

# Getting Started with SimpleScalar Tool Set

Based on the manual by Todd Austin, University of Wisconsin

Originally created by Rong Ge  
Department of Computer Science  
University of South Carolina  
Edited By Michael Narayan

## Objective

- Get you acquainted with SimpleScalar tool set
- Introduce materials that help you work on the upcoming project

## Outline

- Overview of SimpleScalar tool set
- Components of SimpleScalar tool set
- Usage of SimpleScalar tool set

## Simulation in computer architecture

- What will you do to improve performance of a computer system?
  - Propose a new system
- How to know the performance of the proposed system?
  - Simulate it on an existing system
- SimpleScalar tool set is a set of simulators for computer architecture

## SimpleScalar Simulators

- SimpleScalar tool set
  - Designed to measure the performance of several parts of a superscalar processor and its memory hierarchy
  - SuperScalar architecture
    - Derived from the MIPS-IV ISA
    - Further information on <http://www.simplescalar.com>
- Other simulation systems
  - may be similar or very different

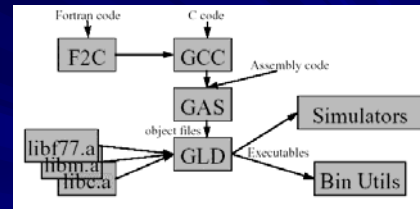
## Advantages to SimpleScalar

- Extensible
  - Source for compiler, libraries and simulators
  - User-extensible instruction format
- Portable
  - Runs on NT and most Unix platform
  - Target can support multiple ISAs
- Detailed
  - Interfaces support Simulators of arbitrary detail
  - Multiple Simulators included with distribution

## History and resources

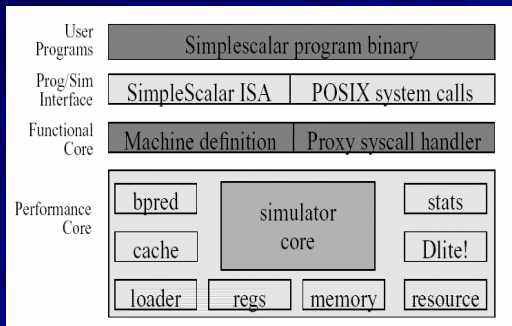
- Written by Todd Austin
- First public release in July, '96 with Doug Burger
- Current version 3.0, version 4.0 soon
- Public release available from University of UW-Madison
  - <http://www.cs.wisc.edu/~mscalar/simplescalar.html>
- Technical report
  - The SimpleScalar tool set, version 2
  - UW-Madison technical report #1342, July 1997

## SimpleScalar tool set overview



- Compiler chain is GNU tools ported to SS ISA
- Fortran codes are converted with AT&T f2c
- Libraries are CLIBC ported to SimpleScalar

## Simulator structure



## Simulation suite overview

sim-fast sim-safe	sim-profile	sim-cache sim-cheetah sim-bred	sim-eio	sim-outorder
300-400 lines functional 4+ MIPS	900 lines functional lots of stats	<1000 lines functional cache stats pred stats	func. timing reproducibility	~4000 lines performance OoO issue branch pred. mis-spec. ALUs cache TLB 200+ KIPS
← performance → ← detail →				

## Why the suite

- Tradeoff between performance and detail
  - The execution of a processor can be modeled as states and the time to transit between states
  - State information
    - Values in all memory locations
    - Values and status of all caches
    - Values and status of the TLB
    - Values and status of the branch prediction table(s) or BTB.
    - All processor state, register file, register update unit (RUU)
  - The more state simulated (more detailed), the longer time taken
- Important to evaluate desired measurements
- Limit the simulation to only the necessary state

## Simulator Executables

- Executable files include:
  - sim-bpred
  - sim-cache
  - sim-cheetah
  - sim-fast
  - sim-outorder
  - sim-profile
  - sim-safe

## Run them in Unix Lab

- Installed in
  - /home/grads/mnarayan/cs4504/simplescalar
  - Compilers in
    - /home/grads/mnarayan/cs4504/simplescalar/bin
  - Simulators in
    - /home/grads/mnarayan/cs4504/simplescalar/simplesim-3.0
- To use them, do
  - setenv PATH \${PATH}:
  - /home/grads/mnarayan/cs4504/simplescalar/bin :
  - /home/grads/mnarayan/cs4504/simplescalar/simplesim-3.0

## Sample commands

- · Compiling a C program (example):
  - sslittle-na-ssstrix-gcc -g -O -o foo foo.c
- · Compiling a Fortran program (example):
  - sslittle-na-ssstrix-gcc -g -O -o foo foo.f
- · Compiling a SimpleScalar assembly program (example):
  - sslittle-na-ssstrix-gcc -g -O -o foo foo.s
- · Running a program (example):
  - sim-safe [- sim options] program [-program opts]

## Global simulator options

- -h ; prints the simulator help message
- -d ; enables debugging messages
- -i ; starts up the Dlite! Debugger
- -q ; quit immediately; (use with -dumpconfig)
- -config <file> ; read processor/architecture configuration parameters from <file>
- -dumpconfig <file> ; save configuration parameters into <file>

## A sample configuration file

- # run pipeline with in-order issue
- -issue:inorder false
- # issue instructions down wrong execution paths
- -issue:wrongpath true
- # register update unit (RUU) size
- -ruu:size 16
- # L1 data cache config, i.e., {<config>|none}
- -cache:dl1 dl1:512:64:2:l
- # L1 data cache hit latency (in cycles)
- -cache:dl1lat 1

## Configuration file (contd)

- # L2 data cache config, i.e., {<config>|none}
- -cache:dl2 ul2:32768:64:1:l
- # L2 data cache hit latency (in cycles)
- -cache:dl2lat 6
- # L1 inst cache config, i.e., {<config>|dl1|dl2|none}
- -cache:il1 il1:512:64:2:l
- # L1 instruction cache hit latency (in cycles)
- -cache:il1lat 1

## What parameters do

- · issue:inorder false
  - issue instructions in order, default false
- · issue:wrongpath true
  - allows instructions to actually go ahead and execute after a misspeculation, default true
- · ruu:size 16
  - the number of instructions and the default is 16.
- · cache:dl1 dl1:512:64:2:l
  - L1 data cache specified by 5 parameters: <name>, <nsets>, <bsize>, <associativity>, and <replacement policy> respectively.

## What parameters do (contd)

- <name>: the cache name; it must be unique
- <nsets>: the number of sets (entries) in the cache
- <bsize>: the block size (in terms of # of bytes)
- <associativity>: the associativity of the cache (must be a power of 2)
- <repl>: the replacement policy of the cache; can be "l", "f", or "r", where "l" is Least Recent Used (LRU), "f" is First in First out (FIFO), and "r" is random placement

- `cache:d1lat 1`
  - Specifies the hit latency of the L1 data cache. The default is one clock cycle.

## Use simulators on your own code

- First, write your code. (c code for instance)
  - Vi test-math.c (normal c code, nothing different)
- Compile and link it for superscalar architecture
  - Simple as you compile and link it for common existing machine, except
  - Using compiler `sslittle-na-sstrix-gcc` other than `gcc`
  - Link different library  
`/home/grads/mnarayan/cs4504/simplescalar/sslittle-na-sstrix/lib` other than `/usr/lib` or `/lib`
  - Example:
    - `sslittle-na-sstrix-gcc -g -O -o test-math test-math.c -L/home/grads/mnarayan/cs4504/simplescalar/sslittle-na-sstrix/lib -lc`
    - You should get `test-math` which is executable for the simulated superscalar architecture

## output from simulation

```

■ Simulate test-math
  * sim-outorder /usr/local/simplescalar/simpesim-2.0/tests/bin/big/test-math

sim: simulation started @ Wed Oct 1 11:02:19 2003, options follow:

# -config          # load configuration from a file
# -dumpconfig     # dump configuration to a file
# -i              false # start in Dilite debugger
...
-seed             1 # random number generator seed (0 for timer seed)
# -ptrace        <null> # generate pipetrace, i.e., <filename>|stdout|stderr <range>
-issue:inorder   false # run pipeline with in-order issue
-ruu:size        16 # register update unit (RUU) size
...
-cache:d1        d1:1:128:32:4:1 # l1 data cache config, i.e., (<config>|<none>)
-cache:d1lat     1 # l1 data cache hit latency (in cycles)
-cache:d2        d2:1:024:64:4:1 # l2 data cache config, i.e., (<config>|<none>)
-cache:d2lat     6 # l2 data cache hit latency (in cycles)
-cache:i1        i1:1:512:32:1:1 # l1 inst cache config, i.e., (<config>|d1|<none>)
-cache:i1lat     1 # l1 instruction cache hit latency (in cycles)
-cache:i2        d2 # l2 instruction cache config, i.e., (<config>|d2|<none>)
-cache:i2lat     6 # l2 instruction cache hit latency (in cycles)
-mem:lat         18 2 # memory access latency (<first_chunk> <inter_chunk>)
...
    
```

## output from simulators (contd)

```

sim: ** starting performance simulation **
pow(12.0, 2.0) = 144.000000
pow(10.0, 3.0) = 1000.000000
str: 123.456
x: 123.000000
str: 123.456
x: 123.456000
str: 123.456
x: 123.456000
123.456 123.456000 123.1000
sinh(2.0) = 3.02688
sinh(3.0) = 10.01737
h=3.60555
atan2(3.2) = 0.98279
pow(3.60555, 4.0) = 169
169 / exp(0.98279 * 5) = 1.24102
3.93117 * 5*log(3.60555) = 10.34355
cos(10.34355) = -0.6068, sin(10.34355) = -0.79486
x 0.5x
x0.5
x 0.5x
-1e-17 != -1e-17 Worked!
    
```

## output from simulators (contd)

```

sim: ** simulation statistics **
sim_num_insts    216432 # total number of instructions committed
sim_num_refs     56969 # total number of loads and stores committed
sim_num_loads    34153 # total number of loads committed
sim_num_stores   22816.0000 # total number of stores committed
sim_num_branches 38648 # total number of branches committed
sim_elapsed_time 2 # total simulation time in seconds
sim_inst_rate    108216.0000 # simulation speed (in insts/sec)
sim_total_insts  238534 # total number of instructions executed
sim_total_refs   62630 # total number of loads and stores executed
sim_total_loads  38192 # total number of loads executed
sim_total_stores 24438.0000 # total number of stores executed
sim_total_branches 43670 # total number of branches executed
sim_cycle        226785 # total simulation time in cycles
sim_IPC          0.9543 # instructions per cycle
sim_CPI          1.0478 # cycles per instruction
...
    
```

## output from simulators (contd)

```

...
i11.accesses     263449.0000 # total number of accesses
i11.hits         247798 # total number of hits
i11.misses       15651 # total number of misses
i11.replacements 15139 # total number of replacements
i11.writebacks   0 # total number of writebacks
i11.invalidations 0 # total number of invalidations
i11.miss_rate    0.0594 # miss rate (i.e., misses/ref)
i11.rep_rate     0.0575 # replacement rate (i.e., repls/ref)
i11.wb_rate      0.0000 # writeback rate (i.e., wrbks/ref)
i11.inv_rate     0.0000 # invalidation rate (i.e., invs/ref)
...
    
```

## output from simulators (contd)

```
dl1.accesses      56465.0000 # total number of accesses
dl1.hits          55886 # total number of hits
dl1.misses        579 # total number of misses
dl1.replacements  89 # total number of replacements
dl1.writebacks    81 # total number of writebacks
dl1.invalidations 0 # total number of invalidations
dl1.miss_rate     0.0103 # miss rate (i.e., misses/ref)
dl1.repl_rate     0.0016 # replacement rate (i.e., repls/ref)
dl1.wb_rate       0.0014 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate      0.0000 # invalidation rate (i.e., invs/ref)
...
```