Using the SimpleScalar Tool Set at UT-CS

Prof. Steve Keckler
skeckler@cs.utexas.edu
Version 1.0: 1/25/99

1 Introduction

The SimpleScalar Tool Set performs fast, flexible, and accurate simulations of a modern microprocessor. It implements the SimpleScalar instruction set, which is a derivative of the MIPS architecture (and therefore is closely related to the DLX instruction set). The tool set itself consists of a collection of microarchitecture simulators that emulate the microprocessor at different levels of detail and a port of the gcc compiler that generates SimpleScalar binaries. The simulators that we will use in this class are as follows.

- `sim-fast` - fast instruction interpreter, optimized for speed. This simulator does not account for the behavior of pipelines, caches, or any other part of the microarchitecture.
- `sim-safe` - slightly slower instruction interpreter, as it checks for memory alignment and memory access permission on all memory operations. This simulator can be used if the simulated program causes `sim-fast` to crash without explanation.
- `sim-profile` - instruction interpreter and profiler. This simulator keeps track of and reports dynamic instruction counts, instruction class counts, usage of address modes, and profiles of the text and data segments.
- `sim-cache` - memory system simulator. This simulator can emulate a system with multiple levels of instruction and data caches, each of which can be configured for different sizes and organizations. This simulator is ideal for fast cache simulation if the effect of cache performance on execution time is not needed.
- `sim-bpred` - branch predictor simulator. This tool can simulate difference branch prediction schemes and reports results such as prediction hit and miss rates. Like `sim-cache`, this does not simulate accurately the effect of branch prediction on execution time.
- `sim-outorder` - detailed microarchitectural simulator. This tool models in detail and out-of-order microprocessor, with all of the bells and whistles, including branch prediction, caches, and external memory. This simulator is highly parameterized and can emulate machines of varying numbers of execution units.

The canonical manual for SimpleScalar is contained in, *The SimpleScalar Tool Set, Version 2.0*, by Doug Burger and Todd Austin. In addition to describing the simulators and options in more detail, that document includes the SimpleScalar instruction set definition.

2 SimpleScalar on CS Machines

The SimpleScalar tools are currently installed on two the Sun Ultra 1 workstations (the cheese machines) located in the basement of Taylor Hall. The installation on the Linux PCs in Painter 1.44 (the hat machines) is ongoing and should be available shortly. SimpleScalar is text based and the tools can be accessed directly from the console or via remote login.

2.1 SimpleScalar Directory Structure

The SimpleScalar release for this class includes the simulator, compiler, and binutils binaries; a precompiled subset of the SPEC95 benchmarks suite, and the necessary input files for the benchmarks. The directory structure for the SimpleScalar tool set installation is as follows:

- `/p/bin/{ss_setup,ss_spec}`
  Utilities for running the SimpleScalar tool set.
- `/p/bin/{sim-*}`
  SimpleScalar simulator executables.
- `/p/bin/simplescalar/test`
  `SimpleScalar test programs.`
- `/p/bin/simplescalar/spec95/benchmarks`
  Subset of the precompiled Spec95 executables.
- `/p/bin/simplescalar/spec95/inputs`
  Test input files for the Spec95 benchmarks.
- `/p/doc/simplescalar`
  SimpleScalar documentation.
- `/p/include/simplescalar`
  Include files for SimpleScalar compiler and simulator builds.
- `/p/lib/simplescalar`
  Archive libraries for compilation of SimpleScalar programs.
- `/p/src/simplescalar`
  SimpleScalar source code.

All of the above directories are links into a physical directory structure that starts at:

- `/projects/cart/students`

Platform specific components are distributed throughout the physical installation and are in variou s `solaris` and `linux` directories. Users of the tool can ignore the physical directory structure and instead consider only the platform independent directory structure described above.

2.2 Running SimpleScalar

To run the SimpleScalar tool set, log on to one of the supported platforms and type the following command at the `csh>` prompt:

```
csh> source /p/bin/ss_setup
```

This shell script will add the `/p/bin` directory to your path. The two platforms that will be supported are `solaris` and `linux`. The `/p` directory structure enables the platform specific binaries to appear to be in the same place in both the linux and solaris environments. You are now ready to run a test of the tools. At the prompt, type:

```
csh> sim-safe /p/bin/simplescalar/test/test-printf.ss
```

This will run the test_printf.ss program on the fast SimpleScalar instruction interpreter and print the program output to the screen. You will see the results of the simulator, listed under `** simulation statistics **`, including the number of instructions executed, the total numbers of loads and stores, etc. You can run this program through the rest of the SimpleScalar simulators as well.

2.3 Configuration Files

The simulators that examine the gory details of the microarchitecture (`sim-profile, sim-cache, sim-bred, sim-outorder`), can all be parameterized. The parameter list can be seen by running the simulator with no arguments:

```
csh> sim-cache
```

The parameters can be set at the command line or in a configuration file. A configuration file can be specified using the `-config` option to the simulators. Configurations can be dumped to a file by the simulator using the `-dumpconfig` option.

```
csh> sim-profile -iprof test-printf.ss  /* generate instruction profile */

csh> sim-profile -dumpconfig profile.cfg  /* generate config file */

csh> sim-profile -config profile.cfg test-printf.ss  /* use config */
```

In this way you can generate a dumpfile with the default simulator parameters, edit the configuration file to change the parameters, and run future simulations without having to enter the parameters on the command line for each simulation.

2.4 Simulation Speeds

Table 1 shows the speeds of the different simulators running on the Sun and Linux platforms. Note that there is approximately 2 orders of magnitude in speed between the fastest and slowest simulators. You can approximate your running time by determining the dynamic instruction count using `sim-fast` and derating by the appropriate factor on the more detailed simulator you wish to use.

TABLE 1. Simulator execution rates on CS instructional computers (instructions/sec)

platform	`sim-fast`	`sim-profile`	`sim-cache`	`sim-bpred`	`sim-outorder`
Sun Ultra I/model 170 167MHz Ultrasparc (cshosts pubultra)	2-3M	700-900K	200-300K	700-800K	40-50K
Intel PC w/ Linux 200MHz PentiumPro (cshosts publinux)	3-4M	1.3-1.8M	280-460K	1.2-1.5M	60-140K

3 Using the precompiled SPEC 95 benchmarks

SimpleScalar is released with the precompiled SPEC95 benchmark suite, including the necessary input files. Since the benchmarks require long argument strings and a variety of different input files, a simple script (`ss_spec`) has been written to encapsulate the hairiness. To run one of the supported SPEC benchmarks, go to your private working directory and type the following:

```
csh> ss_spec <sim_name> <sim_options> <benchmark_name>
```

The `<sim_name>` argument specifies which simulator to run (`sim_fast`, `sim_profile`, etc.), while `<sim_options>` are the parameters to pass to the simulator, such as the configuration file. The

<benchmark_name> argument specifies which supported benchmark to run. Table 2 shows the supported benchmarks and their dynamic instruction count.

TABLE 2. Supported SPEC95 Benchmarks

Benchmark	Description	# instructions
gcc_test	126.gcc: compile genoutput.i	86.8M
compress_test	129.compress: compress 100 character file	3.7M
compress_train	129.compress: compress 10000 character file	35.8M
mgrid_test_2it	107.mgrid: 2 iterations	480.3M
fpppp_train	145.fpppp: 5 atoms	329.8M

For example, the following command runs the compress test through the instruction profiler:

```
csh> ss_spec sim-profile -brprof compress_test
```

The entire command that actually runs the simulator is printed by the ss_spec script so you can see how to run the benchmarks without the script if necessary.

4 Compiling your own programs

This section is under development.

5 Modifying the simulator source code

This section is under development.

Using the SimpleScalar Tool Set at UT-CS

Prof. Steve Keckler
skeckler@cs.utexas.edu
Version 1.0: 1/25/99

1 Introduction

The SimpleScalar Tool Set performs fast, flexible, and accurate simulations of a modern microprocessor. It implements the SimpleScalar instruction set, which is a derivative of the MIPS architecture (and therefore is closely related to the DLX instruction set). The tool set itself consists of a collection of microarchitecture simulators that emulate the microprocessor at different levels of detail and a port of the gcc compiler that generates SimpleScalar binaries. The simulators that we will use in this class are as follows.

- `sim-fast` - fast instruction interpreter, optimized for speed. This simulator does not account for the behavior of pipelines, caches, or any other part of the microarchitecture.
- `sim-safe` - slightly slower instruction interpreter, as it checks for memory alignment and memory access permission on all memory operations. This simulator can be used if the simulated program causes `sim-fast` to crash without explanation.
- `sim-profile` - instruction interpreter and profiler. This simulator keeps track of and reports dynamic instruction counts, instruction class counts, usage of address modes, and profiles of the text and data segments.
- `sim-cache` - memory system simulator. This simulator can emulate a system with multiple levels of instruction and data caches, each of which can be configured for different sizes and organizations. This simulator is ideal for fast cache simulation if the effect of cache performance on execution time is not needed.
- `sim-bpred` - branch predictor simulator. This tool can simulate difference branch prediction schemes and reports results such as prediction hit and miss rates. Like `sim-cache`, this does not simulate accurately the effect of branch prediction on execution time.
- `sim-outorder` - detailed microarchitectural simulator. This tool models in detail and out-of-order microprocessor, with all of the bells and whistles, including branch prediction, caches, and external memory. This simulator is highly parameterized and can emulate machines of varying numbers of execution units.

The canonical manual for SimpleScalar is contained in, *The SimpleScalar Tool Set, Version 2.0*, by Doug Burger and Todd Austin. In addition to describing the simulators and options in more detail, that document includes the SimpleScalar instruction set definition.

2 SimpleScalar on CS Machines

The SimpleScalar tools are currently installed on two the Sun Ultra 1 workstations (the cheese machines) located in the basement of Taylor Hall. The installation on the Linux PCs in Painter 1.44 (the hat machines) is ongoing and should be available shortly. SimpleScalar is text based and the tools can be accessed directly from the console or via remote login.

2.1 SimpleScalar Directory Structure

The SimpleScalar release for this class includes the simulator, compiler, and binutils binaries; a precompiled subset of the SPEC95 benchmarks suite, and the necessary input files for the benchmarks. The directory structure for the SimpleScalar tool set installation is as follows:

- `/p/bin/{ss_setup,ss_spec}`
  Utilities for running the SimpleScalar tool set.
- `/p/bin/{sim-*}`
  SimpleScalar simulator executables.
- `/p/bin/simplescalar/test`
  `SimpleScalar test programs.`
- `/p/bin/simplescalar/spec95/benchmarks`
  Subset of the precompiled Spec95 executables.
- `/p/bin/simplescalar/spec95/inputs`
  Test input files for the Spec95 benchmarks.
- `/p/doc/simplescalar`
  SimpleScalar documentation.
- `/p/include/simplescalar`
  Include files for SimpleScalar compiler and simulator builds.
- `/p/lib/simplescalar`
  Archive libraries for compilation of SimpleScalar programs.
- `/p/src/simplescalar`
  SimpleScalar source code.

All of the above directories are links into a physical directory structure that starts at:

- `/projects/cart/students`

Platform specific components are distributed throughout the physical installation and are in variou s `solaris` and `linux` directories. Users of the tool can ignore the physical directory structure and instead consider only the platform independent directory structure described above.

2.2 Running SimpleScalar

To run the SimpleScalar tool set, log on to one of the supported platforms and type the following command at the `csh>` prompt:

```
csh> source /p/bin/ss_setup
```

This shell script will add the `/p/bin` directory to your path. The two platforms that will be supported are `solaris` and `linux`. The `/p` directory structure enables the platform specific binaries to appear to be in the same place in both the linux and solaris environments. You are now ready to run a test of the tools. At the prompt, type:

```
csh> sim-safe /p/bin/simplescalar/test/test-printf.ss
```

This will run the test_printf.ss program on the fast SimpleScalar instruction interpreter and print the program output to the screen. You will see the results of the simulator, listed under `** simulation statistics **`, including the number of instructions executed, the total numbers of loads and stores, etc. You can run this program through the rest of the SimpleScalar simulators as well.

2.3 Configuration Files

The simulators that examine the gory details of the microarchitecture (`sim-profile, sim-cache, sim-bred, sim-outorder`), can all be parameterized. The parameter list can be seen by running the simulator with no arguments:

```
csh> sim-cache
```

The parameters can be set at the command line or in a configuration file. A configuration file can be specified using the `-config` option to the simulators. Configurations can be dumped to a file by the simulator using the `-dumpconfig` option.

```
csh> sim-profile -iprof test-printf.ss  /* generate instruction profile */

csh> sim-profile -dumpconfig profile.cfg  /* generate config file */

csh> sim-profile -config profile.cfg test-printf.ss  /* use config */
```

In this way you can generate a dumpfile with the default simulator parameters, edit the configuration file to change the parameters, and run future simulations without having to enter the parameters on the command line for each simulation.

2.4 Simulation Speeds

Table 1 shows the speeds of the different simulators running on the Sun and Linux platforms. Note that there is approximately 2 orders of magnitude in speed between the fastest and slowest simulators. You can approximate your running time by determining the dynamic instruction count using `sim-fast` and derating by the appropriate factor on the more detailed simulator you wish to use.

TABLE 1. Simulator execution rates on CS instructional computers (instructions/sec)

platform	`sim-fast`	`sim-profile`	`sim-cache`	`sim-bpred`	`sim-outorder`
Sun Ultra I/model 170 167MHz Ultrasparc (cshosts pubultra)	2-3M	700-900K	200-300K	700-800K	40-50K
Intel PC w/ Linux 200MHz PentiumPro (cshosts publinux)	3-4M	1.3-1.8M	280-460K	1.2-1.5M	60-140K

3 Using the precompiled SPEC 95 benchmarks

SimpleScalar is released with the precompiled SPEC95 benchmark suite, including the necessary input files. Since the benchmarks require long argument strings and a variety of different input files, a simple script (`ss_spec`) has been written to encapsulate the hairiness. To run one of the supported SPEC benchmarks, go to your private working directory and type the following:

```
csh> ss_spec <sim_name> <sim_options> <benchmark_name>
```

The `<sim_name>` argument specifies which simulator to run (`sim_fast`, `sim_profile`, etc.), while `<sim_options>` are the parameters to pass to the simulator, such as the configuration file. The

<benchmark_name> argument specifies which supported benchmark to run. Table 2 shows the supported benchmarks and their dynamic instruction count.

TABLE 2. Supported SPEC95 Benchmarks

Benchmark	Description	# instructions
gcc_test	126.gcc: compile genoutput.i	86.8M
compress_test	129.compress: compress 100 character file	3.7M
compress_train	129.compress: compress 10000 character file	35.8M
mgrid_test_2it	107.mgrid: 2 iterations	480.3M
fpppp_train	145.fpppp: 5 atoms	329.8M

For example, the following command runs the compress test through the instruction profiler:

```
csh> ss_spec sim-profile -brprof compress_test
```

The entire command that actually runs the simulator is printed by the ss_spec script so you can see how to run the benchmarks without the script if necessary.

4 Compiling your own programs

This section is under development.

5 Modifying the simulator source code

This section is under development.