A Novel Approach for Branch Buffer Consuming Power Reduction

Behzad Zamani[1,2], Ehsan Adeli[2], Haleh Gharedaghi[2], Mohsen Soryani[2]
[1]*Engineering Department, Islamic Azad University, ShahreKord Branch*
[2]*Computer Department, Iran University of Science and Technology*
{bzamani,eadeli,gharedaghi,soryani}@iust.ac.ir

Abstract

By increasing the pipeline length in processors, the accuracy of the branch predictor unit plays an important role in processors efficiencies. In addition to the accuracy, consuming power is also an essential in portable systems. Therefore, in the processors these days the prediction unit is used to determine the branch destination, while most of these accesses are not necessary. In this paper, a method is proposed to reduce the consuming power in the jump prediction unit. In this proposed method, the non-necessary accesses to BTB are reduced by taking into account this fact that there exists distances between different consecutive branch instructions. This method decides the access to BTB by a constant value and a counter. After an instruction entrance, the BTB is accessed if the counter is zero, and if the instruction is a branch instruction and exists in the BTB the counter is reset. The simulation and experimental results illustrate the suitable performance of the proposed method in comparisons to the other methods. This superiority is for both the execution time and for the consuming power. Also it is more strengthened by increasing the distance.

1. Introduction

The accuracy of the branch predictor unit gets more important by increasing the pipeline length [1]. In the architecture of the modern processors, other than accuracy, the power consumption should be taken into precise account in the branch predictor unit. This is because up to 10% of the total consuming power of the processor is consumed in this part. [2] Synchronizes access to the cache memory of Branch Target Buffer (BTB) and Prediction History Table (PHT) to increase the processor's clock rate in the fetch step. It is obvious that when accessing BTB and PHT it is not known whether the instruction being fetched is a control instruction or not. As a result, for all the instructions being fetched from the cache memory an access to the branch predictor unit (BTB and PHT) is done and therefore a significant power is being wasted in this unit. This is because accesses to the branch predictor unit for non-control instructions are not necessary. In other words, less than 13% of the program instructions are for control purposes [2, 3], but for all the instructions in the fetch step an access to the branch predictor unit needs to be done [3, 4].

Note that all the predictions of the branch predictor unit are not used for fetching the next instruction. Only the predictions that make a collision to the BTB table and the PHT table prediction predict a branch would be used for next instruction fetch. So solutions should be found to reduce non-necessary accesses within possible limits, to enhance the performance of the branch predictor unit. Also a portion of the information stored in the BTB is redundant; eliminating this information from the BTB makes its size smaller and lessens the power consumption of the unit, without affecting its performance.

In this paper, an approach is proposed to reduce the number of accesses to the BTB unit. This approach assumes that between each two successive branch instructions there exist a number of non-branch instructions. To reduce the non-necessary accesses to the BTB a constant value is used for determining whether to access or not. The proposed approach does not impose any static power to the system due to the need to no extra memory and just by adding a 5-bit counter to the branch predictor unit can reduce the consuming power.

The paper is organized as follows: in section 2 a literature review of the works done in the area of branch predictor unit consuming power reduction is brought. Section 3 the proposed approach is discussed and after that in section 4 demonstrates the simulation results and evaluation process of the proposed system. Finally the conclusion is given.

2. Related Works

Reducing the consuming power of the processors has been greatly of interest in the recent decades. This reduction can be done in the bus, cache or branch predictor units. In this section, some of the papers recently published on how to reduce the consuming power in the branch predictor unit are reviewed.

Skadron et al. in 2000 presented a method for static consuming power reduction by decreasing the BTB unit size [5]. In the target field of the BTB the target addresses of the control instructions are placed, while most of the control instructions are internal braches in a single program and less than 10% are external branch instructions (function call)[4]. Therefore, incorporating this fact, they have divided the BTB into two parts: one is for the internal branches, in which the target addresses are stored relatively, and the other that keeps the target addresses completely. This approach could save almost 25% from the BTB volume.

Petrov and Orailoglu in 2003 proposed a method called Application Customized BTB (ACBTB) [6]. They used a static controller to obtain the critical points and the distance between two successive branches, which are stored in the ACBTB hardware table during the execution. The processor accesses this table in the program critical sections to make decisions.

Monchiero et al. in 2004 used a compiler-based method to reduce the accesses to the BTB unit for the VLIW processors. The compiler scans the code and informs the fetcher whether to access the BTB or not. In other words, the fetcher does not access the BTB unless the compiler lets it to [7].

Parikh et al. in 2004 presented a method named banking [2]. They have divided the BTB and PHT into several parts, in anytime only one of the parts is active. This leads to a less dynamic power usage of the branch predictor unit. The banking algorithm needs an extra decoder to determine the active bank, also bank prediction and disconnecting and connecting the banks are themselves problems. They have also proposed a method to dynamically reduce the accesses to the BTB; it avoids the access to the BTB if there is a non-branch instruction on the cache memory line [2].

Hu et al. in 2005 presented the idea to use Next Branch Distance (NBD) to reduce the accesses to the BTB [3]. In fact, a new section called NBD is added to each entry in the BTB. NBD is an 8-bit number which stores the distance of the branch target from the first control instruction after that. As far as the distance between two consecutive branch instructions does not exceed 50 instructions [2], 8 bits for NBD would be a right choice.

Regularly BTB is updated when a control instruction is reached which causes a branch and does not exist in the BTB. The updating process occurs when the branch instruction is on one of the pipeline steps. But to calculate the NBD, BTB updating should be postponed until the next control instruction is reached. This is because in order to calculate the NBD, we need the address to the first control instruction after the target, so that by subtracting the target address from that NBD is achieved. As this method adds a NDB field to each BTB entry, it increases the size of the BTB and as a result the static power usage is increased.

Furthermore, they have presented a static approach to decrease the accesses to the BTB [3]. Taking into account a constant number, n, the decision whether to access BTB or not is made. This means that they set a counter, and do not refer to the BTB until it equals zero. While the counter is not still zero, it is possible that a branch instruction be reached. This method pays the cost in such cases [3]. In the following an approach is proposed that modifies the Hu et al. method to enhance the performance.

3. Proposed Method

Hu et al. assumed a constant value n to make decisions on accessing BTB. This means that they have assumed that until the next branch instruction there are n non-branch instructions. This algorithm is as what follows [3]:

```
if (counter > 0) then
        counter = counter – 1
else
          access BTB
          if (BTB hits) then
                    counter  = n
          end
end
```

The counter is set when the branch instruction exists in the BTB. In this method the value n affects the consuming power reduction rate. In fact, smaller n reduces less the accesses to the BTB than a larger n, instead searches more instructions in the BTB and is faster. In this method, it is possible that before decreasing the counter to zero, a branch instruction would be reached which increases the execution time by 3 cycles for each instruction.

In the proposed approach in this paper, the aim is to reduce these branches. When the branch instruction is in the BTB and the branch is taken, the counter is set. The proposed algorithm would be of the following form:

```
if (counter > 0) then
        counter = counter – 1
else
          access BTB
          if (BTB hits and branch is taken) then
                    counter  = n
          end
```

```
end
```

It is obvious that the number of the times that the above condition is executed is less than the number of the previous one. So, it is expected that the number of accesses to the BTB be reduced, on the other hand it also reduces the number of the branches before the counter goes zero.

In the Hu et al. method and the proposed method, the determination of the value n is very important. The effectiveness of the method is strongly dependent in this value. For different benchmarks there are different scatters for the branches, therefore a unified behavior for all the benchmarks is not suitable. It is better to change the value of n during the execution of the benchmarks, and adapt it with the structure and the procedure of the branches. When the counter reaches zero, without the entrance of any branch instruction an access to the BTB is made, which is a non-necessary access. To enhance the proposed approach n could be increased. On the other hand, when the counter has not still got zero and a branch instruction is entered, the method decreases n, as a penalty. The modified Hu algorithm could be found as the following:

```
if (counter > 0) then
        counter = counter – 1
          if (instrument is branch) then
                  n  = n - 1
          end
else
          access BTB
          if (instrument isn't branch) then
                  n  = n + 1
          end
          if (BTB hits) then
                  counter  = n
          end
end
```

And the modified proposed algorithm:

```
if (counter > 0) then
        counter = counter – 1
          if (instrument is branch) then
                  n  = n - 1
          end
else
          access BTB
          if (instrument isn't branch) then
                  n  = n + 1
          end
          if (BTB hits and branch is taken) then
                  counter  = n
          end
end
```

In the above method the initial value of n is not very important, so we start with any value of n, the performance would not so differ.

4. Simulations and Results Evaluation

To calculate the measure of power reduction Simwattch and Simplescalar simulators are used and Spec95 benchmarks are taken into use. The configuration of the branch predictor unit, used in these simulations is as in table 1.

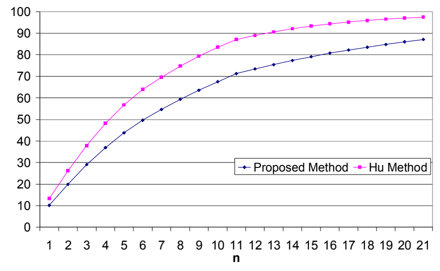
Table 1. The BPU configuration

Branch predictor unit type	bimod
The predictor table size	2048
Size and the number of the sets of the BTB table	512×4

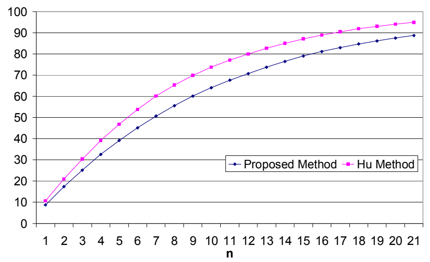
The power of the branch predictor unit in each cycle is almost 4.5231 which are 6.27% of the total chip power. Also the BTB power in the branch predictor unit is of size 4.16837 in each cycle, which this is almost 92.16% of the branch predictor unit consuming power.

The evaluation is done on the Go and Anagram benchmarks. Figure 1 shows the reduction rate of the accesses to the BTB for different value of n. As what the diagram says, when n gets larger the number of accesses falls smaller. In fact, as also mentioned before, the time interval of accesses to BTB is controlled by n, where the larger *n* gets the more the number of accesses reduce. For the two benchmarks Hu method has better performance in reducing the accesses to the BTB, where this difference reaches 15% in some portions. For the two Go and Anagram benchmarks after *n* almost 10, changes get slower.

In the above two methods, since it is possible there is the chance that before n reaches zero a branch occurs, the systems need to pay the cost. This cost is a time delay of three steps in the pipeline and an increase of the consuming power. Therefore, the less the number of branches while n is not zero, the less the consuming power would be. What's more, the execution time of the processes would be less.

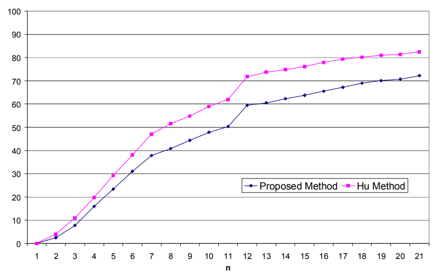


a) Anagram benchmark

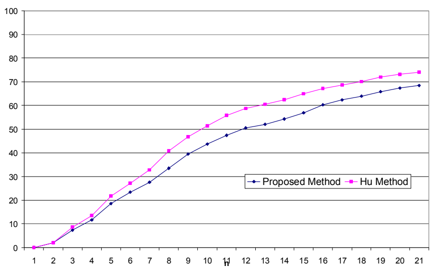


b) Go benchmark

Figure 1. The percentage of the reduction of the accesses to BTB related to the total number of the instructions



a) Anagram benchmark

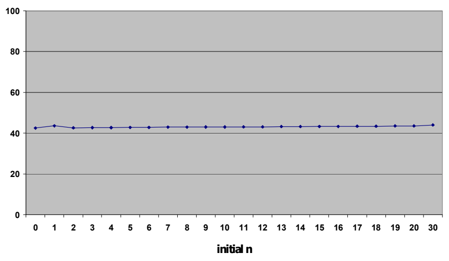


b) Go benchmark

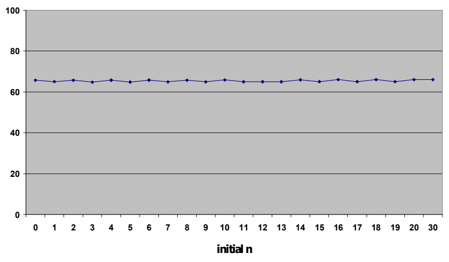
Figure 2. The rate of the number of branches happened in the time interval that n is not zero to the total number of the branch instructions

Figure 2 shows the number of these branches in different values of n. As could be seen in the figure, the number of branch instructions for the proposed approach relative to the one for the Hu method is less. The reason is that in the proposed approach the re-initialization condition occurs less often compared to the same condition in the Hu et al. [3] method. Because if the 'BTB hits and branch is taken' condition occurs, for sure the 'BTB hit' condition also occurs, while the vice versa is not necessarily satisfied. Also after a specific number of n the level difference of the two curves is almost constant. This value of n is different regarding the benchmark type.

Figure 4 shows the consuming power for the two above method in each execution cycle for different values of n. As also obvious in the figure, the proposed approach has less consuming power. As n increases the reduction in the proposed approach get more comparing to the Hu approach. According to the figure, after the values of almost 10 for *n*, the changes in the consuming power decreases, and the execution time increases. Therefore, regarding the execution time and the consuming power the value of n should be chosen correctly. Meanwhile, one can draw histograms for the program using the compiler and initialize n before the program starts



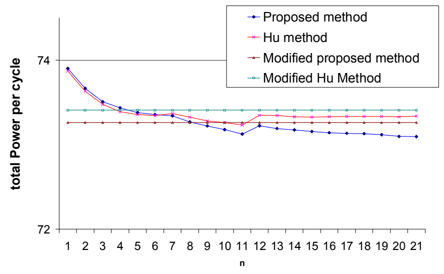
a) The percentage of the branch instructions not accessing the BTB



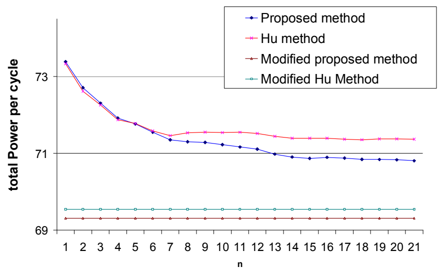
b) The rate of not accessing the BTB to the total number of instructions

Figure 3. Properties of the modified proposed approach for the Anagram benchmark

Figure 3a gives the percentage of the branch instructions that no BTB access is needed for them, for different values of *n* in the modified proposed algorithm. Figure 3b shows the rate of not accessing the BTB to the total number of instructions for different values of *n* in the modified proposed approach. According to the diagrams in figure 3the initial value of *n* does not affect the performance. As the result, it does not matter what initial value of n to be used.



a) Anagram benchmark



b) Go benchmark

Figure 4 The consuming power in each cycle for different values of n

Figure 4 shows the consuming power in each execution cycle for different values of n for the four methods: Hu method, proposed approach, modified Hu method and the modified proposed approach. As shown in the figure, the modified approaches have lower consuming power during the execution and changes of n, comparing to the original approaches. These results are drawn for the two Go and Anagram benchmarks. This is because the algorithm tries to adapt the value of n with the scatters of the branches in the program. Furthermore, it is obvious that the modified proposed approach outperform the modified Hu approach.

5. Conclusions

As the pipeline length in the processors increases, the accuracy of the branch predictor unit plays an important role in the performance of the processors. When designing the branch predictor unit we need to consider the consuming power. The processors these days, in order to determine the branch target and direction in each cycle refer to a unit called branch prediction unit, but most of the accesses to this unit are redundant. To reduce the consuming power in the branch predictor unit, we can restructure the BTB. Of these we can exemplify eliminating the non-necessary address section of the branch instructions in the BTB and reducing the number of accesses to the BTB.

In this paper, a method was proposed to reduce the number of accesses to the BTB. This proposed approach was to modify and enhance the performance of the Hu et al. [3] method. In these two approaches, a counter is assumed to make decision on whether to access the BTB or not. This counter is initialized after a condition is satisfied. These approaches are efficient for reducing the consuming power, and the larger values of the counter would lead to better performance, that's because the access to the BTB is postponed. But on the other hand, as discussed clearly in the paper, the branch instructions may enter before the counter has reached zero. This leads to an increase in the execution time and also the power. The simulation results indicate the better performance of the proposed approach.

After explaining the proposed approach, in order to further enhance the algorithm and solve the problems of initializing *n*, modified versions of the Hu and propped algorithm are also discussed. In these approaches, *n* is adapted during the execution of the program relative to the program structure. The simulation and experimental results show that these modified methods have less consuming power during the execution of the program.

6. References

[1] D.A. Jiménez, S.W. Keckler, and C. Lin, "The impact of delay on the design of branch predictors", *Proc. 33rd Int'l Symp. Microarchitecture*, 2000, pp. 67–77.

[2] D. Parikh, K. Skadron, Y. Zhang, and M. Stan, "Power-Aware Branch Prediction: Characterization and Design", *IEEE Transaction on Computers,* Vol. 53, No. 2, 2004, pp. 168-186.

[3] Y.C. Hu, W.H. Chiao, J.J. Shann, C.P. Chung and W.F. Chen, "Low-Power Branch Prediction," *In Proceedings of the 2005 International Conference on Computer Design (CDES'05)*, pp. 211-217.

[4] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan, "Power issues related to branch prediction", in Proc. 8th HPCA, 2002, pp. 233–244.

[5] K. Skadron, M. Martonosi, and D. Clark, "Speculative updates of local and global branch history: A quantitative analysis", *JILP*, Vol. 2, 2000.

[6] P. Petrov and A. Orailoglu, "Low-power Branch Target Buffer for Application-Specific Embedded Processors," *DSD 2003*, pp. 158-165.

[7] M. Monchiero, G. Palermo, M. Sami, C. Silvano, V. Zaccaria, and R. Zafalon, "Power-aware branch prediction techniques: A compiler-hints based approach for VLIW processors," *In Proceedings 14th Great Lakes Symposium VLSI, Boston, USA*, 2004, pp.440-443.