μOp-Based Value Prediction: a New Approach Enabling Efficient Implementation

Layan Jarjoura

μOp-Based Value Prediction: a New Approach Enabling Efficient Implementation

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering

Layan Jarjoura

Submitted to the Senate
of the Technion — Israel Institute of Technology
Adar bet 5784 Haifa March 2024

This research was carried out under the supervision of Prof. Freddy Gabbay and Prof. Avi Mendlson, in the Faculty of Electrical and Computer Engineering.

The author of this thesis states that the research, including the collection, processing and presentation of data, addressing and comparing to previous research, etc., was done entirely in an honest way, as expected from scientific research that is conducted according to the ethical standards of the academic world. Also, reporting the research and its results in this thesis was done in an honest and complete manner, according to the same standards.

Acknowledgements

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

Instruction-Level Parallelism (ILP) plays a significant role in the design of modern processors. For decades, ILP was considered inherently limited by true data dependencies. The concept of Value prediction (VP) was introduced to speculatively break true data dependencies, thus, allowing Out-of-Order (OoO) processors to achieve higher ILP and gain performance.

To the best of our knowledge, VP has not been implemented in commercial chips since its performance gains do not justify its implementation costs. In this work, a new performance-efficient implementation of VP is proposed. It predicts the destination values of instructions using two phases: a training phase where predictions are trained using a value predictor; and a deployment phase, where out of those predictions, it deploys the high-confidence ones. Importantly, in the case of a prediction having high confidence, the training of the corresponding instruction is stopped. Only when an instruction is found to have been predicted incorrectly, the system is moved back to the training phase.

In our design, the training phase is not part of the OoO pipeline so the constraints on it can be relaxed. This circumvents stressing the CPU front-end which is generally an issue that VP designs face. In the deployment phase, only high-confidence predictions are employed avoiding costly accesses for all other instructions that are either not eligible for prediction or have low confidence, after which prediction validation is performed. This phase is integrated in the pipeline thus it is designed to be simple. In addition, we enable VP for micro-operation (μOp) granularity. Since instructions' addresses in a CISC CPU do not stay the same after decoding them into RISC instructions, typical value predictors become invalid for μOp predictions. Our architecture leverages the micro-operation cache to overcome this problem and enable VP for μOps.

This paves the way for practical adaptations of VP in high-performance processors.

In our experiments, we ran spec2017 benchmarks in addition to EEMBC CoreMark benchmark on sniper x86-simulator. We present results for a 4-issue and an 8-issue superscalar processor augmented with our VP scheme with a stride predictor. The 8-issue processor with VP has, on average, a 2.56% higher instruction-per-cycle (IPC) than its baseline counterpart without VP, on all benchmark tests; the 4-issue processor has, on average, 2.23% higher IPC than its baseline counterpart without VP. To explain low speedup results in the face of high prediction accuracy for specific benchmarks, we

conducted dependency distance analyses and examined them to justify the results.

Chapter 1

Introduction

Modern microprocessors are expected to extract the parallelism at run time out of the process' instruction stream without violating the sequential correctness of the execution. This parallelism is termed extractable ILP and it aims at improving the performance of the processors. Instructions within the sequential program cannot always be eligible for parallel execution due to several constraints. These constraints can be classified into three groups: true-data dependencies, name dependencies, and control dependencies [Joh91]. After performing register renaming, only control and true-data dependencies remain pertinent during run-time.

Control dependencies and name dependencies are not considered an upper bound on the parallelism extractable from a sequential program, since they can be handled or even eliminated by various hardware and software techniques. For example, in order to tolerate the effect of control dependencies, many processors also support speculative execution. This technique allows the processor to continue executing control-dependent instructions before resolving the branch outcome. To maintain the correctness of the program, a speculatively-executed instruction can only retire if the prediction it relies upon was validated; otherwise, it is discarded. Contrarily to name- and control dependencies, only true-data dependencies were thought to reflect the serial nature of a program by dictating in which sequence data should be passed between instructions [Wal91].

To overcome true-data dependencies, and thus improve ILP and single-thread performance, the concept of Value Prediction was proposed by Gabbay et al. [GM96], [GM98b] and Lipatsi et al. [LWS96], independently, in the '90s.

Value Prediction is based on the idea of value locality [LWS96]. Value locality implies that a given dynamic instruction is likely to produce a value that it has previously produced. This suggests that by predicting the value of a value-producing instruction, the consumer instructions can start executing speculatively even before the producer has finished executing.

Value Prediction assumes that there is a connection between different values the code produces and that we can predict these relations at run-time. This allows depen-

dent instructions to issue earlier than previously possible by using predicted operands, and thus uncovers more ILP. Yet, predictions must be verified to ensure correctness. This scheme is termed as Speculative Execution Based on VP.

Several types of predictors were proposed in the literature since VP was introduced. Their common role is aiming at predicting the destination values of instructions. Value predictors can be divided into two main groups according to [SS97]: (1) Computational predictors which predict the next value by performing an operation on previous values, e.g. Last Value predictor [LWS96] which performs the trivial identity operation on the previous value. (2) Context Based predictors which predict values based on previously observed patterns by matching recent value history (context) with previous value history, e.g. FCM predictor [SS97] which identifies patterns in the output of a given static instruction and predicts according to them.

Prediction confidence mechanisms are usually needed in prediction schemes such as in Branch Prediction to avoid costly mispredictions when possible. In Value Prediction, a confidence counter per prediction in a value predictor is usually needed to ensure a high percentage of accurate predictions. High accuracy is important since it was shown that given an enhanced confidence mechanism, simple recovery mechanisms with costly misprediction penalties can be tolerated [PS14a].

Despite Value Prediction being a technique with potential, it experiences drawbacks. It suffers from a trade-off between the performance gains brought by a correct prediction and the high cost of recovering from a misprediction using simple mechanisms. This makes high coverage mostly irrelevant in the presence of low accuracy. Fortunately, it was shown in [PS14a] that with high accuracy, it is possible to achieve performance gains with a simple flushing mechanism, and more complex recovery mechanisms are not needed. In addition, most predictors (e.g., FCM) require either the use of very small tables to reduce access time or give up predicting tight loops [PS14a]. This need for small prediction tables in predictor designs is another challenge of efficient VP implementation.

This work proposes a new architecture for VP that aims at reducing true-data dependencies and as a result, increase ILP, while mitigating the abovementioned challenges.

Our implementation consists of two paths: A non-critical path to train the predictor; and a critical path to deploy predictions with high confidence.

In the non-critical path, training is performed. Only the eligible instructions get allocated an entry in the value predictor, and they remain there only until they gain enough confidence after which they get moved to a Value Renaming Table (VRT) for deployment. This results in avoiding unnecessary predictor accesses and capacity issues in the predictor tables. Most recent works suggesting VP implementation schemes apply their training in the fetch stage which requires multi-port support, fast look-ups, and large predictor tables [She19], [Per15], [PS14b]. In our case, on the other hand, the non-critical path is not part of the pipeline, which circumvents stressing the CPU

front-end.

Another major contribution of this work is enabling of the use of VP for μOp granularity. Since most predictors are accessed using the instruction PC, they are not suitable to predict μOps values. Our architecture leverages the μOp Cache to enable VP for μOps.

Our critical path performs a simple deployment scheme augmented in the pipeline. Only high-confidence predictions are employed avoiding costly look-ups for all other instructions that are either not eligible or have low confidence. Those predictions are stored in the VRT - which is an extension of the ROB - in the entries of the eligible instructions. The ROB hosts the predicted values and every dependent instruction gets fed with the relevant predicted value.

In previous studies [PS14c], [PS15], [PS14b], look-ups of predicted values were from Value Predictors while in our architecture we look up instead from the VRT. The value predictor is used only in the training stage outside the pipeline. When the confidence of the predicted value is high enough, the value graduates to the VRT for faster deployment and to make room for other instructions in the predictor to get trained which eliminates the need for a large predictor structure. Also, whenever we have a value in VRT, we stop training its associated instructions and thus save power and space in the predictor table. Works like [PS15] suggest new, complex predictors to address the storage requirements. Instead, we chose in our experiments a simple predictor but made the allocation of entries in them selective: only eligible μOps get allocated an entry. BeBoP, the architecture proposed in [PS15], applies the training path in the CPUs front-end. In BeBoP, for training, the predictor is accessed during the fetch stage which stresses an already-expensive stage in the pipeline and requires fast look-ups. On the contrary, our training phase has much weaker constraints than the pipeline, avoiding both of these issues.

Our contributions in this paper are as follows:

1. We propose a new approach of handling Value Prediction as part of an OoO core, where the training path is separated from the critical deployment path.

2. We present an architecture which allows the deployment of VP for μ-operations in CISC processors.

3. We design a VP scheme that is orthogonal to the mispredictions recovery mechanism. This enables the use of simple recovery mechanisms.

4. We provide a study of different design parameters and their effect on the performance as well as an evaluation of the presented architecture across multiple benchmarks.

The remainder of the thesis is organized as follows: Chapter 2 discusses related work in value predictors and Value Prediction schemes. In chapter 3, we present the

fundamentals of our proposed Value Prediction scheme. Chapter 4 dives into the micro-architecture of our design. Chapter 5 explains our methodology and describes our evaluation framework and baseline. In chapter 6, we present our findings and results. Lastly, we discuss and conclude the paper in chapter 7 and suggest future research ideas.

Chapter 2

Previous Work

In this chapter, we provide background information about predictors and confidence mechanisms in section 2.2, and a literature overview over Value Prediction schemes and architectures in section 2.1, in addition to μOp Cache preview in section 2.3.

2.1 Value Prediction Schemes

Value Prediction is a technique that aims at speculatively breaking true-data dependencies and improving ILP by predicting destination values of value-producing instructions based on pattern recognition and value locality.

Researchers have studied VP a lot in the past [Per21], [Per15], [PES16], [PS14b], [PS15], [Per21]. The following works have focused on implementation details of VP: EOLE [PS14b] proposed implementation of VP without adding extra read ports to the register file for validation. Its architecture achieves that by early-executing some instructions in the front-end using dedicated execution units when all inputs are available, in addition to late-executing other instructions, and validating all predictions in the in-order back-end. BeBoP [PS15] is a block-based prediction scheme that builds upon EOLE and allows multiple instructions to be predicted in a single access by aggregating all predictions of μOps associated with a single block to be stored in one predictor entry. FVP [BGS$^+$20] focused value prediction as a mechanism to achieve an early execution of critical loads that frequently create performance bottlenecks in the OoO processor so as to manage the value predictor size. Recently, Perais [Per21] proposed targeted VP that leverages register renaming optimizations to reduce hardware complexity, while still preserving most of the gains from VP. The main idea in [Per21] is carefully choosing a very limited (targeted) pool of instructions or distinct values to predict.

Similar to BeBoP, our architecture also works in basic blocks granularity. But unlike BeBoP, we do not use complex predictors which are known to have hardware and complexity overheads. Instead, we chose a simple predictor - the Stride predictor - but made the allocation of entries in it selective, where only eligible μOps get allocated,

to manage table sizes. Regarding the implementation, BeBoP trains its predictor in the CPUs front-end, and the predictor is accessed during the fetch stage which adds stress to an expensive stage in the pipeline and requires fast look-ups. On the contrary, we apply our training phase outside of the pipeline, and design a simple deployment phase as part of the pipeline, avoiding both issues.

Many works have focused on studying the criticality of certain classes of instructions to be value predicted and finding the instructions with the highest gain-to-cost ratio: [Per21] suggests targeting distinct values (like 0x0 and 0x1) and encoding physical register names or small distinct values in register names. [SL19] demonstrated that a combination of value history with branch history is a better predictor of values than either of them used individually. Additionally, this work categorized instructions into different classes and identified three classes that are best to be targeted or prioritized for VP: load instructions, address producing instructions and high fan-out instructions.

In our experiments, we chose to predict register loads and simple arithmetic μOps like ADD and SUB. Our careful choice of predicting loads aligns with the conclusion in [SL19] that load instructions and address-producing instructions are among the critical instruction classes to value-predict. Simple arithmetic instructions that can be predicted using Stride predictors agree with the concept of value locality in general and with the conclusions in [Per21] in particular.

2.2 Value Predictors and Confidence Counters

Several types of predictors were proposed in the literature since VP came to light. Their common role is aiming at predicting the destination values of instructions. Value predictors can be divided into two main groups according to [SS97]:

1. Computational predictors: perform an operation on previous values to yield predicted next values. Examples include Stride Value predictor [GM96] which adds a delta to a previous value, and Last Value predictor [LWS96] which performs the trivial identity operation on the previous value.

2. Context-based predictors: match recent value history (context) with previous value history and predict values based entirely on previously observed patterns. For example, the FCM predictor [SS97] identifies patterns in the output of a given static instruction and predicts according to them; the VTAGE predictor [PS14c] relies on global branch history to provide more accurate predictions; the D-VTAGE predictor [PS15] incorporates stride prediction into VTAGE; the EVES predictor [Sez18] optimizes the coverage and area of D-VTAGE by using smart allocation schemes; the DLVP pedictor [SCD17] predicts addresses based on context to prefetch load values from the data cache. These values are then used for value prediction of corresponding load instructions; the Composite predictor [She19] combines EVES and DLVP to further enhance the coverage of VP.

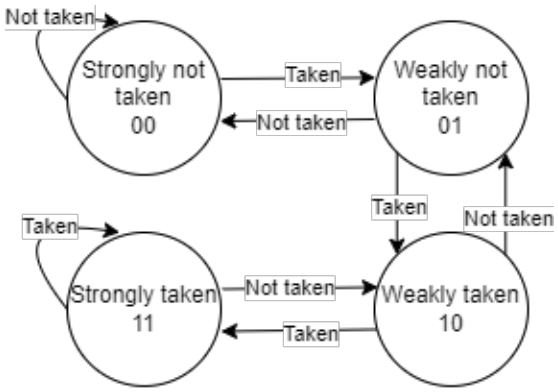


Figure 2.1: Saturating 2-bit counter

Predictors usually incorporate prediction confidence mechanisms. Prediction confidence mechanisms are usually needed in prediction schemes such as in Branch Prediction to increase accuracy. One way to track confidence is using a two-bit saturating counter [Smi81]. This is an automaton consisting of four states that store the history of a branch. The counter is updated every time the branch is executed, as shown in figure 2.1. The outcome of a branch is predicted according to the 2-bit counter value: taken whenever the upper bit is 1, and not taken otherwise. A more complex confidence counter is the forward probabilistic counter [RZ06]. With probabilistic updates, we can increment and decrement the counter by a fractional value. For example, if we set the value of the decrement to be 1/4, this gives the counter a 25% chance of being decremented on negative feedback.

In this study, we compare a baseline CPU without Value Prediction to our suggested VP implementation with a Stride Value Predictor. Our choice is motivated by using the simplest hardware with the smallest overhead while still keeping the desired performance gains. The predictor incorporates a simple three-bit confidence counter for accuracy purposes.

2.3 Micro-Operation Cache

As mentioned in section 2.1, our architecture works in basic blocks to support VP for μOps. It does that with the help of the μOp Cache.

The μOp Cache was introduced in [SMO+01] as an n-associative cache that helps reduce CPU front-end power consumption as well as improve performance. It was later implemented in Intel Pentium 4 industrial processors [HSU+01]. The μOp Cache is a special-purpose cache that dynamically collects and stores already decoded instructions. In the pipeline, it is placed after the x86 decoding stages; so, by storing decoded instructions in it, the whole decode stage can be shut down for significant periods of time while the rest of the execution engine proceeds its work. When a hit in this cache occurs, instructions do not need to be decoded again and can be fed into the

pipeline directly from the cache. Only on a cache miss, the microarchitecture fetches and decodes instructions from the Level 2 cache. The already-decoded μOps from the IA-32 Instruction Decoder are assembled or built - by the μOp Cache - into program-ordered sequences of μOps called traces. The μOps are packed into groups of six μOps per trace line. There can be many trace lines in a single trace. These traces consist of μOps running sequentially down the predicted path of the IA-32 program execution. This allows both the target of a branch and the branch itself to be included in the same trace cache line, even if they are thousands of bytes apart in the program.

Working with BBs means that μOp-Caches map the block to its trace lines according to the address of the BB's first instruction, and the precise address of the μOp is lost. Since the traditional way of accessing a predictor is through the PC of an instruction which is lost after decoding, it is a challenge to implement VP in μOp granularity. However, our architecture overcomes this disadvantage and enables coexistence between Value Prediction and μOp-Caches. Note that our scheme can also be tweaked to get employed in processors that do not have μOp Caches.

Chapter 3

uOp-Based Value prediction Fundamentals

In this chapter, we explain the fundamentals of our idea and the main entities that enable its implementation, then we go over the general prediction flows.

This work builds upon the basic five phases which constitute super-scalar processors' pipeline: Fetch, Decode, Execute, Memory, and Commit. The integration of our proposed architecture within the basic pipeline is shown in figure 3.1. The three orange blocks are the new VP-related added blocks.

The architecture focuses on the decode and commit stages. Two new units are designed and used in this study: the Value Predictor (VP), and the Value Renaming Table (VRT). In addition, the commit stage has been augmented with a prediction validation mechanism. The micro-architectural considerations for both the VP and the VRT are detailed in subsection 4.3.7. The μOp Cache in the CPU's front-end is also leveraged to enable Value Prediction in μOp granularity.

Current modern processors come close to executing as fast as true dependencies allow. The particular dependencies that constrain execution speed constitute the critical path of execution. As can be seen in figure 3.1, our proposed architecture consists of two phases: a training phase which employs the Value Predictor to train predictions outside the critical path of the pipeline; and a deployment phase which employs the VRT in the pipe to deploy confident predictions. The training path can be executed with weaker constraints, i.e. slower clocks, since it lies outside the critical path; and the deployment path is designed to be simple to avoid complexity charges and overheads in the critical path.

3.1 Value Prediction Phases

In this section, we go over the fundamentals of both phases that consist the Value Prediction scheme proposed in this work: the training phase, and the deployment phase.

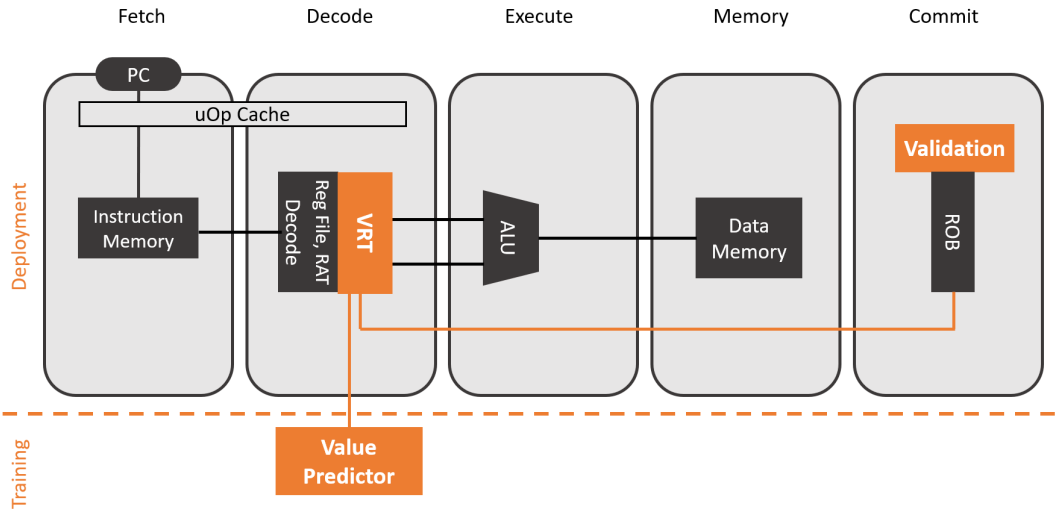


Figure 3.1: The proposed architecture

3.1.1 The Training Phase

Value-generating instructions are defined as instructions that write a value to a destination register, such as ADD, SUB, or LOAD instructions. Upon encountering a new eligible-for-training value-generating μOp, an entry is allocated for it in the VP, marking the start of its value prediction training. A confidence counter is attached to it and initialized to zero at this point. The eligibility criteria that dictates which instructions are eligible for training can be found in section 4.3.3.

On every future encounter of that μOp, the counter is updated: increases when the predicted value is correct, and reset to zero otherwise. When the counter saturates, this indicates that a prediction is ready to be deployed, and its entry gets evicted from the VP and allocated into the VRT. This confidence mechanism ensures less deployed mispredictions which saves precious misprediction penalty cycles. Removing the entries with the high confidence allows for more μOps to start their training without having to enlarge the VP's table size.

The VP can be one of many types as discussed in chapter 1. In this study, we experiment with and provide performance results for a stride predictor.

3.1.2 The Deployment Phase

The confident predictions that get allocated in the VRT according to the scheme explained in subsection 3.1.1 are ready to be speculatively deployed. The μOps that are encountered from now on, and depend on the producer instruction which value is held by the VRT, can use this value and start executing right away, without waiting for the graduation of the producer. The predicted value needs to be validated. In the commit stage of the producer μOp, the real value is sampled and compared to the predicted one. If it matches, the pipeline continues its flow normally. If it does not match, the pipeline is flushed to the last checkpoint which is the latest committed instruction in

the ROB, and the relevant entry in the VRT is emptied which means that the producer μOp is now available for retraining.

The VRT is located in the frontend of the pipeline. It aims at limiting the predictor's training time and lookup power, which enables us to store the predicted values in the predictor only during the training phase and not throughout the life-cycle of the instruction. The VRT also eliminates the need for large predictors since predictions with high confidence get evicted from the predictor and into the VRT.

Chapter 4

Micro-Architecture

In this chapter, we go into a detailed description of the proposed micro-architecture. We explain the flows of each phase in the Value Prediction scheme, how the VP and VRT look-ups are carried out, and eventually analyze the micro-architectural considerations.

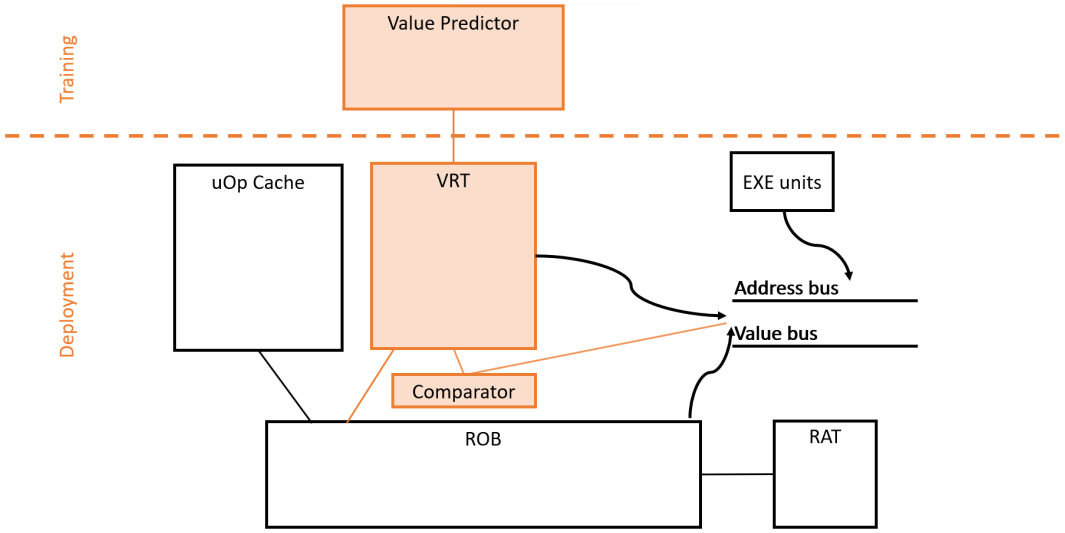


Figure 4.1: Block diagram of the micro-architecture

Figure 4.1 describes the high-level blocks which the micro-architecture consists of. The blocks in orange are value prediction-related. The Value Predictor serves the training phase by collecting predictions until they gain enough confidence to be deployed. The VRT in the deployment phase stores confident predictions to be deployed. That way, the predictor and the VRT will never be storing values for the same instruction concurrently. After an instruction is executed in an execution unit, the unit broadcasts (forwards) the calculated result and the instruction's address to the Address and Value busses where they can be snooped. The verification of predictions is carried out by comparing calculated values snooped from the busses to the confidently predicted values stored in the VRT.

4.1 The Training Phase

In the training phase, the predictor is trained to predict μOp results of value-producing instructions. The training flow is described in figure 4.2 as follows:

1. Eligible, decoded instructions of a BB get fetched from the μOp-Cache to start their training stage.

2. An entry is allocated in the VP for the eligible, value-generating instruction according to its BB's PC, with its confidence counter initialized to zero.

3. After commit stage, the VP snoops on the busses to get the real result of the instruction and compare it to the value it is storing; or alternatively, adding the value to the VP if it has not yet stored any value for the instruction.

4. This comparison occurs on every encounter of the instruction and the confidence counter and/or stored value are updated every time according to the comparison result. When the counter saturates, the corresponding entry is evicted from the VP and into the VRT. The entry is also evicted if the counter does not saturate after a predefined amount of time to avoid impractical VP stores.

Note that the value predictor can be of many types, and as a consequence, the fields described in figure 4.2 might differ from one predictor to another. In the experiments presented in this work, we evaluated our architecture on a stride predictor [GM96].

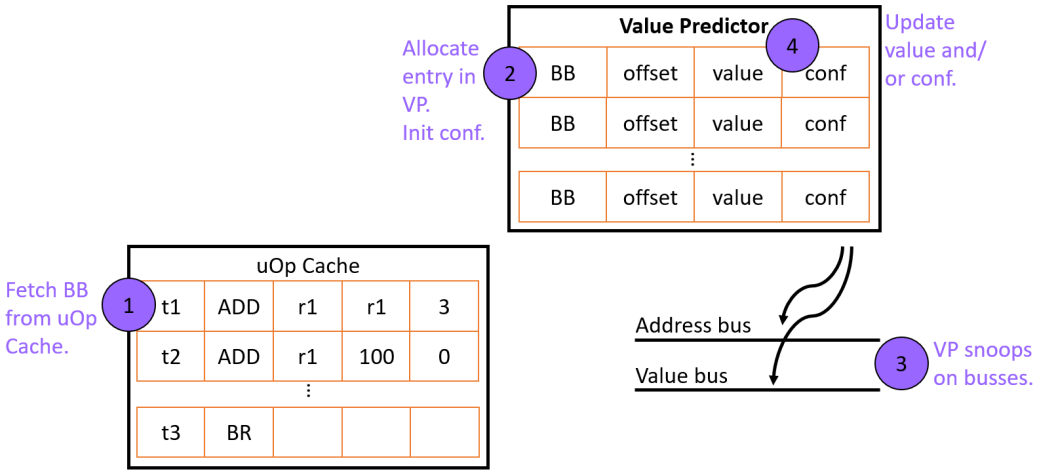


Figure 4.2: The micro-architecture of the training phase

4.2 The Deployment Phase

In the deployment phase, the predictions of high confidence are deployed to improve ILP and save CPU execution time. This is carried out by allowing dependant instructions

to start executing speculatively using the predicted value of the producer instruction. The flow of this phase is described in figure 4.3 as follows:

0. The VRT has confident predictions stored that have been allocated from the VP. Registers that we want to predict in a value-generating instruction will have a special "speculative" bit (symbol * in figure 4.3) indicating the eligibility of the μOp for value prediction.

1. After Register Renaming, a lookup in the VRT is made. If the μOp is found (VRT hit), Value Renaming is performed (connecting r1 with d1). We define Value Renaming as the action of connecting the destination registers stored in the RAT to the predicted values stored in the VRT.

2. The fetched BB is allocated into the RoB with the corresponding predicted values from VRT/RAT to be used speculatively for consequent dependant μOps. If we get a VRT miss, i.e. μOp doesn't exist in VRT, this means there is no confident prediction for this μOp, and the μOp goes back to the training phase.

3. The operations are executed as they traditionally are in x86 processors to get the real results.

4. The execution units forward and broadcast the real result on the busses.

5. For the prediction validation, the ROB snoops on the busses and gets the calculated values and compares them to the predicted values it has, in order. If they match, everything continues as is; if not, we flush to the last checkpoint which is the latest committed instruction in the ROB. ROB needs comparator units, and it needs to signal to the VRT in case of a flush so that the VRT evicts the mispredicted entry.

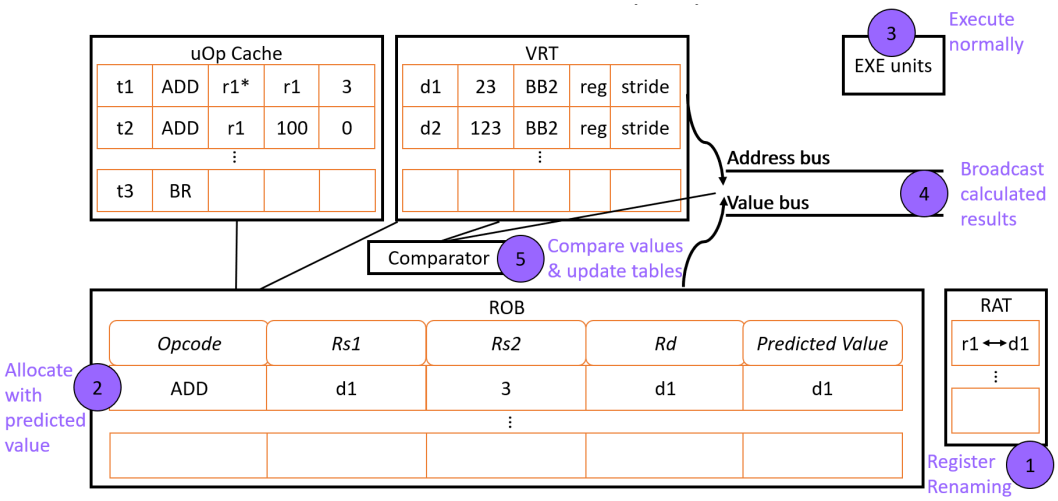


Figure 4.3: The micro-architecture of the deployment phase

4.3 Micro-architectural Considerations

4.3.1 Micro-architecture of VRT

The structure of the VRT is similar to the structure of a μOp Cache, meaning it is an n-associative cache. In our experiments, we picked a fixed size table where n is 1024 with 4 ways in each set. Each way is called an entry and has 8 bytes, thus our VRT is 32K bytes in size. We encountered zero evictions due to conflicts or lack of space.

Each entry consists of a valid bit, LRU counter, PC of the μOp's BB, and a field that stores the predicted value. VRT is accessed simply via the PC of the BB's first instruction.

Works like [YYX05] show that only diminishing improvements can be obtained as predictor table size increase over 2K entries; or 39K bytes in their experiments. In our architecture, we can afford 4K entries since it amounts to 32K bytes in total and- as a positive byproduct- gives the impression of an unbounded predictor.

4.3.2 Micro-architecture of VP

In this work, a Stride Predictor [GM96] was applied in the suggested architecture. It detects stride patterns in the destination values of a particular instruction and predicts the next values according to them. Note that the stride can be zero as well. The value predictor was implemented as a 4-way associative matrix: 2^{10} sets, 4 ways each; 4K entries, or 32KB in total. This predictor was chosen due to its simplicity and low hardware overhead. μOps get allocated in their entries according to the PC of their BB's first instruction which makes look-ups easy and renders history- and context-related look-up keys unnecessary. Quantitative performance results of our architecture with a stride predictor can be found in Chapter 6.

4.3.3 Eligibility Criteria

As for the eligibility criteria, we chose specific types of value-producing μOps to be predicted: loads to registers (MOV, LEA), and simple arithmetic operations (ADD, and SUB). As [SL19] states, load instructions and address-producing instructions are among the critical instruction classes to value-predict, which motivated our choice of predicting loads to registers. Choosing to predict ADD and SUB instructions was motivated by the concept of value locality in general and the conclusions in [Per21] in particular.

We chose to predict one eligible μOp in each BB to avoid keeping track of precise addresses of μOps, but our architecture can be modified to predict multiple μOps in every BB.

4.3.4 Eviction Process

Eviction of an entry from the VP can happen for two reasons: (1) when a prediction reaches a saturated confidence in the VP, the entry is moved to the VRT and invalidated in the VP. (2) when allocating a new entry to the VP and a collision is faced, an entry can be evicted according to the Least Recently Used (LRU) scheme and the new prediction takes its place.

Eviction of an entry from the VRT can occur for two reasons as well: (1) when a new prediction gets allocated into the VRT and a collision is faced, the entry is evicted according to the LRU scheme and the new prediction gets stored in its place. (2) on mispredictions: when validating a predicted value and it's found to be incorrect, the entry is evicted from the VRT immediately, and the corresponding instruction is marked as eligible-for-prediction again.

4.3.5 Misprediction Recovery Mechanism

We employed a flushing mechanism for mispredictions. When a value is mispredicted, its entry gets evicted from the VRT and its corresponding μOp goes back to being eligible for prediction. All while the misprediction penalty is applied (μOps flushed to the last checkpoint, i.e., the latest committed instruction in the ROB) and the μOp gets re-executed in a traditional way.

Note that our architecture is orthogonal to the recovery mechanism. It can work with any preexistent branch prediction recovery mechanism found in the system.

4.3.6 Prediction Confidence Saturation

A 3-bit saturating counter was chosen for each entry in the predictor.

The confidence counter in an entry gets initialized to zero whenever a new entry is allocated into the VP. With each new encounter of the same instruction, the confidence counter increases by 1 if the stored predicted value matches the calculated real one or gets reset to zero whenever an incorrect prediction with low confidence is found in the VP. The counter is considered "saturated" if it reaches the value of 7 ('111' in binary). After multiple encounters, if the confidence counter saturates, the prediction is called a confident prediction, and its corresponding entry is evicted from the VP and allocated into the VRT instead.

As shown in [GKMP98] and [CRT99], a threshold of 15 gives the best results for 4-bit counters. However, a threshold of 7 for 4-bit counters resulted in good gains as well and we decided - for our proof of concept - to go for a 3-bit counter that requires 8 successive correct predictions before considering the prediction confident and moving it to the VRT for deployment. As [CRT99] found, for conservative misprediction recovery mechanisms, the counter needs to quickly turn off once it starts mispredicting before it can cause too much damage. Thus motivating our choice to reset the counter on every encounter of an incorrect prediction.

4.3.7 Instruction-Fetch Rate

[GM98a] shows the effect the instruction window has on VP gains: when increasing the effective instruction-fetch rate, the speedup gained by value prediction grows significantly. They also showed that if the distance between two dependant instruction exceeds a certain limit, the prediction becomes useless since the input value becomes ready on time regardless of the prediction. In our experiments, we chose dispatch width of 4 for the first set of experiments and dispatch width of 8 for the second set. In both sets, the window size was 128.

Chapter 5

Evaluation Methodology

We evaluate our proposed VP architecture on the x86 Sniper simulator [CHE11]. In Sniper, we use Intel's Nehalem micro-architecture as our baseline. We ran all of our experiments on both a 4-issue wide pipeline and an 8-issue wide one. The benchmarks were traced by PIN 3.7 trace generator simulator. It is worth noting that our architecture can be adapted to any general-purpose ISA.

Our baseline is the Nehalem micro-architecture with a width of 4 in the first set of experiments, or a width of 8 in the second. The instruction fetch window is 128 in both sets. In all of the experiments, we integrate our proposed micro-architecture described in chapter 4 into Nehalem and evaluate the results by comparing them against the baseline that's stripped of VP. We use a stride predictor as a proof of concept but other predictor types can be used instead as well.

We use a subset of the SPEC2k17 CPU benchmark suite [BLvK18] in addition to EEMBC's CoreMark [PCLGO09] to evaluate our contributions. Specifically, out of SPEC2k17, we use 8 integer benchmarks and 4 floating-point programs. We compile speed source code of the SPEC2k17 applications and EEMBC to Intel x86 ISA binaries using gcc 7.4.0. with -O3 level optimization.

Table 5.1 shows the SPEC2k17 benchmarks subset. We simulate the tests in two steps: first, we warm up all structures like caches and branch predictors for 1000 billion instructions; and then simulate 10 billion instructions for statistics gathering. For short-running benchmarks (i.e., EEMBC) - we simulate from the start of the benchmark until it completes.

#	Benchmark	Type	Application Area
1	perlbench	Integer	Perl interpreter
2	mcf	Integer	Route planning
3	cactuBSSN	FP	Physics: relativity
4	omnetpp	Integer	Discrete Event simulation - computer network
5	xalancbmk	Integer	XML to HTML conversion via XSLT
6	x264	Integer	Video compression
7	deepsjeng	Integer	Artificial Intelligence: alpha-beta tree search (Chess)
8	imagick	FP	Image manipulation
9	leela	Integer	Artificial Intelligence: Monte Carlo tree search (Go)
10	nab	FP	Molecular dynamics
11	exchange2	Integer	Artificial Intelligence: recursive solution generator (Sudoku)
12	fotonik3d	FP	Computational Electromagnetics (CEM)

Table 5.1: SPEC2k17 benchmarks used in our experiments.

Chapter 6

Experimental Results

We ran two sets of experiments: the first one with a pipeline width of 4; the second one with a pipeline width of 8. Each set was conducted twice: once assuming a zero-cycle mispredictions penalty; and once with different positive penalties. Speedup results of all experiments and their analyses are displayed and explained in the following sections.

6.1 First set: pipeline width 4, zero-cycle penalty

6.1.1 General Speedup Results

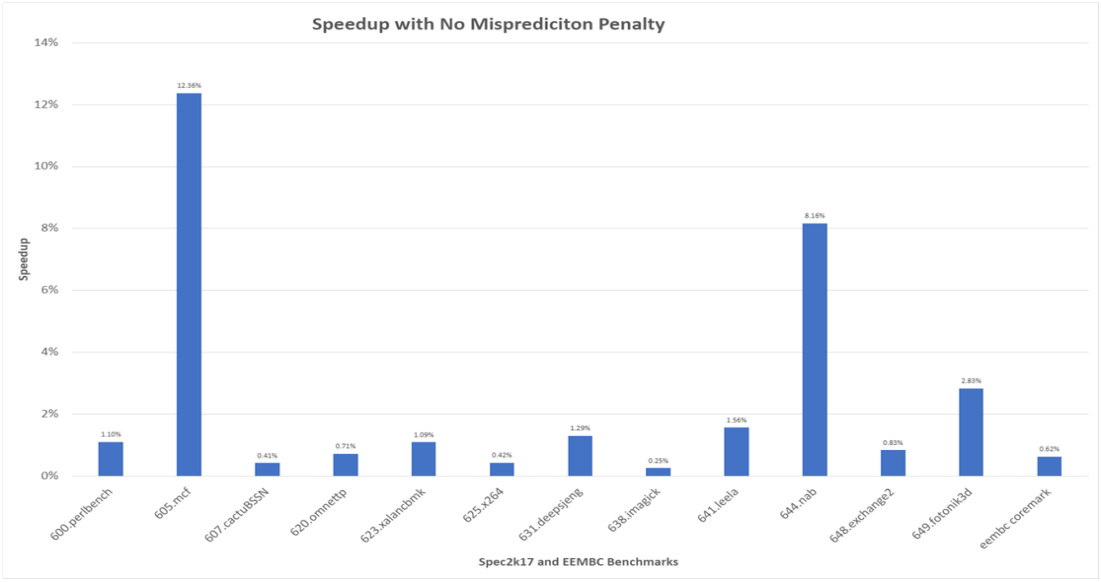
In the first set of experiments, we examine the potential of our proposed architecture in a zero-cycle misprediction penalty setting with a pipeline width of 4. We show and justify the speedup results achieved across all 13 benchmarks.

Figure 6.1a shows the speedup achieved when augmenting our VP scheme to the baseline architecture of width 4. The average (arithmetic mean) speedup of the 13 benchmarks is 2.43%. We can see that benchmarks "mcf" and "nab" obtained the most substantial improvement: 12.36% and 8.16% speedup respectively. The geomean speedup achieved across all benchmarks is 1.20%.

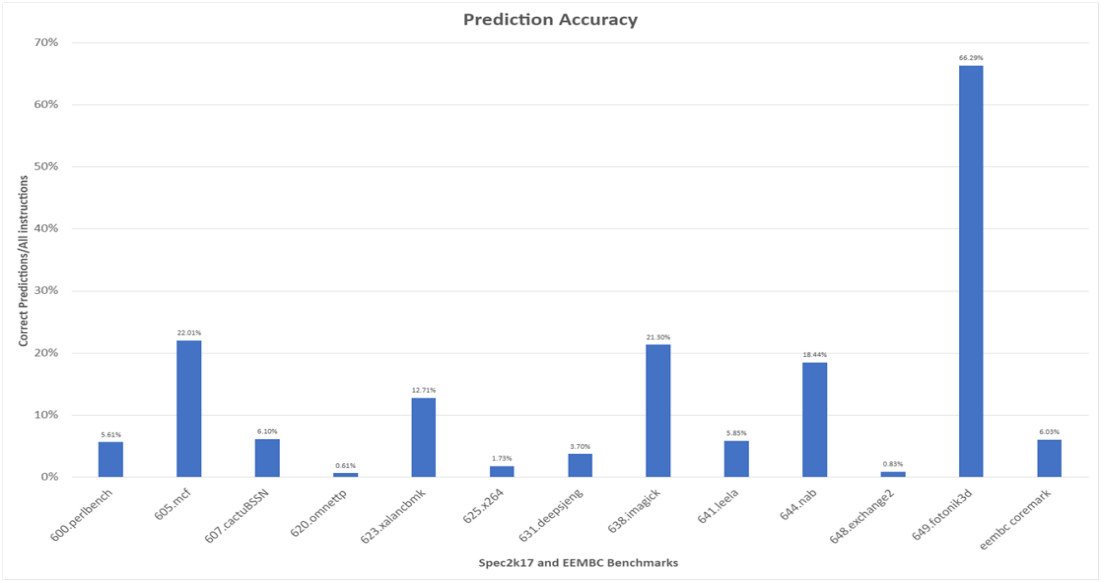
To understand the results better, we now take a look at the fraction of correctly predicted instructions out of all instructions in each test. This is plotted in figure 6.1b. Intuitively, the bigger the fraction of correctly predicted instructions, the better we expect the speedup to be. We can clearly see that "mcf" and "nab" have noticeable prediction accuracy which explains their high speedup gains.

However, we also notice that the two benchmarks "imagick" and "fotonik3d" have relatively high prediction accuracy while we saw in figure 6.1a that their gains are relatively low. The lower percentages of correct predictions for all other benchmarks explain their lower speedups.

In the next subsections 6.1.2 and 6.1.3, we focus on "imagick" and "fotonik3d" benchmarks to explain their speedup results.



(a)



(b)

Figure 6.1: Width 4, 0-cycle penalty. (a) Speedup with No Misprediction Penalty. (b) Percentage of Correctly Predicted Instructions Out of All Instructions

6.1.2 ”imagick” Results

Now, in this section, we attempt to justify the low gains of ”imagick” despite its high prediction accuracy.

But first, we define the term ”Minimal Producer Distance”: An instruction in the ROB can depend on multiple other instructions (producers) in the ROB. ”Producer Distance of an Instruction” is the number of instructions in the ROB between that instruction and its producer. ”Minimal Producer Distance” is the minimal distance between an instruction and its producers in the ROB. An instruction having a minimal producer distance of 0 means that it does not depend on any instructions preceding it. For example, if instruction X has 2 producers in the ROB: Y and Z. Let's assume that the distance between X and Y is 6, and the distance between X and Z is 2. Then, the minimal producer distance for X is 2. Note that even if we predict the parent with distance 6, the instruction is still dependent on the parent with the smaller distance which means it still needs to wait for the closer parent to finish executing before it can start executing itself because the parent with the larger distance is closer up in the ROB's queue. As a consequence, predicting the producer with the larger distance without predicting the one with the smaller distance will render the prediction useless in terms of speeding up the child instruction's execution.

For ”imagick”, we ran a dependency distance analysis whose result is shown in Figure 6.2a. We can see that a large part of the test - 49.47% of the benchmark's instructions - have at least one parent with distance 1 that they depend on.

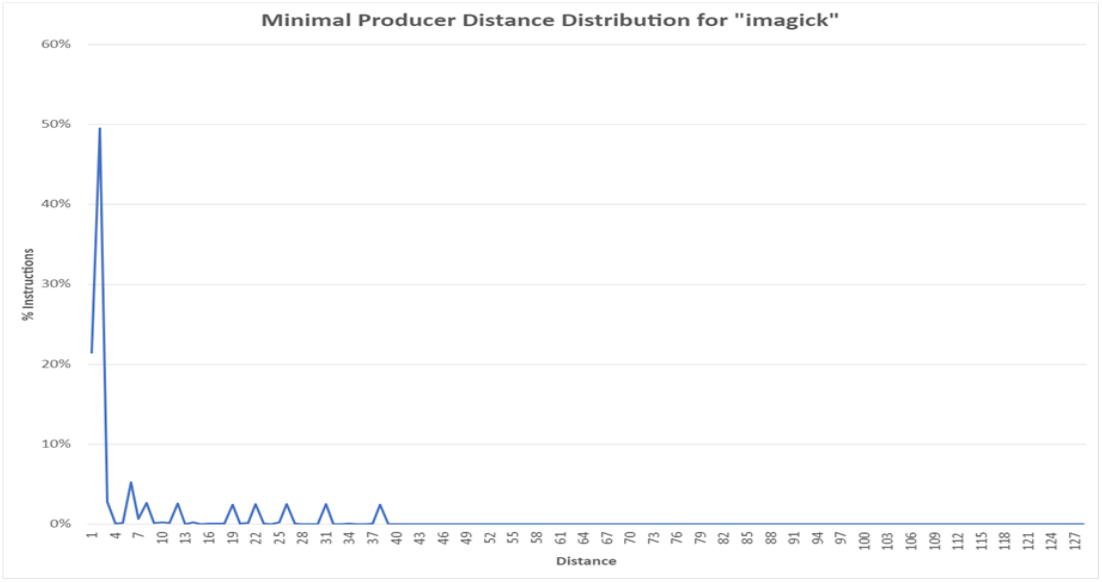
Figure 6.2b, however, shows that only 13.80% of the correctly predicted instructions have a distance of 1. That is, only 2.70% of the test's instructions had a producer distance of 1 in addition to being predicted correctly and *usefully*. The rest of the predictions with distance 1 are ineffectual.

We can see two other spikes in figure 6.2b: distance 2 with 33.13% and distance 37 with 37.43%. However, according to figure 6.2a, only 2.79% of the benchmark has a minimal distance of 2 and only 2.47% of the test has a minimal distance of 37. This indicates minimal effect on the speedup caused by predictions of instructions with distance 2 and 37. In conclusion, a smaller amount of predictions contributed to speedup gains than the relatively high prediction accuracy achieved for this test might wrongly lead one to believe at first glance.

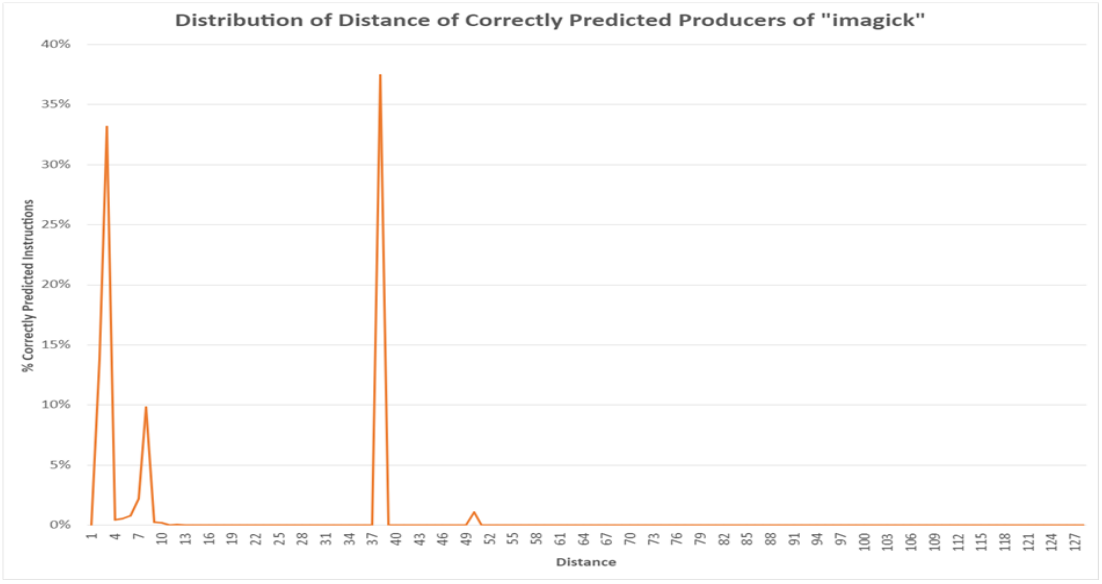
6.1.3 ”fotonik3d” Results

To justify ”fotonik3d” benchmark's results, we also ran a dependency analysis and analyzed it in a similar manner to what was previously shown in section 6.1.2.

Figure 6.3a presents ”fotonik3d”'s minimal producer distance analysis results. We see that 31.91% of the test's instructions have a minimal producer distance of 1. The rest of the distances seen in figure 6.3a have minimal effect on the final speedup due to their uncommonness.



(a)



(b)

Figure 6.2: ”imagick” benchmark. (a) Distribution of the imagick’s Instructions’ Minimal Producer Distance. (b) Distribution of imagick’s Correctly Predicted Producers’ Distances

However, similar to what we previously did for the ”imagick” benchmark, we take a look at figure 6.3b for ”fotonik3d” and see that 31.76% of all correctly predicted instructions in the test have a producer distance of 1. That is, 23.08% of the test's instructions have a producer distance of 1 as well as managed to be correctly and *usefully* predicted. This explains the fair speedup of 2.83% shown in figure 6.1a, unlike the deceiving high prediction accuracy shown in figure 6.1b for ”fotonik3d” benchmark.

6.2 First set: pipeline width 4, positive penalty

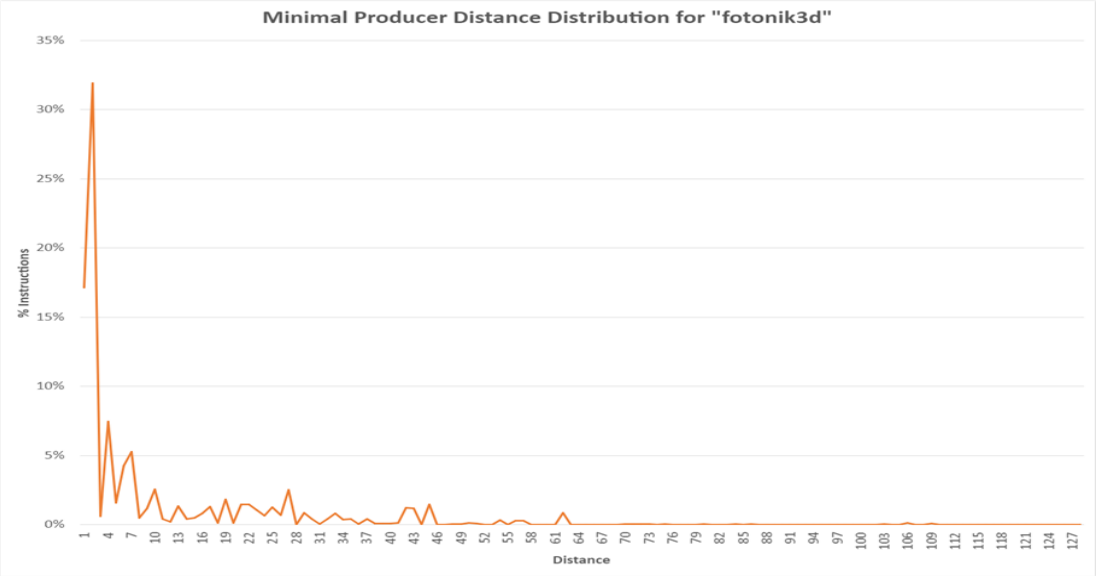
To take misprediction penalty into consideration and better estimate realistic results for the experiments with width of 4, we measured the number of mispredictions (i.e., incorrectly predicted instructions) and calculated the effect of their penalty on the speedup that was achieved when the penalty was set to zero in our simulated VP architecture.

To simulate a flushing mechanism, we calculated 4 possible constants as penalty: a penalty of 5, 10, 15, and 20 clock cycles per misprediction. The results are shown in figure 6.4a.

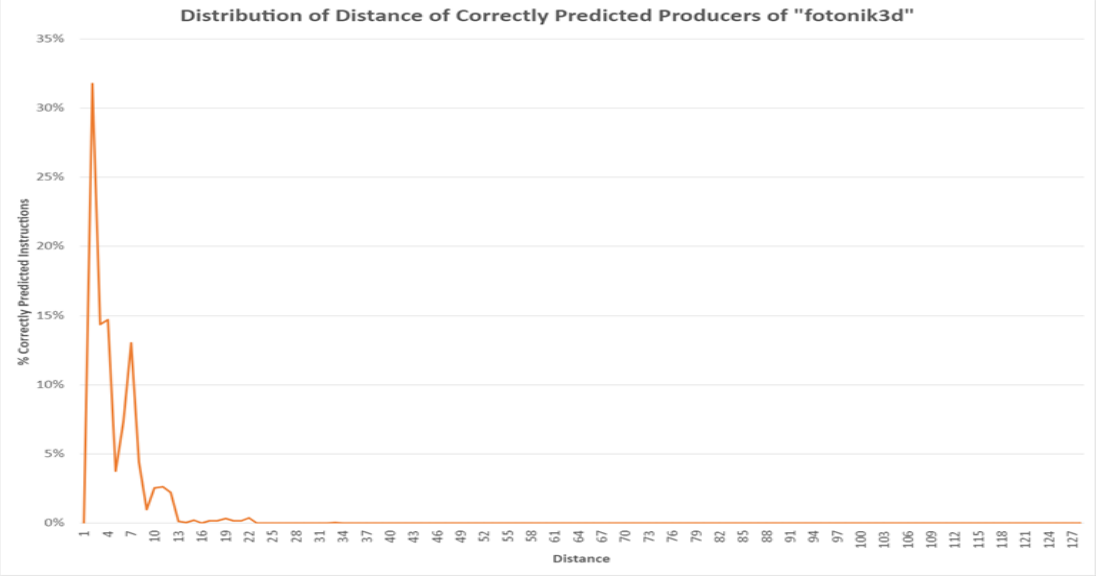
We notice benchmarks ”exchange2” and ”EEMBC CoreMark” have the largest IPC degradation percentages. Since the IPC degradation in this case is due to incorrect predictions causing recovery mechanism penalty cycles, we look at the misprediction percentage out of all the instructions in each test. This misprediction percentage is shown in figure 6.4b. Indeed, ”exchange2” and ”EEMBC CoreMark” have the highest misprediction percentage, excluding ”mcf”, justifying their larger decrement in IPC. The speedup gains brought by the high percentage of correct predictions in ”mcf” seems to compensate for the misprediction percentage shown in figure 6.4b. The lower misprediction percentages for all other tests explains their lower IPC degradation shown in figure 6.4a.

Note that IPC degradation corresponds to speedup degradation since we have the same instruction count and frequency across benchmarks and experiments.

We also calculated the final speedup given a penalty of 5, 10, 15, and 20 clock cycles per misprediction. Figure 6.4c shows the speedup given the different misprediction penalties. As expected, the gains are smaller when penalty is added, and sometimes we get negative speedup (degradation) as well due to incorrect predictions, but ”mcf” and ”nab” still show significant improvement over the baseline even with maximum penalty of 20 cycles per misprediction.

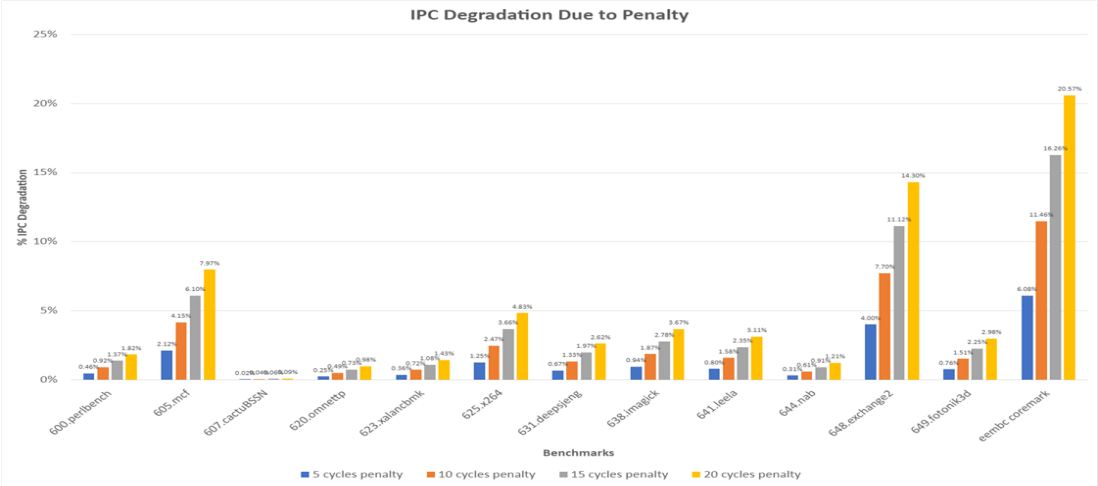


(a)

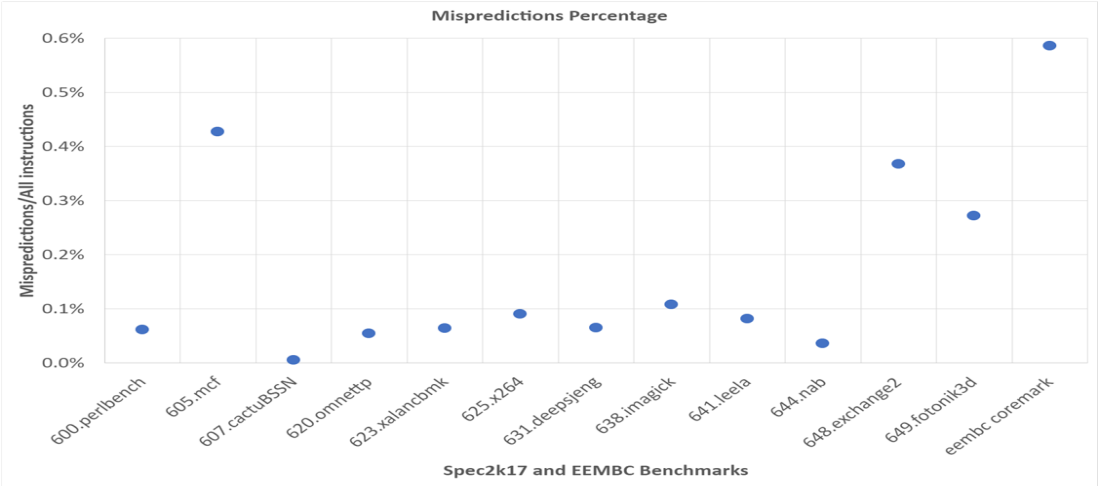


(b)

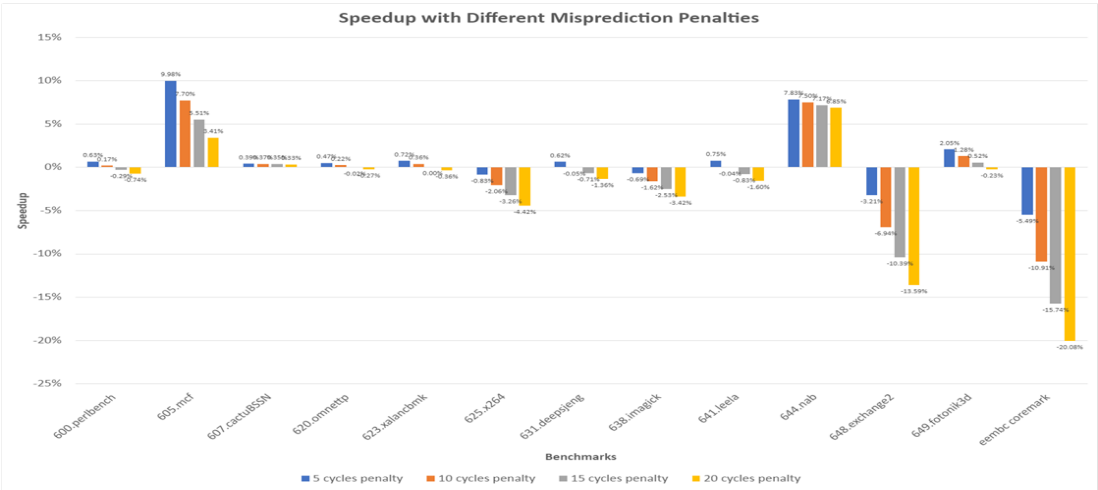
Figure 6.3: "fotonik3d" benchmark. (a) Distribution of the fotonik3d's Instructions' Minimal Producer Distance. (b) Distribution of fotonik3d's Correctly Predicted Producers' Distances



(a)



(b)



(c)

Figure 6.4: Width 4, positive-cycle penalty. (a) IPC Degradation Due to Penalty of 5, 10, 15, and 20 Clock Cycles. (b) Misprediction Fraction Out of All Instructions. (c) Speedup with a Misprediction Penalty of 5, 10, 15, and 20 Clock Cycles

6.3 Second set: pipeline width 8, zero-cycle penalty

6.3.1 General Speedup Results

In the second set of experiments, we are interested in the effects of a different pipeline width on the results. In order to look into that, we examine the potential of our proposed architecture in a zero-cycle misprediction penalty setting with a pipeline width of 8 - to differentiate from the first set on experiments conducted on a pipeline width of 4.

Figure 6.5a shows the speedup gained using our μOp-Based Value Prediction scheme with pipeline width of 8. The average (arithmetic mean) speedup of the 13 benchmarks is 2.56%. We can see clearly in the graph that the "mcf" and "nab" benchmarks achieved the highest speedup - 13.61% and 9.10% respectively, while the rest got more humbling speedup gains. The geomean speedup across all benchmarks is 0.98%.

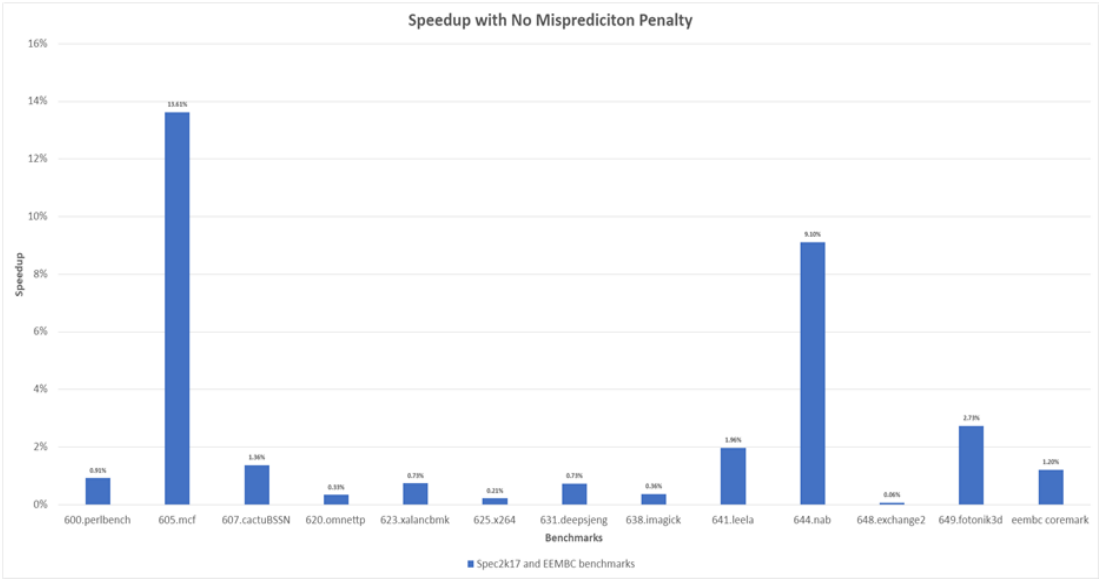
Figure 6.5b shows the prediction accuracy for each test. It shows that our stride predictor succeeded in predicting a high percentage of instructions for the two aforementioned benchmarks "mcf" and "nab", which explains them having high gains. "imagick" and "fotonik3d" aside, the lower percentages of correct predictions for the other benchmarks in figure 6.5b explain their lower speedups.

Again, in the next subsections 6.3.2 and 6.3.3, using dependency distance analysis, we justify the low gains of "imagick" and "fotonik3d" despite their high prediction accuracy, in a similar manner to what we did in the first set of experiments.

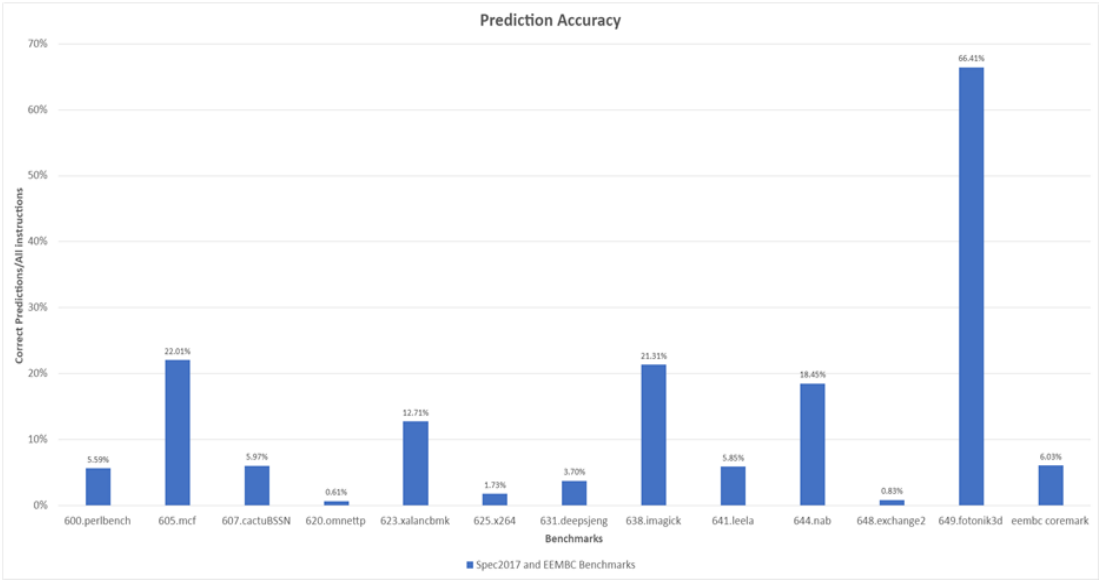
6.3.2 "imagick" Results

We attempt to explain away the low gains of "imagick" despite its relatively high prediction accuracy. The dependency distance analysis for this benchmark is shown in Figure 6.6a. We can see that a large part of the test - 49.47% of all the benchmark's instructions - have at least one parent with distance 1 that they depend on.

However, figure 6.6b shows that only 12.23% of the correctly predicted instructions have a distance of 1. That is, only 5.45% of the test's instructions had a producer distance of 1 in addition to being predicted correctly and *usefully*. The rest of the predictions with distance 1 are ineffectual. We can see two other spikes in figure 6.6b: distance 2 with 29.35% and distance 37 with 33.18%. However, according to figure 6.6a, only 2.79% of the benchmark has a minimal distance of 2 and only 2.48% of the test has a minimal distance of 37. This indicates minimal speedup gains caused by predictions of instructions with distance 2 and 37. As a result, we conclude that a smaller amount of predictions contributed to speedup gains than the relatively high prediction accuracy achieved for this test might indicate deceivingly.

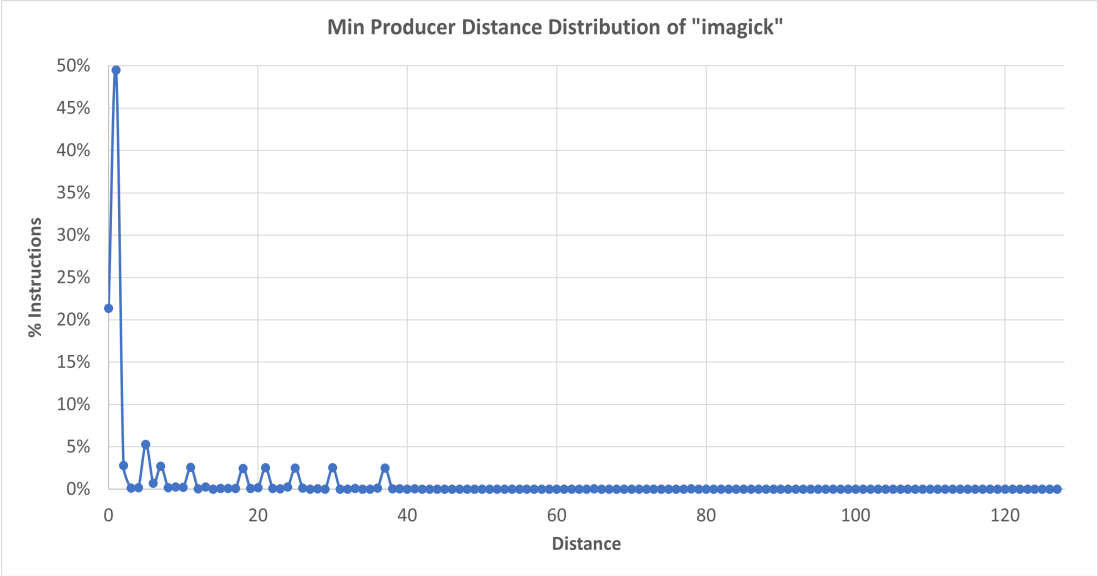


(a)

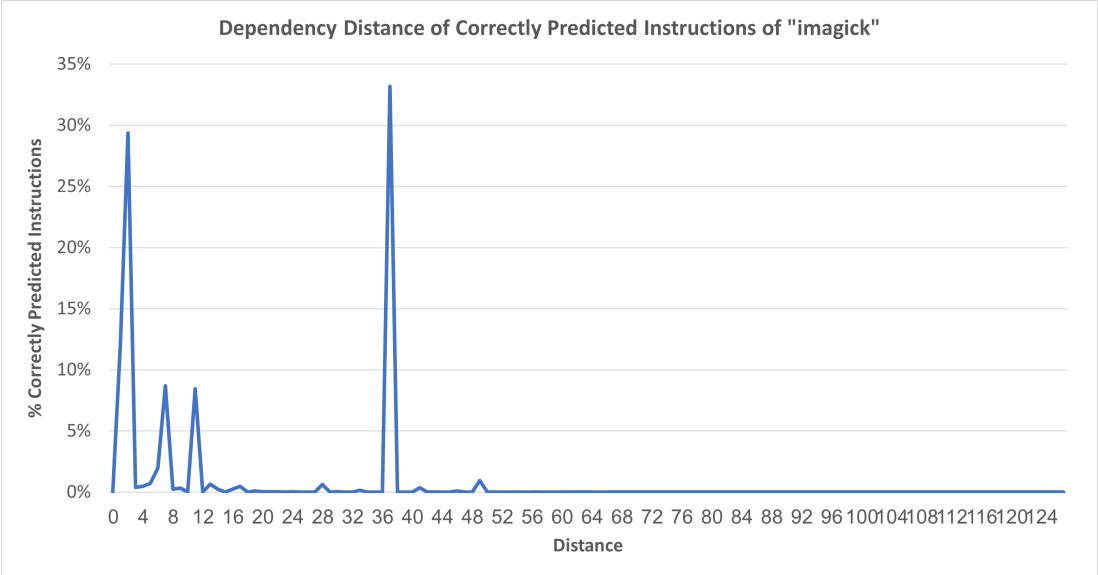


(b)

Figure 6.5: Width 8, 0-cycle penalty. (a) Speedup with No Misprediction Penalty. (b) Percentage of correctly predicted instructions out of all instructions



(a)



(b)

Figure 6.6: ”imagick” benchmark. (a)Distribution of the imagick’s Instructions’ Minimal Producer Distance. (b) Distribution of imagick’s Correctly Predicted Producers’ Distances

6.3.3 "fotonik3d" Results

We conducted a similar analysis for "fotonik3d" benchmark. Figure 6.7a presents its minimal producer distance analysis results. We see that 31.90% of the test's instructions have a minimal producer distance of 1. The rest of the distances seen in figure 6.7a have minimal effect on the final speedup due to their uncommonness.

Looking at figure 6.7b, we see that 27.69% of all correctly predicted instructions in the test have a producer distance of 1. That is, 23.08% of the test's instructions have a producer distance of 1 as well as managed to be correctly and *usefully* predicted. This explains the fair speedup of 2.73% shown in figure 6.5a, unlike the deceiving high prediction accuracy shown in figure 6.5b for "fotonik3d" benchmark.

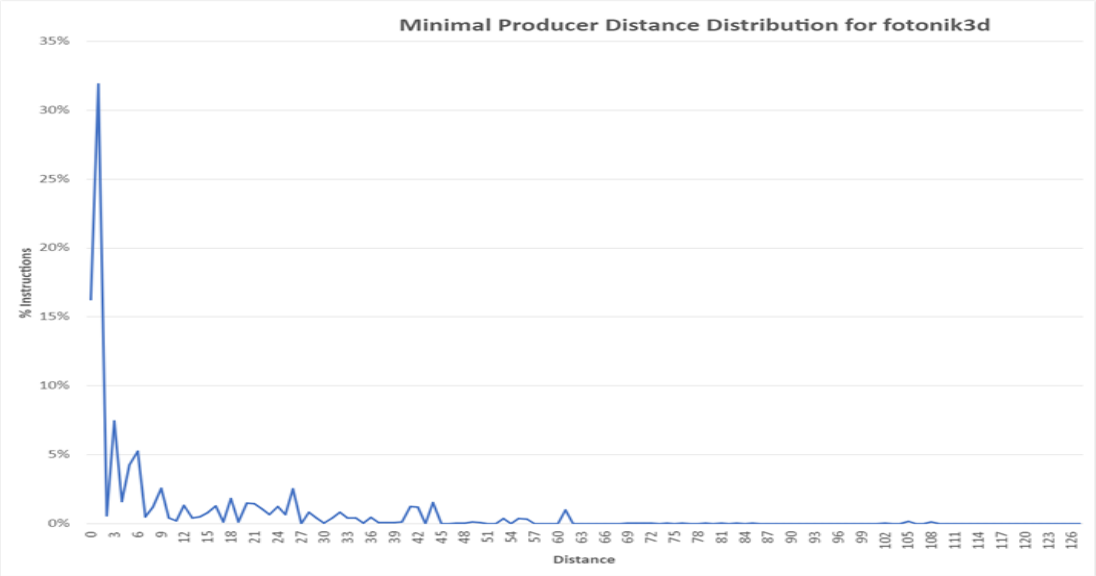
6.4 Second set: pipeline width 8, positive penalty

To take misprediction penalty into consideration and better estimate realistic results, we measured the number of incorrectly predicted instructions (i.e., mispredictions) and calculated the effect of their penalty on the IPC that was achieved when the penalty was set to zero in our VP simulated architecture. We calculated 4 possibilities: a penalty of 5, 10, 15, and 20 clock cycles per misprediction. The results are shown in figure 6.8a.

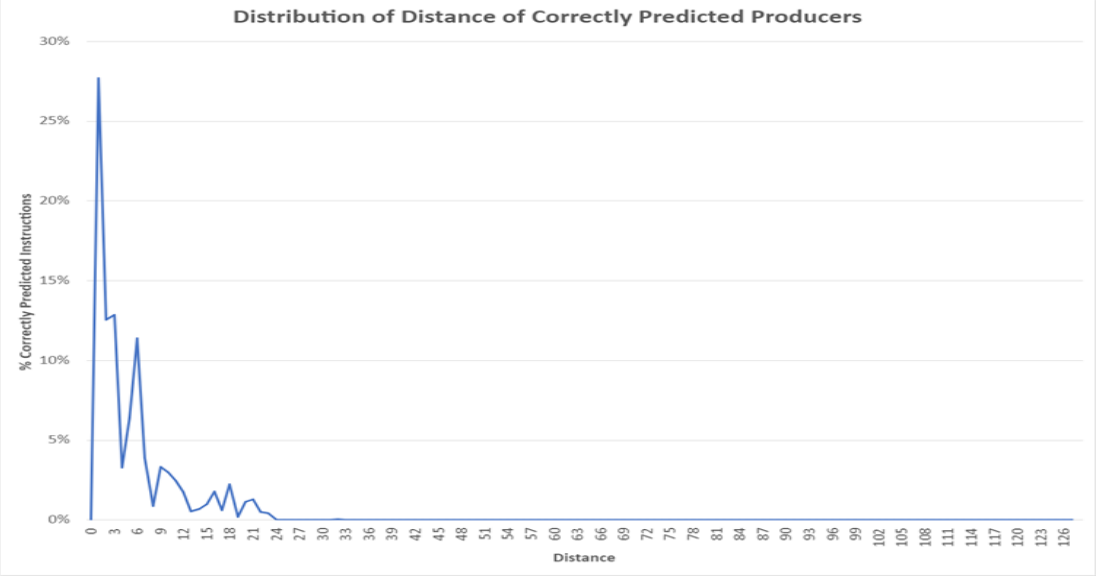
We notice that benchmarks "exchange2" and "EEMBC CoreMark" have the largest IPC degradation percentages. We look at the misprediction percentage out of all the instructions in each test since the IPC degradation in this experiment is due to incorrect predictions resulting in recovery mechanism penalty cycles. This misprediction percentage is shown in figure 6.8b. Again, like in the first set of experiments, "exchange2" and "EEMBC CoreMark" have the highest misprediction percentage, excluding "mcf", justifying their larger decrement in IPC. The speedup gains brought by the high percentage of correct predictions in "mcf" seems to compensate for the misprediction percentage shown in figure 6.8b. The lower misprediction percentages for all other tests explains their lower IPC degradation shown in figure 6.8a.

Note that IPC degradation corresponds to speedup degradation since we have the same instruction count and frequency across benchmarks and experiments.

We also calculated the final speedup given a penalty of 5, 10, 15, and 20 clock cycles per misprediction. Figure 6.8c shows the speedup given the different misprediction penalties. As expected, the gains are smaller when penalty is added, and sometimes we get negative speedup as well, but "mcf" and "nab" still show significant improvement over the baseline even with maximum penalty of 20 cycles per misprediction.

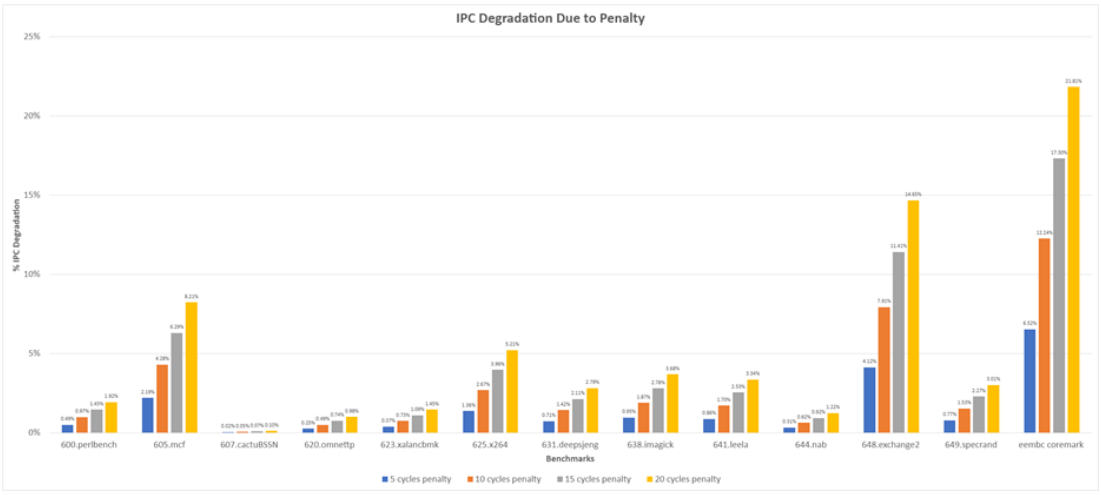


(a)

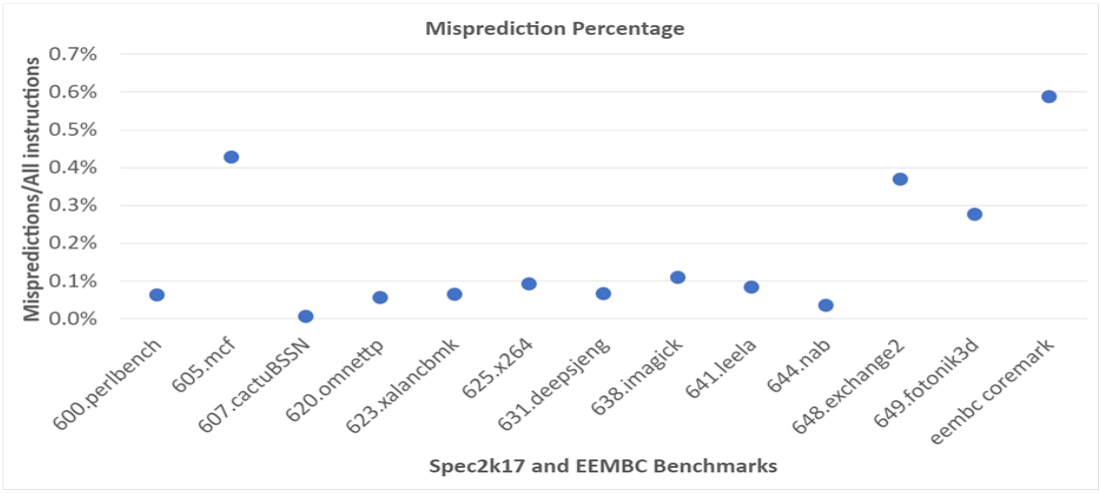


(b)

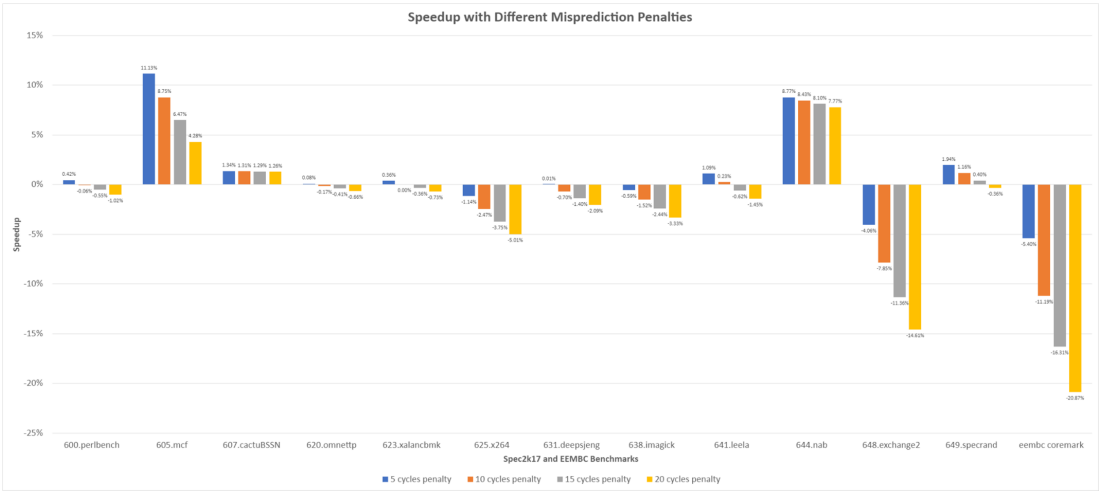
Figure 6.7: "fotonik3d" benchmark. (a) Distribution of the fotonik3d's Instructions' Minimal Producer Distance. (b) Distribution of fotonik3d's Correctly Predicted Producers' Distances



(a)



(b)



(c)

Figure 6.8: Width 8, positive-cycle penalty. (a) IPC Degradation Due to Penalty of 5, 10, 15, and 20 clock cycles. (b) Misprediction Fraction out of All Instructions. (c) Speedup with a Misprediction Penalty of 5, 10, 15, and 20 clock cycles

Chapter 7

Conclusions and Future Research

7.1 Summary

To conclude, in this work, we have presented μOp-Based Value Prediction: a new scheme for Value Prediction that supports μOp granularity predictions and aims at enabling practical implementation for next generations processors. To overcome the main challenges of real-life implementations - namely, complexity overheads - our micro-architecture consists of a training phase that lies outside the main pipeline and uses a value predictor table to train predictions until they reach high confidence; and a low-overhead deployment phase that lies in the pipeline and leverages the μOp Cache and a VRT to deploy confident predictions speculatively. Eventually, a validation mechanism is in place to identify and recover from mispredictions.

We showed that considering both a 4-wide and an 8-wide processor as the baseline, the same processors augmented with μOp-Based VP and validation at commit time can achieve significant speedup gains, and importantly, demonstrate a practical deployment option of VP for μOps in modern chips.

7.2 Discussion and Future Research

In our simulations, we used a constant size for VRT and VP tables. We believe that our scheme has the potential to enable the use of small tables for implementation which makes it area-efficient. Future examination of different VRT and predictor sizes for different applications could develop insight into the most area- and power-efficient size option.

We evaluated our results on a stride predictor and a flushing mechanism for a misprediction recovery mechanism to demonstrate speedup gains as a proof of concept. However, our architecture is orthogonal to both the predictor type and the recovery mechanism. A more accurate predictor should improve gains immediately since we saw that the prediction accuracy we got was not state-of-the-art. Future research could investigate employing other predictors to evaluate gains; and preexistent branch

recovery mechanisms in the system can be also utilized for VP.

An important take-away from our results is that some predictions are more useful than others depending on the predicted producers' distances from their dependants. We conclude that any workload where the predictor can manage to predict a high percentage of the producers with the lowest distances - and not necessarily the highest number of producers in general - will give us the best results since those predictions would be the most effectual.

We showed experimental results on pipeline width of both 4 and 8. The results were similar conceptually. We conclude that our scheme is independent of the issue width of the pipe.

It is important to note that our architecture can be implemented with and without a μOp Cache with minor tweaks to accommodate both RISC and CISC CPUs.

Bibliography

[BGS+20] Sumeet Bandishte, Jayesh Gaur, Zeev Sperber, Lihu Rappoport, Adi Yoaz, and Sreenivas Subramoney. Focused value prediction. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 79–91. IEEE Press, 2020.

[BLvK18] James Bucek, Klaus-Dieter Lange, and Jóakim von Kistowski. Spec cpu2017: Next-generation compute benchmark. In *SPEC CPU2017: Next-Generation Compute Benchmark*, pages 41–42, 04 2018.

[CHE11] T.E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*, pages 52:1–52:12, 01 2011.

[CRT99] Brad Calder, Glenn Reinman, and Dean M Tullsen. Selective value prediction. In *Proceedings of the 26th annual international symposium on computer architecture*, pages 64–74, 1999.

[GKMP98] D. Grunwald, A. Klauser, S. Manne, and A. Pleszkun. Confidence estimation for speculation control. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*, pages 122–131, 1998.

[GM96] Freddy Gabbay and Avi Mendelson. *Speculative execution based on value prediction*. Citeseer, 1996.

[GM98a] F. Gabbay and A. Mendelson. The effect of instruction fetch bandwidth on value prediction. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*, pages 272–281, 1998.

[GM98b] Freddy Gabbay and Avi Mendelson. Using value prediction to increase the power of speculative execution hardware. *ACM Transactions on Computer Systems (TOCS)*, 16(3):234–270, 1998.

[HSU+01] G. Hinton, David Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the pentium 4 processor. *Intel Technology Journal Q1*, 5, 01 2001.

[Joh91] Mike Johnson. *Superscalar multiprocessor design.* Prentice-Hall, Inc., 1991.

[LWS96] Mikko H Lipasti, Christopher B Wilkerson, and John Paul Shen. Value locality and load value prediction. In *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 138–147, 1996.

[PCLGO09] Jason A. Poovey, Thomas M. Conte, Markus Levy, and Shay Gal-On. A benchmark characterization of the eembc benchmark suite. *IEEE Micro*, 29, 2009.

[Per15] Arthur Perais. *Increasing the performance of superscalar processors through value prediction.* Theses, Université de Rennes, September 2015.

[Per21] Arthur Perais. Leveraging targeted value prediction to unlock new hardware strength reduction potential. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 792–803, New York, NY, USA, 2021. Association for Computing Machinery.

[PES16] Arthur Perais, Fernando A. Endo, and André Seznec. Register sharing for equality prediction. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016.

[PS14a] Arthur Perais and Andre Seznec. Practical data value speculation for future high-end processors. In *Proceedings - International Symposium on High-Performance Computer Architecture*, pages 428–439, 02 2014.

[PS14b] Arthur Perais and André Seznec. Eole: Paving the way for an effective implementation of value prediction. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 481–492, 2014.

[PS14c] Arthur Perais and André Seznec. Practical data value speculation for future high-end processors. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 428–439, 2014.

[PS15] Arthur Perais and André Seznec. Bebop: A cost effective predictor infrastructure for superscalar value prediction. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 13–25, 2015.

[RZ06] N. Riley and C. Zilles. Probabilistic counter updates for predictor hysteresis and stratification. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, pages 110–120, 2006.

[SCD17] Rami Sheikh, Harold Cain, and Raguram Damodaran. Load value prediction via path-based address prediction: Avoiding mispredictions due to conflicting stores. In *Load Value Prediction via Path-based Address Prediction: Avoiding Mispredictions due to Conflicting Stores*, 10 2017.

[Sez18] André Seznec. Exploring value prediction with the EVES predictor. In *CVP-1 2018 - 1st Championship Value Prediction*, pages 1–6, Los Angeles, United States, June 2018.

[She19] Rami Sheikh. Efficient load value prediction using multiple predictors and filters. In *Efficient Load Value Prediction Using Multiple Predictors and Filters*, 02 2019.

[SL19] Anjana Subramanian and Calvin Lin. Advancing value prediction by anjana subramanian thesis. In *Advancing Value Prediction by Anjana Subramanian Thesis*, 2019.

[Smi81] James E. Smith. A study of branch prediction strategies. In *International Symposium on Computer Architecture*, 1981.

[SMO+01] Baruch Solomon, Avi Mendelson, Doron Orenstein, Yoav Almog, and Ronny Ronen. Micro-operation cache: a power aware frontend for the variable instruction length isa. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 4–9, 2001.

[SS97] Y. Sazeides and J.E. Smith. The predictability of data values. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 248–258, 1997.

[Wal91] David W Wall. Limits of instruction-level parallelism. In *Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 176–188, 1991.

[YYX05] Xiao Yong, Yang Yanping, and Zhou Xingming. Revised stride data value predictor design. In *Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*, pages 6–pp. IEEE, 2005.

של הפקודות בהם לא נשמרות אחרי פיענוחן לפקודות RISC. הארכיטקטורה שלנו ממנפת את זכרון מטמון המיקרו-פקודות כדי להתגבר על הבעיה הזאת ולאפשר חיזוי ערכים ברמת מיקרו-פקודות.

מחקר זה שואף לסלול דרך למימושים מעשיים של חיזוי ערכים בדורי העתיד של מעבדים מהירים.

בניסויים שלנו הרצנו מבדקים (טסטים) מסויטת הטסטים של Spec2k17 בנוסף לטסט CoreMark של EEMBC על סימולטור sniper x86-simulator. בעבודה זו, אנו מציגים תוצאות של שתי סדרות של ניסויים: סדרת ניסויים על מעבד ברוחב 4, וסדרה על מעבד ברוחב 8. בשני המקרים, התוצאות מציגות מספרים שהוסגו בעזרת מעבד המשלב את המימוש שלנו של חיזוי ערכים עם חוזה מסוג stride כנגד אותו מעבד בלי חיזוי ערכים. הסימולציה עם המעבד ברוחב 8 מראה - בממוצע - 2.56% יותר פקודות מבוצעות בכל מחזור שעון ביחס לאותו מעבד בלי חיזוי ערכים. הטסט שהפגין את השיפור הכי גדול במספר הפקודות המובצעות במחזור שעון הראה שיפור של 13.61%. המעבד ברוחב 4 השיג ביצועים של 2.23% יותר פקודות מבוצעות במחזור שעון בממוצע ביחס לאותו מעבד בלי חיזוי ערכים. הטסט שהשיג הכי הרבה שיפור במספר הפקודות המבוצעות במחזור אחד הציג שיפור של 12.36%. כדי להסביר ולהצדיק את התוצאות של טסטים ספציפיים שהשיגו שיפור נמוך יחסית בביצועים למרות שהחוזה הצליח לחזות אחוז גבוה מהפקודות שלהם בהצלחה, ביצענו ניתוח של מרחקי פקודות תלויות מההורים שלהם.

תקציר

מקביליות ברמת הפקודות משחקת תפקיד חשוב בתכנון מעבדים מודרניים. במשך עשורים, המקביליות הזאת הייתה נחשבת למוגבלת על ידי תלויות ערכי-אמת. בשנות התשעים, הוגדר והוצג המושג של חיזוי ערכים שמטרתו היא לאפשר למעבדים מקביליות נוספת ברמת הפקודות ולהשיג ביצועים טובים יותר. חיזוי ערכים משיג זאת דרך שבירת התלויות הנ"ל בעזרת זיהוי דפוסים בערכי היעד של פקודות וכתוצאה מכך, מאפשר לפקודות להתחיל להתבצע מוקדם יותר - באופן משוער - במקום לחכות לפקודות הקודמות להן לסיים את ביצוען.

הרבה מאמרים במשך השנים תיארו ארכיטקטורות וטכניקות שונות למימוש חיזוי ערכים. חלק מהם הציע לבצע פקודות מסויימות בחזית המעבד, ופקודות אחרות בקצה האחורי של המעבד. חלק התמקד בחיזוי סוגי פקודות אחדים כדי לשלוט בגדלי טבלאות חיזוי קטנות יותר. ארכיטקטורות שונות בספרות השתמשו בסוגי חוזים מרובים. בעבודה שלנו אנחנו משתמשים בחוזה מסוג stride בשל התקורה הנמוכה שלו. חוזה זה מנבא את ערך היעד של פקודה על ידי הוספת דלתא לערך קודם של אותה פקודה בהתבסס על ערכי הפקודה שהחוזה נתקל בהם בעבר.

למיטב ידיעתנו, שבבים מסחריים עדיין לא מימשו חיזוי ערכים כחלק מהארכיטקטורה שלהם כי הרווח בביצועים לא מצדיק את עלות המימוש. בתזה הזאת, אנו מציעים מימוש חדש, יעיל בביצועים, של חיזוי ערכים. הוא חוזה את ערכי היעד של פקודות מסויימות כדי לאפשר לפקודות התלויות בהן להתחיל את ביצוען מוקדם יותר בשימוש בערכים החזויים. המימוש שלנו מורכב משני שלבים: שלב של אימון שבו משתמשים בחוזה כדי לאמן חיזויים עד שהם צוברים מספיק בטחון; ושלב של פריסה שבו משתמשים בחיזויים שכבר צברו ביטחון. חשוב לציין שברגע שערך חזוי מסויים צובר מספיק בטחון, שלב האימון של הפקודה המתאימה לו מפסיק. רק לאחר שערך חזוי מתגלה כלא נכון, המערכת מחזירה את הפקודה המתאימה לו בחזרה לשלב האימון. בדרך זו, אנו חוסכים אנרגיה מיותרת ונמנעים מטבלאות חיזוי גדולות.

בדיזיין שלנו, ובניגוד לתכנון טיפוסי של חיזוי ערכים, שלב האימון אינו מהווה חלק מהצנרת של המעבד. לפיכך, אפשר להחליש את אילוצי הזמן עליו. הבחירה הזאת מאפשרת לנו להימנע מהוספת לחץ על חזית המעבד - בעיה שבדרך כלל מימושי חיזוי ערכים נתקלים בה. שלב הפריסה - לעומת זאת - משולב בצנרת המעבד ולכן הוא מתוכנן להיות פשוט ככל הניתן עם תקורה מינימלית. בשלב הפריסה, רק חיזויים עם רמת בטחון גבוהה נפרסים, מה שעוקף את הצורך בגישות מיותרות לשאר הפקודות הלא זכאיות לחיזוי או שרמת הבטחון שלהן נמוכה.

בנוסף, אנו מאפשרים חיזוי ערכים ברמה של מיקרו-פקודות, דבר שמימושים אחרים רבים בספרות לא מאפשרים. חוזים טיפוסיים נהיים בטלים וחסרי תועלת במעבדי CISC משום שהכתובות המדוייקות

המחקר בוצע בהנחייתם של פרופסור פרדי גבאי ופרופסור אבי מנדלסון, בפקולטה להנדסת חשמל ומחשבים.

מחבר התזה מציין כי המחקר, כולל איסוף, עיבוד והצגת הנתונים, התייחסות והשוואה למחקרים קודמים וכו', נעשה בצורה כנה לחלוטין, כמצופה ממחקר מדעי שנערך על פי הסטנדרטים האתיים של העולם האקדמי. כמו כן, דיווח על המחקר ותוצאותיה בתזה זו נעשו בצורה כנה ומלאה, לפי אותם סטנדרטים.

תודות

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

חיזוי ערכים מבוסס על מיקרו-פקודות: גישה חדשה המאפשרת מימוש יעיל

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בהנדסת חשמל

ליאן ג'רג'ורה

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
אדר ב' ה'תשפ"ד חיפה מרץ 2024

חיזוי ערכים מבוסס על מיקרו-פקודות:
גישה חדשה המאפשרת מימוש יעיל

ליאן ג'רג'ורה