EC513 Spring 2023
Pin Tutorial

Acknowledgement

The following individuals have contributed to developing this tutorial: Multiple folks at Intel, Harish Patil, Zahra Azad, Farbin Fayza, and Ajay Joshi.

GitHub Repository

Clone the repository `https://github.com/bu-ece513-spring2023/pin-bimodal-tutorial. git` for getting started.

git clone https://github.com/bu-ece513-spring2023/pin-bimodal-tutorial.git

Here is a brief description of the provided files.

- *bimodal.H*: Contains the implementation of the bimodal branch predictor with the Pin APIs.

- *branch-predictor.cpp*: A driver script to run the Pin tool. *bimodal.H* is included in this script.

- *makefile*: Makefile to build the pintool.

- *makefile.rules*: Rules for the *makefile*.

- *test-scripts/runall.train.bimodal.sh*: A script to test the bimodal pintool on the training portion of the SPECrate2017 Integer benchmarks.

- *test-scripts/report.train.bimodal.sh*: A script to generate a report from the output files after running the *runall.train.bimodal.sh* script.

Running the Bimodal Branch Predictor

Follow the steps below to run the given pintool. For steps 2-4, you may consider writing a shell script with the given commands to make your job easier. Note that the % sign is used to indicate a shell command.

1. Download the latest Linux Pin kit from `http://pintool.intel.com`

2. Open a terminal in your working directory (where you have the files mentioned above). Load the gcc module.

   % module load gcc/12.2.0

3. Define *PIN_ROOT* as the directory of the downloaded Pin kit in step 1. Then build the branch-predictor pintool with the *make* command.

   % export PIN_ROOT="your Pin kit directory"
   % make clean
   % make

   This should produce a folder named "*obj-intel64*" that contains the object file (*branch-predictor.o*) and the pintool (*branch-predictor.so*).

4. Now we are ready to run the pintool on any test application. The following command is an example of running the generated pintool on the Linux *ls* application.

   % $PIN_ROOT/pin -t obj-intel64/branch-predictor.so - - /bin/ls

   A successful run will produce an output file (*bimodal.out*) containing the number of total instructions, the number of conditional instructions, correct and incorrect prediction counts, and the number of misses per kilo instructions (MPKI) of the branch predictor.
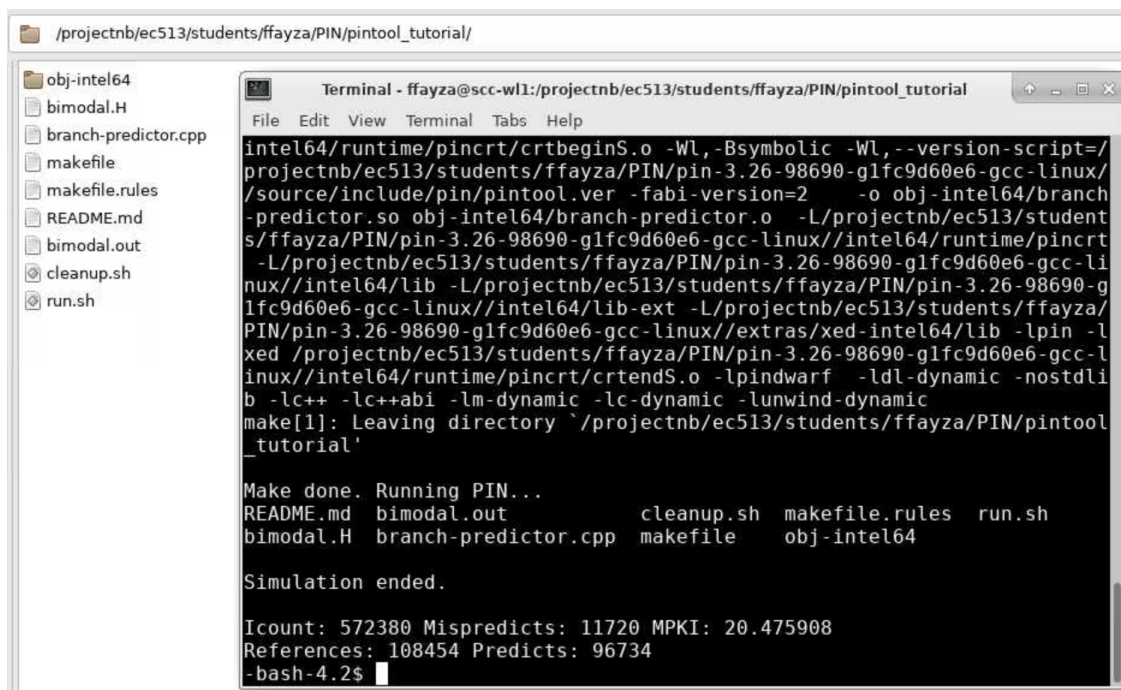
Figure 1 shows a sample script with the commands mentioned in the steps 2-4. Figure 2 shows the outputs after running the sample script.

```
export PIN_ROOT="/projectnb/ec513/students/ffayza/PIN/pin-3.26-98690-g1fc9d60e6-gcc-linux/"

module load gcc/12.2.0
make clean
make

echo -e "\nMake done. Running PIN..."
$PIN_ROOT/pin -t obj-intel64/branch-predictor.so -- /bin/ls

echo -e "\nSimulation ended."
head bimodal.out
```

Figure 1: A sample script for building and running the bimodal pintool.

Figure 2: Output after running the script in Figure 1.

You can test the pintool on any other application of your choice. Here are some useful resources that might come to help.

- Documentation of all Pin APIs: `https://software.intel.com/sites/landingpage/pintool/docs/98484/Pin/html/group__API__REF.html`

- Shell scripting basics: `https://www.tutorialspoint.com/unix/shell_scripting.htm`

Testing the Bimodal Pintool on the SPEC2017 Benchmark

SPEC CPU 2017 is a popular benchmark package used in industries for analyzing the performance of computing systems. It has 43 benchmarks of various applications such as compilers, 3D rendering, fluid dynamics, artificial intelligence, and many more. You can find more information about the SPEC CPU 2017 benchmark here: `https://www.spec.org/cpu2017/Docs/`

We will test our bimodal pintool on the Integer suit of the SPEC CPU 2017 benchmark to check the performance of our branch predictor. This suit is provided in SCC and you can directly use it from there. Follow these steps to run your pintool on the training portion of the provided benchmark.

1. Copy */projectnb/ec513/materials/pin-assignment/benchmark/spec2017.rate.train.harness* to your working directory (can be another folder) by creating a soft link. **Do not copy directly** because it will waste a lot of space. Use the following command on a terminal opened at your working directory:

    % cp -rs /projectnb/ec513/materials/pin-assignment/benchmark/spec2017.rate.train.harness ./spec2017.rate.train.harness

    The *-s* flag will create a soft link instead of copying the physical data.

2. Next, copy the two given scripts in the *test-scripts* folder into your spec2017.rate.train.harness folder.

3. *runall.train.bimodal.sh* is a sample script that runs the bimodal pintool on all the applications of the given spec harness. Additionally, it runs the application itself without the pintool and records the runtime of the application with and without the pintool. To run it, change the *PIN_ROOT*, *PINTOOL*, and *OUT_DIR* variables to your Pin kit directory, pintool, and desired output directory (all the output files will be generated here) respectively as mentioned in the script. Figure 3 shows an example.


```
export PIN_ROOT="/projectnb/ec513/students/ffayza/PIN/pin-3.26-98690-g1fc9d60e6-gcc-linux" #CHANGEME to your pin kit directory
export PINTOOL="/projectnb/ec513/students/ffayza/PIN/pintool_bimodal/obj-intel64/branch-predictor.so" #CHANGEME to your built pintool (.so file)
export OUT_DIR="/projectnb/ec513/students/ffayza/PIN/results/" #CHANGEME to your desired output directory
```

Figure 3: Changes in the *runall.train.bimodal.sh* script.

After these changes, you can run the *runall.train.bimodal.sh* script with this command:

% ./runall.train.bimodal.sh

This will produce the output files containing the branch predictor statistics in the directory you specified with *OUT_DIR*.

Note that the *runall.train.bimodal.sh* script must be inside the harness folder and you must run it from a terminal opened there.

4. The script *report.train.bimodal.sh* can be used to parse all the generated output files and print the stats on the terminal. To run it, just change the *OUT_DIR* variable to the directory where you have your output files. Open a terminal in the harness folder and run *report.train.bimodal.sh*.

> % ./report.train.bimodal.sh

Check the provided excel file *results/bimodal-train-stats.xlsx* on GitHub and verify your produced results. They might not match exactly but the numbers should be close.

Additional Note: Debugging the Pintool with GDB

1. Run GDB: To run GDB in the terminal, you need to open a terminal window and then type the command "gdb" (see Figure 4).

```
[[zazad@scc1 zazad]$
[[zazad@scc1 zazad]$
[[zazad@scc1 zazad]$ gdb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb)
```

Figure 4: Run GDB in the terminal.

2. Get PID: In another window, start your pintool using the following command with the "pause_tool" flag.

> % $PIN_ROOT/pin -pause_tool 20 -t obj-intel64/branch-predictor.so - - /bin/ls

The "pause_tool" flag in Pin is a command line option that can be used to pause the execution of a Pin-based tool when it starts. When the "pause_tool" flag is set, the tool will wait for a debugger to attach to it before continuing execution. This allows you to debug the tool itself, or to set breakpoints in the tool's code to see how it works. Once the tool is paused, you can use a debugger like GDB to attach to it and debug it using the PID reported in the output of this command (see Figure 5).

```
[zazad@scc1 zazad]$
[zazad@scc1 zazad]$ $PIN_ROOT/pin -pause_tool 20  -t obj-intel64/branch-predictor.so -- /bin/ls
Pausing for 20 seconds to attach to process with pid 126596
To load the debug info to gdb use:
****************************************************************
set sysroot /not/existing/dir
file
add-symbol-file /projectnb/ec513/students/zazad/obj-intel64/branch-predictor.so 0x7f208122a040 -s .data 0x7f20815a49a0 -s .bss 0x7f20815a5380
****************************************************************

```

Figure 5: Run Pintool with pause_tool option to get the PID.

3. Debug: After getting the program PID from the previous step, you can attach the program pid to GDB using the following command (also see Figure 6). Here, the "attach" command is used to attach to the Pin process with the specified PID to debug it.
   % (gdb) attach pid

```
[[zazad@scc1 zazad]$
[[zazad@scc1 zazad]$
[[zazad@scc1 zazad]$ gdb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb) attach 126596
Attaching to process 126596
```

Figure 6: Attach GDB to the process using pid.

Here are some common GDB commands:

- **add-symbol-file**: allows you to load symbols for a program that is already running. By loading symbols, GDB can provide information such as function names and local variables, which can be useful when debugging.

- **break**: sets a breakpoint at a specified location in the program.

- **continue**: continues execution of the program until a breakpoint is reached or the program exits.

- **next**: executes the next line of the program, stepping over any function calls.

- **step**: executes the next line of the program, stepping into any function calls.

- **print**: prints the value of an expression. backtrace: displays a stack trace showing the current call stack.

- **quit**: exits GDB.