

Quick Guide to the EECS 452 Computing Environment

Accounts

If you do not already have account on ECE machines, contact me via email to get one set up for this class.

Machines

For this project, you can run your simulations on the following machines:

358-1.ece.northwestern.edu
358-2.ece.northwestern.edu
358-3.ece.northwestern.edu
358-4.ece.northwestern.edu

Please email me if you have any problems accessing these machines.

Binaries

SimpleScalar executes PISA and Alpha binaries. In this class, we will be using Alpha binaries. I have made a number of modifications to SimpleScalar that we will make use of in this class - please read these instructions carefully and only use the tar'ed gzipped version of SimpleScalar located at:

<http://www.ece.northwestern.edu/~memik/courses/452/labs/simplescalar.tar.gz>

Once you have unpacked this file in your home directory, use "make sim-safe" to compile the simulator.

SimpleScalar

SimpleScalar has a variety of simulation tools - in this project we will only be making use of sim-safe.c. For those who want to try a cycle-accurate simulator, we will make use of sim-outorder.c. Sim-safe is a functional simulator - similar to Shade or Atom for those familiar with those tools - but it is not instrumenting a binary, it is an actual execution-driven simulation. You will not be able to measure performance (IPC) with this kind of tool, but you can gather other useful statistics. Here is an example call to sim-safe:

```
sim-safe -max:inst 100000000 prog_name prog_inputs
```

The three defined input parameters are:

- `-max:inst`: This is used to specify how many instructions to execute for.
- `prog_name`: Is the name of the benchmark you want to simulate, see the list below for a complete list of names.
- `prog_inputs`: Inputs to the selected benchmark, see the list again for the input parameters for each benchmark.

I strongly recommend that you use the input and output conventions defined in `simplescalar`. You will want to redirect `stderr` to a file when running jobs. You should use `>&` to do this. For example

```
sim-safe -max:inst 100000000 prog_name prog_input >& test.err
```

will dump `stderr` to `test.err`.

Benchmarks

Here's a list of benchmarks from SPEC 2000 that are commonly used:

- `art`
- `ammp`
- `applu`
- `apsi`
- `crafty`
- `eon`
- `quake`
- `facerec`
- `galgel`
- `gcc`
- `lucas`
- `mesa`
- `mgrid`
- `parser`
- `perlbmk`
- `twolf`
- `vortex`
- `vpr`
- `wupwise`

Common Mistakes

- When you use the `opt_reg_int` function calls, make sure you change both the name of the parameter flag (i.e. `"-num_entries_value_predictor"`) and the storage location for that parameter (i.e. `"&num_entries_value_predictor"`). Many brave hackers have fallen victim to this insidious bug.

- Corrupting the nonspeculative path - if you find your program mysteriously dying - or even worse, if your program is not dying, but your results are very strange - you may be corrupting the nonspeculative path. This means that you are not executing all of the instructions you should be (or possibly in the correct order you should be). This can happen if you incorrectly recover from a branch misprediction, corrupt `regs.reg_NPC`, or simply hinder execution of an instruction. This can have a variety of unpleasant effects - you can go to an invalid address range, divide by zero, or even just have some result calculation be incorrect. The best way to test this is to dump out some portion of your execution as a trace (just print the instructions executed) and compare this to a trace that you can generate with an unmodified and/or correct version of `sim-safe`.
- Memory leaks - be careful of these - when simulating for a long time, a small leak can add up to a big problem.
- Infinite looping - if your simulator just keeps running and running, you may have a misprediction that was partially recovered or recovered incorrectly - or just a really slow simulator. To determine what the problem is, dump out statistics every 10 or 25 million instructions, or every 100 million cycles. Change these numbers as appropriate to avoid dumping too much to disk.
- LAST NOTE - Simulations are not perfect by any means. You can have a simulator that executes cleanly, gives you fantastic results, and is executing the correct instructions - but is still incorrect. You have to be sure that you are realistically simulating various structures - this does not mean that you must implement it exactly as it would be in hardware - but the end result must be that it behaves the same way.