A New Dispatching Mechanism – Using Value Classification and Indexing – for Value Predictors[*]

PO-JEN CHUANG AND YU-SHIAN CHIU
Department of Electrical Engineering
Tamkang University
Tamsui, Taipei County, 251 Taiwan
E-mail: pjchuang@ee.tku.edu.tw

Value prediction can be used to break data dependency between instructions, ensuring simultaneous handling of multiple instructions in processors. When such instruction-level parallelism is lifted, it helps reduce a processor's idle time and thus enhances the performance. To improve the accuracy of value prediction at low additional cost, this paper presents a new dispatching mechanism with special data value classifications and simple indexing devices. The proposed dispatching mechanism classifies data values based on their distribution patterns to attain more balanced utilization of all prediction entries. Three different value indexing devices are also introduced to set up more sophisticated and precise predicting steps. Experimental evaluation shows that the new dispatching mechanism is able to enhance the accuracy of value predictors at small extra cost.

Keywords: data dependency, instruction-level parallelism, value predictors, prediction accuracy, dispatching mechanism, value classification, value indexing

1. INTRODUCTION

Prediction techniques are hotly pursued in recent years to increase the instruction-level parallelism for processors. Value prediction, one of the major prediction techniques, can be used to break data dependency between instructions, thus ensuring simultaneous handling of multiple instructions in processors. With satisfying prediction accuracy, value prediction can raise the instruction-level parallelism (ILP), help reduce idle time and as a result lift up the overall performance for processors.

For illustration, an original pipeline and a pipeline implemented with value prediction are depicted in Figs. 1 and 2.

As Fig. 2 shows [1], when a pipeline implemented with value prediction comes to the instruction state Fetch, the program counter (PC) will be hashed to the value history table (VHT) and then make prediction according to the value stored in the VHT. If the prediction value is correct (*i.e.*, equal to the result of the instruction), instructions can be executed more quickly due to enhanced ILP; if incorrect, the result of the instruction is used to process further work. Instructions that use the incorrect prediction value in their pre-execution must now be re-executed.

Received December 13, 2004; revised April 15 & July 8 & August 31, 2005; accepted October 19, 2005.
Communicated by Chu-Sing Yang.
[*] A preliminary version of this paper was presented at the *2003 National Computer Symposium*, Dec. 2003.
[*] This work was supported in part by the National Science Council of Taiwan, R.O.C., under grants No. NSC 90-2213-E-032-019 and NSC 95-2745-E-032-001-URD.

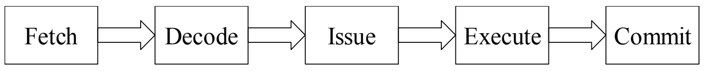

Fig. 1. An original pipeline.

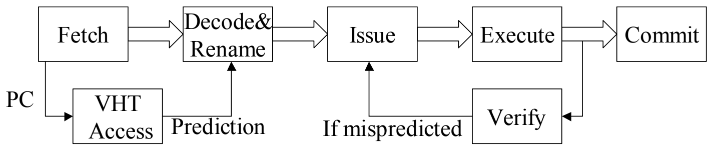

Fig. 2. A pipeline with value prediction.

Table 1. A superscalar flowchart with 3 dependent instructions I, J and K.

Pipeline	Base Superscalar						With VP			
	1	2	3	4	5	6	1	2	3	4
Fetch	I,J,K						I,J,K			
Dec&Ren		I,J,K						I,J,K		
Execute			I	J	K				I,J,K	
Commit				I	J	K				I,J,K

The flowchart of a superscalar working with or without value prediction is provided in Table 1 to show the difference [1]. While the original pipeline needs 6 processing cycles to handle the instructions, the pipeline with value prediction (VP) takes only 4 cycles (with correct prediction values of instructions I and J) to finish the same job.

In case the prediction value turns out incorrect, the additional cost resulting from making predictions will be wasted. Value predictors, such as the last outcome, stride, two-level and hybrid predictors [2-9], avoid such cost waste by adding a *threshold* value [10] to their mechanisms. When a predictor fails to satisfy the threshold, prediction will be aborted.

Some advanced techniques, either history-based, stride-based, hybrid-based or context-based, have been introduced to increase prediction accuracy for value predictors [11-13]. These designs work in the single instruction level and make predictions according to previous instruction results. To store previous instruction results, we need storages. The size of the storage and the way to store data in it may significantly affect the performance of a value predictor in terms of prediction accuracy and hardware cost. It is clear that more stored "history" data may lead to higher prediction accuracy, but without proper dispatching designs, the needed cost will also rise.

A new and cheap dispatching mechanism involving special value classification and indexing is developed in this paper to improve prediction accuracy for lower-cost value predictors, such as the last outcome and stride predictors, at small additional cost. In the proposed dispatching architecture, data values are classified as positive/negative and specific/others based on their distribution patterns to attain even utilization of all prediction entries. Three indexing techniques, including indexing by the last instruction result, indexing by the 2-bit counter, and indexing by the last instruction result and by collocating the positive/negative bit fields, are presented to aid the dispatching operation. Simulation results show that such a new dispatching mechanism is able to improve prediction accuracy for certain value predictors at negligible extra cost.

2. BACKGROUND

Value predictors can be categorized as stride-based (the stride predictor), context-based (the last outcome, two-level and finite context method predictors), and hybrid (the hybrid predictor).

2.1 The Last Outcome Value Predictor [2-5]

The last outcome value predictor uses the instruction address, also known as the program counter (PC), as the hash function's input. The hash function and decoder have four outputs: the index of the classification table (CT), the index of VHT, TAG_CT and TAG_VHT. The "indexed" TAG_CT and TAG_VHT (indexed by the hash function and decoder) are respectively compared with the "calculated" TAG_CT and TAG_VHT (calculated from the hash function). If one of the comparisons turns out unequal, no prediction will be made. If both the indexed TAG_CT and TAG_VHT equal their calculated counterpart, the indexed counter in CT will be compared with the set threshold. When the counter is equal to or bigger than the set threshold, prediction will be made; otherwise prediction is aborted. The predictor uses the values stored in VHT to make predictions. When a prediction is correct (*i.e.*, when the prediction value equals the instruction result), the counter will be increased by 1; otherwise it is decreased by 1.

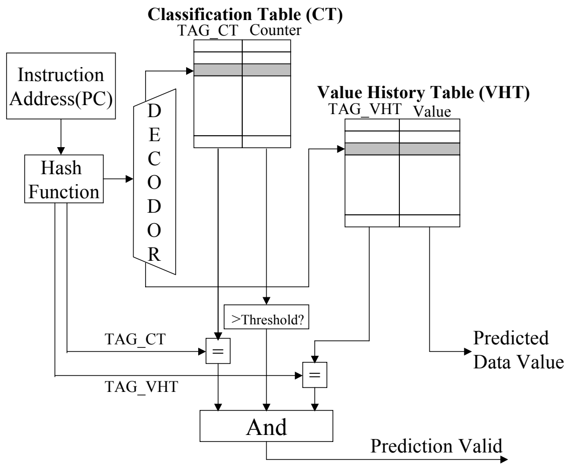

Fig. 3. The last outcome predictor.

The architecture of the last outcome predictor [4], shown in Fig. 3, consists of 5 fields (TAG_CT, TAG_VHT, Threshold, Counter and Value) and 4 functions (And, Hash Function, Decoder and =). *TAG_CT* and *TAG_VHT* store the identity of the instruction currently mapped to the entry; the main function is to check if the entry corresponds to the instruction to be predicted. *Counter* records the prediction history: The counter will be increased or decreased by 1 when the prediction result is correct or incorrect. If the counter reaches the set *Threshold*, prediction will be made. *Value* stores the last result of the instruction. *Hash Function* classifies instructions. *Decoder* helps decode indexes. = checks if the prediction will be made or not, depending on the counter and the threshold.

2.2 The Stride Value Predictor [3, 6, 7]

The stride predictor has four fields, tag, value, state and stride, at each entry (see Fig. 4 below). There are three values in the *state* field: initial, transient and steady. Predictions are made according to the *stride*, *i.e.*, the difference of the two most recent instruction results, and the prediction value will be *value* + *stride*.

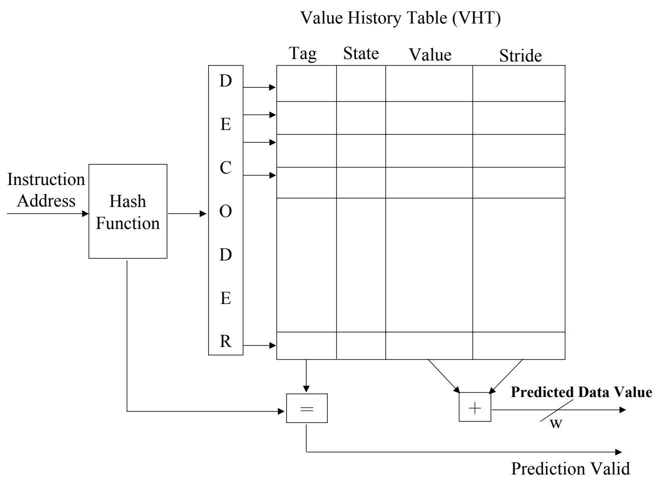

Fig. 4. The stride value predictor.

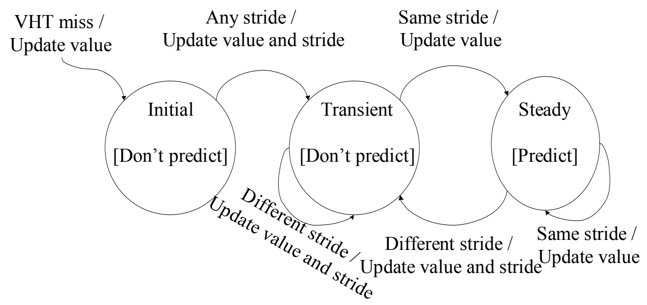
VHT miss / Update value
Any stride / Update value and stride
Same stride / Update value
Initial [Don't predict]
Transient [Don't predict]
Steady [Predict]
Different stride / Update value and stride
Different stride / Update value and stride
Same stride / Update value

Fig. 5. The state transition of the stride predictor.

Fig. 5 shows the three state values: initial, transient and steady. Predictions will be made only in the steady state. The following is an example of the state changes. When an instruction is executed for the first time, its result (result 1) is stored into value, and the state is set to *initial*. When the instruction is executed again, the result (result 2) is stored into value, (result 2 − result 1) is stored into stride and the state is set to *transient*. If another instance of the same instruction happens and produces result 3, the result is again stored into value. Then compare (result 3 − result 2) with the stride value: If equal, the state is set to *steady* and prediction is made by adding together the value and stride fields. When a different stride appears, the state is then set to transient.

2.3 The Two-Level Value Predictor [3, 8]

A two-level predictor contains two tables: the value history table (VHT) and the pattern history table (PHT), as shown in Fig. 6. VHT includes tag, LRU info, data values and a value history pattern (VHP). The two decoders in Fig. 6 are used to decode the values generated in the hash function and VHT. Data values store the last 4 unique results produced by the same instruction. LRU info records the index (0, 1, 2, 3) of the least recently entered value in the data values field. For example, if the data values field stores [21, 35, 46, 98] with 35 being the least recently entered value, the LRU info value will be "1". When a new unique data value is produced, it replaces the least recently entered value of the data values field. VHP stores the $2p$-bit pattern: p refers to the last instruction result and the pattern is the binary sequence code used to index to the data values field. The two-level value predictor has its name because it uses a two-level index to reach PHT. The four up/down counters (C_0, C_1, C_2, C_3) in PHT are independent. If the prediction value equals the instruction result, the counter value will be increased by 3 (not to exceed a predetermined maximum counter value); otherwise it is decreased by 1 (to the minimum of 0). The two-level value predictor works as follows. When an instruction is fetched, its address is used to index to an entry in VHT. If the tag is matched, prediction is made. VHP is sent to index to an entry in PHT. The predicted value is then decided from the max of the 4 counter values in PHT. If the max counter value is greater

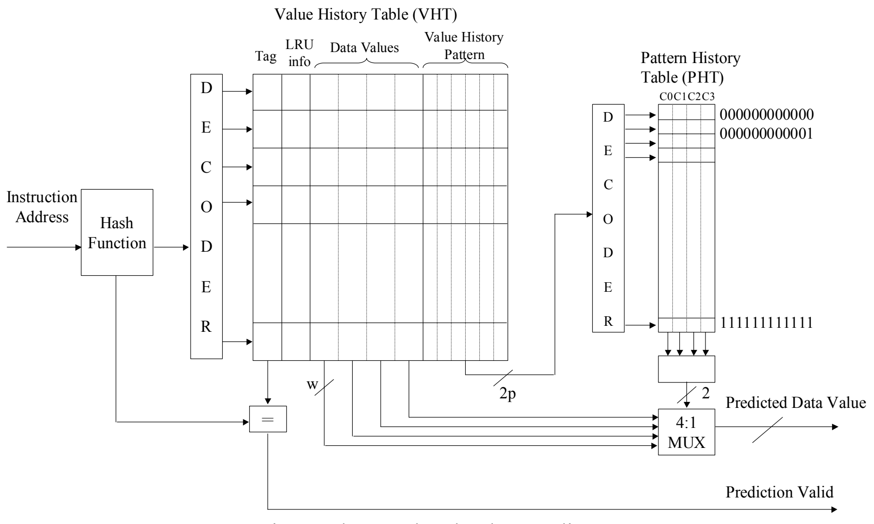

Fig. 6. The two-level value predictor.

than or equal to the threshold value, PHT will send a 2-bit binary code (*i.e.*, 00, 01, 10, or 11) to MUX which then uses the binary code to select a data value as the predicted value. If the max counter value is less than the threshold value, prediction is cancelled.

2.4 The Hybrid Value Predictor [3, 9]

The hybrid value predictor is a combined mechanism of the two-level and stride predictors, retaining advantages and avoiding disadvantages of both predictors. As mentioned, the two-level predictor makes no prediction when all of the counter values are smaller than the threshold value, and the stride predictor makes no prediction in the initial/transient states. As a combined mechanism, the hybrid predictor is able to avoid the problem by making predictions with one predictor when the other fails.

2.5 The Finite Context Method (FCM) Value Predictor [11, 12]

The FCM predictor predicts the next value based on a finite number of preceding values. Specifically, an order k FCM predictor uses k preceding history values as the context. Counters are used to record repetitions of values immediately following some special context patterns, and the value with the maximum counter is selected as the prediction value. As the size of counters is limited, if one counter reaches its maximum value, all of the counters that record the same context will be reset to half. The advantage of going with small sized counters is that the recorded data are always the latest or the most recently used. In general, the n different FCMs leveled 0 to $n - 1$ will compare the context, starting from the highest level to the lower ones, to find the desirable predicted value (with the highest-level).

3. COST EVALUATION FOR VARIOUS PREDICTORS

In developing value predictors, enhancing prediction accuracy has been the major concern; cost is seldom considered or analyzed. As a consequence, accuracy is often pursued at the cost of expensive hardware complexity. For better understanding of the situation, this section provides a cost analysis on existing value predictors.

Note that the sizes of value predictors in our later discussion are decided based on the architectures given in section 2. We assume that there are 12 bits in the tag field, 64 bits in the value field and the number of entries is 4,096. However, there are different parameters for different predictors. For instance, we use 2 bits in the state field and 6 bits in the stride field of the stride predictor. For the first level of the two-level predictor, there are 2 bits in the LRU field and 2×6 bits in the VHP field; for the second level, the number of entries is 4,096 and the second PHT field has 2×4 bits. The same setting (for the above stride and two-level predictors) is adopted for the hybrid predictor. The FCM predictor has 4,096 bits in both the first and second level entries, with 4×64 bits used to calculate the PHT field in the second level.

Table 2 lists the parameters of each value predictor. The parameters are obtained following the Simplescalar simulator [9, 13, 14]. Take the stride predictor as an example. The first level entry is the index of VHT, indicating VHT can store a total of 4,096 entries. As the predictor has no second level, the second level entry and the dispatch bits

Table 2. Parameters of value predictors.

	First level entry	Second level entry	No. of data (per entry)	Dispatch bits (per entry)	Threshold
Last	4,096		1		2
2 level	4,096	4,096	4	12	3
FCM2	4,096	4,096	2 + 4	0	3
Hybrid	4,096	4,096	4 + 1	12	3
Stride	4,096		1 + 1		

Table 3. Cost for value predictors.

	No. of data (per entry)	Difference value (per entry)	Dispatch bits (per entry)	No. of PN-bits (per entry)	Total cost (total bits)
Last	1	0	0		304K
2 level_4	4	0	12		1160K
FCM2	2 + 4	0	0		1536K
Hybrid_4	4	1	12		1176K
Stride	1	1	0		336K
2 level_2	2	0	12		632K
Hybrid_2	2	1	12		648K
Last + Single + PN-bit + SOdisp	1	0	0	1	304K + 4K
Stride + Single + PN-bit + SOdisp	1	1	0	1	336K + 4K

are ignored. The parameter in the number of data is 1 + 1 because it needs to store a data value and a difference value; there is no threshold because prediction is made according to the states, not the threshold value.

Table 3 lists the required cost for various value predictors. The cost is counted by the needed bits which reflect both the storage and time overhead. (Time overhead due to comparators of a predictor is dominated by the bits involved in comparisons.) As we can see, the last and stride predictors have the least cost (304K bits for the former and 336K bits for the latter) because of their simple architectures.

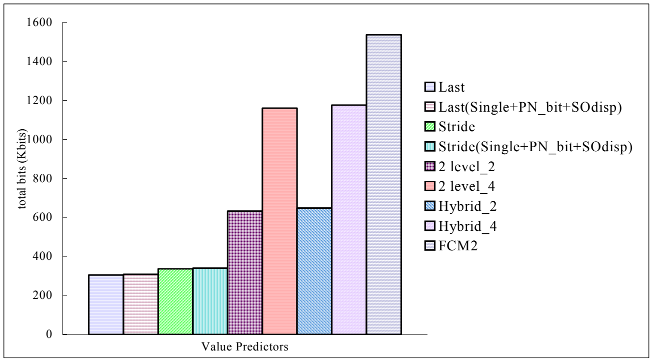

Fig. 7. Cost for various value predictors.

As mentioned, the goal of this paper is to present a cheap dispatching mechanism which can enhance prediction accuracy for lower-cost value predictors, such as the last outcome and stride predictors, to compete with the performance of higher-cost predictors, such as the two-level predictor, at small extra cost. To pinpoint the significance of our investigation, the cost of the last outcome and stride predictors incorporated with our proposed dispatching mechanism is also included in Table 3 and Fig. 7. Note that the cost for adding the proposed dispatching mechanism PN-bit + SOdisp is only 4K which is obtained from PN-bit (1 bit) × 4,096 (number of the first level entries) + Single PN-bit (1 bit) × 1 = 4,097 bits, to be fully discussed in later sections.

4. THE PROPOSED DISPATCHING MECHANISM

In our attempt to develop a new dispatching mechanism, we first classify values into positives and negatives but find that dispatching by such a value classification does not raise prediction accuracy as much as expected. Speculating that this might be caused by uneven distribution of the positive and negative values, we conduct an analysis on value distribution. The result confirms our speculation: There are indeed more positive values than negative ones. A more detailed value analysis is then undertaken to check the number of values in 7 divided locations (shown later in Fig. 16). It turns out the number of values in one specific location (0~268,435,456) almost equals the total number of values in all the other locations. This finding leads to a more effective dispatching design -- dispatching by the specific (location) /others (locations) value classification. (Note that in determining the range of the divided locations, we first partition values into positive and negative but find the result undesirable because of uneven distribution of the positive and negative values. We then attempt to divide the locations into 4 and then more areas until we find out the number of values in interval 0~268,435,456 almost equals the total number of values in all other intervals.)

Besides value classification, three value indexing techniques are also presented to strengthen the proposed dispatching mechanism.

4.1 Previous Dispatching

A large number of value predictors adopt the following dispatching architecture:

$$(\text{PC div } 4) \bmod 2^n \qquad (1)$$

where 2^n is the amount of prediction table entries, PC is the instruction address and mod is the modulo operation.

Built based on programming analysis, the dispatching architecture is simple, fast, easy to implement and able to yield proper performance. It consists of two major operations. First, divide the instruction address by 4. As the last two bits in the instruction address tend to be zero, taking off these two least significant bits will help attain more equalized dispatching (to the prediction table). The formula (PC div 4) can be substituted by right rotating 2 bits as follows:

$$PC >> 2. \qquad (2)$$

Then index to the limited prediction table. The result of (PC div 4) is now divided by 2^n and the remainder is taken as the index value. Taking the remainder as the index value is essential since the number of data to be saved is large and the hardware is limited.

4.2 The Proposed Dispatching Mechanism

The proposed dispatching mechanism is designed following careful investigation on previous dispatching practices and aims to reach favorable performance with possibly the least hardware complexity.

The new dispatching mechanism involves value classification and indexing. Value classification indicates how to classify the prediction table. As mentioned, data values can be classified as positive/negative (bigger or smaller than zero) or as specific/others (in the specific location of 0~268,435,456 or others). Value indexing, on the other side, demonstrates how to index to the classified VHT. For example, the classified VHT for positive/negative value classification will be PVHT (positive value history table) or NVHT (negative value history table), and for specific/others value classification, it will be SVHT (specific value history table) or OVHT (others value history table).

4.2.1 Value classification

(1) The positive/negative value classification dispatching

Data values are first compared with and separated by zero. After analyzing the values in the benchmarks, we find that values bigger than zero outnumber values smaller than zero and set up the positive/negative value classification dispatching. As shown in Fig. 8, the original VHT is equally divided into PVHT and NVHT to save positive and negative values. (The size of PVHT and NVHT together equals to that of the original VHT, making it possible to conduct fair performance comparison between the new dispatching architecture and previous ones.)

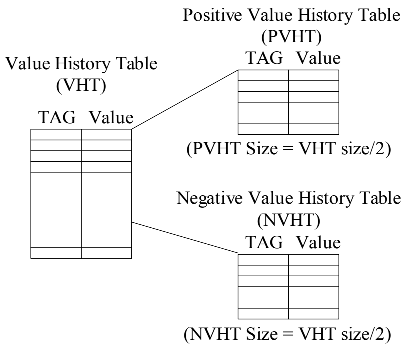

Fig. 8. The dispatching architecture with positive/negative value classification.

(2) The specific/others value classification dispatching

Our value distribution analysis reveals an important fact that the number of values in location 0~268,435,456 almost equals the total number of values in all the other locations. This finding initiates the specific/others value classification dispatching method in Fig. 9.

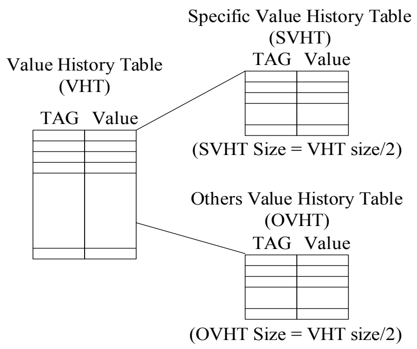

Fig. 9. The dispatching architecture with specific/others value classification.

In this dispatching architecture, the original VHT is equally divided into SVHT and OVHT. SVHT is used to save values located in the specific value location (0~268,435,456 in our simulation); OVHT saves values which are otherwise located.

Due to a more balanced utilization of all prediction entries, specific/others value classification is shown through simulation to outperform positive/negative value classification in raising prediction accuracy. This is because specific/others value classification does a better dispatching job by evenly distributing indexing to SVHT and OVHT without adding extra hardware. (The operations of SOVHT and PNVHT are similar. However, we choose to explain our approaches through the operation of PNVHT in the next section for easier understanding.)

4.2.2 Value indexing

(1) Indexing by the last instruction result

The last instruction result (being positive or negative) is used during prediction to select a table (with positive or negative values). To index by the last instruction result, we need to add one additional bit – the single PN-bit. The single PN-bit stores the last instruction result which probably indexes to different instructions. To give an example, when dispatching is conducted under positive/negative value classification: If the single PN-bit indicates positive (= 1), we choose the table that stores positive values to make prediction; if the bit is negative, then choose the table with negative values for prediction. Once the table is decided, prediction will be made following the steps of the original value predictor. If the prediction result is positive, update PVHT; otherwise, update NVHT. The advantage of indexing by the last instruction result lies in that it requires only one more bit and the additional operation involves only the comparison of the single

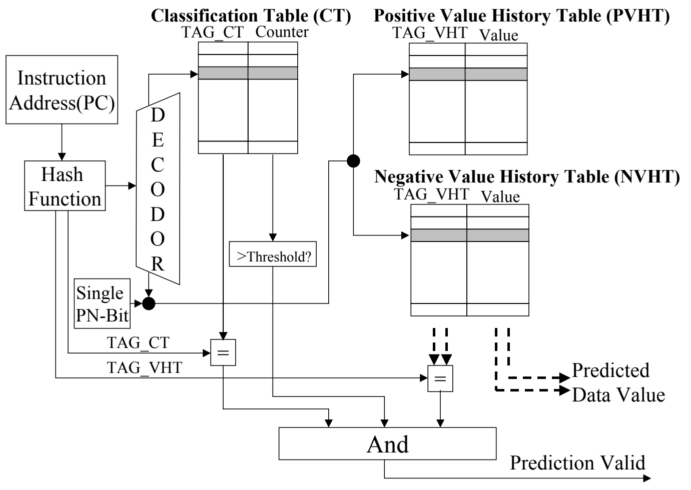

Fig. 10. The architecture for value indexing by the PN-bit of the last instruction result.

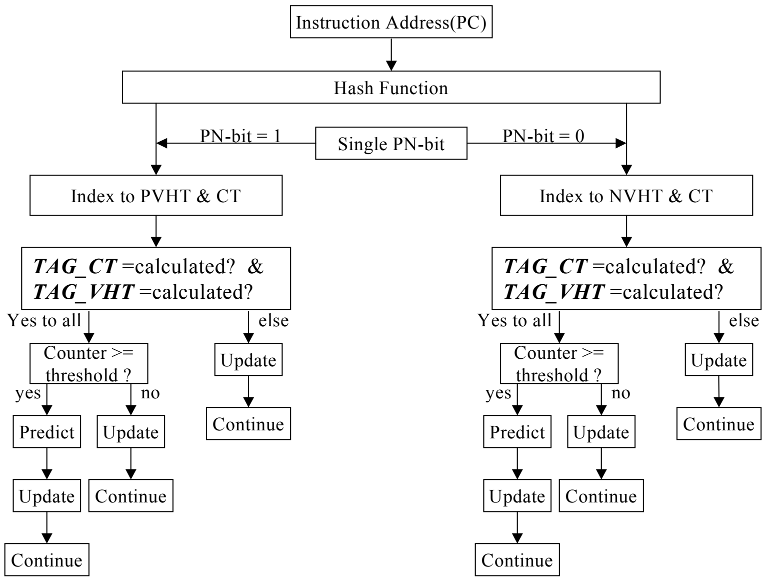

Fig. 11. The flowchart for value indexing by the PN-bit of the last instruction result.

PN-bit with 1 (when both values are equal, index to PVHT or SVHT; otherwise, index to NVHT or OVHT). The indexing architecture is shown in Fig. 10; its flowchart is depicted in Fig. 11.

(2) Indexing by the 2-bit counter

To index by the 2-bit counter, we first dispatch to the index table (IT) whose size is one half of the original VHT. The indexed IT 2-bit counter will decide which table is to be used for prediction. If the 2-bit counter is bigger than or equal to 2, take PVHT or

SVHT for prediction; otherwise, take NVHT or OVHT. Fig. 12 illustrates the architecture of the last outcome value predictor indexing by the 2-bit counter; its flowchart is depicted in Fig. 13. The 2-bit counter first performs value classification and records the significant bit of the last instruction result. With a positive result, update the 2-bit counter of the indexed entry from IT by adding 1 to it (not to exceed the maximum counter value); otherwise, decrease the value by 1 (if it is bigger than the minimum counter value). As the 2-bit counter records the significant bit of the last instruction result and reflects it in the counter value, it will help choose a most recently and frequently used

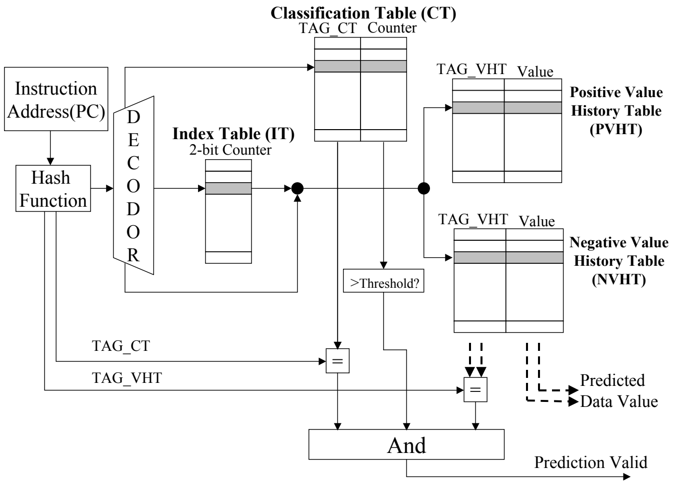

Fig. 12. The architecture of the last value predictor using the 2-bit counter for indexing.

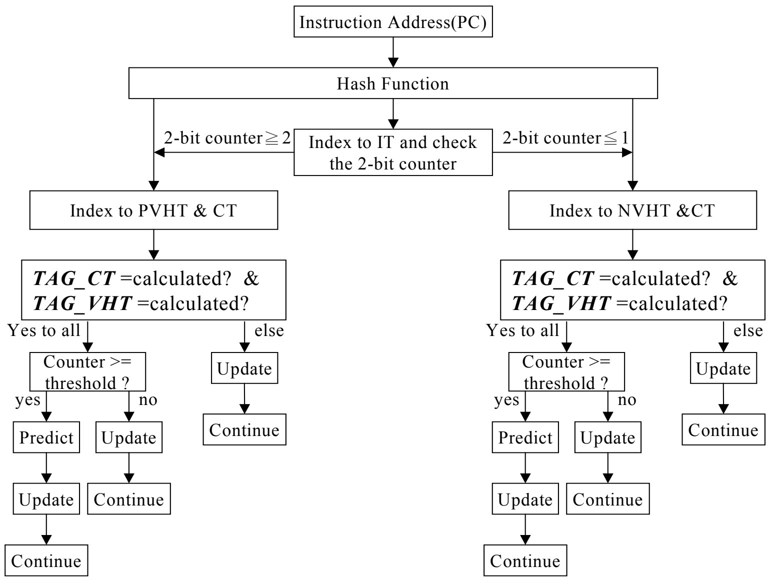

Fig. 13. The flowchart of the last value predictor using the 2-bit counter for indexing.

prediction table to do the prediction. For example, if the value of the 2-bit counter in an entry remains 0 all the time − indicating the recent values for the counter are always negative, NVHT or OVHT will be chosen as the prediction table.

(3) Indexing by the last instruction result and by collocating the positive/negative bit fields

This indexing operation needs an additional bit, the single PN-bit. It also needs a PN-bit field in each entry to record the positive or negative status of the last result of the instruction mapped to the entry. Fig. 14 shows the last outcome predictor with such an indexing technique. (Dashed lines indicate connection with both PVHT and NVHT. For example, the TAG_VHT value generated from the hash function needs to compare with the TAG_VHT value stored in PVHT or NVHT, and the choice of PVHT or NVHT is decided by the single PN-bit.) Fig. 15 gives the corresponding flowchart. The "indexed" TAG_CT and TAG_VHT are respectively compared with the "calculated" TAG_ CT and TAG_VHT, as mentioned in section 2.1. If both the indexed TAG_CT and TAG_ VHT equal their calculated counterparts, compare the PN-bit field with the single PN-bit. If both are positive, choose the table with positive values to make predictions; otherwise choose the table with negative values. Prediction is then handed over to the original predictor. After the prediction is made, update only the indexed table classified by the most significant bit.

With such a comparison mechanism (the PN-bit field), prediction will be more precisely made. Besides, the added PN-bit field makes it possible to store positive or negative values at the same entry in the prediction table. To store positive or negative values is decided by the current instruction results. If the recent instruction results are all positive, indexing will be directed to the prediction table with positive values and the PN-bit will be 1 – indicating the indexed table is with positive values or in location 0~268,435,456. On the other hand, if the recent instruction results are all negative, indexing will be directed to the prediction table with negative values and the PN-bit will be 0, *i.e.*, the indexed table is with negative values or values in locations other than 0~268,435,456.

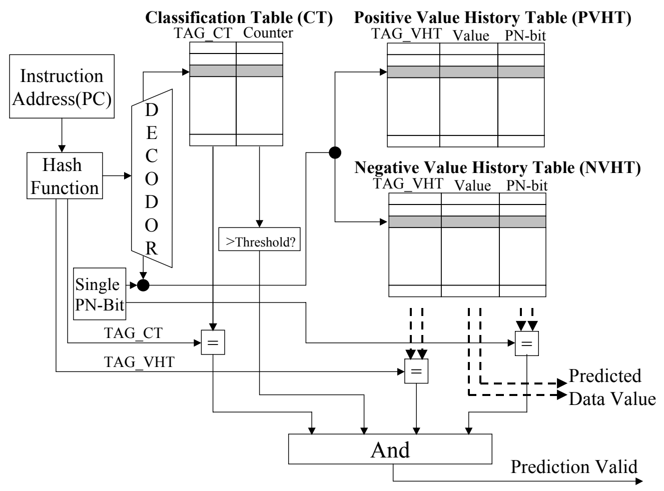

Fig. 14. The architecture of the last value predictor indexing by the last instruction result and by collocating the positive and negative bit fields.

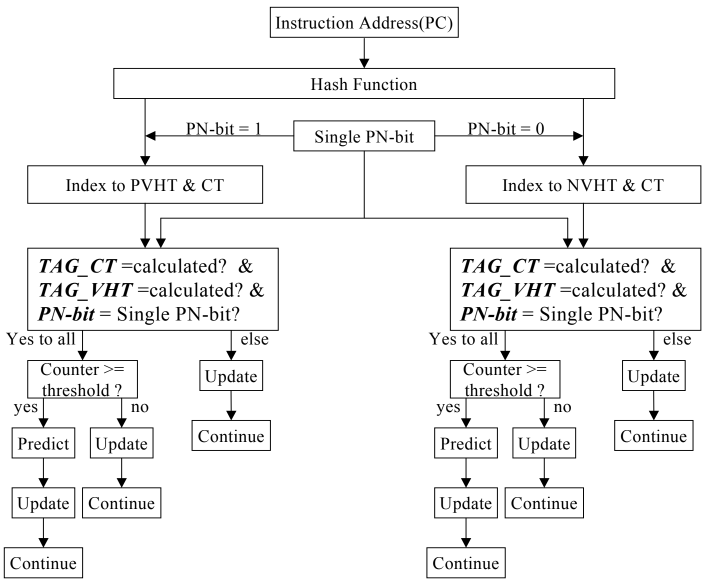

Fig. 15. The flowchart of the last value predictor indexing by the last instruction result and by collocating the positive and negative bit fields.

5. EXPERIMENTAL EVALUATION

5.1 The Simulation Model, Benchmarks and Value Distribution

Simulations can be trace-driven or program-driven. In trace-driven simulations, trace data are saved in computers and used in simulating associated programs. Conducting trace-driven simulations needs a large storage to save the collected trace data. A program- driven simulation uses trace data produced directly by the program itself and as a result each simulation will take longer running time.

This research adopts the trace-driven simulation, using Simplescalar 2.0 [14] in the Linux system. Trace data are produced by Simplescalar and used in the Visual C++ program under Windows 2000. The simulation uses SPECint95 as benchmarks. Description of the 8 benchmarks is given in Table 4; the input and the total number of instructions for each benchmark are provided in Table 5.

Table 4. Description of benchmarks.

Gcc	Compiler
Go	A chess game
M88ksim	A simulator for the 88000 processor
Compress	Data compression program using adaptive Lempel-Ziv coding
Li	Lisp Interpreter
Ijpeg	Jpeg encoder
Perl	Perl interpreter
Vortex	A single user object-oriented database transaction benchmark

Table 5. The input and total number of instructions for each benchmark.

Benchmark	Input	Total execution counts
Gcc	amptjp.i(train)	1,276,486,588
Go	2stone9.in(train)	548,177,629
M88ksim	ctl.in(train)	48,269,168
Compress	test.in(train)	35,818,826
Li	train.lsp(train)	183,304,340
Ijpeg	vigo.ppm(train)	1,464,821,356
Perl	primes.in(train)	10,519,469
Vortex	Vortex.in(train)	2,520,154,589

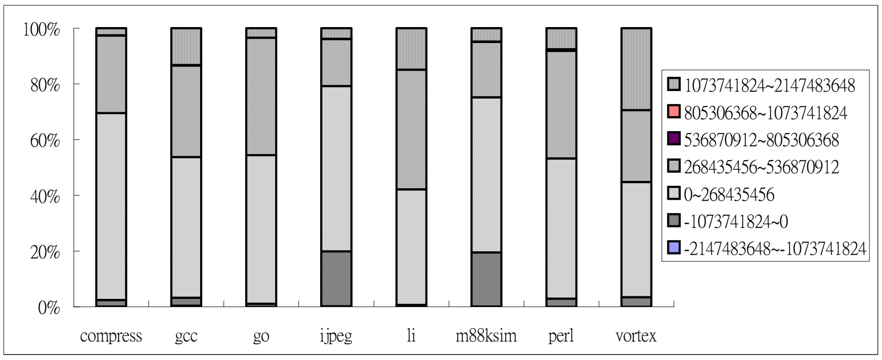

Fig. 16. Value distribution patterns for various benchmarks.

Value distribution patterns for the 8 benchmarks are presented in Fig. 16. The results show a common trend for most benchmarks – the number of values located in 0~268,435,456 is larger than the total number of values in the other locations. The difference is especially evident in benchmark compress.

5.2 Simulation Results

The following terms are provided to facilitate further discussion. (Note that PNVHT and SOVHT need to work with either the Single or the PN-bit indexing.)

Original: the original architecture of a value predictor
Single: the original architecture of a value predictor with the proposed single PN-bit indexing
PN-bit: the original architecture of a value predictor with the proposed PN-bit field indexing
PNdisp: the original architecture of a value predictor with the original VHT being virtually partitioned into two equal sized tables, PVHT and NVHT (the positive/negative value classification dispatching, using PNVHT)
SOdisp: the original architecture of a value predictor with the original VHT being virtually partitioned into two equal sized tables, SVHT and OVHT (the specific/others value classification dispatching, using SOVHT)

+: combined with the proposed value indexing and classification

5.2.1 Accuracy vs. dispatching by positive/negative value classification

Figs. 17 and 18 collect prediction accuracy for the last outcome and stride predictors indexing with and without the single PN-bit/PN-bit field under positive/negative value classification dispatching. Fig. 17 displays that with the proposed single PN-bit/PN-bit field indexing, the last outcome predictor increases the average prediction accuracy. The increase may come from a more sophisticated indexing step: Prediction starts only when both the tag and the PN-bit are compared. In Fig. 18, the stride predictor with the

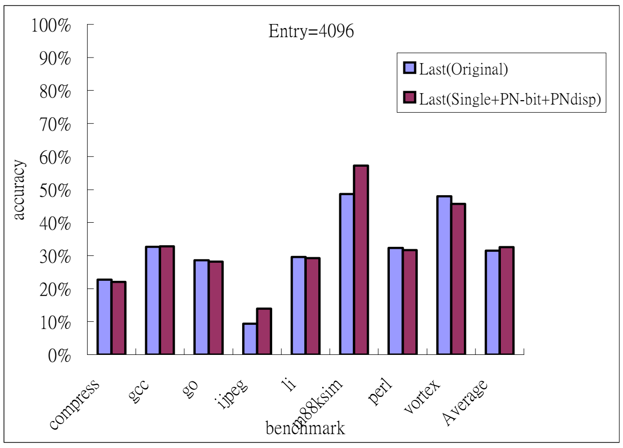

Fig. 17. Prediction accuracy for the last outcome predictor indexing with and without the single PN bit/PN-bit field under positive/negative value classification.

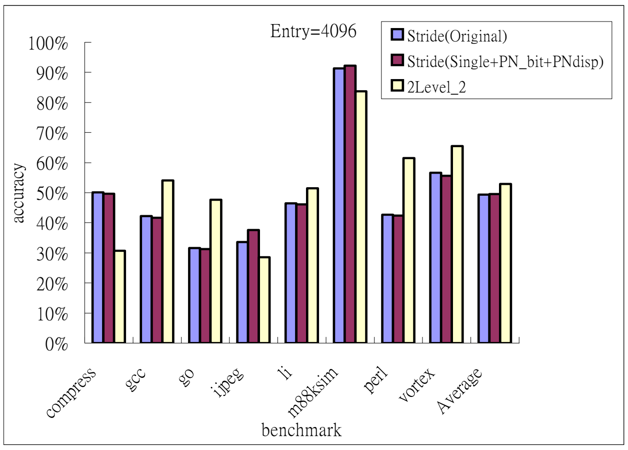

Fig. 18. Prediction accuracy for the stride predictor indexing with and without the single PN-bit/PN-bit field under positive/negative value classification.

proposed PNdisp dispatching produces lower prediction accuracy in some benchmarks but yields higher average accuracy (over the 8 benchmarks) than the original architecture.

We find out during simulation that value distribution in benchmarks ijpeg and m88ksim differs from that in the other benchmarks: The two benchmarks have distinctly larger number of negative values. That explains why both the last outcome and stride predictors turn over higher prediction accuracy in these two benchmarks when dispatching by positive/negative value classification.

5.2.2 Accuracy vs. dispatching by specific/others value classification

Based on our analysis on benchmarks (that the amount of values located in 0~ 268,435,456 approximately equals the total number of values located in all the other locations), we develop a special value classification method to conduct more efficient utilization of all prediction entries. The specific/others value classification dispatching method stores the values in 0~268,435,456 in one table and the values in all other locations in another table. When the amount of values is thus divided into two almost equal parts and stored in two tables, all prediction entries can be utilized in a more balanced and efficient way.

Figs. 19 and 20 give prediction accuracy for the last outcome and stride value predictors indexing with and without the single PN-bit/PN-bit field under the specific/others value classification. The result in Fig. 19 shows the last outcome predictor generates clear accuracy gain in nearly all benchmarks when indexing by the single PN-bit/PN-bit field, and such accuracy gain is obtained at a small additional cost of 4K.

In Fig. 20, the stride predictor indexing by the single PN-bit/PN-bit field produces better accuracy in all 8 benchmarks, and the enhanced average accuracy comes close to that of the two-level_2 value predictor. It should be noted that the two-level_2 predictor takes 632K to attain the slightly superior accuracy, while the stride predictor with the single PN-bit/PN-bit field indexing requires only 340K to get close results. (340K = 336K + 4K. 336K is the stride predictor's original cost; 4K is the cost for the proposed

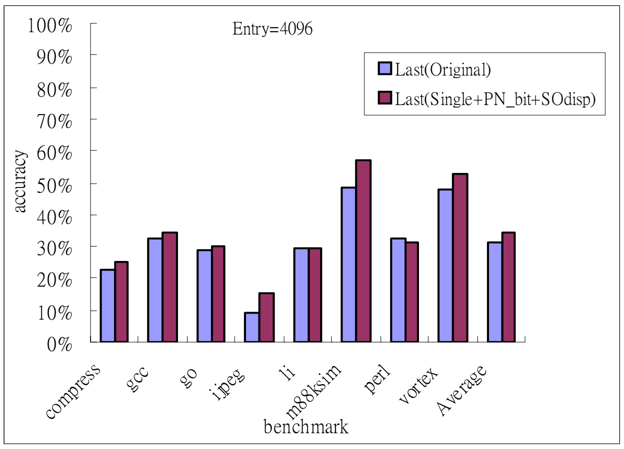

Fig. 19. Prediction accuracy for the last outcome predictor indexing with and without the single PN-bit/PN-bit field under specific/others value classification.

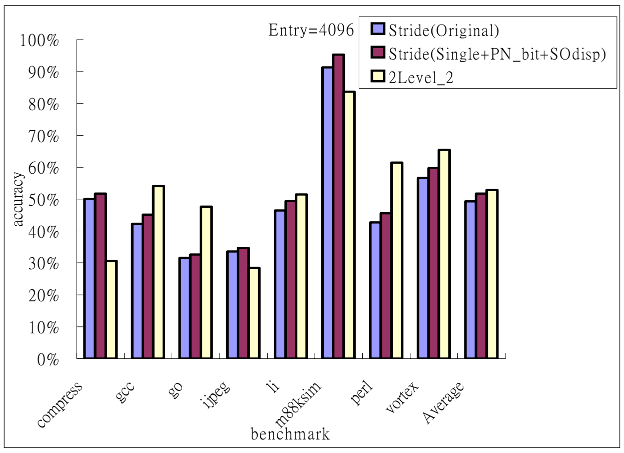

Fig. 20. Prediction accuracy for the stride predictor indexing with and without the single PN-bit/ PN-bit field under specific/others value classification.

single PN-bit/PN-bit field indexing.) Thus, when hardware complexity is taken into account, our proposed indexing technique proves to be efficient in strengthening the performance of value predictors.

6. CONCLUSION

A desirable dispatching mechanism may raise prediction accuracy and thus enhance the performance of value predictors. This paper presents a new dispatching mechanism for value predictors, which involves special value classifications and simple indexing devices, and is demonstrated through experimental evaluation to improve prediction accuracy at very minimal extra cost.

Two value classification methods, positive/negative value classification and specific/others value classification, are introduced according to the special features of value distribution patterns collected in our analysis. The main purpose of dispatching by such value classifications is to attain more balanced utilization of all prediction entries and thus to assist the operation of value predictors. A number of value indexing devices, including indexing by the last instruction result, indexing by the two-bit counter, and indexing by the last instruction result and by collocating the positive/negative bit fields, are also taken to ensure more sophisticated and precise predicting steps.

Extensive simulation runs have been conducted to compare the performance of value predictors dispatching under the two value classifications and different ways of value indexing. The results exhibit that dispatching under specific/others value classification enables predictors to produce better prediction accuracy gain than dispatching under positive/negative value classification, indicating that the former makes it possible to employ prediction entries in a more balanced way. Simulation results also display that predictors indexing by the proposed single PN-bit and the PN-bit field constantly outperform their original forms, and the increased performance, *i.e.*, accuracy gain, is attained at very limited additional hardware cost.

REFERENCES

1. A. Sodani and G. S. Sohi, "Understanding the differences between value prediction and instruction reuse," in *Proceedings of the 31st Annual IEEE/ACM International Symposium on Microarchitecture*, 1998, pp. 205-215.
2. R. Sathe and M. Franklin, "Available parallelism with data value prediction," in *Proceedings of the 5th International Conference on High Performance Computing*, 1998, pp. 194-201.
3. S. J. Lee, Y. Wang, and P. C. Yew, "Decoupled value prediction on trace processors," in *Proceedings of the 6th International Conference on High-Performance Computer Architecture*, 1999, pp. 231-240.
4. M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value prediction," in *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture*, 1996, pp. 226-237.
5. M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," in *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1997, pp. 248-258.
6. F. Gabbay and A. Mendelson, "Can program profiling support value prediction?" in *Proceedings of the 30th International Conference on Microarchitecture*, 1997, pp. 270-280.
7. T. Nakra, R. Gupta, and M. L. Soffa, "Global context-based value prediction," in *Proceedings of the 5th International Conference on High-Performance Computer Architecture*, 1999, pp. 4-12.
8. T. Y. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," in *Proceedings of the 19th International Conference on Computer Architecture*, 1992, pp. 124-134.
9. K. Wang and M. Franklin, "Highly accurate data value prediction using hybrid predictors," in *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, 1997, pp. 281-290.
10. B. Calder, G. Reinman, and D. M. Tullsen, "Selective value prediction," in *Proceedings of the 26th International Conference on Computer Architecture*, 1999, pp. 64-74.
11. B. Goeman, H. Vandierendonck, and K. DeBosschere, "Differential FCM: Increasing value prediction accuracy by improving table usage efficiency," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001, pp. 207-216.
12. Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proceedings of the 30th International Conference on Microarchitecture*, 1997, pp. 248-258.
13. P. J. Chuang, Y. T. Hsiao, and Y. S. Chiu, "An efficient value predictor dynamically using loop and locality properties," *Journal of Supercomputing*, Vol. 30, 2004, pp. 19-36.
14. D. Burger, T. M. Austin, and S. Bennett, "Evaluating future microprocessors: the simplescalar tool set," Technical Report No. CS-TR-96-1308, Dept. of Computer Science, University of Wisconsin-Madison, U.S.A., 1996.

Po-Jen Chuang (莊博任) received the B.S. degree from National Chiao Tung University, Taiwan, R.O.C., in 1978, the M.S. degree in Computer Science from the University of Missouri at Columbia, U.S.A., in 1988, and the Ph.D. degree in Computer Science from the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, U.S.A. (now the University of Louisiana at Lafayette), in 1992. Since 1992, he has been with the Department of Electrical Engineering, Tamkang University, Taiwan, where he is currently a Professor. He was the department chairman from 1996 to 2000. His main areas of interest include parallel and distributed processing, fault-tolerant computing, computer architecture, and mobile computing. Dr. Chuang is a member of the IEEE, the IEEE Computer Society, the ACM, and the IICM.

Yu-Shian Chiu (邱育賢) received the B.S. and M.S. degrees in Electrical Engineering in 2001 and 2003 from Tamkang University, Taiwan, where he is currently pursuing the Ph.D. degree. His research interests include parallel and distributed processing, computer architecture, and mobile computing.