SimpleScalar Tool Set

prepared by Hussein al-Zoubi

Table of contents:

sim-bpred

```
sim-bpred: Version 2.0 of July, 1997.
Copyright (c) 1994-1997 by Todd M. Austin.  All Rights Reserved.
 Usage: sim-bpred {-options} executable {arguments}
 sim-bpred: This simulator implements a branch predictor analyzer.
 #
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a file
-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
immediately

 -bpred          <string>         #        bimod # branch predictor type
{nottaken|taken|bimod|2lev|comb}

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

 -bpred:2lev     <int list...>    # 1 1024 8 0 # 2-level predictor config (<l1size>
<l2size> <hist_size> <xor>)

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

 -bpred:ras      <int>            #            8 # return address stack size (0 for
no return stack)

 -bpred:btb      <int list...>    # 512 4 # BTB config (<num_sets> <associativity>)


  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.
```

sim-cache

```
sim-cache: Version 2.0 of July, 1997.

Copyright (c) 1994-1997 by Todd M. Austin.  All Rights Reserved.
```

```
 Usage: sim-cache {-options} executable {arguments}

 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.

 #
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a file
-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
immediately
-cache:dl1       <string>         # dl1:256:32:1:l # l1 data cache config, i.e.,
{<config>|none}
-cache:dl2       <string>         # ul2:1024:64:4:l # l2 data cache config, i.e.,
{<config>|none}
-cache:il1       <string>         # il1:256:32:1:l # l1 inst cache config, i.e.,
{<config>|dl1|dl2|none}
-cache:il2       <string>         #          dl2 # l2 instruction cache config,
i.e., {<config>|dl2|none}
-tlb:itlb        <string>         # itlb:16:4096:4:l # instruction TLB config, i.e.,
{<config>|none}
-tlb:dtlb        <string>         # dtlb:32:4096:4:l # data TLB config, i.e.,
{<config>|none}
-flush           <true|false>     #        false # flush caches on system calls
-icompress       <true|false>     #        false # convert 64-bit inst addresses to
32-bit inst equivalents
-pcstat          <string list...> #       <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-cheetah

```
sim-cheetah: Version 2.0 of July, 1997.

Copyright (c) 1994-1997 by Todd M. Austin.  All Rights Reserved.

Portions Copyright (C) 1989-1993 by Rabin A. Sugumar and Santosh G. Abraham.

Usage: sim-cheetah {-options} executable {arguments}

sim-cheetah: This program implements a functional simulator driver for
Cheetah.  Cheetah is a cache simulation package written by Rabin Sugumar
and Santosh Abraham which can efficiently simulate multiple cache
configurations in a single run of a program.  Specifically, Cheetah can
```

```
simulate ranges of single level set-associative and fully-associative

caches.  See the directory libcheetah/ for more details on Cheetah.

 #

# -option        <args>           #    <default> # description

#

-config          <string>         #       <null> # load configuration from a file

-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
immediately

-refs            <string>         #         data # reference stream to analyze,
i.e., {inst|data|unified}

-R               <string>         #          lru # replacement policy, i.e., lru or
opt

-C               <string>         #           sa # cache configuration, i.e., fa,
sa, or dm

-a               <int>            #            7 # min number of sets (log base 2,
line size for DM)

-b               <int>            #           14 # max number of sets (log base 2,
line size for DM)

-l               <int>            #            4 # line size of the caches (log base
2)

-n               <int>            #            1 # max degree of associativity to
analyze (log base 2)

-in              <int>            #          512 # cache size intervals at which
miss ratio is shown

-M               <int>            #       524288 # maximum cache size of interest

-c               <int>            #           16 # size of cache (log base 2) for DM
analysis
```

sim-outorder

```
sim-outorder: Version 2.0 of July, 1997.

Copyright (c) 1994-1997 by Todd M. Austin.  All Rights Reserved.

 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 file

-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
immediately

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

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

-fetch:mplat     <int>            #            3 # extra branch mis-prediction
latency

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

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

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

-bpred:2lev      <int list...>    # 1 1024 8 0 # 2-level predictor config (<l1size>
<l2size> <hist_size> <xor>)

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

```
-bpred:ras       <int>           #            8 # return address stack size (0 for
no return stack)

-bpred:btb       <int list...>    # 512 4 # BTB config (<num_sets> <associativity>)

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

-decode:width    <int>           #            4 # instruction decode B/W
(insts/cycle)

-issue:width     <int>           #            4 # instruction issue B/W
(insts/cycle)

-issue:inorder   <true|false>    #        false # run pipeline with in-order issue

-issue:wrongpath <true|false>    #         true # issue instructions down wrong
execution paths

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

-ruu:size        <int>           #           16 # register update unit (RUU) size

-lsq:size        <int>           #            8 # load/store queue (LSQ) size

-cache:dl1       <string>        # dl1:128:32:4:l # l1 data cache config, i.e.,
{<config>|none}

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

-cache:dl2       <string>        # ul2:1024:64:4:l # l2 data cache config, i.e.,
{<config>|none}

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

-cache:il1       <string>        # il1:512:32:1:l # l1 inst cache config, i.e.,
{<config>|dl1|dl2|none}

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

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

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

-cache:flush     <true|false>    #        false # flush caches on system calls

-cache:icompress <true|false>    #        false # convert 64-bit inst addresses to
32-bit inst equivalents

-mem:lat         <int list...>    # 18 2 # memory access latency (<first_chunk>
<inter_chunk>)

-mem:width       <int>           #            8 # memory access bus width (in
bytes)
```

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

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

 -tlb:lat         <int>            #           30 # inst/data TLB miss latency (in
 cycles)

 -res:ialu        <int>            #            4 # total number of integer ALU's
 available

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

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

 -res:fpalu       <int>            #            4 # total number of floating point
 ALU's available

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

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

 -bugcompat       <true|false>     #        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-profile

sim-profile: Version 2.0 of July, 1997.

Usage: sim-profile {-options} executable {arguments}

sim-profile: This simulator implements a functional simulator with profiling support. Run with the `-h' flag to see profiling options available.

```
#
# -option        <args>          #    <default> # description
#
-config          <string>        #       <null> # load configuration from a file
-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
immediately
-all             <true|false>    #        false # enable all profile options
-iclass          <true|false>    #        false # enable instruction class
profiling
-iprof           <true|false>    #        false # enable instruction profiling
-brprof          <true|false>    #        false # enable branch instruction
profiling
-amprof          <true|false>    #        false # enable address mode profiling
-segprof         <true|false>    #        false # enable load/store address segment
profiling
```

```
 -tsymprof        <true|false>     #        false # enable text symbol profiling

 -taddrprof       <true|false>     #        false # enable text address profiling

 -dsymprof        <true|false>     #        false # enable data symbol profiling

 -internal        <true|false>     #        false # include compiler-internal symbols
during symbol profiling

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

sim-safe

```
sim-safe: Version 2.0 of July, 1997.

Copyright (c) 1994-1997 by Todd M. Austin.  All Rights Reserved.

 Usage: sim-safe {-options} executable {arguments}

 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.

 #

# -option        <args>           #    <default> # description

#

 -config          <string>         #       <null> # load configuration from a file

 -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
immediately
```

Command lines for SPEC CPU2000

Integer benchmarks

```
Benchmark: 164.gzip
Command Line: gzip00.peak.ev6 input.source 60

Benchmark: 175.vpr
Command Line: vpr00.peak.ev6 net.in arch.in place.in route.out -nodisp -route_only -
route_chan_width 15 -pres_fac_mult 2 -acc_fac 1 -first_iter_pres_fac 4 -
initial_pres_fac 8


Benchmark: 176.gcc
Command Line: gcc00.peak.ev6 166.i -o 166_2.s

Benchmark: 181.mcf
Command Line: mcf00.peak.ev6 inp.in

Benchmark: 186.crafty
Command Line: crafty00.peak.ev6

Benchmark: 197.parser
Command Line: parser00.peak.ev6 2.1.dict -batch

Benchmark: 252.eon
Command Line: eon00.peak.ev6 chair.control.cook chair.camera chair.surfaces
chair.cook.ppm ppm pixels_out.cook

Benchmark: 253.perlbmk
Command Line: perlbmk00.peak.ev6 diffmail.pl 2 550 15 24 23 100

Benchmark: 254.gap
Command Line: gap00.peak.ev6 -l . -q -m 64M

Benchmark: 255.vortex
Command Line: vortex00.peak.ev6 lendian1.raw

Benchmark: 256.bzip2
Command Line: bzip200.peak.ev6 input.source 58

Benchmark: 300.twolf
Command Line: twolf00.peak.ev6 ref
```

Floating point benchmarks

```
Benchmark: 168.wupwise
Command Line: wupwise00.peak.ev6

Benchmark: 171.swim
Command Line: swim00.peak.ev6

Benchmark: 172.mgrid
Command Line: mgrid00.peak.ev6

Benchmark: 173.applu
Command Line: applu00.peak.ev6
```

```
Benchmark: 177.mesa
Command Line: mesa00.peak.ev6 -frames 1000 -meshfile mesa.in -ppmfile mesa.ppm

Benchmark: 178.galgel
Command Line: galgel00.peak.ev6

Benchmark: 179.art
Command Line: art00.peak.ev6 -scanfile c756hel.in -trainfile1 a10.img -trainfile2
hc.img -stride 2 -startx 110 -starty 200 -endx 160 -endy 240 -objects 10

Benchmark: 183.equake
Command Line: equake00.peak.ev6

Benchmark: 187.facerec
Command Line: facerec00.peak.ev6

Benchmark: 188.ammp
Command Line: ammp00.peak.ev6

Benchmark: 189.lucas
Command Line: lucas00.peak.ev6

Benchmark: 191.fma3d
Command Line: fma3d00.peak.ev6

Benchmark: 200.sixtrack
Command Line: sixtrack00.peak.ev6

Benchmark: 301.apsi
Command Line: apsi00.peak.ev6
```

Notes on using SimpleScalar

(1) Read SimpleScalar Tutorial.

(2) Download 631ssAlpha.tgz (for Linux, 20,033,842 bytes) or 631ssAlpha-Cygwin.tgz (for Cygwin, 19,844,137 bytes). Unzip it (`tar xvzf cpe631Alpha.tgz`).
This archive includes all necessary simulators from SimpleScalar tool suite and Alpha binaries of SPEC CPU2000 benchmarks. Some of the simulators have been modified by our research group members, e.g., *sim-cache* in order to allow you to skip the specified number of instructions.
If you have a PC running Linux you might want to install full SimpleScalar suit which includes program development environment for PISA instruction set architecture (MIPS like) and ARM instruction set architecture. Links are on the course Web site.

(3) Be sure that you have SPEC CPU2000 (SED contact person is Mr. David Austin). You can install or just copy it. Let's say that home directory of SPEC CPU2000 is $SPEC_HOME.

(4) Steps to do:

```
# create a working directory

mkdir work

cd work

mkdir 172.mgrid # e.g., you want to simulate 172.mgrid application

cd 172.mgrid


# now you can copy inputs for this application
# into your working directory;

# with Cygwin you can use Explorer to move necessary input file mgrid.in

cp $SPEC_HOME/spec_cpu2000/benchspec/CFP2000/172.mgrid/data/ref/input/mgrid.in .


# let's say $SS_HOME is where you unzipped 631ssAlpha

# to run simulation type in (one line command)

$SS_HOME/631ssAlpha/mysimplesim_pff_log/sim-cache -fastfwd 500000000 -max:inst
500000000 -redir:sim u2_32KB.txt -cache:il1 il1:512:64:1:f -cache:dl1 dl1:512:64:1:f
-cache:il2 none -cache:dl2 ul2:2048:64:4:l
$SS_HOME/631ssAlpha/spec2000binaries/mgrid00.peak.ev6  < mgrid.in

# this will run sim-cache functional cache simulator for mgrid00 SPEC CPU
application;
# input for this application is given in the file mgrid.in

# tested cache configuration is 8KB L1I, 8KB L1D, and 32KB L2U;
# first 500M instructions will be skipped, and then 500M simulated.


# you can prepare a command file, e.g., 172mgrid.sh to include command lines for
# all runs for your homework (u2 is 32KB, 64KB, ...).
```

Example 1:

```
SS_HOME/631ssAlpha/arAlpha/mysimplesim_pff_log/sim-cache -max:inst 2000000000 -
redir:sim crafty_cache_f2b_l.txt -cache:il1 il1:256:64:1:f -cache:dl1 dl1:128:32:8:r
-cache:il2 dl2 -cache:dl2 ul2:256:64:16:l
$SS_HOME/631ssAlpha/spec2000binaries/crafty00.peak.ev6 < crafty.in
```

This command line runs the sim-cache simulator for 2 billion instructions. It stores the output in crafty_cache_f2b_l.txt file. There are two levels of caches: L1 contains IL1 with 256 sets, 64 B block size, direct mapped, and fifo replacement policy with a total size of 16 KB; and DL1 with 128 sets, 32 B block size, 8-way set associative, and random

replacement policy with a total size of 32 KB.

Example 2:

```
SS_HOME/631ssAlpha/arAlpha/mysimplesim_pff_log/sim-outorder -redir:sim Current-
outorder.txt -cache:il1 il1:64:8:32:l -cache:dl1 dl1:64:8:32:l -fetch:ifqsize 2 -
bpred nottaken -decode:width 1 -issue:width 1 -issue:inorder true -res:ialu 1 -
res:fpalu 1 -res:fpmult 1 -cache:dl2 none -cache:il2 none -mem:width 4 -mem:lat 12 1
$SS_HOME/631ssAlpha/spec2000binaries/gcc00.peak.ev6 scilab.i -o scilab.s
```

This command line runs the sim-outorder simulator. The output goes to Current-outorder.txt file. IL1 has 64 sets, 8 B block size, 32-way set associative, and least recently used replacement policy with a total size of 16 KB. DL1 is the same as IL1. Instruction fetch queue size: 2 instructions. Branch prediction scheme: not-taken. Instruction decode bandwidth: 1 instruction per cycle. Instruction issue bandwidth: 1 instruction per cycle. In-order issue. There is one INT ALU unit, one FP ALU unit, 1 FP multiplier. there is no L2 instruction or data caches. Memory access bus width: 4 B. Memory latency has 12 cycles for the first_chunk, and 1 cycle for inter_chunk.

Example 3:

```
SS_HOME/631ssAlpha/arAlpha/mysimplesim_pff_log/sim-cheetah -redir:sim sim-
cheetah.txt -R opt -C sa -a 5 -b 14 -l 4 -n 2
$SS_HOME/631ssAlpha/spec2000binaries/parser00.peak.ev6 ./2.1.dict - batch < ref.in
```

This command line runs the sim-cheetah simulator. with optimal replacement policy. Set associative cache. The number of sets ranges from 5 to 14 . Block size of 4 B. And associativity ranges from direct-mapped to 2-way set associative.

SimpleScalar resources

- Web page: http://www.simplescalar.com
- Mailing list: http://ord.eecs.umich.edu/ss_archives/
- SimpleScalar Version 4.0 Test Releases: http://www.simplescalar.com/v4test.html
- SimpleScalar Documentation: Documentation
- SimpleScalar users guide: users_guide_v2.pdf

Benchmarks:

- MiBench Embedded Benchmark Suite: http://www.eecs.umich.edu/mibench/

- Standard Performance Evaluation Corporation (SPEC): http://www.spec.org/

Inputs for SPEC CPU applications (http://www.cag.lcs.mit.edu/~kbarr/cag/spec2000-commandlines.html)