1 Abstract

The coursework specification detailed the implementation and analysis of several branch prediction strategies. These included always taken, two-bit, GShare and profiled predictors. The final implementation additionally includes one-bit, global one-bit and global two-bit predictors. Bit predictors, including the GShare predictor, are tested with a table size range from 32 to 16,384 entries. Tests are done on a range of program traces provided with the specification, both using entire files and smaller portions. In summary, the GShare predictor performed best with a large table size for full trace files and two-bit predictors outperformed one-bit predictors.

2 Implementation Overview

The simulator implements several predictors that can be run for a given trace file. An overview of each predictor is given in Figure 1. The one-bit, two-bit and global predictors were developed using material from lectures. The GShare predictor was developed using online resources as a guide on the concept, including the original GShare paper [1], a lecture on branch prediction [2], another university assignment [3] and documentation for the Berkeley Out-of-Order Machine's branch prediction [4]. Running the simulator without a specified trace file will run all trace files in the `trace-files` folder. Each trace file is run with all possible predictor types and table sizes. After running one configuration, the table size and misprediction rate is exported into a `csv` file for the trace file and predictor type. These files served as graph data for the graphs in this report.

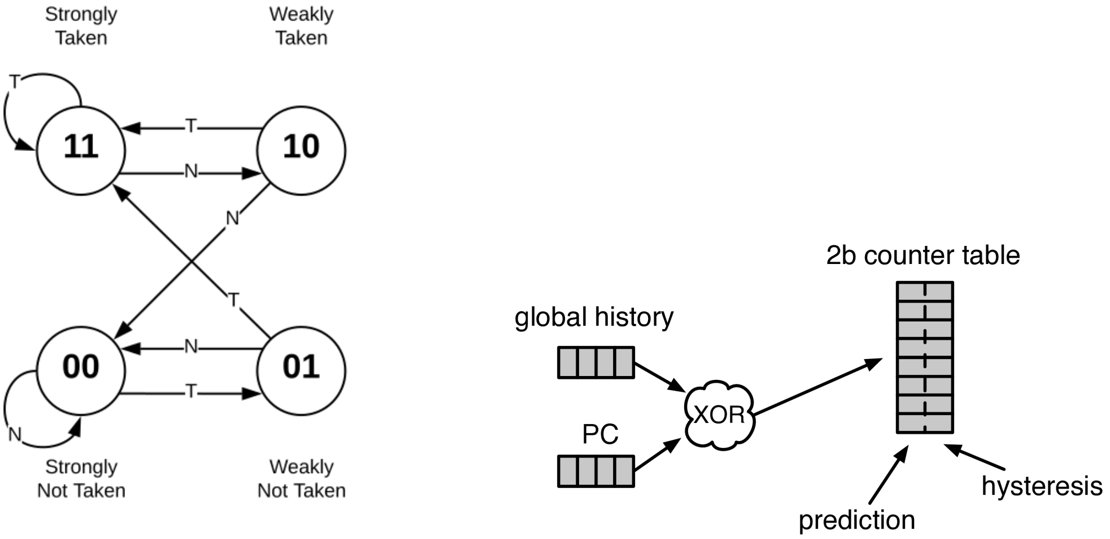
3 Findings

3.1 Full Trace Files

To begin, the predictors were run for the entirety of each of the trace files provided. Most programs showed the trend that two-bit predictors consistently outperformed one-bit predictors. Furthermore the GShare predictor outperformed both other one- and two-bit predictors, but only at sufficiently high table size. This includes the Cactus BSSN (Figure 3(a)), Exchange 2 (Figure 3(b)), GCC (Figure 3(c)), POV-Ray (Figure 3(d)), XZ (Figure 3(e)), WRF (Figure 3(f)) and Leela (Figure 4(a)) trace files. Additionally, WRF (Figure 3(f)) has a very small margin between GShare and the other two-bit predictors. The relative downward trend of misprediction rate with increasing table size is likely due to bigger table size resulting in less overlap between program counter addresses. Two addresses are less likely to share the same table entry and as such less likely to interfere with each other's state. The two-bit predictors outperform the one-bit predictors as two-bit predictors introduce hysteresis (a delay in changing prediction) through their additional bit. The additional bit counters the instability of one-bit predictors, which immediately switch their prediction upon a single mistake. This instability presents an issue in branch patterns that frequently alternate between taken and not taken.

- Always Taken: Always predicts a branch to be taken.
- One-Bit: Keeps a table of booleans used to predict the outcome of a branch. The program counter address space maps directly to the table.
- Two-Bit: Keeps a table of integers used to predict the outcome of a branch. The integer is set from 0 to 3 to represent FSM states (strongly not taken, weakly not taken, weakly taken and strongly taken). The program counter address space maps directly to the table. A diagram of the FSM is shown in Figure 2(a).
- Global: Uses a bit predictor for each possible outcome of previous branches (taken or not taken). Whether to use one-bit or two-bit predictors is set at initialisation. The outcomes are tracked using a boolean value. This value is used to choose the predictor.
- GShare: Uses a long as a global history register for branch outcomes. This is XORed with the program counter address to map into a two-bit predictor table. Bit shifting and setting enable each bit of the long to represent a previous outcome. A diagram is shown in Figure 2(b).
- Profiled: Looks at the outcome of each conditional branch in advance and then chooses to take it if it was taken more than it was not taken.

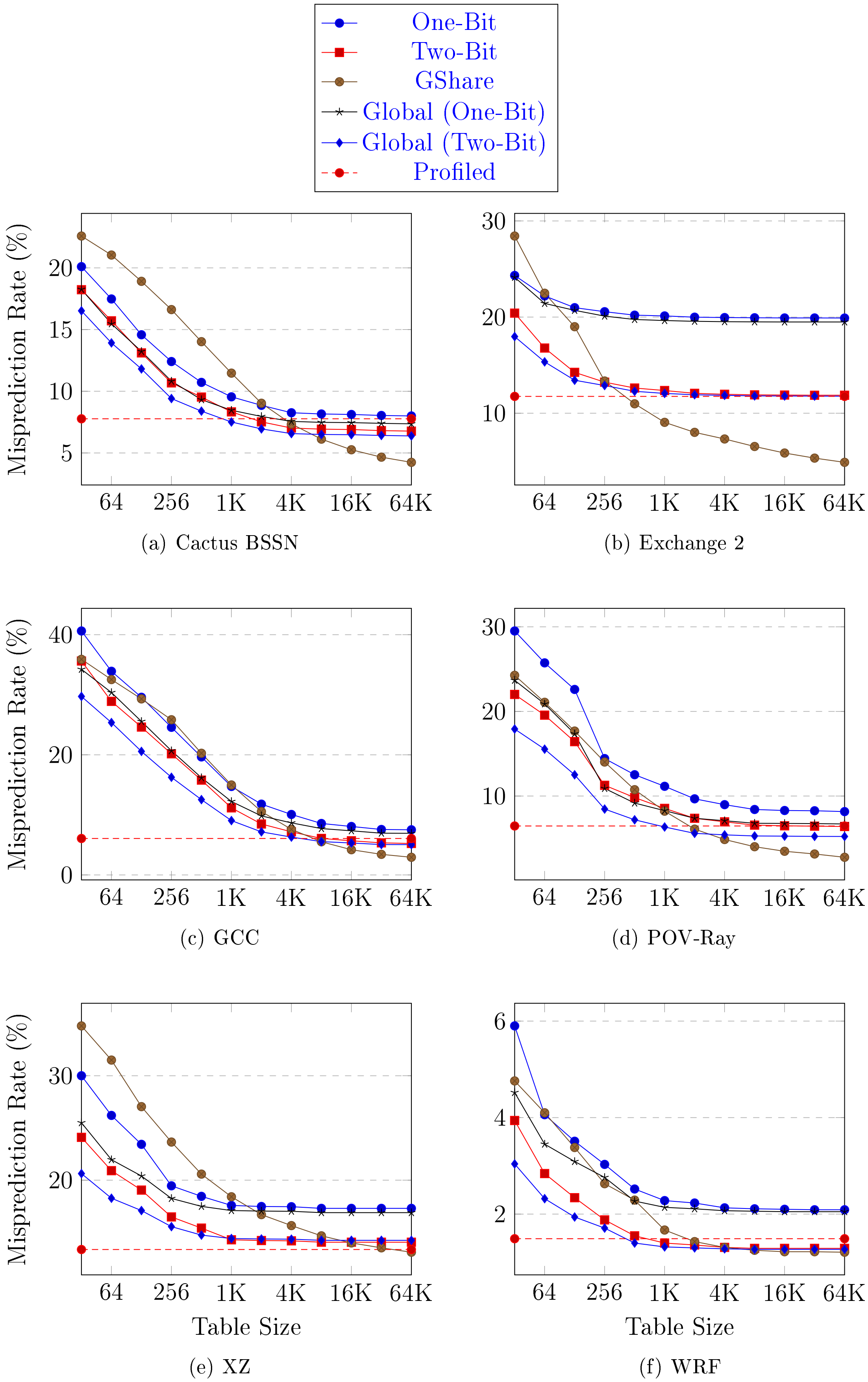
Figure 1: A textual overview of predictor types implemented



(a) Overview of a two-bit predictor's finite-state machine for a table entry [4]

(b) Overview of a GShare predictor, showing XOR mapping into a two-bit predictor table [4]

Figure 2: A visual overview of predictor concepts



(a) Cactus BSSN

(b) Exchange 2

(c) GCC

(d) POV-Ray

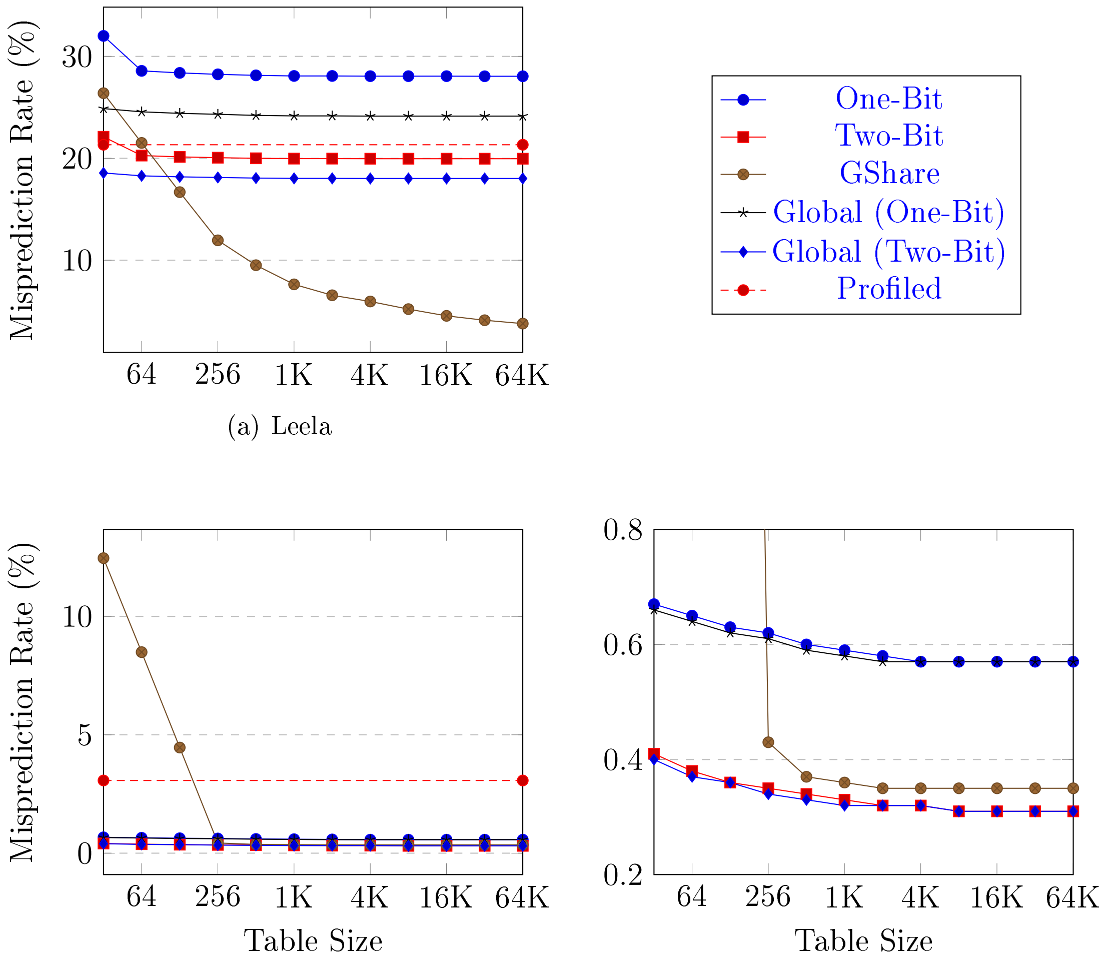
(e) XZ

(f) WRF

Figure 3: Branch predictor performance of several program traces exhibiting common behavior

Also of note is the finding that the standard one- and two-bit predictors in Leela have a relatively constant performance from a table size of 64 (Figure 4(a)). This is likely due to a limited number of program counter addresses used in the address space, making even low table sizes sufficient for maximum predictor performance. While the standard predictors have constant performance, GShare's performance noticeably improves with table size. This is likely due to its increasing ability to use previous branch outcomes to predict future branches.

The only exception to the performance ordering of one-bit, two-bit and GShare is the BWaves trace file (Figures 4(b) and 4(c)). Here, GShare performed worse than the two-bit predictors at all table sizes tested and worse than the one-bit predictors below a table size of 256. It is likely that GShare has significant address conflicts below this table size, causing a much higher misprediction rate.



(a) Leela

(b) BWaves

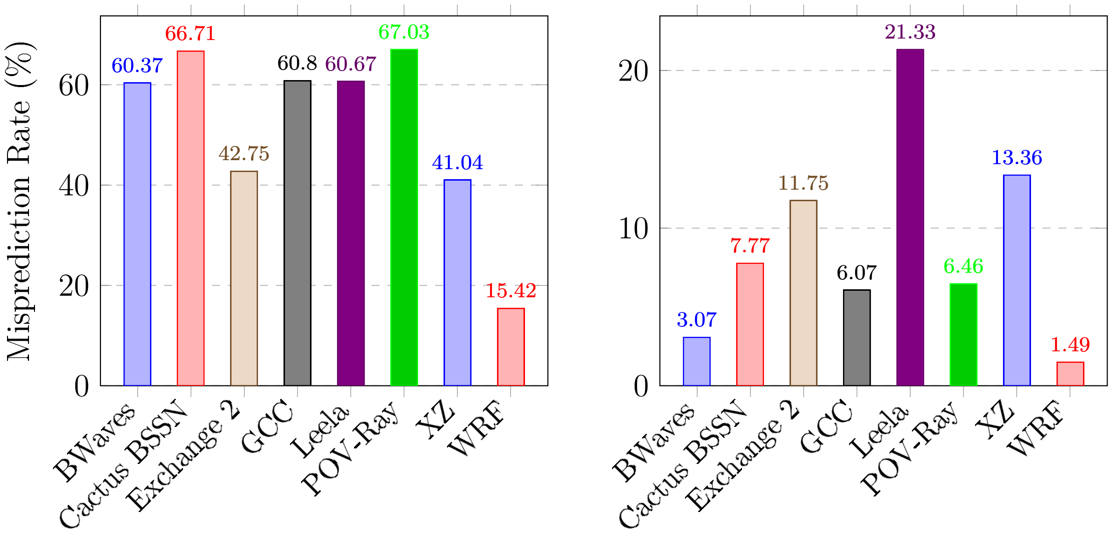
(c) BWaves (Zoomed-In)

Figure 4: Branch predictor performance of several program traces exhibiting unusual behavior

The always taken predictor displayed generally very poor performance, except for the WRF program trace (Figure 5(a)). The misprediction rate of the always taken predictor gives the percentage of conditional branches not taken. This means that the always taken predictor will perform poorly for any programs that do not have a near 100% rate of conditional branches being taken.

Looking at the profiled predictor in isolation, its misprediction rate for the full trace files ranges from 1.49% for WRF to 21.33% for Leela (Figure 5(b)).

Leela's official website states that Leela is a Go playing program that uses a trained neural network. During a game, this neural network is queried to improve prediction from search [5]. The trace file for Leela shows that there are several branches that are reached many times with minor variation. This branch variation matches with how search using a neural network might explore many different options before coming to a conclusion for the next best move.



(a) Always Taken Predictor Performance

(b) Profiled Predictor Performance

Figure 5: Performance of Tableless Predictors

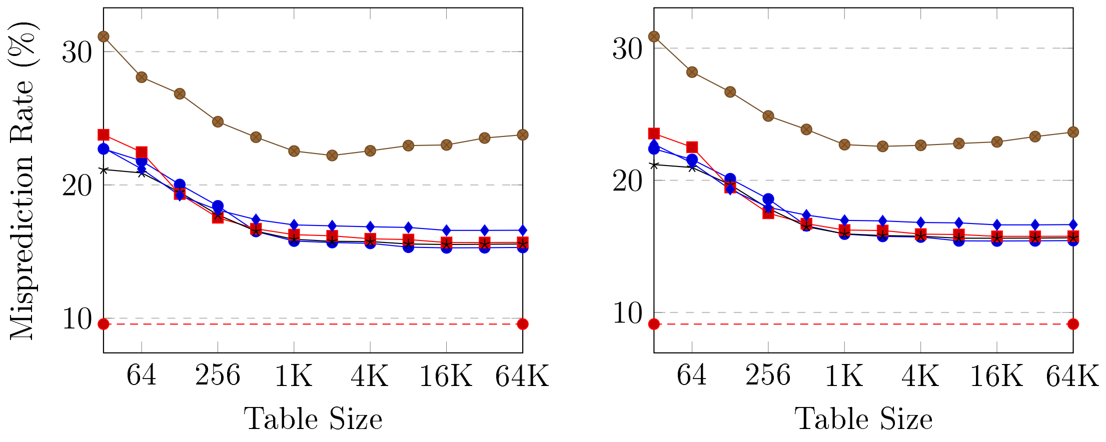
3.2 Partial Trace Files

To look at predictor performance over smaller parts of a trace file, three trace files were broken up into 1,000, 10,000 and 100,000 lines. Cactus BSSN was chosen as it exhibited the most common behaviour identified from the full traces, while Leela and BWaves were chosen as they exhibited behaviour out of the norm. To create the snippets, the code in Listing 1 was run for each trace file and snippet length.

```
    $ tail -n +1000000 trace.out | head -n 1000 > trace1000_offset.out
```

Listing 1: Truncating a trace to 1,000 lines with 1,000,000 offset

Initially, these program trace excerpts used the contents at the start of the full trace file. However, plotting the results of these showed identical results among the different trace files (Cactus BSSN and Leela shown in Figure 6). This suggests that all the trace files share some initialisation code of the platform from which the trace files were generated. To counter this, each snippet is offset by 1,000,000 lines from the start of the corresponding trace file.



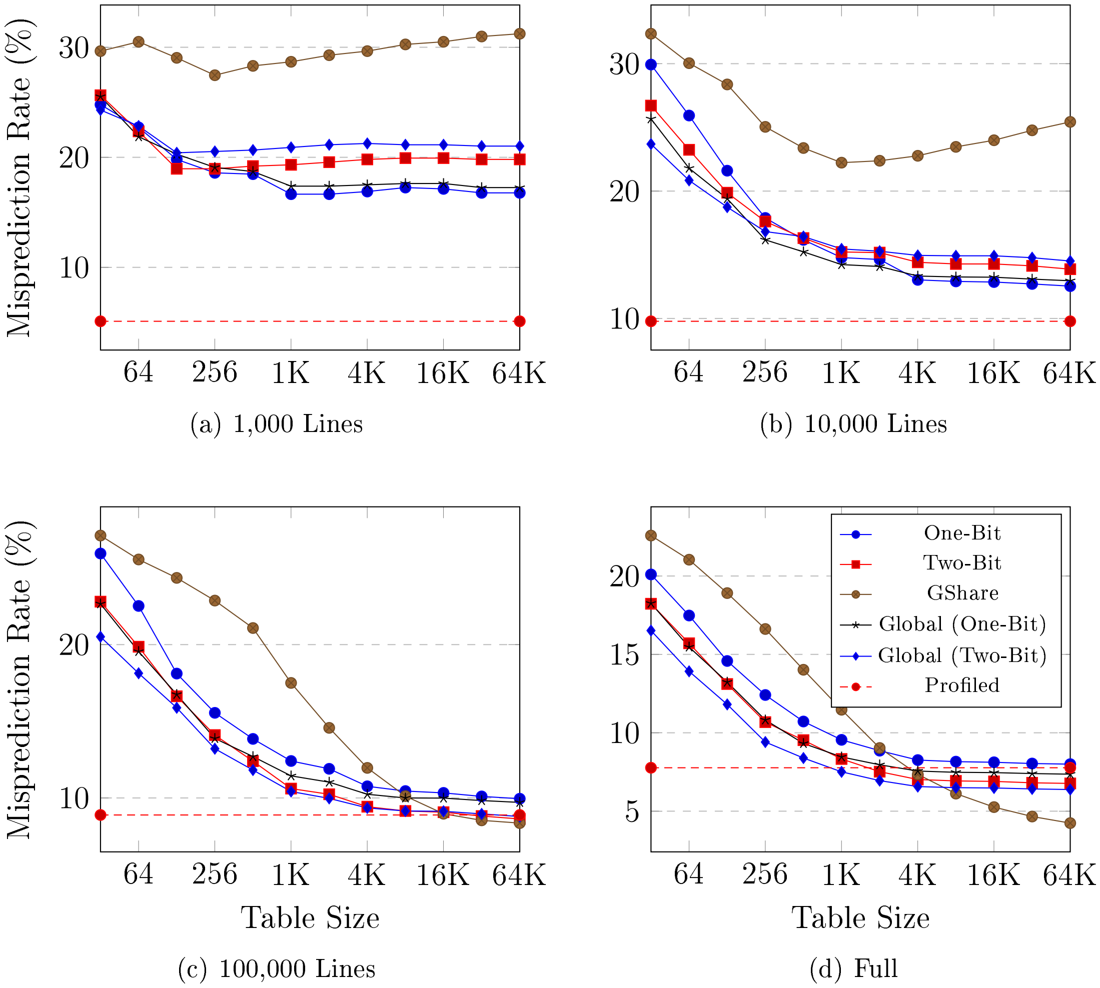
(a) Cactus BSSN

(b) Leela

Figure 6: Predictor performance for the first 10,000 lines of Cactus BSSN and Leela

Running the snippets of Cactus BSSN shows a range of behaviour in predictor performance. For 1,000 lines (Figure 7(a)), the standard bit predictors reach a relatively stable performance from a table size of 128. This stability is likely due to the tables reaching a sufficient size for the short length of the program. Also of note is that the one-bit predictors noticeably outperform the two-bit predictors from a table size of 1,024, which may be the result of a pathological case resulting in increased misprediction for the two-bit predictors.

The one-bit predictors continue to outperform the two-bit predictors for 10,000 lines (Figure 7(b)), but perform worse for 100,000 lines (Figure 7(c)). The performance curve of standard bit predictors roughly reaches its final shape by 10,000 lines, while GShare's curve only matches its final shape roughly near 100,000 lines. Together with GShare's extremely poor performance for 1,000 and 10,000 lines, this fluctuation in shape suggests GShare requires a long program and large table size to outperform other predictors.



(a) 1,000 Lines

(b) 10,000 Lines

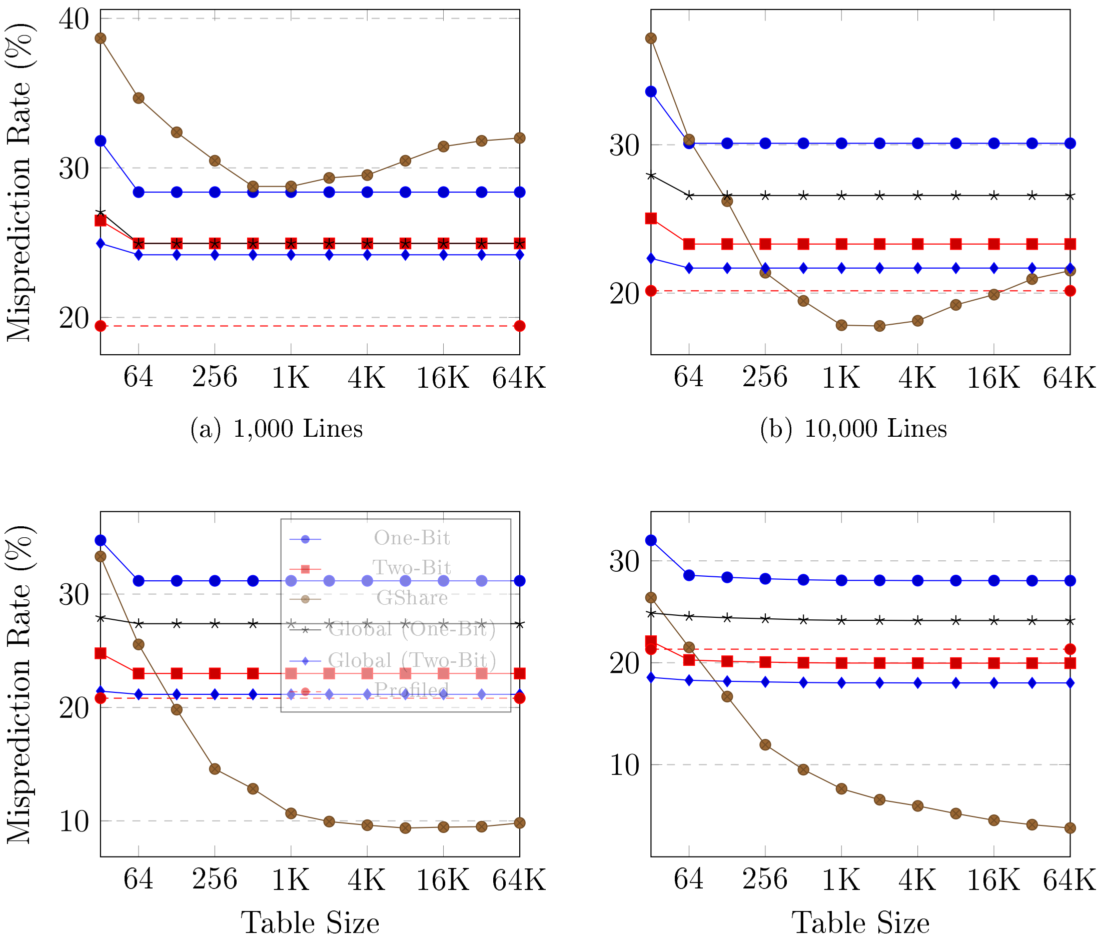
(c) 100,000 Lines

(d) Full

Figure 7: Cactus BSSN run in parts

The partial trace files for Leela have very similar performance profile to the full trace file (Figure 8). The standard predictors have constant performance from a table size of 64, with comparable performance to the full trace file. This backs up the previous findings that the structure of branches in Leela seems to change very little across the runtime of the full program.

GShare already has best results from 10,000 lines (Figure 8(b)), but only gains its final profile at around 100,000 lines (Figure 8(c)). Interestingly, GShare's performance for 10,000 lines is best with a table size of 2,048. This suggests that GShare needs a balance of table size suited to the specific program it is being run for to achieve the best performance. Too large a table might reduce GShare's performance as variation in its global history register could result in the same address being mapped to many different parts of the predictor table. In effect, this would reduce its ability to predict future branches from previous results, giving a higher misprediction rate.



(a) 1,000 Lines

(b) 10,000 Lines

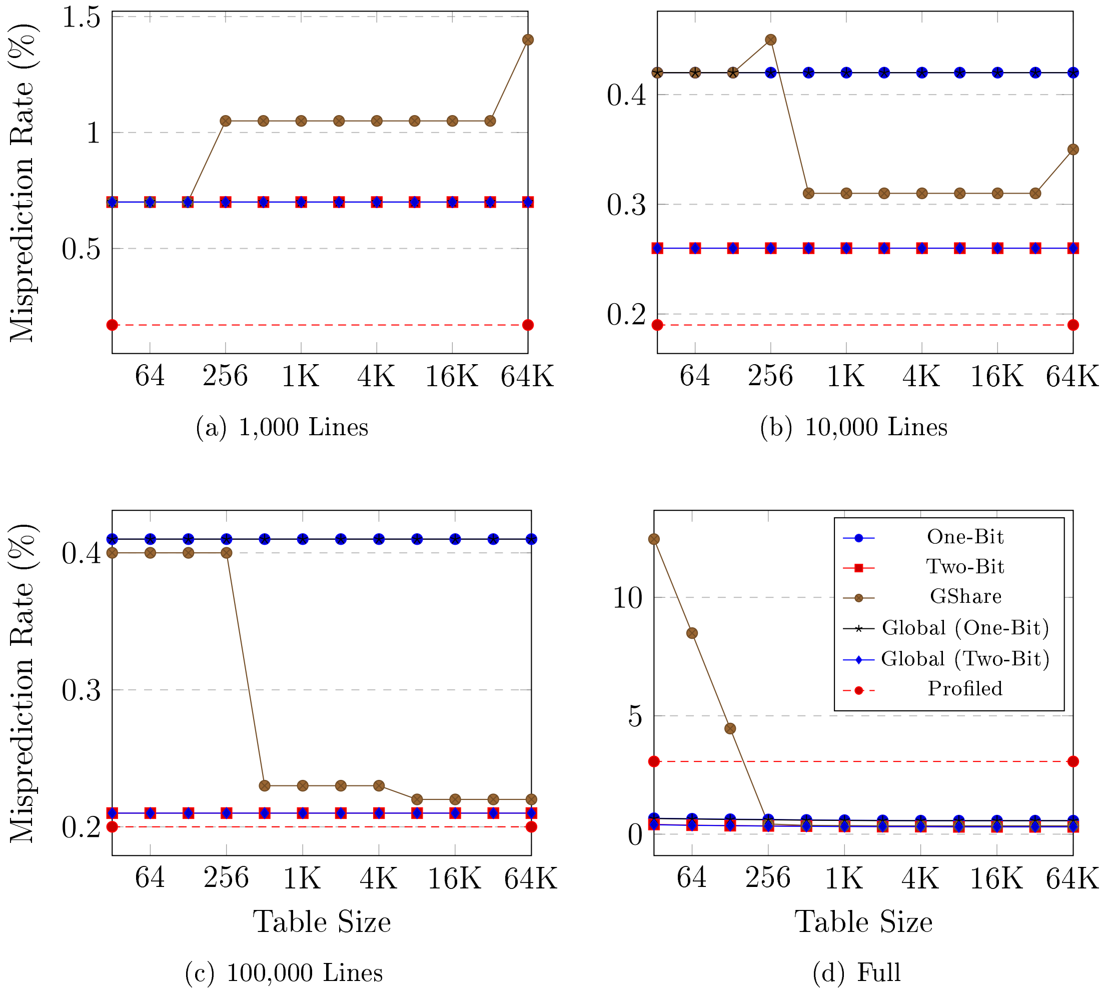
(c) 100,000 Lines

(d) Full

Figure 8: Leela run in parts

The partial trace files for BWaves present a range of different performance profiles (Figure 9). The standard bit predictors seem to have equal, constant performance for 1,000 lines, before diverging to the pattern seen in the full trace file.

The GShare predictor exhibits mixed performance across different trace lengths and table sizes. It follows a profile for 10,000 and 100,000 lines that is similar to the full trace file, but with a significantly less negative impact from a low table size. For 1,000 lines it follows the opposite trend, having the best performance at low table sizes before decreasing in performance for larger table sizes. These mixed results suggest that there may be instructions that present a pathological case for the GShare predictor, but that these instructions do not occur evenly throughout the entire trace.



(a) 1,000 Lines

(b) 10,000 Lines

(c) 100,000 Lines

(d) Full

Figure 9: BWaves run in parts

4 Evaluation

Overall, it seems that the predictors follow a common trend, with one-bit predictors being outperformed by two-bit predictors and the global bit predictors outperforming those without global history. However, there are some exceptions to this in smaller trace files (Figure 7(a)).

GShare seems to be most performant with a high table size for larger program traces, but can occasionally outperform other predictors even at lower trace lengths with a suitable table size (Figure 8(b)).

Among the tableless predictors, the always taken predictor performed poorly both for whole and partial trace files (Figure 5(a)). The profiled predictor consistently showed the highest performance for partial trace files (Figure 9(a)) but had mixed performance for full trace files. In full trace files, it frequently placed between one- and two-bit predictors in terms of performance. For the full XZ trace file, it was only beaten by GShare with the largest table size tested (Figure 3(e)) It may be worth adapting the profiled approach to predict in small blocks of a whole trace file, rather than building its profile from the entire trace file.

5 Conclusion

The final implementation includes a range of branch predictors, including one-bit, two-bit, global one-bit, global two-bit, two-bit GShare, always taken and profiled predictors. Their performance was analysed, identifying common behaviour and investigating exceptions. It was challenging to verify predictor correctness and fully explain behaviour due to the lack of context on the trace files provided beyond file names. With more time, additional predictors could be implemented and tested, such as local and one-bit GShare predictors. Furthermore, expections to common behaviour could be investigated further with more research into each trace file.

References

[1] S. McFarling, *Combining branch predictors.* [Online]. Available: https://inst.eecs.berkeley.edu/~cs252/sp17/papers/McFarling-WRL-TN-36.pdf (visited on 27/03/2024).

[2] B. R. Childers, *Branch prediction.* [Online]. Available: https://people.cs.pitt.edu/~childers/CS2410/slides/lect-branch-prediction.pdf (visited on 27/03/2024).

[3] University of Edinburgh School of Informatics, *Assignment 1: Understanding branch prediction.* [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/car/Pracs/2017-18/Assignment1.pdf (visited on 25/03/2024).

[4] The Regents of the University of California, *The backing predictor (bpd).* [Online]. Available: https://docs.boom-core.org/en/latest/sections/branch-prediction/backing-predictor.html (visited on 27/03/2024).

[5] G.-C. Pascutto, *Leela.* [Online]. Available: https://www.sjeng.org/leela.html (visited on 06/04/2024).