On Table Bandwidth and Its Update Delay for Value Prediction on Wide-Issue ILP Processors

Sang-Jeong Lee, *Member*, *IEEE* and Pen-Chung Yew, *Fellow*, *IEEE*

Abstract

We study two issues which could affect the performance of value prediction on wide-issue ILP processors. One is the large number of accesses to the value prediction tables needed in each machine cycle, and the other is the delay required to update stale values in the value prediction tables. We introduce a prediction value cache (PVC) which augments the instruction cache to hold the prediction values to handle these issues.

Index Terms: decoupled value prediction, prediction value cache, dynamic classification

1 INTRODUCTION

To achieve high performance on wide-issue superscalar processors, it is essential to exploit instruction level parallelism (ILP). But control dependences and data dependences are major hurdles to exploit ILP efficiently. Many speculative execution schemes have been proposed to deal with them. Value prediction is a technique that breaks true data dependences by predicting the outcome of an instruction, and executes speculatively its data-dependent instructions based on the prediction outcome.

There are many studies on value prediction [2-9]. They include last value prediction [5], stride prediction [6,9], context-based prediction [8,9], and hybrid value prediction [6-9]. Last value prediction predicts the result value of an instruction based on its most recently generated value. A stride predictor predicts the value by adding the most recent value to the difference of the two most recently produced values. This difference is called stride. Context-

based predictors predict the value based on the repeated pattern of the last several values observed. FCM (Finite Context Method) [8] and two-level [9] predictors belong to this category. Some hybrid predictors are proposed by combining several predictors. Wang et al. proposed a hybrid predictor that combines a stride predictor and a two-level value predictor [9]. Rychlik et al. [6,7] combine a last value predictor, a stride predictor and a FCM predictor. The choice of a predictor for each instruction is guided by a dynamic classification mechanism. They also developed a selective reissue core (SRC) to recover after mispredictions [7].

In wide-issue superscalar processors, e.g. 8- to 16-issue processors, we may need 8 to 16 simultaneous accesses to the value prediction table in each machine cycle. This will require very expensive hardware because it needs a large number of read/write ports to the value prediction table which may have an adverse effect on the clock cycle time. Most prior studies are thus focused only on the predictor algorithms and their potential performance gain. Some studies such as Gabbay et al. suggest using multiple interleaved banks with a fast distribution network to access the value prediction table [2]. However, potential bank conflicts may exist because several instructions could access the same bank simultaneously. Furthermore, it is quite difficult to implement multiple-bank predictors for context-based value prediction schemes, such as the two-level predictor [9] and the FCM predictor [8].

In a processor, the result of each instruction is produced after its execution. Therefore, the value prediction table can be updated for next prediction only after its execution. If the value prediction table is accessed in the instruction fetch stage and assume we have an execution pipeline with more than 4 stages, it will take more than four cycles to produce the updated values for the prediction table. In a 8-issue processors, there will have been more than 32 instructions executed during this period of time. It is very likely that the same instruction will need to fetch the same value prediction table entry again before it has been updated. This

most likely will cause incorrect value predictions.

In this paper, we address those implementation issues for wide-issue architectures. We augment the instruction cache with a prediction value cache (PVC) to hold prediction values for each instruction in a cache block [4]. It not only allows multiple accesses to the prediction values in each machine cycle, but also allows us to move the value prediction from the instruction fetch stage to a later pipeline stage after the update values are produced. We assume each prediction table has only 2 read/write ports with queues. In addition, we study several schemes to reduce the potential stale value accesses using execution-driven simulations and SPECint95 benchmarks.

2 MOTIVATION

We have simulated several schemes based on the hybrid predictors developed by Wang et. al [9] (see Figure 2). We found that the banked predictors degrade the performance comparing to a hybrid predictor, which allows unlimited simultaneous accesses to the prediction table in each machine cycle, mostly due to bank conflicts. When bank conflicts occur, the prediction table cannot be accessed or updated. Also, we found that when we assume the predictor can update its prediction table instantaneously, the performance is greatly increased.

The main reason for this improvement is due to the stale values in the value prediction table. As the processor increases instruction fetch and issue width for higher ILP, the number of machine cycles between two successive value prediction table lookups to the same table entry could be substantially reduced. Furthermore, the increased number of pipeline stages required by a higher clock rate will increase the number of cycles between the actual prediction value update (which usually occurs in a later stage of the pipeline) and the prediction value usage (which usually occurs in an early stage of the pipeline). Therefore,

many instructions may access stale prediction values from the table before they can be actually updated. In our paper [4], we could see, on average, 36% of instructions access the same table entry again within 5 cycles and 22% of the instructions access the same table entry within 2 cycles. This effect can cause serious performance degradation because stale values mostly are incorrect, especially for stride and context-based predictors which need up-to-date values for correct predictions.

In this paper, we propose several techniques to mitigate these problems. For the required multiple accesses to the prediction table in each machine cycle, we propose using a prediction value cache (PVC) in conjunction with the instruction cache [4]. The organization of PVC is exactly the same as the instruction cache. Hence, the prediction value of each instruction can be fetched exactly the same way the instructions are fetched from the instruction cache, i.e., both PVC and the instruction cache have the same block size and set associativity.

To mitigate the problem of stale prediction values, we associate an age counter with each prediction value in PVC to identify potential stale prediction values. The age counter gives an indication of how many times the prediction value has been accessed before the prediction value is updated. We also use a value prediction scheme which is decoupled from the instruction fetch stage [3]. This allows value prediction to be done in a later stage and shorten the time between the value prediction updates and their usages.

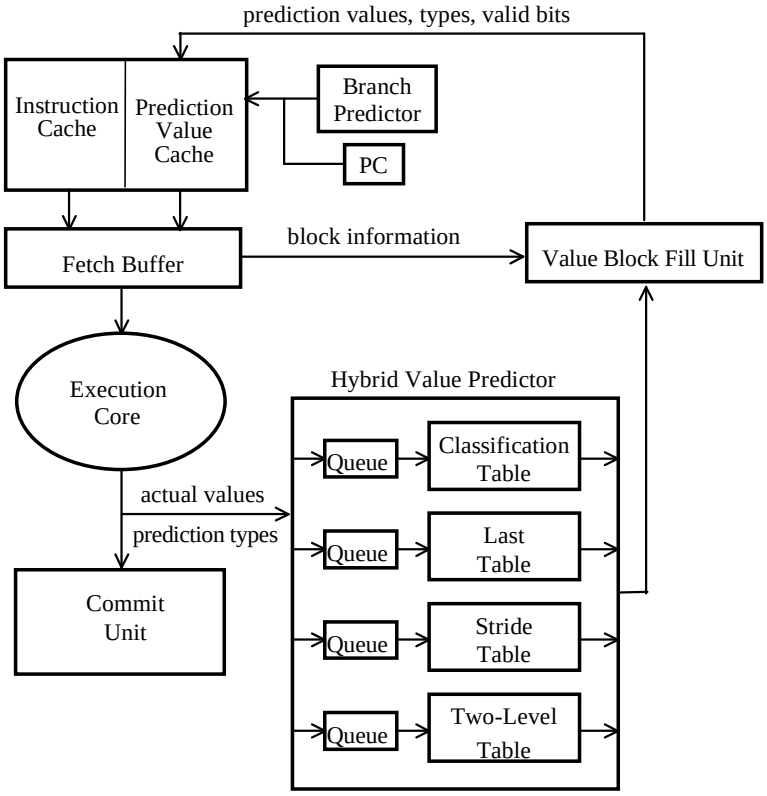
3 DECOUPLED VALUE PREDICTION SCHEME

3.1 Main Components of Decoupled Value Prediction Scheme

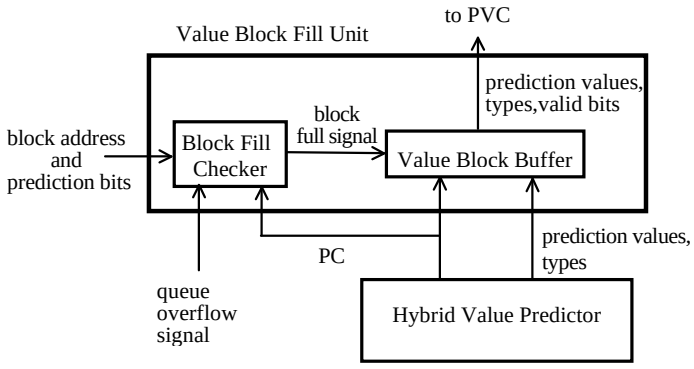
Figure 1 shows the main components of our decoupled value prediction scheme. It has a prediction value cache (PVC) associated with the instruction cache. PVC is organized in the same way as the instruction cache in terms of its block sizes, set associativity and address mapping scheme. Each instruction cache block is augmented with one PVC block, and each

instruction in the instruction cache block has a corresponding entry in the PVC block. Each PVC entry contains the prediction value for the result of the corresponding instruction, its prediction type and a valid bit. Augmenting the instruction cache with PVC provides a very simple mechanism to access multiple prediction values without the complexity of having to access both the branch prediction tables and the value prediction tables in the instruction fetch stage as in other proposed schemes. It also avoids potential bank conflicts among the multiple accesses to the value prediction tables since both the instruction cache and PVC have the exact same organization.

As opposed to the previous schemes, our value predictor is decoupled from the instruction fetch stage and is accessed when the instructions complete their execution (details in section 3.2). The value predictor used is a hybrid predictor which includes: a last value predictor, a stride predictor, a two-level predictor and a classification table for dynamic classification. We assume that each predictor table has only two read ports and two write ports. Since there could be up to 8 instructions accessing and updating the prediction tables for a 8-issue processor, we provide a queue in front of each prediction table (see Figure 1 (a)). Because the predicted values are only advisory and the chance for a queue being full is small if the queue size is chosen properly, to simplify the hardware design, we assume further accesses to a prediction table are dropped when its queue becomes full. The prediction values in PVC may become stale when this happens. However, our simulation shows the overflow rate of the queue is negligible for most benchmarks (details in [4]). According to the prediction type, which is determined by the classification table and is stored in each PVC entry, a request is sent to update the prediction table during the writeback stage with the result its instruction just produced. It also produces a prediction value which is sent to the corresponding PVC entry through VFU (value block fill unit) after the prediction table lookup (see Figure 1 (b)).



(a)



(b)

Figure 1. (a) Decoupled value prediction scheme (b) Value block fill unit

VFU receives the block information after instructions are decoded. It allocates one block for each corresponding instruction cache block. The information VFU received includes the block address, which is the starting address of the cache block, and a prediction bit for each

instruction in the block. The prediction bit is set when the instruction requires a predicted value, i.e. requires value prediction. When VFU receives prediction values, prediction types and the addresses of the instructions from the hybrid value predictor, VFU assembles these information to form a block. When all of the information becomes available in a block, using the block fill checker, VFU forwards the block to the PVC. When the queues overflow, to simplify our scheme, we assume that the requests to the hybrid prediction table are dropped and VFU is notified. The corresponding instructions can thus miss the updated prediction values. However, in our simulations, overflow in queues rarely happen when we use a queue size of 32.

3.2 Instruction Pipeline Design

There are many ways to implement an instruction pipeline which incorporate the value prediction scheme proposed above. We assume an instruction pipeline similar to the one used in the SimpleScalar simulator [1] described as follows.

In the instruction fetch stage, the predicted result values from the PVC are read at the same time when the instructions are fetched from the instruction cache. These predicted result values are forwarded to the data-dependent instructions waiting in the reservation stations through the fetch buffer after the instructions are decoded in the dispatch stage. In the dispatch stage, the block information which consists of the block address and valid bits is sent to the VFU. When all the operands of an instruction are available in the reservation station (regardless of whether their values are predicted or not) and there is an available functional unit, the instruction is scheduled and executed. As in most value prediction schemes which require such speculative execution, the reordering buffer is used to keep both speculative and non-speculative results. After the actual results are produced, the value prediction is verified in the writeback stage. If the prediction is correct, the execution is continued without any

interruption. If the prediction is incorrect, the execution of the data-dependent instructions which have already been issued will be invalidated and reissued with the correct values. We use the selective reissue model proposed by Rychlik et al. [7] which re-executes only those instructions that were executed with mispredicted operands. The actual results and the correctness of the prediction are inserted into the queues according to their prediction types. They will then be used to update the value prediction tables and to obtain the next prediction values.

When a predictor in the hybrid value predictor receives a request from the queue, it will process the request according to its prediction type. The predictor updates the state and the confidence counter of the corresponding entry in the prediction table according to the produced result. If the previous prediction is incorrect, it may need to store the new value in the table according to the prediction type. As soon as it updates the prediction table, it will predict the next value according to the prediction type and send the result to VFU. VFU collects prediction result of a instruction and assemble a block. After a block is assembled, it is forwarded to PVC.

3.3 A Decoupled Hybrid Value Predictor using Dynamic Classification

Our predictor is similar to the one used in [6]. It partitions the requests into several prediction types. The classification is done dynamically with the help of the classification table. The table contains five fields: tag, state, prediction value, stride and class fields. The class field indicates one of the four types: Last, Stride, Two-level, and Unknown. The state field of the classification table is initially set to the initial state. When the value predictor encounters an instruction for the first time, it forwards the instruction to the classification table (see Figure 1). The initial state changes to the transient state. The result value (value1) is stored in the value field and the class is set to the *Unknown type*. If an entry in the

classification table is in the transient state, a stride value (stride1) is calculated by subtracting the old value (value1) in the value field from the new result value (value2) of the instruction. The value2 and the stride1 are stored in the value field and the stride field in the classification table, respectively. The state is then changed to the classified state. If an entry is in the classified state, its stride value (strdie2) will be calculated again. If the stride2 is the same as the stride1 stored in the stride field and the stride value is zero, the prediction type of the instruction is set to the *Last type*. If the stride values are the same but the stride value is not zero, the prediction type of the table is then set to the *Stride type*. The prediction type and the predicted value are sent to PVC. Next time, the instruction is predicted by using the corresponding predictor type. If the newly calculated stride2 is not the same as the stride1 stored in the stride field, the instruction is classified as the *Two-level type*.

Once the prediction type of an instruction is determined, the instruction is removed from the classification table, and its information is forwarded to the corresponding prediction table. Its future predicted values are obtained by accessing the corresponding prediction table specified by its prediction type. A confidence counter is associated with each entry in the prediction tables. If the value of a confidence counter becomes zero after repeated mispredictions, the predictor can no longer accurately predict values for the instruction. In this case, the instruction is removed from the predictor table, and the type of prediction is reset to the *Unknown type*. This instruction needs to be reclassified using the classification table.

Our predictor has several differences from the one used in Rychlik et al. [6]. First, for the classified type, the decoupled scheme does not need another table because the type is stored in PVC. Second, to classify an instruction, Rychlik et al. stores three distinct values in the classification table, but our scheme stores only one value and one stride. Finally, our scheme always tries to predict the value, but in their scheme, when the instruction is removed from

the FCM predictor, the instruction is no longer predicted. Our decoupled predictor provides not only the hybrid prediction mechanism but also multiple prediction tables for simultaneous multiple value predictions.

3.4 Age Counters and Speculative Update

As mentioned in section 2, stale prediction values can cause misprediction and can have a significant impact on performance. Therefore, we investigate schemes to reduce this effect in our decoupled value prediction scheme, especially for the Stride type and the Two-level type. We introduce an age counter in PVC to determine the *staleness* of a predicted value. Initially the age counter is set to zero. Whenever a prediction value is fetched in PVC, its age counter is incremented. When the predicted value in PVC is updated for the predictor, the counter is decremented. If the value of a counter is zero, it means that the value in PVC is not stale. If the value of the age counter is not zero, some action is needed to cope with stale prediction value.

In this paper, we examine three schemes to deal with accesses to potential stale prediction values. In the first scheme, for the Stride or the Two-level type, if its age counter value is not zero, we don't use the prediction value in PVC because its stale prediction value may cause misprediction, i.e. we don't perform value prediction in such cases to avoid misprediction penalty. But this scheme will bypass many opportunities for value prediction because the Stride type and the Two-level type are the dominant types in most of our benchmarks. In the second scheme, we exclude only the Stride type for speculative execution because it is most likely to cause misprediction if it has stale values. In the third scheme, we use a speculative update scheme for the Stride type. In this scheme, instead of skipping value prediction, we update the prediction values speculatively using the value of the age counter. The age counter value actually shows the number of outstanding accesses to the prediction value. Hence, we

can obtain correct prediction values by adding the value in PVC with an adjusted value calculated by multiplying the stride value with the value of the age counter. The obtained prediction value is updated in PVC, and the age counter is reset to zero. We also mark that instruction as a speculative update candidate. When this kind of instruction is encountered again, the prediction value is fetched for speculative execution and then the new prediction value is calculated by adding the value with the stride, and PVC is updated with this new value. In section 5, we will discuss the performance of each scheme.

4 EVALUATION METHODILOGY

We use a 8-issue out-of-order superscalar processor as a baseline architecture. The model has a high instruction fetch bandwidth and a large instruction window (256 entries) as well as large on-chip caches (512 KB). The fetch unit can deliver two basic blocks from the instruction cache but no more than 8 instructions total. The hybrid branch predictor uses a gshare branch predictor with 16K entries and 8-bit history, and a 16K-entry bimodal branch predictor. The PVC has 64KB and direct-map cache with 32 bytes per block. The hybrid value predictor has a 8K-entry value history table (VHT) and a 8K-entry pattern history table (PHT). The value predictor in our proposed scheme has a 4K-entry Last table, a 1K-entry Stride table, a 8K-entry Two-level table and a 1K-entry Classification table. The overall table size of our predictor is roughly 7% larger than the size of a 8K-entry hybrid predictor.

Simplescalar/Alpha 3.0 tool set is used [1]. We add the value predictors and our decoupled scheme on the Simplescalar's sim-outorder simulator for the PISA ISA. The SPECint95 with small input set ,below 100M executed instructions, used in our simulation (For more detailed information of the evaluation methodology, see [4]).

5 PERFORMANCE ANALYSIS

Figure 2 shows the speedup over the baseline architecture. Hybrid-2B and Hybrid-4B use the hybrid predictor schemes similar to [9] with a two-bank and a four-bank prediction table, respectively, to allow multiple accesses in one machine cycle. Hybrid is the hybrid predictor with no limitation on the number of accesses to the prediction table in each machine cycle. Hybrid-Immediate is the predictor that updates prediction table instantaneously with no limitation on the number of accesses to the value prediction table in each cycle. PVC uses our decoupled scheme with only 2 read/write ports in the prediction tables.

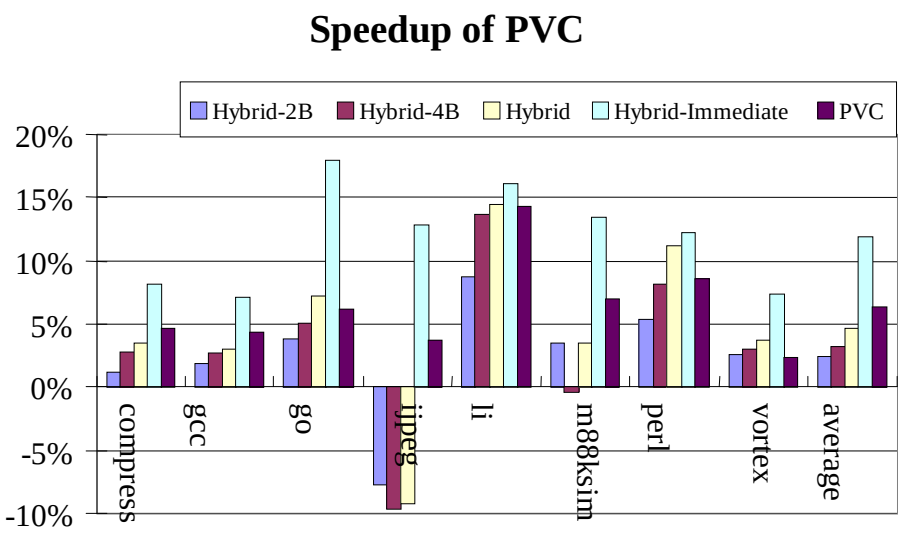


Figure 2. The speedup over the baseline architecture

In Figure 2, the average speedups for Hybrid-2B and Hybrid-4B are 2.4% and 3.1%, respectively. The speedups of Hybrid and Hybrid-Immediate are 4.6% and 11.9%. And the speedup of PVC is 6.3%. PVC shows a better performance than Hybrid, which has no limitation on the number of accesses to the prediction tables. It is because PVC with a dynamic classification capability can predict values more accurately than Hybrid. But PVC cannot reach the performance level of Hybrid-Immediate because of the latency required to classify the prediction types and to update the prediction values. The performance of ijpeg using hybrid schemes shows negative impact of misprediction (ijpeg has the highest miss

prediction rate among SPECint benchmarks).

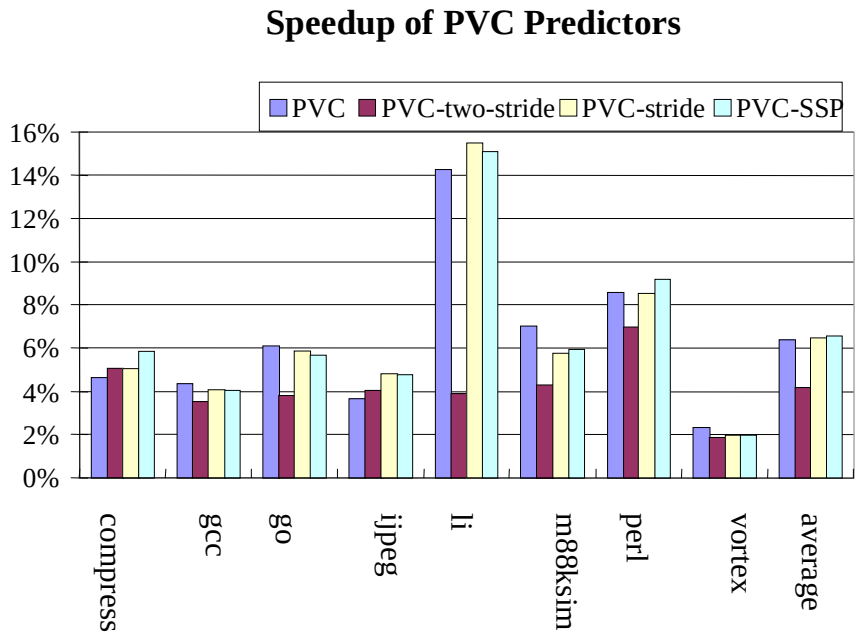


Figure 3. The speedup of PVC predictors over the baseline architecture

Figure 3 shows the speedup over the baseline architecture for several PVC schemes. PVC-two-stride is the scheme which does not perform value prediction for those instructions whose value prediction type is either stride or two-level and their age counter values are not zero. PVC-stride is the scheme which does not perform value prediction for instructions whose value prediction type is stride and their age counter values are not zero. PVC-SSP is the scheme which updates speculatively for the stride type when their age counter values are not zero. In Figure 3, on average, the performance of PVC-stride and PVC-SSP is slightly better than PVC. But PVC-two-stride is worse than PVC because the performance gain from using PVC-two-stride for li is much worse than that from using PVC. It is because PVC-two-stride skips too many value predictions. In the case of compress and ijpeg, PVC-stride and PVC-SSP show somewhat better performance, but m88ksim does not perform well. The main reason that the age counter and speculative update did not show the expected performance

gain is that the number of loop iterations in most benchmarks is too small to cover the latency required to determine the value prediction types dynamically, and hence, to exploit the benefit of the age counter and speculative update. We believe that if compiler analysis can be used to determine the value prediction types, the age counter and the speculative update will be much more effective. However, that is beyond the scope of our paper.

We examined other parameters such as prediction accuracy, distribution of the prediction types (see [4] for more detailed results). *Prediction accuracy* is the percentage of correct predictions among all those instructions which actually obtain prediction values. On average, the prediction accuracy of PVC is 76.3% and that of Hybrid is 65.4%. The PVC shows higher prediction accuracy than the Hybrid because of dynamic classification. We also measured the distribution of prediction types for PVC. For PVC, 37.0% of the instructions which perform value prediction have the Last type, 8% have the Stride type, 43.5% have the Two-level type, 8.5% have the Unknown type, and 2.5% of the instructions cannot be predicted due to queue overflow in the predictors.

6 CONCLUSIONS

In this paper, we propose a value prediction scheme with dynamic classification using a prediction value cache (PVC) for wide-issue processors. By augmenting instruction cache with a PVC and decouple value prediction from the instruction fetch stage, as well as providing multiple value prediction tables and queues, we can provide effective value prediction using realistic 2 read/write ports in prediction tables for a wide-issue (e.g. 8-issue) processor. We also examine the effect of potential delay in updating the value prediction tables which is the main cause of stale prediction values and mispredictions. We examine the performance of some improvement schemes, such as attaching an age counter and using

speculative updates for the Stride type and Two-level type. We found most of them not very effective due to the short average duration between two successive accesses to the same table entry as mentioned above, and the relatively long latency required to dynamically classify access types.

ACKNOWLEDGMENT

The work was supported in part by the U.S. National Science Foundation under Grants MIP EIA-9971666 and MIP-9610379, and a grant from Intel Corporation. The work was done while Sang-Jeong Lee was on a sabbatical leave at the University of Minnesota partially supported by Soochunhyang University, Korea.

REFERENCES

[1] D.Burger and T.Austin, “The Simplescalar Tool Set, Version 2.0”, *Technical Report CS-TR-97-1342*, University of Wisconsin, Madison, June 1997

[2] F.Gabbay, and A.Mendelson, "The Effect of Instruction Fetch Bandwidth on Value Prediction", *Proceedings of the 25th International Symposium on Computer Architecture (ISCA-25)*, p.272-281, 1998.

[3] S.Lee, Y.Wang, and P.Yew, "Decoupled Value Prediction on Trace Processors", *Proceedings of the 6th International Symposium on High Performance Computer Architecture (HPCA-6)*, 2000.

[4] S.Lee, and P.Yew, "On Some Implementation Issues for Value Prediction on Wide-Issue ILP Processors", *International Conference on Parallel Architectures and Compilation Techniques (PACT2000)*, Oct. 2000.

[5] M.Lipasti, and J.Shen, "Exceeding the Limit via Value Prediction", *Proceedings of the 29th Inter-national Symposium on Microarchitecture (MICRO-29)*, Dec. 1996.

[6] B. Rychlik, J.Faistl, B.Krug, A.Kurland, J.Jung, Miroslav, N.Velev, and J.Shen, "Efficient and Accurate Value Prediction Using Dynamic Classification", *Technical Report of Microarchitecture Research Team in Dept. of Electrical and Computer Engineering, Carnegie Mellon Univ.*, 1998.

[7] B.Rychlik, J.Faistl, B.Krug, and J.Shen, "Efficacy and Performance Impact of Value Prediction," *Parallel Architectures and Compilation Techniques*, Paris, Oct. 1998.

[8] Y. Sazeides, and J.Smith, "The Predictability of Data Values", *Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30)*, Dec. 1997.

[9] K.Wang, M.Franklin, "Highly Accurate Data Value Predictions using Hybrid Predictor," *Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30)*, Dec. 1997.