Detecting Global Stride Locality in Value Streams

Huiyang Zhou, Jill Flanagan, Thomas M. Conte
Department of Electrical and Computer Engineering
North Carolina State University
{hzhou,jtbodine,conte}@eos.ncsu.edu

Abstract

Value prediction exploits localities in value streams. Previous research focused on exploiting two types of value localities, computational and context-based, in the local value history, which is the value sequence produced by the same instruction that is being predicted. Besides the local value history, value locality also exists in the global value history, which is the value sequence produced by all dynamic instructions according to their execution order. In this paper, a new type value locality, the computational locality *in the* global value history *is studied. A novel prediction scheme, called the gDiff predictor, is designed to exploit one special and most common case of this computational model, the stride-based computation, in the global value history. Such a scheme provides a general framework to exploit global stride locality in any value stream. Experiments show that there exists very strong stride type of locality in global value sequences. 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 capability of realistically exploiting 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. We study the value delay issue in an out-of-order (OOO) execution pipeline model and propose a new hybrid scheme to maximize the exploitation of the global stride locality. This new hybrid scheme shows 91% prediction accuracy and 64% coverage for all value producing instructions. We also show that the global stride locality detected by gDiff in load address streams provides strong capabilities in predicting load addresses (coverage 63% and accuracy 86%) and in predicting addresses of missing loads (33% coverage and 53% accuracy).*

1. Introduction

Prediction is a powerful technique for accelerating instruction processing. Successful prediction schemes generally predict a single bit of information. For example, predicting a branch direction enables the speculation of control dependencies. Speculating pure data dependencies however requires predicting more than one bit of information. Skepticism regarding these techniques is often centered on the difficulty of predicting 32 (or 64) bits of information. Locality in the value history of a program indicates that the information content is significantly less than 32 bits worth [6, 8, 17, 18, 25]. Exploiting this locality requires a significant investment in hardware for modest prediction accuracy [7, 18, 25, 30]. But if the hardware cost can be tolerated and high accuracy can be reached, value prediction can reduce pipeline stalls by introducing more (speculatively) independent instructions [7, 16, 17, 18, 22, 23], introducing helper instructions such as prefetches [1], enhancing branch prediction [11], and enabling speculative multithreading [15, 19, 31].

This paper introduces novel value prediction schemes that leverage a new kind of value locality based on global stride information. The resulting scheme, called the *gDiff* predictor, is easily combined with past schemes for very potent value prediction strategies. The accuracy achieved by these schemes is 91% and the confidence-gated prediction coverage is 64% for all value producing instructions. Compared to a previously published local stride predictor (accuracy 89% and coverage 55%) and local context predictor (accuracy 87% and coverage 45%), the improved accuracy and coverage result in up to a 53% (19% in average) speedup over a baseline 4-way issue, 64-entry issue queue machine and up to a 17% (4% in average) speedup over the baseline model with a local stride predictor. We also demonstrate the use of the gDiff predictor in exploring the global stride locality in the load address stream. The results show that the gDiff predictor achieves much higher coverage and accuracy in predicting load addresses (accuracy 86% and coverage 63%) as well as predicting addresses of missing loads only (accuracy 52.9% and coverage 32.5%) when compared with local predictors or a Markov predictor with a much larger prediction table.

Value prediction methods exploit the locality in the program's execution history to achieve high prediction accuracy. During program execution, two types of value histories, *local value history* and *global value history*, can be used to predict the value of an instruction. Local value history is a value sequence produced by prior executions of the *same instruction*, whereas global value history is produced by *all dynamic instructions*. If the length (order) of the global value history sequence is made sufficiently long, it will encompass the local value history sequence as well.

In previous research on value predictability, value locality was broadly classified into two categories: *computational* and *context-based* [25]. Most of the proposed value predictors exploit these localities in the local value history. Those predictors include the *last value predictor* [18], the *last N-value predictor* [4], the *stride predictor* [7, 8, 17, 18], and *context predictors* [9, 25, 30]. The hybrid predictors can combine both computational and context-based predictors to exploit both types of localities to achieve higher prediction capability [21, 22, 25, 30].

Compared to local value history, using global value history for prediction is less thoroughly studied in the literature. The *previous instruction* (PI) based predictor [20] was proposed to explore the correlation between two immediately close instructions in the dynamic instruction stream (i.e., the global value history). It may be viewed as the first-order global context-based predictor. In the *dynamic dataflow-inherited speculative context* (DDISC) predictor [28], higher order of context is used and derived from the closest predictable values in the instruction's dataflow path.

In this paper, a new type of the value locality, *computational locality in the global value history,* is studied. A novel predictor scheme, the *gDiff* predictor, is proposed to exploit one special and common case of this computational locality, stride-based global locality. Experiments show that there exists very strong stride-based locality in global value histories; and, many instructions that are hard-to-predict using local history-based predictors become highly predictable using global history-based predictors. Ideally exploiting global stride 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 [9] predictor shows 64% accuracy.

Value delay limits the performance of *any* value prediction schemes, similar to branch outcome delay [10]. Value delay occurs when the required past value is not available for use by the predictor due to pipeline latency. Based on profile runs, the prediction accuracy of the gDiff predictor drops to 52% with a value delay of 16 (i.e., the current prediction cannot use the values that are produced by 16 immediately close value-producing instructions due to delays in the pipeline).

One way to reduce value delay in an out-of-order (OOO) pipeline is to use speculative values at the execution stage instead of waiting for values to be committed in the program order. However, the resulting global value sequence is out of order and is susceptible to execution variations due to cache misses and branch mispredictions, which in turn obscures its value locality. To reduce the execution variation impact, a novel gDiff-based hybrid approach is proposed which achieves a significantly higher prediction capability (91% accuracy and 64% coverage) than a value-delayed gDiff predictor.

The remainder of the paper is organized as follows. Section 2 discusses computational locality in the global value history. Section 3 introduces the gDiff predictor and discusses impact of value delay. Section 4 presents an approach to reduce the value delay by using speculative values. The gDiff-based hybrid predictor is proposed in Section 5. Section 6 uses the gDiff predictor to predict the load address stream and comments on its potential to drive prefetching. Section 7 presents the superscalar performance for gDiff-based value speculation. Finally, Section 8 concludes and discusses future work.

2. Computational locality in global value history

As discussed in Section 1, two types of value histories can be used to make a prediction. If the goal is to predict instruction *I*'s value, the *local* value history is defined as the value sequence produced by the same instruction *I* during its prior executions. In contrast, the *global* value history contains the values produced by all the dynamic instructions before the current dynamic occurrence of the instruction *I*. Two models of value locality exist in value sequences: the *computational* and the *context-based* models. Most proposed value predictors exploit one (e.g., stride predictor and FCM predictor) or both (e.g., DFCM predictor and hybrid predictor) locality models in the local value history. The local value predictability can be further fine-tuned using the control-flow (or path) information [20]. Local predictors perform well when the value sequence produced by an instruction (i.e., its local value history) exhibits strong periodic or stride-type of value patterns.

For some instructions in a program, it is difficult to achieve high prediction accuracy using the local value history alone. 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, …). Apparently, neither computational nor context-based locality exists in this value sequence. Even dividing this

value trace into several sub-traces based on path information reveals neither a computational nor a periodic pattern. This observation can also be confirmed by plotting the value sequence as in Figure 1, where the last three digits of the values are shown (the higher order digits are either the same or zero).

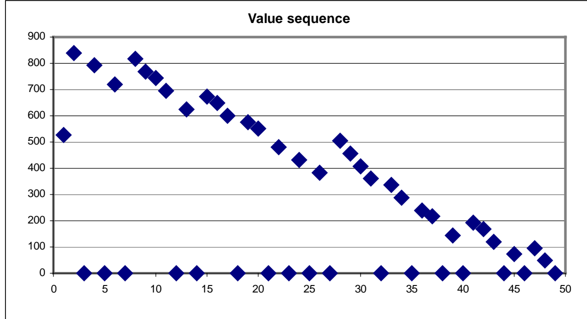

Figure 1. One hard-to-predict value sequence (from one load instruction in the benchmark *parser*).

From Figure 1, we can see that although the dynamic range of the values decreases monotonically, the value sequence looks like random 'noise' and there exists little computational or context-based value locality. The prediction accuracy of this instruction is 4% for the (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 can be seen that the load is the result of register spilling and filling (i.e., the value is stored to memory to free the physical register and reloaded for further use). In this example, the reloaded value is produced by two load instructions (marked 'the correlated load' in Figure 2). If the values produced by those correlated loads can be used for prediction, the accuracy will be 100%. Such locality is termed value locality in global value history, or *global value locality*.

```
....
00400740  lw $v0[2],0($v1[3])    //the correlated load
00400748  sw $v0[2],0($s8[30])
00400750  lw $v0[2],0($s8[30])   // the instruction that we are
                                 // predicting
00400758  bne $v0[2],$zero[0],00400768
....
....
00400798  lw $v0[2],0($s8[30])
004007a0  lw $v1[3],12($v0[2])   //the correlated load
004007a8  sw $v1[3],0($s8[30])
004007b0  j 00400750
....
```

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

We can define global value history in a more formal way: the values produced by dynamic instruction stream (D, D+1, ..., D+N) are labeled x_0, x_1, x_2, ..., x_N . For simplicity, assume every instruction produces a value. Then, the ordered data sequence (x_0, x_1, x_2, ..., x_N) is the global value history of order (i.e., length) $N+1$.

As shown in the example in Figure 2, programs exhibit value locality in the global value history. Similar to local value locality, we can classify global value locality as either computational or context-based. The PI value predictor proposed in [20] can be viewed as an example of exploiting order-1 global context-based locality and the DDISC predictor explores higher order context-based locality with the help of the data flow information [28]. A general form of the global computational value locality can be formalized using linear combination in the following way,

$$x_N = a_{N-1}x_{N-1} + a_{N-2}x_{N-2} + \cdots + a_1 x_1 + a_0 . \qquad \textbf{(1)}$$

where x_N, the prediction value of the instruction (D+N) can be made as the weighted sum of the values (x_{N-1}, ..., x_1) produced by instructions (D+N-1), (D+N-2) ,..., (D+1). There is no prior work on global computational value locality. (However, the locality of similar form in global *branch* history was recently studied and the resulting "perceptron" branch predictor was presented in [12].) For example, the locality in Figure 2 can be expressed as (ignoring the non value-producing stores and branches): $x_N = x_{N-1}$, and this 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 hardware complexity that a general treatment would require. We found that concentrating on the special but most common cases produces powerful predictors. One such case is the variable stride form of locality, as shown in Equation 2.

$$x_N = x_{N-k} + a_0 . \qquad \textbf{(2)}$$

where the prediction x_N is the sum of a value in the global value history (x_{N-k}) and a stride value (a_0). Global stride locality can help in two situations. The first is when the stride locality is embedded in code sequences explicitly or implicitly. Some examples are shown in Figure 3.

```
…
Define (e.g., load ra, rb, rc)    // load value is hard to predict
…
Explicit Use (e.g., add rx, ra, #constant) // the dest of add can be
                                  //predicted well using equation 2
…
Explicit Use (e.g., sub rx, ra, rd)  //  the dest of sub can be predicted
                                  //if rd has strong repeating patterns
…
Implicit Use (e.g., load rx, ry, rz)   //Implicit use through the
                                  //memory (eg., spilling and filling; reloading)
```

Figure 3. Instruction sequence with strong global stride value locality.

In the figure, the hard-to-predict *define* instruction can help to make an accurate prediction of the subsequent *use* instructions, which are also hard to predict based on their local value histories alone. The second is when the stride

locality is embedded in a data structure. Figure 4 shows such an example. As pointed out in [26], if the link elements (i.e., the 'next' field) and the strings are allocated in the same order as they are referenced, there is a near constant stride between the two load addresses when the two fields are referenced.

```
struct  string_list {
            struct  string_list * next;
            char * string;
}
```

Figure 4. Global stride locality embedded in a data structure if '->next' and '->string' are accessed sequentially (from the benchmark *parser*).

3. Exploring the global computational value locality using the GDiff predictor

Values produced by the dynamic instruction stream need to be stored for future predictions in order to exploit locality in global value history. A structure called the global value queue (GVQ) is introduced for this purpose (as shown in Figure 5 for the overall scheme of the gDiff predictor). The GVQ maintains the values of the completed instructions according to their execution order. The PC-indexed prediction table maintains the selected distance (i.e., k for $x_N - x_{N-k}$) used for the prediction and the differences between the instruction's result and the results of n instructions that finished immediately before it (i.e., $x_N - x_{N-i}$, for $i = 1$ to n).

The gDiff predictor operates as follows:

1. **Prediction phase:** when a value producing instruction with address *PC* is dispatched, *PC* is used to index into 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 with the stride value (diff_k) to make a prediction. This is potentially a two-cycle operation, but is hidden in the front part of the pipeline (i.e., during decode/dispatch).
2. **Update phase:** when the value producing instruction with address *PC* is completed, the result of the instruction is used to calculate the difference between it and the values stored in the global value queue. This is done in parallel. Then, the calculated differences (n differences for an order n predictor) are compared against the differences stored in the corresponding entry of the prediction table. If there is a match, the 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. This work occurs after execution and bypass and so is not highly time-critical.

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 6.

```
…
a:  load r1, r2, #20
…                          loop
b:  add r3, r1, #4
…
```

Figure 6. A simple code example.

Assume the value sequence produced by instruction a is (1, 8, 3, 2, …) and instruction b generates (5, 12, 7, 6, …). Also, assume that between the instructions a and b, there are two value producing instructions but they will not alter the value of $r1$ and their values have no correlation with the value sequence produced by instruction a. For this example, the gDiff predictor will learn the stride pattern gradually to predict instruction b

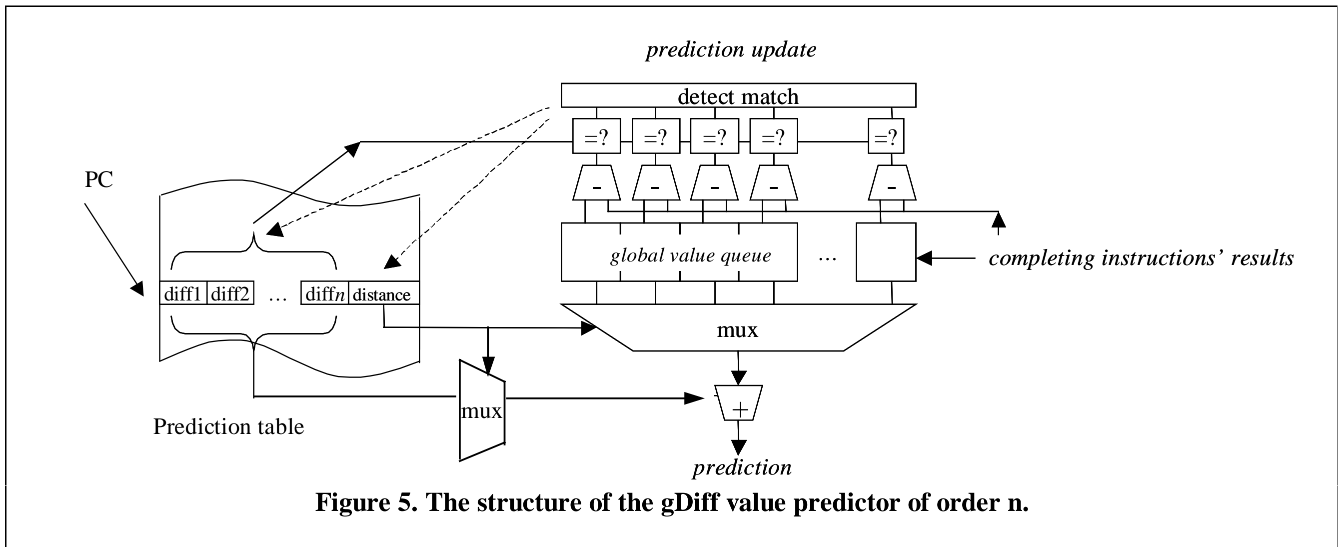

Figure 5. The structure of the gDiff value predictor of order n.

correctly, as shown in Figure 7.

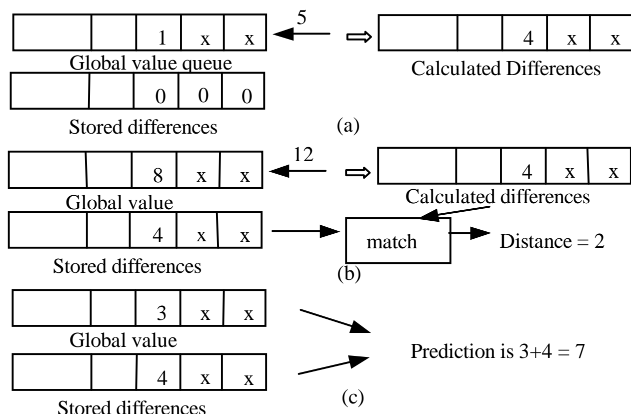

Figure 7. How the gDiff predictor works with the example: (a) the 'snapshot' of gDiff predictor when value '5' of instruction b comes; (b) the 'snapshot' of gDiff predictor when value '12' of instruction b comes; (c) the prediction made for next occurrence of instruction *b*.

As shown in Figure 7(a), when instruction *b* completes with value 5, the difference is calculated between 5 and the values inside the global value queue. Then, the calculated differences are compared with the differences stored in prediction table indexed by *PC* (assuming the initial difference is 0). Since there is no match, the calculated differences are stored in prediction table. Next, when instruction *b* finishes with value 12 (Figure 7(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 seen again (in its dispatch stage), and the gDiff predictor can make the prediction as the sum of selected stride (diff_2) and the value in queue entry number 2 (Figure 7(c)). As shown from this example, the learning time for gDiff predictor is two dynamic value productions.

To examine the degree to which the global stride-based value locality exists in program and how well gDiff exploits this locality, a profile-based simulation was run for SPECint2000 benchmarks with reference input sets (the first 200 million instructions are skipped and the next 1 billion instructions are executed). In this experiment, the queue size of the gDiff predictor (i.e., the order) was limited to 8 and the predictions were made for all value producing integer operations or load instructions. The prediction accuracies observed based on stride predictor (unlimited size) and DFCM predictor (unlimited first level table and 64K second level table) are shown in Figure 8 for comparison.

From Figure 8, 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 the local value locality or global value locality. The reason for the low predictability is due to the hard-to-predict generational values and the long computation chain of these hard-to-predict values. If the global value queue is increased in size to 32 (thus capturing long computation chains), the prediction accuracy for gap increases to 59.7%. Overall, when compared with the predictors that exploit local value localities, the gDiff predictor performs better consistently for all the benchmarks. For benchmarks *parser* and *twolf*, gDiff increases the accuracy up to 34%. This shows that very strong stride-based value locality exists in the global value history and that the gDiff predictor has the potential to exploit it well. A detailed classification of dependencies between correlated instructions and a distribution of correlation distance are discussed in [2].

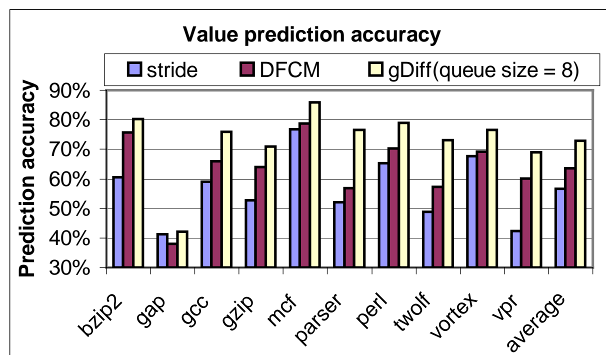

Figure 8. The prediction accuracy for gDiff predictor and local predictors.

The results in Figure 8 are obtained with unlimited prediction table size. Limiting the table size results in aliasing effect as different instructions may index to the same entry. Figure 9 shows this aliasing effect for different table sizes.

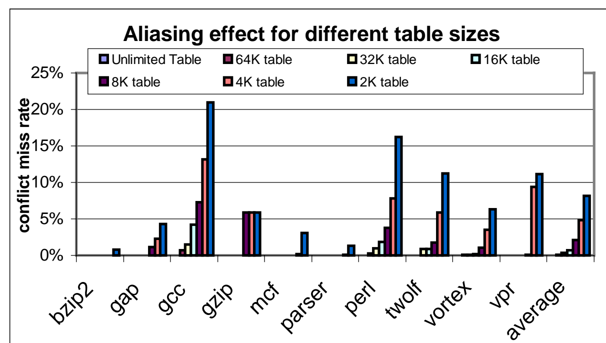

Figure 9. Aliasing effect for different table sizes.

From Figure 9, it can be seen that a table of 8K entries provides a good balance between table size and aliasing. Compared to the infinite table, the prediction accuracy of an 8K-entry table reduces by less than 1%. In the rest of

the paper, a *tagless* prediction table of *8K* entries is used for the gDiff predictor, the local stride predictor, and the first level of DFCM predictor (second level table size remains as 64K), unless stated otherwise.

3.1. Value delay

Although the profile experiments above show that high value predictability can be achieved by exploiting the global stride-based value locality, those experiments did not take any delay in value history into account. Due to pipeline delay, especially in out-of-order (OOO) execution pipelines, the correlated values may not be available when the prediction is being made. Although this issue exists for local value predictors [16, 22] (the local prediction results in Figure 16 confirm this problem), the impact for those predictors is small except for cases such as tight loop code, which calls for the speculative update based on the prediction (the importance of speculative updating branch history is observed earlier in [10]). The impact of the value delay is more dramatic for global value predictors because a value produced by one instruction usually is consumed by other instructions very close to it (i.e., the dependence distance is small). To demonstrate this effect, the value delay was modeled as a parameter *T*, where the prediction can only use the values produced *T* values before the current instruction. Figure 10 shows the prediction accuracy of the gDiff predictor for various value delays.

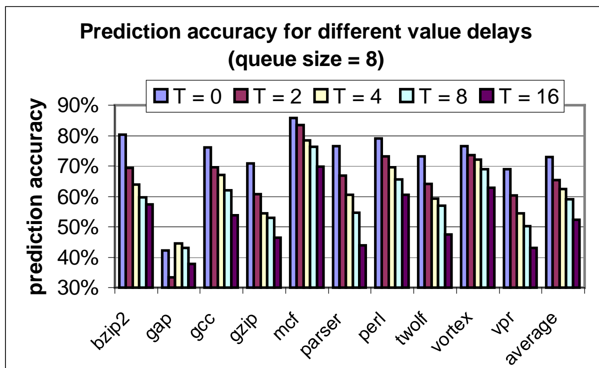

Figure 10. The prediction accuracy using the gDiff predictor with different value delays.

It can be seen that the prediction accuracy of the gDiff predictor is susceptible to value delays for all benchmarks. On average, the prediction accuracy drops from 73% to 52% when the value delay increases from zero to 16 values. (One exceptional case is the benchmark *gap*, for which the highest prediction accuracy is achieved when the value delay is 4— this is a side effect of the long computation chain in *gap* and the limited GVQ size, as discussed above.) These results show that the global value locality is strong between instructions close in time that it becomes weak when the value delay increases. This phenomenon is to be expected as the nature of global value locality is mainly based on computation chain or spill/fill sequences. A value produced out of the computation chain would have less correlation with the values in the chain. This presents a challenge to practically exploiting global value locality, be it either computational or context-based.

4. Reducing the value delay using speculative values

As discussed in Section 3, global value history based on profile runs shows very strong value locality. In an OOO execution pipeline, this value sequence is identical to what is generated at the retire/commit stage. However, using the retiring value sequence incurs very long value delay at least as much as the number of pipeline stages from dispatch to retire. For example, a 4-issue 7-stage pipeline (i.e., *fetch, dispatch, issue, register read, execution, write back*, and *retire*) may incur as many as 20 (5x4) cycles of value delay even if there are no pipeline stalls and no complex instructions (i.e., those requiring more than 1 cycle in execution). Based on the profile results above, such delays reduce the prediction accuracy of the gDiff predictor significantly.

In order to reduce value delay, one approach is to use the values produced earlier than the retire stage, even if the values are speculative (see Figure 11). In this scheme, the speculative results produced at the end of execution stage are used to update the (speculative) global value queue (SGVQ), while the prediction is made at the dispatch stage.

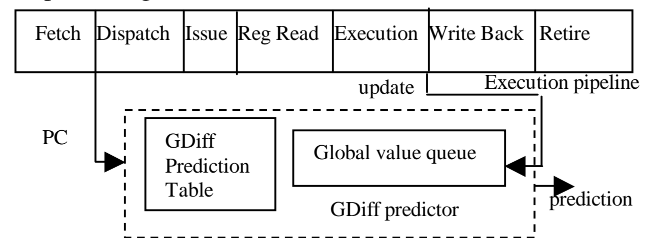

Figure 11. The gDiff predictor with speculative global value queue (SGVQ).

We used a modified out-of-order simulator from the Simplescalar toolset [3] to model the gDiff predictor using speculative values. The underlying processor organization is based on the MIPS R10000 processor, configured as indicated in Table 1. For all benchmarks, the reference input data sets are used and we fast-forward the first 500 million instructions and simulate the next 500 million instructions.

First, we examined how well using speculative values helps to reduce value delays. In the simulations, the value delay is counted as the number of values produced between an instruction's dispatch stage and its write-back

Table 1. Processor configuration.

Instruction Cache	Size = 64 kB; Associativity = 4-way; Replacement = LRU; Line size = 16 instructions (64 bytes); Miss penalty = 12 cycles
Data Cache	Size = 64 kB; Associativity = 4-way; Replacement = LRU; Line size = 64 bytes; Miss penalty = 14 cycles
Superscalar Core	Reorder buffer: 64 entries; Dispatch/issue/retire bandwidth: 4-way superscalar; 4 fully-symmetric function units; Data cache ports: 4
Execution Latencies	Address generation: 1 cycle; Memory access: 2 cycles (hit in data cache); Integer ALU ops = 1 cycle; Complex ops = MIPS R10000 latencies

stage. Figure 12 shows a typical distribution of value delays, in this case based on the benchmark *vortex*. It can be seen that in most cases the value delay is not prohibitively large and the average value delay is approximately 5. This suggests using the speculative values would be successful in reducing value delay effects.

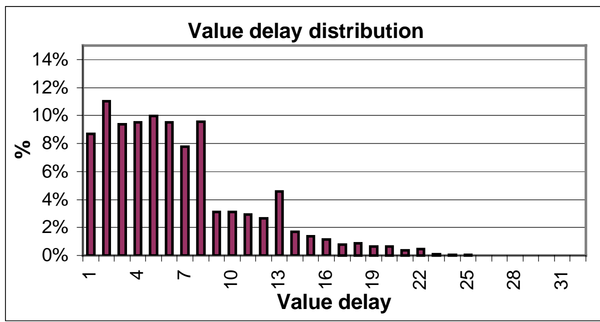

Figure 12. The distribution of value delays based on the *vortex* benchmark.

Next, we analyzed the performance of the gDiff predictor with SGVQ. In this experiment (and all that follow below), a 3-bit confidence mechanism is used to filter the 'weak' predictions. The confidence mechanism works as follows: when a correct prediction is made, confidence is increased by 2; and, it is decreased by 1 if an incorrect prediction is found [28, 30]. A confident prediction is made when the confidence is larger 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 13 shows the simulation results of the gDiff predictor with SGVQ.

In Figure 13, the prediction accuracy and the prediction coverage using a local stride predictor are also shown for comparison. Based on these results, it can be seen that even with reduced value delays, the gDiff prediction results are not as encouraging as the previous profile run results: the average accuracy is 74% and the coverage is 49%, whereas the local stride predictor has 89% accuracy and 55% coverage. The reason behind this is that there are significant execution variations due to cache misses. The impact of execution variations on the gDiff predictor can be explained using the examples shown in Figure 14.

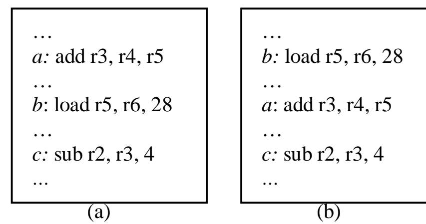

Figure 14. Two code examples to show impact of execution variation due to cache misses: (a) load misses, (b) no load miss occurs.

In Figure 14 (a) and (b), instruction *c* has a strong correlation with the instruction *a*. However, due to the load instruction *b*, the distance between the instructions *a* and *c* may vary based on the hit/miss pattern of the load. Also, I-cache misses and branch mispredictions affect the dynamic scheduling of the instructions, which in turn

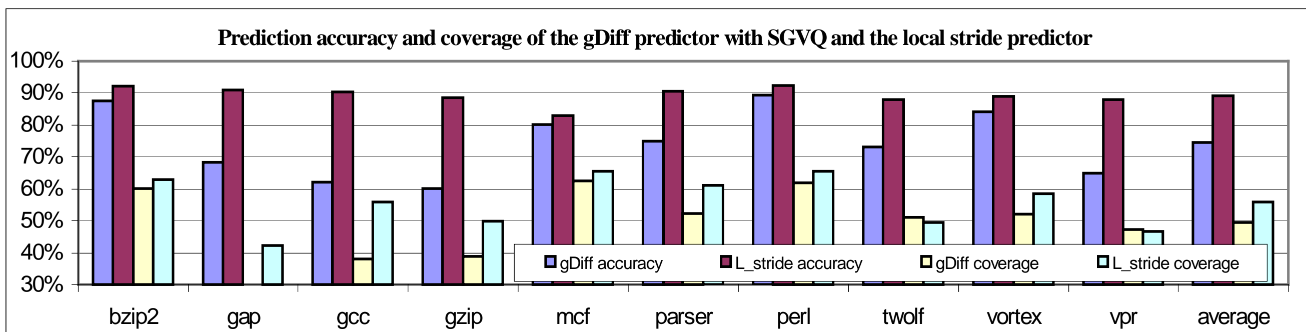

Figure 13. The prediction accuracy of the gDiff predictor with speculative values (queue size = 32) .

accounts for another source of variation in the speculative global value history. Finally, note that in the gDiff implementation of Figure 11, the SGVQ is updated based on speculative execution results and does not squash the values in the case of a branch misprediction.

5. The gDiff predictor with hybrid global value queue

As discussed in Section 4, using the speculative values at execution stage helps to reduce value delay but introduces the problem of variations in the speculative global value queue. In this section, a new gDiff-based hybrid predictor is proposed. This predictor can significantly reduce the impact of pipeline execution variations from cache misses and enhance the performance of the predictor by exploiting more than one type of value locality.

Pipeline execution variations are caused by dynamic run-time events, such as cache misses and mispredictions. Those run-time events affect the dynamic scheduling so that the execution order of instructions may not be the same over different iterations. To remove the variations in the global value queue, the global value 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 the problem, we propose to use another type of speculative value to construct the global value sequence at dispatch time and then update the sequence at write-back time. As gDiff explores global stride-based value locality, a value predictor based on a *different* type of locality, e.g., a local stride predictor or a local context predictor, is appropriate to generate the values at dispatch stage and temporarily fill the entries in the ordered queue. The scheme is shown in Figure 15.

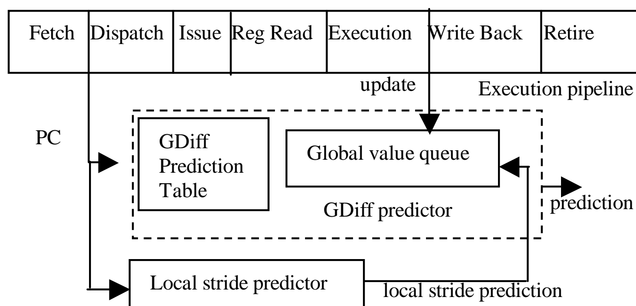

Figure 15. The gDiff predictor with hybrid global value queue (HGVQ).

In Figure 15, the predictions based on the local stride predictor are pushed into the global value queue (GVQ) at dispatch time and updated with the execution results at write-back time. As GVQ contains a hybrid of values from local stride prediction and speculative global execution results, it is called hybrid global value queue (HGVQ). A field is associated with each instruction in the issue queue (or RUU) to direct which entry in the HGVQ the result should update. The performance of the gDiff predictor with HGVQ was evaluated and the simulation results (using the same simulation methodology as in Section 4) are shown in Figure 16.

From Figure 16, it can be seen that the prediction capability of the gDiff predictor is greatly enhanced by the hybrid queue structure. For benchmarks *bzip2*, *gap, gzip*, *mcf*, *parser*, and *perl*, the confident predictions achieve over 90% accuracy, while other benchmarks show 86% to 89% accuracy. Compared with the local stride predictor with the same confidence mechanism, the results show better average prediction accuracies (91% vs. 89%) and higher coverage (64% vs. 55%). This demonstrates that the gDiff predictor captures global stride locality over and above local stride locality. For the local context predictor (DFCM), the confidence gating results smaller prediction coverage compared to local stride or gDiff predictors while the accuracy is in the similar range. Note that in this experiment, all predictors make predictions at dispatch stage and are updated at write-back stage (i.e., speculative update based on prediction is not performed for local stride and local context predictors).

There are several factors accounting for the strong performance of the HGVQ-based gDiff predictor. First, it maximizes opportunities to find global stride-based value locality by constructing the value sequence in instruction dispatch order and by using speculative values to fill in holes in the history. Assembling the value sequence in dispatch order eliminates the execution variation problem due to cache misses; and, the speculative values help to reduce the effective value delay. Secondly, the HGVQ provides an efficient way to exploit two types of value localities (local stride and global stride). For example, if two instructions, *a* and *b*, are both predictable using a local stride predictor and are close to each other (see Figure 17), then clearly there is a global stride-type correlation between the two instructions. However, the ordinary GVQ-based gDiff predictor would fail to exploit this locality since the first load must finish before the dispatch of second load (i.e., due to the value delay of the instruction *a*). With the local stride prediction, although instruction *a* is still in the execution pipeline, the correct prediction (a gDiff-based prediction) of instruction *b* can be made based on the prediction of *a* (a local stride-based prediction). In general, the HGVQ does more than make predictions of a few dependent instructions based on a local prediction possible. Due to the nature of the stride type locality, dependent instructions (e.g., instruction *b* in Figure 17) should be locally predictable as well if the local stride prediction is correct (i.e., those predictions do *not* account for the highly improved coverage shown in Figure 16). The increased coverage comes from the

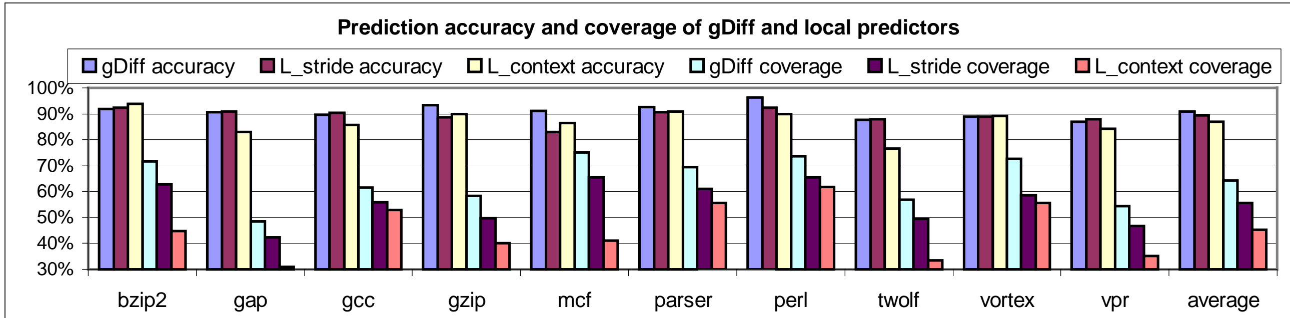

Figure 16. The performance of the gDiff predictor with HGVQ (queue size = 32).

instructions with low local value locality but high global locality, and this added predictive power is one of the most important contributions of the gDiff predictor.

```
…
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 17. The code example to show how the gDiff predictor with HGVQ uses the local prediction.

6. Using gDiff to predict load addresses

The gDiff predictor provides a *general framework* for exploiting global stride locality for *any* value stream. By allowing only load addresses to pass into the GVQ, gDiff detects global stride locality in the load address stream. In this experiment, the address prediction is made at the dispatch stage and updated at the address generation stage for all load instructions. The gDiff and local stride predictors use a tagless, 4K-entry prediction table, while the first-order Markov predictor [13] uses a 4-way, 256K-entry prediction table. For local and gDiff predictors, the prediction coverage is computed as the ratio of confident predictions over total predictions. For the Markov predictor, there is no confidence counter and the confidence gating is achieved with tag matching. Figure 18 shows the prediction capability achieved by each predictor for all load addresses and for addresses of missing loads.

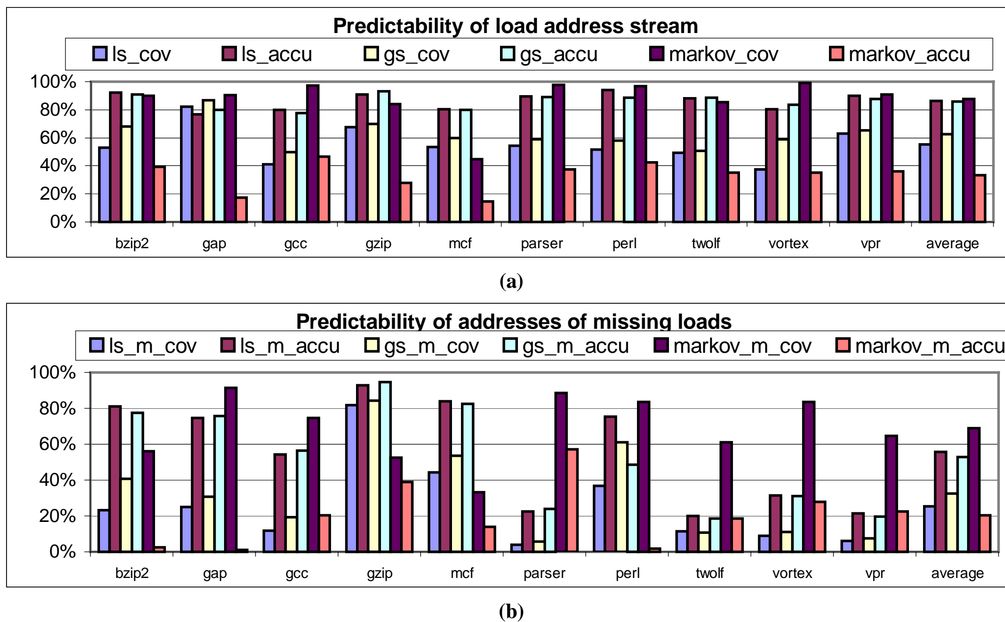

Figure 18. (a) Load address predictability. (b) Predictability of missing loads. (ls: local stride; gs: gDiff; accu: accuracy; cov: coverage; m: missing load).

Table 2. Baseline IPC results (for a 4-way machine model with a 64-entry issue window).

Benchmarks	bzip2	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr
Baseline IPC	2.02	1.23	1.96	1.77	0.66	1.27	1.50	0.99	2.26	1.30

Two important observations can be made from Figure 18. First, gDiff achieves much higher combination of prediction accuracy (86%) and coverage (63%) for load address prediction compared to either local stride (accuracy 86% but coverage 55%) or Markov predictors (accuracy 33% although coverage 87%). The global stride locality detected by gDiff can be used to facilitate the reduction of load-use latency [1]. Secondly, gDiff also performs the best (accuracy 53% and coverage 33%) in predicting addresses of missing loads, while the local stride predictor provides a prediction accuracy of 55% but coverage of only 25%. The Markov predictor has a much higher coverage 69% but a fairly low accuracy of 20%. The Markov predictor usually requires a large prediction table as it is indexed with load addresses. When its size increases from 256K-entry to 2M-entry, the Markov predictor achieves decent average coverage (92%) and accuracy (33%) but still shows much lower prediction capability than gDiff for benchmarks including *bzip2*, *gap*, *gzip* and *perl*. GDiff, on the other hand, provides a relatively cost-effective way in predicting addresses of missing loads when compared with these other schemes. This motivates us to extend gDiff for memory prefetch. These extensions are out of the scope of this paper and left as future work.

7. The performance potential of value prediction using the gDiff predictor

Since the gDiff predictor presents promising prediction results as seen in Figure 16, we investigated the performance impact of using it to break data dependencies. The purpose here is to show the performance potential of the proposed prediction scheme. As such, an aggressive machine model is used, similar to the 'great' latency model described in [24]. The machine can issue branch instructions speculatively and to perform selective reissuing in the case of mispredictions. The performance results based on a 4-wide, 64-entry issue queue machine model (same as what was used in Section 4) are shown in Figure 19 and the baseline IPC results (IPC without value speculation) are shown in Table 2. In the experiment, the local stride and context predictions are made at dispatch stage and they are updated at write-back stage. From the results, we can see that the local context predictor does not perform as well as the local stride predictor. The main reason is due to the small prediction coverage of the local context predictor, as shown in Figure 16. So, we focus on the comparison between the gDiff and the local stride predictor.

From Figure 19, it can be seen that the improved prediction accuracy and coverage of the gDiff predictor show significant performance potential. Taking the benchmark *mcf* as an example, the increased coverage (from 65% to 75% of all the value producing integer instructions) results in a speedup of 17% over the baseline machine with a local value predictor and a speedup of 53% over the baseline model without value speculation. The main reason for the significant speedup is that gDiff can predict many missing load values (71.65% coverage and 88.63% accuracy for all missing loads) correctly to enable more dependent instructions to execute compared to local stride predictor (63.61% coverage and 87.64% accuracy for all missing loads). As *mcf* is highly memory intensive (L1 D-cache miss rate 44.08%), a large window size of 64 enables more missing loads to be predicted leading to higher speedups. Another important reason is that gDiff can accurately capture the stride type of locality (single stride or phased multi-stride) between two load addresses if those two address-generating instructions exist in the HGVQ at the same time. As pointed out in [26], many important loads (i.e., loads that tend to miss) have strong stride relationship between their addresses as an artifact of dynamic memory allocation. The correct

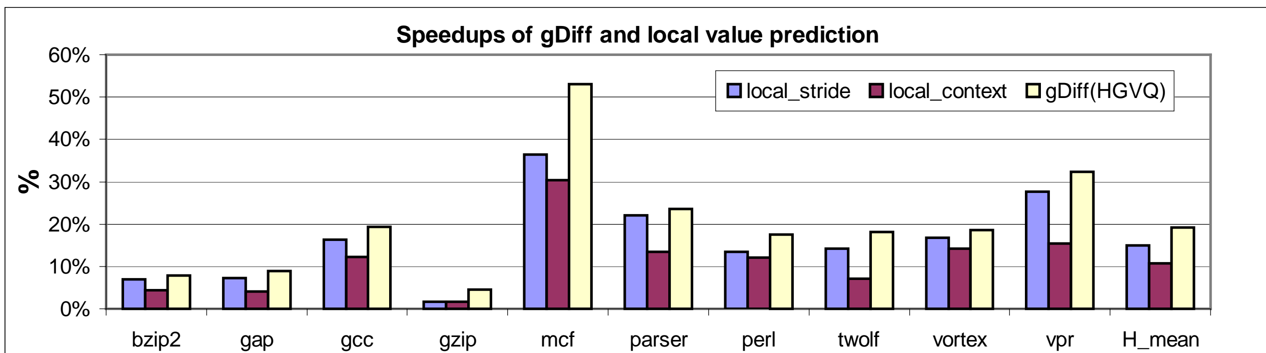

Figure 19. The speedups of value prediction using the local stride and gDiff predictors.

prediction of the load address enables the dependent load to be issued much earlier to overlap the miss latency. However, not all the coverage increase has the same impact on performance for different benchmarks. For example, the gDiff predictor provides 15% prediction coverage increase (from 62% to 71%) for the benchmark *bzip2* compared to the local value predictor, while the resulting speedup increase is just 1%. This implies additional value predictions based on the gDiff predictor does not help in reducing the critical path of the benchmark *bzip2*. On average, the gDiff predictor results in a 19.2% speedup over the baseline machine and a 4% (from 15% to 19%) speedup over the baseline machine with a local stride predictor[1].

8. Conclusion

In this paper, a new type of value locality, global computational locality, is studied and a set of prediction schemes are proposed to exploit this locality and to increase overall value predictability. The main contributions of this work include:

- A new type of value locality is formalized and studied. The localities in global value history present new opportunities to be explored. A novel 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 addressed in this paper and its impact on gDiff predictor is studied. It is shown that value delay is a challenge for any scheme that seeks to exploit global value locality, especially in out-of-order execution pipeline models.
- To reduce the value delay impact in OOO pipelines, this paper proposes using the speculative values at the execution stage instead of waiting for them to be retired in-order. This approach reduces value delay but introduces the execution variation problem. Those variations make it difficult for the gDiff predictor to find global stride locality.
- In order to reduce both value delay and pipeline variation impact, this paper proposes construction of a partially speculative global value sequence at instruction dispatch time using another type of value predictor (e.g., the local stride predictor). The correct values produced after execution are used to update the value sequence. In this way, the gDiff predictor maximizes exploiting global value locality and enables an efficient integration of a different type of value locality. The experiments show that the gDiff predictor achieves an impressive 91% prediction accuracy with 64% coverage. We then demonstrate the usage of gDiff prediction scheme to predict load address stream. The results show that global stride locality detected by gDiff leads to strong capabilities in predicting all load addresses and in predicting addresses of missing loads. The gDiff predictor can also be used to break true data dependencies; and, it shows impressive performance potentials in a 4-wide OOO machine with a 64-entry issue window.

There are several directions for the future work. One interesting work is to extend gDiff to further explore global stride locality in load address stream for memory prefetch and for reducing load-use latency. Another direction would be to study in detail how to interact with the deeper pipeline [27] to convert the newly discovered predictability into higher speedups.

9. Acknowledgement

This work was supported by NSF awards CCR-0208596 and CCR-0072926 and a hardware donation from Hewlett-Packard.

10. References

[1] M. Bekerman, S. Jourdan, R. Ronen, G Kirshenboim, L. Pappoport, A. Yoaz, and U. Weiser, “Correlated Load-Address Predictors”, in *International Symposium on Computer Architecture* (ISCA-26), 1999.

[2] J. T. Bodine, “Exploiting computational locality in global value histories”, *MS. Thesis, ECE Department, N. C. State University*, 2002.

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

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

[5] B. Fields, S. Rubin, and R. Bodik, “Focusing processor policies via critical-path prediction”, in *International Symposium on Computer Architecture* (ISCA-28), 2001.

[6] 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.

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

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

[1] (This suggests that a further enhancement to gDiff would combine it with a critical path predictor [5, 29], but such an extension is beyond the scope of this paper.)

[9] B. Goeman, H. Vandierrndonck, and K. D. Bosschere, "Differential FCM: Increasing value prediction accuracy by improving table usage efficiency," in *7th International Symposium on High-Performance Computer Architecture* (HPCA'01), Jan 2001.

[10] E. Hao, P-Y Chang, and Y. Patt, "The effect of speculatively updating branch history on branch prediction accuracy, revisited", in *27th International Symposium on Microarchitecture* (MICRO-27), 1994.

[11] T. Heil, Z. Smith, and J. E. Smith, "Improving branch predictors by correlating on data values", in *32nd International Symposium on Microarchitecture* (MICRO-32), 1999.

[12] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons", in *7th International Symposium on High Performance Computer Architecture* (HPCA-7), 2001.

[13] D. Joseph and D. Grunwald, "Prefetching using Markov Predictors", *IEEE Transactions on Computers*. Vol. 48, Feb 1999.

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

[15] 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.

[16] 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.

[17] M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value prediction," in *29th International Symposium on Microarchitecture* (MICRO-29), 1996.

[18] 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.

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

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

[21] G. Reinman and B. Calder, "Predictive Techniques for aggressive local speculation", in *31st International Symposium on Microarchitecture* (MICRO-31), 2000.

[22] 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.

[23] R. Sathe and M. Franklin, "Available parallelism with data value prediction", in *High Performance Computing HiPC-5*, 1998.

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

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

[26] M. J. Serrano and Y. Wu, "Memory performance analysis of SPEC2000 for the Intel Itanium Processor", in *4th Workshop on Workload Characterization*, 2001.

[27] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines", in *29th International Symposium on Computer Architecture* (ISCA-29), 2002.

[28] R. Thomas and M. Franklin, "Using dataflow based context for accurate value prediction", in *International Conference on Parallel Architectures and Compilation Techniques* (PACT'01), 2001.

[29] 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.

[30] K. Wang and M. Franklin, "Highly accurate data value prediction using hybrid predictors," in *30th International Symposium on Microarchitecture* (MICRO-30), Nov. 1997.

[31] Y. Wu, D. Chen, and J. Fang, "Better exploration of region-level value locality with integrated computation reuse and value prediction", in *International Symposium on Computer Architecture* (ISCA-28), 2001.