A Replacement Policy Based on Dynamic Profiling and Hashed Data

Marios Kleanthous*,1,4, Sami Yehia†,2,
Yiannakis Sazeides*,1, Emre Ozer‡,3

* *University of Cyprus, 75 Kallipoleos Street, CY-1678 Nicosia, Cyprus*
† *THALES Research & Technology, RD 128 - 91767 Palaiseau cedex, France*
‡ *ARM Ltd, 110 Fulbourn Road, Cambridge CB1 9NJ, United Kingdom*

ABSTRACT

This work proposes to use a small prediction table along with a profiling table to determine the impact of each predictor entry on the performance, and evaluates the idea using a simple stride predictor. Initial results indicate that adding a profiler to a small value predictor can reduce 17% of all stalls on average. More importantly, the results show that our mechanism can approximate the performance of the 128-entry fully associative predictor with just 20% of its size.

KEYWORDS: replacement policy, dynamic profiling, value prediction

1 Introduction

Current superscalar general purpose processors invest in all kinds of ways to predict data in order to exploit ILP. For example, value predictors and branch predictors keep the history of previous instructions in order to correctly predict the data or outcome of upcoming instructions. Unfortunately, these structures are very large and complex most of the time.

Although these structures are big, only a subset of the entries in these structures is effectively useful because of non-regular data or very rarely used entries. We also observed that it is difficult to reduce the number of entries of these structures without degrading its prediction accuracy because smaller structures will cause useful entries to be evicted early.

Embedded processors are unable afford such big and power hungry predictor structures. A cost efficient method is needed to select the best subset of instructions that will update a small predictor effiently. In other words, a new replacement policy must be considered that will also take into account the performance gain of a hit in order to select the best possible subset of instructions for updating.

Therefore, we propose a novel technique that exploits small structures using a profiling table having a number of entries on the same order of existing structures but with less information to determine the impact of each entry on the performance. Knowing the importance of such entries allows the predictor to be updated with the best possible subset of entries.

Our technique potentially targets any microarchitecture structure that uses replacement policies to improve performance. Example of such structures are caches, branch predictors

[1]E-mail: {mklean,yanos}@cs.ucy.ac.cy
[2]E-mail: {sami.yehia}@thalesgroup.com, This work was done while Sami was at ARM Ltd and is the property of ARM Ltd.
[3]E-mail: emre.ozer@arm.com
[4]This author wishes to thank HiPEAC and ARM for offering him the opportunity of an industrial Internship that resulted to this work.

and value predictors. In this work, we use a value predictor to illustrate our approach.

2 Related Work

Prior work mainly proposes replacement policies that decide which entry in the predictor table will be replaced or if an entry will be replaced based on its performance using event counters [KS05, KSD04] or hysteresis counters. Event counters keep track of the performance of each entry based on different events like miss or hit. Hysteresis counters provide delay of the replacement of an entry to avoid premature replacement.

Furthermore, there are various techniques [RMR07, CRT99] to filter the instructions that update the predictors. Such filtering reduces the pressure due to replacements and improves the prediction rate. Also confidence history is used in order to avoid replacing entries that deliver good predictions by other that have poor performance. Still, their method relies on having relatively big structures to achieve accurate predictions.

Using a very small structure of 8 entries for example, it will be very difficult for the hysteresis counters to work since there will not be enough time to get hits and increase the counters. We propose the use of a small predictor with a dynamic profiler. The profiler will have a number of entries on the same order of existing big structures but the size of each entry will be much smaller. The profiler will provide the decisions for replacement based on hashed information and also can achieve the filtering and confidence of unpredictable instructions by keeping track of the performance of each instruction using the profiler.

3 Profiling based replacement policy

We will use a simple Stride Value Predictor [EV93] as our baseline predictor to demonstrate the idea. A Stride Value Predictor is accessed with the PC of a load instruction at the decode stage. If there is a tag match then the destination register of the instruction is speculatively updated and the prediction is checked for correctness on the writeback stage. If the loaded value matches the prediction then the value predictor is updated with the new value and stride. If the loaded value does not match the prediction then the predictor is updated and also the pipeline is flushed and instructions, after the mispredicted load, are re-executed with the correct value. If there is a tag miss in the value predictor during the decode stage, a new entry will be inserted in the value predictor on the writeback stage.

Studying the behavior of load instructions and big value predictors, we discovered that usually a smaller percentage of those instructions are predictable and even a much smaller percentage of those instructions needs to be in the predictor at the same time at any given period of time. Trying to use a very small value predictor, of 8 entries for example, was very difficult for the techniques mentioned before to work since there was not enough time to train the counters that those policies are based on because of the continuous replacements.

The previous observation raises the need for a cheap way to know the performance of the instructions trying to update the value predictor without keeping those instructions in the small 8-entry table that provides the predictions. In order to achieve this, we propose the use of a dynamic profile table with hashed information in combination with the small 8-entry fully associative value predictor. The profile table will provide all the information needed to decide which and when an entry will be inserted in the value predictor.

Figure 1 shows the proposed Profile Table along with its associated 8-entry predictor. The fields of the profiler are as follow: a) the hashed value, which is the last 3 bits of the loaded value, b) the hashed stride, which is the last 3-bits of the delta of the last and previous hashed value, c) the counter, which is a 3-bit counter indicating the performance of the entry, and d)

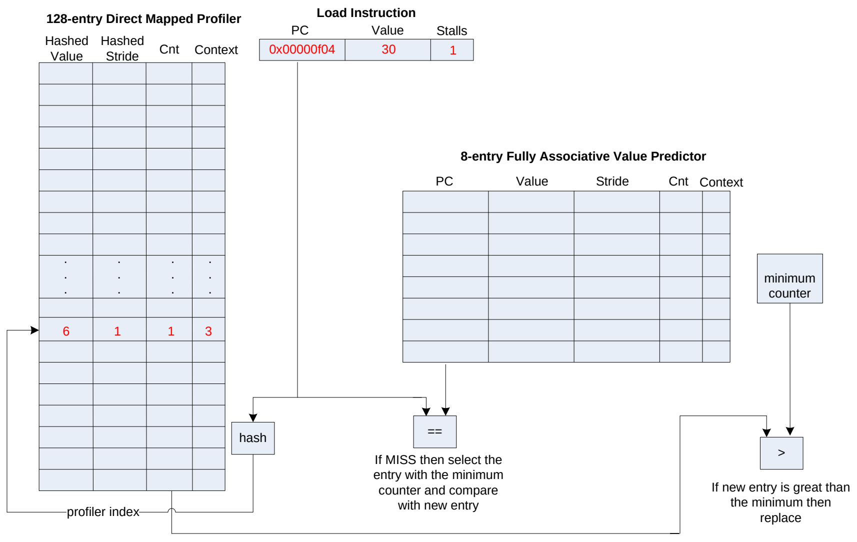
128-entry Direct Mapped Profiler
Hashed Value
Hashed Stride
Cnt
Context
6
1
1
3
Load Instruction
PC
Value
Stalls
0x00000f04
30
1
8-entry Fully Associative Value Predictor
PC
Value
Stride
Cnt
Context
minimum counter
hash
==
If MISS then select the entry with the minimum counter and compare with new entry
>
If new entry is great than the minimum then replace
profiler index

Figure 1: 128-entry direct mapped profiler with an 8-entry fully associative value predictor

the context, which is a unique number assigned heuristically to all loads that belong to the same contexts (4 bit value that becomes zero when it saturates to 16).

The use of hashed values along with the counter allows the profiler to approximately but very accurately calculate how much an entry will contribute to the performance. For example, each time a load instruction reaches the writeback stage it updates the profiler. The hashed value plus the hashed stride are compared with the hash of the loaded value and if they are equal then the counter is updated with the number of stalls that the load caused. Otherwise, if the hashed information does not match then the counter is decremented.

In order to detect and assign the context for an instruction we use a simple heuristic algorithm that detects loops during the execution and assigns a unique context number to all the loads that belong to this loop.

Once an entry has to be replaced in the value predictor the following steps are performed:

1. Access the value predictor and find a candidate entry for replacement
   (a) If one or more entries in the value predictor do not belong to the current context, then one of these entries will be selected
   (b) Else if all the entries in the value predictor belong to the current context then the entry with the minimum counter value will be selected
2. Compare the counter and context of the candidate entry for replacement with the counter and context of the new load instruction
   (a) If the candidate entry belongs to a different context than the current context, or belongs to the current context but has smaller counter value than the new entry, then replace the entry with the new one
   (b) Else if the candidate entry for replacement belongs to the current context and has also bigger counter value than the new load instruction then do nothing

4 Results and Future work

In this section we show that our profiler with a small 8-entry value predictor can approximate a bigger 128-entry value predictor.

We used a 5 stage pipeline model of an ARM core to collect information on the trace and detect potential removal of stalls cause by load dependancies. On every stall cycle we access the value predictor and in case of a correct prediction we can eliminate that stall. Figure 2

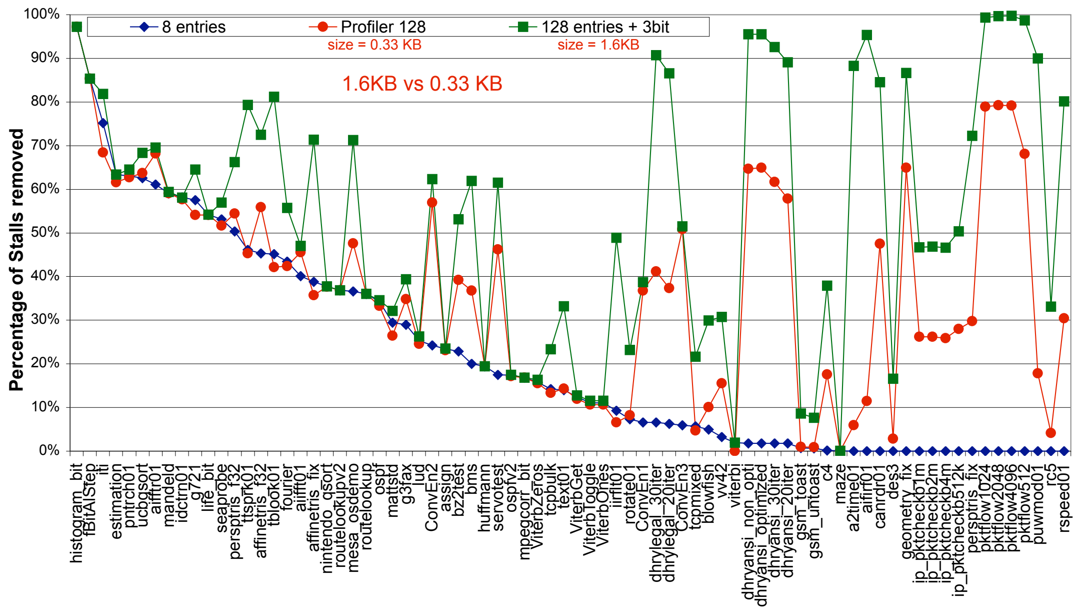

Figure 2: Results for a 128-entry profiler table with an 8-entry value predictor

shows the results using the percentage of stalls successfully removed, correctly predicted, from all the stalls caused during execution. The line "8 entries" correspond to an 8-entry fully associative value predictor using LRU, the line "128 entries + 3bit" corresponds to an 128-entry fully associative value predictor using LRU plus a 3-bit counter for each entry to provide hysteresis during replacement. A hysteris counter is set to maximum value when we have a correct prediction and is decremented on a misprediction. An entry cannot be replaced if the counter is not zero. Finally the "Profiler 128" corresponds to an 8-entry value predictor with the proposed dynamic profiler with 128 entries. The size of this is 0.33KB while the size of the 128-entry predictor is 1.6KB. The results indicate that the potential performance improvement is significant when compared to the LRU policy (17% more stalls removed). Also the results show that the mechanism can approximate the performance of the 128-entry fully associative predictor with just 20% of its size.

For future work, an optimal replacement analysis will be performed to measure the proximity of the profiler based policy to the optimal. We also want to investigate in more detail the effect of our profiler in performance and other structures like branch predictors and caches.

References

[CRT99] Brad Calder, Glenn Reinman, and Dean M. Tullesen. Selective value prediction. In *Proceedings of ISCA99*, 1999.

[EV93] R. J. Eickemeyer and S. Vassiliadis. A load instruction unit for pipelined processors. *IBM Journal of Research and Development*, July 1993.

[KS05] Mazen Kharbutli and Yan Solihin. Counter-based cache replacement algorithms. In *Proceedings of ICCD 2005*, 2005.

[KSD04] Martin Kampe, Per Stenstrom, and Michel Dubois. Self-correcting lru replacement policies. In *Proceedings of 1st conference on Computing frontiers*, 2004.

[RMR07] Roni Rosner, Avi Mendelson, and Ronny Ronen. Trace cache sampling filter. *ACM Transactions on Computer Systems*, 25:1, 2007.