Focalising dynamic value prediction to CPU's context

L.N. Vintan, A. Florea and A. Gellert

Abstract: Value prediction (VP) is a relatively new technique that increases performance by eliminating true data dependency constraints. VP architectures allow data dependent instructions to issue and execute speculatively using the predicted value. This technique is built on the concept of value locality, which describes the likelihood of a previously seen value recurring within a storage location. The authors extend dynamic VP by introducing the concept of register-centric prediction instead of instruction-centric prediction. The value localities obtained on some registers of MIPS architecture were quite remarkable leading to the conclusion that VP might be successfully applied, at least on these favourable registers. The idea of attaching a value predictor for the processor's favourable registers is original and might involve new architectural techniques for improving performance and reducing the hardware cost of speculative micro-architectures. The register VP technique consists in predicting the registers' next values based on the previously seen values. It executes the subsequent data dependent instructions using the predicted values. The speculative execution will be validated when the correct values are known. If the value was correctly predicted the critical path is reduced, otherwise the instructions executed with wrong entries must be executed again. The authors examine different favourable register selections and different basic value predictors to capture certain type of value predictabilities from the SPEC benchmarks (1995 and 2000) to obtain higher prediction accuracies. Their results show that there is a time correlation between the names of the destination registers and the values stored in these registers. The simulations show that the hybrid predictor optimally exploits this correlation with an average prediction accuracy of 85.44%, which is quite remarkable (on some benchmarks the values are over 96%). Considering an eight-issue out-of-order superscalar processor it is shown that register-centric VP produces average speedups of 17.30% for the SPECint95 benchmarks, and of 13.58% for the SPECint2000 benchmarks.

1 Introduction

Value prediction (VP) is a relatively new technique built on the value locality concept, which describes the likelihood of a previously seen value's recurrence within a storage location. The main aim of this paper consists in adapting dynamic VP to the CPU context. The idea of attaching a value predictor to each CPU register (register centric predictor) instead of an instruction or memory-centric predictor [1] is original as far as we know and could involve new architectural techniques for improving performance and reducing the hardware cost of speculative micro-architectures. In our earlier work [1], we performed several experiments to evaluate the value locality exhibited by MIPS general-purpose integer registers. The results obtained on some special registers ($at, $sp, $fp, $ra) were quite remarkable ($\approx 90\%$ value locality degree) leading to the conclusion that VP might be successfully applied at least on these favourable registers.

Whether or not the prediction process has been instruction (producer) or memory-centred with great complexity and timing costs, by implementing the well known VP schemes [2, 3] centred on the CPU's registers will reduce the hardware cost. However, there are some disadvantages. Addressing the prediction tables with the instructions' destination register name (during the decode stage) instead of a program counter will cause some interference. However, we have proved that, with a sufficiently large history a hybrid predictor could eliminate this problem and achieve very high prediction accuracy (85.44% of average on eight MIPS registers using SPEC'95 benchmarks and 73.52% on 16 MIPS registers using SPEC2000 benchmarks). The main benefit of the proposed VP technique consists in unlocking the subsequent dependent instructions.

IEE Proceedings online no. 20045090

doi: 10.1049/ip-cdt:20045090

Paper received 16th July 2004

The authors are with the Computer Science Department, University 'Lucian Blaga' Sibiu, Str. E. Cioran, No. 4, Sibiu-550025, Romania

E-mail: lucian.vintan@ulbsibiu.ro

2 Related work

Lipasti *et al.* [2] first introduced the concept of value locality as the third facet of the principle of locality (temporal and spatial). They defined the value locality as 'the likelihood of the recurrence of a previously seen value within a storage location' [2] in a processor. When the 'storage location' is a single condition bit in a processor, it serves as the basis of branch prediction techniques.

Based on the dynamic correlation between load instruction addresses and the values being loaded, Lipasti and Shen [4] proposed a new data-speculative micro-architectural technique known as load VP that can effectively exploit value locality to collapse true data dependencies exceeding the dataflow limit and enhancing the instruction level parallelism, reduce average memory latency and bandwidth requirement and provide measurable performance gains. Load VP is useful only if it can be done accurately since incorrect predictions can lead to increased structural hazards and longer load latency. Starting from load dynamic behaviour and classifying these separately (unpredictable, predictable and constants), the full advantage of each case can be extracted. The cost of mispredictions can be avoided by detecting the unpredictable loads and also the cost of memory access through identifying highly predictable loads.

Relatively recent studies [5] introduced the store locality concept and store prediction methods, with good results especially for multiprocessor systems. Similarly to the load instructions approach, the store value locality was measured using its PC (instruction-centric) or its accessed data address (memory-centric). In both cases the value locality degree is between 30% and 70%. It introduces the ‘silent store’ concept, meaning that a store writes the same value as its previous instance (34–68% of the dynamic store instructions are silent stores). So, removing these store instructions at some points in the program’s execution (either statically at compile time, or dynamically at run time) some potential benefit can be gained in execution time and/or code size [5]. Also, the pressure on cache write ports, the pressure on store queues and the data bus traffic outside the processor chip are reduced. The free silent store squashing concept is based on the idle read port stealing to perform store verification and an aggressive load/store queue to exploit temporal and spatial locality for store squashing [6].

Sazeides [3] has developed an empirical classification of value sequences produced by instructions. There are two kinds of value predictability in programs: value repetition and value computability. To capture these types of value predictability, the authors proposed two distinct main categories of predictors: computational and contextual. Two important characteristics were also defined for understanding prediction behaviour. One is the learning time (LT), which is the number of values that have to be observed before the first correct prediction. The second is the learning degree (LD), which is the percentage of correct predictions following the first correct prediction.

Computational predictors are predicting based on some previous values in an algorithmic manner, therefore according to a deterministic recurrence formula. An incremental predictor belongs to the computational class. Lipasti and Shen introduced the stride predictor [4] and Sazeides [3] generalised the idea. A stride sequence is a value sequence in which the later value can be computed by the immediate previous value and a stride. Last VPs were used for the first time in [2] to predict load values, and in some subsequent work the VP process was extended to other types of instruction (add, shift, store) [1, 4]. The simulation results indicated that the computational prediction varies between instruction types, indicating that its performance can be further improved if the prediction function matches the functionality of the predicted instruction.

The contextual predictor predicts the next value based on a particular stored pattern (context) that is repetitively generated in the value sequence. Theoretically it can predict any repetitive sequences. A context predictor is of order k if its context information includes the last k values, and therefore, the search is done using this pattern of k values. In this case the prediction process is based on a simple Markov model [7]. The results of laborious simulation on SPEC benchmarks showed that a single predictor cannot capture all the types of predictability patterns that occur in programs. This suggests that a hybrid scheme might be useful for enabling high prediction accuracy at lower cost [2, 8]. Although the hybrid value predictors can provide more correct predictions than single predictors, they consume more hardware resources. More importantly, they can waste the limited hardware resources available since every instruction being predicted occupies a unique entry in each of the component predictors.

Rychlik *et al.* [9] combined a last, a stride, and a two-level VP into an overall hybrid VP. To efficiently utilise the hardware resources, they provided the dynamic classification scheme to dynamically distribute instructions into proper component predictors during run time [9]. Although this dynamic classification scheme uses the hardware resources more efficiently, it cannot provide a higher prediction accuracy than the hybrid VP. Wang *et al.* [10] modified the dynamic classification scheme by reclassifying instructions that cannot be predicted well by their previously assigned component predictor. Their modification improved this kind of hybrid VP.

Calder *et al.* [11] proposed some techniques that give priority for prediction to those instructions that belong to the longest data dependence chains to reduce the pressure on the limited VP resources (such as the limited table size and limited read/write ports). For this reason, partial data dependence graphs for instructions are constructed in the processor’s active instruction window during run time. Also, to detect the time consuming instructions, the compiler’s help is required together with some profiling information.

Tullsen and Seng [12] proposed a technique called register VP that identifies instructions that produce values that are already in the register file. Therefore, the corresponding results are predicted using the values belonging to the register file. This technique mainly uses the previous value in the instruction’s destination register as a prediction for the new result, in a static or dynamic manner. This technique produced speedups of up to 11% for the SPECint95 benchmarks and up to 13% for SPECfp95 benchmarks. In contrast to our work, this approach is instruction-centric, instead of register-centric, as our approach is. Tullsen and Seng [12] pointed out that in their prediction approach ‘confidence counters are associated with instructions rather than registers’. As an alternative, our original register VP technique consists in predicting the register’s next value based on the previously seen values. To implement this strategy we attach a VP to all the CPU’s favourable registers (having a high value locality degree). After the instruction’s decode, in order to predict the instruction’s destination the corresponding register-VP is activated. Based on this approach, we systematically developed some context predictors.

Gabbay and Mendelson [13] developed a so-called register-file predictor that is the closest predecessor to our register VP technique. They predicted the destination value of a given instruction according to the last previously seen value and the stride of its destination register. Unfortunately they did not further pursue this particular idea by systematically developing new register-centric predictors and evaluating them through simulations.

3 Register VPs

The main goal of this Section is to show the current stage in instruction-centric VP development and to adapt these schemes for register VP. Whereas in the producer-centric prediction case the structures are addressed during the instructions’ fetch stage using the PC in the register-centric prediction case the tables are indexed only in the second part of the instructions’ decode stage.

3.1 The ‘last value’ predictors

The ‘last value predictors’ (see Fig. 1) predict the new value as the same as the last value stored in the corresponding register. Exploiting the correlation between register names and the values stored in those registers, will decrease the instructions' latencies.

Each entry in the prediction table has its own automata, which is incremented when the prediction is correct and it is decremented otherwise. Obviously, the utility of VP techniques is emphasised only in the case of a correct prediction otherwise it determines structural hazards and a higher instruction execution latency. Based on the dynamic behaviour of the register content we developed the classification: unpredictable and predictable registers. By treating each group of registers separately the misprediction costs can be avoided. It is necessary to verify the value generated by the value history table (VHT). The automata's state will be changed according to the comparison between the generated value and the predicted value.

The VHT generates the predicted values and the 'state' field could implement the hysteresis mechanism using an automata. Each register used in the prediction mechanism has an entry in the VHT. In this way the number of entries in the prediction table is the same as the number of logical registers.

3.2 Stride predictors

In this case, considering that v_{n-1} and v_{n-2} are the most recent values, the new value v_n will be calculated using the recurrence formula: $v_n = v_{n-1} + (v_{n-1} - v_{n-2})$, where $(v_{n-1} - v_{n-2})$ is the stride of the sequence. The stride could be variable in time and not simply a constant. This technique reduces the number of mispredictions from two to only one in the case of repetitive stride sequences. Figure 2 shows the structure of this predictor.

When a register is used as destination for the first time, there is a miss in the prediction table and no prediction is made. When an instruction stores a value in the destination register that value is stored also in the VHT's '*Val*' field and the automata will be in the initial unpredictable state. If that register is used again as a destination register, no prediction is made but the stride $Str_1 = V_1 - Val$ is calculated, and V_1 and Str_1 are introduced in the '*Val*' and 'Str_1' fields. If it is used again as a destination register no prediction is made, but the stride $Str_2 = V_2 - Val$ is again calculated. The value from the field Str_1 is introduced in the field Str_2. Also V_2 and Str_2 are introduced in the fields *Val* and Str_1 respectively. If the register is used again as an instruction's destination and $Str_1 = Str_2$, the predicted value adding the stride Str_2 to the value stored in the VHT (*Val*) is calculated. If the automata is in the predictable state, the prediction is generated. The automata is incremented if the prediction is correct, otherwise it is decremented.

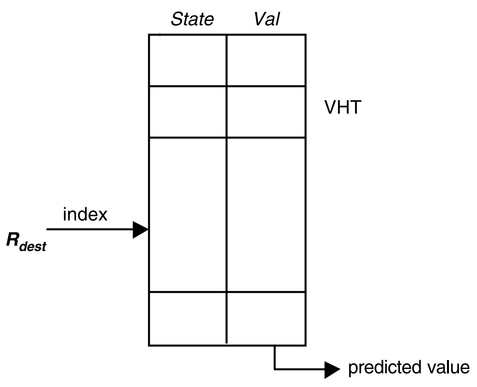


Fig. 1 *'Last value' predictor*

3.3 Context-based predictors

The context-based predictors predict the value that will be stored in a register based on the last values stored in that register. A context is a finite sequence of values with repeated appearance as in a Markov chain.

The predictors that implement the prediction by partial matching (PPM) algorithm [3, 14] represent an important class of context-based predictors. This predictor contains a set of simple Markov predictors as can be seen in Fig. 3.

The predicted value is the value that followed the context with the highest frequency. As can be observed in Fig. 3 the predicted value depends on the context. A longer context frequently drives to a higher prediction accuracy but sometimes it can behave as noise. In the considered sample

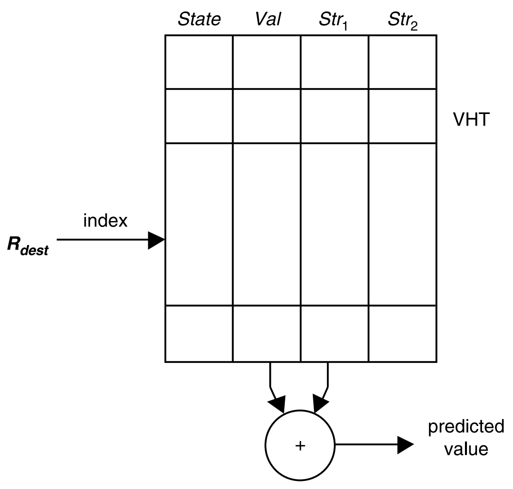


Fig. 2 *Stride predictor*

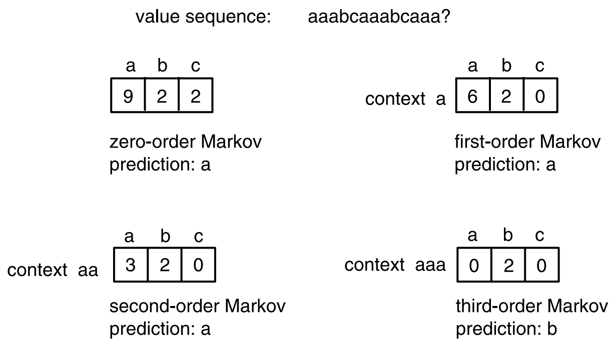

value sequence: aaabcaaabcaaa?
a b c
9 2 2
zero-order Markov prediction: a
context a
a b c
6 2 0
first-order Markov prediction: a
context aa
a b c
3 2 0
second-order Markov prediction: a
context aaa
a b c
0 2 0
third-order Markov prediction: b

Fig. 3 *PPM predictor*

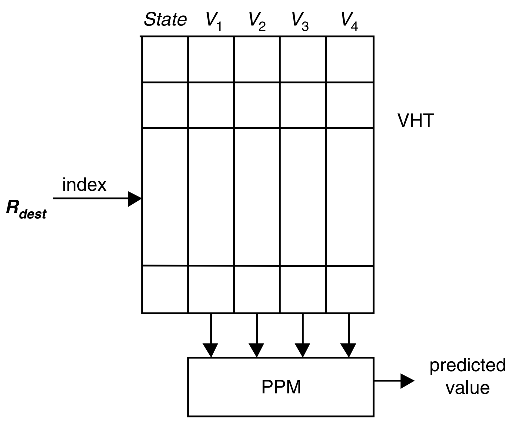


Fig. 4 *Structure of a context-based predictor*

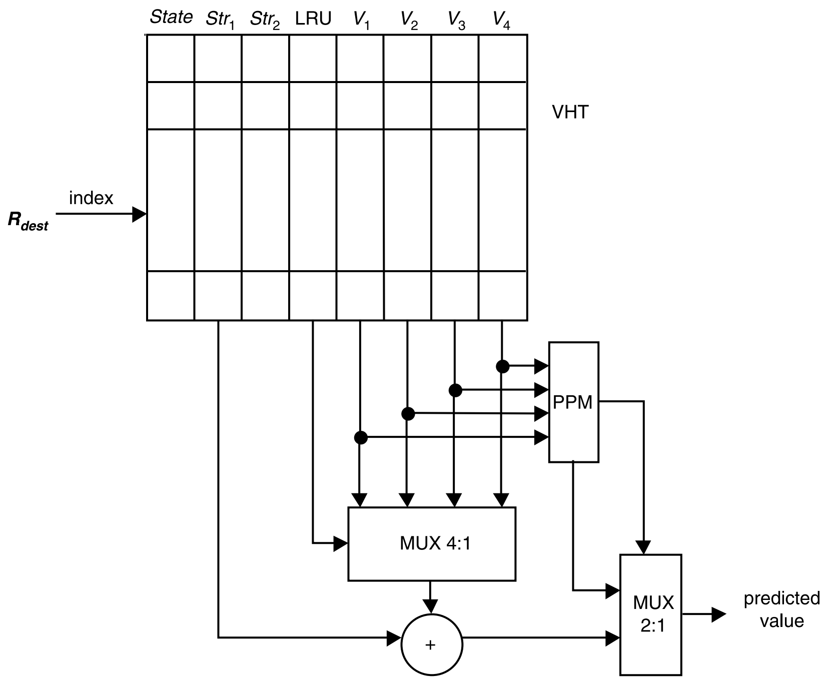


Fig. 5 *Hybrid predictor (contextual and stride)*

only the third-order Markov predictor makes a correct prediction. A complete PPM predictor contains N simple Markov predictors, from 0th order to $(N-1)$th order. If the $(N-1)$th Markov predictor produces a prediction (the context is matched in the sequence) the process is finished, otherwise the $(N-2)$th order Markov predictor will be activated, and so on.

Figure 4 shows the structure of the context-based predictor. Each entry from the VHT has an associated automaton that is incremented when the prediction is correct and is decremented in the case of a misprediction. The fields $V_1, V_2, \ldots, V_4$ store the last four values associated with each register (considering that the predictor works with a history of four values). If the automaton is in the predictable state, it predicts the value that follows the context with the highest frequency (Fig. 3).

3.4 Hybrid predictors

It has been shown that a single type of predictor does not offer the best results. Some types of value sequences generated in programs are better predicted with a certain predictor, and others, with another type of predictor [8]. Therefore, it is natural to consider the idea of hybrid prediction: two or more VPs working together dynamically in the prediction process. Figure 5 shows a hybrid predictor composed of a context-based predictor and by a stride predictor. The context-based predictor always has priority. In this way the value generated by the stride predictor is only used if the context-based predictor cannot generate a prediction. This fixed prioritisation seems not to be optimal. Probably a dynamic prioritisation based on some confidences would be better (the predictor having the highest confidence degree will have priority).

4 Simulation methodology and experimental results

We developed a cycle-accurate execution-driven simulator derived from the sim-outorder simulator in the SimpleScalar tool set [15]. The baseline superscalar processor supports out-of-order instruction issue and execution. We modified it to incorporate the registers' VPs proposed in Section 3. Table 1 shows the configuration of the baseline processor used to obtain the results.

To perform our evaluation, we collected results from different versions of SPEC benchmarks: five integer (*li, go, perl, ijpeg, compress*) and three floating point (*swim, hydro, wave5*) SPEC'95 benchmarks. We simulated seven benchmarks (*gzip, b2zip, parser, crafty, gcc, twolf* and *mcf*) from the CINT SPEC2000 set. We also simulated some SPEC'95 benchmarks to compare their behaviour with that of SPEC2000. In other words, we intend to discover how these different benchmarks influence the VP's micro-architectural features.

The number of instructions fast forwarded through before starting our simulations is 400 million. We used the `-fastfwd` option in SimpleScalar/PISA 3.0 to skip over the initial part of execution in order to concentrate on the main body of the programs. Results are reported for simulating each program for 500 million committed instructions.

Statistical results based on simulation have proved that commonly used programs are characterised by strong value repetitions [2, 16]. The main causes for this phenomenon are: data and code redundancy, program constants, and the compiler routines that resolve virtual function calls, memory aliases etc. The register value locality is frequently met in programs and shows the number of times each register is written with a value that was previously seen in the same register and dividing by the total number of dynamic instructions having this register as its destination field [1, 17].

In Figs. 6 and 7 we calculated the value locality metric using the formula

$$VL_j(R_k) = \frac{\sum_{i=1}^{n} VL_j^k(i)}{\sum_{i=1}^{n} VRef^k(i)} \qquad (1)$$

where n = number of benchmarks (8 for SPEC'95, 7 for SPEC2000), j = history length (4, 8, 16, 32), k = register's number, $VL_j^k(i)$ = number of times when register R_k is written with a value that was previously seen in last j values

Table 1: Machine configuration for baseline architecture

Processor core	fetch/decode/issue width	8 instruction/cycle
	reorder buffer size	128 entries
	load-store queue	64 entries
	integer ALUs	8 units, 1-cycle latency
	integer multiply/divide	4 units, 3/12-cycle latency
Branch predictor	hybrid branch predictor	*gshare* with 16K entries, 14 bit history, *bimodal* with 16K entries
	branch and value misprediction	7-cycle latency
	memory access	60-cycles latency
	memory width	32 bytes
Caches	level-one data cache	4-way set associative, 64 KB, 1-cycle hit latency
	level-one instruction cache	direct mapped, 128 KB, 1-cycle hit latency
	level-two cache (unified)	4-way set associative, 1024 KB, 10-cycle hit latency

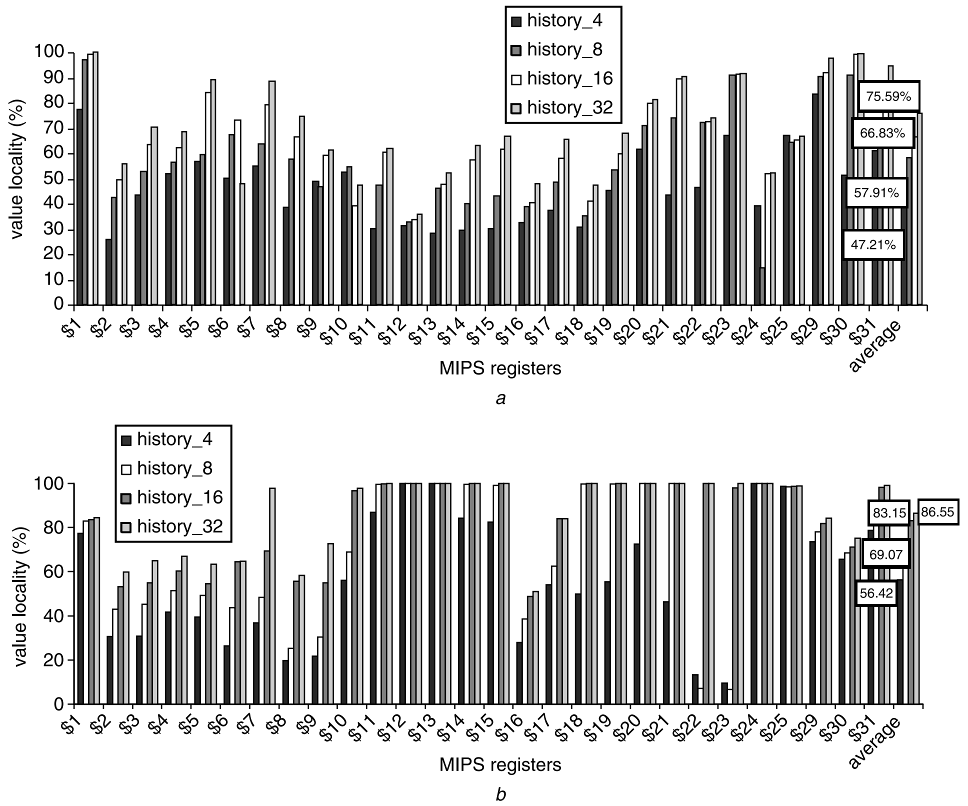


Fig. 6 *Value locality on registers*
a SPEC'95 simulation results
b SPEC2000 simulation results

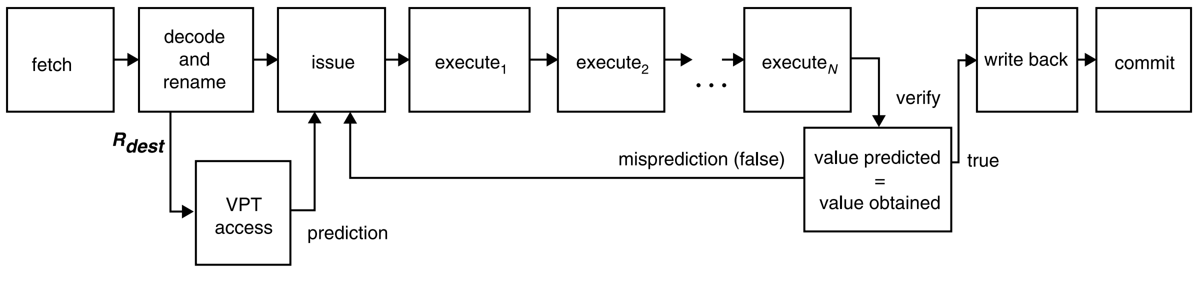


Fig. 7 *Implementation of register VP mechanism in pipeline structure of a general micro-architecture*

of the same register (on benchmark i); and $VRef^k(i)$ = the total number of dynamic instructions that have register R_k as their destination field (on benchmark i).

Figures 6 and 7 emphasise the concept of value locality on registers. As can be observed (Fig. 6), the value locality on some registers is remarkable high (90%), and this predictability naturally leads us to the idea of implementing VP on these favourable registers.

The dynamic VP on registers represents a new technique that allows the speculative execution of the read after write dependent instructions by predicting the values of the destination registers during second half of the instruction's decode stage (see Fig. 7). The VP technique predicts the next register's value based on the last values belonging to that register. It executes the operations using the predicted value. The speculative execution will then be validated when the correct value is known (after the execution stage). If the value was correctly predicted the critical path might be reduced. Otherwise, the instructions executed with wrong entries must be executed again (recovery).

In this work we developed and simulated several different basic value predictors [17], such as the last value predictor, the stride value predictor, the context-based predictor and a hybrid value predictor to capture certain type of value predictabilities from SPEC benchmarks and to obtain higher prediction accuracy. All these predictors were adapted to our proposed prediction model.

After the instruction's decode stage, the VPT table is accessed with the name of the destination register. In the case of a valid prediction, the VPT will generate the predicted value to the subsequent corresponding RAW dependent instructions. After execution, when the real value is known, it is compared with the predicted value. In the case of a misprediction the speculatively executed dependent instructions are re-issued for execution.

Starting with a minimal superscalar architecture, we studied how the simulator's performance will be affected by the variation of its parameters. We now present the results obtained with a register value predictor. Each register has associated four-state confidence automata. A prediction is made only if the automata is in one of the two predictable states. In Figs. 8a and 8b, respectively, each bar represents the average of register VP accuracy obtained for eight SPEC'95 benchmarks and for seven integer SPEC2000 benchmarks, respectively.

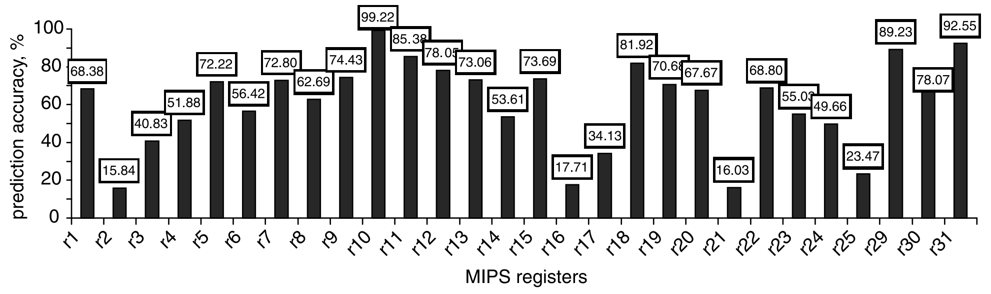
In Figs. 6 and 7 we calculated the prediction accuracy (PA) metric using the following formula:

$$PA(R_k) = \frac{\sum_{i=1}^{n} CPV^k(i)}{\sum_{i=1}^{n} VRef^k(i)} \qquad (2)$$

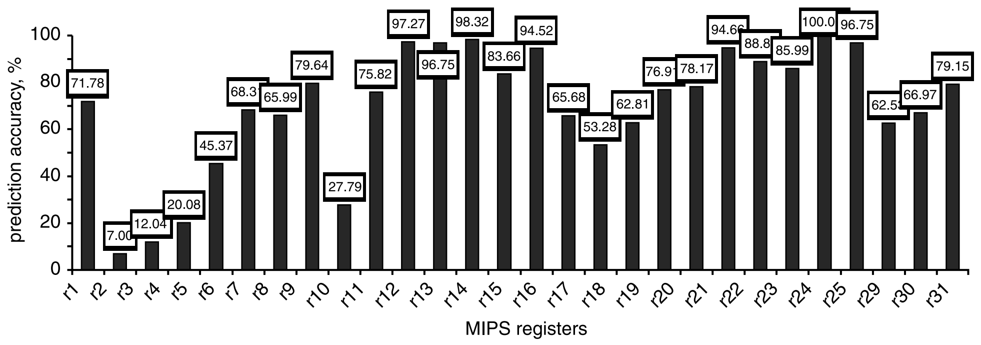
where n = number of benchmarks (8 for SPEC'95 and 7 for SPEC2000), k = register number, $CPV^k(i)$ = number of correctly predicted values of register R_k (on benchmark i), and $VRef^k(i)$ = the total number of dynamic instructions that have register R_k as their destination (on benchmark i).

In the next investigations, we are focusing only on the predictable registers which have a prediction accuracy higher than a certain threshold (60% and 80%, respectively), measured using the hybrid predictor on the SPEC benchmarks (see Fig. 8). As can be seen in Figs. 8a and 8b the registers having a prediction accuracy higher than 60% are: 1, 5, 7–13, 15, 18–20, 22 and 29–31 for SPEC'95, and, 1, 6–8, 10–16, 18–25 and 29–31 for SPEC2000. The statistic results on SPEC'95 benchmarks exhibit a using degree of 19.36% for these 17 registers. This means that 19.36% of instructions use one of these registers as a destination. The equivalent average result on SPEC2000 is 13.24% using 22 general purpose registers.

In Fig. 9 we compared the previously presented VP techniques: last VP (Fig. 1), stride prediction (Fig. 2), context-based prediction (Fig. 4) and hybrid prediction



a



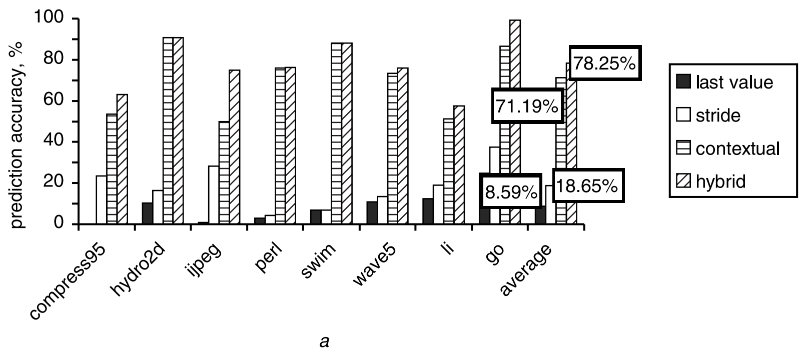
b

Fig. 8 *Register VP using a hybrid predictor (context based, stride)*

History of 256 values

a Pattern has four values (SPEC'95 simulation results)

b Pattern has four values (SPEC2000 simulation results)



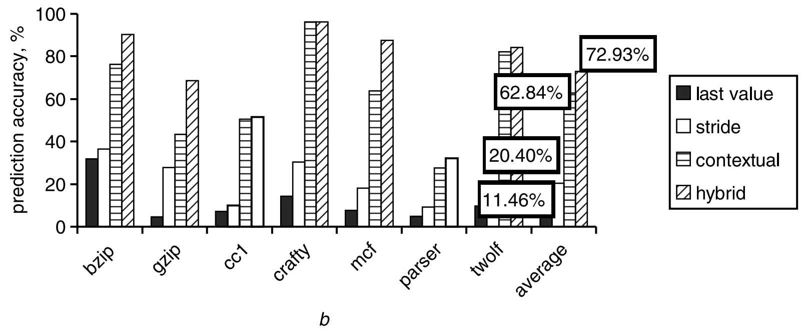


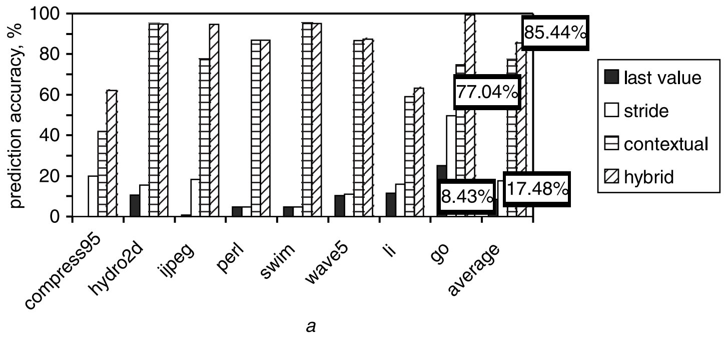
Fig. 9 *Prediction accuracy using favourable registers*

a 17 favourable registers (SPEC'95 benchmarks)
b 22 favourable (SPEC2000 benchmarks)

(Fig. 5). We used in the prediction process only the 17 favourable registers on SPEC'95 benchmarks and 22 favourable registers on the SPEC2000 benchmarks. The context-based and the hybrid predictors use a history of 256 values and a search pattern of 4 values.

These results (see Fig. 9) represent the global prediction accuracies of the favourable registers for each benchmark. Hybrid predictor synergy can be observed. It involves an average prediction accuracy of 78.25% on the SPEC'95 benchmarks and 72.93% on the SPEC2000 benchmarks.

Now we will try a more elitist selection considering only the registers with a prediction accuracy higher than 80% (see Figs. 10*a*, *b*). The selection is again based on Figs. 8*a* and *b*. We can observe that there are 8 registers that fulfill this condition: 1, 10–12, 18, 29–31 on SPEC'95 benchmarks and respectively 16 registers: 1, 8, 11–15, 20–25, 29–31 on SPEC2000 (registers 1, 29–31 are included even if they don't fulfill this condition because they exhibit a high degree of value locality according to Fig. 6*b*; they also have special functions). The global using rate of these registers is 10.58%



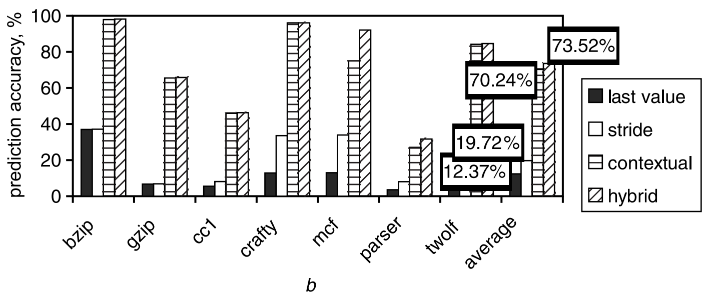
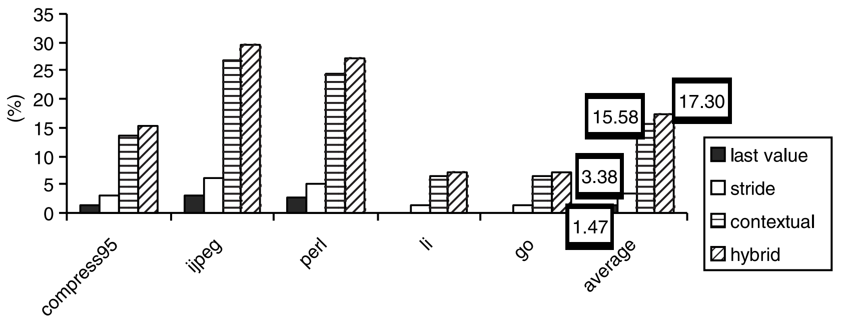
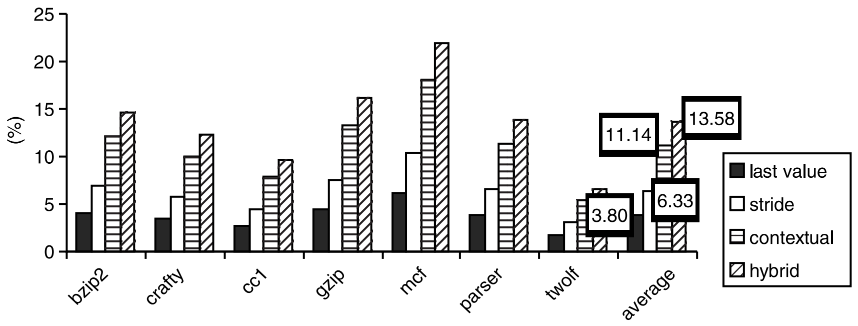


Fig. 10 *Prediction accuracy using favourable registers*

a 8 favourable registers (SPEC'95)
b 16 favourable registers (SPEC2000)



a



b

Fig. 11 *Speedup over baseline machine*
a 8 favourable registers (SPEC'95)
b 16 favourable registers (SPEC2000)

on SPEC'95 benchmarks, and 9.01% on SPEC2000 benchmarks.

Figure 10 emphasises, for each benchmark, the global prediction accuracy obtained with the implemented predictors using 8 and 16 selected registers (threshold over 80%, according to the previous explanations). Each bar represents the register VP for a certain benchmark, measured by counting the number of times when prediction is accurate for any of the favourable registers and dividing by the total number when these registers are written. The simulation results offered by the last value predictor are relatively close to the stride predictor's results. The best prediction average prediction accuracy was obtained with the hybrid predictor was 85.44%, which was quite remarkable (on some benchmarks values over 96% were obtained).

Figure 11 shows the speedup obtained compared to the baseline processor when using each of the register VPs described in Section 3.

5 Conclusions

We have considered the register VP concept. As we have discussed the intention of the register VP concept is to reduce the unfavourable effect of the RAW dependencies, by reducing the wait times of the subsequent dependent instructions. Also, the prediction focused on registers instead of on instructions. This is advantageous because many fewer predictors are needed, thus saving complexity and costs. We proposed to exploit the value locality on registers using different prediction techniques. We used the hybrid predictor presented in Fig. 5 to select the favourable registers. We continued after that with the evaluation of the predictors using registers with a prediction accuracy higher than 60%. The best results were obtained with the hybrid predictor: an average prediction accuracy of 78.25% and a utilisation rate of 19.36%. We then tried a more elitist selection of the registers and we continued the evaluation of the predictors using only the registers with prediction accuracy higher than 80%. The best results were obtained with the hybrid predictor: an average prediction accuracy of 85.44% (on some benchmarks values over 96% were obtained) and a utilisation rate of 10.58%. Also, considering an 8-issue out-of-order superscalar processor simulations shows that utilisation register centric VPs produce average speedups of 17.30% for the SPECint95 benchmarks and 13.58% for the SPECint2000 benchmarks.

We demonstrated that there is a dynamic correlation between the names of the destination registers and the values stored in these registers. The simulations show that the hybrid predictor best exploits this correlation and the value locality concept.

As part of our future work, we intend to improve the implemented VPs. Because the hybrid predictor presented in this work uses a static prioritisation, we want to develop some meta-predictors with a dynamic selection mechanism based on confidences and respectively on neural network (simple perceptrons).

6 References

1 Florea, A., Vintan, L., and Sima, D.: 'Understanding value prediction through complex simulations'. Proc. 5th Int. Conf. on Technical Informatics, Timisoara, Romania, Oct. 2002
2 Lipasti, M.H., Wilkerson, C.B., and Shen, J.P.: 'Value locality and load value prediction'. Proc. 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, October 1996, pp. 138–147
3 Sazeides, Y.: 'An analysis of value predictability and its application to a superscalar processor'. PhD thesis, University of Wisconsin-Madison, 1999.
4 Lipasti, M.H., and Shen, J.P.: 'Exceeding the dataflow limit via value prediction'. Proc. 29th Annual ACM/IEEE Int. Symp. on Microarchitecture, Dec. 1996
5 Lepak, K.M., and Lipasti, M.H.: 'On the value locality of store instructions'. Proc. 27th Annual Int. Symp. on Computer Architecture, Vancouver, Canada, June 2000

6 Lepak, K.M., and Lipasti, M.H.: 'Silent stores for free'. Proc. 33rd Annual ACM/IEEE Int. Symp. on Microarchitecture (MICRO33), California, USA, 2000
7 Rabiner, R.L.: 'A tutorial on hidden markov models and selected applications in speech recognition', *Proc. IEEE*, 1989, **77**, (2)
8 Wang, K., and Franklin, M.: 'Highly accurate data value prediction using hybrid predictors'. Proc. 30th Annual ACM/IEEE Int. Symp. on Microarchitecture, Dec. 1997
9 Rychlik, B., Faistl, J., Krug, B., Kurland, A., Jung, J., Velev, M., and Shen, J.: 'Efficient and accurate value prediction using dynamic classification' (Carnegie-Mellon University, 1998)
10 Wang, Y., Lee, S., and Yew, P.: 'Decoupling value prediction on trace processors'. Proc. 6th Int. Symp. on High Performance Computer Architecture, 1999
11 Calder, B., Reinman, G., and Tullsen, D.: 'Selective value prediction'. Proc. 26th Int. Symp. on Computer Architecture, May 1999, pp. 64–74
12 Tullsen, D.M., and Seng, J.S.: 'Storageless value prediction using prior register values'. Proc. 26th Int. Symp. on Computer Architecture, May 1999
13 Gabbay, F., and Mendelsohn, A.: 'Using value prediction to increase the power of speculative execution hardware', *ACM Trans. Comput. Syst.*, 1998, **16**, (3)
14 Joseph, D., and Grunwald, D.: 'Prefetching using Markov predictors'. Proc. 24th Int. Symp. on Computer Architecture, June 1997, pp. 252–263
15 Simplescalar: 'The SimpleSim tool set', ftp://ftp.cs.wisc.edu/pub/sohi/Code/simplescalar
16 Sodani, A.: 'Dynamic instruction reuse'. PhD thesis, University of Wisconsin-Madison, 2000
17 Gellert, A.: 'Contributions to speculative execution of instructions by dynamic register value prediction'. MSc thesis, University 'Lucian Blaga', Sibiu, 2003 (in Romanian)