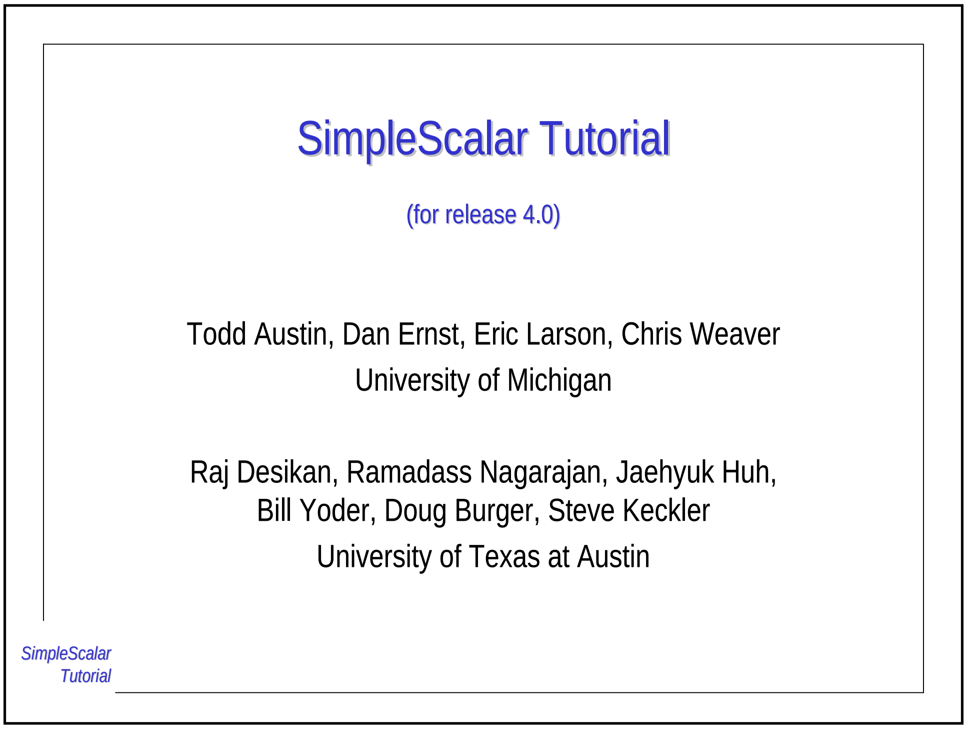
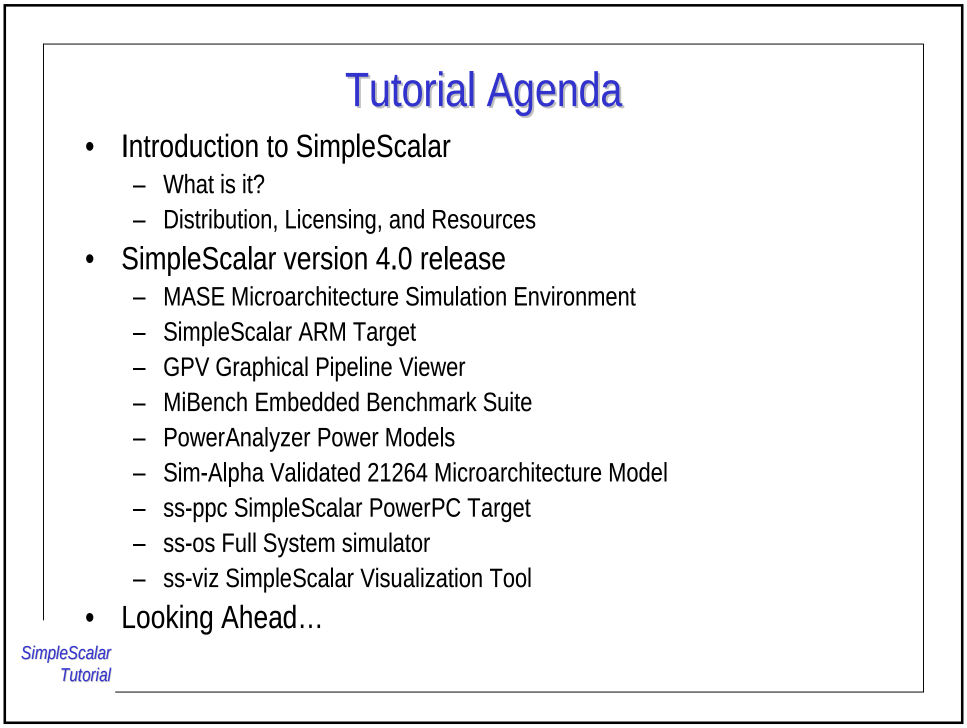


SimpleScalar Tutorial

(for release 4.0)

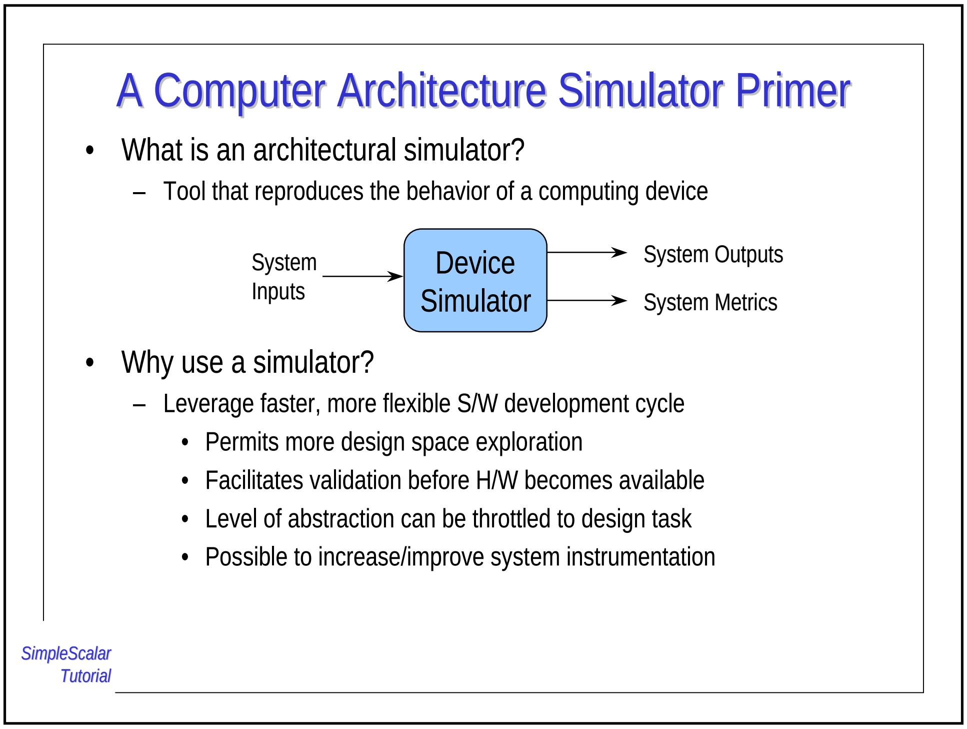
Todd Austin, Dan Ernst, Eric Larson, Chris Weaver
University of Michigan

Raj Desikan, Ramadass Nagarajan, Jaehyuk Huh,
Bill Yoder, Doug Burger, Steve Keckler
University of Texas at Austin



Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…



A Computer Architecture Simulator Primer

- What is an architectural simulator?
  - Tool that reproduces the behavior of a computing device

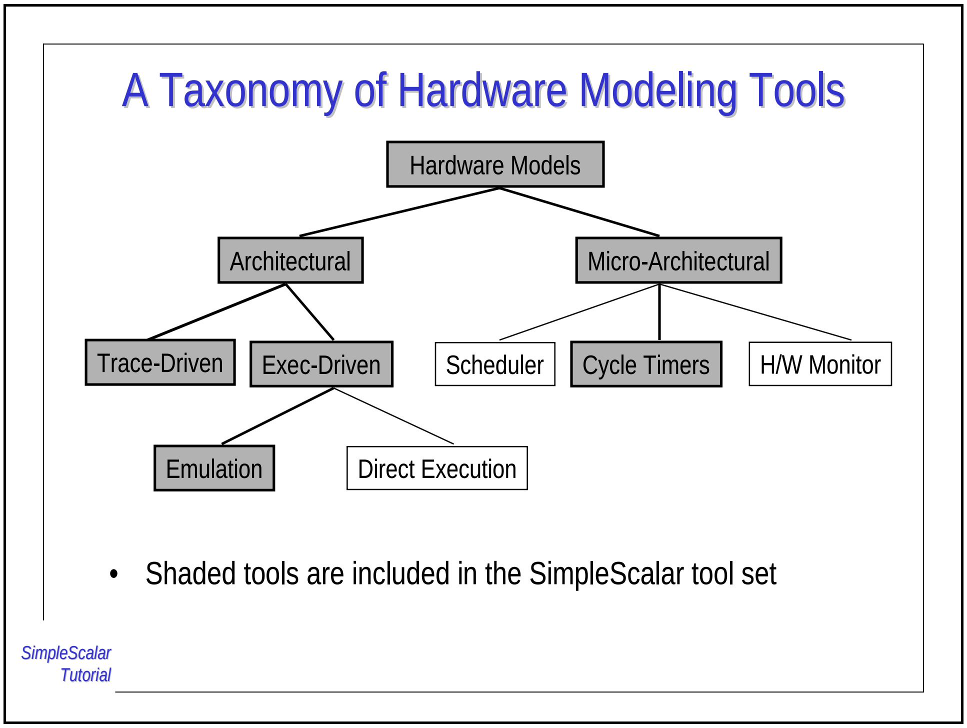
System Inputs

Device Simulator

System Outputs

System Metrics

- Why use a simulator?
  - Leverage faster, more flexible S/W development cycle
    - Permits more design space exploration
    - Facilitates validation before H/W becomes available
    - Level of abstraction can be throttled to design task
    - Possible to increase/improve system instrumentation



A Taxonomy of Hardware Modeling Tools

Hardware Models

Architectural

Micro-Architectural

Trace-Driven

Exec-Driven

Scheduler

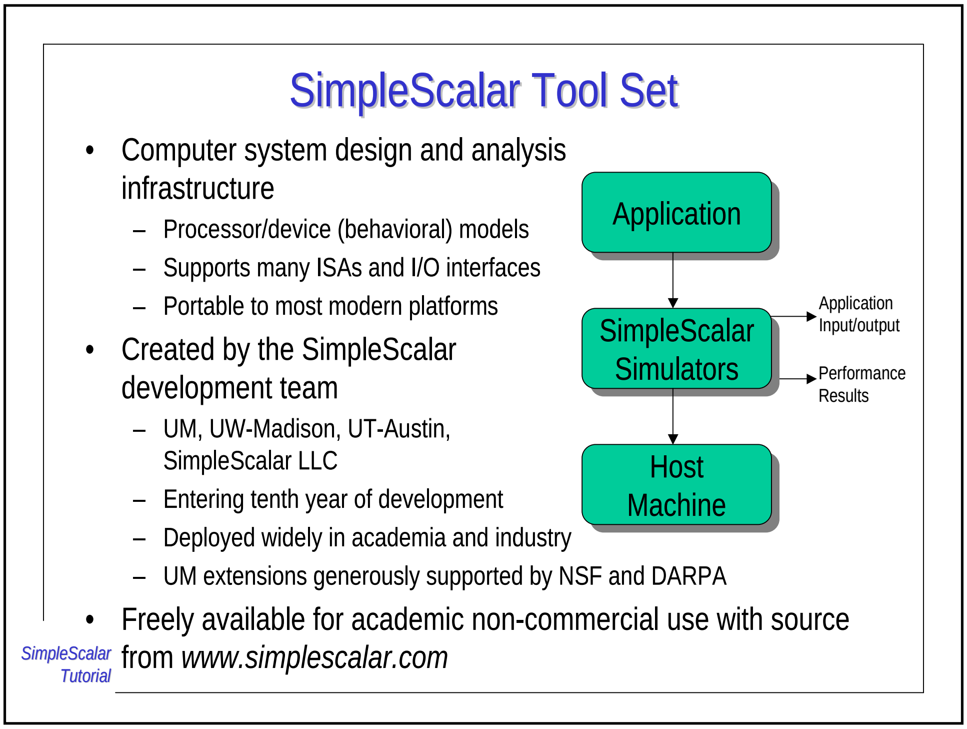
Cycle Timers

H/W Monitor

Emulation

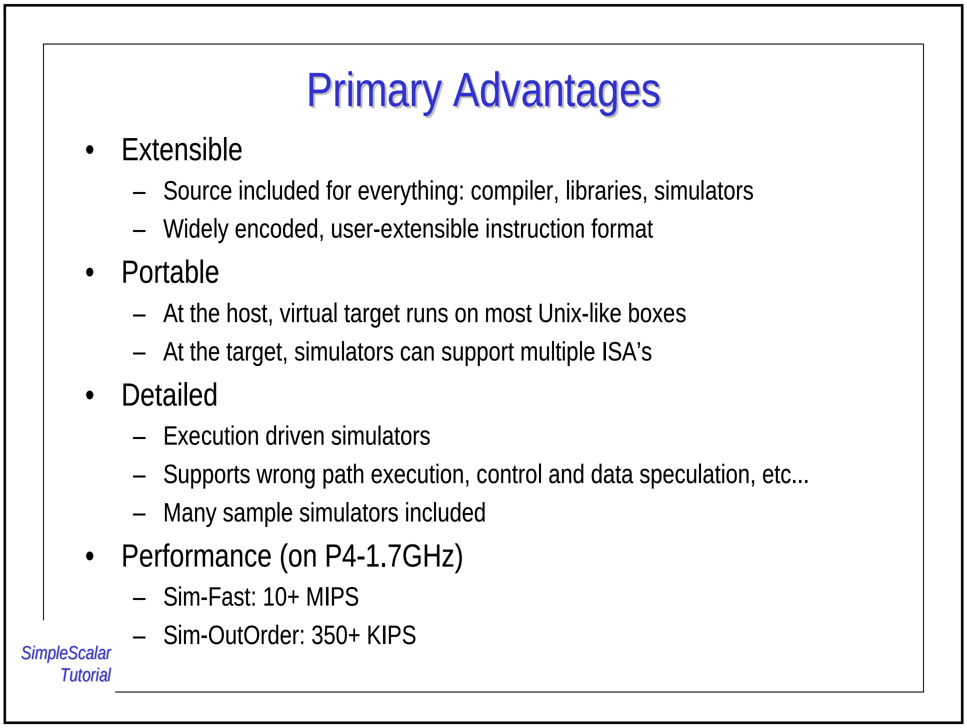
Direct Execution

- Shaded tools are included in the SimpleScalar tool set



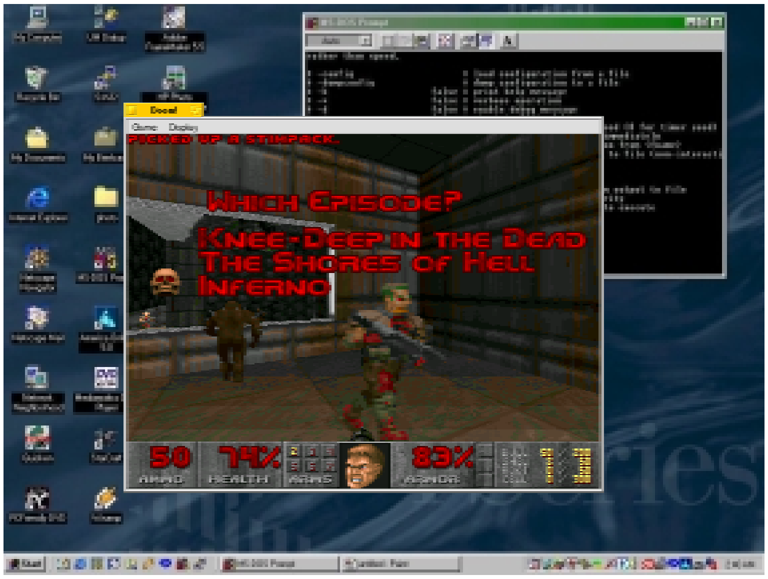
SimpleScalar Tool Set

- Computer system design and analysis infrastructure
  - Processor/device (behavioral) models
  - Supports many ISAs and I/O interfaces
  - Portable to most modern platforms
- Created by the SimpleScalar development team
  - UM, UW-Madison, UT-Austin, SimpleScalar LLC
  - Entering tenth year of development
  - Deployed widely in academia and industry
  - UM extensions generously supported by NSF and DARPA
- Freely available for academic non-commercial use with source from *www.simplescalar.com*

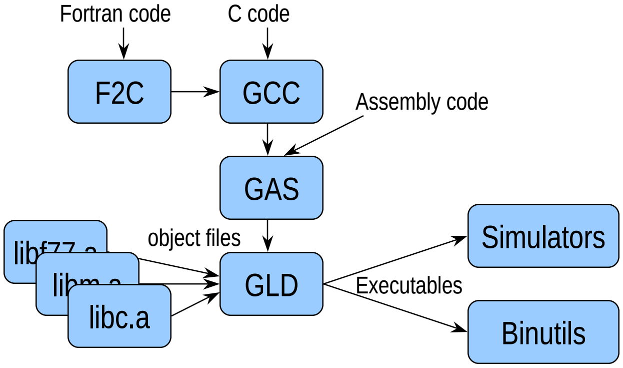


Primary Advantages

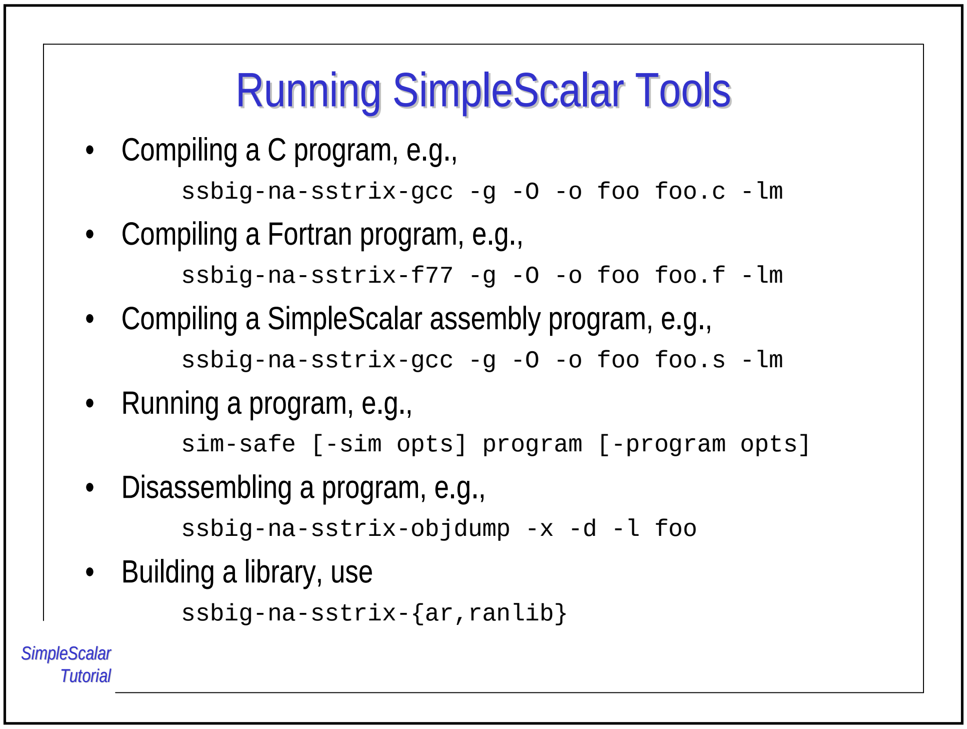
- Extensible
  - Source included for everything: compiler, libraries, simulators
  - Widely encoded, user-extensible instruction format
- Portable
  - At the host, virtual target runs on most Unix-like boxes
  - At the target, simulators can support multiple ISA’s
- Detailed
  - Execution driven simulators
  - Supports wrong path execution, control and data speculation, etc...
  - Many sample simulators included
- Performance (on P4-1.7GHz)
  - Sim-Fast: 10+ MIPS
  - Sim-OutOrder: 350+ KIPS



SimpleScalar Tool Set Overview



- Compiler chain is GNU tools PISA, ARM, etc…
- Fortran codes are compiled with AT&T's f2c, or target FCC
- Libraries are GLIBC ported to SimpleScalar



Running SimpleScalar Tools

- Compiling a C program, e.g.,

```
ssbig-na-sstrix-gcc -g -O -o foo foo.c -lm
```

- Compiling a Fortran program, e.g.,

```
ssbig-na-sstrix-f77 -g -O -o foo foo.f -lm
```

- Compiling a SimpleScalar assembly program, e.g.,

```
ssbig-na-sstrix-gcc -g -O -o foo foo.s -lm
```

- Running a program, e.g.,

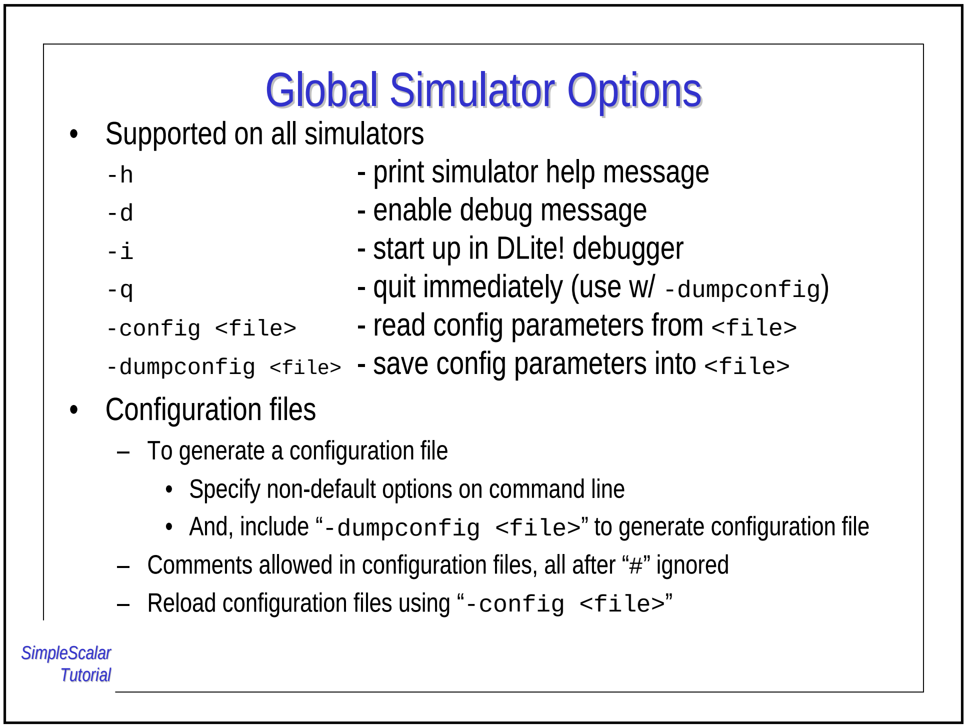
```
sim-safe [-sim opts] program [-program opts]
```

- Disassembling a program, e.g.,

```
ssbig-na-sstrix-objdump -x -d -l foo
```

- Building a library, use

```
ssbig-na-sstrix-{ar,ranlib}
```



Global Simulator Options

- Supported on all simulators

`-h`	- print simulator help message
`-d`	- enable debug message
`-i`	- start up in DLite! debugger
`-q`	- quit immediately (use w/ `-dumpconfig`)
`-config <file>`	- read config parameters from `<file>`
`-dumpconfig <file>`	- save config parameters into `<file>`

- Configuration files
  - To generate a configuration file
    - Specify non-default options on command line
    - And, include "`-dumpconfig <file>`" to generate configuration file
  - Comments allowed in configuration files, all after "#" ignored
  - Reload configuration files using "`-config <file>`"

Sim-Profile: Program Profiling Simulator

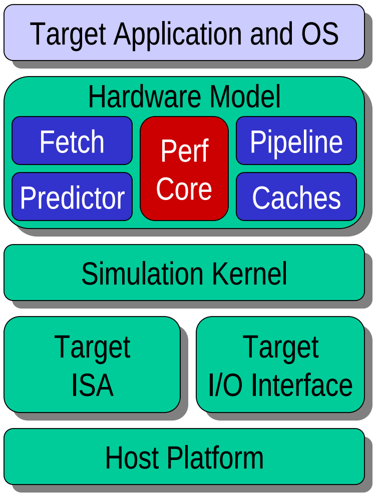
- Generates program profiles, by symbol and by address
- Extra options

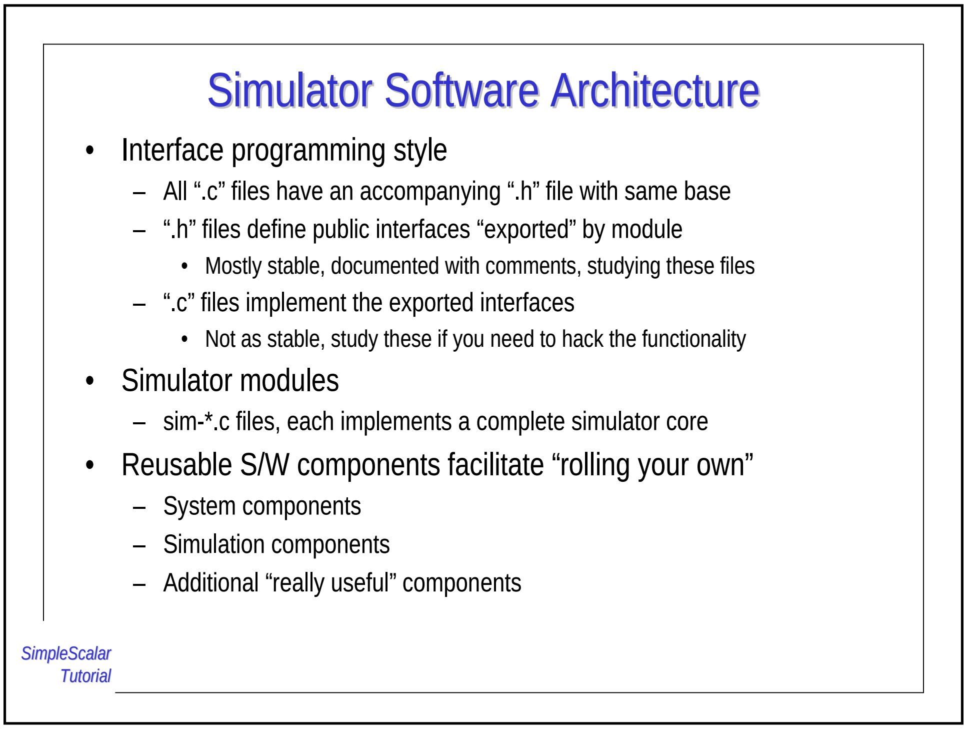
`-iclass`	- instruction class profiling (e.g., ALU, branch)
`-iprof`	- instruction profiling (e.g., bnez, addi, etc...)
`-brprof`	- branch class profiling (e.g., direct, calls, cond)
`-amprof`	- address mode profiling (e.g., displaced, R+R)
`-segprof`	- load/store segment profiling (e.g., data, heap)
`-tsymprof`	- execution profile by text symbol (i.e., funcs)
`-dsymprof`	- reference profile by data segment symbol
`-taddrprof`	- execution profile by text address
`-all`	- enable all of the above options
`-pcstat <stat>`	- record statistic `<stat>` by text address

- NOTE: “`-taddrprof`” `==` “`-pcstat sim_num_insn`”

Simulator Software Architecture

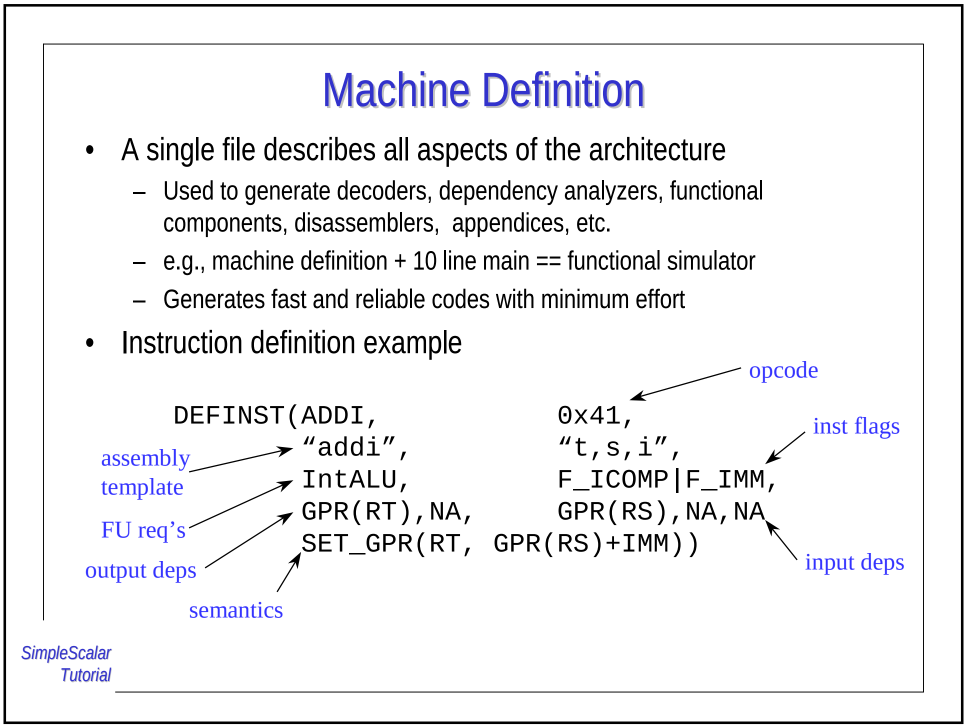
- Target software (apps and OS) runs on simulator
- Performance model tracks time
  - Perf core implements machine
  - Standard modules speed coding
- Simulation kernel provides event simulation services
- Target ISA emulation support
  - PISA, Alpha, StrongARM, PPC, x86
- Target I/O support
  - Syscalls, devices, I/O traces





Simulator Software Architecture

- Interface programming style
  - All “.c” files have an accompanying “.h” file with same base
  - “.h” files define public interfaces “exported” by module
    - Mostly stable, documented with comments, studying these files
  - “.c” files implement the exported interfaces
    - Not as stable, study these if you need to hack the functionality
- Simulator modules
  - sim-*.c files, each implements a complete simulator core
- Reusable S/W components facilitate “rolling your own”
  - System components
  - Simulation components
  - Additional “really useful” components

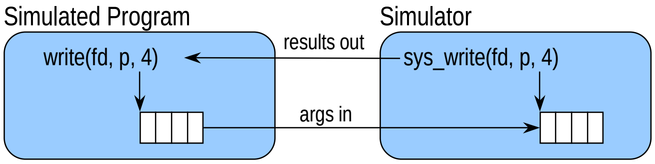


Machine Definition

- A single file describes all aspects of the architecture
  - Used to generate decoders, dependency analyzers, functional components, disassemblers, appendices, etc.
  - e.g., machine definition + 10 line main == functional simulator
  - Generates fast and reliable codes with minimum effort
- Instruction definition example

```
DEFINST(ADDI,           0x41,
        “addi”,         “t,s,i”,
        IntALU,         F_ICOMP|F_IMM,
        GPR(RT),NA,     GPR(RS),NA,NA
        SET_GPR(RT, GPR(RS)+IMM))
```

Simulator I/O



Simulated Program

Simulator

write(fd, p, 4)

results out

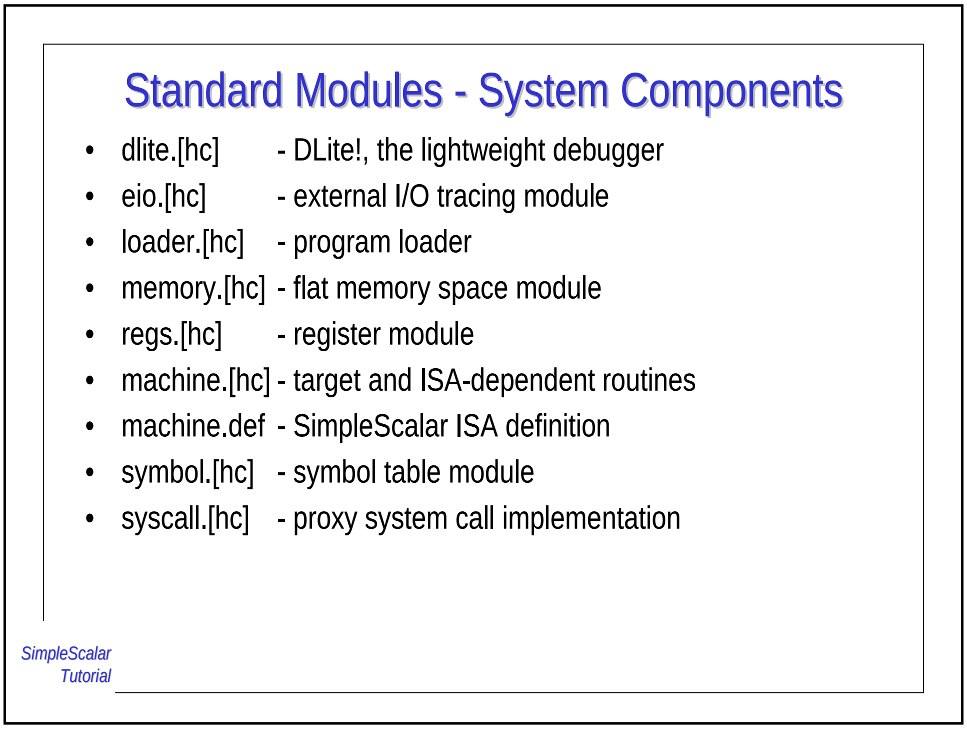
sys_write(fd, p, 4)

args in

- A useful simulator must implement some form of I/O
  - I/O implemented via SYSCALL instruction
  - Supports a subset of Ultrix system calls, proxied out to host
- Basic algorithm (implemented in syscall.c)
  - Decode system call
  - Copy arguments (if any) into simulator memory
  - Perform system call on host
  - Copy results (if any) into simulated program memory

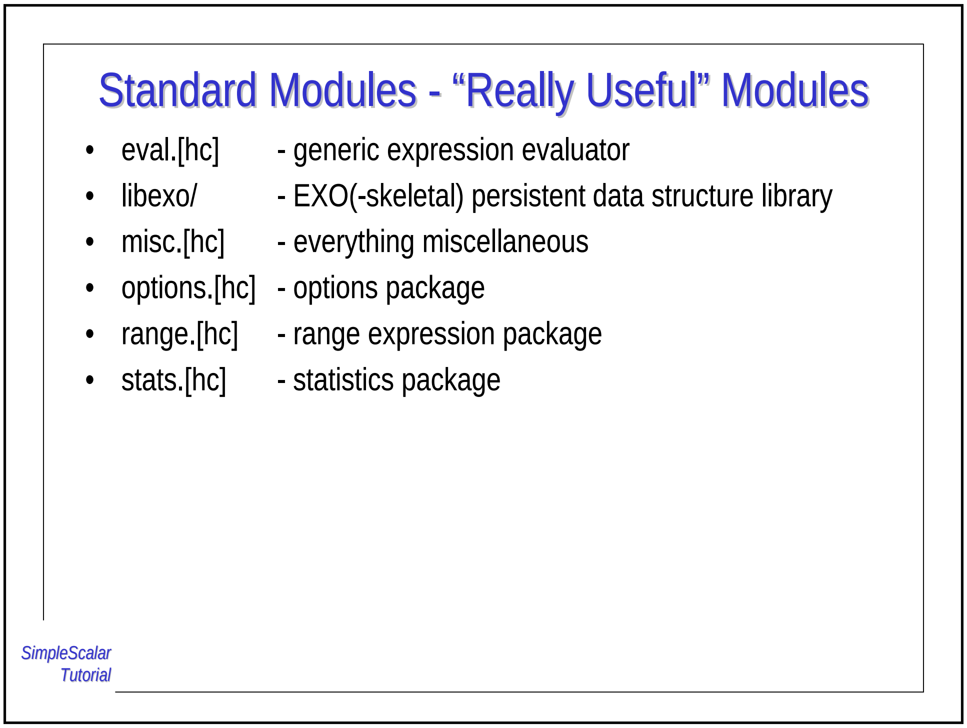
Standard Modules - Simulation Components

- bpred.[hc] - branch predictors
- cache.[hc] - cache module
- eventq.[hc] - event queue module
- libcheetah/ - Cheetah cache simulator library
- ptrace.[hc] - pipetrace module
- res.[hc] - resource manager module
- sim.h - simulator main code interface definitions
- textprof.pl - text segment profile view (Perl Script)
- pipeview.pl - pipetrace view (Perl script)



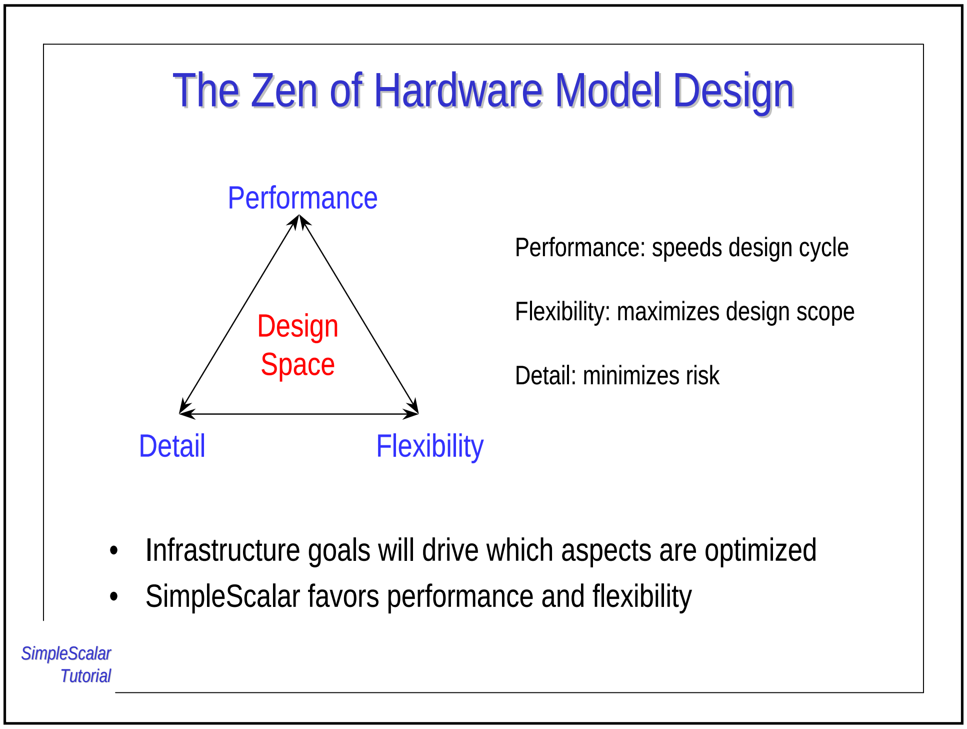
Standard Modules - System Components

- dlite.[hc] - DLite!, the lightweight debugger
- eio.[hc] - external I/O tracing module
- loader.[hc] - program loader
- memory.[hc] - flat memory space module
- regs.[hc] - register module
- machine.[hc] - target and ISA-dependent routines
- machine.def - SimpleScalar ISA definition
- symbol.[hc] - symbol table module
- syscall.[hc] - proxy system call implementation



Standard Modules - “Really Useful” Modules

- eval.[hc] - generic expression evaluator
- libexo/ - EXO(-skeletal) persistent data structure library
- misc.[hc] - everything miscellaneous
- options.[hc] - options package
- range.[hc] - range expression package
- stats.[hc] - statistics package



The Zen of Hardware Model Design

Performance

Design Space

Detail

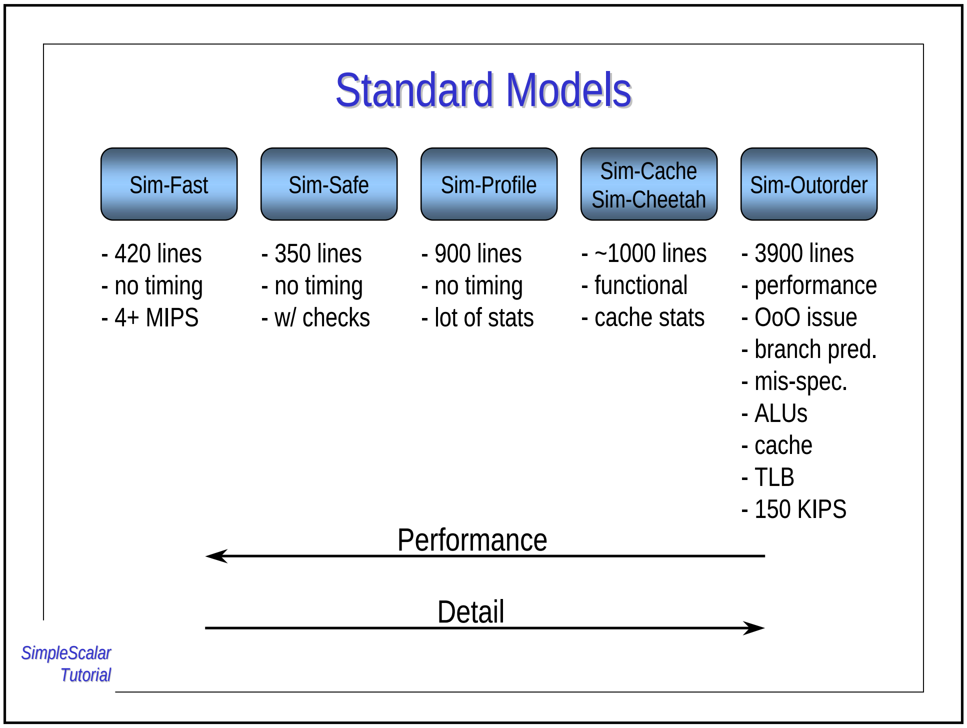
Flexibility

Performance: speeds design cycle

Flexibility: maximizes design scope

Detail: minimizes risk

- Infrastructure goals will drive which aspects are optimized
- SimpleScalar favors performance and flexibility



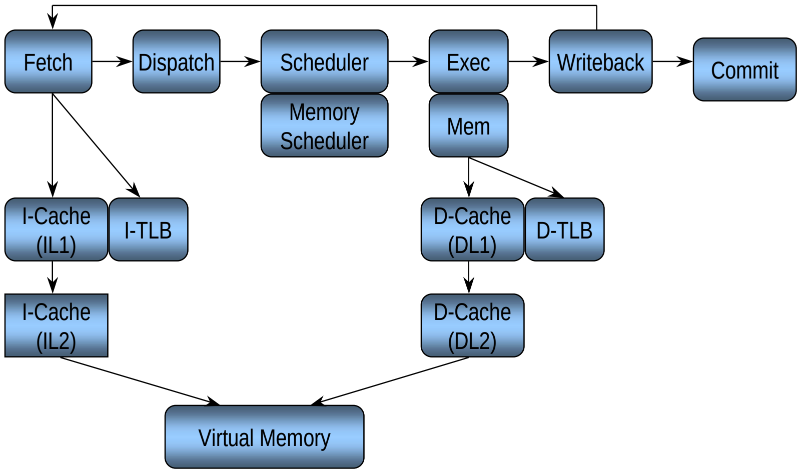
Standard Models

Sim-Fast	Sim-Safe	Sim-Profile	Sim-Cache Sim-Cheetah	Sim-Outorder
- 420 lines	- 350 lines	- 900 lines	- ~1000 lines	- 3900 lines
- no timing	- no timing	- no timing	- functional	- performance
- 4+ MIPS	- w/ checks	- lot of stats	- cache stats	- OoO issue
				- branch pred.
				- mis-spec.
				- ALUs
				- cache
				- TLB
				- 150 KIPS

Performance

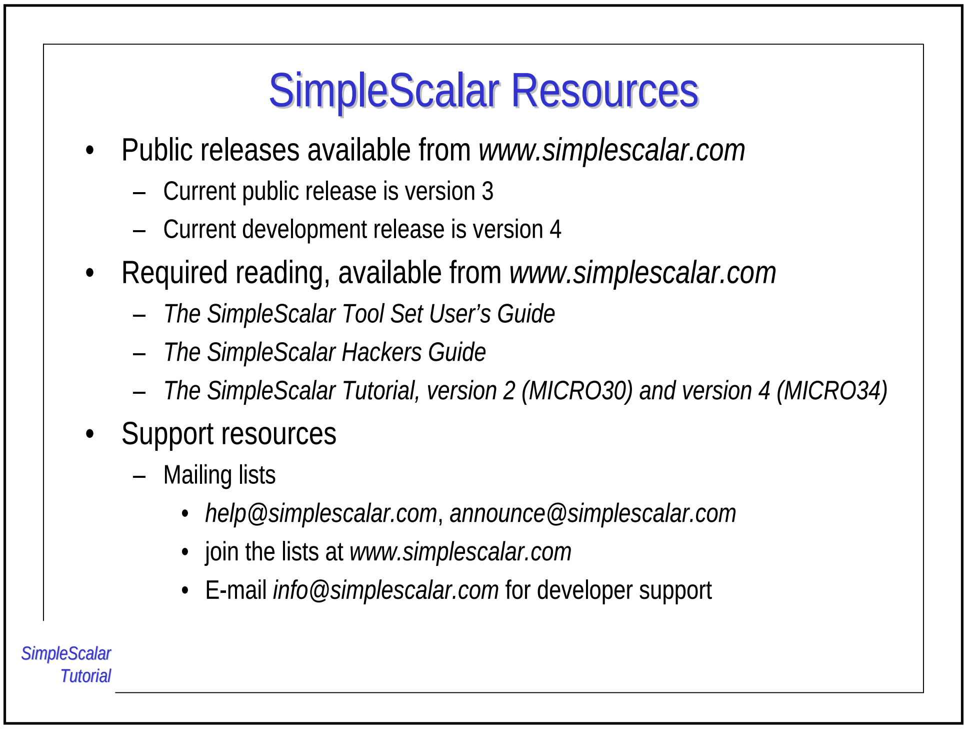
Detail

Out-of-Order Issue Simulator



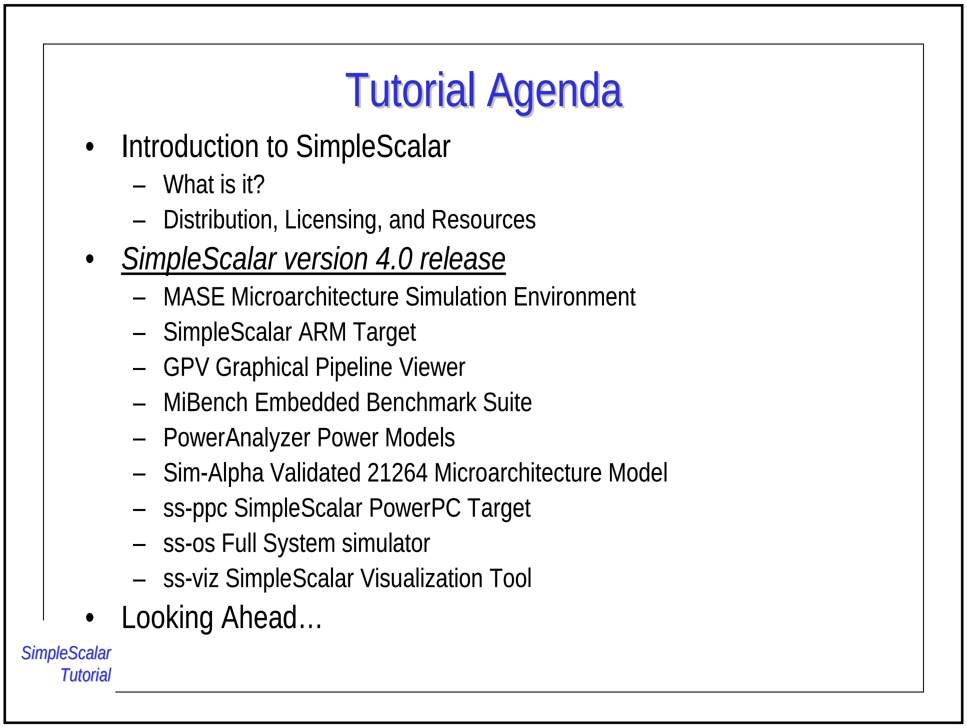
Distribution and Licensing

- Download from *www.simplescalar.com*
  - Code releases and updates
  - Cross-compilers and other tool chains
  - Benchmarks sources, binaries, and test inputs
  - User-contributed developments
- SimpleScalar licensing
  - Non-commercial academic use licenses (research or instruction) are available free of charge
  - Commercial use licenses available from SimpleScalar LLC
    - Required for any use by a for-profit business/institution
    - Two options available: Site and research participation licenses
    - Contact *info@simplescalar.com* for complete details



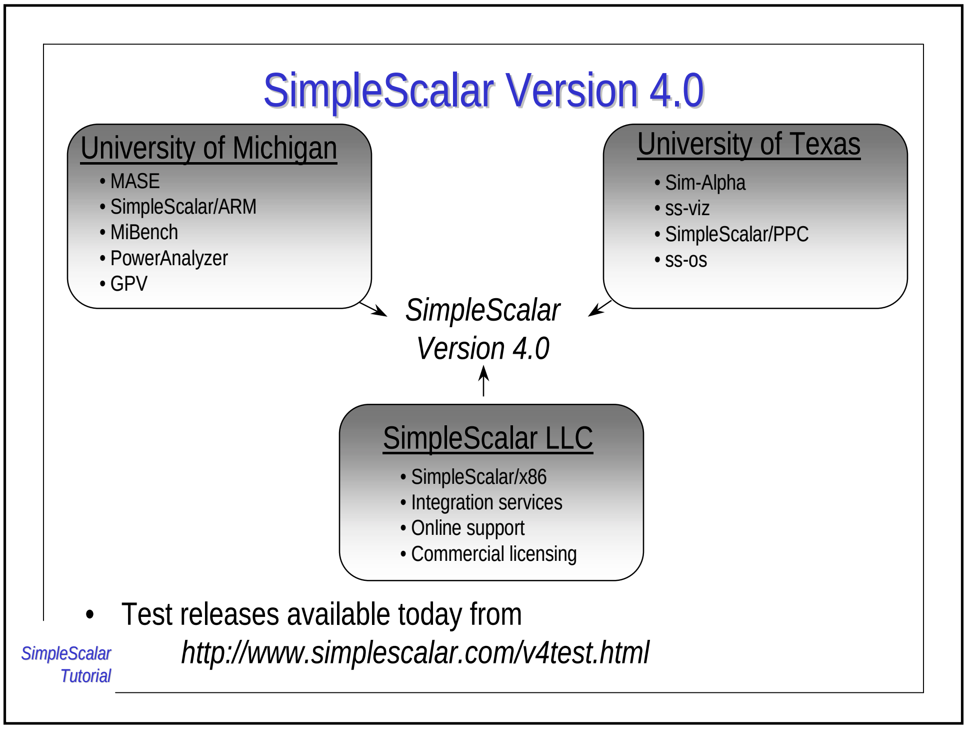
SimpleScalar Resources

- Public releases available from *www.simplescalar.com*
  - Current public release is version 3
  - Current development release is version 4
- Required reading, available from *www.simplescalar.com*
  - *The SimpleScalar Tool Set User's Guide*
  - *The SimpleScalar Hackers Guide*
  - *The SimpleScalar Tutorial, version 2 (MICRO30) and version 4 (MICRO34)*
- Support resources
  - Mailing lists
    - *help@simplescalar.com*, *announce@simplescalar.com*
    - join the lists at *www.simplescalar.com*
    - E-mail *info@simplescalar.com* for developer support



Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- *SimpleScalar version 4.0 release*
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…



SimpleScalar Version 4.0

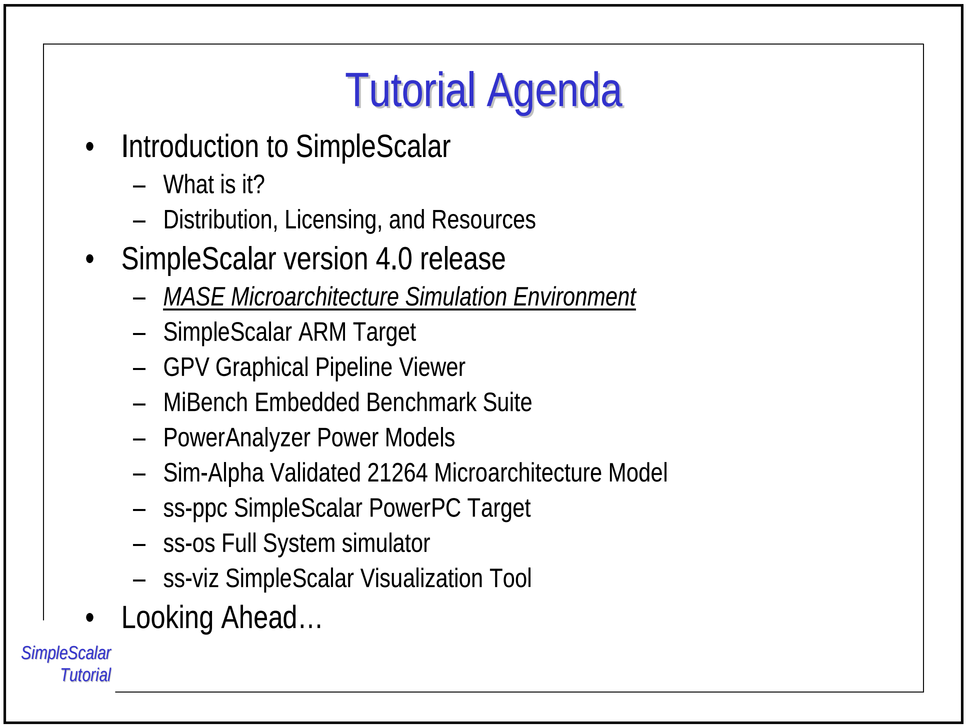
University of Michigan
- MASE
- SimpleScalar/ARM
- MiBench
- PowerAnalyzer
- GPV

University of Texas
- Sim-Alpha
- ss-viz
- SimpleScalar/PPC
- ss-os

SimpleScalar Version 4.0

SimpleScalar LLC
- SimpleScalar/x86
- Integration services
- Online support
- Commercial licensing

- Test releases available today from
  http://www.simplescalar.com/v4test.html



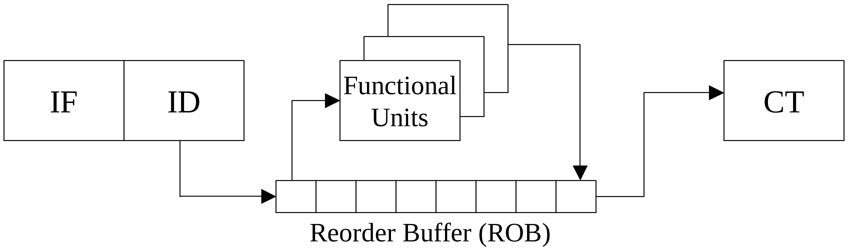
Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - *MASE Microarchitecture Simulation Environment*
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…

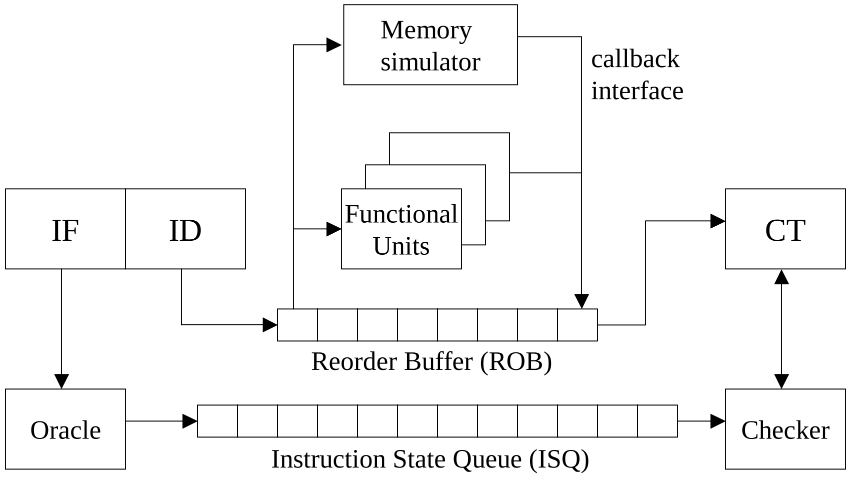
MASE Microarchitectural Simulation Environment

- MASE is a new performance simulation infrastructure for SimpleScalar.
  - Developed by Eric Larson, Saugata Chatterjee, and Dan Ernst
- Features and goals of MASE:
  - Checker improves validation support.
  - Oracle allows for "perfect" studies.
  - Micro-functional performance model increases accuracy.
  - Speculative state management facilities simplify aggressive speculation.
  - Callback interface permits sophisticated memory system simulation.

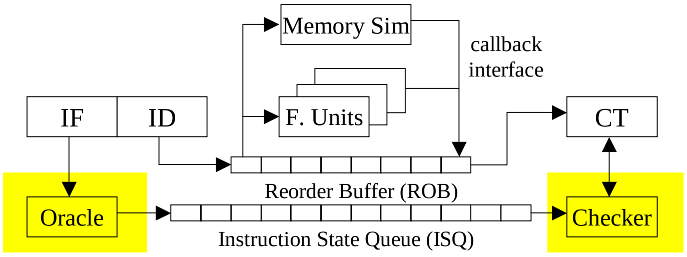
SimpleScalar 3.0 software architecture



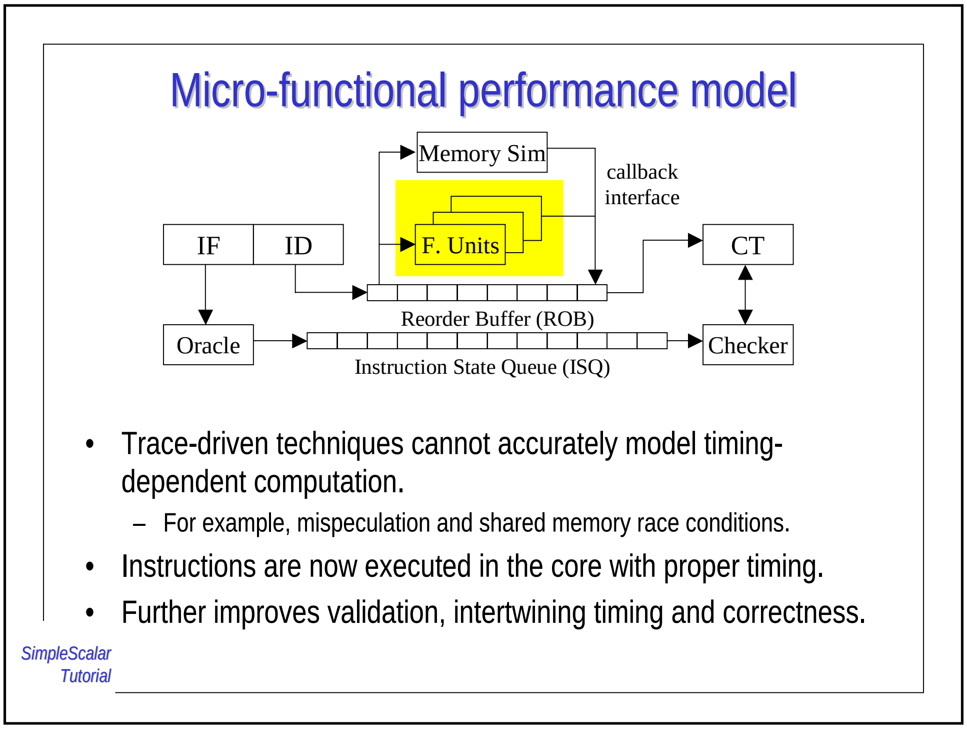
MASE software architecture



Checker and oracle

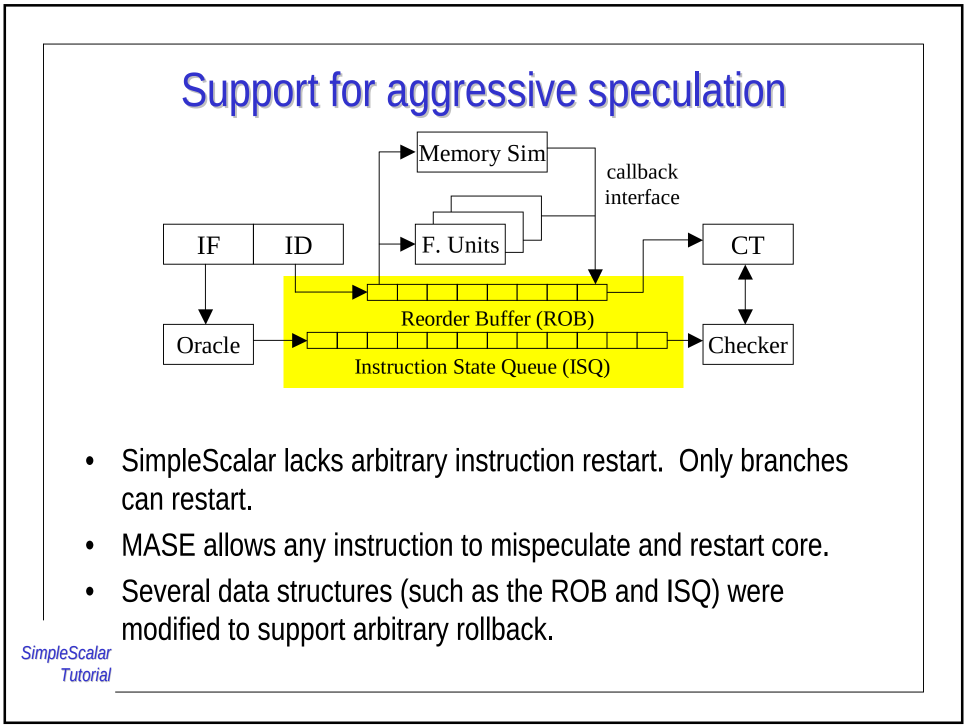


- Permit “perfect” studies and improved validation.
- Oracle executes in fetch and places values into ISQ.
- Checker uses ISQ values to validate core computation.
- Checker will fix any core bug, reducing burden of correctness in core.



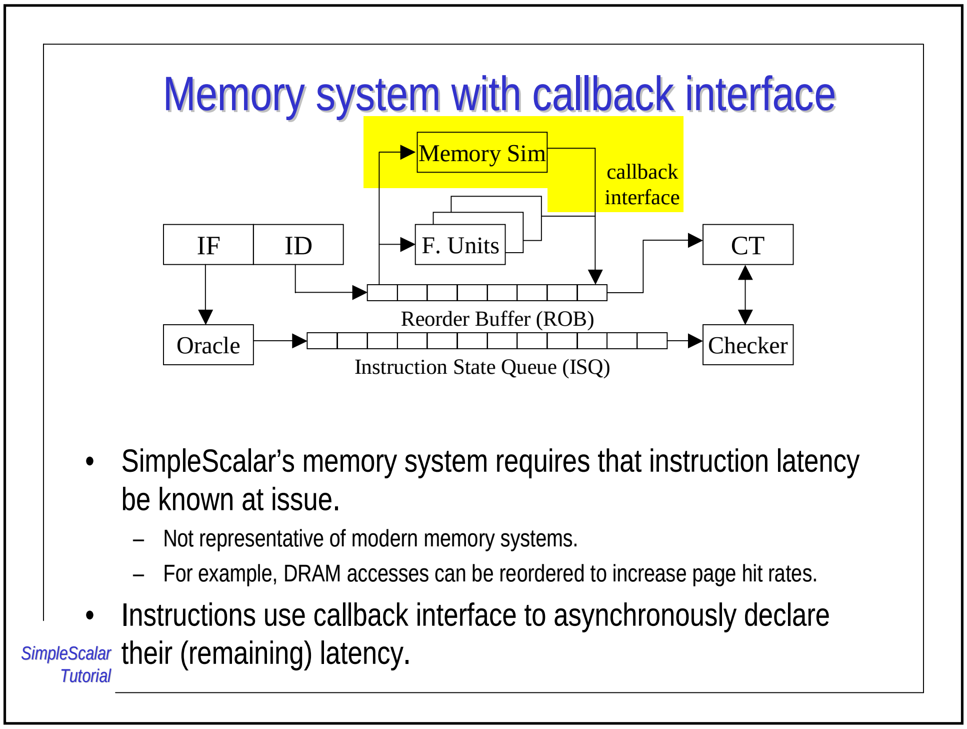
Micro-functional performance model

- Trace-driven techniques cannot accurately model timing-dependent computation.
  - For example, mispeculation and shared memory race conditions.
- Instructions are now executed in the core with proper timing.
- Further improves validation, intertwining timing and correctness.



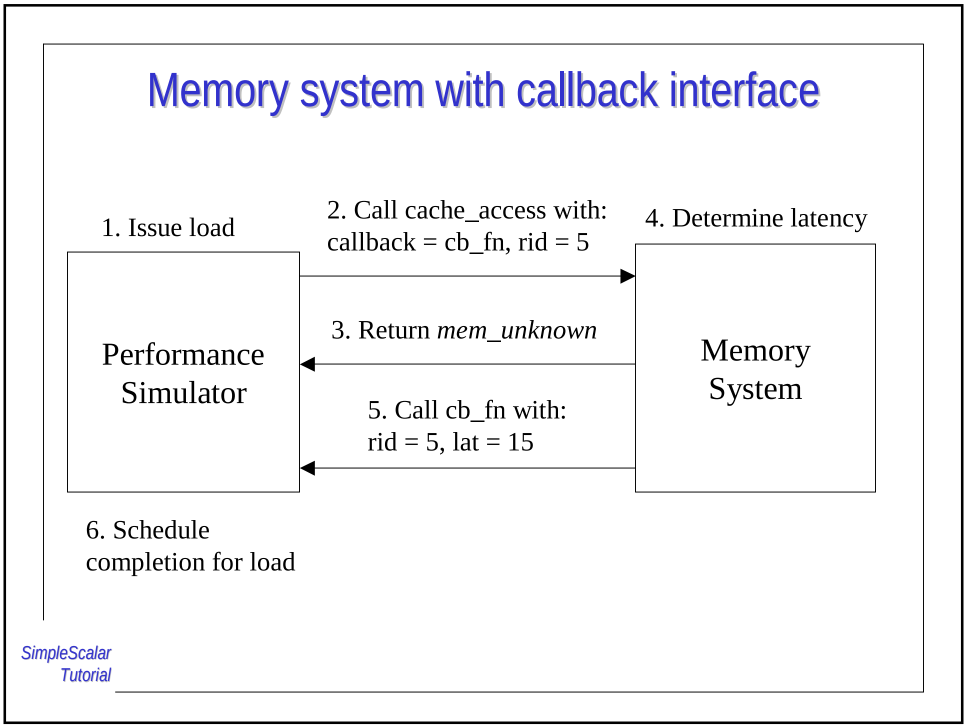
Support for aggressive speculation

- SimpleScalar lacks arbitrary instruction restart. Only branches can restart.
- MASE allows any instruction to mispeculate and restart core.
- Several data structures (such as the ROB and ISQ) were modified to support arbitrary rollback.



Memory system with callback interface

- SimpleScalar's memory system requires that instruction latency be known at issue.
  - Not representative of modern memory systems.
  - For example, DRAM accesses can be reordered to increase page hit rates.
- Instructions use callback interface to asynchronously declare their (remaining) latency.



Memory system with callback interface

1. Issue load

2. Call cache_access with:
callback = cb_fn, rid = 5

4. Determine latency

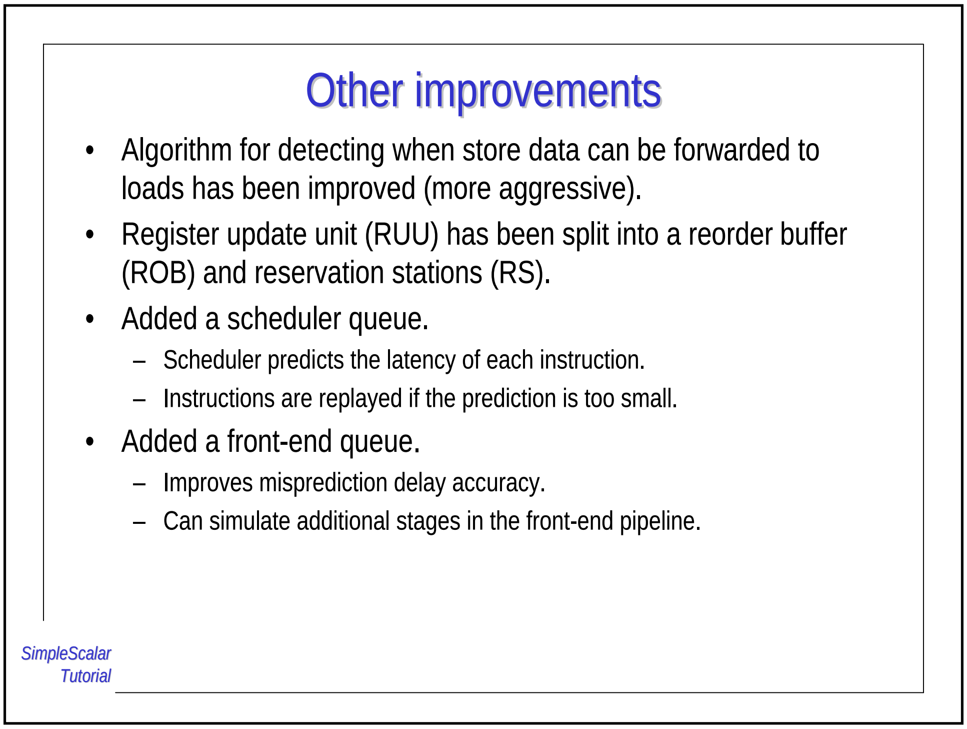
Performance Simulator

3. Return *mem_unknown*

Memory System

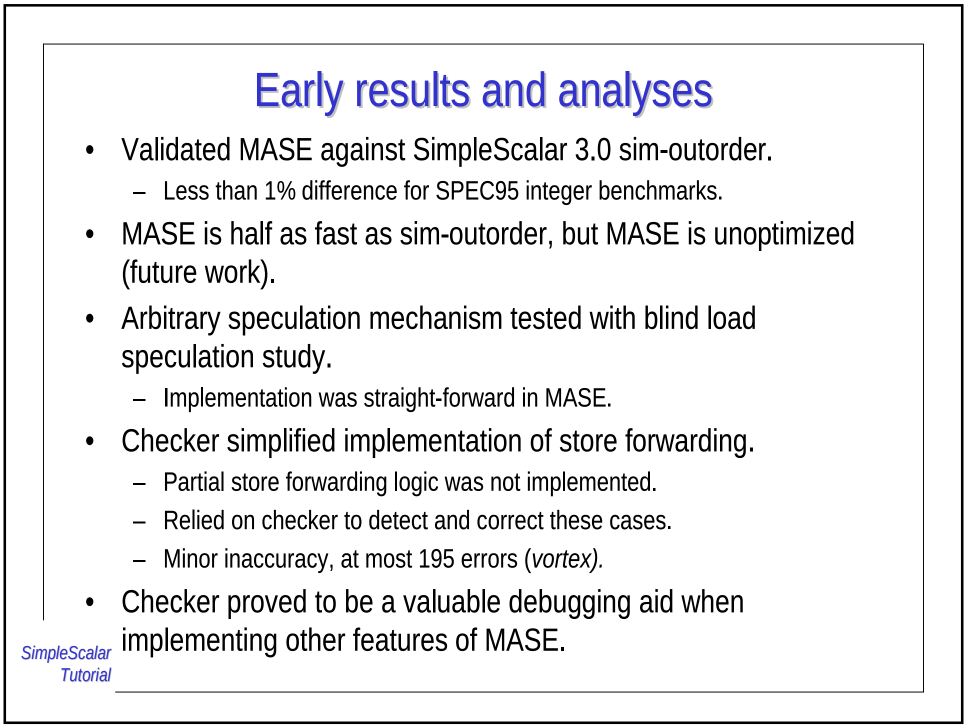
5. Call cb_fn with:
rid = 5, lat = 15

6. Schedule
completion for load



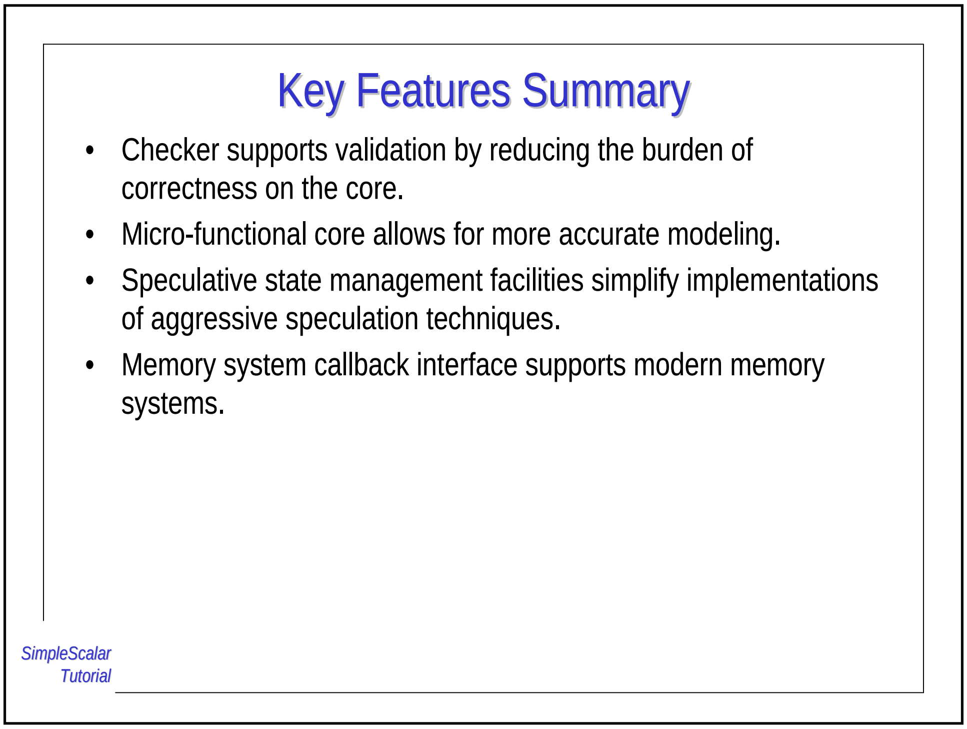
Other improvements

- Algorithm for detecting when store data can be forwarded to loads has been improved (more aggressive).
- Register update unit (RUU) has been split into a reorder buffer (ROB) and reservation stations (RS).
- Added a scheduler queue.
  - Scheduler predicts the latency of each instruction.
  - Instructions are replayed if the prediction is too small.
- Added a front-end queue.
  - Improves misprediction delay accuracy.
  - Can simulate additional stages in the front-end pipeline.



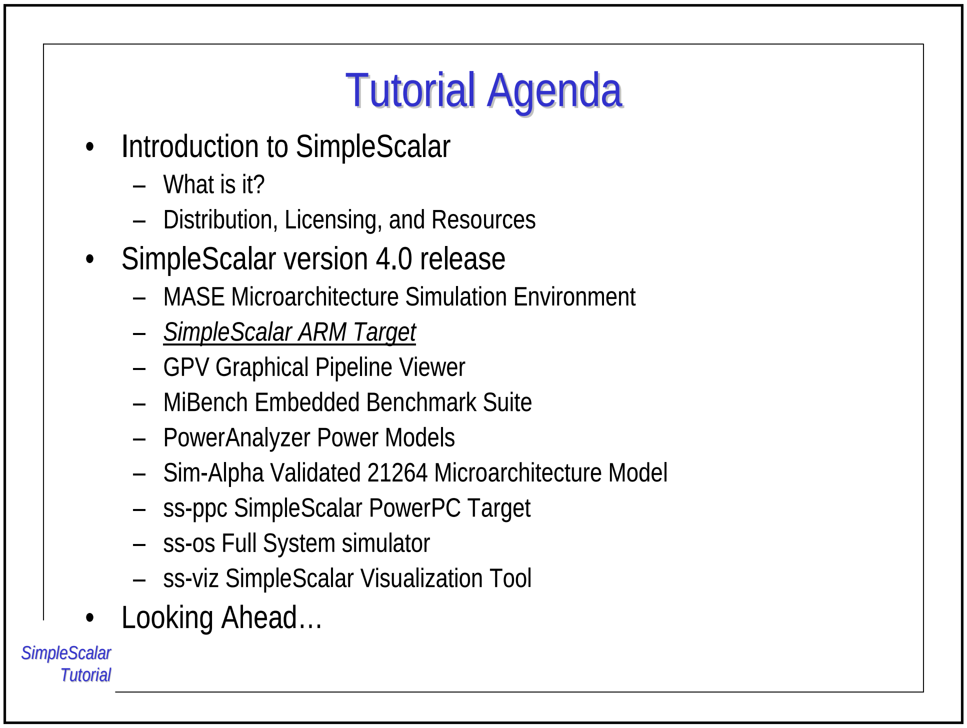
Early results and analyses

- Validated MASE against SimpleScalar 3.0 sim-outorder.
  - Less than 1% difference for SPEC95 integer benchmarks.
- MASE is half as fast as sim-outorder, but MASE is unoptimized (future work).
- Arbitrary speculation mechanism tested with blind load speculation study.
  - Implementation was straight-forward in MASE.
- Checker simplified implementation of store forwarding.
  - Partial store forwarding logic was not implemented.
  - Relied on checker to detect and correct these cases.
  - Minor inaccuracy, at most 195 errors (*vortex).*
- Checker proved to be a valuable debugging aid when implementing other features of MASE.



Key Features Summary

- Checker supports validation by reducing the burden of correctness on the core.
- Micro-functional core allows for more accurate modeling.
- Speculative state management facilities simplify implementations of aggressive speculation techniques.
- Memory system callback interface supports modern memory systems.

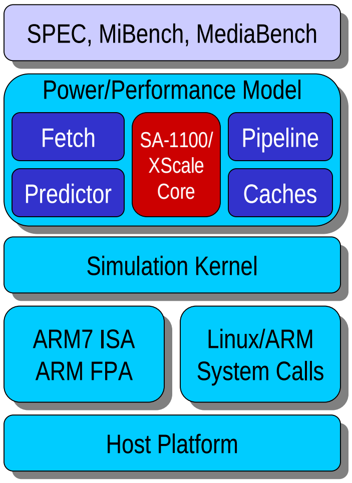


Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - *SimpleScalar ARM Target*
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…

SimpleScalar/ARM Target

- ARM simulation target
  - Developed by Dan Ernst and Chris Weaver
- ARM7 apps run on emulator
  - SPEC, MiBench, MediaBench
- Linux system call I/O emulator
  - Supports file, network, console I/O
- Multiple validated processor models
  - Intel StrongARM SA-1110
  - Intel XScale 80200
  - Performance and power models validated



ARM Target Instruction Emulation

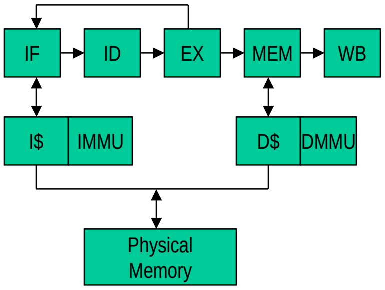
- ARM ISA emulation support added to SimpleScalar tool set
  - ARM 7 integer instruction set support
  - Floating Point Accelerator (FPA) instruction set support
- Linux/ARM system call support added
  - System calls are implemented by the simulator
  - Portable I/O, but does not capture OS execution
- ARM CISC instructions required microcode support
  - Needed for microarchitectural modeling

```
stmdb  r13!,{r4-r8,r10-r15}   ➡   agen  tmp1,r13,0
                                   agen  tmp0,tmp1,-16
                                   stp  r11,[tmp0]
                                   agen  r13,r13,-16
                                   agen  tmp0,tmp1,-12
                                   stp  r12,[tmp0]
                                   agen  tmp0,tmp1,-8
                                   stp  r14,[tmp0]
                                   agen  tmp0,tmp1,-4
                                   stp  r15,[tmp0]
```

Processor Performance Model

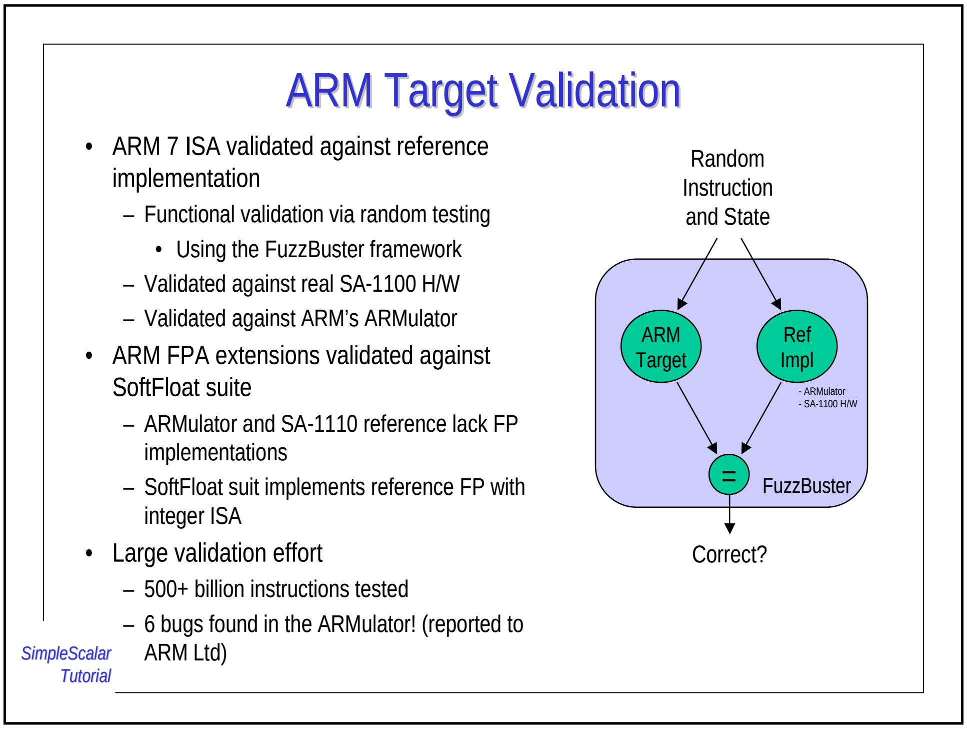
- SA-1 pipeline model implemented
  - Pipeline used in Intel’s SA-11xx
  - Simple five stage pipeline
  - Two level memory hierarchy
- Challenging task due to lack of info on SA-1 microarchitecture
  - Derived many details from the compiler writers guide
  - Used directed black-box testing to fill in the rest of the blanks
- prototype XScale model completed
  - Intel’s new StrongARM processor
  - Based on (sparse) published details
  - Validation ongoing against XScale 80200 evaluation board

SA-1 Pipeline



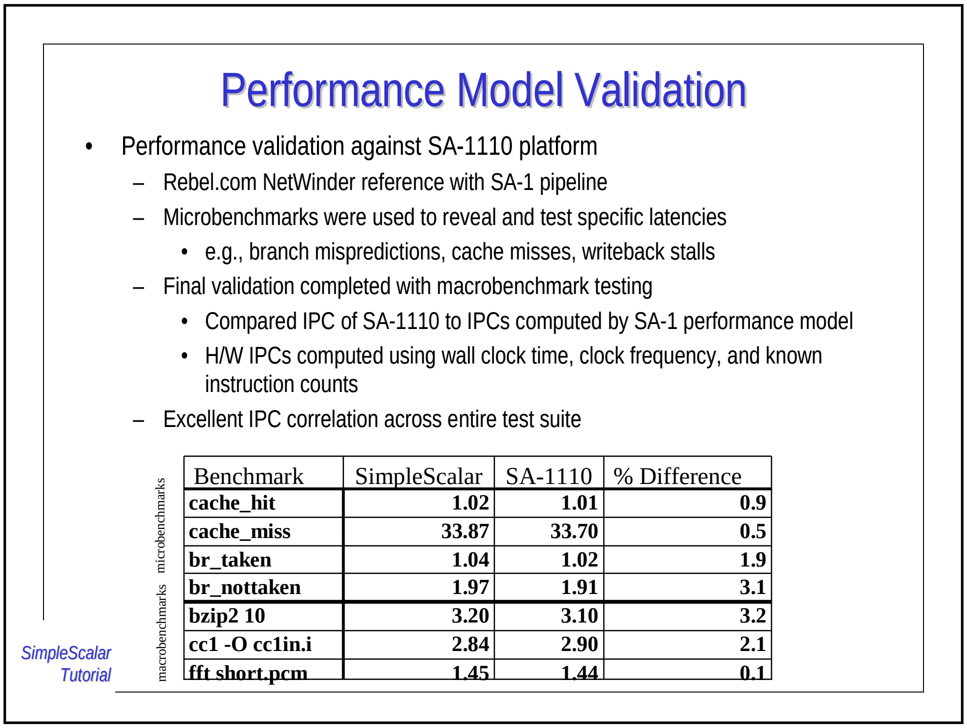
ARM Cross-Compiler Kit

- Permits users to compile ARM binaries w/o ARM hardware
  - Most users lack access to a real ARM target with a native compiler
  - We use Rebel.com’s NetWinder platforms to build native binaries
- GNU GCC targeted to ARM ISA
  - includes soft-float support (permits compilation for non-FP hardware)
- GNU binutils targeted to ARM ISA
  - GNU ld linker
  - GNU binary utilies, e.g., objdump, nm, size, etc…
- Pre-built C libraries for ARM ISA
  - Targeted to Linux system call interfaces
- Portable code base



ARM Target Validation

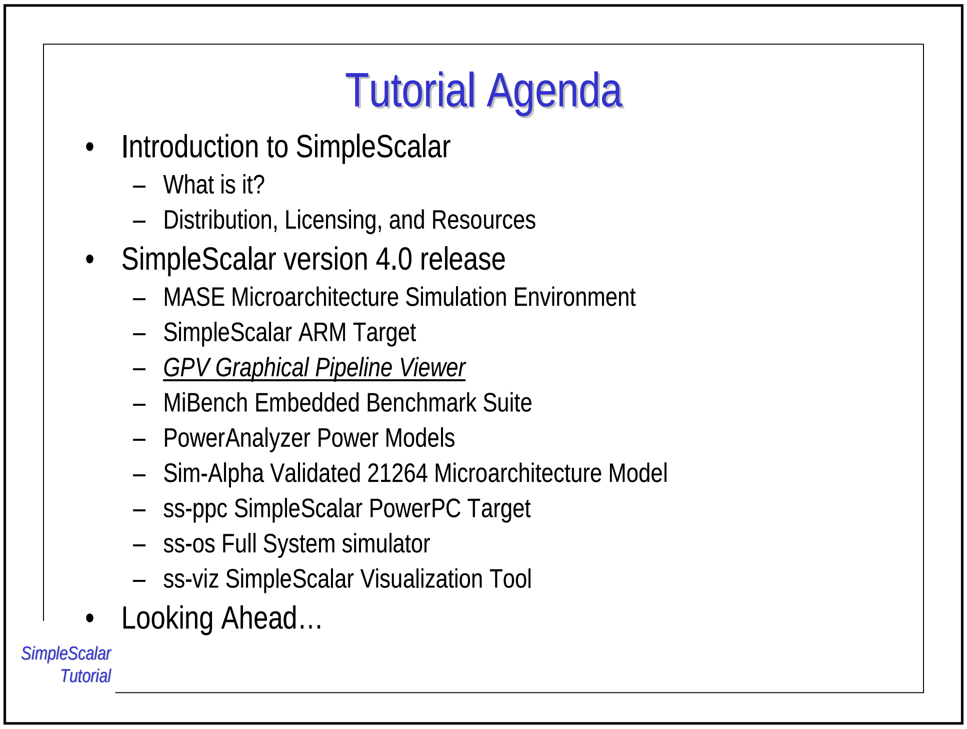
- ARM 7 ISA validated against reference implementation
  - – Functional validation via random testing
    - Using the FuzzBuster framework
  - – Validated against real SA-1100 H/W
  - – Validated against ARM's ARMulator
- ARM FPA extensions validated against SoftFloat suite
  - – ARMulator and SA-1110 reference lack FP implementations
  - – SoftFloat suit implements reference FP with integer ISA
- Large validation effort
  - – 500+ billion instructions tested
  - – 6 bugs found in the ARMulator! (reported to ARM Ltd)



Performance Model Validation

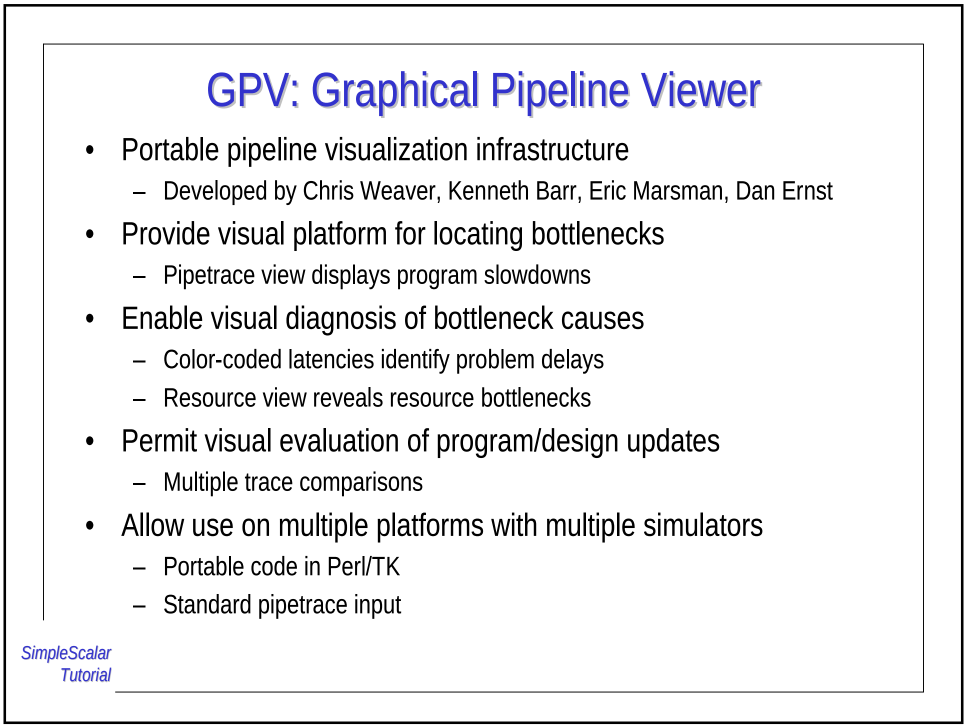
- Performance validation against SA-1110 platform
  - – Rebel.com NetWinder reference with SA-1 pipeline
  - – Microbenchmarks were used to reveal and test specific latencies
    - e.g., branch mispredictions, cache misses, writeback stalls
  - – Final validation completed with macrobenchmark testing
    - Compared IPC of SA-1110 to IPCs computed by SA-1 performance model
    - H/W IPCs computed using wall clock time, clock frequency, and known instruction counts
  - – Excellent IPC correlation across entire test suite

	Benchmark	SimpleScalar	SA-1110	% Difference
microbenchmarks	**cache_hit**	**1.02**	**1.01**	**0.9**
	cache_miss	**33.87**	**33.70**	**0.5**
	br_taken	**1.04**	**1.02**	**1.9**
	br_nottaken	**1.97**	**1.91**	**3.1**
macrobenchmarks	**bzip2 10**	**3.20**	**3.10**	**3.2**
	cc1 -O cc1in.i	**2.84**	**2.90**	**2.1**
	fft short.pcm	**1.45**	**1.44**	**0.1**



Tutorial Agenda

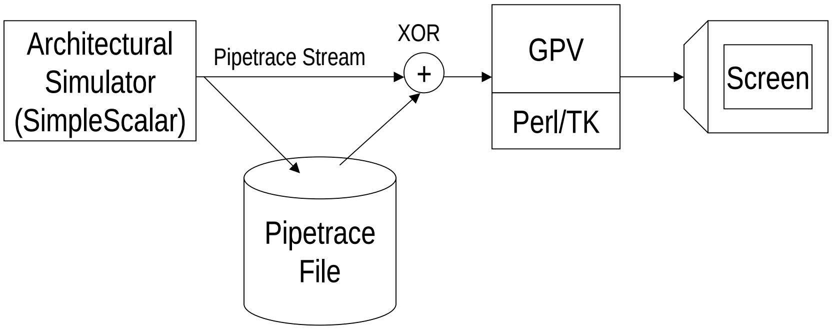
- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - *GPV Graphical Pipeline Viewer*
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…



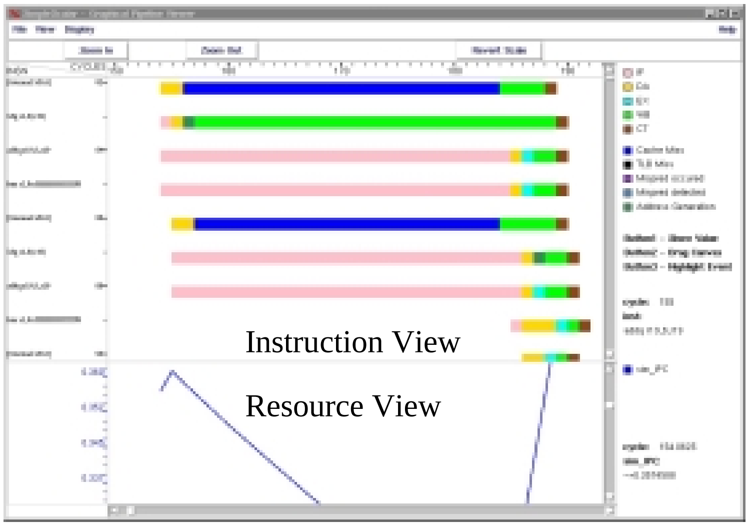
GPV: Graphical Pipeline Viewer

- Portable pipeline visualization infrastructure
  - Developed by Chris Weaver, Kenneth Barr, Eric Marsman, Dan Ernst
- Provide visual platform for locating bottlenecks
  - Pipetrace view displays program slowdowns
- Enable visual diagnosis of bottleneck causes
  - Color-coded latencies identify problem delays
  - Resource view reveals resource bottlenecks
- Permit visual evaluation of program/design updates
  - Multiple trace comparisons
- Allow use on multiple platforms with multiple simulators
  - Portable code in Perl/TK
  - Standard pipetrace input

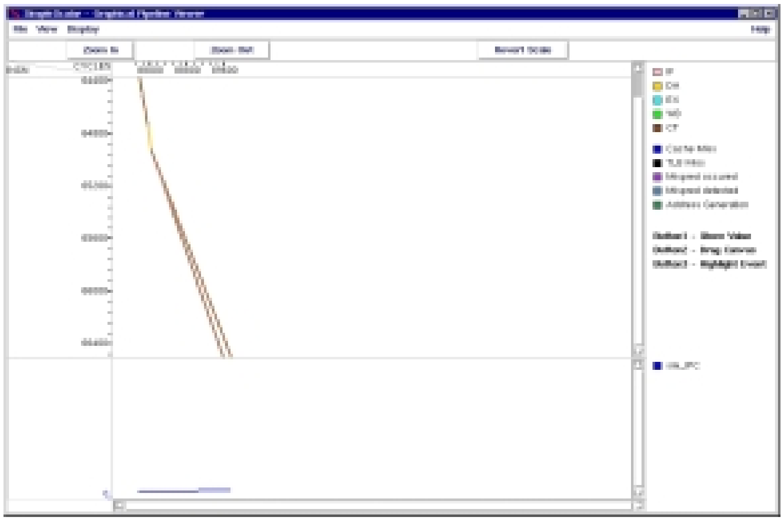
GPV Software Architecture



Main Window

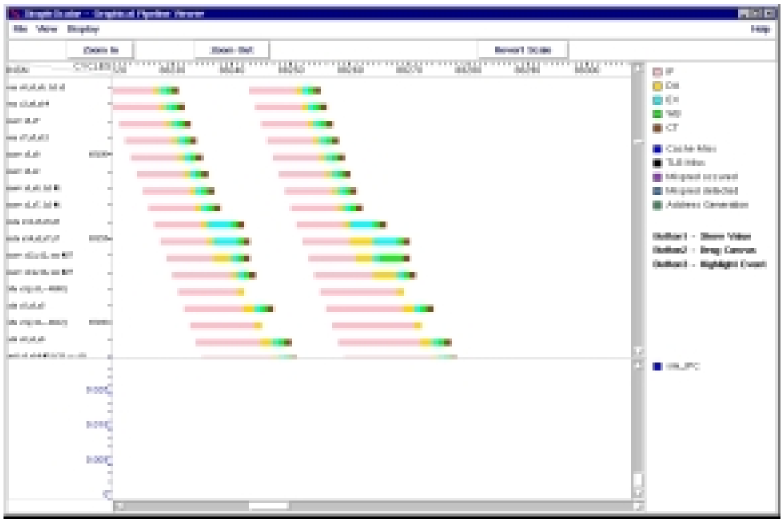


Zoom Feature

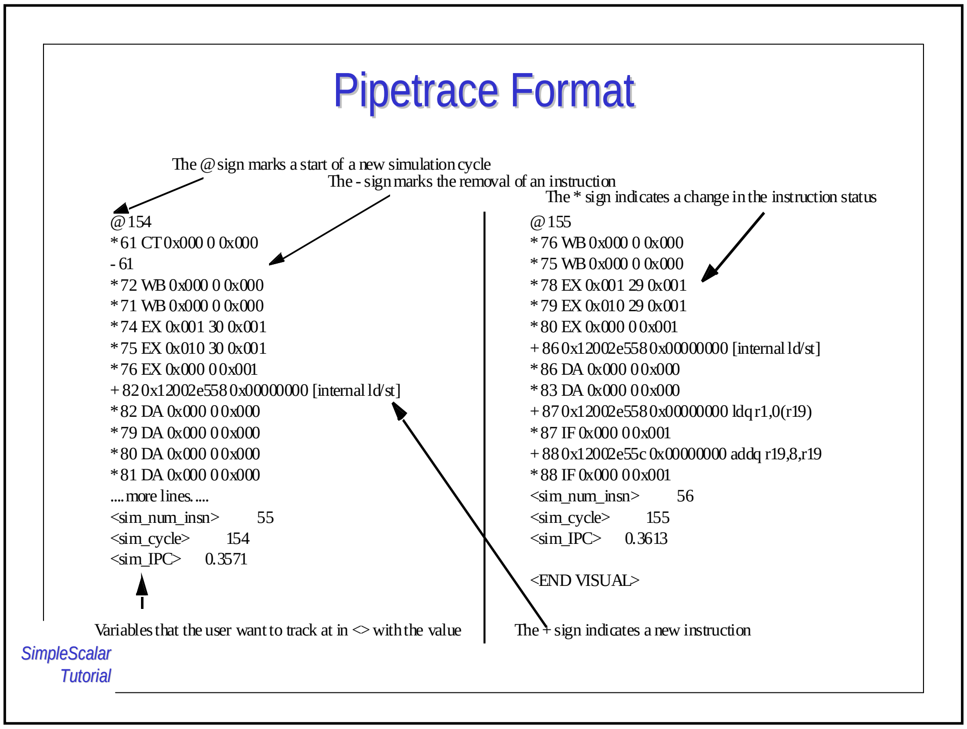


SimpleScalar
Tutorial

Zoom Feature



SimpleScalar
Tutorial



Pipetrace Format

The @ sign marks a start of a new simulation cycle

The - sign marks the removal of an instruction

The * sign indicates a change in the instruction status

```
@154
*61 CT 0x000 0 0x000
- 61
*72 WB 0x000 0 0x000
*71 WB 0x000 0 0x000
*74 EX 0x001 30 0x001
*75 EX 0x010 30 0x001
*76 EX 0x000 0 0x001
+82 0x12002e558 0x00000000 [internal ld/st]
*82 DA 0x000 0 0x000
*79 DA 0x000 0 0x000
*80 DA 0x000 0 0x000
*81 DA 0x000 0 0x000
....more lines.....
<sim_num_insn>        55
<sim_cycle>        154
<sim_IPC>      0.3571
```

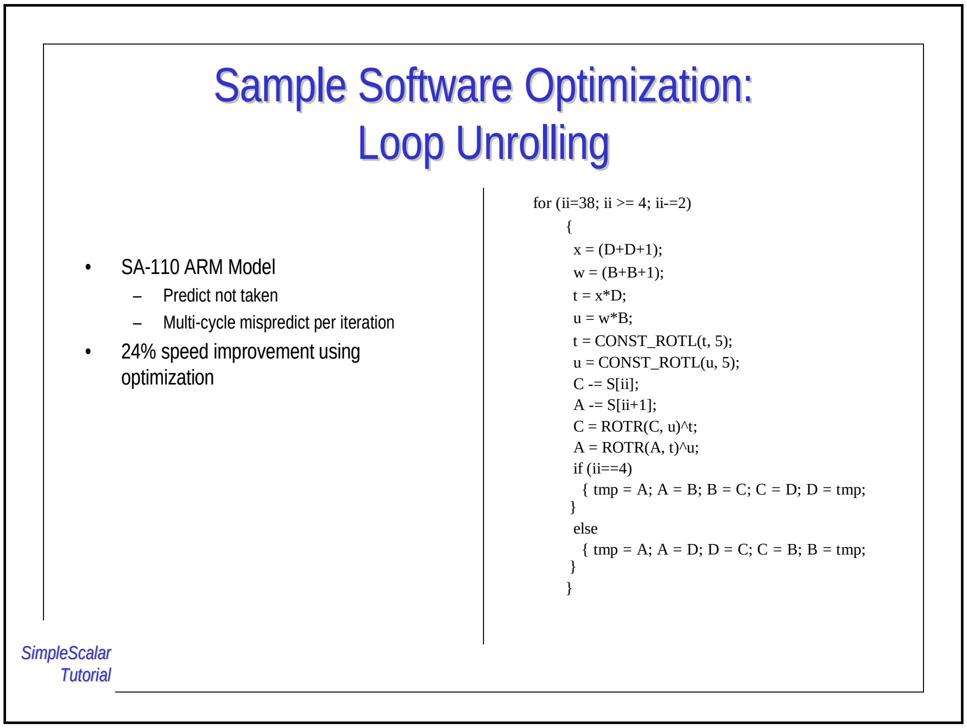
Variables that the user want to track at in <> with the value

```
@155
*76 WB 0x000 0 0x000
*75 WB 0x000 0 0x000
*78 EX 0x001 29 0x001
*79 EX 0x010 29 0x001
*80 EX 0x000 0 0x001
+86 0x12002e558 0x00000000 [internal ld/st]
*86 DA 0x000 0 0x000
*83 DA 0x000 0 0x000
+87 0x12002e558 0x00000000 ldq r1,0(r19)
*87 IF 0x000 0 0x001
+88 0x12002e55c 0x00000000 addq r19,8,r19
*88 IF 0x000 0 0x001
<sim_num_insn>        56
<sim_cycle>        155
<sim_IPC>      0.3613

<END VISUAL>
```

The + sign indicates a new instruction

SimpleScalar Tutorial



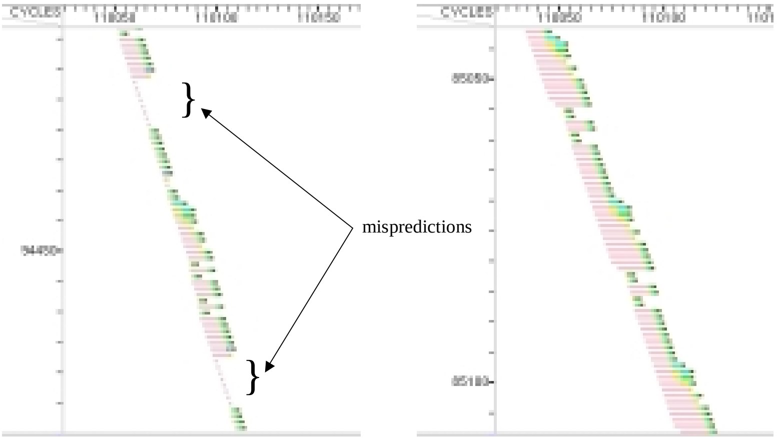
Sample Software Optimization: Loop Unrolling

- SA-110 ARM Model
  - Predict not taken
  - Multi-cycle mispredict per iteration
- 24% speed improvement using optimization

```
for (ii=38; ii >= 4; ii-=2)
    {
      x = (D+D+1);
      w = (B+B+1);
      t = x*D;
      u = w*B;
      t = CONST_ROTL(t, 5);
      u = CONST_ROTL(u, 5);
      C -= S[ii];
      A -= S[ii+1];
      C = ROTR(C, u)^t;
      A = ROTR(A, t)^u;
      if (ii==4)
        { tmp = A; A = B; B = C; C = D; D = tmp;
      }
      else
        { tmp = A; A = D; D = C; C = B; B = tmp;
      }
    }
```

SimpleScalar Tutorial

Base vs. Optimized



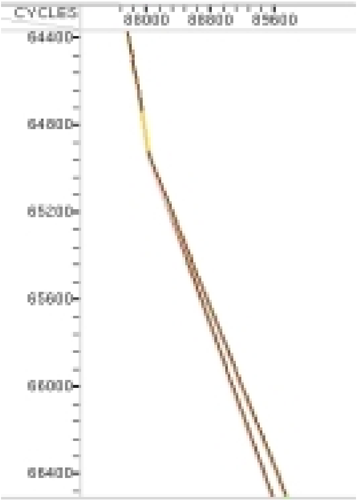
mispredictions

Sample H/W Optimization Add a Multiplier

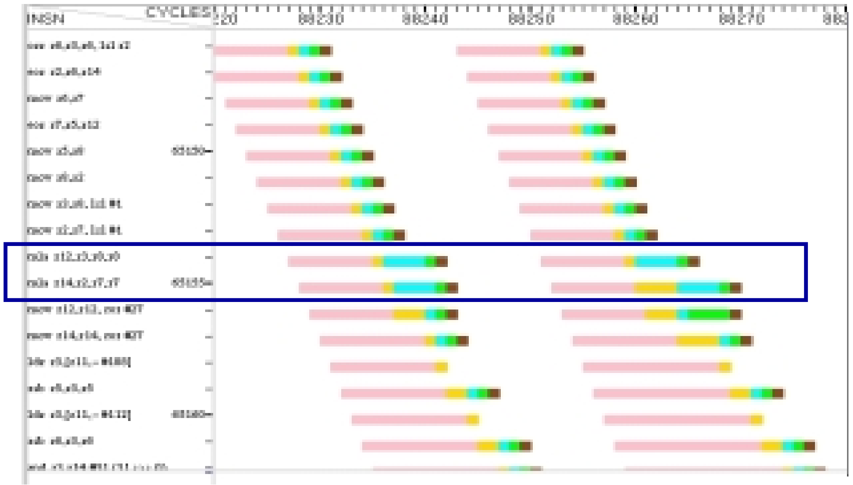
- RC6 does back to back multiplies per iteration
- 4 cycles per multiply on SA-110
- Add Second Multiplier and reschedule code
- 30% speed improvement using optimization

```
for (ii=38; ii >= 4; ii-=2)
       {
         x = (D+D+1);
         w = (B+B+1);
         t = x*D;
         u = w*B;
         t = CONST_ROTL(t, 5);
          u = CONST_ROTL(u, 5);
         C -= S[ii];
         A -= S[ii+1];
         C = ROTR(C, u)^t;
         A = ROTR(A, t)^u;
         if (ii==4)
           { tmp = A; A = B; B = C; C = D; D = tmp;
        }
         else
           { tmp = A; A = D; D = C; C = B; B = tmp;
        }
       }
```

Multiplier Optimization

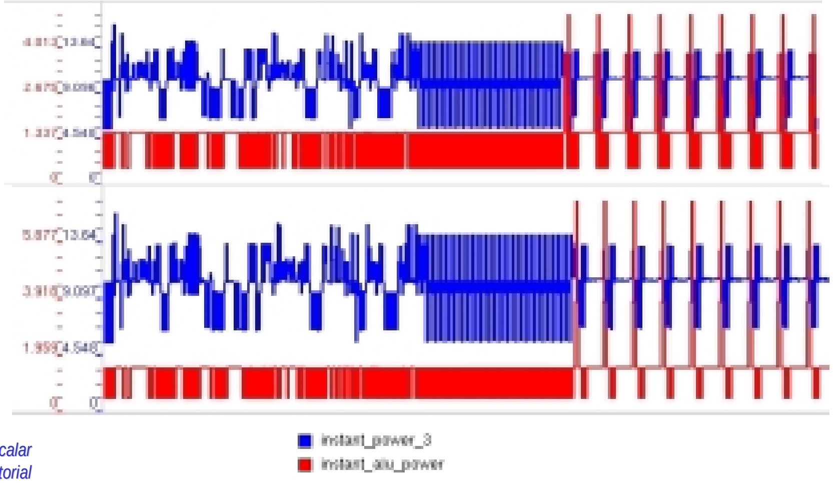


Multiplier Optimization (zoom)



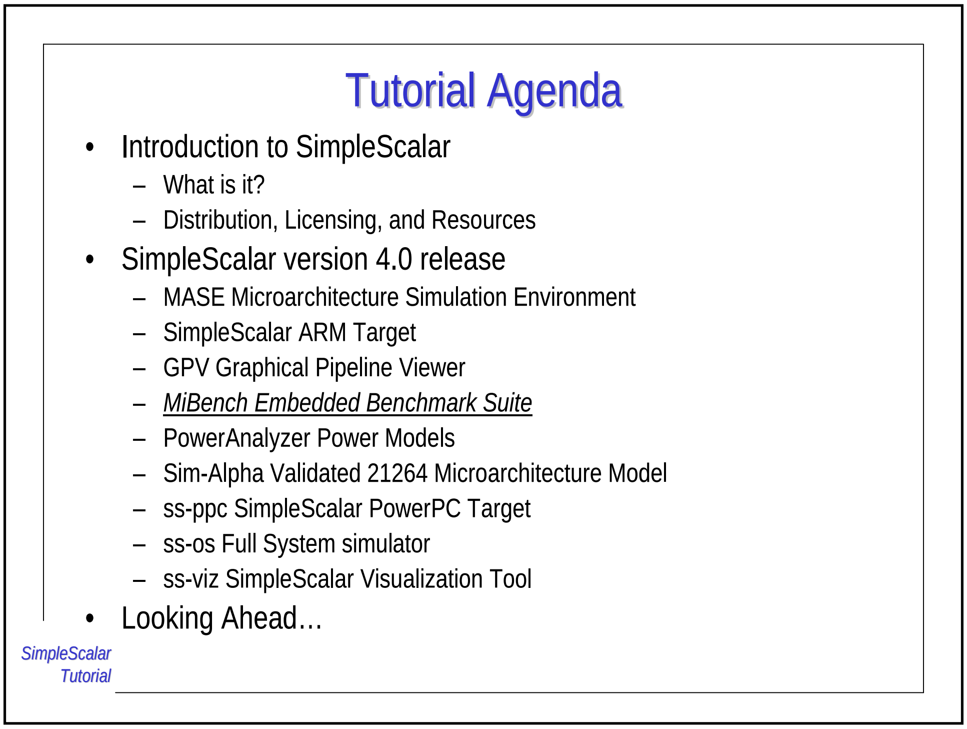
Power usage

(one multiplier top vs two multipliers bottom)



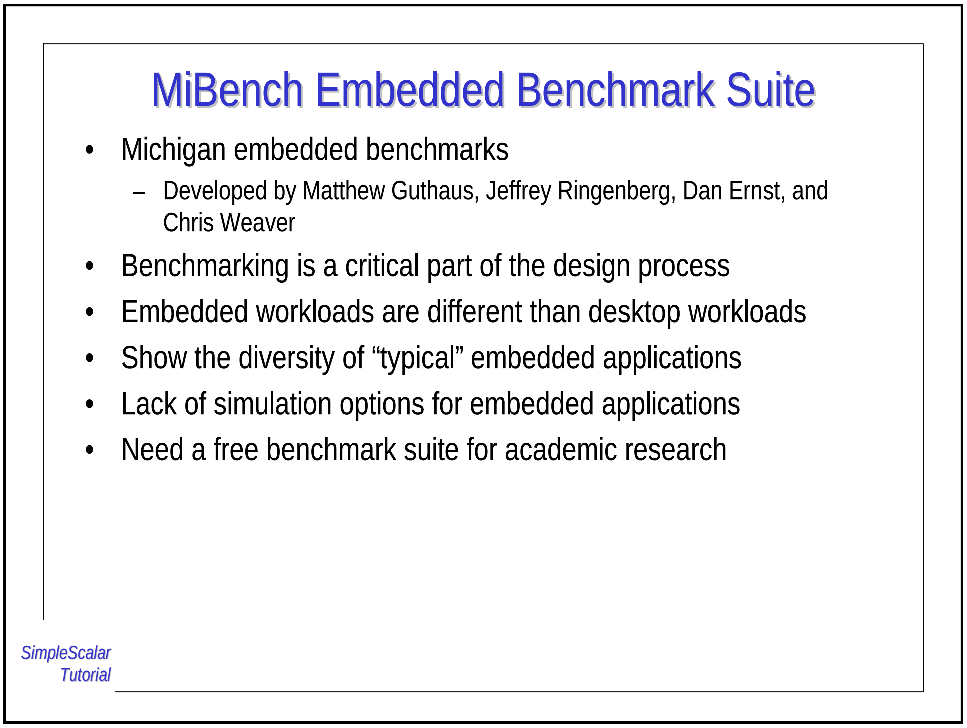
Key Features Summary

- Visualization speeds the process of locating and diagnosing performance bottlenecks
  - Instruction view identifies program slow downs
  - Resource view can be used to locate resource bottlenecks and/or display useful statistics for pipeline analysis
- GPV realized these benefits in an easy to use and portable package



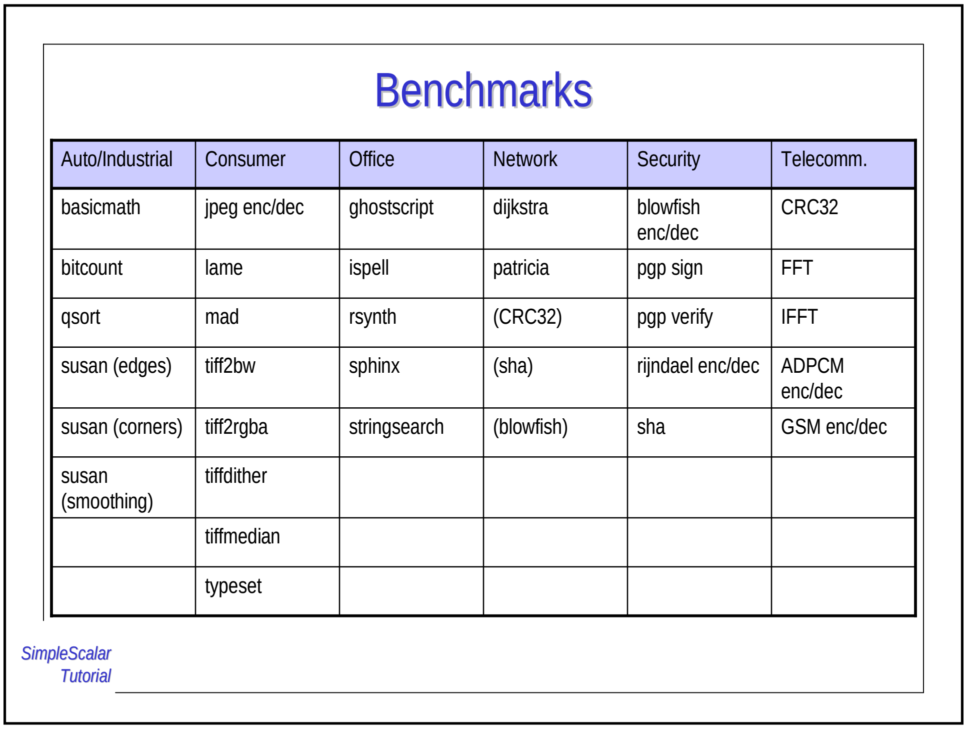
Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - *MiBench Embedded Benchmark Suite*
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…



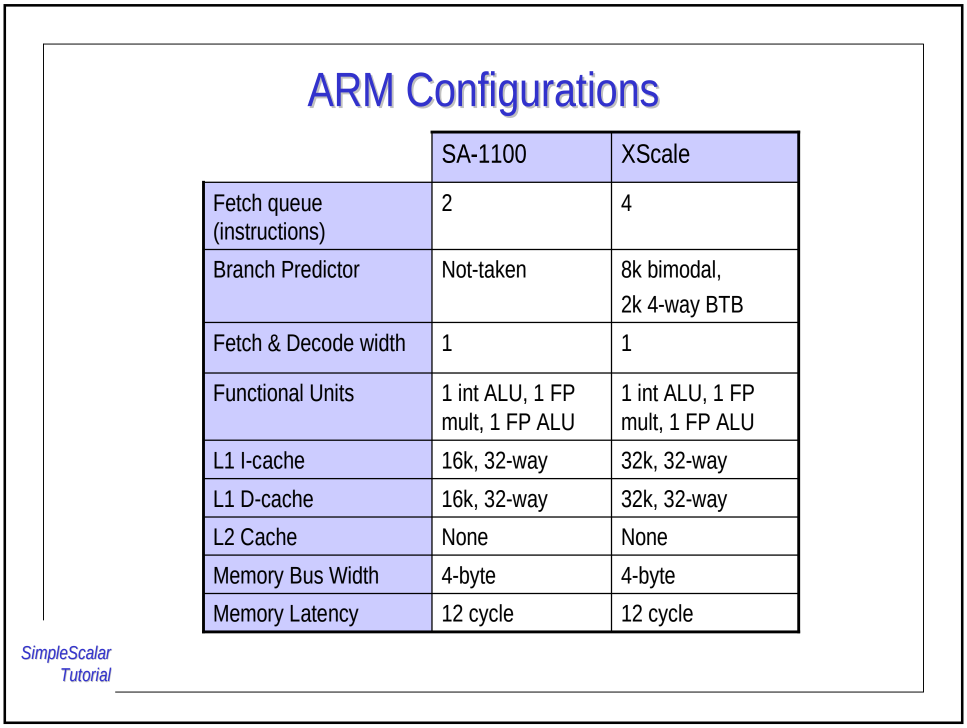
MiBench Embedded Benchmark Suite

- Michigan embedded benchmarks
  - Developed by Matthew Guthaus, Jeffrey Ringenberg, Dan Ernst, and Chris Weaver
- Benchmarking is a critical part of the design process
- Embedded workloads are different than desktop workloads
- Show the diversity of “typical” embedded applications
- Lack of simulation options for embedded applications
- Need a free benchmark suite for academic research



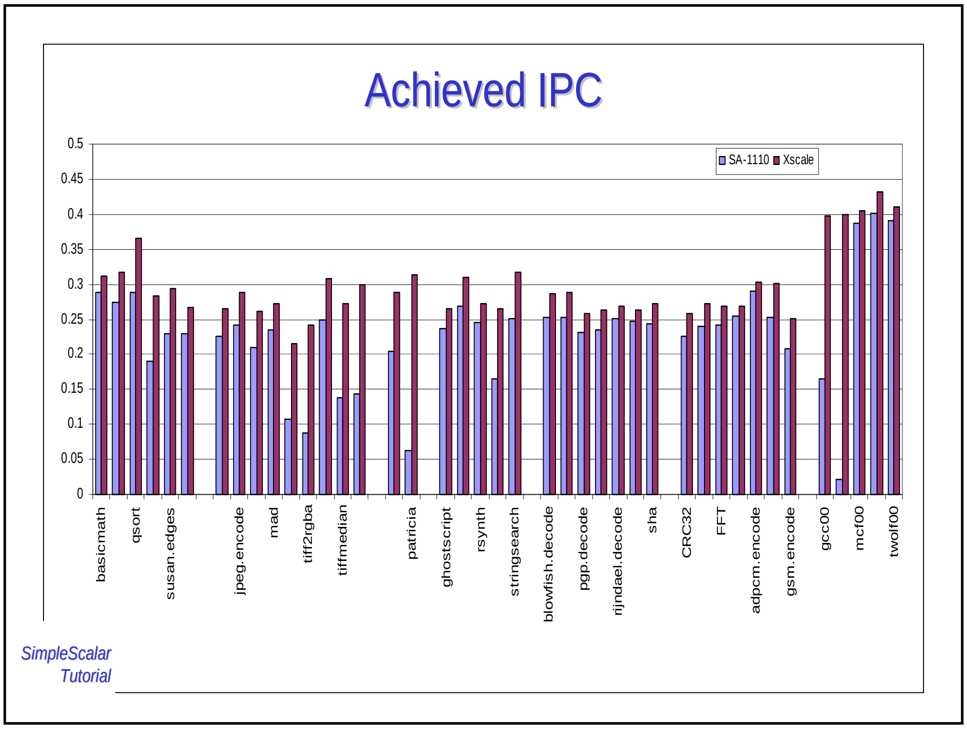
Benchmarks

Auto/Industrial	Consumer	Office	Network	Security	Telecomm.
basicmath	jpeg enc/dec	ghostscript	dijkstra	blowfish enc/dec	CRC32
bitcount	lame	ispell	patricia	pgp sign	FFT
qsort	mad	rsynth	(CRC32)	pgp verify	IFFT
susan (edges)	tiff2bw	sphinx	(sha)	rijndael enc/dec	ADPCM enc/dec
susan (corners)	tiff2rgba	stringsearch	(blowfish)	sha	GSM enc/dec
susan (smoothing)	tiffdither				
	tiffmedian				
	typeset				



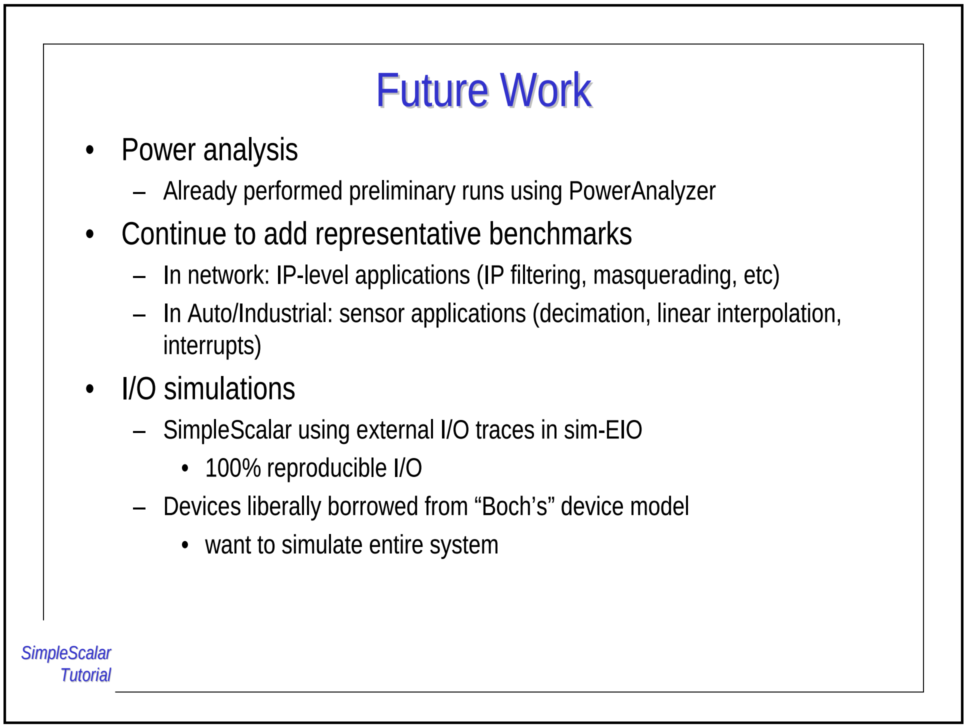
ARM Configurations

	SA-1100	XScale
Fetch queue (instructions)	2	4
Branch Predictor	Not-taken	8k bimodal, 2k 4-way BTB
Fetch & Decode width	1	1
Functional Units	1 int ALU, 1 FP mult, 1 FP ALU	1 int ALU, 1 FP mult, 1 FP ALU
L1 I-cache	16k, 32-way	32k, 32-way
L1 D-cache	16k, 32-way	32k, 32-way
L2 Cache	None	None
Memory Bus Width	4-byte	4-byte
Memory Latency	12 cycle	12 cycle



Achieved IPC

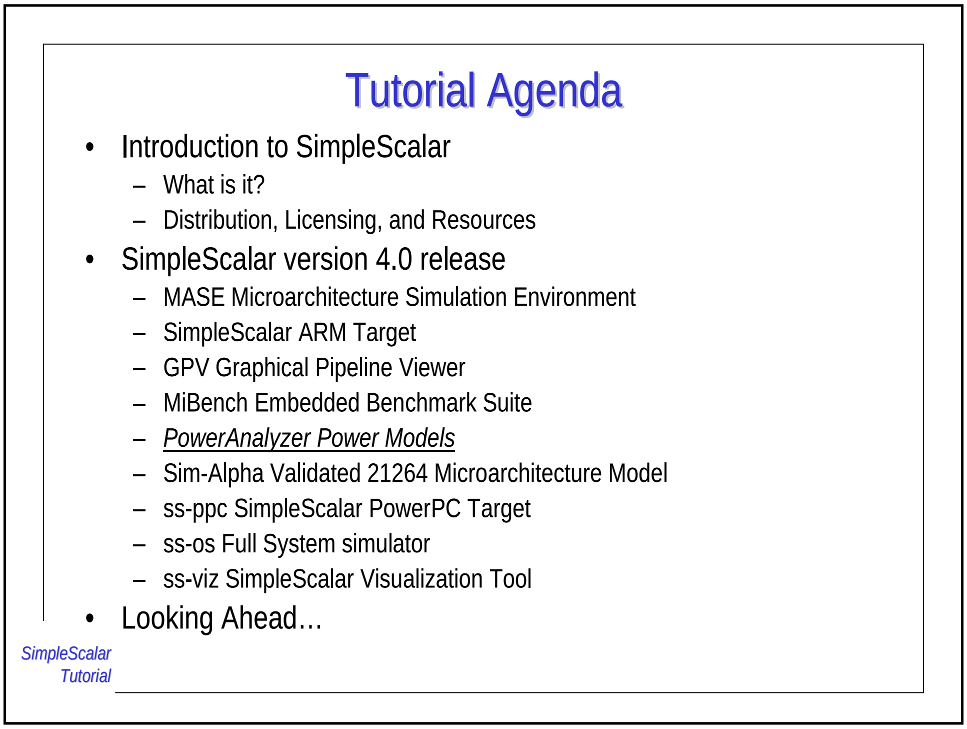
SimpleScalar Tutorial



Future Work

- Power analysis
  - Already performed preliminary runs using PowerAnalyzer
- Continue to add representative benchmarks
  - In network: IP-level applications (IP filtering, masquerading, etc)
  - In Auto/Industrial: sensor applications (decimation, linear interpolation, interrupts)
- I/O simulations
  - SimpleScalar using external I/O traces in sim-EIO
    - 100% reproducible I/O
  - Devices liberally borrowed from “Boch’s” device model
    - want to simulate entire system

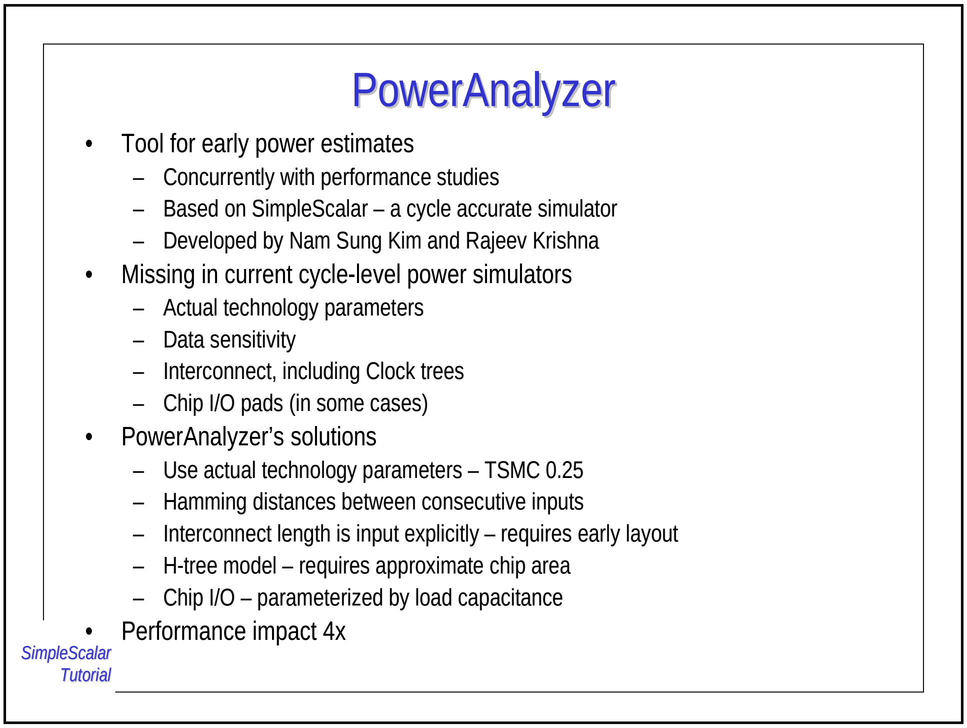
SimpleScalar Tutorial



Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - *PowerAnalyzer Power Models*
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…

SimpleScalar Tutorial



PowerAnalyzer

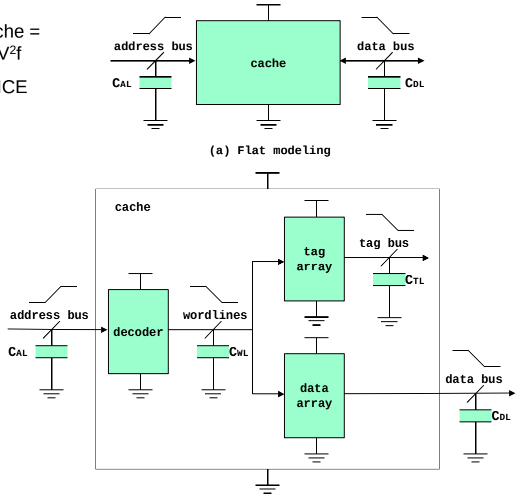
- Tool for early power estimates
  - Concurrently with performance studies
  - Based on SimpleScalar – a cycle accurate simulator
  - Developed by Nam Sung Kim and Rajeev Krishna
- Missing in current cycle-level power simulators
  - Actual technology parameters
  - Data sensitivity
  - Interconnect, including Clock trees
  - Chip I/O pads (in some cases)
- PowerAnalyzer's solutions
  - Use actual technology parameters – TSMC 0.25
  - Hamming distances between consecutive inputs
  - Interconnect length is input explicitly – requires early layout
  - H-tree model – requires approximate chip area
  - Chip I/O – parameterized by load capacitance
- Performance impact 4x

SimpleScalar Tutorial

Modeling µ architectural Blocks

Effective capacitance of cache = (average power of access)/V^2f

Power calculated with HSPICE and CACTI II

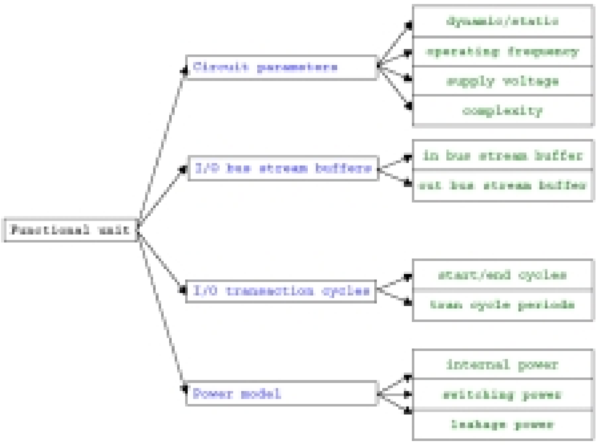


(a) Flat modeling

(b) Hierarchical modeling

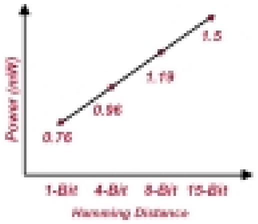
PowerAnalyzer

- Data structure for blocks (simplified)

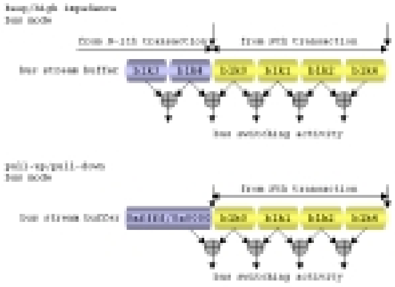


Data sensitivity

8 bit ALU
at 100 MHz



Data sensitivity
on buses

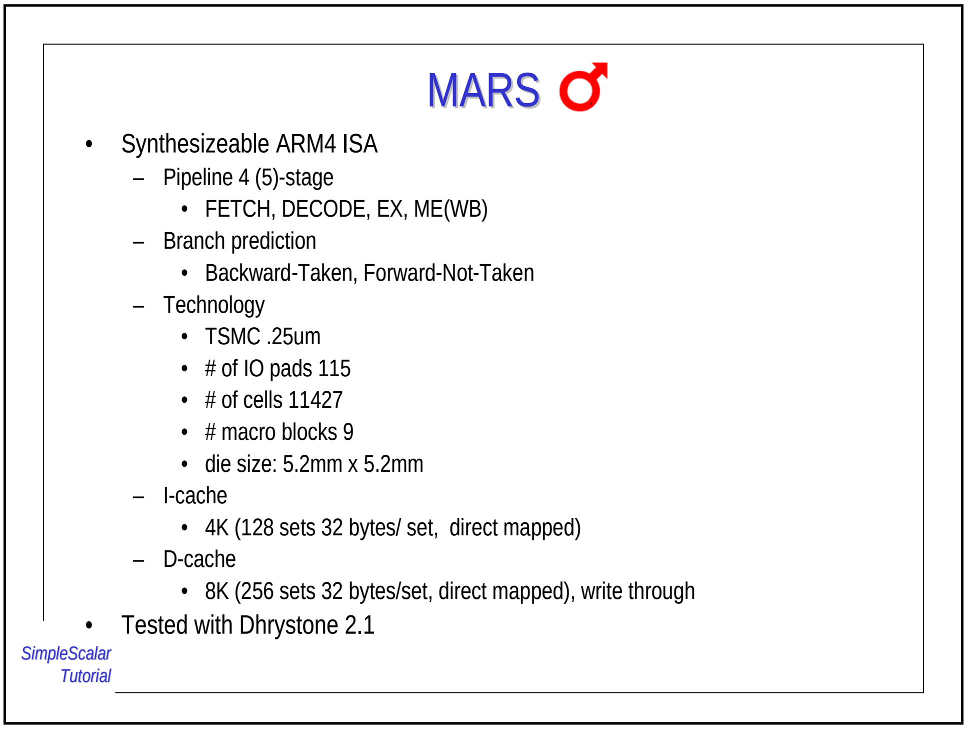


SimpleScalar Tutorial

PowerAnalyzer

- Automatic configuration:
  - Approximate layout – interconnect and clock tree
  - Leakage – total gate width/block (or number of equivalent inverters)
  - Gate count estimation of random logic
- Calibrate against MARS
- Next set of experiments
  - What can we leave out vs technology
    - Interconnect
    - Hierarchy
    - Pads
    - Data sensitivity
    - Leakage
  - Impact on performance of PowerAnalyzer
  - Impact on accuracy of PowerAnalyzer
- Future experiments
  - Microarchitecture power/performance

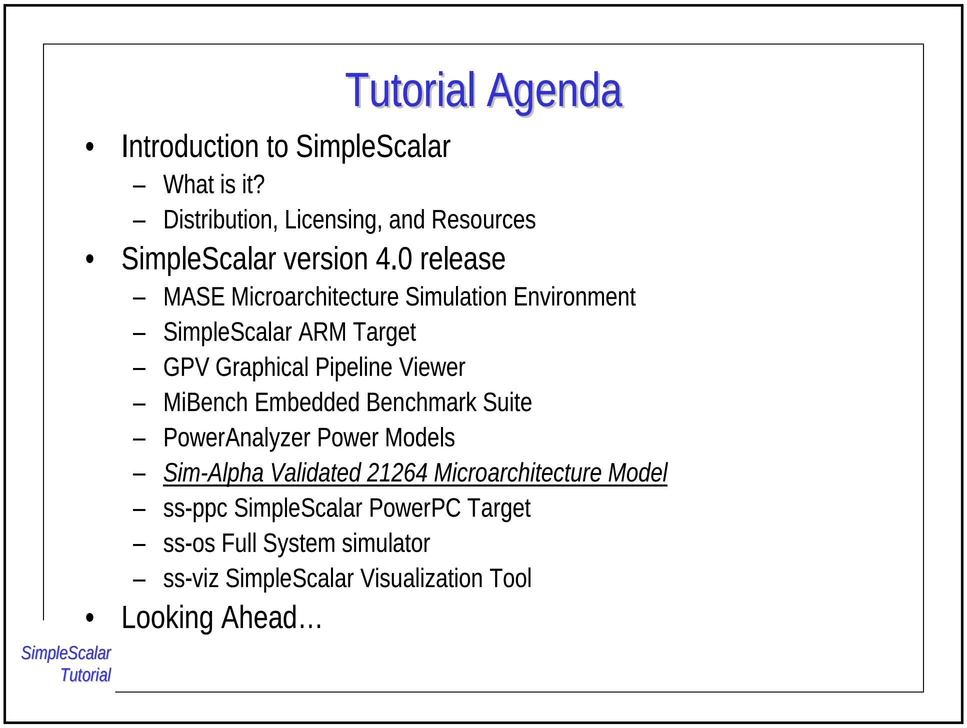
SimpleScalar Tutorial



MARS ♂

- Synthesizeable ARM4 ISA
  - Pipeline 4 (5)-stage
    - FETCH, DECODE, EX, ME(WB)
  - Branch prediction
    - Backward-Taken, Forward-Not-Taken
  - Technology
    - TSMC .25um
    - # of IO pads 115
    - # of cells 11427
    - # macro blocks 9
    - die size: 5.2mm x 5.2mm
  - I-cache
    - 4K (128 sets 32 bytes/ set, direct mapped)
  - D-cache
    - 8K (256 sets 32 bytes/set, direct mapped), write through
- Tested with Dhrystone 2.1

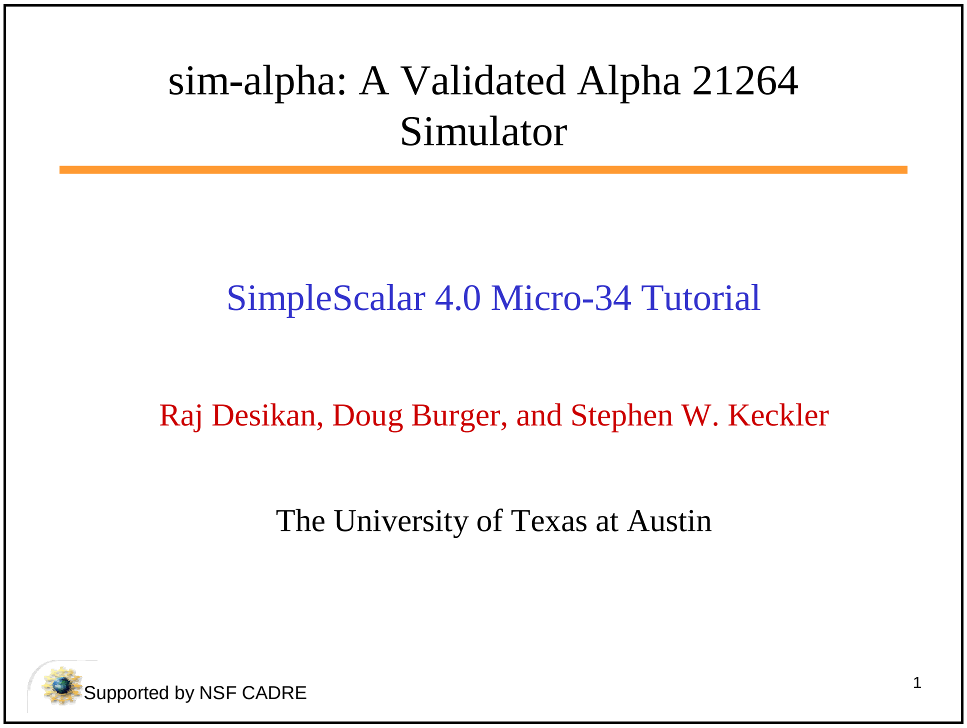
SimpleScalar Tutorial



Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - *Sim-Alpha Validated 21264 Microarchitecture Model*
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…

SimpleScalar Tutorial

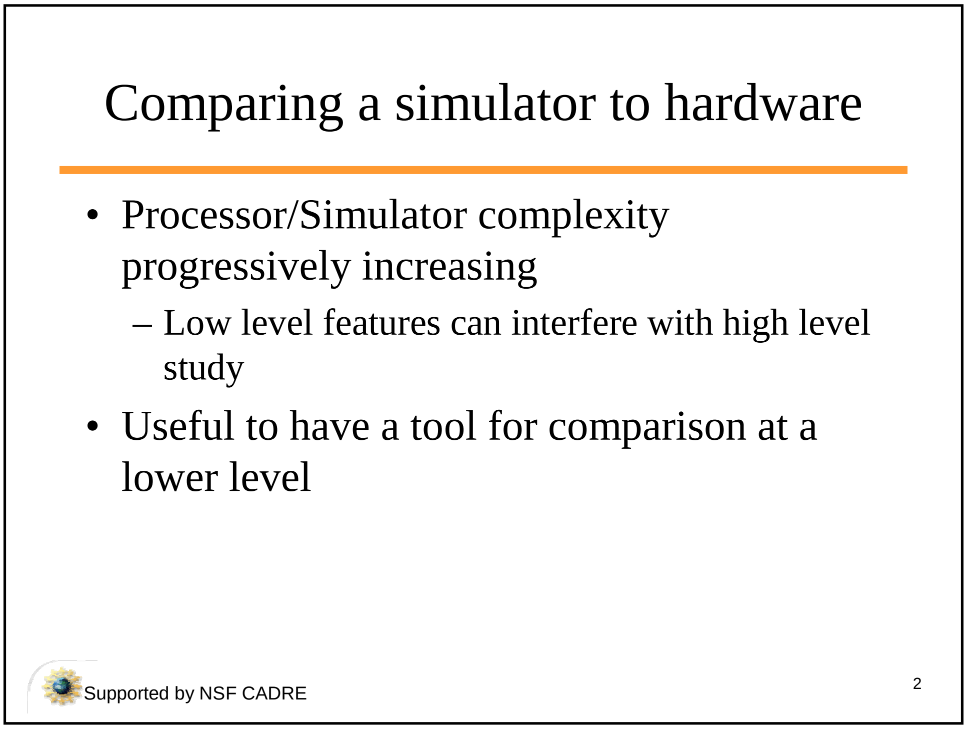


sim-alpha: A Validated Alpha 21264 Simulator

SimpleScalar 4.0 Micro-34 Tutorial

Raj Desikan, Doug Burger, and Stephen W. Keckler

The University of Texas at Austin



Comparing a simulator to hardware

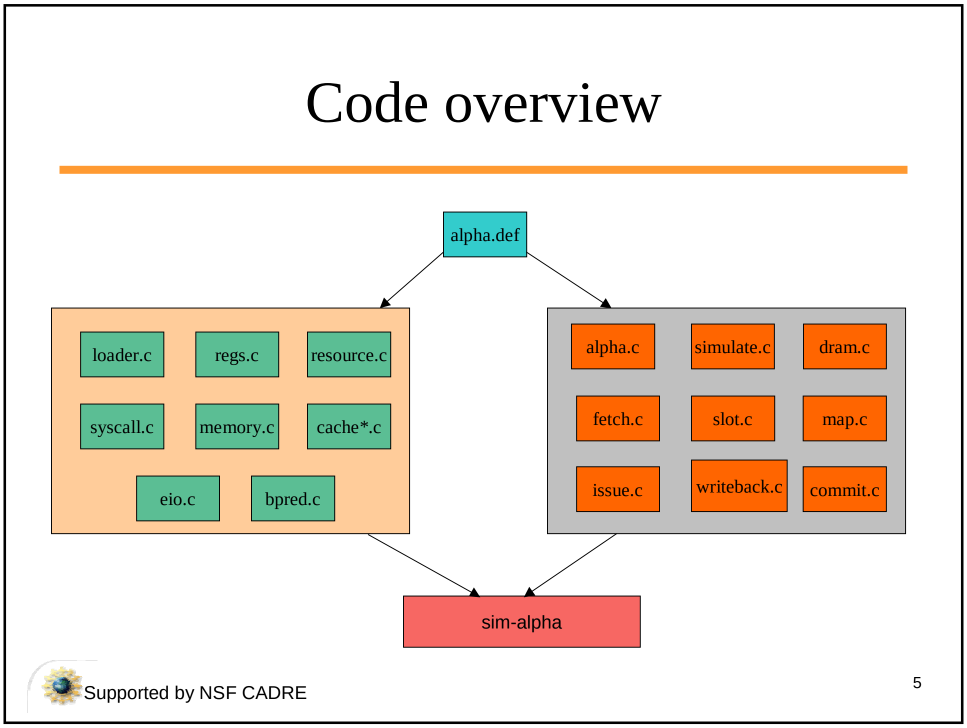
- Processor/Simulator complexity progressively increasing
  - Low level features can interfere with high level study
- Useful to have a tool for comparison at a lower level

The sim-alpha goals

- Extend the SimpleScalar tool set to model an existing microprocessor (EV6 microarchitecture)
- Compare the simulator against actual hardware for accurate modeling
- Release the simulator for use by researchers studying extensions to existing implementations

Using sim-alpha

- make will generate default simulator
- make flexible generates simulator with all bells and whistles
- make functional turns on functional debugger
- sim-alpha –config <config file> binary
- Supports EIO tracing with checkpointing



Code overview

alpha.def

loader.c regs.c resource.c

syscall.c memory.c cache*.c

eio.c bpred.c

alpha.c simulate.c dram.c

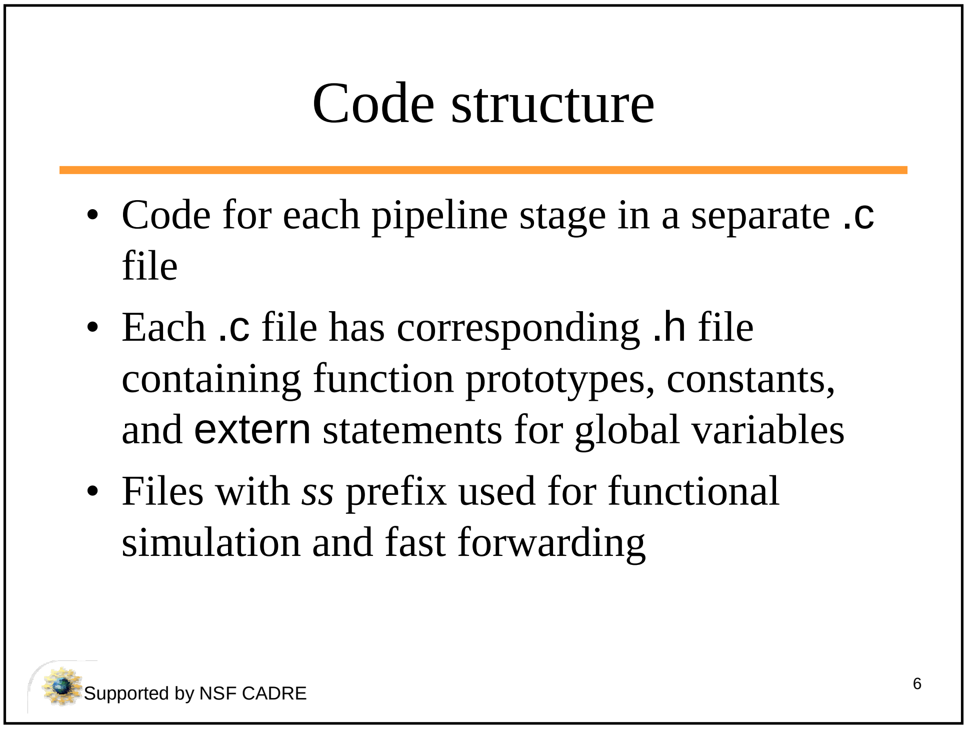
fetch.c slot.c map.c

issue.c writeback.c commit.c

sim-alpha

Supported by NSF CADRE

5

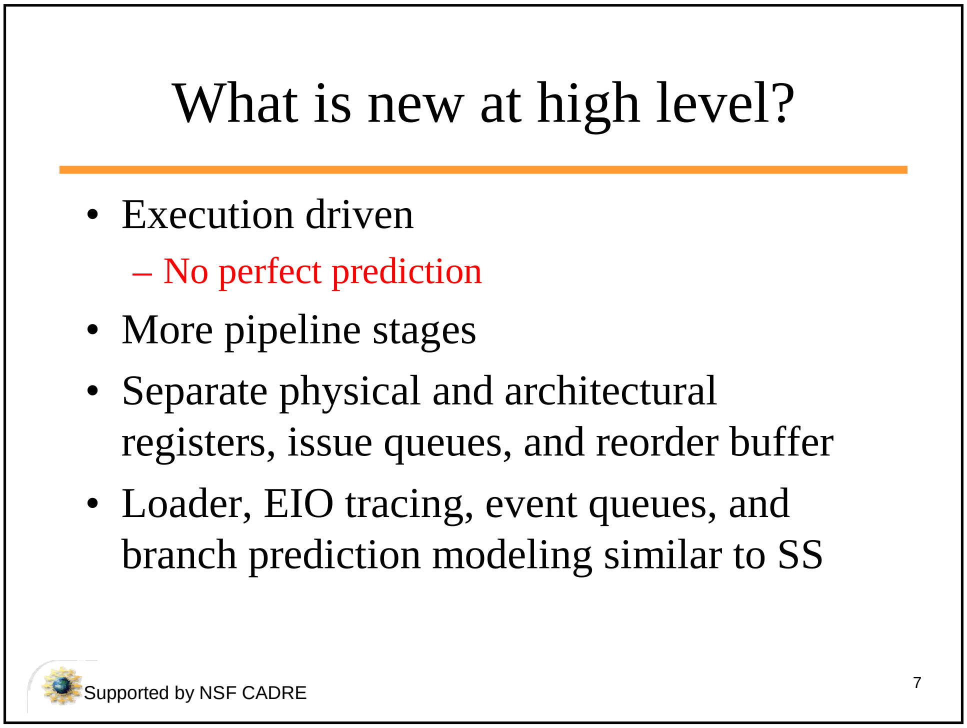


Code structure

- Code for each pipeline stage in a separate .c file
- Each .c file has corresponding .h file containing function prototypes, constants, and extern statements for global variables
- Files with *ss* prefix used for functional simulation and fast forwarding

Supported by NSF CADRE

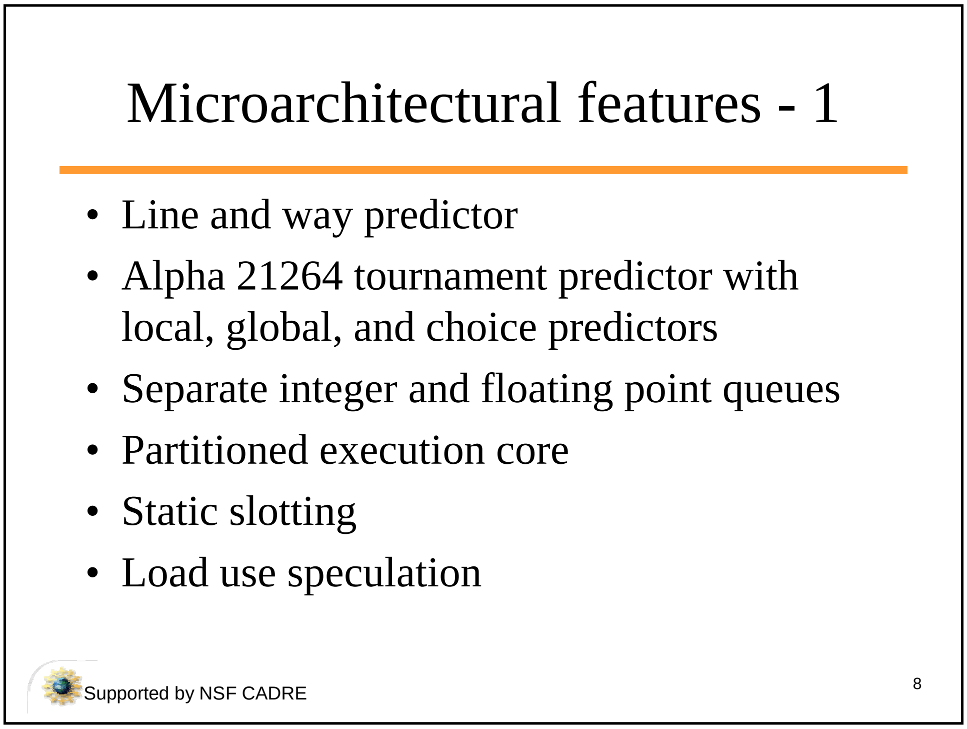
6



What is new at high level?

- Execution driven
  - No perfect prediction
- More pipeline stages
- Separate physical and architectural registers, issue queues, and reorder buffer
- Loader, EIO tracing, event queues, and branch prediction modeling similar to SS

Supported by NSF CADRE



Microarchitectural features - 1

- Line and way predictor
- Alpha 21264 tournament predictor with local, global, and choice predictors
- Separate integer and floating point queues
- Partitioned execution core
- Static slotting
- Load use speculation

Supported by NSF CADRE

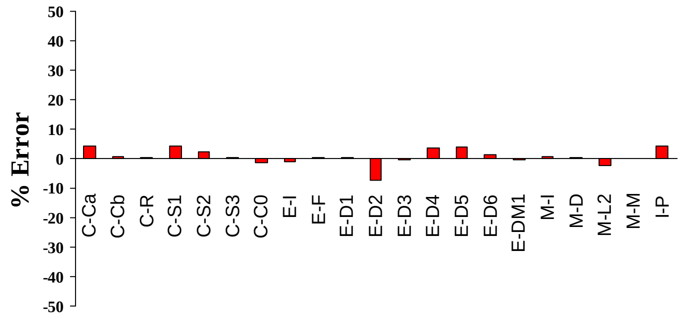
Microarchitectural features - 2

- Separate load and store queues
- Non-homogenous functional units
- Different memory traps
  - – Load-Load trap
  - – Load-Store trap
  - – Mbox traps
- Early instruction retire
- stWait table

Supported by NSF CADRE

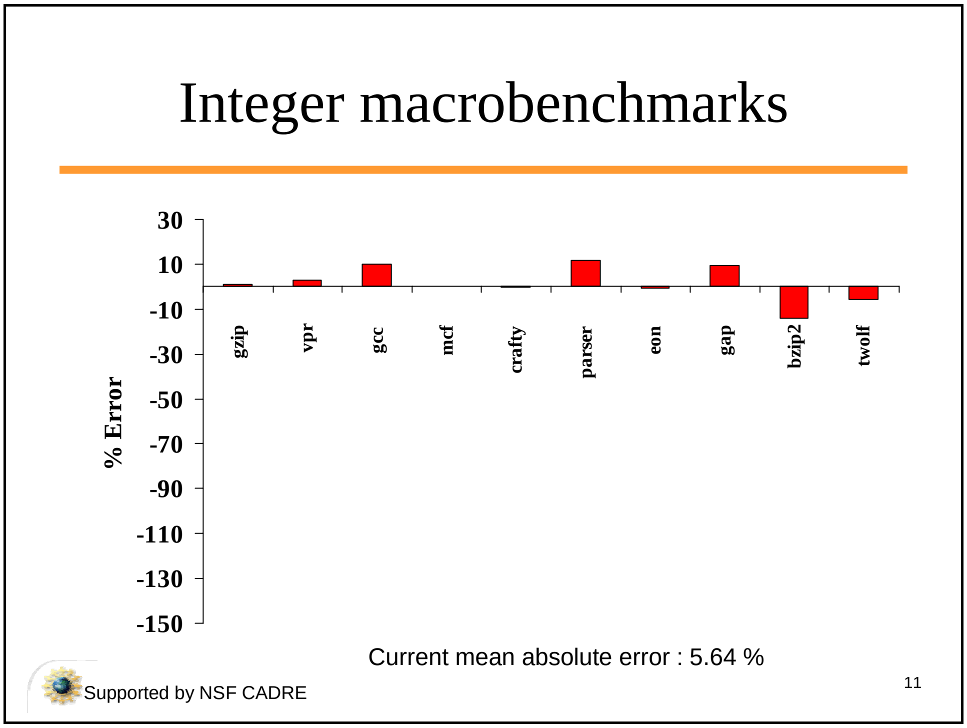
Microbenchmark results

$$\% \text{ Error} = \frac{(\text{Native cycles} - \text{Simulator cycles})*100}{\text{Native cycles}}$$



Current mean absolute error : 1.7 %

Supported by NSF CADRE

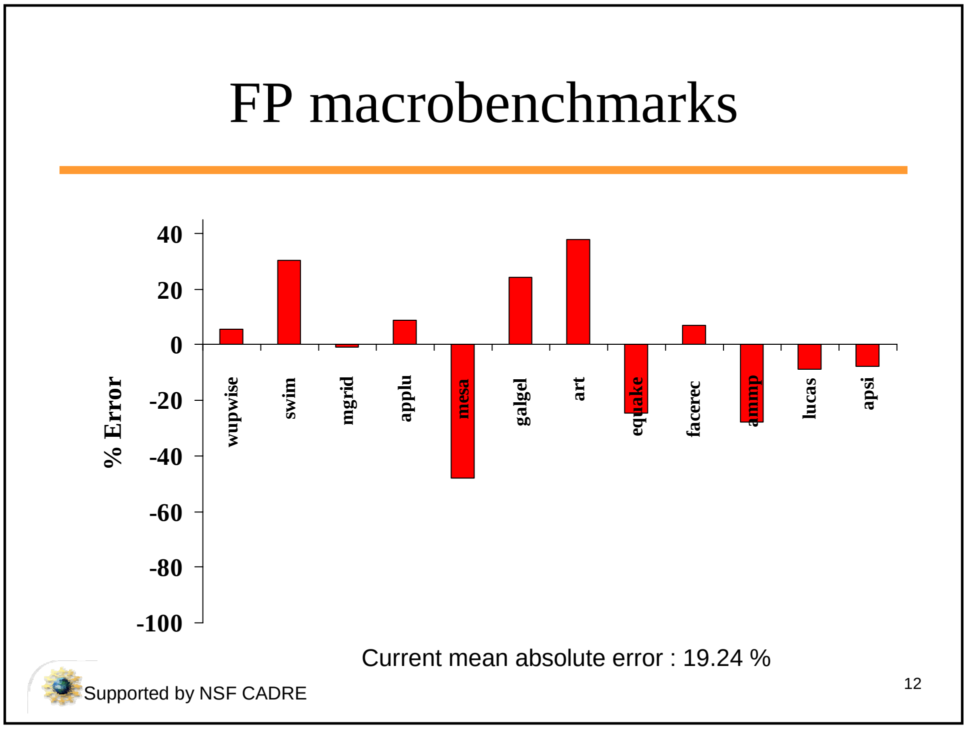


Integer macrobenchmarks

Current mean absolute error : 5.64 %

Supported by NSF CADRE

11

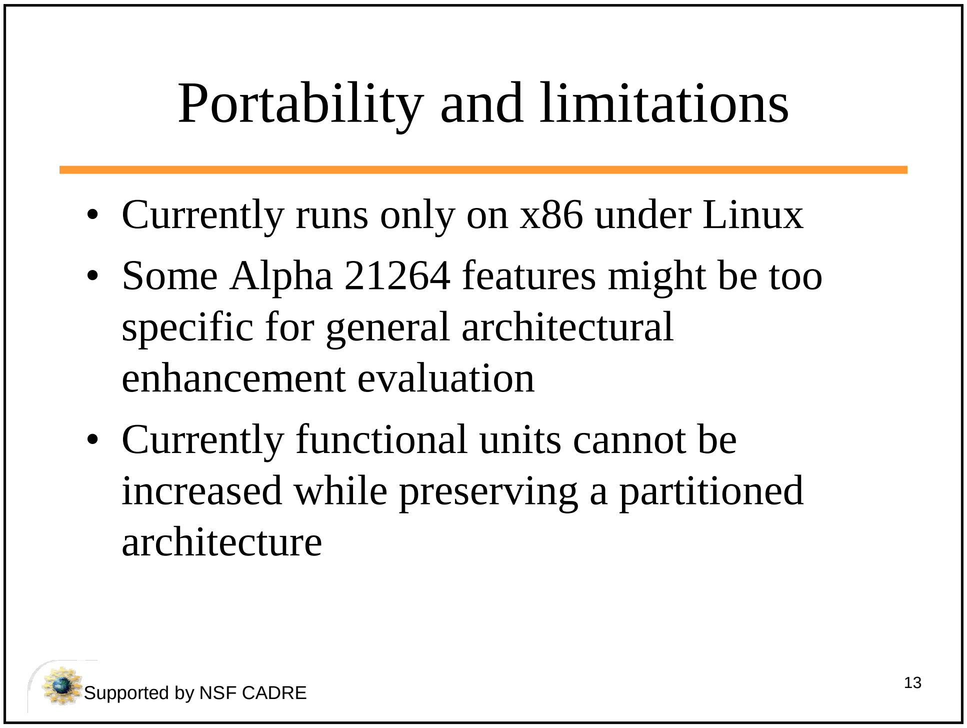


FP macrobenchmarks

Current mean absolute error : 19.24 %

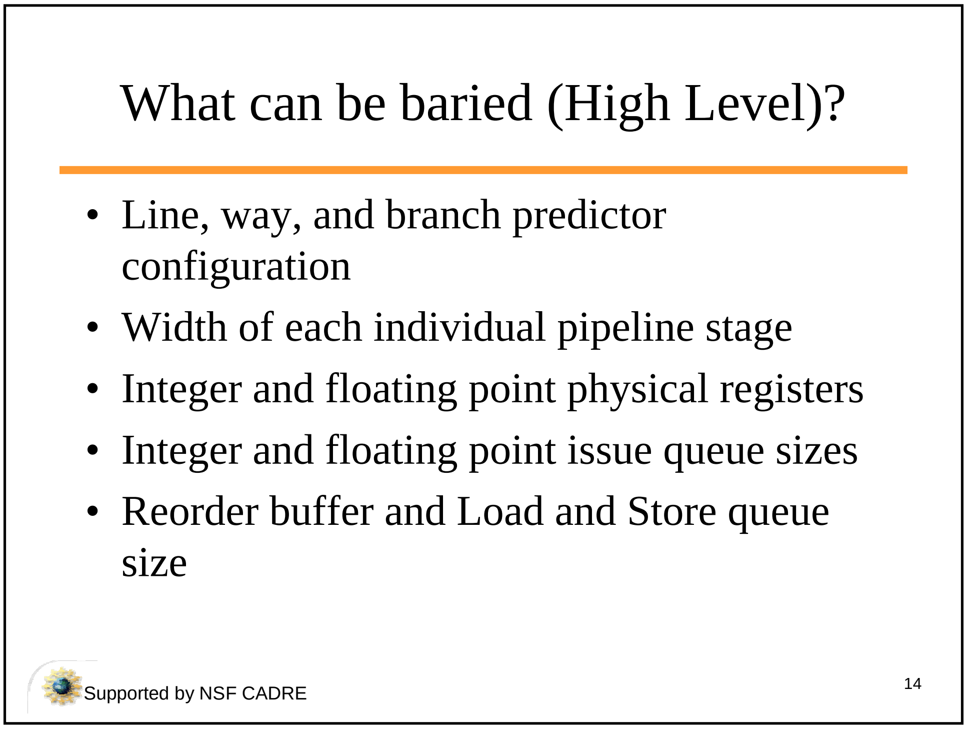
Supported by NSF CADRE

12



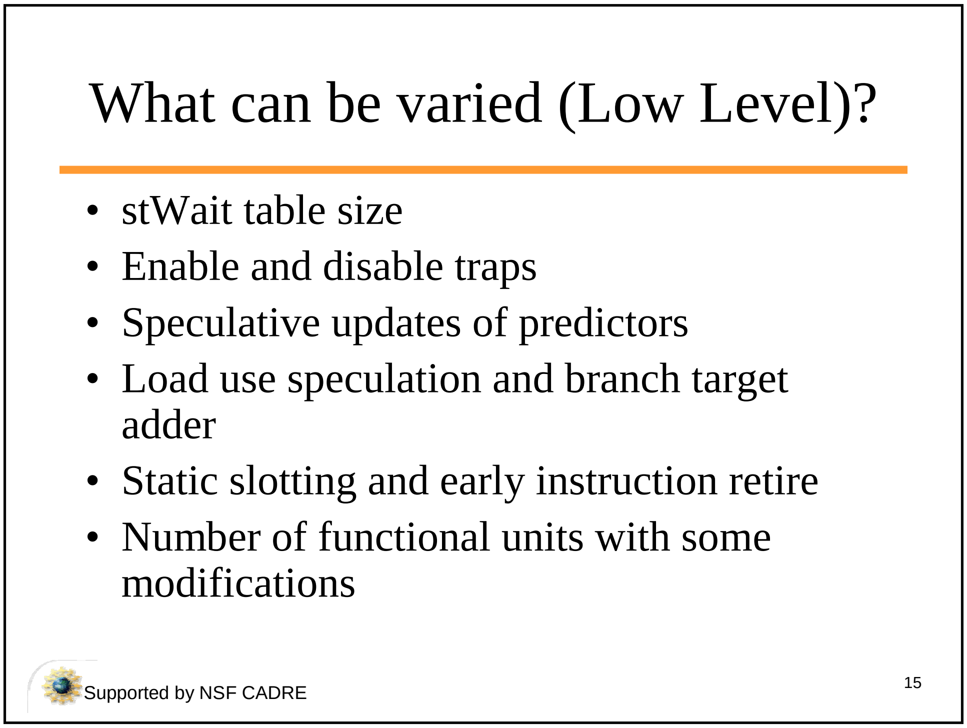
Portability and limitations

- Currently runs only on x86 under Linux
- Some Alpha 21264 features might be too specific for general architectural enhancement evaluation
- Currently functional units cannot be increased while preserving a partitioned architecture



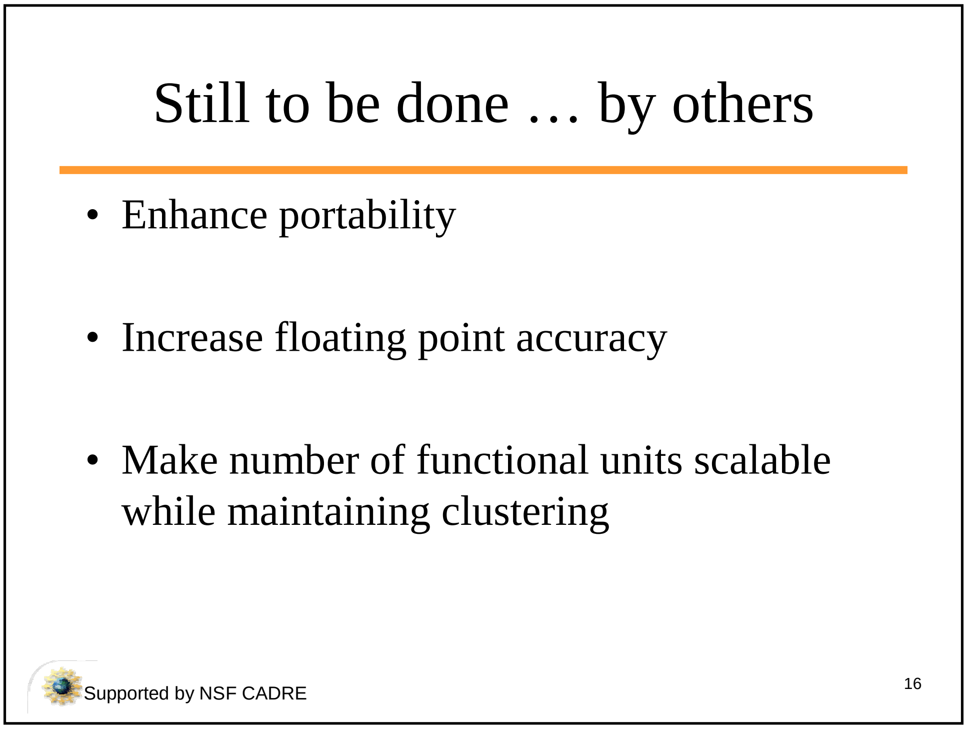
What can be baried (High Level)?

- Line, way, and branch predictor configuration
- Width of each individual pipeline stage
- Integer and floating point physical registers
- Integer and floating point issue queue sizes
- Reorder buffer and Load and Store queue size



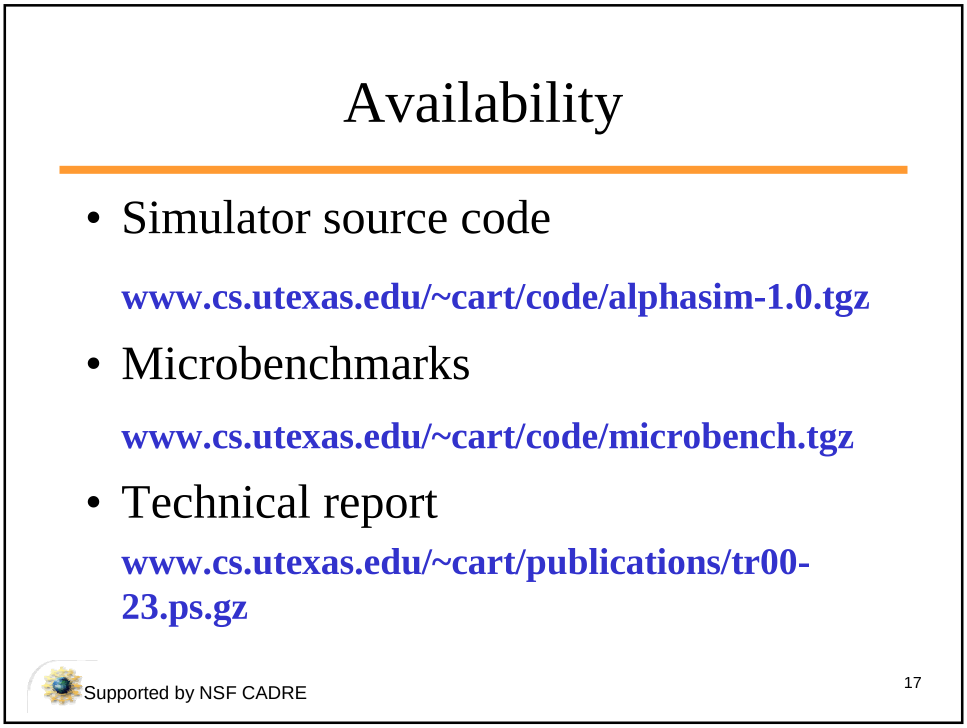
What can be varied (Low Level)?

- stWait table size
- Enable and disable traps
- Speculative updates of predictors
- Load use speculation and branch target adder
- Static slotting and early instruction retire
- Number of functional units with some modifications



Still to be done … by others

- Enhance portability
- Increase floating point accuracy
- Make number of functional units scalable while maintaining clustering



Availability

- Simulator source code

  www.cs.utexas.edu/~cart/code/alphasim-1.0.tgz

- Microbenchmarks

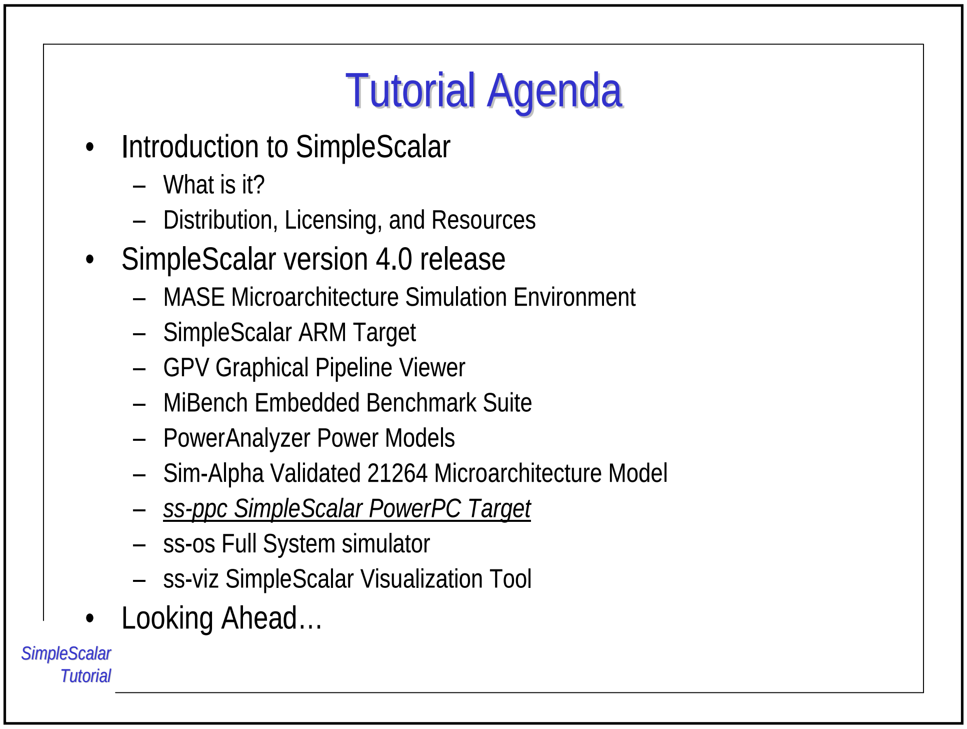
  www.cs.utexas.edu/~cart/code/microbench.tgz

- Technical report

  www.cs.utexas.edu/~cart/publications/tr00-23.ps.gz

Supported by NSF CADRE

17



Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - *ss-ppc SimpleScalar PowerPC Target*
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…

`ss-ppc`

SimpleScalar Simulation of the PowerPC Instruction Set Architecture

SimpleScalar 4.0 Micro34 Tutorial

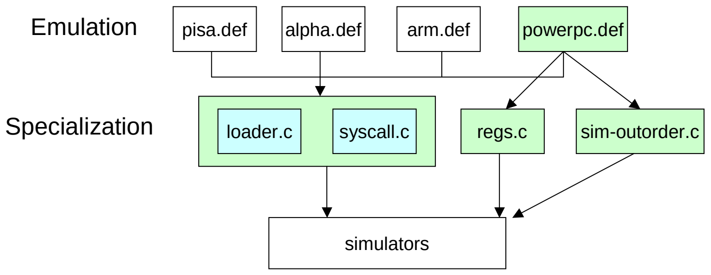
Karu Sankaralingam, Ramadass Nagarajan,
Stephen W. Keckler, Doug Burger

University of Texas at Austin

Supported by NSF CADRE

Overview

- SimpleScalar's port to simulate PowerPC executable files.
- Developed from Version 3.0 code base



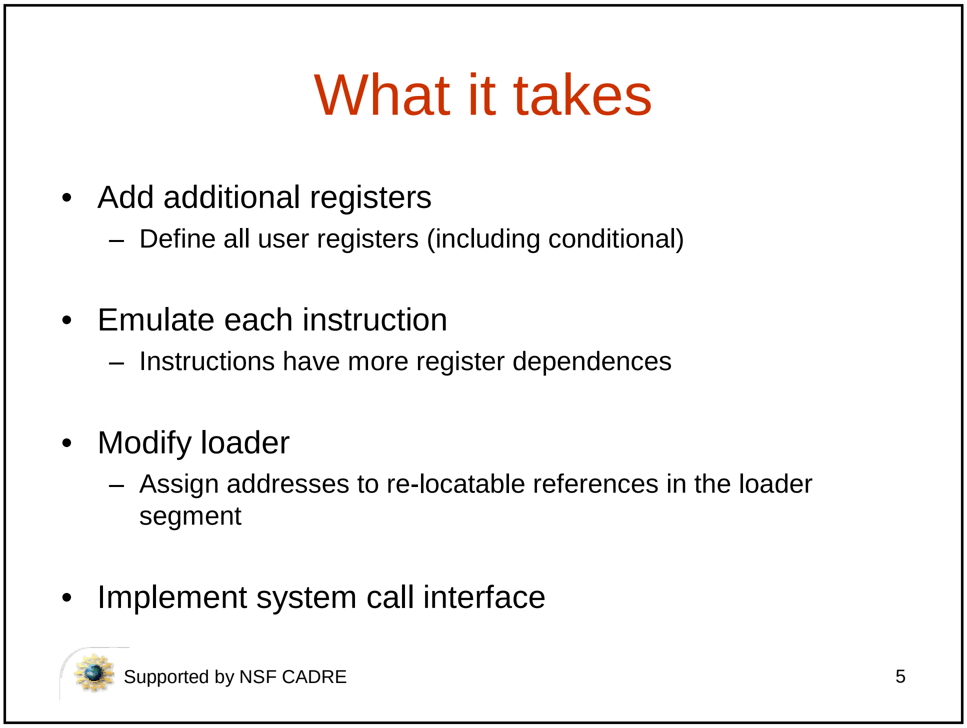
Supported by NSF CADRE

Tools Ported

`sim-fast`	functional simulator
`sim-outorder`	micro-architecture simulator
`sim-eio`	checkpointing and fastforwarding
`sim-profile`	execution profiler
`sim-bpred`	branch prediction simulatior
`sim-cache`	cache simulator
`sim-cheetah`	advanced cache simulator

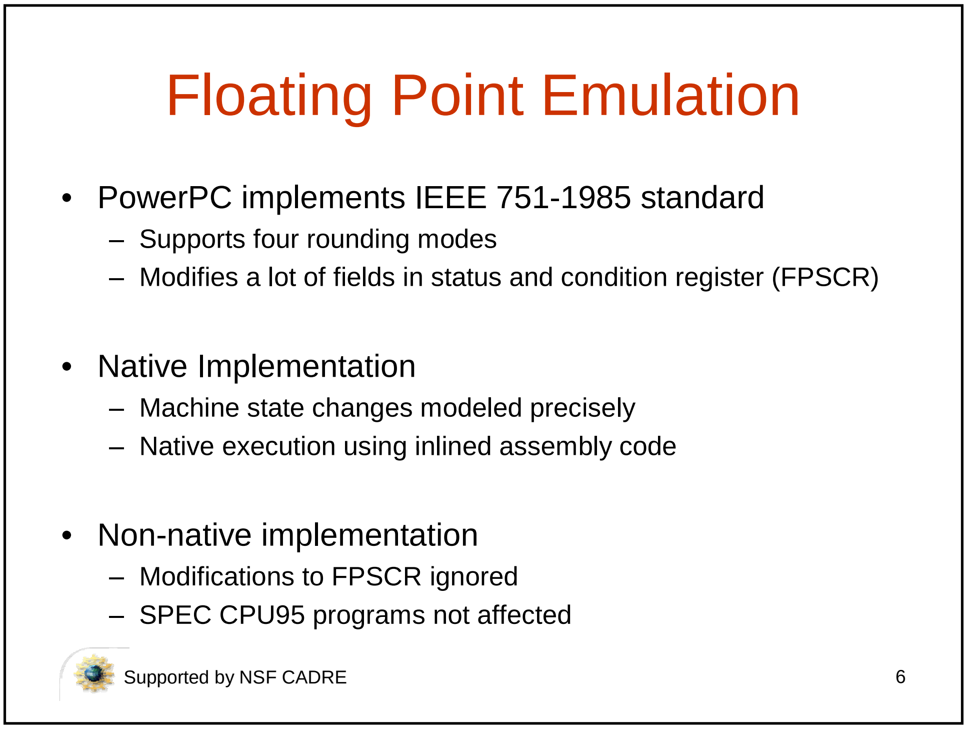
PowerPC ISA

- Instructions
  - 224 instructions in 15 different formats
- Registers
  - 32 GPR, 32 FPR
  - 2 control, 3 condition and exception registers
- Storage model
  - Byte, half-word and word data accesses allowed
  - Misaligned addresses allowed



What it takes

- Add additional registers
  - Define all user registers (including conditional)
- Emulate each instruction
  - Instructions have more register dependences
- Modify loader
  - Assign addresses to re-locatable references in the loader segment
- Implement system call interface



Floating Point Emulation

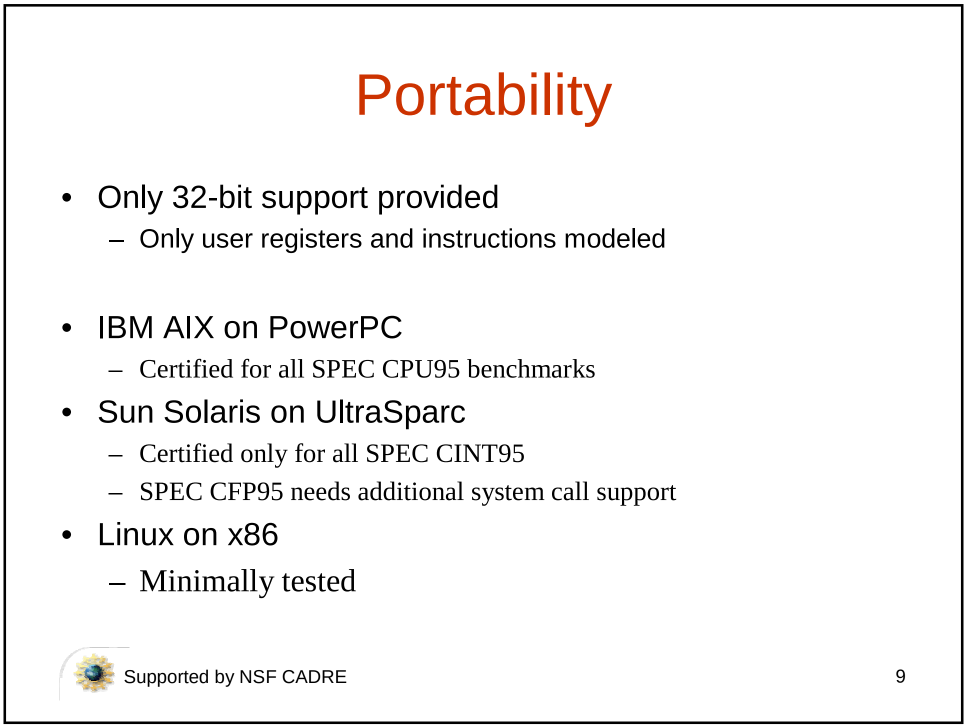
- PowerPC implements IEEE 751-1985 standard
  - Supports four rounding modes
  - Modifies a lot of fields in status and condition register (FPSCR)
- Native Implementation
  - Machine state changes modeled precisely
  - Native execution using inlined assembly code
- Non-native implementation
  - Modifications to FPSCR ignored
  - SPEC CPU95 programs not affected

System calls

- Implemented using corresponding calls on the host machine
- Every syscall is the same sequence of six user instructions
  - Detect using a predecode phase and modify with a special instruction (`sc`)
- Identifying the type of the syscall
  - Loader stores hooks in the TOC

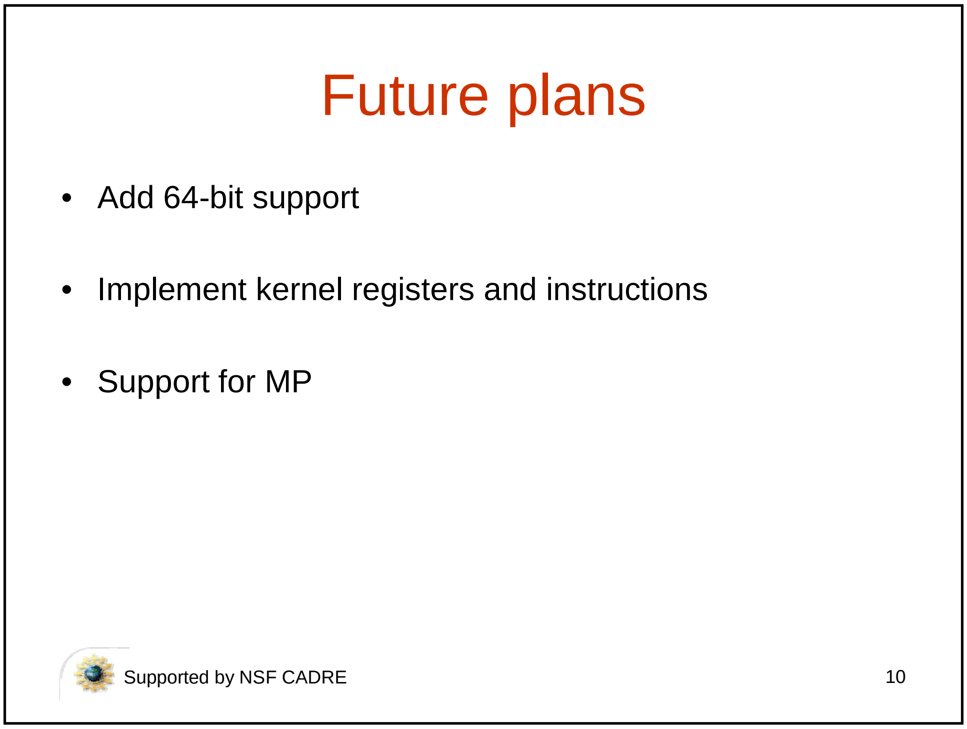
Timing Simulation

- SimpleScalar’s RUU micro-architecture model
- `sim-outorder` port relatively easy
- Implementation issues
  - Stores may update registers
    - passed through writeback stage
  - Load/Store Multiple instructions access multiple words
    - Modeled as atomic operations
  - Memory accesses may be mis-aligned
    - Converted to aligned access(es)



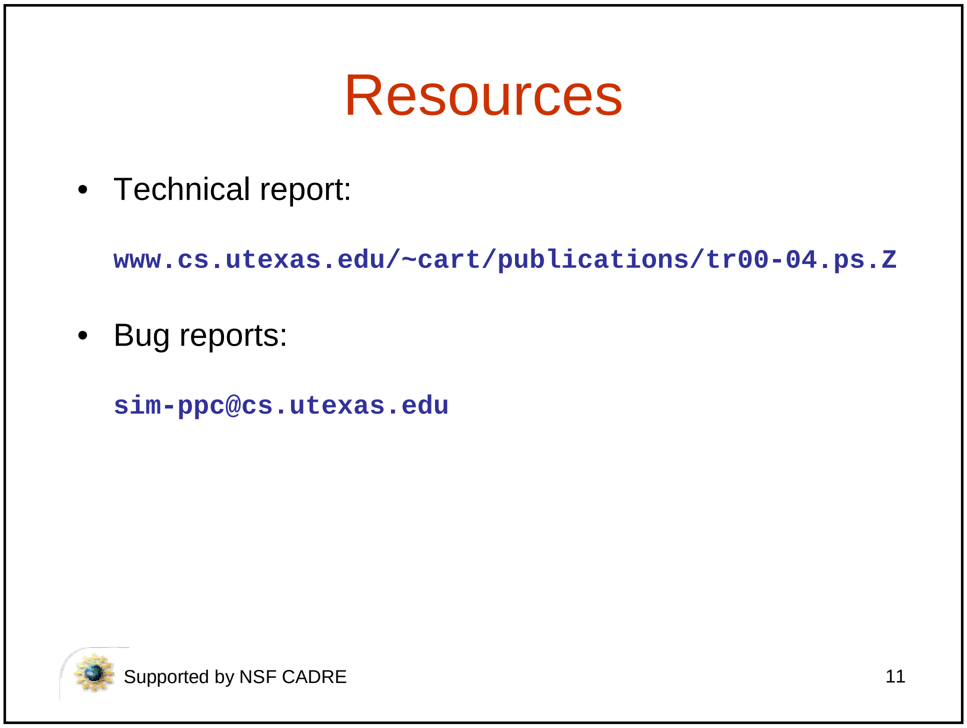
Portability

- Only 32-bit support provided
  - Only user registers and instructions modeled
- IBM AIX on PowerPC
  - Certified for all SPEC CPU95 benchmarks
- Sun Solaris on UltraSparc
  - Certified only for all SPEC CINT95
  - SPEC CFP95 needs additional system call support
- Linux on x86
  - Minimally tested



Future plans

- Add 64-bit support
- Implement kernel registers and instructions
- Support for MP



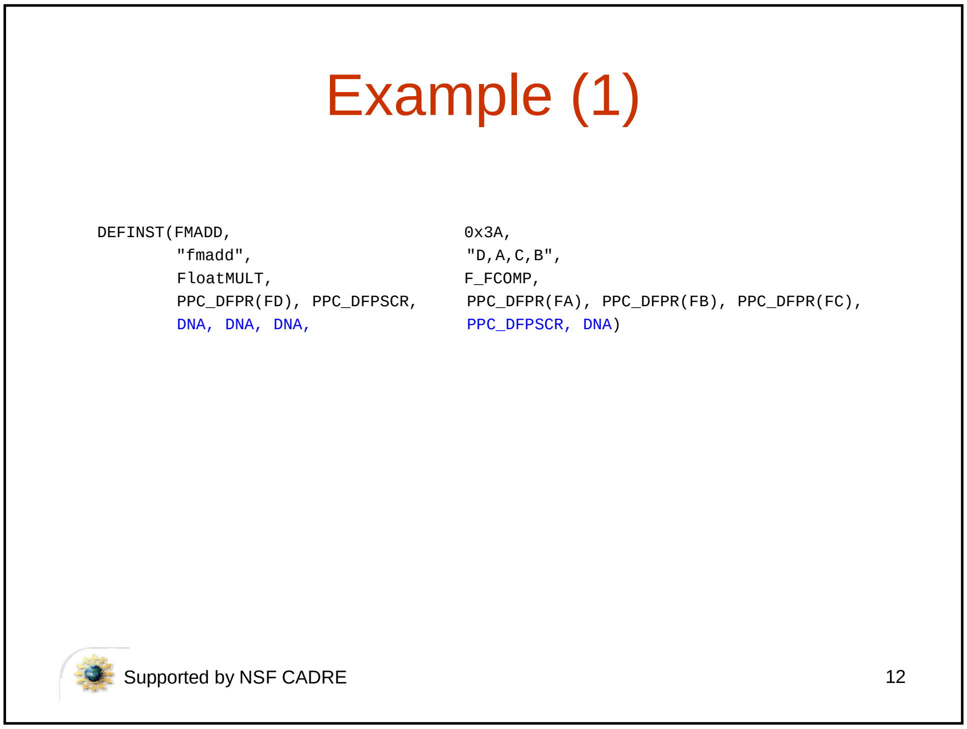
Resources

- Technical report:

  www.cs.utexas.edu/~cart/publications/tr00-04.ps.Z

- Bug reports:

  sim-ppc@cs.utexas.edu



Example (1)

```
DEFINST(FMADD,                                0x3A,
        "fmadd",                               "D,A,C,B",
        FloatMULT,                            F_FCOMP,
        PPC_DFPR(FD), PPC_DFPSCR,              PPC_DFPR(FA), PPC_DFPR(FB), PPC_DFPR(FC),
        DNA, DNA, DNA,                         PPC_DFPSCR, DNA)
```

Example (2)

```
#define FADD_IMPL {
    a = PPC_FPR_DW(RA);  /* copy source registers to temporary
                            variables */
    b = PPC_FPR_DW(RB);

    memcpy(&double_a, &a, sizeof(double) );
    memcpy(&double_b, &b, sizeof(double) );

    /* inline assembly execution */
    asm (“mtsf 0xFF, %2; fadd %0, %3, %4; mffs %1”

    /* copy in result and FPSCR */
          : “=f” (double_dest), “=f” (fpscrout)

    /* give source inputs */
          : “f” (fpscrin), “f” (double_a), “f” (double_b)

    fp1 = (int *) (&fpscrout);
    memcpy(&_fp, (fp1+1), 4);
    dest = (quad_t *) (&double_dest);

    PPC_SET_FPR_DW(FD, *dest);
    PPC_SET_FPSCR( *(int *) (fp1+1));
  }
```

Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - *ss-os Full System simulator*
  - ss-viz SimpleScalar Visualization Tool
- Looking Ahead…

ss-os

SimpleScalar-OS (Sauce)

SimpleScalar 4.0 Micro-34 Tutorial

Jaehyuk Huh,
Karthikeyan Sankaralingam, Vivek Sharma,
Doug Burger, Steve Keckler

University of Texas at Austin

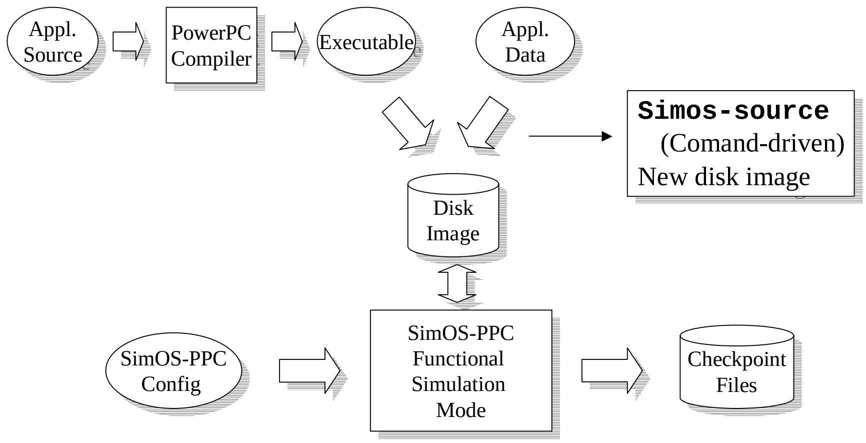
Overview

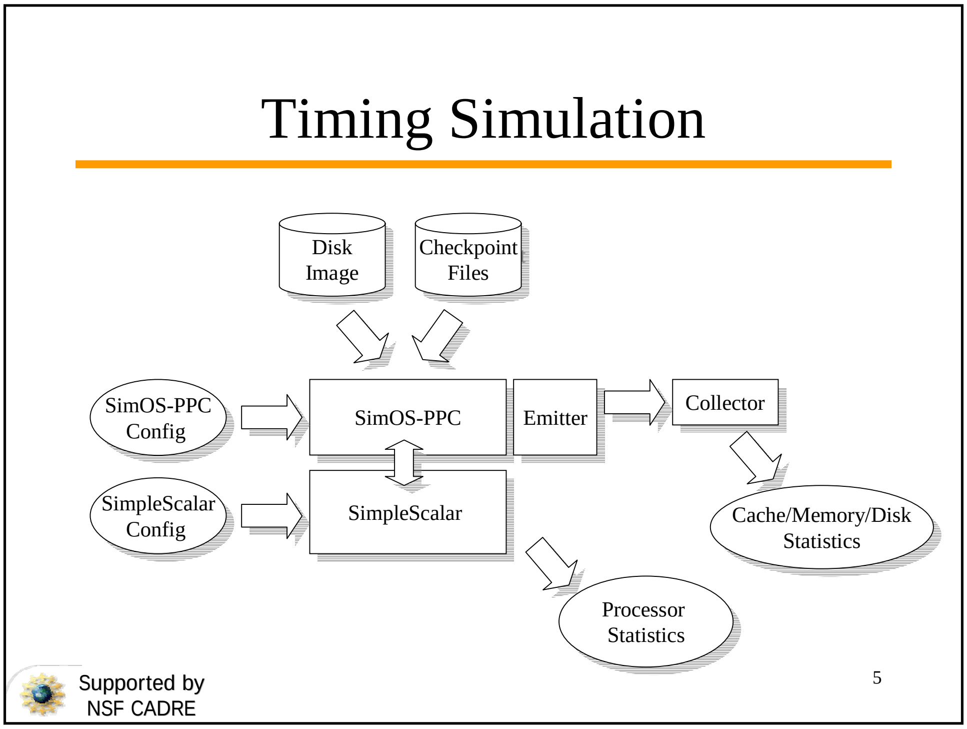
- Need for full system simulation
  - Effect of kernel activity
  - Disk I/O
  - Effect of page and TLB faults
  - Real process (thread) scheduling
- Operating system support for SimpleScalar
  - Integrate ss-ppc simulator with SimOS-PPC
  - Provide full system simulation, running AIX with PowerPC ISA

SimOS-PPC

- PowerPC port based on Stanford SimOS
- Developed by Rick Simpson, Pat Bohrer, Tom Keller, and Ann Marie Maynard at IBM-ARL
- Capability
  - Boot and run AIX with PowerPC ISA
  - 2-level cache system
  - Disk (validated) and network model
  - SMP support
- Limitation: No timing simulation for processors

Setting up Benchmarks

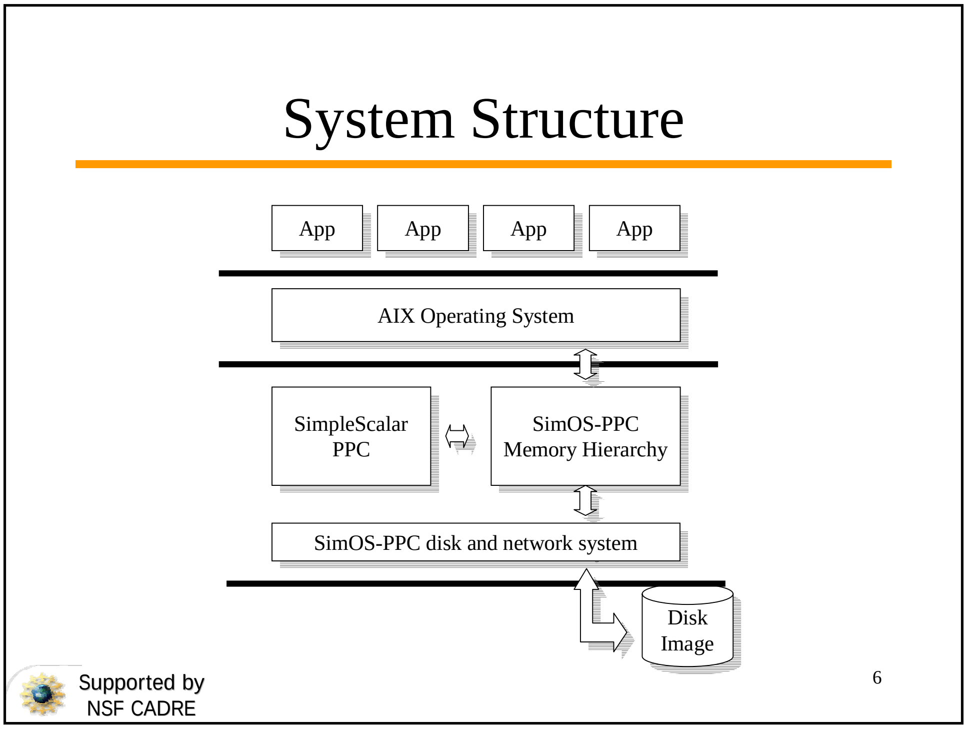




Timing Simulation

Supported by NSF CADRE

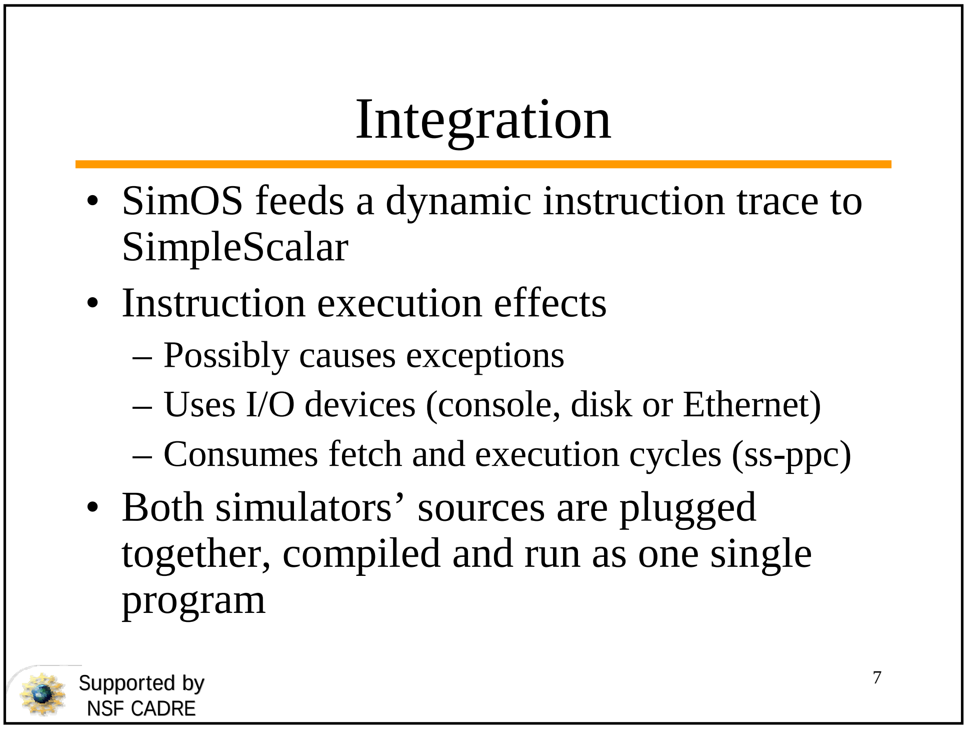
5



System Structure

Supported by NSF CADRE

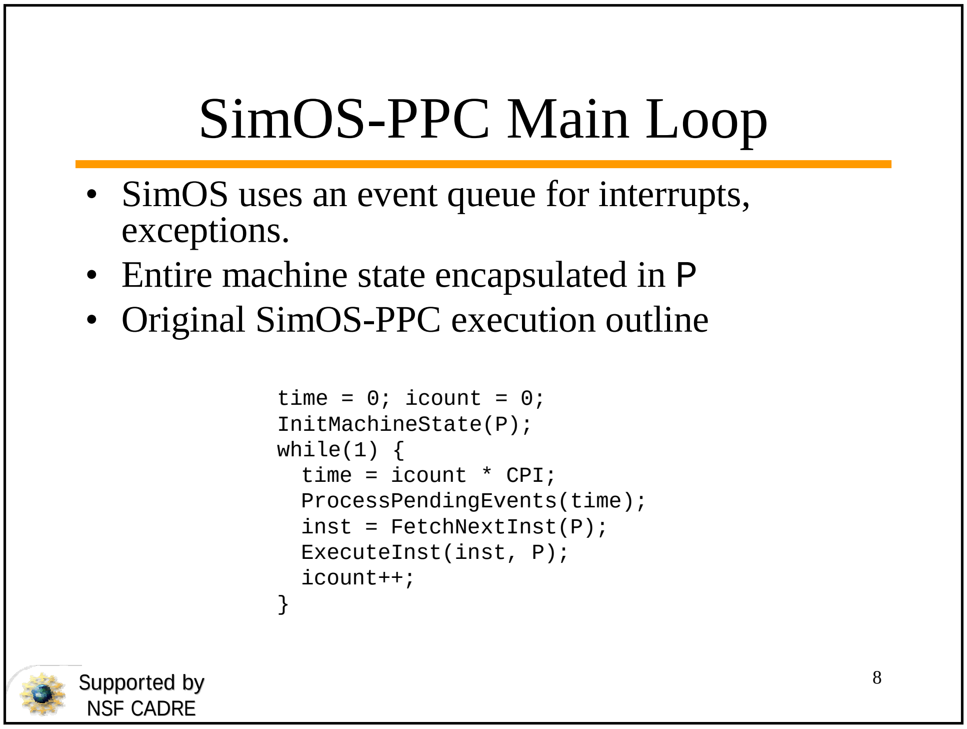
6



Integration

- SimOS feeds a dynamic instruction trace to SimpleScalar
- Instruction execution effects
  - – Possibly causes exceptions
  - – Uses I/O devices (console, disk or Ethernet)
  - – Consumes fetch and execution cycles (ss-ppc)
- Both simulators’ sources are plugged together, compiled and run as one single program

Supported by
NSF CADRE

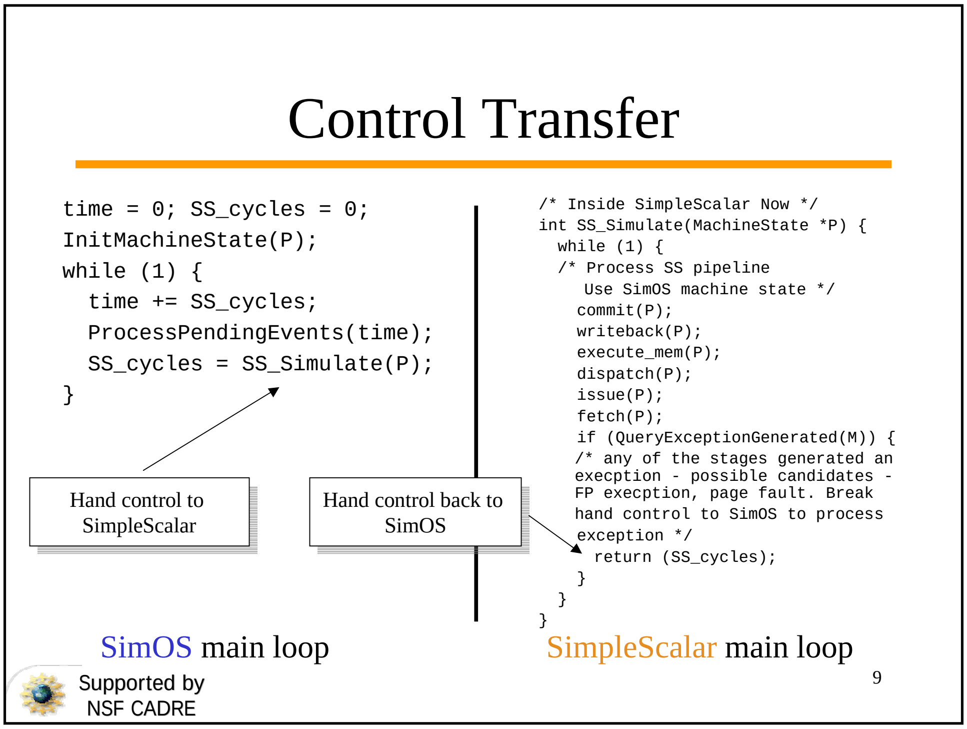


SimOS-PPC Main Loop

- SimOS uses an event queue for interrupts, exceptions.
- Entire machine state encapsulated in P
- Original SimOS-PPC execution outline

```
time = 0; icount = 0;
InitMachineState(P);
while(1) {
  time = icount * CPI;
  ProcessPendingEvents(time);
  inst = FetchNextInst(P);
  ExecuteInst(inst, P);
  icount++;
}
```

Supported by
NSF CADRE



Control Transfer

```
time = 0; SS_cycles = 0;
InitMachineState(P);
while (1) {
  time += SS_cycles;
  ProcessPendingEvents(time);
  SS_cycles = SS_Simulate(P);
}
```

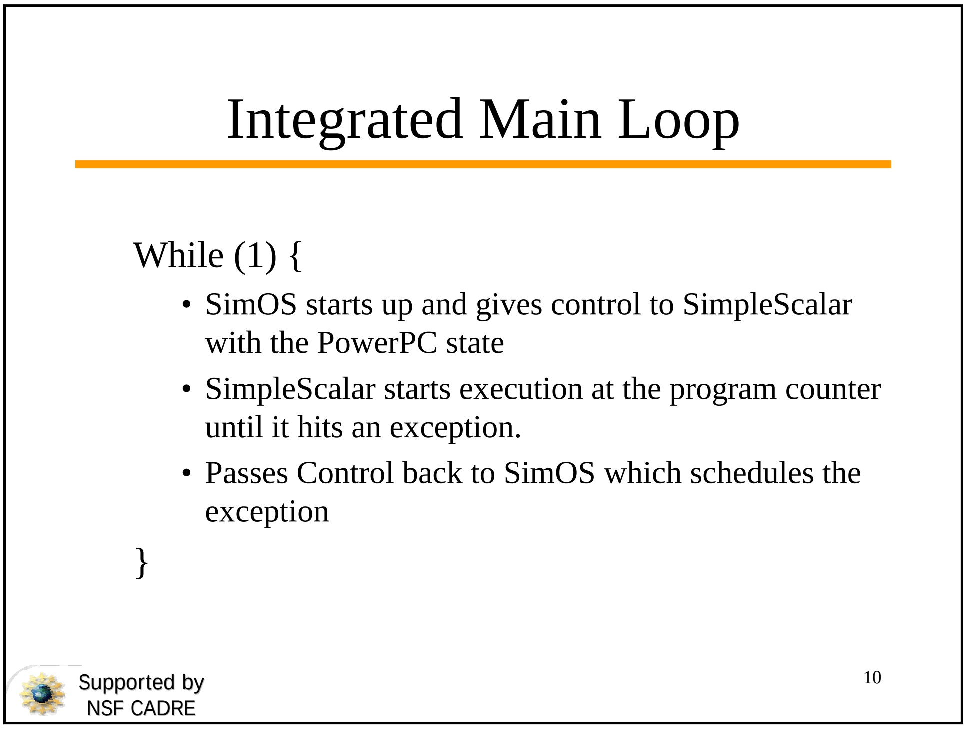
Hand control to SimpleScalar

Hand control back to SimOS

```
/* Inside SimpleScalar Now */
int SS_Simulate(MachineState *P) {
  while (1) {
  /* Process SS pipeline
     Use SimOS machine state */
    commit(P);
    writeback(P);
    execute_mem(P);
    dispatch(P);
    issue(P);
    fetch(P);
    if (QueryExceptionGenerated(M)) {
    /* any of the stages generated an
    execption - possible candidates -
    FP execption, page fault. Break
    hand control to SimOS to process
    exception */
      return (SS_cycles);
    }
  }
}
```

SimOS main loop

SimpleScalar main loop



Integrated Main Loop

While (1) {

- SimOS starts up and gives control to SimpleScalar with the PowerPC state
- SimpleScalar starts execution at the program counter until it hits an exception.
- Passes Control back to SimOS which schedules the exception

}

Disk Images

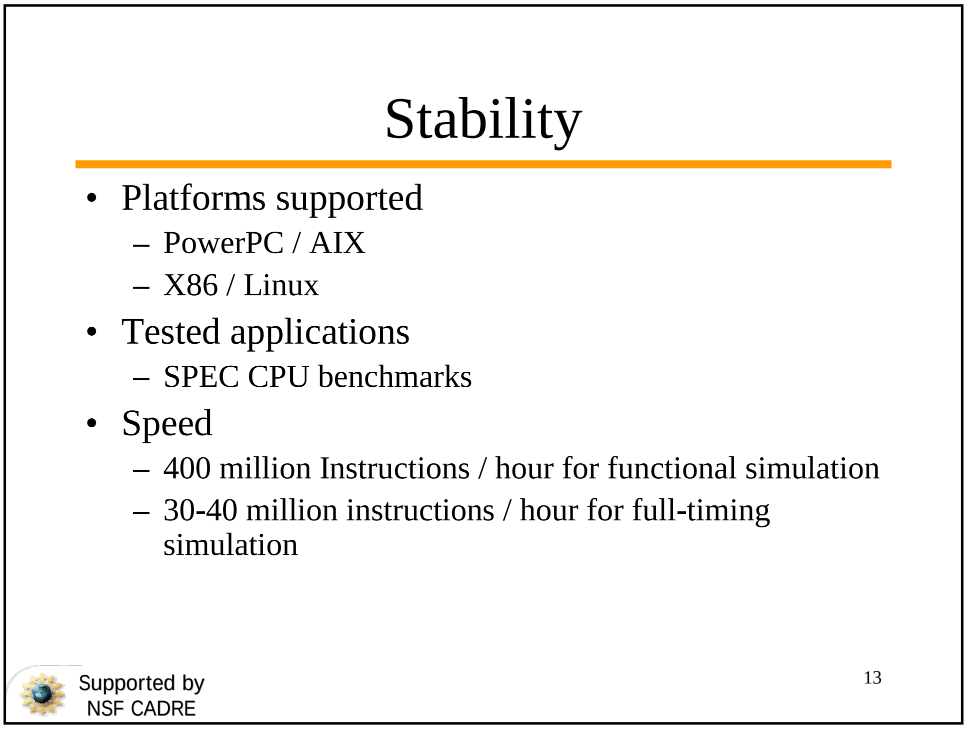
- Disk image keeps the content of simulated disks as a standard UNIX files
- Disk Image Size for AIX support
  - 18 GBytes
  - Real file size: ~1GBytes in sparse file format
- Linux 2.2 :
  - Large disk images need to be split into smaller files (2 GBytes each)

Supported by
NSF CADRE

Issues

- Timing inaccuracies in a few kernel level instructions
- Cache and memory system
  - Use SimOS-PPC code
  - No bus contentions
- TLB handling
  - Hardware-based page table lookup
  - Timing is not accurate

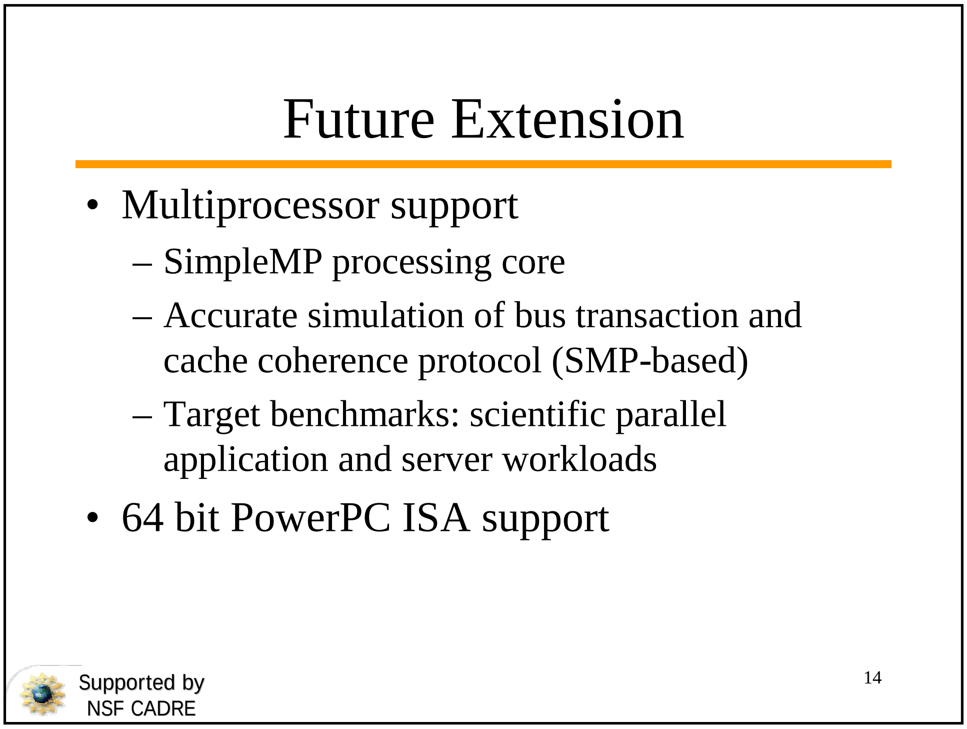
Supported by
NSF CADRE



Stability

- Platforms supported
  - PowerPC / AIX
  - X86 / Linux
- Tested applications
  - SPEC CPU benchmarks
- Speed
  - 400 million Instructions / hour for functional simulation
  - 30-40 million instructions / hour for full-timing simulation

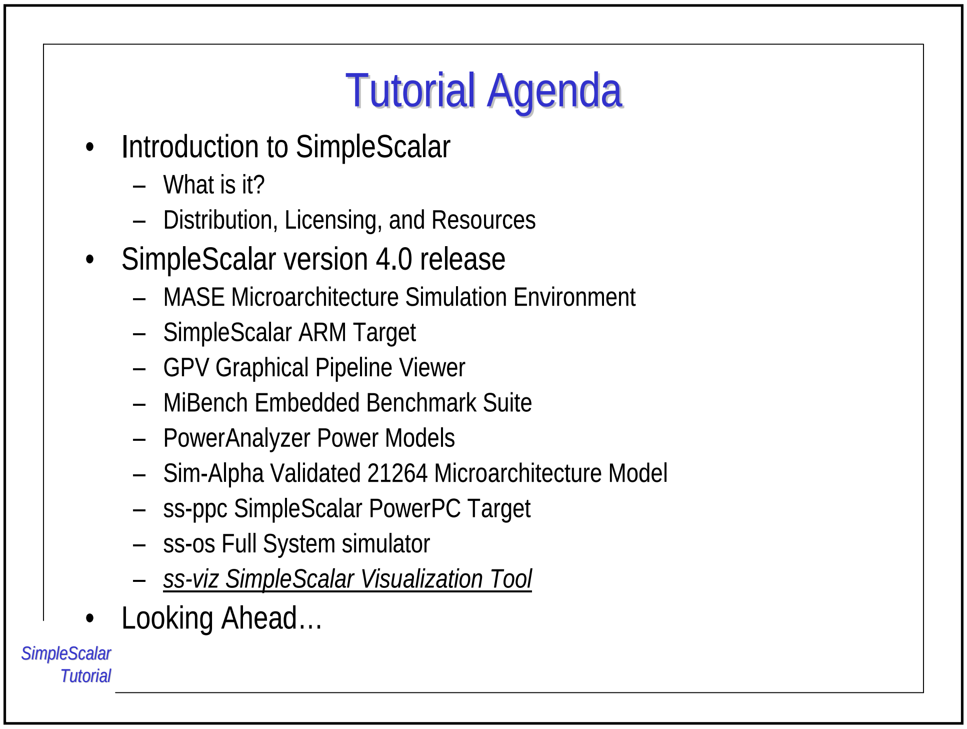
Supported by
NSF CADRE



Future Extension

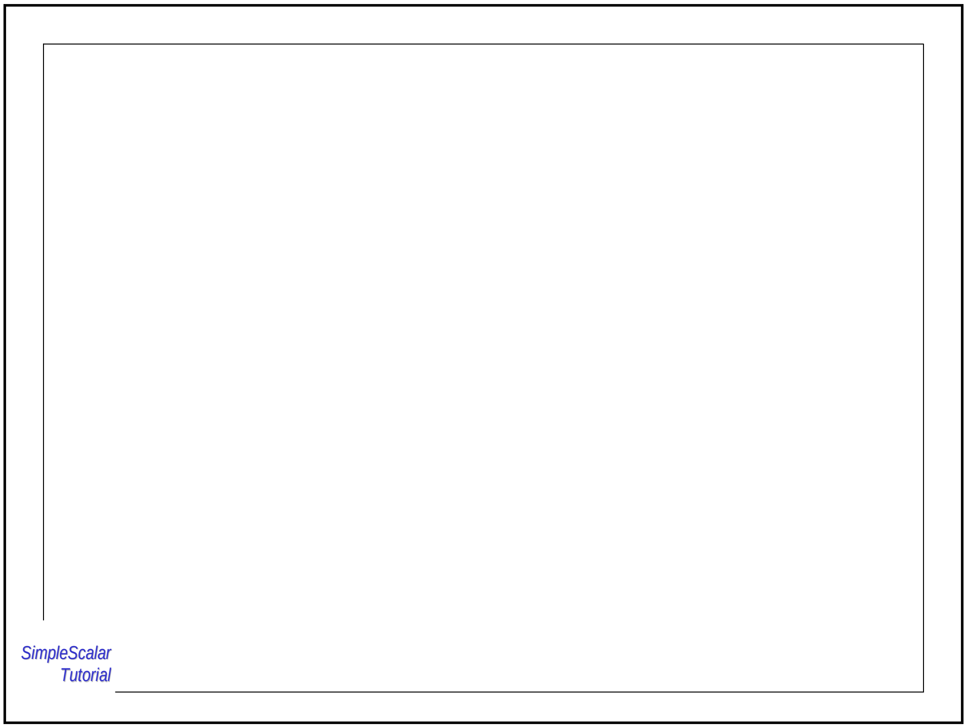
- Multiprocessor support
  - SimpleMP processing core
  - Accurate simulation of bus transaction and cache coherence protocol (SMP-based)
  - Target benchmarks: scientific parallel application and server workloads
- 64 bit PowerPC ISA support

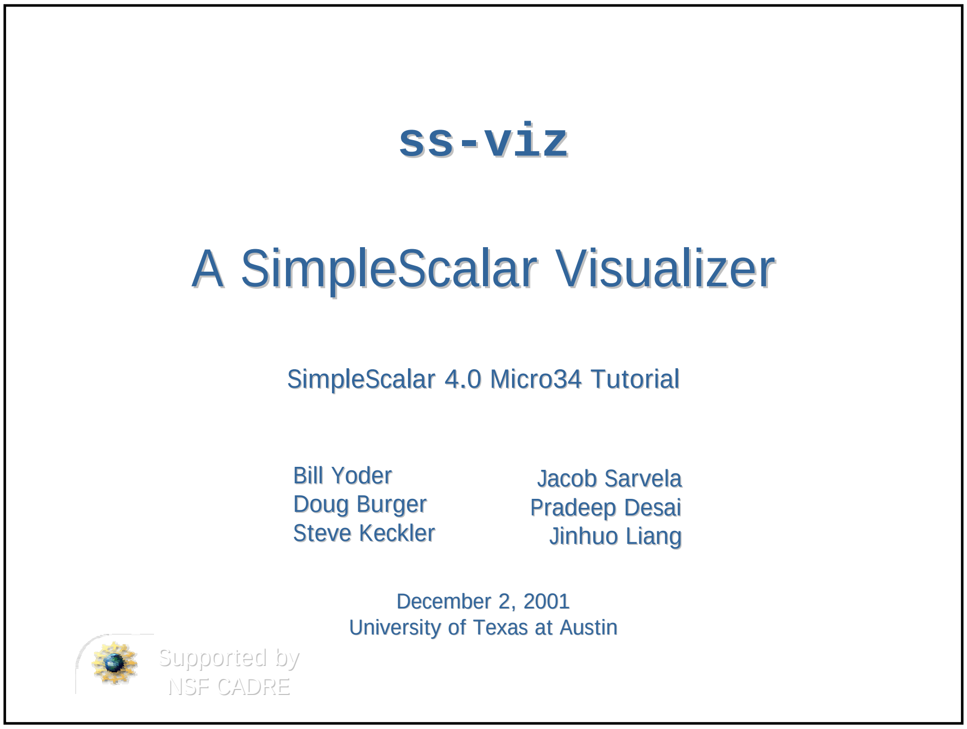
Supported by
NSF CADRE



Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - *ss-viz SimpleScalar Visualization Tool*
- Looking Ahead…





`ss-viz`

A SimpleScalar Visualizer

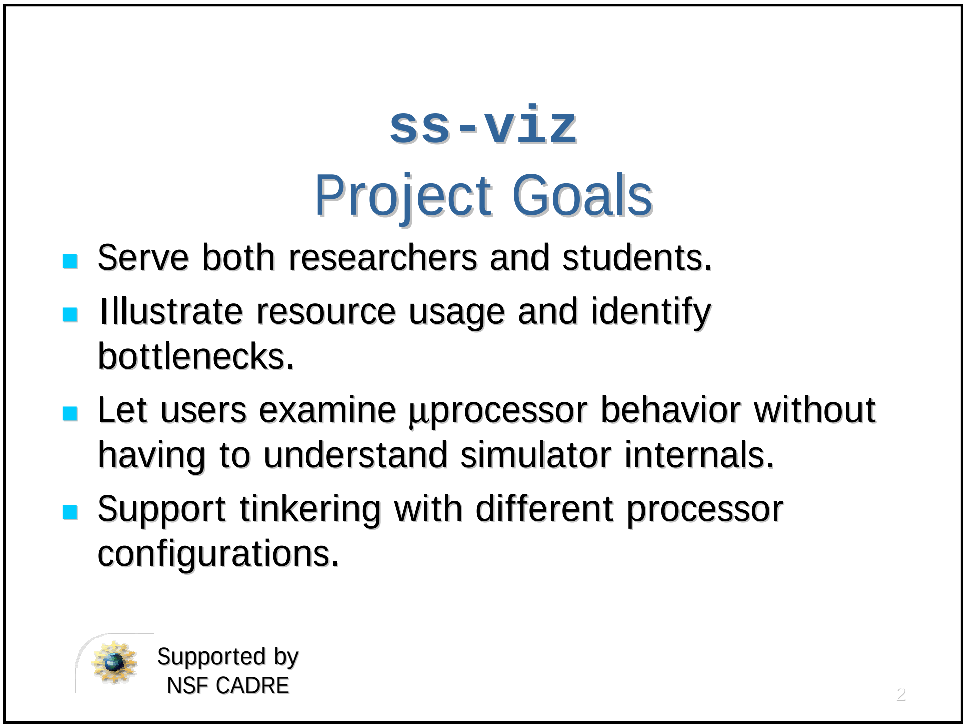
SimpleScalar 4.0 Micro34 Tutorial

Bill Yoder
Doug Burger
Steve Keckler

Jacob Sarvela
Pradeep Desai
Jinhuo Liang

December 2, 2001
University of Texas at Austin

Supported by
NSF CADRE

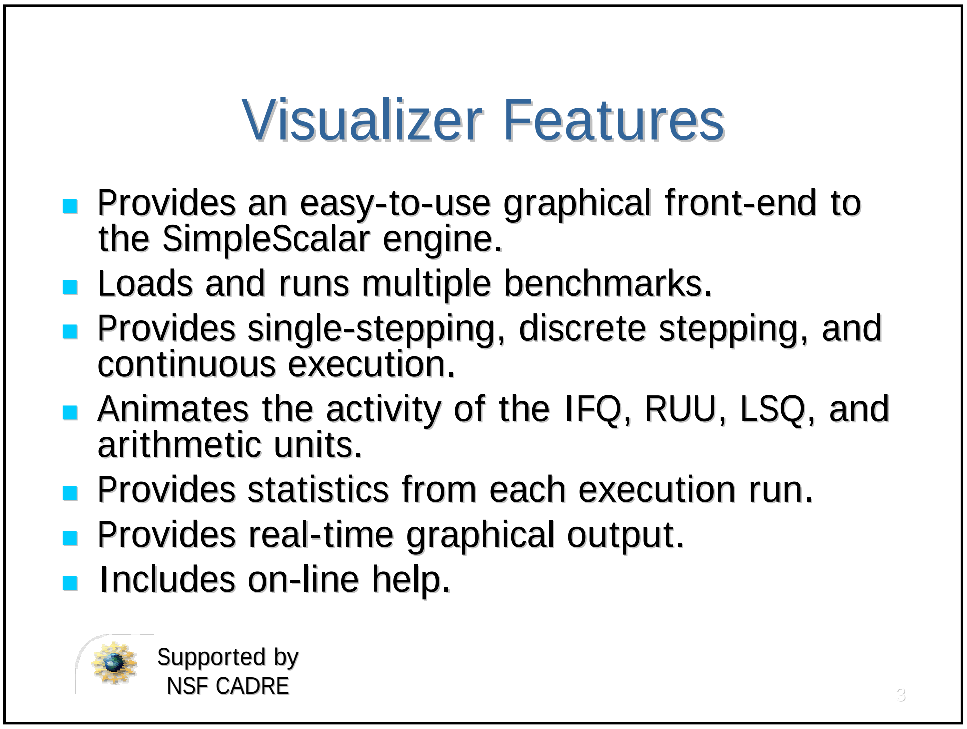


`ss-viz`
Project Goals

- Serve both researchers and students.
- Illustrate resource usage and identify bottlenecks.
- Let users examine µprocessor behavior without having to understand simulator internals.
- Support tinkering with different processor configurations.

Supported by
NSF CADRE

2

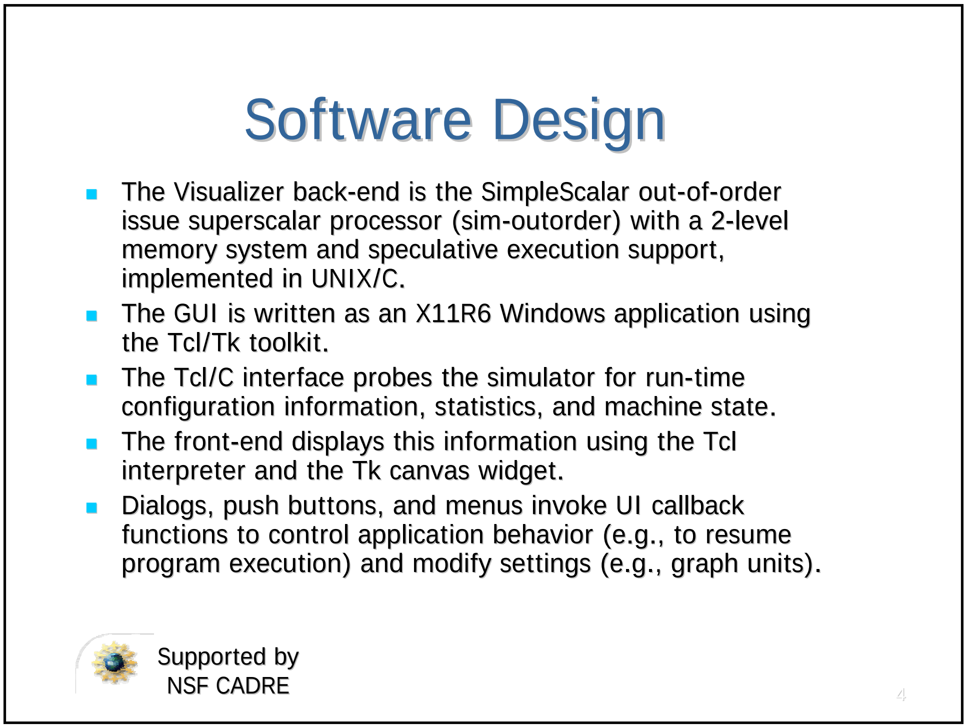


Visualizer Features

- Provides an easy-to-use graphical front-end to the SimpleScalar engine.
- Loads and runs multiple benchmarks.
- Provides single-stepping, discrete stepping, and continuous execution.
- Animates the activity of the IFQ, RUU, LSQ, and arithmetic units.
- Provides statistics from each execution run.
- Provides real-time graphical output.
- Includes on-line help.

Supported by
NSF CADRE

3



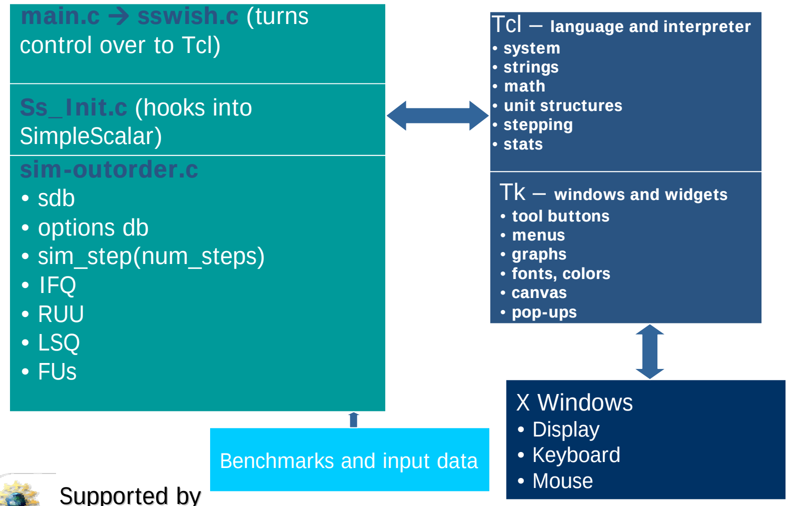
Software Design

- The Visualizer back-end is the SimpleScalar out-of-order issue superscalar processor (sim-outorder) with a 2-level memory system and speculative execution support, implemented in UNIX/C.
- The GUI is written as an X11R6 Windows application using the Tcl/Tk toolkit.
- The Tcl/C interface probes the simulator for run-time configuration information, statistics, and machine state.
- The front-end displays this information using the Tcl interpreter and the Tk canvas widget.
- Dialogs, push buttons, and menus invoke UI callback functions to control application behavior (e.g., to resume program execution) and modify settings (e.g., graph units).

Supported by
NSF CADRE

4

Software Block Diagram



main.c → sswish.c (turns control over to Tcl)

Ss_Init.c (hooks into SimpleScalar)

sim-outorder.c
- sdb
- options db
- sim_step(num_steps)
- IFQ
- RUU
- LSQ
- FUs

Tcl – language and interpreter
- system
- strings
- math
- unit structures
- stepping
- stats

Tk – windows and widgets
- tool buttons
- menus
- graphs
- fonts, colors
- canvas
- pop-ups

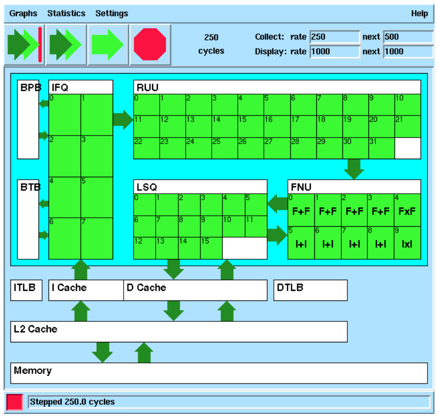
X Windows
- Display
- Keyboard
- Mouse

Benchmarks and input data

Supported by
NSF CADRE

Feedback From Alpha Release

Spring 2001: From two dozen engineering students



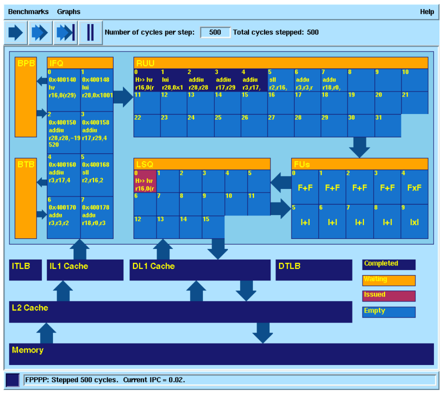
Collect: rate 250 next 500
Display: rate 1000 next 1000

Stepped 250.0 cycles

- ✓ Execution graphs
- ✓ UI concept
- ✓ Statistical info
- ☹ Graphic design
- ☹ Operation
- ☹ Online help

Supported by
NSF CADRE

Today's Status



Number of cycles per step: 500 Total cycles stepped: 500

FPPPP: Stepped 500 cycles. Current IPC = 0.02.

- GUI refurbished with better colors.
- Simplified start-up and user interaction.
- Four units animated (IFQ, RUU, LSQ, FUs).
- HTML help page.
- Various bugs fixed.

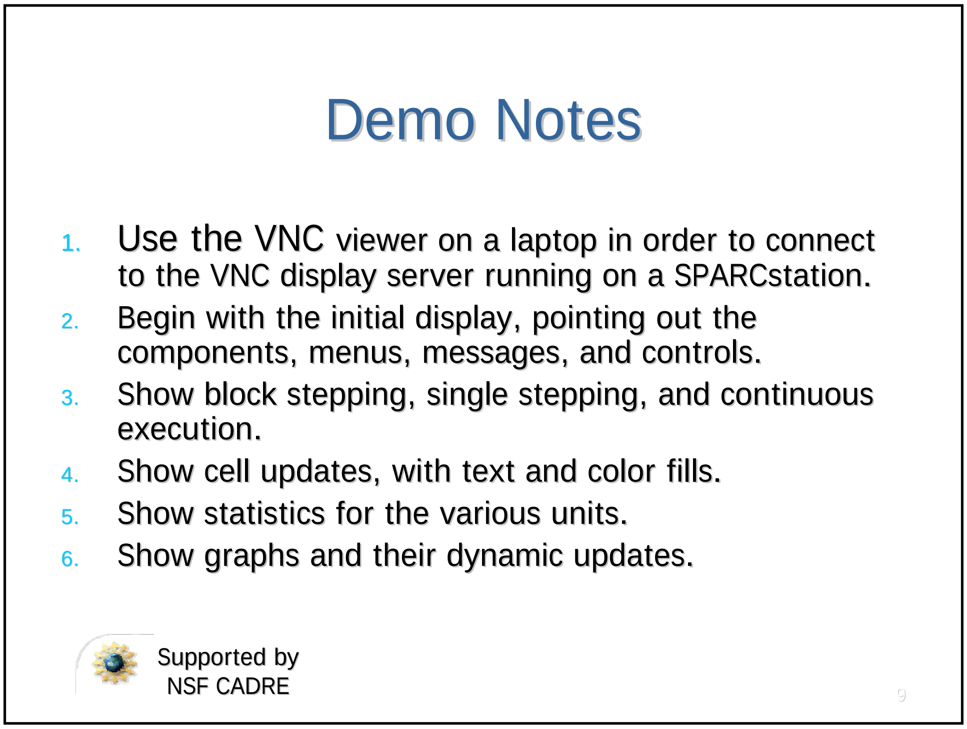
Supported by
NSF CADRE

Future Development

- Portability
  - Package for Solaris/Sparc.
  - Port to Linux/x86.
- Functionality
  - Animate more units, e.g., the L1 and L2 caches.
  - Expose more simulator resources for easy configuration (e.g., the number and type of FUs).
  - Expand on-line help.
  - Enable back-stepping (?!)
- Robustness
  - Improve Tk window management of graphs and window re-sizing.
  - Maintain GUI at benchmark termination.

(Feedback welcome!)

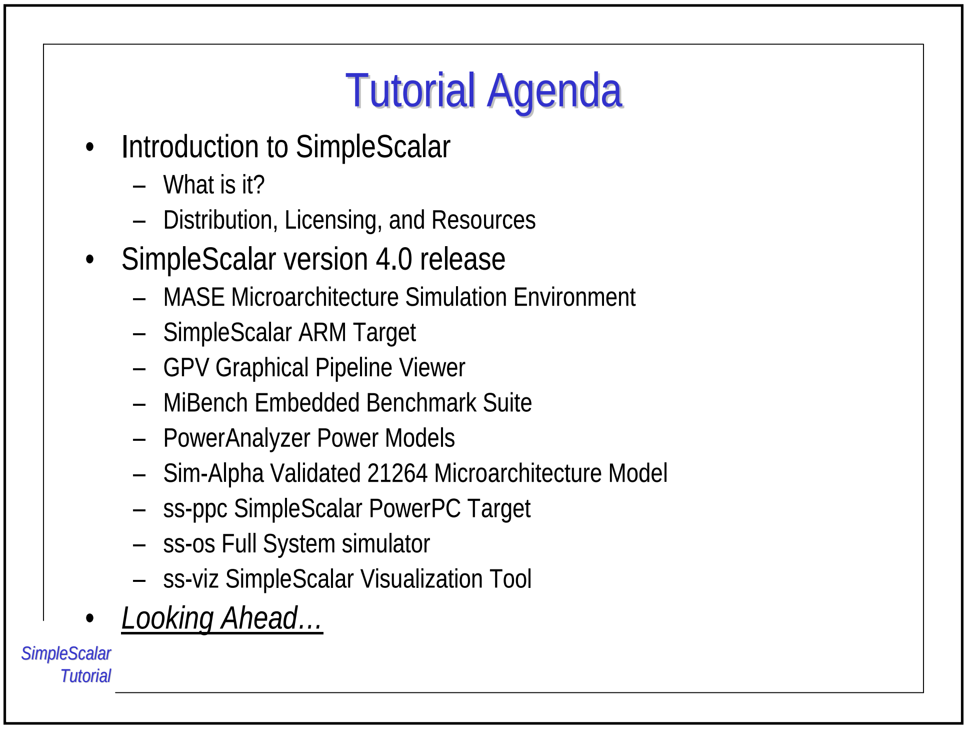
Supported by
NSF CADRE



Demo Notes

1. Use the VNC viewer on a laptop in order to connect to the VNC display server running on a SPARCstation.
2. Begin with the initial display, pointing out the components, menus, messages, and controls.
3. Show block stepping, single stepping, and continuous execution.
4. Show cell updates, with text and color fills.
5. Show statistics for the various units.
6. Show graphs and their dynamic updates.

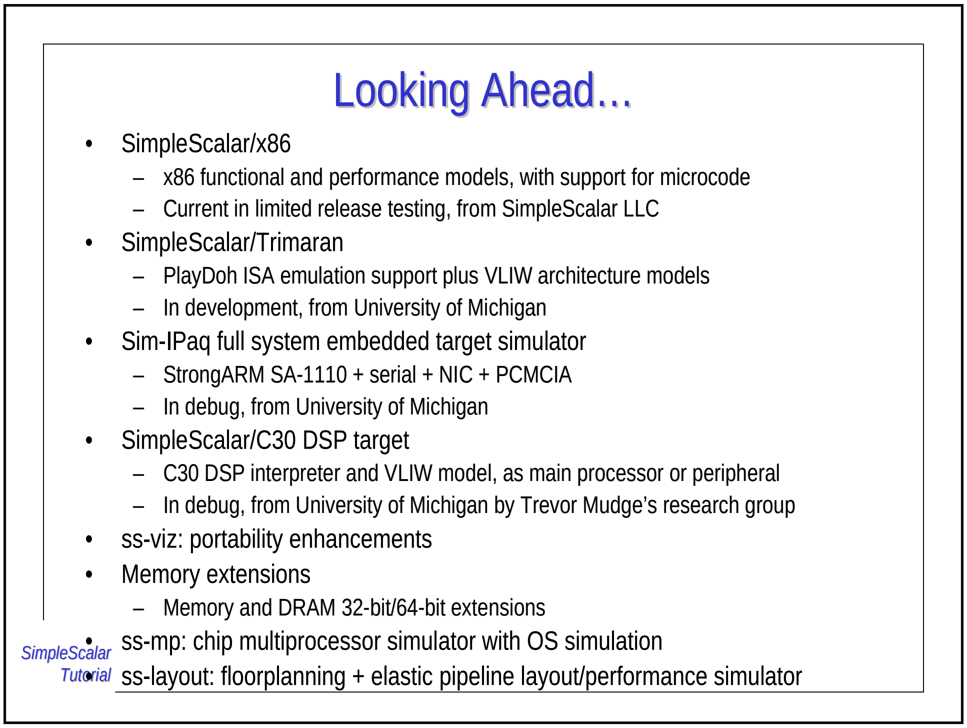
Supported by
NSF CADRE



Tutorial Agenda

- Introduction to SimpleScalar
  - What is it?
  - Distribution, Licensing, and Resources
- SimpleScalar version 4.0 release
  - MASE Microarchitecture Simulation Environment
  - SimpleScalar ARM Target
  - GPV Graphical Pipeline Viewer
  - MiBench Embedded Benchmark Suite
  - PowerAnalyzer Power Models
  - Sim-Alpha Validated 21264 Microarchitecture Model
  - ss-ppc SimpleScalar PowerPC Target
  - ss-os Full System simulator
  - ss-viz SimpleScalar Visualization Tool
- *Looking Ahead…*

SimpleScalar Tutorial



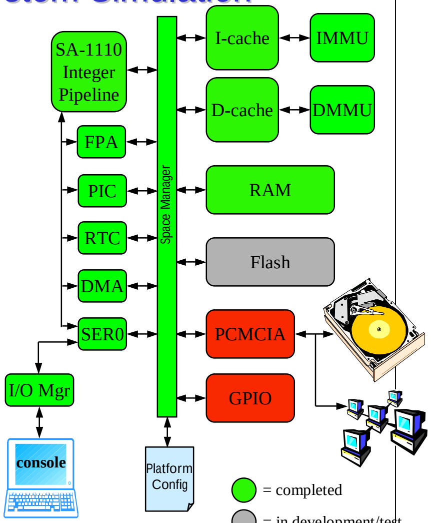
Looking Ahead…

- SimpleScalar/x86
  - x86 functional and performance models, with support for microcode
  - Current in limited release testing, from SimpleScalar LLC
- SimpleScalar/Trimaran
  - PlayDoh ISA emulation support plus VLIW architecture models
  - In development, from University of Michigan
- Sim-IPaq full system embedded target simulator
  - StrongARM SA-1110 + serial + NIC + PCMCIA
  - In debug, from University of Michigan
- SimpleScalar/C30 DSP target
  - C30 DSP interpreter and VLIW model, as main processor or peripheral
  - In debug, from University of Michigan by Trevor Mudge's research group
- ss-viz: portability enhancements
- Memory extensions
  - Memory and DRAM 32-bit/64-bit extensions
- ss-mp: chip multiprocessor simulator with OS simulation
- ss-layout: floorplanning + elastic pipeline layout/performance simulator

SimpleScalar Tutorial

SimpleScalar/ARM System Simulation

- System simulation development
  - ARM7 + FPA + SA-1110 device set
  - Linux + MiBench workload
- Key infrastructure features
  - Space manager directs I/O using a standard extensible interface
  - Platform configuration description file permits multiple target emulation without code changes
  - I/O manager supports recording and playback of external I/O for reproducible real-time experiments
- Status
  - Processor/memory devices deployed
  - VM MMU, RTC, PIC, DMA, SER0 devices completed
  - 8M+ instructions into Linux boot



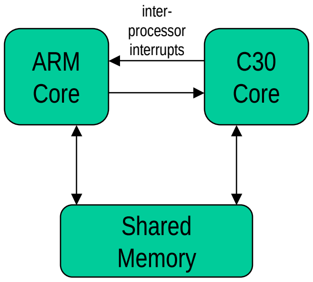
🟢 = completed

● = in development/test

🔴 = next generation

SimpleScalar/C30 Target

- Many embedded targets feature a DSP
  - For fast processing of multimedia workloads
  - e.g., signal processing, codec routines, image processing
  - Typical embedded system architecture couples a general purpose microprocessor with a DSP
- Adding TI TMS320C30 (C30) ISA target
  - Integer and floating-point ISA components
  - Power control instructions
- May be used as a processor or peripheral device
  - Permits use of general purpose processor model and C30 model in tandem
  - Inter-processor communication implemented with bi-directional mailbox primitives
  - Requires a fairly sophisticated compiler tool chain, e.g., GNU GCC for ARM + TI DSP target compiler


inter-processor interrupts
ARM Core
C30 Core
Shared Memory