18-344 Recitation 0

Setting up the tools

08/29/2025

Housekeeping :: Feeling a little rusty?

You are going to be using the Linux CLI a lot.

You are going to write a fair amount of C++ (cool if you've never written C++).

You are likely going to need to write some scripts as well (Bash, Python, etc.).

Check out https://missing.csail.mit.edu/

Also, AI tools can be pretty helpful too! **Make sure to use them in accordance with our Course AI Assistant Policy**: https://course.ece.cmu.edu/~ece344/course_documents/Syllabus.pdf

Housekeeping :: Accessing the ECE number cluster

Hostnames: **<ece000-ece031>.ece.local.cmu.edu**

Specs:
https://cmu-enterprise.atlassian.net/wiki/spaces/ITS/pages/2332131370/ECE+Community+Compute+Clusters#ECE-Community-Cluster-Specifications-(ECE-NUMBER-Cluster)

We recommend using your ECE AFS space.

And, if you'd like to, you can switch to using your ECE home directory.

Check out https://cmu-enterprise.atlassian.net/wiki/x/UwBUhw

Tools :: Introduction

Pin is a dynamic binary instrumentation tool.

Users (you) develop **Pintool**s that run along with **Pin** and the application binary being instrumented to perform program & performance analysis.

DESTINY is a tool for modeling 2D/3D caches designed with SRAM & other cells.

SPEC is a set of benchmark suites for measuring the performance of processors.

We will be using the **SPEC CPU 2017 intspeed** benchmark suite.

Tools :: Environment setup

Source the following script in your `.bashrc`, which sets up the PATH and other environment variables for **Pin**, **DESTINY**, and **SPEC**:

```
/afs/ece.cmu.edu/class/ece344/bin/setup344
```

In the rare cases that you need cross-realm authentication, also add the following:

```
aklog ece.cmu.edu
aklog andrew.cmu.edu
```

Tools :: Directory setup

Go to your home directory in ECE AFS (if you are not already there) and create a directory for this course somewhere in the `~/Private` directory. For example:

```
mkdir /afs/ece/usr/<AndrewID>/Private/18344
```

Now add this to your `.bashrc` so you don't have to always type long paths:

```
export ECE344_HOME="/afs/ece/usr/<AndrewID>/Private/18344"
```

Add this to your `.bashrc` as well:

```
export ECE344_SPEC_CONFIG="$ECE344_HOME/18344-f25-<AndrewID>.cfg"
```

There's no config file there yet, but we are going to get one very soon.

Tools :: SPEC 2017 setup

Make sure to source the updated `.bashrc` if you haven't done so already.

Now copy and rename the SPEC config template:

```
cp /afs/ece/class/ece344/opt/spec2017/config/18344-f25-template.cfg $ECE344_SPEC_CONFIG
```

Open the SPEC config file with your favorite editor, and,

- find and set `output_root` to `/scratch/ece344-<AndrewID>`, and,
- find and set `submit` to `<full-path-to-your-344-home>/run $benchmark $command`
  - e.g., `/afs/ece/usr/<AndrewID>/Private/18344/run $benchmark $command`

Tools :: Testing :: Pin

Copy the lab0 code to your course directory then untar it:

```
cp /afs/ece/class/ece344/assign/lab0.tar.gz $ECE344_HOME
```

```
cd $ECE344_HOME && tar xvzf lab0.tar.gz
```

You'll need to modify the `Load`, `Store`, and `Fini` functions in `lab0/mem-count.cpp`, and potentially add some global variables.

Keep in mind that you only need to record **the number of loads and stores** in the `mem-count` Pintool. When Pin is running along a binary, each load/store instruction will invoke the `Load/Store` function in the Pintool, respectively.

Since there could be many loads/stores, make sure to use data types suitable for holding large integers.

Tools :: Testing :: Pin cont’d

Use the supplied `makefile` to build `mem-count.cpp` into a Pintool:

```
cd $ECE344_HOME/lab0 && make
```

You can find the Pintool at:

```
$ECE344_HOME/lab0/obj-intel64/mem-count.so
```

To instrument a binary (e.g. `curl`) with this Pintool (in the `lab0` directory):

```
pin -t obj-intel64/mem-count.so -o curl.stats -- curl cmu.edu
```

Here, `-t` precedes the path to the Pintool, `-o` precedes the output (defined as a knob), and whatever comes after the double dash are the binary to be instrumented with and its arguments.

Tools :: Testing :: DESTINY

Use DESTINY to model the template 16KB cache:

```
cd $ECE344_HOME/lab0 && destiny SRAM_template.cfg
```

DESTINY:

takes in a configuration file that specifies details about the memory it will model, e.g. size, associativity, technology node, etc.

outputs the details of the memory being modeled.

Tools :: Testing :: SPEC 2017

To run a set of SPEC 2017 benchmarks:

```
runcpu -c $ECE344_SPEC_CONFIG --action=onlyrun --noreportable \
                    --size=test <benchmark>
```

Make sure to always specify `--action=onlyrun --noreportable --size=test` for faster execution times.

`<benchmark>` can a suite of benchmarks, e.g., `intspeed`, or a single benchmark program, e.g. `gcc_s`.

For the list of benchmarks included in the `intspeed` suite, check out https://www.spec.org/cpu2017/Docs/overview.html#Q13.

Note that `perlbench_s`, `deepsjeng_s`, and `specrand_s` may fail.

Tools :: Testing :: SPEC 2017 cont’d

For each benchmark program, `runcpu` looks up the config, and executes the program according to the `submit` field.

Remember this?

```
submit: <full-path-to-your-344-home>/run $benchmark $command
```

`$benchmark` and `$command` will be replaced by `runcpu` and passed to the `run` script.

We can then write the `run` script such that it will instrument the SPEC 2017 benchmarks.

Tools :: Testing :: SPEC 2017 with Pin

First let's create a `run` script and make it executable:

```
touch $ECE344_HOME/run && chmod +x $ECE344_HOME/run
```

Your run script can be written in any language of your choice (it can even be a compiled binary), as long as it gets the job done. Here are two examples:

```
#!/bin/bash

BENCHMARK=$1
COMMAND=${@:2}
PINTOOL="<full-path-to-pintool>"
RESULT="<results-dir>/${BENCHMARK}.stats"

pin -t ${PINTOOL} -o ${RESULT} -- ${COMMAND}
```

```
#!/usr/local/bin/python3

import os, sys

benchmark = sys.argv[1]
command = " ".join(sys.argv[2:])

pintool = "<full-path-to-pintool>"
result = os.path.join("<results-dir>", benchmark + ".stats")

pin_cmd = "pin -t %s -o %s -- %s" % (pintool, result, command)
os.system(pin_cmd)
```

Tools :: Testing :: SPEC 2017 with Pin cont'd

Make sure to replace the placeholders in the example scripts, and to create the results directory (not necessary but highly recommended).

Now we can instrument SPEC benchmarks with our Pintool (to speed up the process, we'll only instrument `gcc_s` here):

```
runcpu -c $ECE344_SPEC_CONFIG --action=onlyrun \
        --noreportable --size=test gcc_s
```

You should then be able to see a `gcc_s.stats` in the results directory.

Tools :: Testing :: SPEC 2017 with Pin cont'd

The `run` script can be used to provide benchmark-specific output filenames as we've seen, and also to sweep across different Pin configurations (e.g. multiple cache sizes) for the same SPEC CPU 2017 benchmark program(s), by adding simple loops in your script file.

You'll need to modify your `run` script several times for the upcoming labs, so it'd be a good idea to keep separate copies of the `run` script and either modify the SPEC config file or use symlinks to choose which `run` script to use.