Highly Accurate Data Value Prediction using Hybrid Predictors

Kai Wang
Datastream Systems, Inc.
50 Datastream Plaza
Greenville, SC 29605, USA
kaiw@dstm.com

Manoj Franklin
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC 29634-0915, USA
mfrankl@blessing.ces.clemson.edu

Abstract

Data dependences (data flow constraints) present a major hurdle to the amount of instruction-level parallelism that can be exploited from a program. Recent work has suggested that the limits imposed by data dependences can be overcome to some extent with the use of data value prediction. That is, when an instruction is fetched, its result can be predicted so that subsequent instructions that depend on the result can use this predicted value. When the correct result becomes available, all instructions that are data dependent on that prediction can be validated. This paper investigates a variety of techniques to carry out highly accurate data value predictions. The first technique investigates the potential of monitoring the strides by which the results produced by different instances of an instruction change. The second technique investigates the potential of pattern-based two-level prediction schemes. Simulation results of these two schemes show improvements over the existing method of predicting the last outcome. In particular, some benchmarks show improvement with the stride-based predictor and others show improvement with the pattern-based predictor. To do uniformly well across benchmarks, we combine these two predictors to form a hybrid predictor. Simulation analysis of the hybrid predictor shows its overall prediction accuracy to be better than that of the component predictors across all benchmarks.

1 Introduction

Instruction-level parallel (ILP) processing—executing multiple instructions in parallel in a uniprocessor—is a subject of great interest among the computer architecture community now. Virtually every processor released in the last 2-3 years and future processors announced by major processor manufacturers exploit ILP in a major way using a variety of techniques.

A major hurdle to ILP processing is the presence of data dependences, which preclude the execution of instructions in parallel. If an instruction is data dependent on a preceding instruction, then it can be executed only after the preceding instruction's result becomes available. Recent studies [8] [9] have shown that it is possible to overcome the hurdles imposed by data dependences with the use of *data value prediction*. That is, the result of an instruction is predicted based on the past behavior of previous instances of the instruction (i.e., previous dynamic instructions with the same PC value), and passed on to subsequent instructions that depend on the result. Later, when the actual operands of the instruction becomes available, the instruction is executed, and the correct result is compared with the value predicted earlier. If the values match, then all of the instructions in the active window that depended on the predicted value are informed of the match. If the values do not match, then the correct result is forwarded to the instructions that require this value, and those instructions are re-executed. Simulation results presented in [9] show value prediction accuracies of about 49%.

Another recently proposed technique that is similar in spirit to data value prediction, but does not involve any prediction, is *dynamic instruction reuse* [14]. In this technique, the operands and result(s) of recently executed instructions are stored in a buffer. When the operands of an instruction become available, they are compared with the operands of the previous instance of the same instruction. If the operands match, then the execution part of the instruction is skipped, and the result is read directly from the buffer. Thus, this technique is useful for shortening the latency of long-latency operations. Experimental results reported in [14] show that the execution part of about 33% of the dynamic instructions can be skipped by using a 1024-entry fully-associative buffer.

In this paper, we investigate various techniques to carry out highly accurate data value predictions.

The schemes that we consider are last outcome-based prediction, stride-based prediction, 2-level prediction scheme, and various hybrids. Simulation results show that hybrid schemes work well across all benchmarks.

Section 2 presents background material on data value prediction. Section 3 describes stride-based value prediction and two-level value prediction. Section 4 presents simulation results obtained using a simulator that we developed. Section 5 presents ways of combining these two schemes, and also gives simulations results for these hybrid schemes. Section 6 presents a summary of this work, and the conclusions of the paper.

2 Background

2.1 Dependences and Their Effect on Parallelism

Dependences between instructions impose a big hurdle to the parallel execution of instructions. Dependences come in 3 kinds—name dependences, control dependences, and data dependences. Name dependences occur due to the reuse of storage locations such as registers and memory locations, and can be easily removed by static or dynamic renaming techniques.

Control dependences occur due to conditional branch instructions that can cause a potential change in control flow based on the outcome of the branch. Three techniques are available to overcome the effects of control dependences: (i) predicated execution, (ii) execute control-independent portions of code in parallel, and (iii) speculate the outcome of control-changing instructions. The first technique converts control dependences into data dependences as follows: the branch condition outcome is evaluated and placed in a predicate register, and the instructions that were control-dependent on the branch are made conditional on the value of the predicate register. The second technique identifies control independent portions of code and executes them in parallel, possibly by means of separate hardware sequencers. Parallelism limit studies in [7] show that executing control independent code alone is not sufficient to overcome the barriers of control dependences. Speculative execution, by means of predicting branch outcomes, should also be done to get reasonable amounts of parallelism.

Like control dependences, *data dependences* also impose a hurdle to the parallel execution of instructions, because an instruction cannot be executed until the results of the instructions that it is data dependent upon are available. As with control dependences, three similar techniques are available to overcome the effects of data dependences: (i) dependence collapsing, (ii) execute data-independent instructions in parallel, and (iii) speculate the result of data-producing instructions. The first technique combines data-dependent instructions into a single instruction with the use of fused functional units [10]. The second technique has been widely used by ILP compilers [6] and dynamically scheduled processors [13]. ILP compilers identify mutually data-independent instructions, and place them side by side to facilitate their parallel execution by the hardware. Dynamically scheduled processors buffer a large window of instructions, and identify ready instructions (i.e., instructions that have their source operands available) every cycle so as to execute them in parallel. Again, parallelism limit studies [1] [2] [7] have shown that with realistic size instruction windows, the amount of parallelism exploitable with scheduling alone is limited. The third technique—data value prediction—attempts to further overcome the hurdles imposed by data dependences, and is the subject matter of this paper.

2.2 Data Value Prediction

The essence of data value prediction is to predict the result of data-producing instructions based on their past behavior, just like predicting the outcome of conditional branches. A good heuristic to use is to record the recent results produced by previous instances of an instruction, and predict the result of the instruction's next instance based on past results. The simplest such scheme, which we call the *last outcome* scheme, would be to store the result produced when the instruction was executed for the last time, and predict the same value when the instruction is encountered in the future. The block diagram of such a scheme is shown in Figure 1. The main part of the predictor is a Value History Table (VHT) that stores the last result produced by the instructions that are currently mapped to it. Each VHT entry has two fields—`Tag` and `Value`. The `Tag` field stores the identity of the instruction that is currently mapped to that entry, and the `Value` field stores the last result for that instruction.

Lipasti *et al* have measured the prediction accuracy obtainable with this approach, and found it to be averaging about 49% for the PowerPC architecture for a sample of benchmarks taken mostly from SPEC '92 and SPEC '95 [9]. That is, the results produced by about 49% of the register result-producing instructions can be accurately predicted by storing the last

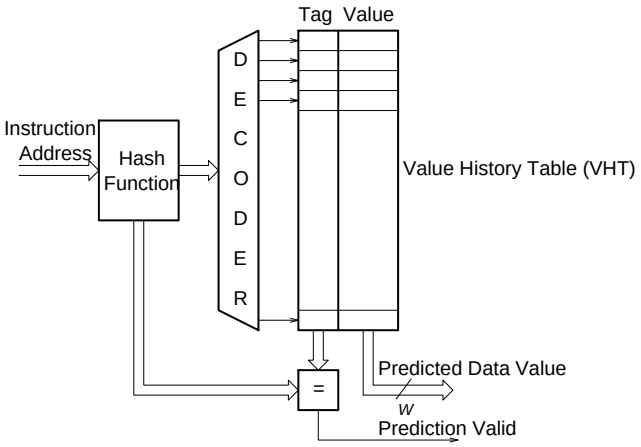

Figure 1: Block Diagram illustrating Last Outcome-based Value Predictor

result of each register result-producing instruction. Lipasti *et al* also measured the theoretical upper limit on the prediction accuracy obtainable by storing the last 4 results of each static instruction, and assuming that the predictor is able to perfectly choose the required result from these 4 values. This hypothetical scheme provided an average prediction accuracy of 61%.

3 Schemes for Accurate Data Value Prediction

Previous studies of realistic data value prediction schemes [8] [9] considered storing the result of the last instance of each register result-producing instruction, and selecting one out of those results as the next prediction. These studies obtained prediction accuracies of only about 49%, with a large number of mispredictions. It is important to decrease the number of mispredictions, because each misprediction causes some instruction re-executions, which can lead to increased structural hazards and increased execution time [9]. In this section, we describe techniques that substantially improve the accuracy of data value prediction.

It is important to note that a plethora of techniques have been proposed for carrying out highly accurate branch predictions. These techniques include 2-level predictors, correlation-based predictors, and hybrid predictors [11] [12] [15]. However, it is not possible to directly apply these techniques to data value prediction, because branch prediction involves merely a binary decision (i.e., an 1-out-of-2 prediction), whereas data value prediction involves a multi-value decision (i.e., an 1-out-of-2^W prediction, where W is the word size of the computer). Nevertheless, it is worthwhile to take cues from the vast body of research on branch prediction.

3.1 Data Value Locality

Our initial objective is to study the amount of past history that needs to be considered in making data value predictions. Too little of history may result in poor prediction accuracy. Too much of history, on the other hand, may result in high hardware overheads, high time overhead for accessing the history, diminishing returns in terms of prediction accuracy, and sometimes even poorer prediction accuracies! As a first step, we stored the most recent 16 value history of each instruction in a separate 16-value buffer, and measured how often the result of the next instruction is present in its buffer. Table 1 gives the percentage of register result-producing instructions that have their result in their 16-depth history buffers. These values are for the MIPS R2000 architecture and for the SPEC '92 integer benchmark suite. From the table we can see that, except for compress and eqntott, more than 70% of the register result-producing instructions have their results available in their 16-depth history buffer. Thus, there is good repeatability of register results.

Program	*Percentage of VP-Eligible Dynamic Instructions whose Results are available in their 16-Depth History Buffers*
compress	39.47%
eqntott	56.36%
espresso	75.26%
gcc (cc1)	74.89%
sc	79.33%
xlisp	72.16%

Table 1: Register Value Locality for a History Depth of 16

Storing 16 values per static instruction, and selecting one among the 16 values is still a cumbersome task for the hardware. Perhaps many of the values present in a 16-depth history buffer are even identical! If the duplicate values are eliminated, then the number of choices the predictor has to deal with is reduced. We shall next see how many of the 16 values are unique. Figure 2 presents a graph showing the cumulative distribution of dynamic instructions based on the number of unique values among their 16-depth history buffers. The X-axis denotes the number of unique values in the 16-depth history buffers, and the Y-axis denotes the

percentage of register result-producing instructions. A particular data point (X_1, Y_1) on the graph indicates that Y_1% of register result-producing instructions have X_1 or less number of unique value in their 16-depth history buffers. The data points for the graph were obtained as follows. A history buffer of 16 entries is provided for each static instruction. Every time a register result is produced by an instruction, the number of unique data values in its history buffer is counted. If the count is C, then the data sets for X $\geq C$ are incremented by one instruction each.

From the data points for X = 1, we can see that about 15-45% of the instructions have all of their previous 16 results to be the same. That is, there is little change in the values produced by different instances of those instructions. Therefore, it would be worthwhile to exploit this behavior; the *last outcome* scheme that we saw in section 2.2 attempts to do precisely this. From the graph, we can also see that about 28-67% of the instructions have 4 or fewer unique values in their 16-depth history buffer. That is, for a large number of instructions, the results are constantly changing, but are still circulating within 4 or fewer values. The results produced by these instructions cannot be accurately predicted by the *last outcome* scheme. We shall next investigate schemes to accurately predict the results of these instructions.

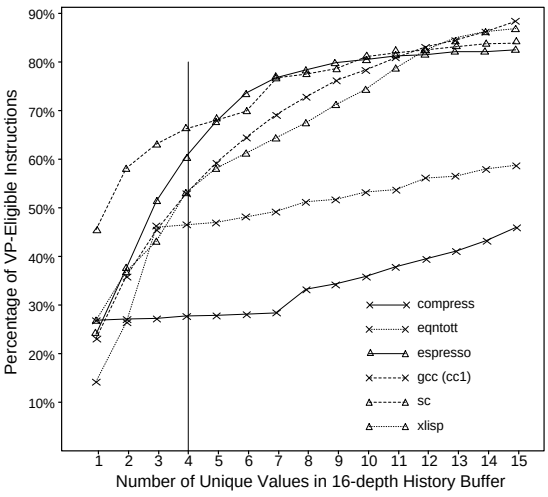

Figure 2: Cumulative Distribution of VP-Eligible Instructions Based on Number of Unique Values in their 16-Depth History Buffers

3.2 Stride-based Value Prediction

One way to capture data value locality is by monitoring the stride by which the results of consecutive instances of an instruction change. If the results vary by a constant stride, then it is easy to predict the results of future instances of that instruction. The concept of stride-based speculation is, by no means, new. It works well because of loop induction variables and programs stepping through arrays in a regular fashion, and has been successfully employed for generating addresses for prefetching data into caches [3] [4].

Figure 3(a) gives a block diagram of a simple stride-based value predictor. Its VHT entry has 4 fields—`Tag`, `State`, `Value`, and `Stride`. The state can have one of 3 values—Init, Transient, and Steady. The state transition diagram is given in Figure 3(b). The basic step in a stride-based predictor is the stride detection phase, which aims at detecting a stride sequence. The first time an instruction is encountered (as evident from a miss in the VHT), no prediction is made. When the instruction produces its result, an entry is allocated in the VHT, and the following actions take place: (i) the result is stored in the `Value` field of that entry, and (ii) the `State` of that entry is

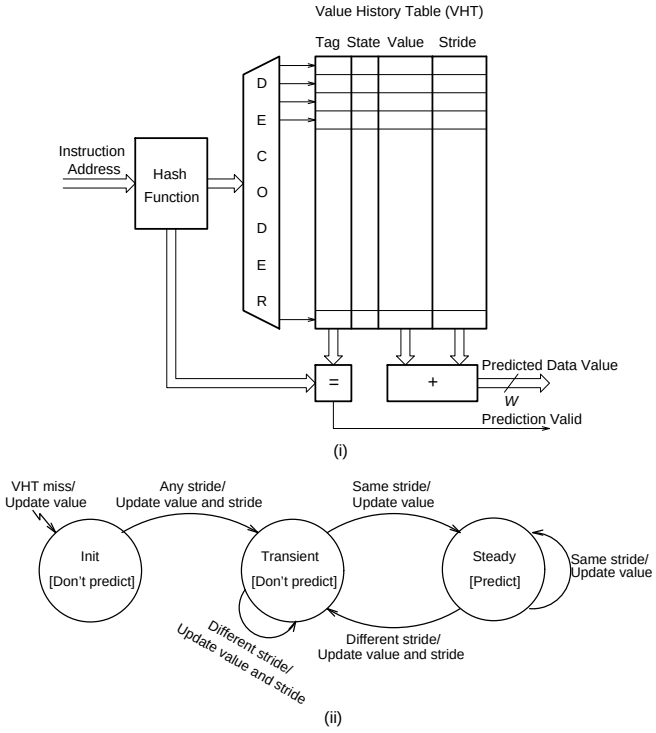

Figure 3: Block Diagram and State Transition Diagram for a Simple Stride-based Value Predictor

set to *Init*. While in the *Init* state, if another instance of the same instruction is encountered, no prediction is made. However, when that instance produces a result ($D1$), that is potentially the beginning of a stride sequence, and the following actions take place: (i) the stride is calculated as $S1 = D1 - Value$ in VHT entry, (ii) $D1$ and $S1$ are entered in the `Value` and `Stride` fields of the VHT entry, and (iii) the `State` is set to *Transient*. While in the *Transient* state, if another instance of the same instruction is encountered, no prediction is made. When that instance produces a result ($D2$), the following actions take place: (i) the stride is calculated as $S2 = D2 - Value$ in VHT entry, (ii) $D2$ is entered in the `Value` field of the VHT entry, and (iii) if $S2$ is same as previous stride, the `State` is set to *Steady*, else $S2$ is entered in the `Stride` field. While in the *Steady* state, predictions are made by adding together the `Value` and `Stride` fields; if a different stride appears, then the `State` is set to *Transient*. This simple 3-state scheme can detect most strides.

3.3 Two-Level Value Prediction

Another scheme to capture the recurrence of a behavior pattern among instruction results is to use an elaborate two-level prediction scheme. Although two-level prediction schemes have provided high accuracy for branch prediction [5] [15], incorporating the 2-level prediction concept into data value prediction is not as straightforward as incorporating it into branch prediction. The primary difficulty is that the result of an instruction can take any one of 2^W values, where W is the width of a register. How do we aspire to capture behavior patterns among 2^W values, for reasonable values of W such as 32 and 64? To solve this problem, we took into consideration our earlier observation from Figure 1 that a substantial percentage of the dynamic instructions have 4 or fewer unique values in their most recent history. By storing a maximum of 4 most recent unique values for each instruction, and by doing a binary encoding of these 4 outcomes, we can capture behavior patterns using a 2-level predictor that performs 1-out-of-4 predictions.

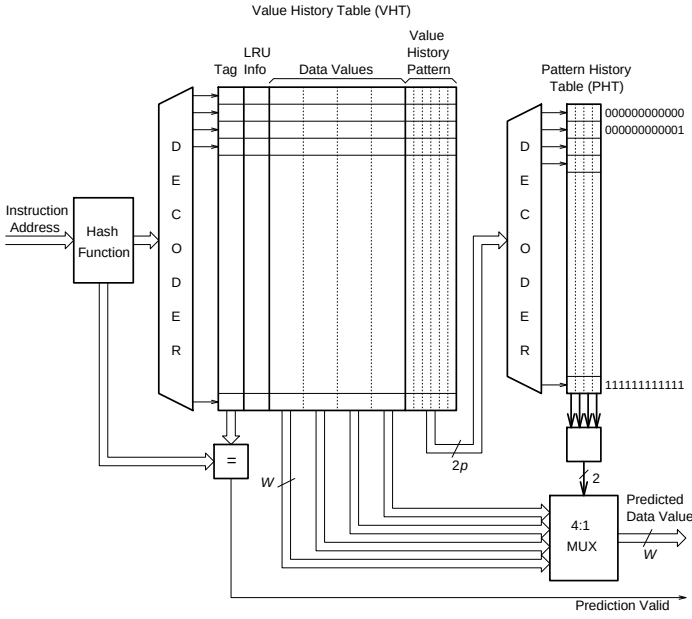

Figure 4: Block Diagram of a 2-Level Value Predictor

Figure 4 shows a block diagram of a 2-level value predictor that stores up to 4 unique values for each static instruction. The VHT has 4 fields—`Tag`, `LRU Info`, `Data Values`, and `Value History Pattern`. The `Data Values` field stores up to 4 most recent unique values. The 4 values are associated with the binary encoding {00, 01, 10, 11}. So long as the different instances of an instruction keep producing one of these 4 values, the result can be predicted by selecting one of the 4 outcomes from {00, 01, 10, 11}, and taking the value currently associated with that outcome. When a fifth unique value is produced, it replaces from the `Data Values` field the least recently seen value. The `LRU Info` field keeps track of the order in which the 4 data values were last seen. The `Value History Pattern` field stores as a $2p$-bit pattern the last p outcomes of an instruction. Because there are 4 possible outcomes for an instruction, 2 bits are required to store each outcome. The second level of the predictor is the Pattern History Table (PHT). For each possible $2p$-bit pattern, a condensed history of the previous outcomes of the pattern is recorded in a PHT entry by means of 4 independent up/down counter values $\{C_0, C_1, C_2, C_3\}$, as in [5].

The 2-level predictor works as follows. When a prediction is to be made for an instruction, the appropriate VHT entry is selected, and its `Tag` field checked to determine if the entry corresponds to that instruction. If so, its `Value History Pattern` value is used to select the appropriate PHT entry. The PHT entry contains 4 count values, from which the maximum value is determined. (If there is a tie, one of the values can be selected at random, or based on last outcome.) If the maximum value is greater than or equal to a specific threshold value, then the outcome corresponding to that count value is selected as the next prediction. If the maximum count value is less than the threshold, then no prediction is made.

The 2-level predictor is updated as follows. The Value History Pattern of the selected VHT entry is shifted left by 2 bits, and the new outcome is entered in the bits left vacant by the shift. The selected PHT entry's counter values are updated in the following manner. The count value corresponding to the correct outcome is incremented by 3 (or less, if the counter saturates), and all other counter values are decremented by 1 (if they are non-zero).

4 Performance Evaluation

We have seen three schemes to carry out accurate data value predictions. This section presents results obtained from a simulation study of these schemes.

4.1 Experimental Setup

The simulation studies are conducted using the MIPS instruction set architecture (ISA), a representative of the class of streamlined ISAs that have emerged recently. The simulator accepts executable images of MIPS programs, and simulates the functional part of their execution.

4.1.1 Benchmarks and Performance Metrics

For benchmarks, we use the SPEC '92 integer suite. All programs were compiled using the MIPS C compiler with the optimization flags distributed in the SPEC benchmark makefiles. To get good insight on the value predictor's performance, we use 3 metrics: (i) *Percentage of instructions correctly predicted*, (ii) *Percentage of instructions mispredicted*, and (iii) *Percentage of instructions not predicted*. All of the metrics are expressed as percentage of the total number of register result-producing instructions.

4.1.2 Default Parameters for the Study

- Number of instructions simulated: The benchmarks were simulated up to 100 million instructions or up to completion, depending on whichever occurred earlier.
- *Value History Table:* For all schemes, the VHT has 4K entries, and is direct-mapped.
- *2-level predictor parameters:* The default pattern size is 6, and the number of values stored per VHT entry is 4. Thus, the PHT has 4K entries. The PHT counters saturate at 12. Three different values are used for the PHT counter threshold—3, 6, and 9.

Data value prediction is carried out only for those instructions that produce a single register result. Thus branch instructions, store instructions, nops, and double-precision instructions are not considered for data value prediction. Table 2 gives the percentage of dynamic instructions that are eligible for data value prediction for each benchmark.

Program	*Percentage of Instructions eligible for Value Prediction*
compress	64.05%
eqntott	60.82%
espresso	69.06%
gcc (cc1)	57.22%
sc	54.07%
xlisp	45.66%

Table 2: Benchmarks and Percentage of Instructions Eligible for Value Prediction

4.2 Experimental Results

Figure 5 presents the results that we obtained in our simulation experiments. The X-axis denotes the benchmarks, and the Y-axis denotes the percentage of VP-eligible instructions. For each benchmark, 6 bar charts are given; these correspond respectively to the (i) last outcome scheme, (ii) last outcome scheme with confidence indicator, (iii) stride scheme, (iv) 2-level scheme (with PHT counter threshold value 3), (v) 2-level scheme (with PHT counter threshold value 6), and (vi) 2-level scheme (with PHT counter threshold value 9). Each bar consists of 3 or fewer parts. The bottom-most part indicates the percentage of VP-eligible instructions that are correctly predicted by a scheme. The next part indicates the percentage that is mispredicted, and the next part indicates the percentage that was not predicted by the scheme.

Let us go through the results of Figure 5 in some detail. A comparison of the first and second bars of each benchmark shows that when value prediction is performed by monitoring the stride, the percentage of mispredictions has dropped substantially, with a corresponding increase in the percentage of instructions that are not predicted. The net result is that the prediction accuracy has increased substantially for the predicted instructions. The last 3 sets of bars, for the 2-level prediction schemes, indicate that these schemes also provide good improvement in prediction accuracy for the predicted instructions. When the threshold is

varied from 3 to 9, there is a substantial drop in mispredictions, with a slight drop in number of correct predictions. Thus, the stride-based value predictor and 2-level value predictors provide reasonably high percentage of correct predictions (about 50% of register result-producing instructions), with very small percentage of mispredictions (about 5% of register result-producing instructions).

5 Hybrid Predictors

The results of Figure 5 show that no single scheme can get high prediction accuracies for every benchmark. For eqntott and espresso, compared to the basic *last outcome* scheme, the stride-based prediction scheme does not give much improvement, whereas the 2-level predictor gives good improvement. On the other hand, for compress and sc, the stride-based predictor gives good improvement, while the 2-level predictor fails to give improvement. (The reason why 2-level predictor performs poorly for compress can be deduced from Figure 2, which shows that compress has much more than 4 unique values in its recent history.) Because different benchmarks have different data value locality characteristics that can be exploited only by different schemes, it is better to use hybrid predictors to get good prediction accuracy over a set of benchmarks.

We investigate two hybrid predictors in this paper. The first one is a combination of last outcome-based prediction and stride-based prediction. The second one is a combination of 2-level prediction and stride-based prediction.

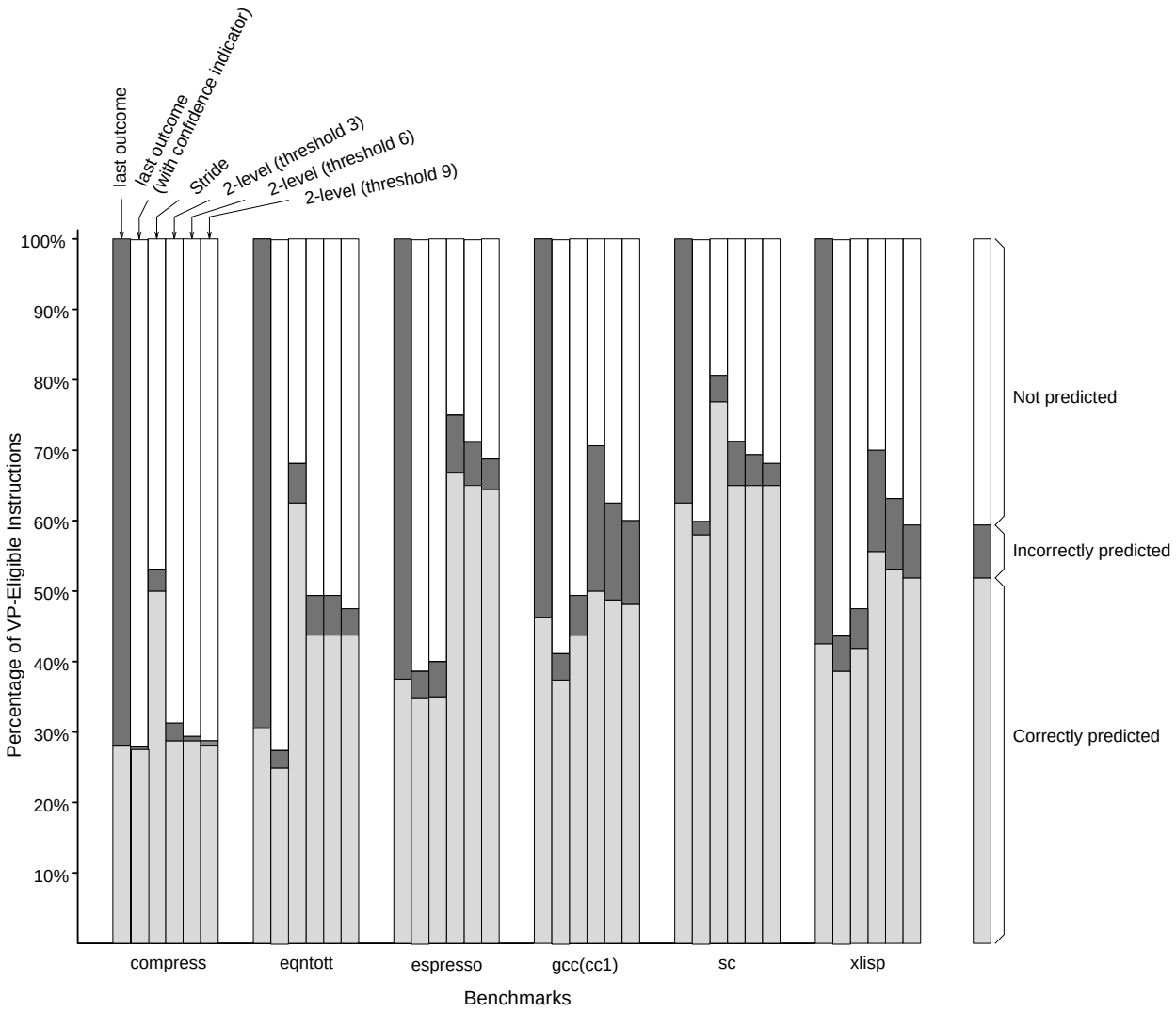

Figure 5: Simulation Results for Last Outcome, Stride-Based, and 2-Level Predictors

5.1 Hybrid of Stride-Based and Last Outcome Predictors

In the stride-based predictor discussed in Section 3, the predictor does not make any prediction if the selected VHT entry is in state *Init* or *Transit*, because a stride has not yet been registered. Our first hybrid predictor is a modification of this stride-based predictor. This hybrid predictor makes predictions even if the VHT entry is in the *Init* state or *Transit* state. In the *Transit* state, it predicts the last outcome, and in the *Init* state, it predicts a value of zero.

5.2 Hybrid of 2-Level and Stride-Based Predictors

The second hybrid predictor that we investigate combines a 2-level predictor and a stride-based predictor. Figure 6 shows the block diagram of this hybrid predictor. Compared to the VHT of the 2-level predictor, this hybrid predictor's VHT entry has two additional fields—`State` and `Stride`. This hybrid predictor works as follows. When a prediction is to be made for an instruction, the appropriate VHT entry is selected, and its `Tag` field checked as before. In parallel, the `Value History Pattern` and the `State` fields are read out for the 2-level predictor and the stride-based predictor. The 2-level predictor makes a prediction if the maximum count value in the selected PHT entry is greater than the specified threshold value. If the 2-level predictor makes a prediction, then that value is selected as the hybrid predictor's prediction. If the 2-level predictor does not make a prediction, then the value predicted (if any) by the stride-based predictor is selected.

5.3 Experimental Results

We conducted simulation studies with the two hybrid predictors. For forming the hybrid predictor, we used the 2-level predictor with PHT counter threshold value 6. Figure 7 shows the simulation results obtained for the two hybrid predictors. For ease of comparison, the results for the last outcome predictor, the stride-based predictor, and the 2-level predictor (with threshold 6) are reproduced from Figure 5.

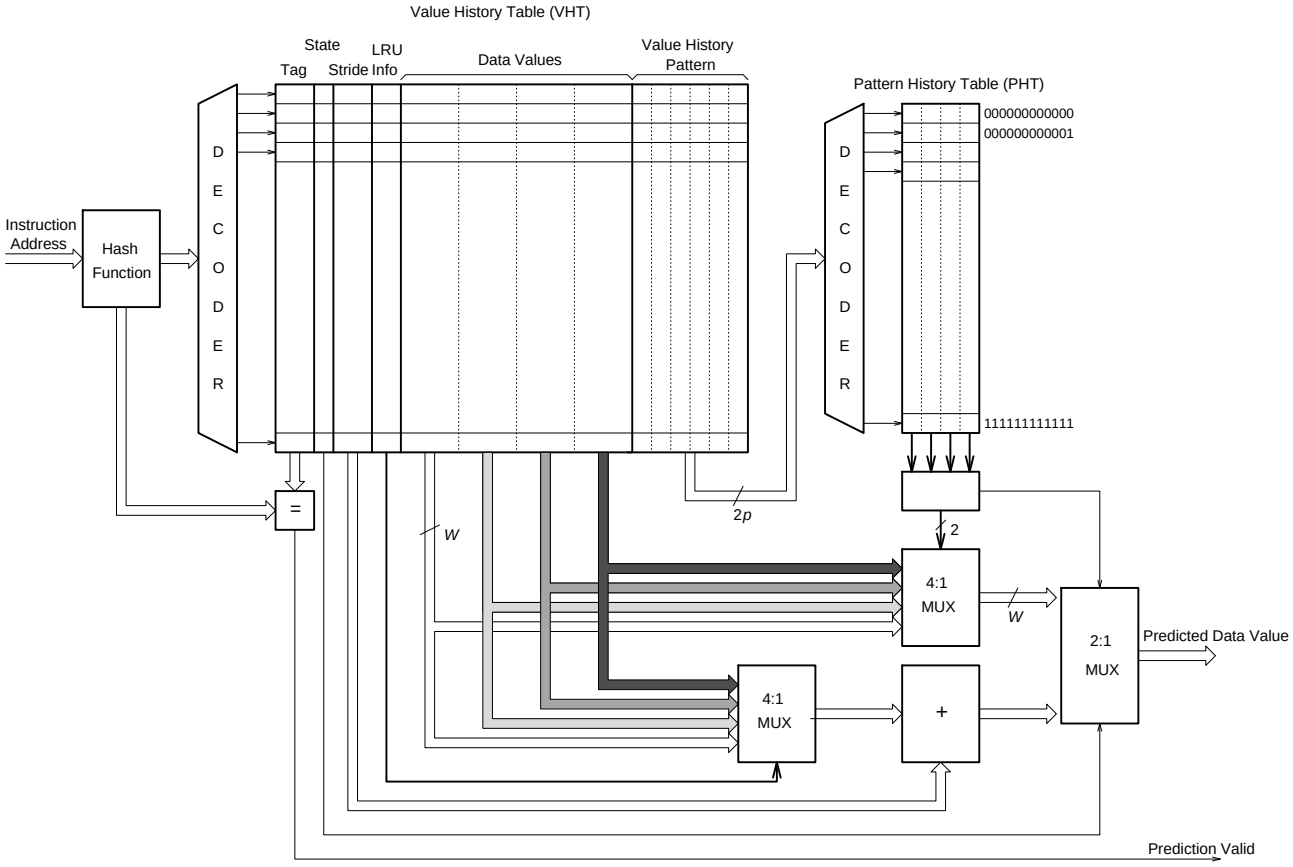

Figure 6: Block Diagram of Hybrid (2-Level, Stride) Predictor

From Figure 7, we can see that with the first hybrid predictor, the percentage of correct predictions is higher than that of both the last outcome predictor and the stride-based predictor. The percentage increase over that of stride-based predictor varies from about 0.5% (for compress) to about 12% (for espresso). But, there is a performance price to pay here. Because this hybrid predictor predicts the results for all register result-producing instructions, the percentage of mispredictions is very high. Therefore, this predictor can be useful only in those microarchitectures that permit negligible penalty for value mispredictions.

From Figure 7, we can also see that with the second hybrid predictor, the percentage of correct predictions has increased substantially. For eqntott and sc, this percentage is now more than 80%. That is, the results of more than 80% of the register result-producing instructions in these two benchmarks can be correctly predicted with this hybrid predictor. For the remaining benchmarks, the percentage of correct predictions varies from about 50% (for compress) to about 72% (for espresso). The percentage of mispredictions with this hybrid predictor ranges from about 5% (for compress) to about 18% (for xlisp).

6 Summary and Conclusions

We have investigated techniques to carry out highly accurate data value prediction in ILP processors. The central idea behind these techniques is to monitor the behavior pattern of each instruction result in order to do better predictions for the future. The stride-based predictor monitors the stride by which the result values produced by successive instances of an instruction vary, and then uses this information to predict the result of future instances of that instruction. The 2-level predictor stores the last 4 unique results produced by different instances of an instruction, and uses a 2-levels of history to predict the result of future instances. The first level table stores the history of the previous results as a pattern, and the second level stores the history of each pattern.

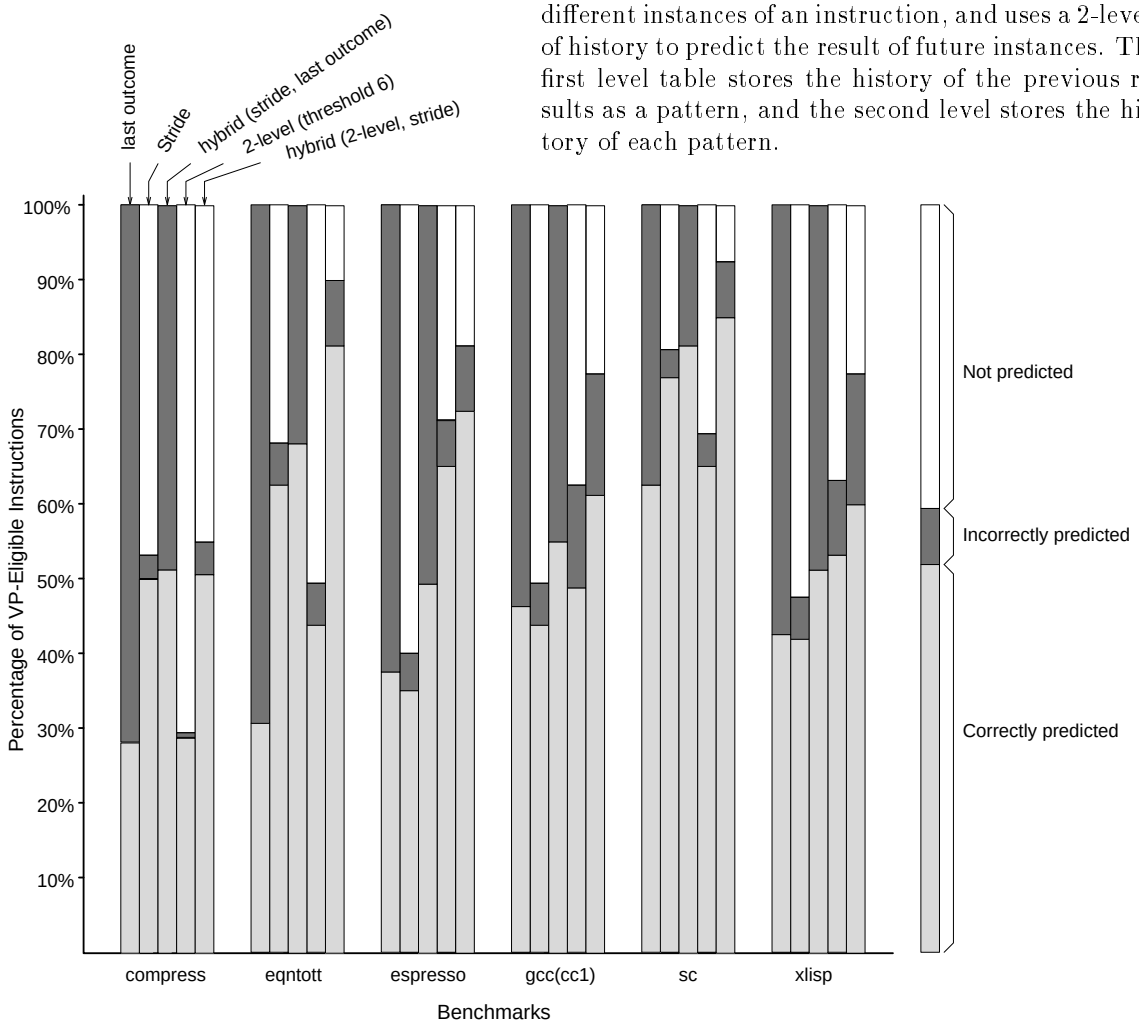

Figure 7: Simulation Results for Hybrid Predictors

We also presented simulation results for these two schemes. These results show that about 50% of the register result-producing instructions can be correctly predicted, with mispredictions averaging about 5%, which is a marked improvement over the previously proposed scheme of using the last outcome as the basis of prediction. Finally, we investigated two hybrid predictors each of which is a combination of two of the above three predictors. One of the hybrid predictors was able to correctly predict the results of about 50-82% of the register result-producing instructions. The percentage of mispredictions with this hybrid predictor ranged from about 5-18%. These results are very promising because fewer mispredictions translates directly into fewer cycles wasted due to (i) instruction re-execution and (ii) structural hazards.

Acknowledgements

This work was supported by the US National Science Foundation (NSF) Research Initiation Award, CCR 9410706. We thank the reviewers for their helpful comments, which have helped improve the quality of the paper.

References

[1] T. M. Austin and G. S. Sohi, "Dynamic Dependency Analysis of Ordinary Programs," *Proceedings of 19th Annual International Symposium on Computer Architecture*, pp. 342-351, 1992.

[2] M. Butler, T. Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebanow, "Single Instruction Stream Parallelism Is Greater than Two," *Proceedings of 18th Annual International Symposium on Computer Architecture*, pp. 276-286, 1991.

[3] T. Chen and J-L. Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors," *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 609-623, May 1995.

[4] F. Dahlgren and P. Stenstrom, "Evaluation of Hardware-Based Stride and Sequential Prefetching in Shared-Memory Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 4, pp. 385-398, April 1996.

[5] S. Dutta and M. Franklin, "Control Flow Prediction with Tree-like Subgraphs for Superscalar Processors," *Proc. 28th International Symposium on Microarchitecture (MICRO-28)*, pp. 258-263, 1995.

[6] W. W. Hwu *et al*, "Compiling for ILP Processors," *Proceedings of the IEEE*, Vol. 83, No. 12, December 1995.

[7] M. S. Lam and R. P. Wilson, "Limits of Control Flow on Parallelism," *Proceedings of 19th Annual International Symposium on Computer Architecture*, pp. 46-57, 1992.

[8] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value Locality and Load Value Prediction," *Proceedings of VIIth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, 1996.

[9] M. H. Lipasti and J. P. Shen, "Exceeding the Dataflow Limit via Value Prediction," *Proceedings of 29th International Symposium on Microarchitecture (MICRO-29)*, pp. 226-237 1996.

[10] R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC Ssytem/6000 Floating-Point Execution Unit," *IBM Journal of Research and Development*, Vol. 34, No. 1, pp. 59-70, January 1990.

[11] R. Nair, "Dynamic Path-Based Branch Correlation," *Proc. 28th Annual International Symposium on Microarchitecture (MICRO-28)*, 1995.

[12] S-T. Pan, K. So, and J. T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pp. 76-84, 1992.

[13] J. E. Smith and G. S. Sohi, "The Microarchitecture of Superscalar Processors," *Proceedings of the IEEE*, Vol. 83, No. 12, pp. 1609-1624, December 1995.

[14] A. Sodani and G. S. Sohi, "Dynamic Instruction Reuse," *Proceedings of 24th Annual International Symposium on Computer Architecture*, 1997.

[15] T-Y Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 124-134, 1992.