Decoupled Value Prediction on Trace Processors

Sang-Jeong Lee†, Yuan Wang‡, and Pen-Chung Yew‡

†Dept. of Computer Science and Engineering, Soonchunhyang Univ., Chungnam, Korea

‡Dept. of Computer Science and Engineering, Univ. of Minnesota, Minneapolis,MN 55455

{ slee, yuan, yew }@cs.umn.edu

Abstract

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 predicted outcome. In this paper, we address several implementation issues for value prediction which are important on wide-issue superscalar architectures, and present a value prediction scheme based on the trace processor [18]. The scheme decouples the value prediction from the instruction fetch stage and use a hybrid predictor with dynamic classification. We use execution-driven simulation to study the performance of such a scheme using SPECint95 benchmarks.

1. Introduction

To achieve high performance on wide-issue superscalar processors, it is essential to maintain high instruction issue rate and to exploit as much parallelism among instructions as possible. Instruction issue rate is mostly limited by the changing of control flow direction and its subsequent nonconsecutive instruction layout in the instruction cache. To overcome such problems, many dynamic branch predictors [3,11,23] and the trace cache have been proposed [4,17]. On the other hand, the major hurdle in exploiting instruction-level parallelism (ILP) is the data dependences among issued instructions. There are two kinds of data dependences: true data dependences and false data dependences (or name dependences). False data dependences can be eliminated by using hardware and software techniques such as register renaming. But true data dependences remain a serious problem for wide-issue processors. Recent proposed schemes using value prediction provide an effective way to address this problem [2,6,7,9,13,19-22]. By predicting the result value of an instruction, the processor can proceed to speculatively execute its data-dependent instructions. These studies show value prediction can provide rather accurate values for many programs [13,19-22], and good performance gain can be obtained [2,7,9,20].

However, most recent value prediction studies assume perfect branch prediction [7,20]. Most studies also assume all fetched instructions could access the value prediction table simultaneously in the instruction fetch stage. However, in wide-issue superscalar processors which allow 8 to 16 instructions per clock cycle, we may need to perform several branch predictions and 8 to 16 simultaneous accesses to the value prediction table in each cycle. This will require very expensive hardware because it needs very high bandwidth branch and value prediction table with many read and write ports to accommodate so many simultaneous accesses. Most prior value prediction studies are thus focussed only on the predictor algorithms and their potential performance gain. Some studies such as in Gabbay et al. [7] suggest using multiple interleaved banks with a fast distribution network to implement the value prediction table. However, potential bank conflicts still exist because several instructions could access the same bank simultaneously. Furthermore, it is quite difficult to implement predictors which allow context-based value prediction, such as the two-level predictor [22] and FCM (Finite Context Method) predictor [21], using their proposed multiple-bank prediction table. Another important drawback of the previously proposed schemes is that they all assume the value prediction table is accessed in the instruction fetch stage. It requires all instructions to access the value prediction table regardless of whether they are needed or not because instructions are not yet decoded in the fetch stage.

In this paper, we address those implementation issues of the value prediction for wide-issue architectures. In particular, we focus on the processors with trace cache because such architectures incorporate multiple branch predictions and speculative execution in a very effective way [4,11,17,18]. We propose a value prediction scheme that is decoupled from the instruction fetch stage and use the prediction scheme with dynamic classification similar to the scheme of Rychlik et al. [19]. Value prediction is carried out in the same stage where the updates to the value prediction table occur, such as the instruction writeback stage, and the predicted values are sent to the value prediction buffer (VPB) associated with the trace cache. The predicted values in VPB are read at the same time the instructions are fetched from the trace cache. Such a scheme can reduce the bandwidth requirement to the value prediction table because the types of the instructions are already known. It is easier to classify the types of value prediction using schemes such as the scheme of Rychlik et al. [19] for more accurate results.

The rest of the paper is organized as follows: Section 2 describes recent related work on the trace cache and the value prediction schemes. Section 3 summarizes the value prediction algorithms used in our decoupled scheme. Section 4 presents our decoupled value prediction architecture and the value prediction with dynamic classification. Section 5 summarizes our evaluation methodology, and in section 6 we present our simulation results. Finally, section 7 offers some conclusions.

2. Related work

In order to supply as many instructions as the processor instruction issue width, the trace cache supplies instructions by caching logically contiguous instructions, called traces, dynamically at run time. Many studies show that the trace cache is a more effective mechanism to supply multiple basic blocks per clock cycle than most conventional instruction fetch mechanisms using multiple branch prediction and realignment hardware [4,17]. The trace cache moves the complexity out of the instruction fetch/issue stages to the trace cache fill stage and is mostly out of the critical path in the instruction execution.

In the case of value prediction, several schemes have been proposed. They include last value prediction [13], stride prediction [6], context-based prediction [21,22], and hybrid value prediction [19,21,22]. 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 the stride. Context-based predictors predict the value based on the repeated pattern of the last several values observed. FCM [21] and two-level [22] predictors belong to this category.

Each of the predictors mentioned above shows good performance for certain data value sequences, but bad for others. Therefore, some hybrid predictors are proposed by combining several predictors. Wang et al. proposed a hybrid predictor which combines a stride predictor and a two-level value predictor [22]. Rychlik et al. [19,20] 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 show that the prediction rate is not a good indicator of performance because over 30% of the value predictions in the SPECint programs have no effect in enhancing the performance [20]. They noticed that updating the value prediction tables after the real values are produced, instead of an ideal immediate update after each prediction table access, can result in lower prediction rate, hence, lower performance. It is because the subsequent instructions may use the stale prediction values before they are updated. They also developed a selective reissue core (SRC) to recover after mispredictions [20]. Calder et al. examine selective value prediction schemes based on the confidence counter value and the instruction types [2].

In spite of those studies on value prediction, there are few studies focussing on multiple value predictions which are required on wide-issue superscalar processors. Gabbay et al. have shown the limitation of multiple accesses to the value prediction table using a realistic implementation [7]. They proposed a highly-interleaved prediction table with a fast distribution network to support high instruction issue rate.

To skip the execution of sequences of instructions by reusing the results of their prior executions, basic block reuse [10] and trace level reuse [8] schemes are proposed. But their focus is more on skipping instruction execution than on predicting the values.

Reinman et al. propose a Fetch Target Buffer (FTB) for fast instruction delivery [16]. They decouple the FTB from the instruction fetch and decode stages for a more scalable design.

3. Value predictor

In this section, we describe the value predictors used in our study. They include: the last value predictor (Last), the stride predictor (Stride), the two-level value predictor (Two-level) and a hybrid predictor. Other predictors such as FCM are not included, but can also be used in our decoupled schemes.

3.1. Last value predictor

The last value predictor uses a prediction table which contains a tag, the last value and a confidence counter in each entry [13]. The prediction table is indexed by the address of an instruction. The predictor stores the last value produced by the instruction, and uses the value as the predicted next result if the value of its confidence counter is greater than a threshold. The confidence counter is incremented by one if the prediction is correct and decremented by one if not. We set the threshold value of the confidence counter to two.

3.2. Stride predictor

The table of a stride predictor has four fields: tag, state, predicted value and stride [6,22]. There are three possible states for the state field: Initial, Transient and Steady. When an instruction accesses the prediction table for the first time, the state is set to the initial state, and its result value is stored in the value field. Next time when the instruction is encountered, the state is changed to the transient state and the stride is calculated by subtracting the value in its value field from the new result value. If the stride remains the same in the next access, the state is changed to the steady state. While in the steady state, predictions can be made by adding the value with the stride. If the prediction fails and the confidence counter becomes 0, the state is returned to the transient state, and the cycle starts again.

3.3. Two-level value predictor

The two-level value predictor consists of two tables: a value history table (VHT) and a pattern history table (PHT) [22]. Each instruction is directly mapped to a VHT entry using its address. A VHT has four fields: tag, LRU, four distinct data values, and its value history pattern. The data value field stores up to four distinct values generated by the instruction. The LRU field keeps track of the order in which the four distinct data values were last observed. The value history pattern records the pattern of the last few outcomes of the instruction. Each pattern is composed of a series of binary coded indices into the four data value fields. This value history pattern is used to index the PHT that has four up-down saturating counters corresponding to the data values field. The counter is incremented or decremented by one according to the prediction result. Prediction is made only if the maximum counter value is greater than or equal to a threshold value, which is set to two.

3.4. Hybrid value predictor

The hybrid value predictor is a combination of a stride and a two-level value predictors [22]. Compared to the VHT of the two-level value predictor, the VHT of the hybrid predictor has two additional fields: state and stride, for the stride prediction. The two-level value predictor is always accessed first, but its prediction is used only if the maximum counter value is greater than or equal to the threshold value. Otherwise, the stride predictor is used.

4. Decoupled value prediction scheme

In this section, we describe our decoupled value prediction scheme and a value predictor using a dynamic classification mechanism.

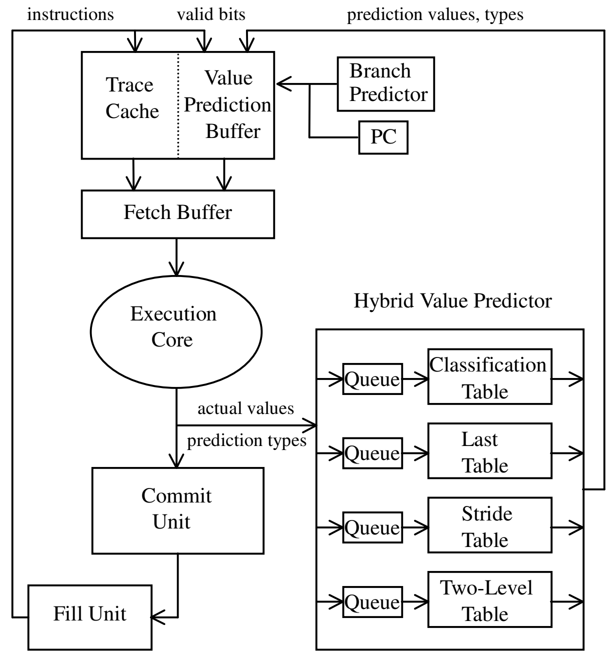

Figure 1. Decoupled value prediction architecture

4.1. Decoupled value prediction architecture

Figure 1 shows the main components of our decoupled value prediction scheme in conjunction with a wide-issue trace processor. It has a value prediction buffer (VPB) associated with the trace cache. Each trace cache block is augmented with one VPB block, and each instruction in the trace cache block has a corresponding entry in the VPB block. The value predictor may predict either source operands or the results of the instructions [14]. Since our proposed scheme move the value prediction out of the instruction fetch, it is easier to implement schemes based on the result prediction. The rest of the paper thus assumes the result prediction is used. Each VPB entry contains the predicted result value of the corresponding instruction, its prediction type and a valid bit. Augmenting the trace cache with VPB provides a very simple mechanism to access multiple predicted 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 table. 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 (see section 4.2). 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 16 instructions accessing and updating the prediction tables, and we assume realistically that each prediction table has only 2 read ports and 2 write ports, we provide a queue in front of each prediction table (see Figure 1). We fix the queue size to 32. 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 the prediction table are dropped when its queue becomes full. The predicted values in VPB may become stale when this happens. However, our simulation shows the overflow rate of the queue is negligible for most benchmarks.

According to the prediction type, which is determined by the classification table and is stored in the VPB entry, a request is sent to update the prediction table with the result its instruction just produced. It also produces a predicted value which is sent to the corresponding VPB entry after the prediction table lookup. These updates continue to be delivered to the VPB even if the trace fetch is stalled due to a trace cache miss, a branch misprediction or an instruction cache miss.

4.2. Instruction pipeline design

There are many ways to implement an instruction pipeline which incorporates the value prediction scheme proposed above. It is beyond the scope of this paper to come up with an optimal design. However, to facilitate our simulation study on the performance of the proposed scheme, we assume the instruction pipeline for the proposed architecture as follows. We first focus on the "fast path" design in the trace processor (which is the execution path used when there is a trace cache hit) [18]. The "slow path" design (the execution path when there is a trace cache miss) is described later.

In the instruction fetch stage, the predicted result values from the VPB are read at the same time when the instructions are fetched from the trace cache. These predicted result values are inserted into the reservation stations of the data-dependent instructions through the fetch buffer (see Figure 1) after the instructions are decoded in the dispatch stage. When all the operands of an instruction are available in the reservation station (regardless of the operand 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. When comparing the actual results with the predicted values, 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 which Rychlik et al. proposed [20]. 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 predicted values.

When a predictor gets a request from the queue, it will process the request similar to other value prediction schemes. 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 VPB.

4.3. Selective prediction

Not every instruction in the trace block needs a predicted value. Only the instructions which have data dependent relation with the predicted value need a predicted value to break the data dependences. For integer and ALU operations (we assume they have one clock cycle latency), they need predicted values to break the data-dependences among instructions in the same trace block so they can be issued in the same cycle. In the case of load instructions, which may have multiple-cycle latency, data-dependent instructions issued simultaneously or in later cycles will need their predicted values.

The fill unit of the trace cache is in charge of constructing trace blocks from the executed instructions. It is usually not in the critical path of the instruction execution. It can easily mark the instructions which require value prediction. Because load instructions are multiple-cycle instructions which most likely will have data-dependent instructions in the same or later trace blocks, we always predict their result values. Using this selective prediction, we can reduce the number of accesses to the value prediction tables by more than 20% on average. It allows the predictor resources to be used more efficiently and effectively.

4.4. A decoupled value predictor using dynamic classification

Our predictor partitions the requests into several prediction types as described in Section 3. The classification is done dynamically with the help of the classification table. The table contains four fields: tag, state, predicted value, stride and class fields. The class field indicates one of the four types: Last, Stride, Two-level, and Unknown types. The state field of the classification table is initialized 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 is stored in the value field and the class is set to the Unknown type. Next, when the instruction is in the transient state, a stride value is calculated by subtracting the old value in the value field from the new result value of the instruction. The new result value and the stride are stored in the value field and the stride field in the classification table, respectively. The state is changed to the classified state. If an instruction is in the classified state, another stride is calculated. If the value of the first stride is equal to the second and the value is zero, the prediction type of the instruction is set to the Last type. The prediction type and the predicted value are sent to VPB. Next time, the instruction is predicted by using the Last predictor. If the value of the stride is not zero, the type of the table is set to the Stride type. If the value of the first stride is not the same as the second stride, the instruction is classified as the Two-level type. Once the prediction type of the instruction is set, the predicted instruction is removed from the classification table. The value is predicted by using the corresponding prediction table specified in the prediction type. If the value of the confidence counter becomes zero after several mispredictions, the predictor can no longer accurately predict a value 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 major differences from the one used in Rychlik et al. [19]. First, for the classified type, the decoupled scheme does not need another table because the type is stored in VPB. 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 accesses.

5. Evaluation methodology

In this section, we describe our performance evaluation methodology. First, we explain the machine model used in our simulation study. Then, we describe the simulation set up and the benchmark programs used.

5.1. Machine model

We use two baseline architectures according to the instruction issue width, which are 8-issue and 16-issue. Table 1 summarizes some of the parameters used in our baseline architectures.

The model has a high instruction fetch bandwidth and a large instruction window as well as large on-chip and off-chip caches. The hybrid branch predictor uses a gshare branch predictor [3] with 16K entries and 8-bit history, and a 16K-entry bimodal branch predictor [15]. This hybrid branch predictor is accessed multiple times in one clock cycle to obtain the multiple-branch prediction needed in the baseline architectures and the trace processors.

Table 1. Machine configuration for baseline architectures

	Parameter	Value	Comments
Processor Core	RUU size	256	Instruction window
	LSQ size	64	Load-store queue
	Fetch w idth	8-16 instr./cycle	8-16 issue baseline
	Decode width	8-16 instr./cycle	8-16 issue baseline
	Issue width	8-16 instr./cycle	8-16 issue baseline
	Commit width	8-16 instr./cycle	8-16 issue baseline
	Functional units	10-20 int alu(1), 4-8 fp add(2), 2-4 int mult(3) div(12), 2-4 fp mult(4) div(12),	8-16 issue baseline, ( ) is latency
Branch Predictor	BTB	2K, 2-way	Branch target buffer
	Hybrid Branch Predictor	gshare 16K, 8bit history bimodal 16K	Gshare + Bimodal
	RAS	32	Return addr. stack
Cache	L1 D- cache	512K, 2-way, 32 byte blocks, 3-cycles latency	Data cache
	L1 I- cache	512K, direct-map, 32 byte blocks	Instruction cache
	L2 cache	unified, 1M, 4-way, 64 blocks	Secondary cache

Table 2. Trace cache and value predictor configuration

	Configuration
Trace cache	512K, 4-way, 16 instructions trace cache block
Hybrid Value Predictor	8K VHT(Value History Table) 8K PHT(Pattern History Table) 4 history data values
Decoupled Value Predictor	4K last table, 1K stride table, 8K two-level table, 1K classification table, 32 queue size

The baseline architectures also constitute the "slow path" [18] in the trace processor when the trace cache misses.

The parameters used in our trace cache and the proposed hybrid value predictor are listed in Table 2. The trace cache has 512KB. It is a 4-way associative cache with 16 instructions per block. We have no limitation on the number of branch instructions in each trace cache 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. These table sizes are derived from the dynamic classification rates of the predicted instructions (see the measured rates in section 6.4). The overall table size of our predictor is roughly 7% larger than the size of a 8K-entry hybrid predictor.

5.2. Evaluation method

By augmenting the baseline architectures with the trace cache and the value predictor, we compare performance for the following configurations:

Base : Baseline architectures
Conv : Base + hybrid value predictor described above
Tc : Base + Trace cache without value prediction
ConvTc : Tc + hybrid value predictor described above
Decl : Tc + our proposed decoupled value predictor

The hybrid data predictor used for Conv and Decl has no limitation on the number of accesses to the value prediction table. The access to the prediction tables for Decl is limited by 2 read ports and 2 write ports.

The instruction pipeline of our machine model is based on SimpleScalar with fetch, dispatch, issue, execution, writeback and commit stages [1]. The model supports out-of-order issue and execution, based on Register Update Unit (RUU). RUU is the unit which combines the instruction window for instruction issue and the reorder buffer for instruction commit [12]. The value predictor updates the prediction tables with the computed value after the writeback stage. The value misprediction penalty of the model is one clock cycle. When a value misprediction detected in the writeback stage, only its data-dependent instructions which have already been issued are reissued. We use selective issue core model [20], in which the speculative result is not broadcast on the result bus.

Simplescalar/Alpha 3.0 tool set is used [1]. We add the value predictors, the trace cache and our decoupled scheme on the Simplescalar's sim-outorder simulator. The SPECint95 used in our simulation is listed in Table 3. The table shows the input, the number of instructions executed, the percentage of register-write instructions that need value prediction, the IPC of the baseline architectures, and the trace cache miss rates for each program.

Table 3. SPECint95 benchmark set

Program	Input	# of instr. executed	% of reg. write instr.	IPC of Base		TC miss rate
				8-issue	16-issue	
compress	10000 e 2231	35,726,960	71%	2.56	2.55	21%
gcc	jump.i	40,268,745	71%	1.97	2.07	64%
go	5 9	81,818,896	83%	1.69	1.74	66%
ijpeg	tinyrose.ppm	74,807,331	75%	3.04	3.30	41%
li	queen6.lsp	41,712,945	66%	1.87	1.98	41%
m88ksim	dhry.big.100	101,391,205	73%	2.87	3.00	32%
perl	scrabbl.in	40,465,023	67%	2.38	2.48	60%
vortex	persons.250	65,086,221	61%	3.38	3.84	61%
average		60,159,666	71%	2.47	2.62	48%

6. Performance analysis

In this section, we examine the performance gain using the proposed schemes. We also investigate the value prediction accuracy, the amount of selective prediction, the characteristics of the dynamic classification, queue size, and the performance impact from the predictor table size.

6.1. Speedup

Figure 2 shows the speedup over the baseline architectures. For 8-issue machines, the average speedups are 7.2% for Conv, 7.8% for Tc, 12.9% for ConvTc, and 13.3% for Decl. For 16-issue machines, the average speedups are 9.9% for Conv, 10.8% for Tc, 18.9% for ConvTc, and 18.2% for Decl. These results show that the machines with the trace cache and the proposed value predictor can attain the best performance.

Considering only the effect of value prediction for ConvTc and Decl: ConvTc has the average speedup of 4.6% and 7.3% over Tc; and Decl has the average speedup of 5.0% and 6.7% over Tc for 8-issue and 16-issue, respectively. Note that ConvTc assumes no limitation on the number of accesses to the value prediction tables in each cycle, whereas our decoupled prediction scheme allows only two reads and two writes per cycle to each prediction table. We can see that our more realistic decoupled scheme can reach a similar performance level as the unlimited case. In the case of ijpeg in ConvTc, the value prediction actually degrades the performance by 12%. This is due to the high value misprediction rate (48%) of ijpeg in ConvTc. In m88ksim, it shows the best speedup for ConvTc and Decl because it has very high prediction accuracy and relatively low trace cache miss rates (32%). Go shows bad performance for Decl due to high miss rates (66%) in the trace cache. One interesting result is that the average speedup of Conv (the value prediction effect) is close to the average speedup of Tc (the trace cache effect).

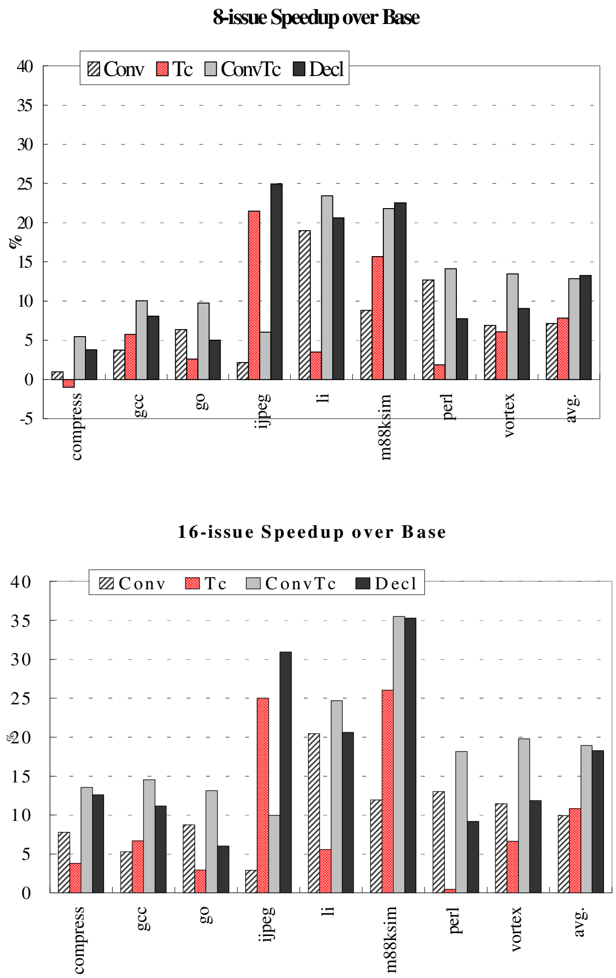

Figure 2. The speedup over the baseline architectures

6.2. Value prediction accuracy

There are two measurements for the correctness of a value prediction. *Prediction rate* is the percentage of correct predictions out of all instructions which perform value predictions (i.e. all register-write instructions). *Prediction accuracy* is the percentage of correct predictions out of actually attempted predictions (i.e. they exclude those predictions which encounter below threshold confidence counter values or trace cache misses). We should measure the performance based on the prediction accuracy, not the prediction rate because it is the prediction accuracy which affects the final performance. Figure 3 shows the prediction accuracy for the hybrid predictor and the proposed decoupled predictor.

The average prediction accuracy is 67% for the hybrid predictor and 77% for the decoupled predictor. Ijpeg using the hybrid predictor has a prediction rate of about 52%. In this case, aggressive fetching mechanisms such as trace cache make the performance worse because of value misprediction penalty.

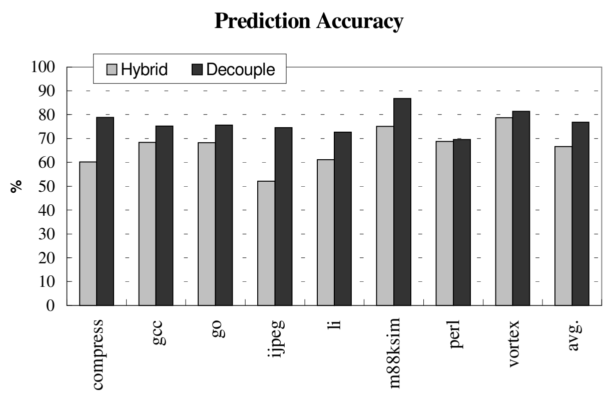

Figure 3. The prediction accuracy for the hybrid predictor and the decoupled predictor

6.3. Selective prediction

In our proposed decoupled scheme, not all register-write instructions need to look up the VPB and the prediction tables. Only the load instructions and the instructions that have data-dependent instructions in the same trace cache block need value prediction. The percentage of the instructions which require value prediction table lookup is 74.0%. The percentage of register-write instructions which have no data dependence in the same trace cache block is 19.6%. This allows us to eliminate 19.6% of unnecessary value predictions. The performance of our proposed decoupled scheme ties closely to the hit rate of the trace cache because we cannot access the VPB when there is a trace cache miss. There is also no value prediction support when the instrucitons are executed in the "slow path". The average percentage of no-lookup to VPB due to trace cache misses is 6.5%.

Figure 5 shows the average number of instructions which require value prediction within a trace cache block for each program. The overall average is 7.4 instructions

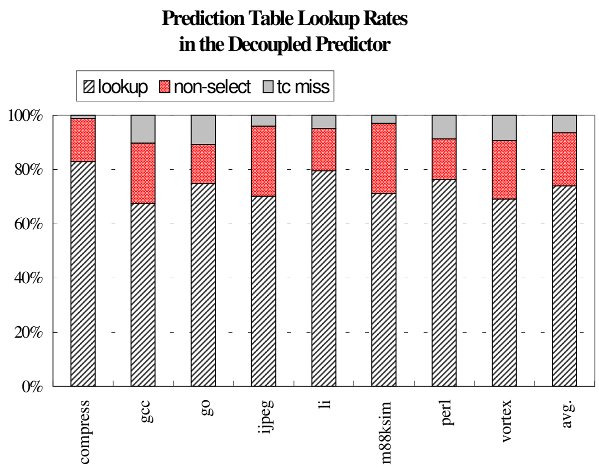

Figure 4. Prediction table lookup rates in decoupled Predictor.

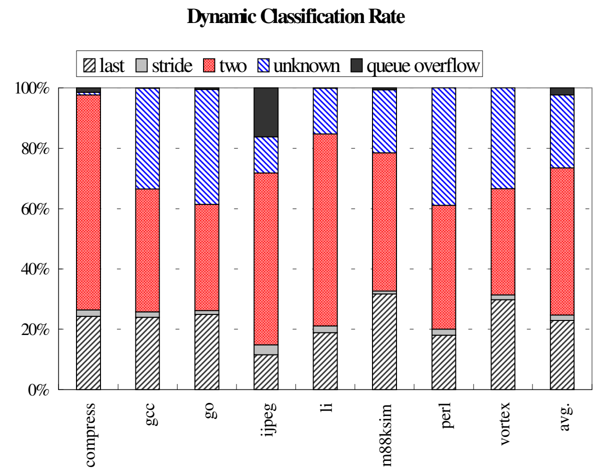

Figure 6. The distribution of the prediction types in the decoupled predictor.

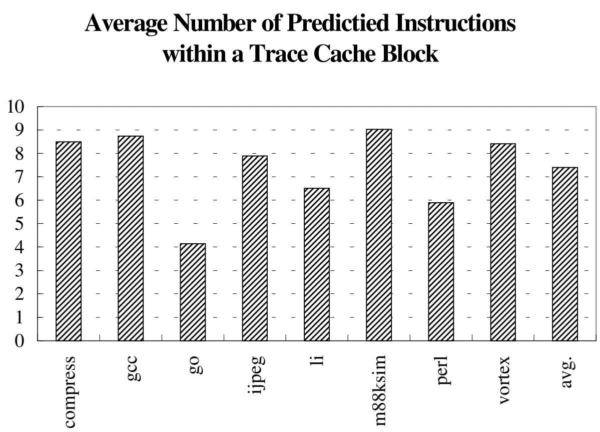

Figure 5. Average number of predicted instructions within a trace cache Block.

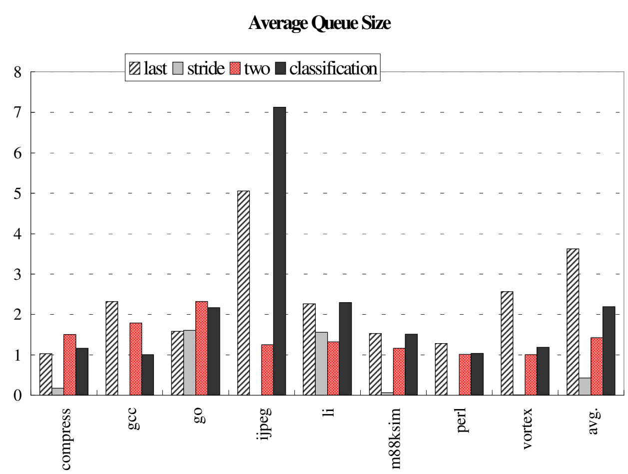

Figure 7. Average queue size in decoupled predictor

for the trace cache with the block size of 16 instructions.

6.4. Dynamic classification

Figure 6 shows the distribution of the prediction types in the dynamic instruction stream using the proposed decoupled predictor. On average, 48.7% of the instructions are classified as Two-level. Note that the instructions are classified as Two-level only when both Last and Stride fail to predict correctly. The percentages of the instructions classified as Last and Stride are 22.9% and 1.9%, respectively. 24.1% of the instructions are classified as unknown.

6.5. Queue size

In the 16-issue configuration, the average queue sizes are 3.6 for Last, 0.4 for Stride, 1.4 for Two-level, and 2.2 for Classification (Figure 7). The average queue sizes are all very small. This indicates that the decoupled predictor effectively distributes the instructions to each table and thus can match the high instruction issue rate of a wide-issue trace processor. Except for ijpeg, the overflow rate for each queue is about or below 1%. But the overflow rate for ijpeg can be as high as 16.2%.

6.6. Predictor table size and trace cache size

Figure 8 shows the average IPC with the predictor sizes of 2K, 4K, and 8K entries for 16-issue configuration. We can see that Decl is less sensitive with the size than Conv.

Figure 9 shows the average IPC of the trace cache with 512KB and 256KB, respectively. The average miss rates for the trace cache are 48% for 512KB and 57% for 256KB. In Figure 9, for 8-issue, 512KB trace cache is the performance loss with a 256KB trace cache over a 0.8% for Tc, 0.4% for ConvTc, and 1.4% for Decl. For 16-issue the losses are 1% for Tc, 0.3% for ConvTc, and 1.9% for Decl. The loss of our decoupled scheme is greater because our scheme is more sensitive to the trace cache miss rate as described earlier.

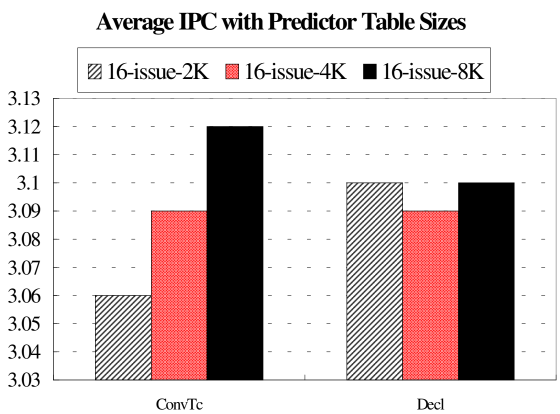

Figure 8. Average IPC with the predictor sizes

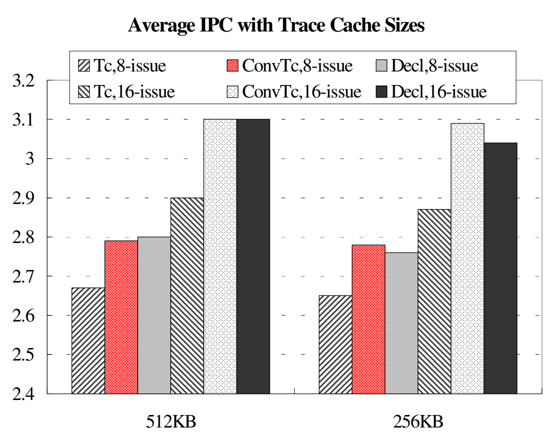

Figure 9. Average IPC for 512KB and 256KB trace cache

7. Conclusions

In this paper, we look at implementation issues regarding multiple value predictions required on wide-issue processor architectures, such as trace processors. By augmenting trace cache with a VPB (Value Prediction Buffer) and decouple value prediction from the instruction fetch stage as well as providing multiple value prediction tables and queues, we can significantly reduce the number of accesses to each value prediction table. It also removes the value prediction from the critical path in the instruction fetch stage. We obtain some performance results using Simplescalar Alpha 3.0 tool set.

The decoupled value prediction scheme provides average speedup of 18.2% over a baseline architecture without trace cache, and 6.7% over a baseline architecture with trace cache on the SPECint benchmarks. We assume 2 read ports and 2 write ports to each value prediction table. This performance gain is as high as using conventional value prediction schemes with unlimited bandwidth to the value prediction tables. We find that most benchmarks have a fairly smooth distribution of desired value prediction schemes. This leads to small average queue sizes and dynamically infrequent overflow conditions. We also show that the decoupled value prediction scheme is not very sensitive to the prediction table size. Because we decouple value prediction from the instruction fetch stage, register-write instructions which are the only instructions requiring value prediction can be identified. This can cut down the unnecessary value prediction by 20%. These results suggest the proposed scheme can effectively handle multiple value predictions on wide-issue superscalars, such as trace processors, with a more realistic hardware implementation.

Acknowledgments

We would like to thank anonymous reviewers for their useful comments.

This work is supported in part by the National Science Foundation under Grant Nos MIP-9610379 and MIP-9971666; and by a gift from Intel Corporation.

Sang-Jeong Lee is supported by the visiting scholar program from Soonchunhyang 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] B.Calder, G.Reinman and D.Tullsen, "Selective Value Prediction", *Proceedings of the 26th International Symposium on Computer Architecture(ISCA-26)*, May 1999.

[3] P.Chang, E.Hao, and Y.Patt, "Alternative Implementations of Hybrid Branch Predictors," *Proceedings of the 28th International Symposium on Mi-*

croarchitecture(MICRO-28), 1995.

[4] D.Friendly, S.Patel, and Y.Patt, "Alternative Fetch and Issue Policies for the Trace Cache Fetch Mechanism," *Proceedings of the 30th International Symposium on Microarchitecture(MICRO-30)* ,Nov, 1997.

[5] D.Friendly, S.Patel, and Y.Patt,"Putting the Fill Unit to Work : Dynamic Optimizations for Trace Cache Microprocessors," *Proceedings of the 31st International Symposium on Microarchitecture (MICRO-31)*, Dec 1998

[6] F.Gabbay, and A.Mendelson, "Speculative Execution Based on Value Prediction", *EE Department TR 1080*, Technion– Israel Institute of Technology, Nov. 1996

[7] 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.

[8] A.Gonzalez, J.Tubella, and C.Molina, "Trace-Level Reuse", *Proceedings of the International Conference on Parallel Processing*, Sept. 1999

[9] J.Gonzalez and A.Gonzalez, "The Potential of Data Value Speculation to Boost ILP", *Proceedings of the International Conference on Supercomputing*, 1998.

[10] J.Huang and D.Lilja, "Exploiting Basic Block Value Locality with Basic Block Reuse", *Proceedings of the 5th International Symposium on High Performance Computer Architecture(HPCA-5)*, 1999.

[11] Q.Jacobson, E.Rotenberg, and J.E. Smith, "Path - Based Next Trace Prediction", *Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30)*, pp. 14-23, Dec. 1997.

[12] M.Johnson, *Superscalar Microprocessor Design*, Prentice Hall, 1991

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

[14] M.Lipasti, *Value Locality and Speculative Execution*, Ph.D. thesis, Carnegie Mellon University, April 1997.

[15] S.McFaring, "Combining Branch Predictors", *Technical Report TN-36*, Digital Equipment Corporation, Western Research Lab, June 1993

[16] G. Reinman, T. Austin, and B. Calder, "A Scalable Front-End Architecture for Fast Instruction Delivery," *Proceedings of the 26th Annual International Symposium on Computer Architecture(ISCA-26)*, May 1999.

[17] E. Rotenberg, S.B., and J.Smith, "Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching", Technical Report: University of Wisconsin - Madison Technical Report #1310, Apr. 1996.

[18] E.Rotenberg, Q.Jacobson, Y.Sazeides, and J. Smith, "Trace Processors",*Proceedings of the 30th International Symposium on Microarchitecture(MICRO-30)*, pp. 138-148, Dec. 1997.

[19] 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.

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

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

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

[23] T. Yeh and Y.Patt, "Two-Level Adaptive Branch Prediction," *Proceedings of the 24th International Symposium Microarchitecture(MICRO-24)*, Nov. 1991.