Implementing Aggressive Branch Prediction in FabScalar

Rangeen Basu Roy Chowdhury, Daniel Howe
Electrical & Computer Engineering Department
North Carolina State University
rbasuro@ncsu.edu , dchowe@ncsu.edu

***Abstract*—When selecting a style of branch predictor to be implemented in FabScalar [1] as an improvement on the existing Bimodal predictor, it is necessary to consider many alternatives. The quality of these alternatives must then be judged based on the performance benefits they bring as well as the cost of gaining these performance benefits. This paper aims to study several variations on the Gshare style branch predictor to determine which would be the most feasible to implement in FabScalar. It then goes on to compare the new implementation against the old as well as considering possible future work. The branch predictor ultimately constructed is a Gshare style predictor that uses stale global history information to allow for accurate multiple branch predictions per cycle at a minimal hardware cost. Although the C++ simulation results were encouraging, the performance of the RTL was not as good.**

branch predictor; bi-modal; gshare; FabScalar

I. INTRODUCTION

Today's microprocessor designs try to squeeze out as much performance as possible while staying within power and area budgets. One conceptually easy way to make processors faster is to increase the frequency thus reducing the amount of time required to complete a provided task. Processor pipelines have to be made deeper and deeper to achieve this reduction in cycle time. However, increasing the pipeline depth has a negative impact on mis-prediction recovery as the number of cycles between predicting a branch and resolution (execution) of a branch increases. This leads to a reduced overall IPC and an increased energy consumption since more and more speculative work needs to be squashed as the pipeline depth increases. Having a predictor with higher accuracy offsets these effects by decreasing the number of mis-predictions and hence reduced recovery penalty.

Prior to this point, FabScalar HDL model had a Bimodal multi-branch predictor [2] that could predict 8 branches at a time. The performance in terms of IPC and energy consumption was not acceptable and owed to the inherent lack of accuracy of Bimodal prediction. History based branch predictors have similar hardware costs as Bimodal branch predictor but have significantly higher prediction accuracy [11]. However, their complexity increases when multiple predictions are required in the same cycle. Techniques that can efficiently reduce this complexity without an adverse effect on accuracy were explored in this study. Higher prediction accuracy will allow a higher IPC and a lower overall energy consumption. It will also allow high bandwidth fetch mechanisms such as Trace Cache [6] and Branch Address Caches [9], which rely on accurate branch prediction to form traces of non-contiguous basic blocks, to be much more effective.

In this project we investigate the accuracy of various aggressive branch predictor styles and their effect on IPC. The considered branch predictor designs are either direct implementations from previous work or variations of them. Based on the simulation results, an optimum design is chosen and implemented as a synthesizable HDL model and integrated into the existing FabScalar design. The design was synthesized and a one to one comparison was made with the Bimodal predictor to make sure that cycle time was not negatively impacted.

The following section describes in detail the different styles of history based branch predictor that were explored. Section II also describes our methodology and presents exploration results obtained through simulation. Section III explains our choice of branch predictor style implemented in FabScalar RTL and presents detailed micro-architecture and RTL simulation results. Section IV presents some discussions and suggests future work. Section V concludes the study.

A. Abbreviations

BHR – Branch History Table, PHT – Prediction History Table (Counter Table), BPU – Branch Prediction Unit, PC – Program Counter

II. DESIGN SPACE EXPLORATION

The main difficulty in designing a history based multi-branch predictor is making the indexing style consistent for each branch and for each instance of the same branch. Achieving this in a bimodal predictor is fairly simple as it always uses the PC of a branch to index into the PHT and hence uses the same index to predict different instances of the same branch. In case of a history based predictor, the history used to hash the PC of a branch may be inconsistent between instances of the same branch. This study examined a few probable hashing algorithm that try to make the indexing consistent.

A. *Styles of Predictor*

We explored the prediction potential of several styles and variations of branch predictors. As a baseline standard, a 64K bi-modal branch predictor is used. The different predictor styles explored are briefly described here.

1) Standard Gshare - GS

This is a standard Gshare style predictor capable of predicting multiple branches in a single fetch bundle based on a speculatively updated global history register and the PC of each branch instruction. This predictor XORs bits of the branch's PC with the global history register to read into a PHT of 2-bit saturating counters (Fig. 1).

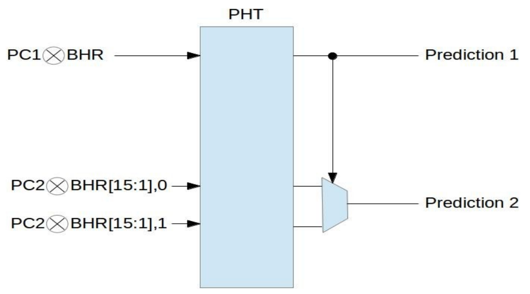


Fig. 1. Standard Gshare - GS

2) Branch Sequence Gshare - BSGS

This is similar to GS except it additionally uses the order of branches in the fetch bundle to index into a set of *N* PHTs. (i.e. the first branch in the fetch bundle references the first PHT, the second branch uses the second PHT, etc.) This style is functionally similar to the first however, each prediction does not reference the same PHT. This means that a branch at PC=X with global history=Y may get a different prediction depending on its location in the fetch bundle. This technique may be capable of gathering some local history to make more accurate predictions although this might require some decode information for the instructions at the beginning of the lookup cycle. This multiple PHTs are full sized PHTs (i.e. 64 K) and so would require *N* times the area. This study was more of a control experiment than a practical implementation. This is represented in Fig. 2.

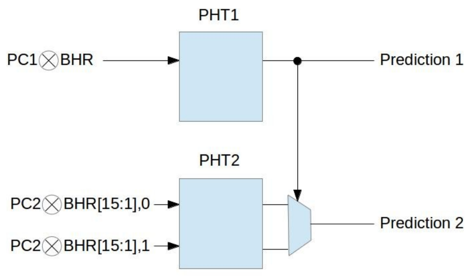


Fig. 2. Branch Sequence Gshare - BSGS

3) First Branch Gshare - FBGS

This is a Gshare style predictor that predicts multiple branches based only on the global history register and the PC of the first branch in the fetch bundle. The global history is updated speculatively after all predictions for the current bundle are made. This predictor has a single PHT that contains four 2-bit counters per entry. This predictor is modeled to account for potential difficulties in implementing other Gshare style predictors since it may not be possible to know the PC of later branches in the fetch bundle immediately. This indexing style is still consistent as it uses the PC of the first branch as opposed to PC of the first instruction in the bundle and hence it is not affected by fetch block misalignment i.e. fetch bundles spanning across multiple cache lines. See Fig. 3 for the diagram .

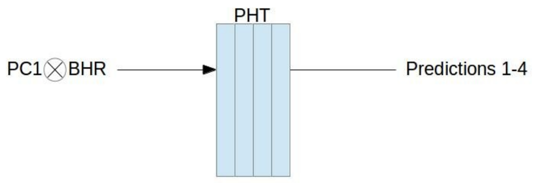


Fig. 3. First Branch Gshare - FBGS

4) Global History Two Level Predictor - GAg

On the opposite spectrum, a global history based predictor that relies only on the global history register to index into the PHT [10] for making multiple predictions was also examined. When considering this predictor style it seems likely to be relegated to another baseline reference, where the standard bi-modal predictor relies entirely on the addresses of branches this style relies solely on global history (Fig. 4). Never the less, this style was simulated to ensure that a fall-back implementation, relying on this style, would still be a viable improvement over the existing bi-modal implementation. This style is also one of the most likely ones for use with a trace cache as the PCs of the instructions in a trace are unknown due to them being in non contiguous basic blocks.

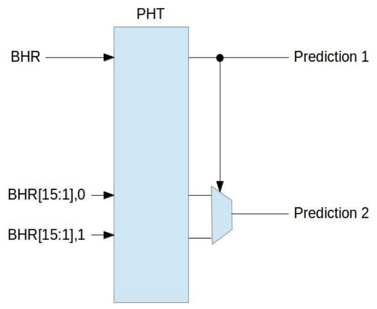


Fig. 4. Global History Two Level Predictor - GAg

5) PC Set Gshare - PSGS

As an option which may improve accuracy by extracting more locality information from the PC of branches, a Gshare style predictor that also uses lower order PC bits to further index into a set of PHTs is simulated. This style allows for simple and also beneficial banking of a single PHT, though in simulation the set of PHTs was not modeled as a single banked PHT but as full sized individual PHTs which are indexed by lower order PC bits and sub-indexed by XORing the global history with the branch address to read the 2-bit counters. See Fig. 5 for a clearer diagram.

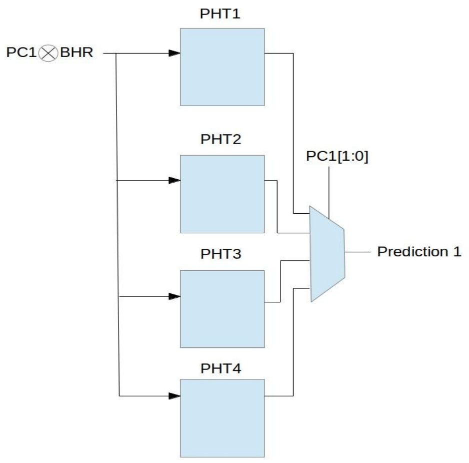


Fig. 5. PC Set Gshare - PSGS

6) Stale History Gshare - SHGS

To account for the possibility of not being able to make predictions and speculatively update the global history within a single cycle, a Gshare style predictor which uses slightly stale global history is considered. While making prediction for a branch, this style always ignores the three most recent predictions prior to the branch and uses rest of the BHR to hash the PC. Though not as aggressive as possible, this model still gives a good impression of the effects of limiting the global history when making branch predictions. This style is referenced as SHP, Stale History Predictor (Fig. 6).

B. *Methodology*

The simulator used for this project was a modification of a C++ based pipeline simulator that using the SimpleScalar ISA. This is a nearly cycle accurate, execution driven simulator; no frequency is considered, and the simulator operates on an actual dynamic instruction stream as opposed to a static trace previously generated.

Five SPEC benchmarks namely gcc, gzip, parser, twolf, and vortex were used for the different analysis runs. For each benchmark, each style of predictor was simulated under several conditions, however some of the properties were held constant across all simulations. Benchmarks were limited to 100,000,000

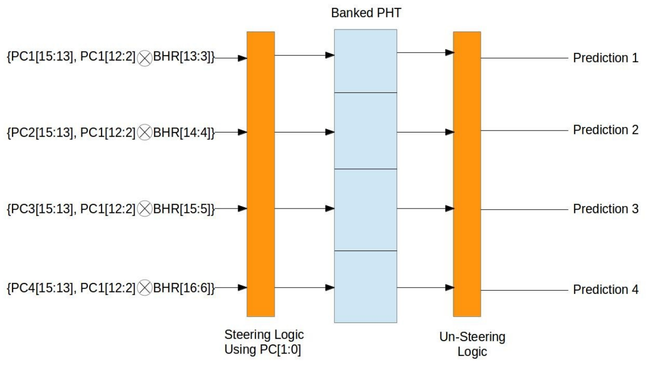

Banked PHT
{PC1[15:13], PC1[12:2]⊗BHR[13:3]}
{PC2[15:13], PC1[12:2]⊗BHR[14:4]}
{PC3[15:13], PC1[12:2]⊗BHR[15:5]}
{PC4[15:13], PC1[12:2]⊗BHR[16:6]}
Prediction 1
Prediction 2
Prediction 3
Prediction 4
Steering Logic Using PC[1:0]
Un-Steering Logic

Fig. 6. Stale History Gshare - SHGS

instructions after skipping the first 1,000,000,000 instructions to ignore any initialization phase the benchmarks may go through.

To begin with, the benchmarks were simulated under heavily idealized conditions: ideal instruction and data caches, very large sized queue structures, oracle memory disambiguation, and large widths (other than fetch and issue). Under these conditions the fetch (and issue) width was varied from 4 to 32 to ensure predictor accuracy as fetch width, and therefore a higher likelihood of multiple branches in a single fetch bundle, increase. Issue width was not held at a large value since instructions are not issued until there are at least 'Issue width' instructions queued, and for small fetch widths this would take multiple cycles and degrade performance unnecessarily. The results from these simulations allow the unadulterated effects of the branch predictors to be measured since the only factor being modified is the type of branch predictor.

Finally, to compare the predictor styles in the realistic settings similar to FabScalar, results were also gathered for simulations where none of the idealizations mentioned above were used. This portion of the study also limited the fetch width to four instructions.

In each of these simulations the accuracy of the predictor was measured as percentage of conditional branches mis-predicted out of the total number of conditional branches executed. This metric is used to compare the quality of the predictors since this value is fairly impervious to variations in the processor architecture and idealizing assumptions; it depends more on the properties of the workload. Also, as a more tangible metric of comparison, the IPC for each simulation is also recorded. It stands to reason that a predictor with an overall and generally higher accuracy would correspond to a higher IPC, though admittedly, seeing this correlation via measurement is reassuring.

C. *Simulation Results*

Fig. 7 shows the average accuracy of each predictor style based on the benchmark being used. This number is the arithmetic average of all five simulations' accuracy results, per benchmark and per predictor style. The average was used since variation across the benchmarks for a particular predictor style

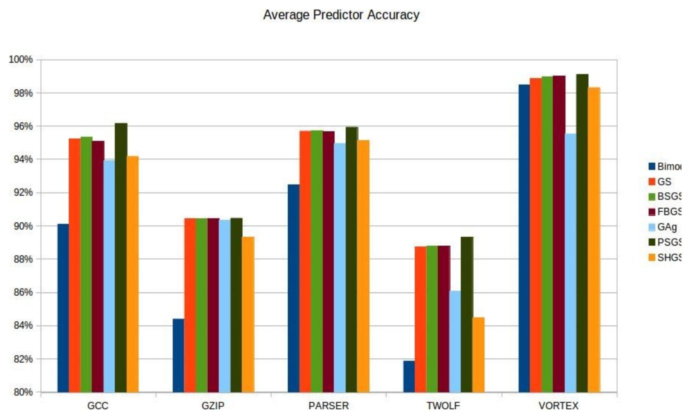


Fig. 7. Prediction Accuracy

was minimal. These results show that gshare style predictors can outperform a similarly sized bimodal in terms of the number of predictions accurately made. It is perhaps surprising to note that the GAg predictor is also more accurate than the bimodal for 4 of the 5 benchmarks despite being the generally weakest of the predictor styles that consider global history. It is noted that the highest performing predictor style for all benchmarks is the PSGS. This is likely because the PC information is doubly utilized as it is both XORed with the BHR and used to select a PHT set to index into.

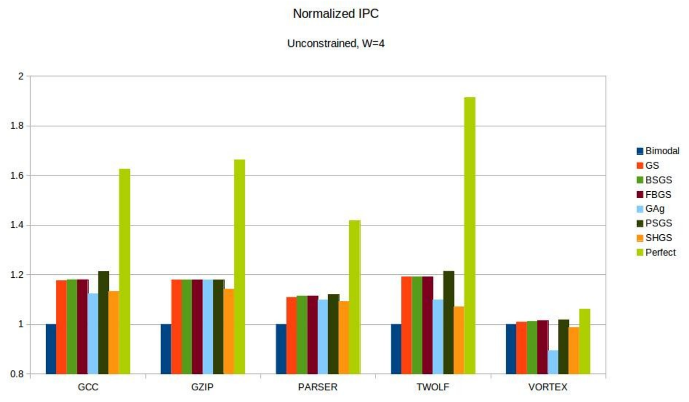


Fig. 8. Normalized IPC for 4 Wide Fetch

Fig. 8 and Fig. 9 show the IPC, normalized with respect to the bi-modal predictor, for the highly idealized simulations when the fetch width is equal to 4 and to 8, respectively. These results indicate how improved predictor accuracy yields improved performance when predictor accuracy is the only limiting factor

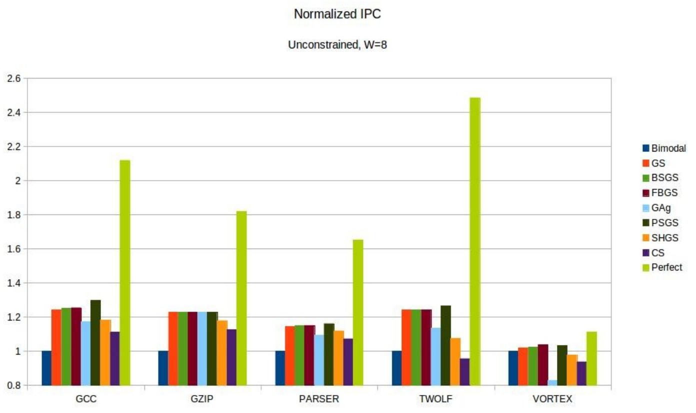


Fig. 9. Normalized IPC for a 8 Wide Fetch

in an architectural design. Also shown is the comparative improvement having 100% accurate branch prediction would give for each benchmark. From these plots we can see how the trends in IPC follow the trends in predictor accuracy.

Presented in Fig. 10 are the normalized IPCs for a more realistic model. The results are similar to those from the idealized models but the extent to which IPC is improved is

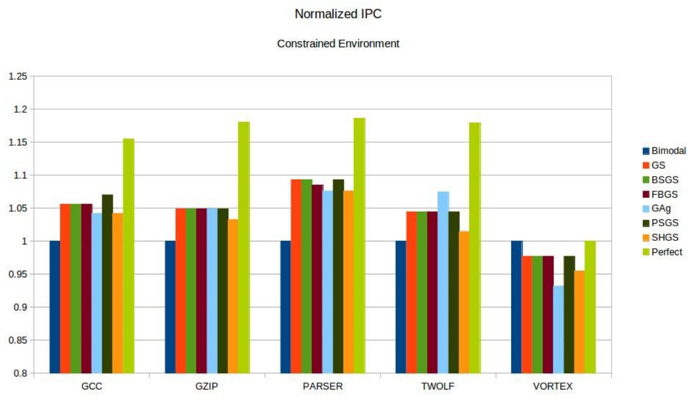


Fig. 10. Normalized IPC for a Constrained System

limited by various other constraints such as cache misses and load violations.

As stated above, from these results it can be observed that for all five benchmarks the Gshare style predictors have improved accuracy over the baseline Bimodal predictor. Also, for all of the benchmarks the predictor style with the greatest accuracy is the PSGS style since it maximizes the use of branch location information as well as utilizing the global history.

III. HDL IMPLEMENTATION

As discussed in the previous section, the PSGS has the best performance in terms or accuracy and consequently IPC. However, looking at the hardware complexity (Fig. 5) it is apparent that this design will have a negative impact on cycle time due to higher memory access latency caused by the larger number of read ports. This style also requires multiple, full sized PHTs while giving only marginal improvements over the other Gshare styles. Similarly the additional area required to implement the BSGS and FBGS styles limits their practicality. The standard GS style would seem like an ideal choice however the additional read and write ports to enable multiple predictions is a high cost. Likewise, the GAg predictor requires similar porting into the PHT to achieve multiple predictions per cycle.

This leaves the SHGS style which can be implemented using a single PHT with only one read and one write port per desired prediction. If banked, this gives several small memories with only a single read port and a single write port per bank. Since this style was consistently better than a similarly sized bi-modal predictor, the prediction accuracy was not significantly lower than other Gshare style predictors, and because the hardware costs of implementation were minimized, it was decided to implement this style of branch predictor.

Fig. 11. Gshare Predictor Micro-architecture

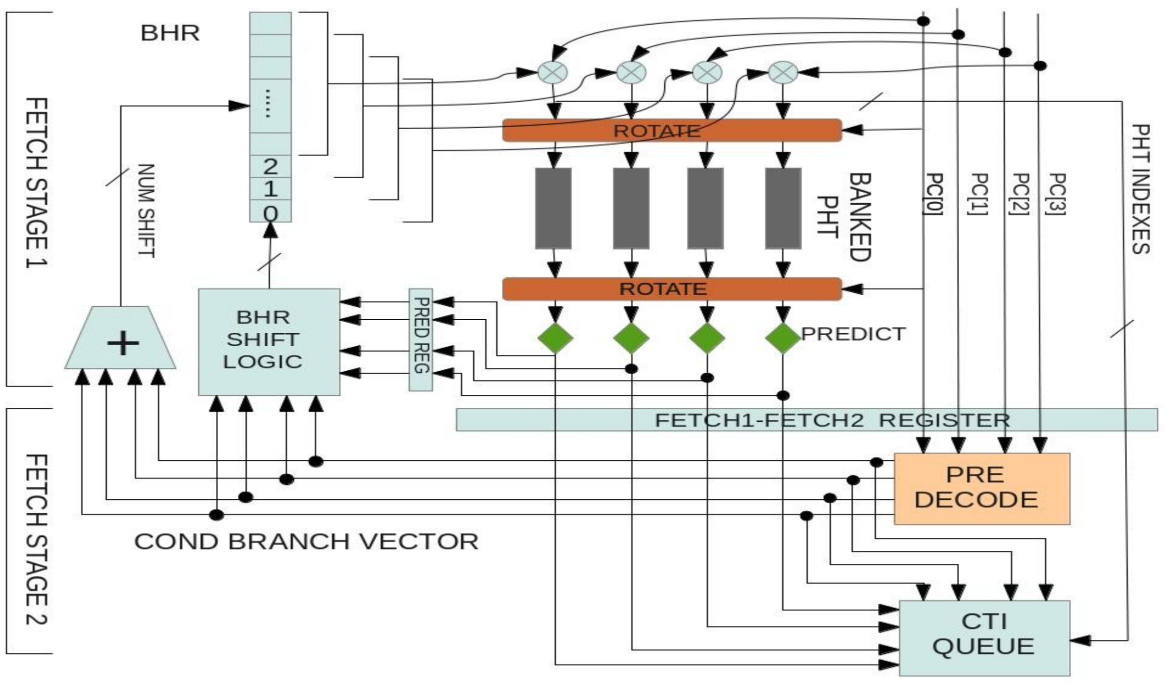
A. Predicting multiple branches

The predictor considers each instruction in the fetch bundle as a branch as it does not have the decode information for the instructions. For a 4-wide fetch, the hashing logic generates four different hashes using the PC of the instruction and a slightly stale BHR. Lower 2 bits of the PC, after discarding the lowest 3 bits (instruction byte offsets) are used to select the PHT bank from which the counter is read. The next (*log2K-2*) bits of the PC, where *K* is the total PHT size, are XORed with appropriate part of the BHR, after skipping three most recent history, to generate an index for each instruction. The indexes are aligned with the appropriate bank by rotating them. The counters read out are rotated back to be aligned with the actual fetch bundle and these counters are used to make the predictions. The index for each instruction is sent to Fetch Stage 2 where they are stored in the CTI queue to be used later while updating the PHT.

B. Updating the BHR

FabScalar has a two stage fetch where the first fetch stage generates a speculative next PC to fetch from the I-Cache. This calculation is based on information from the BTB and predictions made by the branch predictor. The second stage validates the PC calculated in stage 1 using decode information from a control pre-decoder and the predictions from the branch predictor.

The position of the branches in the raw fetch bundle in stage 1 are not known until Fetch Stage 2 and so the BHR can not be updated with the predictions made until the next cycle. The decode information from Fetch stage 2 is used to speculatively



update the BHR with predictions of only the conditional branches.

C. Updating the PHT

The pattern history table is updated when a branch retires and its actual outcome is known. The CTI queue provides the index and bank of the pattern history table to be updated. The update process is also pipelined. The current value of the counter is first read out from the required index and bank. This is then updated based on the branch outcome and written into the correct bank of the PHT in the next cycle. This makes sure the PHT is consistently updated while meeting cycle time requirements.

D. RTL Simulation Results

Six benchmarks from the SPECINT suite namely, bzip, gzip, gap, mcf, parser and vortex were used for evaluating the design in FabScalar. Fig. 13(a) presents accuracy of the Gshare predictor. As can be seen it performs quite well for three out of the six benchmarks and is slightly worse for the other three benchmarks. Fig. 13(b) shows the IPCs obtained for the design using Gshare normalized with respect to the design using bi-modal. The configuration of the processor pipeline used for RTL simulations is given in Table I.

TABLE I. CORE CONFIGURATION OF FABSCALAR

Parameter	Configuration
Fetch Width	4
Dispatch Width	4
Issue Width	4
Commit Width	4
Active List	128
Physical Register File	96
Issue Queue	32
Load/Store Queue	32
CTI Queue	16
Branch Predictor Table	64 K Entries
Branch Target Buffer	512 K Entries
Load/Store Lane	1
Control Execution Lane	1
Simple Lane	1
Complex Lane	1

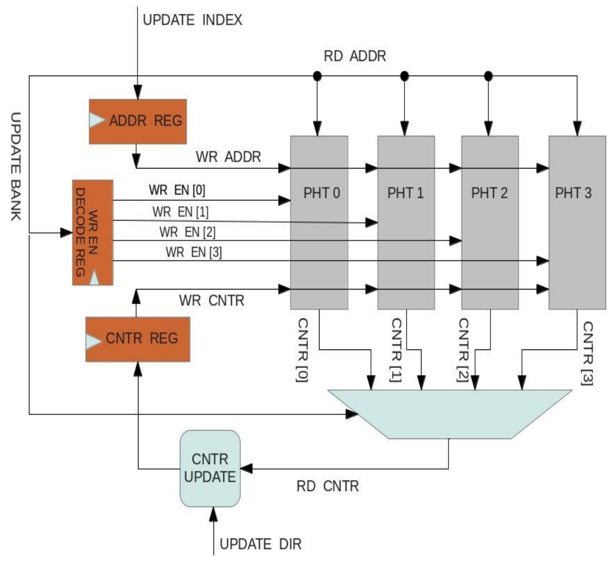
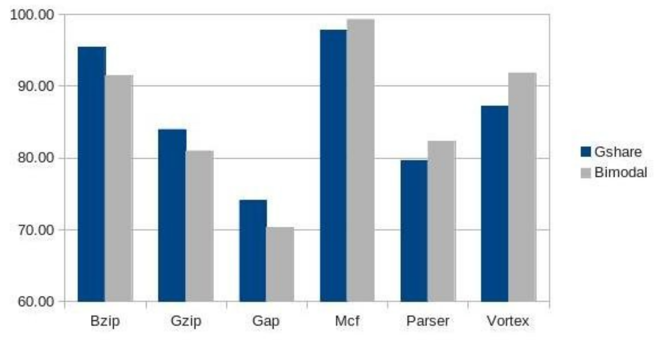


Fig. 12. PHT Update Micro-architecture



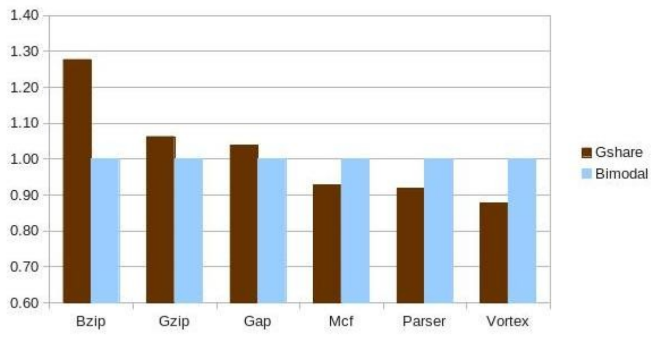


Fig. 13. (a) Predictor Accuracy (b) IPC Achieved

IV. DISCUSSIONS

A. *Mismatch Between C++ and RTL Results*

C++ simulation study showed improvements in most benchmarks when using the Gshare but similar improvements were not seen with the RTL. One striking difference that the C++ simulator has compared to the RTL is the availability of decode information and so it knows which instruction in a fetch bundle are branches and only makes predictions for those instructions. In case of the RTL, the decode information is not available and hence the history bits used to hash the PC may not be very consistent from one instance of a branch to the next. This can lead to significant inaccuracies for some cases and indeed that might be the case for the benchmarks that do not show much improvement over bi-modal.

B. *Limitations of the study*

The pipeline in the simulator was not modeled exactly as the RTL and hence might give optimistic results for some of the benchmarks. The RTL also assumes perfect I-Cache and D-Cache and the benefits of having a branch predictor with higher accuracy are not as pronounced as having realistic caches. In case of realistic caches, cache misses feeding mis-predictions [CFD Paper] can have significant impact on the IPC.

C. *Future Work*

- The Gshare branch predictor can be pipelined. This will allow the use of a much larger PHT without impacting the cycle time. Block ahead prediction is an option that can be explored [7][8].

- A hybrid branch predictor style [5] can be implemented leveraging the existing Bimodal predictor and the Gshare predictor. This is in fact quite common in most high performance processor as none of the two perform better than the other for all available workloads.

- The I-Cache can be augmented with a small predecode cache or some sort of a decode predictor table [4] that can quickly provide only relevant decode information to the branch predictor so that the hashing algorithm can be more consistent.

V. CONCLUSION

This paper has presented the results of a study into a variety of Gshare style branch predictors considering the improvement in accuracy over the Bimodal style of predictor as well as performance trade offs between the various styles themselves. It has been shown that a PC set Gshare style predictor can make branch predictions with a higher degree of accuracy than other Gshare style predictors as well as outperforming Bimodal prediction. The results gathered also demonstrated the correlation between increasing predictor accuracy and increasing IPC. Considering the results of this branch prediction study and practical physical implementation costs, a specific branch predictor style was selected to be implemented in the FabScalar RTL. This implementation showed improvement over the existing Bimodal predictor for several benchmarks tested. Finally this paper has proposed further work which can potentially improve the performance and functionality of FabScalar.

REFERENCES

[1] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi and E. Rotenberg. FabScalar: Automating superscalar core design. Micro, IEEE 32(3), pp. 48-59. 2012.

[2] J. Gandhi 1987-. FabFetch electronic resource] : A synthesizable RTL model of a pipelined instruction fetch unit for superscalar processors. 2010. Available: http://www2.lib.ncsu.edu/catalog/record/NCSU2282189; Get an online version (NCSU only) (http://www.lib.ncsu.edu/resolver/1840.16/6114).

[3] E. Hao, Po-Yung Chang and Y. N. Patt. The effect of speculative updating branch history on branch prediction accuracy, revisited. Presented at Microarchitecture, 1994. MICRO-27. Proceedings of the 27th Annual International Symposium on. 1994, .

[4] D. A. Jimenez. Reconsidering complex branch predictors. Presented at High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. the Ninth International Symposium on. 2003, .

[5] Po-Yung Chang, E. Hao and Y. N. Patt. Alternative implementations of hybrid branch predictors. Presented at Microarchitecture, 1995. Proceedings of the 28th Annual International Symposium on. 1995, .

[6] E. Rotenberg, S. Bennett and J. E. Smith. Trace cache: A low latency approach to high bandwidth instruction fetching. Presented at Microarchitecture, 1996. MICRO-29. Proceedings of the 29th Annual IEEE/ACM International Symposium on. 1996, .

[7] A. Seznec and A. Fraboulet. Effective ahead pipelining of instruction block address generation. Presented at Proceedings of the 30th Annual International Symposium on Computer Architecture. 2003, Available: http://doi.acm.org/10.1145/859618.859646.

[8] A. Seznec, S. Jourdan, P. Sainrat and P. Michaud. Multiple-block ahead branch predictors. SIGPLAN Not. 31(9), pp. 116-127. 1996. Available: http://doi.acm.org/10.1145/248209.237169.

[9] T. Yeh, D. T. Marr and Y. N. Patt. Increasing the instruction fetch rate via multiple branch prediction and a branch address cache. Presented at Proceedings of the 7th International Conference on Supercomputing. 1993, Available: http://doi.acm.org/10.1145/165939.165956.

[10] T. Yeh and Y. N. Patt. Two-level adaptive training branch prediction. Presented at Proceedings of the 24th Annual International Symposium on Microarchitecture. 1991, Available: http://doi.acm.org/10.1145/123465.123475.

[11] Scott McFarling. Combining Branch Predictors, HP Labs Tech Report 1993, Available: http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf

Appendix I

Division of Labor

Rangeen Basu Roy Chowdhury	Daniel Howe
Tasks : • Literature survey • Deciding predictor styles • RTL coding and verification • RTL Simulation and obtaining results • Trial synthesis run **Paper:** • Wrote the introduction, HDL and Discussions parts. • Did the detailed microarchitecture diagrams. • Did final merging and formatting of paper, final edit and bibliography **Contribution Factor:** 1	**Tasks:** • Literature survey • Deciding predictor styles • Modeling the styles in C++ simulator • Simulation and consolidating results **Paper:** • Wrote abstract, C++ Simulation and conclusion parts • Diagrams of the different styles • Graphs for the simulation results **Contribution Factor:** 1