Improving Multicore Architectures by Selective Value Prediction of High-Latency Arithmetic Instructions

Claudiu BUDULECI, Arpad GELLERT, Adrian FLOREA, Remus BRAD
Computer Science and Electrical Engineering Department, Lucian Blaga University of Sibiu,
Emil Cioran Street, No. 4, 550025 Sibiu, Romania
claudiu.buduleci@ulbsibiu.ro

***Abstract*—This work is an original contribution consisting in the implementation and evaluation of a selective value predictor in a multicore environment, with focus on long latency arithmetical instructions, having the goal to break the dataflow bottleneck of each core, thus increasing the overall performance. The Sniper simulator was used to augment the Intel Nehalem architecture with a value predictor and to estimate the computing performance, area of integration, power consumption, energy efficiency and chip temperature for the enhanced architecture. We run simulations and study the impact of the number of values which are used for prediction for each instruction. By increasing the history length, we measured on average more than 3 % increase in performance (core speed-up), a reduction in chip temperature from 57.8 °C to 56.17 °C, and lower energy consumption in most cases compared with the baseline configuration. We also realized a comparison between the value prediction and dynamic instruction reuse techniques in equitable condition (to exploit the same value locality), where we highlight the advantages and disadvantages of each technique in the given context.**

***Index Terms*—multicore processing, computer simulation, prediction methods, benchmark testing, microprocessors.**

I. Introduction

Breaking the dataflow bottleneck of microprocessors is still an open research topic and researchers are still finding ways of pushing this limit, using well known techniques like out-of-order processing, dynamic instruction reuse (DIR), and value pre-diction (VP). VP is a speculative technique, which allows the execution of dependent instructions much earlier in the processor pipeline. The dependent instructions are executed using a speculated value provided by the predictor based on the current instruction context, and after the execution of this instruction the computed value is compared with the speculated value. In the case of a match, all the instructions which used the speculated value are committed, but if a mismatch is found then the whole pipeline is flushed, and all instruction are executed again from the point of speculation. Flushing the pipeline and executing again the instructions is not only time consuming, but also consumes additional power which impacts the overall energy consumption and chip temperature. DIR is an anticipative technique which consists of a reuse buffer which stores the input values and the results of instructions. When an instruction is executed, the reuse buffer is consulted, and in case an entry is found having the same input, then the result is taken out of the buffer and feed to dependent instructions inside the pipeline, without the need to re-execute the instruction. In comparisons with the VP, the DIR technique does not use any speculation and the mentioned drawbacks are not applicable.

In our previous work [1] we emphasized the fact that computer applications contain a lot of time-consuming arithmetic instructions, and any performance gain is beneficial. More than that, most of those instructions are subject to repetition.

This work provides a brief overview of the state-of-the-art VP techniques and their challenges in multicore architectures, multicore simulators and benchmarks used in computer architecture research. We implemented a selective VP designated for the long-latency arithmetic instructions in a multicore environment and study the impact of the history length variation over the interested metrics: prediction accuracy, performance, integration area, power consumption, energy consumption and chip temperature. A comparison was made between the VP and DIR techniques in relation with the number of cores, in equitable conditions (exploiting the same value locality). To validate our enhancement of the Intel Nehalem architecture, we used parallel computer benchmarks, which distribute the workload among multiple cores. We have simulated the enhanced processor using Sniper [2], which is a multicore, high-speed and accurate x86 simulator, able to execute parallel programs. To run the simulations, we have used the supercomputer from the Advanced Computer Architecture and Processing Systems (ACAPS) Research Center from Sibiu.

This paper is structured as follows. In Section 2 we present an overview of the latest VP techniques, multi-core simulation and benchmarking, along with the most recent data security concerns introduced by the speculative techniques. Section 3 presents in detail our modification to the Sniper simulator. Section 4 shows our simulation methodology and environment. In Section 5 we show and discuss our simulation results, and finally Section 6 concludes this work and present future work possibilities.

II. Related Work

A. Multicore Simulation

Some of the most popular open-source multicore simulators used in academic research are Graphite [3], Sniper [2] and Gem5 [4].

Graphite is a distributed parallel simulator framework for

Digital Object Identifier 10.4316/AECE.2024.02007

multicore systems. It can simulate microarchitectures having a few cores up to thousands of cores. For simulations it uses techniques like direct execution, multi-machine distribution, analytical modelling, and lax synchronization.

Build upon Graphite, Sniper is a more recent x86 multicore simulator which uses the interval simulation concept. Cycle-by-cycle simulation requires a vast number of details for each instruction which gets through the simulated pipeline execution phases. Managing those details makes simulation cumbersome and slow. To increase the simulation speed, Sniper uses the interval simulation method which approximates the core's performance by applying at well-defined time intervals an analytical model to estimate the timing within the interval. To integrate our proposed technique we chose Sniper, as it provides a good trade-off between simulation time and accuracy.

Gem5 offers a simulation framework that is easily customizable, supports various CPU models and instruction set architectures (ALPHA, ARM, MIPS, SPARC, Pow-er and x86), a flexible memory system that contains different cache coherence protocols and interconnect models.

B. Security and Data Consistency Concerns

Even if we have not implemented security and data consistency techniques in VP, the role of this section is to show the importance those topics and the need to approach it from more perspectives besides performance, energy consumption, integration area and temperature.

Breaches like Meltdown [5] and Spectre [6] affected millions of microprocessors, while having the same philosophical root cause, driven by a hardware vs. software discipline seclusion paradigm, concretely by the lack of cross-discipline collaboration and communication. Sensitive data, which is protected by a memory protection mechanism, can be read out by exploiting those breaches, dodging the activation of the memory protection mechanism in time. The authors of the breaches have discovered a new way to exploit the side effects of out-of-order execution (Robert Tomasulo's algorithm) and speculative execution (such as BP and VP). These breaches depend on a narrow window of time to access the sensitive data. Meltdown takes advantage of out-of-order execution, whereas Spectre exploits speculative execution. Both breaches rely on the fact that the data is cached before the memory protection exception is triggered.

A pure hardware-based approach to mitigate security concerns in a system with a VP unit, denoted VPsec, was proposed in [7]. The authors suggested an enhanced predictor which is capable to detect abnormal behaviors during the program executions and to react. In case an abnormal behavior is detected, the predictor can decide to provide a random (infected) value, for misguiding the attacker, which will read the altered value instead of the correct one. The approach imposes no software overhead (no increase in memory consumption) and, under typical attack scenarios, it maintains most of the performance advantages of VP. The simulation results of VPsec confirm its effectiveness in countering attacks while preserving the performance of modern microprocessors.

Another security concern presented in [8] with focus on VP shows that sensitive information can be leaked through the microarchitectural state maintained by predictors in contemporary processors. The security of VPs has been examined in this study, and novel security attacks have been presented. Until now these attacks have not been considered for VPs functional units as potential vulnerabilities. Additionally, the study illustrates the existence of various VP attack variations, as determined through a new attack model. The research emphasizes the significance of conducting security assessments of processor features before their implementation in silicon, to comprehend their security during the design phase.

Towards implementing VP techniques in microprocessors which support multithreading or integrate multiple cores, one must ensure that it is implemented correctly, without introducing unintended side effects.

For example, in [9], the authors discovered that the VP technique can cause unexpected erroneous results if is not correctly implemented. They demonstrate on a pointer-based data structure, a situation where the reader gets a value which is invalid, neither old nor new, making this scenario an unintended side effect of the VP technique. Different techniques were analyzed and proposed to eliminate these consistency model violations. Unfortunately, they have a negative impact on performance and are adding even more complexity to the microarchitecture. Although, the sequential consistency issues rarely occur in practice, the designer of the chip must ensure a solution to avoid them when implementing VP.

C. Value Prediction

VP is a speculative micro-architectural technique that improves the instruction-level parallelism. The parallelism is increasing each time the value of an instruction is correctly predicted. The technique was proposed in the period 1995-1997 by four distinct groups in [10-15].

Lately, researchers focus on the VP technique and lagged the DIR. The rationale of the statement is based on the following observation. Recently, at the prestigious ISCA conference, the 1st Championship Value Prediction (https://www.microarch.org/cvp1/cvp1) was organized. Also, other similar competitions will be organized, with involvement of major companies in the field, such as Intel, Qualcomm, Nvidia, Samsung, etc.

The fundamental difference between VP and DIR is the following: VP is a speculative technique and DIR is non-speculative. This means that the DIR technique eliminates the speculative execution of instructions. Basically, the recovery time paid in case of a wrong prediction does not exist in DIR. This puts DIR in advantage over VP. Both architectural techniques can exploit the same value locality, in a history context of n values.

Given the execution phases of an instruction (fetch, decode, issue, execute, writeback and commit), the VP table is accessed in the frontend and based on the information from the prediction table a prediction can be made. In case a prediction is done, the processor executes speculatively the subsequent dependent instructions using the predicted value. When the outcome of the instruction is available, it is compared with the prediction and, if they are matching (correct prediction), the speculated instructions are finally committed. In case of a wrong prediction the speculative

execution must revert to the point when the prediction was done, the instruction must be executed again with the correct value. This implies an additional recovery time penalty.

One remark is that the VP unit can be accessed earlier in the pipeline stages in case the prediction is not done on values which are available only after the decode stage (e.g., register values, data addresses). If this is the case, then the predictor can be consulted directly in the fetch phase, using only the program counter (PC) or with a more complex access using both the PC and part of the global branch history.

Researchers proposed different implementation techniques of VP which will be summarized in the next sections, grouping them in the following categories: computational, contextual and store/load VPs.

D. Computational Predictors

In [12], the authors proposed a simple VP, called last-VP, which uses as current prediction the last computed value of a previous execution of the instruction. Such a predictor decides what value it will predict based on a quite rudimentary decision mechanism using confidence counters, thus achieving a performance gain from 4.5 % up to 23 % using this technique.

A more advanced technique is the stride predictor, which was introduced in [15]. The advantage of the stride predictor over the last-VP, is that it can predict values which were not previously seen. Generally, stride predictors are simplistic models because they only add to the last value the difference of the last two recent values that were generated by an instruction.

Most of the modern context-based predictors are incorporating in a way or another, a stride predictor, making it an essential part of modern VPs. In [16], context-based, stride and hybrid predictors are applied for register-centric VP. Basic stride predictors are good at predicting arithmetical strides but are problematic when they must deal with stride patterns that occur on an interval basis. However, a newer predictor, called Stride Equality Prediction (SEP), is proposed in [17] by Yang et al., and is specialized in identifying and predicting interval stride patterns. They achieved a 5.3 % improvement compared with the state-of-the-art E-Stride predictor [18]. They also enhanced the Context-Based Computational Value TAGE (CBC-VTAGE) with SEP and achieved 1.5 % better results without adding extra costs.

E. Context Predictors

Context-based predictors try to identify patterns between multiple produced values of one dynamic instruction and to predict the next value based on the identified pattern. Predictors using a finite context method are based on a mechanism that estimates the next value based on a finite number of previous values. A predictor of this kind of order K will use the last K previous values. In other words, such predictors implement learning mechanisms that are used to predict future values.

A finite context method (FCM) was proposed in [14] along with an intensive study for the predictability of data values. They define and compare different types of predictor models, computational (stride and last value) vs. the proposed contextual predictor. The contextual predictor can record previous values for multiple instructions, thus maintaining a history and perform a prediction based on the identified patterns. In average, the contextual predictor has an 20 % better accuracy than the computational predictors.

One general drawback of the FCM predictor is that the size of the value history table is increasing exponentially along with the length of the history. The efficiency of prediction accuracy is correlated with the prediction history. Usually, more history means better accuracy in prediction.

To solve the exponential size increase in the FCM predictor, in [19] the authors proposed an optimized variant named Differential FCM (DFCM). They are proposing to predict strides instead of values, thus achieving an increase of 33 % in prediction accuracy. Using strides instead of actual values to identify the difference over time makes the predictor more space-efficient and allows a longer history pattern to be recorded and used for prediction. Another improved aspect of DFCM over the initial implementation is the ability to identify patterns faster, especially in case of constants.

The DFCM predictor was further improved in [20] achieving an 28.1 % improvement in overall speedup. There are four notable improvements proposed on top of the standard DFCM: early update policy, a value estimator which correlates dependencies for prediction, blacklist usage for hard to predict instructions, and the introduction of dynamic context length determination and adaptation.

The VTAGE predictor [21] is using multiple sources of information to perform a prediction. It uses as input the global branch history, difference of successive values and history of local values. Also, a combination of multiple prediction tables are checked in parallel, and it will use the entry with the longest history for prediction. Saturated confidence counters are also considered for the final verdict. The prediction table is accessed using a hash, which is computed using multiple global branch history lengths in combination with the PC, such combination resulting a geometric progression. This predictor was further improved in the E-VTAGE [18] variant with the following enhancements:

- usage of tags and associativity on the PC indexed component;
- was optimized to reduce the needed storage;
- has improved the confidence management;
- reduced the number of burst miss predictions.

The E-Stride predictor is an improved classical stride predictor. It was adapted in such way that it can predict inflight instructions by considering the speculative instances of instructions and multiple improvements.

In [18], the authors are proposing a state-of-the-art VP named Enhanced VTAGE Enhanced Stride (EVES). It combines two predictors E-VTAGE and E-Stride which are not relaying only on using the last result of the instruction to compute the prediction. Each predictor is addressing different types of instructions. According to the Championship Value Prediction competition, the speedup achieved using this predictor is the highest, 23.8 % using 8 KB storage. Increasing the storage up to 32KB the obtained speedup is 28.6 % and using unlimited storage space a 45 % performance increase was measured. It won the first place, thus becoming the state-of-the-art of VPs.

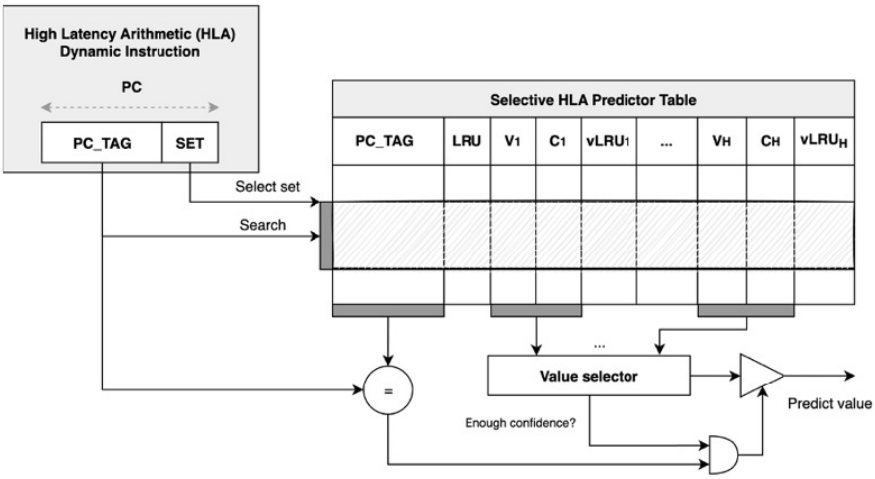

Figure 1. SHLA-VP scheme

Another modern VP is proposed in [22], which won the second place in the Value Prediction Championship held in 2018, in the 8 KB storage budget category, achieving a 17.1 % performance speedup. This predictor, named Context-Base Computational VP with Value Compression (CBC-VTAGE), enhances the original VTAGE predictor scheme. The predictor proposed in [23] won the 2nd place in the Value Prediction Championship held in 2018, in the 32 KB storage budget section achieving a 4.3 % performance speedup. It also won the 3rd place in the 8 KB storage category, obtaining a speedup of 3.4 %. The predictor is called H3VP: History-Based Highly Reliable Hybrid Value Predictor. It is composed of a common history table and three independent predictors (arithmetic, 2-periodic and 3-periodic).

F. Store/Load Predictor

In [24], the authors propose a Decoupled Load Value Predictor to tackle two important challenges when dealing with VP. First, is related to the store instructions which are changing the content of memory locations, thus making the values in the predictor deprecated and the predictor itself prone to an increase in misprediction rate. It takes time until the modified value via a store operation is loaded and used in the pipeline, meanwhile the predictor most probably will speculatively insert the deprecated value in the pipeline. The effect of a misprediction consists in costly pipeline flushes, which denotes the second challenge: to minimize the number of mispredictions. To mitigate this issue most of the VPs are using confidence counters, but, according to the authors, there is a probability which negatively impacts the training time and the overall prediction coverage.

The idea of this predictor is to not use the outcome value for prediction, instead to use the memory address and to rely on the data cache to distribute the predicted values just in time for prediction. In this way, they can predict the value of a subsequent load instruction considering the value from the data cache. This idea is used to tackle the first challenge. As for the second, they implement a new context-based address prediction scheme, which uses the load-path history to improve the accuracy. The performance improvement is 4.8 % in average, having a maximum of 71 %, using a memory allocation of 8 KB.

Selective VP applied on long-latency Load instructions within superscalar processors was evaluated in [25-31], whereas its integration into multicore microarchitectures was discussed in [32-33].

In [32], the scheme applied in a multicore environment is called Selective Load Value Predictor (SLVP) and targets the prediction of long-latency load instruction's outcome. The selectiveness is used to maintain an energy-efficient architecture. The overall performance speedup was 4 % in average and the authors observed an 1.25 % reduction in energy consumption.

In [33], the authors expanded the idea of load VP, by using a perceptron-based classification method to categorize load instructions into predictable and unpredictable classes. The prediction technique has been incorporated into the Sniper multicore simulator. The primary objective of the load VP is to predict the values of crucial load instructions and to enable the processing of dependent instructions in a speculative manner. VP is considered only if the corresponding load is found in the predictable state, since high prediction accuracies are required. Assessments were conducted on the Splash-2 parallel benchmarks, revealed an average relative speedup of 4.21 % compared to the baseline multicore architecture, with a maximum of approximately 17 %.

III. Technical Modifications

A. Value Prediction Scheme

In Fig. 1 is depicted the scheme of the Selective High Latency Arithmetic (SHLA) VP, denoted SHLA-VP. It comprises a set-associative enhanced last VP, which can keep up to H result values, produced by one instruction. The values are kept in dedicated entries from V1 up to VH. For each value, a two-bit confidence counter C and a two-bit vLRU field is associated. The vLRU field is used to determine which one of the H values is replaced with a newer one. In case of correct prediction, the corresponding vLRU field is set to the maximum value and is decremented for the others. The LRU field is used to decide on which entry gets evicted within the set-associative table, the well-known LRU algorithm is applied. The table is accessed using two sub-set bitfields from the instruction PC: PC_TAG represented by the most significant bits and the SET given by the least significant bits.

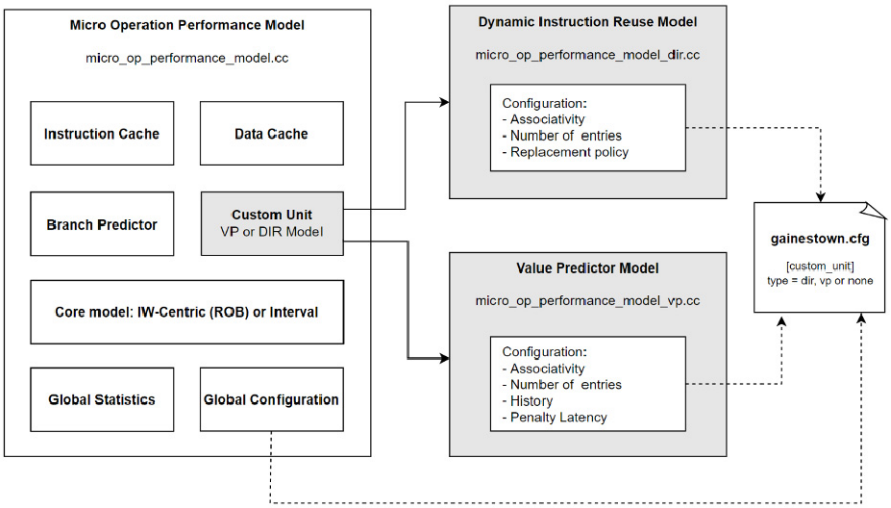

Figure 2. High-level structure of the Micro Operation Performance Model

In the current implementation, each core has its own local implementation of the presented VP model, no information is shared between cores. All parameters are listed in Table I.

TABLE I. LIST OF PARAMETERS OF THE VP MODEL

Parameter	Value
associativity	Specifies the associativity of the table: {1, 2, 3, 4, 8, 16}. Note: value "1" – means a direct mapped table.
num_entries	The total number of entries allocated inside table.
history	The number of result values which are kept for one entry.
penalty_latency	The latency penalty in case of a wrong prediction [cycles].
block_size	Block size [bytes].
size	Total size of the unit [KB].

B. *Micro-Operation Performance Model*

The solution to make the operand values accessible in Sniper, was presented in [34], and in this work we apply the same approach. The SHLA-VP is modeled inside the Core Performance Model of the simulator, which lays in the backend, more concrete in the Micro-Operation Performance model area (an overview of this model is presented in Fig. 2).

In this area, the dynamic instructions are further split into multiple micro-operations. Based on those operations, the intra-instruction dependencies are determined, and the latency of each micro-operation is calculated. It communicates with instruction and data caches to calculate the access latency, for example in case of a miss in one of the caches penalty latency is added to the instruction execution time.

For branch instructions, the dedicated BP model is accessed. After all the micro-operations are evaluated and the corresponding latency is assigned, they are pushed into the selected core model which further simulates them. When the core model finishes the simulation, it returns the number of committed instructions and the latency necessary to commit those instructions.

We inserted a placeholder for our VP or DIR table, namely "Custom Unit". The active custom unit is selected by the "custom_unit/type" configuration parameter, which can be placed for example inside the "gainestown.cfg" configuration file or set via a command line parameter (e.g., "-g --custom_unit/type=dir").

Based on this parameter the DIR or VP model is selected for one simulation.

Each custom unit is implemented in separated source files and can be configured via dedicated parameters. For example, there is no need to rebuild the simulator to change the associativity of the selected model.

C. *Latency Adaptations*

Considering the architecture of the micro-operation performance model presented in B we adapted the micro-operations before the core model simulation is performed. In this way, the existing models were not modified, and the proposed solution is working independently regarding the simulation model. Sniper has multiple models for core simulation: one-ipc, interval and instruction-window centric (rob timer).

According to [2], the one-ipc model offers the lowest simulation accuracy against real hardware, it is also the simplest model in terms of modelling complexity and is prone to misleading and incorrect result. The interval simulation model offers high simulation speed, modest accuracy, and short simulation time. Instruction-window centric simulation is the newest implemented model and offers better estimations due to the modelling of a reorder buffer. The simulation time is slightly increasing when compared to the interval model.

For VP, we distinguish between three outcomes: no prediction, correct prediction, and wrong prediction. No prediction means that the table is accessed and the information (LRU counters, confidence counters, values, etc.) is updated. In this case we do not apply adaptations regarding latency, but we increment the read/write counters necessary to estimate for power consumption estimations. A correct prediction will impact the latency in the following manner:

- all execution micro-operations of the current instruction are squashed and not feed into the core simulation model (analogy to unblock the execution of instructions which are waiting for the outcome of the current instruction, without interfering with the simulation model);
- the number of executed instructions is incremented by one to reflect that the execution of the current instruction was correctly speculated;

- one cycle is considered for accessing the table, the cycle being added to the last stage of store micro-operation.

For a wrong prediction we add to the execution latency of the current instruction an extra 17 cycles of penalty, the same as it is used to treat a mispredicted branch instruction. The misprediction penalty (in cycles), was measured in [35] (paragraph 3.6) for BP in Intel Nehalem by Agner Fog.

IV. Simulation Environment and Metrics

The host configuration used to run the simulations was a computer with Intel Xenon Gold 6240R CPU (2.4 GHz, 48 physical and 96 virtual cores), 128 GB DRAM (2933 MHz) and 2 TB SSD storage.

To run the simulator, we used a virtual machine running Ubuntu 18.04 64-bit version. The simulated microarchitecture is Intel Nehalem codename Gainestown using a clock speed of 2.66 GHz. The baseline configuration is presented in Table II.

TABLE II. BASE CONFIGURATION OF THE SIMULATED ARCHITECTURE (INTEL NEHALEM – GAINESTOWN)

	Parameter Name		Value
Intel Nehalem – Gainestown	L3 Cache (Shared)	Size	8192 KB
		Associativity	16
	L2 Cache	Size	256 KB
		Associativity	8
	L1 Data Cache	Size	32 KB
		Associativity	8
	L1 Instruction Cache	Size	32 KB
		Associativity	4
	Frequency		2.66 GHz
	Number of cores		4

We are interested in the following metrics: performance, area of integration, dynamic power consumption, energy consumption and maximum temperature. For DIR we are interested also in the reuse rate and for VP in the prediction accuracy, which are the main performance indicators for these units.

Regarding the performance metric, we are measuring it as instructions per cycle (IPC) from two perspectives: core and processor. For the core performance we are averaging the calculated IPC of all used cores and for processor performance we sum the number of instructions executed by all cores and divided by the longest cycle time among cores. Below are presented the equations, which were used to determine the performance, equation (1) for IPC, (2) for the performance of the core and (3) to compute the processor performance.

The relative speedup is also calculated according to (4).

$$IPC = \frac{I}{N} \tag{1}$$

$$Core\ Performance = \sum_{i=1}^{C} \frac{IPC_i}{C} \tag{2}$$

$$Processor\ Performance = \sum_{i=1}^{C} \frac{I_i}{MAX_{1 \le k \le C}\{E_k\}} \tag{3}$$

where:

I = the number of instructions executed;

N = the number of cycles necessary to execute the instructions;

C = the number of cores;

E_k = execution time in cycles for core k.

$$Relative\ Speedup = \frac{IPC_E - IPC_B}{IPC_B} \times 100\ [\%] \tag{4}$$

where:

IPC_B = IPC of the baseline configuration;

IPC_E = IPC of the enhanced architecture (with DIR or VP unit).

$$Energy = \frac{AVG(P) \times MAX(C)}{f_{CPU}}\ [J] \tag{5}$$

where:

P = the dynamic power consumption;

C = the number of cycles;

f_{CPU} = frequency of the simulated processor [Hz].

$$Energy\ Reduction = \frac{E_B - E_E}{E_B} \times 100\ [\%] \tag{6}$$

where:

E_B = energy consumption of the baseline configuration;

E_E = energy consumption of the enhanced architecture (with DIR or VP unit).

The area of integration is measured in mm^2 and the dynamic power consumption in Watt [W], the values being estimated by invoking the McPAT [36] framework. The temperatures are estimated in Celsius degrees using the HotSpot tool. We are interested in the maximum recorded value. The energy consumption of the whole chip is determined according to equation (5) and it is measured in Joules (J). The energy reduction percentage is also considered, the computation is done applying formula (6).

As for benchmarking, we use the programs from the Splash-2 [37] suite with the large input dataset. Both units are targeting the same x86 high latency arithmetic instructions (DIV, IDIV, DIVSD, VDIVSD, MUL, IMUL and SQRTSD).

V. Experimental Results

A. VP Prediction Accuracy Study

The following simulations were done successively on a quad core system to study the prediction accuracy in variation to the number of entries in the table, table's associativity, and the history length.

First, we vary the number of entries E = {128, 256, 512, 1024, 2048}, associativity A = 4 and the number of results history length H = 4. In Fig. 3 the prediction accuracy is presented and on average the values are laying in the interval 77.2 % and 75.5 %. For half of the benchmarks, we achieved ~99 % accuracy, meaning that the values were predicted correctly in most cases. A high contributor to this achievement is the implementation of confidence counters, which feeds a predicted value only when the confidence is above the defined threshold. The optimal number of entries is E = 512.

Next, we study the influence of associativity variation on prediction accuracy, it is varied as follows A = {1,2, 4, 8}. As for the other parameters we chose the number of entries E = 512 and history length H = 4.

The results are shown in Fig. 4, and in comparison with the results achieved when we vary the number of entries (Fig. 3), we observe a decrease from ~1.7 % to 0.34 % in accuracy.

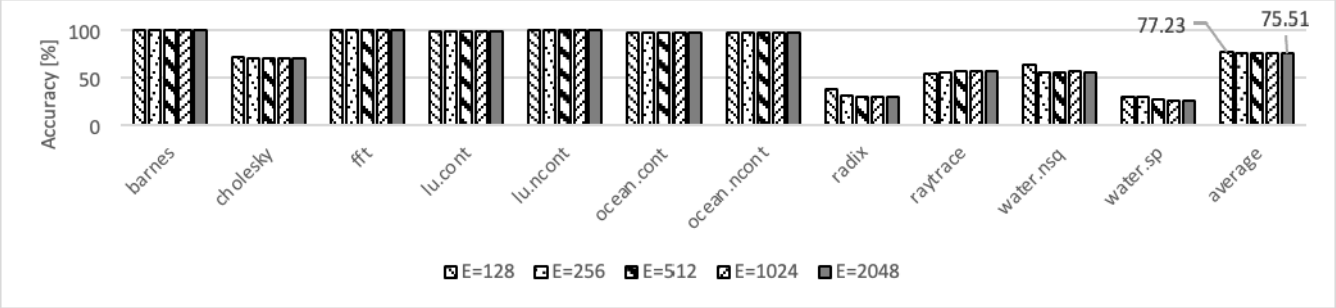

Figure 3. Averaged prediction accuracy per core for all benchmarks by varying number of entries (E = {128, 256, 512, 1024, 2048}; A = 4; H = 4; C = 4)

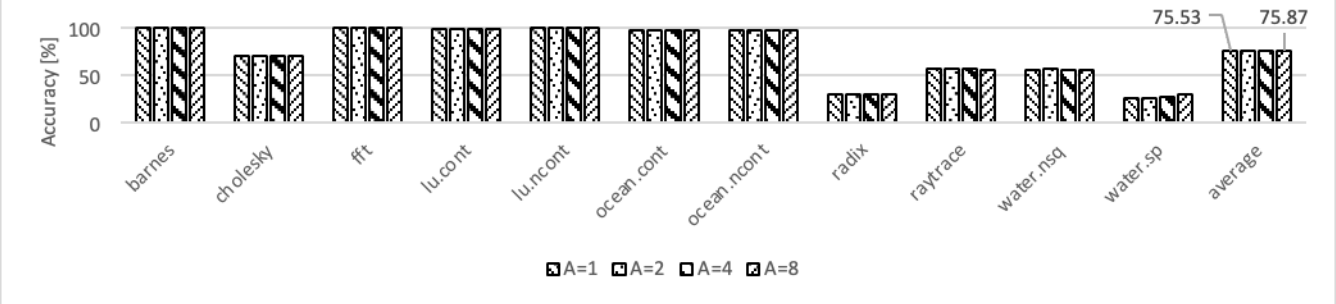

Figure 4. Averaged prediction accuracy per core for all benchmarks by varying the associativity (E = 512; A = {1,2, 4, 8}; H = 4; C = 4)

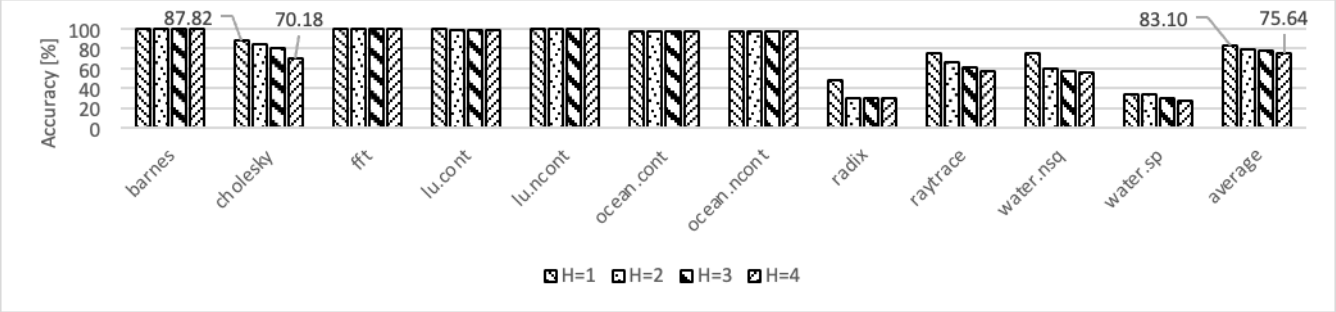

Figure 5. Average prediction accuracy per core for all benchmarks by varying the history length (E = 512; A = 4; H = {1, 2, 3, 4}; C = 4)

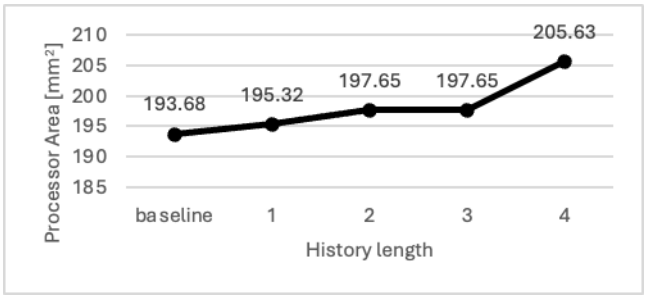

Figure 6. Area of integration in relation to the history length (E = 512; A = 4; H = {1, 2, 3, 4}; C = 4)

On average, the achieved results are looking stable, meaning that the associativity variation does not significantly impact the accuracy of prediction. For next simulations we chose the associativity to be 4, an associativity greater than 4 being uncommon.

Lastly, the variation of the number of result values which are kept for each instruction in the prediction table is studied. The VP table is configured as following E = 512 entries and an associativity of A = 4. We are scaling the length of history as follows: H = {1, 2, 3, 4}. On average, in Fig. 5 it is observable a decreasing trend from 83.10 % to 75.64 %. On half of the benchmarks the accuracy is not influenced by the history length variation. But on the "cholesky", "radix", "raytrace", "water.nsq" and "water.sp" benchmarks, we achieved a decrease in the accuracy along with the increase in history length. In this context, it looks like if more values are kept, the harder it gets to speculate with the correct value. Comparing the achieved accuracy with the ones measured on the previous result sets (number of entries and associativity variation) we can say that by varying the history length we achieved the biggest impact on the prediction accuracy.

The processor area integration in relation to the history length of the VP is visible in Fig. 6. It ranges from 139.68 mm^2 on the baseline configuration, up to 205.63 mm^2 for the configuration with a history length of four. One remark is in the case of history of two and three, the integration footprint being the same for both configurations. In case of a history of two, 180 bits are required to keep all the necessary information, thus rounding it up to the nearest power of two, we end up with a line of 256 bits (32 bytes). In case of a history of three, we need 238 bits to store the information and after the rounding we achieved 256 bits (32 bytes). In this case, a more realistic memory model is used. So, both configurations are having the same footprint for a VP table on each core, making the overall chip to have the same integration area. Overall, the integration area is increasing along with the number of history elements which are configured in the VP.

Further we can see the impact of the history length variation over the performance in Fig. 7, namely the relative core speedup. Interestingly, on average, the speedup is decreasing from 3.40 % to 3.25 %. The highest speedup achievement of 13.81 % was measured on the "lu.cont" benchmark on the configuration with a history length of 1. An interesting behavior is visible on the "cholesky"

benchmark. In the configuration where we are keeping only the last value produced for each instruction, we achieved a 0.51 % speedup. Increasing the number of values which are stored has a negative impact on the performance, decreasing up to -1.82 %. This happened because the prediction accuracy is also decreasing when we increase the number of values, from 87.82 % (H = 1) to 70.18 % (H = 4). The achievement of the negative value reflects the speculative execution drawback of the VP technique, making the system overall to perform worse than the baseline configuration. As a conclusion, the history length variation has small impact on the speedup, for most of the benchmarks we measured a positive speedup except for one.

The average dynamic power consumption of the processor in relation to the number of values stored for each instruction is presented in Fig. 8 for all benchmarks. On average, we measured a 34.39 W power consumption on the baseline configuration. In comparison to the baseline configuration, the power consumption overhead introduced by the predictor is quite small, on average ranging from 35.41 W for the configuration with a history length of 1, up to 35.48 W on the configuration with a history of 4. The highest power consumption of 64.84 W was measured on the "lu.cont" benchmark, the increase being justified by the fact that on this benchmark we achieved the highest performance increase of 13.81 %. Another notable increase correlated with performance increase, is visible on the "water.nsq" benchmark, with a power consumption of 47.63 % on the configuration with a history length of four. A reduction in power consumption is visible on the "cholesky" benchmark, on the maximal configuration (H = 4), this can be correlated with the fact that on this program we achieved a decrease in the overall performance (it takes longer time to execute the same program).

The energy consumption in relation to the number of result values which are kept for each instruction is summarized in Fig. 9 for all benchmarks. On average, the variation is quite small, with 8.97 J on the baseline configuration, and from 8.94 J when only one result value is stored, up to 8.99 J on the highest configuration.

As a conclusion, the history length variation did not influence in a significant way the overall energy consumption of the chip.

Regarding the chip's maximum temperature, we achieved its reduction by increasing the number of stored elements. The results are visible in Fig. 10, on average ranging from 57.8 °C, up to 56.17 °C. The biggest contributors to the decrease of chip temperature are due to the increase in integration area, stable energy consumption and minimal increase in the overall dynamic power consumption. One remark is for the configurations with the history of two and three, hence the predictors have the same integration area also the estimated chip temperatures are approximately the same, the trend is visible on all benchmarks.

B. Comparison of VP and DIR

This section presents a comparison between SHLA-DIR [1] vs. VP in fair conditions, to exploit the same value locality degree in a multicore environment. The DIR and the VP are both only using the last available result for an instruction. We showed in earlier simulations that the VP has the capability to store more than one result for one instruction by varying the history length (from 1 to 4), but for a fair comparison we chose to store only one value. Other parameters which are common for both schemes are chosen as follows: the number of entries is 512 and the associativity is set to 4.

A first comparison insight is given by the core and overall processor performance metrics. The first metric represents the average performance per core and the second metric the overall performance of the system. Fig. 11 summarizes the core performance and we can see that compared with the baseline configuration, in both cases we achieved a modest improvement. Thus, on average, cores that include an RB or a VP table are running faster. We can see that using the speculative VP technique, a higher IPC was achieved compared with the non-speculative DIR technique.

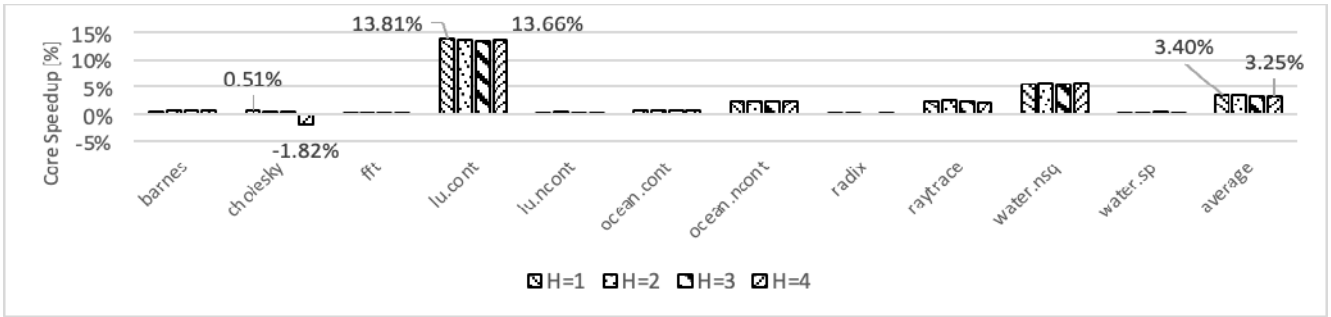

Figure 7. Relative core speedup for all bechmarks by varying the history length (E = 512; A = 4; H = {1, 2, 3, 4}; C = 4)

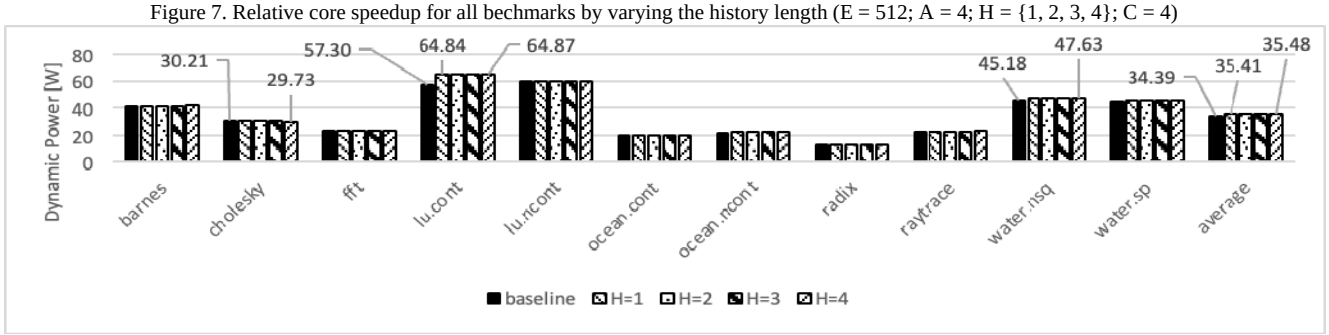

Figure 8. Average of processor dynamic power consumption for all benchmarks by varying the history length (E = 512; A = 4; H = {1, 2, 3, 4}; C = 4)

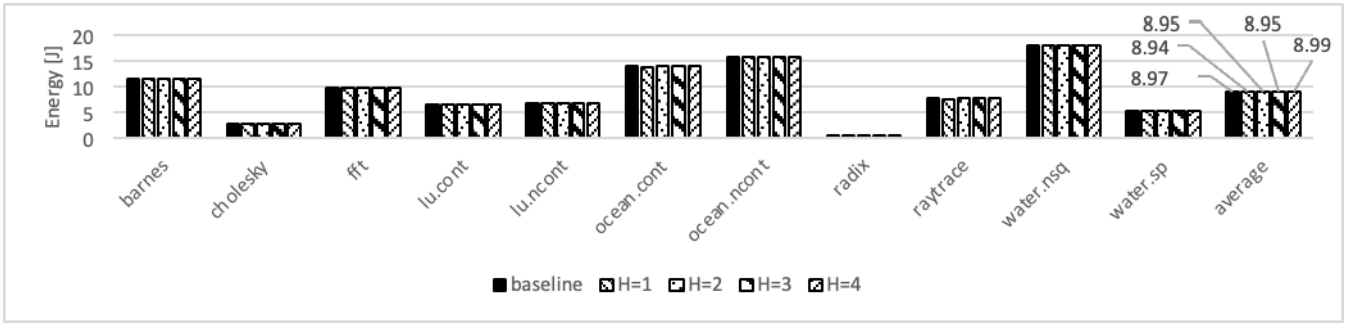

Figure 9. Average of processor energy consumption for all benchmarks by varying the history length (E = 512; AS = 4; H = {1, 2, 3, 4}; C = 4)

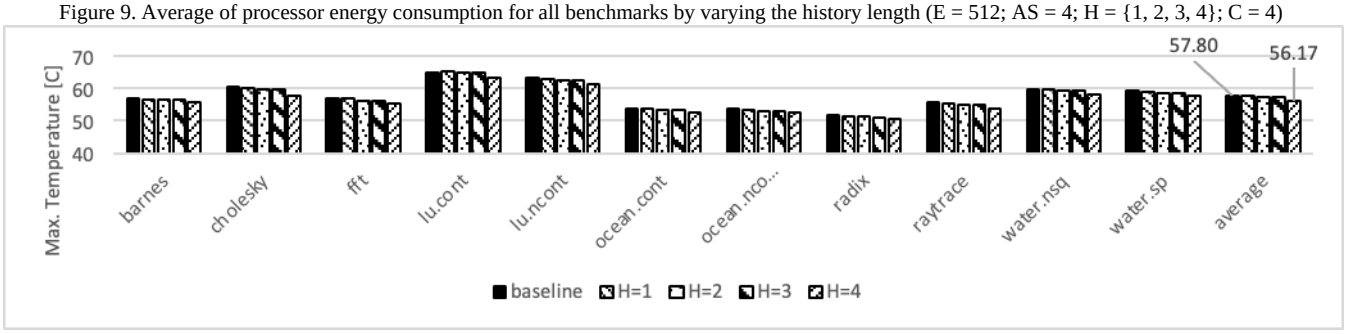

Figure 10. Maximum chip temperature for all benchmarks by varying the history length (E = 512; AS = 4; H = {1, 2, 3, 4}; C = 4)

A closer look at this metric is given by the relative speedup, which is computed in Fig. 13. Now we can see the difference as percentage. For both techniques the increase trend is visible for the simulated configuration ranging from 1 core up to 16 cores. Now, comparing the 16 vs 32 cores configurations we can see a small decrease in performance with the higher core number. It happens because of the scalability limits of the benchmarks. Adding more resources, beyond a certain limit, often leads to the situation where the communication interfaces and synchronization mechanisms between cores negatively impacts the overall performance. For this benchmark suite, the scalability sweet spot can be found in the 16-core configuration, where using an RB we achieved a 2.45 % speedup, whereas a 5.29 % increase in performance was difference between the two techniques has its roots in the achieved by using a VP.

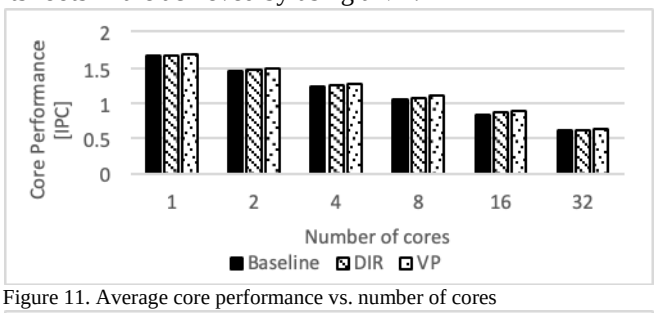

Figure 11. Average core performance vs. number of cores

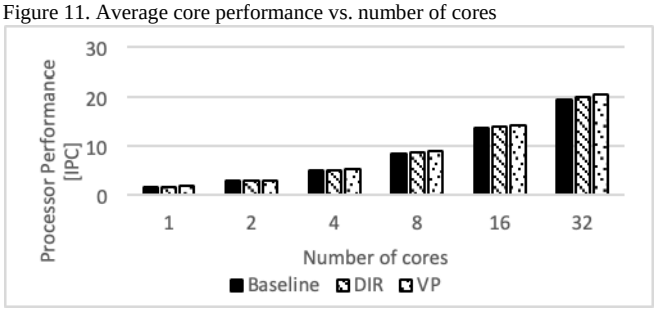

Figure 12. Average processor performance vs. number of cores

The main reason for this speedup is the fundamental differences between them, meaning that the DIR technique requires the values of the operands to check if the result can be reused (non-speculative). But the VP technique can provide a speculated value of the fetched instruction much earlier, it does not need to wait until the value of the operands are available.

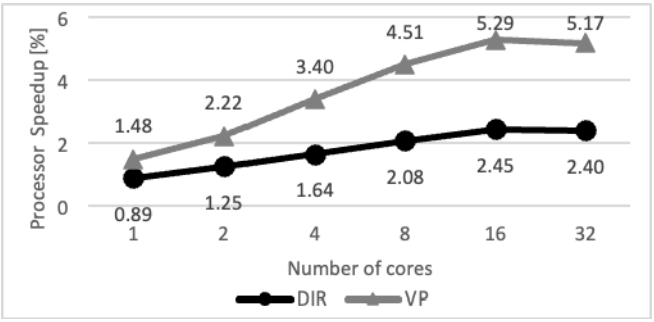

Figure 13. Average processor speedup vs. number of cores

The time required to fetch and compute the operand values is highly valuable in this scenario. Unlocking in a speculative way the execution of dependent instructions is a major advantage for the VP compared with the RB.

Regarding the processor area integration, we have summarized two perspectives. The first one represents the raw area numbers visible in Fig. 14. We can easily observe that the configurations which are including an RB have a bigger footprint than the ones which include a VP. In Fig. 15 we can see the differences between both configurations, with respect to the baseline configuration which does not include any architectural enhancement (DIR or VP). The functional unit of the DIR technique occupies around 2.9 mm2 spaces on each core, on the other hand the VP has as a footprint of only 0.412 mm^2, thus, making the VP unit to be more efficient in terms of integration area.

The processor's average dynamic power consumption in relation to the number of cores is visible in Fig. 16 and in Fig. 17 the percentage increase is calculated with respect to the baseline configuration. We can see that the power consumption increases along with the number of cores, because more hardware resources are used to run the same program and it well correlates with the processor

performance increase. Using the DIR technique, we achieved a lower increase in power consumption and performance. For the configurations which are using a prediction table the increase in power consumption is much higher, as well the performance. We can say that configurations with an RB consume less power, than the ones with a VP.

Interesting results were achieved on the energy consumption metric, depicted in Fig. 18 and Fig. 19. On most of the enhanced configurations we achieved, overall, a lower energy consumption. Although, the power consumption is increasing, the energy consumption is decreasing along with the number of cores. The highest difference is on the 32-core configuration using VP, we measured a value of -0.48 % less energy consumption compared with the baseline configuration. Based on the results we can say that the VP technique is more energy-efficient than the DIR.

Maximum chip temperatures and temperature reduction were plotted in Fig. 20 respectively Fig. 21. We can see that the trend is the same on all the simulations, configurations which include an RB unit achieved a bigger reduction in temperatures, compared with the ones with a VP. This happens because the integration areas of configurations which comprise an RB are higher than the ones with a VP. Using the RB unit, lower processor performances were achieved, similar energy consumption and the increase in integration area means that it has more space for the heat to dissipate.

All those factors are contributing to the lower temperature achievement using the DIR technique, compared with those which include an VP.

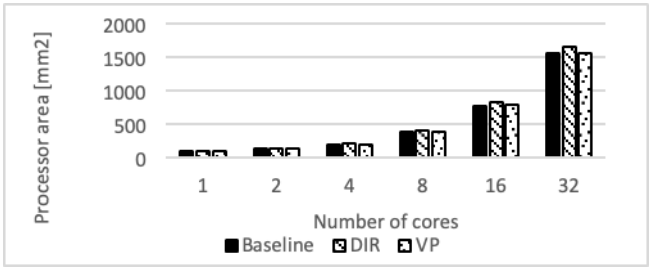

Figure 14. Processor area vs. number of cores

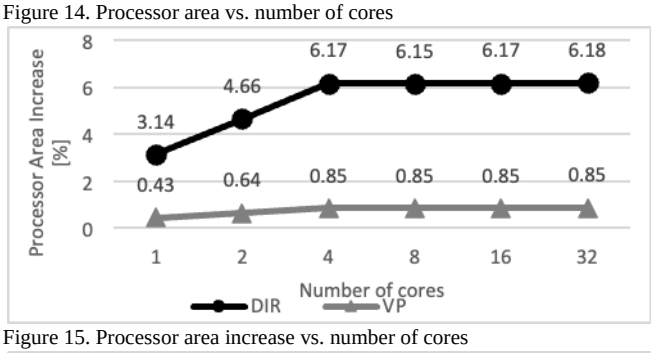

Figure 15. Processor area increase vs. number of cores

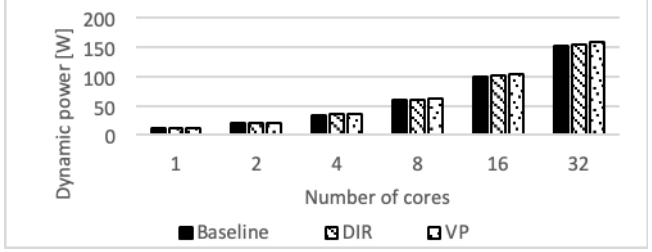

Figure 16. Average of processor dynamic power consumption vs. number of cores

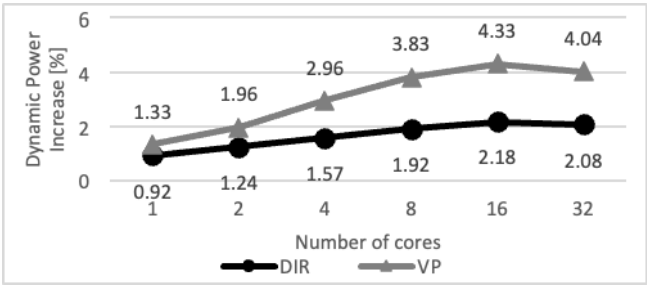

Figure 17. Average of processor power reduction vs. number of cores

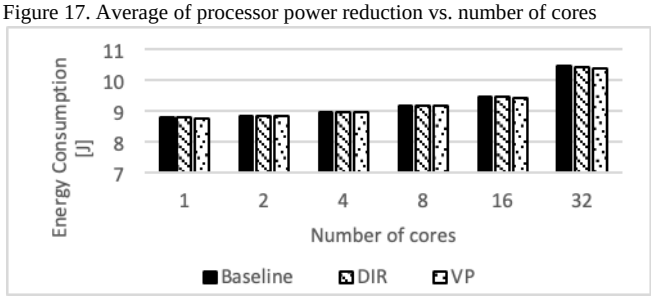

Figure 18. Processor energy vs. number of cores

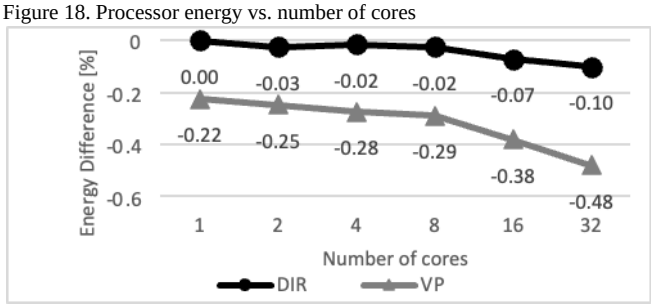

Figure 19. Processor energy reduction vs. number of cores

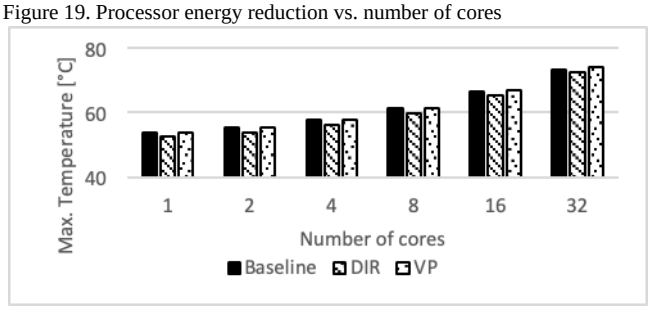

Figure 20. Maximum chip temperature vs. number of cores

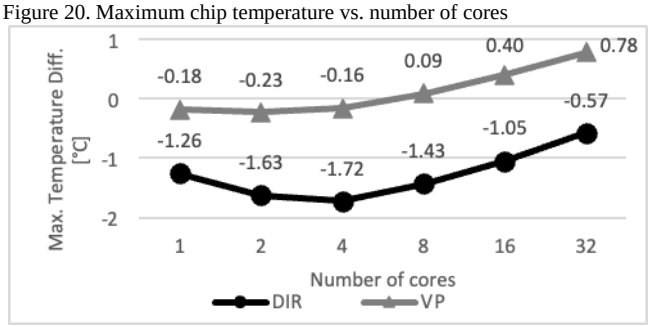

Figure 21. Maximum chip temperature vs. number of cores

VI. Conclusions and Further Work

In this work we show that the variation of the number of entries of the predictor has low impact on prediction accuracy. Changing the associativity seems to have neglectable impact on the accuracy in our tests. The highest impact on accuracy was achieved by varying the history length, more than that we measured an increase in performance, a reduction in chip temperature, lower energy consumption in most cases, as trade-off we observed an increase in power consumption and integration area.

Our original comparison between VP and DIR techniques, in fair conditions, is summarized in Table III.

We can conclude that the most suitable configuration depends on the needs of the user, the tradeoffs in each dimension are easily observable in the table. For example, in case one requires a high-performance configuration, then the VP unit can be chosen considering the following aspects: it has a smaller integration footprint in comparison with a DIR unit and will consume less energy despite the higher power consumption.

TABLE III. OVERVIEW OF THE COMPARISON DIR VS. VP

	Unit	
Metric (baseline difference)	**DIR**	**VP**
Speed-up	-	+
Area of integration	+	-
Power consumption	-	+
Energy reduction	+	-
Max. temperature	-	+

Further, we plan to use existing state-of-the-art multi-objective optimization methods and tools, for performing an automatic design space exploration to search for optimal configurations considering the trade-offs between the following metrics: integration area, chip temperature, processing performance, energy consumption and security. We plan to enhance our Framework for Automatic Design Space Exploration (FADSE) [38] with newer state-of-the-art optimization techniques and algorithms located in the Pareto-Fuzzy paradigm. We also plan to integrate the modern benchmarking suite Splash-4 [39] into the Sniper simulator and to model multiple complex context-based VP schemes.

CONFLICT OF INTEREST

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

PUBLISHER'S NOTE

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editor and the reviewers. Any statements, claims, performances and results are not guaranteed or endorsed by the publisher.

REFERENCES

[1] C. Buduleci, A. Gellert, and A. Florea, "Selective high-latency arithmetic instruction reuse in multicore processors," in 2023 27th International Conference on System Theory, Control and Computing (ICSTCC), Timisoara, Romania: IEEE, Oct. 2023, pp. 410–415. doi:10.1109/ICSTCC59206.2023.10308483

[2] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," ACM Trans. Archit. Code Optim., vol. 11, no. 3, pp. 1–25, Oct. 2014. doi:10.1145/2629677

[3] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, Bangalore: IEEE, Jan. 2010, pp. 1–12. doi:10.1109/HPCA.2010.5416635

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," SIGARCH Comput. Archit. News, vol. 39, no. 2, pp. 1–7, May 2011. doi:10.1145/2024716.2024718

[5] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown." arXiv, Jan. 03, 2018. Accessed: Feb. 26, 2024. [Online]. Available: http://arxiv.org/abs/1801.01207

[6] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," 2018. doi:10.48550/ARXIV.1801.01203

[7] R. Sheikh, R. Cammarota, and W. Ruan, "Value prediction for security (VPsec): Countering fault attacks in modern microprocessors," in 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC: IEEE, Apr. 2018, pp. 235–238. doi:10.1109/HST.2018.8383922

[8] S. Deng and J. Szefer, "New predictor-based attacks in processors," in 2021 58th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA: IEEE, Dec. 2021, pp. 697–702. doi:10.1109/DAC18074.2021.9586089

[9] M. M. K. Martin, D. J. Sorin, H. W. Cain, M. D. Hill, and M. H. Lipasti, "Correctly implementing value prediction in microprocessors that support multithreading or multiprocessing," in Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34, Austin, TX, USA: IEEE Comput. Soc, 2001, pp. 328–337. doi:10.1109/MICRO.2001.991130

[10] L. Widgen and E. Sowadsky, "Operand cache addressed by the instruction address for reducing latency of read instruction," U.S. Patent US5919256A, Jul. 06, 1999

[11] F. Gabbay and A. Mendelson, "System and method for concurrent processing," U.S. Patent US5996060A, Nov. 30, 1999

[12] M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value prediction," in Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29, Paris, France: IEEE Comput. Soc. Press, 1996, pp. 226–237. doi:10.1109/MICRO.1996.566464

[13] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," SIGPLAN Not., vol. 31, no. 9, pp. 138–147, Sep. 1996. doi:10.1145/248209.237173

[14] Y. Sazeides and J. E. Smith, "The predictability of data values," in Proceedings of 30th Annual International Symposium on Microarchitecture, Research Triangle Park, NC, USA: IEEE Comput. Soc, 1997, pp. 248–258. doi:10.1109/MICRO.1997.645815

[15] F. Gabbay and A. Mendelson, "Speculative execution based on value prediction," Technion - Israel Institute of Technology, EE Department TR 1080, 1996

[16] L. N. Vintan, A. Florea, and A. Gellert, "Focalising dynamic value prediction to CPU's context," IEE Proc., Comput. Digit. Tech., vol. 152, no. 4, p. 473, 2005. doi:10.1049/ip-cdt:20045090

[17] L. Yang, L. Huang, R. Yan, N. Xiao, S. Ma, L. Shen, and W. Xu, "Stride equality prediction for value speculation," IEEE Comput. Arch. Lett., vol. 21, no. 2, pp. 57–60, Jul. 2022. doi:10.1109/LCA.2022.3195411

[18] A. Seznec, "Exploring value prediction with the eves predictor," in 1st Championship Value Prediction, Los Angeles, CA, USA, Jun. 2018

[19] B. Goeman, H. Vandierendonck, and K. De Bosschere, "Differential FCM: increasing value prediction accuracy by improving table usage efficiency," in Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, Monterrey, Mexico: IEEE Comput. Soc, 2001, pp. 207–216. doi:10.1109/HPCA.2001.903264

[20] N. Deshmukh, S. Verma, P. Agrawal, B. Panda, and M. Chaudhuri, "DFCM++: Augmenting DFCM with early update and data dependence-driven value estimation," in 1st Championship Value Prediction, Los Angeles, CA, USA, Jun. 2018

[21] A. Perais and A. Seznec, "Practical data value speculation for future high-end processors," in 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), Orlando, FL, USA: IEEE, Feb. 2014, pp. 428–439. doi:10.1109/HPCA.2014.6835952

[22] Y. Ishii, "Context-base computational value prediction with value compression," in 1st Championship Value Prediction, Los Angeles, CA, USA, Jun. 2018

[23] K. Koizumi, K. Hiraki, and M. Inaba, "H3VP: History based highly reliable hybrid value predictor," in 1st Championship Value Prediction, Los Angeles, CA, USA, Jun. 2018

[24] R. Sheikh, H. W. Cain, and R. Damodaran, "Load value prediction via path-based address prediction: avoiding mispredictions due to conflicting stores," in Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge Massachusetts: ACM, Oct. 2017, pp. 423–435. doi:10.1145/3123939.3123951

[25] A. Gellert, “Advanced prediction methods integrated into speculative computer architecture,” PhD Thesis, “Lucian Blaga” University of Sibiu, Computer Science Department, Sibiu, 2008

[26] A. Gellert, Beyond the limits of modern processors. Bucharest Matrix Rom, 2008

[27] A. Gellert, A. Florea, and L. Vintan, “Exploiting selective instruction reuse and value prediction in a superscalar architecture,” Journal of Systems Architecture, vol. 55, no. 3, pp. 188–195, Mar. 2009. doi:10.1016/j.sysarc.2008.11.002

[28] A. Gellert, G. Palermo, V. Zaccaria, A. Florea, L. Vintan, and C. Silvano, “Energy-performance design space exploration in SMT architectures exploiting selective load value predictions,” in 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), Dresden: IEEE, Mar. 2010, pp. 271–274. doi:10.1109/DATE.2010.5457197

[29] A. Gellert, H. Calborean, L. Vintan, and A. Florea, “Multi-objective optimisations for a superscalar architecture with selective value prediction,” IET Comput. Digit. Tech., vol. 6, no. 4, p. 205, 2012. doi:10.1049/iet-cdt.2011.0116.

[30] Gellert, A. Florea, U. Fiore, P. Zanetti, and L. Vintan, “Performance and energy optimisation in CPUs through fuzzy knowledge representation,” Information Sciences, vol. 476, pp. 375–391, Feb. 2019. doi:10.1016/j.ins.2018.03.029

[31] A. Gellert, “Prediction-based modeling and estimation in advanced computing systems,” Habilitation Thesis, “Lucian Blaga” University of Sibiu, Sibiu, 2023

[32] A. Gellert and L. Vintan, “A multicore architecture with selective load value prediction,” Proceedings of The Romanian Academy, Series A: Mathematics, Physics, Technical Sciences, Information Science, vol. 19, no. 4, pp. 597–604, 2018

[33] A. Gellert, M. Vintan, and L. Vintan, “Perceptron-based selective load value prediction in a multicore architecture,” Romanian Journal of Information Science and Technology, vol. 22, no. 3–4, pp. 215–227, 2019

[34] C. Buduleci, A. Gellert, A. Florea, and A. Matei, “Extending sniper with support to access operand values: A case study on reusability measurement,” in 2022 23rd International Carpathian Control Conference (ICCC), Sinaia, Romania: IEEE, May 2022, pp. 70–75. doi:10.1109/ICCC54292.2022.9805869

[35] A. Fog, “The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers,” Technical University of Denmark, Nov. 2022

[36] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), New York, NY, USA, 2009, pp. 469-480

[37] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in Proceedings 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Italy: ACM, 1995, pp. 24–36. doi:10.1109/ISCA.1995.524546

[38] H. Calborean, “Multi-objective optimization of advanced computer architectures using domain-knowledge,” PhD Thesis, “Lucian Blaga” University of Sibiu, Sibiu, 2011

[39] E. J. Gomez-Hernandez, J. M. Cebrian, S. Kaxiras, and A. Ros, “Splash-4: A modern benchmark suite with lock-free constructs,” in 2022 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA: IEEE, Nov. 2022, pp. 51–64. doi:10.1109/IISWC55918.2022.00015