Further re-examining complex branch predictors

Aaron Fox
Department of Computer Science and Engineering
University of Louisville
Louisville, KY, 40203
aaron.fox@louisville.edu

***Abstract*—As microarchitectures become increasingly optimized over time, the general trend of significant increases in clock times, deeper pipelines, and throughput rates have led designers of new microarchitectures to use the increased capabilities in part to design increasingly complex and more accurate branch predictor systems. However, as discovered in previous surveys of branch predictor systems, this increase in complexity and focus on improving accuracy does not always lead to better processor performance. Past microarchitectures have not emphasized the scalability of the IPC of processors with respect to the increased hardware size of branch predictor systems. This has resulted in the decreasing of IPC using past branch predictor systems as the size of the hardware used to implement them increases. Several alternatives have been proposed to help overcome this issue over the past decades that aim to simplify predictor systems to help improve performance. The aim of this project is to continue this exploration of the branch predictor's impact on performance and its relationship with prediction accuracy. Commonly used branch predictor systems in the past are used as a baseline for analyzing different branch predictors systems' effects on processor performance and branch prediction accuracy, and more recent innovations in predictor systems are also compared to help evaluate the impact of complexity on modern predictor systems as well. Through simulations, this experiment serves to support the assertion that simpler, pipelined branch predictor systems generally perform better than more complex and more accurate systems as hardware budgets increase, proving the need for scalability and measuring performance of branch predictors over the slightly less useful metric of misprediction rate, and this paper seeks to assert this for modern predictor systems as well.**

Keywords—branch predictors, perceptron predictor, dynamic branch prediction, predictor accuracy, pipelining, hybrid branch predictor.

I. Introduction: A Survey of Past Branch Predictors and a Comparison to Modern Prediction Systems

Branch Predictors are a vital component in the operations of modern processors. As the accuracy of branch predictors has strong correlations with overall processor performance, an efficient and accurate branch predictor is essential to a well-functioning modern processor. A good branch predictor can also decrease the required energy output of a processor by avoiding unnecessary computations involved with incorrect predictions and computing branches that need not be processed. Although the practice of improving a branch predictor's accuracy to reduce energy output and improve performance appears straightforward, the added complexity required to obtain this increased accuracy often interferes with the intended goal of improved overall processor performance, as noted by Jiménez et al [1]. The goal of this paper is to survey the same branch predictors first used in Jiménez's paper roughly two decades ago to evaluate and compare this conclusion with modern implementations of branch prediction systems that are common in the literature today. In this paper, experiments are designed, implemented, and evaluated to provide an analysis and comparison with past branch predictors and modern prediction systems. Conclusions and analysis discussing the results are then followed.

II. Relevant Background of Branch Predictors

A. *Why Branch Predictors Impact Processor Performance*

As most software programs necessitate the use of branching instructions which determine the flow of a program, there is almost always uncertainty in the execution flow of instructions. This can lead to a steep loss in processor performance when instructions that aren't needed are executed or a needed instruction has not yet been executed by the time it is required. A not insignificant measure of accurate branch prediction is therefore needed to help alleviate this penalty, potentially relieving a processor from overly obstructive latencies and wasted computational energy [3].

B. *Challenges of Designing a Branch Predictor*

Unfortunately, there is not a direct linear relationship between accuracy and processor performance, as Jiménez et al. noted. In fact, there are many complex relationships to consider when implementing the design of a branch prediction system. Many tradeoffs must be considered when analyzing the hardware budget scalability, area required, energy use, misprediction rate, and prediction latency, to name a few factors of the multivariate equation that arises in the design of a branch predictor implementation. As the main conclusion of "Reconsidering complex branch predictors" illustrates, the tradeoff between added complexity to make a branch more accurate often may lead to an implementation to become less practical and, in fact, may worsen the performance versus a less accurate branch predictor. This necessitates a survey of both modern and past branch predictor systems to help analyze and evaluate some of the many variables and considerations inherent to the design of branch prediction systems.

C. *Survey of Past and Future Work*

As emphasized by Jiménez et al., the increase in sophisticated techniques and emphasis on improving precision does not always lead to better processor performance. The ability for past microarchitectures to scale up the instructions per cycle (IPC) of their processors in relation to larger hardware budgets for predictor systems has not always been emphasized by microarchitecture designers. Consequently, the processor performance has often decreased using these past branch predictor systems as the hardware budget of the predictors has increased in size. In "Reconsidering complex branch predictors," an implemented algorithm called gshare.fast was used as an improved alternative for designing more scalable branch predictors for larger hardware sizes. This design aimed to avoid overhead and simplify the predictor system so that it could be pipelined and produce good predictions in only one cycle. The aim of this project is to continue this exploration of the branch predictor's impact on performance and its relationship with prediction accuracy. Commonly used branch predictor systems in the past, such as the hybrid 2Bc+gskew predictor, the Bimodal predictor, and the aforementioned gshare.fast are used as a baseline for analyzing each branch predictors' effect on processor performance and branch prediction accuracy. Additional dynamic branch prediction techniques used more recently -such as geometric history length branch predictors, virtualized tagged branch predictors, and neural-based branch predictors- are compared and analyzed with the results of the improved gshare.fast algorithm with the expectation that over the seventeen years since the gshare.fast algorithm was published, there are potential new prediction systems that provide improvements to both processor performance and branch predictor accuracy in the literature.

III. A Short Summary of Systems Evaluated

A. *Bimodal Branch Predictor*

The bimodal predictor is one of the simplest branch prediction systems known in the literature. This system simply uses the address of a fetched branch prediction instruction as means of indexing the global history prediction bits [2]. This system normally indexes the lowest bits of the address which is then used to index into the table entry. The corresponding branch prediction bits that are output are then used for predicting the branch outcome, and, when the actual branch results of the software are learned, the table entry is replaced with the correct results. Because of the simple nature of the bimodal predictor, this is the cheapest and simplest hardware implementation for a branch predictor system of all the systems evaluated in this paper.

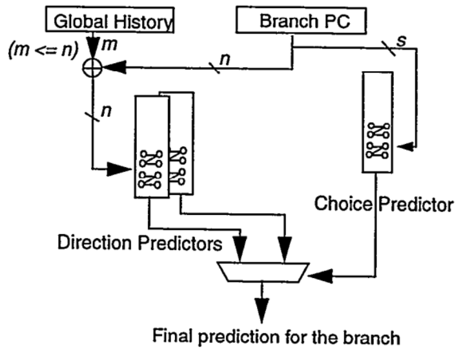


Figure 1. Bimodal branch predictor system [2]

B. *Gshare.fast*

In "Reconsidering Complex Branch Predictors," Jiménez et al. designed a new branch predictor system that improves upon the classic gshare algorithm which functioned like the aforementioned bimodal branch predictor but also recorded the global history (hence the naming of Gshare) of branches to aid in indexing the prediction bit table, helping to provide a more accurate prediction than the bimodal scheme alone. Gshare.fast improved on this gshare implementation by implementing pipelining of the branch predictor which, in effect, brought the latency of the branch predictor to just one cycle. This implementation used a four-stage pipeline where the first three stages read the pattern history table and the last stage made the prediction. Different bits, called "Branch Present" and "New History Bits," are pipelined through the stages to help record speculative global history. The pipeline system itself works to predict a fetched branch by storing eight two-bit counters in the first stage into an eight-entry pattern history table. The Branch Present and New History Bits can be shifted in from stage 2 if needed, and the first stage's respective bits are pipelined forward to the second stage as well. The same effective steps occur for the second and third stages until the fourth stage XORs the results of the shifted up pattern history table with the global history record to give the final prediction. This, in effect, allows the branch predictor system implemented in gshare.fast to deliver a prediction in a single pipelined cycle.

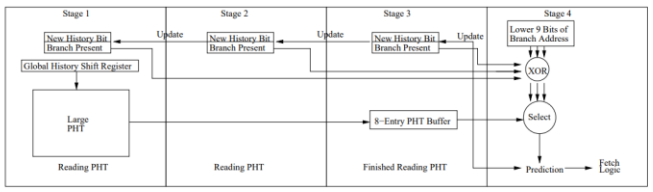


Figure 2. Pipeline system of *gshare.fast* [1]

C. *2bc+gskew Branch Predictor*

Because non-biased and biased branches are not uncommon in most written software, one branch predictor system alone often fails to provide sufficient accuracy, for a branch prediction system alone often cannot account for both types of branches. Hybrid branch predictor systems such as the 2bc+gskew predictor system allow for higher accuracy than the single branch predictor systems alone [4]. The 2bc+gskew predictor combines an *e-gskew* predictor (which, by "skewing" the branch predictor design, avoids aliasing, i.e. hashing to the same entries, in tables, similar to how cache misses are avoided by de-aliasing in caching systems) and a bimodal two-bit count predictor as previously discussed into four predictor-table banks along with a meta-predictor bank, which uses the current branch address and global history records to help determine the final prediction. The e-gskew predictor banks are updated whenever a misprediction occurs. All the respective banks then "vote" together to help determine the final prediction. Since oftentimes the global history record is not required, only the simpler bimodal predictor is used, thus improving performance time and raising accuracy. While accuracy is improved, this hybrid model is noticeably far more complex than the simpler bimodal predictor first discussed.

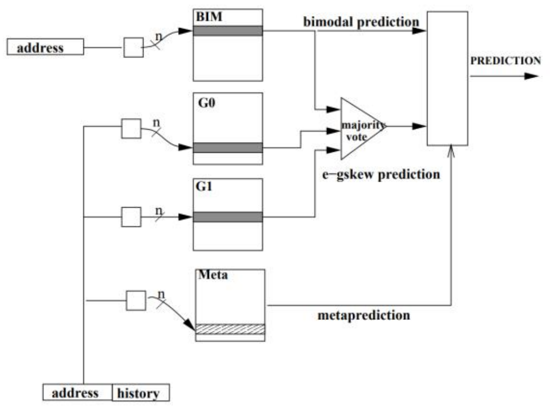


Figure 3. The 2bc+gskew predictor system [4]

D. *Geometric History Length Branch Predictor*

The optimized version of the geometric history length predictor allows for keeping track of very long global history records (up to 200 bits) so that the predictions can make use of more past data in determining predictions, leading to a higher overall accuracy [5]. The optimized geometric history length predictor is able to still maintain efficiency in doing this by maintaining many short history tables, which is achieved by nature of the geometric series that is used to determine the length of each of the tables. The geometric history length branch predictor is able to take advantage of using many of these shorter tables for more recent branch correlations, allowing for very accurate predictions. Using a geometric series to determine the length of the branch predictor tables and summing all of the tables together to form a prediction proves to be a sophisticated process. Therefore, while this predictor system is prominent in its high accuracy and can be pipelined to help mitigate the inherent delays, the latency involved with the complexity and the managing of so much memory can become obstructive.

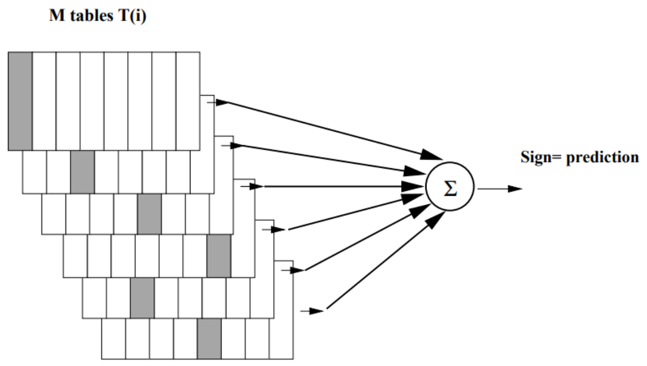
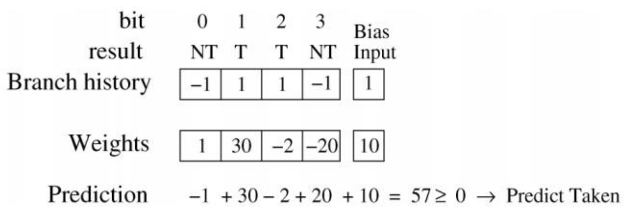


Figure 4. The Geometric History Length Branch Predictor [5]

E. *Neural-Based Dynamic Branch Predictors*

Neural-based dynamic branch predictors, such as the perceptron predictor, take advantage of simple neural methods such as the perceptron, which is simply a single-layer neural network. Because of the nature of neural networks, the history length of the tables can scale linearly with the hardware resources, an important factor for modern predictor systems as their respective hardware budgets continue to increase over time [6]. The weights vector inside the perceptron in this predictor system represent the correlation of the different branches. The inputs to the network range from -1 to 1, depending on whether the respective input branches are taken. After summing and multiplying the weights by the branch inputs and adjusting the input values depending on whether or not the predictions are correct, the output of the neural classifier will serve as the prediction of the next branch, whose output is later fed back into the input of the network again. An example of the inputs and outputs of the perceptron branch predictor can be seen in Figure 5. In this figure, the perceptron has determined the respective weighted vector for each input bit to the system. Then, based on the previous two-bit branch results and the bias results, the inputs are summed and multiplied together to provide an output. If that resulting output is greater than zero, then the branch is decided to be taken. The perceptron predictors require tuning to ensure the high accuracy that it can achieve and can vary substantial amounts in complexity depending on the deepness of the neural network layers. Despite the added complexity of the neural network in this system, there are still noticeable benefits of the IPC performance over more classical techniques such as the bimodal predictor. Additionally, the ability for the perceptron predictor to scale linearly in table length history size with its hardware budget proves very useful for scaling branch predictor systems up as microprocessors become increasingly larger.



bit	0	1	2	3	Bias
result	NT	T	T	NT	Input
Branch history	−1	1	1	−1	1
Weights	1	30	−2	−20	10

Prediction $-1 + 30 - 2 + 20 + 10 = 57 \geq 0 \rightarrow$ Predict Taken

Figure 5. Sample Perceptron Branch Predictor Output [6]

F. *Virtualized L-TAGE Branch Predictor*

A more recent innovation in branch predictors concerns the use of virtualization in branch predictor systems. The application of virtualization to branch predictor systems attempts to take advantage of the effective increased capacity of predictors in a more spread-out, sleeker design. By tabulating all of the branch predictor metadata into a virtual table in a lower cache and using similar concepts of cache locality to store more necessary branch data on on-chip caches, faster data retrieval for branch information is able to be achieved while still maintaining large capacities for storing branch metadata. This virtualization was applied to the state-of-the-art L-TAGE predictor so that it could use a paging scheme to simulate how cache localities work with reference to spatial and temporal localities. By extending the common concepts of locality that is well-known in the realm of the caching literature to branch predictors, the developers of this virtualized L-TAGE predictor are able to efficiently provide necessary branch data with lower latencies for most needed calls, simulating how current methods of retrieving more recently data quicker from on-chip caches versus having a larger latency in having to retrieve data from a slower L2 cache that was less likely to be needed. This clever replication of a well-known concept in the realm of computer science has been shown to improve accuracy while requiring less overall storage. To be sure, this system is noticeably more complex than the simpler models mentioned previously, requiring non-insignificant overhead and lag in the process of replicating the concepts of cache locality.

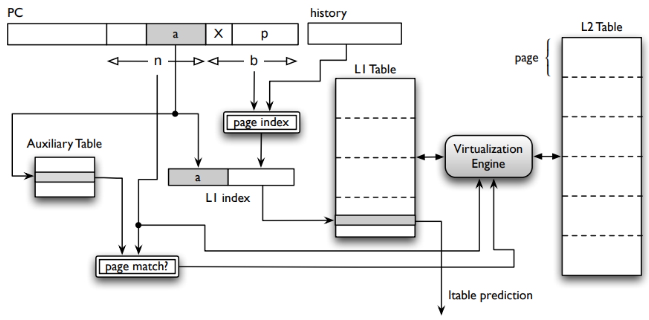


Figure 6. The Paging Scheme Architecture of the Virtualized L-TAGE Branch Predictor [7]

IV. Experimental Protocol

A. *Simulation Framework*

To aid in comparing each of the branch predictor systems surveyed in Section III, simulations were run to compare, evaluate, and analyze the trade-offs of running each system. The Journal of Instruction-Level Parallelism hosts a Championship Branch Prediction competition every year, where the goal of the competition is to use a fixed storage budget to implement the best branch prediction algorithms on a common evaluation framework that the committee distributes. The committee therefore provides the simulation functional infrastructure that was used to evaluate the submitted branch prediction algorithms they used to determine the winners of the competition. These functions simulate 21 large trace workloads consisting of over 30 million instructions each. There are seven different benchmarks used in the traces available, such as SPECint and SPECfp. Each branch prediction simulator was then simulated on an Intel Core i7-6600U CPU @ 2.60 GHz, 2808 MHz with 2 Cores and 4 Logical Processor using the simulation infrastructure developed by the Championship Branch Prediction competition in order to evaluate their results experimentally.

B. *Method of Testing*

A decision was then made on which of two versions of SimFlex, one of the most popular computer architecture simulation softwares publicly available: Flexus or SMARTS. Since the Flexus software allows full-system simulation using component interface models that help focus on compiler-time optimization and integration, Flexus was chosen over the more statistical-sampling-oriented SMARTS, which focuses on only a smaller portion of the microarchitecture overall. The Flexus software was then used to gather the data run on the Championship Branch Prediction's simulation infrastructure. The benchmarks of SPECinf and SPECfp were used to help gauge the performance of each prediction algorithm as they were readily available from the Championship Branch Prediction's infrastructure. The misprediction rate is measured for each algorithm as one gauge, and the instructions per cycle of each algorithm is also considered as the more important factor to judge each branch predictor per Jiménez's original paper and thus is the most vital metric used in this study to help judge which prediction algorithm is best at scaling performance with the size of the hardware budget. Each branch prediction system, that is the bimodal, gshare.fast, 2bc+gskew, geometric history length, neural-based dynamic, and the virtualized L-TAGE branch predictors, were implemented in C++ according to their respective paper's specifications reformatted to the required framework input traces and output results of the Championship Branch Prediction infrastructure. Some of the code implementations that were readily available in open source implementations, such as the bimodal and 2bc+gskew branch predictor systems, were cloned and adapted to fit the proper input and outputs results of the simulation infrastructure. Each branch predictor system was run on the SPECfp and SPECint benchmarks eight times each for the 16KB, 32KB, 64KB, 128KB, and 256KB budgets. The results were then averaged for each respective benchmark, hardware budget, and predictor system and then tabulated and visualized graphically in the Results section below.

C. *A Note on Optimistic Assumptions*

It must be noted that, generally, the optimal variables of each branch predictor function were used to provide the most efficient misprediction rates and IPCs as found by the original research papers for each predictor. For example, the history length of 300 was chosen for the geometric history length branch predictor, for the original paper referenced found that a maximum history length of 300 generally leads to a useful drop in misprediction rate relative to the other history table sizes without adding so much complexity so as to lower the performance relative to the IPC. Another example concerns the use of 9 tagged tables in the Virtualized L-TAGE branch predictor, for the original paper came to the conclusion that this number led to an optimal misprediction rate relative to other sizes of tagged tables and also yielded an optimal IPC compared to the other sizes. Additionally, for all systems that took advantage of records such as global histories used the optimal global history length after testing and averaging the results of various history lengths. For example, the 2bc+gskew hybrid predictor found that a global history length for the two benchmarks considered in this experiment had an optimal length at 10. The diminishing returns after 10 often led to a lower IPC at little to no benefit. The optimistic assumptions of each system were used as an attempt at a fair and flat rule in obtaining the best results from each system. It should also be noted that, depending on different benchmarks and other variables tested, the results of this experiment could be altered. The latencies for all predictor systems were also optimistically assumed using Hewlett-Packard's CACTI 7.0 analytical tool for modeling the caches and memories of each predictor system in an attempt to make the starting ground fair for each system.

V. Results

A. *Graphical Summary*

After running each branch predictor system eight times over for each benchmark and hardware budget for each system and then averaging the results, the results were tabulated. The results are summarized with the metric of accuracy via the use of misprediction rate in Figure 7.

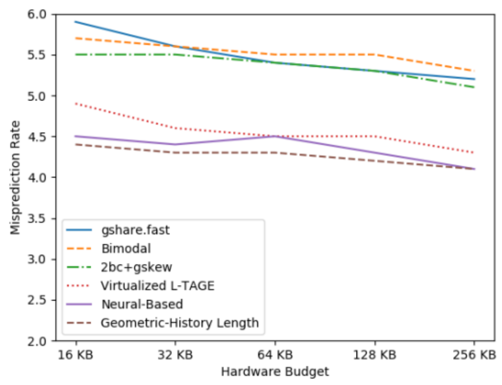


Figure 7. Average misprediction rates of each predictor system

The results summarized by the respective benchmarks for each prediction rate at a budget of 64 KB are summarized in the graph found in Figure 8.

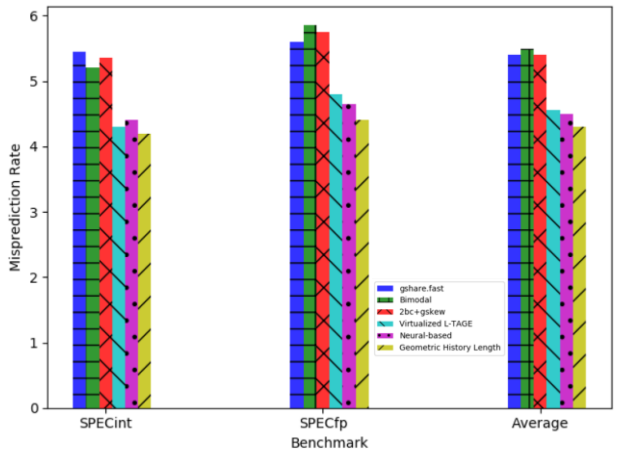


Figure 8. Average misprediction rates with a 64 KB budget for both benchmarks

The realistic average IPC of each prediction method for 1-cycle prediction are shown in Figure 9.

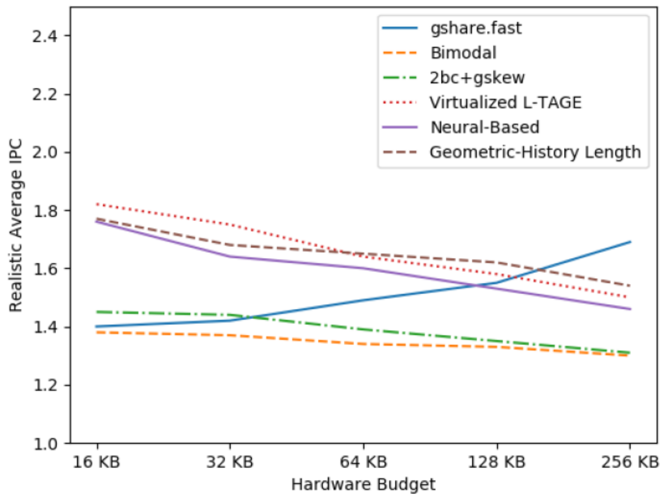


Figure 9. Realistic average IPC of each predictor system for 1-cycle prediction.

The last results figure, Figure 10, illustrates the average performance measured by IPC for each predictor system with a hardware budget of 64 KB.

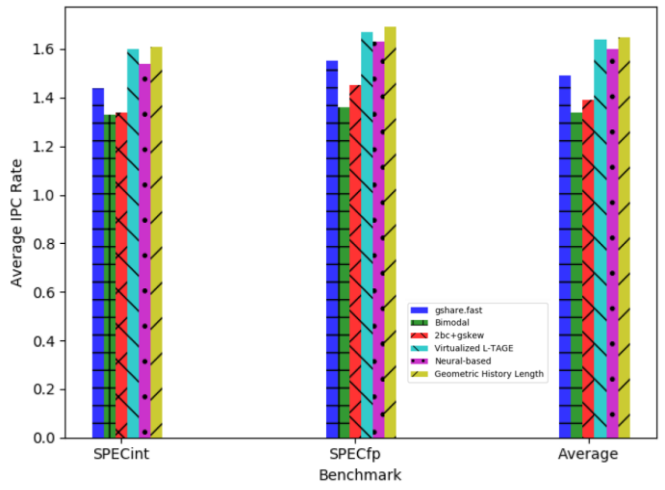


Figure 10. Realistic average IPC of each predictor system for 1-cycle prediction at a hardware budget of 64 KB.

VI. Analysis and Discussion of Results

A. *Analysis of Misprediction Rates*

In Figure 7, the disparity found in the misprediction rate between the three more recent branch predictor systems that have since been published after the penning of "Reconsidering complex branch predictors" versus the three classic prediction algorithms that existed prior to and during the penning of the paper is expected and consistent with the current literature. That is to say, the virtualized L-TAGE, neural-based, and geometric history length predictors all noticeably outperform the simpler and more classic algorithms of bimodal, 2bc+gskew, and the gshare.fast branch predictor systems. The more sophisticated tactics of the more recent implementations, which range from a basic layer of a neural network to a virtualized cache-like system of using predictor capacity to the use of a geometric series to exploit long global history records, allow for more innovative and precise measures to ensure a better overall accuracy using the metric of misprediction rate.

As more hardware capability becomes available, it becomes increasingly obvious that the misprediction rate improves as the global history records become longer and longer, allowing for more data to be used to base the prediction off of. This was the case for the 2bc+gskew and geometric history length predictors, for their misprediction rate certainly decreased as a result of having a larger history record to tabulate and form better informed predictions based off of larger history records. The neural-based perceptron predictor likewise was able to take advantage of this, and it also could have implemented a larger neural network with more features, allowing for more accuracy from having more variables to record data and influence the final better informed prediction. The general downward trend in the misprediction rate therefore occurs as expected as the hardware budget increases from 16 to 256 kilobytes.

Figure 8 illustrates how, for certain hardware budgets, different branch predictors may become more and more appropriate. This shows the need for planning out and testing how different predictor systems perform for different hardware budget sizes. This makes it clear that, when microarchitects are deciding on which branch predictor system to implement in their architecture, they should first consider which specific hardware budget they are using. The snapshot of the different predictor systems at a hardware budget of 64 kilobytes shows how a certain predictor system may be more appropriate at the budget of 64 KB, such as the gshare.fast system being able to perform with better accuracy at that budget than the bimodal branch predictor was able to perform. If Figure 8 was instead a snapshot of the benchmark performance at a hardware budget of 16 KB, a different story would reveal itself: the simpler bimodal predictor system happens to perform better at that specific budget over gshare.fast. This case vividly shows how important it is for microarchitects to therefore consider the desired hardware budget system before deciding on which predictor system to use, for different predictor systems perform better at different sizes per these results.

B. *Analysis of IPC*

In respect of Jiménez et al.'s research in "Reconsidering Branch Predictors," the most important measure of this experiment concerns the actual realistic performance of the predictor systems as the hardware budget scales up using the metric of cycles per instruction. The CPI at each hardware budget is deemed more important than the accuracy as measured by the misprediction rate because, in general, the actual realistic performance of the processor is far more important than the arbitrary accuracy of a branch predictor system. This is true to the phrase of not missing the forest for the trees: microarchitects shouldn't focus on simply improving one aspect of the architecture as much as possible; they should strive to adjust the components so that they work together with the entire system such that the whole system has an optimal and better performance with each component, not just a better single component.

By the vital metric of this experiment, therefore, the realistic performance of the gshare.fast algorithm as it improves as the hardware budget scales up proves to be the main point of this experiment: as Figure 9 shows, the gshare.fast branch predictor appears to far outperform all the other predictors as the budget increases. In fact, it is the only predictor system to noticeably increase as the budget increases. This is in contrast to the other predictor implementations which have a general negative trend in performance as the hardware budget increases.

The trend of the predictor systems as seen in figure 9 is due, in large part, to the inherent complexity of the predictor systems. From neural networks to geometric series to the concepts of cache locality, there are many complexities and intricacies involved with the sophisticated, yet very clever, means of lowering the misprediction rates. Because these techniques grow more complex over time with larger data sets from larger global history records, more latencies and complexities are incurred as the size of the predictor hardware budget grows. The gshare.fast implementation, however, maintains its steady growth as the hardware budget grows as it is specifically designed to pipeline its predictions in a single cycle, as Jiménez et al. discuss in their paper.

Figure 10, however, does provide an important caveat. When microarchitects are desigining the architecture of the processor, they must strongly consider which hardware budget size they are going to use. If they are using a size of around 64 kilobytes, for example, it appears that the more complex geometric history length predictor actually outperforms the gshare.fast implementation. That is to say, if a microarchitect had to design at this budget, the extra accuracy in predictions can be considered worthwhile at this smaller budget. However, if the designers of the processor wished to be able to scale the hardware budget of their systems up over time as technology got better, they would likely be better off resorting to a more stable system that has proven to scale well with an increasing hardware budget, which is fast.gshare.

C. *Tradeoffs*

When designing a branch predictor system, considerations must be considered in the tradeoff between increased predictor accuracy and the overall performance of the processor. Branch predictor designers may want to choose a more accurate system that looks to be fantastic on paper with respect to its misprediction rate, but they must pause to consider the tradeoff of using these more complex implementations that allow for this increased accuracy. Especially if the designers are considering a larger predictor hardware budget, they would very likely be better off trading the better accuracy for a more simple and pipelined predictor system that could deliver its predictions faster, therefore leading to better performance for a larger hardware budget. However, if the designer is working with a smaller hardware budget, they may not have to worry about the trade-off of accuracy for performance and can afford to use a more complex predictor system such as a neural-based or a geometric history length predictor which can properly be used for improving performance of the overall processor system.

VII. Conclusions and Future Work

A. *Conclusion*

This research originally set out to question the validity of the assertion that simpler, pipelined branch predictors generally always perform better at larger hardware budgets over more complex, non-pipelined predictor systems, even for more modern predictor systems that have been placed into the literature since the original paper on this topic was realized. The research and data of this experiment affirm this statement. This paper therefore provides further proof and data that simpler, pipelined implementations of branch predictor systems are very often the optimal choice for larger branch predictor hardware budgets. This is an important conclusion for architect designers to observe as more transistors are able to be placed onto chips and larger, complex, and more scalable branch predictor systems will need to be proposed. It is therefore in the best interest of these designers to continue research into more pipelined implementations of the more complex branch prediction systems discussed so that they can focus on performance more than the red herring of prediction accuracy.

B. *Future Work*

Much more design and consideration into simpler pipelined versions of the more complex branch predictor systems tested in this research is needed. If these more complex systems can be simplified to use pipelining like the gshare.fast implementation, it is possible that both a higher accuracy and scalable performance at larger hardware budgets can be combined. This would lead to more formidable and more useful branch predictor systems, and this would ultimately lead to faster and better processors as the size of hardware implementations increase over time.

References

[1] D. A. Jiménez, "Reconsidering complex branch predictors*," The Ninth International Symposium on High-Performance Computer Architecture, 2003*. HPCA-9 2003. Proceedings., Anaheim, CA, USA, 2003 , pp. 43-52 ,doi: 10.1109/HPCA.2003.1183523.

[2] C. Lee, I. -. K. Chen and T. N. Mudge, "The bi-mode branch predictor," *Proceedings of 30th Annual International Symposium on Microarchitecture*, Research Triangle Park, NC, USA, 1997, pp. 4-13, doi: 10.1109/MICRO.1997.645792.

[3] G. H. Loh, "Revisiting the performance impact of branch predictor latencies," 2006 IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, USA, 2006, pp. 59-69, doi: 10.1109/ISPASS.2006.1620790.

[4] A. Seznec, P. Michaud, "De-aliased hybrid branch predictors," PhD Dissertation, CAPS, INRIA, 1999

[5] A. Seznec, "Analysis of the O-GEometric history length branch predictor," *32nd International Symposium on Computer Architecture (ISCA'05)*, Madison, WI, USA, 2005, pp. 394-405, doi: 10.1109/ISCA.2005.13.

[6] D. A. Jiménez and C. Lin. 2002, "*Neural methods for dynamic branch prediction*," ACM Trans. Comput. Syst. 20, 4, 2002, pp. 369–397, doi:https://doi.org/10.1145/571637.571639

[7] M. Sadooghi-Alvandi, K. Aasaraai and A. Moshovos, "Toward virtualizing branch direction prediction," *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2012, pp. 455-460, doi: 10.1109/DATE.2012.6176514.