Programming project 2, milestone 2 — due on Thursday 9 November 2000

The aim is to examine virtual memory management algorithms employed by the operating system.

For the first milestone, you were to collect a series of programs of interest to you, your benchmarks, and to assemble the tools to trace the execution of a program and report all memory references.

A tool you can use: SimpleScalar

SimpleScalar 3.0, my patches for RedHat, and for Debian, configured and built for the Alpha target, tested on UNM CS Debian Linux hosts, can be found in directory /nfs/faculty/darko/exports/simplesim-3.0.

EIO files for replaying execution of Alpha binaries are in the subdirectory EIO-traces/AlphaAXP.

The EIO files provided allow you to execute SPEC95 benchmarks (a set of programs widely used in computer architecture studies as well as in commercial comparisons) on particular inputs called the training set.

For example, you can do:

cd /nfs/faculty/darko/exports/simplesim-3.0

./sim-safe EIO-traces/AlphaAXP/SPEC95.099.go.train.50-9-2stone9.eio.gz

which will run the SPEC95 integer benchmark 099.go, the game of go, under the SimpleScalar simulator tool sim-safe.

You will get output like this:

```
sim-safe: SimpleScalar/Alpha Tool Set version 3.0 of September, 1998.
Copyright (c) 1994-1998 by Todd M. Austin.  All Rights Reserved.

sim: loading EIO file: EIO-traces/AlphaAXP/SPEC95.099.go.train.50-9-2stone9.eio.gz
sim: command line: ./sim-safe EIO-traces/AlphaAXP/SPEC95.099.go.train.50-9-2stone9.eio.gz

sim: simulation started @ Sun Oct 29 16:56:42 2000, 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 **
Initial position from 2stone9.in
  1 B C3
  2 W pass
  3 B G7
 .......
 .......
 53 B F2
 54 W pass
```

```
 55 B pass
Game over

sim: ** simulation statistics **
sim_num_insn              527768219 # total number of instructions executed
sim_num_refs              162784067 # total number of loads and stores executed
sim_elapsed_time                246 # total simulation time in seconds
sim_inst_rate          2145399.2642 # simulation speed (in insts/sec)
ld_text_base           0x0120000000 # program text (code) segment base
ld_text_size                 344064 # program text (code) size in bytes
ld_data_base           0x0140000000 # program initialized data segment base
ld_data_size                 620432 # program init'ed `.data' and uninit'ed `.bss' 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          0x0120007d10 # 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                  116 # total number of pages allocated
mem.page_mem                   928k # total size of memory pages allocated
mem.ptab_misses            13412161 # total first level page table misses
mem.ptab_accesses         854320772 # total page table accesses
mem.ptab_miss_rate           0.0157 # first level page table miss rate
```

You can tell from the above summary that the tool sim-safe contains all the elements needed for your task. You just need to modify sim-safe to *print* the trace of memory accesses. (Of course, you must first make your own copy of the simplesim-3.0 directory structure.)

There are several other tools in SimpleScalar that you can familiarize yourself with; sim-outorder, for instance, will do a detailed simulation of a superscalar architecture. Note that with an out-of-order architecture the ordering of memory instructions may be different, but this fact does not concern us in this project.

You can find documentation on SimpleScalar (release 2.0, but applicable to 3.0) at
http://www.cs.wisc.edu/~mscalar/simplescalar.html.

Second week goals.

Produce traces of memory accesses for programs in your benchmark suite.

To be representative of present-day loads, a trace should contain billions of references. For the purposes of the project, tens of millions will suffice. Nevertheless, traces are very long. Devise a way to compress traces, so they can be stored on disk.

If we were interested in a cache behavior study, we would use all the information in the memory access trace file. However, we are interested in virtual memory behavior (paging behavior). Therefore, we can reduce the amount of information we need to keep considerably. If the page size is P bytes, then instead of the complete virtual address a, we just need the page number, $\lfloor \frac{a}{P} \rfloor$. Often it will happen that many consecutive memory accesses map to the same page number, but we only need to keep one page access in the page access file.

Your document should detail: the benchmark suite, the tracing tools, the format of the trace file, the compression mechanism, and the statistics of the tracing process (for each benchmark, how many instructions/cycles/seconds it took, how many memory reads and writes, the size of the memory access trace file uncompressed and compressed, the size of the page access trace file uncompressed and compressed).