Global Context-Based Value Prediction[1]

Tarun Nakra, Rajiv Gupta and Mary Lou Soffa
Department of Computer Science
University of Pittsburgh
{nakra,gupta,soffa}@cs.pitt.edu

Abstract

Various methods for value prediction have been proposed to overcome the limits imposed by data dependencies within programs. Using a value prediction scheme, an instruction's computed value is predicted during the fetch stage and forwarded to all dependent instructions to speed up execution. Value prediction schemes have been based on a **local context** *by predicting values using the values generated by the* **same** *instruction. This paper presents techniques that predict values of an instruction based on a* **global context** *where the behavior of* **other** *instructions is used in prediction. The global context includes the path along which an instruction is executed and the values computed by other previously completed instructions. We present techniques that augment conventional last value and stride predictors with global context information. Experiments performed using path-based techniques with realistic table sizes resulted in an increase in prediction of 6.4-8.4% over the current prediction schemes. Prediction using values computed by other instructions resulted in a further improvement of 7.2% prediction accuracy over the best path-based predictor.*

1 Introduction

The efficiency of modern architectures is highly dependent on the amount of parallelism extracted both by the compiler and the hardware. The barriers to parallelism are the dependencies that exist between instructions of a program. These dependencies are classified as 1) *control dependencies*, arising from conditional branches in a program, and 2) *data dependencies*, arising from the flow of data between instructions.

To overcome the limitations imposed by control dependencies, branch outcomes are predicted and instructions are executed speculatively. Current branch predictors have been shown to accurately predict more than 98% of the branches [9].

Data dependencies existing between the instructions are more prevalent than control dependencies. To overcome these dependencies, it is possible to perform speculation on data as well. The data speculation could be performed based on the storage addresses [5, 10] or data values [7]. Recent studies on predicting data values have shown that the benefits of value prediction have significant potential for performance improvement [7, 12, 13]. Moreover, studies performed to observe the patterns of values computed by an instruction have found that instructions demonstrate locality of data values [8]. The sequences that occur in certain patterns are easier to predict than others. Typically, the patterns of values include:

(1) Constant values - these are the easiest to predict by always predicting the same constant value. Constant values have been shown to occur often in benchmark programs [8].

(2) Values differing by a stride - these usually occur in case of loop induction variables or variables that are dependent on induction variables.

(3) Other sequences - these include sequence of values that differ by non-constant strides. This category is the hardest to predict.

The techniques that perform prediction based on a local context of an instruction include **last value prediction** in which the value predicted is the one that was computed in the last execution of that instruction. Previous work using last value prediction indicates successful prediction for 40-50% of the dynamic instruction instances [7]. Another technique is **stride prediction**, which predicts values to be the sum of the last computed value of the same instruction and a stride. Stride prediction has been compared with last value prediction and shown to improve the prediction performance significantly [4]. In other work, **finite context method** (FCM) predictors are proposed that

[1]Partially supported by National Science Foundation Grant CCR-9808590 and a grant from Hewlett Packard to the University of Pittsburgh

predict values based on previously observed patterns of values [12]. The recent history of values computed by an instruction is matched with previous value history of the instruction. The next value of the instruction is predicted to be the next value in the previous pattern. These predictors have been shown to improve the accuracy of last value and stride predictors by as much as 20% . However, this is an upper bound since the study uses unbounded history table sizes. Also, **hybrid predictors** [14] have been proposed that attempt to detect sequences that are composition of stride and non-stride sequences. Due to their complexity, the FCM predictor and the proposed hybrid predictors do not have efficient implementations.

This paper proposes techniques that associate a more global context with an instruction to help predict its value. This global context association is analogous to branch prediction performed by correlating with other branches [15] and has not been considered for value prediction before. Two types of global contexts are considered.

- **Path information (branch history):** Different values for an instruction are predicted along different paths within a program. Branch history is used to separate the different paths. In the work by Lipasti[6], branch history is used to either select instructions for prediction, or predict input operands for estimating data dependencies. However, branch history is not involved in the actual prediction of computed values.

- **Values produced by recently completed instructions:** Values are predicted by using the correlation between values produced by instructions close to each other within the dynamic instruction stream.

Consider the code sequence shown in Figure 1. Instructions 8, 9 and 10 in block 5 compute three address values from previously defined base and offset values. $Addr_a$ depends on a base value that has either the value Y or X, depending on whether the condition $Condition_1$ evaluates to true or false, while the offset value is fixed at 10 for all loop iterations. $Addr_b$ uses a base value that is fixed for all loop iterations and an offset that is incremented by either 1 or 2, depending on the outcome of $Condition_1$. Finally, $Addr_c$ depends on base and offset values, both of which are different for the two paths within the loop.

Let us now consider the prediction of these values, assuming the *true, false, false* sequence of outcomes for $Condition_1$ repeats itself. The conventional last value and stride predictors would not work in this case. It is possible to perform prediction using the FCM or the hybrid predictor by maintaining the patterns of values. However, new patterns of values would be generated often and the number of patterns generated would be large. Thus, frequent mispredictions can be expected.

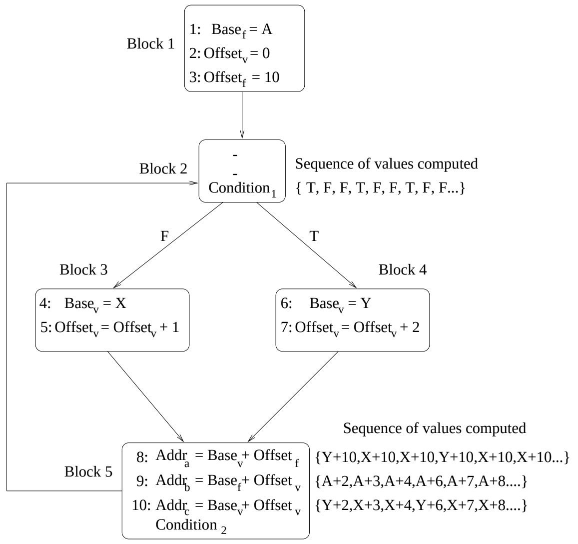

Figure 1. Example

It is possible to predict these values using control flow information. Consider the sequence of values computed by instruction 8 as shown in the figure. We can predict the next value by looking at the most recent outcome of $Condition_1$. If $Condition_1$ evaluates to false/true, the next value will be the last value of instruction 8 along the false/true branch (i.e., *Y+10/X+10*).

Consider the sequence of values computed by instruction 9. We notice that this sequence is similar to a stride sequence. In case the true branch is taken, the stride is 2; otherwise it is 1. Thus, the next value can be predicted from the stride pattern of the offset occurring along a path.

Finally, the computed value of instruction 10 depends on base and offset values, both of which depend on the control path taken from block 2 to block 5. In this case, it is possible to perform prediction by remembering both the last value and the stride values for each incoming path.

In all the above cases, we can perform the prediction for any arbitrary sequence of branch outcomes. Values of an instruction along different paths are stored separately and branch history is used to predict the value on the current path. In this example, a branch history composed of the last two branches is sufficient for correct prediction. In general, outcomes from several branches may be required.

Let us modify basic block 5 in the example as shown

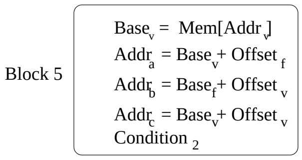

Figure 2. Modified block 5 of example

in Figure 2. The assignment to base value $Base_v$ is moved from blocks 3 and 4 to block 5. This value is now loaded from a memory location whose address, $Addr_v$, may be different for different iterations of the loop. In this case, the value of $Addr_a$ may not be predictable using path information since a different value can be produced for separate iterations of the loop along the same path. However, this value can be predicted as the sum of the loaded value of $Base_v$ and $Offset_f$. This implies that we can predict the value of $Addr_a$ by correlating it with the value loaded by a previous instruction. Essentially, we need to capture the dependence occurring between the instruction computing $Base_v$ and the one using this value. Notice that the values that were predicted using path information could also be predicted using value correlation. For example, values of $Addr_b$ and $Addr_c$ can be predicted by correlating them with the value of instruction computing $Offset_v$.

In this paper, we present prediction techniques that combine the use of branch history with last value and stride predictors. Also, a technique that predicts values based on the correlations between instructions is developed. The predictors resulting from these techniques were experimentally evaluated and compared with conventional predictors.

The rest of the paper is organized as follows. Sections 2 and 3 present the new prediction schemes based on path information. Section 4 presents a prediction scheme that uses correlation between instructions. Each prediction scheme is evaluated by doing a performance comparison with the conventional prediction schemes. The conclusions are presented in Section 5.

2 Path-based Last Value Prediction

Our first scheme, Path-based Last Value (PLV) Prediction, extends the last value predictor by storing the most recent values of an instruction for different branch histories. The history of recent branch outcomes is maintained and used, along with the instruction address, to predict the next value of the instruction. For the example in Figure 1, two different values of $Addr_a$ are stored and then, based on the path followed, the appropriate value computed on that path is predicted.

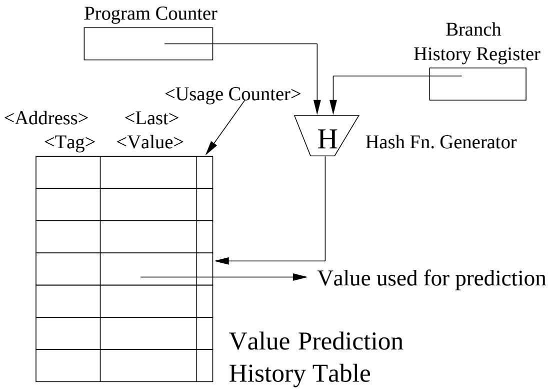

Figure 3. Per path Last value Predictor Microarchitecture

The implementation of this predictor is depicted in Figure 3. The components of the predictor are as follows:

Branch History Register (BHR): This register is similar to the history registers used in branch prediction techniques [9]. The register stores the most recent branch outcomes. After each branch is evaluated, the entire history contents are shifted left by one bit and the least significant bit is set/reset depending on whether the branch was taken/not taken. The number of paths that can be represented is limited by the size of the register.

Value Prediction History Table (VPHT): This table stores the values of instructions for different branch histories. It differs from the value prediction table proposed in [7] in that this table may store several values computed by an instruction for different branch histories. The table is set-associative and accessed using the instruction address and BHR bits. Each entry stores the predicted value and a usage counter for performing replacement. The counter is a saturating counter incremented after every correct prediction and decremented upon a misprediction. Whenever an entry needs to be replaced, the one with the lowest counter value is selected for replacement.

Hash Function Generator : The VPHT is accessed using both the address of the instruction and the history register. This unit performs the hashing of the two for indexing into the VPHT. For instruction i and BHR's value b, the function used is

$$f_1(i,b) = ((Addr(i) << Size(BHR)) + b) \bmod Size(VPHT)$$

where *Addr(i)* denotes the address value of instruction i, *Size(BHR)* and *Size(VPHT)* give the total number of distinct entries in buffer BHR and VPHT respectively. This function maps an instruction within a segment of

Benchmark	Input set	Instrns. analyzed
Unix Utilities		
diff	4K C files	5.4M
gawk	parse 250K input	6.9M
grep	search 1.28M file	20.9M
SPEC INT 95 programs		
li	train.lsp	125.3M
perl	primes.in	10.3M
ijpeg	vigo.ppm	1282.4M
go	2stone9.in	397.1M
SPEC FP 95 programs		
applu	applu.in	147.1M
fpppp	natoms.in	112.5M
swim	swim.in	176.8M
wave5	wave5.in	934.5M

Table 1. Description of benchmarks

the VPHT of size $2^{Size(BHR)}$. Within each segment, different branch histories for the same instruction are mapped to different entries of the table. This function is easy to implement in hardware and all experimental results presented in this paper are based upon this function.

For each instruction, the predictor computes the index in the VPHT by hashing the instruction's address bits with the current branch history contents. If the index maps to a valid entry, the corresponding value is used for prediction. After execution, the computed value is compared with the predicted one. Upon a misprediction, the instruction using the predicted value is squashed and re-issued with the correct value. Upon the instruction's completion, its last value information for the current branch history is updated in the VPHT.

The performance of the PLV predictor was analyzed using trace-driven simulations and instrumenting the code to perform the prediction. The experimental framework consisted of an instruction-set simulator SHADE [2], a code instrumentation tool from SUN Microsystems, which simulates the SPARC (Versions 8 and 9) instruction sets. The instructions of the benchmarks were instrumented to record the instruction values and branch histories. Performance was analyzed by executing the traces generated by the instrumented code over the PLV predictor's simulated microarchitecture on a 64-bit SPARC Ultra-2 processor running SunOS 5.5.1. The benchmarks were compiled using the SUN Workshop C Compiler, Version 4.2 with the -O option, and run until completion. The data input to the benchmarks is described in Table 1. Only the integer instructions were analyzed in the benchmarks. Since previous work [7] shows that floating point instructions exhibit value prediction patterns similar to those of integer instructions, similar performance is expected for floating point instructions.

We implemented the PLV predictor and compared its performance with the performance of the last value predictor and the FCM predictor. The FCM predictor was implemented as a two-level table, with tables of both levels having the same size. The first level table entries consisted of an address tag and partial values from the last three instances of the instruction. The address tag along with the stored values are hashed to a second table storing predicted values for the instruction. The hashing function used performs exclusive-OR on the most significant bits of the stored values.

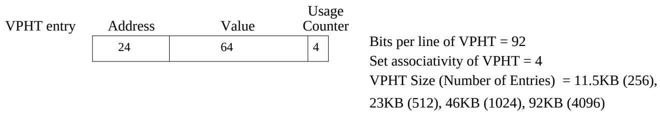

Figure 4. Table entry line for PLV predictor

For each scheme, we performed experiments for different table sizes. The different sizes of the VPHT in PLV predictor are shown in Figure 4. Each entry is 4-way set associative. Since it is possible that some of the programs require only a small branch history and others require greater path information, we ran our experiments for three different branch history depths, last 2, 4 and last 8 branches. It was observed that for smaller table sizes a small branch history performed best since larger branch histories result in significant aliasing. Overall the branch history size of 2 produced the best results as compared with the other history sizes and therefore we selected this number for all our experiments.

Performance comparison of the PLV predictor with the last value and the FCM predictors, for same number of entries per table, is shown in Figure 5. The figure shows the percentage of analyzed instructions that were successfully predicted. The first bar for each benchmark shows the result of applying the FCM technique. The second and third bars show results of applying the last value technique and PLV prediction technique respectively. From the comparisons, it can be seen that the FCM method did not match up to the other techniques for realistic table sizes. This is due to the large number of patterns that need to be stored, resulting in frequent collisions within the tables. Also, the PLV predictor outperforms the conventional predictors for most of the cases. For a 256 entry VPHT, there is slight degradation in performance in cases of *ijpeg*, *swim* and *wave5*, as compared to last value prediction. The degradation can be attributed to the aliasing problem, apparent for smaller table sizes, since we need to store several values for different branch histories for an instruction. With an increase in the number of entries, the aliasing is reduced and a considerable

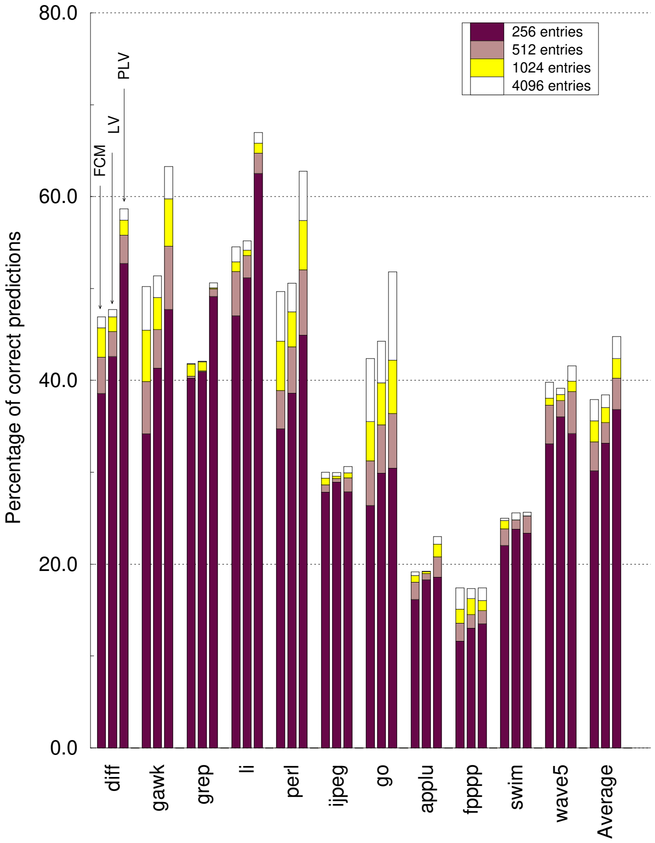

Figure 5. Performance of FCM vs. Last value vs. PLV Predictors in Percentage Predictions

performance improvement is observed. The average performance improvement over last value prediction is 6.4% for the table size of 4096 entries. For larger table sizes, this improvement is higher for larger branch history but this is not apparent from the results shown which use a fixed branch history size throughout.

In the above experiments, the branch history and predictor table state are updated immediately after the values are available. In a real machine, this update may take a few cycles, in which case the instruction is not allowed to execute immediately. In order to take these extra cycles into account, we ran the benchmarks using a delayed update of 2 cycles as an estimate of the extra cycles needed for an update. This resulted in reduction in prediction accuracy in the range of 2-7% in case of both the conventional and new prediction schemes. The reduction in accuracy is due to the VPHT remaining inconsistent after executing an instruction for 2 cycles, when the table would be updated by the instruction's value. However, the average performance improvement decreased only by a fraction of a percentage indicating the improvement for the proposed predictors was not impacted severely by delayed updates.

3 Per-Path Stride Prediction

We also developed two prediction schemes that improve stride prediction by incorporating path information. The first of these two schemes, Per-Path Stride (PS) prediction, stores different stride values for an instruction along different paths. The extension of stride prediction is analogous to extending last value prediction within the PLV predictor. Path information can be useful for stride prediction when a loop variable is updated by different amounts along different paths within a loop. In the example of Figure 1, the variable $Offset_v$ demonstrates this characteristic. While the strides are stored separately for different branch histories, the last value is stored globally for all the strides. The table storing the strides is accessed using the instruction address and branch history bits.

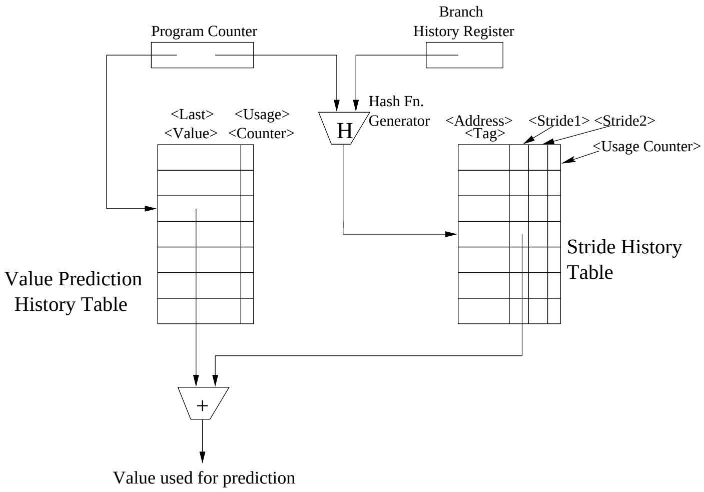

Figure 6. Per-path Stride Predictor Microarchitecture

The components of the predictor are shown in Figure 6. Besides using the components of the PLV predictor, this scheme makes use of an additional table, the **Stride History Table(SHT)**. The SHT stores the stride values for different execution histories of an instruction. It is accessed by hashing the address bits and the branch history register value. The stride update policy used here is the **two-delta policy** [3]. In this policy, the stride gets updated only if the difference between the two most recent values of an instruction occurs twice in a row. To implement this policy, two strides are stored for an instruction. The first one stores the difference between the last two values computed by the instruction while the other stride stores the value being used for prediction. The latter value is updated when the first stride has the same value twice in a row.

For each analyzed instruction, the predictor uses its lower (k bits) to map to the VPHT. The last value of the instruction is obtained from this table. Simultaneously, the address bits are hashed with the branch history to map to the SHT. This mapping accesses the stride used for the instruction and current branch history. The sum of the two values is used to predict the next value. Similar to the PLV scheme, the correctness of the prediction is checked after execution of the instruction is completed, and the update of the last value and stride is performed following completion.

The second path-based stride predictor scheme combines the previous two proposed schemes by maintaining a last value for each path, as well as a stride value for each path. This scheme, Per-Path Stride Per-Path Last Value (PS-PLV) prediction, is used to handle conditions such as that of $Addr_c$ in the example of Figure 1, whose value depends on operands evaluated differently along different paths. The implementation is shown in Figure 7. It is similar to the PLV predictor with the exception that each entry in the VPHT now also contains a stride value.

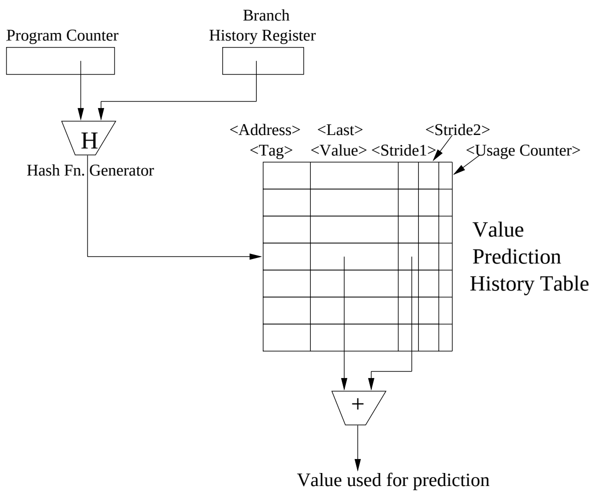

Figure 7. Per-path Stride Per-path Value Predictor Microarchitecture

Each instruction is hashed to index the VPHT based on its address and branch history. This table stores the last value as well as the strides for each branch history. A single access results in accessing both the values to be used for prediction. These values get updated during the completion stage of the instruction, as in the previous methods.

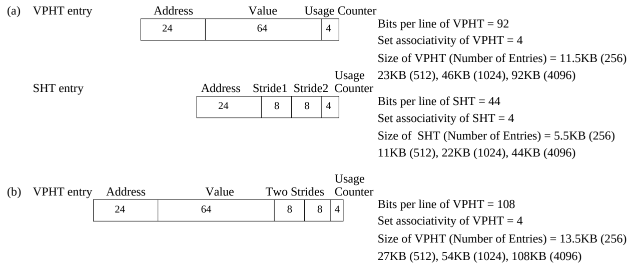

Figure 8. Table entry lines for (a) PS predictor (b) PS-PLV Predictor

The configurations of PS and PS-PLV predictors that were used are shown in Figure 8. They were implemented and compared with conventional stride predictors. The performances of the PS and PS-PLV predictors are shown in Figure 9. For each benchmark, the first bar shows the result of applying conventional stride prediction. The second and third bars show results of applying the PS and PS-PLV prediction techniques respectively. The fourth bar shows the result of applying a hybrid predictor that combines the PS and PS-PLV prediction techniques. Details of this hybrid predictor are provided below. Similar to the study for the PLV predictor, we chose a branch history size of 2 for our analysis. From the figure, we observe that the PS predictor almost always outperforms the conventional stride predictor (except for the benchmark *fpppp* which has a marginal degradation in performance). The PS-PLV predictor also gives a better performance when compared to the conventional stride predictor for most of the cases. However, the performance improvement is less than with PS predictor. Overall, the PS predictor shows an average improvement of 8.4% for 4096 entries while the PS-PLV predictor improves performance by an average of 6.9%.

From the above results, we infer that the PS predictor improves conventional prediction significantly. Notice that both PS and PS-PLV prediction schemes would be able to capture different sets of prediction cases (as illustrated by the example of Section 1). Hence a hybrid predictor that uses both these prediction schemes would potentially give an additive improvement in performance. Such a hybrid predictor was implemented involving both the PS and PS-PLV prediction mechanisms. The method used for prediction was based on a confidence mechanism associated with the instruction. The confidence mechanism used was a 4-bit counter value that was updated depending on which prediction mechanism performed correctly on the instruction. The performance of such a predictor is shown in Figure 9. The improvement in performance over conventional stride prediction for 256 entries was

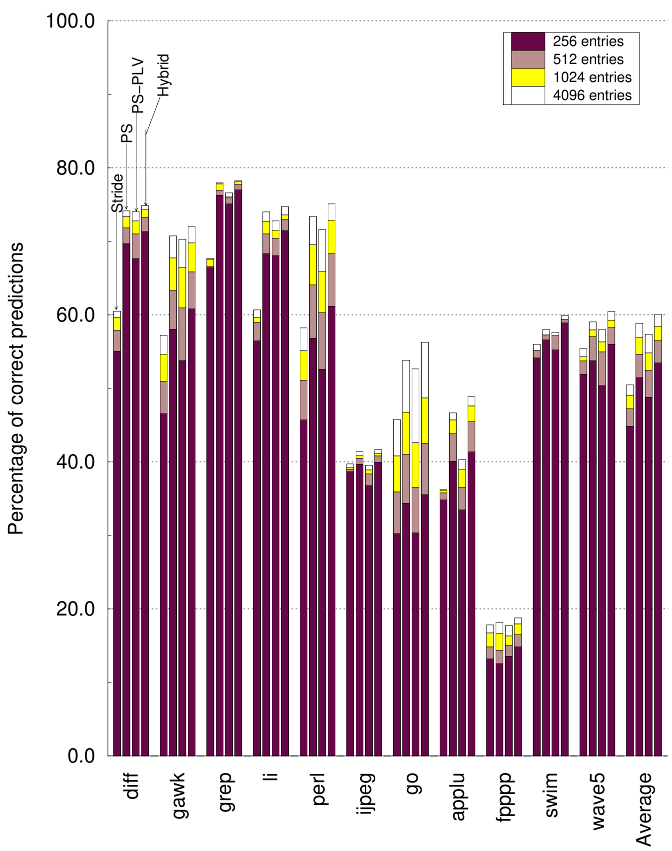

Figure 9. Performance of Stride vs. PS vs. PS-PLV vs. Hybrid Predictors in Percentage Predictions

10.4% and was 2% over the best path-based stride predictor. The main issue to be explored in implementing a hybrid predictor of this form is to select which predictor to use every time a prediction is to be made.

The performances of all the predictors were reduced by 3-8% when the the predictors were implemented with a delayed update. However, the performance improvements of the new predictors were reduced marginally.

4 Path and Instruction-based Value Prediction

It is possible that information other than path information may be useful in predicting an instruction value. This was demonstrated in the example when $Base_v$ was moved to block 5 in Figure 2, and $Addr_a$ could no longer be predicted using path history. A scheme that performs prediction of the value of an instruction by correlating it with the value computed by the previous instruction in the instruction sequence was

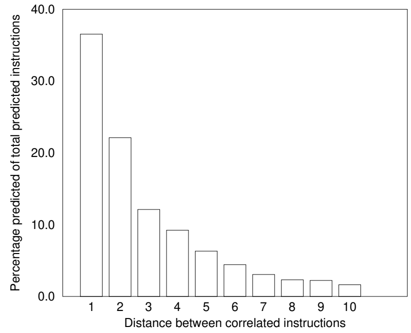

Figure 10. Percentage predicted vs. Distance between correlation

shown to predict $Addr_a$. However, using simply the last instruction will not allow us to predict $Addr_b$ and $Addr_c$ in this case. In order to correlate with a previous instruction value, we need to store values generated by a number of previously completed instructions. The correlation is detected by selecting an instruction from these instructions and using its value to predict the current instruction's value. However, the issue here arises as to which instruction to select out of the recently completed instructions.

We performed experiments to observe the detectable correlation with recently completed instructions, the results of which are presented in Figure 10. From this figure, we notice that 37% of the instructions are cor-

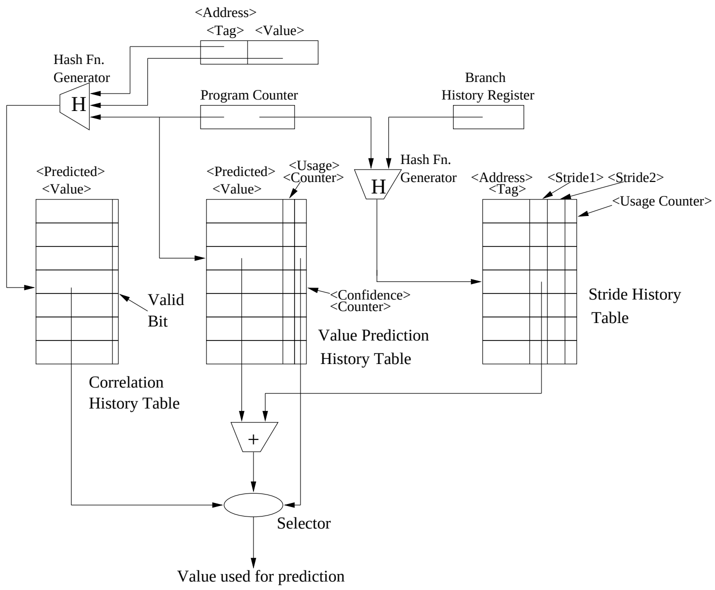

Figure 11. Per-Stride Previous Instruction-based Predictor Microarchitecture

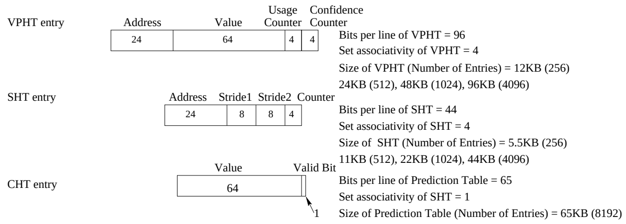

Figure 12. Table entry lines for PS-PI Predictor

rectly predictable due to their correlation with the immediately preceding instruction appearing in the dynamic sequence. Therefore, to maintain simplicity of the predictor we choose to correlate the instruction to be predicted with the immediately preceding instruction.

Prediction using instruction correlation is able to capture prediction cases that are independent of paths. Hence, we implemented a hybrid predictor that performs prediction using instruction correlation in conjunction with the PS predictor. The PS predictor was chosen as part of the hybrid predictor since it produced

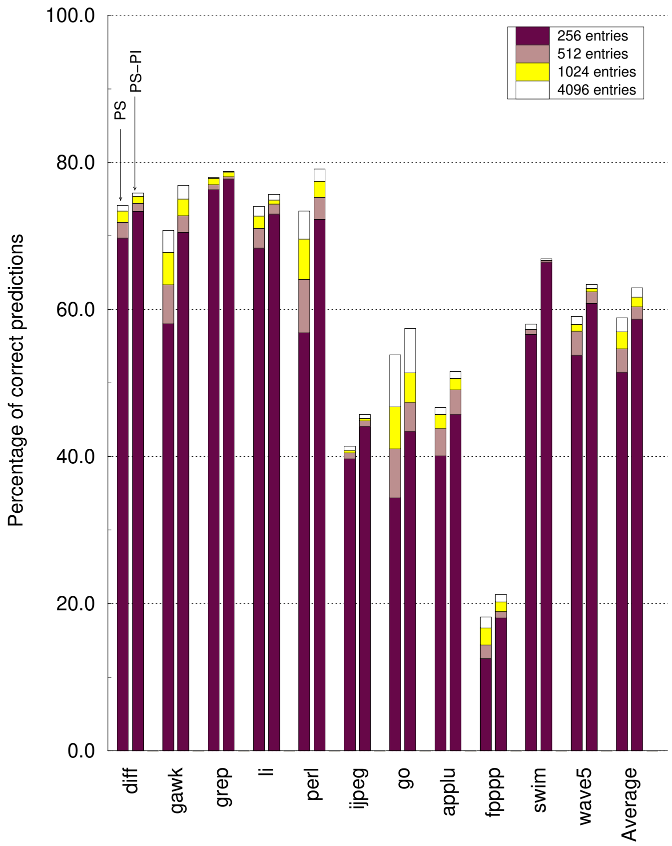

Figure 13. Performance of PS vs. PS-PI Predictors in Percentage Predictions

the best performance of the path based predictors. The new predictor, **Per-Path Stride Per-Instruction (PS-PI)**, attempts to predict the instruction value using either path information or the previous instruction's value. The prediction component that is selected, each time an instruction is predicted, is based on a confidence mechanism. Each entry of the VPHT has a 4-bit confidence counter whose value is incremented whenever the PS component makes a correct prediction and decremented when the instruction-based component predicts correctly. The value of this counter is used to decide which prediction component to use for predicting the next value of the instruction. In case the instruction does not have an entry in the VPHT, the instruction-based component is used for prediction.

The implementation of the PS-PI predictor is shown in Figure 11. The left half of the figure depicts the previous-instruction-based component of the predictor. It uses a table, **Correlation History Table (CHT)**, which stores values that are predicted using correlation with previously computed values. In order to perform such correlation, the address and value computed by the last instruction are always saved and used (along with the program counter) to map to the entry storing the predicted value in the table. The right half of the figure shows the PS component described in the last section. The selector chooses one of the two values predicted based on the confidence counter value in the VPHT.

The size of the CHT was fixed at 8K entries. We analyzed the performance of previous-instruction-based correlation by comparing the PS-PI predictor with the PS predictor using the experimental design described in Section 2.1. The results are depicted in Figure 13. The first bar shows the result of applying the PS predictor and the second bar shows the result of applying the PS-PI predictor. We observe that the improvement is significant for all benchmarks. The average improvement is 7.2% for table sizes of 256 entries and similar performance for other table sizes.

When the predictor was modified to incorporate delayed update of branch history and tables as discussed in the last two sections, the prediction accuracy of all the predictors was reduced in the range of 2-8.5%. However, the degradation in performance improvement over PS predictor was less than 2%.

5 Conclusions

From our experiments, a number of conclusions can be drawn.

(1) Path information is helpful in performing prediction of data values of instructions. Three novel schemes have been proposed for performing value prediction sensitive to path information. The first scheme, PLV predictor, improves the last value prediction by as much as 15% for some benchmarks and 6.4% on average. The other two proposed predictors, PS and PS-PLV, extend the stride predictors. The PS prediction scheme shows an improvement exceeding 15% for some benchmarks and 8.4% on average. For PS-PLV prediction, the improvement exceeds 13% for some benchmarks and 6.9% on average. The PS and PS-PLV stride predictors are combined in a hybrid predictor, which improves prediction accuracy by 10.4% over convention stride prediction and by 2% over the best path-based predictor.

(2) Prediction based on correlating instruction values can produce additional benefits over path-based predictors. The proposed previous-instruction-based prediction scheme is incorporated within a hybrid predictor, PS-PI. Analysis of the PS-PI predictor indicates that this method can capture several cases of instruction values that are not predictable by path-based predictors. The percentage of instructions of this kind exceeds 15% for some benchmarks and 7.2% on an average.

(3) Significant performance improvement is attainable with a simple hardware scheme and realistic table sizes. The proposed prediction schemes provide improvement over the conventional last value and stride predictors for limited number of entries per table. The FCM predictor does not match up to these predictors for the table sizes under consideration. We have shown that each of the proposed schemes would require very simple hardware for implementation.

(4) Delaying update of information as is done in real microarchitecture does not affect the performance significantly. We performed experiments by updating branch history a few cycles after the branch outcome is available. In an attempt to give a more realistic view of the performance, we performed experiments by updating branch history 2 cycles after the branch outcome is available. Results indicate that the degradation in performance due to delayed update is minimal.

References

[1] T. F. Chen and J. L. Baer. A performance study of software and hardware data prefetching schemes. *Proceedings of the 21st International Symposium on Computer Architecture*, pages 223–232, April 1994.

[2] R. F. Cmelik and D. Keppel. Shade: A fast instruction set simulator for execution profiling. *Tech. Rep. TR-93-12, Sun Microsystems Laboratories*, July 1993.

[3] R. J. Eickemeyer and S. Vassiliadis. A load instruction unit for pipelined processors. *IBM Journal of Research and Development*, pages 547–564, 1993.

[4] F. Gabbay and A. Mendelson. Speculative execution based on value prediction. *EE Department TR#1080, Technion - Israel Institute of Technology*, November 1996.

[5] D. M. Gallagher, W. Y. Chen, S. A. Mahalke, J. C. Gyllenhaal, and W. W. Hwu. Dynamic memory disambiguation using the memory conflict buffer. *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 183–193, October 1994.

[6] M. H. Lipasti. Value locality and speculative execution. *Ph.D. Thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University*, May 1997.

[7] M. H. Lipasti and J. P. Shen. Exceeding the dataflow limit via value prediction. *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 226–237, December 1996.

[8] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value locality and load value prediction. *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 138–147, October 1996.

[9] S. McFarling. Combining branch predictors. *Tech. Rep. DEC WRL TN-36*, pages 281–290, June 1993.

[10] A. Moshovos and G. Sohi. Streamlining inter-operation memory communication via data dependence prediction. *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 235–245, December 1997.

[11] S. T. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 76–84, October 1992.

[12] Y. Sazeides and J. E. Smith. The predictabilty of data values. *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 248–258, December 1997.

[13] Y. Sazeides, S. Vassiliadis, and J. E. Smith. The performance potential of data dependence speculation and collapsing. *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 238–247, December 1996.

[14] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 281–290, December 1997.

[15] T. Y. Yeh and Y. N. Patt. Alternate implementation of two-level adaptive branch prediction. *Proceedings of the 19th International Symposium on Computer Architecture*, pages 124–134, May 1992.