Simple Scalar

CSCE 4610 Computer Architecture

Instructor: Dr. Krishna Kavi
Grader: Shashank Adavally

Simple Scalar

- Simple Scalar is open source simulator copyright with Todd M. Austin.
- Detailed Micro architectural Modeling: Alpha, PISA (MIPS-like), ARM, x86 ISAs
- Multiple cache structures, speculative execution, out-of-order, branch prediction
- Downloadable
  - www.simplescalar.com
  - http://csrl.unt.edu/simplescalar/
- Simplescalar version 3 is installed at UNT on
  - watney.cse.unt.edu
Simple Scalar

- SimpleScalar location on watney.cse.unt.edu:
  SSDIR=/opt/Simplescalar

- 4 Versions of executable for PISA by cache type:

  $ SSDIR/ss_split
  $ SSDIR/ss_split_set
  $ SSDIR/ss_unified
  $ SSDIR/ss_unified_set

- Precompiled benchmarks will be saved in benchmarks folder.

Simple Scalar

- compiler chain is GNU tools ported to SimpleScalar
- Fortran codes are compiled with AT&T’s f2c
- libraries are GLIBC ported to SimpleScalar
Several simulators: outorder, fast, safe etc.

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

```
./results/qsort_1.sim

# -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 /home/mpich/nm0312/results/shant1 # 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
-fastfwd 0 # 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:mlplat 3 # extra branch mis-prediction latency
-fetch:speed 1 # speed of front-end of machine relative to execution core
-bpred nottaken # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
```
Simple Scalar

- ./results/qsort_1.sim

- 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
- commit:width 4 # instruction commit B/W (insts/cycle)
- ruu:size 16 # register update unit (RUU) size
- lsq:size 8 # load/store queue (LSQ) size
- cache:dl1 dl1:128:32:2:l # l1 data cache config, i.e., {<config>|none}
- cache:dl2 dl2:1024:32:1:l # l2 data cache config, i.e., {<config>|none}
- cache:dl1lat 1 # l1 data cache hit latency (in cycles)
- cache:dl2lat 10 # l2 data cache hit latency (in cycles)
- res:ialu 4 # total number of integer ALU's available
- res:imult1 1 # total number of integer multiplier/dividers available
- res:memport 2 # total number of memory system ports available (to CPU)
- res:fpalu 4 # total number of floating point ALU's available

The cache configuration 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

Example script

# minimal script to run simple scalar
SSDIR=/opt/simplescalar

Echo "Start of Script"

-redir:sim ~/results/qsort_1.sim $SSDIR/benchmarks/mibench/automotive/qsort/qsort_small
$SSDIR/benchmarks/mibench/automotive/qsort/input_small.dat >~/results/qsortt1.txt

Echo "End of Script"
What evaluations can be performed with existing sim-outorder PISA executables?

- Compare unified vs split cache effectiveness
- Compare performance of various set associativity settings
- Analyze programs with respect to cache hit/miss rates
- Tradeoffs between L1 cache size vs speed (latencies)
- Analyze differences in branch prediction settings
- Analyze differences in issue width
- Analyze differences in number/organization of FUs

Simple Scalar

- What evaluations can be performed by recompiling sim-outorder executables?
  - Analyze pipeline changes (depth, organization, etc.)
  - Change cache replacement, Branch predict algorithms

- What evaluations can be performed by recompiling sim-outorder, compiler, tools, libraries, benchmarks?
  - Compare different ISA’s
  - Add/Delete/Modify instructions being simulated

- Simplescalar does not support multiprocessor simulation. But it can be used as uni-processor with superscalar.