G-Select Implementation and Analysis

Ibrahim Binmahfood
Department of Electrical & Computer Engineering
Portland State University
Portland, OR, USA
ibrah5@pdx.edu

Phil Nevins
Department of Electrical & Computer Engineering
Portland State University
Portland, OR, USA
pnevins@pdx.edu

Anjela Albaka
Department of Electrical & Computer Engineering
Portland State University
Portland, OR, USA
Aalbaka@pdx.edu

Kunjan Vyas
Department of Electrical & Computer Engineering
Portland State University
Portland, OR,USA
kunjan@pdx.edu

***Abstract* - This paper delves into the design, implementation, and evaluation of the Gselect branch predictor within the SimpleScalar CPU simulator framework. Gselect is a dynamic branch prediction technique that combines global history and program counter information to improve branch prediction accuracy. The predictor operates by indexing a table of 2-bit counters using a concatenation of bits from the Global History Register (GHR) and a portion of the Program Counter (PC). We implemented Gselect in the SimpleScalar toolset to explore its performance impact and validate its efficacy under varying configurations. The results demonstrate a significant improvement in prediction accuracy over traditional static and simpler dynamic predictors, emphasizing Gselect's potential for enhancing instruction-level parallelism in modern processors.**

I. INTRODUCTION

Branch prediction is an essential component of modern high-performance processors, enabling efficient handling of control dependencies in pipelined architectures. Incorrect predictions can result in significant performance penalties due to pipeline flushes and execution stalls, particularly in deeply pipelined or superscalar designs. Among the diverse strategies for branch prediction, dynamic schemes that leverage runtime behavior have emerged as the most effective. Gselect is one such dynamic predictor that combines global history and program counter information to generate highly accurate predictions.

In this study, we focus on the implementation and analysis of the Gselect predictor in the SimpleScalar CPU simulator, a widely used framework for architectural research. By leveraging bits from the GHR and PC to index a prediction table, Gselect captures the correlation between past branch behavior and the current program state. This paper provides an in-depth exploration of the predictor's design, its integration into SimpleScalar, and the resulting performance metrics across various workloads. The remainder of the paper is organized as follows: Section II covers the theoretical foundation of Gselect, Section III details the implementation methodology in SimpleScalar, Section IV presents experimental results, and Section V concludes with insights and potential future work.

II. RELATED WORK

Branch performance issues have been studied widely. Several authors have suggested ways of predicting the direction of conditional branches with hardware that uses the history of previous branches. The different proposed predictors take advantage of different observed patterns in branch behavior, for example McFarling’s paper presented a method of combining the advantages of these different types of predictors. This method uses a history mechanism to keep track of which predictor is most accurate for each branch so that the most accurate predictor can be used.

III. THE G-SELECT BRANCH PREDICTOR

Gselect is a type of branch predictor that combines global and local information to make predictions about the outcome of a branch. Specifically, it uses a portion of the Global History Register (GHR) and the Program Counter (PC) to index into a table of counters for prediction. Gselect belongs to the class of global branch predictors.

IV. METHOD OF IMPLEMENTATION

In the SimpleScalar simulator, we implemented the Gselect branch predictor as part of the branch prediction unit (bpred.c/bpred.h), utilizing several essential components and processes. The Global History Register (GHR) serves as a shift register that captures recent branch outcomes, where a "1" indicates a taken branch and a "0" represents a not-taken branch. This history is crucial for our predictions, and its length determines how many bits of global history we use. Additionally, we rely on a Prediction Table, which consists of an array of saturating counters, typically 2-bit, to store the prediction data. To calculate the index for accessing this table, we concatenate specific bits from the program counter *PC* with bits from the *GHR*. Once the index is determined, we retrieve the saturating counter value to make a prediction. If the counter value exceeds a predefined threshold, such as 2 for a 2-bit counter, we predict the branch as "taken." After the branch resolves, we update the saturating counter based on the actual outcome, incrementing it for a taken branch or decrementing it otherwise. Simultaneously, the *GHR* is shifted to incorporate the resolved branch outcome, ensuring that our predictor remains accurate and adaptive to changing program behavior.

```
if (pred_dir->config.gselect.history_bits
> 0) {
          index = ((baddr >> MD_BR_SHIFT)
& (pred_dir->config.gselect.size - 1))
                  ^
(pred_dir->config.gselect.history & ((1 <<
pred_dir->config.gselect.history_bits) -
1));
        } else {
          index = (baddr >> MD_BR_SHIFT) &
(pred_dir->config.gselect.size - 1);
        }
```

In our implementation, we attempted to match the initial conditions outlined in McFarling's paper to ensure consistency and accuracy in our simulations. Specifically, we simulated only 10 million instructions out of order for each benchmark, with all branch prediction counters warmed up by fast forwarding 10 million instructions. This is mostly the same behavior as setting the branch predictor counters initially to "taken". The benchmarks we used with the SimpleScalar simulator included a variety of tests from the SPEC2000 suite, some of which were also referenced in McFarling's work. These benchmarks included *gcc*, *li*, *tomcatv*, and *fpppp*. However, we encountered issues with *tomcatv*, which did not perform as expected during simulation, and *fpppp* was excluded due to the time it required to run. Consequently, these two benchmarks were omitted from our analysis.

V. RESULTS AND ANALYSIS

For each benchmark, we recorded the average accuracy of various branch prediction methods. For the *li* and *gcc* benchmarks, we observed that the accuracy for each prediction method exceeded 80%. Among these, the combined bimodal Gselect and Gshare predictors emerged as the most accurate for the *li* benchmark. A similar trend was evident for the *gcc* benchmark, where these combined predictors consistently performed the best.

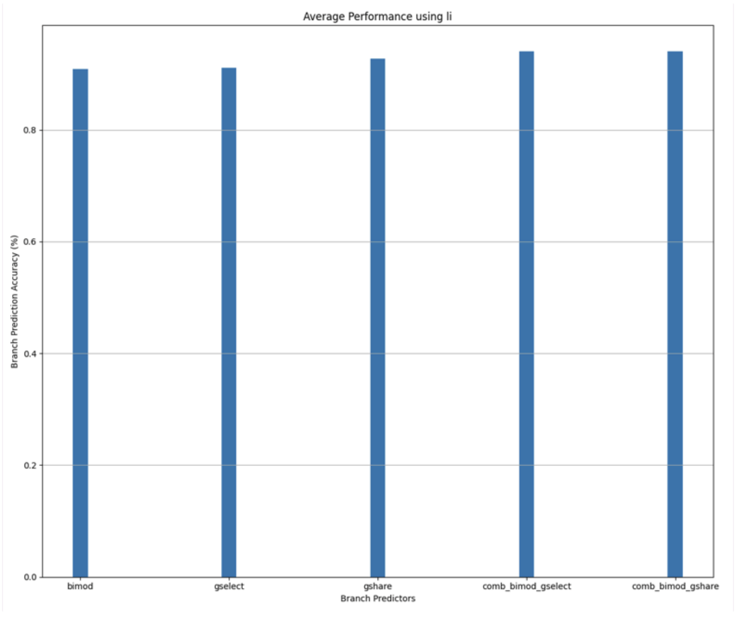

Figure 1: Performance using li benchmark

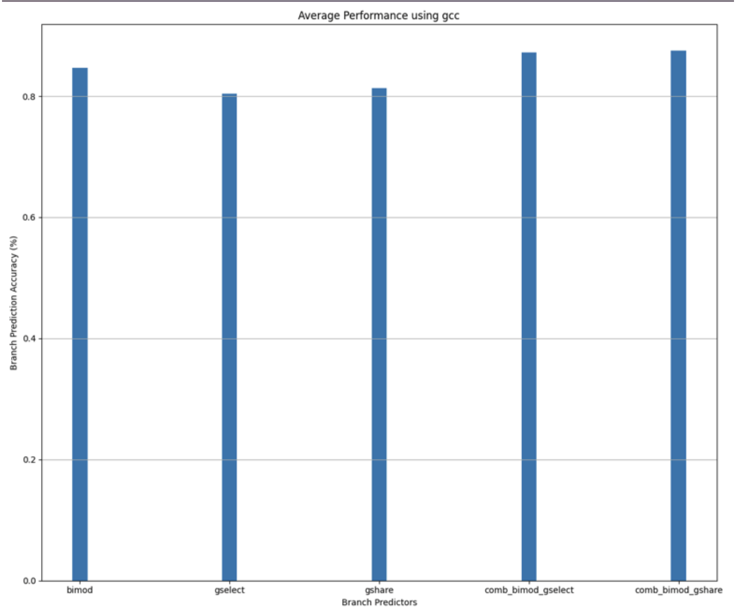

Figure 2: Performance using gcc

We further analyzed the relationship between predictor size and average prediction accuracy across all benchmarks. The results showed that while Gshare slightly outperformed Gselect, the difference in performance was minimal. However, Gshare has a larger hardware footprint, making Gselect a viable alternative when footprint is a concern. Interestingly, the trends for combined bimodal Gshare and combined bimodal Gselect predictors were nearly identical across different predictor sizes.

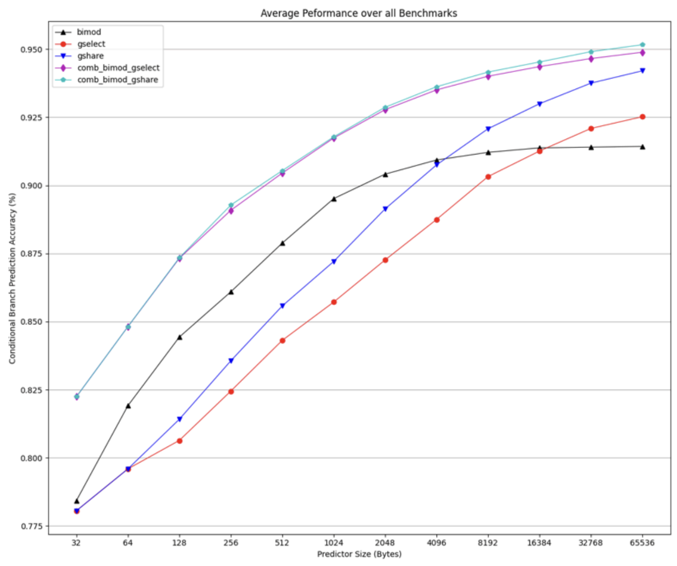

Figure 3: Performance over all benchmarks vs Predictor Size

This indicates that combining Gselect with bimodal prediction can achieve comparable accuracy to Gshare while enabling simpler hardware implementation.

Additionally, we plotted IPC (instructions per cycle) against predictor size. Here again, the combined bimodal Gselect predictor performed nearly as well as the combined bimodal Gshare predictor.

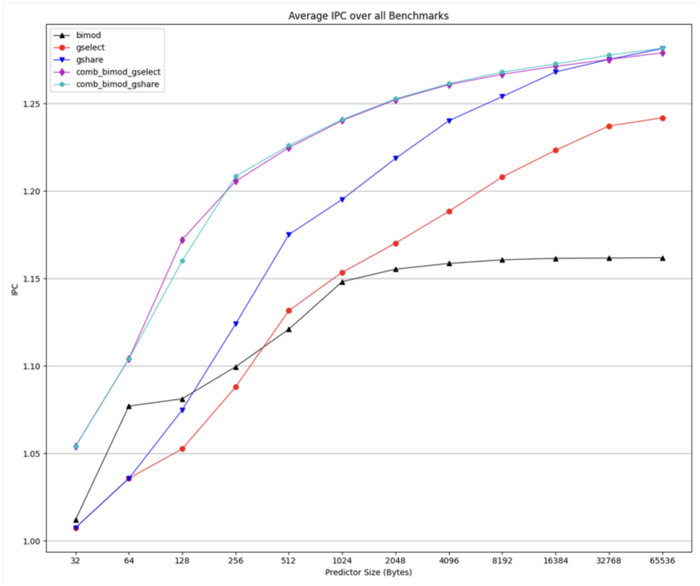

Figure 4: IPC over all benchmarks vs Predictor Size

Given that our simulation setup matched the granularity of McFarling's paper, the identical trends observed in our results validate the correctness of our implementation.

VI. CONCLUSION

In conclusion, our project successfully implemented and evaluated various branch prediction methods within the SimpleScalar simulator, closely adhering to the initial conditions outlined in McFarling's paper. Through rigorous testing on selected benchmarks using our automation script, we demonstrated that *gselect* is an inexpensive alternative without much performance degradation. Additionally, combined predictors, such as bimodal Gselect and bimodal Gshare, deliver similar accuracy, with performance exceeding 80% for key benchmarks like *li* and *gcc*. Our analysis revealed that Gselect offers a competitive alternative to Gshare, particularly when hardware footprint is a constraint, without significantly compromising accuracy. By validating our implementation against established trends, we ensured the reliability of our results, providing valuable insights into the trade-offs between accuracy and complexity in branch prediction strategies.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to our professor, Yuchen, and our TA, Sean, for their invaluable guidance and support throughout this project. Their expertise and feedback were critical in helping us navigate challenges and achieve our objectives. We also extend our thanks to both T. M. Austin and S. McFarling for their work, "A User's and Hacker's Guide to the SimpleScalar Architectural Research Tool Set" and "Combining Branch Predictors," which served as essential references and inspirations for our exploration of branch prediction strategies.

REFERENCES

[1] S. McFarling, "Combining branch predictors," in Proc. 6th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI), 1993, pp. 213-223. doi: 10.1145/166321.166343

[2] T. M. Austin, A User's and Hacker's Guide to the SimpleScalar Architectural Research Tool Set, Intel MicroComputer Research Labs, Jan. 1997.