ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **22**, Number 3–4, 2019, 215–227

Perceptron-Based Selective Load Value Prediction in a Multicore Architecture

Arpad GELLERT , Maria VINŢAN , and Lucian VINŢAN

Computer Science and Electrical Engineering Department, Lucian Blaga University of Sibiu
Emil Cioran Str., No. 4, 550025 Sibiu, Romania
E-mails: arpad.gellert@ulbsibiu.ro, maria.vintan@ulbsibiu.ro, lucian.vintan@ulbsibiu.ro

Abstract. In this work we have extended the load value prediction concept, previously implemented by us in a multicore architecture, with perceptron-based classification of load instructions into the predictable and unpredictable categories. The prediction scheme has been integrated into the Sniper multicore simulator. The goal of the load value predictor is to anticipate the values of critical load instructions and to unlock in a speculative manner the subsequent dependent instructions' processing. Since high prediction accuracies are necessary, we use the predicted value only if the corresponding load instruction is classified as being predictable. The evaluations performed on the Splash-2 parallel benchmarks have shown an average relative speedup of 4.21% over the baseline multicore architecture, with a maximum of about 17%.

Key-words: Multicore architecture; issue bottleneck; selective load value prediction; perceptron; speculative execution; Splash-2 parallel benchmarks; Sniper multicore simulator.

1. Introduction

In the last period, the research in the computer architecture domain has been focused on multicore systems which exploit thread-level parallelism, frequently neglecting the instruction-level parallelism, investigated for a long period especially within superscalar microarchitectures. In this work, we have improved the selective load value predictor for a state-of-the-art multicore architecture, developed previously in [8], by enhancing it with perceptron-based classification of dynamic load instructions into the predictable and unpredictable classes.

Perceptrons have been successfully applied in [21, 10, 11] and [12] for efficient dynamic branch prediction within two-level adaptive schemes that are using fast per branch single-cell perceptrons instead of two-bit saturating counters. The branch address is hashed to select the corresponding perceptron's weights, which are used to generate a prediction based on the global

/ local branch history. The perceptron, one of the simplest neural networks, is a natural choice for branch prediction because it can be efficiently implemented in hardware.

In this work, we adapted the perceptron-based branch prediction scheme presented in [10] and [11] for load value prediction, in order to ameliorate the so-called issue bottleneck (data-flow bottleneck). For each load instruction stored in the *Load Value Prediction Table* (LVPT), the corresponding perceptron's weights are stored, too. Actually, the perceptron's hardware structure, representing a binary adder (Wallace-Tree of 3 to 2 Carry-Save, having a logarithmic depth), is global. Based on the previous behaviors of a certain load instruction, the corresponding perceptron classifier (a unique hardware structure together with the load instruction's corresponding weights) determines if that load instruction is predictable or not. Only if a load instruction is predictable, the predicted value is used to speculatively unlock the subsequent dependent instructions' processing. Thus, if the prediction is correct, the dependent instructions are earlier executed, which contributes to a global speedup of the multicore system. On the other hand, if the prediction is wrong, a recovery process is necessary, in order to re-establish the correct architectural state of the corresponding core (thread). Therefore, high prediction accuracies are necessary, which can be obtained with efficient predictors applied only on the load instructions which are dynamically classified as predictable. Thus, with good prediction accuracies on a significant number of dynamic loads, this technique can provide a speedup. If the speedup is sufficiently high, it can also reduce the energy consumption, as we already have shown in the related papers [6, 7] and [8].

The rest of the paper is organized as follows. Section 2 presents the recent related work. Section 3 presents the perceptron-based selective load value prediction mechanism. Section 4 presents the simulation methodology. Section 5 discusses the experimental results and, finally, Section 6 presents the conclusions and the further work suggestions.

2. Related Work

In 2004 there were published about 70 papers, especially at the most prestigious computer architecture conferences, focused on value prediction techniques implemented in single threaded architectures. We have also applied value prediction methods in superscalar-, multithreaded- and multicore-microarchitectures, in several previous works like [5, 6, 7, 8] and [9]. Value prediction focused on the logical registers of superscalar architectures, as a low-power alternative to the instruction-centric value prediction paradigm, have been successfully applied by us in [22, 23] and [24]. The Sniper state-of-the-art multicore simulator and the Splash-2 parallel benchmark suite have been successfully used for microarchitectural evaluations in [4] and [8]. Further, we will limit our presentation only to some of the most recent valuable papers focused on this issue.

In [14], the authors have investigated load value estimation in applications that accept inexactness. Thus, rollbacks are eliminated, since re-execution in the case of misprediction is not necessary. They show that without rollbacks, the load value estimation is still presenting low error in the application's output. Spatio-value approximation is also exploited through the so-called Bunker Cache in [15]. In contrast, our method targets all the types of applications by not accepting any inexactness of the speculated data values.

In [17], the authors proposed a confidence estimation mechanism for value prediction in monocore architectures. They used 3-bit confidence counters and they predicted only on saturated counter (so on highest value). They incremented the counters on correct prediction and reset them (to zero) on misprediction. The authors reported prediction accuracies between 95%

- 99%. They have also introduced the VTAGE context-based value predictor (derived from the previous ITTAGE predictor) which uses global branch history and path information to predict values. In [19], the authors proposed a block-based value prediction scheme which associates the predicted values with fetch blocks. They have also presented the Differential VTAGE predictor (D-VTAGE), which uses stride-based value prediction. The obtained average speedup was 11.2%. In contrast with our work, which investigates value prediction in a multicore system, all these authors were focused on monocore processors.

In [18], Perais and Seznec proposed the *Early Out-of-order Late Execution* (EOLE) monocore architecture, which delays the value prediction validation within the pipeline until the commit stage. Thus, the authors avoid selective replay and enforce complete pipeline squash on misprediction, significantly simplifying the hardware design. Additionally, the authors further reduced the design's complexity by dynamically classifying instructions into early execution, out-of-order execution and late execution instructions. The instructions having immediate or predicted operands are executed early and in-order, in the front end. On the other hand, predicted instructions and high confidence branches are executed late and in-order in a pre-commit processing stage. Both the early and late execution instructions avoid the out-of-order engine, reducing thus the pressure on this unit which led to lower energy consumption. The same classification of the instructions into the early execution, out-of-order execution and late execution categories is applied in [20], too. In contrast with this briefly presented work, limited to monocore approaches, we applied the value prediction on critical load instructions (with miss in the Data L1 Cache) processed within a complex multicore architecture.

In [3], Endo *et al.* have evaluated the potential of value prediction in the context of the EOLE microarchitecture and the D-VTAGE value predictor, considering different compilation options. The authors observed a large benefit from load value prediction, especially in the case of unoptimized codes. In [16], Orosa *et al.* proposed a load value prediction mechanism which predicts the load address first. The predicted load address is then used to index a Value Table (VT) in order to predict the load's value. For a better coverage, the authors improved the hit rate of the VT with an adaptive algorithm which is prefetching future predicted addresses into the VT.

In [13], a very interesting original work, the authors have shown that value prediction can violate the sequential consistency in multithreaded and multicore architectures. The problem can occur when value prediction is applied on codes manipulating pointer-based shared variables. As a concrete instructive example, the authors are considering two distinct threads: one thread is inserting to the front of a list (writer), while the other thread is reading the first element of the list (reader). No further synchronization is necessary between these two threads. The reader or writer may execute its code first, or the instructions may occur in an interleaved manner. If the reader thread is executed first (with load value prediction), the predicted value V1 of its L1 load instruction is speculatively used as an address for another subsequent load instruction L2. An initially wrong prediction (V1) might be erroneously seen as being correct at the time of verification, reading thus a possible wrong value V2 with L2 instruction, since another thread or core (the writer) has already modified the values V1 and V2, between (L1) prediction and verification (late validation). Thus, the authors have firstly shown that predicting the value of an instruction with later validation is not sufficient in a multicore architecture. Despite the fact that this process might be counterintuitive, however it can appear. To solve such possible consistency problems of shared variables, we applied in the simulator the value-based detection solution proposed in [13]. According to this approach, all the load instructions executed with directly or transitively predicted address are re-executed when the address becomes non-speculative. If the corresponding values match, the sequential consistency was not violated and the execution can continue. If the values do not match, all the subsequent data-dependent instructions are re-executed with the right values, restoring in this way the sequential consistency. We applied this selective re-issue mechanism in our work, too.

3. Perceptron-Based Selective Load Value Prediction

In this work, we improve our *selective load value prediction* (SLVP) scheme implemented in a multicore architecture [8] by classifying the critical load instructions into predictable or unpredictable through simple one-cell perceptrons instead of saturating counters. The main goal is to increase the predictability of the SLVP unit. We adapted the perceptron-based branch prediction presented in [10] and [11] for load value prediction. The whole load value prediction process is presented in Figure 1. The LVPT has a TAG field consisting in the most significant part of the load instruction's address stored in the *Program Counter* (PC), a LRU field necessary for the decisions regarding the replacements within the set-associative table, a *Locality History Register* (LHR) containing the last behaviors of the load instruction (see its detailed description below, in paragraph 3.2), the set of perceptron weights W, a history of the load instruction's last distinct produced values, each such value V having associated a confidence automaton C and a vLRU field. The role of the vLRU fields is to decide which one of the last H stored values must be replaced when a new one occurs.

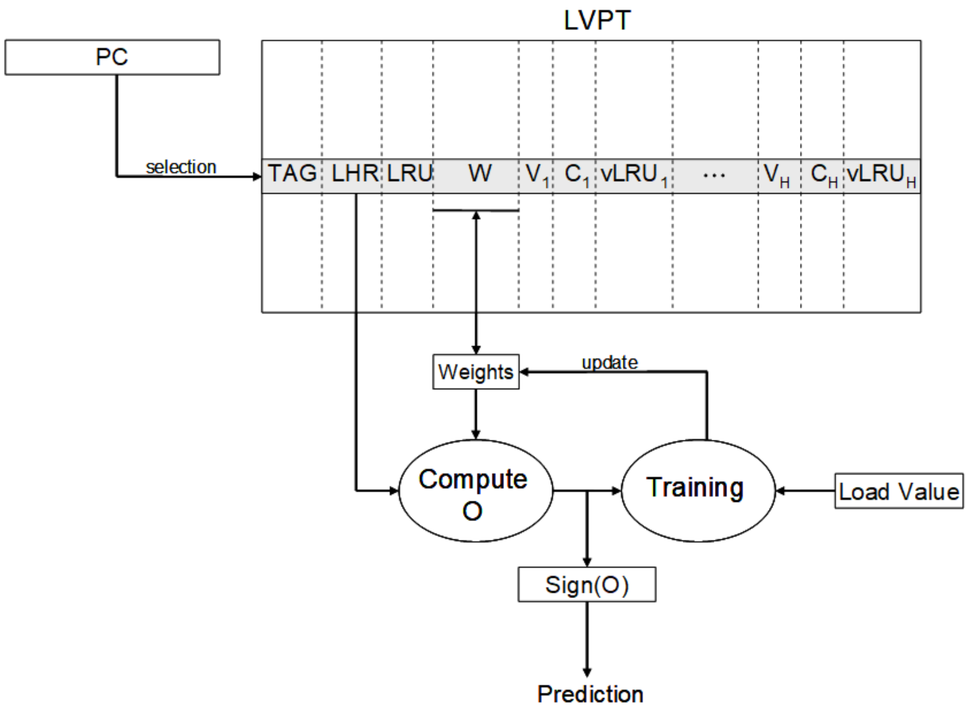

Fig. 1. The whole load value prediction process.

Further we present the selective load value prediction mechanism, the perceptron-based procedure of dynamically classifying load instructions into the predictable or unpredictable classes, as well as the update mechanism.

3.1. The Selective Load Value Prediction Mechanism

If the currently executed load instruction is involving a miss in the Data L1 Cache, the LVPT is accessed during its pipelined issue stage. The LVPT set is selected using the least significant part of the PC and then an associative search is performed for the TAG within the selected set. In the case of miss in the LVPT, the load instruction is considered unpredictable and it is normally executed through the instructions' pipeline. If there is a hit in the LVPT, the highest confidence is evaluated to select the corresponding value. The LHR and the W fields are used for the perceptron's forward stage, in order to determine if the load instruction is predictable or not. In the unpredictable case, the load is normally executed, thus without prediction and speculation. Otherwise, the value selected from the LVPT is speculatively forwarded to the in-flight Read After Write (RAW) dependent instructions and they are executed in a speculative manner, potentially reducing the processing time. In the commit stage, the predicted value is compared with the real value. In the case of misprediction, a recovery process is necessary in order to squash the wrong speculative results and selectively re-execute the RAW dependent instructions with the correct produced values.

During the commit stage, every critical load updates the corresponding SLVP entry: the confidences, the vLRU fields and, additionally, the data value, in the case of a new produced value. The confidence automaton is incremented for the correct value and decremented for the wrong values. If the actual produced value is not belonging to the stored history, it overwrites the value having the lowest vLRU. The vLRU fields are set to their maximum value for the correctly predicted value and decremented for the others. The LHR is correspondingly updated, too (see paragraph 3.2). The backward step is also applied, if necessary, to adjust the weights according to the simplified implemented gradient descent learning rule.

In the case of miss in the LVPT, the entry with the lowest LRU from the set is selected. The new TAG is inserted into the selected entry, the V_1 field is updated with the produced data value and all the confidences from that entry are reset. The first vLRU is set to the maximum, whereas all the other vLRUs are reset. Finally, the LRU of the selected entry is set to its maximum value and the LRUs corresponding to the other entries from the corresponding set are decremented. The LHR bits are reset, excepting LHR_0 which is kept on 1. The weights W are reset, too.

3.2. The Selective Load Value Prediction Mechanism

The LHR contains the last n behaviors (L_i, $1 \leq i \leq n$) of a certain load instruction: L_i=0 when the corresponding Load's real output value (V_R) is not in the V set, or L_i=1 when the produced output value belongs to the V set. The V set from a certain LVPT entry is containing the last H distinct values produced by the previous dynamic instances of the corresponding load instruction. The LHR is logical right-shifted after the commit of each load instruction whose behavior (0 or 1) is inserted as the most significant bit (MSB) into the LHR. We keep LHR_0 always on logical 1 (L_0=1) for the bias weight w_0. The detailed perceptron-based load value prediction scheme is depicted in Figure 2.

The MAX circuit works as follows: if MAX (C_1, C_2, C_3, C_4) = C_K then the MAX circuit generates the binary value of k, codified on 2 bits. Based on this value, the 4:1 multiplexer

(MUX) will select the predicted value V_k, k=1÷4.

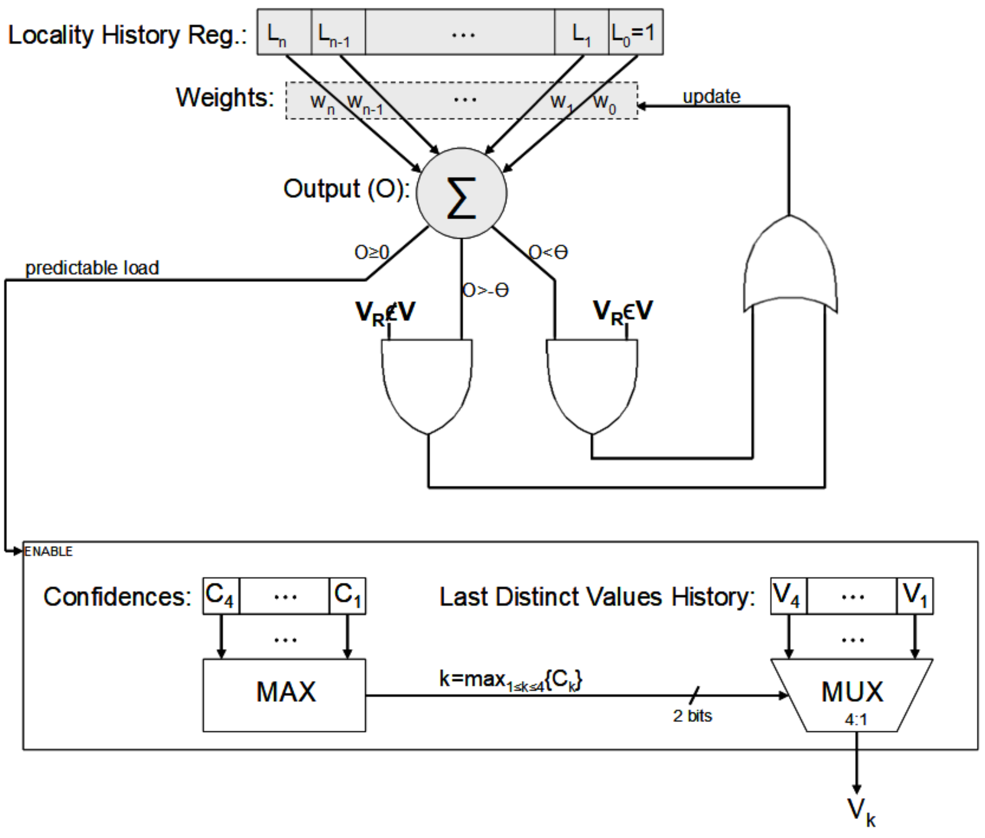

Fig. 2. Perceptron-Based Load Value Prediction Scheme (H=4).

The weights are signed integers, represented on one byte, which cannot exceed the value of Θ. The threshold Θ is also used to decide if the training stage is necessary. As it was shown in [10], the value of Θ is "exactly" $\lfloor 1.93n + 14 \rfloor$.

3.2.1. The Forward Stage

The forward stage consists in calculating the perceptron's output sign which is used to decide if the corresponding load instruction is predictable or not. The output O is given by the following formula (as in [12] for branch prediction):

$$O = w_0 + \sum_{i=1}^{n} \begin{cases} w_i, \; if L_i = 1 \\ -w_i, \; if L_i = 0 \end{cases} \quad (1)$$

where w_i represent the perceptron's corresponding weights (w_0 being the bias weight). The formula (1) is equivalent with the following one:

$$O = w_0 + \sum_{i=1}^{n} w_i \cdot (-1)^{L_i+1} \tag{2}$$

Implementing formula (1) in hardware is not very complicated. According to [11], the above sum can be obtained using a Wallace-tree of 3-to-2 carry-save adder, which reduces the process of adding n bytes to the problem of adding just two bytes. The final two signed integers are added with a carry-look-ahead adder. Taking into account that the Wallace-tree and the carry-look-ahead adder have logarithmic depths, the computation is relatively quick. Only the sign bit of the result is needed to make a decision (the load instruction is predictable or not). A prediction is generated only in the case of a positive output.

3.2.2. The Backward Stage

The backward (learning) stage is applied if the output is in contradiction with the fact that the real value V_R belongs or not to the V set, or if the magnitude of the output is less or equal with the threshold Θ. The pseudocode of the backward algorithm is given below:

```
If (O<0 and $V_R \in$ V) or (O$\geq$0 and $V_R \notin$ V) or $|O| \leq \Theta$ then
    If $V_R \in$ V then
        If $w_0 < \Theta$ then
            $w_0 := w_0 + 1$
        Endif
    Else
        If $w_0 > (-\Theta)$ then
            $w_0 := w_0 - 1$
        Endif
    Endelse
    For i:=1 to n in parallel do
        If ($V_R \in$ V and $L_i$=1) or ($V_R \notin$ V and $L_i$=0) then
            If $w_i < \Theta$ then
                    $w_i := w_i + 1$
            Endif
        Endif
        Else
            If $w_i > (-\Theta)$ then
                    $w_i := w_i - 1$
            Endif
        Endelse
    Endfor
Endif
```

According to the single-cell perceptron's stochastic gradient descent learning method, the simplified implemented learning algorithm increments the bias weight w_0 if V_R belongs to V and decrements it otherwise. Furthermore, it increments the weight w_i if L_i agrees with the fact that V_R belongs or not to V and it decrements that weight in the case of disagreement. Thus, the weights with mostly agreement become positive with large magnitude and those with mostly disagreement become negative with large magnitude. In these cases there is a high correlation between the output and the weight. The correlation between the output and a certain weight is weak if its magnitude is close to 0.

4. Experimental Research Methodology

We have integrated our developed load value prediction technique into the Sniper 6.1 state-of-the-art multicore simulator [1]. The simulated microarchitecture is Intel Nehalem and can include between 1 and 16 cores running at 2.66 GHz.

Table 1. Parameters of the simulated architecture

DL1 / IL1 cache	Size	16 KB
	Block size	64 B
	Associativity	4
	Latency	3 cycles
L2 cache	Size	256 KB
	Block size	64 B
	Associativity	8
	Latency	9 cycles
L3 cache	Size	8192 KB
	Block size	64 B
	Associativity	16
	Latency	35 cycles
Memory	Latency	175 cycles
SLVP	Entries	32
	Associativity	2
	Latency	1 cycle
	Recovery	7 cycles

The baseline configuration of the simulated microarchitecture is presented in Table 1. The latencies have been determined using the Membench tool in [2] and configured in gainestown.cfg. The cache sizes and associativity degrees have been configured in nehalem.cfg. The L3 cache is shared among all cores. We have used for the evaluations the large datasets of the Splash-2 parallel benchmarks [26]. The simulations have been performed on a host computer with Intel Core 2 CPU running at 2.4 GHz and a DRAM memory of 2 GB, under the Fedora 22 operating system (kernel 4.0.8-300). In order to integrate our Selective Load Value Predictor into the Sniper simulator, it was necessary to apply the following setups:

Table 2. Sniper configuration

Configuration	File
[general] issue_memops_at_functional = false	*gainestown.cfg*
[perf_model/core] type = interval	*nehalem.cfg*
[perf_model/dram/queue_model] enabled = false	*base.cfg*
[network/emesh_hop_by_hop/queue_model] enabled = false	*base.cfg*
[network/bus/queue_model] enabled = false	*base.cfg*

The counters necessary for statistics like the number of loads, the number of critical loads, the number of load predictions and the number of correct load predictions have been defined in performance_model.h, registered as output metrics in performance_model.cc and written to the sim.out file within gen_simout.py. The predictor's structure and functions have been implemented in micro_op_performance_model.cc and called in the handleInstruction function from the same source file. We presented these technical details because they are not documented.

We have determined the relative speedup of a multicore architecture with value prediction (RS_{VP}) with respect to the baseline architecture, using the following equation:

$$RS_{VP} = \frac{C_B - C_{VP}}{C_B} \cdot 100[\%] \tag{3}$$

where C_B and C_{VP} are the execution times in cycles for the baseline and for the VP-based architectures, respectively.

5. Experimental Results

We considered as a good starting point for our evaluations the optimal configuration obtained in our previous related work [8]. Thus, we evaluated a dual-core microarchitecture with a level 1 data cache of 16 KB, and a 2-way associative LVPT of 32 entries, storing just one value per entry. We use confidence automatons belonging to the [0, 3] interval attached to the load's values, whose role is just to select for prediction the value with the highest locality. We have varied the size of the LHR, which is equal with the number of weights (W), and we compared these perceptron-based LVPT configurations with our previous counter-based method. The relative speedups obtained over the baseline architecture are depicted in Figure 3.

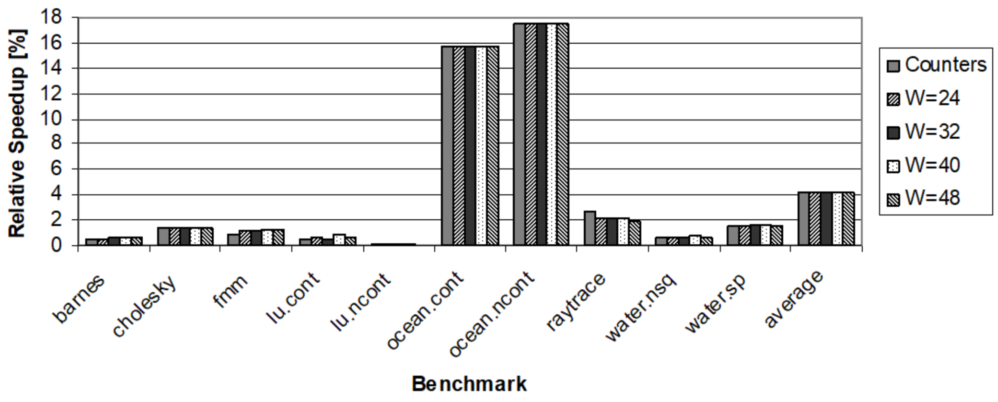

Fig. 3. Comparing the relative speedups of the counter-based and perceptron-based methods.

As Figure 3 shows, there are only insignificant differences between different LHR sizes. The results are in concordance with those obtained in [8]: on the most of the benchmarks, the relative speedup is less than 2%, but on the *ocean* benchmarks it is over 15%. Figure 4 focuses on the averages among benchmarks.

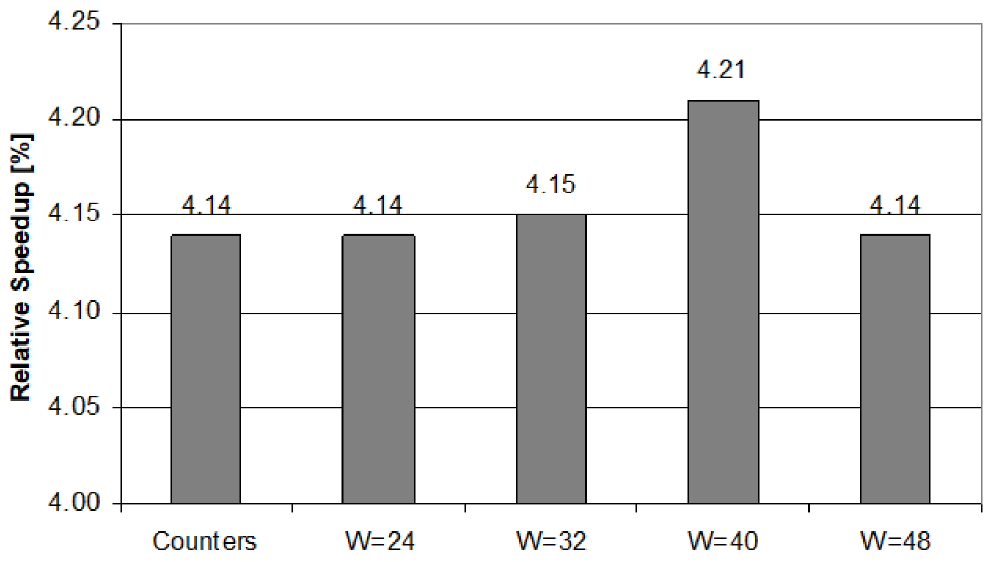

Fig. 4. The average relative speedups of the counter-based and perceptron-based methods.

As Figure 4 shows, the best configuration is using 40 weights, involving thus an LHR of 40 bits (load instruction behaviors). With that configuration, the average relative speedup was 4.21% which is slightly better than the 4.14% obtained previously with the counter-based method. Next, we have analyzed this best perceptron-based method in comparison with the previous counter-based method.

Table 3. Analyzing the optimal configuration

Splash-2 Benchmarks	Counter-Based Method		Perceptron-Based Method (W=40)	
	Coverage [%]	Prediction Accuracy [%]	Coverage [%]	Prediction Accuracy [%]
barnes	2.49	82.07	2.55	86.85
cholesky	9.34	75.30	9.70	72.73
Fmm	22.48	89.87	24.60	86.94
lu.cont	0.20	84.32	0.15	86.63
lu.ncont	0.13	99.79	0.18	99.13
ocean.cont	22.02	99.12	22.09	99.24
ocean.ncont	23.63	99.09	23.70	99.36
raytrace	24.62	87.00	27.06	73.17
water.nsq	19.88	88.25	20.63	96.22
water.sp	20.85	66.88	26.46	91.12
average	14.56	87.17	15.71	89.14

Table 3 presents the coverage, computed as the number of correctly predicted loads divided to the number of critical loads, and also the prediction accuracy, computed as the number of correctly predicted loads divided to the number of predicted loads. The results show that both indicators are better for the proposed perceptron-based method, meaning that a higher number of loads are predicted and the prediction is provided with a higher accuracy.

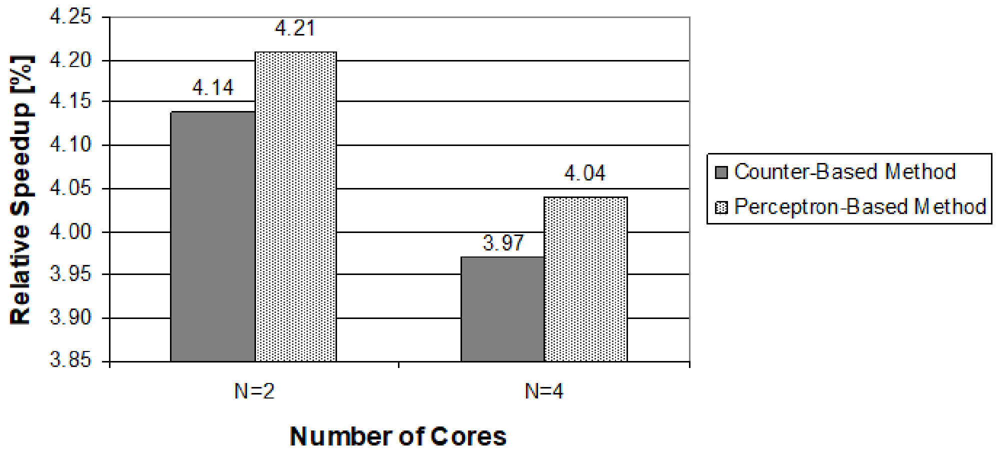

Fig. 5. The average relative speedups of the counter-based and the optimal perceptron-based methods considering different core numbers.

Finally, we have compared the optimal perceptron-based configuration (W=40) with the counter-based one, considering dual-core and quad-core architectures. Figure 5 presents the relative speedups obtained with respect to the corresponding baseline architectures, averaged on all the benchmarks. The relative speedup is lower as the number of cores grows, but the performance gain is at the same level.

6. Conclusions and Further Work

In this work, we have enhanced the selective load value prediction mechanism with hardware perceptron-based classification of load instructions as predictable or unpredictable. Thus, we have inserted into the LVPT's hardware structure two new fields, the locality history register and the set of weights, which are used to determine if the corresponding load instruction is predictable or not. The goal of the load value predictor is to anticipate the values of critical load instructions and to unlock in a speculative manner the subsequent RAW dependent instructions' execution. Since each misprediction implies a recovery process, we are interested in obtaining a high prediction accuracy. Therefore, we predict only the results of critical load instructions identified as predictable by our developed perceptron. In the experimental process, we have varied the size of the perceptrons between 24 and 48 bits. The results have shown that the best configuration implies 40 weights within each perceptron. With the dual-core configuration, consisting in a level 1 data cache of 16 KB and a 2-way associative LVPT with 32 entries and one value per entry, we have obtained an average relative speedup of 4.21% over the baseline architecture (the same configuration, but without LVPT), with a maximum speedup of about

17% (on the *ocean.ncount* benchmark).

As a further work, we propose to generalise the implemented value prediction method to Intel Nehalem's all long-latency machine instructions (multiplication, division, square-root, etc.), through some dedicated hardware value predictors. In this way, the obtained speedups would be, for sure, much more significant. Also we intend to further apply an automatic multi-objective optimization method for our speculative multicore system, based on our already developed complex and effective software optimization tools [25]. Obviously, this automatic design space exploration will provide a better optimal solution, but, due to the enormous design space, not certainly the ideal one.

Acknowledgements. We express our gratitude to Dr. Wim Heirman (Intel) for his support in configuring the Sniper simulator and also to our former MSc student Claudiu Bruda for providing his useful and competent help in identifying the correct location where we could integrate the load value predictor.

References

[1] T.E. CARLSON, W. HEIRMAN, S. EYERMAN, I. HUR, L. EECKHOUT, *An Evaluation of High-Level Mechanistic Core Models*, ACM Transactions on Architecture and Code Optimization (TACO), **11**(3), 2014.

[2] T.E. CARLSON, W. HEIRMAN, *The Sniper User Manual*, 2013.

[3] F.A. ENDO, A. PERAIS, A. SEZNEC, *On the Interactions Between Value Prediction and Compiler Optimizations in the Context of EOLE*, ACM Transactions on Architecture and Code Optimization, **14**(2), 2017.

[4] A. FLOREA, C. BUDULECI, R. CHIS, A. GELLERT, L. VINTAN, *Enhancing the Sniper Simulator with Thermal Measurement*, The Eighteenth International Conference on System Theory, Control and Computing, Sinaia, 2014.

[5] A. GELLERT, A. FLOREA, L. VINTAN, *Exploiting Selective Instruction Reuse and Value Prediction in a Superscalar Architecture*, Journal of Systems Architecture, Elsevier, **55**(3), pp. 188–195, 2009.

[6] A. GELLERT, G. PALERMO, V. ZACCARIA, A. FLOREA, L. VINTAN, C. SILVANO, *Energy-Performance Design Space Exploration in SMT Architectures Exploiting Selective Load Value Predictions*, International Conference on Design, Automation and Test in Europe (DATE 2010), pp. 271–274, Dresden, Germany, 2010.

[7] A. GELLERT, H. CALBOREAN, L. VINTAN, A. FLOREA, *Multi-Objective Optimizations for a Superscalar Architecture with Selective Value Prediction*, IET Computers & Digital Techniques, **6**(4), pp. 205–213, Stevenage, United Kingdom, 2012.

[8] A. GELLERT, L. VINTAN, *A Multicore Architecture with Selective Load Value Prediction*, Proceedings of the Romanian Academy, Series A, **19**(4), pp. 597–604, 2018.

[9] A. GELLERT, A. FLOREA, U. FIORE, P. ZANETTI, L. VINTAN, *Performance and Energy Optimisation in CPUs through Fuzzy Knowledge Representation*, Information Sciences, Elsevier, **476**, pp. 375–391, 2019.

[10] D. JIMÉNEZ, C. LIN, *Dynamic Branch Prediction with Perceptrons*, Proceedings of the Seventh International Symposium on High Performance Computer Architecture (HPCA-7), Monterrey, Nuevo Leon, Mexico, 2001.

[11] D. JIMÉNEZ, C. LIN, *Neural Methods for Dynamic Branch Prediction*, ACM Transactions on Computer Systems,**20**(4), 2002.

[12] D. JIMÉNEZ, *Fast Path-Based Neural Branch Prediction*, 36th International Symposium on Microarchitecture, San Diego, CA, USA, 2003.

[13] M.M.K. MARTIN, D.J. SORIN, H.W. CAIN, M.D. HILL, M.H. LIPASTI, *Correctly Implementing Value Prediction in Microprocessors that Support Multithreading or Multiprocessing*, 34th Annual ACM/IEEE International Symposium on Microarchitecture, Austin, Texas, 2001.

[14] J.S. MIGUEL, M. BADR, N.E. JERGER, *Load Value Approximation, Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 127–139, Cambridge, UK, 2014.

[15] J.S. MIGUEL, J. ALBERICIO, N.E. JERGER, A. JALEEL, *The Bunker Cache for Spatio-Value Approximation*, 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, Taiwan, October 2016.

[16] L. OROSA, R. AZEVEDO, O. MUTLU, *AVPP: Address-First Value-Next Predictor with Value Prefetching for Improving the Efficiency of Load Value Prediction*, ACM Transactions on Architecture and Code Optimization, **15**(4), January 2019.

[17] A. PERAIS, A. SEZNEC, *Practical Data Value Speculation for Future High-End Processors, 20th International Symposium on High Performance Computer Architecture*, pp. 428–439, Orlando, FL, USA, February 2014.

[18] A. PERAIS, A. SEZNEC, *EOLE: Paving the Way for an Effective Implementation of Value Prediction*, 41st Annual International Symposium on Computer Architecture, pp. 481–492, Minneapolis, MN, USA, June 2014.

[19] A. PERAIS, A. SEZNEC, *BeBoP: A Cost Effective Predictor Infrastructure for Superscalar Value Prediction*, 21st International Symposium on High Performance Computer Architecture, pp. 13–25, San Francisco, CA, USA, February 2015.

[20] A. PERAIS, A. SEZNEC, *EOLE: Combining Static and Dynamic Scheduling through Value Prediction to Reduce Complexity and Increase Performance*, ACM Transactions on Computer Systems, **34**(2), May 2016.

[21] L. VINTAN, *Towards a Powerful Dynamic Branch Predictor*, Romanian Journal of Information Science and Technology, **3**(3), pp. 287–301, ISSN: 1453-8245, Romanian Academy, Bucharest, 2000.

[22] L. VINTAN, A. GELLERT, A. FLOREA, *Register Value Prediction Using Metapredictors*, Proceedings of the Eighth International Symposium on Automatic Control and Computer Science, Iasi, October 2004, republished in Bulletin of the Polytechnic Institute of Iasi, Fasc. 1–4, Section IV, Tomul L (LIV), pp. 109–122, Iasi, 2004.

[23] L. VINTAN, A. FLOREA, A. GELLERT, *Focalising Dynamic Value Prediction to CPU's Context*, IEE Proceedings Computers & Digital Techniques, **152**(4), pp. 473–481, Stevenage, United Kingdom, July 2005.

[24] L. VINTAN, A. GELLERT, A. FLOREA, *Value Prediction Focalized on CPU Registers*, Advanced Computer Architecture and Compilation for Embedded Systems (ACACES 2005), Academia Press, pp. 181–184, Ghent, Belgium, July 2005.

[25] L. VINTAN, R. CHIS, Md. ALI ISMAIL, C. COTOFANA, *Improving Computing Systems Automatic Multi-Objective Optimization through Meta-Optimization*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 35, Issue 7, pp. 1125–1129, July 2016.

[26] S.C. WOO, M. OHARA, E. TORRIE, J.P. SINGH, A. GUPTA, *The SPLASH-2 Programs: Characterization and Methodological Considerations*, 22nd International Symposium on Computer Architecture, pp. 24–36, Italy, June 1995.