International Journal of High Speed Computing

A SIMULATION STUDY OF COMBINING LOAD VALUE AND ADDRESS PREDICTORS

TOSHINORI SATO*
Department of Artificial Intelligence
Kyushu Institute of Technology
680-4, Kawazu, Iizuka-Shi, Fukuoka 820-8502, Japan
e-mail: tsato@mickey.ai.kyutech.ac.jp

ABSTRACT

In this paper, we evaluate a variety of combinations of a load value predictor and a load address predictor. We consider a dynamic hybrid predictor using a predictor selection counter, a static hybrid predictor utilizing execution profiles, and a cooperative predictor. The cooperative predictor is a load value predictor supported by a load address predictor when it is unable to predict a load value. The static hybrid and the cooperative predictors have a benefit that the hardware cost of the selection counter is removed. On the other hand, the dynamic hybrid and the cooperative predictors are free from tedious process of profiling. Based on cycle-by-cycle simulations, we have evaluated the variations and found that the cooperative predictor exploits instruction level parallelism most effectively.

Keywords: instruction level parallelism architecture, dynamic speculation of data dependence, hybrid predictors, execution profiles, optimizing compilers

1. Introduction. Recently, the practice of speculation in resolving data dependences has been studied as a means of extracting more instruction level parallelism (ILP). An outcome of an instruction is predicted by value predictors. The instruction and its dependent instructions can be executed simultaneously, thereby exploiting ILP aggressively. Value predictors are classified into two types according to predicted instruction class. One type predicts execution results of all register-writing instructions[5,7,18,22]. The other is that predicts only load data values[6,8,10,15,21,23]. In this paper, we focus on the load value predictor because the propagation of predictability often rises in load instructions[2].

* Parts of this work were performed while the author was with Toshiba Microelectronics Engineering Laboratory.

In order to utilize value predictors effectively, the following two conditions must be considered. One is prediction accuracy, and the other is prediction coverage. We define the prediction accuracy as the number of instructions whose outcome is correctly predicted over the total number of the predicted instructions, and the prediction coverage as the number of correctly predicted instructions over the all load instructions. Since misspeculations cause penalties, prediction accuracy should be high. Predictability is different according to individual instructions. Thus, the prediction accuracy is improved if only easily predictable instructions are predicted. However, this implies a reduction in prediction coverage. It has been found that the dominant factor in performance improvement is not prediction accuracy but prediction coverage[12,20]. Therefore, we must improve both prediction accuracy and prediction coverage.

For the purpose of increasing the prediction coverage as well as the prediction accuracy, we propose a combination of a load value predictor and a load address predictor. Address prediction enjoys better accuracy than value prediction, but does not result in as big a win when correct. Therefore, address prediction is proposed to be used in cases where value prediction is unable to predict a load. We call this predictor a cooperative predictor[13]. In this paper we compare the cooperative predictor with a variety of combinations consisting of a load value predictor and a load address predictor.

The organization of the rest of this paper is as follows. In Section 2, related works are surveyed. The dynamic and static hybrid predicting methods used are explained in Section 3. The cooperative predictor is also explained. In Section 4, our evaluation methodology is described, and simulation results are shown in Section 5. Finally, our conclusions are presented in Section 6.

2. Related Work. There are many studies of value predictors. These predictors are classified into two types according to the predicted instruction class. One predicts execution results of all types of instructions which write values into registers[5,7,18,22]. The other predicts only load values[6,8,10,15,21,23]. The last value predictor proposed by Lipasti et al.[7,8] introduces the value prediction concept, and is based on value locality. According to this method, an instruction uses the same value which was generated the last time that instruction was executed. An operand prefetch cache[23] also uses the last outcome of a load instruction. The stride predictor proposed by Gabbay et al.[5] keeps not only the last outcome of an instruction but also a stride which is the difference between last two outcomes of the instruction. The predicted value is the sum of the last outcome and the stride.

In order to improve prediction accuracy, several hybrid predictors[18,22] have been proposed. A hybrid predictor is a combination of several value predictors with a selector choosing the best predictor on a case by case basis.

Another approach to improving value prediction accuracy is to use program execution profiles[2,5]. Using these profiles, instructions are classified according to their predictability, and a compiler provides this information to the processors. The prediction accuracy of load values can be improved by utilizing information gathered from store instructions. The schemes proposed by Moshovos et al.[10], Sato[15], and Tyson et al.[21] are very similar to each other, and thus we refer to them as renaming-based value predictors in this paper. The renaming-based predictors speculatively streamline a stored value to a load instruction. The load value predictor used in this paper is based on one of the renaming-based predictors[15].

There are many proposals on predicting the data addresses of load instructions [6,14,17,23]. The basic schemes used in them are similar. They keep histories of memory references and predict a data address as the sum of a previous data address and a stride which is the difference between the last two addresses generated by an instruction. The data address prediction method utilized in this paper is based on [14].

There are few studies investigating predictors that combine a load value predictor and a load address predictor. The predictor proposed by Gonzalez et al.[6] predicts both load values and addresses, but the load values are obtained using the predicted load addresses. Thus, the load address predictor and the load value predictor do not work independently. On the other hand, our proposed load value predictor does not use the predicted address and hence works independently of the load address predictor. Recently, we have proposed a predictor named cooperative predictor[13]. The cooperative predictor is a load value predictor supported by a load address predictor when it can not predict a load value. Furthermore, we have proposed to check speculatively whether a predicted load value is correct by using the load address predictor. We call this technique speculative verification. This work is similar to recent work done independently by Reinman et al.[11]. Their Load-Spec-Chooser and Check-Load-Chooser predictors resemble the cooperative predictor and the one using the speculative verification respectively. In [16] we evaluated a static hybrid predictor consisting of a load value predictor and a load address predictor. On the other hand, Reinman et al. have not considered any static hybrid predictor. In this paper, we will evaluate a variety of the combinations of the predictors and compare them with each other.

3. Combinations of Value and Address Predictors. A variety of combinations of a load value predictor and a load address predictor are considered in this section. First, we explain a load value predictor and a load address predictor which are components of the combinations. We then discuss the variations.

3.1. Load value predictor. In this subsection, we explain our renaming-based load value predictor[15]. First, we explain the concept of two-hop reference address renaming. Next, we describes three tables which are utilized to implement the proposed load value prediction scheme. They are store-indexed value table (SIVT), load-indexed store table (LIST), and data-indexed store table (DIST). The SIVT and LIST cooperate with each other and forward a stored data value to a load instruction. The DIST links a pair of a load and a store instructions. And next, the load value prediction mechanism is explained.

The load value prediction scheme studied by Lipasti et al.[8] uses only load instruction information. The value prediction accuracy will be improved if the store instruction information is available, since the data value read by a load instruction is stored by a store instruction. If the store instruction renames the data address into a tag and the load instruction refers the data using the same tag, it becomes possible to forward the data from the store instruction to the load instruction before the load instruction calculates the data address. However, it is difficult to deliver the tag defined by the store instruction into the load instruction. Therefore, we propose that the store and load instructions define their tags independently. If the tag defined by the load instruction is translated into the one defined by the store instruction, the store and load instruction use a same tag effectively. This scheme is shown in Fig.1. The `tag(l)` defined by the load instruction is translated into the `tag(s)`, which is defined by the store instruction. The load instruction refers the temporal memory with the translated tag. We call this scheme 2-hop reference address renaming.

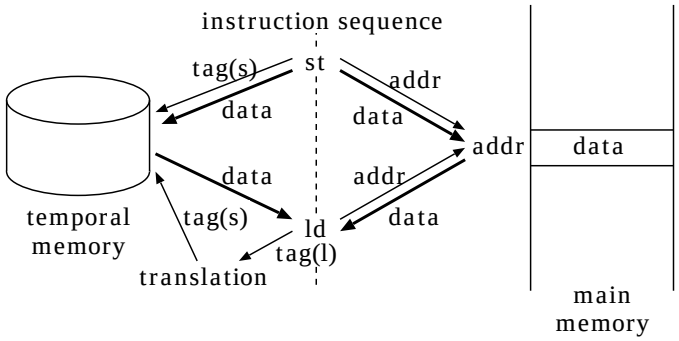
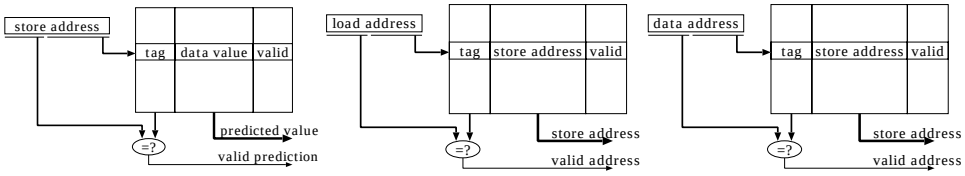


FIG. 1. *2-hop reference address renaming*

If an instruction address are used as the tag, a data address is renamed as follows. When a store instruction is executed, the data value is stored also in a temporal memory with the store instruction address used as a tag. A load instruction translates the load instruction address into the store

instruction address and refers the temporal memory with the translated address. And thus, the load instruction can obtain the data value from the temporal memory before the data address is calculated.

The load value predictor based on the 2-hop reference address renaming consists of the following three tables, which are the SIVT, LIST, and DIST. The SIVT is indexed by a store instruction address, and provides the data value written by the store instruction. Each entry of the SIVT holds a data value and is indexed by a store instruction address. When the SIVT is accessed with a store instruction address, it supplies the data value which the store instruction wrote at the last time. Thus, the SIVT renames a data address to a store instruction address. The LIST translates a load instruction address into the address of the store instruction which refers the same memory location accessed by the load instruction. Each entry of the LIST keeps a store instruction address and is indexed by the load instruction address. When a load instruction refers the LIST, it obtains the store instruction address. This store instruction wrote the data which the load instruction will read. Hence, the LIST renames a store instruction address to a load instruction address. The DIST links a load instruction with a store instruction whose reference address is same with that which the load instruction refers. It is indexed by a data address and supplies the store instruction address. When a load instruction refers the DIST with the data address, it obtains the address of the store instruction which wrote the data at the last time.



(i) SIVT (ii) LIST (iii) DIST

FIG. 2. *Renaming-based load value predictor*

The SIVT, LIST, and DIST have a similar structure with cache memory. Fig.2(i) shows the direct-mapped SIVT. Each entry consists of a tag address field (`tag`), a data value field (`data_value`), and a valid bit (`valid`). Fig. 2(ii) shows the direct-mapped LIST. Each entry consists of a tag address field (`tag`), a store instruction address field (`store_address`), and a valid bit (`valid`). Fig.2(iii) shows the direct-mapped DIST. Each entry consists of a tag address field (`tag`), a store instruction address field (`store_address`), and a valid bit (`valid`).

Next, we explain the load value prediction mechanism using the simple pipeline described in Fig.3. A data value is predicted before an instruction

is issued. When the instruction is fetched, the LIST is accessed with the instruction address and provides a store instruction address. The store instruction address is used to refer the SIVT at the decode stage. The SIVT provides a data value stored by the store instruction. When the instruction accessing the LIST is a load instruction, the data value is used as the predicted one and instructions following the load instruction are executed speculatively using the predicted value. Note that the LIST and SIVT are accessed at different stages, and hence there are not any impact of serialized lookups of the LIST and SIVT on processor cycle time.

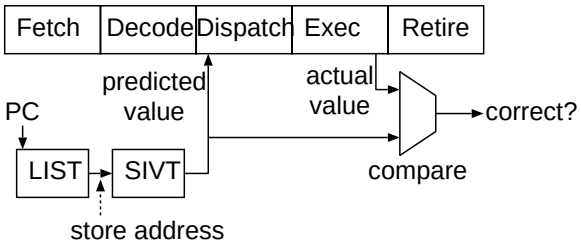


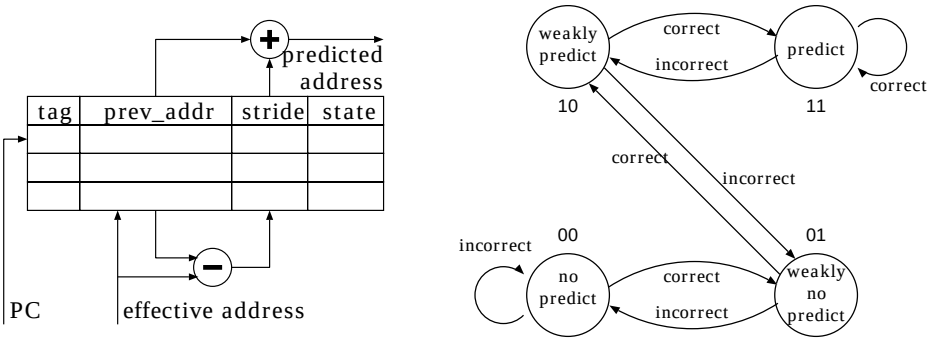
FIG. 3. *Pipeline diagram*

The SIVT, DIST, and LIST are updated as follows. The SIVT is updated when a store instruction obtains the data value which will be stored in memory even if the data address has not been calculated. At this time, the instruction knows the store instruction address and the data value. Therefore, it is possible to keep the value in the SIVT indexed by the store instruction address. The DIST is updated when a store instruction has calculated the data address. At this time, the instruction knows the store instruction address and the data address. Hence, the DIST indexed by the data address can hold the store instruction address. The LIST is updated, whenever the load instruction finishes. The DIST is referred and it provides a store instruction address. At this time, the data address is calculated and it is possible to access the DIST. Using the store instruction address provided by the DIST, it is possible to link the load and store instructions in the LIST.

From the above explanations, the load data value prediction is performed. The DIST links a load and a store instruction which refers the same memory location, and the LIST keeps the link. When the load instruction is executed, the LIST supplies the store instruction address which is used to access the SIVT. The load instruction can obtain the data value provided by the SIVT before it calculates the data address. In summary, in order to predict the load value without calculating the data address, the 2-hop reference address renaming from the data address into the load instruction address is performed.

3.2. Load address predictor. In this subsection, we describe a load address prediction mechanism[14]. In order to predict data addresses, we utilize the reference prediction table (RPT)[3].

The RPT, which has a similar structure with the instruction cache, is proposed by Chen et al. for hardware prefetching[3]. We apply the RPT to predict the data address because of its simplicity. Fig.4(i) shows the RPT structure. The RPT keeps track of previous memory references. An entry of the RPT is indexed by the instruction address and holds the previous data address (`prev_addr`), the stride value (`stride`), and the state information (`state`). The stride is the difference between last two data addresses generated by an instruction. The state information encodes the past history and indicates whether next prefetching is initiated. An example of the state information is decided according to Fig.4(ii). Note that the state transition described in Fig.4(ii) is different from the original one proposed in [3]. It is quite similar to the two bit saturated counter (2bC) used in branch predictors. The reason why we choose the 2bC scheme is that the RPT using the 2bC state machine can more aggressively speculate the data dependences than that using the original state machine[14]. There are four states, which are "predict", "weakly predict", "no-predict", and "weakly no-predict".



(i) Reference prediction table

(ii) State transition

FIG. 4. *Reference address prediction*

The predicted address is generated as follows. The RPT is accessed at instruction fetch stage. The previous data address and the stride value are supplied from an entry of the RPT indexed by the program counter (PC), if the tag field is matched. The predicted address is the sum of `prev_addr` and `stride`. The state information is also provided. If the state is "predict" or "weakly predict", the predicted address is valid. Otherwise, the prediction is not initiated. Next, we explain the state transition of the RPT. If a prediction is correct, the counter is incremented. Otherwise, it is decremented. When the most significant bit is `1`, the state is "(weakly) predict" and thus

the predicted address is valid.

3.3. Dynamic hybrid predictor. A dynamic hybrid predictor consists of a load value predictor, a load address predictor, and a predictor selection counter. Whenever a load instruction is encountered both the value and the address predictors initiate prediction. The selection counter decides which predictor is used. When the counter value is larger than a threshold, the value predictor is selected. Otherwise, the address predictor is used. The state transition of the counter is shown in Table 1. The counter is incremented if a value prediction is correct. The counter is decremented if only address prediction is correct. Note that priority is given to the value predictor because the address prediction enjoys better accuracy than value prediction but does not result in as big a win when correct.

TABLE 1
State transition of selection counter

value	address	counter
correct	correct	+1
correct	fail	+1
correct	no predict	+1
fail	correct	−1
no predict	correct	−1
other cases		no change

3.4. Static hybrid predictor. A static hybrid predictor consists of a load value predictor and a load address predictor. It does not have a predictor selection counter since the selection is statically decided at compile time using execution profiles. When a load instruction is encountered, only the predictor which is statically chosen initiates prediction.

For each load instruction the profile includes information consisting of the number of times the load is executed, the number of times the value predictor correctly predicts the load, the number of times the value is mispredicted, the number of times the address predictor correctly predicts the load, and the number of times the address is mispredicted. Using the information, load instructions are classified into three categories, `value_prediction`, `address_prediction`, and `no_prediction`. The classification can be based on the prediction accuracy. For example, when it is decided that predictor selecting threshold is 50%, the predictor whose prediction accuracy is larger than the threshold (50%) is selected for each instruction. If both predictors have the accuracy which is larger than the threshold (50%), the predictor whose accuracy is higher is selected.

Compilers use the profile to determine classification information, which

is supplied to the processors. It is necessary to modify instruction set architecture (ISA) in order to inform the processors of this class determination. Each load instruction is enhanced with 2-bit prediction field. Table 2 shows an encoding of the prediction field; this enhancement is simple to implement.

TABLE 2
Encoding of prediction field

bit 1	bit 0	classification
0	-	`no_prediction`
1	0	`value_prediction`
1	1	`address_prediction`

The static hybrid predictor requires a smaller hardware cost than the dynamic one because it does not have a selection counter. This predictor also reduces interference and aliasing in the prediction tables since the component predictors are statically selected and only the predictor that is actually used should be updated. These will increase both prediction accuracy and prediction coverage.

3.5. Cooperative predictor. A cooperative predictor is a load value predictor supported by a load address predictor[13]. Whenever a load instruction is encountered both the value and the address predictors initiate prediction. The predicted address is used only when the value predictor is unable to predict a load value, and thus the cooperative predictor does not have any selection counter. Therefore, its hardware cost is as same as that of the static hybrid predictor.

Examples are useful to understand how the cooperative predictor works. Fig.5 shows an example of an instruction sequence and there is a data dependence between instructions `I1` and `I2`. It is assumed that register $r12$ is ready and $r10$ will be ready at one cycle later. It is executed as shown in Fig.6 when the dependence is not speculated. Figs. between 7 and 9 explain how the instructions speculate the dependence. When the load value for `I1` is correctly predicted, the instruction sequence goes as shown in Fig.7. `I2` is dispatched using the predicted load value (**r11**). The predicted value is in bold. When the actual load value is obtained, it is compared with the predicted one and the speculation finishes since the prediction is correct. The dependence length is decreased by three.

```
I1:  load  r11 ← r10(4)
I2:  add   r13 ← r11 + r12
```

FIG. 5. *Instruction sequence example*

	time →			
I1		ad←r10+4	r11←mem[ad]	
I2				r13←r11+r12

FIG. 6. *Without any predictions*

	time →			
I1		ad←r10+4	r11←mem[ad]	*compare* r11
I2	r13←**r11**+r12			

FIG. 7. *Correct value prediction*

	time →			
I1	**r11**←mem[**ad**]	ad←r10+4 *compare* ad		
I2		r13←**r11**+r12		

FIG. 8. *Correct address prediction*

	time →			
I1	**r11**←mem[**ad**]	ad←r10+4 *compare* ad	r11←mem[ad]	
I2		r13←**r11**+r12 *invalidated*		r13←r11+r12

FIG. 9. *Failed address prediction*

When the load value can not be predicted the instruction sequence is as same as that shown in Fig. 6. If both load value and address predictors are used, that is the cooperative predictor is used, the sequence goes as shown in Figs.8 and 9 according to the address prediction result. The load value is obtained using the predicted address (**ad**). The predicted address and the value obtained using the predicted address are in bold. The comparison of the predicted address with the actual one can be done in parallel with the address calculation[4], and thus the data fetching using the actual address is suppressed when two addresses match. `I2` speculates data dependence using the load value which is read using the predicted address, and when the prediction is correct the speculation finishes as shown in Fig. 8. Otherwise, `I1` fetches data again using the actual address and `I2` is invalidated and reissued as shown in Fig.9.

We expect that the cooperative predictor works efficiently even if there is not any predictor selection counter. The reason is explained in Fig.10. In

Fig. 10(i), a store and a load instructions which refer the same memory location appear in turn in a single loop. On the other hand, in Fig.10(ii), a store and a load instructions which refer the same memory location are included different loops, respectively. The renaming-based value predictor[10,15,21] is good at the former case but poor at the latter case. On the contrary, the stride-based address predictor[6,14,17,23] is good at the latter case. Therefore, the two predictors can works complementary. That is, the predictors covers different classes of load instructions and hence the cooperative predictor does not need the selection counter.

```
loop:    :              loop1:   :
       store                     store
         :                         :
       load                      bne loop1
         :             loop2:      :
       bne loop                  load
                                   :
                                 bne loop2

       (i)                       (ii)
```

FIG. 10. *Two types of loops*

In this paper, we do not consider the speculative verification because it has little contribution to processor performance over the cooperative predictor[13].

3.6. Comparison between predictors. Table 3 summarize characteristic of each predictor. The second column indicates if the predictor specified by the first column has the selection counter. The next column presents if the predictor needs process of profiling. And the last column shows how adaptable the predictor to dynamic behavior of a program.

TABLE 3
Predictor characteristics

	counter	profile	adaptability
cooperative	NO	NO	good
dynamic	YES	NO	very good
static	NO	YES	poor

From the view of hardware cost, the cooperative predictor and the static hybrid predictor is more effective than the dynamic hybrid one. On the other hand, the static hybrid predictor relies on the tedious process of the

profiling. The other two predictors are free from the profiling. Furthermore, its adaptability to dynamic execution of a program is poorer than the other two. From these considerations, the cooperative predictor is the best choice among these predictors if contributions to processor performance are same between these predictors.

4. Evaluation Methodology. In this section, we describe the evaluation methodology by explaining a processor model and benchmark programs.

4.1. Experimental model. An execution-driven simulator that models wrong path execution caused by misspeculations is used for this study. We implemented this simulator using the SimpleScalar tool set[1]. The SimpleScalar/PISA ISA is based on the MIPS ISA.

The simulator models a realistic 8-way out-of-order execution superscalar processor based on a register update unit (RUU)[19] which has 64 entries. Each functional unit can execute any operation. The latency for execution is 1 cycle except in the case of multiplication (4 cycles) and division (12 cycles). A 4-port, non-blocking, 64KB, 32B block, 4-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 a 1MB, 64B block, 8-way set associative L2 cache which has a miss latency of 18 cycles for the first word plus 2 cycles for each additional word. No memory operation can execute that follows a store whose data address is unknown. A 64KB, 32B block, 4-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 branch target buffer, a 4K-entry gshare-type branch predictor[9], and an 8-entry return address stack are used. The branch predictor is updated at the instruction commit stage.

For value prediction, we assume that the DIST, SIVT, and LIST have 4096-entry direct-mapped tables respectively. The RPT utilized for address prediction is also assumed to be a 4096-entry direct-mapped table. In the case of the dynamic hybrid predictor, a 4096-entry direct-mapped predictor selection counter which consists of 2-bit up-down saturated counters is used, and the threshold for selection is 2.

In order to recover the processor state when a misspeculation occurs, an instruction reissue mechanism is used[12]. The instruction reissue mechanism re-executes only instructions dependent upon a misspeculated instruction. Those instructions are detected selectively and reissued inside the extended RUU. It is found that the reissue mechanism can be practically implemented without a large hardware cost.

4.2. Workloads. The SPEC95 CINT benchmark suite is used for this study. The `test` input files provided by SPEC are used. Table 4 lists the benchmarks and the input sets. We used the object files provided by University of Wisconsin Madison[1], except `132.ijpeg` which is compiled by GNU GCC (version 2.6.3) with the optimization option, `-O3`. Each program is executed to completion or for the first 100 million instructions. We count only committed instructions.

TABLE 4
Benchmark program and input file

program	input file
099.go	null.in
124.m88ksim	ctl.in
126.gcc	cccp.i
129.compress	test.in
130.li	test.lsp
132.ijpeg	specmun.ppm
134.perl	primes.in
147.vortex	vortex.in

The static classification strategy is as follows. A load instruction is predicted by one of the predictors if its prediction accuracy is larger than a threshold. When both predictors satisfy the condition, the predictor whose prediction accuracy is larger than that of the other is selected. If the accuracy of the value predictor and that of the address predictor is same, the former is chosen. We evaluate six thresholds, which are 0%, 50%, 60%, 70%, 80%, and 90%.

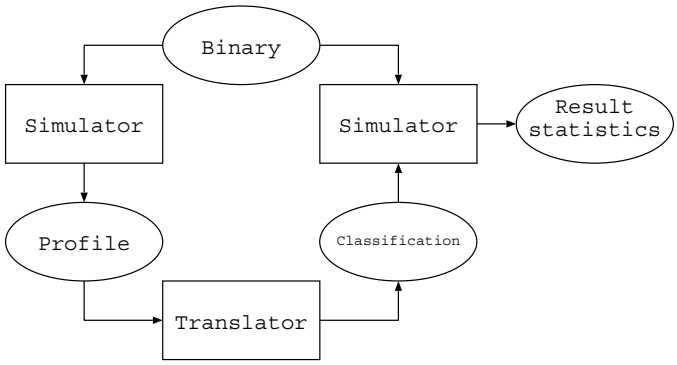


FIG. 11. *Classification methodology*

Compilation using execution profiles is modeled as follows. Our static

classification methodology utilizing execution profiles is explained in Fig.11. First, a binary code is executed on the simulator and its profile is gathered. The execution profile is processed by a translator and the classification information is obtained. The binary code is executed again on the simulator. In the second simulation, the class information is supplied to the simulator in order to utilize the execution profile. We use the same input files for the profiling and the re-execution since it has been found that program execution characteristics are quite similar for different input files[2,5].

5. Simulation Results. This section presents simulation results. First, we show the prediction coverage of each predictor for various benchmarks. Next, we divide these results according to the effects of the predictor's component predictors. Finally, the impact of each predictor on processor performance is investigated.

5.1. Prediction coverage. Fig.12 presents the prediction coverage when the load value predictor is utilized. Each bar is divided into three parts. The bottom part (black) indicates the percentage of the load instruction whose data value is correctly predicted. The middle part (white) indicates the percentage that is mispredicted. And the top part (gray) indicates the percentage that is not predicted. The prediction coverage depicted as the bottom part is low and is 46.6% on average. Therefore, it is expected to improve the prediction coverage without diminishing the prediction accuracy.

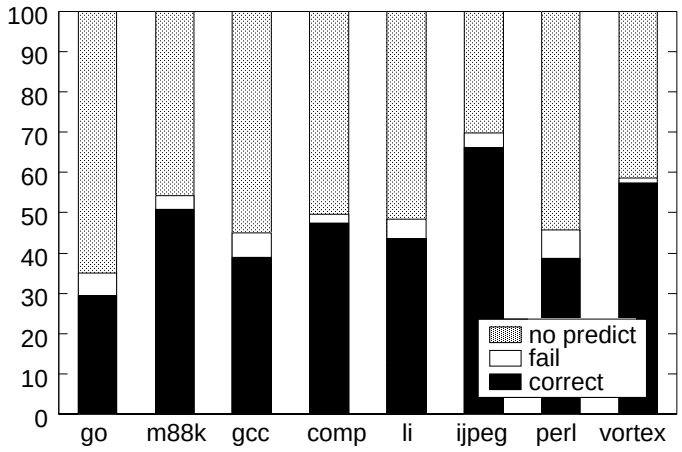


FIG. 12. *(%)Dynamic prediction coverage of value predictor*

Fig.13 shows the prediction coverage when the load address predictor is utilized. As same with Fig.12, each bar is divided into three parts. The

coverage is 51.7% on average and is slightly larger than that of the value predictor. As can be seen, the prediction coverage is significantly high for `124.m88ksim`, `129.compress`, and `132.ijpeg`, and thus the value predictor seems to be efficiently supported by the address predictor. For the remaining programs, the prediction coverage is small. However, if the address predictor covered the load instructions which can not be covered by the value predictor, total prediction coverage would be increased.

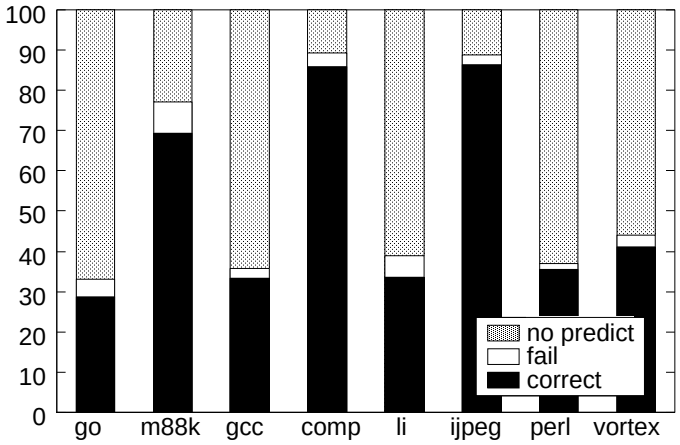


FIG. 13. *(%)Dynamic prediction coverage of address predictor*

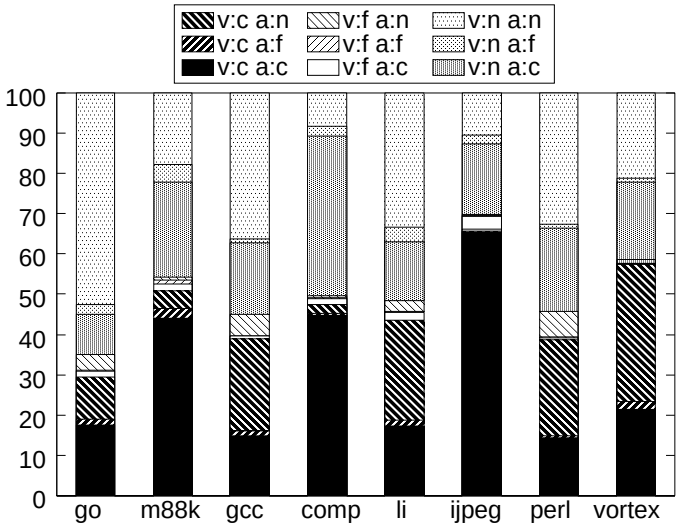


FIG. 14. *(%)Dynamic prediction coverage of value and address predictors*

Fig.14 shows dynamic prediction coverage when both load value and

address predictors are used. Each bar is divided into nine parts. They are combinations of the three cases (`correct`, `fail`, `no predict`) of the value prediction and of address prediction. In Fig.14, the value and address predictions are denoted with `v:` and `a:` respectively. In addition, the `correct`, `fail`, and `no predict` cases are denoted with `c`, `f`, and `n`, respectively.

Using the information in Fig.14, Fig.15 depicts the prediction coverage when an ideal selector is used. That is, a correct predictor is always selected if at least one predictor predicts the load correctly. The coverage becomes 68.4% on average with a maximum of 88.7%. This is an upper bound and may not be realized by practical selection mechanisms. Our goal is to attain the near optimal coverage using the combinations of two predictors.

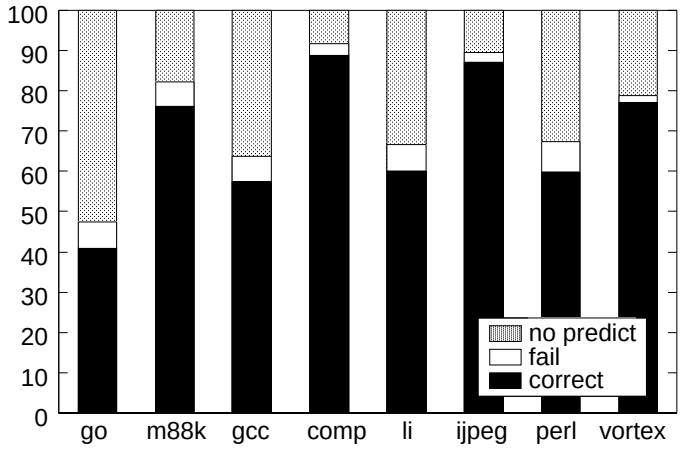


FIG. 15. *(%)Dynamic prediction coverage of ideal predictor*

Fig.16 shows the prediction coverage of the static hybrid predictor. Each program is statically analyzed and the percentage of instructions that are candidates for prediction is shown. For each group of six bars, the bars from left to right indicate the static prediction coverage when the predictor selection threshold is varied between 0% and 90%. Each bar is divided into two parts. The lower part indicates the percentage of load instructions selected for `value_prediction` and the upper part indicates the percentage selected for `address_prediction`. Naturally, prediction coverage decreases as the prediction threshold increases. From Fig.16, when the threshold is 0% the percentage of load instructions which are selected as `value_prediction`, `address_prediction`, and `no_prediction` are 41.6%, 19.7%, and 38.7% on average respectively. When the threshold increases to 90%, they are 36.1%, 14.3%, and 49.6%, respectively.

Fig.17 depicts the dynamic prediction coverage of the predictors. For each group of seven bars, the first bar (see from left to right) indicates the

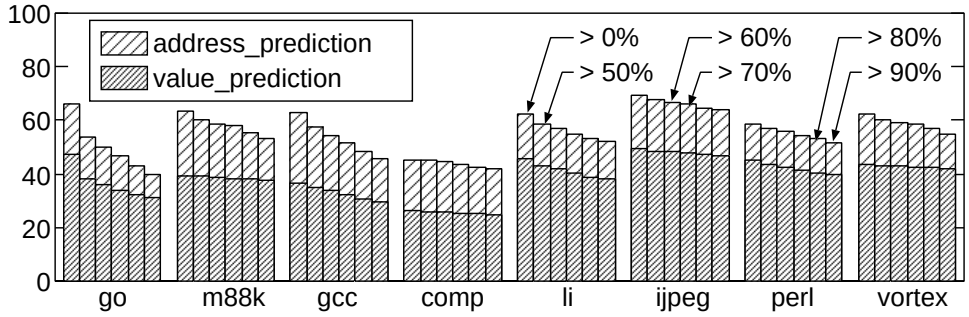


FIG. 16. *(%)Prediction coverage of static hybrid predictor*

coverage of the dynamic hybrid predictor. Next six bars indicate those of the static hybrid predictor when the selection threshold is varied between 0% and 90%. The remaining bar indicates that of the cooperative predictor. Each bar is divided into five parts. The bottom part indicates the percentage of load instructions whose data value is correctly predicted. The next part indicates the percentage of load instructions whose data address is correctly predicted. The next part indicates the percentage of load instructions whose data value is mispredicted. The next part indicates the percentage of load instructions whose data address is mispredicted, and the top part indicates the percentage of load instructions for which neither data value nor address are predicted. For the dynamic hybrid predictor, only the selected component predictor is considered for calculating the prediction coverage.

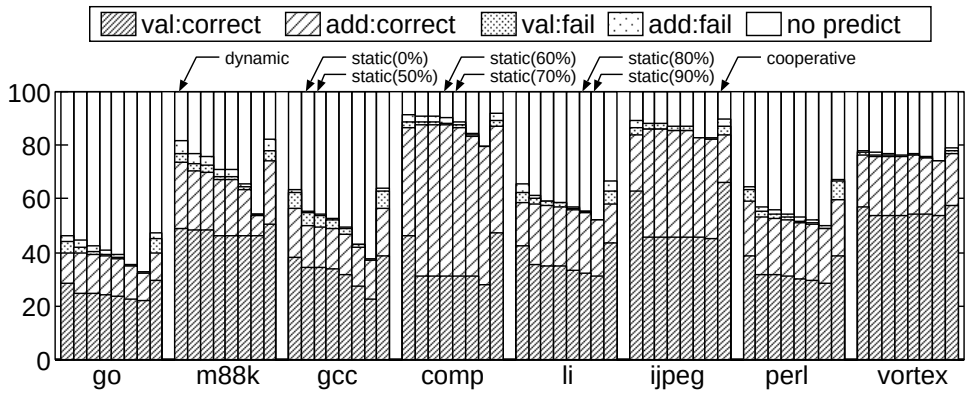


FIG. 17. *(%)Dynamic prediction coverage of each predictor*

First, it is observed that the coverage of the dynamic hybrid predictor is comparable to that of the ideal predictor shown in Fig.15. For example,

in the case of `099.go` the coverage of the dynamic hybrid predictor is 39.5% and that of the ideal one is 40.8%. We are satisfied with these results, and thus the goal of this paper is to achieve comparable coverage using the static hybrid predictor or the cooperative predictor because they have the benefit that the hardware cost of the selecting mechanism is removed.

Second, in the case of the static hybrid predictor, the dynamic coverage of the component value predictor is smaller than the static coverage for most cases. On the other hand, the dynamic coverage of the component address predictor is larger than the static coverage for most cases. As a result, `129.compress`, `132.ijpeg`, and `147.vortex` achieve high prediction coverage. It is also seen that total prediction coverage is approximately same between the dynamic and the static hybrid predictors when the prediction threshold of the static hybrid predictor is relatively low. We define total prediction coverage as the sum of value prediction coverage and address prediction coverage. From this observation, it is expected that performance impact of the static hybrid predictor will be comparable to that of the dynamic one.

Third, it can be seen that the dynamic hybrid predictor tends to select the value predictor more often than the static one does. Thus, the static hybrid predictor will not attain processor performance improvement comparable to that of the dynamic one if address prediction does not result in as big a win when correct as the value predictor.

Forth, for the cooperative predictor, the total prediction coverage is 67.0% on average with a maximum of 87.3%. The difference from the optimal coverage is only 1.4%. This is very encouraging. From Fig.17 it is observed that the value predictor is efficiently supported by the address predictor in the cases of `124.m88ksim`, `129.compress`, and `132.ijpeg`, as we expected. The remaining programs are also gained by the address predictor. This confirms that the address predictor covers the load instructions which can not be covered by the value predictor.

Fifth, the total prediction coverage of the cooperative predictor is comparable to that of the dynamic hybrid one. It is also found that percentages selecting the value and the address predictors are approximately the same between the cooperative and the dynamic hybrid predictors. From these observations, the contribution to processor performance of the cooperative predictor can be as much as that of the dynamic hybrid predictor.

Lastly, the percentage of mispredictions should be considered. The percentage of the cooperative predictor is slightly larger than that of the dynamic hybrid predictor. Therefore, if the misprediction penalty is severe, the cooperative predictor will not achieve performance improvement comparable to that of the dynamic hybrid predictor.

5.2. Prediction accuracy. Tables 5 and 6 present the prediction accuracy of the component value and address predictors respectively. As can be easily seen, the accuracies of the cooperative predictor is same to those of the dynamic hybrid one. This is because the cooperative predictor as well as the dynamic hybrid one always utilizes and updates both component predictors. In the case of the value predictor, the prediction accuracy is always improved when the candidates for prediction are statically selected using the execution profiles. This is natural because only load instructions whose prediction accuracy are high are selected. This is also due to the reduction of interference and aliasing in the prediction tables described in Section 3.4. For the address predictor, the prediction accuracy of the static hybrid predictor is lower than that of the dynamic one in the cases of `099.go`, `124.m88ksim`, `132.ijpeg`, and `134.perl`. This is because the priority in selection is given to the value predictor. The address predictor only predicts the loads which are not well predicted by the value predictor. This causes the diminishing of the address prediction accuracy. On the other hand, the address predictor of the dynamic hybrid predictor predicts all load instructions. Therefore, the address prediction accuracy of the static hybrid predictor is lower than that of the dynamic one. Referring back to Fig.17, the reduction of the address prediction accuracy is undesirable for `132.ijpeg` and `134.perl` since the address prediction coverage is considerably high for these programs.

TABLE 5
(%)Prediction accuracy of value predictor

program	dynamic hybrid	static hybrid						cooperative
		> 0%	> 50%	> 60%	> 70%	> 80%	> 90%	
go	84.46	92.37	95.77	96.81	97.95	99.18	99.66	84.46
m88ksim	93.55	94.41	94.51	98.17	98.17	98.17	98.19	93.55
gcc	86.52	88.33	89.44	90.47	92.73	96.42	99.32	86.52
compress	95.87	98.15	98.20	98.20	98.20	98.20	99.99	95.87
li	90.08	93.57	95.40	96.18	98.07	98.92	99.60	90.08
ijpeg	94.88	99.71	99.75	99.79	99.83	99.86	99.94	94.88
perl	84.78	93.77	95.44	96.12	97.44	97.96	98.54	84.78
vortex	97.92	99.36	99.42	99.53	99.66	99.79	99.86	97.92

TABLE 6
(%)Prediction accuracy of address predictor

program	dynamic hybrid	static hybrid						cooperative
		> 0%	> 50%	> 60%	> 70%	> 80%	> 90%	
go	86.78	85.51	88.65	90.97	92.41	95.67	98.10	86.78
m88ksim	90.00	85.40	87.07	88.34	88.64	91.60	99.29	90.00
gcc	92.77	94.36	95.83	96.93	97.48	98.11	98.38	92.77
compress	96.19	96.19	96.19	96.45	97.33	99.65	99.73	96.19
li	86.19	97.43	98.93	98.95	98.96	99.13	99.95	86.19
ijpeg	97.02	95.07	95.39	96.23	96.41	98.94	99.04	97.02
perl	96.27	93.28	93.29	94.17	94.20	94.26	94.54	96.27
vortex	93.00	95.85	96.54	97.24	97.46	98.48	99.16	93.00

5.3. Impact on processor performance. Fig.18 shows the processor performance improvement rate. For measuring performance, we use commit-

ted instructions per cycle (IPC). Only useful instructions are considered for counting the IPC. We define the performance improvement rate as the increased IPC over the IPC of the baseline model. For each group of seven bars, the first bar (see from left to right) indicates the improvement rate for the value predictor and the second one indicates that for the address predictor. The next bar indicates that for the dynamic hybrid predictor. The next six bars indicate those for the static hybrid predictor when the selection threshold is varied between 0% and 90%. The remaining bar indicates that for the cooperative predictor.

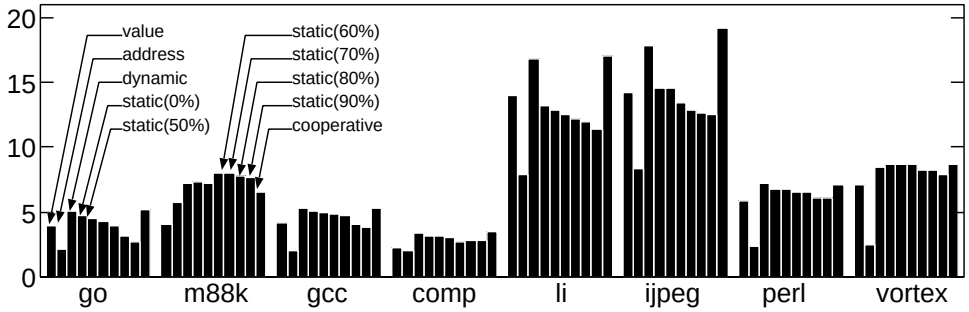


FIG. 18. *(%)Processor performance improvement*

First, it is observed that for most programs except `124.m88ksim` the component value predictor has much more contribution to processor performance than the component address predictor. Referring back to Figs.12 and 13, for five of the eight programs the prediction coverage of the value predictor is larger than that of the address predictor, thus this results are natural. Even for `130.compress` and `132.ijpeg`, where the coverage of the value predictor is smaller than that of the address one, the performance improvement of the value predictor is larger than the address predictor. Only in the case of `124.m88ksim`, the address predictor has more contribution to performance than the value predictor. Thus, it is seen that correct value prediction results in bigger contributions to processor performance than correct address prediction.

Next, we compare the static hybrid predictor with the dynamic one. The notable difference between the improvements for the dynamic and the static hybrid predictors is observed in the cases of `130.li` and `132.ijpeg`. The improvements for the static hybrid predictor are much lower than that for the dynamic one. In the cases of `130.li` and `132.ijpeg`, the improvement of the static hybrid predictor is even smaller than that of the component value predictor. As can be seen from Fig.17, total prediction coverage is approximately the same for the dynamic and the static hybrid predictors, and thus it is expected that the improvement rate for the static hybrid predictor

is comparable to that for the dynamic one. The difference is determined by how often each component predictor is selected. In the dynamic hybrid predictor the value predictor is used more than 70% of the time. On the other hand, in the static hybrid predictor it is used less than 60% of the time. These observations confirm that correct value prediction results in bigger contributions to processor performance than correct address prediction.

The percentage that the value predictor is selected is summarized in Table 7. Generally, the dynamic hybrid predictor tends to choose the value predictor more often than the static one. This is due to the selection policy explained in Section 3.3. Since the policy gives priority to the value predictor when both predictors result in correct predictions, the value predictor is selected more often in the dynamic hybrid predictor. As we have considered above, this results in big success in the cases of `130.li` and `132.ijpeg`. For `129.compress`, the percent usage is considerably different between the dynamic and the static hybrid predictors while there is not significant difference in performance contribution between the predictors. In this case, the performance improvement rates of the component predictors are approximately same. Moreover, the total prediction coverage of the static hybrid predictor is larger than that of the dynamic one. Therefore, the difference of the percent usage is compensated by the contribution of the address predictor and it does not much influence on the performance improvement rate.

TABLE 7
Percentage selecting value predictor

program	dynamic	static						cooperative
		> 0%	> 50%	> 60%	> 70%	> 80%	> 90%	
go	71.93	60.23	60.13	61.17	61.51	64.44	68.08	73.65
m88ksim	63.95	66.69	67.41	66.49	66.81	71.75	86.44	66.05
gcc	69.86	70.62	70.66	70.57	69.14	65.58	61.08	70.43
compress	52.55	35.22	35.21	35.34	35.92	37.94	34.75	54.01
li	70.86	61.75	61.73	61.41	60.00	59.29	60.14	72.75
ijpeg	74.06	51.83	51.95	52.44	52.55	54.93	54.97	77.82
perl	66.61	60.03	59.25	59.43	58.37	57.88	57.76	67.92
vortex	74.27	70.08	70.28	70.56	70.96	71.77	72.73	74.22

An interesting observation is that the performance improvement rate for the static hybrid predictor is sometimes larger than that for the dynamic one. Such results are seen for `124.m88ksim` and `147.vortex`. For `124.m88ksim`, it is observed from Table 7 that the percentage selecting value predictor of the static hybrid predictor is larger than that of the dynamic one. On the other hand, we have found that the component address predictor attains much more performance improvement than the value predictor. These observations might look paradoxical. However, they are not. The large contribution of the address predictor is due to its large prediction coverage. We have already confirmed that correct value prediction generally results in bigger contributions to processor performance than correct address prediction. Therefore, larger percentage selecting value predictor of the static hybrid predictor results in that the performance improvement rate for the static

hybrid predictor is larger than that for the dynamic one. For `147.vortex`, the reason is simple. From Fig.17, the total prediction coverage as well as the coverage of the component predictors are approximately same between the dynamic and static hybrid predictors. Moreover, we can find that the prediction accuracies of the static hybrid predictor are better than those of the dynamic one from Tables 5 and 6. Thus, the performance improvement rate for the static hybrid predictor becomes larger than that for the dynamic one.

It is also important to note that the improvement rate is in one example increased (for `124.m88ksim`) when the prediction threshold is 60%. Tables 5 and 6 explain that the difference in prediction accuracy between the predictors whose thresholds are 50% and 60% is significant. It is also observed from Table 7 that the difference in the selection rates of component predictors for the static hybrid predictors whose thresholds are 50% and 60% is small. Therefore, the improvement rate is increased. As explained in Section 3.4, interference and the aliasing of the prediction tables are reduced in this case, and thus the prediction coverage is improved. This also increase the performance improvement rate.

In the cases of `090.go`, `126.gcc`, and `134.perl`, it is observed that the performance improvement is reduced as the prediction coverage becomes lower and that the dynamic hybrid predictor whose prediction coverage is higher results in the larger performance improvement.

For the static hybrid predictor, we can summarize that the processor performance improvement rate for the static hybrid predictor is comparable to that for the dynamic one when the prediction threshold of the static hybrid predictor is relatively low, when the predictor decides to predict as many load instructions as possible.

Next, let us look at the cooperative predictor. From Fig.17 and Table 7, we observe that both the total prediction coverage and its items are approximately same between the dynamic hybrid predictor and the cooperative predictor. Hence, it is expected that performance contribution of the cooperative predictor is comparable to that of the dynamic hybrid predictor. This expectation is confirmed by the simulation results shown in Fig.18. For most programs except `124.m88ksim` the cooperative predictor improves processor performance most effectively. Therefore, the cooperative predictor is more effective for improving processor performance than the dynamic hybrid predictors since the hybrid predictor is larger in hardware budget than the cooperative predictor due to the predictor selection counter. In other words, the predictor selection counter is not needed to decide which predictor is used from the load value and the address predictors. This means that our expectation explained by Fig.10 in Section 3.5 is correct, which is that the two predictor works complementary.

Considering the misspredict penalty, we have mentioned in Section

5.1 that the number of misspredictions caused by the cooperative predictor is larger than that of the dynamic hybrid predictor and thus the cooperative predictor could not attain performance contribution comparable to the dynamic hybrid predictor if the miss penalty was severe. From Fig.18, it is found that this forecast is imaginary fear. Due to the instruction reissue mechanism, the misspeculation penalty is very small. Thus, misspredictions does not result in serious impact on processor performance.

From these investigations, we have found that among the combinations evaluated in this paper the cooperative predictor is the most effective one consisting the load value and the address predictors, because the cooperative predictor does not relay on the profiling which is essential for the static hybrid predictor and because it removes the hardware cost of the selection counter which is included in the dynamic hybrid predictor.

6. Conclusions. In this paper, we evaluate a variety of combinations of the load address predictor and the load value predictor in order to improve prediction coverage as well as prediction accuracy. For each instruction, by selecting the more accurate predictor, the prediction accuracy is improved and thus the prediction coverage is increased. We investigate the dynamic hybrid, the static hybrid, and the cooperative predictors. The cooperative predictor is basically the load value predictor, which is supported by the load address predictor when it can not predict a load. The static hybrid predictor and the cooperative predictor have the benefit that the hardware cost of the selecting mechanism is removed. We evaluated the predictors using a cycle-by-cycle simulator and found the followings.

- The contribution to processor performance of the value predictor is in general larger than that of the address predictor.
- The contribution of the static hybrid predictor to processor performance is comparable to that of the dynamic one.
- The component value and address predictors are effective to different instruction classes, and thus any adaptive mechanism is not needed to select one from the two predictors.
- The performance contribution of the cooperative predictor is also comparable to that of the dynamic hybrid predictor.

In summary, the cooperative predictor is the best choice among the combinations of the load value and the address predictors.

Acknowledgments. This paper is one result of our ongoing research in high performance microprocessor *COSMOS* at Kyushu Institute of Technology. More information is available at `http://www.mickey.ai.kyutech.ac.jp/~tsato/`. The author is grateful to doctors Mitsuo Saito, Haruyuki Tago and Shigeru Tanaka in Toshiba Microelectronics Engineering Laboratory for their continuous encouragements.

REFERENCES

[1] D.Burger and T.M.Austin, "The SimpleScalar tool set, version 2.0", *ACM SIGARCH Computer Architecture News*, 25(3), (1997).

[2] B.Calder, P.Feller, and A.Eustace, "Value profiling", in *Proc. 30th Int. Symp. on Microarchitecture*, 1997.

[3] T-F.Chen and J-L.Baer, "Effective hardware-based data prefetching for high-performance processors", *IEEE Trans. Comput.*, 44(5), (1995).

[4] J.Cortadella and J.M.Llaberia, "Evaluation of A+B=K conditions without carry propagation", *IEEE Trans. Comput.*, 41(11), (1992).

[5] F.Gabbay and A.Mendelson, "Can program profiling support value prediction?", in *Proc. 30th Int. Symp. on Microarchitecture*, 1997.

[6] J.Gonzalez and A.Gonzalez, "Speculative execution via address prediction and data prefetching", in *Proc. 11th Int. Conf. on Supercomputing*, 1997.

[7] M.H.Lipasti and J.P.Shen, "Exceeding the dataflow limit via value prediction", in *Proc. 29th Int. Symp. on Microarchitecture*, 1996.

[8] M.H.Lipasti, C.B.Wilkerson, and J.P.Shen, "Value locality and load value prediction", in *Proc. Int. Conf. on Architectural Support for Programming Languages and Operating Systems VII*, 1996.

[9] S.McFarling, "Combining branch predictors", WRL Technical Note TN-36, Digital Western Research Laboratory, 1993.

[10] A.I.Moshovos and G.S.Sohi, "Streamlining inter-operation memory communication via data dependence prediction", in *Proc. 30th Int. Symp. on Microarchitecture*, 1997.

[11] G.Reinman and B.Calder, "Predictive techniques for aggressive load speculation", in *Proc. 31st Int. Symp. on Microarchitecture*, 1998.

[12] T.Sato, "Analyzing overhead of reissued instructions on data speculative processors", in *Proc. Workshop on Performance Analysis and its Impact on Design held in conjunction with 25th Int. Symp. on Computer Architecture*, 1998.

[13] T.Sato, "Reducing miss penalty of load value prediction using load address prediction", in *Computer Architecture '99 Australasian Computer Science Communications*, 21(4), ed. J.Morris, (Springer-Verlag Singapore, 1999).

[14] T.Sato, "Improving efficiency of dynamic speculation via data address prediction using instruction reissue mechanism", *Trans. of IPSJ*, 40(5), (1999) (In Japanese). Parts of this paper are presented in T.Sato, "Speculative resolution of ambiguous memory aliasing", in *Innovative Architecture for Future Generation High-Performance Processors and Systems*, eds. A.Veidenbaum and K.Joe, (IEEE-CS Press, 1998).

[15] T.Sato, "Load value prediction via two-hop reference address renaming", *Trans. of IPSJ*, 40(5), (1999) (In Japanese). Parts of this paper are presented in T.Sato, "Load value prediction using two-hop reference address renaming", in *Proc. 4th Int. Conf. on Computer Science & Infomatics*, 1998.

[16] T.Sato, "Profile-based selection of load value and address predictors", in *High Performance Computing*, LNCS#1615, eds. C.Polychronopoulos, K.Joe, A.Fukuda, and S.Tomita, (Springer-Verlag Berlin/Heidelberg, 1999).

[17] Y.Sazeides, S.Vassiliadis, and J.E.Smith, "The performance potential of data dependence speculation & collapsing", in *Proc.29th Int. Symp. on Microarchitecture*, 1996.

[18] Y.Sazeides, J.E.Smith, "The predictability of data value", in *Proc. 30th Int. Symp. on Microarchitecture*, 1997.

[19] G.S.Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers", *IEEE Trans. Comput.*, 39(3), (1990).

[20] D.M.Tullsen and J.S.Seng, "Strageless value prediction using prior register values", in *Proc 26th Int. Symp. on Computer Architecture*, 1999.

[21] G.Tyson and T.M.Austin, "Improving the accuracy and performance of memory communication through renaming", in *Proc. 30th Int. Symp. on Microarchitecture*, 1997.

[22] K.Wang and M.Franklin, "Highly accurate data value prediction using hybrid predictors", in *Proc. 30th Int. Symp. on Microarchitecture*, 1997.

[23] L.Widigen, E.Sowadsky, and K.McGrath, "Eliminating operand read latency", *ACM SIGARCH Computer Architecture News*, 24(5), (1996).