SimpleScalar and SPEC 2000 installation instructions

Urvashi Jouhari
Department of Computer Engineering and Computer Science, California State University Long Beach
{urvashi.jouhri@student.csulb.edu}

***Abstract-* SimpleScalar simulation toolsets are the most commonly used computer architecture simulation simulator. They provide users the capability to run experiments in an easy and efficient manner to determine the performance of a processor under defined conditions. SPEC 2000 is one of the benchmark suites developed by the Standard Performance Evaluation Corporation (SPEC). Even though newer models have overtaken SPEC 2000, the SPEC 2000 serves as an efficient tool for basic evaluations of the processor performance. This paper aims to provide the installation instructions for installing the SimpleScalar toolsets and SPEC 2000 benchmark suite on a Linux platform. In particular, the instructions are based on modeling the behavior of the cache for analysis of cache performance but they may be extended for other simulation experiments.**

***Keywords* – Cache, SimpleScalar, SPEC 2000, Installation**

I. INTRODUCTION

The SimpleScalar tool set is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. [1] Todd Austin initially developed the SimpleScalar tool set with the assistance of Doug Burger and Guri Sohi. SimpleScalar tool sets are now developed and supported by the SimpleScalar LLC. SimpleScalar tools permit users to setup experimental simulations on processors. This includes modeling of the cache in terms of block size, L1-cache size, L2-cache size, associativity and replacement policies. SimpleScalar also contains tools for debugging, verifying infrastructure and resources for statistical analysis. SimpleScalar simulators can emulate the Alpha, PISA, ARM, and x86 instruction sets. [1] SimpleScalar tools can be built on either UNIX or Windows NT-based Operating Systems. Most 32-bit and 64-bit systems support it. However the most commonly used operating system is the Linux (64-bit) operating system.
SPEC CPU2000, commonly known as SPEC 2000 is a standardized CPU intensive benchmark suite. It was designed by Standard Performance Evaluation Corporation (SPEC) to provide a comparative measure of compute intensive performance across the widest practical range of hardware. [2] Source code benchmarks in the SPEC 2000 are developed via real user applications. These benchmarks prove as a base that permit the testing and analysis of processors, memories and compilers based on defined criteria.

This paper aims to provide the installation instructions required to build the SimpleScalar tool sets and set up the SPEC 2000 benchmark suite in order to run simulations on a Linux (64-bit) operating system. The installation of both the SimpleScalar tool sets and the SPEC 2000 benchmark suite were tested on the Linux platform, Ubuntu 12.10. This paper is organized as follows. Section II explains the installation of SimpleScalar manually. Section III explains the installation of SimpleScalar using a build file. Section IV provides the instructions required to execute the SPEC 2000 benchmarks. Examples to the experimental results along with their output files are included in Sections V and Section VI concludes the paper.

II. INSTALLATION OF SIMPLESCALAR MANUALLY

1. Download the necessary Source code files.

   Simpletools-2v0.tgz
   Simplesim-3v0d.tar.gz
   Simpleutils-990811.tar.gz
   Gcc-2.7.2.3.ss.tar.gz

   The files are available at www.SimpleScalar.com
   Alternatively, the files are uploaded on www.stanford.edu/~urvashi/simpletools

2. Under the home directory, create a directory SimpleScalar and copy the four source code files into that directory.

3. Launch the 'Terminal' and type:
```
$ export HOST=i686-pc-linux
$ export IDIR=/home/USER_NAME/SimpleScalar
$ export TARGET=sslittle-na-sstrix
$ mkdir $IDIR
$ cd $IDIR
```
   This creates the directory "SimpleScalar". The terminal should now display:
```
 USER_NAME@ubuntu /SimpleScalar$
```

4. Copy the four source codes files in the "SimpleScalar" directory.

5. Installing Simpletools
   In the 'Terminal' type:
```
$ cd $IDIR
$ tar xzvf simpletools-2v0.tgz
$ rm -rf gcc-2.6.3
```

6. Installing SimpleUtils
   In the 'Terminal' type:
```
$ tar xzvf simpleutils-990811.tar.gz
$ cd simpleutils-990811
```

7. Now, before proceeding, some errors need to be fixed

   In directory "ld" find file ldlex.l and replace all instances of
   yy_current_buffer with YY_CURRENT_BUFFER.

   This could be done manually by editing the file OR
   In the 'Terminal' type:
```
$ find . -type f -print0 | xargs -0 sed -i -e 's,yy_current_buffer,YY_CURRENT_BUFFER,g'
$ ./configure –host=$HOST –target=$TARGET –with-gnu-as –with-gnu-ld –prefix=$IDIR
$ make
$ make install
```

8. Installing Simplesim Simulator

   In the 'Terminal' type:
```
$ cd $IDIR
$ tar xzvf simplesim-3v0d.tgz
$ cd simplesim-3.0
$ make config-pisa
$ make
```
   The installation of the simplesim simulator may be tested by the following command:
```
$ ./sim-safe tests/bin.little/test-math
```

9. Installing GCC Cross-Compiler

   In the 'Terminal' type:
```
$ cd $IDIR
$ tar xzvf gcc-2.7.2.3.ss.tar.gz
$ cd gcc-2.7.2.3
$ export PATH=$PATH:/home/USER_NAME/SimpleScalar/sslittle-na-sstrix/bin
$ ./configure –host=$HOST –target=$TARGET –with-gnu-as –with-gnu-ld –prefix=$IDIR
```

10. Again, a few errors need to be fixed.
    In the 'Terminal' type:
    ```
    $ make
    ```

    - Fixing Error 1
      In the 'Terminal' type:
      ```
      $ chmod +w protoize.c
      $ gedit protoize.c
      ```

      Once protoize.c opens in gedit, edit line 60 of protoize.c, and replace

      #include <varargs.h> with #include <stdarg.h>

    - Fixing error 2
      ```
      $ chmod +w obstack.h
      $ gedit obstack.h
      ```

      Once obstack.h opens in gedit, edit obstack.h at line 341 and change

      ```
      *((void **)__o->next_free)++=((void *)datum);
      ```
      with

      ```
      *((void **)__o->next_free++)=((void *)datum);
      ```

11. To avoid parse errors while compiling, copy the patched files located in the patched directory.
    In the 'Terminal' type:

    ```
    $ cp ./patched/sys/cdefs.h ../sslittle-na-sstrix/include/sys/cdefs.h
    $ cp ../sslittle-na-sstrix/lib/libc.a ../lib/
    $ cp ../sslittle-na-sstrix/lib/crt0.o ../lib/
    ```

12. Download the "ar-ranlib.tar" file from www.stanford.edu/~urvashi/simpletools, un-tar it and place its contents in the directory "sslittle-na-sstrix/bin"

13. Confirm that these files have "execution & write permission"
    In the 'Terminal' type:

    ```
    $ cd $IDIR/sslittle-na-sstrix/bin
    ls –al
    ```

    If you see each file of this folder with write(w) & execution(x) permission then do nothing. Otherwise, assign them the permission by typing in the 'Terminal':

    ```
    chmod +w <filename>
    chmod +x <filename>
    ```

    ```
    $ make
    ```

14. A number of insn-output.c errors will be received. This can be solved by adding line breaks after each of the three FIXME (line 675, 750 and 823) in the insn-output.c

    In the 'Terminal' type:
    ```
    $ gedit insn-output.c
    ```

    Add the line breaks at lines 675,750 and 823

    In the 'Terminal' type:
    ```
    $ make
    ```

15. In objc/sendmsg.c, add the following code at line 35

    ```
    #define STRUCT_VALUE 0
    ```

    ```
    $ cd $IDIR/gcc-2.7.2.3/objc
    $ chmod +w sendmsg.c
    $ gedit sendmsg.c
    $ cd ..
    $ make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
    ```

16. The last make command leads to an error message that requires the cxxmain.c file to be edited.
    In the 'Terminal' type:
    ```
    $ chmod +w cxxmain.c
    $ gedit cxxmain.c
    ```

    In file cxxmain.c, that is remove the following lines (lines 2978-2979)

    ```
    char * malloc ();
    char * realloc ();
    ```

17. Now, In the 'Terminal' type:

    ```
    $ make LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
    $ make install  LANGUAGES="c c++" CFLAGS="-O" CC="gcc"
    ```

18. This completes the manual installation of SimpleScalar on Ubuntu.

19. The installation of SimpleScalar may be tested by using a simple C-program such as:

    ```
    #include <stdio.h>
    int main()
     {
            printf("Hello, World! \n");
            return 0;
    }
    ```

20. Save the C program as "hello.c" in the SimpleScalar directory.

21. In the 'Terminal' type:
    ```
    $IDIR/bin/sslittle-na-sstrix-gcc -o hello hello.c

    $IDIR/simplesim-3.0/sim-safe hello
    ```

22. If SimpleScalar is installed correctly, the program will execute successfully.

III. INSTALLATION OF SIMPLESCALAR USING A BUILD FILE

The following describes the procedure for the installation of SimpleScalar on the Linux platform Ubuntu. This was tested on the Ubuntu version 12.10.

1. Download the necessary source code files from www.stanford.edu/~urvashi/SimpleScalar/

2. Download the build file from www.stanford.edu/~urvashi/Build/

3. In the 'Home' directory create a directory 'SimpleScalar'

4. Copy the build file and the downloaded source code files into the 'SimpleScalar' directory.

5. In the 'Terminal' type:

   ```
   $ cd SimpleScalar
   $ chmod +x SimpleScalar-build.sh
   $ export IDIR=$PWD
   $ export HOST=i686-pc-linux
   $ export TARGET=sslittle-na-sstrix
   $ ./SimpleScalar-build.sh
   ```

   This should begin building the source code files.

6. Once the 'Terminal displays "USER_NAME@ubuntu /SimpleScalar$", the installation of SimpleScalar and its necessary code files has completed.

7. The installation of SimpleScalar may be tested by using a simple C-program such as:

   ```
   #include <stdio.h>
   int main()
    {
           printf("Hello, World! \n");
           return 0;
   }
   ```

8. Save the C program as "hello.c" in the SimpleScalar directory.

9. In the 'Terminal' type:
   ```
   $IDIR/bin/sslittle-na-sstrix-gcc -o hello hello.c

   $IDIR/simplesim-3.0/sim-safe hello
   ```

10. If SimpleScalar is installed correctly, the program will execute successfully.

IV. RUNNING TESTS ON SIM-CACHE

1. Once the SimpleScalar execution is complete, the performance of the cache may be tested using the "sim-cache" simulator.

2. Write a simple C-program such as:
   (This program was written to test the affect of different block sizes of a 8KB L1 cache)

```
#define SIZE_OF_ARRAY 16*1024

#define LOOPS 100

int main()
{
  char array[SIZE_OF_ARRAY];
  register int out_loop;
  register int in_loop;
  register int solution = 0;

  for (out_loop = 0; out_loop < NUM_LOOPS; out_loop++)
    {
      for (in_loop = 0; in_loop < ARRAY_SIZE; in_loop++)
              {
               solution *= array[in_loop];
              }
    }
  return solution;
}
```

3. Save the program as Cachetest.c

4. In the 'Terminal type:

   ```
   $IDIR/bin/sslittle-na-sstrix-gcc Cachetest.c

   $IDIR/simplesim-3.0/sim-cache -redir:prog /dev/null -redir:sim Cachetest_output a.out –cache:dl1 (cache specifications)
   ```

5. Once the program completes execution, it creates an output file Cachetest_output in the SimpleScalar directory with the simulation results.

6. A sample output is included in Section V.

IV. INSTALLING SPEC 2000

1. Open the SimpleScalar directory and **delete** the "simplesim-3.0" directory that was previously created.

2. Download the simplesim files http://www.stanford.edu/~urvashi/Simplesim/

3. Copy the "simplesim-3v0e.tgz" file into the SimpleScalar folder

4. Now in the terminal type

   ```
   $ cd SimpleScalar
   $  tar xzvf simplesim-3v0e.tgz

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

5. Download the SPEC 2000 necessary files from http://www.stanford.edu/~urvashi/Spec2000/

6. Copy the downloaded files in the SimpleScalar directory.

7. Open the 'Terminal' and type

```
cd SimpleScalar
$ tar xzvf spec2000binary.tgz
$ tar xzvf spec2000args.tgz
$ tar xzvf runscripts.tgz
```

8. Depending on the choice of the benchmark enter the following commands in the Terminal

```
$ cp runscripts/RUN(benchmark) spec2000args/(benchmark)
$ cd spec2000args/(benchmark)
$ ./RUN(benchmark) ../../simplesim-3.0/sim-outorder ../../spec2000binaries/(benchmark)00.peak.ev6 –config
(configuration file).config – redir:sim (output file name) .txt
```

9. Once the configuration file is done executing, a file with the specified output file name is created in the directory of the chosen benchmark.

V. SAMPLE OUTPUTS AND CONFIGRATION FILES

1. Sample output to "hello.c"

```
sim: simulation started @ Mon Apr  1 16:58:29 2013, options follow:

sim-safe: This simulator implements a functional simulator.  This
functional simulator is the simplest, most user-friendly simulator in the
SimpleScalar tool set.  Unlike sim-fast, this functional simulator checks
for all instruction errors, and the implementation is crafted for clarity
rather than speed.

# -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
# -chkpt               <null> # restore EIO trace execution from <fname>
# -redir:sim           <null> # redirect simulator output to file (non-interactive only)
# -redir:prog          <null> # redirect simulated program output to file
-nice                       0 # simulator scheduling priority
-max:inst                   0 # maximum number of inst's to execute

sim: ** starting functional simulation **
Hello, World! I think this crap actually works!!!!!!

sim: ** simulation statistics **
sim_num_insn                7778 # total number of instructions executed
sim_num_refs                4132 # total number of loads and stores executed
sim_elapsed_time               1 # total simulation time in seconds
sim_inst_rate          7778.0000 # simulation speed (in insts/sec)
ld_text_base          0x00400000 # program text (code) segment base
ld_text_size               71984 # program text (code) size in bytes
ld_data_base          0x10000000 # program initialized data segment base
ld_data_size                8352 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base         0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size              16384 # program initial stack size
```

```
ld_prog_entry           0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian             0 # target executable endian-ness, non-zero if big endian
mem.page_count                  26 # total number of pages allocated
mem.page_mem                  104k # total size of memory pages allocated
mem.ptab_misses                 26 # total first level page table misses
mem.ptab_accesses           489324 # total page table accesses
mem.ptab_miss_rate          0.0001 # first level page table miss rate
```

2. **Sample output to "Cachetest.c"**
   Cache parameters chosen: L1 size=8KB
   Block size=8bytes
   Associativity=1
   Replacement policy=LRU

```
sim: simulation started @ Mon Apr  1 17:04:59 2013, options follow:

sim-cache: This simulator implements a functional cache simulator.  Cache
statistics are generated for a user-selected cache and TLB configuration,
which may include up to two levels of instruction and data cache (with any
levels unified), and one level of instruction and data TLBs.  No timing
information is generated.

# -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
# -chkpt               <null> # restore EIO trace execution from <fname>
# -redir:sim     output_block # redirect simulator output to file (non-interactive only)
# -redir:prog       /dev/null # redirect simulated program output to file
-nice                       0 # simulator scheduling priority
-max:inst                   0 # maximum number of inst's to execute
-cache:dl1       dl1:1024:8:1:l # l1 data cache config, i.e., {<config>|none}
-cache:dl2       ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:il1       il1:256:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il2                  dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-tlb:itlb        itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
-tlb:dtlb        dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
-flush                  false # flush caches on system calls
-cache:icompress        false # convert 64-bit inst addresses to 32-bit inst equivalents
# -pcstat              <null> # profile stat(s) against text addr's (mult uses ok)

  The cache config parameter <config> has the following format:

    <name>:<nsets>:<bsize>:<assoc>:<repl>

    <name>   - name of the cache being defined
    <nsets>  - number of sets in the cache
    <bsize>  - block size of the cache
    <assoc>  - associativity of the cache
    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random
  Examples:   -cache:dl1 dl1:4096:32:1:l
           -dtlb dtlb:128:4096:32:r

 Cache levels can be unified by pointing a level of the instruction cache
 hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache
 configuration arguments.  Most sensible combinations are supported, e.g.,

  A unified l2 cache (il2 is pointed at dl2):
    -cache:il1 il1:128:64:1:l -cache:il2 dl2
    -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

  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 functional simulation w/ caches **

sim: ** simulation statistics **
sim_num_insn              14752723 # total number of instructions executed
sim_num_refs               1642123 # total number of loads and stores executed
sim_elapsed_time                 2 # total simulation time in seconds
sim_inst_rate         7376361.5000 # simulation speed (in insts/sec)
il1.accesses              14752723 # total number of accesses
il1.hits                  14752346 # total number of hits
il1.misses                     377 # total number of misses
il1.replacements               171 # total number of replacements
il1.writebacks                   0 # total number of writebacks
il1.invalidations                0 # total number of invalidations
il1.miss_rate               0.0000 # miss rate (i.e., misses/ref)
il1.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
il1.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses               1642193 # total number of accesses
dl1.hits                   1435736 # total number of hits
dl1.misses                  206457 # total number of misses
dl1.replacements            205433 # total number of replacements
dl1.writebacks                 639 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.1257 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.1251 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0004 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses                207473 # total number of accesses
ul2.hits                    206830 # total number of hits
ul2.misses                     643 # total number of misses
ul2.replacements                 0 # total number of replacements
ul2.writebacks                   0 # total number of writebacks
ul2.invalidations                0 # total number of invalidations
ul2.miss_rate               0.0031 # miss rate (i.e., misses/ref)
ul2.repl_rate               0.0000 # replacement rate (i.e., repls/ref)
ul2.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
itlb.accesses             14752723 # total number of accesses
itlb.hits                 14752717 # total number of hits
itlb.misses                      6 # total number of misses
itlb.replacements               0 # total number of replacements
itlb.writebacks                 0 # total number of writebacks
itlb.invalidations              0 # total number of invalidations
itlb.miss_rate             0.0000 # miss rate (i.e., misses/ref)
itlb.repl_rate             0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate               0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate              0.0000 # invalidation rate (i.e., invs/ref)
dtlb.accesses             1642193 # total number of accesses
dtlb.hits                 1642182 # total number of hits
dtlb.misses                    11 # total number of misses
dtlb.replacements               0 # total number of replacements
dtlb.writebacks                 0 # total number of writebacks
dtlb.invalidations              0 # total number of invalidations
dtlb.miss_rate             0.0000 # miss rate (i.e., misses/ref)
dtlb.repl_rate             0.0000 # replacement rate (i.e., repls/ref)
dtlb.wb_rate               0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate              0.0000 # invalidation rate (i.e., invs/ref)
ld_text_base           0x00400000 # program text (code) segment base
ld_text_size                23008 # program text (code) size in bytes
ld_data_base           0x10000000 # program initialized data segment base
ld_data_size                 4096 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base          0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size               16384 # program initial stack size
ld_prog_entry          0x00400140 # program entry point (initial PC)
ld_environ_base        0x7fff8000 # program environment base address address
ld_target_big_endian            0 # target executable endian-ness, non-zero if big endian
mem.page_count                 14 # total number of pages allocated
mem.page_mem                  56k # total size of memory pages allocated
mem.ptab_misses           1228814 # total first level page table misses
mem.ptab_accesses        61211958 # total page table accesses
mem.ptab_miss_rate         0.0201 # first level page table miss rate
```

3. **Sample configuration file:**
   Cache Parameters chosen : L1 size=8KB
   L2 size=256 KB

```
# load configuration from a file
# -config

# dump configuration to a file
# -dumpconfig

# print help message
# -h                    false

# verbose operation
# -v                    false

# enable debug message
# -d                    false

# start in Dlite debugger
# -i                    false

# random number generator seed (0 for timer seed)
-seed                       1
```

```
# initialize and terminate immediately
# -q                         false

# restore EIO trace execution from <fname>
# -chkpt                     <null>

# redirect simulator output to file (non-interactive only)
# -redir:sim                 <null>

# redirect simulated program output to file
# -redir:prog                <null>

# simulator scheduling priority
-nice                        0

# maximum number of inst's to execute
-max:inst                    0

# number of insts skipped before timing starts
-fastfwd                     0

# generate pipetrace, i.e., <fname|stdout|stderr> <range>
# -ptrace                    <null>

# instruction fetch queue size (in insts)
-fetch:ifqsize               4

# extra branch mis-prediction latency
-fetch:mplat                 3

# speed of front-end of machine relative to execution core
-fetch:speed                 1

# branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
-bpred                       2lev

# bimodal predictor config (<table size>)
-bpred:bimod           2048

# 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:2lev            1 256 8 0

# combining predictor config (<meta_table_size>)
-bpred:comb            1024

# return address stack size (0 for no return stack)
-bpred:ras                   4

# BTB config (<num_sets> <associativity>)
-bpred:btb             256 2

# speculative predictors update in {ID|WB} (default non-spec)
# -bpred:spec_update         <null>

# instruction decode B/W (insts/cycle)
-decode:width                4
```

```
# instruction issue B/W (insts/cycle)
-issue:width                4

# run pipeline with in-order issue
-issue:inorder              false

# issue instructions down wrong execution paths
-issue:wrongpath            true

# instruction commit B/W (insts/cycle)
-commit:width               4

# register update unit (RUU) size
-ruu:size                   32

# load/store queue (LSQ) size
-lsq:size                   32

# l1 data cache config, i.e., {<config>|none}
-cache:dl1          dl1:256:32:1:l

# l1 data cache hit latency (in cycles)
-cache:dl1lat               1

# l2 data cache config, i.e., {<config>|none}
-cache:dl2          ul2:4096:64:1:f

# l2 data cache hit latency (in cycles)
-cache:dl2lat               6

# l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1          il1:64:32:4:f

# l1 instruction cache hit latency (in cycles)
-cache:il1lat               1

# l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2                  dl2

# l2 instruction cache hit latency (in cycles)
-cache:il2lat               6

# flush caches on system calls
-cache:flush                false

# convert 64-bit inst addresses to 32-bit inst equivalents
-cache:icompress            false

# memory access latency (<first_chunk> <inter_chunk>)
-mem:lat            100 20

# memory access bus width (in bytes)
-mem:width                  8

# instruction TLB config, i.e., {<config>|none}
-tlb:itlb           itlb:16:4096:4:l
```

```
# data TLB config, i.e., {<config>|none}
-tlb:dtlb             dtlb:32:4096:4:l

# inst/data TLB miss latency (in cycles)
-tlb:lat                      100

# total number of integer ALU's available
-res:ialu                      2

# total number of integer multiplier/dividers available
-res:imult                     1

# total number of memory system ports available (to CPU)
-res:memport                   2

# total number of floating point ALU's available
-res:fpalu                     1

# total number of floating point multiplier/dividers available
-res:fpmult                    1

# profile stat(s) against text addr's (mult uses ok)
# -pcstat                 <null>

# operate in backward-compatible bugs mode (for testing only)
-bugcompat                  false

#command line parameters
-max:inst 100000000
-fastfwd 300000000
```

4. Sample output file created from execution of sample configuration file:

```
sim: simulation started @ Tue Apr 23 15:44:27 2013, 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
# -chkpt               <null> # restore EIO trace execution from <fname>
# -redir:sim        l1_8k.txt # redirect simulator output to file (non-interactive only)
# -redir:prog          <null> # redirect simulated program output to file
-nice                       0 # simulator scheduling priority
-max:inst           100000000 # maximum number of inst's to execute
-fastfwd            300000000 # number of insts skipped before timing starts
# -ptrace              <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
```

```
-fetch:ifqsize              4 # instruction fetch queue size (in insts)
-fetch:mplat                3 # extra branch mis-prediction latency
-fetch:speed                1 # speed of front-end of machine relative to execution core
-bpred                   2lev # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
-bpred:bimod             2048 # bimodal predictor config (<table size>)
-bpred:2lev        1 256 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:comb              1024 # combining predictor config (<meta_table_size>)
-bpred:ras                  4 # return address stack size (0 for no return stack)
-bpred:btb              256 2 # BTB config (<num_sets> <associativity>)
# -bpred:spec_update   <null> # speculative predictors update in {ID|WB} (default non-spec)
-decode:width               4 # instruction decode B/W (insts/cycle)
-issue:width                4 # instruction issue B/W (insts/cycle)
-issue:inorder          false # run pipeline with in-order issue
-issue:wrongpath         true # issue instructions down wrong execution paths
-commit:width               4 # instruction commit B/W (insts/cycle)
-ruu:size                  32 # register update unit (RUU) size
-lsq:size                  32 # load/store queue (LSQ) size
-cache:dl1       dl1:256:32:1:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat               1 # l1 data cache hit latency (in cycles)
-cache:dl2       ul2:4096:64:1:f # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat               6 # l2 data cache hit latency (in cycles)
-cache:il1       il1:64:32:4:f # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1lat               1 # l1 instruction cache hit latency (in cycles)
-cache:il2                dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2lat               6 # l2 instruction cache hit latency (in cycles)
-cache:flush            false # flush caches on system calls
-cache:icompress        false # convert 64-bit inst addresses to 32-bit inst equivalents
-mem:lat               100 20 # memory access latency (<first_chunk> <inter_chunk>)
-mem:width                  8 # memory access bus width (in bytes)
-tlb:itlb        itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
-tlb:dtlb        dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
-tlb:lat                  100 # inst/data TLB miss latency (in cycles)
-res:ialu                   2 # total number of integer ALU's available
-res:imult                  1 # total number of integer multiplier/dividers available
-res:memport                2 # total number of memory system ports available (to CPU)
-res:fpalu                  1 # total number of floating point ALU's available
-res:fpmult                 1 # total number of floating point multiplier/dividers available
# -pcstat              <null> # profile stat(s) against text addr's (mult uses ok)
-bugcompat              false # operate in backward-compatible bugs mode (for testing only)

  Pipetrace range arguments are formatted as follows:

    {{@|#}<start>}:{{@|#|+}<end>}

  Both ends of the range are optional, if neither are specified, the entire
  execution is traced.  Ranges that start with a `@' designate an address
  range to be traced, those that start with an `#' designate a cycle count
  range.  All other range values represent an instruction count range.  The
  second argument, if specified with a `+', indicates a value relative
  to the first argument, e.g., 1000:+100 == 1000:1100.  Program symbols may
  be used in all contexts.

    Examples:   -ptrace FOO.trc #0:#1000
                -ptrace BAR.trc @2000:
                -ptrace BLAH.trc :1500
                -ptrace UXXE.trc :
                -ptrace FOOBAR.trc @main:+278
  Branch predictor configuration examples for 2-level predictor:
    Configurations:   N, M, W, X
      N   # entries in first level (# of shift register(s))
      W   width of shift register(s)
      M   # entries in 2nd level (# of counters, or other FSM)
      X   (yes-1/no-0) xor history and address for 2nd level index
    Sample predictors:
      GAg     : 1, W, 2^W, 0
      GAp     : 1, W, M (M > 2^W), 0
      PAg     : N, W, 2^W, 0
      PAp     : N, W, M (M == 2^(N+W)), 0
      gshare  : 1, W, 2^W, 1
  Predictor `comb' combines a bimodal and a 2-level predictor.

  The cache config parameter <config> has the following format:

    <name>:<nsets>:<bsize>:<assoc>:<repl>

    <name>   - name of the cache being defined
    <nsets>  - number of sets in the cache
    <bsize>  - block size of the cache
    <assoc>  - associativity of the cache
    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

    Examples:   -cache:dl1 dl1:4096:32:1:l
                -dtlb dtlb:128:4096:32:r

  Cache levels can be unified by pointing a level of the instruction cache
  hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache
  configuration arguments.  Most sensible combinations are supported, e.g.,

    A unified l2 cache (il2 is pointed at dl2):
      -cache:il1 il1:128:64:1:l -cache:il2 dl2
      -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

    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: ** fast forwarding 300000000 insts **
warning: partially supported sigprocmask() call...
warning: partially supported sigaction() call...
warning: unsupported setsysinfo() call...
sim: ** starting performance simulation **

sim: ** simulation statistics **
sim_num_insn              100000000 # total number of instructions committed
sim_num_refs               40597581 # total number of loads and stores committed
sim_num_loads              30290152 # total number of loads committed
sim_num_stores         10307429.0000 # total number of stores committed
sim_num_branches             573315 # total number of branches committed
sim_elapsed_time                181 # total simulation time in seconds
sim_inst_rate           552486.1878 # simulation speed (in insts/sec)
sim_total_insn            100589340 # total number of instructions executed
```

```
sim_total_refs            40697693 # total number of loads and stores executed
sim_total_loads            30381427 # total number of loads executed
sim_total_stores       10316266.0000 # total number of stores executed
sim_total_branches          644751 # total number of branches executed
sim_cycle                281058104 # total simulation time in cycles
sim_IPC                     0.3558 # instructions per cycle
sim_CPI                     2.8106 # cycles per instruction
sim_exec_BW                 0.3579 # total instructions (mis-spec + committed) per cycle
sim_IPB                   174.4242 # instruction per branch
IFQ_count                996627534 # cumulative IFQ occupancy
IFQ_fcount               248554626 # cumulative IFQ full count
ifq_occupancy               3.5460 # avg IFQ occupancy (insn's)
ifq_rate                    0.3579 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                 9.9079 # avg IFQ occupant latency (cycle's)
ifq_full                    0.8844 # fraction of time (cycle's) IFQ was full
RUU_count               8113007208 # cumulative RUU occupancy
RUU_fcount               244149429 # cumulative RUU full count
ruu_occupancy              28.8659 # avg RUU occupancy (insn's)
ruu_rate                    0.3579 # avg RUU dispatch rate (insn/cycle)
ruu_latency                80.6547 # avg RUU occupant latency (cycle's)
ruu_full                    0.8687 # fraction of time (cycle's) RUU was full
LSQ_count               3704414490 # cumulative LSQ occupancy
LSQ_fcount                       0 # cumulative LSQ full count
lsq_occupancy              13.1802 # avg LSQ occupancy (insn's)
lsq_rate                    0.3579 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                36.8271 # avg LSQ occupant latency (cycle's)
lsq_full                    0.0000 # fraction of time (cycle's) LSQ was full
sim_slip               11956163669 # total number of slip cycles
avg_sim_slip              119.5616 # the average slip between issue and retirement
bpred_2lev.lookups          689757 # total number of bpred lookups
bpred_2lev.updates          573315 # total number of updates
bpred_2lev.addr_hits        467190 # total number of address-predicted hits
bpred_2lev.dir_hits         467887 # total number of direction-predicted hits (includes addr-hits)
bpred_2lev.misses           105428 # total number of misses
bpred_2lev.jr_hits             496 # total number of address-predicted hits for JR's
bpred_2lev.jr_seen             661 # total number of JR's seen
bpred_2lev.jr_non_ras_hits.PP           46 # total number of address-predicted hits for non-RAS JR's
bpred_2lev.jr_non_ras_seen.PP          163 # total number of non-RAS JR's seen
bpred_2lev.bpred_addr_rate    0.8149 # branch address-prediction rate (i.e., addr-hits/updates)
bpred_2lev.bpred_dir_rate    0.8161 # branch direction-prediction rate (i.e., all-hits/updates)
bpred_2lev.bpred_jr_rate    0.7504 # JR address-prediction rate (i.e., JR addr-hits/JRs seen)
bpred_2lev.bpred_jr_non_ras_rate.PP    0.2822 # non-RAS JR addr-pred rate (ie, non-RAS JR hits/JRs seen)
bpred_2lev.retstack_pushes          940 # total number of address pushed onto ret-addr stack
bpred_2lev.retstack_pops          608 # total number of address popped off of ret-addr stack
bpred_2lev.used_ras.PP          498 # total number of RAS predictions used
bpred_2lev.ras_hits.PP          450 # total number of RAS hits
bpred_2lev.ras_rate.PP    0.9036 # RAS prediction rate (i.e., RAS hits/used RAS)
il1.accesses             103400270 # total number of accesses
il1.hits                 100910644 # total number of hits
il1.misses                 2489626 # total number of misses
il1.replacements           2489370 # total number of replacements
il1.writebacks                   0 # total number of writebacks
il1.invalidations                0 # total number of invalidations
il1.miss_rate               0.0241 # miss rate (i.e., misses/ref)
il1.repl_rate               0.0241 # replacement rate (i.e., repls/ref)
il1.wb_rate                 0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
```

```
dl1.accesses              40598320 # total number of accesses
dl1.hits                  33198327 # total number of hits
dl1.misses                 7399993 # total number of misses
dl1.replacements           7399737 # total number of replacements
dl1.writebacks             3004849 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.1823 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.1823 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0740 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses              12894468 # total number of accesses
ul2.hits                  10862216 # total number of hits
ul2.misses                 2032252 # total number of misses
ul2.replacements           2028156 # total number of replacements
ul2.writebacks             1037055 # total number of writebacks
ul2.invalidations                0 # total number of invalidations
ul2.miss_rate               0.1576 # miss rate (i.e., misses/ref)
ul2.repl_rate               0.1573 # replacement rate (i.e., repls/ref)
ul2.wb_rate                 0.0804 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
itlb.accesses            103400270 # total number of accesses
itlb.hits                103400184 # total number of hits
itlb.misses                     86 # total number of misses
itlb.replacements               24 # total number of replacements
itlb.writebacks                  0 # total number of writebacks
itlb.invalidations               0 # total number of invalidations
itlb.miss_rate              0.0000 # miss rate (i.e., misses/ref)
itlb.repl_rate              0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
dtlb.accesses             40598409 # total number of accesses
dtlb.hits                 40266838 # total number of hits
dtlb.misses                 331571 # total number of misses
dtlb.replacements           331443 # total number of replacements
dtlb.writebacks                  0 # total number of writebacks
dtlb.invalidations               0 # total number of invalidations
dtlb.miss_rate              0.0082 # miss rate (i.e., misses/ref)
dtlb.repl_rate              0.0082 # replacement rate (i.e., repls/ref)
dtlb.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
sim_invalid_addrs                0 # total non-speculative bogus addresses seen (debug var)
ld_text_base          0x0120000000 # program text (code) segment base
ld_text_size               1056768 # program text (code) size in bytes
ld_data_base          0x0140000000 # program initialized data segment base
ld_data_size             198995952 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base         0x011ff9b000 # program stack segment base (highest address in stack)
ld_stack_size                16384 # program initial stack size
ld_prog_entry         0x0120015b30 # 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                5033 # total number of pages allocated
mem.page_mem                40264k # total size of memory pages allocated
mem.ptab_misses            7441536 # total first level page table misses
mem.ptab_accesses       3037148960 # total page table accesses
mem.ptab_miss_rate          0.0025 # first level page table miss rate
```

VI. CONCLUSION

The paper specifies installation instructions to the SimpleScalar toolset and SPEC2000 benchmark suite. Also, instructions for the modeling of cache using the sim-cache tool of SimpleScalar are described. In addition, instructions for modeling of cache with a configuration file using the SPEC 2000 benchmark are provided. Sample configuration files, programs and outputs are provided. All the necessary downloads and resources are made easily available.

VII. REFERENCES

1. http://www.simplescalar.com/
2. http://www.spec.org/cpu2000/
3. Doug Burger, Todd M. Austin, “SimpleScalar Tool Set, Version 2.0”
4. Todd Austin, “SimpleScalar Hacker’s guide”
5. Todd Austin, Dan Ernst, Eric Larson, Chris Weaver, RajDesikan, Ramadass Nagarajan, Jaehyuk Huh, Bill Yoder, Doug Burger, Steve Keckler, “SimpleScalar Tutorial”.
6. http://harryscode.blogspot.com/2008/10/installing-simplescalar.html
7. http://www.simplescalar.com/docs/install_guide_v2.txt