ECE-97-8 December 1997

Implementations of Context Based Value Predictors

Yiannakis Sazeides and James E. Smith
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
1415 Engr. Dr.
Madison, WI 53706
yanos@ece.wisc.edu, jes@ece.wisc.edu

Abstract

Execution paradigms that eliminate data dependences based on value prediction have been shown to have significant performance potential. High accuracy value prediction is essential for the success of such paradigms. Recently it was shown that *context based* prediction can predict values with high accuracy. A context based predictor learns the values that follow a particular context (sequence of values) and predicts one of the values when the context repeats. However, in the earlier study unbounded tables were used.

In this paper we discuss implementations of context based value predictors using two level table organizations. The new elements introduced in this paper are fixed size tables and sharing of prediction information among instructions. A number of issues unique to context based value prediction are considered. A variety of implementations of the proposed value predictor are evaluated by comparing their prediction performance to the "ideal" performance with unbounded tables. We found that such predictors obtain accuracies that approach that of "ideal" predictors. The proposed predictor was also compared against stride and last value predictors. Stride and last value do better than context based prediction for small tables but context based prediction is superior with increasing table size. The sensitivity of the context based prediction to the size of the input data was explored and found to be relatively insignificant.

1 Introduction

Speculation is one of the approaches widely advocated for high performance processor design. Speculation can improve performance by proceeding with instruction execution using predicted program information instead of waiting for the deterministic information. The success of a speculative approach depends highly on the accuracy with which program information is predicted.

Although just about any program information could be predicted, conditional branch outcomes are the only example that has been studied extensively. Another dominant information type that should be predicted is data values.

It would appear that value prediction has little chance for success because data can take a large range of values. Recently it was shown, to the contrary, that values produced by instructions can be very predictable [1, 2]. In [1] was shown that some values produced by instructions exhibit "locality"; that is, they tend to repeat for a large fraction of time, and therefore may be predicted correctly if an appropriate mechanism is used.

In [2] value predictors were classified as *computational* and *context-based*. A context based predictor learns the values that follow a particular context (sequence of values) and predicts one of the values when the context repeats. Results in [2] with unbounded tables show that context based value prediction outperforms stride-based computational prediction by more than 25% and that almost all correct predictions with stride-based prediction are also predicted correctly by the context based predictor. These results suggest that context based prediction is an important value prediction approach. However, the previous work in [2] did not consider many implementation aspects of such predictors.

In this paper we discuss the implementation of high accuracy value predictors. We propose a context based value predictor with a two level table organization. One of the arguments for the choice of this particular organization is its resemblance to finite context predictors used in text compression that have in general the best known prediction (compression) performance [3]. A number of issues unique to context based prediction are discussed and possible solutions are proposed. A variety of implementations of the proposed value predictor are evaluated by comparing prediction performance with the "ideal" performance reported in [2]. The performance of the proposed value predictor is also compared against computational predictors (last value and stride). Another interesting issue that is explored is the sensitivity of the context based prediction to the size of the input data.

1.1 Background - Value Prediction Models

In this section we present the classification of value predictors introduced in [2]. This classification will be useful for the discussion in subsequent sections.

A computational prediction model produces a prediction by performing a computation on previous values. Computational predictors can learn to predict correctly fast, and can even predict values not previously

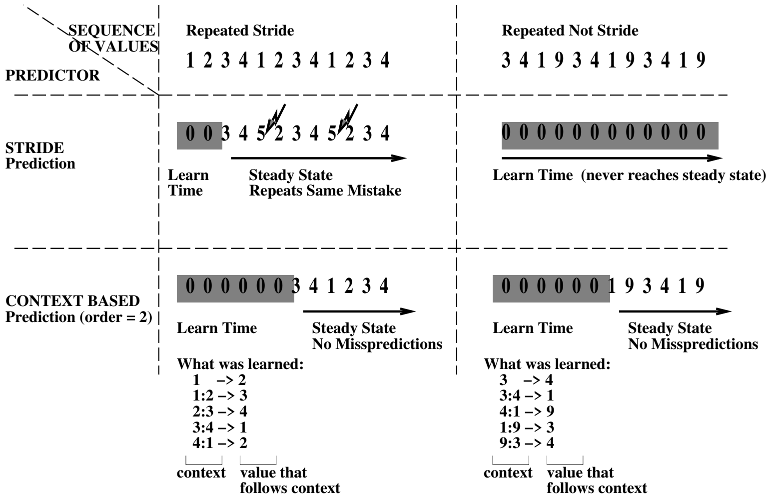

Figure 1: Differences between Stride and Context Based Value Prediction

observed. However, they are only capable of predicting sequence of values that correspond to the computation of the predictor. Examples are *stride* prediction, that produce a prediction by adding a constant to the most recent value, and *last value* prediction, that predicts the next value to be the same as the previous one. Stride value prediction was used in [4] and last value prediction was used in [1, 5].

A context based prediction model learns the values that follow a particular context and predicts one of the values when the same context repeats. The context is a sequence of k previous values. A context based predictor is said to be of *order-k* when the context includes information from k values. Context based prediction enables the prediction of any repeated sequence of values. The main limitation of context based prediction is that for a value to be predicted correctly it must have followed the same context at least once before. This is not a significant limitation for two reasons: program execution is highly iterative and loops are usually executed enough times so that the cost for learning is not significant. Furthermore, if the context predictor can share the information learned between different instructions, then when similar sequences are generated by several instructions they need to be learned by one instruction and other instructions generating identical sequences can be predicted correctly. Some of the important differences between functional and context based prediction are illustrated in Figure 1.

1.2 Related Work and Issues

In [1], it was first reported that data values produced by instructions exhibit "locality". The potential for value predictability was reported in terms of "history depth" locality, that is how many times a value produced by an instruction repeats when checked against the most recent n values. A pronounced difference is observed between the locality with history depth 1 and history depth 16. It is suggested that if a mechanism could be devised to correctly select between the n most recent values, high predictability can be expected. The mechanism proposed for prediction, however, exploits the locality of history depth 1 and is based on last value prediction that predicts the most recent value (last) will also be the next. In [1] the last value prediction was used to predict load values and in a subsequent work to predict all values produced by instructions [5]. Significant performance benefits were obtained with last value prediction driving the speculative execution of instructions.

Stride predictors traditionally have been used for address prediction and data prefetching [6, 7]. They have also been proposed for speculative execution of load and store instructions [8, 9]. Stride prediction was proposed for value prediction in [4], and its accuracy and performance potential were compared against last value prediction.

Value prediction can draw from a variety of work on the prediction of control dependences. A significant breakthrough in control dependence prediction was the use of correlation information in the prediction process. Two level tables have been the underlying organization of successful correlation predictors [10, 11, 12, 13]. Recently it was demonstrated theoretically that two level tables, as used for control dependence prediction, resemble the prediction models used for text compression that are known to have the best prediction performance [14]. This is an important observation because it re-enforces the approach used for control dependence prediction and also suggest that methods based on compression can be used for data value prediction.

Data can take a large range of values, this gives rise to the problem of how to reduce the values of a context to an index suitable for indexing into predictor structures. Though this problem has not been considered for values, solutions have been proposed for encoding addresses used for branch and jump prediction [11, 12, 13]. These schemes perform different functions on the lower bits of the addresses. The higher bits are usually ignored because they seldom change. Such an approach may be inadequate for value predictions since values can take large range of values and high order bits can be as significant as low bits.

In [2] value prediction was studied at a fundamental level with no cost considerations. Static instructions were treated individually, and prediction tables were of unbounded size. Consequently the potential and effects of sharing prediction information between instructions was not explored. Furthermore, performance reduction (if any) due to the use of fixed table size was not considered.

This paper differentiates from previous work, firstly, by considering the implementation of context based value predictor, secondly, enabling sharing of prediction information among instructions, and thirdly, adopting a hash function that uses bits from an entire value instead only from the lower order.

1.3 Overview

The paper is organized as follows: in Section 2 we describe a generic value prediction organization. The simulation methodology is described in Section 3, and the experimental results are discussed in Section 4. We conclude and present future direction in Section 5.

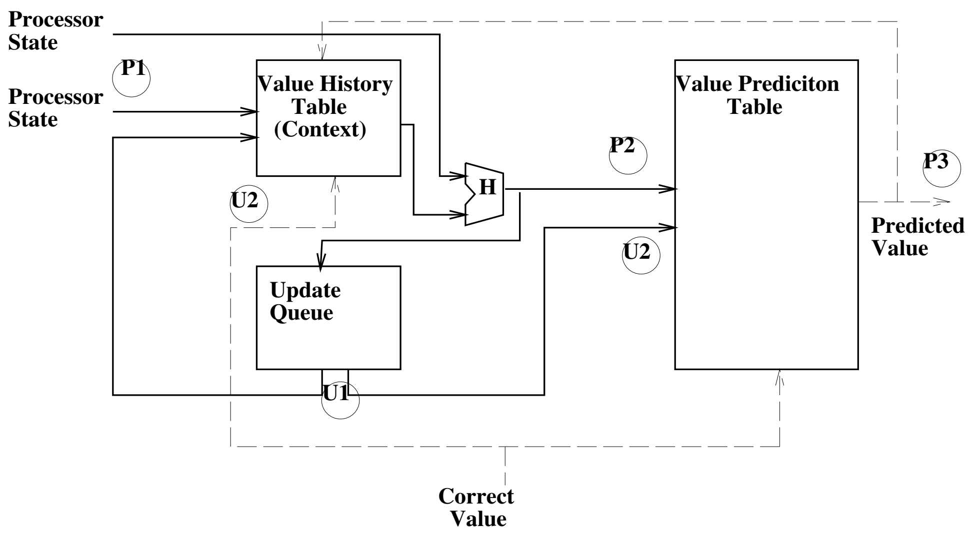

Figure 2: Generic Two Level Table for Value Prediction

2 A Two Level Table Organization for Context Based Value Prediction

A two level organization is proposed for value prediction. The main elements of a generic two level value predictor are shown in Figure 2. The main storage structures are the *value history table (VHT)*, that contains the context - previous history of values used to select the next prediction, the *value prediction table (VPT)*, that contains the predictions, and the *update queue*, that contains information that will enable the correct updates of the history and prediction tables when the correct value for a prediction becomes available.

The annotation in Figure 2 describes the events that can take place in this model when predictions and updates occur. To make a prediction the following are performed: (**P1**) Using information derived from processor state, an index is formed to access an entry in the value history table. (**P2**) The context selected from the value history table and information derived from processor state are reduced using a hash function to an index for accessing the value prediction table to get a prediction. The selected context is saved in the update queue to be used later when the update for the instruction is performed. (**P3**) The contents of the value prediction table are accessed and used to form a prediction. Due to the delay between the time of a prediction and the time when the correct value becomes available, it may be desirable to update the value history table speculatively using the predicted value.

The steps to perform an update are: (**U1**) When the correct value becomes available the update queue is accessed and the saved context is retrieved. (**U2**) Using the correct value, the value history and value prediction tables are updated.

Almost limitless possibilities exist with regard to the information used to access predictor tables as well as the information stored in the predictor. Some of the possibilities are discussed next.

Value History Table Index: can be derived from processor state using information such as the address and the type of the instruction to be predicted, global branch history, register values, bits in the pipeline etc.

A variety of schemes for forming value history table indices have been proposed for conditional branches in [15]. The choice of a particular index function could be dependent on the size of the value history table.

Contents of Value History Table: include k values (an order-k context) possibly in hashed form. The context is used to identify a point in a value sequence so that the next value can be predicted. The context is analogous to the history register or path information used for branch prediction and jump prediction [11, 12, 13]. In theory, the larger the order, the better for prediction accuracy, however, in a practical scheme this may result in increased aliasing. Other contents of the value history table include *prediction confidence* and *update confidence*.

The prediction confidence mechanism is the means used to reduce misspeculation and can be implemented as a finite state machine such as a saturating counter that is incremented/ decremented on success/failure. Prediction confidence can also be associated with a value prediction table entry. There is a subtle difference between the two choices, the value history table choice will reflect the confidence for an instruction, whereas the value prediction table choice the confidence following a certain context. The decision for either scheme may be dependent on the size of the value history table. Update confidence can be used to eliminate start up updates that may pollute the value prediction table. The start up confidence can be a state machine used to determine when an update to the value prediction table should take place. A VHT table entry may also contain tag information aiming to reduce mispredictions due to aliasing.

Value History Table Size: for branches is known that high accuracy can be attained even with a single value history table entry (global history registers) [10, 16]. This is the case because branch values are single bit and a global history register, in combination with address information, can do relatively well in identifying a branch in a program. For value prediction, however, the granularity of the predicted values is larger and prohibits forming an index to the value prediction table using the full value history. The reduction of the context will result in value prediction table conflicts from different contexts. There are at least two approaches and their combination that can be used to alleviate this problem:

- Increase value history table size so that instruction context becomes more unique; and,
- Incorporate additional information in the hash function to help identify the predictions in the value prediction table that are associated with an instruction.

Hash Function: the hash function forms an index into the value prediction table using context information from the value history table and information derived from processor state. The hash function can be better conceptualized as two distinct functions: a **context function** that reduces the k values of a context, and a **combine function** that merges the state from the processor and the output of the context function.

Context Function: Numerous context functions can be defined, in this work we consider the following three important classes of functions:

Select-Concatenate (C): select certain bits from each value in the context and concatenate them together. The bit positions and the number of bits selected from each context value can be different.

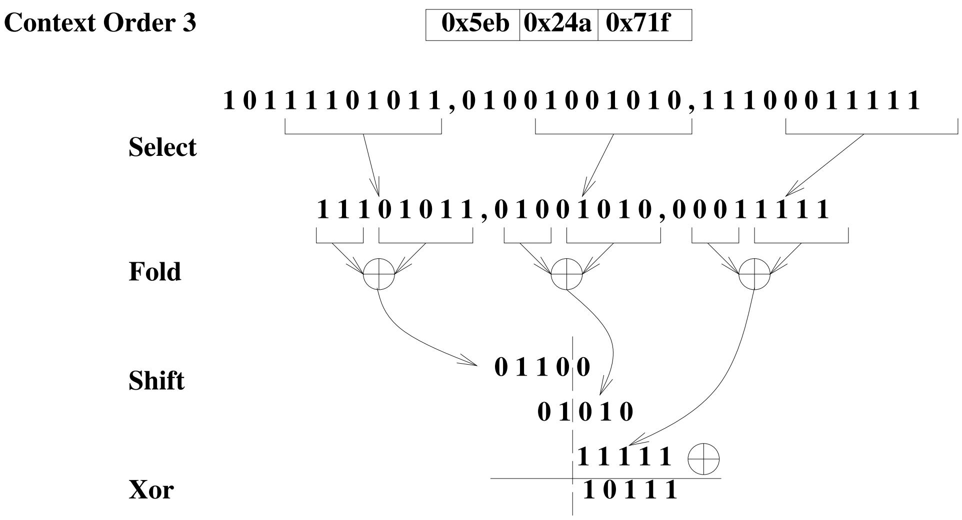

Figure 3: *Select Fold Shift Xor* Function

Select-Fold-Concatenate (FC): extends the previous function with the insertion of a folding step. The folding reduces the selected bits into a smaller number of bits. A folding operation that can be considered is the bitwise xor of the selected bits.

Select-Fold-Shift-Xor(FS): uses a different operation for combining the folded context values. Each folded value is shifted by a number of bits and then all values are xored together. It should be noted that C and FC functions are special FS cases.

A function for reducing an order-k context into an index with b bits can then be described in terms of the following operations: $\mathbf{R}(\mathbf{r}_{k-1}\ ..\ \mathbf{r}_0)\mathbf{F}(\mathbf{f}_{k-1}..\ \mathbf{f}_0)\mathbf{S}(\mathbf{s}_{k-1}\ ..\ \mathbf{s}_0)$. The R operation selects r_j bits from the jth value in the context, the F operation folds the selected value into f_j bits which are left shifted by s_j bits and xored together with all values in the context. The b least significant bits of the resulting value is the index. In general r_j should be a bit vector but in this work we assume that r_j specifies the number bits to be selected starting from the least significant position of a value and ending at bit position r_j-1. In Figure 3 is shown an example of how the FS function is used to form an index of 5 bits using a context with three values. The function used in Figure 3 can be defined as R(8,8,8)F(5,5,5)S(4,2,0)

The combine function: can be as simple as the identity of the output of the context function or can be more interesting and include processor state information. Processor state information can be effective in identifying the predictions associated with a particular instruction in the value prediction table. Information that can be used is a part of the address of the predicted instruction, branch history etc. An effective operation for combining all the information together is the xor function[16].

Value Prediction Table Size: the table size is limited by cost and directly determines the size of the index generated by the hash function and possibly could influence the number of values used in a context.

Contents of Value Prediction Table: can include one or more predictions per entry, a selection mechanism for choosing among many predictions, and confidence mechanisms for predictions and replacement.

The replacement policy can be a finite state machine such as a saturating counter that is incremented/decremented on success/failure. When the counter is below a threshold then an old prediction can be replaced with a new one. The same state machine, or a separate one, can be used to maintain the confidence in the correctness of a prediction. Recall that in some cases a value history table prediction confidence may be more appropriate.

An alternative replacement mechanism maintains two predictions *p1* and *p2*. The one, p1, is the prediction and the other, p2, is used to determine when p1 is replaced. P2 is always updated, however, p1 is only updated with the value in p2 only when a new value matches the value in p2 [8].

If more than one prediction is maintained per entry then it will be necessary to have a voting scheme for which prediction to select. Examples of voting schemes include counters and a most recently used flag. Counters can be incremented every time the predicted value is the actual value that follows the context. For counters there are two issues that need to be addressed: what to do when the maximum is reached? and what is the maximum count? In the area of text compression it was found that small counters, 5-8 bits, work very well. When a counter reaches its maximum, it is suggested to half all counters in the same context [3].

Update Queue: the simplest update order for the predictor state is program order. This may be too conservative and could hinder prediction performance. However, a variety of alternative schemes can be conceived where updates take place out of order while individual predictor entries are updated in order. An example of such scheme could be to update each value history table entry in order only with respect to the updates that go to that specific entry and update the value prediction table strictly in program order. Other variations can be devised but are not further discussed in this work.

3 Simulation Methodology

This study considers the prediction of values that are results of instructions and are written into general purpose registers (this excludes store and loads addresses and branches - although context based prediction could be useful for them, in particular addresses). The study does not address microarchitectural issues in an effort to focus and understand the process of value prediction. Three predictors are used in the simulations, context based, stride and last value.

The parameters that are varied in the simulations for context based prediction are:

- Value history table index (truncated pc),
- Order (number of values in a value history table entry),
- Value history table entries (1, 4, 16, 64, 256, 1K, 4K, 16K, 64K)
- Context function (C, FC, FS),
- Combine function (identity, program counter based), and

Description		Context Function	
Code	Order	C	FC
P-3	6	R(3,3,3,3,3,3)	R(32,32,32,32,32,32)F(3,3,3,3,3,3)
P-5	4	R(5,5,5,3)	R(32,32,32,32)F(5,5,5,3)
P-6	3	R(6,6,6)	R(32,32,32)F(6,6,6)
P-65	4	R(6,5,4,3)	R(32,32,32,32)F(6,5,4,3)
P-8	5	R(8,4,2,2,2)	R(32,32,32,32,32)F(8,4,2,2,2)
P-18	1	R(18)	R(32)F(18)
Description		FS	
R-1	$\lceil t/1 \rceil$	R(32,..,32)F(t,..,t)S(,..order+0,0)	
R-3	$\lceil t/3 \rceil$	R(32,..,32)F(t,..,t)S(,..order+0,0)	
R-5	$\lceil t/5 \rceil$	R(32,..,32)F(t,..,t)S(,..order+0,0)	
R-7	$\lceil t/7 \rceil$	R(32,..,32)F(t,..,t)S(,..order+0,0)	
R-18	$\lceil t/18 \rceil$	R(32,..,32)F(t,..,t)S(,..order+0,0)	

Table 1: Context Functions Description

Benchmark	Input Flags	Dynamic Instr. (mil)	Instructions Predicted (mil)	Note
compress	30000 e 2231	8.2	5.8 (71%)	one iteration
gcc	gcc.i	203	137 (68%)	reference flags
go	9 9	132	105 (80%)	
ijpeg	specmun.ppm	129	108 (84%)	test input
m88k	ctl.raw	493	345 (70%)	train input
perl	scrabbl.in	40	26 (65%)	train input and dictionary
xlisp	7 queens	202	125 (62%)	

Table 2: Benchmarks Characteristics

- Value prediction table entries (4K, 16K, 64K, 256K, 1M).

The different context functions used, their order and a code that is used subsequently to identify them for the presentation of results are shown in Table 1. The C and FC functions are defined for an index that is 18 bits long (a 2^{18} entry value prediction table). The class of FS functions considered is given a general definition. A specific FS configuration depends on the size of the value prediction table, t $=\log_2$(Number of Value Prediction Table Entries).

For last and stride value predictors we only varied the number of entries in the prediction table. For last value the predictor was updated always with the most recent value. For stride prediction the 2-delta algorithm was used[8] .

A value history table entry contains the context values in hashed form. An entry in the value prediction table contains a single prediction and a replacement finite state machine. The replacement mechanism is a 3 bit saturating counter that is incremented/decrement by 1/1 on a success/failure. A prediction is replaced with a new one when the replacement confidence is less than 4. Updates to the value history table always

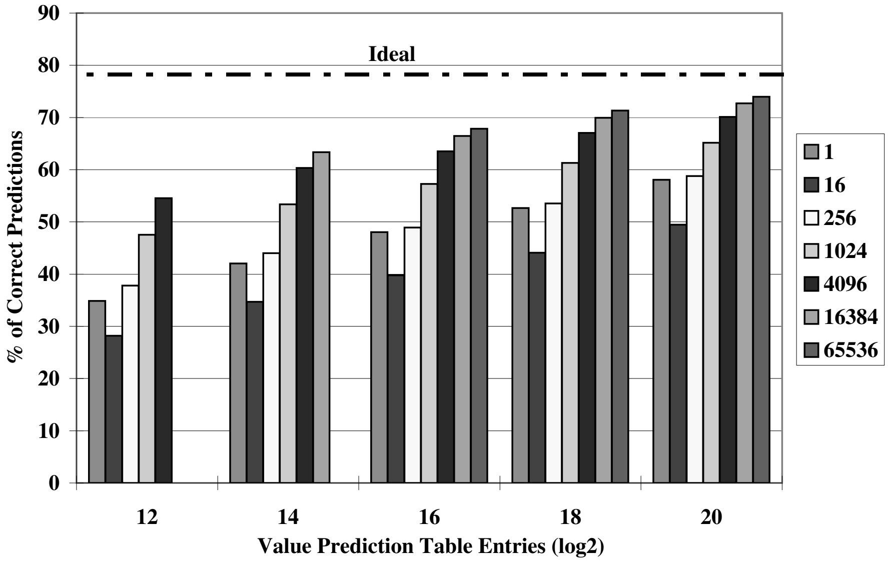

Figure 4: Predictability using Context Based Predictor

take place. The simulations did not consider the effects of timing between predictions and updates. As a result the update queue design space was not explored and the impact of speculative updates was not considered. Confidence mechanisms issues were also not investigated.

It is noteworthy that the above suggest value prediction has an enormous design space that we roughly explored.

Trace driven simulation was used to evaluate the different configurations of the various predictors. The SPECINT95 benchmarks shown in Table 2 were simulated using the simplescalar tool [17]. The benchmarks were compiled with -O3 optimization. Integer benchmarks were selected because they tend to have less data parallelism and may therefore benefit more from data predictions.

In the results section unless indicated otherwise the following default parameters for context based prediction are used: R-5 context function, the combine function is identity and the value prediction table has 2^{20} entries. Whenever "ideal" results are used they correspond to the results reported for an order-3 unbounded context based predictor in [2]. For averaging we used arithmetic mean, so each benchmark effectively contributes the same number of total instructions.

4 Results

4.1 Two Level Context Based Prediction

The results in Figure 4 show prediction accuracy averaged over all benchmarks. Each bar corresponds to a different value history table size, from 1 to 65536. An "ideal" line is drawn for comparison purposes.

The results show that prediction accuracy can approach the ideal with bounded table sizes. Although the tables are large, they are not impossibly large, and all value-producing instructions are predicted. If a subset

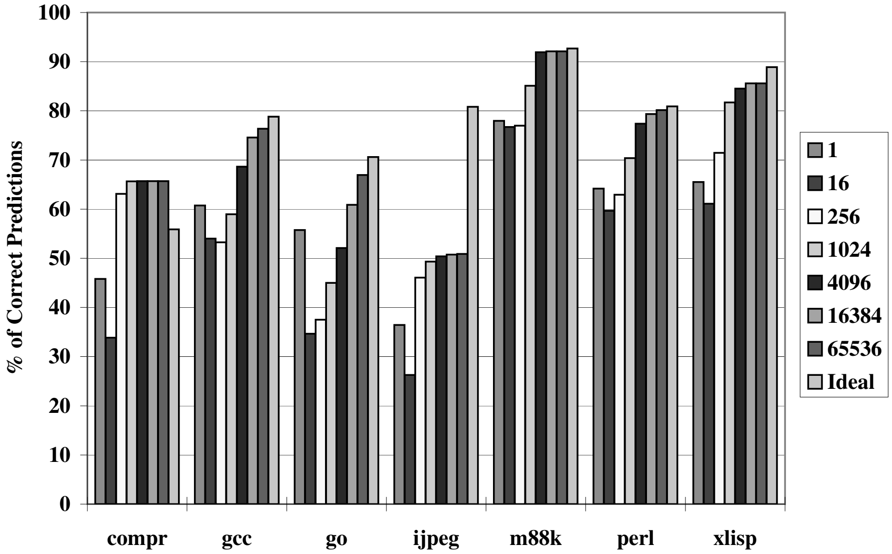

Figure 5: Overall Benchmark Predictability

of instructions were predicted, as would be the case in a real application, then undoubtedly smaller tables could be used. Furthermore, hybrid tables combining a context predictor and a computational predictor would likely reduce the table space required for the context predictor. That is, these results indicate the feasibility of context based prediction, especially if additional engineering is applied.

The data indicate it is important to have both large value history and value prediction tables. For instance, very large value prediction tables with small history tables do worse than smaller value prediction tables and larger history tables (compare configurations 2^{10}-2^{20} and 2^{12}-2^{18}). This suggests that using the truncated pc as an index function to the value history table results in significant aliasing.

An interesting result is that single entry value history tables exceed the performance of larger tables, until the table size reaches 2^8 entries. A single history entry in the trace driven simulation environment captures complete global path information. Complete global path information is more useful than the incomplete global history that results from a small number of distinct history table entries. And because of aliasing in the multiple entry tables, the histories of individual instructions are also mixed together. Larger value history tables, however, reduce the effects of aliasing, and although they do not have as much global value information, they do have more accurate information per instruction. This provides overall higher prediction accuracies. Returns from using larger value context tables are diminished when value history tables become greater than 2^{14} entries.

The individual benchmark behavior is shown in Figure 5. Each bar corresponds to a different value history table size. The data shown correspond to the configuration with a value prediction table with 2^{20} entries. Several interesting observations can be made.

The prediction accuracy for most benchmarks is almost equal to the ideal. The exceptions to this are **compress** and **ijpeg**. For compress the performance is significantly higher than the "ideal" - almost by 10%.

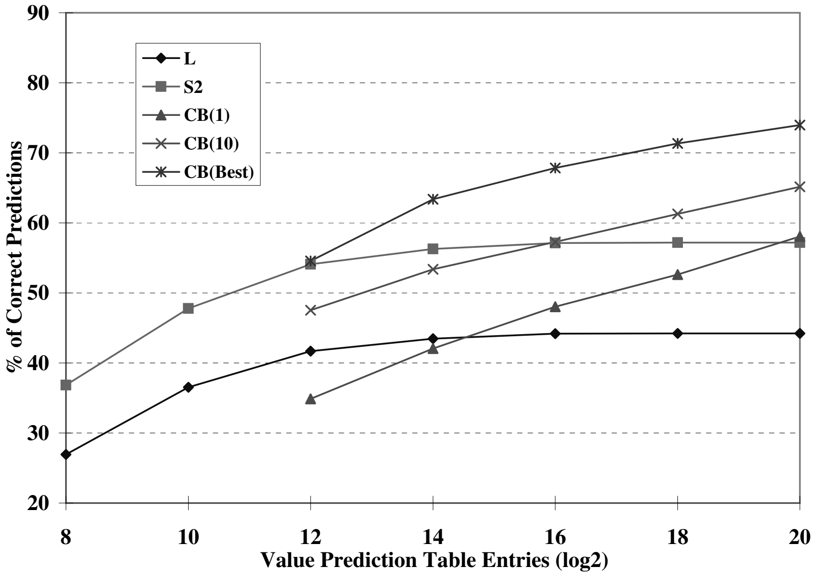

Figure 6: Computational and Context Based Predictors

This is attributed to the sharing of prediction information that is enabled by having a common value prediction table for all instructions - sharing was not considered for the ideal study. Sharing allows instructions to take advantage of information learned by other instructions and therefore not all instructions have to incur a start up learning time. Sharing however can be detrimental if the conflicting instructions do not share similar behavior. This could be one of the reasons for the misbehavior of ijpeg. One other reason for the discrepancy is that the ideal scheme better exploits the repetitive nature of the benchmark - jpeg processes the same data 35 times. An unbounded table can learn all required information through the first iteration, whereas a scheme with a bounded table to may need relearn during each iteration because of limited table space. To confirm this hypothesis we ran the both the ideal and bounded-size predictors for one iteration, and they provided comparable performance.

4.2 Computational and Context Based Prediction

The proposed context based predictor (**CB**) is compared against two computational predictors, last value (**l**) and stride (**s2**). The size of the prediction table is varied and the performance of the three schemes is compared. For context based prediction three curves are shown in Figure 6: a predictor with a single value history table entry, CB(1), a predictor with 2^{10} value history table entries,CB(10), and the best overall behavior for a given value prediction table size, CB(Best). The overall benchmark behavior is shown Figure 6. The results reveal several interesting points.

For small table sizes the computational predictors outperform the context based predictors. Overall, the stride predictor has better performance than the last value predictor and both reach their maximum performance with tables of 2^{14} or more entries. With larger table size the context based prediction increasingly outperforms the computational prediction. Higher performance is achieved when using large value context tables.

The above suggest that if high prediction accuracy is required context based prediction needs to be employed. However, if value prediction with large tables is infeasible, then either context predictors with smaller table requirements should be deviced or a hybrid scheme should be examined. In a hybrid approach a computational predictor, such as stride, can be used to capture the majority of the correct predictions while the context based predictor deals with the remaining predictions. Note that a hybrid approach may not necessarily translate to fewer value prediction table entries. This can be the case when the sequences captured by a computational predictor do not occupy many table entries.

4.3 Context Function and Table Utilization

Prediction accuracy is studied in terms of the three context functions discussed in Section 4.1. The results for the C and FC functions are shown in Figure 7 and for the FS function in Figure 8. The data in Figure 7 are for a 2^{16} entry value history table and a 2^{18} entry value prediction table. The data in Figure 8 are for different table configurations. The data show that folding, (FC), does better as compared to no folding, (C), and that the shift/xor function, (FS), outperforms the other two. The results suggest it is important to consider bits from the entire value to increase prediction accuracy. Another interesting observation is that, when using FS function, with increasing value prediction table size the fewer the values the better the prediction accuracy. This is probably due to the availability of more bits to encode recent history.

The hash function used, influences the utilization of the value prediction table entries. As a result it may be possible to devise functions that obtain same or better performance if the context function is improved (or conversely reduced table size requirements). To study the efficiency of the FS function we show in Figure 9 the table utilization for a 2^{20} value prediction table for gcc. The graph was obtained by sorting table entries in increasing order according to their utilization. This graph suggests that entries that are not accessed frequently do not contribute to prediction accuracy. Such entries account for the majority of the value prediction table. This suggests that schemes with fewer second level table entries may be able to obtain high accuracy, provided functions that can exploit them can be designed.

Considering the enormous design space for context functions and their effect on prediction accuracy and cost, it is essential to explore them in detail. One interesting possibility is to identify when an address prediction is made and use a context function that accounts for the tendency of higher order bits to remain unchanged.

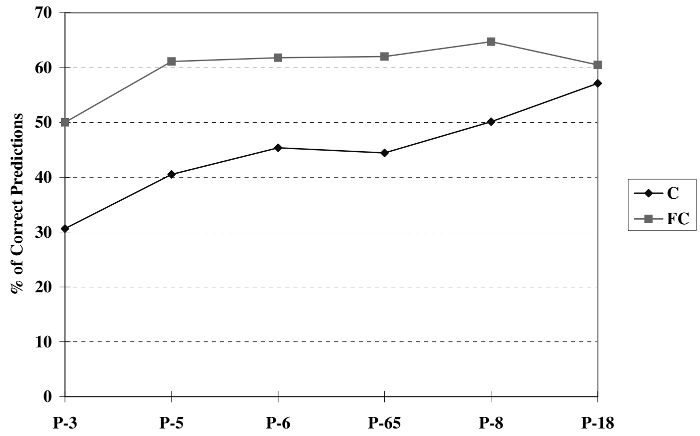

Figure 7: Prediction Using the C and FC Functions

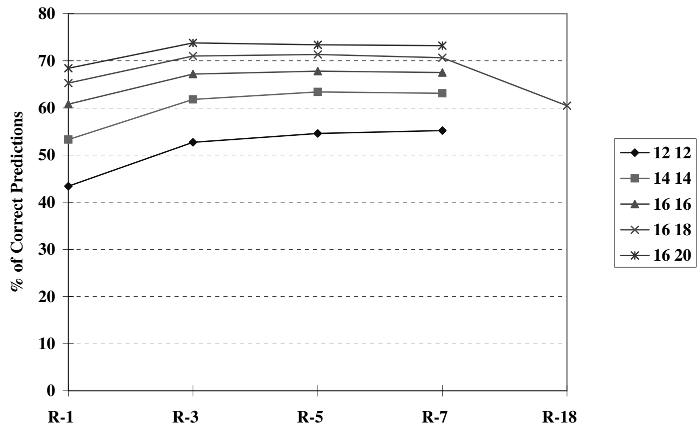

Figure 8: Prediction Using the FS Function

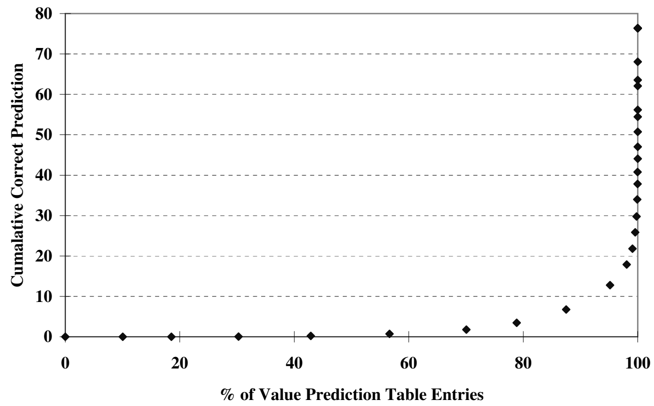

Figure 9: Value Prediction Table Utilization for gcc

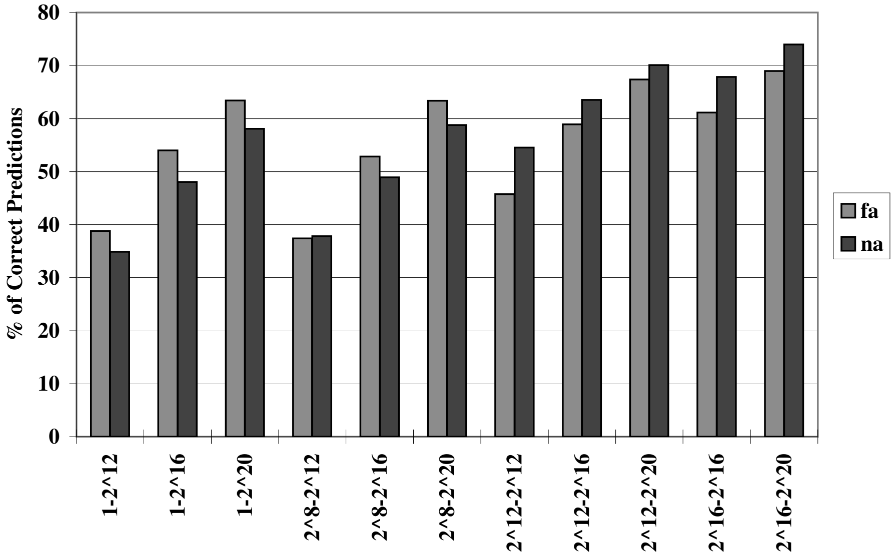

Figure 10: Predictability with and without Combining the PC

4.4 Combining Function

Combining functions based on the program counter were considered . The simulated function xors the truncated address of the predicted instruction with the output of the context function. The results with (**fa**) and without combining (**na**)are shown in Figure 10.

The results suggest that configurations with small value context tables can benefit significantly from such an approach (as much as 7%). Combining the address of a predicted instruction into the index that accesses the prediction table helps identify the predictions associated with an instruction more accurately. This reduces/increases the aliasing/utilization in the prediction table. However as tables get bigger address combining results in an increase in aliasing, and it becomes better to access the prediction table with no combining.

Experiments were also conducted using combining of global branch history. This approach resulted in worst prediction accuracy. This is attributed to the increase in the number of value prediction table entries that an instruction references, resulting in more conflicts and slower learning. Another combining function that was simulated but had low performance was one that xors only $\log_2$(Size of Value History Table) address bits with the context function.

4.5 Sensitivity to Input Data Size

The sensitivity to the size of the data input was examined for two benchmarks. The input files and their runtime characteristics of the two benchmarks are given in Table 3. The benchmarks were run for a variety of prediction configurations and the obtained results for the two benchmarks are shown in Figures 11-12. The results for gcc show a higher accuracy with increasing size, whereas for compress a decrease can be observed. The variation in both cases, however, was not significant. The observed invariance can be

gcc		compress	
File	Predictions (mil)	File	Predictions (mil)
jump.i	106	sm: 30000 e 2231	5.8
gcc.i	137	med: 100000 e 2231	18.2
stmt.i	372	lar: 1000000 e 2231	183

Table 3: Different Input Files for gcc and compress

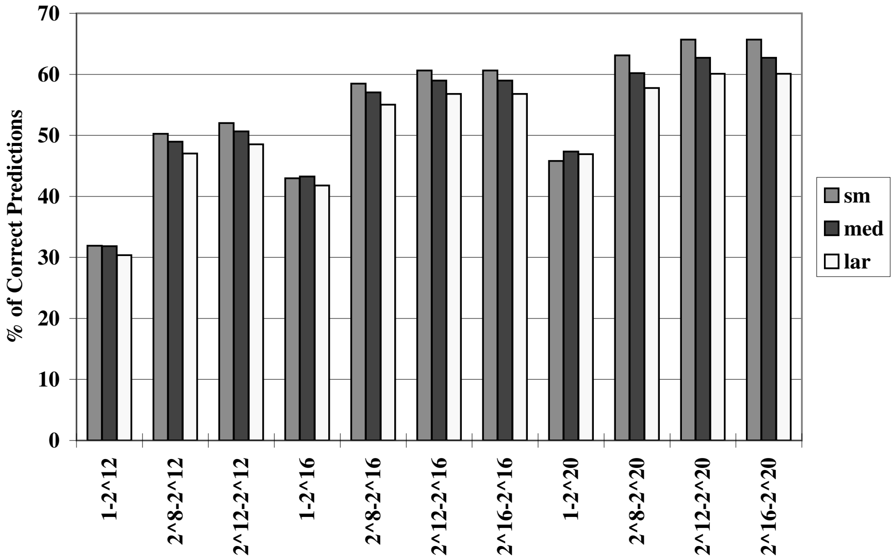

Figure 11: Sensitivity of compress to different inputs

attributed to the way data are processed in programs. Data can be processed once and either never referenced again or referenced again after some time period. Depending on whether the period between references to the same data item is short or long, the information acquired for the particular data may be maintained or evicted from the predictor. If the period between references is a function of the data set size then it should be expected that prediction information will be evicted more frequently with larger data sets. Neither gcc nor compress did exhibited such behavior.

5 Summary and Conclusions

In this paper we studied a two level implementations of context based value predictors. The new elements introduced by this study are bounded size tables and sharing of prediction information among instructions. A number of issues unique to context based prediction are considered. Trace driven simulation was used to study prediction accuracy. The results suggest accuracies that approximate the "ideal" can be obtained with the use of large tables. For small table sizes the proposed predictor has inferior performance compared with equal size computational (last value and stride) predictors. However, with increasing size the prediction

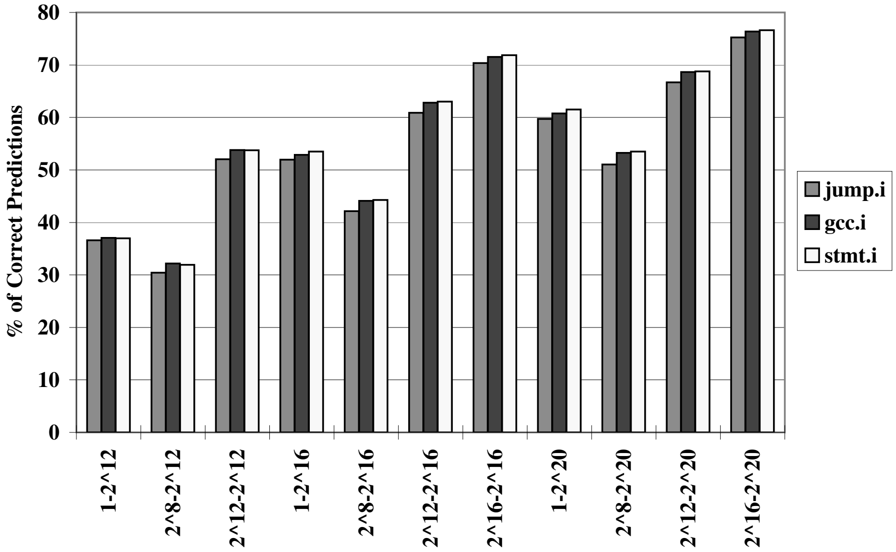

Figure 12: Sensitivity of gcc to different inputs

accuracy of computational predictors saturates and two level value predictors provide higher accuracy. It is known that computational predictors can predict a large fraction of the predictions captured by context based prediction [2]. This indicates that a hybrid scheme, using stride and context base prediction, may be useful in maintaining high prediction accuracies and reducing the table size requirements of context based prediction.

Our experimentation demonstrated that context functions that consider bits from the entire value have better performance as compared to functions that do not. The utilization of the table resources with the use of the FS hashing function was shown to be non-uniform. Many entries in large tables are not used or used very little. These entries contribute little (if any) to the overall predictability. This suggests that mechanisms, such as improved hash functions that use the table more efficiently should be considered. Such mechanisms can potentially result in significant reductions in table sizes while maintaining high prediction accuracy.

Results were obtained for a combining function that xors the address of the predicted instruction with the output of the context function. Better prediction accuracies, as compared to not combining, were observed for small tables. However with increasing table sizes combining was detrimental to the performance.

A study of the sensitivity of two benchmarks to different input data sizes suggests that larger input data size may not result in worse prediction performance.

It is safe to say that this study raises more questions than it answers, however we believe it represents a first step in the challenging process of designing high accuracy value predictors.

6 Acknowledgements

This work was supported in part by NSF Grants MIP-9505853 and MIP-9307830 and by the U.S. Army Intelligence Center and Fort Huachuca under Contract DABT63-95-C-0127 and ARPA order no. D346. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U. S. Army Intelligence Center and Fort Huachuca, or the U.S. Government.

The authors would like to thank Stamatis Vassiliadis for his helpful suggestions and constructive critique while this work was in progress.

References

[1] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and data speculation," in *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 138–147, October 1996.

[2] Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, December 1997.

[3] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text Compression*. Prentice-Hall Inc., New Jersey, 1990.

[4] A. Mendelson and F. Gabbay, "Speculative execution based on value prediction," Tech. Rep. (Available from http://www-ee.technion.ac.il/fredg), Technion, 1997.

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

[6] T. F. Chen and J. L. Baer, "Effective hardware-based data prefetching for high performance processors," *IEEE Transactions on Computers*, vol. 44, pp. 609–623, May 1995.

[7] S. Mehrotra and L. Harrison, "Examination of a memory access classification scheme for pointer intensive and numeric programs," in *Proceedings of the 10th International Conference on Supercomputing*, May 1996.

[8] R. J. Eickemeyer and S. Vassiliadis, "A load instruction unit for pipelined processors," *IBM Journal of Research and Development*, vol. 37, pp. 547–564, July 1993.

[9] J. Gonzalez and A. Gonzalez, "Speculative execution via address prediction and data prefetching," in *Proceedings of the 11th International Conference on Supercomputing*, pp. 196–203, July 1997.

[10] T.-Y. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," in *Proceedings of the 19th International Symposium on Computer Architecture*, pp. 124–134, May 1992.

[11] R. Nair, "Dynamic path-based branch correlation," in *Proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 15–23, December 1995.

[12] Q. Jacobson, S. Bennet, N. Sharma, and J. E. Smith, "Control flow speculation in multiscalar processors," in *Procedings of the 3rd International Symposium on High-Performace Computer Architecture*, pp. 218–229, February 1997.

[13] P.-Y. Chang, E. Hao, and Y. N. Patt, "Target prediction for indirect jumps," in *Proceedings of the 24th International Symposium on Computer Architecture*, pp. 274–283, June 1997.

[14] I.-C. K. Cheng, J. T. Coffey, and T. N. Mudge, "Analysis of branch prediction via data compression," in *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[15] T.-Y. Yeh and Y. N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," in *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 257–266, May 1993.

[16] S. McFarling, "Combining branch predictors," Tech. Rep. DEC WRL TN-36, Digital Western Research Laboratory, June 1993.

[17] D. Burger, T. M. Austin, and S. Bennett, "Evaluating future microprocessors: The simplescalar tool set," Tech. Rep. CS-TR-96-1308, University of Wisconsin-Madison, July 1996.