CS 203: Advanced Computer Architecture

Practical Load Value Predictor - Project Report

-Bhagyesh Gaikwad

Introduction:

Predictors are used in computer architecture so that we could increase throughput and reduce the latency caused by the gap between software and hardware. Branch prediction is proven to cope with the latency by a great amount. Over the years, it has moved from concept stage to actual implementation and exists in the majority of the processors available in the market today.

The purpose of load value prediction is to be able to predict the value of a memory load instruction. This sounds impossible as we could have a billion possible values for a load value. But it is observed that these values can be predicted to a large extent by observing the program's temporal and spatial locality.

We have chosen to implement a 'Practical Load Value Predictor' with various design tweaks. The inclination towards choosing this over Machine learning Load Value Predictor was that, even if we are able to predict load values with high accuracy we felt that it would defeat the purpose of saving the computation time which we wish to overcome. We also wanted to test it on basic designs as there are no particular features that are influential here for prediction other than some primary modifications that we aim to explore in this project.

Methodology:

- We have implemented the below three designs for load value predictors:

1. **Last Value Predictor:**

    - In this design we use the last value that was loaded from the current address that we are accessing.
    - We have used it as a baseline predictor and improved the design further from this design.
    - Design:
        - We have used a python dictionary to store the Program Counter and Memory Address as key and store the load value for the same.
        - This saved us space as creating a 2d array was consuming a lot of memory where every address from the mapped arrays wasn't accessed.

2. **Global Load Value Predictor:**

    - Here we speculate that the values for the current address would be influenced by the last n values for the load instruction.

- This design takes into consideration that irrespective of the memory addresses accessed, the global values for the load instruction would likely repeat.
- Design:
  - We have used a python dictionary to store only the Program Counter as key and store the load value for the same.
  - To choose n we ran our predictor from range 0-20 to test the accuracy and saw that the accuracy decreased as we increased n.
  - We thus averaged out the first 10 values and chose ‘n=5’ in our code for simplicity of not evaluating it for every train and test file.
  - We predicted a value that had the highest count among the vector for the load.
  - This vector for every load is updated by removing the oldest value and appending the latest one.

3. **Local Load Value Predictor:**

- In this design we speculate that the values for the current address would be influenced by the last n values local values for the load instruction at a particular memory address.
- The motivation for the design was that the load values would be influenced by the memory address which a load instruction accesses every time.
- Design:
  - Similar to ‘Global Load Value Predictor’ above except we use the key for dictionary as Program Counter and Memory Address

Predictable Loads Last Value Predictor:

- This design is influenced by the fact that many of the load values keep changing for some of the load which increase the misprediction rate.
- Thus the cost of mispredictions can be mitigated by identifying the unpredictable loads, and not predict the values for them at all.
- This design is theresore an extension to the ‘Last Value Predictor’
- To achieve this we kept a threshold of incorrect value that a particular instruction would be tolerated, after the threshold is reached we stop predicting the load values.
- Design:
  - To determine the threshold value, we took the mean number that a load instruction would have. (i.e Total Load Values / Load Instructions)
  - We tweaked the design here from the ones above, we used a 2-d matrix of unique Program Counters v/s unique Memory addresses to store the last load values.
  - We maintained a list of unpredictable loads in which further access were skipped if the load instruction passed the threshold value.
  - In this process we have already predicted some values from the unpredictable loads which were taken into consideration during calculation of accuracy.

Results & Evaluation:

- We have used a train and a test file for which we have achieved the below results:

Predictor	Train Accuracy (%)	Test Accuracy (%)
Last Value Predictor	63.35	64.08
Global Load Value Predictor	47.62	48.53
Local Load Value Predictor	55.95	56.94
Predictable Loads - Last Value Predictor	82.87	82.92

- From the table above we can observe that the Global and Local Load value predictors have performed worse than the Last Value predictor.
- This pertains to the fact that there is seasonality in the series that we are predicting and it is heavily influenced by the last value which is repeated frequently.
- This also points to the fact that the values here are more temporally predictable than spatially predictable.

- The 'Predictable Loads - Last Value Predictor' has outperformed the others by quite a margin.
- For practical purposes we feel that this is a much better predictor to be used as we save a lot of time by not flushing ROB and restarting everytime we mispredict.
- The threshold can be increased depending on the computational time where we expect to maintain a positive ratio for predicted vs not predicted loads.

References:

1. Mikko H. Lipasti, Christopher B. Wilkerson, and John Paul Shen. Value Locality and Load Value Prediction in ACM, 1996

2. Martin Burtscher and Benjamin G. Zorn. Load Value Prediction using Prediction Outcome Histories, 1998

3. Martin Burtscher. An improved index function for (D)FCM Predictors in ACM, 2002