Implementation and Comparison of Bi-Modal Dynamic Branch Prediction With Static Branch Prediction schemes

Abstract

There's a steady hindrance faced by high-performance processors in pipeline delays that are caused by a conditional branch instruction. To resolve the issue, the prediction of branches presents significant challenges. This paper aims to present the bi-modal prediction algorithm that stresses the role of dynamic branch prediction to boost the accuracy rate while mitigating the branch misprediction rate. Further, the paper compares the different schemes of static branch prediction with bi-modal dynamic branch prediction.

The results suggest that the bi-modal dynamic branch prediction provides higher accuracy than always taken and always not-taken static branch prediction by the rate of 11.30% and 18.20% respectively. Moreover, the bi-modal dynamic branch prediction limits the misprediction rate by 11.33% and 18.16% when respectively compared with the always taken and always not-taken static branch prediction.

Keywords: The *Pipeline Processor; A Static and Dynamic Branch Prediction; Accuracy Rate; Misprediction rate.*

Formatted: Indent: First line: 0 cm

1. Introduction

The advanced processors of this time allow the deep pipeline to improve the flow of instructions and speed-up the processor performance [1]. All processors developed since 1985 are using a pipeline for the execution of instructions and to improve the processor performances [2]. With the superscalar idea of the pipeline execution found in Intel processors [3], which involved various parallel execution units, it's manifold proficient to keep the pipeline full, by executing parallel instructions.

While trying to keep the execution pipeline completely used, the processors in equipment play out an enhancement optimization alluded to as speculative execution. With continuous execution, when any branch guidance is executing, this processor predicts which branch will be taken. It will at that point utilize this conjecture to start, conceivably rashly, executing the directions from the

anticipated branch way. At the point when the theory is right, it's alluded to as a Branch expectation precision rate hit, and when it's inaccurate, it's alluded to as a branch misprediction rate.
Pipelining is a usage system utilized in cutting edge chips, where the processor starts executing second guidance before the first has been registered. A pipeline is a progression of stages where some work is done at each stage. With pipelining, the execution will be done in five stages: Fetch (F), Decode (D), Execute (E), Memory (M), and Writeback (W).

1. The fetch stage is reliable for obtaining the requested instruction from the memory.

1. The decode stage takes the output of the fetch stage as input, and it's responsible for interpreting the guidance and conveying the different control lines to different pieces of the processor.

1. The execution stage considers the output of a decode stage as input, and some ALU calculations are performed in this stage.

1. The memory arrangement is liable for putting away and stacking qualities to and from memory.

Nowadays, every CPU processor is implemented with the pipeline to improve their efficiency, but a problem occurs when a conditional branch instruction comes in pipeline stages. Several hazards or delays cause a problem in the processor to affect the performance of processors- causing them to slow-down.

The hazards can be of three types: the data hazard, the structure hazard, and the controlling hazard, which is also known as branch hazard.

~~The advanced processors of this time allow the deep pipeline to improve the flow of instructions and speed up the processor performance [1]. All processors developed since 1985 use a pipelineuse pipeline for the execution of instructions and improve the processor performances [2]. With the superscalar idea of the execution pipeline found in Intel processors [3], which involved various parallel execution units, it's manifold proficient to keep the pipeline full, by executing parallel instructions.~~

~~While trying to keep the execution pipeline completely used, the processors in equipment play out an enhancement optimization alluded to as speculative execution. With continuouscontinous execution, when any branch guidance is executing, this processor predicts which branch will be taken. It will at that point utilize this conjecture to start, conceivably rashly, executing the directions from the anticipated branch way.~~

At the point when the theory is right, it's alluded to as a Branch expectation precision rate hit, and when it's inaccurate, it's alluded to as a branch misprediction rate.

Pipelining is a usage system utilized in cutting edge chipschip, where the processor starts executing a second guidance before the first has been registered. A pipeline is a progression of stages where some work is done at each stage. With pipelining, the execution will be done in five stages: **Fetch (F), Decode (D), Execute (E), Memory (M)** and **Write back (W).**

1) The fetch stage is reliable for obtaining the requested instruction from the memory.

2) The decode stage takes the output of the fetch stage as input, and it's responsible for interpreting the guidance and conveying the different control linesline to different pieces of the processor.

3) The execution stage considers the output of a decode stage as input, and some ALU calculations are performed in this stage.

4) The memory arrangementarrange is liable for putting away and stacking qualities to and from memory.

Nowadays, every CPU processor is implemented with the pipeline to improve their efficiency, but a problem occurs when a conditional branch instruction comes in pipeline stages. Several hazards or delays cause a problem in the processor to affect the performance of processors causing them to slow down.

The hazards can be of three types: **the data hazard**, **the structure hazard**, and **the control hazard,** which is also known as branch hazard.

Formatted: Indent: Left: 0 cm

1.1 Data Hazard

Also, knownknows as a pipeline data hazard, this hazard arises from the dependence of instruction on earlier instructions or data that is running in the queue or pipeline because data that is needed to execute the instructions areis not yet accessible [4].

1.2 Structure Hazard

This hazard arises when an instruction can't be executed in the best possible clock cycle because, at times, the hardware does not support multiple combinations of instructions that need to be executed [5].

1.3 Control Hazard

Formatted: Font: Italic
Formatted: Font: Italic

A branch in a sequence of instructions causes a problem. However, until the branch is resolved, the processor will not know from where to fetch the next instructions, and this causes a problem [5]. This situation gives rise to a control hazard [2,4].
Accurate branch prediction is essential to sustain an efficient pipeline, which is critical forto boosting the performance of modern processors. Branch prediction is essential in modern processors to mitigate the branch delay problems. Dedicated hardware is provided in advanced processors to correctly predict the direction, as well as the outcome of conditional branches.
If the branch direction is correctly predicted then the processor executes the continuous flow of instruction, and in case of not correctly predicted- the processor leads to flushing of the pipelines. Therefore, it's critical to predictpredicting the branch correctly. The branch prediction method falls into two categories of prediction:

A) Static branch prediction
B) Dynamic branch prediction.

- **Static branch prediction:** This method is usually carried out by the compiler.

This prediction is entitled static for the reason that the prediction is known before the execution of the program.

Some of the static branch prediction includes:

- Predict all branches that are taken.
- Predicting all branches that are not-taken.
- Predict backward taken and forward not-taken.
- Prediction based on profiling.

- **Dynamic branch prediction:** This method makes the decisions on the last executions of a program and has been more accurate than the static branch schemes.

Some of the dynamic branch prediction includes:

- One level or bimodal dynamic branch prediction scheme.
- The two-level adaptive prediction scheme.
- Correlated prediction scheme.
- Hybrid based prediction scheme.
- Neural branch predictor scheme.

Nevertheless, there have been a handful specialized development techniques that have upgraded the execution of pipeline processors [7]. However, Branch prediction is a method by which the performance of the pipeline processors can be increased [8]. Hence, to overcome their overheads, accurate prediction of the branch outcome is vital [9].

Contributions:

The current paper is the extension ~~in continuation~~ of our previous work [10], wherein we employed ~~implied~~ the static branch prediction schemes by using alpha benchmarks files. The performance of schemes was taken in terms of conditional branches, unconditional branches, branch address accuracy rate, and branch address misprediction rate. In this paper, we target at a dynamic branch prediction scheme that gives enhanced accuracy as compared to the static branch prediction schemes. The paper is structured in the following style:

Section1 introduces the role of branch prediction. **Section 2** presents a concise background of dynamic branch prediction schemes. The role of dynamic branch prediction is discussed in **Section 3**. **Section 4** outlines the experimental framework of different benchmarks files using the bi-modal dynamic branch prediction, the simulation results are presented in **Section 5,** and in **Section 6**, the paper concludes with a comment on the scope of future work.

2. Related work

Branch prediction performance issue has been studied extensively in the literature; therefore, this section focuses on the different schemes that are used for the branch prediction. A large amount of speculative work has been carried out to increase the

performance of processors by using several structures. Few of them from the deepfrom deep past were based on static predictions while others were dynamic in nature i.e. predictors that change its behavior in run-time.

J.E.Smith [2] introduces the comparative study of branch prediction strategies. He briefly discussed the seven hardware different strategies of branch prediction to improve the performance of processorsprocessor including static, dynamic and improved dynamic strategies.

Lee and A.J.Smith [11] evaluate the branch target buffer with many prediction schemes. The Branch target buffer is a small associative memory that contains the addresses of the most recent branches and their target locations.

The 2-level adaptive schemes were introduced by **Yeh and Putt** [12], [13] wherein two types of tables were talked about.

Pan et.al [14] proposed a correlation-basedproposed correlation based predictor that additionally uses the behavior of other branches to make a prediction.

Sweety and P. Chaudhary [8] introduced a brief review of the branch prediction and discussed the schemes of branch prediction which they categorized into 2 schemes, viz. Static branch prediction schemes and Dynamic branch prediction schemes.
The static branch prediction scheme only predicted whether all the branches are taken or not-taken.

In another paper, **Sweety and P. Chaudhary** [10] implemented the static branch prediction scheme by using some benchmarksbenchmarks file. The result concludes that the Always Taken Scheme provides a better result when compared to Always Not-taken Scheme.

The dynamic branch prediction scheme was firstly reported by **Lee and Smith** [11] and provides 68% accuracy in the prediction of branches.

However, according to the **McFarling** [15] dynamic branch prediction schemes provideprovides at least 90% accuracy in the prediction of branches. Nevertheless, as compared to the static schemes, a dynamic scheme has been studied extensively and gives better accuracy than static schemes.

In the dynamic branch prediction, firstly a two-bitfirstly two-bit dynamic branch predictor was introduced by **Hennessy and Patterson** [16].

To overcome the limitations of a two-bit dynamic branch predictor and tofor achieve improved accuracy of branch prediction, **Su and Zhou** [17] provided special aspects to analyzeanalysis the performance.

Table 1: Different types of Dynamic branch predictor with their features and challenges:

Branch Predictors	Features and challenges	Reference
Smith Algorithm's	Improve the performance by a small increment, but it doesn't use the store history tables of instructions.	[2, 10]
Two-level predictor	Uses two separate levels of branch history tables. However, However, trade-off between sizes of two-tables occurs.	[11,18-20]
Index sharing predictor	Size of the historyof history table is large as compared to the two-level predictors. Hashing branch history register and PC together leads to better accuracy in processor performance.	[21–23]
Agree Predictor	Reduces destructive aliasing interference by reinterpreting the pattern history table counter.	[25, 26]
Hybrid branch predictor	Combines two or more predictors to make one final prediction, but sometimessometime partially misunderstandsmisunderstand the hybrid path at the time of prediction.	[24, 25]
Path based neural branch predictor	Uses ahead pipelining to balance idleness issuesissue, be that as it may, it replaces more force and region effective convey spare adders of unique perceptron branch indicator with a few convey finishing adders, prompting higher multifaceted nature.	[19, 24,26]
PiecewisePiece-wise linear neural branch predictor	Gives much higher precision, however fundamentally builds the check pointing and recuperation overhead and the quantity of adders.	[27, 28]

3. The Dynamic branch prediction

The fact is well-established that in pipeline architecture processors, conditional instructions break continues the flow of instruction during the execution of instructions. This break results in a form of delay which creates a decrease in the processor performance.
To overcome this delay and improve the performance of the processor, one can attempt to guess the directions of the conditional instructions.

Hence, branch prediction improves the performance of a pipeline~~of pipeline~~ by providing a correctly~~correct~~ predicted path for the instructions after a conditional branch has been encountered.

In other words, branch prediction is the ability to make an educated guess about the way a branch will go; will the branch be taken or not. If the mispredicted path occurs, the pipeline gets flushed and again that path is predicted until the conditional branch address is matched [33].

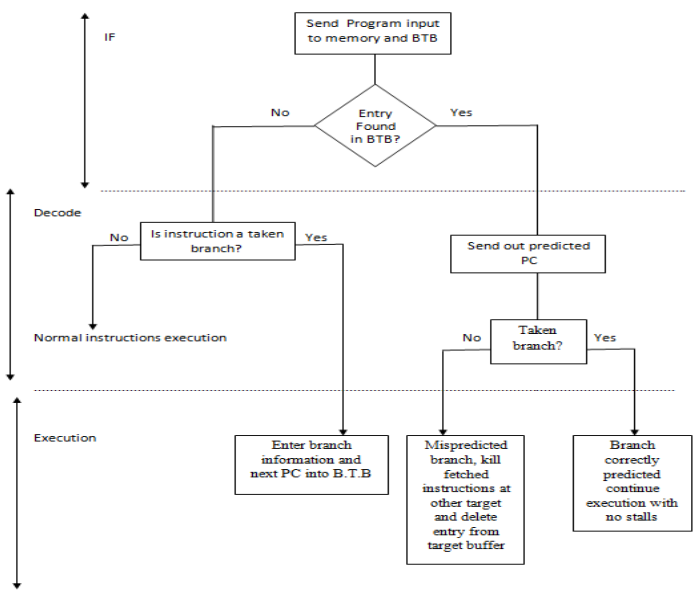

Fig.1. Flow chart of executing instruction using dynamic behaviour in the pipeline

Dynamic branch predictors choose the instructions dynamically to execute the next, possibly reordering sequences of the instructions to avoid the stalls or control hazard. The performance of the predictors depends on the accuracy as well as on~~as the on~~ the misprediction rate of instructions.

Performance = f (accuracy rate, misprediction rate)

- The accuracy rate is defined as the prediction path that is correct for the conditional branches.
- The misprediction rate is defined as the prediction path that is not correct for the conditional branches.

In Dynamic prediction, the prediction of branches depends on the recent execution history of each branch and that history is stored in the Branch History Table (BHT) [34]. But, if the prediction of branches goes wrong, the pipeline gets stalled~~stall~~ while re-fetching the instructions, and simultaneously the branch~~simultaneously branch~~ history table gets updated. This prediction maintains a history table that stores information about the executed instructions [18]. The BTB (Branch Target Buffer) is illustrated in Fig. 2. to detect the presence of a branch instruction in the progression of got directions before it has been decoded. It also provides the branch target address when the branch is anticipated as taken. The BTB is listed by lower bits of the program counter and every section is labelled with the branch address for check.

3.1 Bi-modal Dynamic Branch Prediction Scheme

The Bi-modal predictor scheme tracks the individual behavior of branches.This prediction provides a counter for each entry when any conditional instruction occurs [35]. The value of the counter lies between 0 and 3. The two-bit values represent viz.; 00, 01, 10, 11 [11]. The following values and their prediction selection are represented in Table 2.

Table 2. Representation of states of counter values and their prediction selection

Counter values	Prediction Selection
00	strongly not-taken
01	weakly not-taken
10	weakly taken
11	strongly taken

The bimodal branch prediction scheme is similar to the theory of finite state machines. In this, the counter is incremented or decremented according to whether the~~to the~~ branch is taken or not taken. In fig inside the circle is the prediction whether it will be taken or not taken state and outside label is what is the actual~~is actual~~ prediction state.

The finite states of the bimodal branch prediction scheme are represented in Fig. 2

Formatted: Indent: Left: 0 cm, First lin

Formatted: Font color: Auto

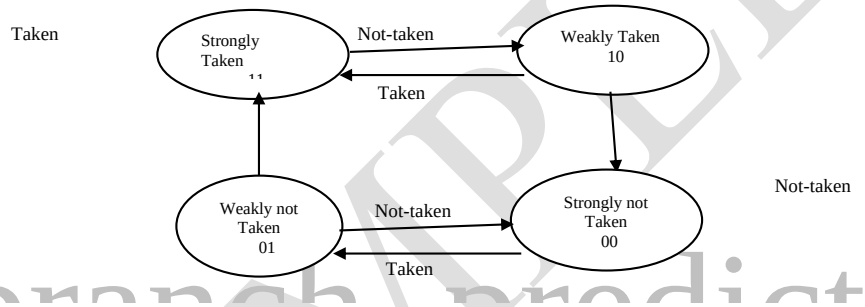

Fig. 2: The finite states in bi-modal branch prediction scheme

The above diagram explanation is followed by the table below:

History Bits	Resulting Description	Prediction Made	If branch taken	If not Taken
11	Strongly Taken	Branch Taken	Remains in same state	Decreased to weakly taken
10	Weakly Taken	Branch Taken	Upgraded to strongly taken	Decreased to weakly not-taken
01	Weakly not taken	Branch not taken	Upgraded to weakly taken	Decreased to strongly not-taken

00	Strongly not-taken	Branch not-taken	Upgraded to weakly not-taken	Remains in same state

4. Experimental Framework

This section describes the simulation experiment, benchmarks, and statistics of different benchmarks files using the bi-modal dynamic branch prediction.

4.1. Simulation Experiment

The simulation of the proposed work of the bi-modal dynamic scheme is done using the modified version of SimpleScalar~~Simple Scalar~~ toolset simulator 3.0, which is developed at the University of Wisconsin in Madison.

The toolset of this simulator includes performance visualization tools, statistical analysis resources, debug, and verification infrastructure. Fig.3. outlines the configuration of the simulator for dynamic branch prediction.

4.2 Benchmarks

Now the paper of study is directed through a unique~~through unique~~ dynamic simulation driven by utilizing the SPEC alpha benchmarks which consists of five different benchmark programs: tests-args, test-dirent, test-fmath, test-long, and test-lawler.

At the point when any benchmark~~benchmarks~~ runs on the tool, the subsequent yield gives the complete number of directions, numerous branches executed, and the quantity of branches whose address is hit and miss.

Table.3 represents SPEC alpha benchmarks files and the input used, the total number of instructions executed, the number of loads and store instructions, the total number of conditional and unconditional branch instructions, and the total number of instructions per branch.

Table 3. SPEC alpha benchmarks information

Benchmark	Input file	Total number of instructions	Total number of load and store instructions	Total number of conditional branch instructions	Total number of unconditional instructions	Total number of instructions per branch
anagram	test-anagram.in	4904	1825	870	683	5.6368
args	Test-args.in	6447	2252	1211	1041	5.3237
dirent	tes-dirent.in	4498	1738	799	939	5.6295
llong	test-llong.in	10572	3425	1876	1549	5.6354
lslwr	Test-lslwr.in	4871	1867	863	1004	5.6443

4.3 Benchmark Statistics

Conditional branch instructions dramatically affect the performance of a pipeline and each branch behaves differently with different applications. It is important to generalize the performance of a pipeline by predicting the direction of branches by using any predictor with some benchmark files.
In our previous work, we had implemented static branch prediction schemes using always taken and always not-taken prediction schemes.

In the current paper, SPEC-alpha benchmarks files are run on the Simple-scalar tool by using a bi-modal dynamic branch predictionpredictor scheme. To calculate the performance of a bi-modal dynamic branch prediction scheme, each benchmark filefiles runsrun on the Simple-scalar tool to get the branch accuracy rate,= and branch misprediction rate.

5. Simulation Results

This section describes the performance accuracy of the branch address hit rate and branch misprediction rate of the bi-modal dynamic branch prediction scheme, and comparison of the static branch prediction scheme, as already implemented in our previous papers.

5.1 Performance impact of benchmark files

Fig. 4. shows the performance analysis for five SPEC-alpha benchmarks. Taking the reference from the figure above, it can be identified that each benchmark~~benchmarks~~ file~~files~~ has~~have~~ a different number of branch executed instructions. Each benchmark~~benchmarks~~ file~~files~~ provides~~provide~~ a different rate for branch address hits as well as for the branch misprediction rate. To determine the accuracy rate as well as the misprediction rate of branch instructions, simulations were carried out on a modified version of SimpleScalar~~Simple Scalar~~ 3.0 simulator.

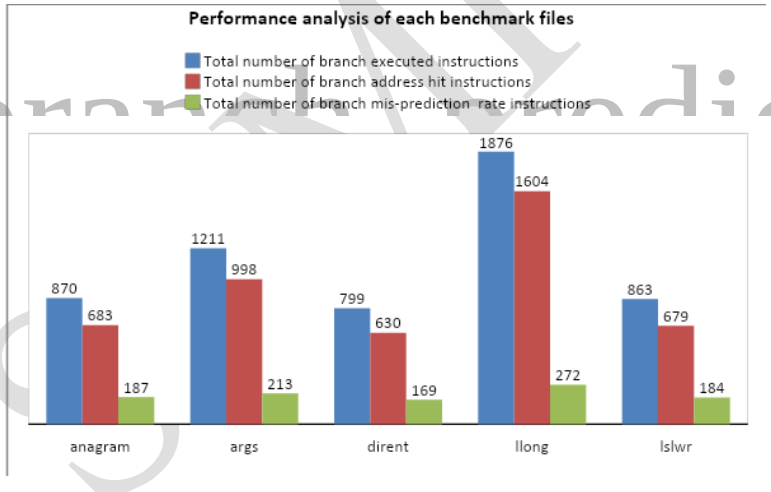

Fig. 4: Performance impact of each benchmark files: total number of branch executed instructions, branch address hit, and branch misprediction rate.

5.2 Performance impact of branch prediction accuracy

The impact of the branch prediction accuracy rate is identified when a predicted direction of a branch is equal to the actual direction of that branch.

Table 4 indicates the branch prediction address hit rate accuracy for each benchmarks file.

This table provides a comparison of the static scheme namely, the always-taken prediction scheme and always not-taken prediction scheme with a bi-modal dynamic branch prediction scheme. Evidently, it refers to~~that~~ the accuracy rate of address hit in~~hit is higher in~~ the bi-modal dynamic branch prediction scheme.

The current paperwork implements a bi-modal dynamic branch prediction scheme by analyzing the same benchmarks file and the result of the branch accuracy rates are better than the previous schemes.

Formatted: Indent: Left: 0 cm

Formatted: Font color: Auto

Table 4: Comparison of branch prediction address hit rate

Branch Prediction schemes	SPEC-alpha Benchmarks Files					Total Accuracy
	Anagram	args	Dirent	llong	Lslwr	
Always-taken prediction scheme	75.75%	70.08%	75.31%	62.66%	63.40%	69.44%
Always not-taken prediction scheme	6.40%	63.55%	59.14%	61.29%	62.33%	62.54%
Bi-modal prediction scheme	78.50%	82.41%	78.84%	85.50%	78.47%	80.74%

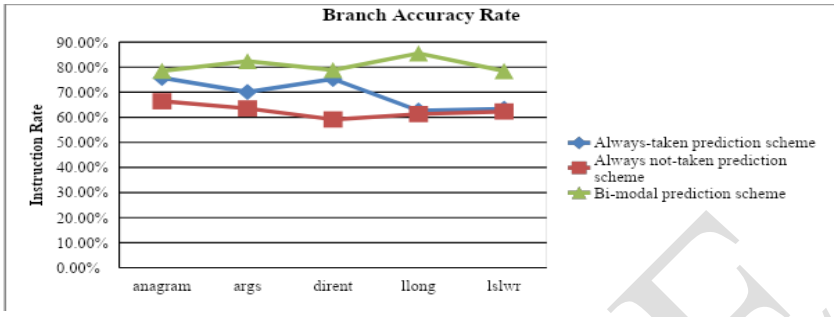

Fig.5: Performance accuracy of branch address accuracy rate of different schemes of the prediction scheme.

As evident in fig.5. the bi-modal prediction scheme shows higher accuracy as compared to the other schemes. As a matter of fact, the accuracy rate is different in each benchmark file in the corresponding scheme. The average accuracy rate of bi-modal prediction is 11.30% higher than always taken prediction scheme, and 18.20 % higher than always not-taken prediction scheme.

5.2 Performance impact of branch misprediction rate:

The impact of branch misprediction rate is observed when a predicted direction of a branch is not equal to the actual direction of that branch.

Table 5 reflects the branch misprediction rate for each benchmarks file. This table suggests the comparison of the static schemes such as always-taken prediction scheme and always not-taken prediction scheme with bi-modal dynamic branch prediction scheme. Visibly, it shows that the misprediction rate is lowest in the bi-modal dynamic branch prediction scheme. The results of always taken prediction schemes and always not-taken prediction schemes were implemented in our previous work. In this paper, we implemented the bi-modal dynamic branch prediction scheme by using the same benchmarks file and the result of the branch misprediction rate is better than the previous schemes.

Fig. 5 reflects that the bi-modal prediction scheme provides the lowest misprediction rate when compared to the other schemes. Furthermore, the misprediction rate is

different in each benchmark file in every scheme. The average misprediction rate of bi-modal prediction is 11.30% lower than always taken prediction scheme, and 18.18 % lower than always not-taken prediction scheme.

6. Conclusion and Future Scope

In the current constructed paper, a bi-modal dynamic branch prediction scheme is implemented that has a lower misprediction rate and higher accuracy rate than the static prediction schemes. Leveraging the benchmarks files, the accuracy rate of bi-modal prediction is 11.30% higher than always taken prediction scheme, and 18.20 % higher than always not-taken prediction scheme.

On the other hand, the misprediction rate is 11.30% lower than always taken prediction schemesscheme, and 18.18 % than always not-taken prediction schemesscheme. These results satisfactory suggest that the dynamic branch prediction scheme provides improved results than static branch prediction schemes.

The branch prediction accuracy can be optimized by combining static with dynamic branch prediction. Some of the branches can be predicted with a static bias bit, while the others having less biased behavior can use the dynamic predictor. Machine learning also mechanizes a genetic algorithm or neural network to improve the accuracy rate of the branch predictor.

References

[1] Shen, J. P., & Lipasti, M. H. (2013). *Modern processor design: fundamentals of superscalar processors*. Waveland Press.

[2] Smith, J. E. (1998, August). A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)* (pp. 202-215).

[3] Le, H. Q., Van Norstrand, J. A., Thompto, B. W., Moreira, J. E., Nguyen, D. Q., Hrusecky, D.,& Kroener, M. (2018). IBM POWER9 processor core. *IBM Journal of Research and Development*, *62*(4/5), 2-1.

[4] Pandey, A. (2016, August). Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor. In *2016 International Conference on Inventive Computation Technologies (ICICT)* (Vol. 2, pp. 1-4). IEEE.

[5] Y. He, H. Wan, B. Jiang, and X. Gao, “A new method to prevent control hazard in pipeline processor by using an auxiliary processing unit,” *MATEC Web Conf.*, vol. 139, p. 00085, 2017.

[6] A. Changela, “Hazard Detection and Data Forwarding Scheme for 5-stage Pipeline Structure of RISC Processor,” *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 4, no. 6, pp. 11693–1169, 2016.

[7] Wang, Y., & Chen, L. (2015). Dynamic Branch Prediction Using Machine Learning. *ECS-201A, Fall*.

[8] Sweety and P. Chaudhary, "Implemented Static Branch Prediction Schemes for the Parallelism Processors," 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 2019, pp. 79-83.

[9] Youssif, A., Ismail, N., & Torkey, F. (2004, September). Comparison of branch prediction schemes for superscalar processors iceec 2004. In *Electrical, Electronic and Computer Engineering, 2004. ICEEC’04. 2004 International Conference on* (pp. 257-260).

[10] Lee, J. K., & Smith, A. J. (1984). Branch prediction strategies and branch target buffer design. *Computer*, (1), 6-22.

[11] Yeh, T. Y., & Patt, Y. N. (1992). Alternative implementations of two-level adaptive branch prediction. *ACM SIGARCH Computer Architecture News*, *20*(2), 124-134.

[12] Yeh, T. Y., & Patt, Y. N. (1993, May). A comparison of dynamic branch predictors that use two levels of branch history. In *Proceedings of the 20th annual international symposium on computer architecture* (pp. 257-266).

[13] Pan, S. T., So, K., & Rahmeh, J. T. (1992, September). Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems* (pp. 76-84).

[14] McFarling, S. (1993). *Combining branch predictors* (Vol. 49). Technical Report TN-36, Digital Western Research Laboratory.

[15] Hennessy, J. L., & Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Elsevier.

[16] Su, Z., & Zhou, M. (1995). A comparative analysis of branch prediction schemes. *University of California at Berkeley, Computer Architecture Project.*

[17] Rajper, A., Talpur, S., & Rajper, N. J. (2017, April). Analysis of Performance of Instruction Pipeline with Transactional Slice Mechanism in CMP. In *2017 UKSim-AMSS 19th International Conference on Computer Modelling & Simulation (UKSim)* (pp. 209-214). IEEE.

[18] Egan, C., Steven, G., Quick, P., Anguera, R., Steven, F., & Vintan, L. (2003). Two-level branch prediction using neural networks. *Journal of Systems Architecture*, *49*(12-15), 557-570.

[19] Steven, G., Anguera, R., Egan, C., Steven, F., & Vintan, L. (2001, September). Dynamic branch prediction using neural networks. In *Proceedings Euromicro Symposium on Digital Systems Design* (pp. 178-185). IEEE.

[20] Cheng, C. C. (2000). The schemes and performances of dynamic branch predictors. *Berkeley Wireless Research Center, Tech. Rep*.

[21] González, A. M. (1993). A survey of branch techniques in pipelined processors. *Microprocessing and microprogramming*, *36*(5), 243-257.

[22] Mittal, S. (2019). A survey of techniques for dynamic branch prediction. *Concurrency and Computation: Practice and Experience*, *31*(1), e4666.

[23] Chaudhary, P. (2019, February). Implemented Static Branch Prediction Schemes for the Parallelism Processors. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)* (pp. 79-83). IEEE.

[24] Jin, W., Shi, F., Song, Q., & Zhang, Y. (2013). A novel architecture for ahead branch prediction. *Frontiers of Computer Science*, *7*(6), 914-923.

[25] Otiv, S., Garikipati, K., Patnaik, M., & Kamakoti, V. (2014, June). H-pattern: A hybrid pattern based dynamic branch predictor with performance based adaptation. In *Proc. 4th JILP Workshop Comput. Architecture Competitions: Championship Branch Prediction*.

[26] Pan, G., & Lu, M. (2006). Analysis of Branch Predictors.

[27] Jiménez, D. A., & Lin, C. (2001, January). Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture* (pp. 197-206). IEEE.

[28] Vintan, L. N., & Iridon, M. (1999, July). Towards a high performance neural branch predictor. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)* (Vol. 2, pp. 868-873). IEEE.

[29] Ribas, V. M., Figueiredo, M. F., & de Lara, R. A. (2003). Simulating a simple neural network on branch prediction. *Acta Scientiarum Technology*, *25*(2), 153-160.

[30] Amant, R. S., Jiménez, D. A., & Burger, D. (2008, November). Low-power, high-performance analog neural branch prediction. In *2008 41st IEEE/ACM International Symposium on Microarchitecture* (pp. 447-458). IEEE.

[31] Sbera, M., VINTAN, L. N., & FLOREA, A. (2014). Static and Dynamic Branch Prediction Using Neural Networks. *no. May*.

[32] Arora, H., Kotecha, S., & Samyal, R. (2013, December). Dynamic Branch Prediction Modeller for RISC Architecture. In *2013 International Conference on Machine Intelligence and Research Advancement* (pp. 397-401). IEEE.

[33] Parasanna, S., Sarma, R., & Balasubramanian, S. (2017). A study on improving branch prediction accuracy in the context of conditional branches. *Int J Eng Technol Sci Res*, *4*, 654-662.

[34] Hennessy, J. L., & Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Elsevier.

branch predicti