The SimpleScalar Architectural Research Tool Set

Todd M. Austin
Gurindar S. Sohi

Computer Sciences Department
University of Wisconsin – Madison

`http://www.cs.wisc.edu/~austin/austin.html`

Talk Outline

Tool Set Overview

Simulation Tools

Insights and Experiences

Future Directions

Tool Set Overview

Architectural research test bed: compilers, libraries, simulators

Targeted to SimpleScalar architecture

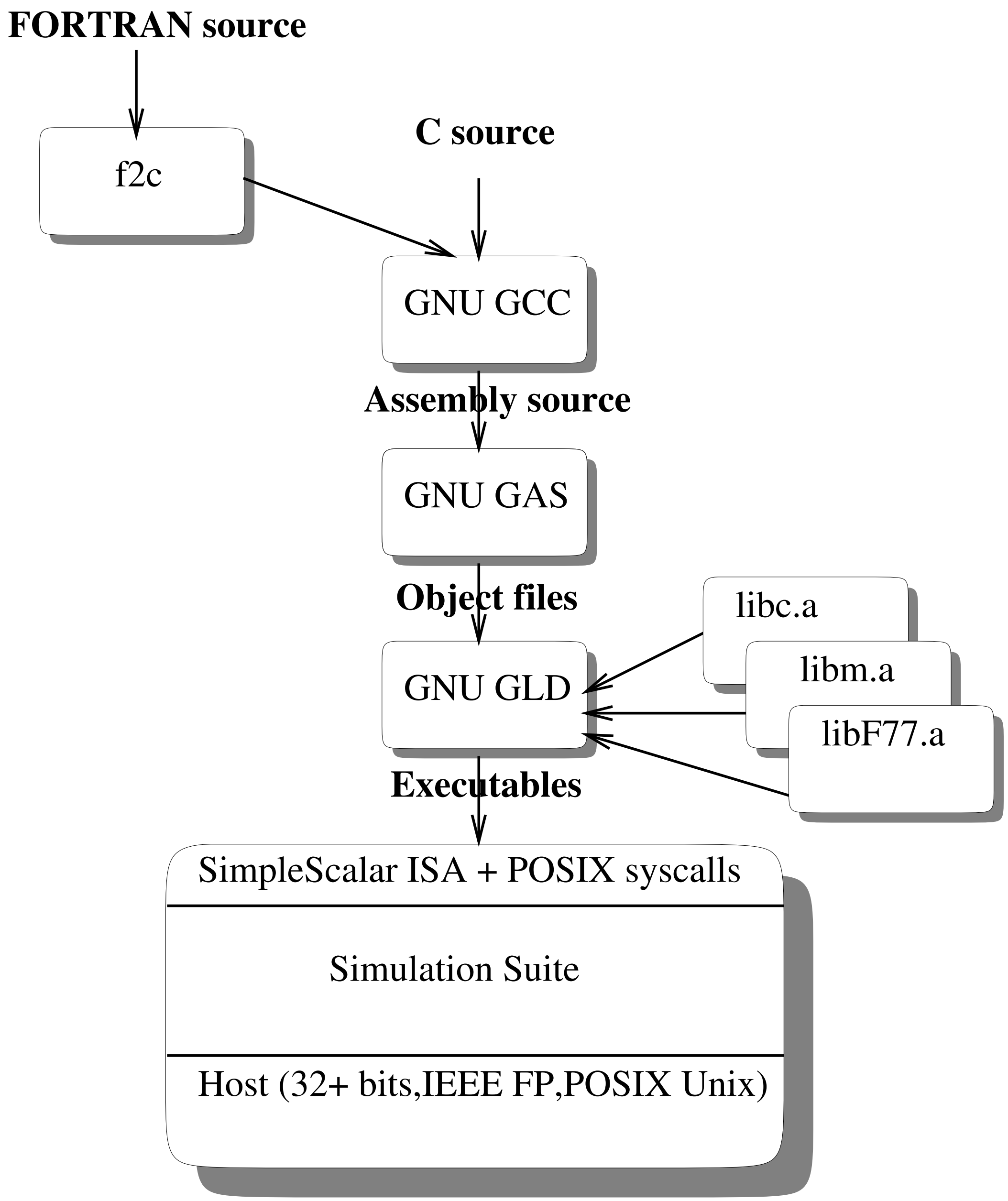
Hosted on most any Unix box

Grew out of tool set assembled by Wisconsin Multiscalar group

Why we like it:

- extensible
- portable
- detailed
- fast

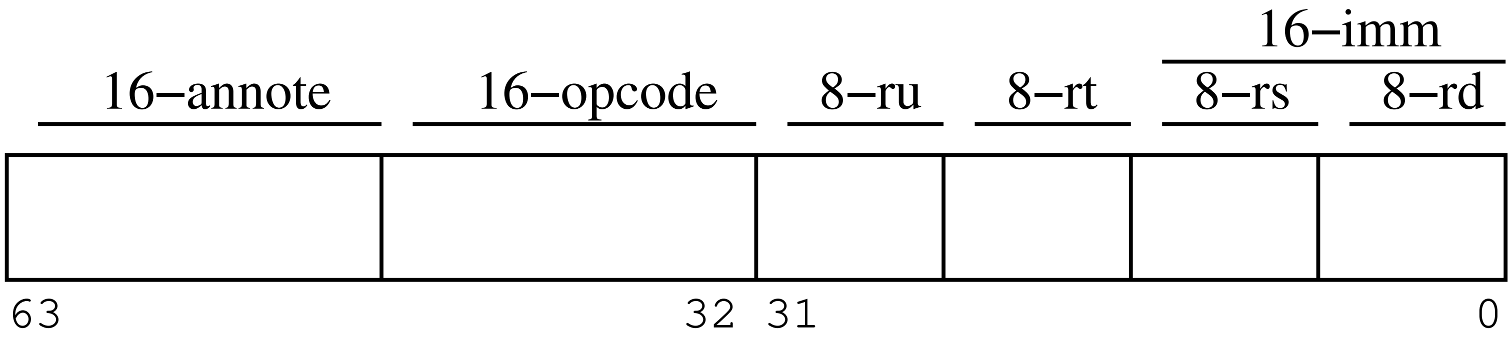
Tool Set Overview



The SimpleScalar Architecture

MIPS-I ISA, plus:

- `register+register` addressing
- post-increment and -decrement addressing
- no architected delay slots



Simulation Tools Overview

Extensive simulator suite:

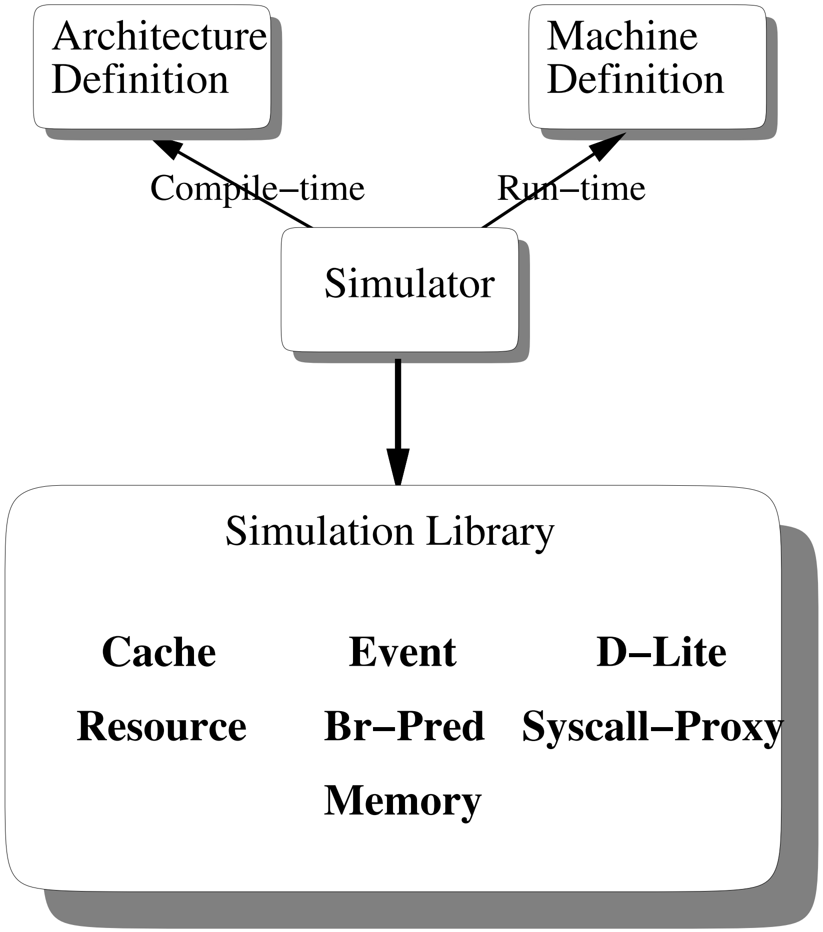
- functional, debugger, profiler, trace generator, cache, in-order issue, out-of-order issue
- easy to roll your own

Easily extensible through architecture and machine definitions

Track current technologies: out-of-order issue, non-blocking loads

Fast runners: functional-2.5M inst/s, out-of-order-150,000 inst/s

Simulator Structure



- small simulator implementations: out-of-order - 1900 lines
- stable, general interfaces

Architecture and Machine Definitions

Architecture Definition:

- decode mask
- (dis-)assembler template
- functional unit requirements
- instruction flags
- input/output dependencies
- evaluation template (uses sim-defined reg/mem accessors)

Machine Definition:

- decode/issue/retire widths
- cache configuration
- functional unit configuration
- etc., *ad infinitum*

Insights and Experiences

1.5 years in development (ancestors date back to mid/late '80s)
Getting heavy use at Madison, public release forthcoming

Insights:

- effective performance analysis requires fast simulators
- execution-driven simulation is worth the trouble
- distributed batch processing is a really good thing

High-Performance Simulation

Becoming more important:

- ILP: mis-speculation is costly, parallel operations serialized
- workloads getting larger

Getting fast:

- use appropriate level of detail
- squeeze constants out of simulators

Results:

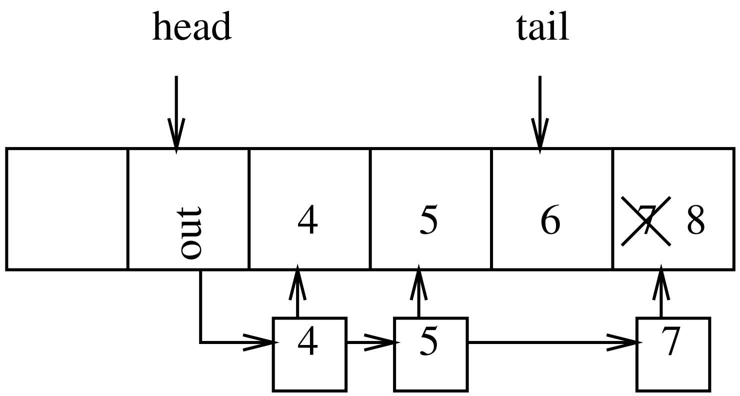
- functional simulator: 2.5M inst/s
- out-of-order issue timing simulator: 150,000 inst/s

Fast Functional Simulation

- predecode text segment
- no looping, use direct dispatch
- force frequently used values into registers
- inline everything
- efficient memory allocation and access

Fast Out-of-Order Issue Timing Simulation

- fast functional simulation
- hashed access to associative cache sets
- dependence chains to avoid linear scans
- capability-based pointers for fast mis-speculation recovery



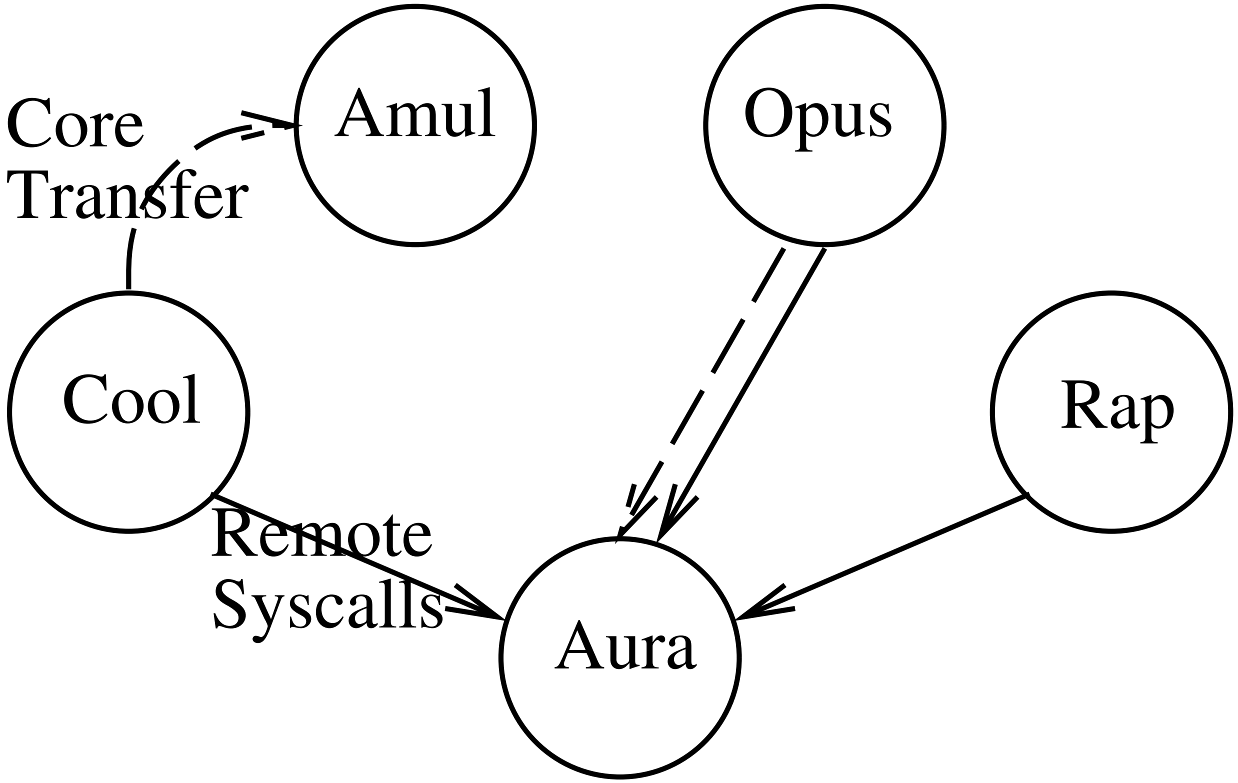
Execution- vs. Trace-Driven Simulation

- about as fast: 2.5M inst/s vs. disk: .5M+ inst/s
- performance difference less noticeable with complex simulator
- slightly more complicated: requires eval expr, syscalls
- more reliable: track and verify execution state
- more capable/accurate:
    - can see process state
    - can process mis-speculated instructions

Experiment: GCC with and without issue of mis-speculated insts

- 1% more cycles executed
- 8% more data cache accesses
- 0.17% lower data cache miss ratio

Distributed Batch Processing



- *Condor*: many jobs run on idle machines w/ local resources
- requires large local storage spaces
- excellent throughput: 2.5 CPU-months in 6 days
- high variance in response time

Looking Ahead

- summer release (watch comp.arch)

- OS port (MIPS Linux?)

- simulator ports: MIPS, SPARC, x86 coming