**Mahindra University, Spring 2023**

**EE507 Advanced Computer Architecture**

**Branch Prediction Programming Assignment**

Rishab Ramanathan

19XJ1A0558

**Introduction**

The assignment consists of 5 different branch prediction algorithms, which include: static always taken, static always not taken, dynamic last taken, dynamic bimodal, and dynamic bimodal with branch history. I understand that a sixth assignment, involving free choice in prediction algorithm, was also part of the assignment, but I couldn’t complete it within the given duration.

All programs are written in C (purely for the purpose of speed).

The trace file accuracies and time outputs are detailed in a separate Outputs.docx in the same folder. This report summarizes the findings of those outputs. Execution times may vary with a different systems, the logged times are run on my personal computer (CPU - 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz 8-core)

**The Predictors**

1) Static Always Taken / Static Always Not Taken:

Static Taken maxes accuracy for Trace 4 (~94.5%), and minimizes accuracy for Trace 8 (~33.4%), while the opposite is true for Static Always Not Taken. Higher accuracy percentages for static taken would generally imply the existence of loop branching statements (continuously large number of taken branch, with only the last branch not taken). On average, static taken gives 49% mean accuracy (0.185 deviation), and static not taken gives 51% accuracy with the same amount of deviation.

Being a fairly straightforward algorithm to implement, the time for the simulator to run is purely dependent on number of lines in the trace file, roughly averaging at 0.077 seconds per million lines.

2) Dynamic Last Taken

I chose to use a dynamically generated linked list for this implementation for the purpose of experimentation.

- As the complete list and number of branches in the trace is unknown (by the very nature of branch prediction, only one target address instruction can be at a time in a scalar architecture), the memory in the code is allocated dynamically, with each unique target address holding an individual node.

- Each new trace is searched for uniqueness in the list and is either (1) pushed into the list if it doesn’t exist in the history or (2) updated with the last taken value.

I chose to use a dynamic memory format, since a static list (arrays) would require unique trace size estimation, and overflow in unique traces would cause inaccuracies in prediction (not every address would be possible to be mapped to it’s last dynamic instance).

Accuracy among all the traces averages to ~85.15%, with maximums seen in trace 1 (97%) and minimums seen in trace 2 (61%). Owing to the algorithm, an increase in accuracy would ideally be seen in target addresses maintaining the previous taken state, and conversely a drop in accuracy would imply volatility (the same target shifting between taken and not taken states frequently).

This simulated implementation is not feasible in a hardware environment, as there is no real method to dynamically generate physical memory space. Furthermore, the use of linked lists greatly slows down the complete processing of each trace. This is why the next two branch predictors are implemented keeping a static memory space (arrays) in mind.

3) Dynamic Bimodal

The predictor is implemented in software using a static array of size 2^n, with n as a command line input. N has been limited to 0 < n < 32, as 0 as a size would lead to no prediction, and 32 is one above the maximum possible bit shift possible in an unsigned integer type in C. ([Left bit shift limited to 31](https://stackoverflow.com/questions/7401888/why-doesnt-left-bit-shift-for-32-bit-integers-work-as-expected-when-used)) Bit shifting was used in the simulator to truncate a binary sequence to a specific length (n bits), hence a maximum of 32 was impossible for the aforementioned reasons.

N = 20, 24, 31 all give almost identical counts of accuracy for the given traces. Accuracy saturation occurs with n ~ 20 on calculating average accuracy (~89.5% with deviation of 0.048) for all traces, most likely because the unique branches in the traces don’t overflow the array memory once 2^n becomes sufficiently larger than the unique count of branches. This saturation value may change with an increase in the trace file size/amount of unique branches.

Consequently, accuracy drops sharper and is more volatile with reduced accuracy (~68.3%) and increased deviation (0.121) for n = 2.

This increase in accuracy in comparison with Dynamic Last Taken implies the sort of singular “outlier” taken states in the traces, that would end up as a mis-prediction in dynamic\_LT, but is modulated in dynamic\_bm. An argument could be made of introducing more states (8,16), but that would again translate into issues with volatile branches in trace files, and increased hardware requirements in the form of the size of bits of the register storing the state.

Naturally, an increased size of n would mean a larger memory register of n bits, involving more expensive hardware. Furthermore, larger n’s would translate to read/write latency, with increased complexity in the mux/demux circuits to read the memory register. This hardware increase is more apparent with the introduction of branch history.

4) Dynamic Bimodal with Branch History

With n, a new factor h is also introduced, and set to the same limits as n, with an extra limitation that h < n.

For analysis, n was taken as 20, since that was the inferred saturation limit. H was accordingly varied.

Since h is subject to the same restrictions as n, h cannot be practically set to 0 (refer to the previous bit shift issue in C), hence the simulator consciously forces the history register to be empty for each new iteration, and effectively becoming a normal bimodal predictor in practice. This can be proven as the accuracy is identical for both predictors.

The predictor enjoys a steady increase in accuracy for the trace files with an increase in h size, with the maximum encountered at h = n (for n = 20 and h = 20, mean accuracy comes up to 96.3%, with a deviation of 0.019). Introducing the history register improved the accuracy and more importantly, maintained stability in accuracy through all the trace files.

Just like dynamic bm, the increasing size of h would translate to larger history registers, and a larger capable XOR gate to hash the history and the memory registers for indexing, involving further hardware expenses.