18-344 Recitation 3

HW1 Review & Lab 1 Overview

Logistical Notes

Logistical Notes

- HW line up with lectures
  - Release after the lecture and are due before lecture (9:30am)
- HW2 Released - Due Sept 21 9:30am
- Lab 1 Released - Due Sept 21 9:30am

Amdahl’s Law

Defining Speed up

“Your friend proposes an optimization which would speed up store operations by 30%. Calculate the new run time”

When we say: X is N times faster than Y, We mean:

N = TimeY/TimeX = 1.30

Examples in HW1

Stores_old = 15ns
Stores_new = Stores_old/1.3 = 15ns/1.3 = 11.53ns

11.53 * 1000 * 0.2 = 2307.69s

1950ns + 2250ns + 2307.69ns ≅ = 6608ns

Note:

(15ns * 0.7) * 1000 * 0.2 = 2100ns

Amdahl’s Law Approach

Store Time = 15 * 1000 * 0.2 = 2250ns

Total Time = 6600ns

2250ns/6600ns = .34

S = 1/(1-p + p/s) = 1/(.66 + .34/1.3) = 1.08

6600/1.08 ≅ 6082ns

Amdahl’s Law cont’d

Note: Amdahl’s deals with proportions of time not with proportions of operations

$S = 1/(1-p + p/s) = 1/(0.8 + .2/1.3) = 1.04$ (incorrect)

$6600/1.04 \cong 6295$ns

ISA Design

General Principles

- Don’t over specify… If Hardware & Software don’t need to BOTH know this information it doesn’t need to be in the spec*
- SW:
    - If a compiler doesn’t NEED to know it, it doesn’t matter
- HW:
    - If a micro-architect doesn’t NEED to know it, it doesn’t matter

*Things get fuzzier on how to define “need”

** lots of exceptions based on who makes the ISA

Caches

- Caches are a hardware construct
  - What you built in 213 is NOT a cache, it was a cache simulator
- Caches generally shouldn’t be in the ISA*
  - They do (dynamic) run-time analysis of code to optimize memory accesses
  - What happens if new caching technologies are developed?
    - Better replacement policies
    - Space optimizations

*many ISA’s break this rule: x86 cache hinting

Specifying Delays

- Answer didn’t matter if you interpreted “delay” to be time or cycles
- Instruction delays are implementation specific
    - Optimizing for power vs Optimizing for speed will have different delays
    - Improvements in execution techniques or a paradigm shift in architecture will make these specifications obsolete
- This can be made worse if you specify techniques to deal with these delays in your ISA: MIPS R3000 Branch Delay Slot and Load Delay Slot

Logistical Notes

- Partnered lab
- Lab1 builds off the infrastructure setup in Lab0
  - Pintool
  - Runcpu
  - SPEC2017

Version Control

- We recommend using some version control system such as git
- If you choose to backup your code to a cloud service such as GitHub **PLEASE MAKE SURE IT IS PRIVATE**

Collaboration Tools

- You can share files over the afs space by creating a shared directory (see: https://github.com/CMU-18240/240-How-to/wiki/Configuring-AFS-folder-permissions-with-FS)
- If you are using git, you might also collaborate via a cloud system like github.
- Teletype (VScode, other ‘smart’ editors)
- Partner Program

Getting Started

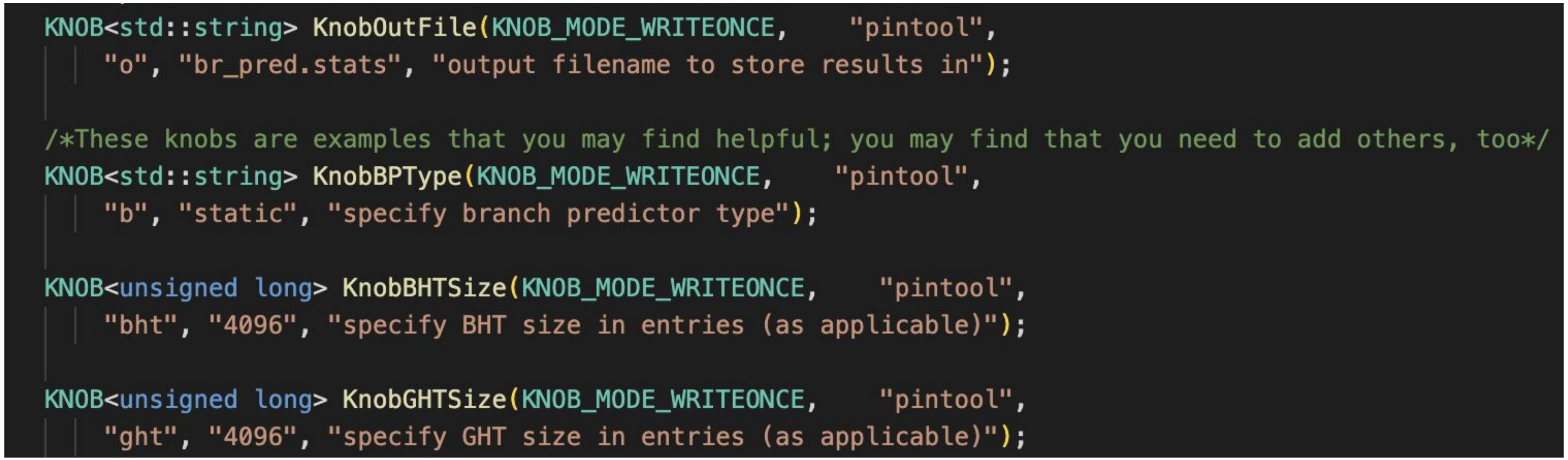
- Starter code at: /afs/ece.cmu.edu/class/ece344/assign/lab1.tar.gz
- Extract into your private class folder using: `tar -xvzf <file_name>.tar.gz`
- Read handout.txt for Lab1 implementation and deliverable details

Goals

- Implement four branch outcome prediction algorithms that we learned in class and compare their accuracy and implementation complexity
- The branch predictors that you will implement are:
  - Static predictor (e.g., always-taken or always-not-taken)
  - Bimodal / saturating counter predictor
  - Two-level (e.g., GAp or PAg) predictor
  - GShare predictor
- You will implement predict(), update(), and any other necessary functionality for four branch predictors (we recommend implementing these in the existing bp.cpp file)

Lab1 Knobs

- Found in bp_main.cpp:



```
KNOB<std::string> KnobOutFile(KNOB_MODE_WRITEONCE,    "pintool",
    "o", "br_pred.stats", "output filename to store results in");

/*These knobs are examples that you may find helpful; you may find that you need to add others, too*/
KNOB<std::string> KnobBPType(KNOB_MODE_WRITEONCE,    "pintool",
    "b", "static", "specify branch predictor type");

KNOB<unsigned long> KnobBHTSize(KNOB_MODE_WRITEONCE,    "pintool",
    "bht", "4096", "specify BHT size in entries (as applicable)");

KNOB<unsigned long> KnobGHTSize(KNOB_MODE_WRITEONCE,    "pintool",
    "ght", "4096", "specify GHT size in entries (as applicable)");
```

- Refer to Recitation 2 for additional Knob details

Implementation Tips

Your code will run every time a branch is encountered in the program

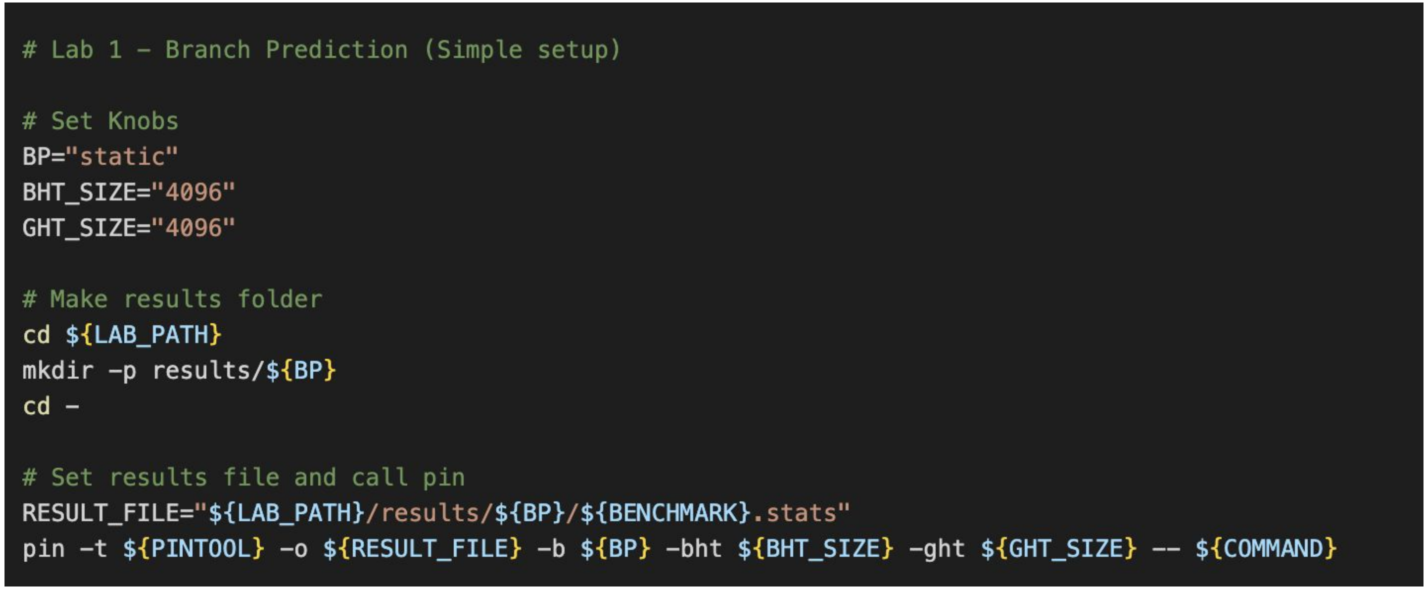
- Keep your code light weight
- In general, avoid C++ data structures like Map
    - This is an AVL tree (remember that from 15122?)
    - These AVL trees will rebalance after every call
- Stick to C arrays and you should be able to keep your code quick and efficient

Testing Infrastructure

- Lab0 should have ironed out all the issues but let's double check
- The static predictor (always taken) is already implemented in the starter code
- Lets try running the pintool
    - First need to run `make` to generate pintool (.so file)
    - Remember to run `make` every time you edit the source files

Testing Infrastructure - edit run.sh/py

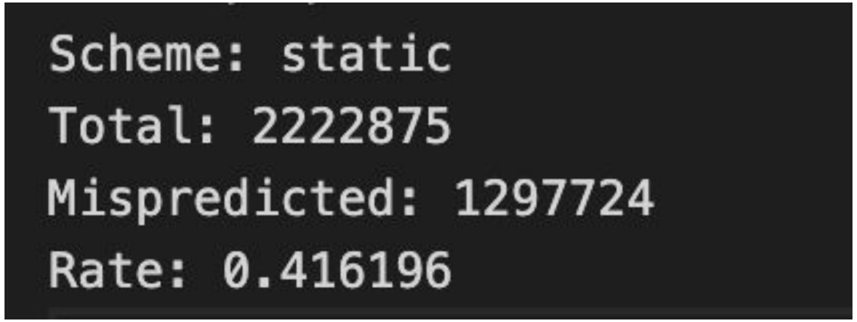
- We need to change run.sh/py to point to the new lab1 files as well as set knobs when calling the pintool



```
# Lab 1 - Branch Prediction (Simple setup)

# Set Knobs
BP="static"
BHT_SIZE="4096"
GHT_SIZE="4096"

# Make results folder
cd ${LAB_PATH}
mkdir -p results/${BP}
cd -

# Set results file and call pin
RESULT_FILE="${LAB_PATH}/results/${BP}/${BENCHMARK}.stats"
pin -t ${PINTOOL} -o ${RESULT_FILE} -b ${BP} -bht ${BHT_SIZE} -ght ${GHT_SIZE} -- ${COMMAND}
```

Testing Infrastructure - call runcpu

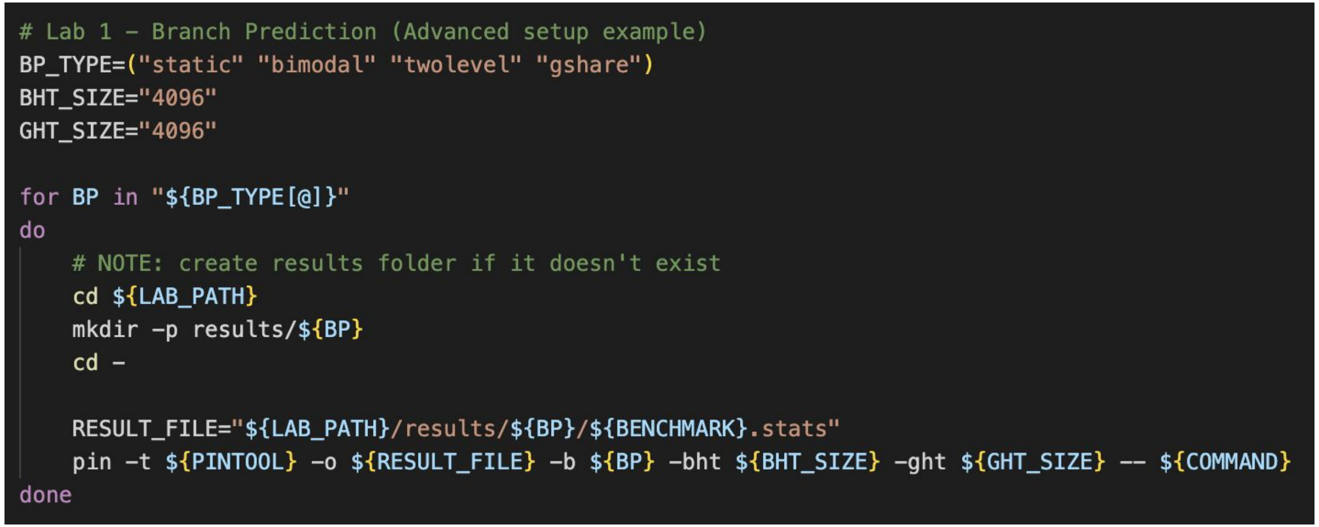
- Command:
  `runcpu -c /path-to-config/18344-f22-<andrewid>.cfg --action=onlyrun--noreportable --size=test <selected-benchmark-suite>`
- There should now be stats in the results folder for the static branch predictor



```
Scheme: static
Total: 2222875
Mispredicted: 1297724
Rate: 0.416196
```

Advanced edits to run.sh

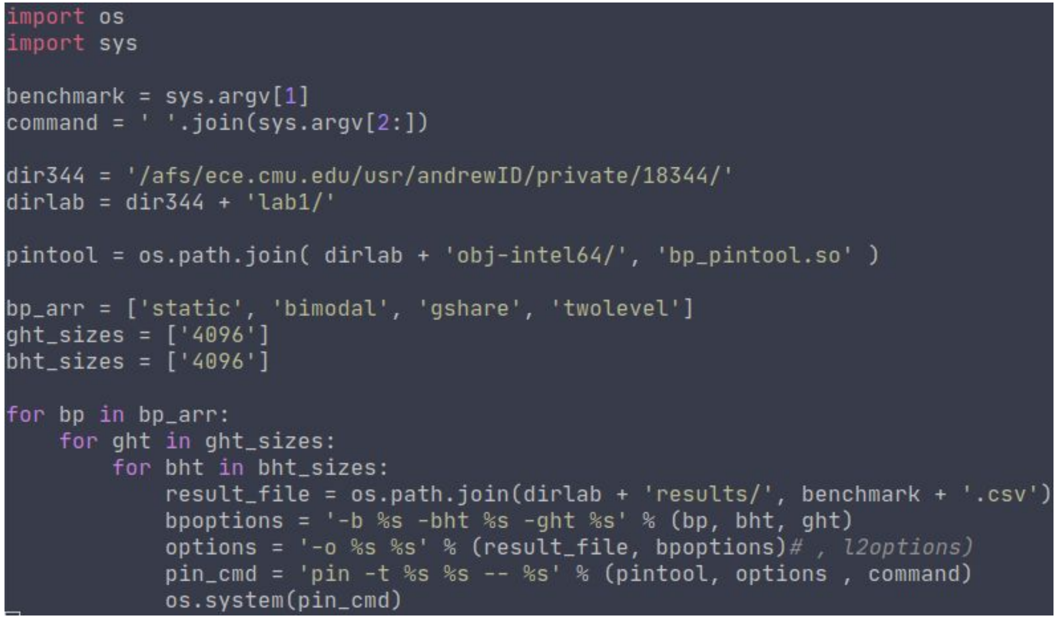
- You will want to automate the process of selecting knob {b, bht, ght} values
- Example which iterates over each BP type (keeping bht, ght constant)



```
# Lab 1 - Branch Prediction (Advanced setup example)
BP_TYPE=("static" "bimodal" "twolevel" "gshare")
BHT_SIZE="4096"
GHT_SIZE="4096"

for BP in "${BP_TYPE[@]}"
do
    # NOTE: create results folder if it doesn't exist
    cd ${LAB_PATH}
    mkdir -p results/${BP}
    cd -

    RESULT_FILE="${LAB_PATH}/results/${BP}/${BENCHMARK}.stats"
    pin -t ${PINTOOL} -o ${RESULT_FILE} -b ${BP} -bht ${BHT_SIZE} -ght ${GHT_SIZE} -- ${COMMAND}
done
```

- Note: You could have several nested for loops iterating over different knob values

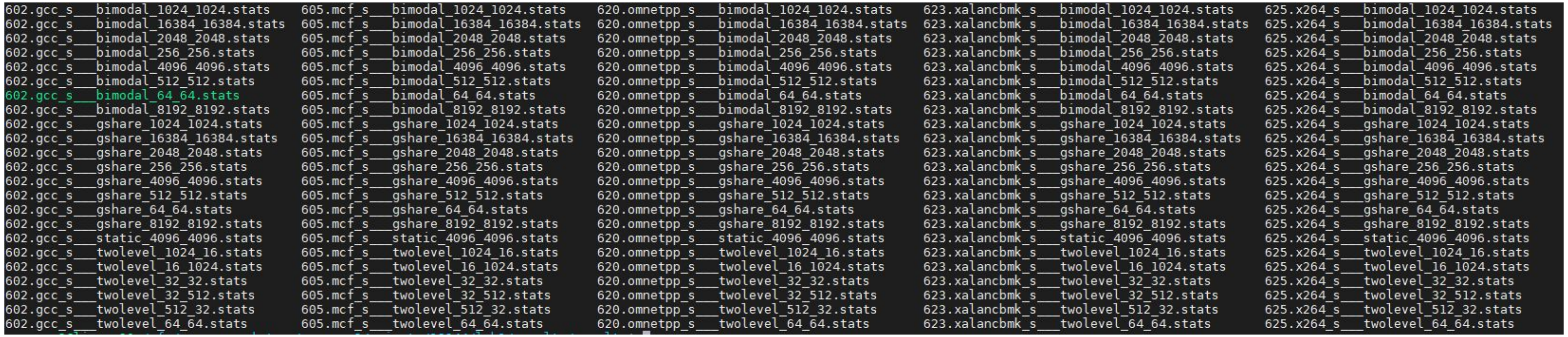
The same thing, but in Python!



```python
import os
import sys

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

dir344 = '/afs/ece.cmu.edu/usr/andrewID/private/18344/'
dirlab = dir344 + 'lab1/'

pintool = os.path.join( dirlab + 'obj-intel64/', 'bp_pintool.so' )

bp_arr = ['static', 'bimodal', 'gshare', 'twolevel']
ght_sizes = ['4096']
bht_sizes = ['4096']

for bp in bp_arr:
    for ght in ght_sizes:
        for bht in bht_sizes:
            result_file = os.path.join(dirlab + 'results/', benchmark + '.csv')
            bpoptions = '-b %s -bht %s -ght %s' % (bp, bht, ght)
            options = '-o %s %s' % (result_file, bpoptions)# , l2options)
            pin_cmd = 'pin -t %s %s -- %s' % (pintool, options , command)
            os.system(pin_cmd)
```

Outputting Data

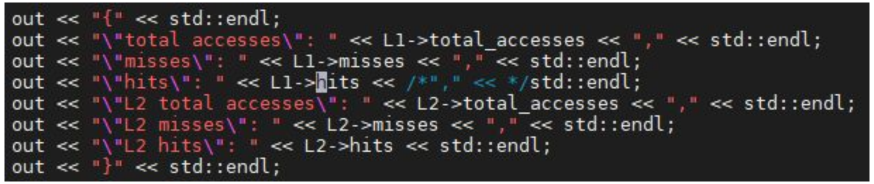
The default outputs aren’t exactly scalable



```
602.gcc_s___bimodal_1024_1024.stats    605.mcf_s___bimodal_1024_1024.stats    620.omnetpp_s___bimodal_1024_1024.stats    623.xalancbmk_s___bimodal_1024_1024.stats    625.x264_s___bimodal_1024_1024.stats
602.gcc_s___bimodal_16384_16384.stats  605.mcf_s___bimodal_16384_16384.stats  620.omnetpp_s___bimodal_16384_16384.stats  623.xalancbmk_s___bimodal_16384_16384.stats  625.x264_s___bimodal_16384_16384.stats
602.gcc_s___bimodal_2048_2048.stats    605.mcf_s___bimodal_2048_2048.stats    620.omnetpp_s___bimodal_2048_2048.stats    623.xalancbmk_s___bimodal_2048_2048.stats    625.x264_s___bimodal_2048_2048.stats
602.gcc_s___bimodal_256_256.stats      605.mcf_s___bimodal_256_256.stats      620.omnetpp_s___bimodal_256_256.stats      623.xalancbmk_s___bimodal_256_256.stats      625.x264_s___bimodal_256_256.stats
602.gcc_s___bimodal_4096_4096.stats    605.mcf_s___bimodal_4096_4096.stats    620.omnetpp_s___bimodal_4096_4096.stats    623.xalancbmk_s___bimodal_4096_4096.stats    625.x264_s___bimodal_4096_4096.stats
602.gcc_s___bimodal_512_512.stats      605.mcf_s___bimodal_512_512.stats      620.omnetpp_s___bimodal_512_512.stats      623.xalancbmk_s___bimodal_512_512.stats      625.x264_s___bimodal_512_512.stats
602.gcc_s___bimodal_64_64.stats        605.mcf_s___bimodal_64_64.stats        620.omnetpp_s___bimodal_64_64.stats        623.xalancbmk_s___bimodal_64_64.stats        625.x264_s___bimodal_64_64.stats
602.gcc_s___bimodal_8192_8192.stats    605.mcf_s___bimodal_8192_8192.stats    620.omnetpp_s___bimodal_8192_8192.stats    623.xalancbmk_s___bimodal_8192_8192.stats    625.x264_s___bimodal_8192_8192.stats
602.gcc_s___gshare_1024_1024.stats     605.mcf_s___gshare_1024_1024.stats     620.omnetpp_s___gshare_1024_1024.stats     623.xalancbmk_s___gshare_1024_1024.stats     625.x264_s___gshare_1024_1024.stats
602.gcc_s___gshare_16384_16384.stats   605.mcf_s___gshare_16384_16384.stats   620.omnetpp_s___gshare_16384_16384.stats   623.xalancbmk_s___gshare_16384_16384.stats   625.x264_s___gshare_16384_16384.stats
602.gcc_s___gshare_2048_2048.stats     605.mcf_s___gshare_2048_2048.stats     620.omnetpp_s___gshare_2048_2048.stats     623.xalancbmk_s___gshare_2048_2048.stats     625.x264_s___gshare_2048_2048.stats
602.gcc_s___gshare_256_256.stats       605.mcf_s___gshare_256_256.stats       620.omnetpp_s___gshare_256_256.stats       623.xalancbmk_s___gshare_256_256.stats       625.x264_s___gshare_256_256.stats
602.gcc_s___gshare_4096_4096.stats     605.mcf_s___gshare_4096_4096.stats     620.omnetpp_s___gshare_4096_4096.stats     623.xalancbmk_s___gshare_4096_4096.stats     625.x264_s___gshare_4096_4096.stats
602.gcc_s___gshare_512_512.stats       605.mcf_s___gshare_512_512.stats       620.omnetpp_s___gshare_512_512.stats       623.xalancbmk_s___gshare_512_512.stats       625.x264_s___gshare_512_512.stats
602.gcc_s___gshare_64_64.stats         605.mcf_s___gshare_64_64.stats         620.omnetpp_s___gshare_64_64.stats         623.xalancbmk_s___gshare_64_64.stats         625.x264_s___gshare_64_64.stats
602.gcc_s___gshare_8192_8192.stats     605.mcf_s___gshare_8192_8192.stats     620.omnetpp_s___gshare_8192_8192.stats     623.xalancbmk_s___gshare_8192_8192.stats     625.x264_s___gshare_8192_8192.stats
602.gcc_s___static_4096_4096.stats     605.mcf_s___static_4096_4096.stats     620.omnetpp_s___static_4096_4096.stats     623.xalancbmk_s___static_4096_4096.stats     625.x264_s___static_4096_4096.stats
602.gcc_s___twolevel_1024_16.stats     605.mcf_s___twolevel_1024_16.stats     620.omnetpp_s___twolevel_1024_16.stats     623.xalancbmk_s___twolevel_1024_16.stats     625.x264_s___twolevel_1024_16.stats
602.gcc_s___twolevel_16_1024.stats     605.mcf_s___twolevel_16_1024.stats     620.omnetpp_s___twolevel_16_1024.stats     623.xalancbmk_s___twolevel_16_1024.stats     625.x264_s___twolevel_16_1024.stats
602.gcc_s___twolevel_32_32.stats       605.mcf_s___twolevel_32_32.stats       620.omnetpp_s___twolevel_32_32.stats       623.xalancbmk_s___twolevel_32_32.stats       625.x264_s___twolevel_32_32.stats
602.gcc_s___twolevel_32_512.stats      605.mcf_s___twolevel_32_512.stats      620.omnetpp_s___twolevel_32_512.stats      623.xalancbmk_s___twolevel_32_512.stats      625.x264_s___twolevel_32_512.stats
602.gcc_s___twolevel_512_32.stats      605.mcf_s___twolevel_512_32.stats      620.omnetpp_s___twolevel_512_32.stats      623.xalancbmk_s___twolevel_512_32.stats      625.x264_s___twolevel_512_32.stats
602.gcc_s___twolevel_64_64.stats       605.mcf_s___twolevel_64_64.stats       620.omnetpp_s___twolevel_64_64.stats       623.xalancbmk_s___twolevel_64_64.stats       625.x264_s___twolevel_64_64.stats
```

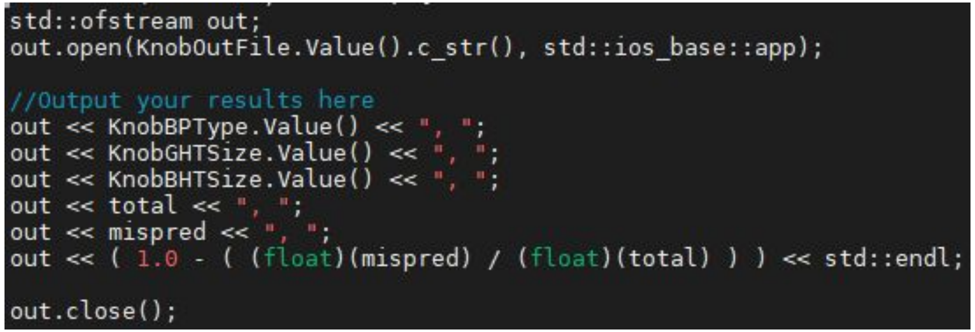
Outputting Data cont’d

Choose a data format you like!



```
out << "{" << std::endl;
out << "\"total accesses\": " << L1->total_accesses << "," << std::endl;
out << "\"misses\": " << L1->misses << "," << std::endl;
out << "\"hits\": " << L1->hits << /*"," << */std::endl;
out << "\"L2 total accesses\": " << L2->total_accesses << "," << std::endl;
out << "\"L2 misses\": " << L2->misses << "," << std::endl;
out << "\"L2 hits\": " << L2->hits << std::endl;
out << "}" << std::endl;
```

Like json



```
std::ofstream out;
out.open(KnobOutFile.Value().c_str(), std::ios_base::app);

//Output your results here
out << KnobBPType.Value() << ", ";
out << KnobGHTSize.Value() << ", ";
out << KnobBHTSize.Value() << ", ";
out << total << ", ";
out << mispred << ", ";
out << ( 1.0 - ( (float)(mispred) / (float)(total) ) ) << std::endl;

out.close();
```

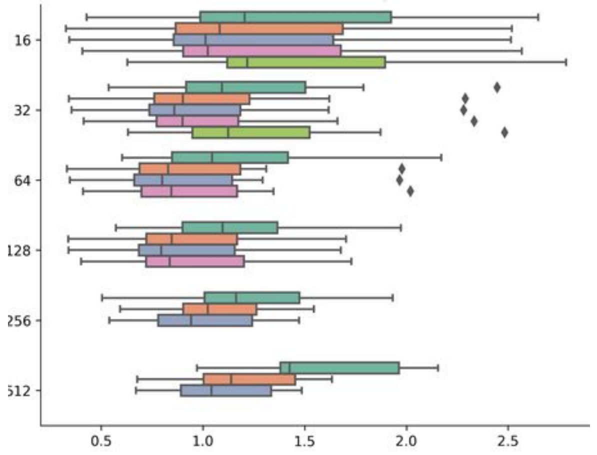
Or csv

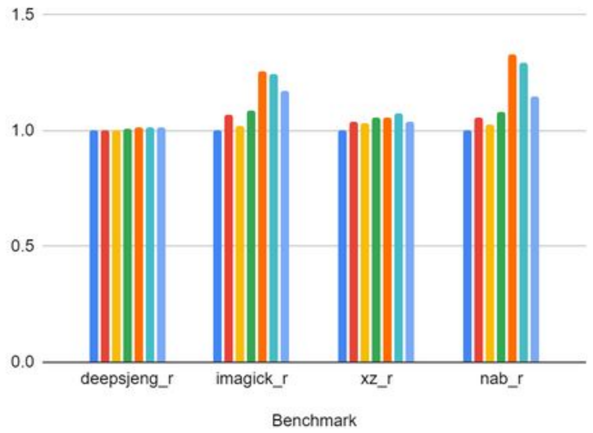
Nb: this file appends data instead of just replacing

Visualizing Results

Choose a graphing systems

- Your favorite spreadsheet software (e.g. Google Sheets or Excel)
- Plotting tools like Matplotlib
    - You can use scientific notebooks like JuPyter notebook
- Matlab???





Branch Predictor Review