COL 718 Architecture of High Performance Computers
Assignment 1
Devise a Branch Predictor

August 18, 2022

This assignment has 2 parts. In the first part, you need to devise a branch predictor, that satisfies the given area constraints and achieves a given minimum accuracy. Then you have to compare the results obtained using a predictive algorithm from the WEKA toolkit or the Caffe toolkit. In the second part, you have to install the Tejas architectural simulator, and understand its operation. This assignment is to be done individually.

1 Implement a Branch Predictor

Framework

- Download the assignment and extract it.
- The folder *traces* contains five traces of branch instructions against which your predictor is to be tested. You **DO NOT** have to perform any file operations. The given framework does all that for you. Please concern yourself with only modifying the files *Predictor2400.java*, *Predictor6400.java*, *Predictor9999.java,* and *Predictor32000.java*. **No other file is to be modified**.
- Run the following commands:
  - ant clean: to clean class files and jar files from previous builds.
  - ant: to compile the java files.
  - ant make-jar: to create a jar file (akin to an X86 executable).
  - java -jar jar/BranchPredictor.jar traces/trace1 2400: this runs the simulator with the input trace *trace1*. It also tells the simulator that the maximum allowed predictor size is 2400 bits. More on this later.

Designing a Branch Predictor

As mentioned earlier, you need to design a branch predictor that achieves good accuracy within an area budget. Specifically, the question has four subparts:

maximum predictor size = (i) 2400 bits, (ii) 6400 bits and (iii) 9999 bits (iv)32000 bits. So, you need to design a predictor with any number of tables and registers with the constraint the sum of the table sizes (number of entries in the table bits per entry) and register sizes (number of bits in the register) **MUST** be less than the specified maximum.

Implementing your Design

- All your changes are to be limited to the files *Predictor2400.java*, *Predictor6400.java*, *Predictor9999.java* and *Predictor32000.java*. **DO NOT** change any other file.
- For sub-part:
  - (i), viz. 2400 bits, implement your predictor in the file Predictor2400.java .
  - (ii), viz. 6400 bits, implement your predictor in the file Predictor6400.java .
  - (iii), viz. 9999 bits, implement your predictor in the file Predictor9999.java .
  - (iv), viz. 32000 bits, implement your predictor in the file Predictor32000.java. Implement either the Tage [5] or the Preceptron [6] predictor.
- Each of these Predictor <XXXX>.java files have three functions that need to be implemented:
  - `Predictor()`: This is the constructor. Here you instantiate your tables and registers. **IMPORTANT:** you are allowed to instantiate objects of only the given Table and Register classes. Instantiating any objects or arrays of your own will result in a zero. You may declare your own variables, both as class members and locally in the functions.
  - `boolean predict(long address)`: Perform the branch prediction based on the branch instructions program counter, i.e. address. Return true for taken, false for not taken. To access your tables and registers, use the public functions defined in Table.java and Register.java.
  - `void train(long address, boolean outcome, boolean predict)`: Train the predictor. address refers to the branch instructions program counter, outcome is the actual outcome of the branch, predict is the prediction your predictor made.

- You may declare any other functions you may need within the Predictor<XXXX>.java files. You may declare your own variables, both as class members and locally in the functions.
- Run the commands specified above (ant clean, ant....) to run the simulator. Try for the five different traces, as well as the three different values for the maximum predictor size.

Evaluating your design

Create a .tar.gz archive of the files *Predictor2400.java*, *Predictor6400.java*,*Predictor9999.java* and *Predictor32000.java*. Name it <entry-number>.tar.gz . Run the command python *test.py* <path-to-archive>. It should tell you for the twenty different runs (five traces, four predictor sizes):

- if you instantiated objects or arrays other than Table and Register
- if compilation succeeded
- if there was a runtime error
- if you violated the maximum size constraint
- number of branches simulated and the accuracy achieved
- if the achieved average accuracy for each predictor size (calculated over the five traces) is greater than the stipulated minimum. Run the command python test.py clean to clean the logs, class files and jar files from the previous runs (deletes the directories *bin*, *jar* and *logs*).

Comparing with a Machine Learning Algorithm

Use a prediction algorithm from either of the toolkits WEKA [1] or Caffe [2]. Report the accuracy obtained and the confusion matrix. Specify the classification algorithm and test options chosen. Are the predictions skewed towards one of the classes?

2 Install the Tejas Simulator and Understand its operation

1. Tejas is an open-source, Java-based multicore architectural simulator developed in-house by the Srishti group. It will be used in the subsequent assignments. The instructions to set up Tejas can be found at http://www.cse.iitd.ac.in/tejas/install.html. Additional details about Tejas can be found in [3] and [4].

2. Run *'HelloWorld.c'* program on the simulator.

3. Explore the simulator's working and summarize your findings.

Submitting Your Assignment

We will be evaluating using the same *test.py* script, but with a different set of traces. So, make sure your implementation does not violate any of the rules. Submit only the **.tar.gz** archive, created and named as described above on Moodle. The deadline is **September 5, 2022.**

Grading Scheme

The marks will be based on your accuracy achieved relative to the other students in the class as well as creativity. The top submission achieving the maximum accuracy will receive bonus marks. Also, you need to make a report which includes all the information related to the branch predictors you designed and the findings of the Tejas simulator.

References

1. www.cs.waikato.ac.nz/ml/weka/

2. http://caffe.berkeleyvision.org/

3. www.cse.iitd.ac.in/srsarangi/files/papers/patmospaper.pdf: Sarangi, Smruti R., et al. Tejas: A java based versatile micro-architectural simulator. Power and Timing Modeling, Optimization and Simulation (PAT- MOS), 2015 25th International Workshop on. IEEE, 2015.

4. http://www.cse.iitd.ac.in/tejas/overview.html

5. Seznec A. A case for (partially)-tagged geometric history length predictors. Journal of Instruction Level Parallelism. 2006.

6. Jimnez DA, Lin C. Dynamic branch prediction with perceptrons. InProceedings HPCA Seventh International Symposium on High-Performance Computer Architecture 2001 Jan 19 (pp. 197-206). IEEE.