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

January 31, 1998

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, the goal of the earlier study was to determine the limits of value predictability using unbounded tables.

In this paper we discuss implementations of context-based value predictors using two level table organizations. The important new elements introduced in this paper are fixed size tables and sharing of prediction information among instructions. Some other dimensions of context-based prediction - such as context order, aliasing and hash functions - are also examined. A variety of implementations of the proposed value predictor are evaluated for SPECINT95 benchmarks by comparing their prediction performance to the "ideal" performance with unbounded tables. Such predictors obtain accuracies that approach that of "ideal" predictors. Aliasing is found to be a significant cause for misprediction. Aliasing is mitigated by using tables that capture accurate value history per instruction. Prediction rates for SPECFP95 benchmarks are evaluated and found to often exceed 90% accuracy. The proposed predictor is 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 is 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 deterministically computed information. The success of a speculative approach depends highly on the accuracy with which program information is predicted. Although just about any program information can potentially be predicted, conditional branch outcomes are the only example that has been studied extensively. Empirical evidence suggest that data values is another dominant information type that should be predicted for higher performance[1, 2, 3, 4, 5].

It would at first 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, 3, 6, 7]. 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 [7] 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 [7] 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 [7] 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 similar to the ones used for branch prediction[8]. 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[9, 10]. A variety of implementations of the proposed value predictor are evaluated by comparing prediction performance with the "ideal" performance reported in [7]. A number of issues relevant to context-based prediction such as order, aliasing and hash functions are investigated. 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 [7]. 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 quickly learn to predict correctly, and can even predict values not

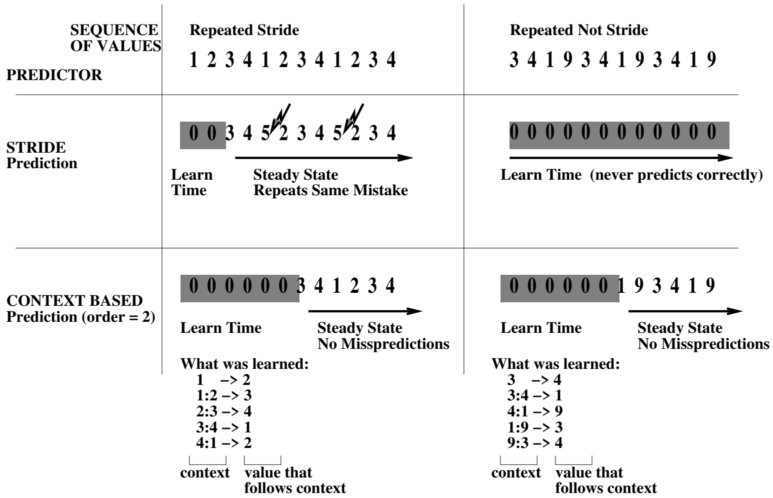

Figure 1: Differences between Stride and Context-Based Value Prediction

previously 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 [3] and last value prediction was used in [1, 2].

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.

Figure 1 compares the performance of a stride predictor against a context-based predictor when predicting a repeated stride sequence and a repreated non-stride sequence. The stride predictor, following a short start-up learn time, starts predicting correctly the stride sequence, however, it repeats the same misprediction each time the sequence begins. The stride predictor fails to predict correctly any value in the non-stride sequence. In comparison, the context-based predictor requires longer learning time - it needs to observe a sequence at least once before it can start predicting it correctly - but can be highly accurate for *any* repeating sequence.

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 in [1], 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. Last value prediction was used to predict load values and in a subsequent work to predict all values produced by instructions[2]. 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 [11, 12]. They have also been proposed for speculative execution of load and store instructions [13, 14, 15]. Stride prediction was proposed for value prediction in [3], and its accuracy and performance potential were compared against last value prediction.

A two level value predictor was proposed in [6]. The basic two level scheme maintains n unique values per program counter (**PC**). The selection among the different values is based on a unique index associated with each value (if 4 values are maintained a 2 bit index is used). A history of p such indices, maintained per PC, is used to access another table containing counts that indicate which of the values should be selected. The main limitations of the proposed scheme is that it can only exploit locality of up to four unique values, uses inefficiently a table entry when an instruction generates less than four unique values, and it can not share prediction information between different instructions. Consequently, prediction accuracy is limited, since instructions very often have value locality that exceeds four[2]. Fundamentally the proposed scheme may be limited to having few unique values per PC because an associative search is likely to be required to establish if the actual value is in the list of values maintained per PC. An interesting extension that may overcome partially some of the above limitations is the use of hybrid predictors[6].

Value prediction can draw from a variety of work on the prediction of control dependences[16]. 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 [8, 17, 18, 19, 20]. 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 [10]. 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 [18, 19, 20]. 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 [7] 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 lower order bits.

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.

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 via 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 or other VPT and/or processor state information.

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.

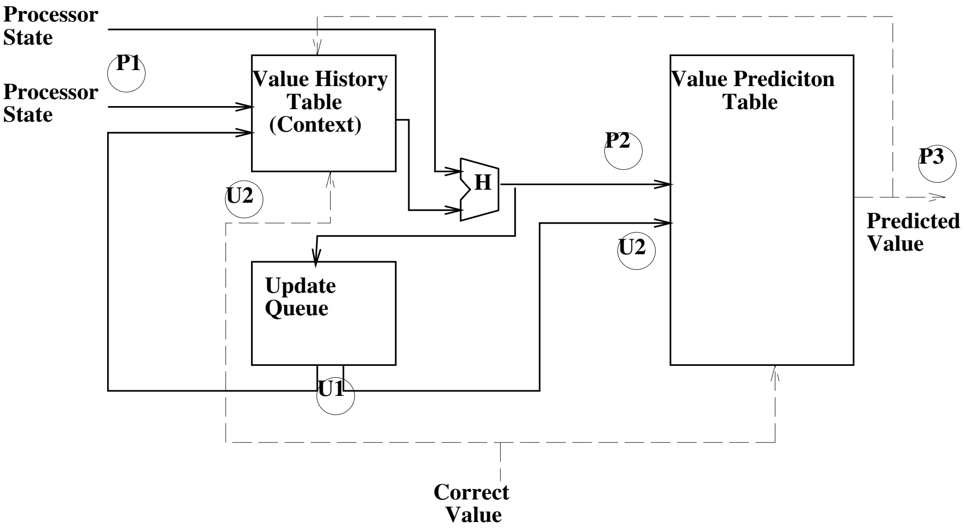

Figure 2: Generic Two Level Table for Value Prediction

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 [21]. 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 [18, 19, 20]. 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[22]. Prediction confidence can also be associated with a value prediction table entry. There is a subtle difference between the two choices, using the value history table gives confidence for a specific instruction, whereas using the value prediction table gives confidence following a specific context. The decision for using either scheme may be dependent on the size of the value history table; possibly large tables will favor the former and small the latter.

Update confidence can be used to eliminate start-up updates that may pollute the value prediction table. An example policy would be to not update the value prediction table from a value history entry that has not been referenced more times than the number of values in the context(order). The start-up confidence can be implemented as a state machine that determines when to update the value prediction table.

A VHT table entry may also contain tag information aiming to reduce mispredictions due to aliasing by not using/updating a prediction when there is not a tag match. The tag replacement can also be guided by a state machine.

Value History Table Size: for branches it is known that high accuracy can be attained even with a single value history table entry (global history registers) [8, 23]. This is the case because branch values are single bits 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 (context aliasing). There are at least two approaches, and in some combination they can be used to alleviate this problem:

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

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 could 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. Figure 3 shows an example of the FS function being used to form an index of 5 bits with a context of three values. The

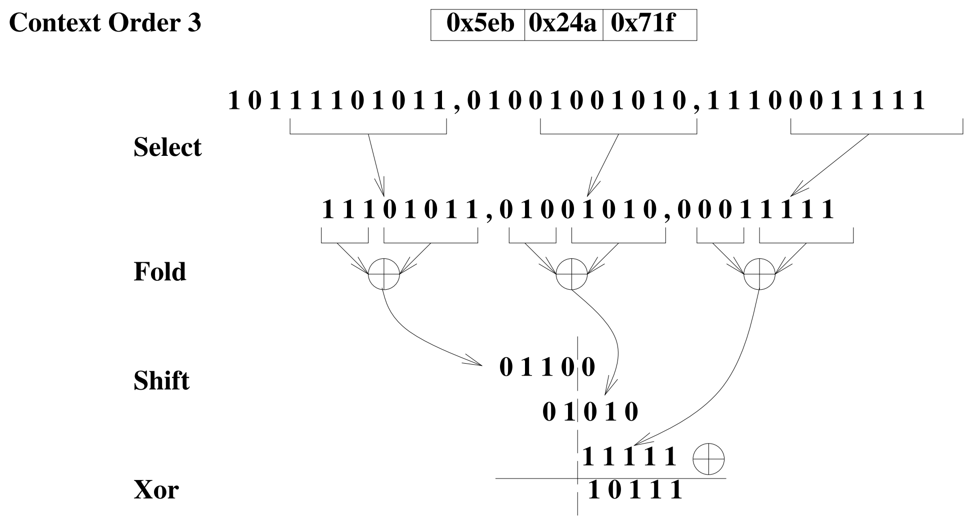

Figure 3: *Select Fold Shift Xor* Function

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 PC address of the predicted instruction, branch history etc. An effective operation for combining all the information is the xor function[23].

Value Prediction Table Size: the table size is limited by cost and access latency, 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 may 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 is to maintain two predictions *p1* and *p2*, known as *2-delta*[13]. 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 when a new value matches the value in p2[13].

If more than one prediction is maintained per entry then it will be necessary to have a voting scheme to determine which prediction should be used. 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 halve all counters in the same context [9].

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 (1,2,3,5,7,10),
- Value history table entries (1, 16, 256, 1K, 4K, 16K, 64K),
- Context function (C, FC, PFS (defined below), FS),
- Combine function (identity, truncated PC), and
- Value prediction table entries (1K, 4K, 16K, 64K, 256K).

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 16 and 18 bits long (for 2^{16} and 2^{18} entry value prediction tables). 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). The PFS is defined similarly to FS with the exception that its R function uses t bits from each value instead of 32.

A value history table entry contains the context values. An entry in the value prediction table contains a single prediction and a replacement finite state machine. The replacement mechanism is a 2 bit saturating

C and FC Functions for 64K and 256K VPT		
Order	C	FC
1	R(16)	R(32)F(16)
2	R(8,8)	R(32,32)F(8,8)
3	R(6,5,5)	R(32,..,32)F(6,5,5)
5	R(4,3,3,3,3)	R(32,..,32)F(4,3,3,3,3)
7	R(4,2,2,2,2,2,2)	R(32,..,32)F(4,2,2,2,2,2,2)
Order	C	FC
1	R(18)	R(32)F(18)
2	R(9,9)	R(32,32)F(9,9)
3	R(6,6,6)	R(32,..,32)F(6,6,6)
5	R(6,3,3,3,3)	R(32,..,32)F(6,3,3,3,3)
7	R(6,2,2,2,2,2,2)	R(32,..,32)F(6,2,2,2,2,2,2)
FS Context Function for 2^t VPT		
Order	k	FS
1	$\lfloor t/1 \rfloor$	R(32,..,32)F(t,..,t)S(0)
2	$\lfloor t/2 \rfloor$	R(32,..,32)F(t,..,t)S(0,t-k)
3	$\lfloor t/3 \rfloor$	R(32,..,32)F(t,..,t)S(0,t-2k,t-k)
5	$\lfloor t/5 \rfloor$	R(32,..,32)F(t,..,t)S(0,..,t-2k,t-k)
7	$\lfloor t/7 \rfloor$	R(32,..,32)F(t,..,t)S(0,..,t-2k,t-k)
10	$\lfloor t/10 \rfloor$	R(32,..,32)F(t,..,t)S(0,..,t-2k,t-k)

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%)	
applu	5x5x5 grid	60	55 (92%)	test input
fpppp	4 atoms	150	125 (83%)	test input
mgrid	32x32, 8 itter	119	114 (96%)	test input
swim	256x256 grid	200	177 (88%)	test input

Table 2: Benchmarks Characteristics

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 2. Updates to the value history table always 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, all the above suggests, that value prediction has an enormous design space that we roughly explored.

For last and stride value predictors we only varied the number of entries in the prediction table. A replacement in the two predictors is guided by a 2 bit saturating counter similar to the context-based predictor.

Trace driven simulation was used to evaluate the different configurations of the various predictors. The SPECINT95 and SPECFP95 benchmarks shown in Table 2 were simulated using the simplescalar tool [24]. The benchmarks were compiled with -O3 optimization.

In the results section unless indicated otherwise the following default parameters for context-based prediction are used, order-5 FS context function and the combine function is identity. For instructions producing 64 bit results - due to simulation platform limitations - a separate but identical predictor is used to predict the most significant 32 bits. A 64 bit prediction is correct if all 64 bits are correct. In the results we often refer to a specific context-based configuration using a triple **x-y-z**, the first number represents the $\log_2$(VHT entries), the second number the $\log_2$(VPT entries) and the third number to the predictor order. When a configuration is described as using global value history is equivalent to saying that is using a single VHT entry. Whenever "ideal" results are used they correspond to the results reported for an order-3 unbounded context-based predictor in[7]. For averaging we used arithmetic mean, so each benchmark effectively contributes the same number of total instructions. Unless indicated otherwise overall results are only for the SPECINT95 benchmarks.

4 Results

4.1 Two Level Context-Based Prediction

The results in Figure 4 show prediction accuracy averaged over all benchmarks. Each curve corresponds to a different value prediction table size, from 2^{10} to 2^{18}. An "ideal" line is drawn for comparison purposes.

The results show that prediction accuracy can approach the ideal with bounded table sizes. Although some of the tables are large, they are not impossibly large, and all value-producing instructions are predicted. If a subset of instructions were predicted/updated, as could be the case in a real application, then undoubtedly smaller tables could be used. An example of this will be to have tagged VHT entries that are allocated after is learned that a prediction for an instruction would have enabled earlier execution. 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

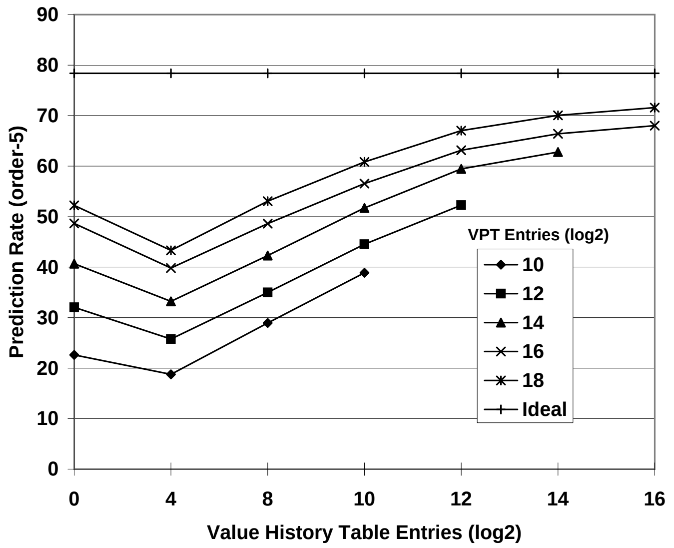

Figure 4: Predictability using Context-Based Predictor

if additional engineering is applied.

The data indicate it is important to have both large value history and value prediction tables. For instance, large value prediction tables with small history tables do worse than smaller value prediction tables and larger history tables (compare configurations 2^{10}-2^{16} and 2^{12}-2^{14}). 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 (local) 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. A possible direction of future research can be aimed to identify predictors that can provide accuracy similar to large VHT tables (local value history) with cost not much higher than a global value history table.

The individual benchmark behavior is shown in Figure5. Each graph contains the results for a different size VPT table. Each bar corresponds to a specific value history table size. Also, each set of results includes the prediction accuracy when an alias-free VHT table is used (denoted by AF) . This is useful in determining the prediction potential when there is no aliasing. Several interesting observations can be made.

The prediction accuracy for most benchmarks, with increasing VPT size, is approaching the ideal. The exceptions to this are **compress** and **ijpeg**. For compress the performance is significantly higher than the

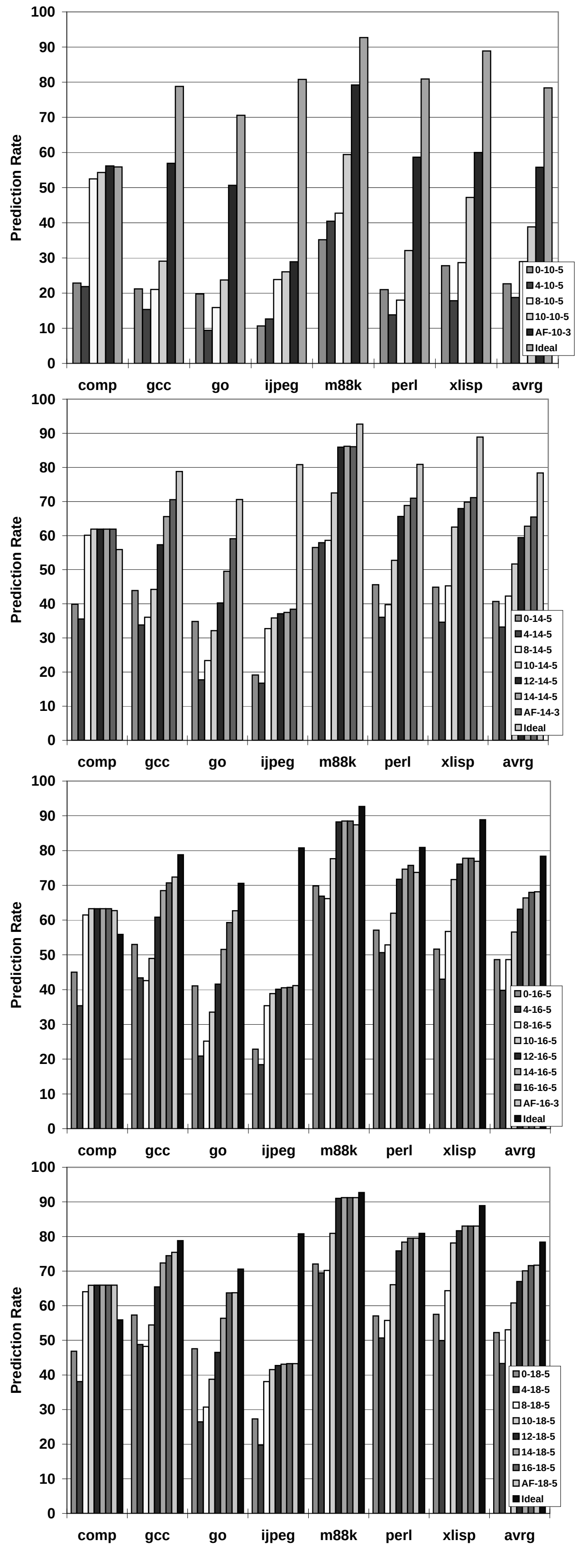

Figure 5: Overall Benchmark Predictability for 2^{10}-2^{18} VPT

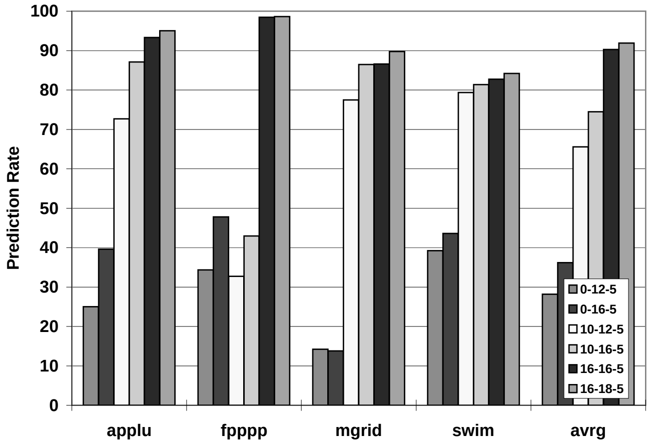

Figure 6: Overall Benchmark Predictability for Floating Point Benchmarks

"ideal" - almost by 10%. 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 may need to relearn during each iteration because of limited table space. To confirm this hypothesis we ran both the ideal and bounded-size predictors for one iteration, and they provided comparable performance.

Another important observation from the data is that aliasing in the value history table is a significant source of misprediction. In particular, an alias-free value history table provides dramatic prediction improvement for smaller value prediction tables. This underlines the importance of having per PC history and hence the need for large VHT tables. The benefits for large value prediction tables are significant but not as pronounced because longer context can identify the next prediction more accurately. Also, we can observe that a VHT table with 2^{14} entries or more it approximates the performance of an alias-free value history table.

Figure 6 shows the prediction accuracy for floating point benchmarks for a variety of predictor configurations. A comparison to the prediction of integer benchmarks, indicates that the floating point benchmarks are more predictable than integer benchmarks. Otherwise, the overall observations made for integer benchmarks appear to hold also for floating point benchmarks.

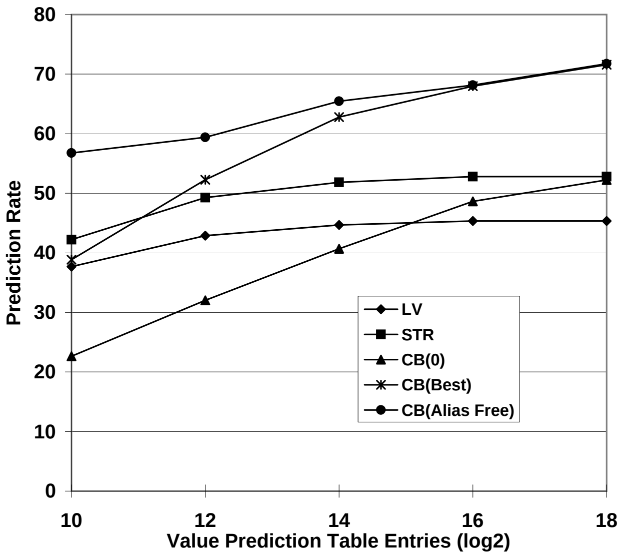

Figure 7: Computational and Context-Based Predictors

4.2 Computational and Context-Based Prediction

The proposed context-based predictor (**CB**) is compared against two computational predictors, last value (**LV**) and stride (**STR**). The overall benchmark behavior is shown Figure 7. 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 7: a predictor with a single value history table entry, CB(0), a predictor with the best overall behavior in the simulations for a given value prediction table size and order-5, CB(Best), and a predictor with alias-free value history table, CB(Alias Free). 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 increasing value prediction table size the context-based prediction increasingly outperforms the computational prediction. Higher performance is achieved when using large value history tables. Another important results is that global value history configurations were outperformed by the stride predictor. Subsequently we will describe how to improve global history performance.

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 alternative context predictors with smaller table requirements should be devised 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 smaller value prediction tables. This can be the case when the sequences captured by the computational predictor did not occupy many value prediction table entries.

4.3 Context Function and Table Utilization

Prediction accuracy is studied in terms of the context functions discussed in Section 1. The results for the C, FC, PFS and FS functions are shown in Figure8. The results can be divided according to the value history table used into global (2^0 entry VHT) and local (2^{16} entry VHT).

The data for all configurations show that the functions that consider bits from the entire value outperform the functions that only use a subset of them (FC and FS versus C and PFS). Another interesting observation is that, with global value history FC outperforms the FS function however with local history the opposite is true. The data also suggest that for local history tables the FS functions have the best overall performance. Another interesting observation is that increasing order results in higher prediction accuracy. However, the prediction accuracy levels off, or even deteriorates, for large orders. This can be attributed to increase aliasing when hashing larger number of values.

To study the efficiency of the FS context function we show in Figures 9-11 the cumulative total prediction and cumulative normalized correct prediction for different predictor configurations over all integer benchmarks. Each graph was obtained by sorting VPT table entries in decreasing order according to their utilization. This graph suggests that entries that are not accessed frequently do not contribute to total prediction and even less to the prediction accuracy. Such entries account for the majority of the value prediction table. This suggests that schemes with fewer value prediction table entries may be able to obtain the same accuracy (or conversely attain higher prediction with the same table size), 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.

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 (**xor PC**) and without combining (**Identity**) are shown in Figure 12.

The results suggest that configurations with small value history tables can benefit significantly from such an approach. Combining the PC 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. It is noteworthy that with combining the global value history outperforms the stride prediction for larger VPT tables (see Figure 7). However as value history 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

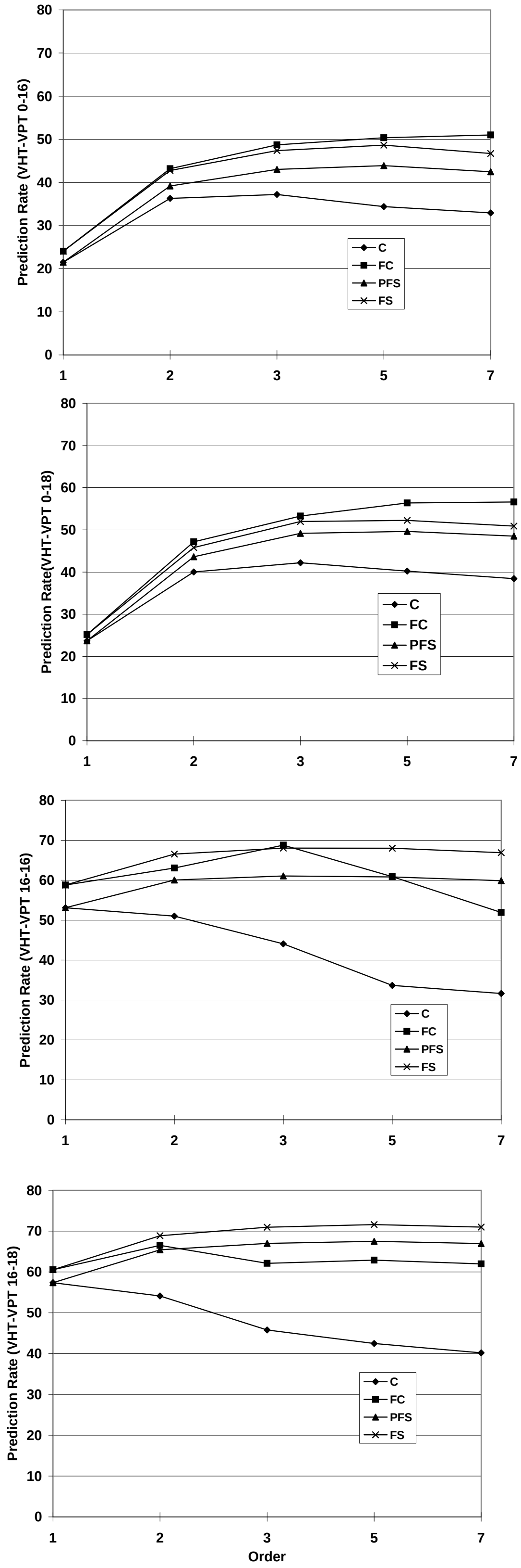

Figure 8: Prediction Accuracy for Different Context Functions, Predictor Configuration and Order

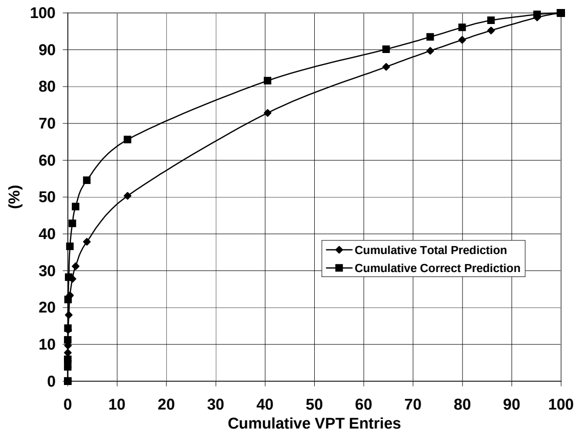

Figure 9: Value Prediction Table Utilization for 12-12 Configuration

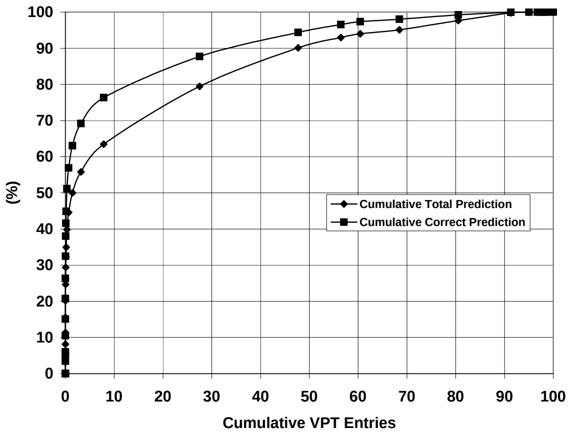

Figure 10: Value Prediction Table Utilization for 16-16 Configuration

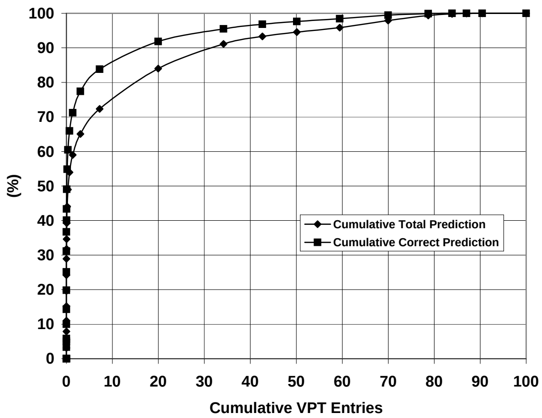

Figure 11: Value Prediction Table Utilization for 16-18 Configuration

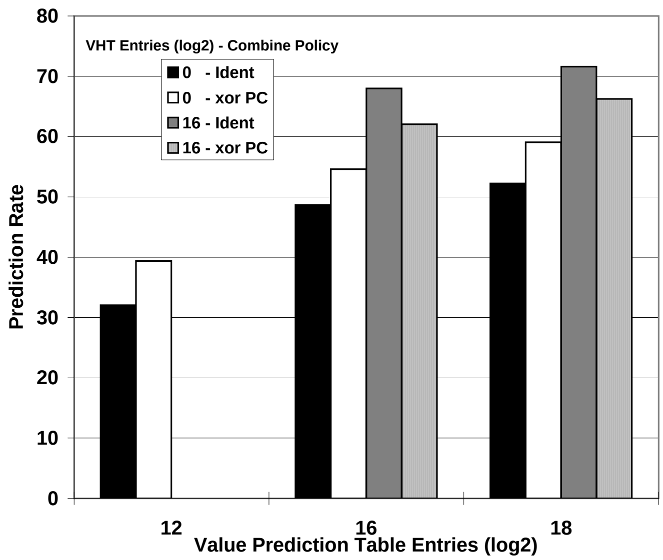

Figure 12: Predictability with and without Combining the PC

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 case that was studied - but had low performance - was the combining of $\log_2$(VHT entries) address bits.

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 13-14. 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 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 exhibited such behavior, however.

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

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

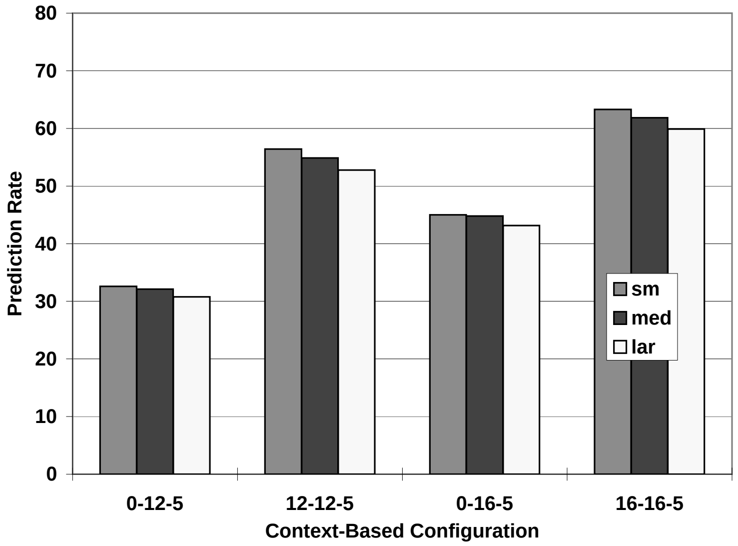

Figure 13: Sensitivity of compress to different inputs

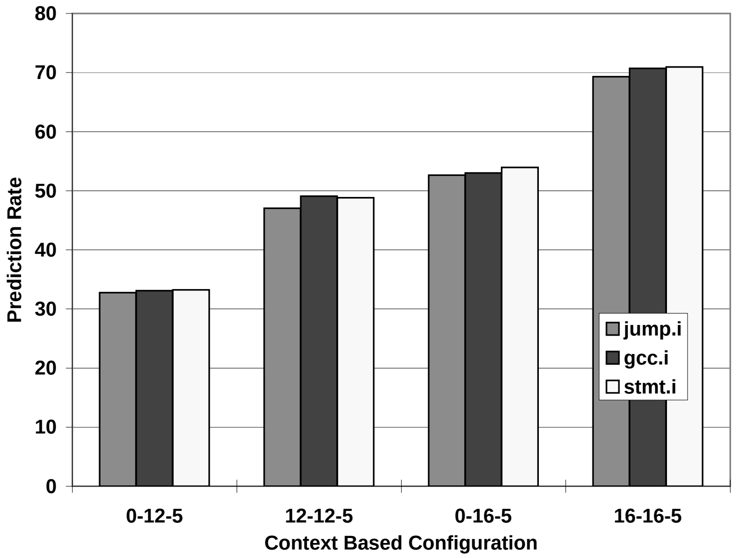

Figure 14: Sensitivity of gcc to different inputs

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 accuracy of computational predictors saturates and context-based value predictors provide higher accuracy. It is known that computational predictors can predict a large fraction of the predictions captured by context-based prediction [7]. 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 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] 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.

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

[4] M. H. Lipasti and J. P. Shen, “Superscalative microarchitecture for beyond ad 2000,” *Computer*, September 1997.

[5] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. E. Smith, “Trace processors,” in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, December 1997.

[6] K. Wang and M. Franklin, “Highly accurate data value prediction using hybrid predictors,” in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, December 1997.

[7] 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.

[8] T.-Y. Yeh and Y. N. Patt, “Two-level adaptive branch prediction,” in *Proceedings of the 24th Annual ACM/IEEE International Symposium on Microarchitecture*, November 1991.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] J. E. Smith, “A study of branch prediction strategies,” in *Proceedings of the 8th International Symposium on Computer Architecture*, pp. 135–148, May 1981.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] E. Jacobsen, E. Rotenberg, and J. E. Smith, “Assigning confidence to conditional branch predictions,” in *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 142–152, December 1996.

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

[24] 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.