Abstract

BODINE, JILL THERESE. Exploiting Computational Locality in Global Value Histories. (Under the direction of Thomas M. Conte.)

Value prediction is a speculative technique to break true data dependencies by using history to predict values before they are computed. Previous research focused on exploiting two types of value locality (computation-based and context-based) in the local value history, which is the value sequence produced by the same instruction that is being predicted. Besides local value history, value locality also exists in global value history, which is the value sequence produced by all dynamic instructions according to their execution order. In this thesis, a new type of value locality, computational locality in global value history is studied. A prediction scheme, called gDiff, is designed to exploit one special and most common case of this computational model, the stride-based computation, in global value history. Experiments show that there exists very strong stride type of locality in global value sequences and ideally the gDiff predictor can achieve 73% prediction accuracy for all value producing instructions without any hybrid scheme, much higher than local stride and local context prediction schemes. However, the ability to realistically exploit locality in global value history is greatly challenged by the value delay issue, i.e., the correlated value may not be available when the prediction is being made. The value delay issue is studied in an out-of-order (OOO) execution pipeline model and the gDiff predictor is improved by maintaining an order in the value queue and utilizing local stride predictions when global values are unavailable to avoid the value delay problem. This improved predictor, called hgDiff, demonstrates 88%

accuracy and 69% prediction coverage on average, outperforming a local stride predictor by 2% higher accuracy and 13% higher coverage.

EXPLOITING COMPUTATIONAL LOCALITY IN GLOBAL VALUE HISTORIES

by
JILL THERESE BODINE

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER ENGINEERING

Raleigh

2002

APPROVED BY:

_____ _____

_____

Chair of Advisory Committee

Biography

Jill T. Bodine received a Bachelor of Science degree in Computer Science from Michigan State University in 1992. She was employed from 1992-1998 with IBM's Networking Software Division and from 1998-2000 with Ericsson's Cellular Phone Division, both in Research Triangle Park, North Carolina. Jill's research was performed while she was a member of the Tinker group in the Center for Embedded Systems Research at North Carolina State University.

Acknowledgements

Special acknowledgements are given to Huiyang Zhou and Dr. Tom Conte for their input and assistance in pursuing this research and in initial versions of this thesis submitted to prior conferences.

Table of Contents

List of Figures

List of Equations

Chapter 1. Introduction

1.1. Motivation

Value prediction is a promising way to break true data dependencies in a program. In ILP (instruction level parallel) processing, value prediction can be used to avoid pipeline stalls by introducing more (speculatively) independent instructions. In TLP (thread level parallel) processing, it can be used to support speculative multithreading. However, in spite of its promising performance potential, the benefits of value prediction are quickly offset by the high cost of misprediction. Therefore, a highly accurate value predictor is required along with an accurate confidence mechanism for value prediction to reach viability. Since value predictors are impacted by cache misses and variations in control flow, the prediction accuracy of state-of-the-art value predictors has not been good enough to win commercial implementation. In this thesis, a new type of value locality is studied and two new value predictors, *gDiff* and *hgDiff*, are proposed to improve prediction accuracy.

Similar to branch prediction techniques, value prediction methods exploit localities in the value history to achieve high prediction accuracy. During program execution, two types of value history, *local value history* and *global value history*, can be exploited to predict the value of an instruction. Local value history is the value sequence produced by prior executions of the *same instruction* that is being predicted. Global value history, on the other hand, is the value sequence consisting of the values produced by *all the dynamic instructions* according to the order of the completion of their execution. If the length/order of the global value history is large enough, global value history can encompass the local value history.

Value locality can be broadly classified as *computation-based* (i.e., the prediction is a function of prior values) or *context-based* (i.e., value patterns are detected and the prediction is based on the next value in the prior pattern).

However, there is one issue that limits the exploitation of value locality in *global* value history (both computational and context-based): the *value delay*. Value delay is defined as the number of values in the dynamic instruction stream that are not available for use in predicting an instruction due to pipeline latency. Value delay affects the prediction accuracy of the gDiff predictor significantly by forcing the predictor to generate a prediction based on instructions that are far away in the dynamic instruction stream. Based on profile runs, the accuracy of the gDiff predictor drops to 52% (from 73%) when a value delay of 16 is assumed (i.e., the current prediction can not utilize the values that are produced by the previous 16 value-producing instructions, due to pipeline latencies).

One way to reduce the value delay problem in an out-of-order (OOO) pipeline model is to use speculative values once they are available from the execution stage, instead of waiting for the values to be committed in program order. Utilizing the speculative values helps to reduce but not eliminate value delay since pipeline latency still exists. Further, the resulting global value sequence is out-of-order and may vary from one instance to the next due to cache misses and branch mispredictions. Such variations make it difficult for a global predictor such as *gDiff* to identify the previous instruction from which to base a value prediction.

To solve the execution variation impact on the global value sequence, a new hybrid approach, called *hgDiff*, is proposed based on the *gDiff* predictor. In this scheme, the global value sequence is constructed in the order of instruction dispatch. Since execution

results of the instructions are not available when they are dispatched, the global value sequence is constructed using the local stride predictions and is updated later with the instruction-generated values at the write back stage. As the local stride predictor and the gDiff predictor exploit different types of value locality, the *hgDiff predictor* using a confidence mechanism produces high prediction accuracy (88%) and prediction coverage (69%)

1.2. Prior Work

There has been significant study on the potential of value prediction as a means to increase instruction level parallelism and avoid pipeline stalls caused by data dependencies [1, 2, 4, 7, 17]. In addition, value prediction can be used to eliminate inter-thread dependencies and enable efficient thread-level parallelism [8, 16, 21]. Value prediction can be exploited in superscalar models and with compiler supported during instruction scheduling [9, 10]. Due to the cost of re-executing speculative instructions after misprediction, mechanisms have been designed to select highly predictable instructions that will reduce the critical path of execution [14] and minimizes the instances of recovery. Using control flow information to predict values has provided increased value predictor accuracy [20]. The fact that 75% of instructions are redundant in results produced motivates studies comparing value prediction to instruction reuse [22].

Most previously proposed value predictors exploit locality in *local* value history. These predictors include last value predictor [1], last N-value predictor [11], stride predictor [1, 2, 4, 5], and context predictor [3, 6, 12]. Hybrid predictors combine both

computational and context predictors to exploit both types of localities to achieve higher prediction accuracy [3, 6].

Compared to local value history, locality in global value history is less thoroughly studied. The PI (previous instruction) based predictor [13] was proposed to explore the locality between two adjacent instructions in the dynamic instruction stream (i.e., the global value history). It may be viewed as a first-order global context-based predictor.

Previous research [3] based on local value history has shown that there are two models of value locality existing in the value sequence: the computational model and the context-based model. Most proposed value predictors exploit one (e.g., stride predictor and FCM predictor) or both locality models (e.g., DFCM predictor and hybrid predictor) in local value history. Also, the local value history can be further fine-tuned using the control-flow (or path) information [13]. Those predictors perform well in predicting many instructions when the value sequence produced by those instructions (i.e., local value histories) shows strong stride-type or periodic types of value patterns [2, 3, 4, 5, 6, 11, 12, 13, 18, 19].

In this thesis, *computational locality in the global value history* is studied. A new predictor scheme, the *gDiff predictor*, is proposed to exploit one special and common case of this computational locality based on stride. Experiments show that there exists very strong stride based locality in value sequences and many instructions that are hard-to-predict using local history based predictors become highly predictable. Ideally exploiting the global stride type value locality using the gDiff predictor can produce average prediction accuracy as high as 73% when predicting all the value producing

instructions without any hybrid scheme, while the local stride predictor shows 57% accuracy and the local DFCM [12] predictor shows 64% accuracy.

The SimpleScalar tool set provides an implementation environment that supports the study of value prediction [15].

1.3. Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 discusses computational locality in the global value history and the gDiff predictor as well as a hybrid of the gDiff predictor, hgDiff. The existence of value delay and using speculative values to reduce value delay is also discussed in Chapter 2 along with the sensitivity analysis of the hgDiff predictor to variations in queue size, table size and available difference bits. Chapter 3 concludes this thesis and discusses future work.

Chapter 2. Hybrid local/global value prediction mechanisms

2.1. Global Value Locality

As discussed in chapter 1, there are two types of value history that can be used to make a prediction: local value history and global value history. To define the two types of history assume instruction *I* is being predicted. The local value history is the value sequence produced by instruction *I* during its prior executions. In contrast, global value history contains the values produced by all the dynamic instructions before the current occurrence of instruction *I*.

For some instructions, it is very hard to achieve high prediction accuracy using local value history based predictions. For example, the value sequence produced by one load instruction in the benchmark *parser* has the following form: (xx528, xx840, 0, xx792, 0, xx720, 0, xx816, xx768, xx744, xx696, xx624, xx672, …). Neither (simple) computational nor context-based locality exists in this value sequence. Even dividing this value trace into several sub-traces based on path information would reveal neither computational pattern nor periodic pattern. This observation can also be confirmed by plotting the value sequence as in Figure 1, where the last three digits of the value are shown (the higher order digits are either the same or zero).

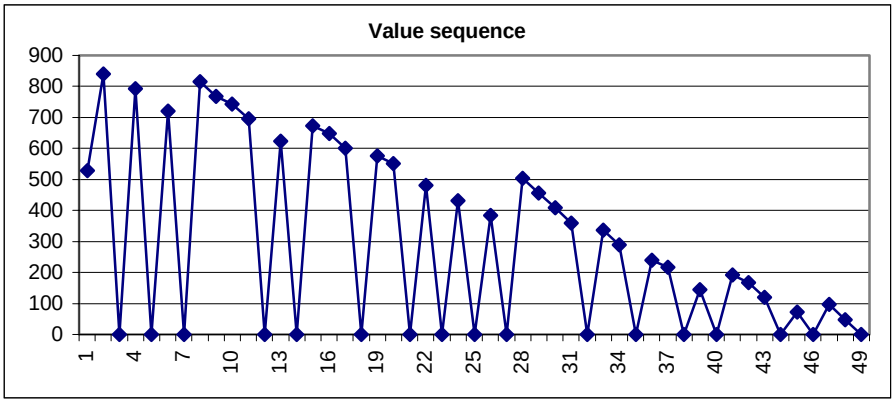


Figure 1. One hard-to-predict value sequence (from one load instruction in the benchmark parser)

Figure 1 demonstrates that although the value sequence demonstrates spurious short patterns of 2 and 3 values that appear to follow a stride pattern, the overall sequence appears random. There exists no significant computational or context-based value locality. The prediction accuracy of this instruction is 4% for (local) stride predictor and 2% for the DFCM predictor. However, when the code around this load instruction is analyzed, as shown in Figure 2, it is apparent that this load is the result of register spilling and filling (i.e., the value is stored to memory to free the architectural register and reloaded for further uses). In this example, the reloaded value is produced by two load instructions (marked 'the correlated load' in Figure 2). As a result, if the values produced by those correlated loads can be used for prediction, the accuracy will be 100%. Such locality is value locality in global value history or *global value locality*.

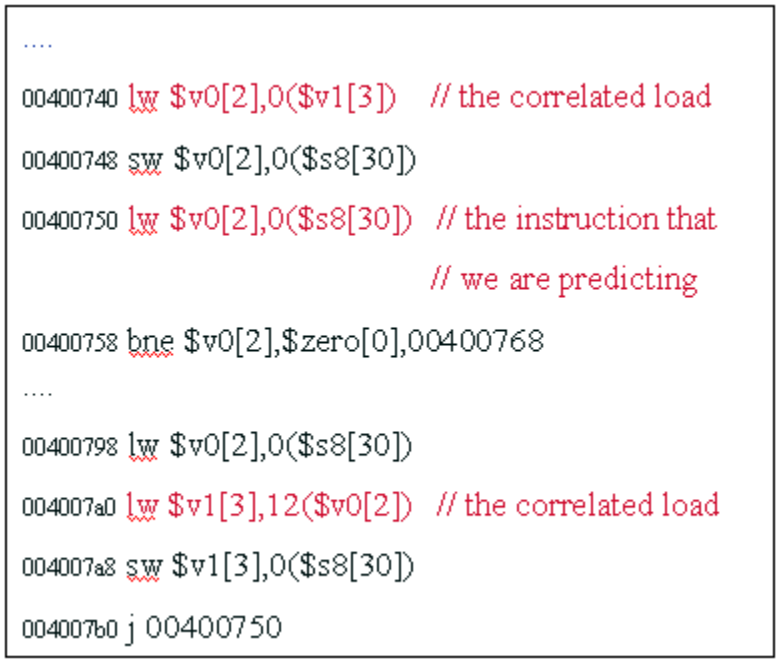


Figure 2. The code example of global value locality (extracted from benchmark parser)

To facilitate the discussion of global value locality, global value history is defined in a more formal way: a dynamic instruction stream (D, D+1, ..., D+N) produces values labeled x_1, x_2, ..., x_N . For simplicity, assume every instruction produces a value. Then, the ordered data sequence (x_1, x_2, ..., x_N) is the global value history of order N. Note that in the dynamic instruction stream, some static instructions can occur multiple times due to loops. As a result, if the order N is large enough, the global value history encompasses the local value history for one or more static instructions.

As shown in the example in Figure 2, the program exhibits value locality in the global value history. Similar to local value locality, global value locality can be classified either as computation-based or context-based. The global computation-based value locality can be formalized as shown in Equation 1.

$$x_N = a_{N1} x_{N1} + a_{N2} x_{N2} + \ldots + a_1 x_1 + a_0$$

Equation 1. Global computation-based value locality

Equation 1 demonstrates that x_N, the prediction value of the instruction (*D*+*N*), can be made as the weighted sum of the values (x_{N-1}, …, x_0) produced by instructions (*D*+*N*-1), (*D*+*N*-2) ,…, *D*.

For the example in Figure 2, the locality can be expressed (ignoring the non-value producing stores and branches) as: $x_N = x_{N-1}$ and it holds for both control paths leading to the load instruction that is being predicted.

Exploring the general form of computational locality as specified in Equation 1 is not easy due to the mathematical nature of the problem and the required hardware complexity. However, study can be concentrated on the special but most common sub-cases, similar to approaches exploiting local value history. One such case is the stride form of locality, as shown in Equation 2.

$$x_N = x_{N-k} + a_0.$$

Equation 2. Stride locality

The prediction x_N is the sum of a value in the global value history (x_{N-k}) and a stride value (a_0).

By exploiting such a locality, instructions of the sequence as shown in Figure 3 can be predicted, where the hard-to-predict *define* instruction can help to make an accurate prediction of the subsequent *use* instructions, which in turn are also hard to predict based on their local value histories.

```
…
Define (e.g., load ra, rb, rc)        // load value is hard to predict
…
Use (e.g., add rx, ra, #constant)     // the dest of add can be predicted well using equation 2
…
Use (e.g., sub rx, ra, rb)            // the dest of sub can be predicted if rb has strong repeating patterns
…
```

Figure 3. Instruction sequence with strong global stride value locality

In the next section, a predictor scheme, called *gDiff* predictor, is proposed to exploit global stride type value locality. The problem of value delay (i.e., when the prediction of X_N is being made, X_{N-k} is not available due to the pipeline latency of instruction (N-k), especially when k is small) will also be discussed.

2.2. The gDiff predictor

To exploit the locality in global value history, values produced by the dynamic instruction stream need to be stored for future predictions. A structure called the global value queue (GVQ) is designed for this purpose, as shown in Figure 4 for the overall scheme of the gDiff predictor.

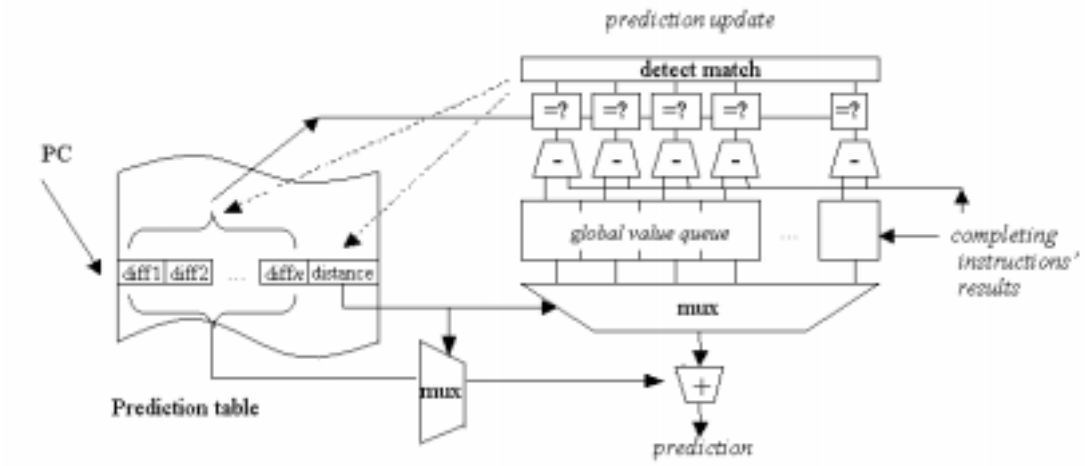


Figure 4. The structure of gDiff value predictor of order-n

As shown in Figure 4, GVQ maintains the values of the completed instructions according to the program order. The PC-indexed prediction table maintains the differences between an instruction's result and the results of n immediately preceding instructions (i.e., $X_n - X_{n-i}$, for $i = 1$ to n) and the selected distance (i.e., k for $X_n - X_{n-k}$) used for the prediction.

The gDiff predictor works as follows.

1. Prediction phase: When a value-producing instruction is dispatched, its PC is used to index the prediction table. Then, the value stored at entry k (specified by the distance field of the prediction table entry) of the GVQ is read out and added to the stride value $diff_x$ to make a prediction.
2. Update phase: When a value-producing instruction completes, the difference between the result of the instruction and all values stored in the global value queue is computed. Then, the calculated differences (n differences for an order-n predictor) are compared to the differences stored in the corresponding entry of the prediction table. If one or more matches are found, the lowest matching distance is stored in the distance field. If there is no match, the calculated differences are stored in the prediction table and there is no update of the distance field. At the same time, the current result is shifted into the GVQ.

A simple example is used to show how the predictor makes use of the global value locality. Consider the dynamic instruction stream produced from the code structure shown in Figure 5.

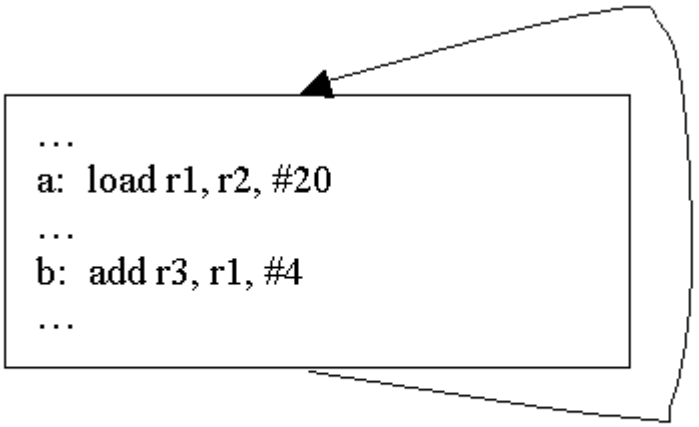


Figure 5. A simple code example

Assume the value sequence produced by instruction *a* is (1, 8, 3, 2, …). Instruction *b* will generate (5, 12, 7, 6, …). Also, assume there are two value producing instructions between instructions *a* and *b* but they will not alter the value of r1 and are not relevant to this example. For this example, the gDiff predictor will learn the stride pattern gradually to predict instruction *b* correctly, as shown in Figure 6.

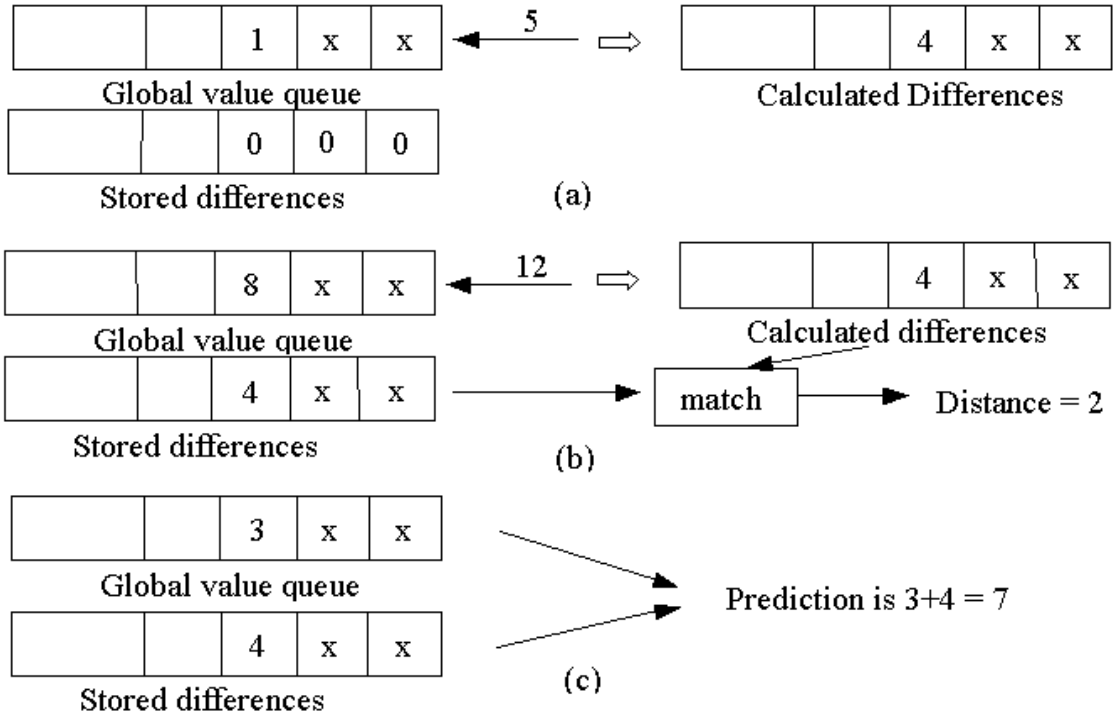


Figure 6. How the gDiff predictor works.

As shown in Figure 6(a), when instruction *b* completes with value 5, the difference is calculated between 5 and the values stored in the global value queue. Then, the calculated differences are compared with the differences stored in the prediction table indexed by PC (assume the initial difference is 0). No match is detected and the calculated differences are stored in prediction table. Next, when instruction b finishes with value 12 (Figure 6(b)) and the differences are calculated, there is a match between the calculated differences and the stored differences. Then, the index of the match is stored as the selected distance, 2 in this example. After the distance is set, when instruction *b* is met again (in its dispatch stage), the gDiff predictor can make the prediction as the sum of diff_2 and the value in queue entry number 2 (Figure 6(c)). As shown from this example, the learning time for gDiff predictor is two values.

To examine the degree to which global stride-based value locality exists in programs and how well gDiff exploits this locality, a trace-driven simulation was run for the SPECint2000 benchmarks. (1 billion instructions are executed and the first 200 million instructions are skipped in the analysis). In this experiment, the queue size of the gDiff predictor (i.e., the order) is limited to 8 and the predictions are made for all value producing integer operations or load instructions. Also, the prediction accuracy based on stride predictor (unlimited size) and DFCM predictor (unlimited first level table and 64K second level table) is shown in Figure 7 for comparison.

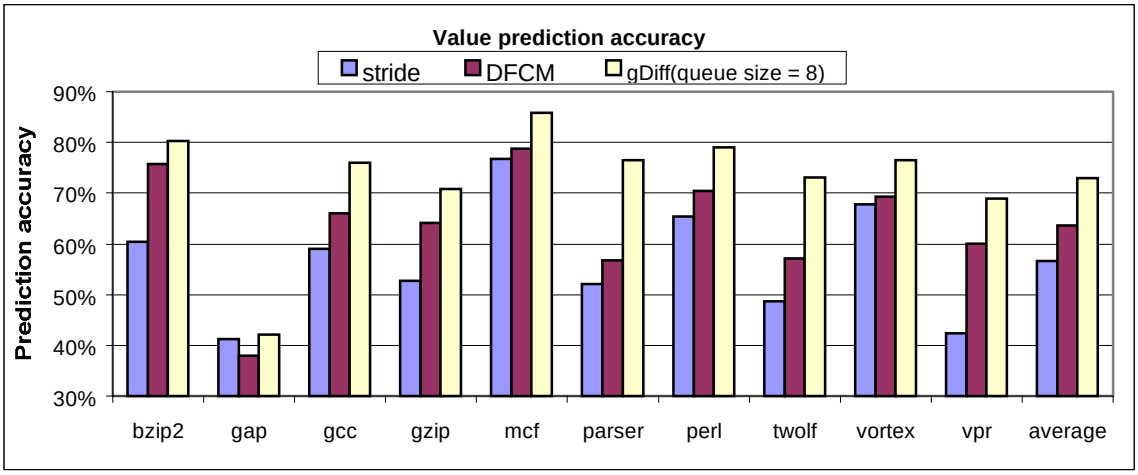


Figure 7. The prediction accuracy of the gDiff predictor and other local predictors

From Figure 7, it can be seen that the gDiff scheme predicts values very accurately for most benchmarks, up to 86% in the benchmark *mcf* and 73% on average. The exception is the benchmark *gap*, whose value predictability is fairly low (~40%), whether using local or global value locality. The reason for the low predictability is due to the hard-to-predict redundant [22] values and the long computation chain of the hard-to-predict values. If the global value queue capacity is increased to 32 (capturing long computation chains), the prediction accuracy achieves 59.7% since the window of available values

captures the redundancy. Overall, as compared to the predictors exploiting local value locality, the gDiff predictor performs better consistently for all the benchmarks. For benchmarks *parser* and *twolf*, gDiff increases the accuracy up to 24%. This shows that very strong stride-based value locality exists in the global value history and that the gDiff predictor exploits it fairly well with a modest queue size.

2.3. Value delay

Although the experiments show that high value predictability can be achieved by exploiting global stride-based value locality, there is one issue that cannot be ignored for any realistic exploitation of global value locality: *the value delay problem*. Due to the pipeline delay, especially the out-of-order execution pipeline delay, the correlated values may not be available when the prediction is being made. The impact of this value delay issue is minimal on the local value predictors except for tight loop structures. But, for global value based predictors, as a value produced by an instruction usually is consumed by other instructions very close to it (i.e., the dependence distance is small), the impact of the value delay is more dramatic. In one experiment based on profile runs, the value delay is modeled as a parameter T so that the prediction can only make use of the values produced T values before the current instruction. Figure 8 shows the prediction accuracy of gDiff predictor for different value delays.

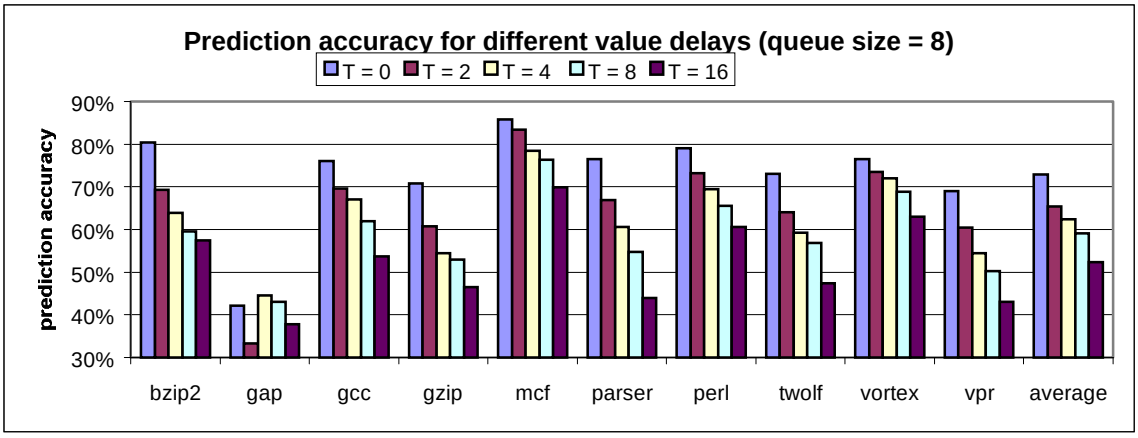


Figure 8. The prediction accuracy of the gDiff predictor with different value delays

From the simulation results in Figure 8, it can be seen that the prediction accuracy of the gDiff predictor is susceptible to value delay for all the benchmarks. On average, the prediction accuracy drops from 73% to 52% when value delay increases from zero to 16 values. One abnormal case is the benchmark *gap*, for which the highest prediction accuracy is achieved when value delay is 4. This is a side effect of the long computation chain in *gap* and the limited GVQ size, as discussed earlier. These results show that the degree of global value locality is large between one instruction and the instructions close to it and is reduced as value delay increases. This phenomenon is expected as the nature of global value locality is based on data dependencies (or a spill/fill sequence). A value produced out of the computation chain would have less locality with the values in the chain. This fact presents a great obstacle for practically exploiting the global value locality, both computation-based and context-based. As the OOO execution produces the most significant value delay due to the dynamic scheduling, the value delay problem is studied in an OOO execution context and the speculative values are used to reduce the value delay impact.

Global value histories based on profile runs show very strong value locality. In an OOO execution pipeline, this value sequence is reflected at the retire stage. However, using the retired value sequence incurs very long value delay, at least as much as the number of pipeline stages between dispatch and retire. For example, a 4-way issue 6-stage pipeline (i.e., fetch, dispatch, issue, execution, write back, and retire) may incur a value delay of as much as 16 (4 cycles x 4 values) even if there are no pipeline stalls and no multi-cycle instructions (requiring more than 1 cycle in execution). Based on the profile runs, such delay reduces the prediction accuracy of the gDiff predictor significantly.

In order to reduce such value delay, one option is to use the values produced earlier than retire stage even if the values are speculative, as shown in Figure 9. Speculative results produced at the end of the execution stage are used to update the speculative global value queue (SGVQ), while the prediction is made at the dispatch stage. As the prediction and the update are made at different pipeline stages, a timestamp is maintained in each SGVQ entry so that the differences are calculated at update only for those values produced earlier than the prediction time.

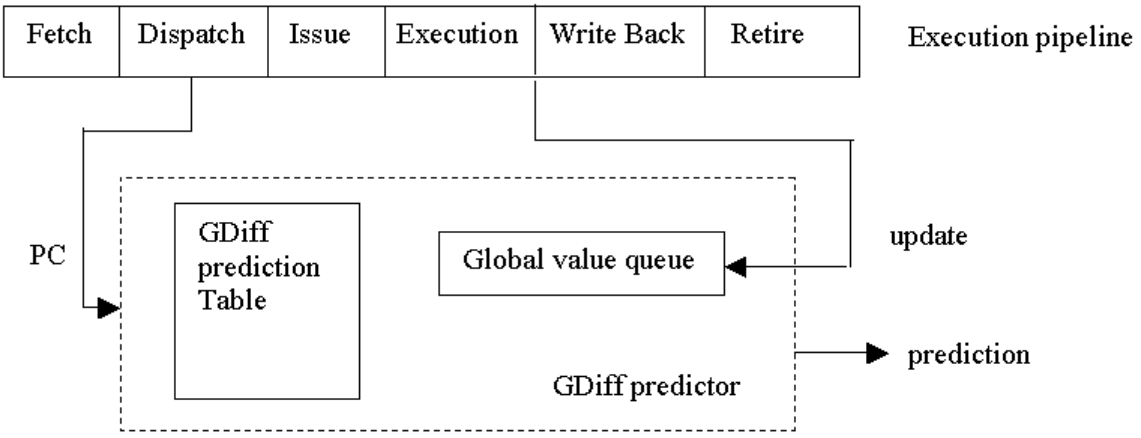


Figure 9. The gDiff predictor with the speculative global value queue (SGVQ)

The out-of-order simulator provided by the SimpleScalar toolset [15] was modified to model the gDiff predictor using speculative values. The simulation is based on a 4-way issue OOO superscalar processor with a 16KB 4-way set-associative D-cache (12 cycle miss latency), a 16KB direct-mapped I-cache (12 cycle miss latency), and a 2K-entry bimodal branch predictor. For all benchmarks except *vortex* (run to completion), the first 200 million instructions are skipped and the next 800 million instructions are simulated.

First, using speculative values to reduce the value delay was examined. Figure 10 shows the average pipeline delay for value producing instructions from dispatch to write back and from dispatch to retire based on the benchmark *vortex*. It can be seen that using speculative values allows values to be written into the queue approximately 2 cycles earlier than waiting for the retire stage. Using speculative values reduces value delay by (Cycle Difference * IPC). If an IPC rate of 2 is assumed, using speculative values reduces value delay by 4 values. Using results obtained from the profile runs, this reduction in value delay could have a significant positive impact on gDiff prediction accuracy.

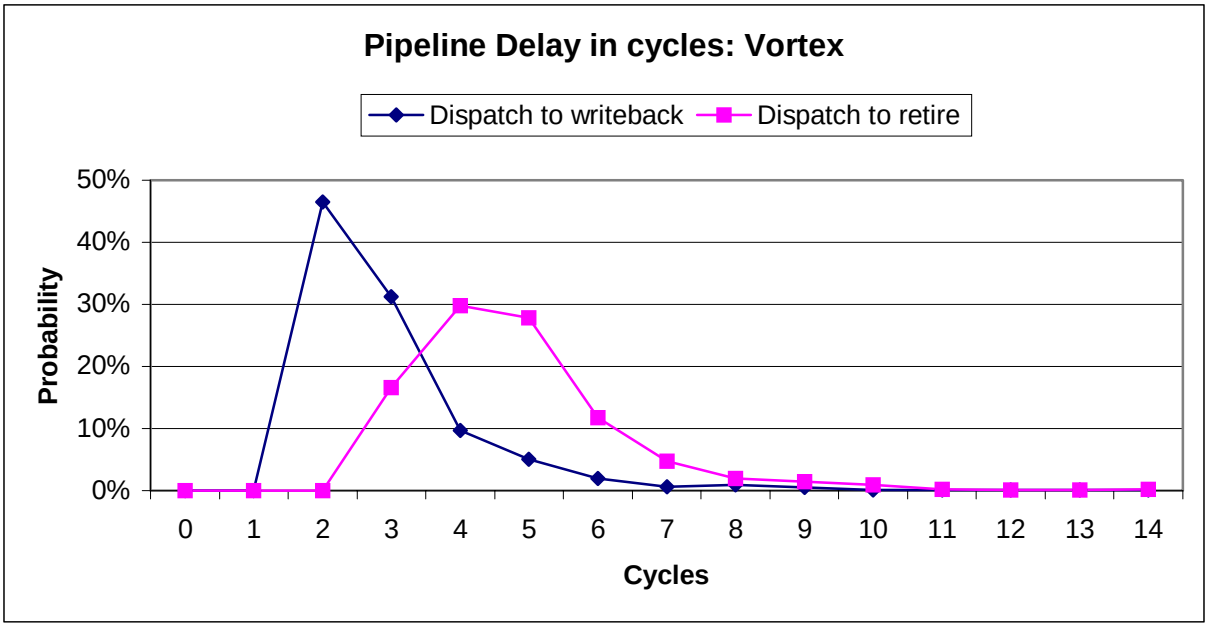


Figure 10. The distribution of pipeline latencies for all value-producing instructions in the vortex benchmark

Next, the performance of the gDiff predictor with speculative values in the value queue is analyzed. In this experiment (and all afterwards), a 3-bit confidence counter is used to filter the 'weak' predictions. The confidence mechanism is important since the misprediction recovery is costly and may significantly offset the benefits of value prediction [6]. The confidence mechanism works as follows: the confidence counter is increased by 2 for a correct prediction and decreased by 1 for an incorrect prediction. A prediction is confident if the counter is greater than or equal to 4. The ratio of resulting number of confident predictions over the total number of value producing instruction is the prediction coverage. Figure 11 shows the simulation results of the gDiff predictor with SGVQ. The prediction accuracy and prediction coverage using local stride predictor are also shown for comparison.

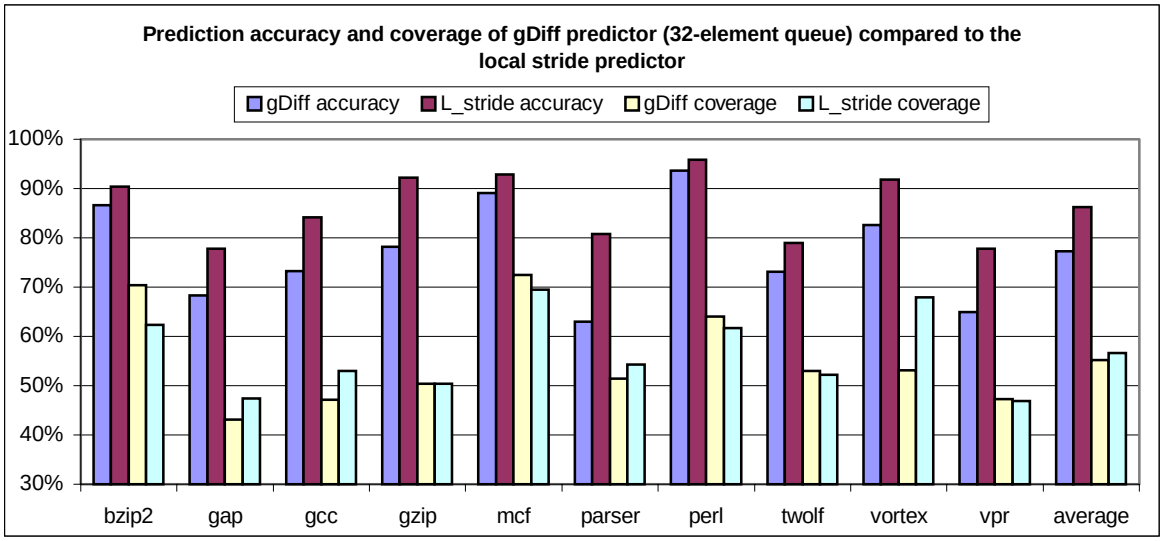


Figure 11. The prediction accuracy of the gDiff predictor with speculative values

Based on those results, it can be seen that, even with the reduced value delay, the gDiff prediction results are not encouraging. The average accuracy is 77% and the coverage is 55%, whereas the local stride predictor has 86% accuracy and 56% coverage.

Using a confidence mechanism increases the prediction accuracy but decreases the coverage. To define the performance of the predictor in terms of both coverage and accuracy, the *prediction power* of a predictor is defined as the accuracy of the predictor multiplied by the instruction coverage provided. This allows predictors with varying strengths in accuracy and coverage to be compared with one another on the basis of power.

Figure 12 consolidates the data in Figure 11 by presenting it as the power of the predictor. This concisely depicts that the local stride predictor outperforms gDiff.

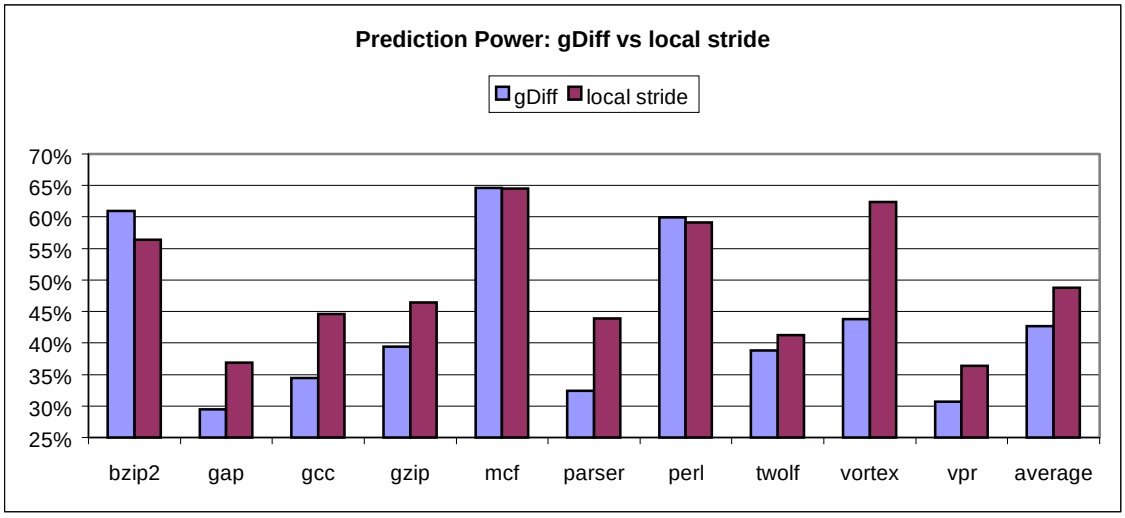


Figure 12. Power of the gDiff predictor compared to local predictors.

The lower accuracy and coverage of the gDiff predictor can be explained by execution variations due to cache misses. When the value sequence in the value queue assumes a different order than prior executions, all subsequent predictions will be incorrect until the expected order resumes or until the predictor learns the new distance. Figure 13 explains this phenomenon.

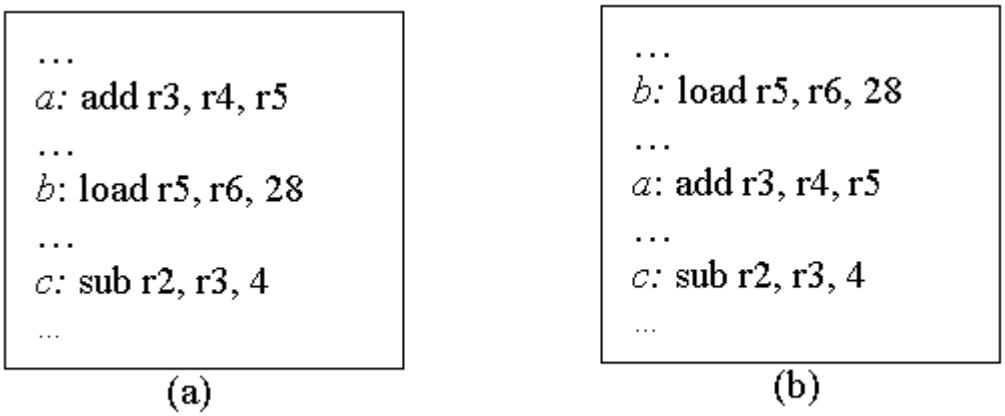


Figure 13. Two code examples to show the impact of execution variation due to cache misses

In Figure 13, instruction *c* has strong locality with instruction *a*. However, due to the load instruction *b*, the distance between instructions *a* and *c* may vary based on the hit/miss pattern of the load. Also, I-cache misses and branch mispredictions may affect the dynamic scheduling of the instructions, which accounts for another source of variation in the speculative global value history. Note that in the gDiff implementation of Figure 9, the value queue used by gDiff is updated based on speculative execution results and does not squash the values in the case of a branch misprediction.

2.4. Hybrid gDiff predictor (hgDiff)

As discussed in section 2.3, using the speculative values at write back stage helps to reduce the value delay, but introduces the problem of variations in the speculative global value queue. In this section, a new hybrid predictor is proposed. It eliminates the impact of pipeline execution variations caused by cache misses and enhances the performance of the predictor by exploiting more than one type of value locality.

The pipeline execution variations are caused by dynamic events, such as cache misses. Run-time events affect the dynamic scheduling so that the execution order of instructions may not be the same over different iterations. As a result, to remove the variations in the global value sequence due to cache misses, the sequence needs to be constructed before dynamic scheduling (i.e., in the dispatch stage). However, the execution result of an instruction is not available at dispatch time. To solve this problem, another type of speculative value is used to construct the global value sequence at dispatch time and the sequence is updated/refined at write back time. As gDiff explores the global stride type of value locality, a value predictor based on a different type of locality, e.g., local stride

predictor, is appropriate to generate the values at the dispatch stage and temporarily fill the entries in the ordered queue. The scheme, referred to as *hgDiff*, is shown in Figure 14.

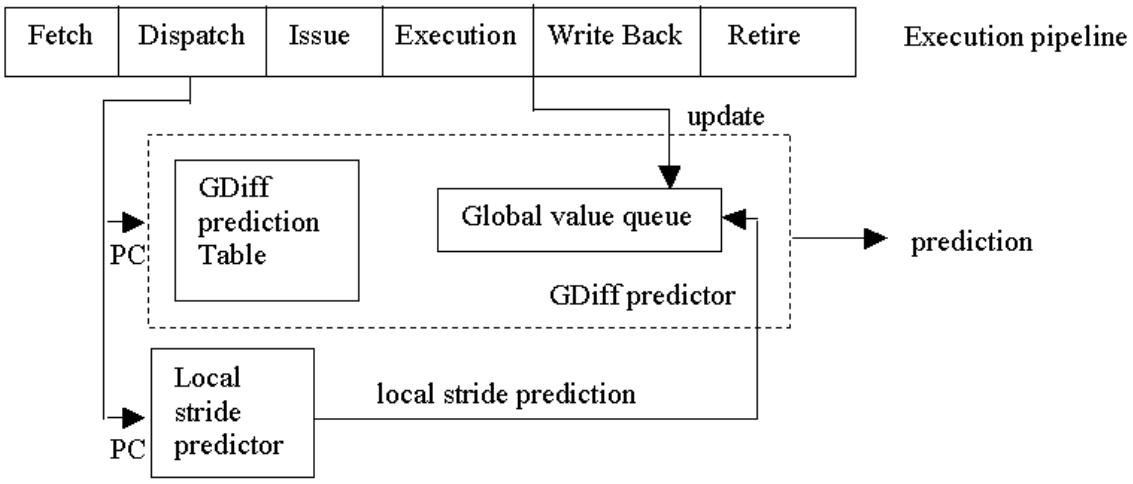
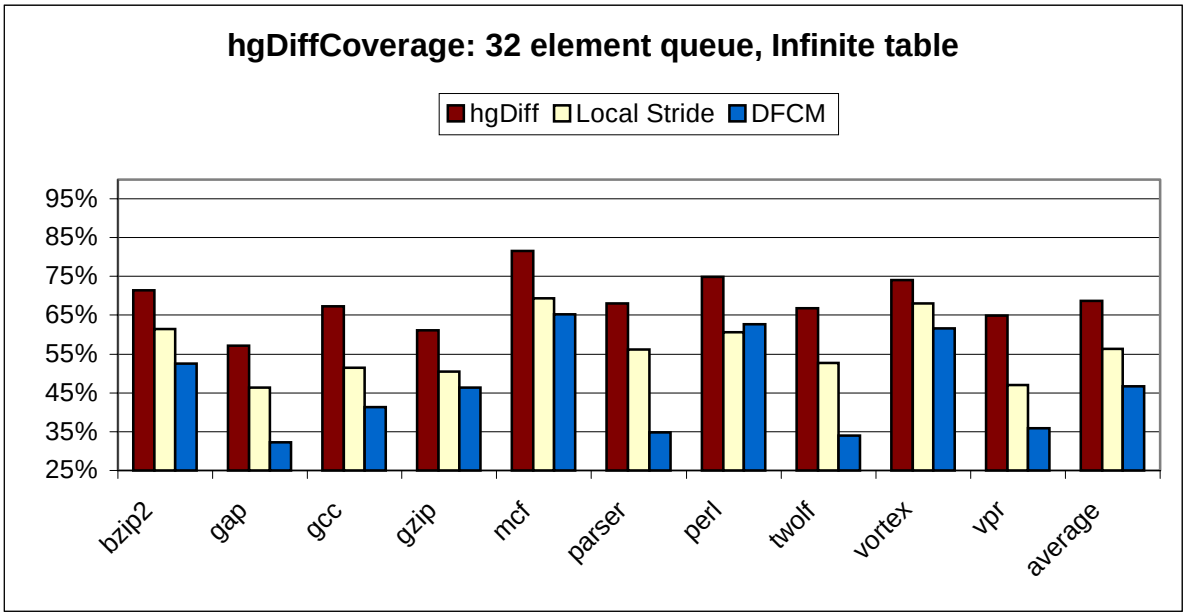


Figure 14. The hybrid gDiff predictor (hgDiff)

In Figure 14, predictions based on the local stride predictor are pushed into the global value queue at dispatch stage and updated with results at write back stage. As field is added to each entry in the RUU to indicate which entry in the value queue the result should update.

To evaluate the performance of the hgDiff predictor, the predictor was implemented using the SimpleScalar toolset using the same simulation methodology as in section 15. The simulation results are shown in Figure 15.



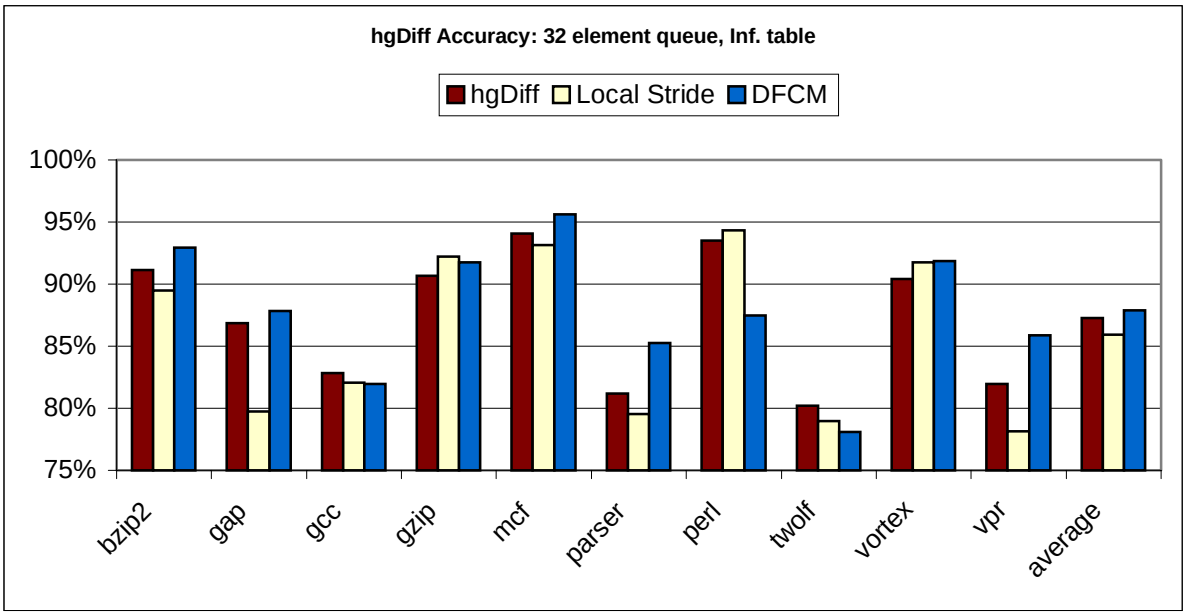


Figure 15. The performance of the hgDiff predictor

From Figure 15, it can be seen that the prediction capability of the gDiff predictor is greatly enhanced by the hybrid queue structure. For benchmarks *bzip2*, *gzip*, *mcf*, *perl*, and *vortex*, the confident predictions achieve over 90% accuracy, while other benchmarks show 80% to 87% accuracy. Compared with the local stride predictor with the same

confidence mechanism, the results show similar prediction accuracy for most benchmarks except for *gap*, for which the accuracy of the hgDiff predictor reaches 87% while the local stride predictor shows 80% accuracy. However, in terms of prediction coverage, the hgDiff predictor always shows significantly higher coverage for all benchmarks, with the largest increase of 70% from the benchmark *vpr* (from 46% to 78%), and 24% on average (from 56% to 69%), compared with the local stride predictor. Also, as indicated by profile runs, the benchmark *gap* is the most difficult to predict. The hgDiff predictor performs fairly well on the *gap* benchmark with prediction accuracy of 87% and coverage of 77%. (Surprisingly, the prediction accuracy and coverage of the hgDiff predictor for *gap* is even better than the results using gDiff and profile runs).

There are several factors accounting for the promising performance of the hgDiff predictor. First, it maximizes the exploitation of global value locality by constructing the value sequence in instruction dispatch order and utilizing speculative values. Maintaining the value sequence in dispatch order eliminates the variation problem due to cache misses and speculative values help to reduce the effective value delay.

Secondly, hgDiff provides a natural way to integrate the exploitation of a different type of value locality (the local stride in this case). For example, two instructions *a* and *b*, both predictable using a stride predictor, are close to each other, as shown in Figure 16. Clearly, there exists stride type locality between the two instructions. However, the original gDiff predictor may fail to exploit such a locality since the first load must finish before the dispatch of second load (i.e., the value delay of the instruction *a*). With the local stride prediction, although instruction *a* is still in the execution pipeline, the correct

prediction (gDiff prediction) of instruction *b* can be made based on the prediction of *a* (local stride prediction).

```
…
a: load r2, r28, #constant    //producing values 1, 1, … , 1 (local stride predictable)
…
b: load r3, r30, #constant    //producing values 3, 3, … , 3 (local stride predictable)
…
```

Figure 16. Code example demonstrating how hgDiff utilizes the local prediction

The potential of the hgDiff predictor is defined as the number of correct predictions that are available in the global value queue without being limited by the capability of the selection mechanism. Figure 17 shows the accuracy of the hgDiff predictor with a "perfect" selector compared to the power of the hgDiff implementation with a 3-bit confidence mechanism as described earlier. It is reasonable to compare the accuracy of the perfect selector with the power using the prior confidence mechanism since the perfect selector does not include a confidence mechanism and therefore provides 100% coverage (i.e., an attempt is made to predict all instructions that generate a value.) The results in Figure 18 show that an average of 75% of all value producing instructions in the dynamic instruction stream could be accurately predicted using the hgDiff predictor if an adequate selection mechanism could be defined that more accurately identifies locality distances.

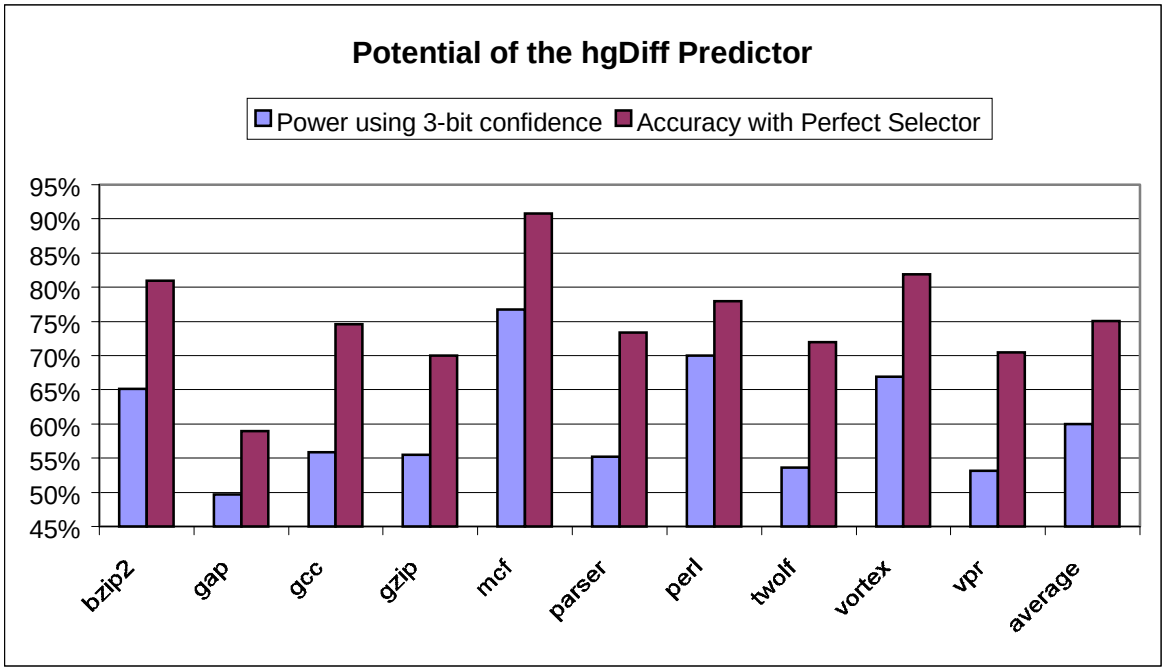


Figure 17: Potential of the hgDiff predictor

2.5. Sensitivity Analysis

The hgDiff predictor demonstrates promising prediction capabilities with order-32 as indicated by the previous results. In order to be feasible in a real implementation, the complexity of the predictor would be constrained by a reduced order-N (smaller value queue), a reduced subtractor, and a limit to the size of the prediction table

In an attempt to identify how a reduced order will impact the performance of the hgdiff predictor, the distribution of the selected distances in the global value queue for all confident and correct predictions was recorded. The results are shown in Figure 18 and demonstrate that most confident and correct predictions tend to use entries close to the head of the value queue. This phenomenon is expected since the locality is stronger when the locality distance is small. This allows a reduction of the queue size without significant performance penalty. From Figure 18, it is apparent that 94% of all confident and correct predictions are predictable using one of the first 12 elements in the global value

queue. For this reason, limiting the hgDiff predictor to order-12 by storing only12 elements in the global value queue should be adequate to achieve desirable results with the hgDiff predictor.

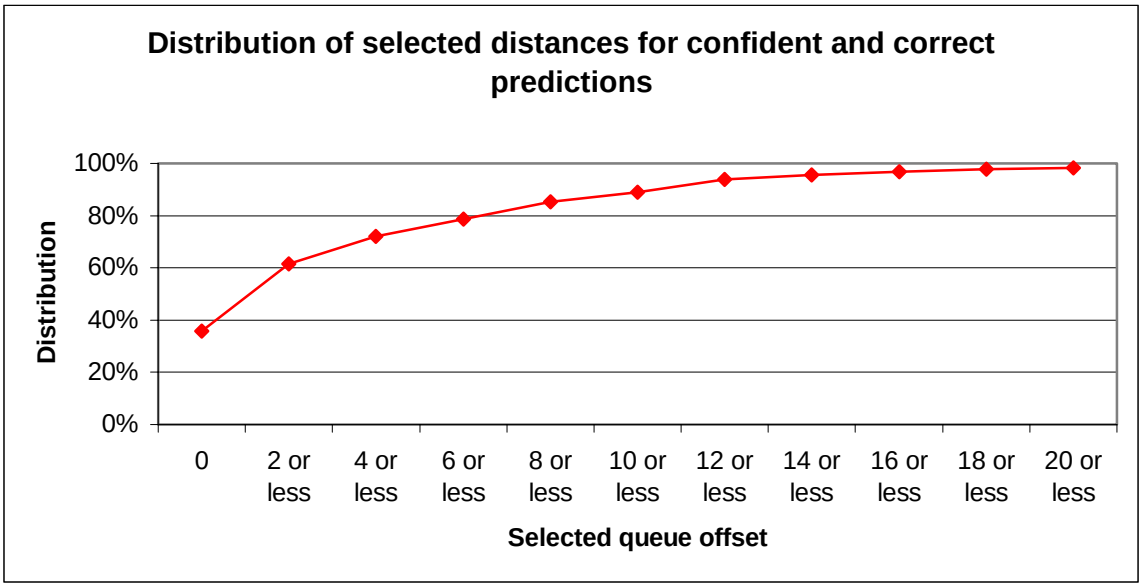


Figure 18. The distribution of entries in the global value queue selected by the hgDiff predictor for confident and correct predictions

Figure 18 also shows that 36% of all confident and correct predictions use offset 0 in the global value queue. This corresponds to using the local stride prediction and demonstrates one of the advantages of using the hybrid version of the gDiff predictor. This result also introduces the possibility of building a global value predictor that predicts using prior values directly without calculating a stride.

The results of running hgDiff with various limits on the queue size are presented as prediction power in Figure 19. The prediction power results correspond to the results in Figure 18, indicating that hgDiff performance shows a moderate consistent decline (3%) when the queue size is reduced to 16 and 12 but a significant decline when the queue size is reduced from 12 to 8 (10%) and from 8 to 4 (14%).

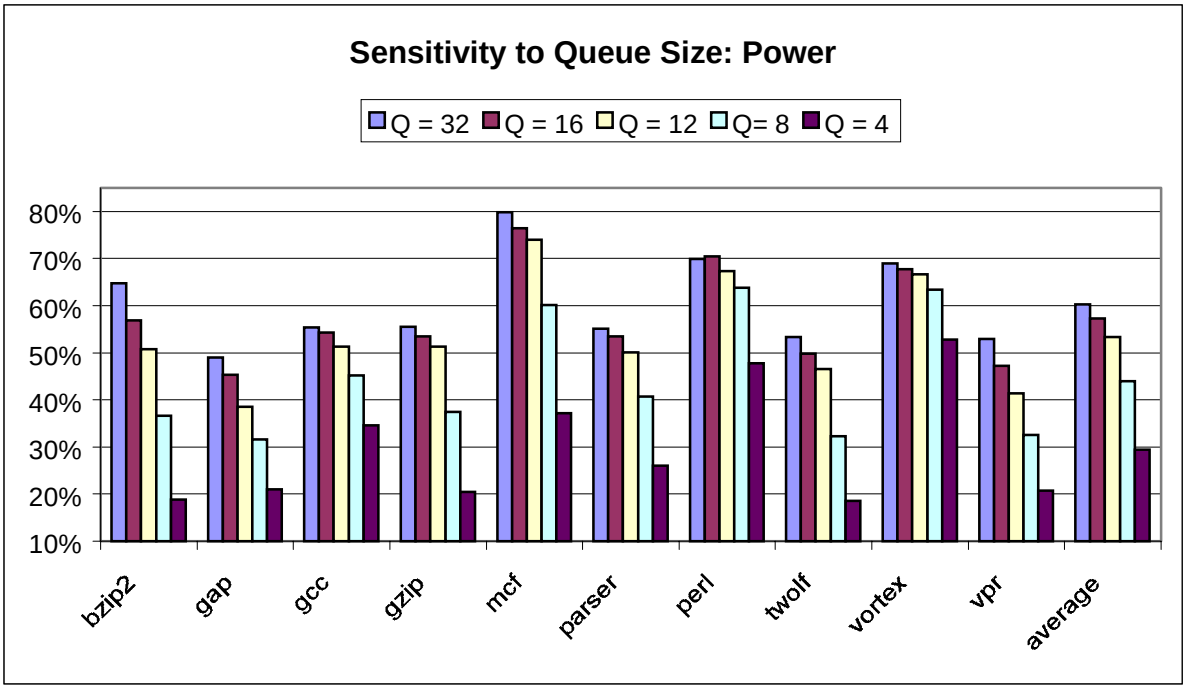


Figure 19. Prediction power of hgDiff with limited queue size

As the implementation of a subtractor in hardware is expensive, the hgDiff predictor is feasible if the number of bits required to calculate and store differences is small. Figure 20 shows the distribution of difference bits in correct and confident predictions across all benchmarks studied. It is apparent from these results that the majority of correct and confident predictions result in differences that require 7 or less bits as this accounts for 81% of confident and correct predictions.

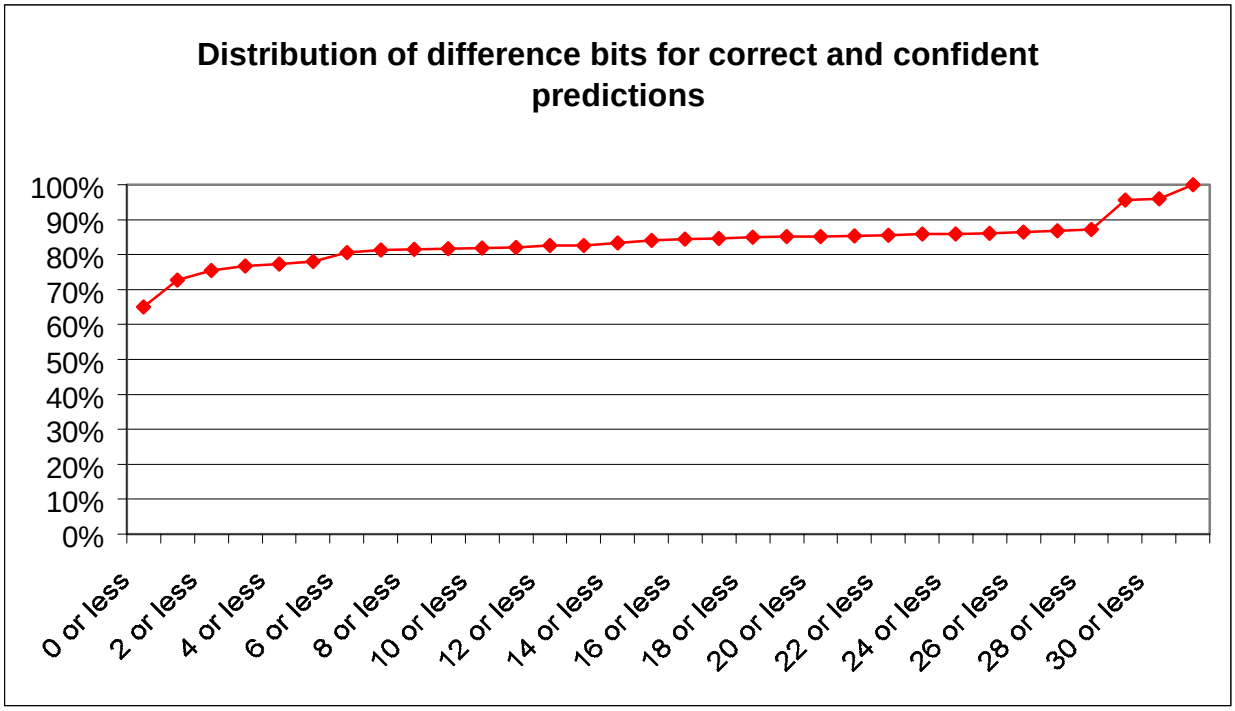


Figure 20. Distribution of difference bits required for confident and correct predictions

Interestingly, the tail of the curve in Figure 20 jumps sharply at the point of 29 bits indicating that 13% of the correct and confident predictions required 29 or greater bits. This occurs when a striding index correlates with a striding address such as in the case of array accesses. The hgDiff predictor selects the best distance for a prediction as soon as a calculated difference to a specific offset in the queue occurs twice and does not change that prediction if the prediction is confident. Therefore, in the best case, the hgDiff predictor selects a distance after the second instruction execution and successfully predicts the third. However, the local stride predictor would not confidently learn the correct stride until one cycle later. This causes an index to confidently correlate with a prior address despite the fact that the local stride prediction (closer in the queue and requiring fewer difference bits) was available and correct.

The power of the hgDiff predictor with constraints on the number of difference bits is presented in Figure 22. The results depicted in this figure correspond to the distribution of difference bits presented in Figure 21. This further indicates that reducing the number of difference bits has very minor impact on the prediction power since only 5.5% average decrease in power is demonstrated when the difference bits are reduced from unlimited (32) to 4 bits.

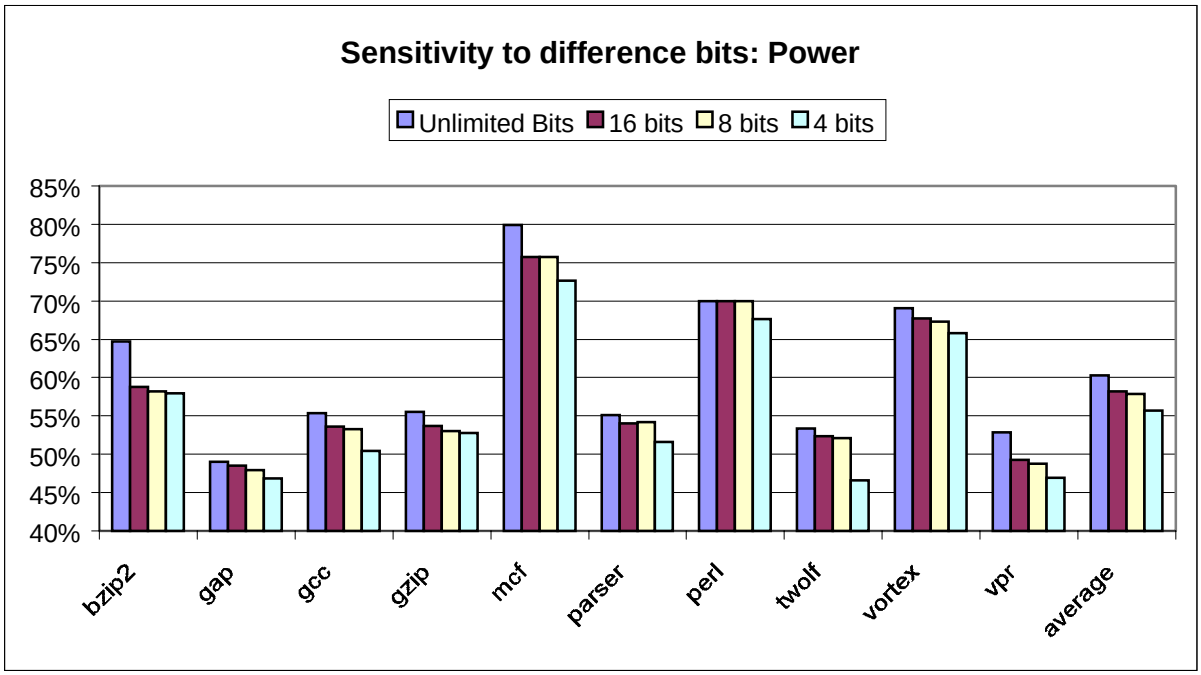


Figure 21. Prediction power of hgDiff with limits on the number of difference bits

The final parameter that was varied in the hgDiff sensitivity analysis was the size of the prediction table. Figure 23 shows the power of the hgDiff predictor when table size is varied. The graph demonstrates that reducing the table size to 16K shows a decrease in power of less than 1% when compared to an infinite table. Further reducing the table size results in additional decline of 1% with 8K, 2% with 4K and an additional and 3% with 2K. Therefore, the total decline in power is just above 6% when comparing an infinite table to a 2K table. This indicates that the hgDiff predictor is only moderately sensitive

to table size and a 16K prediction table can be used with negligible impact to prediction accuracy and coverage.

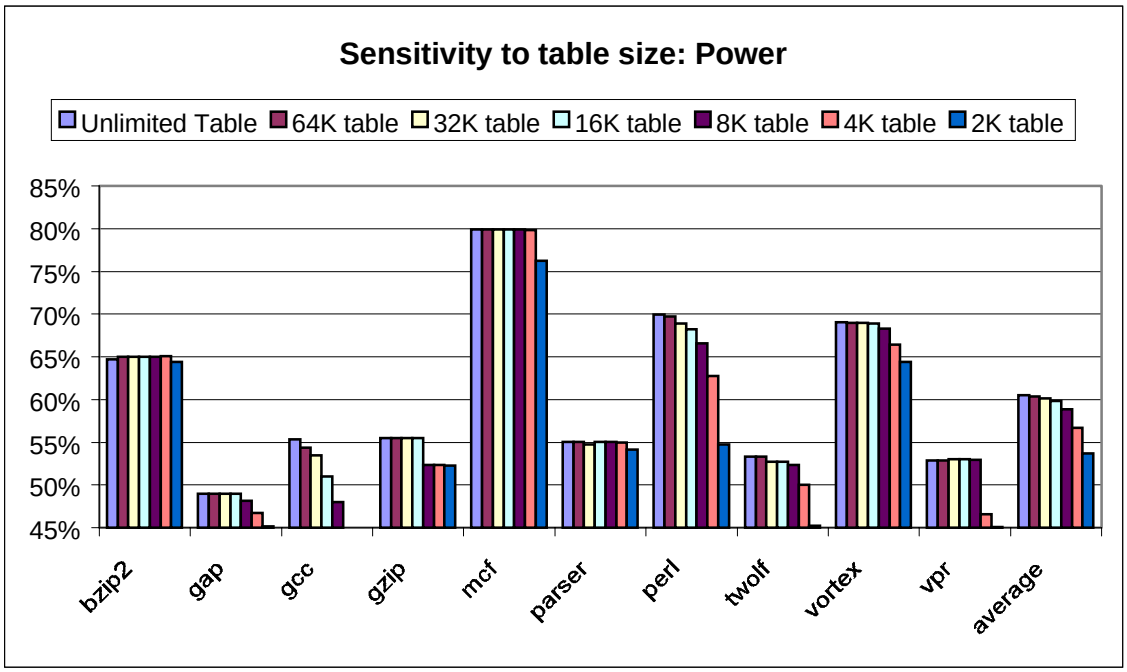


Figure 22. Power of the hgDiff predictor with varying prediction table sizes

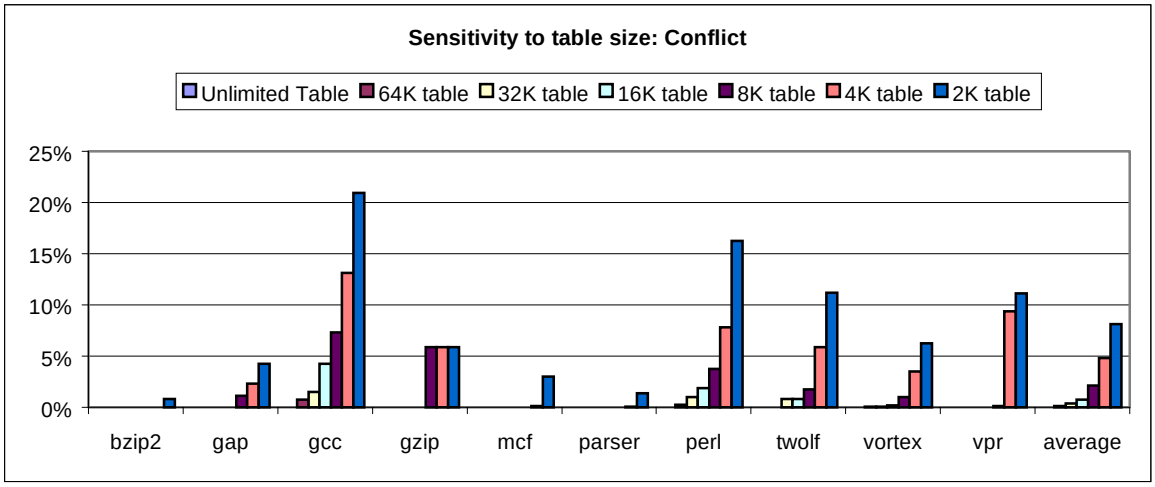


Figure 23. Conflict in the prediction table with varying table sizes

The performance of the hgDiff predictor with varying prediction table sizes corresponds to the degree of conflict that occurs. Figure 23 demonstrates that conflict in

a 16K predictor table occurs in only 4 of the 10 benchmarks evaluated. As table size is decreased further, conflict increases as the same rate as power decreases.

Chapter 3. Conclusions and Future Work

3.1. Conclusions

In this thesis, a new type of value locality, global computational locality, is studied and a prediction scheme is proposed to exploit this locality to increase the value predictability. The main contribution of this work includes:

- A new type of value locality is studied and formalized. The locality in global value history presents new opportunities that can be explored in both the architectural and compiler domains. A prediction scheme, the gDiff predictor, is proposed to exploit the global stride value locality dynamically. Experiments demonstrate that there exists very strong stride type locality in global value history and ideally, the gDiff predictor can achieve 73% prediction accuracy when predicting all the value producing instructions.
- The value delay issue is emphasized in this thesis and its impact on the gDiff predictor is studied. It is shown that value delay presents a challenge for any realistic exploitation of the global value locality, especially in out-of-order execution pipeline models. However, by combining the global value predictor with a local value predictor as proposed in the hybrid gDiff predictor (hgDiff), the value delay issue can be avoided.
- To reduce the value delay impact in OOO execution environment, speculative values are used at write back stage instead of waiting for them to be retired in-order. This solution reduces the value delay but introduces execution variations due to cache misses and branch mispredictions. However, ordering the values in the value queue

based on dispatch order prevents the variations in execution time from impacting the success of the hgDiff predictor.

In order to reduce the impact of both value delay and pipeline execution variation, the global value sequence is constructed at instruction dispatch time using the local stride predictions. Then, the speculative values produced at write back stage are used to update the value sequence. In this way, the hgDiff predictor maximizes the exploitation of global value locality and while enabling predictions based on local value history for instructions that are highly predictable with existing techniques.

The experiments show that the hgDiff predictor achieves an impressive 88% prediction accuracy with 69% coverage significantly outperforming an unconstrained local stride predictor, which demonstrates 86% accuracy and 56% coverage. Additionally, the complexity of the hgDiff predictor can be managed by significantly reducing the order of the predictor, the size of the subtractor, and the size of the prediction table without seriously affecting prediction power. With the prediction accuracy and coverage demonstrated, the hgDiff predictor moves value prediction closer to commercial feasibility.

3.2. Future Work

There are several directions for future work in the area of exploiting global computational value locality and in improving the gDiff and hgDiff predictors.

One item would be to convert the identified predictability into speedup numbers, either in a single superscalar processor model or the multi-threading execution model. This would reveal if the instructions correctly predicted by the hgDiff predictor are on the

critical path and are among the set of instructions that are hard to predict by existing predictors.

Studying the variation in instruction locality is another area of further research. More study is required to determine if the source instruction for a given correlated dependent instruction is constant or if it varies with execution phases. If the static instruction locality distance is found to be constant (i.e. a given instruction consistently correlates with the same prior instruction regardless of execution order), a 2-level predictor could be developed that may produce higher accuracy than the 1-level predictor proposed here. With this 2-level predictor, the global value queue stores the address of a correlated instruction and provides an index into a second level table. The second level table stores the values produced by a single instruction. This solution would eliminate the problem of varying locality distances caused by execution and control variations and would eliminate the need for an ordered value queue. This solution has the potential to provide comparable prediction power to the hgDiff predictor without the use of an additional local stride predictor.

If the instruction locality distance were found to be constant, another interesting direction would be to use compiler techniques to exploit the global locality among instructions [9, 10, 20]. Using profile data, the compiler detects correlated instructions as described in [9]. *LDPRED* and *UDPRED* operations are inserted in the instruction stream along with compensation code to statically break the data dependencies among correlated instructions. Dependencies are inserted between the update of the predictor for a source instruction and the load prediction for future correlated instructions. Therefore, both value delay and variations in execution order would not affect the capabilities of the

predictor. The compiler technique would also allow a value generated by a single instruction to be used as the prediction source for multiple subsequent instructions when a chain of global value locality is determined. This is in contrast to the superscalar hgDiff implementation which always predicts using the nearest correlated instruction and is therefore significantly impacted by value delay.

To minimize the impact of control flow variations on the predictor, the hgDiff predictor could use control flow information to select a best distance for each incoming path [13].

Lastly, the perfect selector results presented here indicate the potential of the hgDiff predictor. Further study of a selection mechanism to more accurately identify the correct distance in the global value queue to be used for a prediction could increase hgDiff prediction accuracy and coverage. It may be possible to identify a pattern of distances and select the next best distance based on the next element in the pattern. A study of the distribution of selected distances for specific opcodes could produce a heuristic for selecting correlation distance based on the opcode of the current and preceding instructions.

References

[1] M.H. Lipasti, C. B. Wikerson and J. P. Shen, “Value locality and load value prediction,” In 7th International Conference on Architectural Support for Programming Language and Operation Systems (ASPLOS-7), Oct, 1996.

[2] M. H. Lipasti and J. P. Shen, “Exceeding the dataflow limit via value prediction,” In 29th International Symposium on Microarchitecture (MICRO-29), 1996.

[3] Y. Sazeides and J. E. Smith, “The predictability of data values,” in 30th International Symposium on Microarchitecture (MICRO-30), Nov. 1997

[4] F. Gabbay and A. Mendelson, “Speculative execution based on value prediction,” EE Department Tech Report 1080, Tachnion - Israel Institute of Technology, Nov. 1996.

[5] F. Gabbay and A. Mendelson, “Can program profiling support value prediction?”, in 30th International Symposium on Microarchitecture (MICRO-30), Nov. 1997.

[6] K. Wang and M. Franklin, “Highly accurate data value prediction using hybrid predictors,” in 30th International Symposium on Microarchitecture, Nov. 1997.

[7] B. Rychlik, J. Faistl, B. Krug, and J. P. Shen, “Efficacy and performance impact of value prediction,” in International Conference on Parallel Architectures and Compilation Techniques (PACT'98), 1998

[8] P. Marcuello, J. Tubella, and A. Gonzalez, “Value prediction for speculative multithreaded architectures,” in 32nd International Symposium on Microarchitecture (MICRO-32), 1999

[9] C. Fu, M. D. Jennings, S. Y. Larin, and T. M. Conte, “Value speculation scheduling for high performance processors,” In 8th International Conference on Architectural Support for Programming Language and Operation Systems (ASPLOS-8), 1998

[10] T. Nakra, R. Gupta, and M. L. Soffa, “Value prediction in VLIW machines,” International Symposium on Computer Architecture (ISCA-26), May 1999.

[11] M. Burtscher and B. G. Zorn, “Exploring last n value prediction,” In International Conference on Parallel Architectures and Compilation Techniques (PACT'99), 1999

[12] B. Goeman, H. Vandierrndonck, and K. D. Bosschere, “Differential FCM: Increasing value prediction accuracy by improving table usage efficiency,” International Symposium on High-Performance Computer Architecture (HPCA'01), Jan 2001.

[13] T. Nakra, R. Gupta, M. L. Soffa, “Global context-based value prediction,” in 5th International Symposium on High Performance Computer Architecture (HPCA-5), 1999.

[14] Y. Sazeides, “Modeling value prediction,” in 8th International Symposium on High Performance Computer Architecture (HPCA-8), 2002.

[15] D. Burger and T. Austin, "The SimpleScalar tool set, v2.0," Computer Architecture News (ACM SIGARCH newsletter), vol. 25, June 1997.

[16] S. Lee, Y. Wang, and P. Yew, “Decoupled value prediction on trace processors”, in 6th International Symposium on High Performance Computer Architecture (HPCA-6), 2000.

[17] S. Lee and P. Yew, “On some implementation issues for value prediction on wide ILP processors”, in International Conference on Parallel Architectures and Compilation Techniques (PACT'00), 2000.

[18] R. Sathe and M. Franklin, “Available parallelism with data value prediction”, in 5th International Symposium on High Performance Computer Architecture (HPCA-6), 2000.

[19] E Tune, D. Liang, D. Tullsen, and B. Calder, “Dynamic prediction of critical instructions”, in 7th International Symposium on High Performance Computer Architecture (HPCA-7), 2001.

[20] E. Larson and T. Austin, “Compiler controlled value prediction using branch predictor based confidence”, 33rd International Symposium on Microarchitecture (MICRO-33), 2000.

[21] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. E. Smith. "Trace Processors". 30th International Symposium on Microarchitecture, pp. 138-148, December 1997.

[22] A. Sodani and G. S. Sohi, “Understanding the Differences Between Value Prediction and Instruction Reuse “, 31st International Symposium on Microarchitecture (MICRO-31), Nov-Dec 1998.