CS6600-Assignment 3

Lakshya J, Arun Krishna

November 2022

1 Basic Predictors

First we give a brief explanation of the simplest branch predictors implemented and still in use until now.

1.1 Bimodal Branch Predictor

An n-bit bimodal branch predictor is a saturating predictor where the entry to the hash map is incremented when the branch is taken and decremented otherwise. The most significant bit determines whether the branch is taken or not.

1.2 Local History Branch Predictor

Each history table entry records the direction taken by the most recent n branches whose addresses map to this entry, where n is the length of the entry in bits. The second table is an array of m-bit counters identical to those used for bimodal branch prediction. However, the branch history may reflect a mix of histories of all the branches that map to each history entry and since there is only one counter array for all branches, there may be conflict between patterns.

1.3 Global History Predictors

In the local branch prediction scheme, the only patterns considered are those of the current branch. Global Branch Prediction takes advantage of all the recent branch patterns observed to make a prediction. The dependency of the branches on other branches can be exploited in a global branch predictor. It can also duplicate the behaviour of a local branch predictor. One simple implementation is using shift registers. Since global history predictor can capture more information than just the last few branches, it performs better than the bimodal predictor.

1.4 Global Predictor with Index Selection

Since only global history is not as good as local predictor, we can include the branch address as well to make a branch prediction (gselect). There is a trade-off between using more history bits or more address bits. This is an implementation of branch predictor with correlation.

1.5 Global History with Index Sharing

There is a lot of redundancy in the counter used by gselect. When there are enough address bits to identify the branch, we can expect the frequent global history combinations to be rather sparse. Taking advantage of this, one proposes hashing the branch address the global history together. gshare XORs the branch address and the global history producing unique predictions for varied cases compared to gselect.

1.6 Combining bimodal and gshare

Global information can be used if it is worthwhile, otherwise the usual branch direction as predicted by the bimodal scheme can be used. The combined predictor always does better than either predictor alone.

1.7 Perceptron Model

This predictor uses only pattern based information and the same weight vector is trained for all the branches taken. A path based predictor on the other hand also uses path information where the i^{th} correlating weight is chosen for the i^{th} branch address. Perceptron branch predictors achieve higher accuracy by capturing correlation from very long histories, however they have higher training latency due to higher complexities.

1.8 Conclusion

For small predictors, the bimodal scheme is relatively good. As the number of counters doubles, roughly half as many branches will share the same counter. As more counters are added, eventually each frequent branch will map to a unique counter. The information content in each additional address bit declines to zero for increasingly large counter tables. The perceptron model is useful only in the case of linearly separable data, which is not the case in most of the workload suites.

2 Our Predictor

After going through basic implementations of many algorithms for branch prediction, we have decided to use an algorithm inspired by 'Stacking', which is a common technique used in machine learning. Stacking is an ensemble method

that enables the model to learn how to use combine predictions given by learner models with meta-models and prepare a final model with accurate prediction. Weighted Average Ensemble uses a diverse collection of model types as contributing members. It involves tuning the coefficient weightings for each model using an optimization algorithm and performance on a holdout dataset.
In our case we am initializing the weights to equal values at the beginning, and let the model learn dynamically the weights according to the individual predictions made by it. In case of incorrect predictions we reduce the weight associated with that model, otherwise we increase it.

We have combined the bimodal, gshare and perceptron (Jiménez, Daniel A., and Calvin Lin. "Dynamic branch prediction with perceptrons." Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture. IEEE, 2001.) model to create another model with 4 weights being updated at every time step based on whether the individual branch predicted correctly or not. We update the weights based on the difference between the predicted output by the stacked model versus the prediction by the model associated with that weight.

Since each of the models decide whether or not to take a branch based on a threshold, we center all the observed outputs around 0 and update the weights accordingly. The fourth weight is the bias term, which we update whenever the final stacked model predicts wrongly.

We have added a table of four more weights to leverage the predictions made by the different models and weigh it accordingly to get an expected better output on the whole. At each prediction this weight is updated, just like the individual models used to build it. We have not made any changes to the individual models and have kept it as is as we want to compare how the combined model performs against the individual best model.

This algorithm is influenced by the Multiperspective Predictors (Jiménez, Daniel A. "Multiperspective perceptron predictor." 5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5). 2016.) where multiple different predictors are combined to give a single predictor and the feature selection for the weighting of these predictors is done through a genetic algorithm based approach.

The following pseudocode outlines the algorithm used:

Algorithm 1 Implemented Algorithm

Prediction Part
$w \leftarrow \{1, 1, 1, 0\}$
$base\_preds \leftarrow \{bimodal\_pred, gshare\_pred, perceptron\_pred\}$
$base\_outs \leftarrow \{bimodal\_out, gshare\_out, perceptron\_out\}$
$model\_out \leftarrow \{w[0] * base\_outs[0] + w[1] * base\_outs[1] + w[2] * base\_outs[2] + w[3]\}$
if $model\_out > 0$ **then**
    $model\_pred = 1$
else
    $model\_pred = 0$
end if

Updation Part
Update bimodal, gshare and perceptron models
if $model\_pred\ != \ taken$ **then**
    for $i = 0; i < 3; i+ = 1$ **do**
        if $base\_preds[i]$!= $taken$ **then**
            $w[i]$ -= $base\_outs[i] - model\_out$
        else
            $w[i]$ += $base\_outs[i] - model\_out$
        end if
    end for
else
    if $model\_pred == True$ **then**
        $w[3]$ += $w[0] + w[1] + w[2]$
    else
        $w[3]$ -= $w[0] + w[1] + w[2]$
    end if
end if

3 Results

The configuration is as follows:
Basic BTB sets: 1024 ways: 8 indirect buffer size: 4096 RAS size: 64

The results of our predictor for each of the traces are as follows: In the case of 437.leslie3d-134B benchmark, the model learns to embody the perceptron model completely, as the perceptron model individually gives the same results. In other cases it is noticed that our model performs worse than the best individual model.

The model performs the worst in the case of 433.milc.127B, with a branch prediction accuracy of around 56%.

Benchmark	Branch Prediction Accuracy	MPKI	Avg. ROB Occupancy
400.perlbench-41B	92.8733%	15.001	33.1898
403.gcc-16B	99.5196%	0.94292	39.7531
410.bwaves-1963B	67.5105%	9.05782	232.501
416.gamess-875B	88.6357%	10.6852	84.18454
435.gromacs-111B	87.8498%	8.35898	29.818
437.leslie3d-134B	98.224%	0.4712	326.604
444.namd-120B	95.7701%	2.52516	96.2247
401.bzip2-226B	90.6153%	14.958	26.5084
429.mcf-184B	84.4256%	37.4434	16.7112
433.milc-127B	56.4296%	37.4172	50.4812

Table 1: Statistics of our model on the different benchmarks

It is most likely because 4 weights is a small number for a model to embody multiple complexities. Not only that, since the model placed after the stacked model is linear, non-linearities are not properly encapsulated using our model. This can be introduced using non-linear activation functions, however calculating the loss for backpropagation will be harder.