Low-Cost Value Predictors Using Frequent Value Locality

Toshinori Sato†‡ and Itsujiro Arita†

†Department of Artificial Intelligence
‡Center for Microelectronic Systems
Kyushu Institute of Technology
tsato@ai.kyutech.ac.jp

Abstract. The practice of speculation in resolving data dependences has been recently studied as a means of extracting more instruction level parallelism (ILP). Each instruction's outcome is predicted by value predictors. The instruction and its dependent instructions can be executed in parallel, thereby exploiting ILP aggressively. One of the serious hurdles for realizing data speculation is the huge hardware budget required by the predictors. In this paper, we propose techniques that exploit frequent value locality, resulting in a significant budget reduction. Based on these proposals, we evaluate two value predictors, named the *zero-value predictor* and the *0/1-value predictor*. The zero-value predictor generates only value 0. Similarly, the 0/1-value predictor generates only values 0 and 1. Simulation results show that the proposed predictors have greater performance than does the last-value predictor which requires a hardware budget twice as large as that of the predictors. Therefore, the zero- and the 0/1-value predictors are promising candidates for cost-effective and practical value predictors which can be implemented in real microprocessors.

1 Introduction

Modern microprocessors boost performance by exploiting instruction level parallelism (ILP). However, several obstacles limit ILP. These include dependences between instructions and are classified into three classes — control, name, and data dependences. This paper focuses on reducing data dependences. Recent studies have shown that these data dependences can be speculatively resolved by value prediction. An outcome of an instruction is predicted by means of value predictors, and the instruction and its dependent instructions can then be executed in parallel, thereby exploiting ILP aggressively.

However, there is a serious hurdle for exploiting data speculative execution in real microprocessors. In order to efficiently utilize value prediction without incurring misspeculation penalties, value predictors must operate at high predictability. Some predictors such as hybrid[21] and context-based[18] predictors achieve significantly high predictability rates at considerable hardware cost. This hardware cost is one of the serious hurdles for realizing data speculation, and

thus it should be alleviated. Furthermore, even embedded microprocessors support out-of-order execution[10]. Thus, low-cost value predictors are required.

Many studies[3–5, 12–15] have been devoted to reducing the hardware cost of value predictors. One way of reducing this cost is through the use of simple circuits. For example, direct-mapped tables are smaller than highly associative ones. Reducing the number of entries in the value prediction table by carefully selecting predicting instructions[4, 5, 12, 13] is also a simple technique to reduce cost. Tag array size can be saved by employing partial resolution, using fewer tag address bits than necessary to uniquely identify every instruction[15]. Full resolution of value predictors is not necessary since they do not have to be correct all the time. Data array size can be reduced if the value predictor keeps only low-order bits of data values, exploiting narrow width values. It has been found that across SPECint95 benchmark programs, over 50% of integer operands are 16 bits or less[1]. Therefore, it is possible to keep only low-order bits of data values in the value predictor in order to reduce its data array size[3, 14]. In this paper, we propose an alternative technique to obviate the data array cost.

The organization of the rest of this paper is as follows: Section 2 surveys related work. Section 3 describes our evaluation environment. Section 4 presents how frequent value locality is found and proposes cost-effective value predictors which are evaluated in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

Value prediction[8, 11] is a speculative technique which executes instructions using predicted data values. Data dependences are speculatively resolved and thus ILP is increased. Many studies have proposed value prediction mechanisms, some of which achieve predictability rates as high as 80%[18, 21]. However, these predictors such as 2-level, hybrid, and context-based predictors require considerable hardware cost for realizing their high predictabilities.

Morancho et al.[12], Rychlik et al.[13], Del Pino et al.[5], and Calder et al.[4] have examined the capacity constraints of value predictors. Morancho et al.[12], Rychlik et al.[13], and Del Pino et al.[5] proposed to reduce hardware cost by classifying instructions based on their value predictability. Easily predictable instructions use simpler predictors such as last-value and stride predictors, whose hardware cost is low. High-cost predictors such as 2-level and the context-based predictors are used only for hard-to-predict instructions. Calder et al.[4] proposed filtering instructions based on their level of criticality. Only instructions regarding critical paths are held in the value predictor, resulting in capacity saving. On the other hand, Fu et al.[7] and Tullsen et al.[20] remove value prediction hardware completely with the aid of the compiler management of values in registers.

Partial resolution is first evaluated in the research area of branch prediction. Fagin[6] proposed to reduce the tag array bitwidth in branch target buffers (BTBs). He investigated the tradeoff between bitwidth and branch prediction accuracy, and found that it is possible to restrict the bitwidth of the tag array

to 2 bits without severe performance loss. We evaluated the partial resolution of several value predictors[15] and found that no tag bits are required for the last-value predictor and that only 2 bits are required for the hybrid predictor.

Recently, we examined a technique for reducing hardware cost by exploiting narrow width values[14]. Brooks et al.[1] found that across SPECint95 benchmark programs, over 50% of integer operands are 16 bits or less. That is, only low-order bits of the value prediction table are utilized. Therefore, we propose to keep only low-order bits of data values in the prediction table in order to reduce its hardware cost. Simulation results showed that performance improvement due to data speculation is maintained accompanied by the hardware cost savings of 45.1%[14]. A similar technique was proposed by Burtscher et al.[3], where infrequently changing high-order bits are shared by several values held in the prediction table.

3 Evaluation Environment

In this section, we describe our evaluation environment by explaining a processor model and benchmark programs.

3.1 Processor Model

We use two types of simulators for this study. One is a functional simulator for evaluating predictability and the other is a timing simulator for evaluating processor performance. The timing simulator models wrong path execution caused by misspeculations. We implemented the simulators using the SimpleScalar tool set (ver.3.0a)[2]. SimpleScalar/PISA instruction set architecture (ISA) is based on MIPS ISA.

The timing simulator models a realistic 8-way out-of-order execution superscalar processor based on a register update unit[19] which has 128 entries. Each functional unit can execute any operation. The latency for execution is 1 cycle except in the cases of multiplication (4 cycles) and division (12 cycles). A 4-port, non-blocking, 128KB, 32B block, 2-way set-associative L1 data cache is used for data supply. It has a load latency of 1 cycle after the data address is calculated and a miss latency of 6 cycles. It has a backup of an 8MB, 64B block, direct-mapped L2 cache which has a miss latency of 18 cycles for the first word plus 2 cycles for each additional word. A memory operation that follows a store whose data address is unknown cannot be executed. A 128KB, 32B block, 2-way set-associative L1 instruction cache is used for instruction supply and also has the backup of the L2 cache which is shared with the L1 data cache. For control prediction, a 1K-entry 4-way set associative BTB, a 4K-entry gshare-type 2-level adaptive branch predictor, and an 8-entry return address stack are used. The branch predictor is updated at the instruction commit stage.

For the purpose of comparison with our proposal, we also evaluate the last-value predictor[11]. Its main structure is the Value History Table (VHT). The VHT has `Tag`, `Data Value`, and `Conf` fields. The `Tag` field is for identifying each

instruction. As determined by a previous study[15], the `Tag` field does not always contribute to predictor performance and thus we omit it in this study. The `Data Value` field stores the last result of the associated instruction. When the same instruction is encountered, the `Data Value` is used for its predicted value. The `Conf` field is a saturating up-down counter that determines whether oa not the instruction should be predicted. The counter is incremented or decremented whenever a prediction is correct or incorrect, respectively. In our study, 2-bit saturating counters are used. The threshold value necessary to trigger a prediction is 2. We use a direct-mapped table for the predictor, since we are interested in cost-effective predictors.

When a misspeculation occurs, it is necessary to revert the processor state to a safe point where the speculation is initiated. We use an instruction reissue mechanism which selectively flushes and reissues misspeculated instructions. We have already proposed a practical instruction reissue mechanism[17]. When comparing an actual value with its associated predicted value, a one cycle penalty of the comparison stage is included when every mispredicted instruction is reissued.

3.2 Benchmark Programs

This study uses the SPECint95 benchmark suite. We focus on improving the performance of only integer programs because it tends to be more difficult to obtain high levels of parallelism from these types of programs than from floating-point programs. The input files are modified to achieve a practical evaluation time. We use the object files provided by the University of Wisconsin-Madison[2]. Each program is executed to completion. The candidate instructions for value prediction are register-writing ones, and did not include branch and store instructions. We counted only committed instructions.

4 Cost-Effective Value Predictors

Frequent value locality[22] is a new kind of locality, and is distinct from traditional value locality[11] in that the former is observed across several instructions and the latter is defined in each instruction. This section investigates how frequent values are observed and proposes cost-effective value predictors.

4.1 Frequently Occurring Values

Zhang et al.[22] observed that at any given point in the program's execution, a small number of distinct values occupy a vast fraction of referenced locations, when the values involved in memory accesses are traced. Furthermore, values 0 and 1 are the most frequently accessed values in the case of SPECint95 programs. We extend their study to trace all register-writing instructions. The simulation results are summarized in Figure 1. Each bar is divided into two parts. The bottom part (black) indicates the percentage of dynamic instructions which generate the value 0. The top one (gray) is for the value 1. On average, 15.3% of dynamic

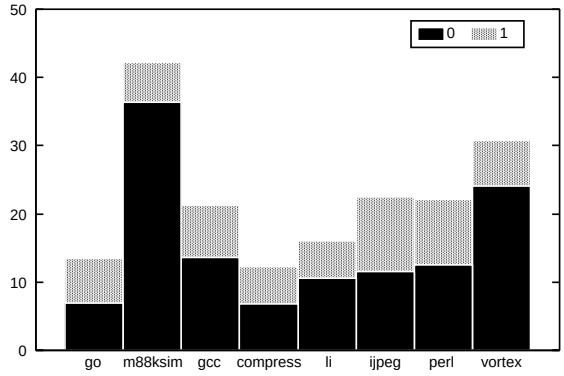


Fig. 1. (%)Frequent value locality

instructions generate the value 0. While this is much smaller than the frequency in memory access, which is nearly 50%, this frequency is still large enough to exploit these characteristics. In addition, the percentage of dynamic instructions which generates the value 1 is 7.3% on average. Tables 1 and 2 summarize instructions which frequently generate the values 0 and 1, respectively, for each benchmark program. The first column is the program name. The second column presents the top five instructions which frequently produce the value 0 or 1. The next one indicates the percent occupancy that is defined as the number of the associated instructions over that of all dynamic instructions which generate the value. The last column is percent frequency, meaning how frequently the associated instruction generates the value. This can be explained by using the case of `099.go`. In this case, 34.79% of all dynamic instructions which produce the value 0 is instruction `lw`. In addition, 9.07% of all dynamic instructions `lw` generates the value 0. As can be easily seen, the instruction `lw` most frequently produces the value 0. The next frequent instruction is instruction `addu`. It is interesting that any subtract instructions do not appear in Table 1. It is also interesting that logical instructions occupy small parts. In contrast, logical instructions frequently generate value 1 as can be seen in Table 2. In the following subsection, we propose cost-effective value predictors utilizing the localities of values 0 and 1.

4.2 Zero-Value Predictor

Since the value 0 is the most frequent value generated, processors may benefit from a value predictor which provides only value 0. We call this predictor *zero-value predictor*[16]. It can be implemented by removing the `Data Value` field completely from the VHT. That is, the zero-value predictor consists of only `Conf` and `Tag`[1] fields. This significantly reduces the hardware cost of the predictor.

[1] As we will see in Section 5.1, the `Tag` field is optional.

Table 1. Instructions frequently generating value of zero

program	inst.	%occu.	%freq.	program	inst.	%occu.	%freq.
099.go	lw	34.79	9.07	130.li	lw	49.86	11.58
	slt	27.05	51.92		addu	23.23	15.99
	addu	18.15	5.42		andi	12.04	63.56
	sll	5.70	2.93		lbu	7.61	26.76
	alti	5.46	37.62		addiu	1.81	0.85
124.m88ksim	lw	16.71	29.69	132.ijpeg	lw	22.64	9.84
	addu	16.67	45.55		addu	15.86	5.64
	sll	13.83	98.82		slt	15.15	57.81
	and	11.07	56.89		sll	11.22	11.33
	andi	8.36	37.48		sltu	10.72	47.78
126.gcc	lw	32.93	14.30	134.perl	lw	30.98	11.02
	addu	23.79	19.27		addu	18.41	12.75
	sltiu	6.23	41.83		slt	8.48	74.71
	sll	6.23	14.96		andi	6.79	50.39
	slt	5.21	45.91		sltu	5.17	25.32
129.compress	lw	22.20	5.84	147.vortex	lw	53.04	27.54
	slti	18.10	45.61		addu	37.06	41.26
	slt	17.46	31.19		sltu	2.58	37.42
	sltu	7.54	42.00		andi	1.39	16.19
	and	7.23	31.55		lbu	1.13	41.54

4.3 0/1-Value Predictor

Figure 1 shows that the value 1 also occurs frequently. Hence, it is expected that applying the frequent one-value locality by building the *0/1-value predictor* improves the value predictabilities. In order to distinguish the values 0 and 1, the 0/1-value predictor should have a 1-bit `Data Value` field in the VHT. In addition, its replacement policy becomes a complex issue. When an instruction produces the value 0 or 1, the data value field simply holds the value. However, if the generated value is neither 0 nor 1, we can determine a replacement policy according to many choices. In this paper, we use a simple replacement policy. Only when the generated value is 1 does the data value field in the VHT keep the value 1. Otherwise, it keeps the value 0. In other words, the 0/1-value predictor has a priority on the value 0. This decision is proper since the value 0 is the most frequently generated value, as we have already seen in Figure 1.

5 Evaluation

In this section, we present simulation results. First, we evaluate predictability. We define predictability as the number of instructions that are (correctly and incorrectly) predicted by a value predictor over that of all register-writing instructions. We also define prediction coverage as the number of instructions

Table 2. Instructions frequently generating value of one

program	inst.	%occu.	%freq.	program	inst.	%occu.	%freq.
099.go	lw	30.57	7.56	130.li	sltu	22.18	90.51
	slt	26.40	48.08		sltiu	16.63	95.05
	addiu	25.04	8.76		lbu	16.62	30.36
	slti	9.54	62.38		slti	16.13	100.0
	addu	3.20	0.91		andi	8.56	23.48
124.m88ksim	sltu	33.32	66.15	132.ijpeg	slti	26.17	87.34
	slti	33.27	99.93		lw	19.94	10.86
	addiu	33.18	16.34		addiu	19.65	12.78
	sra	0.07	0.39		sltu	9.35	52.22
	lw	0.03	0.01		slt	8.82	42.19
126.gcc	addiu	20.93	7.17	134.perl	sltiu	20.01	88.54
	lw	18.69	4.54		sltu	19.08	74.68
	sltiu	15.50	58.17		addiu	18.84	9.30
	slti	15.15	71.08		lw	10.69	3.04
	slt	10.99	54.09		lbu	6.59	31.95
129.compress	slt	45.45	68.81	147.vortex	addiu	52.04	18.03
	slti	25.47	54.39		sltu	15.49	62.58
	sltu	12.30	58.00		lw	11.26	1.61
	addiu	4.95	1.14		lhu	6.51	35.90
	sra	2.57	8.61		addu	5.50	1.71

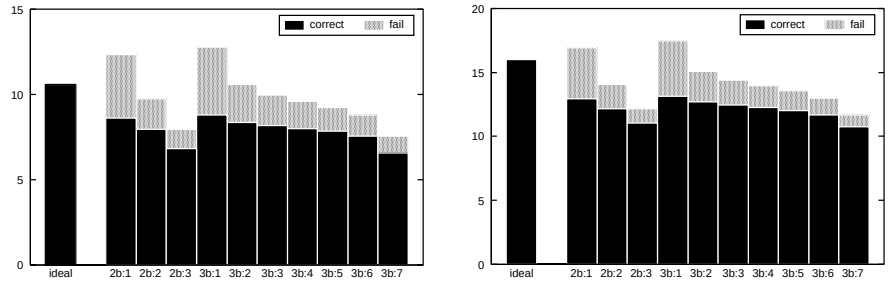
correctly predicted over that of all register-writing instructions. On the other hand, prediction accuracy, which is the percentage of instructions correctly predicted over all predicted instructions, can be easily obtained using the following equation.

$$(Prediction\ accuracy) = \frac{(Prediction\ coverage)}{(Predictability)}$$

After that, processor performance is evaluated.

5.1 Predictability

There is a tradeoff between predictability and prediction accuracy. If every instruction initiates value prediction in order to increase predictability, hard-to-predict instructions must be included and thus the prediction accuracy is degraded. Therefore, only easily predictable instructions should be carefully selected. In the cases of the zero- and the 0/1-value predictors, their predictabilities are severely limited since they can predict only the value 0 or 1. And thus, the tradeoff point in the predictors is more difficult to find than in any conventional value predictors. From these considerations, we investigate how a confidence mechanism affects predictabilities. After the confidence mechanism is determined, we evaluate the relationship between the table capacity and pre-



(i) zero-value (ii) 0/1-value

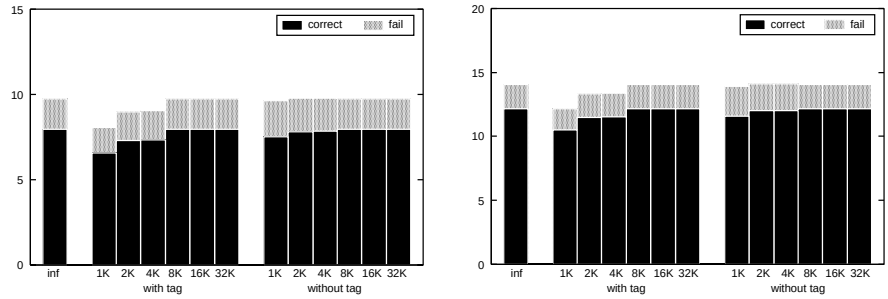
Fig. 2. (%)Predictability (infinite capacity): 130.li

dictabilities. Please remember that we use the functional simulator for evaluating predictabilities.

Confidence We evaluate 2-bit and 3-bit saturating up-down counters for the confidence mechanism. The threshold value for triggering every prediction varies between 1 and 3 for the 2-bit counter and between 1 and 7 for the 3-bit counter. Figure 2 summarizes the results for the zero- and the 0/1-value predictors. Due to space limitation, only results for `130.li` are shown. It should be noted that the VHT has infinite capacity. The left-most bar indicates the upper boundary for the predictability which is determined by the frequent value localities observed in Section 4.1. The remaining bars indicate the predictabilities and the prediction coverages for all confidence mechanisms evaluated. Each simulation result is indicated by a group consisting of the bit length of the counter and the threshold value. For example, if a simulation result is denoted as `3b:5`, it presents the confidence mechanism as the 3-bit counter, and the threshold value is 5. Each bar is divided into two parts. The lower part (black) indicates the percentage of the instructions whose data value is correctly predicted. The upper part (gray) indicates the percentage that is mispredicted. That is, the lower part is the prediction coverage, while the sum of the two parts is the predictability.

We can find the following. Figure 2 indicates that, first, there is no significant difference between the 2-bit and the 3-bit counters. Thus, in regard to hardware cost efficiency, we select the 2-bit counter for the confidence mechanism. Second, in the case of a threshold value of 1, the percentage of misprediction is considerably high. On the other hand, a threshold value of 3 limits predictability. These observations indicate that the threshold value is 2. Throughout the rest of this paper, the 2-bit saturating up-down counter with a threshold value of 2 is used.

Capacity Next, we present how table capacity affects predictabilities. Figure 3 summarized the results. The left-most bar indicates the case of infinite capacity,

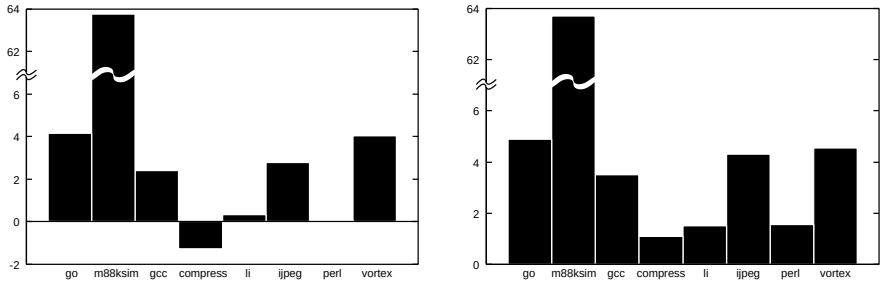


(i) zero-value (ii) 0/1-value

Fig. 3. (%)Predictability (finite capacity): 130.li

which we have seen above. The remaining bars indicates the cases of finite capacity tables. For each group of six bars, the bars from left to right are for the results of 1K-, 2K-, 4K-, 8K-, 16K-, and 32K-entry VHTs, respectively. Each bar is divided into two parts as in Figure 2. The left group presents the results of the VHT which has the `Tag` field, and the right one presents that of the VHT whose `Tag` field is removed. We would like to consider the VHT without tag because the evaluated predictors are special kinds of last-value predictor and hence it is expected that the tag contributes little to the predictabilities and processor performance improvement as we have already seen in [15]. It is also desirable that removing the tag further significantly reduces the hardware cost. However, the tagless VHT increases aliasing. Different instructions share a single predictor entry. In general, an aliasing may be constructive, destructive, or neutral. However, in the case of the zero-value predictor, the predicted value is only 0 and hence destructive aliasing rarely occurs. In order to increase constructive aliasing, the zero-value predictor can consist of two individual tables, one of which is for high confidence instructions while the other is for low confidence ones, such as the bi-mode branch predictor[9]. On the other hand, the 0/1-value predictor suffers destructive aliasing more often than does the zero-value predictor since it holds both values 0 and 1 in each entry. This problem also can be mitigated by the using bi-mode predictor, which has two dedicated tables for predicting the values 0 and 1. We leave consideration of the bi-mode prediction for future study.

The following are observed. First, the VHT without tag marks slightly larger predictability than does the one with tag at the risk of large misspredictions, as we expected. Second, for most programs, the predictability of the VHT with 8K-entry is comparable to that of the VHT with infinite capacity. Based on these observations, we determine that the table capacity is 8K. This results in an 8K-entry tagless zero-value predictor whose hardware cost is only 2K bytes. This is approximately equal to the costs of 256- and 512-entry last-value predictors with and without tag, respectively, and is less than 2% in the area of on-chip



(i) 1-cycle delay for comparison (ii) no delay for comparison

Fig. 4. (%)Processor performance improvement (zero-value)

caches in modern microprocessors. On the other hand, the cost for the 8K-entry 0/1-value predictor is 3K bytes since it has a 1-bit `Data Value` field.

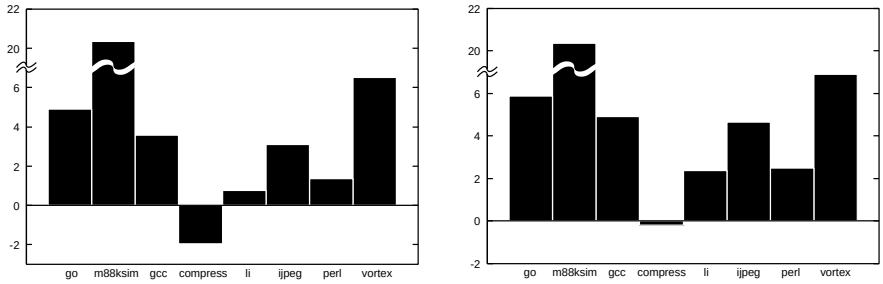
5.2 Processor Performance

In this subsection, we evaluate how much the zero- and the 0/1-value predictors contribute to processor performance. Each predictor has an 8K-entry VHT whose `Tag` field is removed. For measuring performance, we use committed instructions per cycle (IPC). Only useful instructions are considered for counting the IPC. We do not count nop instructions. We define percent speedup as the increased IPC over the IPC of the baseline model. After that, we compare the proposed value predictors with the last-value predictor. It should be noted that a timing simulator is used for evaluating processor performance.

Figure 4(i) presents the percent speedup when the 8K-entry zero-value predictor is utilized. Except for `129.compress`, substantial speedup is observed. Especially in the case of `124.m88ksim`, a 63.8% speedup is achieved. We can see that the speedup figure closely follows the pattern of the frequent value locality depicted in Figure 1. While the zero-value predictor can generate only value 0, data speculation contributes considerably to processor performance. From these observations, it is confirmed that utilizing the frequent zero-value locality is an effective way of exploiting ILP.

Since the zero-value predictor generates only value 0, the comparison between a predicted value (always 0) and its actual value is very simple. Therefore, it may be possible to verify each prediction during the execution stage, and thus the one cycle penalty of each mispredicted instruction can be removed. The results of this scenario are summarized in Figure 4(ii). Processor performance is improved for all programs.

The results for the 0/1-value predictor are presented in Figure 5. While the absolute values of the speedup are different from those of the zero-value predictor, similar tendencies can be observed.



(i) 1-cycle delay for comparison (ii) no delay for comparison

Fig. 5. (%)Processor performance improvement (0/1-value)

Table 3. Hardware cost

predictor type	capacity (entries)	cost (bytes)
zero-value	8K	2K
0/1-value	4K	1.5K
	8K	3K
last-value	512	2K
	1K	4K
	2K	8K

Next, we compare the zero- and the 0/1-value predictors with the last-value predictor which is the simplest value predictor so far. We evaluate six cases. They are summarized in Table 3, which also explains the hardware cost of every predictor evaluated. Note that no predictor has the `Tag` field. We can see that the 8K-entry zero-value predictor is equivalent in hardware cost to the 512-entry last-value predictor, and that the 4K-entry 0/1-value predictor has a smaller hardware cost than does the 512-entry last-value predictor.

The percent of performance improvement is shown in Figure 6. For each group of six bars, the first one indicates the speedup of the 8K-entry zero-value predictor. The second and the third bars indicate those of the 4K- and 8K-entry 0/1-value predictors. The remaining bars from left to right are for the 512-, 1K-, and 2K-entry last-value predictors, respectively. Note that the last-value predictor suffers a one cycle penalty from every mispredicted instruction but that the zero- and the 0/1- value predictors do not. Primary observation is that the zero-value predictor contributes more to processor performance than does the 1K-entry last-value predictor, which is twice as large in hardware cost as the 8K-entry zero-value predictor. Moreover, for most programs, the zero-value predictor is comparable in speedup to the 2K-entry last-value predictor. In the case of `124.m88ksim`, the zero-value predictor overwhelms the last-value

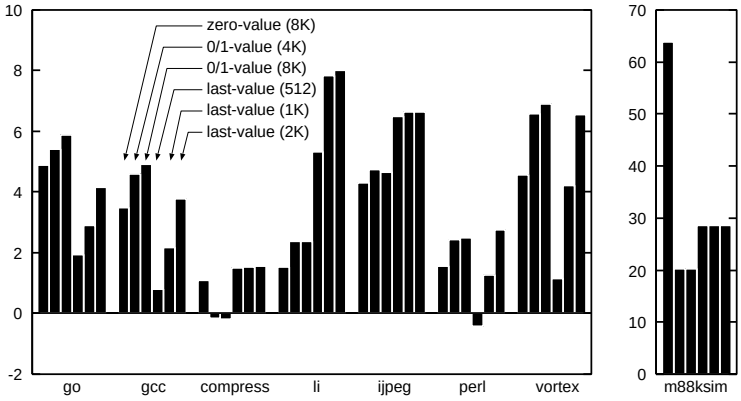


Fig. 6. (%)Comparison with last-value predictor

predictor. This is a good example for explaining the benefit gained from the frequent zero-value locality. Because the last-value predictor evaluated has a quite small capacity, each entry holding the value 0 is replaced by the other values. This diminishes the benefit. On the other hand, the 8K-entry zero-value predictor has sufficient capacity for capturing the frequent zero-value locality. From these observations, we can confirm the cost-efficiency of the zero-value predictor.

The 0/1-value predictor has different characteristics from the zero-value predictor. In the cases of `099.go`, `126.gcc`, `134.perl`, and `147.vortex`, even the 4K-entry 0/1 value predictor attains a speedup comparable to the 2K-entry last-value predictor, which requires a hardware cost five times as large. In the cases of `132.ijpeg` and `124.m88ksim`, the discrepancy of speedup between the 0/1-value and the last-value predictors is modest. These observations confirm that the 0/1-value predictor is also a cost-effective alternative to the zero-value predictor. However, in the cases of `129.compress` and `130.li`, the discrepancy is significant. When we compare the 0/1-value predictor with the zero-value predictor, the most interesting result is found in the case of `124.m88ksim`. While the predictability of the 0/1-value predictor is larger than that of the zero-value predictor and the prediction accuracies of both predictors are almost perfect, the speedup in the former is less than one third of the latter. This is due to the serious destructive aliasing between the values 0 and 1. If the 0/1-value predictor cannot predict instructions producing the value 0 on critical paths while the zero-value predictor can, its speedup is severely diminished. Nonetheless, it is confirmed that the 0/1-value predictor is as cost-effective as the zero-value predictor, since the former is faster in speedup than the latter except for the cases of `129.compress` and `124.m88ksim`.

6 Conclusions

Recently, the practice of speculation in resolving data dependences has been studied as a means of aggressively extracting ILP. One of the serious hurdles for realizing data speculation is the huge hardware budget required by the value predictors. This is because the predictors which achieve high predictability for alleviating misspeculation penalties require considerable hardware cost. This paper uses these observations to explore ways to reduce the hardware budget of data value predictors.

Detailed simulation results show the existence of is frequent value locality in programs. We find that across SPECint95 benchmark programs, on average 15.3% and 7.3% of dynamic instructions generate the value 0 and 1, respectively. Based on this observation, we propose cost-effective value predictors, named the zero-value predictor and the 0/1-value predictor. The zero-value predictor generates only value 0, and the 0/1-value predictor generates both values 0 and 1. The 8K-entry tagless zero- and 0/1-value predictors capture most of the frequent value localities. Their hardware costs are less than 2% in the area of on-chip caches in modern microprocessors. Simulation results demonstrate that the 8K-entry zero-value predictor is faster in processor performance speedup than is the 1K-entry last-value predictor, whose hardware is twice as large as that of the zero-value predictor. Furthermore, the 4K-entry 0/1-value predictor achieves a speedup comparable to the 2K-entry last-value predictor, which is five times as large in hardware cost as is the 0/1-value predictor. These observations confirm that the zero- and the 0/1-value predictors are promising candidates for cost-effective and practical value predictors which can be implemented in real microprocessors in the near future.

Future study includes investigation and evaluation of the bi-mode zero- and 0/1-value predictors for reducing mispredictions caused by the destructive aliasing.

Acknowledgments

This work is supported in part by a Grant-in-Aid for Encouragement of Young Science (#12780273) and a Grant-in-Aid for Scientific Research (B) (#12780273) from the Japan Society for the Promotion of Science.

References

1. Brooks D., Martonosi M.: Dynamically exploiting narrow width operands to improve processor power and performance. 5th Int. Symp. on High Performance Computer Architecture (1999)
2. Burger D., Austin T.M.: The SimpleScalar tool set, version 2.0. ACM SIGARCH Computer Architecture News, 25(3) (1997)
3. Burtscher M., Zorn B.G.: Hybridizing and coalescing load value predictors. Int. Conf. on Computer Design (2000)

4. Calder B., Reinman G., Tullsen D.M.: Selective value prediction. 26th Int. Symp. on Computer Architecture (1999)
5. Del Pino S., Pinuel L., Moreno R.A., Tirado F.: Value prediction as a cost-effective solution to improve embedded processor performance. 3rd Int. Meeting on Vector and Parallel Processing (2000)
6. Fagin B.: Partial resolution in branch target buffers. IEEE Trans. Comput., 46(10) (1997)
7. Fu C-y., Jennings M.D., Larin S.Y., Conte T.M.: Software-only value speculation scheduling. Tech. Rep., Dept of Electr. and Comp. Eng., North Carolina State University (1998)
8. Gabbay F.: Speculative execution based on value prediction. Tech. Rep., Dept of Electr. Eng., Technion (1996)
9. Lee C-C., Chen I-C.K., Mudge T.N.: The bi-mode branch predictor. 30th Int. Symp. on Microarchitecture (1997)
10. Levy M. NEC processor goes out of order. Microprocessor Report, 15(9) (2001)
11. Lipasti M.H., Wilkerson C.B., Shen J.P.: Value locality and load value prediction. Int. Conf. on Architectural Support for Programming Languages and Operation Systems VII (1996)
12. Morancho E., Llaberia J.M., Olive A.: Split last-address predictor. Int. Conf. on Parallel Architectures and Compilation Techniques (1998)
13. Rychlik B., Faistl J.W., Krug B.P., Kurland A.Y., Sung J.J., Velev M.N., Shen J.P.: Efficient and accurate value prediction using dynamic classification. Tech. Rep., Dept of Electr. Comp. Eng., Carnegie Mellon University (1998)
14. Sato T., Arita I.: Table size reduction for data value predictors by exploiting narrow width values. 14th Int. Conf. on Supercomputing (2000)
15. Sato T., Arita I.: Partial resolution in data value predictors. 29th Int. Conf. on Parallel Processing (2000)
16. Sato T., Arita I.: Reducing hardware budget of data value predictors by exploiting frequent value locality. IEICE Tech. Rep. CPSY2000-62 (2000)
17. Sato T.: Evaluating the impact of reissued instructions on data speculative processor performance. Microprocessors and Microsystems, 25(9) (2002)
18. Sazeides Y., Smith J.E.: Implementations of context based value predictors. Tech. Rep., Dept of Electr. Comp. Eng., University of Wisconsin-Madison (1997)
19. Sohi G.S.: Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers. IEEE Trans. Comput., 39(3) (1990)
20. Tullsen D.M., Seng J.S.: Storageless value prediction using prior register values. 26th Int. Symp. on Computer Architecture (1999)
21. Wang K., Franklin M.: Highly accurate data value prediction using hybrid predictors. 30th Int. Symp. on Microarchitecture (1997)
22. Zhang Y., Yang J., Gupta R.: Frequent value locality and value-centric data cache design. Int. Conf. on Architectural Support for Programming Languages and Operation Systems IX (2000)