VPC-based Indirect Branch Predictor

Ankur Roy Chowdhury
ankurrc@tamu.edu

Objective

This project aims to simulate an indirect branch predictor based on the VPC algorithm [1], executed within the CBP2 (ver.3) infrastructure.

Introduction

Indirect branch prediction (IBP) involves two stages: branch direction prediction and target address prediction. While branch direction prediction is a binary decision, indirect target prediction is an N-ary decision. Also, modern OOP-based languages rely heavily on dynamically-dispatched function calls that benefit from IBPs. Most existing IBPs, however, are expensive in terms of hardware resources [2,3], resulting in high power consumption and complexity. Therefore, a low-cost - yet efficient - IBP is required. The VPC-based predictor is one such IBP.

Overview

A Virtual Program Counter (VPC) predictor treats a single indirect branch as multiple conditional branches (virtual branches) in hardware for prediction purposes. Conceptually, each virtual branch has its own unique target address, and the target address is stored in the BTB with a unique virtual PC. The processor uses the outcome of the existing conditional branch predictor to predict each virtual branch. The processor accesses the conditional branch predictor and the BTB with the virtual PC address of a virtual branch. If the prediction for the virtual branch is "taken," the target address provided by the BTB is predicted as the next fetch address (i.e. the predicted target of the indirect branch). If the prediction of the virtual branch is "not-taken," the processor moves on to the next virtual branch: it tries a conditional branch prediction again with a different virtual PC. The processor repeats this process until the conditional branch predictor predicts a virtual branch as taken. VPC prediction stops if none of the virtual branches is predicted as taken after a limited number of virtual branch predictions.

Algorithm

- *Prediction*
  1. Set VPCA to PC value and VGHR to GHR value. Set *iteration* to 1.
  2. Repeat steps 3-6.
  3. Get the *predicted target* from the BTB indexed by VPCA, and the *predicted direction* from the branch predictor accessed via the VPCA and the VGHR values.
  4. If *predicted direction* is "TAKEN" and *predicted target* is not empty, then set the next instruction address to *predicted target* and exit the loop.
  5. If *predicted target* is empty from step 3 or the loop has reached maximum iterations (MAX_ITER), then *STALL* (set True) the pipeline, and exit loop.
  6. Set next value for VPCA by hashing the PC with the current *iteration* value, and set VGHR by left-shifting the current VGHR value. Increment the *iteration*.

- *Training on correct prediction*
  1. Set VPCA to PC value and VGHR to GHR value. Set *iteration* to 1.
  2. Repeat Steps 3-5, while *iteration* is less than the *predicted iteration* value from prediction stage.

3. If iteration is equal to *predicted iteration,* then update the corresponding VGHR entry for the VPCA value as "TAKEN". Also, update the replacement policy bits of the BTB for the VPCA.
4. Else, update the corresponding VGHR entry as "NOT TAKEN".
5. Set next value for VPCA by hashing the PC with the current *iteration* value, and set VGHR by left-shifting the current VGHR value. Increment the *iteration*.

- *Training on mis-prediction*
  1. Set VPCA to PC value and VGHR to GHR value. Set *iteration* to 1.
  2. In case a target was found but mis-predicted, repeat steps 3-6 while *iteration* is less than MAX_ITER and *found correct target* is False.
  3. Set predicated target by accessing the BTB entry for the VPCA.
  4. If predicated target is equal to CORRECT TARGET from the predication stage, *,* then update the corresponding VGHR entry for the VPCA value as "TAKEN". Also, update the replacement policy bits of the BTB for the VPCA.
  5. Else, update the corresponding VGHR entry as "NOT TAKEN".
  6. Set next value for VPCA by hashing the PC with the current *iteration* value, and set VGHR by left-shifting the current VGHR value. Increment the *iteration*.
  7. In case no target was found in prediction stage, then set VPCA value to VPCA corresponding to the virtual branch with a BTB-Miss or Least-frequently-used target among all virtual branches and VGHR value to VGHR corresponding to the virtual branch with a BTB-Miss or Least-frequently-used target among all virtual branches.
  8. Insert into BTB, the CORRECT TARGET corresponding to the VPCA and update the branch predictor with the values derived from the VGHR, VPCA and set it as "TAKEN".

Project Plan

- *Phase 1* (10/01): Choose and implement the Conditional Branch Predictor.
- *Phase 2* (10/11): Implement VPC-branch prediction algorithm.
- *Phase 3* (10/18): Implement VPC-training algorithm.
- *Phase 4* (10/24): Integrate and review.

References

1. Kim, H., Joao, J. A., Mutlu, O., Lee, C. J., Patt, Y. N., & Cohn, R. (2007). VPC prediction. *ACM SIGARCH Computer Architecture News, 35*(2), 424. doi:10.1145/1273440.1250715
2. A. Seznec and P. Michaud. A case for (partially) TAgged Geometric history length branch prediction. *Journal of Instruction-Level Parallelism (JILP)*, 8, Feb. 2006.
3. J. Kalamatianos and D. R. Kaeli. Predicting indirect branches via data compression. In *MICRO-31*, 1998.
4. D. A. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *HPCA-7*, 2001.