Technical Report CMµART-1998-04

Efficacy and Performance Impact of Value Prediction

Bohuslav Rychlik, John Faistl, Bryon Krug, John P. Shen

Electrical and Computer Engineering
Carnegie Mellon University
{bohuslav, shen}@ece.cmu.edu

Value prediction is a technique that bypasses inter-instruction data dependences by speculating on the outcomes of producer instructions, thereby allowing dependent consumer instructions to execute in parallel. This work makes several contributions in value prediction research. A hybrid value predictor that achieves an overall prediction rate of up to 83% is presented. The design of a value-predicting wide superscalar machine with its speculative execution core is described. This design is able to achieve 8.6% to 20% IPC improvements on the SPEC benchmarks. Additionally, it is shown that prediction rate is not a good indicator of speedup because over 40% of predictions made may not be useful in enhancing performance, and a simple hardware mechanism that eliminates many of these useless predictions is introduced. Finally, the interactions between value prediction and issue policy as well as pipeline depth are studied. It is shown that a machine using a new relaxed in-order issue policy in conjunction with a deeper pipeline can achieve 16% to 36% speedup with value prediction.

1. Introduction

In search for higher levels of instruction level parallelism (ILP), speculative execution techniques are increasingly being adopted. One well established technique is branch prediction, which involves predicting the directions and targets of branch instructions [7][19]. Branch prediction allows execution to temporarily ignore dependency edges in the control flow graph. Value prediction [10][9] is a similar technique that involves predicting data values produced by instructions. It allows the processor to temporarily ignore dependence edges in the data flow graph when scheduling instructions for execution.

1.1 Previous Work

Value prediction is possible because of value locality [10] - the likelihood that values repeat in the same storage locations. [10] proposed a Load Value Prediction Unit that buffers and predicts the values produced by each load instruction. The study reports minimal speedup obtained from value prediction on two realistic PowerPC machine models and the Alpha 21164 machine model. A follow-up study [9] extends load value prediction to all register writing instructions and evaluates potential speedup on a machine limited only by instruction window size.

Stride value prediction [12] allows accurate prediction of values that have not previously been seen. A stride predictor stores the last value and a stride value, which are added together to generate a prediction. [11] studied the potential of classifying instructions as last value- or stride-predictable by profiling and reported ILP increases on an abstract machine.

In a theoretical analysis of value predictability [16], the finite context method (FCM) value predictor is introduced. The FCM predictor stores value history patterns for instructions and uses the history to index into a prediction table. The prediction returned is the most likely successor to the given history (context). The FCM predictor described attains very high accuracy with unlimited prediction table size.

A study of hybrid value prediction is presented in [18], in which a stride predictor and a pattern based predictor are evaluated. The pattern based predictor uses a two-level scheme to pick a prediction from a history of values previously produced by an instruction. The combination of the stride and pattern based predictors is shown to be more effective than either of them individually. In [1], a hybrid load address predictor is used to predict load addresses.

1.2 Current Objectives

This work not only introduces a hybrid predictor that achieves high prediction rate but also assesses its effectiveness in impacting the IPC performance (average instructions completed per cycle) of wide superscalar machines. Furthermore, we show that higher prediction rate does not necessarily result in higher IPC, and introduce a simple hardware mechanism to identify the subset of predictions that can lead to performance increase. We also investigate how value prediction speedup is affected by instruction issue policy and pipeline depth.

Section 2 gives background information on value prediction and introduces definitions and concepts used later. Section 3 describes the experimental framework used in this work. Section 4 discusses the hybrid value predictor used and the prediction rates obtained. Section 5 describes the baseline machine model and the microarchitecture modifications needed to facilitate value prediction. It also introduces a hardware mechanism to identify the instructions that are useful to predict. Section 6 explores the interactions of value prediction with issue policy and pipeline depth.

2. Background

2.1 Value Prediction Concepts

In order to perform value prediction, two mechanisms are needed in the microarchitecture: a *value predictor* and a *speculative execution core (SEC)*. The role of the value predictor is to provide predicted values. Typically, it is accessed with the instruction fetch address during the fetch cycle and returns the predicted value. When the instruction finishes execution, the predictor is updated with new information (e.g., the result value) in anticipation of the next prediction.

Introducing value prediction into a processor has new consequences. There are now three distinct classes of values to deal with:

1. *Predicted values* are those supplied by the value predictor; they are not directly produced by instructions in the program. Predicted values must not affect the architectural state of the machine because they can be incorrect.
2. *Speculative values* are results produced by executing instructions with at least one predicted or speculative source operand. Speculative values also must not update the architectural state.
3. *Final values* are results produced by executing instructions with no predicted or speculative source operands. Final values affect architectural state. In a machine with no speculative execution, all values are final.

The value predictor may predict either source operands or the result of an instruction [8]. Since results eventually become operands, the approaches are conceptually identical. The source operand prediction scheme can allow early dispatch at the cost of a more complex value predictor and dispatch mechanism. Because of these complications, result prediction is used in this work.

The value predictor may predict different types of values: integer, floating point, condition code, and special purpose register values. We restrict value prediction to general purpose integer values in this work. The value predictor employed is described in Section 4.

In general, the *speculative execution core* allows computations to proceed using predicted and/or speculative values. Because predicted and speculative values may be available before final values, instructions can be speculatively issued for execution earlier, producing speculative values. However, to ensure correctness, the SEC can only architecturally write back final values, introducing the need for validation. When successful, the *validation operation* transforms a speculative value into a final value. Otherwise, the speculative value is discarded and the instruction must be reexecuted.

Several variations of the SEC are possible [8], each with varying misprediction penalties and potential speedup. We use a speculative execution core which does not forward speculative results to dependent instructions, similar to the mechanism in [10][9]. Dependent instructions can only be issued speculatively using predicted values. The SEC is further described in Section 5.

2.2 Potential Performance Improvements

Value prediction can yield execution speedup in two distinct ways. First, it can decrease the execution latency of producer-consumer instruction pairs, as described by the *pipeline contraction framework* in [8]. Second, it can potentially better utilize execution resources on a finite-size machine. A traditional microprocessor can only issue instructions when their operands become available, even though there might be contention for execution resources at that time. With value prediction, the microprocessor can potentially schedule instruction execution earlier, when the execution resource is available. Since a machine with unlimited resources always has functional units available, it cannot benefit in this way. This is why [9] reported in some cases lower speedups with value prediction in the infinite machine model.

3. Experimental Framework

All the data reported in this paper are generated by an execution-driven performance simulator [2]. The simulator uses a cycle-accurate machine model based on the PowerPC 604 [3][6][17]. All key aspects of the PowerPC 604 microarchitecture are modeled.

The SPEC95 benchmark set is used for this work, including eight integer and six floating point programs. The benchmark programs, input sets, and run lengths are summarized in Table 1. To reduce simulation times, we use small input files and limit benchmark length to 100 million instructions. All user library calls are modeled, though system calls are not.

Name	Input Set	Instruction Count	Integer Register Writes
SPECint			
compress	10000 e 2231	39,719,131	22,504,999
gcc	-O regclass.i -s regclass.s	100,000,000	58,852,647
go	5 9	79,544,303	51,285,296
ijpeg	tinyrose.ppm	92,017,609	51,284,350
li	queen6.lsp	56,572,774	27,917,704
m88ksim	dhry.big.100iter, cacheoff	100,000,000	63,239,173
perl	trainscrabbl.in	50,054,404	26,594,210
vortex	tiny.in	100,000,000	50,181,852
	Total:	617,908,221	351,860,231
SPECfp			
applu	tiny.in	38,462,337	14,551,298
fpppp	tiny.in	50,112,487	18,626,903
mgrid	tiny.in	100,000,000	28,577,932
swim	tiny.in	38,797,820	11,937,334
tomcatv	tiny.in	47,187,886	26,353,196
turb3d	tiny.in	100,000,000	57,404,056
	Total:	373,560,530	157,450,719

Table 1: Benchmark Set

4. The Hybrid Value Predictor

Previous work in value prediction has classified predictable value sequences as either constants, strides, or repeating non-strides, and discussed the types of predictors best able to predict these sequences [16][11]. Hybrid value predictors have been proposed as a means of achieving high prediction rates by partitioning the instruction stream according to the type of value sequence generated by each instruction [18][1][15]. In the following subsections, the predictors composing the hybrid predictor employed are described, followed by a discussion of the classification mechanism and the specific predictor configurations used later in the paper.

4.1 Component Predictors

4.1.1 Stride+ Predictor

Stride predictors [12] predict value sequences which vary by a constant delta or stride. The Stride+ predictor is based on the two-delta predictor [16][4], with modifications to make it smaller and more accurate [15]. The Stride+ predictor is indexed with the instruction address. Each entry contains a partial instruction address tag, the last value produced by the instruction, the two delta (stride) fields, and saturating counters to track confidence and replacement.

The Stride+ predictor uses only eight bits for each of the two stride fields for size efficiency. Eight bits are sufficient to represent over 98% of the strides for correct predictions on SPECint benchmarks [15]. The Stride+ predictor implemented is set-associative, with a Modified Least Confident replacement algorithm that uses both correctness and replacement information. The replacement counter is incremented on a correct prediction, and decremented when a prediction is incorrect *or* when one of the other entries in the same set is replaced. In addition to evicting less predictable instructions, the scheme also replaces instructions with high predictability if they are not used repeatedly.

4.1.2 FCM Predictor

To predict arbitrary repeating patterns, [16] proposed a two-level finite context method (FCM) predictor. While FCM predictors can achieve high prediction rates by learning repeating sequences, large prediction tables may be required.

The FCM predictors implemented in this work are composed of two levels of tables. The first level is instruction address indexed. Each entry contains a partial instruction address tag and a partial set of bits from the last three values produced by the instruction (the 32, 24, and 16 least significant bits from each). Storing only partial history values reduces the size of the first-level table. The second-level table is indexed by hashing the history value bits and the instruction address. Second-level table entries contain a tag, the prediction value, and confidence count.

4.2 Classification Unit

The classification operation in a hybrid predictor determines which component predictors are accessed and updated and whether a prediction is made. Classification can be static [11] or dynamic [18][1][15].

The dynamic classification algorithm implemented for this work uses the prediction confidence mechanisms of the Stride+ and FCM predictors. Both predictors are accessed and two confidence bits are returned with each prediction. A high degree of prediction accuracy can then be obtained by selecting the more confident predictor. In the case of a tie, the FCM is selected. The update policy is simpler: both predictors are updated.

4.3 Two Select Designs

Parameter	**VP1**	**VP2**
Stride+ Sets	512	4096
Stride+ Associativity	6	4
Stride+ Replacement Policy	Modified least confident (MLC)	Modified least confident (MLC)
Stride+ Replacement Counter	4 bits	4 bits
Stride+ Bits / Line	79	75
FCM 1st level Sets	256	4096
FCM 1st level Associativity	4	4
FCM 1st level Bits / Line	96	92
FCM 2nd level Sets	512	16384
FCM 2nd level Associativity	4	4
FCM 2nd level Bits / Line	55	49
FCM Replacement Policy	Least recently predicted	Least recently predicted
FCM Hash Function	Shift XOR of history and address	Shift XOR of history and address
Total size	**56 Kbytes**	**678 Kbytes**

Table 2: VP1 and VP2 Hybrid Predictor Configurations

The design space for the component predictors was extensively explored and narrowed to two designs. VP1 is a small, 56 KB predictor. VP2 is over 12 times larger. The configurations for the two predictors are described in Table 2.

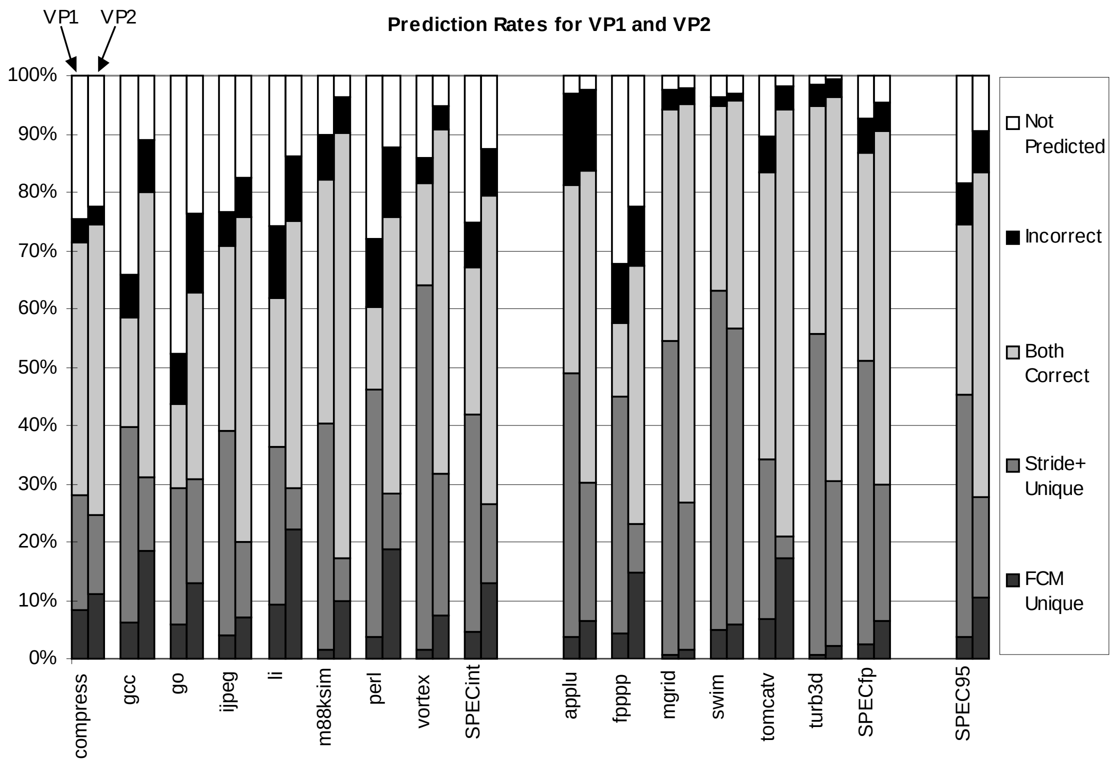


Figure 1: Prediction Rates for the VP1 and VP2 Hybrid Predictors

Figure 1 shows performance results for VP1 and VP2 in terms of prediction rates. *Prediction rate* is defined in this paper to mean the percentage of correct predictions over all possible predictions. *Prediction accuracy* is the percentage of correct predictions over actually attempted predictions. Both VP1 and VP2 attain very high prediction rates of 74% and 83%, respectively. VP1 achieves its prediction rate at 1/12th the size of VP2. Prediction accuracies for VP1 and VP2 are 91.5% and 92.2%, respectively. The prediction rate of integer values for the SPECfp benchmarks is higher (by 12% for VP1 and 20% for VP2) than for the SPECint programs.

5. The Value Predicting Machine

5.1 Baseline Machine Model

The eight-wide baseline machine model is an extension of the PowerPC 604 microarchitecture (a four-wide superscalar). The baseline machine is extended to support up to eight instructions per cycle by increasing the pipeline buffer widths and the number of functional units, rename registers, reservation station entries, and reorder buffer entries. Figure 2 is a diagram of the baseline machine.

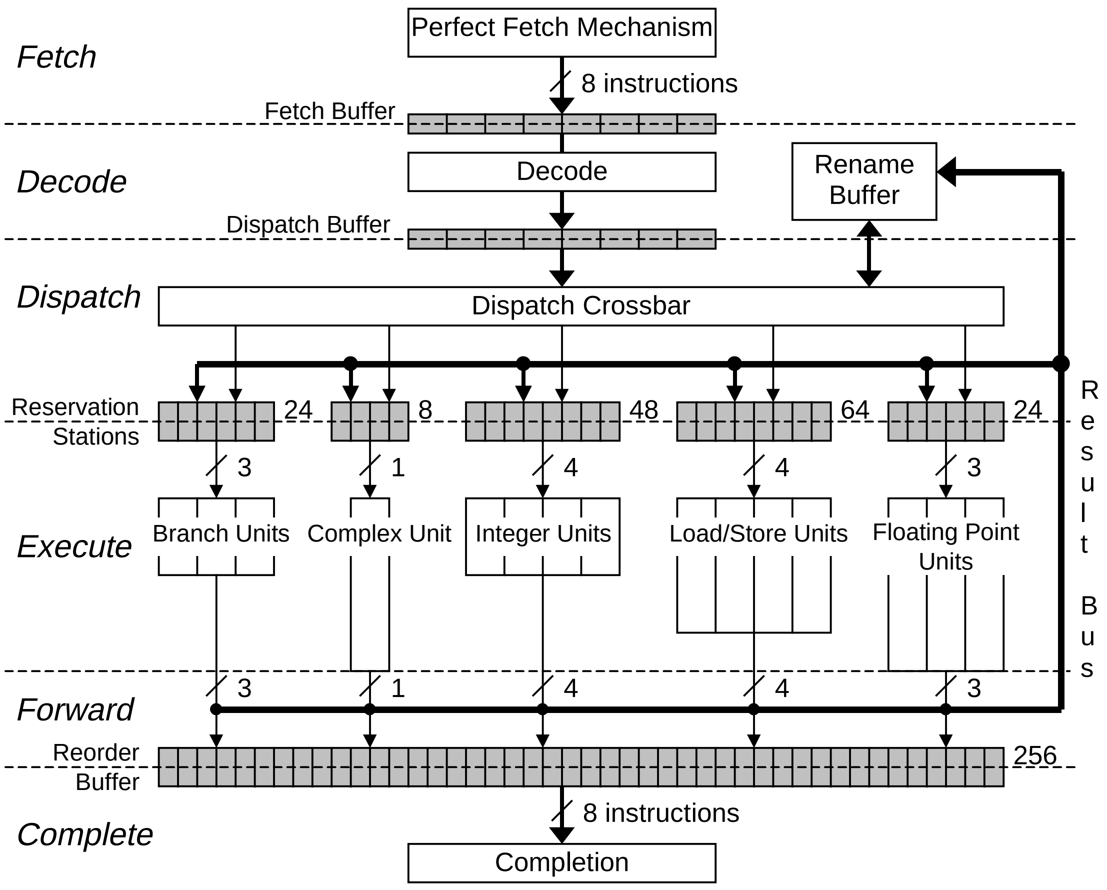


Figure 2: Baseline Machine Organization

The baseline machine is implemented with a perfect fetch mechanism. This fetch mechanism is capable of supplying eight instructions every cycle and does not suffer from instruction cache misses or branch mispredictions. Perfect fetching is implemented to keep constant pressure on the functional units and focus on the data flow bottleneck. In a real implementation, high fetch bandwidths could potentially be achieved using a trace cache [13][14], although such an implementation is subject to miss penalties.

Each reservation station services a cluster of identical functional units. Routing of instructions from the in-order front end to the functional units is split into *dispatch* to the five clustered reservation stations and *issue* to the 15 functional units. The machine has 128 integer, 64 FP, and 32 condition code rename registers with enough read ports to support eight instruction dispatch. The baseline machine also has a distributed issue window in the form of 168 total reservation station entries.

Instructions are issued to the simple integer, complex integer, and floating point units out-of-order. The load/store and branch units always execute instructions in-order. All instruction execution latencies are the same as in the PowerPC 604 microprocessor. The realistic memory hierarchy is summarized in Table 3.

Cache	Ports	Miss Latency	Sets	Associativity	Block Size	Capacity
Instruction L1	∞	0	∞	∞	-	∞
Data L1	4	6	256	4	32 bytes	32 KB
Unified L2	1	36	2048	8	32 bytes	512 KB

Table 3: Memory Hierarchy

In the PowerPC 604, instruction results are forwarded to the rename buffers and reservation stations during the last execution stage. Thus, simple integer instructions can execute and forward their results in one cycle. Due to the increased number of result forwarding paths and the increased load on each of these paths, the baseline machine has a dedicated *forward* stage during which results are broadcast on the result bus. Therefore, it takes two cycles for simple integer instructions to execute and forward their results. This more realistic design can lower the machine performance.

5.2 Value-Predicting Machine Model

The baseline machine model is modified to support result value prediction. In addition to adding the hybrid value predictor, the value-predicting machine has a modified rename buffer and a *speculative execution core* (SEC). Instructions can be speculative or non-speculative and predicted or not predicted. Speculative instructions have at least one predicted source operand; predicted instructions have a predicted result.

5.2.1 Rename Buffer

The rename buffer is used to store result predictions so they can be read by dependent instructions. The value predictor is accessed with each instruction's fetch address in the fetch stage. When an instruction allocates a rename buffer entry for a destination, the prediction, if any, is stored in the allocated entry. An additional bit is used to indicate whether the value is predicted. When a dependent instruction is dispatched, it reads the rename tag and potentially either predicted or final data.

5.2.2 Speculative Execution Core (SEC)

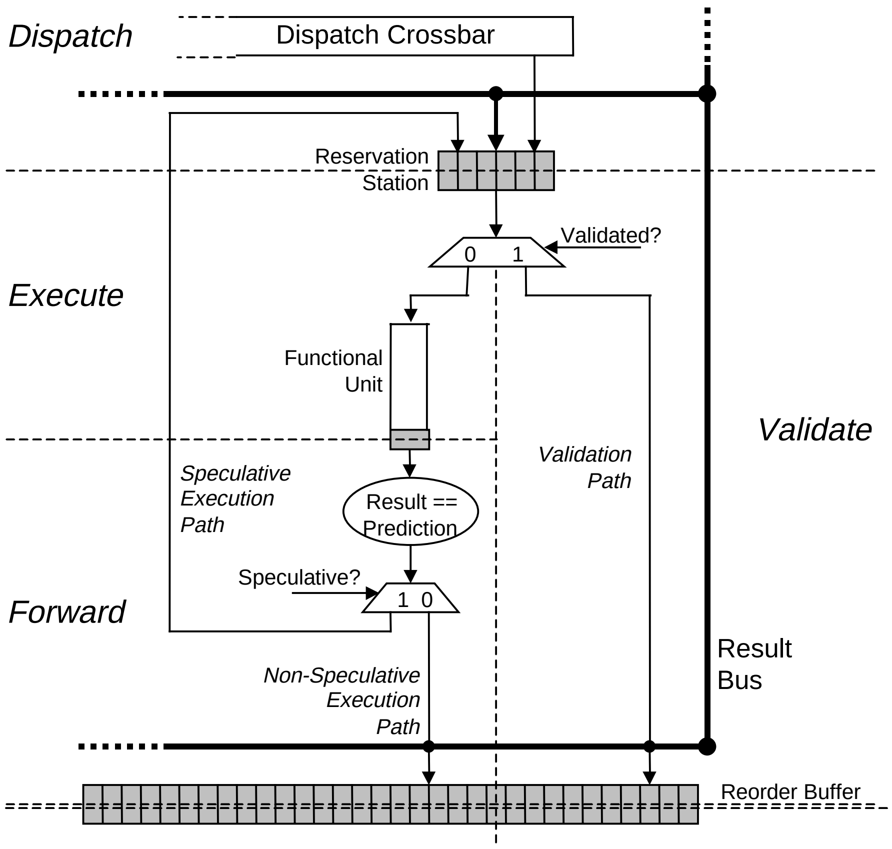


Figure 3: Speculative Execution Core (SEC) Implementation

Figure 3 illustrates the SEC implementation for a single functional unit. In addition to the *non-speculative execution* path, the SEC adds the *speculative execution path*. A comparator is added to the forward stage of both execution paths. It is used to compare an instruction's computed and predicted result, if any, generating the *prediction comparison bit*. Both the computed result and the prediction comparison bit are then forwarded either (a) along the result bus if the execution is non-speculative, or (b) locally back to the instruction's reservation station entry if the execution is speculative.

The SEC also adds the *validation path*, which is simply a forwarding path. All three issue paths are described in detail in subsections 5.2.2.1-5.2.2.3.

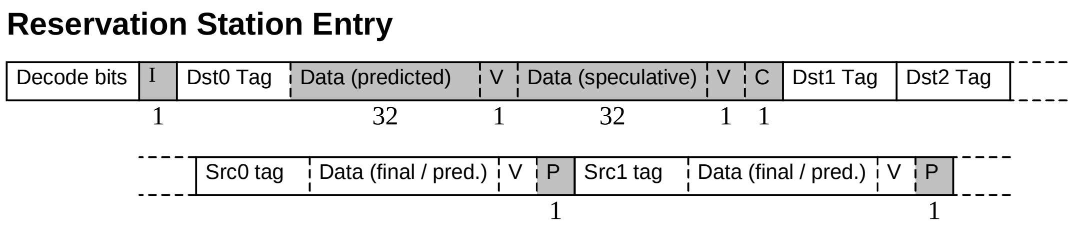


Figure 4: Value-predicting Machine Reservation Station Entry

The path to which an instruction is issued is determined by the issue logic from information in the instruction's reservation station entry. Figure 4 shows a diagram of a reservation station entry in the value-predicting machine. New fields required by the SEC are gray. Table 4 describes what each field means and when it is modified.

Field	Description	Modified
Decode bits	Various decoded bits such as opcodes.	During allocation
I	Issued: 1 if instruction has been issued with the current set of source operand values, 0 otherwise.	Set when speculatively issued Cleared when new source operand data is latched: 1. Current source data is invalid, or, 2. Current source data was incorrectly predicted
V	Valid bit for preceding data.	Data field is written
DstN: Tag	Rename buffer tag for destination N.	During allocation
Dst0: Data (predicted)	Integer result prediction made during fetch	During allocation (if result was predicted)
Dst0: Data (speculative)	Result of speculative execution.	The instruction finishes executing speculatively
Dst0: C	Comparison result: 1 if the speculative result equals the predicted result.	The instruction finishes executing speculatively
SrcN: Tag	Rename buffer tag for source N.	During allocation
SrcN: Data	Data for source operand N. predicted or final.	During allocation (final or predicted) When forwarded result is latched (final)
SrcN: P	Predicted: 1 if the data for source operand N is a predicted value, 0 otherwise.	Set during allocation if a predicted value was read from the rename buffers Cleared when forwarded result is latched.

Table 4: Reservation Station Entry Fields

The value-predicting machine only supports the prediction of one result, which must be stored in Dst0. Additionally, only complex and simple integer instructions can be speculatively issued, while results of complex, simple integer, and load/store instructions can be predicted. Loads are not issued with predicted operands as they can potentially be handled by other techniques [1][5].

As in the baseline machine, all source operands must be valid (final or predicted) for an instruction to be a candidate for issue. The following sections describe the circumstances under which an instruction is issued to each of the three paths. Note that an instruction may be issued to the validation path simultaneously with another instruction being issued to one of the execution paths.

5.2.2.1 Non-Speculative Execution Path

Instructions are issued to the *non-speculative execution path* whenever the **I** bit (see Table 4) is zero and all source operands are final. This occurs when:

1) All source operands are already final at dispatch.

2) The last invalid source operand is replaced with its final value from the result bus. All other operands are already final.

3) The last predicted source operand is replaced with its final value whose prediction comparison bit indicates misprediction. All other operands are already final.

Reservation station entries are deallocated when instructions issue non-speculatively. During the forward stage, the final result and its prediction comparison bit are broadcast on the result bus.

5.2.2.2 Speculative Execution Path

Instructions are issued to the *speculative execution path* whenever the **I** bit is zero and at least one source operand is predicted. This occurs when:

1) During dispatch, all operands are valid, but at least one source operand is a predicted value and remains a predicted value until issue.
2) The last invalid source operand is replaced with its final value. All other operands are valid, but at least one is predicted.
3) A predicted source operand is replaced with its final value whose prediction comparison bit indicates misprediction. All other operands are valid, but at least one is predicted.

A speculatively issued instruction does not deallocate its reservation station entry. After execution the speculative result and its prediction comparison bit are *not* broadcast on the result bus. Instead, they are locally forwarded back to the same reservation station entry.

5.2.2.3 Validation Path

Instructions are issued to the *validation path* whenever the **I** bit is one and all source operands are final. This only happens when:

1) The last predicted source operand of a previously speculatively executed instruction is replaced with its final value whose prediction comparison bit indicates correct prediction. All other source operands are final.

This indicates that the instruction has already executed with the final source operand values. This instruction is validated and the speculative destination value contained in the reservation station becomes final. This final value and its prediction comparison bit are broadcast from the reservation station onto the result bus. The reservation station entry is deallocated.

5.2.3 SEC Operation

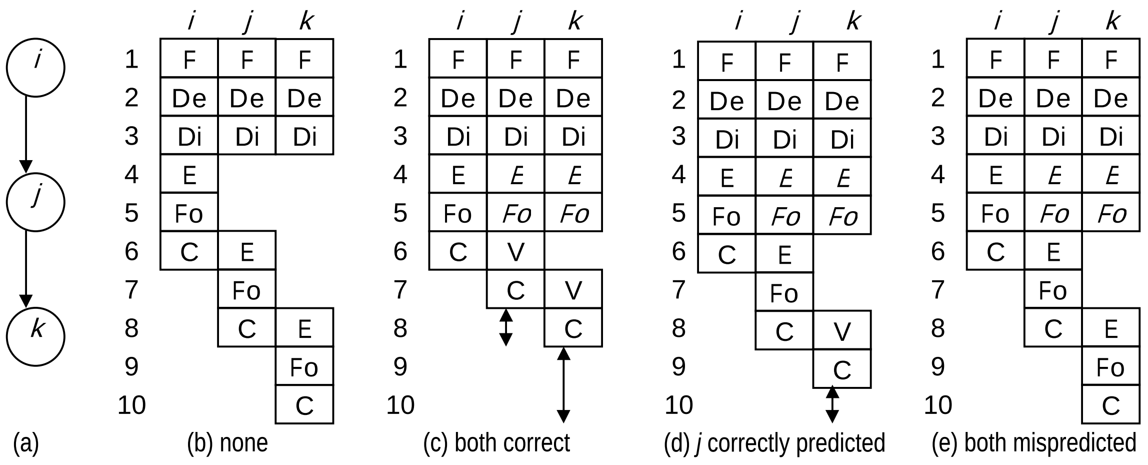


Figure 5: Speculative Execution Example

Figure 5 shows four example execution traces of three sequentially-dependent instructions. The data dependences between the instructions are shown in Figure 5(a) and this example assumes there are no other dependence or resource constraints. Figure 5(b) shows how the instructions can best execute on the baseline machine with no value prediction. The labels inside the boxes indicate which pipe stage each instruction is in each cycle: fetch, decode, dispatch, execute, forward, validate, or complete. The complete stage indicates the earliest cycle the instruction could architecturally retire. Because instructions *j* and *k* must wait for their operands to become available, the quickest that the sequence can complete without value prediction is in 10 cycles. Figures 5(c, d, and e) show three possible execution scenarios when the results of instructions *i* and *j* are predicted. In Figure 5(c), both predictions are correct and the sequence is executed in only 8 cycles. While the speculative issue (italicized) of instructions *j* and *k* can temporarily bypass the data dependences, the validation of *j* and *k* must obey the data dependences. The validation of instruction *k* is delayed to cycle 7 because its final source operands only become available (through the validation of instruction *j*) at the end of cycle 6. Also, note that correctly predicting the results of *i* and *j* allows *j* and *k* to remove 1 cycle each from the critical path.

Figures 5(d and e) show cases when one and two mispredictions occur. Note by comparing Figures 5(b and e) that there is no compulsory misprediction penalty in this machine. The only potential misprediction penalties are additional resource hazards.

5.3 Performance Results

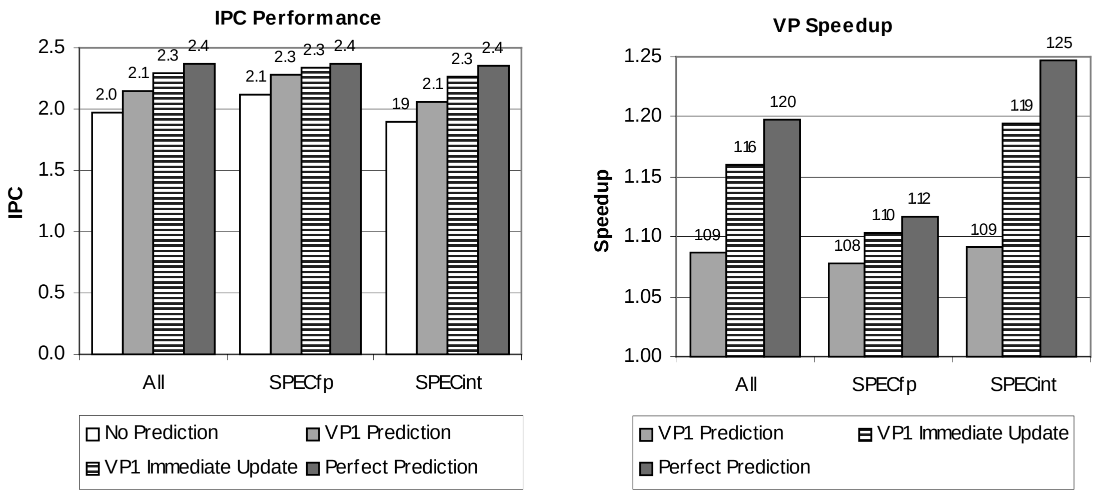


Figure 6: Impact of Value Prediction on IPC

Figure 6 shows the performance impact of adding the VP1 predictor and the SEC to the baseline machine. *VP speedup* is the IPC of each value-predicting machine configuration divided by the IPC of the baseline machine (No Prediction). Speedup of the VP2 configuration is consistently within 2% of the VP1 performance; therefore, only VP1 results are presented.

In Figure 6, performance using the VP1 predictor is shown using both realistic update and immediate update. Immediate update is an idealization that updates the predictor with the final result value after getting a prediction at fetch time; this is used to obtain the prediction rates in Section 4. Realistic update sends the final result to the predictor after non-speculative execution or validation. Prediction rates with realistic update drop from 74% to about 50%. While the VP1 configuration is able to yield an overall 9% speedup over the baseline machine, this speedup increases to 16% with immediate update. Although realistic update is used in the remainder of this paper, the above results suggest a speculative update policy approximating immediate update can yield better performance.

A perfect value predictor (always correct) is also evaluated to determine maximum obtainable speedup on the value-predicting machine. The Perfect configuration achieves near 25% improvement on SPECint benchmarks, suggesting better prediction accuracy can yield better performance.

5.4 Prediction Usefulness Tracking

As part of the model instrumentation, counters are added to track how often predicted values are read from the rename registers. On average, 31% of SPECint and 62% of SPECfp integer predicted values are never read. The cause is that the predicted value is overwritten by the final result before any dependent instruction is dispatched, making the prediction useless. To reduce wasted utilization of the predictor resources on useless predictions, a simple *prediction usefulness tracking mechanism* is introduced. The mechanism consists of an additional *data requested* bit associated with each integer rename register entry. The bit is cleared at allocation and set whenever a dependent instruction reads the register. In this implementation, only simple and complex integer instructions can set the bit because they are the only instruction types allowed to issue speculatively. When any integer-writing instruction finishes, it reads the data requested bit and only updates the value predictor if it is set.

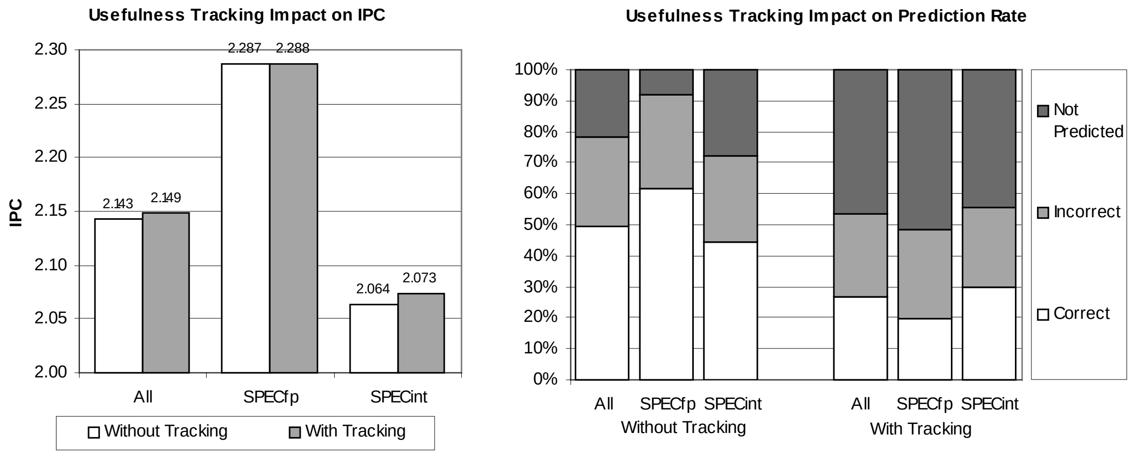


Figure 7: Impact of Prediction Usefulness Tracking With VP1 Predictor

With the usefulness tracking mechanism in place, the percentage of useless predictions drops to 6.6% for SPECint and 27.7% for SPECfp. Figure 7 shows the impact on IPC and prediction rate. When the mechanism is used, prediction rate drops from 50% to 27%, but IPC slightly increases! This implies that overall prediction rate is not a good indicator of the IPC speedup. Only the prediction rate for those instructions closely followed by their consumers matters. The simple usefulness tracking mechanism described attempts to identify this subset of instructions.

Prediction accuracy also drops from 63% to 50%, suggesting the prediction tracking mechanism should be used during predictor design to tailor it for useful predictions. All subsequent results in this paper use the prediction usefulness tracking mechanism.

6. Machine Organization and VP Speedup

This section investigates the interaction of value prediction with issue policy (out-of-order vs. in-order) and pipeline depth (shallow vs. deep).

6.1 Issue Policy

Data ready instructions can be issued from the reservation stations either in-order or out-of-order. In-order issue is a simpler implementation but can unnecessarily stall some data ready instructions. Out-of-order issue avoids these stalls but can require a much more complex implementation.

With value prediction, a new in-order issue policy is available. We term this policy *relaxed in-order issue*. It relaxes some of the constraints imposed by strict in-order issue without adding the complexity of out-of-order issue.

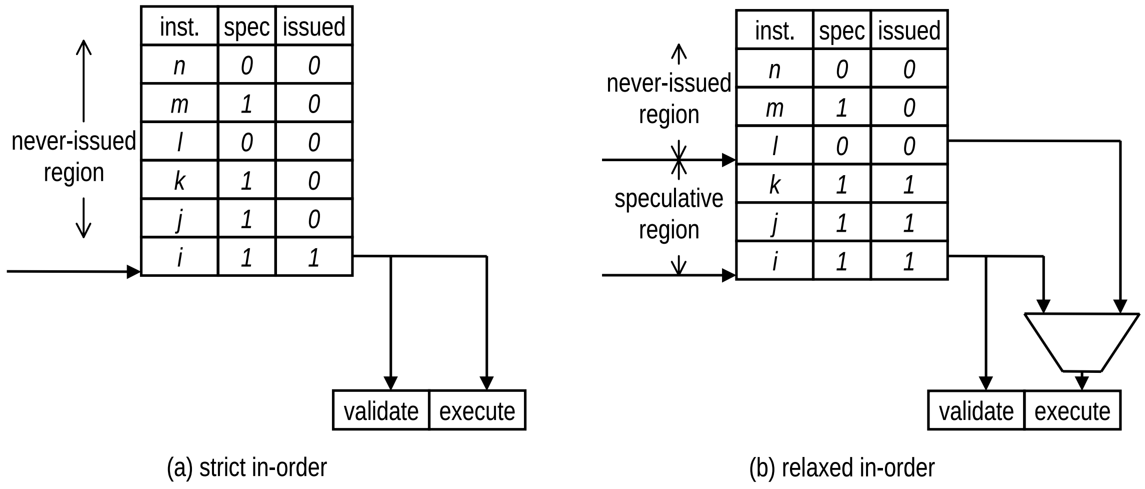


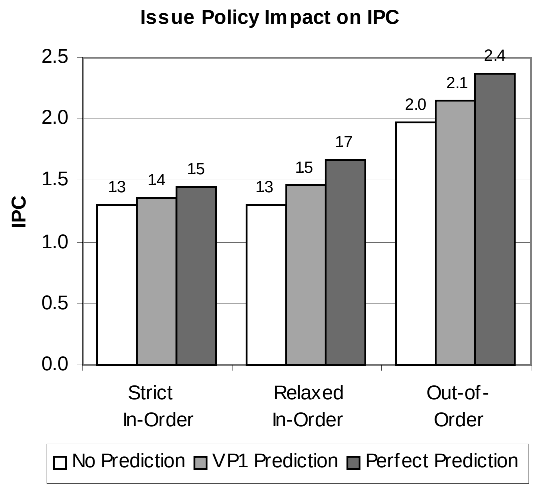
Figure 8: Strict In-order Issue and Relaxed In-order Issue

Figure 8 illustrates the difference between the strict and relaxed in-order issue policies for issue width of one using abbreviated reservation station diagrams. In each diagram the *spec* field indicates whether an instruction has any predicted operands and the *issued* field (the *I* bit described in Section 5.2.2) indicates whether the instruction has been executed.

The strict in-order policy uses a single pointer to the oldest instruction to identify the only possible next issue candidate. A speculative instruction may still be executed early, but subsequently issue stalls until the instruction is validated and removed from the reservation station. Then the pointer can move to the next oldest instruction.

In contrast, the relaxed in-order issue policy uses two pointers to identify the only two possible issue candidates: the oldest instruction and the oldest instruction that has never been executed. Initially, both point to the same entry and move in unison as long as only non-speculative instructions are issued. However, when an instruction is issued speculatively, only the second pointer advances. In the following cycle, the next instruction could be issued while the speculatively issued instruction waits for validation. When the oldest instruction is validated or reissued with final operands, it is removed from the reservation station and the first pointer advances.

Unlike for out-of-order issue, the complexity of relaxed in-order issue is not dependent on the number of reservation station entries and is similar to that of strict in-order issue. Relaxed in-order allows speculative instructions to reissue or validate out-of-order with respect to non-speculative instructions.



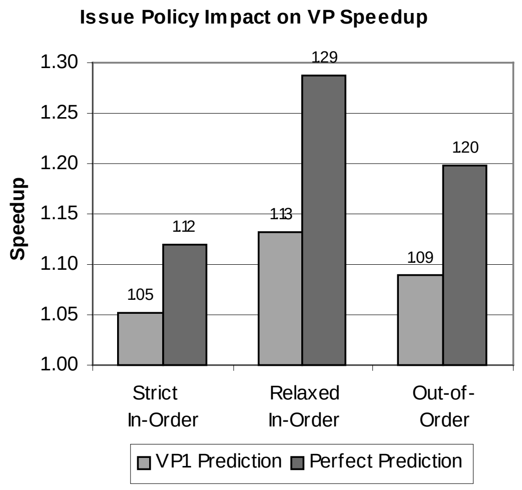


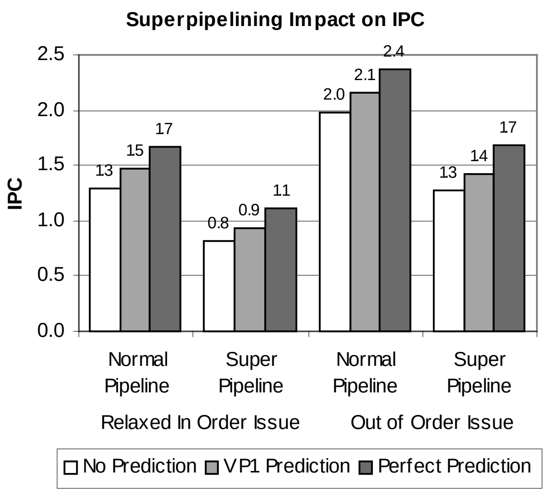
Figure 9: Impact of Issue Policy on VP Speedup

Figure 9 illustrates the effect of issue policy on the speedup obtained with value prediction. The issue policy is varied only for the simple integer, complex integer, and FP pipelines; branch and load/store pipelines remain strictly in-order. While out-of-order issue attains the highest IPC (up to 2.4), relaxed in-order issue benefits the most (up to 29%) from value prediction.

6.2 Pipeline Depth

As described in Section 5.2.3, the SEC can use correct value predictions to achieve a one-cycle reduction in execution latency of a producer-consumer pair with two-cycle execution latency each. Specifically, the potential latency reduction of a producer-consumer pair execution is equal to the execution latency of the consumer minus the validation cost. Therefore, if the execution latency of the consumer increases relative to the validation latency, the potential latency reduction also increases.

This hypothesis is investigated by superpipelining the machine with the assumption that the forward and validation stages remain single cycle. All execution and memory latencies are doubled. Thus, a simple integer instruction now takes two cycles to execute and one cycle to forward its result. Though significant in a real machine, no branch misprediction penalties are incurred because of the perfect instruction supply assumption.



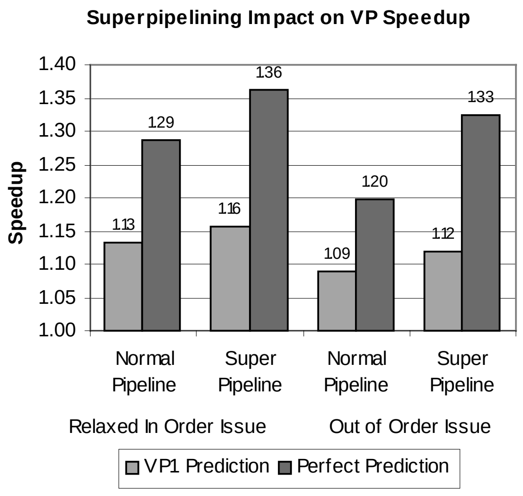


Figure 10: Superpipelining Impact on VP Speedup

Figure 10 shows that a superpipelined machine benefits more from value prediction than a normal machine. Issue policies are also varied between out-of-order and relaxed in-order. The VP1 configuration yields a speedup of 12% on the superpipelined value-predicting machine with out-of-order issue as compared to only 9% on the normal pipeline.

6.3 Complexity Tradeoffs

Notice in Figure 10 that the greatest VP speedup is achieved by the relaxed in-order superpipelined combination (16% for VP1 and 36% for Perfect). Since in-order issue is less complex than out-of-order issue and less likely to impact cycle time, in-order issue is a better match for a superpipelined, high clock frequency implementation. Value prediction techniques also appear to be compatible with superpipelined implementations. Access to the value predictor could potentially be pipelined over multiple stages in the front-end of the machine and the SEC adds little additional logic to the critical path.

This suggests that value prediction may be another way to attain the levels of performance provided by out-of-order issue. In Figure 11, the IPC of superpipelined implementations is adjusted for their increased clock frequency to produce a normalized performance metric proportional to execution time. This metric is IPC for the normal pipeline and 1.8 x IPC for the superpipelined implementation (assuming a 10% clock frequency decrease due to pipeline inefficiencies).

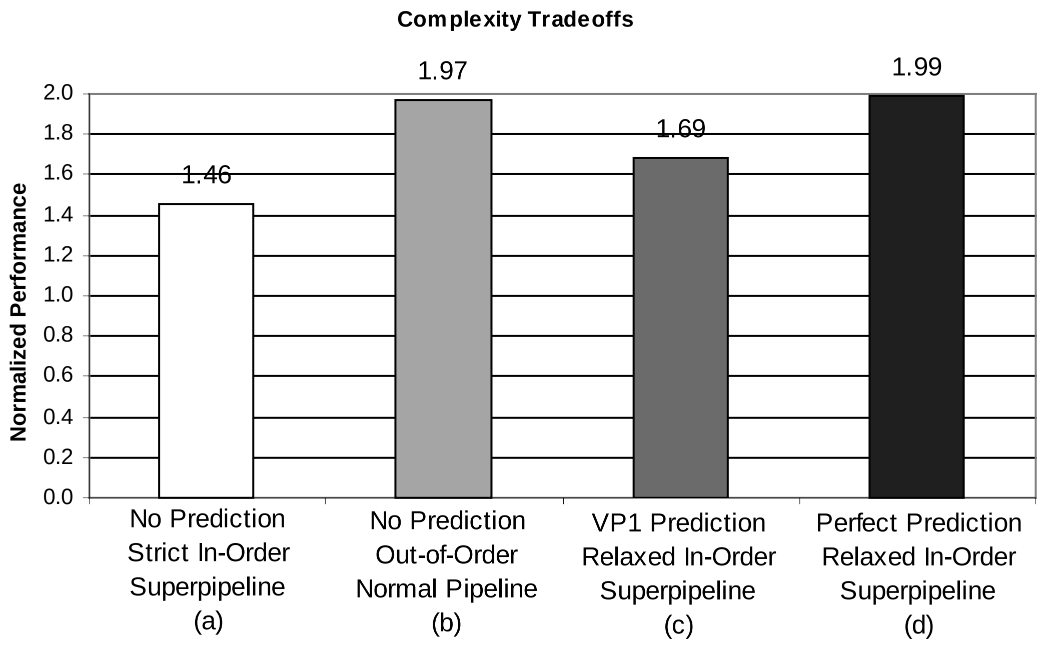


Figure 11: Complexity Tradeoffs (out-of-order vs. value prediction)

Figure 11 presents several approaches to machine design. The left two bars represent two traditional approaches. Machine (a) uses simple in-order issue and consequently reaches the lowest performance, even though it is superpipelined. Machine (b) adds complexity with out-of-order issue to improve performance but may be difficult to superpipeline. Machine (c) also adds complexity but in the form of value prediction instead of out-of-order issue. It uses the relaxed in-order issue policy, which allows it to be superpipelined. The VP1 predictor configuration does not match the performance of the out-of-order machine. However, the performance of machine (d) suggests that a value-predicting machine with relaxed in-order issue and a superpipelined implementation has the potential of outperforming the traditional out-of-order superscalar.

7. Conclusion and Future Work

We make several contributions in this paper. First, we present realistic (VP1) and large (VP2) hybrid value predictors that attain overall prediction rates of 74% and 83%, respectively, on the SPEC benchmarks. Second, we describe in detail a speculative execution core that facilitates the use of value prediction in a superscalar machine and obtain 8.6% and 20% IPC improvement using realistic and perfect value predictors, respectively. Third, we describe a simple hardware mechanism for tracking prediction usefulness and reduce the percentage of useless predictions from 41.9% to 12.5%. Finally, we present a new relaxed in-order issue policy and explore the relationship between issue policy, pipeline depth, and value prediction speedup.

This work focuses on evaluating the effectiveness of value prediction in a wide superscalar machine. The proposed implementation predicts only general purpose integer results and does not forward speculative results to allow further speculation.

Additionally, loads/store instructions are not allowed to issue speculatively. Less conservative assumptions could result in higher IPC. This potential further increase in IPC remains to be explored.

This work also suggests that overall prediction rate is not the appropriate measure of the effectiveness of value predictors. The appropriate measure of a value predictor's performance is the percentage of useful predictions that are predicted correctly. A simple mechanism for providing usefulness feedback to the predictor is proposed. However, more sophisticated mechanisms for utilizing usefulness information to increase the percentage of correct *and* useful predictions should be explored.

Finally, this work suggests that value prediction speedup is dependent on other aspects of the machine organization. Adding value prediction to existing machines may result in little performance improvement. Instead, value prediction is a potential alternative to out-of-order execution complexity. Value prediction may also be compatible with more deeply pipelined implementations. More work remains to be done to determine how value prediction can be effectively and efficiently implemented as an enabling technology.

8. References

[1] Bryan Black, Brian Mueller, Stephanie Postal, Ryan Rakvic, Noppanunt Utamaphethai, and John P. Shen, "Load Execution Latency Reduction." Proceedings of the International Conference on Supercomputing, Melbourne, July 1998.

[2] A. Cagney. PSIM User's Guide. Available as ftp://cambridge.cygnus.com/pub/psim/index.html, August 1996.

[3] K. Diefendorff, and E. Silha, "The PowerPC User Instruction Set Architecture." In IEEE Micro, pp. 30-41, 1994.

[4] R. J. Eickemeyer and S. Vassiliadis, "A Load Instruction Unit For Pipelined Processors," IBM Journal of Research and Development, vol. 37, pp. 547-564, July 1993.

[5] J. Gonzalez and A. Gonzalez, "Speculative Execution via Address Prediction and Data Prefetching." In Proceedings of the 11th ACM International Conference on Supercomputing, pp. 196-203, July 1993.

[6] IBM Microelectronics Division, *PowerPC604 RISC Microprocessor User's Manual*, 1994.

[7] J. Lee and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design." IEEE Computer, 21(7):6-22, January 1984.

[8] M. H. Lipasti, *Value Locality and Speculative Execution,* Ph.D. thesis, Carnegie Mellon University, April 1997.

[9] M.H. Lipasti and J.P. Shen, "Exceeding the Dataflow Limit via Value Prediction," Proceedings of the 29th International Symposium on Microarchitecture (MICRO-29), pp. 226-237, December 1996.

[10] M.H. Lipasti, C.B. Wilkerson, and J.P. Shen, "Value locality and load value prediction," Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS- VII), October 1996.

[11] F. Gabbay and A. Mendelson, "Can Program Profiling Support Value Prediction?" In Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30), pp. 270-280, December 1997.

[12] F. Gabbay and A. Mendelson, "Speculative Execution Based on Value Prediction." Technical Report 1080 , Technion-Israel Institute of Technology, November 1996.

[13] E. Rotenberg, S. Bennett and J. E. Smith, "Trace Cache: A Low Latency Approach to High Bandwidth Instruction Fetching," In Proceedings of the 29th International Symposium on Microarchitecture, pp. 24-34, December 1996.

[14] E. Rotenberg, Q. Jacobson, Y. Sazeides, and J. Smith, “Trace Processors,” Proceedings of the 30th International Symposium on Microarchitecture, December 1997.

[15] Reference withheld to preserve anonymity.

[16] Y. Sazeides and J.E. Smith, "The Predictability of Data Values," Proceedings of 30th International Symposium on Microarchitecture (MICRO-30), December 1997.

[17] S. Song, M. Denman, and J. Chang, The PowerPC 604 RISC Microprocessor. In IEEE Micro, pp. 8-17, 1994.

[18] K. Wang and M. Franklin, "Highly Accurate Data Value Prediction using Hybrid Predictors," Proceedings of 30th International Symposium on Microarchitecture (MICRO-30), December 1997.

[19] T-Y. Yeh, D. Marr, and Y. Patt, “Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache.” In Proceedings of the 7th ACM International Conference on Supercomputing, pp. 67-76, July 1993.