Branch Prediction Neural Networks for Hard to Predict Branches

ECE338 - Parallel Computer Architecture

Spyros Liaskonis
George Kapakos

Conventional Branch Predictor Architectures

- Existing Conventional Branch Predictors:
  - 1-bit Branch Predictor
  - 2-bit Branch Predictor
  - Gshare Predictor
  - The Perceptron Predictor
  - **TAGE-SC-L Predictor**

TAGE-SC-L Branch Predictor

- The TAGE-SC-L branch predictor is a **state-of-the-art** mechanism that enhances prediction accuracy by combining multiple strategies:
  - **TAGE Predictor**: Utilizes tagged tables indexed by hashed subsequences of global branch history, approximating Partial Pattern Matching (PPM).
  - **Loop Predictor**: Specialized component designed to identify and predict regular loop patterns, improving accuracy for loop-related branches.
  - **Statistical Corrector (SC)**: Employs a perceptron-based model to adjust predictions from the TAGE and loop predictors, especially in cases where statistical biases are detected.

Branch Prediction Metrics

- Metrics for the Evaluation of the Classification Neural Net:
  - **Mispredictions per Kilo (MPKI)**: As MPKI we set the erroneous predictions every 1000 instructions. Gives us the comparison between different branch predictors.
  - **Instructions per Cycle (IPC)**: Describes the boost in the CPU, showcasing how many instructions are executed in each cycle.
  - **Accuracy** : Measures how often a model correctly predicts the outcome. Accuracy is equal to the number of correct predictions divided to all the predictions.

Near-Perfect Accuracy Isn't Enough: H2P Branches Still Hurt Performance

- State-of-the-art predictors (e.g. TAGE-SC-L) are near-perfect:
  - >99% accuracy on SPEC Int 2017 [1], [2]
- **However,** they systematically mispredict a small, crucial set of branches (H2Ps)
- These mispredictions can reduce Instructions Per Cycle (IPC):
  - **Up to 14%** on Skylake (2015 micro-architecture)
  - **Up to 37.4%** in future, wider/deeper pipelines. (Tarsa et al. [3])

What are Hard-To-Predict (H2P) branches?

According to Tarsa et al. [2], a hard-to-predict branch is a branch that:

- Appear more than 15.000 times (per 30 million instructions).
- Misspredicted at least 1000 times.
- Less than 99% accuracy when using the TAGE-SC-L predictor.

CNNs for H2P Branches

1. CNN Predictor Helper (Tarsa et al. [3])
   - **Assists baseline predictors** (e.g., TAGE-SC-L) on H2Ps.
   - **Trained offline** for a specific static branch identified via profiling.
   - A ternary (2-bit) version of the CNN is developed to meet CPU hardware constraints:
     - Filters and weights are quantized to $\{-1, 0, +1\}$.
     - Inference is done via bitwise operations and popcount, avoiding full matrix ops.
     - Memory footprint: as low as 336 bytes per predictor.

CNNs for H2P Branches

On SPECint 2017:

- Full-precision CNNs improved 47% of H2Ps with a **36.6% average reduction in mispredictions.**
- Quantized CNNs improved 24% of H2Ps with a **14.4% reduction.**
- Even when TAGE is scaled 8× (to 64KB), CNNs still improve 21% of H2Ps, proving the fundamental advantage of better pattern matching.

Based on the work by Tarsa et al. [3], Zangeneh et al. developed a convolutional neural network called BranchNet [4]

- Different variants of BranchNet have been developed:
  - Big-BranchNet
  - Mini-BranchNet

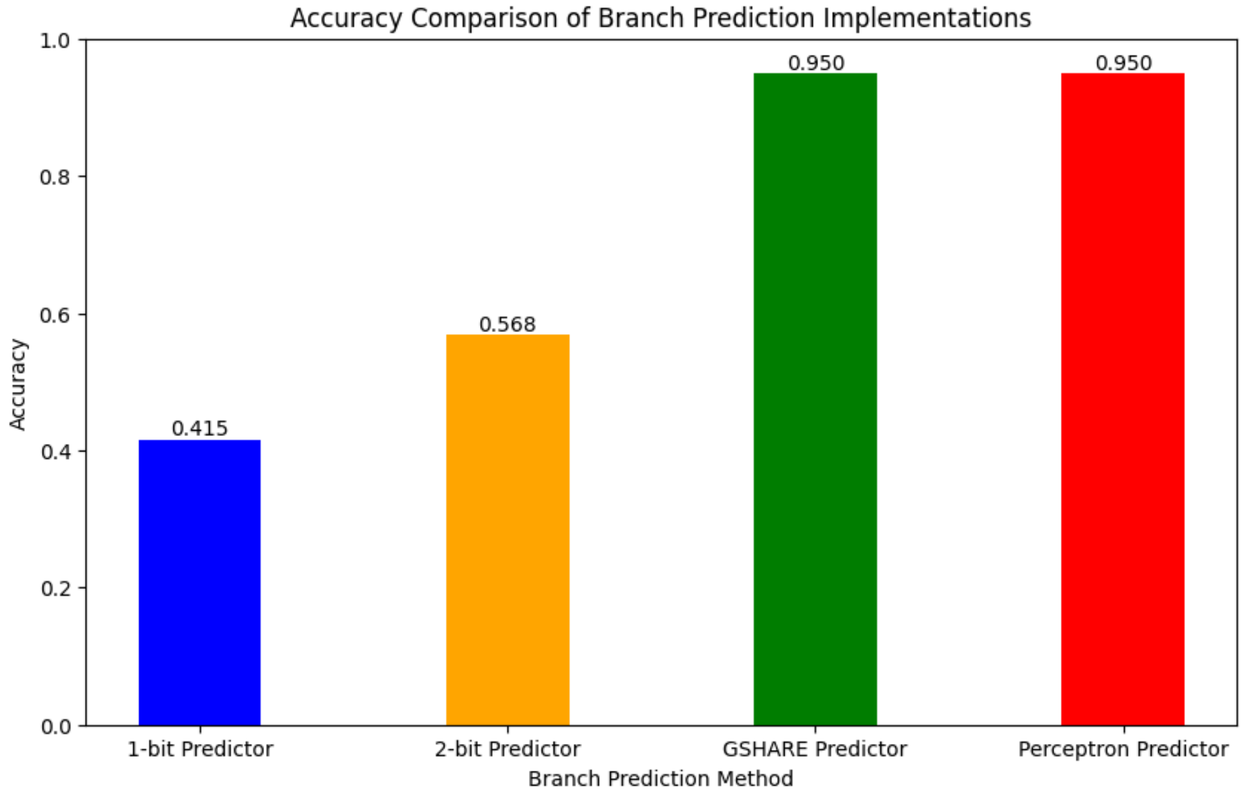
Progress so far...

- Discovered and familiarized ourselves with benchmarks and tools that we are going to use:
  - **SPEC2017** is a benchmark suite used for the evaluation of compute-intensive performance of modern processors. We are going to generate branch prediction workloads.
  - **Intel Pin:** instrumentation tool to generate traces from given programs/ workloads (e.g. SPEC 2017).
  - **ChampSim:** trace-based simulator for a microarchitecture study.
  - **SimPoint** is used to identify and focus on the most common execution periods of the program. We specify the % of the program that is the most represented.

Progress so far…

- Researched already implemented Neural Net Architectures:
  - BranchNet
  - TarsaNet
  - LSTM [5]
- Implementation of Conventional Branch Prediction on SW, we implemented all the conventional branch predictor architectures.
- Familiarized ourselves with Intel's Pin tool, used for trace creation.
- Evaluation of those methods on a dataset.

Results of the Conventional Architecture Implementations



Workflow

Current workflow consists of:

1. Generating traces from applications (e.g. Spec Int 2017) using Intel's Pin Tool.
2. Using these traces with ChampSim, which will simulate a given branch predictor
3. Get results for different branch predictor architectures.

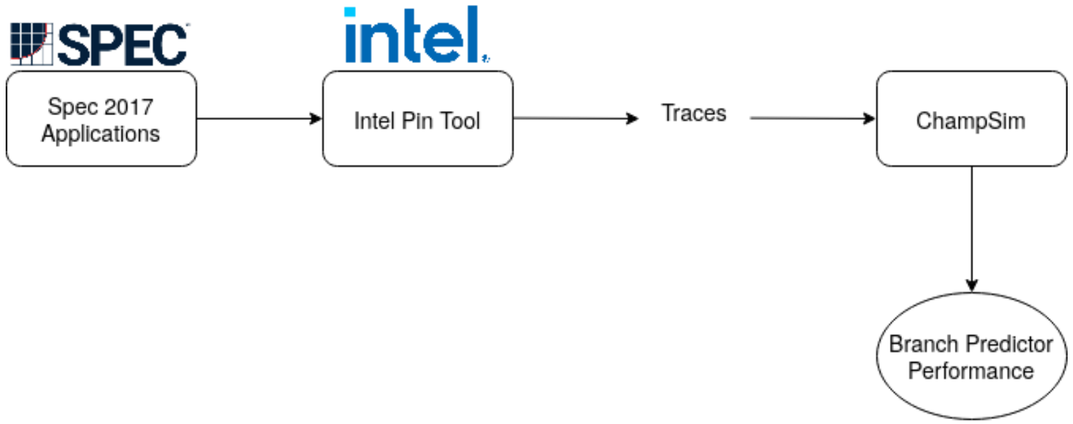


Figure 1: Workflow for evaluating a branch predictor

What needs to be done:

1. Identify hard-to-predict branches (using ChampSim).
2. Use ChampSim for generating tuples like: {Branch IP, Outcome}.
3. Apply preprocessing, encoding, and generate a dataset for training the networks.
4. Develop neural network architecture (based on BranchNet) and evaluate on the implemented workflow.

References

[1] **A Fog. The microarchitecture of intel, amd, and via cpus. An Optimization Guide for Assembly Programmers and Compiler Makers.** Copenhagen University College of Engineering, 2018.

[2] A Seznec. TAGE-SC-L Branch Predictors Again. In Proc. 5th Championship on Branch Prediction, 2016.

[3] Tarsa, S. J., Lin, C., Keskin, G., Chinya, G., & Wang, H. (2019). **Improving Branch Prediction By Modeling Global History with Convolutional Neural Networks.** ArXiv. https://arxiv.org/abs/1906.09889

[4] Siavash Zangeneh, Stephen Pruett, Sangkug Lym, and Yale N. Patt. **BranchNet: A convolutional neural network to predict hard-to-predict branches**. In 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 118–130, 2020.

[5] aristotelis96 (2022) thesis-branch-prediction-ml Github

Thanks for your attention !

Any Questions ?