Register Value Prediction using Metapredictors

Lucian N. Vintan, Arpad Gellert and Adrian Florea

"Lucian Blaga" University of Sibiu, Computer Science Department, Str. E. Cioran, No. 4, Sibiu-550025, ROMANIA, Tel./Fax: +40-269-212716, E-mail lucian.vintan@ulbsibiu.ro, arpad.gellert@ulbsibiu.ro, adrian.florea@ulbsibiu.ro

Abstract? **Value Prediction (VP) is a relatively new technique that increases performance by eliminating true data dependencies constraints. Value prediction architectures allow data dependent instructions to issue and execute speculatively using the predicted values. This technique is built on the concept of value locality, which describes the likelihood of a previously seen value's recurrence within a storage location.**

This paper extends the dynamic value prediction by exploiting 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 conclusion that value prediction might be successfully applied, at least on these favorable registers. The idea of attaching a value predictor for the processor's favorable registers might involve new architectural techniques for improving performance and reducing the hardware cost of speculative microarchitectures. The register value prediction technique consists in predicting registers' next values based on the previously seen values. It executes the subsequent data dependent instructions using the predicted values and 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.

In our previous work we implemented a hybrid predictor composed by a context-based predictor and respectively a stride predictor, working together. The context-based predictor had always priority; in this way the value generated by the stride predictor was used only if the context-based predictor cannot generate a prediction. Obviously this fixed prioritisation is not optimal. In this paper our goal is to increase the prediction accuracy using a dynamic prioritisation based on some confidences. We introduce several different metaprediction structures, in order to properly select the current best predictor: two non-adaptive metapredictors and an adaptive one, represented by a neural network. The experimental results obtained using metaprediction applied only to the best four favorable registers (having high value locality degrees) show an average prediction accuracy of 91.40%, measured on SPEC benchmarks. The accuracy gain obtained on these registers versus the old hybrid predictor is 2.27%.

Index Terms? **metapredictors, dynamic value predictors, register value prediction, neural networks.**

I. Introduction

Our proposed register value prediction (RVP) focuses dynamic value prediction to CPU's context. It allows dependent instructions speculative execution by predicting the values of their corresponding destination registers. The RVP technique predicts the values of registers based on the last values stored in those registers; it executes the operations using the predicted values and the speculative execution will be validated when the correct values are known, after the complete instruction's execution. If the value was correctly predicted the critical path might be reduced, otherwise the instructions executed with wrong entries must be executed again (recovery).

Whether until now the prediction process was instruction (producer) or memory centered with great complexity and timing costs, by implementing the well-known value prediction schemes [8], [12] centered on CPU's registers will mainly decrease the hardware cost. However, there are some disadvantages. Addressing the prediction tables with instructions' destination register name (during the decode stage) instead of Program Counter will cause some interferences and delays. However, we proved that with a sufficiently large history a hybrid predictor could eliminate this problem and achieve very high prediction accuracy**.** The main benefit of the proposed VP technique consists in unlocking the subsequent dependent instructions.

In our previous works we developed and simulated several different basic value predictors, such as the *last value predictor*, the *stride value predictor*, the *context-based predictor* and a *hybrid value predictor*. The *last value predictors* predict the new value as the same with the last value stored in the corresponding register. The *stride value predictors* identify stride sequences and predict them correspondingly. The *context-based predictors* predict the next value based on a particular stored pattern that was repetitively generated in the value sequence. An important class of contextual based predictors implement the "*Prediction by Partial Matching*" algorithm (PPM), based on a set of Markov predictors. We also implemented a hybrid predictor composed by a context-based predictor and respectively a stride predictor, working together. The context-based predictor had always priority; in this way the value generated by the stride predictor was used only if the

context-based predictor cannot generate a prediction. Obviously this fixed prioritization is not optimal; this will be solved in this paper through our proposed metapredictors.

Our goal is to improve the presented hybrid predictors, in order to obtain better prediction accuracy. We introduce several different metapredictors, used for a dynamic selection of the current best predictor: two non-adaptive (static) metapredictors and an adaptive (dynamic) neural metapredictor.

The organization of the rest of this paper is as follows. In section II we review related work in the field of value prediction. Section III presents the metaprediction concept and describes the implemented predictors. Section IV includes simulation methodology and experimental results obtained using a simulator that we developed. Finally, section V suggests directions for future works and concludes the paper.

II. Related Work

Lipasti and Shen [8] 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" [8] in a processor. When the "storage location" is a single condition bit into a processor, it serves as the basis of branch prediction techniques. Statistical results based on laborious simulations have proved that common-used programs are characterized by strong value repetitions. Value prediction techniques exploit value locality to collapse true data dependencies exceeding the dataflow limit.

Based on the dynamic correlation between load instruction addresses and the values being loaded, Lipasti [7] proposed a new data-speculative micro-architectural technique entitled *load value prediction* 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.

Relatively recent studies [5] introduced the Store locality concept and Store prediction methods, with good results especially for multiprocessor systems. It is introduced the “silent Store” concept, meaning that a Store writes the same value like 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, there are reduced: the pressure on cache write ports, the pressure on store queues and the data bus traffic outside the processor chip. The free silent store squashing concept is based on idle read port stealing to perform store verifies and aggressive load/store queue to exploit temporal and spatial locality for store squashing [6].

In [12] Sazeides developed an empirical classification of value sequences produced by instructions. There are two kinds of value predictability existing in programs: value repetition and value computability. In order to capture these certain types of value predictability, the authors have been proposed two distinct main categories of predictors: computational and contextual. Two important characteristics were also defined for understanding prediction behavior. 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 in [7] and Sazeides [12] generalized 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 value predictors were used for the first time in [8] to predict load values. In some subsequent work the value prediction process was extended to other types of instruction (add, shift, store) [7], [2]. The simulation results indicated that the performance of computational prediction varies between instructions 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 they 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 length. In fact, in this case the prediction process is based on a simple Markov model [10]. The results of laborious simulation on SPEC benchmarks pointed out that a single predictor cannot capture all the various 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 [8], [17]. 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 [11] combined a last, a stride, and a two-level value predictor to an overall hybrid value predictor. In order to efficiently utilize the hardware resources, they provided the dynamic classification scheme to dynamically distribute instructions into proper component predictors during run-time [11]. Although this dynamic classification scheme uses the hardware resources more efficiently, it cannot provide higher prediction accuracy than the hybrid value predictor. Lee and Yew [18] modified the dynamic classification scheme by reclassifying instructions after they cannot be predicted well by their previously assigned

component predictor. Their modification improved this kind of hybrid value predictor.

Calder [1] proposed some techniques that give priority for prediction to those instructions that belong to the longest data dependence chains in order to reduce the pressure on the limited value prediction resources (such as the limited table size and limited read/write ports). For this reason, it is constructed partial data dependence graphs for instructions in the processor's active instruction window during run-time. Also, for detecting the time consuming instructions it is required the compiler's help together with some profiling information.

Tullsen and Seng [15] proposed a technique entitled register-value prediction 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. Mainly, this technique uses the previous value in the instruction's destination register as a prediction for the new result, in a statically or dynamically manner. In contrast to our work, this approach is instruction-centric, like all developed papers in this research area, instead of register-centric, as our approach is. According to this, the authors pointed out that in their prediction approach "confidence counters are associated with instructions rather than registers". As an alternative, our original register value prediction technique consists in predicting register's next value based on the previously seen values. In order to implement this strategy we attach a value predictor for all the CPU's favorable registers (having a high value locality degree). After instruction's decode, in order to predict instruction's destination the corresponding register value predictor is activated. Based on this approach, we developed in a systematical manner some context predictors

Gabbay and Mendelson [3] developed a so called register-file predictor that is the closest predecessor to our register value prediction technique. They predict the destination value of a given instruction according to the last previously seen value and the stride of its destination register. Unfortunately the authors did not pursue further this particular idea by systematically developing new register-centric predictors and evaluating them through simulations.

III. META PREDICTION

A hybrid instruction value predictor combines two or more component predictors that each predicts the value of the current instruction destination. The hybrid predictor therefore needs a selection mechanism to predict which of the predictors is likely to be most accurate at a certain moment. This prediction of prediction accuracy is called in literature *metaprediction.*

In this paper we used the following component predictors, all of them centered on CPU's registers instead on program's instructions: a *last value* predictor, a *stride* predictor and a *context-based* predictor. Every component predictor provides two values: the predicted value and its confidence. A confidence mechanism performs speculation control by limiting the prediction to those that are likely to be correct. A high confidence represents the continuous correct predictability in a given history of that register.

The metapredictor represents a second prediction level and it selects dynamically, based on their last behaviors (confidence), one of the predictors (last value, stride or contextual) in order to predict the next value for a certain destination register. The architecture used for metaprediction is presented in Fig. 2.

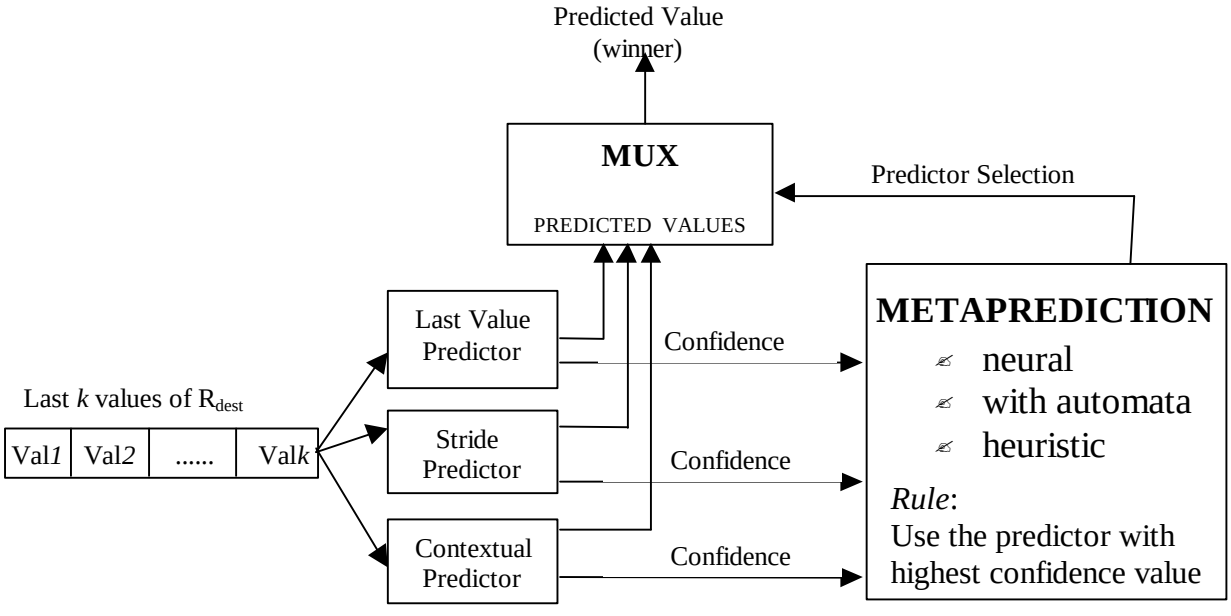

Fig. 2. The architecture used for metaprediction

A. *Heuristic Non-Adaptive Metapredictors*

The first non-adaptive metapredictor (we called it "heuristic") selects one of the predictors based on their last *k* behaviors (correct/incorrect prediction). A logical one ('1') means a correct prediction and a logical zero ('0') means a miss-prediction. These behaviors are stored in a logic left shift register associated with each predictor (Fig. 3).

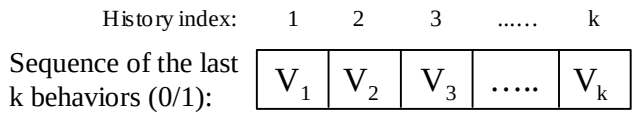

Fig. 3. Logic left shift register associated with each predictor

The predictor with the highest number of favorable behaviors (number of logical ones) has priority. The predicted value is used only if the selected predictor encountered a number of favorable behaviors higher than a certain threshold. In other words a prediction is generated only in the case of an acceptable confidence.

B. *Non-Adaptive Metapredictors With Automata*

The other one non-adaptive metapredictor uses three 4-states confidence automata – saturating counters: one for the context-based predictor, one for the stride predictor and another one for the last value predictor. The automata initially are in the *unpredictable* state and they are adjusted after each prediction, when the real values are known. The predictor with the highest confidence has priority and a prediction is generated only in the corresponding two predictable states. The structure of the 4-states automata is presented in Fig. 4.

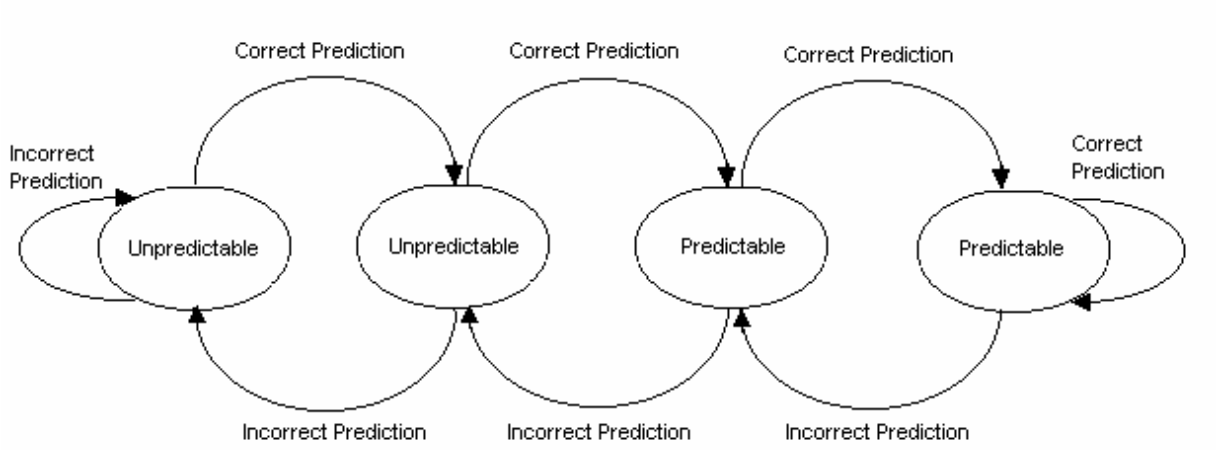

Fig. 4. The structure of the 4-state automata

C. *Neural Adaptive Metapredictors*

Another idea consists in implementing a dynamic metapredictor represented by a simple neural network (NN) in order to select the best predictor at a certain moment. It is well known that a great advantage of the artificia l neural networks is their capacity to learn based on examples (supervised learning). The network extracts the information from the training samples. In this way it is able to synthesize implicitly a certain model of the problem. In other words, the neural network builds up alone an algorithm to solve the problem. We firstly implemented as a metapredictor a multi-layer perceptron with one hidden layer and back-propagation learning algorithm just for understanding the idea's potential performance. The input vector consists of one *k*-bit confidence sequence for each predictor (N=3k), representing the last *k* binary codified behaviors: 1 for a correct prediction and 0 for misprediction. The neural network returns through its output layer the selected predictor, using one neuron for each predictor (P=3). It is selected the predictor corresponding to the neuron with the highest output value, but a prediction is generated only if this value is greater than a certain threshold. In our previous works [16] we proved based on experiments that the optimal number of hidden layer neurons is N+1, N being the number of input layer neurons. In Fig. 5 is presented the structure of the adaptive neural metapredictor with one hidden layer.

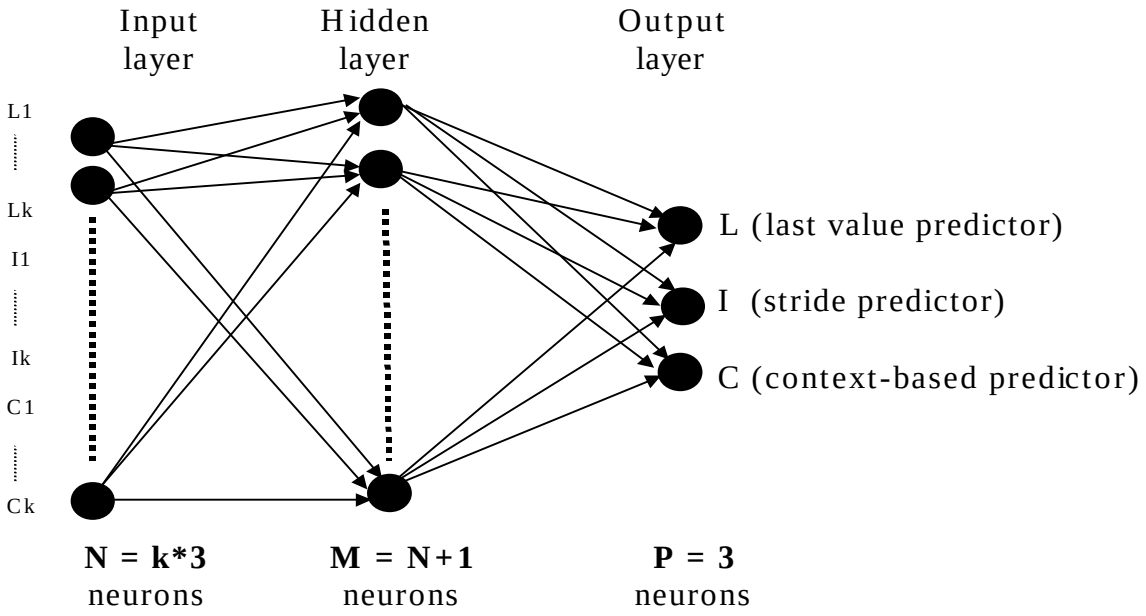

Fig. 5. The structure of the neural metapredictor

For the training/learning process we used the well-known *back-propagation* algorithm [9], adapted as below:

1. Create a feed-forward network with N inputs, $M = N + 1$ hidden units and P output units.

2. Initialize all network weights $W^1_{i,j}$; $i = \overline{1, N}$; $j = \overline{1, M}$ and $W^2_{i,j}$; $i = \overline{1, M}$; $j = \overline{1, P}$, to small random numbers belonging to the [0.3, 0.7] interval.

In the following t_k represents the value of k's neuron from the output layer and O_k is the desired value of the same neuron.

3. Until $E(\overline{W}) = \frac{1}{2} \sum_{k \in Outputs(P)} (t_k - O_k)^2 < T$ (*threshold*), *do*:

3.1. Input the instance $\overline{X}$ to the network and compute the output $\overline{O}$ (matrix product).

$$\overline{O} = \overline{X} \cdot \overline{W^1} \cdot \overline{W^2} \quad (1)$$

3.2. For each network output unit k, $k = \overline{1, P}$, calculate its error term δ_k.

$$\delta_k = O_k (1 - O_k)(t_k - O_k) \quad (2)$$

3.3. For each hidden unit h, $h = \overline{1, M}$, calculate its error term δ_h

$$\delta_h = O_h (1 - O_h) \sum_{k \in Outputs(P)} W^2_{k,h} \delta_k \quad (3)$$

3.4. Update each network weight $W_{i,j}$

$$W_{i,j} = W_{i,j} + \Delta W_{i,j} \quad (4)$$

$$\Delta W_{i,j} = \eta \delta_i X_{i,j} \quad (5)$$

where η is the learning step.

We used the following activation function:

$$F(x) = \frac{1}{1 + e^{-x}} \quad (6)$$

IV. Experimental Results

We developed an execution driven simulator derived from *Simple Scalar* tool set [13]. It was simulated the execution of 5 million dynamic instructions from different SPEC'95 benchmarks. Statistical results based on simulation have proved that common-used programs are characterized by strong value repetitions [8], [14]. 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 exhibits the number of time 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 destination field [2], [4]. For the Value Locality metric we used the following formula:

$$VL_j(R_k) = \frac{\sum_{i=1}^{n} VL^k_j(i)}{\sum_{i=1}^{n} V\,\mathrm{Re}\,f^k(i)} \quad (7)$$

n = number of SPEC'95 simulated benchmarks;
j = history length (1, 4, 8 respectively 16);
k = register's number;

$VRef^{k}(i)$ = number of dynamic instructions having register R_k as destination (on benchmark *i*);

$VL_j^k(i)$ = number of times when register R_k is written with a value that was previously seen in last j values of the same register (on benchmark *i*).

The next figure gives emphasis to the concept of value locality on MIPS registers. As it 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 value prediction implemented on these favorable registers. In the next investigations, we are focusing only on the predictable registers, having value locality higher than a certain threshold (70%).

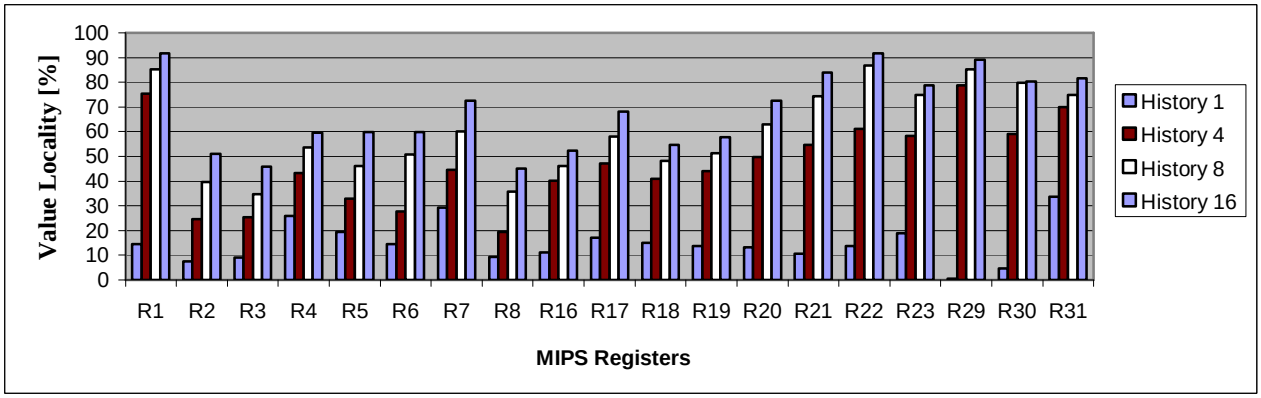

Fig. 6. Value locality on registers

In this work we developed and evaluated several different metapredictors (presented in paragraph III) in order to capture certain type of value predictabilities from SPEC'95 benchmarks and to obtain higher prediction accuracy. The prediction accuracy represents the number of correctly predicted registers divided to the total number of dynamic instructions having these registers as destination. The confidence metric further introduced represents the number of correctly predicted values related to a certain register (Ri) when the attached predictor was into a predictable state divided to the total number of predictable states associated to that register (Ri).

Starting with a minimal superscalar architecture, we studied how will be affected the simulator's performance by the variation of its parameters. We began evaluating the heuristic non-adaptive metapredictors (presented in Fig. 3) which select one of the predictors based on their last k behaviors stored in a logic left shift register associated with each predictor. The predictor with the highest number of favorable behaviors (number of logical ones) has priority. The predicted value is used only if the selected predictor encountered a number of favorable behaviors at least equal with a certain threshold. Figures 7 and 8 show how the threshold's value affects the prediction accuracy and respectively the confidence, using a heuristic metapredictor with k=3.

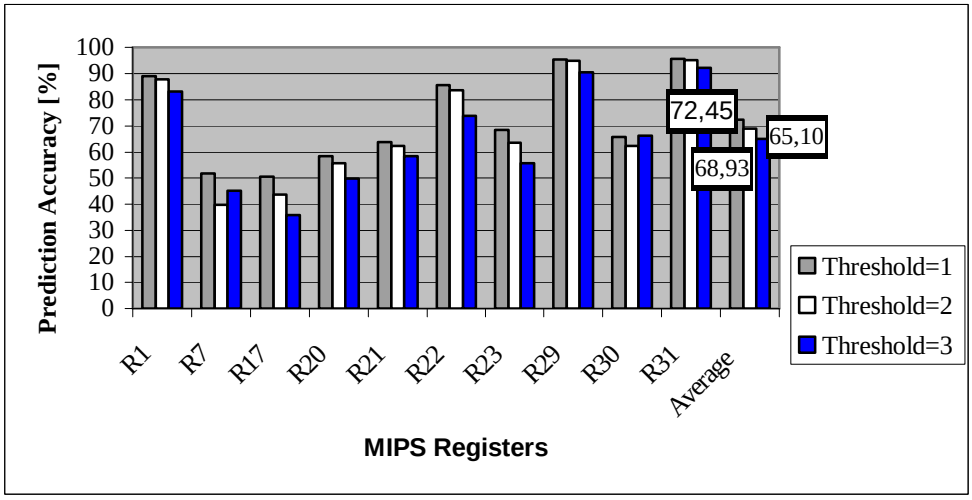

Fig. 7. Non-adaptive Metapredictor: prediction accuracy for different thresholds (k=3)

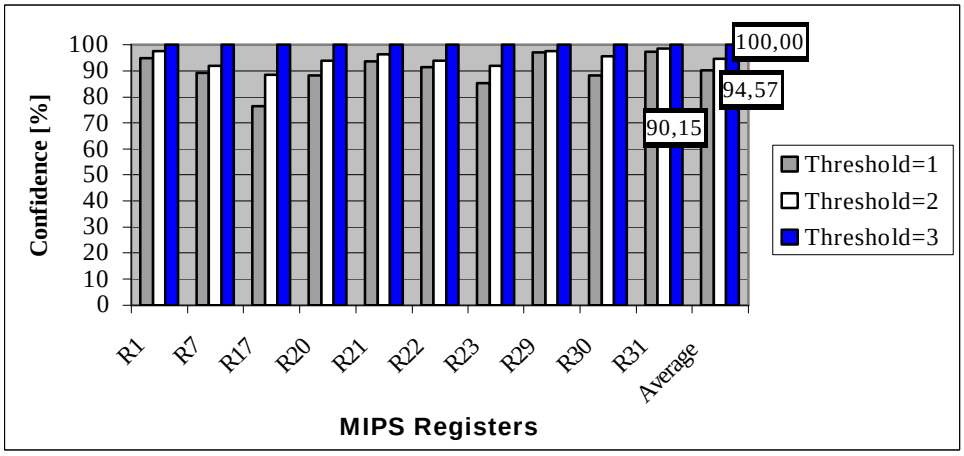

Fig. 8. The non-adaptive Metapredictor: confidence for different thresholds (k=3)

We continued our study evaluating the non-adaptive metapredictors with confidence automata (presented in Fig. 4). Each predictor has associated a 4-state automata and the predictor with the highest confidence has priority. A prediction is generated only in the predictable strong states. Figures 9 and 10 show how the threshold's value affects the prediction accuracy and respectively the confidence of these predictors.

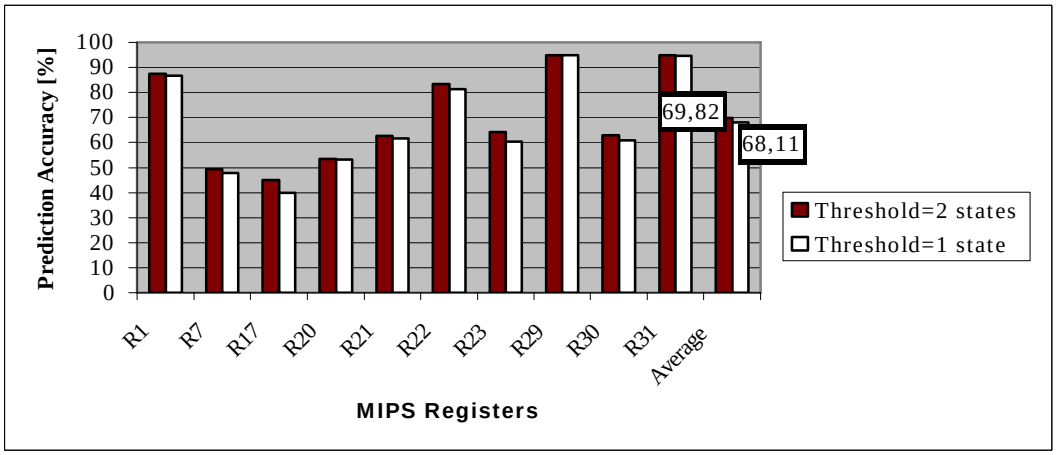

Fig. 9. The prediction accuracy measured for different thresholds (prediction in 1 or 2 strong states) using a metapredictor with automata

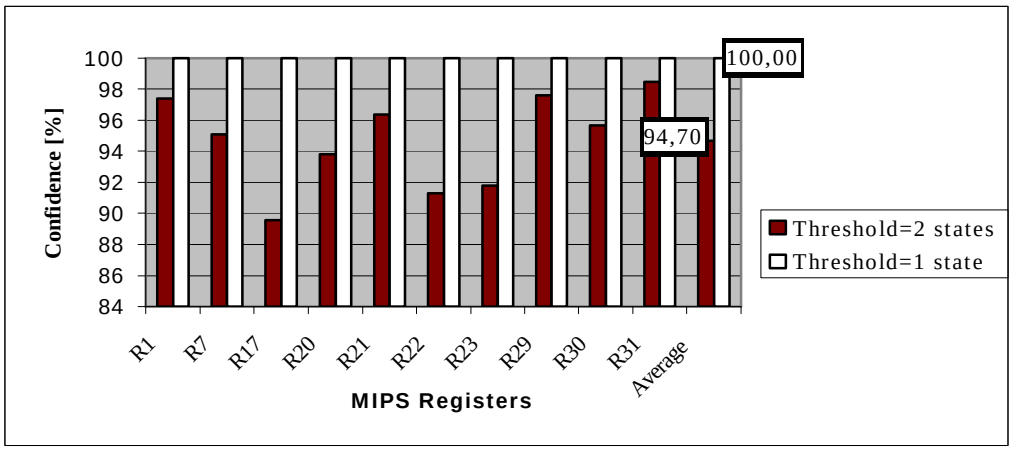

Fig. 10. The confidence measured for different thresholds (prediction in 1 or 2 strong states) using a metapredictor with automata

We continued our evaluations with the adaptive neural metapredictor (Fig. 5), with an input vector consisting of one *k*-bit sequence for each predictor (N=3k), representing the last *k* binary codified behaviors: 1 for a correct prediction and 0 for misprediction. The neural network returns through its output layer the selected predictor, using one neuron for each predictor (P=3). It is selected the predictor corresponding to the neuron with the highest output value, but a prediction is generated only if this value is greater than a certain threshold. Figures 11 and 12 show how the threshold's value affects the prediction accuracy and respectively the confidenc e of the neural network:

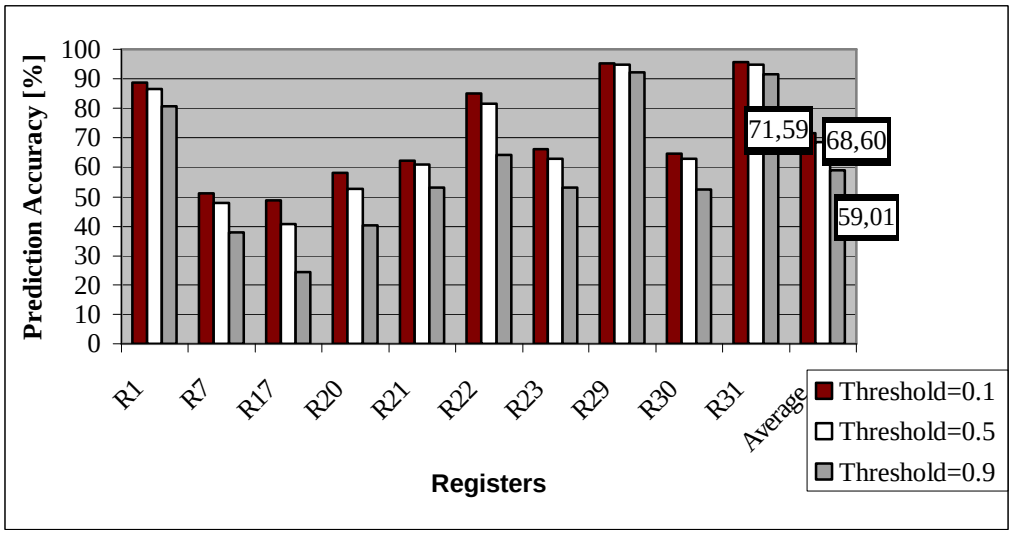

Fig. 11. The neural metapredictor's prediction accuracy for different thresholds (k=3)

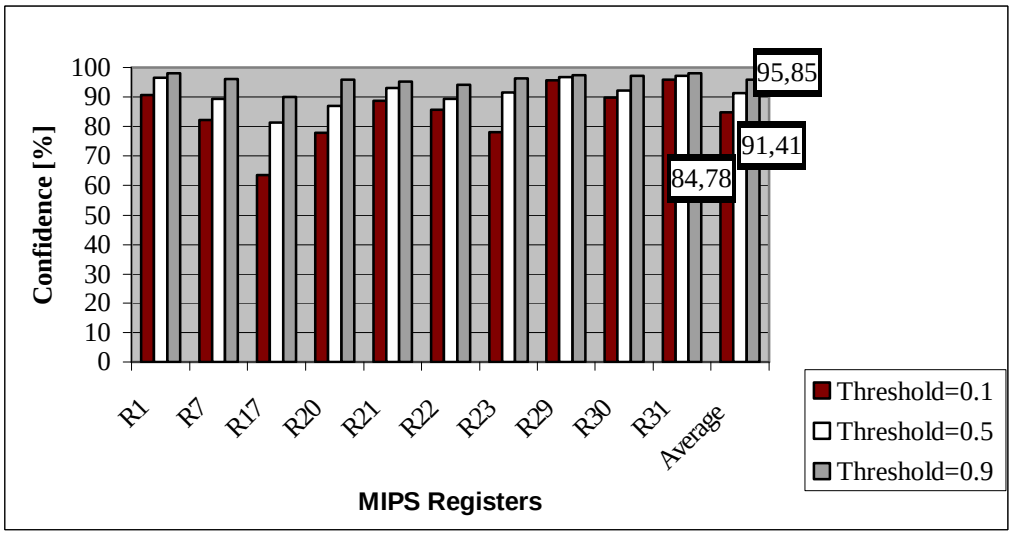

Fig. 12. The neural metapredictor's confidence for different thresholds (k=3)

It can be observed that each metapredictor, for a high threshold value, has less global prediction accuracy but a higher confidence. Practically, the probability that local prediction generated by high confidence states to be correct significantly increases thro ugh reducing the cases when the structure makes a prediction. The disadvantage is that the percentage of cases in which is made a prediction dramatically decreases.

The next parameter we varied is the history length (k). Fig. 13 shows how is affected the prediction accuracy and respectively the confidence of a neural network by the behavior history length, using a threshold of 0.1. It can be observed that the optimal value of k is 3.

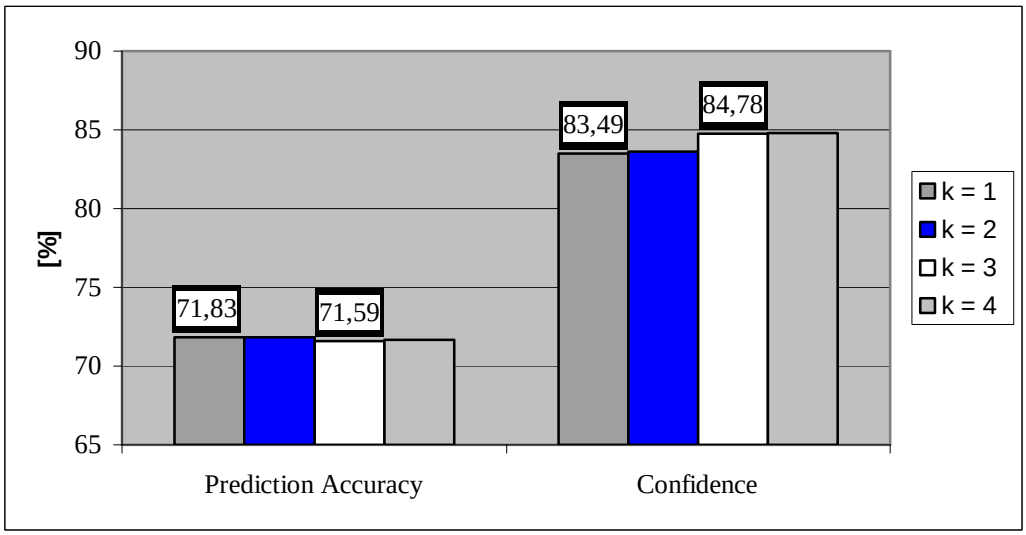

Fig. 13. Study of prediction accuracy and confidence varying the behavior history length for a threshold of 0.1

Taking into account of all previous experimental results the next table exhibits an interesting conclusion: in spite of pretty small global prediction accuracy (72.45% in average on all favorable registers) there are some registers (R1, R22, R29 and R31) with prediction accuracies greater than 90%. The accuracy gain obtained on these registers versus our previous hybrid predictor [2], [4] is 2.27%.

Register Value Prediction Accuracy

Register number	**Metapredictor with automata (threshold=1 state)**	**The non-adaptive (heuristic) metaprediction: (threshold = 1)**	**The neural metapredictor (threshold=0)**	**Hybrid predictor**
1	89.034	89.028	88.999	86.81
22	85.477	85.647	84.146	81.43
29	95.289	95.287	95.284	94.74
31	95.642	95.643	95.599	94.48
Average	**91.3605**	**91.40125**	**91.007**	**89.365**
Accuracy gain *obtained vis a vis of hybrid predictor* **[%]**	2.23	**2.27**	1.83	

TABLE I. The prediction accuracy obtained on the best four favorable registers

V. CONCLUSIONS

In this paper we proposed to improve the register value prediction's accuracy by metaprediction. We introduced several different metapredictors, in order to properly select the current best predictor: two static metapredictors based on predictors' confidences, and a dynamic one, represented by a neural network. One of the static metapredictors uses three 4-state confidence automata: one for the context-based predictor, one for the stride predictor and another one for the last value predictor. The predictor with the highest confidence has priority and a prediction is generated only in the predictable strong states. Using this hybrid predictor we obtained an average global prediction accuracy of 69.81% with a confidence of 94.7%. Predicting only in the strongest state of the automata we obtained an average global prediction accuracy of 68.1% with a confidence of 100%.

The other static metapredictor selects one of the predictors based on their last *k* behaviors (correct/incorrect prediction). These behaviors are stored in a logic left shift register associated with each predictor. The predictor with the highest number of favorable behaviors (number of logical ones) has priority. As an example, with a selection

based on the last three behaviors of each predictor (k=3), and generating prediction only if the number of favorable predictions is at least 2 (threshold=2), we obtained a global prediction accuracy of 68.93% with a confidence of 94.57%. For a threshold of 1 (predict only in the strongest state) the global prediction accuracy is 65.09% with a confidence of 100%.

Another idea consists in implementing a dynamic metaprediction structure represented by a simple neural network (NN) in order to select the best predictor at a certain moment. We implemented as metapredictor a multi-layer perceptron with one hidden layer and back-propagation learning algorithm. The input vector consists of one k-bit sequence for each predictor, representing the last k binary codified behaviors: 1 for a correct prediction and 0 for misprediction. The neural network returns through its output layer the selected predictor, using one neuron for each predictor. It is selected the predictor corresponding to the neuron with the highest output value, but a prediction is generated only if this value is greater than a certain threshold. With a neural selection, based on the last three behaviors (k=3) and a threshold value of 0.1 we obtained an average global prediction accuracy of 71.58% with a confidence of 84.77%. For a threshold of 0.9 the global prediction accuracy is 59.01% with a confidence of 95.85%.

Judging from the confidence point of view it can be observed that the non-adaptive metapredictor with automata and prediction in the strongest state is the optimal. Using this metapredictor we obtained a global prediction accuracy of 68.11% with a confidence of 100%. A further challenge will be to implement in hardware an efficient neural dynamic metapredictor.

REFERENCES

[1] Calder B., Reinman G. and Tullsen D. *Selective Value Prediction*, Proceedings of the 26th International Symposium on Computer Architecture, pages 64-74, May 1999.

[2] Florea A., Vintan L., Sima D. *Understanding Value Prediction through Complex Simulations*, proceedings of the 5th International Conference on Technical Informatics, University "Politehnica" of Timisoara, Romania, October, 2002.

[3] Gabbay F., Mendelsohn A. *Using Value Prediction To Increase The Power Of Speculative Execution Hardware*, ACM Transactions On Computer Systems, vol 16, nr. 3, 1998.

[4] Gellert A. *Contributions to speculative execution of instructions by dynamic register value prediction*, MSc Thesis, University "Lucian Blaga" of Sibiu, Computer Science Department, 2003 (in Romanian).

[5] Lepak K. M., Lipasti M. H. *On the Value Locality of Store Instructions*, Proceedings of the 27th Annual International Symposium on Computer Architecture, Vancouver, June 2000.

[6] Lepak K. M., Lipasti M. H. *Silent Stores for Free*, Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture (MICRO33), California, USA, 2000.

[7] Lipasti M. H., Shen J. P. *Exceeding the Dataflow Limit via Value Prediction*, Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture, December 1996.

[8] Lipasti, M. H., Wilkerson, C. B., Shen, J. P. *Value Locality and Load Value Prediction*, Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 138-147, October 1996.

[9] Mitchell T., *Machine Learning*, McGraw-Hill, 1997.

[10] Rabiner R. L. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, Vol. 77, No. 2, February 1989.

[11] Rychlik B., Faistl J., Krug B., Kurland A., Jung J., Velev M. and J. Shen. *Efficient and Accurate Value Prediction Using Dynamic Classification*, Technical Report, Department of Electrical and Computer Engineering, Carnegie Mellon Univ., 1998.

[12] Sazeides Y. *An analysis of value predictability and its application to a superscalar processor*, PhD Thesis, University of Wisconsin-Madison, 1999.

[13] Simplescalar, *The SimpleSim Tool Set*, ftp://ftp.cs.wisc.edu/pub/sohi/Code/simplescalar

[14] Sodani A. *Dynamic Instruction Reuse*, PhD Thesis, University of Wisconsin – Madison, USA, 2000.

[15] Tullsen D. M., Seng J. S. *Storageless Value Prediction using Prior Register Values*, Proceedings of the 26th International Symposium on Computer Architecture, May 1999.

[16] Vintan L., *Towards a High Performance Neural Branch Predictor*, International Joint Conference on Neural Networks, Washington DC, USA, July 1999.

[17] Wang K., Franklin M. *Highly Accurate Data Value Prediction using Hybrid Predictors*, Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, December 1997.

[18] Wang Y., Lee S., and Yew P. *Decoupling Value Prediction on Trace Processors*, Proceedings of the 6th International Symposium on High performance Computer Architecture, 1999.