The Effect of Instruction Fetch Bandwidth on Value Prediction

Freddy Gabbay and Avi Mendelson
Department of Electrical Engineering
Technion - Israel Institute of Technology,
Haifa 32000, Israel.
{fredg@psl, mendlson@ee}.technion.ac.il

Abstract

Value prediction attempts to eliminate true-data dependencies by dynamically predicting the outcome values of instructions and executing true-data dependent instructions based on that prediction. In this paper we attempt to understand the limitations of using this paradigm in realistic machines. We show that the instruction-fetch bandwidth and the issue rate have a very significant impact on the efficiency of value prediction. In addition, we study how recent techniques to improve the instruction-fetch rate affect the efficiency of value prediction and its hardware organization.

1. Introduction

The fast growing density of gates on a silicon die, allows modern microprocessors to increasingly employ multiple execution units that are capable of executing several instructions in parallel. Most of the recent microprocessor architectures assume sequential programs as an input and a parallel execution model, where the hardware is expected to extract the parallelism at run-time out of the instruction stream. The efficiency of such architectures is highly dependent on both the hardware mechanisms and the programs' characteristics; i.e., the instruction-level parallelism (ILP) the programs exhibit ([11]). Programs' characteristics affect the ILP in the sense that instructions cannot always be ready for parallel execution due to three types of constraints: *true-data dependencies*, *name dependencies (false dependencies)* and *control dependencies* ([11], [25]). Both control dependencies and name dependencies are not considered an upper bound on the extractable ILP, since they can be handled or even eliminated in several cases by various hardware and software techniques ([2], [4], [5], [11], [12], [23], [24], [26], [27]). As opposed to name dependencies and control dependencies, only true-data dependencies were considered to be a fundamental limit on the extractable ILP since they reflect the serial nature of a program by determining the sequence in which data should be passed between instructions. This kind of extractable parallelism is represented by the *dataflow graph of the program* ([11]).

Recent studies ([13], [14], [7], [8]) have proposed a novel *hardware-based* paradigm, termed value prediction, that allows *superscalar* processors to exceed the limits of true-data dependencies. This paradigm was able to eliminate true-data dependencies by predicting at run-time the outcome values of instructions and executing the true-data dependent instructions based on that prediction. As a result, it has been shown that the limits of true-data dependencies can be exceeded without violating the sequential program correctness.

Lipasti *et al.* reported in [14] that value prediction can improve the performance of the PowerPC 620 (running the Spec92 benchmarks) by approximately 5% (on average). In addition, they examined the performance speedup of value prediction in an "infinite machine model" and reported that value prediction can gain approximately 30% speedup (on average). Their infinite machine model was not "ideal" since it was limited both by the branch prediction accuracy and the fetch bandwidth (single taken branch per cycle). In our previous studies ([7], [8]) we examined the performance gain of value prediction when using an ideal machine model, i.e., a machine that is only limited by the true-data dependencies appearing in the program, and by the instruction window size. Such a machine was also assumed to be free from other constraints, like control dependencies, name dependencies and structural resources conflicts. We reported that value prediction can increase the performance of such a machine by approximately 70% (on average) and in several benchmarks the increase was even more than 200%. Even though we used a different set of benchmarks (Spec95) and different prediction mechanisms, it seemed that this was not the only reason for the significant difference between these reported observations. In order to be more certain about these observations, we even measured the performance gain of a realistic processor under our simulation environment. Our observations were found to be similar to those reported by Lipasti *et al.* In this paper we try to find answers to these observations, i.e., to explore the differences between the performance gain of an ideal machine and a realistic processor. Our study focuses on two main aspects: First, we explore the limitations of using value prediction in realistic machines. Second, with regard to future microprocessor architectures, we search for the environment and the

conditions which would allow next-generation processors to gain more benefits from value prediction. As opposed to all previous works in this area that focused on studying the characteristics of value prediction either in present-generation processors or in an ideal processor, in this paper we intend to focus on next generation processors.

The effectiveness of value prediction can be examined in different aspects, e.g., the prediction accuracy, the overall performance gain etc. In this paper we introduce another aspect that has not been studied yet in the previous studies in this area. We show that not all the value predictions during the run of a program can be exploited even though they are correct (notice that a similar problem was identified by [10] and [21] regarding load address prediction). There are a significant number of cases where the dependent instructions are fetched too late to the processor and all their input values become ready (since they are truly computed). In all these cases, even though the predictor yields a correct prediction, the prediction becomes useless.

We show in this paper that the effectiveness of value prediction is significantly affected by the instruction-fetch bandwidth. In general, when examining the effect of the instruction-fetch bandwidth in our analysis we also involve the impact of the entire processor bandwidth (dispatch and issue rate). Particularly, the effective instruction-fetch bandwidth is limited by several major factors ([18]): the instruction cache hit rate, the branch prediction accuracy, the branch misprediction penalty, the branch prediction throughput (number of branches predicted per cycle) and number of taken branches in the instruction stream. The last factor introduces the problem of noncontiguous instruction fetching, i.e., the dynamic instruction stream resides in the cache but not in contiguous locations. In general the effect of the instruction-fetch rate on ILP (when value prediction was not employed) has been broadly studied by previous studies ([6], [16], [17], [18], [19]). In this paper we intend to focus only on its impact on the effectiveness of value prediction and show that its significance may not be straightforward. In addition, we do not intend to examine the impact of all the previously mentioned factors on value prediction. Rather we focus only on those factors related to the program control flow.

The work presented in this paper consists of two major parts. In the first part we explore the inherent properties of true-data dependencies in computer programs which limit the effectiveness of value prediction in current microprocessor architectures. In the second part we examine the effectiveness of value prediction in future microprocessor which employ high-bandwidth instruction-fetch mechanisms, e.g., the trace cache ([18]). In addition, we also discuss the high-level hardware revisions needed in order to combine value prediction in such microprocessors.

The rest of this paper is organized as follows: Section 2 summarizes previous studies related to this paper. Section 3 explores the fundamental effect of instruction-fetch rate on the efficiency of value prediction. Section 4 discusses the limitations of using the conventional value prediction hardware in high-bandwidth instruction-fetch processors and proposes hardware solutions to overcome these limitations. Section 5 presents performance analysis and Section 6 concludes this paper.

2. Previous related studies

This section summarizes some of the previous studies related to the scope of this paper. In the first subsection we survey the related works in the area of value prediction and in the second subsection we survey recent works related to high-bandwidth instruction-fetch processors.

2. 1.Recent studies in the area of value prediction

The pioneer papers in the area of value prediction ([13], [14], [7], [8]) reported that many of the data values produced during the execution of instructions in a program tend to be predictable. These works introduced two different fundamental value prediction methods: last-value prediction ([13], [14]) and stride value prediction ([7], [8]). Last-value prediction predicts the destination value of an individual instruction based on the most recent value it has generated (or computed), while stride value prediction makes the prediction based on recently seen value plus a calculated stride. The stride value is the delta between two recent consecutive destination values. The performance potential of the prediction methods was examined on both a realistic model of a present microprocessor ([14]) and on an ideal execution environment ([7]) that is only limited by its instruction window size and the true-data dependencies in a program. Further study of different value prediction methods was also presented in [22].

Various statistical characteristics of value prediction were explored in [7] and their significance was presented as well. One of these important properties indicates that the tendency of instructions in a program to be value predictable does not spread uniformly. This property established the basis for using a classification mechanism which assigns confidence to the predictions made by the value predictor. An early classification method was hardware-based ([14]) and in a later paper ([9]) an alternative profiling-based method was proposed and examined.

2. 2.Recent high-bandwidth instruction-fetch mechanisms

Various recent studies have proposed and examined different high-bandwidth instruction-fetch mechanisms.

They all attempt to fetch multiple basic blocks (possibly noncontiguous) in each cycle.

The *Branch Address Cache* was proposed in [28] as an extension of the branch target buffer ([23], [27]). The branch address cache was capable of producing multiple branch prediction of the next basic blocks to be executed. The multiple target addresses that the branch address cache produced were sent to a highly interleaved instruction cache in order to be fetched in a single cycle. A similar approach to the branch address cache was also presented in [3] and allowed the use of more accurate branch predictors. Another scheme was proposed in [1] that allowed fetching two noncontiguous instruction cache lines. In addition it used a *collapsing buffer* in order to detect short branches within a cache line and purge the instructions between the branch and the target. The *fill-unit* was proposed in [15] and suggested caching RISC instructions that were driven from a CISC instruction stream. As a result this mechanism allowed effective fetching of more instructions each cycle in case of hit. In [17] the trace cache concept was first introduced. This concept was further explored in [18] and compared to other high-bandwidth instruction-fetch mechanisms. The trace cache concept suggests using a special instruction cache capable of capturing dynamic instruction stream (trace). A trace cache block can store multiple noncontiguous basic blocks which may contain more than one taken branch. Further studies of trace cache related issues were also presented in [16] and [19]. In addition, other alternative techniques, *partial matching* and *inactive issue*, were presented in [6] for improving the effective fetch rate of the trace cache.

3. The fundamental effect of instruction-fetch rate on the efficiency of value prediction

In this section we initially study the fundamental influence of the instruction-fetch rate on the effectiveness of value prediction. In addition, we explore the inherent characteristics of true-data dependencies in programs that induce the linkage between the instruction-fetch bandwidth and the effectiveness of value prediction.

3. 1.Simulation methodology

For our experiments we use the 8 Spec95 integer benchmarks (table 3.1). All benchmarks were compiled with the *gcc 2.7.2* compiler with *all available optimizations*. The benchmarks' traces, that were driven to our simulator, were produced by the *Shade* simulator ([20]) running on Sun-Sparc processor. Each benchmark was traced for 100 million instructions. We found that using longer traces, in term of the instruction count in the trace, barely affects the results of our experiments and therefore this trace length satisfied our needs.

SPEC95 Integer Benchmarks	
go	Game playing.
m88ksim	A simulator for the 88100 processor.
gcc	A GNU C compiler version 2.5.3.
compress95	Data compression program using adaptive Lempel-Ziv coding.
li	Lisp interpreter.
ijpeg	JPEG encoder.
perl	Anagram search program.
vortex	A single-user object-oriented database transaction benchmark.

Table 3.1 - Spec95 integer benchmarks.

Our current simulation methodology is implementation-independent. It assumes an ideal machine model that is only limited by true-data dependencies in the program and the instruction window size (limited to up to 40 instructions). This machine is assumed to be free from control dependencies, name dependencies and structural resources conflicts. In addition, in each of the experiments in this section, we artificially limit the fetch / issue rate to up to 4, 8, 16, 32 or 40 instructions per cycle (the number of taken branches per cycle is unlimited). Our value prediction mechanism consists of the stride predictor ([7], [8]) and a classification unit that employs a set of saturated counters ([14], [8]) to increase the confidence in the prediction. Both the prediction table and the set of saturated counters are assumed to be infinite. In addition, it is assumed that the value predictor is updated speculatively after the lookup, and in case of an incorrect update the correct value is stored in the prediction table as soon as it is known. We believe that the usage of such a simulation methodology allows us to remove implementation-dependent characteristics and enables us to gain better understanding of the examined phenomenon. In further sections we intend to involve some of the microarchitecture aspects in our analysis.

3. 2.The effect of instruction-fetch rate in an ideal execution environment

Figure 3.1 illustrates the speedup gained by using value prediction (under the assumptions of subsection 3.1) for machines with different instruction-fetch rates. Note that the speedup represents the contribution of value prediction only, since the performance gain of value prediction is measured relative to a machine with the same instruction-fetch rate. It can be clearly observed that the instruction-fetch bandwidth dramatically affects the performance gain of value prediction. When the instruction fetch rate is limited to up to 4 instructions per cycle the speedup is barely noticeable on the chart. When increasing the instruction-fetch rate to up to 8, 16, 32 and 40 instruction per cycle, value prediction yields a performance

speedup (average) of 8%, 33%, 70% and 80% respectively. In the benchmarks *m88ksim* and *vortex*, the effect of increasing the instruction-fetch bandwidth on the speedup gained by value prediction is very significant. For instance, when increasing the instruction fetch rate from 4 to 16 instruction per cycle, the speedup gained by value prediction moves from 4% to 112% in *m88ksim*, and from 1.5% to 83% in *vortex*. In the other benchmarks like *go*, *gcc*, *li* and *perl* the effect of the instruction-fetch rate is also significant. In the benchmark *gcc*, for instance, increasing the instruction fetch rate to up to 8, 16, 32 and 40 instructions per cycle yields a performance speedup of 2%, 14%, 32% and 34% respectively. These observations clearly show that the instruction-fetch bandwidth has a significant impact on the obtainable speedup gained by value prediction. In addition, they also establish the first indication which strengthens our main claim in this paper - the effectiveness of value prediction is best observable when the processor employs high-bandwidth instruction-fetch mechanisms.

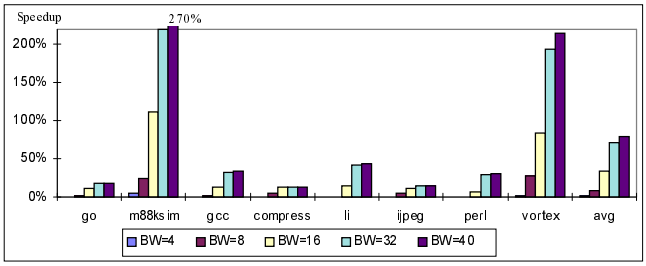

Figure 3.1 - The effect of instruction-fetch rate in an ideal execution environment.

3. 3.The Dynamic Instruction Distance (DID)

In order to understand the origin of this observation we examine some of the inherent properties of true-data dependencies in programs. A convenient way to illustrate true-data dependencies in a program is the dataflow graph representation ([11]). The dataflow graph (*DFG*) is a directed graph *G(V,S)*, where each node, $v \in V$, represents a single instruction, and each arc, $s_{ij} \in S$, represents a true-data dependency between two nodes *i* and *j* ($i,j \in V$). The DFG that we use in our analysis is not limited to a single basic block in a program. We construct the DFG out of the entire execution trace of the program, regardless of basic block boundaries. The main advantage of such representation is that it also includes loop-carried dependencies and inter-basic-blocks data dependencies. To each node, $v \in V$, we assign an individual number that represents the appearance order of the instruction in the trace of the program. Once each node has been assigned to its appearance order number, we can define a new term - *dynamic instruction distance*. A dynamic instruction distance (DID) of an arc, $s_{ij} \in S$ between nodes *i* and *j* ($i,j \in V$), is defined as illustrated by equation 3.1:

$$DID(S_{ij}) = |j-i|$$

Equation 3.1 - The dynamic instruction distance.

Figure 3.2 illustrates a DFG where each node was assigned to its appearance order number in the trace and each arc (data dependency) was assigned to its corresponding DID.

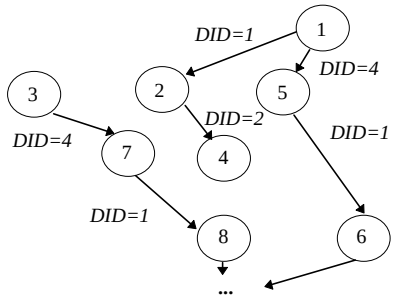

Figure 3.2 - DID measured on a data-flow graph.

The DID of a given arc in the DFG represents the distance, in terms of number of instructions, between the producer of a value and the consumer of that value. Measuring this distance for the data dependencies in a program can provide us with a substantial understanding about the impact of instruction-fetch bandwidth on the efficiency of value prediction. If the distance between the producer of a value and the consumer(s) of that value is very long relative to the instruction-fetch rate then it is very unlikely that both the producer and the consumer will be fetched at the same cycle, and as a result they are even unlikely to be executed at the same cycle together. In that case it is likely that when the consumer is issued, its input values are already available and hence value prediction is not needed in this case. On the other hand, if the distance between the producer and the consumer is short relative to the instruction-fetch rate, it is likely that both the producer and the consumer will be fetched at the same cycle. In addition when the consumer is issued it is also likely that its input value is not ready and therefore value prediction can serve this case. In order to illustrate these principles we use the DFG example in figure 3.2 and execute it on a machine with 4-instruction fetch / issue bandwidth. We assume that the machine's pipeline is organized in four stages: Fetch, Decode / Issue, Execute and Commit. We also ignore all name dependencies, control dependencies, structural conflicts and memory wait cycles. For simplicity we assume that the execution latency of each instruction is a single cycle. In addition, in order to focus on the impact of instruction-fetch bandwidth on efficiency of value prediction only and eliminate the effect of the prediction accuracy, we assume a perfect value predictor (with accuracy of 100%). In table 3.2 we illustrate the progress of instructions through the pipeline of such a machine. In the first cycle four instructions (1, 2, 3 and 4) are fetched from memory. In the second cycle 4 new instructions are fetched (5, 6, 7 and 8) and the first 4 instructions are passed to the

decode / issue stage. When issuing these instructions, only the dependent instructions (instructions 2 and 4 which depend on instructions 1 and 2 respectively), use value prediction and therefore they can also be executed in the third cycle. When issuing instructions 5, 6, 7 and 8, it can be observed that only instructions 6 and 8 will use value prediction. The input values of these instructions are not ready because they depend on values that have not been computed yet (produced by instructions 5 and 7 respectively). On the other hand, although instructions 5 and 7 are dependent on instructions 1 and 3 respectively they will not use value prediction since when they issue all their input values become ready. It can also be observed that the DID of the data dependencies associated with these instructions is greater than or equals the instruction-fetch rate (4 instructions), and hence we could expect that when they are issued their input values are ready.

Cycle	Fetch	Decode/Issue	Execute	Commit
1	1, 2, 3, 4			
2	5, 6, 7, 8	1, 2, 3, 4		
3		5, 6, 7, 8	1, **2**, 3, **4**	
4			5, **6**, 7, **8**	1, **2**, 3, **4**
5				5, **6**, 7, **8**

Table 3. 2 - An example of instructions progressing in a pipeline.

In figure 3.3 we present measurements of the average DID for Spec-int95 benchmarks. In order to calculate the average DID we constructed a DFG as illustrated by figure 3.2 for each benchmark, and for each arc in the graph we computed its corresponding DID. The average DID was obtained as the arithmetic average of the DID's among all the arcs in the DFG. It can be observed in figure 3.3 that all the programs exhibit an average DID that is greater than the instruction-fetch bandwidth of present processors, i.e., 4 instructions per cycle. These measurements provide the explanation for our observations in figure 3.1. In particular, they explain why the usage of value prediction in present processors, which are limited in their instruction-fetch bandwidth, is not expected to yield a very significant performance gain. When the instruction-fetch rate is low, a significant portion of the data dependencies spans across instructions that were fetched consecutively. As a result these instructions will be eventually issued and executed in a sequential manner, not because of the dependencies between them (which could be potentially eliminated by using value prediction), but because they are fetched to the processor in a sequential manner. When the effective instruction-fetch rate is increased, more data dependencies are likely to span across instructions that were fetched simultaneously, and as a result they are more likely to be issued simultaneously. In this case, the effect of the dependencies on the execution rate is much more significant, and thus the effect of value prediction in these cases is expected to be more noticeable.

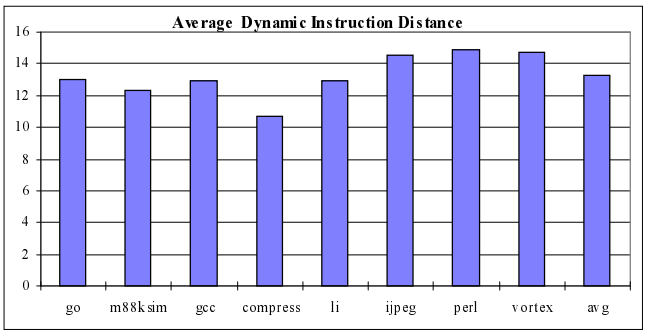

Figure 3.3 - Average DID measurements.

In figure 3.4 we illustrate as histograms the distribution of the measured DID's in the DFG for each of the benchmarks. It can be seen that approximately 60% (on average) of the true-data dependencies span across instructions in a greater or equal distance of 4 instructions. This implies that in current processors (limited to fetch up to 4 instruction per cycle) even if we had an ideal value predictor, approximately more than half of the dependencies in a program would not use the prediction, since as the dependent instructions are issued their input values become ready.

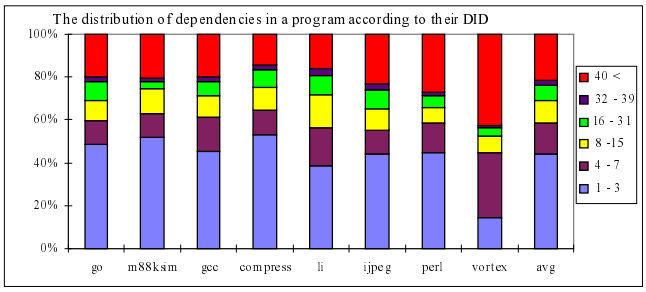

Figure 3.4 - The distribution of dependencies in a program according to their DID.

Our next step in this analysis is to explore the correlation between the DID and the value predictability of instructions. Figure 3.5 summarizes these measurements and strengthens our previous claims. We draw up a histogram that illustrates the distribution of the true data dependencies attempted to be value-predicted (using an infinite stride prediction table). We measured the fraction of the unpredictable dependencies and the predictable ones. Only for the value-predictable dependencies did we illustrate their fraction according to the DID. These histograms can be obtained by using the previous DFG's that were used to calculate the average DID in figure 3.3. We can scan all the arcs in the DFG and mark only the value-predictable arcs (representing dependencies which can be potentially eliminated). Once all the arcs in the DFG

have been scanned, we scan only the arcs that were marked as predictable and draw a histogram according to their DID.

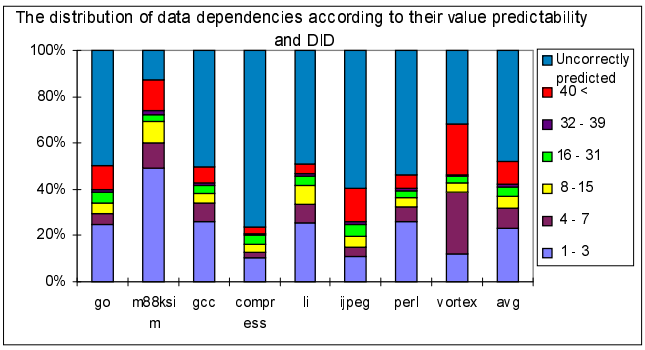

Figure 3.5 - The distribution of data dependencies according to their value predictability and DID.

It can be observed from this figure that the benchmarks which are expected to be most affected when increasing the instruction fetch rate (when using value prediction) are *m88ksim* and *vortex*. The fraction of data dependencies which are value predictable and have a DID greater than or equal to 4 is approximately 40% in *m88ksim* and more than 55% in *vortex*, while in the other benchmarks it is approximately 20-25%. This means that when increasing the instruction fetch rate both *m88ksim* and *vortex* would reveal bigger fraction of data dependencies which may potentially benefit value prediction (relative to the other benchmarks). In addition, it can also be observed that only 23% (on average) of the data dependencies are both predictable and span a distance of less than 4 instructions. This means that current processors (limited in their instruction-fetch bandwidth) can effectively exploit little of the potential of value prediction, and as a result obtain a relatively less significant performance gain. Moreover, even if a data dependency is value predictable and spans a distance of less than 4 instructions, it does not necessarily imply that it can be exploited by current processors due to several reasons: First, even if the DID between the producer and the consumer of a value is 1, 2 or 3, there is still a chance of 50% that the producer and the consumer can be fetched consecutively from memory, issued sequentially, and eventually also executed in a sequential manner. Second, even if the dependency between the producer and the consumer is both value-predictable and short (in terms of DID), the other operands of the consumer may not necessarily be ready because they are data dependent on values that had neither been computed yet, nor correctly value-predicted. We have found that the frequency of the latest case among the short (less than 4 instructions) and value predictable data dependencies is less than 10% (on average). Therefore, our main conclusion from this part of this study is that the major part of the correct value predictions can only be exploited when the fetch rate is increased.

4. The limitations of value-prediction hardware in high-bandwidth instruction-fetch processors and a possible hardware solution.

In the previous section we have shown that it is very significant to use value prediction in microprocessors which employ high-bandwidth instruction-fetch mechanisms. In this section we discuss the limitations of using the conventional value predictors in such processors, and offer a possible hardware solution that overcomes these limitations. Our solution is proposed for microprocessors which employ a trace-cache, however, the principles of our scheme can be also adopted when using other high-bandwidth instruction-fetch schemes.

4. 1.Limitations of the conventional hardware.

Using conventional value prediction schemes, such as those presented in [7] and [14], in high-bandwidth instruction-fetch processors introduces several fundamental problems. In present processors the prediction table is searched in the fetch stage ([14]) and in case of hit the predicted values are assigned to the fetched instructions and progress with them through the pipeline stages. Note that since current superscalar processors fetch up to 4 instructions per cycle, the prediction table could be organized in an interleaved manner in such a way that it could satisfy all the fetched instructions. In addition, as long as the processor is not capable of fetching instructions beyond a single taken branch (every cycle), it is guaranteed that all the fetched instructions are consecutive and hence more than one copy of a given instruction can never be fetched at the same cycle. However, when the processor can fetch instructions beyond a single taken branch per cycle, it is possible that more than one copy of an instruction is fetched at the same cycle. For instance, in the case of a loop, the processor can fetch more than one iteration of the loop at the same cycle, and as a result it cannot be guaranteed that all the fetched instructions are consecutive. Hence, using an interleaved prediction table for such cases can be useless, since several copies of a given instruction may need to access the prediction table simultaneously. This situation introduces two main problems. First, even if the table is interleaved, all the copies of the same instruction need to access the same bank port simultaneously (figure 4.1). In addition, even if we ignore the first problem, we still need to determine how to produce the predicted values in such a case. If we use last-value prediction, we can supply the same value to all the copies of the instruction simultaneously. However, if we use stride value prediction, we need to compute these values in advance. For instance, suppose that we fetch in each cycle 3 iterations of a loop, and one of the instructions in the loop computes an index of arrays. In each iteration the value of this instruction always increases in a fixed delta relative to

the value in the previous iteration. If 3 iterations of the loop are fetched simultaneously, as illustrated by figure 4.2, the predictor needs to produce the predicted values for the 3 copies of this instruction. The current stride predictor is capable of producing a predicted value for only one copy of an instruction every cycle. In order to overcome these limitations we need to redesign the conventional predictors.

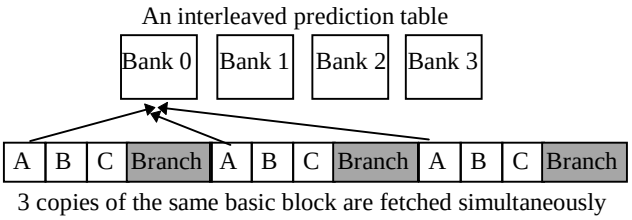

Figure 4.1 - A Possible table's ports conflict.

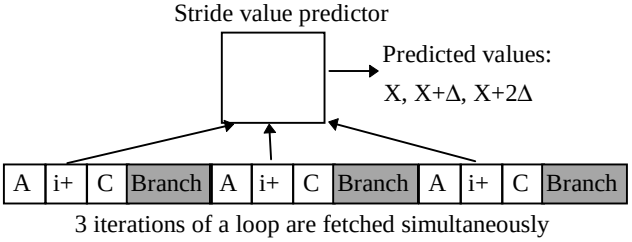

Figure 4.2 - Multiple copies of a given instruction require a production of multiple predicted values.

4. 2.A hardware solution

Our hardware solution, illustrated by figure 4.3, is currently proposed for microprocessors which employ a trace cache. The principle of this solution is based on a fast distribution network. The network distributes the addresses of instructions in the trace to a highly-interleaved prediction table. Subsequently, the predicted values are collected from the prediction table banks and are distributed to the instructions in the trace, in such a way that each instruction is assigned to its predicted output value.

As the trace-cache is accessed, the addresses of the instructions in the trace are stored in a special buffer, termed the *trace addresses buffer*. These addresses are read by the *address router*. The job of the address router is to distribute the addresses of the instructions in the trace to the interleaved prediction table. In the current scheme, we assume that the bank number is determined according to the low order bits of the address (forming a modulo operation). In addition, the trace router needs to resolve bank conflicts, when they occur. Bank conflicts may occur in two cases: 1. When different instructions attempt to access the same bank simultaneously; and 2. when more than one copy of the same instruction appears in the trace (e.g. - in the case of a loop). In the first case the conflict can be resolved by assigning priorities to the instructions in the trace. For instance, if two different instructions conflict the same bank port, grant only the earlier instruction. In the second case, when several copies of the same instruction attempt to access the bank port, the router's job is to merge these accesses to a single access. Once the router resolves the bank conflicts the requests are sent to the prediction table banks. When we use last-value prediction, the predictor returns the last-value produced by the individual instruction. In the case of stride value prediction the predictor returns two values: 1. The last-value and 2. the stride value itself.

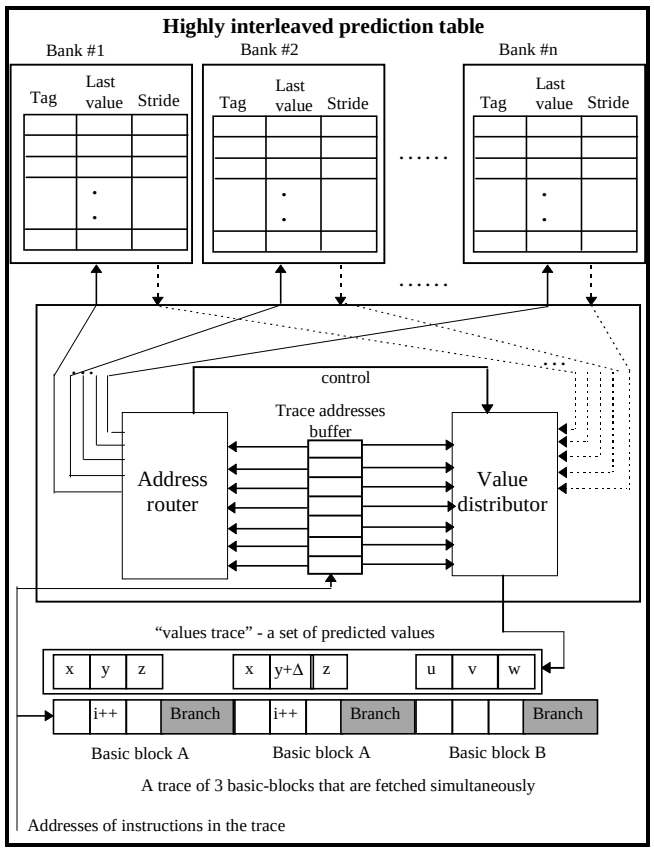

Figure 4.3 - The proposed hardware solution.

The predicted values that were produced by the interleaved table are collected by the *value distributor*. The job of the value distributor is to distribute the returned values (from the predictor) and to assign them to the corresponding instructions in the trace. Note that the distributor is informed about the addresses of the instructions in the trace when they are stored in the trace addresses buffer. In addition the mapping between these addresses and the prediction table banks are sent to the distributor through special control lines from the router. Through these control lines the router also informs the distributor about the conflicts (if any occurred). For instance, it informs the distributor which instructions were granted access to the table in a conflict and which were forbidden. In addition it also informs the distributor when several prediction requests were merged to a single request.

All this information is essential for the distributor in order to perform the re-mapping from the bank number to the instruction address in the trace. In addition when an instruction was not granted access to the table (because of a conflict) the distributor is responsible for informing this instruction that its predicted value will not be provided (e.g. - we can use a sort of "valid-bit" assigned to the predicted value of every instruction).

When value-prediction requests are merged to an individual request, it is necessary to distribute the returned value from the predictor among the merged requests. When we use last-value prediction, the distribution in this case is simple since the same value is distributed to all the copies of the instruction. However, when using stride value prediction, to each of the distributed values should be added a fixed delta (stride) relative to the previous value. All the values in this sequence need to be computed by the value distributor. Note that we have previously indicated that when using stride value prediction the value predictor returns both the last-value and the stride value. Hence, both these values are used by the value distributor in order to compute the other values in the sequence. For instance, it can be observed in figure 4.2 that 3 basic blocks are fetched from the trace cache: A, A and B. In the two copies of basic block A we observe an instruction, marked as "i++", that always adds a fixed delta value relative to its previous last value. Once the value predictor sends the last-value and the stride, the value-distributor computes the sequence of values y and $y+\Delta$.

The evaluation of the hardware complexity of the proposed scheme is beyond the scope of this paper. In addition, although the proposed scheme may increase the latency of the prediction process, it is not expected to degrade the effectiveness of value prediction. The value prediction table access starts at the earliest pipeline stage, the fetch stage, since the addresses of the fetched instructions (in the trace) are known by then. The earliest pipeline stage where the predicted values may be needed is the execution stage, since dependent instructions may execute speculatively based on these values. From the moment the prediction table is accessed (fetch stage) until the earliest moment where those predicted values may be needed there are at least 2 clock cycles (at least 1 clock cycle in the fetch stage and at least 1 clock cycle in the decode / issue stage). This implies that we can also pipeline the accesses to our prediction table, such that in the first stage (fetch) the address router distributes the instructions' addresses and in the second stage the value-distributor distributes the predicted values to the instruction in the trace. In this way we can tolerate the latency of our scheme and still gain the required throughput.

In figure 4.3 we illustrated how our hardware scheme can be integrated with the stride predictor. Note that instead of using either last-value predictor or stride predictor we can also use a *hybrid predictor* such as was proposed in [9]. A hybrid value predictor consists of two prediction tables: last-value prediction table and relatively small stride prediction table (since the number of instructions which exhibit stride patterns is relatively small). In addition the hybrid predictor can be assisted by *opcode hints*, inserted by the compiler, in order to classify instructions to each of the prediction tables according to their value predictability patterns (a broad study on these issues was presented in [9]). The usage of such a hybrid predictor in the proposed hardware scheme introduces several significant advantages. First, when merging value prediction requests, the value distributor needs to distribute the predicted value among the merged requests. As long as the predicted value is returned from the last-value prediction table, the value distributor does not need to perform any computation, but only to distribute the same value to all the requests that were merged. Only when the predicted value is returned from the stride prediction table, does the value distributor need to compute the distributed values among the merged requests. In this way we can significantly reduce the number of computations made by the value distributor. Second, the address router can be assisted by the opcode hints when distributing the addresses to the prediction tables. The opcode hints indicate which instructions are allowed to be predicted and which are not. This can significantly reduce the number of conflicts that need to be resolved by the router, since not all the instructions in the program considered only as candidates for value prediction. In addition, the opcode hints can assist the router by indicating to which prediction table (stride or last-value) the request should be distributed.

5. Performance analysis

In this section, we examine the performance gain of our value-prediction hardware scheme in high-bandwidth instruction-fetch microprocessors. The machine's execution model which was simulated in our experiments considers a finite instruction window of 40 instructions and 40 execution units. The decode / issue width of our machine is also limited to up to 40 instructions. Our machine allows register renaming, branch prediction and value prediction. A branch misprediction penalty is 3 clock cycles and value misprediction penalty is 1 clock cycle ([14], [9]). In case of value misprediction, the machine invalidates only the dependent instructions and reschedules them (when available) for execution. In addition, for the prediction of values we used the stride predictor (using a 2-bit saturated counter classification mechanism).

Our first set of experiments assumes a perfect branch predictor, and an instruction-fetch mechanism that can fetch up to n taken branches each cycle but no more than the decode / issue width (40 instructions). The range of values chosen for n is: 1, 2, 3, 4 and 40. Figure 5.1 illustrates the speedup of our machine when using value prediction

relative to the *same machine* (with the same instruction-fetch mechanism) but without value prediction.

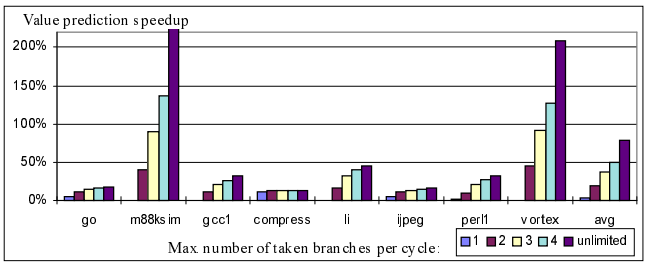

Figure 5.1 - Value prediction speedup when using an ideal BTB.

It can be clearly observed that when we increase the number of taken-branches allowed to be fetched per cycle (i.e., increasing the effective instruction-fetch rate), the speedup gained by value prediction grows significantly. For instance, when we allow fetching up to 1 taken branch each cycle the average speedup is barely noticeable (approximately 3%), but when increasing the instruction-fetch rate and allowing up to 4 taken branches per cycle the averages speedup becomes nearly 50%.

Our second set of experiments is similar to the first set, but this time we use a 2-level BTB in a PAp configuration ([27]) instead of the perfect branch predictor. The first level size of the BTB is 2K entries organized as a 2-way set associative table. Each branch has a 4-bit history register. For simplicity, we assume that our BTB allows predictions of multiple branches at the same cycle ([18]) The average prediction accuracy that this BTB gained in our experiments is 86%. Figure 5.2 summarizes the results for this set of experiments. Similarly to the previous set of experiments, we observe the significant impact of the instruction-fetch rate on the speedup gained by value prediction. When the fetch rate is limited up to 1 taken branch each cycle, the average speedup is approximately 3%, but when increasing the instruction-fetch rate and allowing up to 4 taken branches per cycle the average speedup becomes approximately 20%.

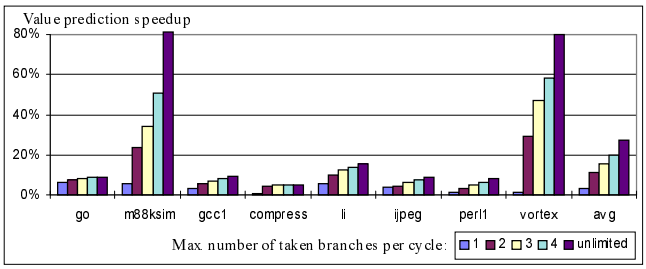

Figure 5.2 - Value prediction speedup when using a 2-level BTB.

It can be observed by comparing figures 5.2 and 5.1 that the branch prediction accuracy has also a significant effect of speedup gained by value prediction. For instance, when our machine allows fetching up to 4 taken branches each cycle the speedup drops by approximately 30% when using a realistic BTB relative to an ideal branch predictor. This observation indicates that any small improvement in the BTB accuracy can considerably affect the performance gain of value prediction.

The next two experiment sets are summarized in figure 5.3. In these two sets we examine the speedup gained by value prediction when using a trace cache with a similar organization to the one used in [18]. The trace cache configuration consists of 64 entries organized as a direct-mapped cache. Each entry can store up to 32 instructions or up to 6 basic blocks. In the first set of experiments we used the trace cache with a perfect branch predictor and in the second set of experiments we used the 2-level BTB with the same configuration as before. In both sets of experiments the instruction window size is 40. Our experiments indicate that when using a trace cache, value prediction itself can increase the performance by more then 10% (on average). In addition, we also observe that the performance gain that value prediction can obtain is highly dependent on the prediction accuracy of the BTB. The upper bound of the BTB effect is obtained when using an ideal branch predictor. In this case the speedup gained by using value prediction is less than 40% (on average).

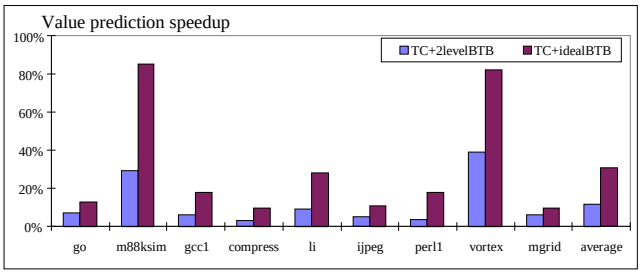

Figure 5.3 - Value prediction speedup when using a trace cache.

Note that the reported results in figure 5.3 can be significantly improved by tuning the performance of the BTB and the trace cache, i.e., using a more accurate BTB and also improving the performance of the trace cache.

6. Conclusions

In this paper we explored the limitations of using value prediction in present processors. In addition, unlike previous studies in this area, we also searched for the environment that would allow next-generation processors to gain more benefits from value prediction. We showed that the effectiveness of value prediction is significantly influenced by the instruction-fetch bandwidth and the instruction issue rate.

Initially, we explored the inherent properties of true-data dependencies in computer programs which limit the effectiveness of value prediction in current microprocessor

architectures. Our observations indicate that many data dependencies span across instructions that are fetched to the processor consecutively. As a result these dependent instructions are executed in a sequential manner, not because of the dependencies between them, but because they are fetched sequentially. In such a case although the value predictor yields a correct value prediction, the prediction becomes useless since the input value becomes ready. In addition, we examined the usage of value prediction in high-bandwidth instruction-fetch processor and presented the limitations of using the conventional value predictors. We also offered a possible high-level hardware solution that overcomes these limitations and examined its performance gain.

The innovation in this paper is very important since it reveals that value prediction has tremendous potential that can be best exploited when using high bandwidth instruction-fetch mechanisms. In addition it encourages us to further explore this new direction in order to better exploit the potential of value prediction and efficiently integrate it with the next generation of microprocessors.

References

[1] T. Conte *et al*. Optimizations of Instruction Fetch Mechanisms for High Issue Rates. Proc. of the International Symposium on Computer Architecture, June, 1995.

[2] S. Davidson, D. Landskov, B. D. Shriver and P. W. Mallet. Some Experiments in Local Microcode Compaction for Horizontal Machines. IEEE Transactions on Computers, Vol. C-30, no. 7, July, 1981, pp. 460-477.

[3] S. Dutta and M. Franklin. Control Flow Prediction with Tree-Like Subgraphs for Superscalar Processors. Proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitecture, November, 1995.

[4] J. R. Ellis. Bulldog: A Compiler for VLIW Architecture. MIT Press, Cambridge, Mass., 1986.

[5] J. A. Fisher. The Optimization of Horizontal Microcode Within and Beyond Basic Blocks: An Application of Processor Scheduling with Resources. Ph.D. dissertation, TR COO-3077-161. Courant Mathematics and Computing Laboratory, New York University, NY, October, 1979.

[6] D. H. Friendly, S. J. Patel and Y. N. Patt. Alternative Fetch and Issue Policies for the Trace Cache Fetch Mechanism. Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, December, 1997.

[7] F. Gabbay and A. Mendelson. Speculative Execution based on Value Prediction. EE Department TR #1080, Technion - Israel Institute of Technology, November, 1996 (http://www-ee.technion.ac.il/~fredg).

[8] F. Gabbay and A. Mendelson. An Experimental and Analytical Study of Speculative Execution based on Value Prediction. EE Department TR #1124, Technion - Israel Institute of Technology, June, 1997.

[9] F. Gabbay and A. Mendelson. Can Program Profiling Support Value Prediction? Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, December, 1997.

[10] J. Gonzales and A. Gonzales. Speculative Execution via Address Prediction and Data Prefetching. Proc. of the 11th International Conference on Supercomputing , July 1997.

[11] M. Johnson. Superscalar Microprocessor Design. Prentice Hall, Englewood Cliffs, 1990, N.J.

[12] M. S. Lam. Software Pipelining: An Effective Scheduling Technique for VLIW Processors. Proc. of the SIGPLAN'88 Conference on Programming Languages Design and Implementation, ACM. June, 1988, pp. 318-328.

[13] M. H. Lipasti, C. B. Wilkerson and J. P. Shen. Value Locality and Load Value Prediction. Proc. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct., 1996.

[14] M. H. Lipasti and J. P. Shen. Exceeding the Dataflow Limit via Value Prediction. Proc. of the 29th Annual ACM/IEEE International Symposium on Microarchitecture, Dec., 1996.

[15] S. W. Melvin, M. C. Shebanow and Y. N. Patt. Hardware Support for Large Atomic Units in Dynamically Scheduled Machines. Proc. of the 21st Annual ACM/IEEE International Symposium on Microarchitecture, 1988, pp. 60-63.

[16] S. J. Patel, D. H. Friendly and Y. N. Patt. Critical Issues Regarding the Trace Cache Fetch Mechanism. Technical Report CSE—TR-335-97, Univ. of Michigan, May, 1997.

[17] A. Peleg and U. Weiser. Dynamic Flow Instruction Cache Memory Organized Around Trace Segments Independent of Virtual Address Line. U. S. Paten Number 5,381,533, 1994.

[18] E. Rotenberg, S. Bennett and J. Smith. Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching. In proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture, December 1996.

[19] E. Rotenberg, Q. Jacobson, Y. Sazeides and J. Smith. Trace Processors. In proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, Dec., 1997.

[20] Introduction to Shade, Sun Microsystems Laboratories, Inc. TR 415-960-1300, Revision A of 1/Apr/92.

[21] Y. Sazeides, S. Vassiliadis and J. E. Smith. The Performance Potential of Data Dependence Speculation & Collapsing. in Proceeding of the 29th Annual ACM/IEEE International Symposium on Microarchitecture, pp. 238-247, Dec., 1996.

[22] Y. Sazeides and J. E. Smith. The Predictability of Data Values. Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, Dec., 1997.

[23] A. Smith and J. Lee. Branch Prediction Strategies and Branch-Target Buffer Design. Computer 17:1, January, 1984.

[24] J. E. Smith. A Study of Branch Prediction Techniques. Proc. of the 8th International Symposium on Computer Architecture, June, 1981.

[25] D. W. Wall. Limits of Instruction-Level Parallelism. Proc. of the 4th Conference on Architectural Support for Programming Languages and Operating Systems. Apr. 1991.

[26] S. Weiss and J. E. Smith. A Study of Scalar Compilation Techniques for Pipelined Supercomputers. Proc. of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems. Oct., 1987.

[27] T. Y. Yeh and Y. N. Patt. Alternative Implementations of Two-Level Adaptive Branch Prediction. Proc. of the 19th International Symposium on Computer Architecture. May, 1992. pp. 124-134.

[28] T. Y. Yeh, D. Marr and Y. N. Patt. Increasing Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache. Proc. of the 7th ACM International Conference Supercomputing. July, 1993.