Introduction to SimpleScalar

SimpleScalar (http://www.simplescalar.com) is an open source computer architecture simulator. SimpleScalar is a set of tools that model a virtual computer system with CPU, Cache and Memory Hierarchy. Using the SimpleScalar tools, users can build modeling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. In addition to simulators, the SimpleScalar tool set includes performance visualization tools, statistical analysis resources, and debug and verification infrastructure.

Supported Instruction Set Architectures (ISA)

SimpleScalar can simulate Alpha and PISA (Portable ISA). The tool set takes binaries compiled for the SimpleScalar architecture and simulates their execution on one of several provided processor simulators. The machine running SimpleScalar is called the Host machine or Host while the ISA that one is targeting such as Alpha or PISA is called Target. In this course, we will configure the SimpleScalar to simulate Alpha only

Installing SimpleScalar and cross-compiler

Simplescalar currently only works on Linux machines. We do not support running this simulator on other OS such as Windows and Mac.

1) Get SimpleScalar and cross-compiler, and setup the environment

Download the SimpleScalar and cross-compiler from following two links

https://drive.google.com/open?id=0BxgRHQKhRziWbGFWaGdNOW9jZUE
https://drive.google.com/open?id=0BxgRHQKhRziWbFlfcWR6R2lFOEE

Create the folder "cse420" under your home folder, move the files you just downloaded into it, and then decompress them.

```
cd ~
mkdir cse420
cd cse420
mv ~/Download/simplesim-3v0e.tgz .
mv ~/Download/gcc-alpha-osf.tar.gz .
```

```
tar zxvf simplesim-3v0e.tgz
tar zxvf gcc-alpha-osf.tar.gz
```

Two folders, simplesim-3.0 and gcc, will be created. simplesim-3.0 is the simplescalar simulator while gcc is the cross-compiler.

Please strictly follow the path we use above. Otherwise, you have to open the file `gcc/local/alpha-dec-osf4/bin/as`, and change the path in the first two lines based on where you decompress your cross-compiler.

Then, add the `$HOME/cse420/gcc/local/bin` to the system PATH variable.

```
echo export PATH="\$HOME/cse420/gcc/local/bin:\$PATH" >> ~/.bashrc
source ~/.bashrc
```

This allows us using cross-compiler command without indicating the path such as `/home/<user name>/cse420/gcc/local/bin`

2) Build SimpleScalar for Alpha ISA simulation.

You need to have common compile tools such as "gcc" and "make" installed in your machine. Using following command to install through OS predefined repository if you have not.

```
sudo apt-get install gcc make g++   # For Ubuntu and Debian users
yum install gcc make g++            # For RedHat and CentOS users
```

Then, go to SimpleScalar's folder and build it

```
cd simplesim-3.0
make config-alpha && make
```

Upon finishing, you will see files `sim-fast`, `sim-cache`, …, and `sim-outorder` are generated under current folder.

sim-outorder is a detailed micro-architectural 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. We will use sim-outorder throughout all the program assignments.

3) Prepare you test source code and cross-compile it for Alpha architecture

Assuming that we create a C source code called "test.c" under folder cse420, we can cross-compile it by following commands.

```
cd ~/cse420
alpha-dec-osf4-gcc test.c -o test
```

Since our cross-compiler does not support C++, do not use any C++ syntax such as cout, class, or include C++ header such as <iostream>.

NOTE: In case you encounter an error "bash: /path/to/gcc/local/bin/alpha-dec-osf4-gcc :No such file or directory" for no reason, type "sudo apt-get install libc6:i386" in the terminal. This error might arise due to running of a 32-bit program on a 64-bit machine.

4) Run the program using simplescalar.

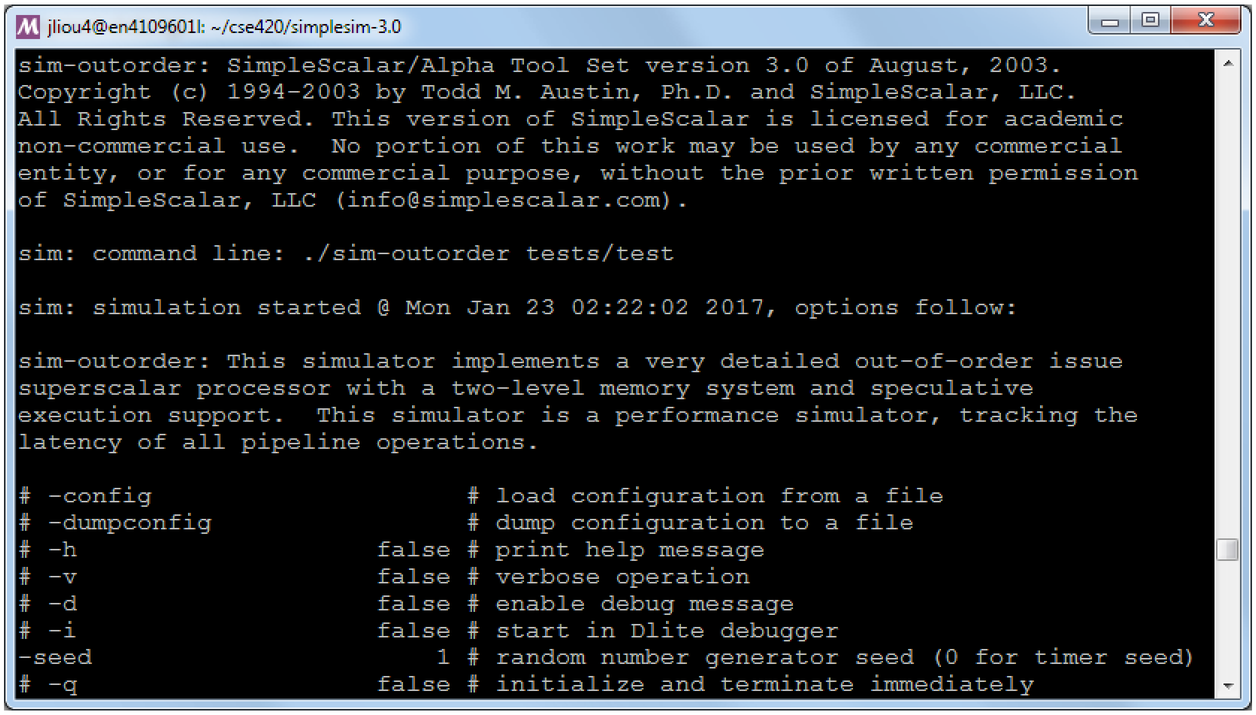
We will use the program we create from last step.

```
cd ~/cse420/simplesim-3.0
./sim-outorder ~/cse420/test
```

A bunch of information will be printed in the terminal. You might want to redirect those information to a file. Such as

```
./sim-outorder ~/cse420/test > output.txt
```

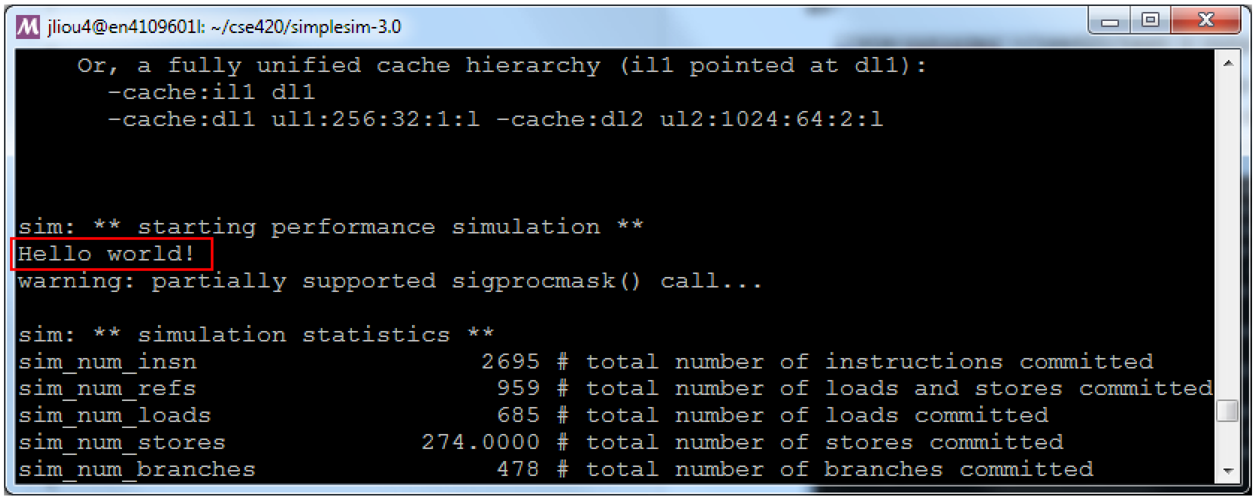
The output can be roughly split into 3 parts. The first part states the system configuration used for this simulation like following.



jliou4@en4109601l: ~/cse420/simplesim-3.0

```
sim-outorder: SimpleScalar/Alpha Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: ./sim-outorder tests/test

sim: simulation started @ Mon Jan 23 02:22:02 2017, options follow:

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.

# -config                          # load configuration from a file
# -dumpconfig                      # dump configuration to a file
# -h                         false # print help message
# -v                         false # verbose operation
# -d                         false # enable debug message
# -i                         false # start in Dlite debugger
-seed                            1 # random number generator seed (0 for timer seed)
# -q                         false # initialize and terminate immediately
```

The second part records your program's standard output. If your program trying to print anything on the terminal, it will show here. For example, in my program, it prints "Hello world!", so the output will looks like

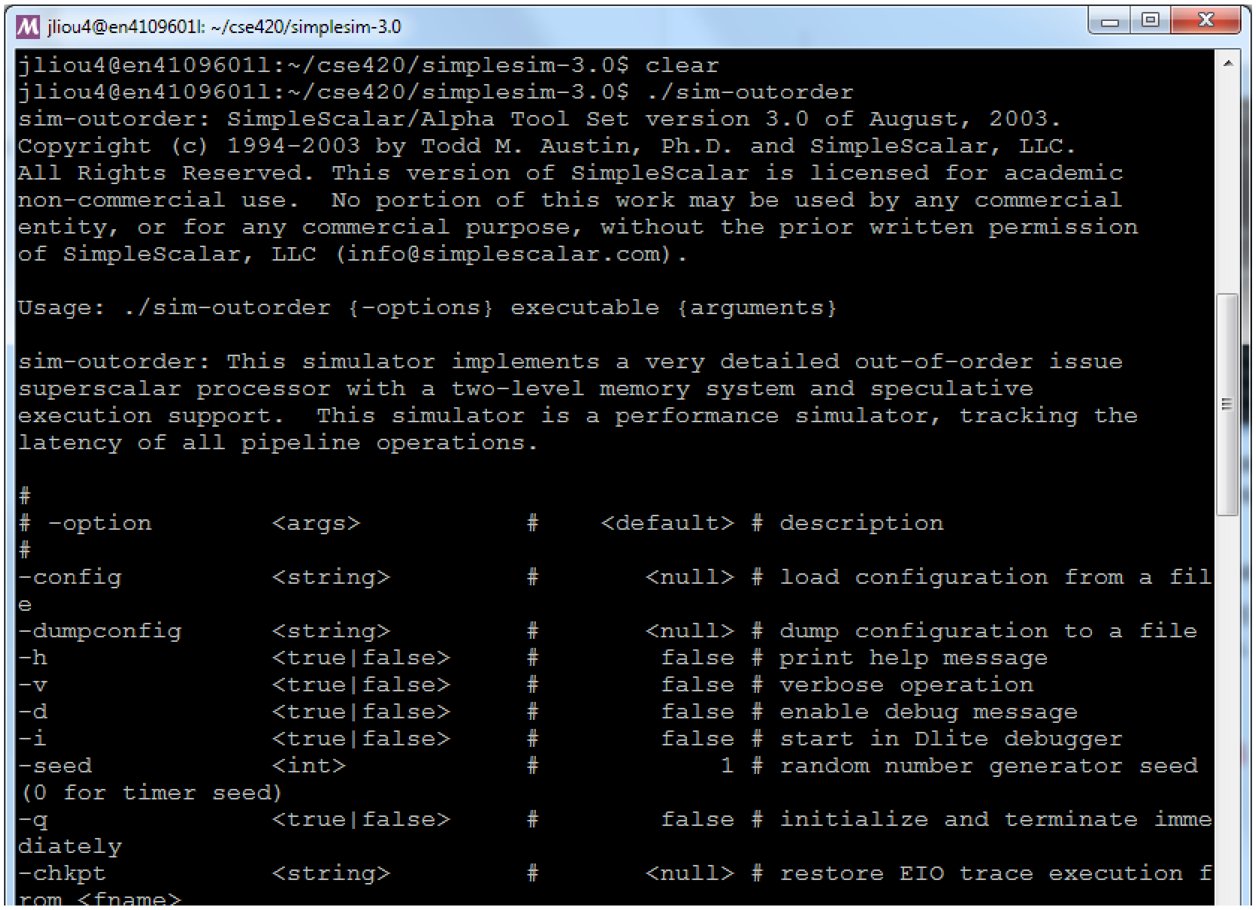


```
    Or, a fully unified cache hierarchy (il1 pointed at dl1):
      -cache:il1 dl1
      -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l



sim: ** starting performance simulation **
Hello world!
warning: partially supported sigprocmask() call...

sim: ** simulation statistics **
sim_num_insn                   2695 # total number of instructions committed
sim_num_refs                    959 # total number of loads and stores committed
sim_num_loads                   685 # total number of loads committed
sim_num_stores             274.0000 # total number of stores committed
sim_num_branches                478 # total number of branches committed
```

The third part is the simulated system performance result. You could observe bunch of information including cache hit/miss count, total instruction count, total simulated cycles, and so on.

5) Configure the simulation environment

sim-outorder is highly parametrized. You could view all available flags and settings by simply executing sim-outorder without any flags. Like following:



```
jliou4@en4109601l:~/cse420/simplesim-3.0$ clear
jliou4@en4109601l:~/cse420/simplesim-3.0$ ./sim-outorder
sim-outorder: SimpleScalar/Alpha Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

Usage: ./sim-outorder {-options} executable {arguments}

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.

#
# -option          <args>          #     <default> # description
#
-config            <string>        #        <null> # load configuration from a fil
e
-dumpconfig        <string>        #        <null> # dump configuration to a file
-h                 <true|false>    #         false # print help message
-v                 <true|false>    #         false # verbose operation
-d                 <true|false>    #         false # enable debug message
-i                 <true|false>    #         false # start in Dlite debugger
-seed              <int>           #             1 # random number generator seed
 (0 for timer seed)
-q                 <true|false>    #         false # initialize and terminate imme
diately
-chkpt             <string>        #        <null> # restore EIO trace execution f
rom <fname>
```

In addition to the descriptions of each flags, it also mentions what the default value is for each setting. Those are the values automatically applied to the simulator if you did not explicitly set them.

Here is an example simulating a system with multi-level cache hierarchy.

```
./sim-outorder  -cache:il1  il1:128:64:1:l  -cache:il2  il2:128:64:4:l  -cache:dl1 dl1:256:32:1:l -cache:dl2 dl2:1024:64:2:l ~/cse420/test
```

The flag `-cache:il1 il1:128:64:1:l` means

L1 instruction cache:	name is il1
	128 sets,
	block size is 64 bytes,
	direct map(1 way),
	LRU policy(l)

As you can see, the command will become much longer as more system setting are put together. To avoid such tedious, long command, SimpleScalar also supports reading config file for system configuration by using -config. There is a great config file example in simplesim-3.0/config/default.cfg. Here is an example using config file to run the simulator.

```
./sim-outorder -config config/default.cfg ~/cse420/test
```

For more information, please google “sim-outorder simplescalar”.