I.J. Modern Education and Computer Science, **2017, 5, 27-33**
Published Online May 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2017.05.04

Two-Level Alloyed Branch Predictor based on Genetic Algorithm for Deep Pipelining Processors

Shivam Goyal
JECRC University, Jaipur, Rajasthan 303905, India
E-mail: shivamgoyal.sg05@gmail.com

Jaskirat Singh
JECRC University, Jaipur, Rajasthan 303905, India
E-mail: jaskirat.singh@jecrcu.edu.in

***Abstract*—**To gain improved performance in multiple issue superscalar processors, the increment in instruction fetch and issue rate is pretty necessary. Evasion of control hazard is a primary source to get peak instruction level parallelism in superscalar processors. Conditional branch prediction can help in improving the performance of processors only when these predictors are equipped with algorithms to give higher accuracy. The Increment in single miss-prediction rate can cause wastage of more than 20% of the instructions cycles, which leads us to an exploration of new techniques and algorithms that increase the accuracy of branch prediction. Alloying is a way to exploit the local and global history of different predictors in the same structure and sometimes also called hybrid branch prediction. In this paper, we aim to design a more accurate and robust two-level alloyed predictor, whose behavior is more dynamic on changing branch direction.

***Index Terms*—**Control hazard, Branch prediction, Alloying, hybrid branch predictors, dynamic, robust.

I. Introduction

High-performance computers are embedded with the processors with deeper pipelining and also with higher parallelism in instruction fetch, and are called superscalar processors. The Increment in instruction fetch parallelism causes control and data hazards. Data forwarding units can be implemented to remove data hazards whereas branch predictors deal with control hazards. A lot of cycles get wasted in a single miss-prediction, that's why we need branch predictors having very high level of accuracy. One way of deploying this is to make adaptive branch predictor [11]. Such predictor uses the information about the behavior of a branch for fixed number of iterations (local history) or/and behavior of fixed number of past branches (global history) stored in it at the time of execution [3]. The new way of dealing with this problem is to use multiple advanced dynamic branch predictors in the same structure to maintain the accuracy of prediction throughout. This idea came into existence because every branch predictor has its characteristics and behaves differently with the change in frequency of branch directions. Branch prediction is an on-going subject.

Branch prediction is also a point of interest for many researchers. There are several ways of deploying the branch predictors, but two-level alloyed predictor is considered to be most accurate [7]. The technique that a highly accurate predictor uses is recording the branch history and observing each branch scenario which predictor came across. Predictor learns from these scenarios and starts predicting better. Static selection is undesirable as the direction of individual branches varies dynamically, and there could be a need for both global and local history [1].

The need for global and local history made alloyed branch prediction technique came into existence. Alloying gives higher prediction accuracy over conventional branch predictors with the addition of nominal hardware to store branch/branches history. No advancement in prediction mechanism limits alloying technique to get efficiencies ranging in 70%-95% and in some situations the miss-prediction rate can be as high as 35%. A new source of miss-prediction comes into account if we deploy components for both local and global history [9]. This miss-prediction happens when branch needs local history and predictors track the global one or vice-versa and hence poor behavior. This kind of prediction is called wrong-history miss-prediction. The Wrong history here doesn't mean that history bits are wrong, but the track is inappropriate for a particular branch [3]. Separate structures for global and local history can solve this problem but still alloyed branch predictor lacks in good prediction mechanism. This lacking in accuracy is the reason why alloyed predictors use other schemes such as Tournament predictors or Genetic Programming to increase performance. An energy efficient branch predictor with less history size and self-learning capacity is our goal so that the predictions will become more robust and dynamic according to change in frequency of branch direction in

instruction fetch.

This paper explains why and when our predictor performs well. The Genetic algorithm with a particular fitness function we have chosen works well for the class of continuously flipping branches (if-else branches) along with those branches whose flipping tendency is very less (loop branches). We know that programs tend to have branches, of which, these two kinds frequently occur but when they do not, our predictor may not perform as well as other techniques. Thus, our predictor works best in the conditions where branch transition probabilities are either too high or too less.

Our predictor gives a hike in performance and reaches the accuracies as high as 98% with the128K budget in the two extreme cases mentioned above. On the other hand for other cases, the performance is not remarkable, and mispredictions rate reaches to 20-25%.

This paper makes the following contributions. In section-I, we introduce the basic branch problem and validation of scheme proposed in this paper. In section-II, we provide a deep literature survey, which explains a wide variety of branch resolving techniques and covers major related work. Furthermore, in section-III, we proceed in our paper by giving a thorough explanation of how to integrate genetic algorithm in branch prediction. This section also explains genetic operations involved in GA. In section IV we explained the whole methodology, which we have used while integrating genetic algorithm with alloyed branch predictor. This section focuses on modeling GA and explains how genetic operations work to select the best predictor in present situation. Section V includes results of our simulations and performance comparison of conventional branch predictors with proposed predictor has been done. Lastly, in section VI we concluded the performance analysis done in the previous section. This section also gives the validation of proposed work. Tabulated results are attached at the end of the paper.

II. Related Works

The literature on branch prediction schemes is huge [13][2][11][14][4][6]. Few of them from deep past were based on static predictions while others were dynamic in nature i.e. predictors which change its behavior in run-time. Researchers have done Substantial research to decrease aliasing problem, power consumption problem, delay and hardware in branch predictors [15][1][3]. Bi-Modal branch predictor [13] is the simplest dynamic branch predictor. It is more accurate than one-bit branch predictor and one of the most widely used predictors. The accuracy of this predictor depends on the dynamic behavior of the branch, which limits the accuracy to 50% [2]. It also has a high level of aliasing in PHT (Pattern history table) that affects the accuracy. Two-level Adaptive Branch predictors and hybrid predictors such as YAGS (Yet Another Global Scheme) branch predictors can improve efficiency. These predictors consume less power and have improved accuracy with little aliasing [11][15][16]. The main advantage we get from the adaptive or dynamic branch predictor is they learn the repetitive pattern of change in branch direction if any, studies the branch behavior and acts accordingly. It does not alter its prediction very quickly but changes with changing scenario [11].

A more accurate 2 level branch predictor, which takes the recent behavior of other branches into consideration, is called Correlating branch predictor [2][14]. A previously executed branch can affect the direction and outcome of the future branch if a previous branch runs source operands of the next branch. Correlated branch predictor handles these correlations between the branches and improves prediction mechanism by identifying the correlated branches from large history table. G-share branch predictor eradicates the problem of aliasing [12] in earlier branch predictors. When multiple branches mapped onto the same location in branch history table, it is called aliasing. Therefore, many positions of branch history table remain unmapped. G-share branch predictor [2] uses the XOR function to produce new bits, which will further point to the different locations in the history table. In G-share prediction mechanism XORing of ‘k’ least significant bits of current PC address is being done with address bits of global history table [9]. This XORing gives a new address, which will point to a unique location of branch history table and we take prediction bit of that address to predict the current branch. Generation of unique address every time in G-share predictors prevents aliasing. Removal of aliasing helps in increasing the prediction accuracy but with a margin of 2-5%, and there is still a scope to recuperate the branch prediction techniques and algorithms [4]. The right amount of research on resolving multiple branches in a single cycle has also been done in recent past [17].

Alloyed branch prediction has its importance in its robustness in using the local and global history in the same structure. Alloyed or hybrid predictors are designed to fulfill the requiring need of improvement in the accuracy of predictors and also its an effort to develop a structure to predict branches (T/NT) more dynamically. Alloyed predictors changed the prediction mechanism slightly which was earlier unaltered in conventional branch predictors [6]. Hybrid predictors with different PHT for taken and not taken branches are efficient enough to improve prediction accuracy but with a large hardware budget.

Branch prediction structures that use more than two single scheme predictors were introduced in the past. These predictors use two kinds of predictors: one with less warm-up time along with those who have highest prediction efficiency but have great warm-up time. These predictors are made keeping the fact in mind that most efficient predictors have large history tables and they need training time to get higher accuracies i.e. warm-up time of these predictors are large. Hybrid predictors [18] Initially use small single scheme predictor with less warm-up time. In the meantime, the history table gets filled with enough prediction bits to train the other predictor embedded in the structure. This predictor then

uses this history to predict with better accuracy and maintain it. Predictor that uses neural networks (perceptrons) has also been proposed [5]. These predictors with machine learning techniques are more sophisticated than conventional predictors. They claim to give higher accuracies of prediction in nominal hardware budget (4K byte).

III. Branch Prediction With Genetic Algorithm (GA)

Branch prediction with GA is a new way to predict the branch direction (T/NT). GA has a knack to generate potential solutions and to improve them, as it evolves. GA is all about moving to the fittest solutions and maintaining the selection of the same. This property of GA motivates us to design an environment, where GA selects the most suitable predictor to be used and also picks up the prediction bit of the same predictor [5].

A. Why GA

The motivation of integrating our primary alloyed branch prediction mechanism with the GA is to improve the conventional prediction scheme and to have a substantial growth in prediction accuracy. GA is efficient in elucidating two of the biggest problems, which we come across while predicting through conventional Hybrid predictors: One is how to know which predictor is making the correct predictions and the second is how to ensure that the prediction bit generated by our conventional predictors is right in the present scenario. GA does this job efficiently as we can encode the type of predictor with its characteristics in the binary string that are called chromosomes. The genetic operations help us in selecting the winner chromosome. The prediction of the predictor encoded in the winner chromosome can then be made.

B. How GA Works

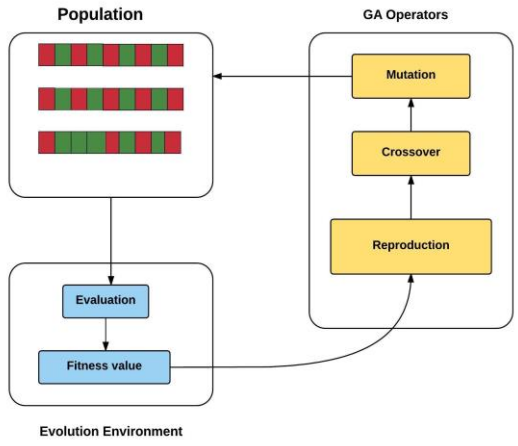


Fig.1. GA mechanism

Generating Initial Population- The very first step in GA is to create an initial population of solutions, which we call chromosomes. These chromosomes are the binary strings as shown in Figure 4 whose bits are the properties of the different predictors such as the warm-up time, more accurate predictor, and probability difference between the last few taken/not-taken branches.

Applying Fitness Function- The next step is to find the winner chromosome among all the chromosomes selected and for this, we use fitness function over our chromosomal strings as shown in the second phase of Figure 4. The fittest chromosome is selected and the prediction bit of the predictor encoded in the winner chromosome is taken into account [8].

Mutation -Mutation is done on a single bit or multiple points of current generation. It is a step to prevent convergence in multiple offspring. Mutation is done in the binary chromosomal string to flip few of the bits, which will try to make every new generation unique [8].

Crossover - Crossover is done to copy useful bits from winner chromosome to other offspring. It is done to generate healthier future generation from the recipient chromosome. In this way, the new generation produced will be healthier than the old generation of solutions [8].

IV. Proposed Methodology

The use of 2-bit counter in traditional predictors such as correlated and G-share gives miss-prediction whenever counter is at corrupt state. This type of miss-prediction reduces the accuracy. Which led us to create a mechanism, which can work accurately even when the counter goes wrong. Integrating GA with alloyed predictor does the same. GA not only selects the prediction bit of best predictor but also tries to predict branch correctly when selected predictor fails. So in this way scope of wrong predictions decreases. The criterion of prediction used by GA in the cases when the counter goes wrong is to calculate probability difference of last 8 T/NT branches. GA gives its forecast as taken if the probability of the number of taken branches is more or vice-versa.

A. GA Implementation

GA starts with encoding binary strings called chromosomes, and each chromosomal string contains 9 bits, and each bit has its significant meaning regarding the features of predictor: Starting two bits signifies the type of predictor: (00) for bi-modal, (01) for G-share, (11) for correlated predictor. Next, two bits mean the wake-up time of the particular predictor. The wake-up time is highest for bi-modal and lowest for the correlated predictor, therefore initially (00) is used for highest and (11) is for the lowest wake-up time. Least significant 5 bits show the probability difference of the last 8 T/NT branches, where 4 bits represent the binary equivalent of the integral value of probability difference and fifth bit is signed bit for the difference.

The next step in GA is to apply fitness function over the chromosomes to find most suitable predictor. Fitness function used in our algorithm is the number of ones in the chromosome, more the number of ones in the binary string, healthier the chromosome will be.

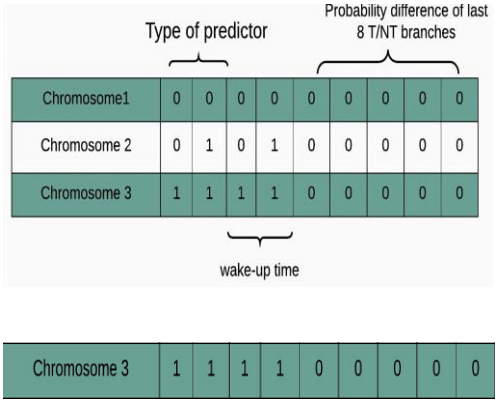


Fig.2. (a) Initial Populations (b) Winner Chromosome

Further, the algorithm proceeds by setting mutation probability (0.55) and mutation point (5th bit) as shown in Figure 6 (a).

Mutation point can be a single bit or multiple bits. This process is simple, which includes flipping the bit of the chromosome on variation point.

The next stage in GA is to copy the relevant bits from winner chromosome to rest of the chromosomes, and this process is called crossover. In our chromosome strings, relevant bits are the number of ones, so we do bitwise XORing of rest of the chromosome with winner chromosome. This step helps in copying the bits containing "1" present in recipient chromosome at the crossover points to rest of the offspring.

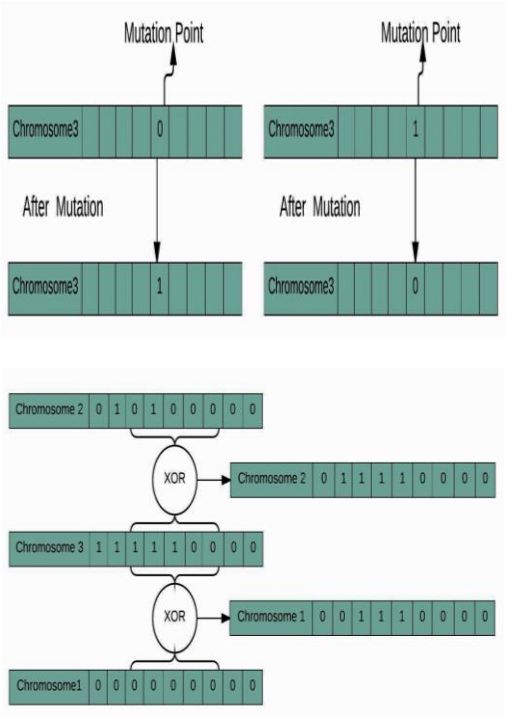


Fig.3. (a) Chromosomes before and after Mutation (b) Chromosome before and after crossover

After crossover, we get healthier generation. It can be observed from Figure 6 (b) that new chromosomal strings contain comparatively more number of ones, which means offspring of the new generation are healthier than the previous generation. This process is thus repeated in the next iteration, results in the production of new offspring from previous ones. It results in shifting to the fittest predictor, which will result in appropriate predictions every time. New population after the first iteration is shown in Figure 7.

Chromosome1	0	0	1	1	1	0	0	0	0
Chromosome 2	0	1	1	1	1	0	0	0	0
Chromosome 3	1	1	1	1	1	0	0	0	0

Fig.4. Chromosome After Applying GA

Further improvement in prediction mechanism is achieved by selecting the winner chromosome along with calculating the probability of last 8 taken/Not taken branches. This probability is useful in the cases where Winner chromosome fails to predict correctly. This failure happens when the state of 2-bit counter goes wrong in such situations prediction is considered to be the one, which is more probable for e.g. If the number of taken branches are more in last eight branches then next prediction is found to be taken or vice-versa. The probability difference, which gets calculated earlier and mapped into the last 5 bits of chromosomal string, is used in this case.

V. Results

The designed alloyed branch predictor with Genetic algorithm simulates performance analysis on MATLAB. A 16- Instruction RISC ISA (Instruction set architecture) of a processor simulated with an instruction memory of 1MB having instruction length of 16-bits. Two branch instructions included in this ISA: Branch if Zero and branch if negative. Only the fetch stage of a processor is simulated in MATLAB with the complete proposed predictor working in the fetch stage. This simulation includes over 0.5 million instructions of which about 25% instructions are branch one. The mechanism to flip branch direction is kept random it depends upon the branch transition probabilities we set. Setting the branch transition probabilities allows us to have a controlled flow of branch instructions. High branch transition probability means a more random flipping of branches, which ultimately creates a robust environment for our predictor.

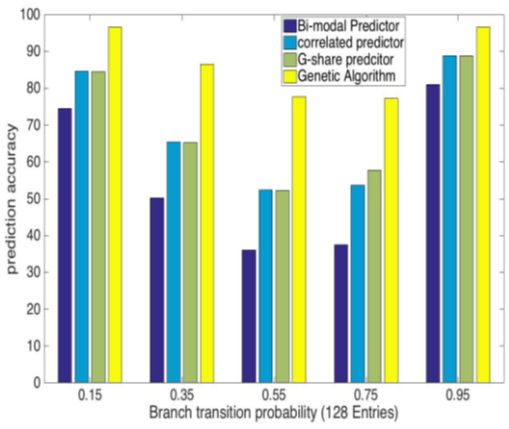
To test and compare GA with other kinds of branch predictors, we did a rigorous analysis of designed predictor on several benchmark programs. Each program has 0.5 million instructions in which more than 0.15 million instructions are branches. Statistically, every fourth instruction being a branch gives us an idea about the number of branches in a program. The branch resolving structure is kept just after the fetch stage to predict correct branch direction. The branch transition probability varies from 0.35 - 0.95 for each benchmark program. Different branch transition probabilities in

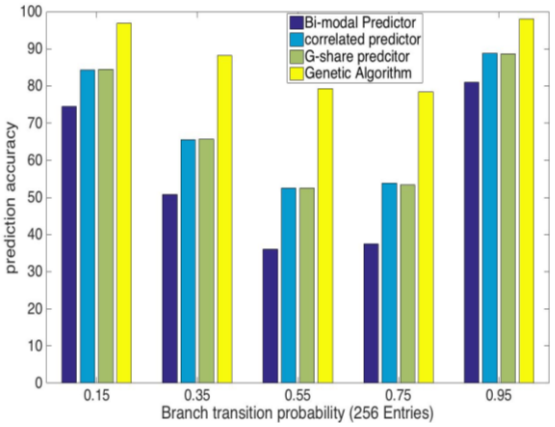
every benchmark program provide randomness in forward and backward branches. It means that larger the transition probability, greater is the flipping of branch directions. Branch resolving mechanism consists of three conventional predictors with genetic algorithm applied to it. To make the comparison more extensive we did various simulations with different branch transition probabilities. Each simulation with one value of transition probability tested for various history length of 128, 256, 512, 1024, 2048 and 4096 entries.

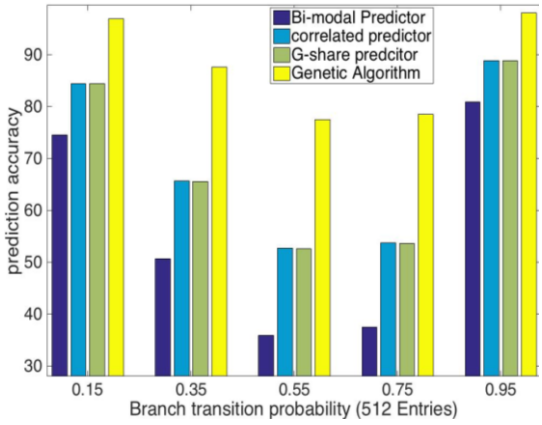
Observation from tabulated as well as graphical results explains that although our prediction based on GA is not attaining high accuracy for average branch transition probabilities like 0.35 to 0.75, it's raising with increasing history lengths and is ranging between 77%-89%. Furthermore, it is also not able to attain very high accuracy for low transition probability branches (0.15), but the prediction accuracy is consistent and near about 96%. On the other side, efficiency is as high as 98% for high transition probability branches (0.95) in significant history length. Our GA-based predictor is giving exceptional accuracy for high transition probability branches, and it is consistent for all history length.

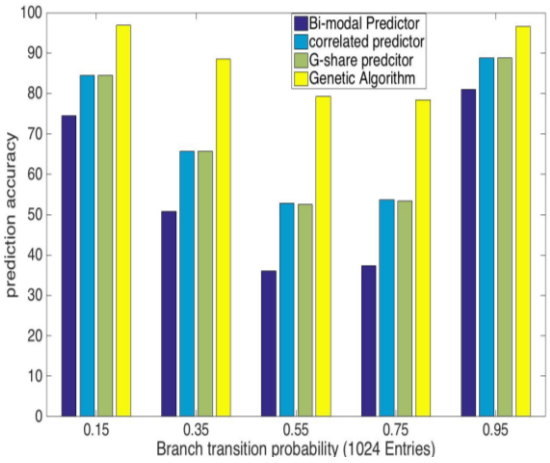
Our predictor can choose the best predictor out of three conventional branch predictors, which includes in the structure. This decision of which predictor is fittest is critical and should be done very carefully as the selected suitable predictor is going to generate the decision bits. Our GA-based predictor is also able to make its prediction whenever the chosen conventional predictor makes a wrong move. This prediction made by GA is correct almost all the time as this decision made by calculating the branch probability differences of taken and not-taken branches executed in the past.

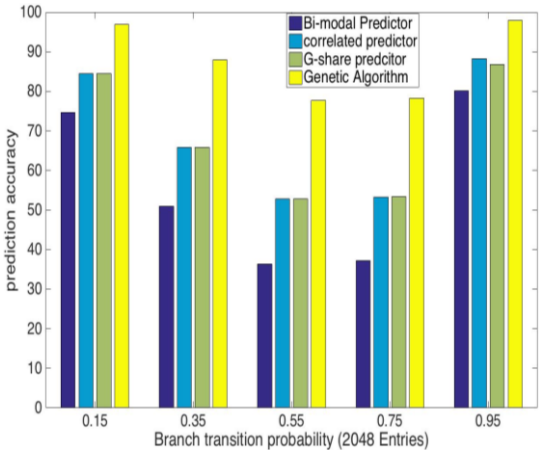
GA provides a considerable amount of increment in the prediction accuracies and can also perform well even if the history table is short. Moreover, we can see that during instruction fetch, branches that are usually encountered are either with very low transition probability (loop branches) or with very high transition probability (if-else branches). Higher accuracies in 0.15 and 0.95 branch transition probabilities make our branch-resolving structure more suitable for the environment where branch direction is randomly flipping at a very high rate and also in the environment where it is not flipping much.











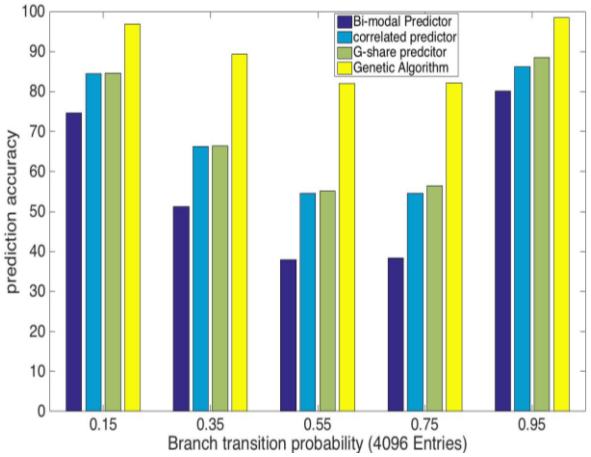


Fig.5. Prediction Accuracy for Different Predictors at Different History Length

VI. Conclusion

In this paper, we systematically studied the efficiency of GA in the context of alloyed branch prediction. Our idea of applying GA in choosing fit predictor among conventional branch predictors proved to be a source of better prediction mechanism. It is evident from our results that applying GA can improve the prediction mechanism and gives a considerable amount of increment in prediction accuracy for different branch transition probabilities. This increment is more than 10% in few cases of branch transition probabilities such as 0.35, 0.55 and 0.75, whereas it's more than 20% for 0.15 and 0.95-branch transition probabilities. Moreover, the substantial increment in accuracy can be achieved even for small history length whereas for large history length of 4096 entries; accuracy reaches as high as 98%. This makes our overall prediction mechanism more robust and reliable in conditions where branch directions are continuously flipping as seen in the case of if-else branches and also in conditions where branches are flipping at a very slow rate as seen in loop branches.

References

[1] Huang, Mingkai, Dan He, Xianhua Liu, Mingxing Tan, and Xu Cheng. "An Energy-Efficient Branch Prediction with Grouped Global History". In Parallel Processing (ICPP), 2015 44th International Conference on, pp. 140-149. IEEE, 2015.

[2] Wu, Di. "High Performance Branch Predictors for Soft Processors." PhD diss., University of Toronto, 2014.

[3] Lu, Z., Lach, J., Stan, M. R. & Skadron, K. "Alloyed branch history: Combining global and local branch history for robust performance". International Journal of Parallel programming, 31(2), pp.137-177, 2003

[4] Eden, Avinoam Nomik. "Of limits and myths in branch prediction." PhD diss., University of Michigan, 2001.

[5] Jiménez, Daniel A., and Calvin Lin. "Dynamic branch prediction with perceptrons." In High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on, pp. 197-206. IEEE, 2001.

[6] Skadron, Kevin, Margaret Martonosi, and Douglas W. Clark. "A taxonomy of branch mispredictions, and alloyed prediction as a robust solution to wrong-history mispredictions." Parallel Architectures and Compilation Techniques, 2000. Proceedings. International Conference on. IEEE, 2000.

[7] Lee, Chih-Chieh, I-CK Chen, and Trevor N. Mudge. "The bi-mode branch predictor." In Microarchitecture, 1997. Proceedings. Thirtieth Annual IEEE/ACM International Symposium on, pp. 4-13. IEEE, 1997.

[8] Emer, Joel, and Nikolas Gloy. "A language for describing predictors and its application to automatic synthesis." In ACM SIGARCH Computer Architecture News, vol. 25(2), pp. 304-314. ACM, 1997.

[9] McFarling, Scott. Combining branch predictors. Vol. 49. Technical Report TN-36, Digital Western Research Laboratory, 1993.

[10] Porter, Leo, and Dean M. Tullsen. "Creating artificial global history to improve branch prediction accuracy." In Proceedings of the 23rd international conference on Supercomputing, pp. 266-275. ACM, 2009.

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

[12] Mudge, T., Lee, C. C., & Sechrest, S. "Correlation and aliasing in dynamic branch predictors". In *Computer Architecture, 1996 23rd Annual International Symposium on* pp. 22-22, IEEE. May 1996

[13] Lee, C. C., Chen, I. C., & Mudge, T. N. "The bi-mode branch predictor". In *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on* pp. 4-13. IEEE, December 1997

[14] Thomas, R., Franklin, M., Wilkerson, C., & Stark, J. "Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history". In *ACM SIGARCH Computer Architecture News* Vol. 31(2), pp. 314-323. ACM, June 2003

[15] Chang, P. Y., Evers, M., & Patt, Y. N. "Improving branch prediction accuracy by reducing pattern history table interference". *International journal of parallel programming*, *25*(5), pp.339-362, 1997

[16] Eden, A. N., & Mudge, T. "The YAGS branch prediction scheme". In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture* pp. 69-77, November 1998

[17] Yeh, T. Y., Marr, D. T., & Patt, Y. N. "Increasing the instruction fetch rate via multiple branch prediction and a branch address cache". In *ACM International Conference on Supercomputing 25th Anniversary Volume* pp. 183-192, June 2014

[18] Evers, M., Chang, P. Y., & Patt, Y. N. "Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches". In *ACM SIGARCH Computer Architecture News* Vol. 24(2), pp. 3-11, May 1996

SUMMARY OF RESULTS

Table 1. Prediction Accuracy (%) when Branch Transition Probability is 0.15

Type of predictor	128 Entries	256 Entries	512 Entries	1024 Entries	2048 Entries	4096 Entries
Bi-modal	74.7332	74.5448	74.5563	74.5511	74.6037	74.7038
Correlated	84.5975	84.3679	84.4228	84.4531	84.4303	84.4414
G-share	84.4102	84.4034	84.4354	84.4142	84.4714	84.5864
Genetic algorithm	96.5381	96.9031	96.8808	96.9300	96.9128	96.9225

Table 2. Prediction Accuracy (%) when Branch Transition Probability is 0.35

Type of predictor	128 Entries	256 Entries	512 Entries	1024 Entries	2048 Entries	4096 Entries
Bi-modal	50.2328	50.7558	50.7404	50.7338	50.8696	51.2456
Correlated	65.4424	65.5290	65.6538	65.7279	65.7691	66.3231
G-share	65.2612	65.7008	65.5584	65.7390	65.7595	66.4679
Genetic algorithm	86.5602	88.2742	87.6113	88.4593	87.8712	89.4123

Table 3. Prediction Accuracy (%) when Branch Transition Probability is 0.55

Type of predictor	128 Entries	256 Entries	512 Entries	1024 Entries	2048 Entries	4096 Entries
Bi-modal	36.0330	36.0663	35.9496	36.0571	36.2910	37.9508
Correlated	52.3182	52.5252	52.6824	52.8677	52.8014	54.4699
G-share	52.2722	52.5704	52.6413	52.5481	52.8517	55.1713
Genetic algorithm	77.6711	79.3096	77.4923	79.3159	77.6484	81.9421

Table 4. Prediction Accuracy (%) when Branch Transition Probability is 0.75

Type of predictor	128 Entries	256 Entries	512 Entries	1024 Entries	2048 Entries	4096 Entries
Bi-modal	37.4525	37.4347	37.5521	37.3746	37.1817	38.3432
Correlated	53.7087	53.8850	53.7928	53.6989	53.2564	54.5643
G-share	57.7247	53.4613	53.6096	53.4522	53.4178	56.3654
Genetic algorithm	77.2412	78.3468	78.4986	78.3978	78.2489	82.2012

Table 5. Prediction Accuracy (%) when Branch Transition Probability is 0.95

Type of predictor	128 Entries	256 Entries	512 Entries	1024 Entries	2048 Entries	4096 Entries
Bi-modal	81.0605	80.9782	80.8621	80.6099	80.1575	80.1775
Correlated	88.7983	88.7320	88.8631	88.4700	88.1646	86.2334
G-share	88.8023	88.6610	88.2921	87.9359	86.8487	88.4656
Genetic algorithm	96.5824	98.1196	98.0172	97.8342	97.8487	98.5313

Authors' Profiles

Shivam Goyal is currently an M.Sc. candidate in the Department of Electronics and Communication Engineering at JECRC University at Jaipur (India). He did his B.Tech in Electronics and Communication Engineering from JECRC UDML College of engineering, Jaipur, India. His research interests include high-performance computing and Microarchitecture.

Jaskirat Singh is an Assistant Professor – II in the Department of Electronics and Communication Engineering, JECRC University, Jaipur (India). He did his B.Tech in Electronics and Communication Engineering from Sikkim Manipal Institute of Technology, Sikkim Manipal University, Sikkim (India). He did his MSc. Engg. In Electrical and Computer Engineering from Lakehead University, Canada. His research interests include Supervised, Reinforcement and Deep Learning.

How to cite this paper: Shivam Goyal, Jaskirat Singh,"Two-Level Alloyed Branch Predictor based on Genetic Algorithm for Deep Pipelining Processors", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.5, pp. 27-33, 2017.DOI: 10.5815/ijmecs.2017.05.04

