SimpleScalar Tutorial

COMP4611 Tutorial 5

Oct 15th-19th

1

Outline

- Introduction to SimpleScalar
- SimpleScalar installation and practice
- PISA *objdump* installation and practice
- Instruction count and CPI calculation

2

What is SimpleScalar

- A tool set for users to build applications that simulate real programs running on a range of modern processors and systems
- SimpleScalar tool set includes a set of sample simulators to simulate different operations of processors (i.e. branch prediction)

3

Install Cygwin

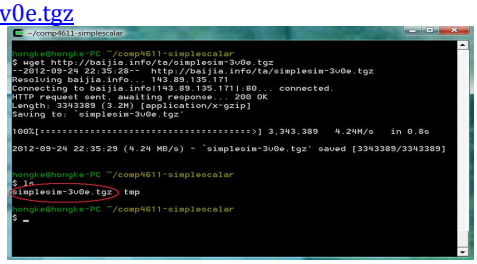
- We will run SimpleScalar on Cygwin
- Go to http://cygwin.com/install.html to install Cygwin
  - ftp.iij.ad.jp(ftp) as a recommended mirror site
  - Select *Category* (the default) package installation option
  - Make sure *gcc*, *make*, *wget, tar* are installed

4

Where to get SimpleScalar

- Open Cygwin to download SimpleScalar:

  $ wget http://baijia.info/ta/simplesim-3v0e.tgz

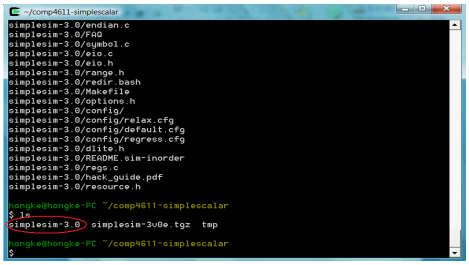


5

Where to get SimpleScalar

- Extract the SimpleScalar package:

  $ tar –xzvf simplesim-3v0e.tgz



6

How to install SimpleScalar

- Configure the installation target:

  $ cd simplesim-3.0/

  $ make config-pisa

```
~/comp4611-simplescalar/simplesim-3.0
hongke@hongke-PC ~/comp4611-simplescalar
$ cd simplesim-3.0/

hongke@hongke-PC ~/comp4611-simplescalar/simplesim-3.0
$ make config-pisa
rm -f config.h machine.h machine.c machine.def loader.c symbol.c syscall.c
ln -s target-pisa/config.h config.h
ln -s target-pisa/pisa.h machine.h
ln -s target-pisa/pisa.c machine.c
ln -s target-pisa/pisa.def machine.def
ln -s target-pisa/loader.c loader.c
ln -s target-pisa/symbol.c symbol.c
ln -s target-pisa/syscall.c syscall.c
rm -f tests
ln -s tests-pisa tests

hongke@hongke-PC ~/comp4611-simplescalar/simplesim-3.0
$
```


What is PISA and Alpha

- SimpleScalar can simulate programs in Alpha or PISA binary
- PISA (Portable ISA) instruction set is a simple MIPS-like instruction set
- Alpha ISA is a 64-bit RISC ISA

How to install SimpleScalar

- Compile the source code of SimpleScalar:

  $ make

```
~/comp4611-simplescalar/simplesim-3.0
gcc `./sysprobe -flags` -DDEBUG -O0 -g -Wall   -c sim-profile.c
gcc -o sim-profile `./sysprobe -flags` -DDEBUG -O0 -g -Wall   sim-profile.o main
.o syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o eval.o options.o
 stats.o eio.o range.o misc.o machine.o libexo/libexo.a `./sysprobe -libs` -lm
gcc `./sysprobe -flags` -DDEBUG -O0 -g -Wall   -c sim-cache.c
gcc `./sysprobe -flags` -DDEBUG -O0 -g -Wall   -c cache.c
gcc -o sim-cache `./sysprobe -flags` -DDEBUG -O0 -g -Wall   sim-cache.o cache.o
main.o syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o eval.o optio
ns.o stats.o eio.o range.o misc.o machine.o libexo/libexo.a `./sysprobe -libs` -
lm
gcc `./sysprobe -flags` -DDEBUG -O0 -g -Wall   -c sim-outorder.c
sim-outorder.c: In function `ruu_dispatch':
sim-outorder.c:3723: warning: unused variable `temp_qword'
sim-outorder.c: In function `sim_main':
sim-outorder.c:4455: warning: unused variable `temp_qword'
gcc `./sysprobe -flags` -DDEBUG -O0 -g -Wall   -c resource.c
gcc `./sysprobe -flags` -DDEBUG -O0 -g -Wall   -c ptrace.c
gcc -o sim-outorder `./sysprobe -flags` -DDEBUG -O0 -g -Wall   sim-outorder.o ca
che.o bpred.o resource.o ptrace.o main.o syscall.o memory.o regs.o loader.o endi
an.o dlite.o symbol.o eval.o options.o stats.o eio.o range.o misc.o machine.o li
bexo/libexo.a `./sysprobe -libs` -lm
my work is done here.
hongke@hongke-PC ~/comp4611-simplescalar/simplesim-3.0
$
```


How to verify SimpleScalar works

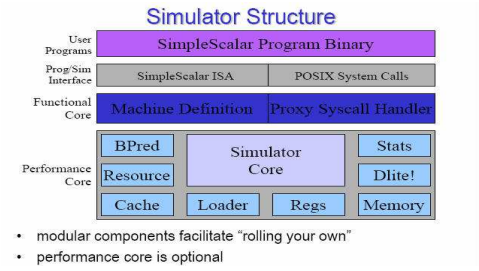
- Verify if the installation is successful:

  $ make sim-tests

```
~/comp4611-simplescalar/simplesim-3.0
../sim-outorder -redir:prog results/anagram.eio-progout \
                -redir:sim results/anagram.eio-simout  \
                eio.little/anagram.eio
../sim-outorder -redir:prog results/test-math.eio-progout \
                -redir:sim results/test-math.eio-simout  \
                eio.little/test-math.eio
../sim-outorder -redir:prog results/test-printf.eio-progout \
                -redir:sim results/test-printf.eio-simout  \
                eio.little/test-printf.eio
../sim-outorder -redir:prog results/test-fmath.eio-progout \
                -redir:sim results/test-fmath.eio-simout  \
                eio.little/test-fmath.eio
../sim-outorder -redir:prog results/test-llong.eio-progout \
                -redir:sim results/test-llong.eio-simout  \
                eio.little/test-llong.eio
../sim-outorder -redir:prog results/test-lswlr.eio-progout \
                -redir:sim results/test-lswlr.eio-simout  \
                eio.little/test-lswlr.eio
make[2]: Leaving directory `/home/hongke/comp4611-simplescalar/simplesim-3.0/tes
ts-pisa'
make[1]: Leaving directory `/home/hongke/comp4611-simplescalar/simplesim-3.0/tes
ts-pisa'

hongke@hongke-PC ~/comp4611-simplescalar/simplesim-3.0
$
```


Structure of SimpleScalar



- modular components facilitate "rolling your own"
- performance core is optional

Where to get SimpleScalar program binary?

Exercise

- How to install SimpleScalar for Alpha binary?
  - Clean the previous installation for PISA binary

    $ cd simplesim-3.0

    $ make clean
  - Configure the installation target:

    $ make config-alpha
  - Compile the source code
  - Verify the installation

Benchmark on SimpleScalar

- Download some benchmark programs at http://www.eecs.umich.edu/~taustin/eecs573_public/instruct-progs.tar.gz
- Extract the benchmark package:

  $ tar –xzvf instruct-progs.tar.gz

  $ mv benchmarks simplesim-3.0/

  $ cd simplesim-3.0

  $ ls benchmarks/

13

Benchmark on SimpleScalar

- Run GCC Alpha binary on SimpleScalar

  $ cd benchmarks

  $ ../sim-safe cc1.alpha -O 1stmt.i

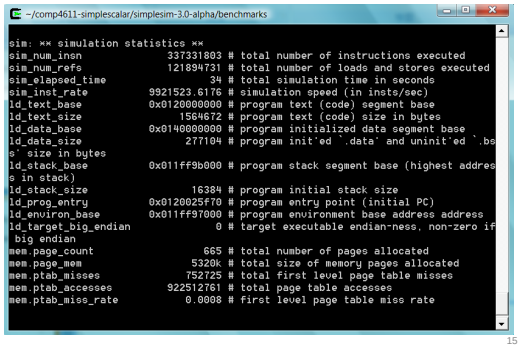
  simulator program — Alpha binary — Input file

- Compare the simulation result:

  $ diff 1stmt.s 1stmt.s.ref

14

Simulation Summary



```
~/comp4611-simplescalar/simplesim-3.0-alpha/benchmarks

sim: ** simulation statistics **
sim_num_insn              337331803 # total number of instructions executed
sim_num_refs              121894731 # total number of loads and stores executed
sim_elapsed_time                 34 # total simulation time in seconds
sim_inst_rate          9921523.6176 # simulation speed (in insts/sec)
ld_text_base           0x0120000000 # program text (code) segment base
ld_text_size                1564672 # program text (code) size in bytes
ld_data_base           0x0140000000 # program initialized data segment base
ld_data_size                 277104 # program init'ed `.data' and uninit'ed `.bs
s' size in bytes
ld_stack_base          0x011ff9b000 # program stack segment base (highest addres
s in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry          0x0120025f70 # program entry point (initial PC)
ld_environ_base        0x011ff97000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if
 big endian
mem.page_count                  665 # total number of pages allocated
mem.page_mem                  5320k # total size of memory pages allocated
mem.ptab_misses              752725 # total first level page table misses
mem.ptab_accesses         922512761 # total page table accesses
mem.ptab_miss_rate           0.0008 # first level page table miss rate
```

15

Benchmark on SimpleScalar

- Run GO Alpha binary on SimpleScalar

  $ cd benchmarks

  $ ../sim-safe go.alpha 50 9 2stone9.in > OUT

- Compare the simulation result:

  $ diff OUT go.out

16

Introduction to PISA *objdump*

- A tool for disassembling PISA binary code into PISA assembly code
- Manual of *objdump* available at http://sourceware.org/binutils/docs/binutils/objdump.html

17

Install PISA *objdump* and *gcc*

- Download *objdump* and *gcc* for PISA at http://www.eecg.toronto.edu/~moshovos/ACA06/files/ss-gcc.usrlocal.tar.bz
- Install *objdump* and *gcc* for PISA:

  ~~$ tar –cvjf ss-gcc.usrlocal.tar.bz~~

  $ tar –xvjf ss-gcc.usrlocal.tar.bz

  $ mv usr/local/ss /usr/local

  $ cd usr/loca/bin

  $ ls ss-gcc ss-objdump

18

PISA *objdump* Demo

- A simple C program – *hello.c*

```
int main()
{
    int i, a;
    a = 2;
    for (i = 0; i < 1000; i++)
        a++;
}
```

19

PISA *objdump* Demo

- Compile *hello.c* into PISA binary code – *hello*

  $ ./ss-gcc -o hello hello.c

- Disassemble *hello* into PISA assembly code – *hello.asm*

  $ ./ss-objdump -d hello > hello.asm

  $ less hello.asm

20

PISA assembly of main()

```
004001f0 <main> addiu/00 $sp[29],$sp[29],-32
004001f8 <main+8> sw/00 $ra[31],28($sp[29])
00400200 <main+10> sw/00 $s8[30],24($sp[29])
00400208 <main+18> addu/00 $s8[30],$zero[0],$sp[29]
00400210 <main+20> jal/00 00400468 <__main>
00400218 <main+28> addiu/00 $v0[2],$zero[0],2
00400220 <main+30> sw/00 $v0[2],20($s8[30])
00400228 <main+38> sw/00 $zero[0],16($s8[30])
00400230 <main+40> lw/00 $v0[2],16($s8[30])
00400238 <main+48> slti/00 $v1[3],$v0[2],1000
00400240 <main+50> bne/00 $v1[3],$zero[0],00400250 <main+60>
00400248 <main+58> j/00 00400298 <main+a8>
00400250 <main+60> lw/00 $v1[3],20($s8[30])
```

21

PISA assembly of main()

```
00400258 <main+68> addiu/00 $v0[2],$v1[3],1
00400260 <main+70> addu/00 $v1[3],$zero[0],$v0[2]
00400268 <main+78> sw/00 $v1[3],20($s8[30])
00400270 <main+80> lw/00 $v1[3],16($s8[30])
00400278 <main+88> addiu/00 $v0[2],$v1[3],1
00400280 <main+90> addu/00 $v1[3],$zero[0],$v0[2]
00400288 <main+98> sw/00 $v1[3],16($s8[30])
00400290 <main+a0> j/00 00400230 <main+40>
00400298 <main+a8> addu/00 $sp[29],$zero[0],$s8[30]
004002a0 <main+b0> lw/00 $ra[31],28($sp[29])
004002a8 <main+b8> lw/00 $s8[30],24($sp[29])
004002b0 <main+c0> addiu/00 $sp[29],$sp[29],32
004002b8 <main+c8> jr/00 $ra[31]
```

22

hello on SimpleScalar

- Simulate *hello* binary on SimpleScalar

  $ cp hello ~/simplesim-3.0

  $ cd ~/simplesim-3.0

  $ ./sim-safe –v hello &> hello.ss

  $ less hello.ss

  $ cat hello.ss | grep 'bne' | wc

  $ cat hello.ss | grep 'j/' | wc

23

Instruction Statistics

- ***Static instruction statistics*** is the statistics about the program's binary code (i.e. how many instructions are there in the program)

- ***Dynamic instruction statistics*** is the statistics of the dynamic instruction flow fetched and executed by the processor

24

Instruction Statistics of *hello*

- What is the static instruction count in main() of *hello*?
- What is one instruction that is executed most frequently by processors in main() of *hello*?

25

Calculate CPI of *hello*

- For a given program, ***CPI = Total program execution cycles / Instruction count***
- Is ***Instruction count*** equal to static or dynamic instruction count?
- How to get ***Total program execution cycles***? (Hint: SimpleScalar summary)

26

References

- SimpleScalar LLC: www.simplescalar.com
- Setting up Cygwin: cygwin.com/cygwin-ug-net/setup-net.html
- Introduction to SimpleScalar: www.ecs.umass.edu/ece/koren/architecture/Simplescalar/SimpleScalar_introduction.htm
- GCC port for SimpleScalar: www.eecg.toronto.edu/~moshovos/ACA06
- GNU *objdump:* sourceware.org/binutils/docs/binutils/objdump.html

27