# Value Prediction for Speculative Multithreaded Architectures

Pedro Marcuello, Jordi Tubella and Antonio González

Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya,
Jordi Girona 1-3, Edifici D6, 08034 Barcelona, Spain
e-mail: {pmarcue,jordit,antonio}@ac.upc.es

## Abstract

*The speculative multithreading paradigm (speculative thread-level parallelism) is based on the concurrent execution of control-speculative threads. The efficiency of microarchitectures that adopt this paradigm strongly depends on the performance of the control and data speculation techniques. While control speculation is used to predict the most effective points where a thread can be spawned, data speculation is required to eliminate the serialization imposed by inter-thread dependences.*

*This work studies the performance of different value predictors for speculative multithreaded processors. We propose a value predictor, the increment predictor, and evaluate its performance for a particular microarchitecture that implements this execution paradigm (Clustered Speculative Multithreaded architecture).*

*The proposed trace-oriented increment predictor clearly outperforms trace-adapted versions of the last value, stride and context-based predictors, specially for small-sized history tables. A 1-KB increment predictor achieves a 73% prediction accuracy and a performance that is just 13% lower than that of a perfect value predictor.*

## 1. Introduction

Current superscalar microarchitectures suffer from some important constraints that may prevent them from exploiting large amounts of instruction-level parallelism (ILP), even with the much larger transistor budget that will be available in the near future (an increase by a factor of about 25 in the next 10 years is expected, according to the SIA technology roadmap [14]).

The size of the instruction window is one of these crucial constraints to attain high ILP. As several studies have shown (see [19] among others), to achieve a high IPC rate (instructions committed per cycle), a large instruction window is required. However, the average size of it is basically determined by the performance of the control speculation approach. One of the most severe limitations of the control speculation approach implemented in superscalar processors relies on the fact that branches are predicted in sequential order, and as soon as a single branch is mispredicted, all instructions after it are squashed.

Some alternative microarchitectures have been recently proposed to avoid such a limitation in control speculation. The basic idea is to have a two-level speculation approach: on top of the conventional control speculation of a superscalar processor, these new microarchitectures also try to predict certain points in the control flow that are very likely to be visited in the future, regardless of the outcome of the forthcoming branches. The microarchitecture spawns speculative threads starting from such control-flow points. We refer to such points in the control flow as control quasi-independent points.

Speculative threads together with the non-speculative one are executed concurrently since the processor provides support for multiple hardware contexts. Each thread has its local instruction window built in the same way as superscalar processors do. This type of execution model was proposed in the Expandable Split Window paradigm [4] and the Multiscalar microarchitecture [15]. Recent microarchitecture proposals that are also based on this generic execution model are: SPSM [3], Superthreaded [17], Trace Processors [11], Speculative Multithreaded [9][10], Dynamic Multithreaded [1], and extensions to multiprocessor architectures ([7][16] among others).

A critical issue of such architectures is the approach to identify the most effective points where speculative threads can be spawned. In addition to being highly predictable from the point of view of the control flow, the speculation mechanism should consider the data dependences among concurrent threads. For non-numeric codes with limited amount of ILP such as the SpecInt95, threads are very likely to be dependent, and thus, the microarchitecture should provide mechanisms to ensure that all dependences among threads (inter-thread dependences) are obeyed. Note that constraining the thread-level speculation just to data independent threads would result in almost no parallel threads for most SpecInt95 programs.

The performance of speculative multithreaded architectures is very dependent on the approach to dealing with inter-thread dependences. A simple approach is to force a serialization between the producer and the consumer of every dependence, for instance, through the re-execution of instructions when a dependence violation is detected. However, data value speculation can significantly boost the performance by predicting those values that flow through such dependences. If values are correctly predicted, dependent threads are executed as if they were independent.

In this paper, we study the performance of data value speculation for this type of microarchitectures. In particular, we focus on Clustered Speculative Multithreaded processors and present a comparative study of the performance of value predictors previously proposed in the literature. We also present a new predictor that is oriented to estimate the outputs of a thread based on its inputs. We refer to it as *increment*

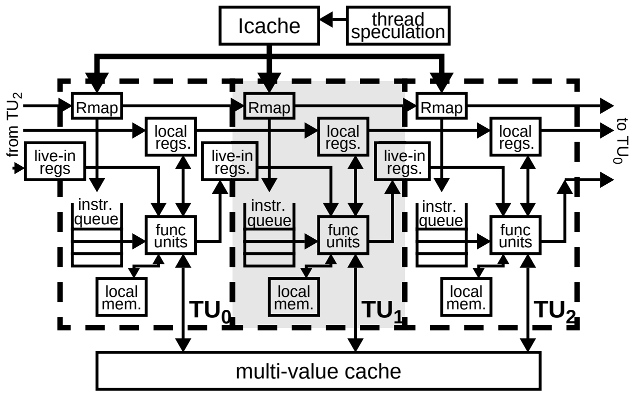


**Figure 1:** A Clustered Speculative Multithreaded processor with three thread units.

predictor. We show that an increment predictor provides the best performance among single predictors. It can achieve a 73% accuracy with a table of just 1 KB for the SpecInt95. In addition, its performance can be increased when combined with a context-based value predictor to form a hybrid predictor. In this case, an 16-KB predictor has a 80% prediction accuracy and provide a 45% speed-up over a single-threaded execution.

The rest of this paper is organized as follows. Section 2 reviews the Clustered Speculative Multithreaded architecture. Section 3 describes different value predictors and evaluates their prediction accuracy. Performance figures obtained by a timing simulator are presented in section 4. Section 5 reviews some related work and finally, section 6 summarizes the main conclusions of this work.

## 2. Clustered Speculative Multithreaded Architecture

This work focuses on the potential of value prediction to boost the performance of processors that exploit speculative multithreading. As a case study, we have considered the Clustered Speculative Multithreaded microarchitecture. This processor architecture (shown in Figure 1), is made up of several thread units interconnected by means of a ring topology, being each thread unit similar to a superscalar core.

The thread speculation logic of a Clustered Speculative Multithreaded processor is responsible for detecting those parts of a sequential program that can be executed by different threads. This architecture considers the beginning of loops as control quasi-independent points, since when these points are visited, they are very likely to be visited again in the near future (regardless of most of the branches inside the loop bodies). Thus, each speculative thread corresponds to a different iteration of an innermost loop. We will refer to the code executed by a speculative thread as a *loop trace*. The thread speculation logic also provides support for predicting the trace input/output data values and inter-thread memory dependences. If a value cannot be predicted by this logic, the threads are spawned anyway, but all the instructions that are dependent of the non-predicted value wait for the completion of the producer, which will store the value in a special register file (live-in register file) so that the following thread can read it. This architecture uses the MultiValue Cache to deal with memory dependences. More details about this microarchitecture can be found elsewhere [9][10].

### 2.1. The Performance Potential of Value Prediction

The main functionality of value prediction in a multithreaded environment is its ability to transform dependent threads into pseudo-independent ones. This ability becomes more important if the processor has a clustered architecture since these pseudo-independent instructions of speculative threads do not compete for the available resources.

Dependences among threads can be through registers and memory. We have evaluated than inter-thread register dependences are much more abundant that memory ones. On average, for the SpecInt95 suite, we have measured that a loop trace has 3.1 and 0.6 register and memory inputs respectively that are written by any of the previous 3 loop traces (see Figure 6). Figure 2 shows the speed-ups achieved by a Clustered Speculative Multithreaded processor with perfect register value prediction and perfect register and memory value prediction against the same clustered processor when it does not include any value predictor and it has to synchronize dependent instructions. On average, the speed-up achieved by predicting register values is quite important (close to 40%) and the improvement provided by predicting memory values is small in comparison with predicting only register values (only a 4% increase). Therefore, in this work, we focus on predicting register values, even though we consider that predicting memory values can also be important for the performance of some other codes.

## 3. Value Predictors

Value prediction is a technique that tries to exploit the fact that values tend to repeat or follow a known pattern over a large fraction of time. Therefore, values may be predicted correctly if an appropriate mechanism is used. These mechanisms are based on tables that store information reflecting the history that has been observed in the recent past.

This section is devoted to analyze the accuracy of value prediction in the context of a Clustered Speculative Multithreaded processor. The performance of this type of architecture strongly depends on the ability to predict the input or output values of speculative threads. As seen in the previous section, speculative threads correspond to loop iteration traces. In this way, we define a *trace input* as a value (in a register or

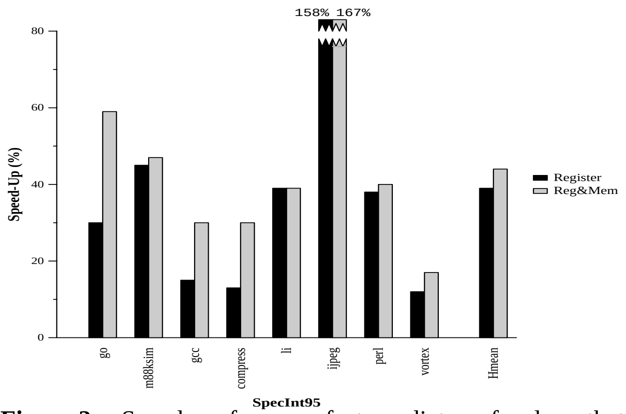


**Figure 2:** Speed-up for a perfect predictor of values that flow through inter-thread dependences, either in registers or memory.

memory location) that is live at the beginning of the trace (the trace uses it without having been computed by the same trace). In the same manner, we define a *trace output* as a value (in a register or memory location) that is computed by the trace. Thus, the analysis of value prediction focuses on trace input or output values, since these are the values that flow through inter-thread dependences.

The following subsections describe the value predictors investigated in this work. We classify the predictors into those that exploit correlation with past values of the same instruction operand and those that exploit correlation with values of the same trace. The former are called *instruction-based predictors* while the latter are called *trace-based predictors*.

Some of the predictors (last value, stride, context and hybrid predictors) have been thoroughly studied in the context of a superscalar and VLIW processors, but their performance for speculative multithreaded processors is unknown. In addition, we propose a new value predictor, the *increment* predictor, which is shown to have extremely high accuracy with very low cost.

## 3.1. Instruction-Based Value Predictors

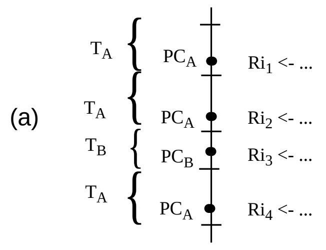
These predictors have in common that their history tables store information about the values seen by individual instruction operands. Well-known predictors are the last value (LV) [8], stride (STR) [5][12], context-based (FCM) [13] and hybrid schemes such as the stride-context (HYB-S) predictor.

The LV predictor assumes that the next value an instruction operand is the value in its previous execution. The STR predictor speculates that the new value seen by an instruction is the sum of the last value and a stride, which is the difference between two consecutive values. The predicted stride is replaced when a new stride has been seen twice in a row. The FCM predictor considers that values follow a repetitive sequence, and thus, estimates the new value based on the sequence of previous values of the same instruction operand. The HYB-S predictor is composed of a stride predictor and a context-based predictor. In this case, the choice between both predictors is guided by confidence counters.

Typically, a FCM predictor uses a Value History Table (VHT) containing the last *n* values seen by each instruction operand. These values are hashed to form an index to a Value Prediction Table (VPT). In all figures presented for FCM predictors (included the hybrid version) we have considered that the VHT contains the last 3 values and these values are 0-bit, 2-bit and 4-bit shifted, respectively, before xor-ing them in order to obtain the index to the VPT [13]. We also assume that the number of entries in each table is the same.

## 3.2. Trace-Based Value Predictors

The performance of instruction-based predictors can be improved if information about the trace to which the instruction operand to be predicted belongs is also included in the history tables. This gives way to the so-called trace-based value predictors. Traces being considered in this paper are the loop traces that the thread speculation unit of the Clustered Speculative Multithreaded architecture delimits for speculation. Nevertheless, this kind of predictors can be used for other types of traces.



(b) $Ri_4 = Ri_2 + \text{STRIDE}(PC_A)$

(c) $Ri_4 = Ri_3 + \text{INCREMENT}(T_A, Ri)$

**Figure 3:** Stride and increment predictors: a) 4 consecutive traces ($T_A$, $T_A$, $T_B$ and $T_A$) which reference register Ri; b) predicted value of $Ri_4$ using a stride predictor; c) predicted value of $Ri_4$ using the increment predictor.

Let us see a common case where instruction-based value predictors fail. The instruction-based stride predictor behaves badly when consecutive loop traces correspond to different paths of the same loop. Figure 3 shows an example where traces $T_A$, $T_A$, $T_B$ and $T_A$ are consecutively executed and Ri is the destination operand produced by instructions at $PC_A$ and $PC_B$. Consider also that these instructions compute an output trace value. The stride predictor would speculatively compute the value of Ri in the last trace ($Ri_4$) as the last value produced by the same instruction ($Ri_2$) plus the stride computed for that instruction operand. This may give an incorrect value since trace $T_B$ modifies the value of Ri.

### 3.2.1. Increment Predictor (INCR)

A stride predictor computes a difference between two consecutive values of an operand at the same instruction address. Writes to the same storage location produced between these two instructions affect the accuracy of the predictor. Instead, it may be better to base the value prediction of a storage location on the difference (the increment) of its value between two given points of the execution that always correspond to the same high-level structure, such as the beginning and the end of a loop trace [10].

The increment predictor predicts every trace output value as the value of that storage location at the beginning of the trace plus an increment. This increment is computed as the value at the end of the trace minus the value at the beginning of the trace in previous executions of the same trace. The predicted increment is updated when a new increment has been seen twice in a row.

Regarding Figure 3, note that the value of $Ri_3$, which is an output value of trace $T_B$ is also the value of register Ri at the beginning of the fourth trace ($T_A$). In this way, the value $Ri_4$ is predicted as $Ri_3$ plus the increment observed for this register in trace $T_A$ in the past. $Ri_3$ may in turn contain a predicted value, which was computed as $Ri_2$ plus the increment observed for this register in trace $T_B$ in the past. This scheme may be more accurate than an instruction-based predictor, since different traces are considered to update operands in a different way.

### 3.2.2. Other Trace-Based Predictors

Note that the instruction-based predictors presented in the previous section can be extended to correlate their predictions with previous instances of the same instruction operand in the same trace. This extension to convert them into trace-based predictors only requires minor modifications in the implementation, namely, the indexing function in the history table should consider both the instruction address and the trace identifier.

#### 3.2.3. Hybrid Increment-Context Predictor (HYB-I)

A hybrid scheme composed of the proposed increment predictor and a context-based predictor (HYB-I) will also be analyzed. For hybrid predictors, the choice between the two predictions is guided by confidence fields located in each individual predictor, which are implemented by means of 3-bit up/down saturating counters.

### 3.3. Indexing Schemes

For instruction-based predictors, the history tables are indexed through the instruction address of the operand to predict. If a source operand is to be predicted, a bit is appended to the instruction address in order to identify each source operand. A destination operand is directly identified by its instruction address. We refer to this indexing scheme as *PC-based indexing*.

Trace-based value predictors access the history tables through a trace identifier and a operand identifier (e.g. register identifier). For traces, we have considered a pseudo-identifier that consists of the instruction address of the first instruction of the trace along with a bit vector with the result of all conditional branches in the trace. It is not a unique identifier because a trace can have indirect unconditional branches, and their target branch addresses are not considered. We refer to this indexing scheme as *trace-based indexing.*

Note that trace-based predictors could alternatively index the history tables based on the instruction address of the producer (resp. first consumer) of the output (resp. input) value. Both types of indexing, PC- and trace-based indexing, are considered for trace-based predictors.

### 3.4. Prediction Accuracy

This section analyzes the accuracy of the different value predictors for the two indexing functions and different table capacities. The differences in predictability of inputs and outputs is also investigated. The objective is to devise the configurations with most potential, whose impact on IPC will be later analyzed in section 4.

In this section we use a trace-driven simulation of the SpecInt95 benchmark suite. The programs were compiled with the Compaq/Alpha compiler for an AlphaStation 600 5/266 with full optimization (-O4), and instrumented by means of the Atom tool. Programs used the reference input data during 200 millions of instructions after skipping the first 500 millions of instructions.

Loop traces being considered in this analysis are on average 36 instruction-length. Moreover, instructions belonging to loop traces represent almost the 62% of total instructions. Individual data for every program is depicted in Table 1.

#### 3.4.1. Predicting Register Values through PC-Based Indexing

A proper selection of the values to be predicted may have an important impact on the performance of data value speculation

| | instr in loop traces | instr / loop trace |
|---|---|---|
| go | 44.30 % | 40.57 |
| m88ksim | 83.00 % | 54.26 |
| gcc | 54.36 % | 32.09 |
| compress | 74.99 % | 16.19 |
| li | 38.82 % | 25.87 |
| ijpeg | 81.93 % | 43.38 |
| perl | 52.19 % | 50.90 |
| vortex | 75.75 % | 247.33 |
| **AVERAGE** | **61.73 %** | **36.13** |

**Table 1:** Loop trace statistics.

techniques [2]. We first compare the difference in predictability between trace input and output values. Note that predicting the outputs of previous loop traces is another way to obtain the input values of a loop trace. Value predictors being analyzed here use a PC-based indexing mechanism.

Figure 4.a shows the prediction accuracy for trace input register values whereas trace output register values are analyzed in Figure 4.b. The impact of the capacity of the history tables on the prediction accuracy is depicted along the X-axis. The INCR and HYB-I predictors are not depicted for input values since they only predict trace output values.

As observed for superscalar processors [13], a FCM can achieve a high prediction accuracy but it requires very large history tables. LV and STR predictors can achieve a better accuracy for small-sized ones. For large tables, the LV predictor is the least accurate. A remarkable result is that input values are more predictable than output values (70% of inputs for a 64-KB table using a STR predictor and 60% of outputs using a HYB-I predictor with the same capacity). Another important result is that a STR predictor outperforms an INCR predictor by around a 10%. The difference in performance of the respective hybrid predictors is not so high; in fact, HYB-I has a slightly advantage over HYB-S which suggests that the type of patterns predicted by the STR and the FCM have more overlap than those predicted by the INCR and the FCM.

Nonetheless, among all the inputs or outputs of a trace, only the prediction accuracy of those that are used speculatively will have an impact on performance. In other words, if a given input or output is already available at the time it is used, or an output is never utilized, the performance of the processor will be the same regardless of the result of its prediction. To estimate this

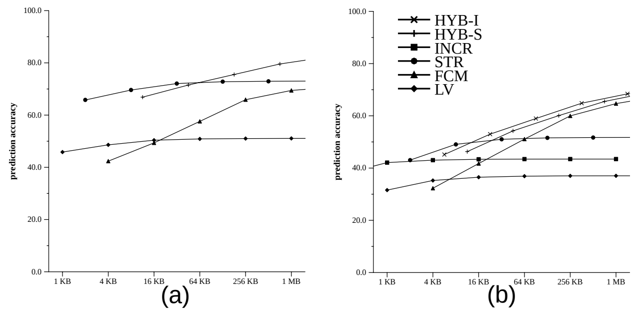


**Figure 4:** Predicting register values of loop traces using PC-indexed predictors: a) trace input values; b) trace output values.

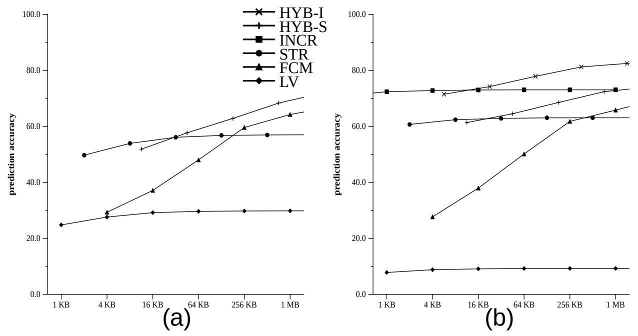


**Figure 5:** Predicting distance-3 values of loop traces using PC-indexed predictors: a) input values; b) output values.

effect, we compute the prediction accuracy for those input values produced by any of the previous 3 traces and those output values consumed by any of the following 3 traces. We refer to these values as distance-3 inputs and outputs respectively.

For distance-3 inputs (see Figure 5.a), the prediction hit rate diminishes when compared with that of predicting all values (it goes from 70% for a 64-KB HYB-S predictor as it can be seen in figure 4.a, to 60%). However, for distance-3 output values (see Figure 5.b), this trend is reversed. A 64-KB HYB-I predictor increases the prediction accuracy by 20% when compared with its accuracy for all outputs.

This is due to the fact that traces have in average more register outputs than inputs (8.2 vs 5.0), as shown in Figure 6. However, the average number of distance-3 register outputs is lower than distance-3 register inputs. The figure also includes statistics for memory values, showing that the average number of distance-3 memory inputs and outputs are rather low.

Another remarkable fact is that the INCR predictor outperforms the STR predictor by about 10% for distance-3 output values. This is explained by the fact that the stride predictor suffers from interferences from other instructions with different addresses that write to the same storage location, as discussed in section 3.2, whereas these interferences are avoided by a trace-based predictor such as the INCR, even if the indexing function uses only the instruction address.

As conclusions up to this point, for a speculative multithreaded architecture based on loop traces, the most predictable values are trace outputs. Moreover, the INCR predictor for small sized tables and its hybrid version, the HYB-I predictor, for larger tables outperform the other value predictors. An increment predictor can achieve a quite high hit rate with very small tables (73% for a 1 KB table).

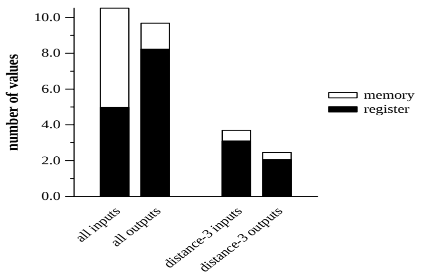


**Figure 6:** Average number of inputs/outputs and distance-3 inputs/outputs per trace.

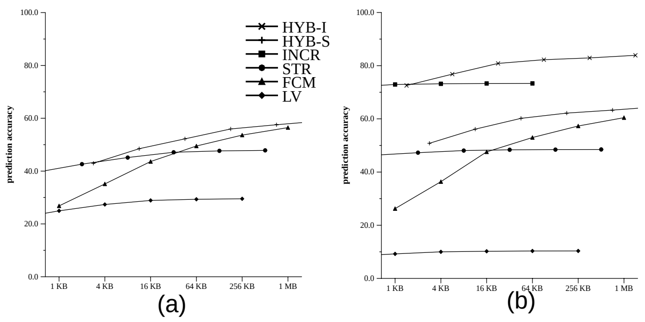


**Figure 7:** Predicting distance-3 values of loop traces using trace-based indexing: a) input values; b) output values.

### 3.4.2. Trace-Based Indexing

For trace-based predictors, the trace identifier can be included in the indexing function. Figure 7 shows the prediction accuracy for distance-3 input/output vales. It can be observed that the hit rate for input values decreases when compared with PC-based indexing (60% for a 64-KB PC-indexed HYB-S versus 50% for the trace-indexed version of the same predictor). However, the performance for output values increases. Although the INCR predictor obtains a similar performance (73% hit rate for the whole range of table capacity), the HYB-I predictor can achieve a 80% hit ratio with relatively small tables (16 KB in total). This is mainly caused by the significant performance boost of trace-based indexing for the FCM predictor, which is due to the benefits of using different value sequences for different traces.

As conclusions of this analysis on prediction accuracy, the four selected predictors for a further analysis in the context of a cycle-based timing simulator are those with the highest prediction accuracy for a moderate-sized history table (16 KB): HYB-I, INCR, HYB-S and FCM, with a trace-based indexing function.

For this four predictors, Figure 8 shows the percentage of traces whose all distance-3 outputs are correctly predicted. This gives an estimation of the percentage of traces that can be executed as if they were parallel. Note that many traces can be parallelized due to value prediction, even with small predictors (50% for 1-KB INCR predictor). With large history tables, this percentage can be as much as 70% (with a HYB-I predictor).

## 4. Performance Evaluation

In this section, we present performance figures that show the impact of different value predictors on the performance of a

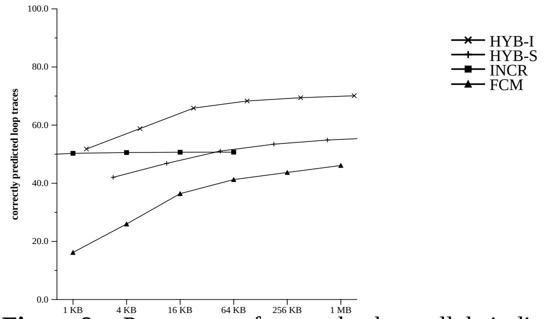


**Figure 8:** Percentage of traces that have all their distance-3 output values correctly predicted.

Clustered Speculative Multithreaded processor. The predictors with most potential according to the analysis of section 3 are considered: increment (INCR), context-based (FCM), hybrid of increment and context-based (HYB-I) and hybrid of stride and context-based (HYB-S). All of them use a trace-based indexing scheme.

## 4.1. Experimental Framework

The performance of the value predictors on a Clustered Speculative Multithreaded processor was evaluated through timing trace-driven simulation of the SpecInt95 benchmark suite. The programs were compiled with the Compaq/Alpha compiler for an AlphaStation 600 5/266 with full optimization (-O4) and instrumented by means of the Atom tool. They were run until completion with the train input data sets.

The baseline Clustered Speculative Multithreaded processor has 4 thread units. The fetch bandwidth of the architecture is up to 4 instructions per cycle or up to the first taken branch, whichever is shorter, and the fetch policy for threads with different control flow is round-robin. The MultiValue cache, which is responsible of serializing memory-dependent instructions from different threads, has 128 entries and a 64-KB non-blocking, 2-way set-associative L1 cache with an 32-byte block size and up to 4 outstanding misses. The L1 latency is 3 cycles for a hit and 7 for a miss. An ideal L2 cache memory is considered. The Loop Iteration Table (LIT) has 8 entries and a perfect trace predictor is assumed. Each thread unit has the following features: issue up to 4 instructions per cycle, 64-entry reorder buffer, local $2^{14}$-entry gshare and the following funtional units (latency in brackets): 2 simple integer (1), 1 integer multiplication (4), 2 simple FP (4), 1 FP multiplication (6), and 1 FP division (17).

The size of the value predictors has been approximately limited to 16 KB. In particular, each table of the FCM predictor has 1024 entries, while the table of the INCR predictor has 4096 entries. For the hybrid predictors, each table of the HYB-S predictor has 512 entries, whereas for the HYB-I they have 1024 entries. Note that the size of the INCR predictor could be significantly reduced without affecting performance, according to the analysis presented in section 3. Output values are always predicted, and in case of misprediction, in addition to wait for the correct value to be produced, a 1-cycle penalty is assumed.

## 4.2. Performance Figures

Figure 9 shows the instructions committed per cycle (IPC) for each SpecInt95 benchmark and each value predictor. We can see that the IPC for the different predictors is correlated with their prediction accuracy (see Figure 7). The HYB-I predictor outperforms the others for the whole range of applications. However, we can see that the differences between the HYB-I and the INCR predictors are rather low. We can also observe that the HYB-S predictor does not substantially improve the performance in comparison with the FCM predictor except for several benchmarks such as *li*, *ijpeg* and *perl*.

As it was shown in Figure 7, the shape of the curve that plots prediction accuracy versus table capacity for the INCR predictor is almost flat beyond 1 KB. In addition, it is much simpler than the other three predictors. Thus, we can conclude that the INCR predictor is the most cost-effective predictor.

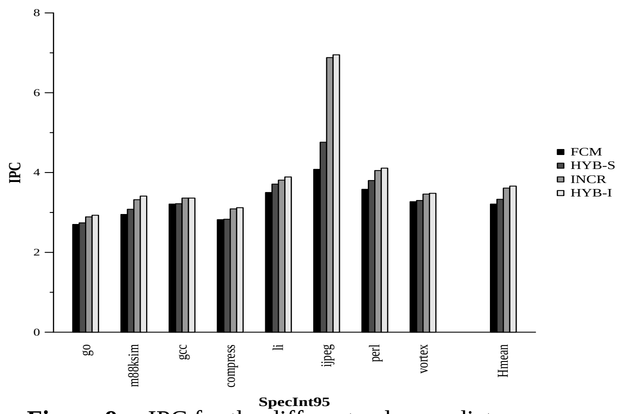


**Figure 9:** IPC for the different value predictors.

Figure 10 shows the speed-up achieved by a Clustered Speculative Multithreaded processor with an increment predictor against the single-threaded execution of the code. The maximum performance for a perfect value predictor is also shown for comparison. Note that the average speed-up is over 40%, which is quite significant for a benchmark suite that is very hard to parallelize. For *ijpeg*, which is the program with more parallelism, the IPC is twice that of a single-threaded execution. Moreover, the performance provided by the INCR predictor is close to that of a perfect value predictor. On average it is just 13% lower and in some programs such as *gcc* it is almost the same (just 3% lower).

## 5. Related Work

Several value predictors have been proposed in the past. The first proposal was the last value predictor, presented by Lipasti, Wilkerson and Shen [8]. Afterwards, more complex value predictors have been proposed such as the stride predictor [5][12] and the FCM [13] context-based value predictor among others, and some combinations of them to obtain hybrid value predictors [18].

These predictors, as well as variations of them, have been studied for superscalar processors. González and González [6] pointed out that value prediction had more potential in multithreaded processors, but they just evaluated a stride predictor in an ideal machine with unlimited resources in most of its components.

On the other hand, several works proposing multithreaded architectures which provide support for speculative threads

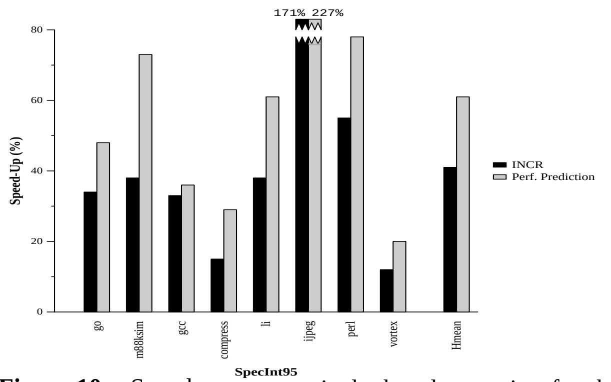


**Figure 10:** Speed-up versus single-thread execution for the INCR predictor and a perfect predictor.

have recently appeared. Pioneer work on this topic was the Expandable Split Window paradigm [4] and the follow-up work on Multiscalar processors [15]. Other proposals are the SPSM architecture [3], the Superthreaded architecture [17], the Trace Processor [11], the Clustered Speculative Multithreaded architecture [9][10] and the Dynamic Multithreaded processor [1].

The Multiscalar, SPSM and Superthreaded architectures do not have any mechanism for data value speculation and data dependences are always enforced by executing the producer instruction before the consumer one.

The Dynamic Multithreaded, the Clustered Speculative Multithreaded architectures and Trace Processors provide support for value speculation. Whereas the mechanism provided in the Dynamic Multithreaded is very simple since the register file of the parent is copied into the child register file, the Trace Processor and the Clustered Speculative Multithreaded architecture use more complex and suitable predictors like context-based and the increment.

Moreover, several works have studied the effectiveness of speculative threads on multiprocessor platforms ([7][16] among others), but they do not provide any mechanism for predicting data values.

## 6. Conclusions

We have studied the performance of value prediction for speculative multithreaded architectures and have presented a new value predictor targeted to this type of architectures, which is referred to as increment predictor. The increment predictor computes a new value for a trace output as its value at the beginning of the trace plus an increment.

Experimental results have shown the importance of choosing the correct values to be predicted. Output trace values are more predictable than inputs. Moreover, trace-based indexing outperforms PC-based indexing. We have also shown that the increment predictor obtains the highest prediction accuracy with small-sized history tables. This accuracy is increased for larger history tables by means of a hybrid predictor that combines an increment and a context-based predictors. Average accuracy for SpecInt95 ranges from 73% to 84% depending on the capacity of the history table.

A Clustered Speculative Multithreaded processor with a 16-KB increment predictor achieves an average IPC of 3.61, which is 9% higher than the IPC of a hybrid stride/context predictor. The speed-up over a single-threaded execution is over 40%.

We can conclude that predicting the values that flow through inter-thread dependences is crucial for the performance of processors that exploit speculative thread-level parallelism.

## 7. Acknowledgments

This work has been supported by grants CICYT TIC 511/98, ESPRIT 24942 and AP96-52274600. The research described in this paper has been developed using the resources of the European Center for Parallelism of Barcelona (CEPBA).

## 8. References

[1] H. Akkary and M.A. Driscoll, “A Dynamic Multithreading Processor”, in *Proc. of the 31st. Int. Symp. on Microarchitecture,* 1998.

[2] B. Calder, G. Reinman and D.M. Tullsen, “Selective Value Prediction”, in *Proc. of the 26th. Int. Symp. on Computer Architecture*, 1999.

[3] P.K. Dubey, K. O’Brien, K.M. O’Brien and C. Barton, “Single-Program Speculative Multithreading (SPSM) Architecture: Compiler-Assisted Fine-Grained Multithreading”, in *Proc. of the Int. Conf on Parallel Architectures and Compilation Techniques*, pp. 109-121, 1995.

[4] M. Franklin and G. Sohi, “The Expandable Split Window Paradigm for Exploiting Fine Grain parallelism”, in *Proc. of the Int. Symp. on Computer Architecture*, pp. 58-67, 1992.

[5] F. Gabbay and A. Mendelson, “Speculative Execution Based on Value Prediction”, *Technical Report, Technion*, 1997.

[6] J. González and A. González, “Data Value Speculation in Superscalar Processors”, in *Microprocessors and Microsystems,* 22(6), pp. 293-302 November 1998

[7] L. Hammond, M. Willey and K. Olukotun, “Data Speculation Support for a Chip Multiprocessor”, in *Proc. Int. Conf. on Architectural Support for Prog. Lang. and Op. Systems,* 1998

[8] M.H. Lipasti, C.B. Wilkerson and J.P. Shen, “Value Locality and Load Value Prediction”, in *Proc. of the 7th. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 138-147, Oct. 1996.

[9] P. Marcuello, A. González and J. Tubella, “Speculative Multithreaded Processors”, in *Proc. of the 12th Int. Conf. on Supercomputing*, pp. 77-84, 1998.

[10] P. Marcuello and A. González, “Clustered Speculative Multithreaded Processors”, in *Proc. of the 13th Int. Conf. on Supercomputing*, 1999.

[11] E. Rotenberg, Q. Jacobson, Y. Sazeides and J.E. Smith, “Trace Processors”, in *Proc. of 30th. Int. Symp. on Microarchitecture*, pp. 138-148, 1997.

[12] Y. Sazeides, S. Vassiliadis and J.E. Smith, “The Performance Potential of Data Dependence Speculation & Collapsing”, in *Proc. of the 29th. Int. Symp on Microarchitecture*, Dec. 1996.

[13] Y. Sazeides and J.E. Smith, “Implementations of Context-Based Value Predictors”, Technical Report #ECE-TR-97-8, University of Wisconsin-Madison, 1997.

[14] SIA Semiconductor Industry Association, *The National Technology Roadmap for Semiconductors,* 1997.

[15] G. Sohi, S.E. Breach and T.N. Vijaykumar, “Multiscalar Processors”, in *Proc. of Int. Symp. on Computer Architecture*, pp. 414-425, 1995.

[16] J. Steffan and T. Mowry, “The Potential of Using Thread-Level Data Speculation to Facilitate Automatic Parallelization”, in *Proc. 4th Int. Symp. on High-Performance Computer Architecture,* pp. 2-13, 1998

[17] J.Y. Tsai and P-C. Yew, “The Superthreaded Architecture: Thread Pipelining with Run-Time Data Dependence Checking and Control Speculation”, in *Proc. of the Int. Conf. on parallel Architectures and Compilation Techniques*, 1996.

[18] K. Wang and M. Franklin, “Highly Accurate Data Value Prediction Using Hybrid Predictors”, in *Proc of the 30th Int. Symp. on Microarchitecture,* pp. 281-190, 1997.

[19] D.W. Wall, “Limits of Instruction-Level Parallelism”, *Tech. Report WRL 93/6,* Digital Western Research Lab., 1993.