The Potential of Data Value Speculation to Boost ILP

José González and Antonio González

Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Barcelona, Spain

Email: {joseg,antonio}@ac.upc.es

Abstract

Instruction Level Parallelism (ILP) is one of the key issues to boost the performance of future generation processors. Data dependences may become one of the main bottlenecks that limit the performance of such future architectures due to the serialization that they introduce in the execution of programs. Data value speculation is a novel mechanism that may avoid such serialization by predicting the values that flow among dependent instructions. This work presents a study of the limits of the performance potential of data value speculation to boost ILP. Both realistic and probabilistic value predictors are considered in this paper. The interaction with some other critical features of the microarchitecture is researched: instruction window size, branch prediction accuracy and instruction latency. Results show that data value speculation has a significant potential to boost ILP. However, for a relatively small instruction window, like the one that can currently be built with the best branch predictors, the achieved speedup is moderate. The speedup provided by data value speculation is also dependent on the criticality of the correctly predicted instructions. Finally, it is shown that the speedup that data value speculation may achieve in superscalar processors, in the way that it has been used so far, can be approximated by a linear function of the prediction accuracy.

Keywords: Data value speculation, limits of ILP, value prediction.

1 Introduction

Future microprocessors, which will likely have a large instruction window, may have a limited performance due to dependences among the instructions of programs. These dependences can be classified into three types: name dependences, control dependences and data dependences[7].

Name dependences are caused by the reuse of storage locations, i.e. when two different instructions write their result in the same register or in the same memory location. Register renaming [19] and memory renaming [13][21] are two well-known techniques that deal with this problem by assigning a new storage location to each instruction that produces a value.

Control dependences are caused by branch instructions. They may introduce a serialization in the execution of a program since the following instructions depend on the branch outcome. The mechanism that current microprocessors use to avoid this limitation is known as *control speculation*. It is based on predicting the outcome of branches, and executing speculatively instructions from the predicted path.

Data dependences appear when an instruction produces a value that is to be read by another subsequent instruction. These dependences impose a serialization on the execution of a program since the consumers have to wait until the producers are executed. *Data value speculation* is a new technique that tries to avoid this serialization. It works in a similar way as control speculation: the results or the operands of some instructions are predicted before they are actually computed and those instructions that depend on the predicted ones are speculatively executed. All that a processor needs in order to implement data value speculation is a *value predictor*, that is, a hardware that carries out the predictions, and a mechanism to recovery from misspeculations.

Previous work on data value speculation has focused on proposing different value predictors and analyzing their impact on particular microarchitectures. This work is concerned with the limits of the performance potential of data value speculation. We evaluate its benefits for a processor with infinite resources and perfect branch prediction. Then, the influence of some critical processor features is considered: instruction window size, branch prediction accuracy and instruction latency. Finally, the potential benefits of enhancing the value predictor accuracy is analyzed by means of a probabilistic predictor with a parameterized accuracy.

The rest of the paper is organized as follows: Section 2 reviews the related work. The methodology to evaluate the ILP that can be exploited by both an ideal processor and a processor with a limited instruction window and non-perfect branch prediction, with and without data value speculation, is described in Section 3. Section 4 presents the value predictor considered in this work. Section 5 analyzes the results of this study and finally, section 6 summarizes the main conclusions of this work.

2 Related Work

Studying the limits of ILP under some constraints has been the aim of a plethora of works [1][2][8][9][17][20][23]. Each work analyzes the impact that some microarchitectural features have on

ICS 98 Melbourne (Australia)

the microprocessor performance. These features may be the instruction window size, branch prediction accuracy, fetch bandwidth, register renaming, memory aliasing, etc. The main conclusion that one can extract from those works is that there is not only one key issue that determines the processor performance. However, data dependences may be one of the important bottlenecks of future microprocessors. For instance, in [7], the results show that the amount of ILP that is expected to achieve a processor with infinite resources and perfect branch prediction is not much higher that a few hundred instruction per cycle (IPC), and, for some applications only a few tens of IPC.Recently, data value speculation has been proposed as a technique to go beyond the limits imposed by data dependences. This technique was first presented by Lipasti *et al* in [11]. In that work, the values that loads read from memory are predicted based on the observation that some loads repeatedly read the same values. The type of predictor that they use is called *last value predictor*. In [5] values of loads are also predicted by predicting the addresses (using a *stride predictor*) and prefetching the value from memory. This approach is based on the fact that load addresses are more predictable than load values. In that work, store addresses are also predicted in order to improve the memory disambiguation scheme by predicting memory data dependences based on the predicted addresses. This latter feature is usually known as *data dependence speculation*. In [9], Lipasti and Shen extend their last value predictor initially proposed for loads to predict the results of all the instructions. In [15], Sazeides and Smith show that the results that an instruction generates may follow a regular pattern that a *stride predictor* can fail to predict. In [16], the implementation of a new predictor called *context value predictor* is introduced. This scheme can predict the results of instructions that follow a regular pattern. In [22], Wang and Franklin present a hybrid predictor. The implementation of this predictor is similar to that of a *2-level* branch predictor. The impact of different value predictors on the performance of a processor with a 40-entry instruction window is evaluated in [3]. Finally, the potential of using profiling to help the predictors is analyzed in [4].

These previous works on data value speculation focus on either improving the value predictor accuracy or evaluating their impact on a particular processor microarchitecture. None of them presents a study of the limits of the performance potential of data value speculation when some current performance bottlenecks are removed, like the instruction window size, the branch prediction accuracy, etc, which is the target of this work. A very limited intent is presented in [10], where value prediction is evaluated for an instruction window of 4096 entries but with a limited fetch bandwidth and branch predictor.

3 Methodology

This section describes the methodology employed in this work to obtain the ILP of some applications under different scenarios regarding prediction schemes and hardware resources.

3.1 Experimental Framework

The evaluation methodology is trace-driven. The trace of each program has been generated using the ATOM tool [18]. This trace feeds an analysis program, which computes the performance, reported as Instructions Per Cycle (IPC), achieved by a particular architectural configuration. Table 1 shows the list of Spec95 programs that have been simulated along with their input set and the number of instructions executed. Go,Gcc, Li and Perl are SpecInt whereas the rest belong to the SpecFp suite. Each program has been compiled for a DEC AlphaStation600 5/266 with '-O4' optimization flag.

Program name	Input set	Million of instructions executed
Go	9 9	122
Perl	test set	150
Li	test set	150
Gcc	genrecog.i	143
Hydro2d	test set	150
Turb3d	test set	150
Applu	test set	150
Fpppp	test set	150

Table 1: Benchmarks from Spec95

3.2 Approach to compute the IPC

The approach used to compute the IPC of each program is based on an extension to the scheme proposed in [1] in order to include branch prediction and data value prediction. By using the trace of each program, a *Data Dependence Graph (DDG)* is built, where instructions are the nodes and the values that flow among instructions are the edges. Each edge is labeled with the cycle on which the value will be available, which is referred to as its *compute time*. In the case of an infinite instruction window, this cycle is computed as the maximum among the edges that arrive to the node plus the latency of the instruction. In other words, an instruction produces a value when the instructions on which it is dependent have finished, plus its latency. The IPC of a program is computed dividing the number of executed instructions by the cycle when the latest result of any instruction is produced. Since we are just interested in the global IPC, only the leaves of the DDG must be stored during the simulation.

When a finite instruction window and in-order retirement are considered, an instruction cannot start its execution until the instruction W (W is the instruction window size) locations above in the trace and the instructions on which it is data dependent have produced its result.

As far as branch prediction is concerned, the effect of branch mispredictions are considered by delaying the firing of instructions below the mispredicted branch until the branch is computed plus a misprediction penalty, which is considered to be a fixed number of cycles.

When data value speculation is considered, a new variable is managed by the simulator: the *prediction time*, which refers to the cycle when a predicted value is available. It depends on the type of prediction considered:

- Serialized: Only one prediction per static instruction can be performed at most per cycle. That is, an instruction is not predicted until the previous execution of the same static instruction has been predicted.
- Non serialized: Multiple executions of the same static instruction can be predicted at the same time. For instance, this scheme may predict the result of all the executions of a static instruction inside a loop all at once.

In this work, only serialized predictors are considered. Results for non-serialized predictors are shown in [6].

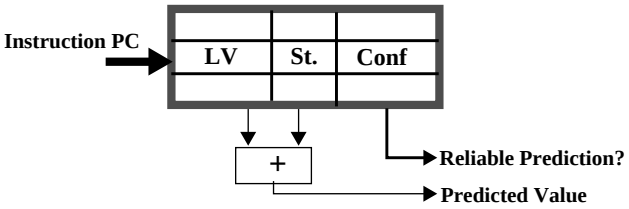

Figure 1: A stride-based predictor

Regarding a serialized predictor, the *prediction time* of a given dynamic instruction is computed as the time when the value produced by the last instruction that was mapped to the same entry is available (either predicted or computed), plus the latency to perform a prediction. In this work, the prediction latency is considered to be 1 cycle. This means that only one prediction per each predictor entry is allowed in each cycle.

The cycle when the correct value of a dynamic instruction is available is computed as the minimum between the prediction time and the compute time.

Further details about implementation aspects can be found in [6].

4 Data Predictors

The value predictor used in this work is a *stride-based predictor*. This type of predictor has been already used in previous works [3][5][14]. Figure 1 shows its basic structure. It is implemented by means of a table of 4096 entries, direct mapped and non-tagged. It is indexed using the least significant bits of the instruction PC. Each entry stores the following information:

- Last Value (LV): Last value produced by the instruction.
- Stride (St): This field contains the difference between two consecutive values.
- Confidence (Conf): A two bit up/down saturating counter that is used to assign confidence to the predictions.

Each instruction that is to be predicted accesses the table using the PC and computes the predicted value by adding the stride to the last value. If the most significant bit of the confidence flag is set, then the prediction is assumed to be correct, and the dependent instructions are allowed to use this value speculatively. Otherwise, the instruction is not executed until the input operands are available.

This type of predictor is used to predict the results of arithmetic instructions and loads. Besides, the same type of predictor is used to predict the addresses and values of store instructions. A perfect memory disambiguation scheme and an infinite memory renaming mechanism are assumed. That is, a store can be executed as soon as both its address and value are either computed or predicted. A load value is either predicted correctly or the load has to wait until its input operands are available and the previous store to the same address has been executed.

In this work, each type of predicted instructions has its own table.

5 Results

This section analyzes the potential benefits of data value speculation for an ideal processor with infinite resources: fetch bandwidth, number of functional units, register and memory ports, register and memory storage, etc. The potential improvement is shown in terms of speedup, which is computed by dividing the IPC achieved by the processor when data value speculation is considered by the IPC obtained by the same processor without data value speculation.

Two different branch prediction schemes have been considered: a perfect branch prediction (PBP) and a "realistic" branch predictor (GsBP), which is a *gshare* [12] with a 2^{16} entry pattern history table. This table is indexed by xoring a 16-bit history register with the 16 least significant bits of the branch PC. The instructions that follow a mispredicted branch have to wait until the branch computes its outcome plus 1 cycle of penalty.

5.1 Impact of the instruction window size

We will first analyze the impact of data value speculation when the main performance bottleneck is the instruction window size, in addition to data dependences.

Results in this section are presented dividing data value speculation into three different categories: predicting only memory instructions, predicting simple arithmetic instructions, that is, additions, substractions, etc. (both integer and floating point), and predicting all previous instructions together. Figure 2 shows the performance of data value speculation for an ideal machine, with a finite instruction window and assuming that all the instructions have a 1-cycle latency. The Y axis represents the speedup provided by data value speculation, whereas the X axis represents the instruction window size. Notice that the scale of the Y axis changes depending on the program. Each graph has six different curves. Solid lines represent the speedup when data value speculation is performed by predicting memory instructions, arithmetic instructions and all together, in a processor with perfect branch prediction. Dotted lines represent the performance of data value speculation when a *gshare* branch predictor is considered. Table 2 shows the percentage of instructions correctly predicted and confident in their predictions. This percentage is split into the different instruction types and it is computed by dividing the number of instructions correctly predicted by the total number of instructions of each type. The grey columns show the speedup achieved by predicting loads, stores and arithmetic instructions in a processor with an infinite instruction window. The last column shows the misprediction ratio achieved by the gshare branch predictor.

The results in Figure 2 show that the size of the instruction window has an important influence on the performance of data value speculation. All the programs except *go* increase their speedup when the window grows. Notice that a decrease in speedup does not mean a decrease in IPC. It means that, when the instructions window size is increased, the IPC for data value speculation increases more slowly than the IPC when data values are not speculated. The particular behavior of go may suggest that for this program the impact on the execution time of the instructions that have a stride behavior is not very important, and the main benefits in terms of IPC come from increasing the window size.

Looking at the results for different types of instructions, it seems that data value speculation benefits most from arithmetic prediction. The reason may be that in this first study all instructions have the same latency, and the percentage of arithmetic instructions is greater than the percentage of memory instructions.

As far as an infinite instruction window is concerned, one can see that there is not a correlation between the value prediction accuracy and the speedup obtained. For instance the predictor accuracy for *perl* is better than for *li* for all types of instructions, but the speedup of *li* is 16 times higher. This fact points out that, as far as the benefits of data value speculation are concerned, not only the

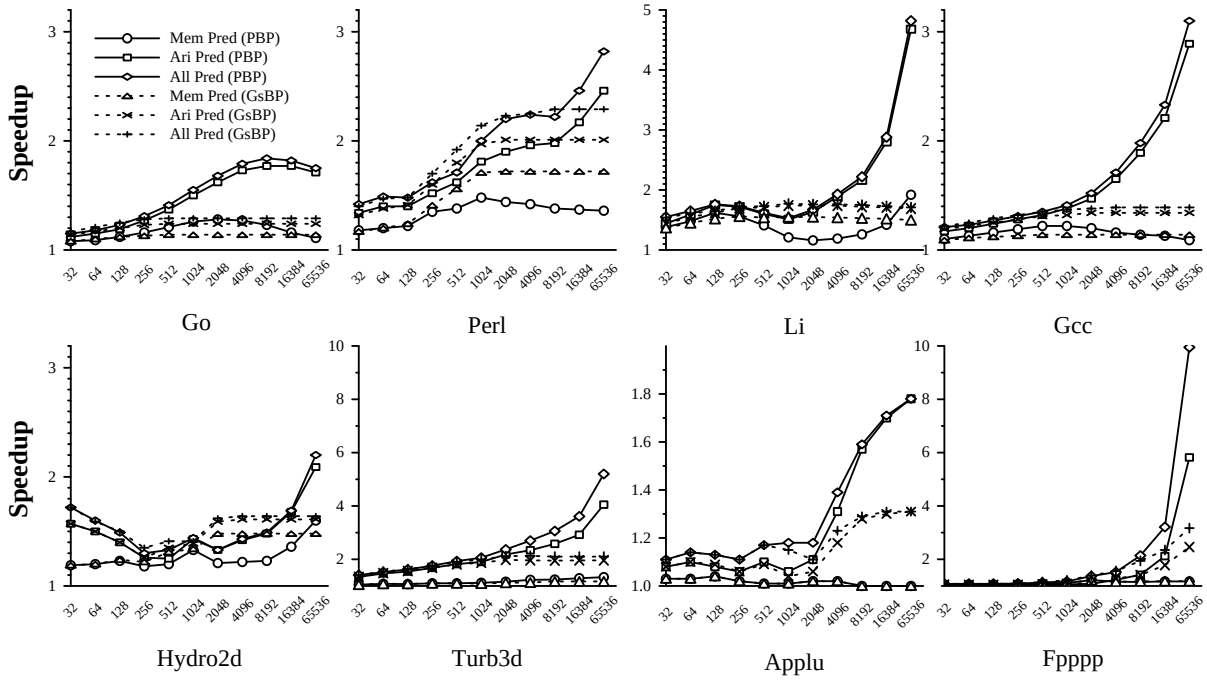

Figure 2: Speedup achieved by data value speculation depending on the instruction window size

percentage of predicted instructions is important but also whichinstructions are predicted, i.e. those instructions placed on the critical path are more attractive to predict than those that have a much lower influence on the total execution time of the program.

	Ld	St Addr	St Value	Iarth	Farth	Speedup PBP	Speedup GsBP	Gshare miss
Go	35	48	30	29	-	1.10	1.29	16.39
Perl	66	70	63	58	-	5.90	2.29	3.03
Li	50	27	42	55	-	82.10	1.71	3.07
Gcc	39	42	41	42	-	6.63	1.39	7.88
H2d	78	86	85	78	86	8.62	1.65	0.48
T3d	65	59	69	53	72	16.29	1.93	2.29
Applu	33	78	15	42	1	10.44	1.47	0.44
Fppp	37	94	18	57	0.5	136.01	3.48	4.10

Table 2: Prediction accuracy for each type of instructions and speedup achieved predicting all the instructions for an infinite instruction window. The last column shows the gshare misprediction rate.

Branch prediction has a very important influence on the potential benefit of data value speculation. Looking at Figure 2, one can see that the speedup achieved for realistic branch predictor does not increase once a certain window size is reached. This is due to the fact that the effective instruction window, i.e. the number of entries of the instruction window that are used in average, is determined by the branch predictor accuracy. Once the instruction window reaches a certain size, any additional increase in size is almost useless since the branch mispredictions do not allow the processor to take benefit of it. In this case, the performance for both scenarios (with and without data value speculation) hardly changes and the graphs in Figure 2 flatten out. This is especially true for integer programs, where branch prediction is a main performance bottleneck.

For an infinite instruction window, the difference between perfect and realistic branch prediction is really significant, and only *go* seems to benefit more from data value speculation when branches are predicted. This particular benchmark has the highest branch misprediction rate, and its performance seems to be determined by such branches. When data value speculation is performed along with a realistic branch prediction, the penalty of mispredicted branches is reduced because the prediction of their inputs allows for an earlier computation of the branch outcome. This fact may explain that the speedup is a bit higher for a realistic branch prediction than for a perfect branch prediction.

Finally, an important conclusion from the results in Figure 2 is that the benefits of data value speculation are moderate. For instance, if the instruction window is not higher than 4096, the speedup of data value speculation is lower than 3 for all the cases, even when a perfect branch predictor is considered. For a 256-entry instruction window, which may seem more realistic for forthcoming microprocessors, the speedup achieved by data value speculation is lower than 2 for the 8 analyzed applications. One may argue that the accuracy of the stride predictor is not very good. However, we will show later that the benefits that one may expect from enhanced predictors are quite moderate. Another argument may be that instructions with a stride behavior may not be usually in the critical path. Evidence of this fact will be shown in section 5.3.

5.2 Instructions latencies

Figure 3 shows the speedup obtained when instructions have a “realistic” latency. Table 3 shows the latency assumed for each type of instructions. Loads and stores have a 2-cycle latency, one for address computation and another one for access memory, since an infinite cache memory is assumed. Each graph has 6 different curves. Solid curves represent the speedup that data value

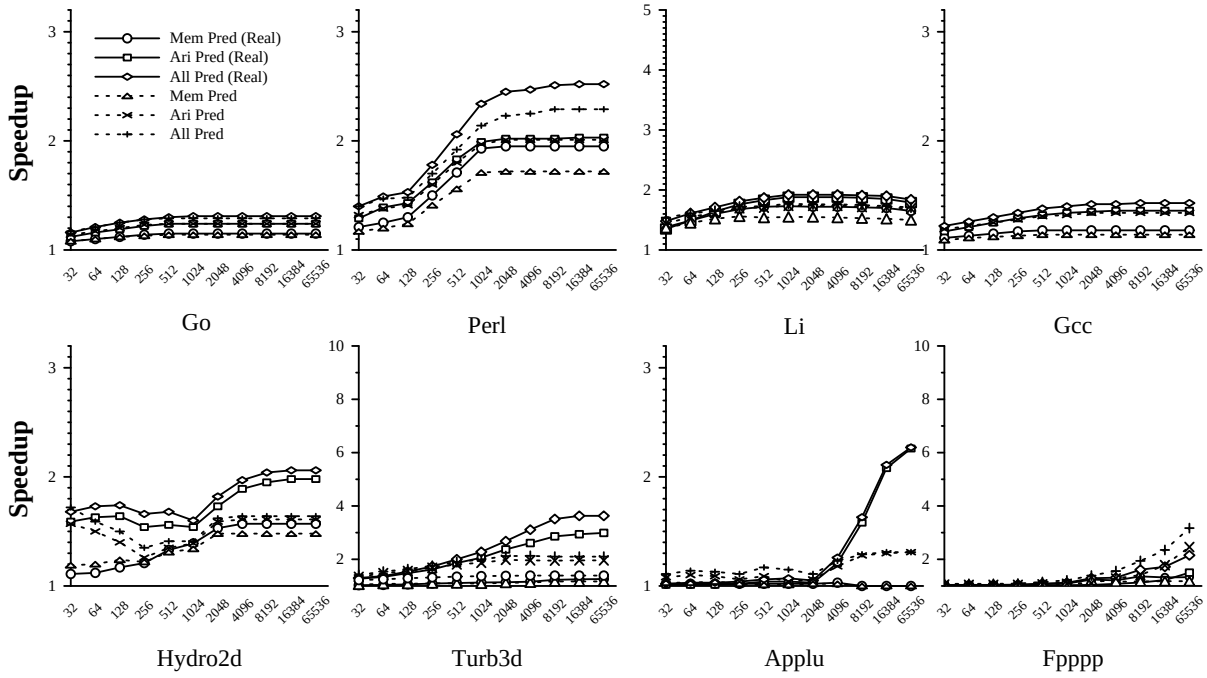

Figure 3: Speedup achieved by data value speculation using a gshare branch predictor for both "realistic" and 1-cycle latencies.

speculation achieves when instructions have a "realistic" latency. This is computing by dividing the IPC obtained when data value speculation is performed and latencies are those of Table 3 by the IPC achieved without data value speculation with the same latencies. Branches are predicted using the *gshare* predictor. The dotted lines show the speedup that data value speculation provides when all instructions have a 1-cycle latency, and branches are predicted using the *gshare*. These lines are the same than the dotted lines in Figure 2.

The results show that when realistic latencies are considered, the impact of data value speculation is somewhat greater than that observed for 1-cycle latency instructions. Notice that the latency of simple arithmetic instructions remains the same, and therefore there is not difference between the dotted and the solid lines for arithmetic predictions. On the other hand, increasing the memory latency to 2 cycles has an influence on the speedup since these instructions become more important on the critical path. Combining arithmetic and memory prediction benefits the overall results.

Inst. Type	**Latency**
Int. Arith	1
Int. Mult.	9
In.t Div.	67
Fp. Arith.	4
Fp. Mult.	4
Fp. Div.	16
Loads/Stores	2

Table 3: Latency of instructions

5.3 Impact of data prediction accuracy

Data value speculation is a quite novel area of research in which many important issues require much more effort in order to understand the phenomenon and its potential benefits. A few different value predictors have been proposed so far and their prediction accuracy is still moderate, which may be one of the reasons why the benefits of the data value speculation mechanism proposed so far are also moderate. In this section we are interesting in assessing how much benefit one may expect from more accurate predictors.

To measure this potential, we consider a probabilistic value predictor with a parameterized prediction accuracy. A probabilistic predictor consists of a pseudo-random number generator that produces a value between 0 and 1 for each instruction of the trace that is predicted. If this number is below a certain threshold, which is the prediction accuracy, the instruction is processed as correctly predicted.

Figure 4 shows the speedup obtained when data value speculation is performed using a probabilistic predictor for a fixed instruction window size of 256 entries. The X axis represents the prediction accuracy whereas the Y axis shows the speedup (solid lines), which is computed by dividing the IPC with data value speculation by the IPC without it. There are two different lines, one for a perfect branch predictor and another for the *gshare* predictor. Besides, the dotted lines show the speedup obtained when branches are predicted using a probabilistic branch predictor. This speedup is computed by dividing the IPC achieved by a processor with a certain branch prediction accuracy over the IPC achieved when branches are not predicted. Notice that in this case data value speculation is not involved at all.

The aim of this figure is not to compare the performance of branch prediction and value prediction but to illustrate the quite different impact of prediction accuracy on the processor performance. Notice also that the objective of these graphs is not to conclude: having a value predictor of accuracy X, the speedup would be Y.

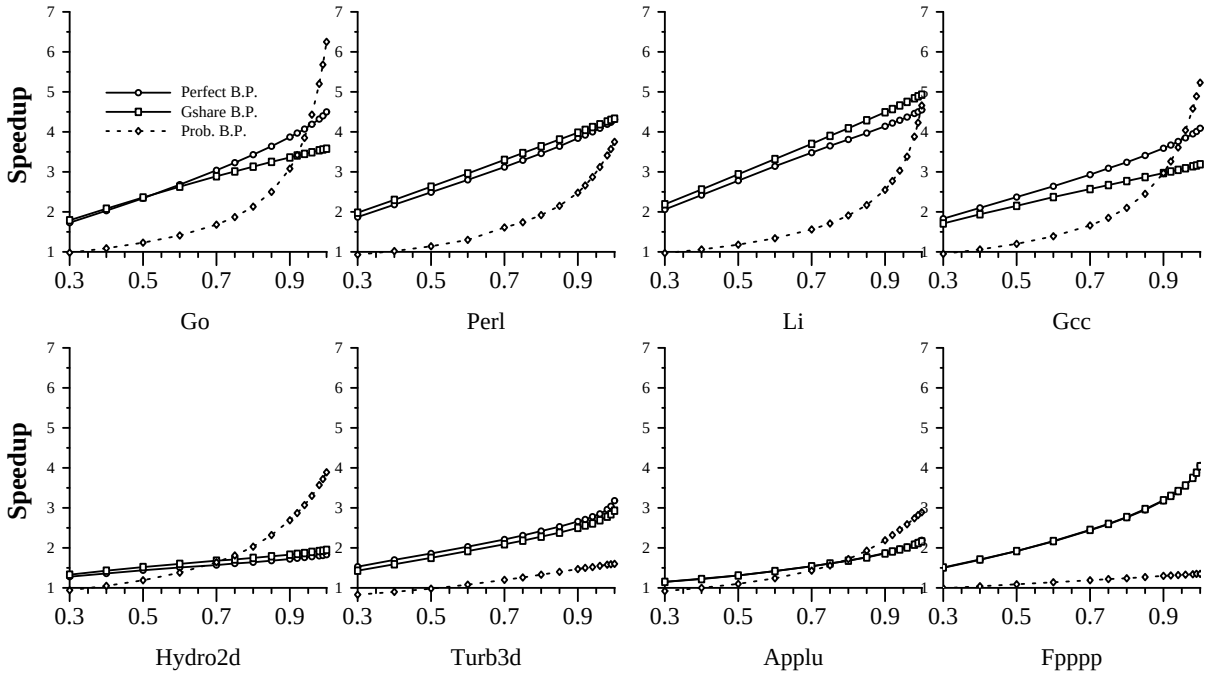

Figure 4: Speedup obtained by a probabilistic data value predictor (solid lines) and a probabilistic branch predictor (dotted lines). The window size considered is 256 entries.

The point is the shape of the curves. Looking at Figure 4, it seems clear that the data value speculation curves have a linear behavior, whereas the branch prediction curves have an exponential shape. For instance, in *go*, increasing the branch prediction accuracy from 90% to 96% makes the speedup to move from 3 to 4.45 (48% of increase on speedup). On the other hand, the same range of improvement for the value prediction accuracy moves the speedup from 3.87 to 4.19 (8% of increase). The same behavior can be observed in *perl*, *li*, *hydro2d gcc* and *applu*. However, *turb3d* and *fpppp* do not seem to get much benefit of branch prediction, and that is due to their low percentage of branches (1% for *fpppp* and 3.8% for *turb3d*).

A main conclusion drawn from Figure 4 is that, for branches, it is worthwhile building predictors that improve the accuracy just a little bit (notice that the latest branch predictors proposed in the literature have an accuracy higher than 85% for all the Spec95int benchmarks), whereas for value predictors it is worthwhile to come up with a predictor that improves the accuracy from the 50% to 90% but improving the predictor accuracy from 90% to 95% may have a little impact on the processor performance.

Comparing the results of Figure 4 and Figure 2 one may derive an estimation of the criticality of the instructions that have a stride behavior. For instance, a stride value predictor for *gcc* has an average prediction accuracy of about 40% (see Table 2). Looking at Figure 2, for a window size of 256 entries, the speedup obtained is 1.16, whereas using a probabilistic predictor with an accuracy of the 40%, the speedup is near 2 (Figure 4). *Perl* has a prediction accuracy of 64% when stride predictor is used, and the corresponding speedup is 1.5. When a probabilistic predictor of 60% of accuracy is considered, the speedup grows until 3. This trend is observed mainly for integer codes, and it is not so clear for floating point benchmarks. For instance, *hydro2d* has a prediction accuracy of 82% when a stride predictor is used and the corresponding speedup is 1.30. On the other hand, when a probabilistic predictor of 80% of accuracy is considered, the speedup achieved is 1.8 (notice that this difference is not so high as that observed for integer codes). The same happens to *turb3d* and *applu*.

Figure 5 shows the same results for a window size of 4096 entries. The graphs have the same shape as those in Figure 4. That is, the speedup of data value speculation follows a linear function whereas the speedup of branch prediction has an exponential shape. Furthermore, the increase in performance achieved when branch prediction accuracy experiences a slight improvement beyond the 90% is really significant (look for instance at *go, gcc, hydro2d* or *applu)*.

To summarize, a small improvement on the value predictor accuracy has a little impact on the processor performance, no matter in which range of accuracy the predictor is working. On the other hand, for branch predictors, a little improvement on accuracy may be reflected in a significant speedup when the prediction accuracy is quite high. The impact of a given increment in accuracy is more noticeable when the predictor accuracy approaches values close to 100%.

Finally, whereas branch prediction has received much attention during the last years, data value speculation is just a novel mechanism that needs further research. It has been shown that values can be predicted and there is a potential benefit on this prediction, but, maybe, the real potential of data value speculation is beyond boosting the processor performance by predicting individual instructions sequentially. The results of this work suggest that investigating different approaches to implement and use data value speculation may be the most rewarding research direction.

6 Conclusions

In this work we have presented a study of the potential of data value speculation to increase the ILP. This study has been carried out by considering a processor with infinite resources in terms of

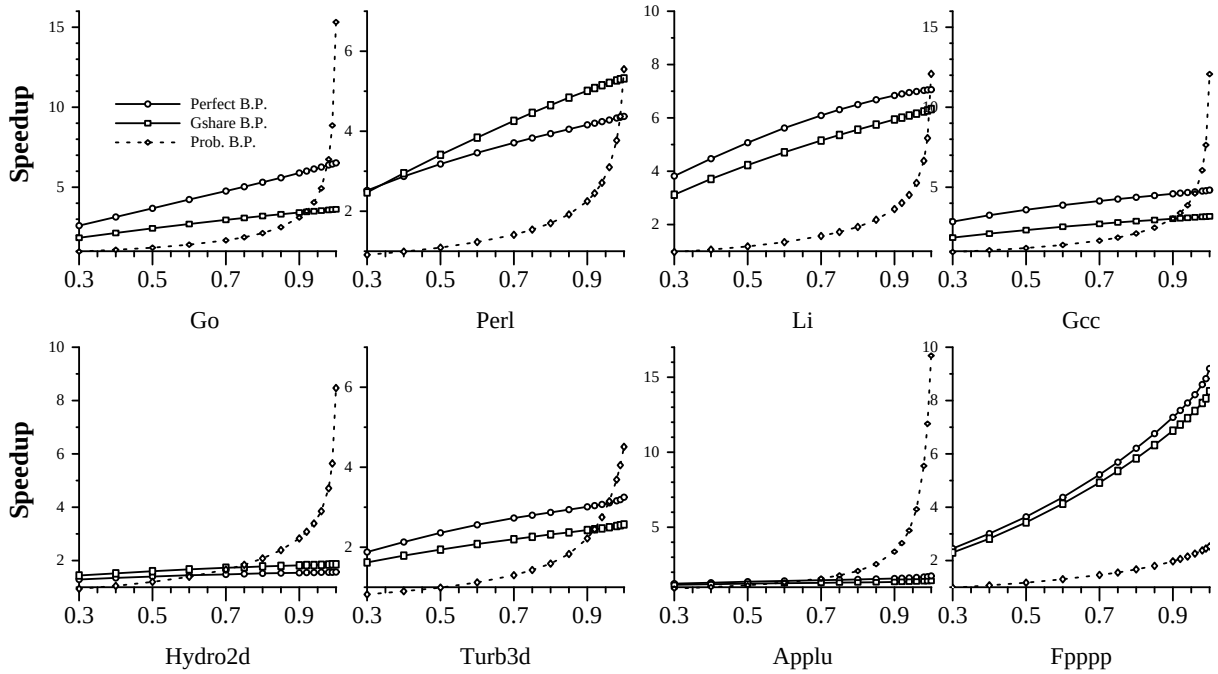

Figure 5: Speedup obtained by a probabilistic data value predictor (solid lines) and a probabilistic branch predictor (dotted lines). The window size considered is 4096 entries.

functional units, memory system, number of ports, register file, cache memory, etc. Other parameters like the instruction window size, the branch prediction accuracy or the instructions latencies have been varied to analyze their interaction with data value speculation. The main conclusions that can be drawn from this study are the following:

- Avoiding the ordering imposed by data dependences by means of data value speculation techniques has a significant potential to improve the performance of future generation superscalar processors. However, the benefits when a state-of-the-art branch predictor is also considered are relatively moderate.
- The benefit of data value speculation increases significantly as the instruction window grows and it is also dependent on the criticality of the correctly predicted instructions. For instance, it has been observed that many instructions whose result follows a stride pattern are not in the critical path for integer codes.
- A side-effect benefit of data value speculation is that the branch misprediction penalty is reduced since branch outcomes are computed earlier. For instance, it has been observed for *go* that the benefits of data speculation are higher for a realistic branch predictor than for a perfect branch predictor.
- Finally, when data value speculation is implemented in the way that it has been proposed so far for superscalar processors, it produces a benefit that can be approximated by a linear function of the prediction accuracy. This suggests that the main efforts in this area should be directed towards new implementations and/or uses of data value speculation.

Finally, we believe that much more research is still needed to better understand the data value speculation techniques and their performance potential under different scenarios.

7 Acknowledgments

This work has been supported by the Spanish Ministry of Education under grant CYCIT TIC 429/95 and the Direcció General de Recerca of the Generalitat de Catalunya under grant 1996FI-03039-APDT.

The research described in this paper has been developed using the resources of the Center of Parallelism of Barcelona (CEPBA).

8 References

[1] T.M. Austin and G. S. Sohi. “Dynamic Dependency Analysis of Ordinary Programs”. In *Proc. of Int. Symp. on Computer Architecture*, pp 342-351, 1992.

[2] M. Butler T.Y. Yeh, Y. Patt, M. Alsup, H. Scales and M. Shebanow. “Single Instruction Stream Parallelism is Greater than Two”. In *Proc. of Int. Symp. on Computer Architecture*, pp. 276-286, 1991.

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

[4] F. Gabbay and A. Mendelson. “Can Program Profiling Support Value Prediction?”. In *Proc. of Int. Symp. on Microarchitecture*, pp 270-280, 1997.

[5] J. González and A. González. “Speculative Execution via Address Prediction and Data Prefetching”. In *Proc. of the International Conference on Supercomputing*, pp 196-203, 1997.

[6] J. González and A. González. “Limits of Instruction Level Parallelism with Data Speculation”. Technical Report #UPC-DAC-1997-34, 1997.

[7] J.L Hennessy and D.A. Patterson. *Computer Architecture. A Quantitative Approach.* Second Edition. Morgan Kaufmann Publishers, San Francisco 1996.

[8] N.P. Jouppi and D.W. Wall. “Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines”. in *Proc. of the ACM Conf. on Architectural Support for Programming Languages and Operating Systems*, 1989.

[9] M.S. Lam and R.P. Wilson. “Limits on Control Flow on Parallelism”. In *Proc. of Int. Symp. on Computer Architecture*, pp 46-57, 1992

[10] M.H. Lipasti and J.P. Shen. “Exceeding the Dataflow Limit via Value Prediction”. In *Proc. of Int. Symp. on Microarchitecture*, 1996.

[11] M.H. Lipasti, C.B. Wilkerson and J.P. Shen. “Value Locality and Load Value Prediction”. In *Proc. of the ACM Conf. on Architectural Support for Programming Languages and Operating Systems*, 1996.

[12] S. McFarling. “Combining Branch Predictors”. Digital Western Research Lab Technical Note TN-36, 1993

[13] A. Moshovos and G.S. Sohi. “Streamlining Inter-operation Memory Communication via Data Dependence Prediction”. In *Proc. of Int. Symp. on Microarchitecture*, pp 235-245, 1997.

[14] Y. Sazeides, S. Vassiliadis and J.E. Smith. “The Performance Potential of Data Dependence Speculation & Collapsing”. In *Proc. of Int. Symp. on Microarchitecture*, 1996.

[15] Y. Sazeides and J.E. Smith. “The Predictability of Data Values”. In *Proc. of Int. Symp. on Microarchitecture*, pp 248-258. 1997.

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

[17] M.D. Smith, M. Johnson and M.A. Horowitz. “Limits on Multiple Instruction Issue”. In *Proc. of the ACM Conf. on Architectural Support for Programming Languages and Operating Systems*, 1989

[18] A. Srivastava and A. Eustace. “ATOM: A system for building customized program analysis tools”. In *Proc of the 1994 Conf. on Programming Languages Design and Implementation*, 1994.

[19] R.M. Tomasulo, “An Efficient Algorithm for Exploiting Multiple Arithmetic Units”, *IBM Journal of Research and Development*, 11(1), pp. 25-33, Jan. 1967.

[20] K.B. Theobald, G.R. Gao and L.J. Hendren. “On the Limits of Program Parallelism and its Smoothability”. In *Proc. of Int. Symp. on Microarchitecture*, pp 10-19, 1992.

[21] G. Tyson and T. Austin. “Improving the Accuracy and Performance of Memory Communication Through Renaming”. In *Proc. of Int. Symp. on Microarchitecture*, pp 218-227. 1997.

[22] K. Wang and M. Franklin. “Highly Accurate Data Value Prediction using Hybrid Predictors”. In *Proc. of Int. Symp. on Microarchitecture*, pp 281-290, 1997.

[23] D.W. Wall. “Limits of Instruction-Level Parallelism” Technical Report WRL 93/6 Digital Western Research Laboratory, 1993.