A mechanism for mitigation of branch prediction calculation latency

Uroš Radenković[*], Marko Mićović[*] and Zaharije Radivojević[*]

[*] University of Belgrade, School of Electrical Engineering, Belgrade, Serbia

{uki, micko, zaki}@etf.bg.ac.rs

***Abstract*— Branch predictors are one of the most significant components in the contemporary pipelined processors. They are used to predict which instruction should be executed right after a branch instruction. Prediction itself must be made before completion of the branch instruction. This requirement makes an accuracy the main characteristic of a branch predictor. Accuracy of a branch predictor naturally must be very high to avoid wasted processor cycles while executing instructions which are not part of the correct execution path. Additionally, in case of a misprediction processor must engage recovery procedure in order to return back to the correct execution path which reduces performance of the processor. Branch predictors use various algorithms and structures to calculate a prediction consuming certain time in the process. This time is called a branch predictor latency. There are two types of branch predictor latency. First type is time needed for prediction calculation itself and second type is time needed to update internal state of branch predictors. This paper focuses on time needed for prediction calculation and ways this type of branch predictor latency could be mitigated. There is a proposition of a mechanism that initiates prediction calculation in advance i.e. before branch instruction has even been fetched for execution by the processor. Proposed mechanism consists of two different branch predictors and internal structures for information buffering. First branch predictor is slow in terms of time needed for prediction calculation and has very high accuracy whilst second branch predictor is fast but less accurate. Depending on the execution traces used for testing proposed mechanism achieves accuracy from 76 to 99 percent.**

Index terms: branch predictor, prediction calculation latency, pipelined processor

I. INTRODUCTION

Pipelined processors use various branch predictors which consume a different amount of time for prediction calculation. If branch predictor is more accurate it more than likely consumes more time for the prediction calculation. Having in mind the high frequency of branch instructions in program execution there is a need for a mechanism that mitigates branch predictor calculation latency.

Branch predictors use various algorithms and structures to calculate a prediction consuming a certain time in the process. This time is called a branch predictor latency. There are two types of branch predictor latency. The first type is the time needed for prediction calculation itself and the second type is the time needed to update the internal state of branch predictors.

This paper proposes a new mechanism which should be able to mitigate branch predictor calculation latency. Based on the survey of existing mechanisms presented in papers [1, 2] proposed mechanism consists of two branch predictors. Main difference between the proposed mechanism and existing mechanisms mentioned above is in the initiation moment of prediction calculation. Existing mechanisms initiate prediction calculation when branch instruction has been fetched for execution by the processor. Proposed mechanism initiates prediction calculation even before fetching of a branch instruction. Similar to the proposed mechanism is the mechanism presented in paper [3], but it uses a different algorithm to determine initiation moment of prediction calculation.

This paper is organized as follows: The second section describes how the proposed mechanism was developed. The third section contains details of the proposed mechanism. The fourth section describes the simulation used for testing. The fifth section presents the test results and discusses achieved results. The sixth section is the conclusion and contains some directions for future work.

II. METHODOLOGY

As a first step, a software system for branch predictor simulation based on execution traces has been developed in order to evaluate the accuracy of different existing branch predictors [4]. The software system incorporates the implementation of ten different branch predictors which are widely known and whose details are available in the literature [5]. These implemented branch predictors have been tested using various execution traces.

In the second step, two branch predictors were chosen based on their accuracy. First branch predictor had to be slow in terms of time needed for prediction calculation and very accurate whilst the second branch predictor had to be fast but less accurate. TAGE predictor [6] which is very accurate was chosen as a first branch predictor and Bimodal predictor [7] as a second branch predictor. Bimodal predictor was chosen based on the assumption that it takes only one cycle for prediction calculation. New mechanism for mitigation of branch prediction calculation latency is proposed which uses these two previously chosen branch predictors.

In the end, the proposed mechanism was tested using several execution traces. Trace-driven simulation was used with an accuracy as the main characteristic which was measured.

III. DETAILS OF PROPOSED MECHANISM

This section contains details of the proposed mechanism. All parts of the proposed mechanism are described and the ways that they are connected. Also, the data that the proposed mechanism uses for calculation prediction are described.

A. *Main Idea of Proposed Mechanism*

Proposed mechanism for mitigation of branch prediction calculation latency consists of two branch predictors and three buffers. First branch predictor (BP1) is slow but very accurate. Second branch predictor (BP2) is fast but less accurate than the first one. The working principle of the proposed mechanism is that the prediction calculation by the BP1 is initiated even before branch instruction has been fetched for execution by the processor. The purpose of the BP2 is to calculate a prediction for every branch instruction. Prediction calculated by the BP2 is used when the BP1 does not calculate prediction on time or does not calculate prediction for that particular branch instruction.

B. *Working Principle of Proposed Mechanism*

The problem is how to determine when prediction calculation by BP1 should be initiated for the given branch instruction. There is no clear way of knowing when that branch instruction will be fetched for execution by the processor. For that reason, small buffer called Finished Branch Instructions Buffer (FBIB) stores information about most recently finished branch instructions. FBIB stores pairs which consist of the branch instruction address (BIA) and preceding instruction address (PIA). Preceding instruction is an instruction which has been executed *N* cycles before the branch instruction. The number of cycles *N* represents the time which is necessary for the BP1 to finish prediction calculation where *N* is the same for every branch instruction. When the instruction from PIA has been fetched again by the processor then FBIB is accessed using PIA as a key in order to get (PIA, BIA) pair. Afterwards, prediction calculation by the BP1 is initiated for the branch instruction from BIA previously read from FBIB. At Figure 1. is shown principle schema of the proposed mechanism.

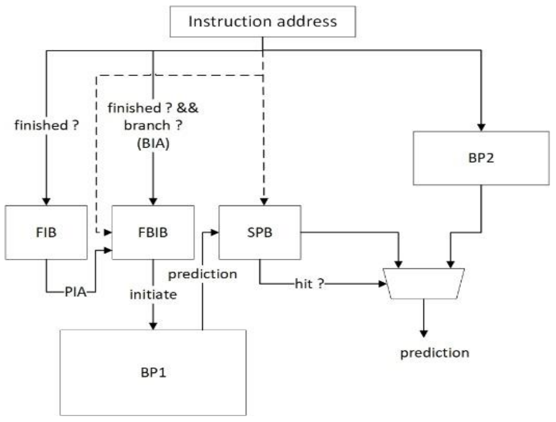

Figure 1. Principle schema of mechanism for mitigation of branch predictor calculation latency

Second small buffer called Finished Instructions Buffer (FIB) stores information about the most recently finished instructions (including all types of instruction). FIB is used when a new (PIA, BIA) pair should be inserted into FBIB in order to acquire PIA.

Third small buffer called Small Prediction Buffer (SPB) is a cache memory which stores predictions calculated by the BP1. If there is a hit in SPB, when branch instruction has been fetched for execution by the processor, the prediction is read from SPB. In case there is a miss in SPB the BP2 will provide a prediction.

C. *Details of FBIB, FIB and SPB*

As mentioned above, FBIB stores pairs which consist of the branch instruction address (BIA) and preceding instruction address (PIA) as elements. FBIB uses a modified FIFO replacement policy. When a new pair (NEW_PIA, NEW_BIA) should be inserted into FBIB two checks must be performed. The first check determines whether FBIB already contains a pair whose BIA element is equal to NEW_BIA. If FBIB contains such a pair, insertion of the new pair is omitted. The second check determines whether FBIB already contains a pair whose PIA element is equal to NEW_PIA. If such a pair exists in FBIB, then the BIA element of that pair is set to the NEW_BIA. Also, in this case, insertion of the new pair is omitted. Only when both checks are successfully performed new pair is inserted into FBIB using FIFO policy. This replacement policy ensures that there are no two pairs in FBIB with same BIA or PIA elements. Therefore, branch instruction, for which prediction calculation by the BP1 should be initiated, can be unambiguously determined.

FIB is a buffer which stores addresses of the most recently finished instructions. It stores information about all types of instruction. It uses FIFO replacement policy and may contain multiple identical addresses. Its minimum size is *N*, where *N* is the number of cycles necessary for the BP1 to calculate prediction.

SPB is a fully associative cache memory. Cache lines consist of a valid bit, tag index and prediction which is calculated by BP1 (prediction is the data of this cache). An address of branch instruction is used as a tag index. The valid bit tells whether the prediction from the cache line has been previously used. The prediction from SPB can be used only if the valid bit is set and the valid bit must be cleared when prediction from SPB is used. This means that any particular prediction from SPB can be used only once.

IV. IMPLEMENTATION OF SIMULATION

Trace-driven simulation was used to measure the accuracy of the proposed mechanism. The software system for simulation was implemented in Java programming language. It consists of ten different branch predictors that are implemented based on details that can be found in the literature.

The software system for simulation reads branch instructions one by one from program trace which is held in the textual file. For each branch instruction read, the proposed mechanism calculates prediction. After that the proposed mechanism updates its internal state based on prediction and true outcome of the branch which is read also from the program trace. The time needed to update the internal state is not simulated and the update is performed immediately after prediction calculation.

During the execution of the simulation, the software system collects information into textual output file. Information which are collected include:

- Number of correct predictions (hits)
- Number of wrong predictions (misses)
- Number of all executed branch instructions
- Percent of accuracy (hits/(hits + misses))
- Average distance between successive branch instructions
- Duration of simulation

V. Results

This section describes program traces that were used for testing. Also, the diagrams that represent accuracy of the proposed mechanism are presented in this section and at the end of this section comments on the results are given.

A. *Program traces*

Program traces, that were used for testing the proposed mechanism, are subsets of SPEC CPU2006 benchmarks [8]. Program traces consist of 10M to 100M instructions of which 1M to 17M are branch instructions. Every row in program trace contains a record that describes one microoperation. Following pieces of information from records were used for testing:

- Address of branch instruction
- Target branch address
- Branch outcome

In every cycle during testing one instruction is fetched for execution. From the moment when it is initiated to calculate prediction for some given branch instruction, BP1 is busy for next *N* cycles and cannot be initiated again for any other branch instruction that could be fetched afterwards. Number of branch instructions and average distance between adjacent branch instructions are shown in Table I. Average distance is average number of instructions that were fetched between adjacent branch instructions.

TABLE I.
NUMBER OF BRANCH INSTRUCTION AND
DISTANCE BETWEEN ADJACENT BRANCH INSTRUCTIONS

Trace	Num. of branch	Distance
art-100M	15061011	5,64
gcc-10M	1867817	4,35
gcc-50M	9293598	4,38
go-100M	16371394	5,11
hmmer-100M	11993628	7,34
libquantum-100M	16719565	4,98
mcf-100M	19518883	4,12
sjeng-100M	17102954	4,85
sphinx3-100M	17826992	4,61

B. *Accuracy of the proposed mechanism*

As previously mentioned, TAGE predictor was chosen as the BP1. It uses 128KB of space for its internal structures. Based on its size and complexity the assumption is that it takes *N* cycles for prediction calculation. Bimodal predictor was used as the BP2. It only uses about 2KB of space because the assumption is that it takes only one cycle for prediction calculation. It is fast and can finish the calculation of prediction in one cycle.

FIB has *N* entries where single entry keeps the address of the instruction. FBIB contains 16 entries where the size of the entry is twice the length of the address of the instruction, because FBIB stores (PIA, BIA) pairs. SPB is fully associative cache memory with 16 entries. Single entry (cache line) stores valid bit, one bit for prediction and tag bits (address of the branch instruction).

Number of cycles *N* that represents branch prediction calculation latency for the BP1 was varied between values from 2 to 9 in order to simulate different BP1 branch prediction calculation latency. The results were better when *N* had smaller values. At the Fig 2., Fig. 3. and Fig. 4. are shown diagrams that represent accuracy of the proposed mechanism when *N* is 2, 4 and 7, respectively, for the various execution traces. Depending on the execution traces used for testing proposed mechanism achieves accuracy from 76 to 99 percent. Diagrams show accuracy when only BP2 calculate prediction and when proposed mechanism (BP1 and BP2) calculate prediction. In most of the cases proposed mechanism (BP1 and BP2) achieved better results. Proposed mechanism (BP1 and BP2) achieves worse results than alone usage of BP1 only if BP1 is able to calculate prediction within single cycle (which is naturally expected).

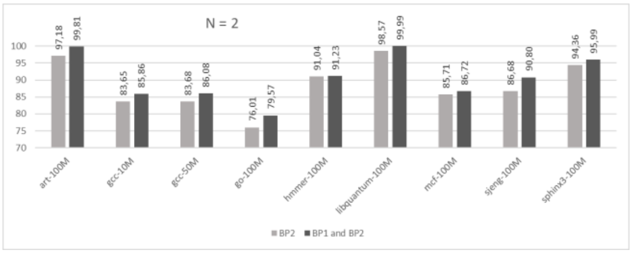

Figure 2. Accuracy when branch prediction calculation latency of BP1 is 2 (N=2)

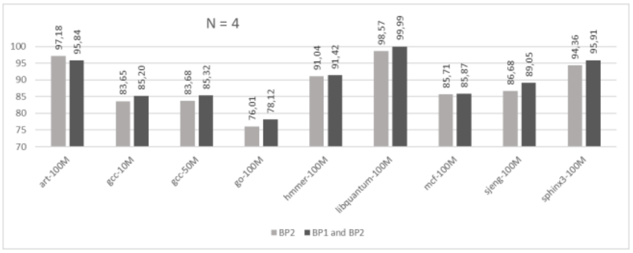

Figure 3. Accuracy when branch prediction calculation latency of BP1 is 4 (N=4)

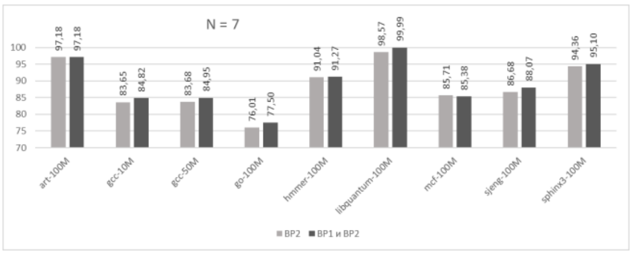

Figure 4. Accuracy when branch prediction calculation latency of BP1 is 7 (N=7)

C. *Comments on the results*

Several things influence the accuracy of the proposed mechanism and they were noticed during the testing phase. One of them is that BP1 calculates the prediction only for a subset of branch instructions. Also, update of the internal state of BP1 is performed only if the final prediction is calculated by BP1 (if there is hit in SPB). Because of that, BP1 cannot track the history of all branch instructions. BP1 calculates ahead the prediction based on current internal state. In the meantime, while BP1 is calculating the prediction, other branch instructions can be executed. BP1 does not have information about those instructions and it does not have a complete global history of branch instructions. Correlation between branch instructions is not tracked properly and it may affect the accuracy of BP1.

BP1 will not be able to calculate the prediction every time for some branch instruction, because BP1 may be involved in the calculation of prediction for other branch instruction. For example, prediction for some branch instruction may be calculated first time by BP1, the second time by BP2 and again by BP1. In this scenario, BP1 will not have a complete local history of execution for that branch instruction, because BP1 performs updating of internal state only when it calculated the prediction for that branch instruction.

The proposed mechanism is tested with values 2 to 9 for *N*. With values above 9 for *N*, the proposed mechanism was not tested, because those values are about two times greater than the average distance between two adjacent branch instructions in used program traces. The greatest accuracy the proposed mechanism achieved when *N* had smaller values (2 and 3). The proposed mechanism achieved greater accuracy than alone BP2 on the most program traces. On some program traces the accuracy is greater for about 3%. As mentioned above, the accuracy is worse than the accuracy achieved by BP1 (TAGE predictor) in one cycle.

VI. Conclusion and future work

This paper proposes the mechanism for mitigation of branch prediction calculation latency, describes the trace-driven simulation of the mechanism and shows results of testing. The software system for branch predictor simulation has been developed and it is used for testing the proposed mechanism. The main characteristic which was measured is the accuracy. At the end, this paper gives comments on the achieved results and things that influence the results.

The proposed mechanism only focuses to hide the time needed for prediction calculation, whilst the time needed to update the internal state of the branch predictor is ignored. The proposed mechanism may be upgraded to include the effect of time needed to update the internal state.

Acknowledgment

Research presented in this paper was supported by Ministry of Education, Science and Technological Development of Republic of Serbia, project III44009.

References

[1] Loh, Gabriel H. "Revisiting the performance impact of branch predictor latencies." *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on*. IEEE, 2006.

[2] Jin, Wenbing, et al. "A novel architecture for ahead branch prediction." *Frontiers of Computer Science* 7.6 (2013): 914-923.

[3] Jiménez, Daniel A., Stephen W. Keckler, and Calvin Lin. "The impact of delay on the design of branch predictors." *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*. ACM, 2000.

[4] U. Radenkovic, M. Micovic, F. Hadzic, Z. Radivojevic "A tool for testing branch predictors" Proceedings of 25th conference "YU INFO 2019", pp. 77-81, Serbian Informatics Society, Kopaonik, Serbia, March, 2019.

[5] Mittal, Sparsh. "A Survey of Techniques for Dynamic Branch Prediction." arXiv preprint arXiv:1804.00261 (2018).

[6] Seznec, André and Pierre Michaud. "A case for (partially) TAgged GEometric history length branch prediction. " *J. Instruction-Level Parallelism* 8 (2006): n. pag.

[7] Lee, Chih-Chieh, I-Cheng K. Chen, and Trevor N. Mudge. "The bi-mode branch predictor." Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture. IEEE Computer Society, 1997.

[8] Computer Architecture, Computer and Informatics Science University of Pennsylvania (https://www.cis.upenn.edu/~milom/cis501-Fall12/traces/trace-format.html, 15.08.2019.)