A Machine Learning Pipeline Stage for Adaptive Frequency Adjustment

Arash Fouman Ajirlou, *Student Member, IEEE* and Inna Partin-Vaisband, *Member, IEEE*

Abstract—A machine learning (ML) design framework is proposed for adaptively adjusting clock frequency based on propagation delay of individual instructions. A random forest model is trained to classify propagation delays in real time, utilizing current operation type, current operands, and computation history as ML features. The trained model is implemented in Verilog as an additional pipeline stage within TigerMIPS processor. The modified system is experimentally tested at the gate level in 45 nm CMOS technology, exhibiting simultaneously a speedup of 70 percent and an energy reduction of 30 percent with coarse-grained ML classification as compared with the baseline TigerMIPS. A speedup of 89 percent is demonstrated with finer granularities with a simultaneous 15.5 percent reduction in energy consumption.

Index Terms—Computer systems organization, microprocessors and microcomputers, hardware, pipeline, processor architectures, pipeline processors, pipeline implementation, VLSI systems, impact of VLSI on system design, VLSI, system architectures, integration and modeling, design methodology, cost/performance, machine learning, classifier design and evaluation

1 Introduction

The primary design goal in computer architecture is to maximize the performance of a system under power, area, temperature, and other application-specific constraints. Heterogeneous nature of VLSI systems and the adverse effect of process, voltage, and temperature (PVT) variations have raised challenges in meeting timing constraints in modern integrated circuits (ICs). To address these challenges, timing guardbands have constantly been increased, limiting the operational frequency of synchronous digital circuits. On the other hand, the expanding variety of functions in modern processors increases delay imbalance among different signal propagation paths. Bounded by critical path delay, these systems are traditionally designed with pessimistically long clock period, yielding underutilized IC performance. Moreover, power efficiency of these underutilized systems also degrades due to the increasing power leakage. Alternatively, when designed with relaxed timing constraints, integrated systems are prone to functional failures. To simultaneously maintain correct functionality and increase system performance, numerous optimization techniques as well as offline and online models have recently been proposed, including pipelining, multicore computing, dynamic frequency and voltage scaling (DVFS), and ML driven models [1], [2], [3], [4], [5], [6], [7], [8], [9], [10].

• *The authors are with the Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL 60607 USA. E-mail: {afouma2, vaisband}@uic.edu.*

Manuscript received 5 July 2020; revised 17 Dec. 2020; accepted 17 Jan. 2021. Date of publication 9 Feb. 2021; date of current version 11 Feb. 2022. (Corresponding author: Arash Fouman Ajirlou.) Recommended for acceptance by Y. Yang. Digital Object Identifier no. 10.1109/TC.2021.3057764

Propagation delay in a processor is a strong function of the type, input operands, and output of the current operation, and computation history [4]. Computation history accounts for data overwrite and crosstalk noises. For example, the delay overheads for keeping a bit value constant, changing a single bit from 0 to 1, or changing the same bit from 1 to 0 are all different. Furthermore, due to capacitive coupling between adjacent wires, these delays can significantly vary based on toggling schemes of the individual bits within the multi-bit input and output buses. To illustrate the significant difference in propagation delay between fast and slow execution of an operation in a typical processor, one million random instructions are executed on a conventional 32-bit MIPS processor, exhibiting a broad spectrum of propagation delays for each of the AND, ADD, ADDI, and MULT operations, as shown in Fig. 1. The worst-case clock period, as reported by Synopsys Design Compiler, is 4 ns. As expected, majority of operations are completed within a small portion of the clock period. For example, the mean propagation delay of the 248,667 random MULT instructions (2.2 ns) is twice longer than the mean propagation delay of the one million random instructions (1.1 ns). Thus, executing all the instructions with the worst-case clock signal, or categorizing the execution delay based on operation type yields a significant delay overhead in modern ICs.

While multicore approaches have been proposed to enhance system performance, the scalability of modern multicore systems is limited by the design complexity of instruction level parallelism and thermal design power constraints [12], [13]. Note that while multicore and out-of-order CISC processors are often utilized in high-end applications, RISC processors comprise 99 percent of the overall microprocessors shipped annually around the world [11]. Thus, speeding single thread, RISC architectures is an important cornerstone for enhancing performance in modern processors [11], [14]. This is, therefore, the primary focus of the proposed

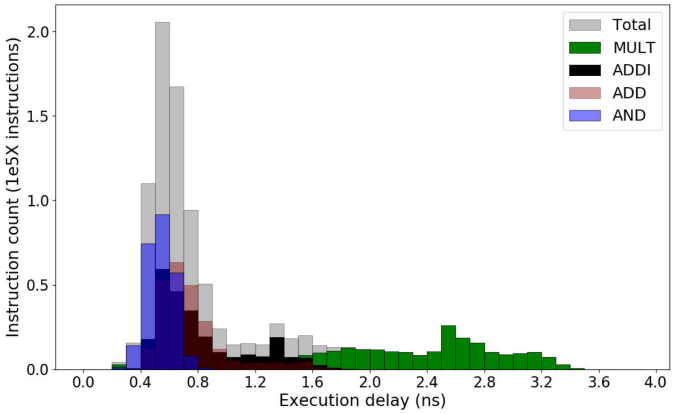

Fig. 1. Distribution of propagation delays of one million random ADD, ADDI, AND, and MULT instructions, executed on a RISC processor.

approach. To the best of the authors' knowledge, this paper is the first to employ ML for adaptively adjusting the clock frequency at the instruction-level. Note that with the proposed method, clock frequency is adaptively adjusted (by switching to a preferred physical clock signal) *per instruction* in real time, yielding a fundamentally different approach as compared with the traditional, task-based dynamic frequency scaling. Also note that a design of multiple clock signals, as required with the proposed method, is a common practice in modern ICs [15]. The main contributions of this work are as follows:

1) A systematic flow is proposed and implemented as a unified platform for extracting ML input features from an instruction and classifying the instruction execution delay in real time.
2) A random forest (RF) model is trained to classify individual instructions into delay classes based on their type, input operands, and the computation history of the system.
3) A new pipeline stage is designed and integrated within a pipelined MIPS processor.
4) The proposed method is synthesized and verified on LegUp [16] benchmark suite of programs with Synopsys Design Compiler in 45 nm CMOS technology node.

The rest of the paper is organized as follows. Prior and related work is described in Section 2. The proposed unified platform and the design methodology are explained in Section 3. ML algorithms for classification of instruction delay are described in Section 4. In Section 5 the implementation details of the system are introduced. Experimental results are presented in Section 6. Conclusions and future work are discussed in Section 7, and the paper is summarized in Section 8.

2 Prior and Related Work

Multiple approaches have been proposed for efficiently tuning the operating point (i.e., voltage supply and clock frequency) of a system at various levels of a computing system, including application- and task-based methods and instruction-level speculations.

Predicting timing violations in a constraint-relaxed system is impractical with deterministic approaches, due to the wide dynamic range of input and output signals (typically 32 or 64 bits), variety of operations in a modern processor, and delay dependence on the runtime and physical characteristics of the system (e.g., crosstalk noise). ML based approaches for predicting timing violations of individual instructions have recently been proposed, which consider the impact of input operands and computation history on timing violations [4], [17], [19]. While significant for the design process of next generation scalable high performance systems, these approaches have several limitations:

1) Instruction output is considered as a ML feature and exploited in these systems for predicting the timing characteristics of the individual instructions. These predictions are, however, carried out before the instruction execution, when the instruction output is not yet available, limiting the effectiveness of these methods in practical systems.
2) The modules under the test are studied separately and evaluated in an isolated test environment without considering the effects of other processing elements (e.g., arithmetic modules, buffers or multiplexers). The high reported accuracy is, therefore, expected to degrade if the methods are applied to a complex system (e.g., a practical execution unit).
3) Power and timing overheads due to additional hardware are not considered in these papers.

Granularity of prediction is another primary concern. A bit-level ML based method has been proposed in [18] for predicting timing violations with reduced timing guardbands. While up to 95 percent prediction accuracy has been reported with this method, the excessively high, per bit granularity of the ML predictions is expected to exhibit substantial power, area, and timing overheads. These overheads are, however, not evaluated in [18]. Furthermore, a procedure for recovery upon a timing error is not provided and the recovery overheads are also not considered.

As an alternative to fine-grain high-overhead ML methods, multiple coarse-grain schemes for timing error detection and recovery have been proposed to mitigate the adverse effect of the pessimistic design constraints. A better-than-worst-case design approach has been introduced in [5]. With this approach, the clock period is set to a statistically nominal value (rather than worst-case propagation delay) and the history of timing erroneous program counters is kept in a ternary content-addressable memory (TCAM). The TCAM is exploited for predicting timing violations of the instructions based on previous observations. Note that the system only warns against those timing violations that have been previously recorded. Alternatively, unseen violations are not predicted with this approach. Owning to the apparent simplicity of this approach, only bi-state operating conditions (i.e., nominal and worst-case clock frequencies) can be efficiently utilized with this method. Alternatively, the design complexity and system overheads are expected to significantly increase with the increasing number of frequency domains.

An efficient, coarse grain approach has recently been proposed, which determines a coarse grain DVFS scheme prior to program execution [10]. This approach relies on diagnosis of critical paths based on activation patterns and

adjusts the supply voltage level according to this information. A fine grained clock and/or voltage adjustment is, however, impractical with this approach because the execution time strongly depends on the individual operands and the computation history, which are not available prior to program execution.

In BandiTS [20], a reinforcement learning approach has been proposed to estimate the timing error probability (TEP) within a program time interval, given timing speculation (TS) ratios, $TSR = t_{clk}/t_{nom}$ for various values of the reduced clock period t_{clk}, and the worst-case clock period t_{nom}. The TS-based TEP problem is modeled in [20] as the classical multi-armed bandit problem [21], where the TS ratios and TEPs correspond to, respectively, the arms and stochastic rewards. The primary limitation of that work is the lack of details about the hardware implementation and overheads. In addition, the maximum achievable performance gain of only 25 percent has been reported. Furthermore, BandiTS approach exhibits per-task clock granularity and scales the clock frequency for a batch of instructions. Higher performance gain is possible with fine-grain, per instruction clock frequency adjustment, as shown in this paper.

A thermal-aware voltage scaling has been proposed in [22]. Voltage selection algorithm has been developed and integrated within FPGA synthesis process to aggressively scale the core and block RAM voltages, utilizing the available thermal headroom of the FPGA-mapped design. As a result, 36 percent reduction in power consumption has been demonstrated. Driven by workload and thermal power dissipation, this method, however, supports only coarse-grain voltage and frequency scaling.

Predicting program error rate in timing-speculative processors has been proposed in [23]. A statistical model is developed for predicting dynamic timing slack (DTS) at various pipeline stages. The predicted DTS values are exploited to estimate the timing error rate in a program. The implementation overheads, and the potential performance or power consumption gains are, however, not reported in [23].

An offline model for TS processors has been introduced in [24]. This probabilistic model is trained to optimally select a better-than-worst-case, nominal clock frequency. The provided hardware-based speculation, however, does not consider the overall workload or specific finer units, limiting the fidelity of the method. Alternatively, the adverse effect of process variations on the propagation delay is considered, strengthening the approach in [24]. Note that PVT variations are also considered with the proposed approach of classifying instructions into delay intervals in real time, as described in the following sections.

Finally, ML based methods for modeling system behavior have also been proposed. For example, in [6], linear regression has been leveraged for modeling the aging behavior of an embedded processor based on current instruction and its operands, as well as the computation history and overall circuit switching activity. As a result, the timing guardband designed to compensate for aging in digital circuits can be effectively reduced, in presence of graceful degradation [6]. Reallocation of delay budget has, however, not been considered with this method.

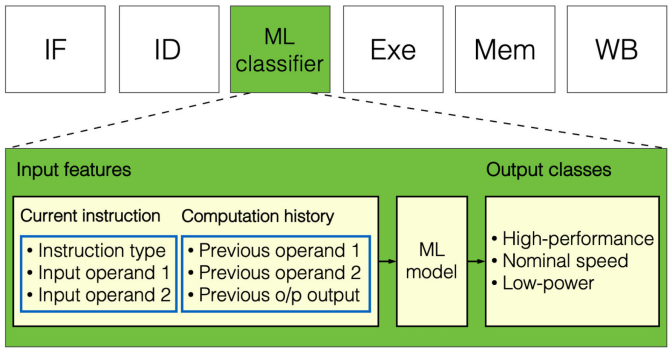

Fig. 2. The proposed pipeline with the additional ML stage. In this configuration, six ML features and three delay classes are illustrated.

ML ICs can exhibit a prohibitively high power consumption and physical size. Furthermore, auxiliary ML ICs can introduce additional delay and increase design complexity, depending upon the application characteristics. To efficiently exploit ML methods for managing frequency in modern processors, delay, power, and area of ML ICs should be considered.

3 The Proposed ML-Based Frequency Adjustment

In this paper, a design methodology is proposed for ML driven adjustment of operational frequency in pipeline processors. To reduce the gap between the actual propagation delay and the clock period, individual instructions are classified into the corresponding propagation delay classes in real time, and the clock frequency is accordingly adjusted by switching to a preferred physical clock signal. The classes are defined by segmenting the worst-case clock period into shorter delay fragments. Each class is characterized by a specific supply voltage and clock frequency. The primary design objective is to maximize system performance within an allocated energy budget. The overall delay and energy consumption are evaluated with the additional ML components, and considering the overheads of incorrect predictions. The proposed scalable framework allows for other control configurations to be defined in a similar manner for various design objectives. The real-time clock adjustment is enabled by the recent advancement in clock management circuits [27].

In order to evaluate this method, a pipelined 32-bit RISC processor (TigerMIPS [25]) is utilized as the baseline processor. The ML classifier is designed as an additional pipeline stage within the baseline MIPS processor, as shown in Fig. 2. The inputs to the additional ML pipeline stage are the current instruction and its operands, as well as the computation history, as defined by the toggled inputs bits (i.e., current inputs are XORed with the previous inputs) and output of the previous operation. The choice of these parameters is in accordance with the results in [4] and [6]. These inputs are utilized as ML features for predicting the delay class of the current instruction based on the trained ML model. It is important to note that more complex, slower ML models can also be trained with this methodology, as long as the overall design complexity and hardware costs of the system satisfy the specified constraints. To meet the

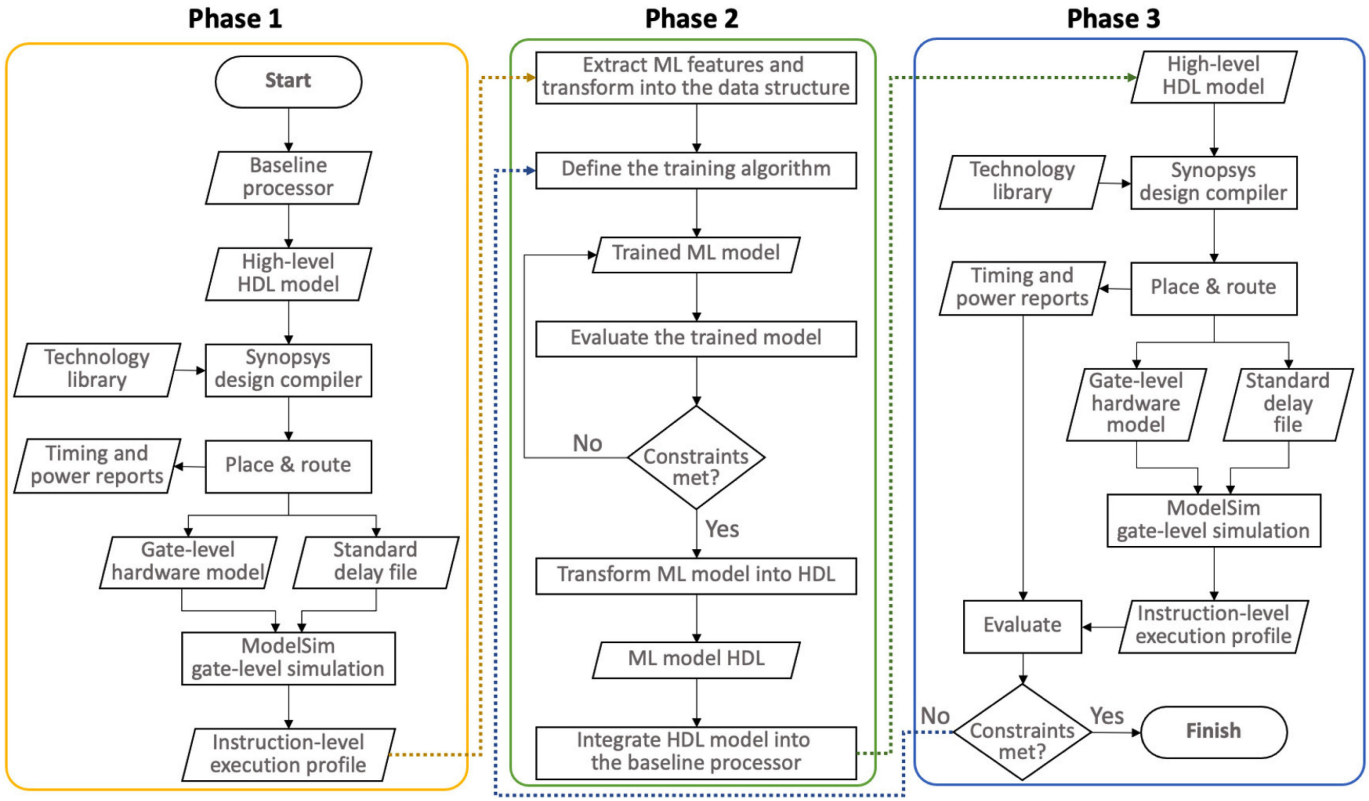

Fig. 3. Systematic flow for designing ML predictor within a typical pipelined processor.

overall throughput constraints, the trained models can be implemented as multiple pipeline stages, mitigating the additional latency introduced by the ML functions. Finally, the granularity of the output delay (e.g., three delay classes are illustrated in Fig. 2) can be varied to meet the timing constraints within the energy budget.

A systematic flow has been developed, prototyped, and verified on TigerMIPS with LegUp benchmark suite. The flow comprises three primary phases, as shown in Fig. 3. The individual phases are described in the following subsections.

3.1 Phase 1: Baseline Processor Synthesis and Profiling

During the first phase, the high-level hardware description language (HDL) model of the baseline processor is synthesized into gate-level description model. During this phase, timing information is generated in the IEEE standard delay format (SDF). Based on this information, the gate-level simulation (GLS) is performed and the instruction-level execution profile is generated. A profile comprises a list of instructions, the fetched or forwarded operands, the output of the operations, and the propagation delays. In addition to the execution profile, post place-and-route (PAR) reports, including timing and power information, are collected in this phase.

3.2 Phase 2: ML Training

In the second phase, the gate-level profiles from Phase 1 are parsed and utilized as ML features. Based on the extracted features, a preferred ML model is trained in Python with Scikit-learn ML library [26]. A HDL code (e.g., Verilog in this paper) of the trained model is generated and integrated within the baseline processor as a single (or multiple, as needed) pipeline stage(s) between the decode and execute stages (see Fig. 2).

3.3 Phase 3: Verification and Evaluation

During the last phase, the modified high-level HDL model of the system with the ML pipeline stage is synthesized and profiled, as described in Phase 1. To guarantee functional correctness, the output signal is double-sampled to detect timing violations, and timing-erroneous instructions are re-executed with the worst-case clock frequency. Similar to the baseline iteration, the post PAR reports are extracted for evaluating the timing and energy characteristics of the system. Finally, the profiling of the modified system is executed during this phase to evaluate the overall speedup of the system.

To optimize the final solution in terms of the operational frequency and energy consumption, the proposed flow is executed iteratively with various ML algorithms and clock fragments, as shown with the feedback in Fig. 3. The clock signal of the pipeline registers is assumed to be near-instantly switched based on the individual classification results, as has been experimentally demonstrated in [27].

4 Machine Learning Models

Owing to the unique learning characteristics and hardware tradeoffs of neural networks (NNs), support vector machines (SVMs), and random forest models, all these ML models are considered in this paper. Each model is trained based on the instruction profiles extracted from a synthetically generated dataset of 3,000 random instructions per class.

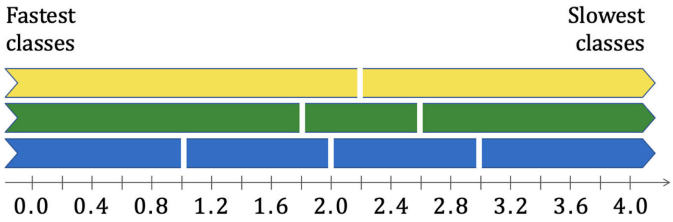

Fig. 4. Delay boundaries with respect to the worst-case delay of 4 ns for the two-, three-, and four-class configurations.

The delay boundaries of the individual classes are experimentally determined with respect to the worst-case delay of 4 ns as follows: {[0.0,2.2],(2.2,4.0]} for the two-class configuration, {[0.0,1.8],(1.8,2.6],(2.6,4.0]} for the three-class configuration, and {[0.0,1.0],(1.0,2.0],(2.0,3.0],(3.0,4.0]} for the four-class configuration (see Fig. 4).

The feature vector of the ith instruction comprises six elements, $x_i = (instr, op_1, op_2, Xop_1, Xop_2, output)$. The first feature, $instr$, comprises four subfeatures, representing the type of the operation in one-hot format,

$$instr = \begin{cases} 1000, & \text{if arithmetic} \\ 0100, & \text{if arithmetic with immediate operand} \\ 0010, & \text{if logical} \\ 0001, & \text{if multiplication or division} \end{cases}.$$

The subsequent four elements are defined by the operands. The features op_1 and op_2 are the first and second operands of the instruction, and the features Xop_1 and Xop_2 are the XORed values of the first and second operands with their respective previous values. The last feature, $output$, is the output of the preceding instruction. The last three elements of the feature vector are exploited to capture the effect of computation history on the instruction delay. Note that the operands and output of the preceding instruction are 32-bit long, as determined by the 32-bit baseline processor utilized in this work. Thus, the distribution of these features significantly differs from the distribution of the operation type subfeatures. To balance the overall distribution of the individual features, the input features are preprocessed and scaled to follow a normal distribution using *quantile transformer* in Python scikit-learn library. An example of operand and output features with and without the transformation is shown in Fig. 5 for arithmetic and logical instructions. Note that the type subfeatures remain unchanged.

To evaluate the efficiency and efficacy of the proposed method, propagation delay classification is investigated with three common ML algorithms: NN, SVM, and RF. The configuration, including the hyperparameters, performance, and hardware costs of RF models (which exhibit superior performance in systems explored in this work) are described in the following subsection. The configurations of NN and SVM (which might be of interest in other systems) are described in appendices A.1 and A.2, respectively, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2021.3057764. All the algorithms are five-fold cross-validated based on three thousand randomly generated instructions per class. While finding an effective metric for stability of the evaluation is still an

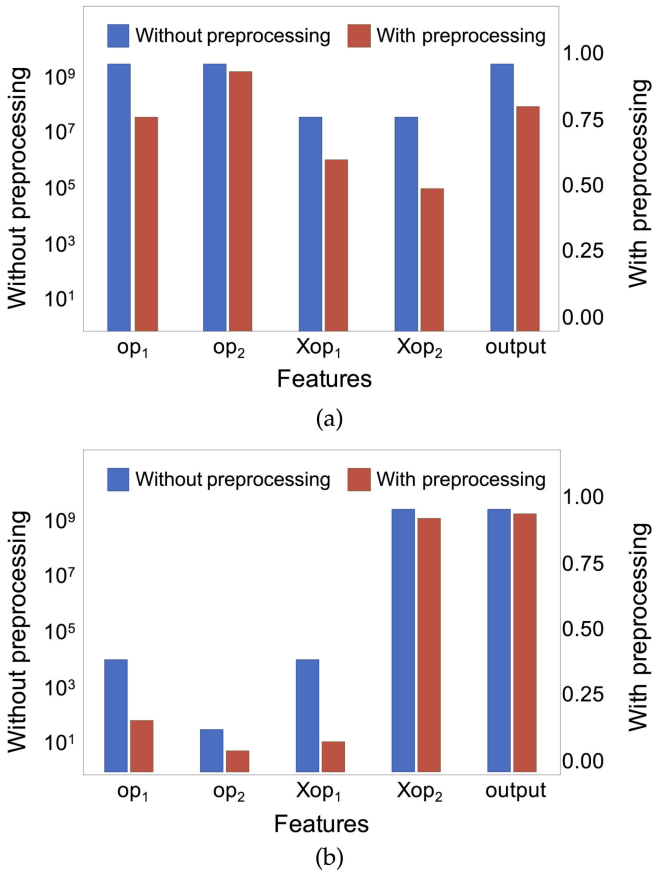

Fig. 5. A typical feature vector with and without the ML preprocessing, (a) for arithmetic operation with immediate operand, and (b) for logical operation. Note that the values without preprocessing are shown on a logarithmic scale, while the values with preprocessing are shown on a linear scale.

open question, k-fold cross-validation with $5 \leq K \leq 20$ is typically used, as these K values have been demonstrated to simultaneously minimize the bias and variance across many studied test sets [28], [29], [30], [31]. Thus, $K = 5$ is used in this work. ML accuracy is reported as the F1-score of delay classification and the resultant speedup for each benchmark program has been considered in determining the performance of each ML algorithm. Hardware cost is evaluated as the number of additional transistors required for implementing the individual ML algorithms and has also been considered in determining the performance of the ML algorithms. Among the evaluated ML algorithms, the RF classifier is preferred in this work due to the favorable tradeoff between the performance gain and hardware costs, as well as the relative simplicity of the RF algorithm, as explained in the following subsections.

4.1 Random Forest

RF classifier is an ensemble of decision tree classifiers. The input samples are split into multiple sample subsets and each decision tree is trained on a training subset. The final classification decision for each sample is made based on the result of averaging the individual tree decisions (i.e., ensembling). RF models benefit from the accuracy, training speed, and interpretability of the decision tree model, while the ensembling mitigates overfitting, otherwise common to decision tree classifiers. Thus, RF is often preferred in

TABLE 1
Top (within 1 percent of the highest F1-score) RF Configurations and Their Respective Performance Metrics (i.e., speedup, hardware cost (in million transistors), and speedup per hardware metric (SPH))

	4 classes					
	max_depth	n_estim	F1-score	Speedup	HW cost	SPH
1	30	50	0.852	1.835	0.177	10.357
2	10	200	0.850	1.925	0.480	4.012
3	30	200	0.850	1.889	0.709	2.666
4	10	100	0.849	1.913	0.240	7.978
5	50	200	0.849	1.833	0.815	2.249
6	50	100	0.846	1.856	0.407	4.554
7	20	200	0.845	1.874	0.624	3.004
8	50	50	0.843	1.836	0.204	9.011
9	30	100	0.842	1.838	0.354	5.187
10	10	50	0.840	1.902	0.120	15.859
Average			0.847	1.870	0.413	6.488
Positive standard deviation (σ^+)			0.002	0.016	0.137	2.638
Negative standard deviation (σ^+)			0.002	0.014	0.078	1.250

	3 classes					
	max_depth	n_estim	F1-score	Speedup	HW cost	SPH
1	20	50	0.949	1.879	0.156	12.040
2	30	100	0.947	1.856	0.354	5.238
3	20	200	0.946	1.851	0.624	2.966
4	40	200	0.945	1.847	0.768	2.403
5	40	50	0.944	1.843	0.192	9.591
6	50	200	0.944	1.839	0.815	2.257
7	10	200	0.944	1.848	0.480	3.853
8	30	200	0.942	1.814	0.709	2.561
9	10	100	0.940	1.828	0.240	7.621
10	10	50	0.939	1.826	0.120	15.224
Average			0.944	1.843	0.446	6.375
Positive standard deviation (σ^+)			0.002	0.008	0.117	2.764
Negative standard deviation (σ^+)			0.002	0.007	0.111	1.360

	2 classes					
	max_depth	n_estim	F1-score	Speedup	HW cost	SPH
1	40	50	0.981	1.688	0.192	8.785
2	30	200	0.981	1.686	0.709	2.380
3	30	100	0.981	1.683	0.354	4.750
4	40	200	0.981	1.686	0.768	2.193
5	10	50	0.981	1.686	0.120	14.058
6	20	200	0.981	1.683	0.624	2.696
7	10	200	0.980	1.686	0.480	3.515
8	50	200	0.980	1.687	0.815	2.070
9	10	100	0.980	1.685	0.240	7.024
10	20	100	0.979	1.683	0.312	5.393
Average			0.980	1.685	0.461	5.286
Positive standard deviation (σ^+)			2.0E-04	0.001	0.111	2.401
Negative standard deviation (σ^+)			4.3E-04	0.001	0.104	1.034

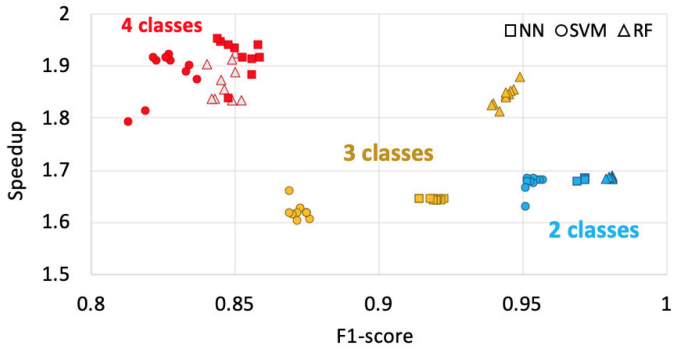

Fig. 6. Speedup versus F1-score for two-, three-, and four-class configurations. The results are extracted from RF raw data (listed in Table 1) and NN and SVM raw data (listed in Tables 4 and 5 in the Appendix sections), available in the online supplemental material.

scientific and practical applications [4], [32]. The computational and hardware complexity of RF is a strong function of the number and depth of the decision trees. The depth of the individual trees is dependent on the number of features and their correlation. In this work, a RF grid search is performed over the following ranges of hyperparameters:

1) Number of trees in the forest, $n\_estims \in \{1, 10, 50, 100, 200\}$,
2) Maximum number of levels in each tree, $max\_depth \in \{10, 20, 30, 40, 50\}$.

The results of the top estimators (within 1 percent of the highest F1-score) are listed in Table 1. The hardware cost of an RF classifier is evaluated based on the number of required comparators, $\mathcal{O}(n\_estims \times \log_2(max\_depth))$, and reported in terms of the total number of RF transistors. Transistor count for a single comparator is determined based on [33].

4.2 ML Algorithm Tradeoffs

The tradeoffs between the speedup and F1-score are summarized in Fig. 6 for all the classifiers. An interesting observation from the analysis in Fig. 6 is that no significant difference is exhibited among the speedups of the NN, SVM, and RF classifiers in two- and four-class configurations albeit the differences in F1-scores. Alternatively, with three-class configuration, RF considerably outperforms the other two methods, implying a complex relation between the number of classes, classifier type, and the resultant speedup. In particular, the type of misclassification significantly impacts the overall speedup. For example, classifying a slow instruction into a faster class results in an incorrect execution output, incurring a four-clock-cycle penalty for re-executing IF, ID, ML, and EX stages. Alternatively, misclassifying a fast instruction results in correct output albeit the lower or no performance gain. In addition, if a fast instruction is classified into an intermediate-delay class (for example, in the case with three delay classes), the overall performance of the system is still increased (but not maximized) as compared to the execution with the worst-case clock period. To understand the significance of speedup and overhead due to each of the ML classifiers, a speedup-per-hardware (SPH) metric is considered. The SPH results are shown in Fig. 7 for RF, NN, and SVM classifiers in two-, three-, and four-class configurations. Based on these results,

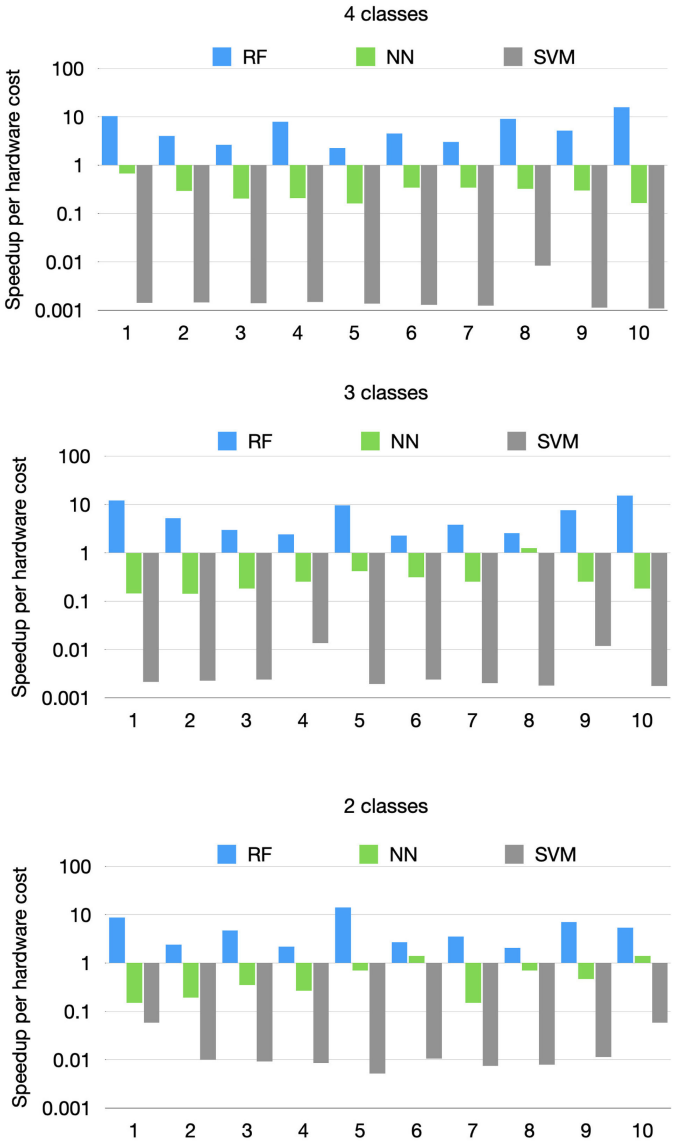

Fig. 7. Speedup per hardware cost (SPH) for two-, three, and four-class configurations. The hardware cost is evaluated based on the number of transistors needed to realize each classifier. The SPH performance is highest with RF classifier as compared with the SVM and NN based classifiers for each of the classifier configurations. The results are extracted from RF raw data (listed in Table 1) and NN and SVM raw data (listed in Tables 4 and 5 in the Appendix sections), available in the online supplemental material.

RF exhibits the best tradeoff between the hardware cost and speedup, as well as the lowest design complexity and hardware overheads. RF classifier is, therefore, preferred in this work as a demonstration vehicle of the proposed framework.

5 Implementation

The proposed framework is implemented with RF model within TigerMIPS and evaluated based on LegUp benchmarks. The details of the implementation are described in this section.

5.1 Unified Platform

A holistic platform is developed based on the proposed system design methodology, as illustrated in Fig. 3. The framework is unified within a shell programming platform supported with several peripheral programs developed in C++ and Python. The synthesis steps, as described in Fig. 3, are sequentially executed from *Start* to *Finish*.

During the first phase, Synopsys Design Compiler is called with the high-level HDL model of the baseline processor. The profiler triggers are added to the system and GLS is performed in Modelsim.

The second phase is triggered upon the completion of the instruction profiling. An external parser program is called to transform the instruction profiles into the ML feature data structure and eliminate outliers. The model is trained to classify propagation delays into user-defined number of classes based on a user-specified learning algorithm and delay boundaries. The ML accuracy and estimated speedup are evaluated upon the training completion. If the design requirements are met, the ML software model is transformed into the high-level HDL code. Otherwise, ML model is retrained with new parameters.

Upon training completion, the HDL code of the ML model is instantiated within the original HDL model of the baseline processor. Finally, the procedure in Phase 1 is repeated in Phase 3 with the modified processor model, and the overall system performance and overheads are evaluated.

5.2 Baseline Processor

The proposed framework is demonstrated on TigerMIPS. In addition to the basic MIPS units, such as Instruction Fetch (IF), Instruction Decode (ID), Execute (Exe), Memory access (Mem), and Write-back (WB), TigerMIPS comprises advanced units, such as, forwarding unit, branch handling unit, stall logic, and instruction and data caches, which are common in modern pipeline processors.

5.3 Synthesis and Profiling

The baseline model is synthesized in 45 nm NanGate CMOS technology node [34] with Synopsys Design Compiler and evaluated throughout this paper with power supply of 1 volt. Upon completion of the synthesis, triggers are implemented in Verilog HDL, enabling data and timestamp sampling at the input and output of the execution unit within the MIPS pipeline. The profiling is performed based on GLS with Modelsim simulator.

5.4 Integration, Verification, and Evaluation

The trained ML model is first validated in Python. The HDL code of the validated ML model is integrated into the baseline processor. The modified processor is synthesized and its functionality is verified through GLS. The post PAR reports are utilized to evaluate the modified system with respect to specified design constraints.

6 Experimental Results

To demonstrate the framework, LegUp high-level synthesis benchmark suite coupled with LLVM compiler toolchain [35] is utilized for profiling and verification during GLS. The clock frequency of the baseline processor is set to 250 MHz, as determined based on the worst-case propagation delay reported by Synopsis Design Compiler. The trained RF model is tested with nine standard benchmark programs

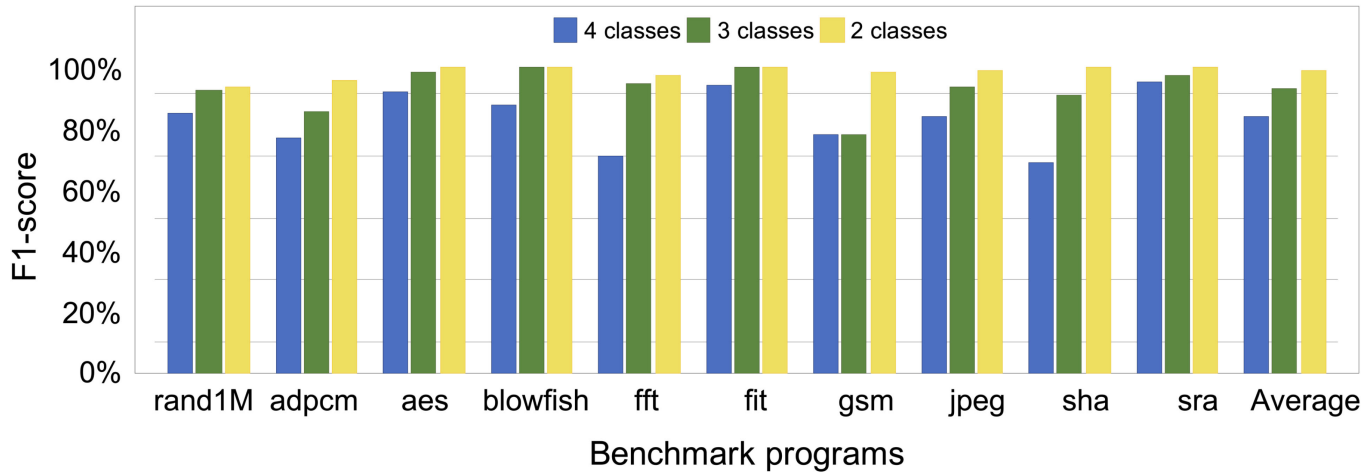

Fig. 8. Inference RF classification based on the LegUp benchmark suite with two, three, and four classes.

available within the LegUp benchmark suite and an additional synthetically generated benchmark with one million random instructions. The F1-score is shown in Fig. 8 for two, three, and four ML delay classes, yielding above 95 percent F1-score for majority of the programs with two delay classes. Resultant speedup for the individual benchmarks is shown in Fig. 9, including the practical speedup (with the misclassification penalty), no-penalty speedup (without the misclassification penalty), and ideal speedup (with 100 percent classification accuracy). The energy overhead due to the additional ML hardware and classification errors is listed in Table 2. To account for delay overheads due to the misclassification of a slow instruction into a higher performance class, a re-execution penalty of four clock cycles (compensating for IF, ID, ML, and EX stages) is considered within the performance results, as reported in Fig. 9. The no-penalty speedup is also presented in Fig. 9, visualizing the penalty due to the misclassification of a fast instruction into a slow class. Note that the overall speedup with four-class configuration is higher than the speedup with two-class configuration, albeit the higher classification accuracy with two delay classes. Alternatively, higher misclassification rate with four delay classes yields higher re-execution energy consumption, as listed in Table 2. Also, note that a negative energy overhead indicates a reduction in the overall energy consumption (i.e., power-delay product).

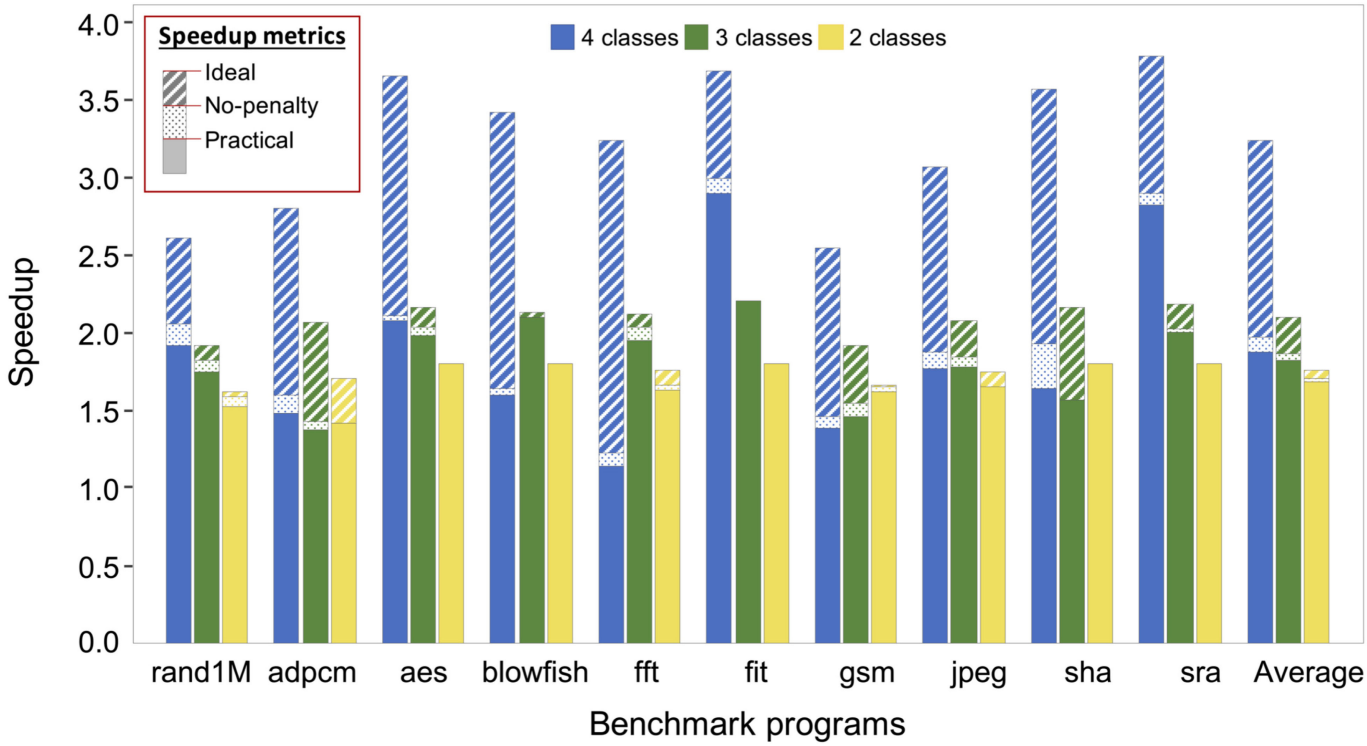

Fig. 9. Experimental speedup with the proposed ML framework with two, three, and four delay classes. Practical, no-penalty, and ideal speedups are presented for each benchmark and class. The practical speedup considers the experimental classification accuracy and delay overheads due to misclassification of a slow instruction into a fast class. The no-penalty speedup considers the experimental accuracy, but disregards the idle time due to misclassification of a fast instruction into a slow class. Finally, the ideal speedup is the theoretical maximum with 100 percent classification accuracy.

TABLE 2
Experimental Power and Energy Overhead of the Proposed ML Method

Benchmark	4 classes			
	Practical speedup	Power overhead	Energy overhead	Instruction count
rand1M	1.923	38.5%	-27.99%	1000000
adpcm	1.497	56.49%	4.55%	30197
aes	2.087	45.14%	-30.46%	11223
blowfish	1.633	65.01%	1.02%	199759
fft	1.165	42.45%	22.28%	11001
fir	2.908	25.94%	-56.7%	7024
gsm	1.382	45.77%	5.45%	7671
jpeg	1.792	55.04%	-13.49%	1133161
sha	1.657	62.07%	-2.22%	345576
sra	2.840	20.62%	-57.53%	1775
Average	1.889	45.7%	-15.51%	274738.7
Positive standard deviation (σ^+)	0.35	5.81%	8.7%	374831.68
Negative standard deviation (σ^+)	0.17	6.58%	15.50%	92682.62
Area overhead	42.1%			
Benchmark	**3 classes**			
	Practical speedup	Power overhead	Energy overhead	Instruction count
rand1M	1.765	23.3%	-30.151%	1000000
adpcm	1.389	33.69%	-3.768%	30197
aes	1.987	27.06%	-36.051%	11223
blowfish	2.125	39.27%	-34.456%	199759
fft	1.957	25.47%	-35.872%	11001
fir	2.222	16.14%	-47.727%	7024
gsm	1.465	27.87%	-12.729%	7671
jpeg	1.786	32.62%	-25.74%	1133161
sha	1.578	37.93%	-12.566%	345576
sra	2.013	7.18%	-46.745%	1775
Average	1.829	27.05%	-28.58%	274738.7
Positive standard deviation (σ^+)	0.11	3.09%	8.41%	374831.68
Negative standard deviation (σ^+)	0.13	5.76%	4.84%	92682.62
Area overhead	29.1%			
Benchmark	**2 classes**			
	Practical speedup	Power overhead	Energy overhead	Instruction count
rand1M	1.530	14.29%	-25.323%	1000000
adpcm	1.418	22.4%	-13.654%	30197
aes	1.818	17.1%	-35.595%	11223
blowfish	1.818	27.23%	-30.024%	199759
fft	1.646	16.14%	-29.45%	11001
fir	1.818	10.5%	-39.225%	7024
gsm	1.635	18.34%	-27.642%	7671
jpeg	1.665	22%	-26.727%	1133161
sha	1.818	27.23%	-30.024%	345576
sra	1.818	5.5%	-41.975%	1775
Average	1.699	18.073%	-29.964%	274738.7
Positive standard deviation (σ^+)	0.05	2.84%	3.49%	374831.68
Negative standard deviation (σ^+)	0.07	3.06%	3.24%	92682.62
Area overhead	20.6%			

TABLE 3
Comparison Between the Proposed Method and Existing State-of-the-Art Methods

Algorithm		Performance gain	Energy overhead	ML based
SLoT [4]		23%	N/A	Yes
Early Prediction [5]		20%	3%	No
Clim [17]		24%	N/A	Yes
SLBM [18]		15%	N/A	Yes
Adaptive Clock Management [27][1]		18.2%	-30.4%	No
This work	2 classes	70%	-30%	Yes
	3 classes	83%	-28.6%	
	4 classes	89%	-15.5%	

[1]*The fabricated chip, as reported in [27], is 5 percent faster than the simulated system. Thus, the results from the fabricated chip are used for fairer comparison.*

Performance comparison between the proposed method and state-of-the-art (ML and non-ML) DVFS approaches is listed in Table 3. For example, both the proposed framework and the approach in [5] consider binary classification with two execution delay classes. The proposed method exhibits on average a 50 percent higher performance gain (i.e., speedup) as compared with [5] and yields 33 percent energy savings whereas a 3 percent energy overhead has been reported in [5]. As compared with the adaptive approach in [27], the proposed method exhibits up to 70 percent higher performance gain when executed with finest granularity (i.e., 4 classes). The particularly high, 70 percent performance gain in this case results in lower energy savings (i.e., 15.5 percent with the four-class configuration as compared with 30.4 percent in [27]). Alternatively, energy savings similar to [27] are possible with the proposed two-delay class configuration, yielding a 50.8 percent higher performance gain than [27].

Power overhead per instruction for two-, three-, and four-delay class configurations are also determined for the programs in the LegUp benchmark suite. The average power overhead (due to the additional ML stage and re-execution of misclassified instructions) is shown in Fig. 10. The average power is linearly reduced with the increasing

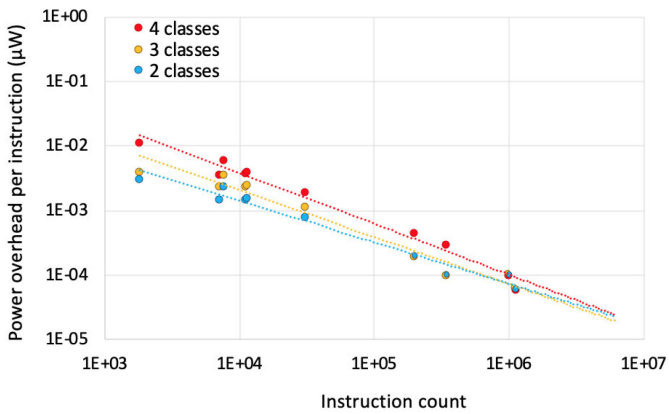

Fig. 10. Power overhead per instruction for two-, three-, and four-delay class configurations based on the benchmarks in Table 2.

number of program instructions, exhibiting an overhead of less than 0.02 microwatts in practical applications with more than one million instructions. Furthermore, the additional average power consumption rapidly converges for various number of classes, as shown in Fig. 10. Thus, when optimizing the number of delay classes in processors with large workload, power overhead is a secondary factor. Finally, the steeper decrease in the power overhead with the four-class configuration supports the previous assertion regarding the gain-overhead tradeoff with finer granularity of delay classes: as the number of instructions increases, the higher accuracy with four-class configuration mitigates the adverse effects of misclassifications on the overall system frequency.

7 Conclusion and Future Work

The proposed unified framework facilitates efficient utilization of the time and hardware resources in the system. In addition, this approach enables the design of ML pipeline stages, while satisfying design constraints, as shown in Fig. 3. Finally, classification of instructions into delay intervals in real time alleviates the path propagation variances imposed by PVT variations and system aging. To enhance the performance gain, the proposed approach should be preferred with those applications and systems characterized by considerable variations in the propagation delay of the individual instructions. For example, RISC processors often exhibit instruction delay disbalance due to variety of independent functionalities typically implemented in RISC based devices.

This method is practical with pipelined, MIPS-like processors, in which the overall delay is dominated by the delay of the execution stage. Although, the proposed method is explored in this work with a single core RISC processor, further increases in energy efficiency and overall system performance are expected if the approach is adjusted for modern architecture processors with out-of-order (OOO) execution and multicore processors with multiple frequency domains. Note that some of the state-of-the-art OOO microprocessors still borrow from the RISC architecture. For example, in high-end OOO CISC ISA processors, the CISC instructions are broken by the ISA interpreter into multiple RISC-like microinstructions [37], increasing the confidence of successfully scaling the proposed approach to these OOO microprocessors. Nevertheless, the challenges of implementing ML-based frequency adjustment in modern high-end OOO processors remain unresolved until carefully studied.

To exploit the positive impact of out-of-order execution and multicore systems on performance and energy efficiency in commercial class processors, the following methodologies are proposed for future investigation.

7.1 Single-Core, Single-Clock Delay-Based Out-of-Order Execution

To support out-of-order execution, instructions within a delay class should be bundled into a delay-class specific reservation station (RS). Instructions stored in an RS are individually executed at a constant frequency until the RS is emptied or a dependency is determined, preventing further execution of instructions in the RS. Such bundling of instructions reduces the number of clock signal transitions among various frequencies, increasing the performance and power efficiency of the system.

7.2 Single-Core, Multi-Clock Delay-Based Out-of-Order Execution

As previously, to support out-of-order execution, instructions should be bundled based on the delay classes and stored within the matching RS's. To support multi-clock execution, the ALUs and FPUs within the execution unit should be operated at different clock frequencies, as determined by the granularity of the delay classes. Intuitively, the parallelization of execution from different delay classes with this approach decreases the number of clock adjustments, increasing the system performance and energy efficiency.

7.3 Multi-Core, Multi-Clock Delay-Based Out-of-Order Execution

To leverage the advantages of processing with multiple clock domains in multicore systems, bundled instructions within the individual clock domains (as defined in Section 7.1) should be shared among all the system clock domains, mitigating the additional cost of multiple clocking (as described in Section 7.2). To enable the sharing of bundles, efficient bundle scheduling and low overhead communication channels are required. While the number of clock adjustments is expected to further reduce with this approach, additional overheads due to intelligent communication of bundles among the cores should be considered. Alternatively, by complementing the traditional DFS, DVFS, and thread scheduling mechanisms with the ML-based frequency adjusting, additional savings are expected with the proposed approach. Finally, the proposed method can be adjusted in a similar manner to classify instruction propagation delay of various pipeline stages.

Existing approaches are focused on offline speculations, statistical models, per-task (workload-based) frequency scaling, and prediction of timing errors at an operating point of a system. Alternatively, the proposed method demonstrates the benefits of fine-grain, instruction-level frequency adjustment, simultaneously utilizing most of the clock period slack and mitigating the adverse effects of PVT variations and aging.

8 Summary

In this work, an additional ML pipeline stage is proposed for increasing the overall system performance by enhancing the temporal resource utilization. This additional stage is designed to classify instructions into propagation delay classes. The system clock frequency is adaptively adjusted based on the individual delay class predictions. Pipelining is exploited to mitigate the effect of the ML stage latency on the overall system performance. Practical ML features are extracted based on current instruction and computation history. ML hardware and misclassification power and delay overheads are considered within the reported results. Tiger-MIPS is utilized as the baseline processor. The processor is enhanced with the ML predictor and simulated with the

LegUp benchmark suite. Based on the experimental results, up to simultaneously 89 percent performance gain and 15.5 percent energy savings are achieved with four delay classes. Alternatively, the reduction of 30 percent in energy consumption with 70 percent performance gain is demonstrated with two delay classes. A unified shell programing platform with peripheral programs is designed to provide a systematic design flow for ML driven pipelined processors.

REFERENCES

[1] B. Fields, R. Bodik, and M. D. Hill, "Slack: Maximizing performance under technological constraints," in *Proc. 29th Annu. Int. Symp. Comput. Archit.*, 2002, pp. 47–58.

[2] V. Zyuban *et al.*, "Integrated analysis of power and performance for pipelined microprocessors," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 1004–1016, Aug. 2004.

[3] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2003, pp. 81–92.

[4] X. Jiao, Y. Jiang, A. Rahimi, and R. K. Gupta, "SLoT: A supervised learning model to predict dynamic timing errors of functional units," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2017, pp. 1183–1188.

[5] S. H. Hashemi, A. F. Ajirlou, M. Soltani, and Z. Navabi, "Early prediction of timing critical instructions in pipeline processor," in *Proc. 15th Biennial Baltic Electron. Conf.*, 2016, pp. 95–98.

[6] I. Moghaddasi, A. Fouman, M. E. Salehi, and M. Kargahi, "Instruction-level NBTI stress estimation and its application in runtime aging prediction for embedded processors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1427–1437, Aug. 2019.

[7] P. Gepner and M. F. Kowalik, "Multi-core processors: New way to achieve high system performance," in *Proc. Int. Symp. Parallel Comput. Electr. Eng.*, 2006, pp. 9–13.

[8] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proc. Int. Symp. Low Power Electron. Des.*, 2004, pp. 32–37.

[9] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Trans. Circuits Syst. I: Fundam. Theory Appl.*, vol. 47, no. 3, pp. 415–420, Mar. 2000.

[10] H. Cherupalli, R. Kumar, and J. Sartori,"Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 671–681, 2016.

[11] J. L. Hennessy and D. A. Patterson,"A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[12] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput CNN inference on embedded ARM big.LITTLE multi-core processors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2254–2267, Oct. 2020.

[13] M. Rapp, M. Sagi, A. Pathania, A. Herkersdorf, and J. Henkel, "Power-and cache-aware task mapping with dynamic power budgeting for many-cores," *IEEE Trans. Comput.*, vol. 69, no. 1, pp. 1–13, Jan. 2020.

[14] C. Isci, A. Buyuktosunoglu, C. Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2006, pp. 347–358.

[15] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proc. 8th Int. Symp. High Perform. Comput. Archit.*, 2002, pp. 29–40.

[16] A. Canis *et al.*, "LegUp: High-level synthesis for FPGA-based processor/accelerator systems," in *Proc. 19th ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2011, pp. 33–36.

[17] X. Jiao *et al.*, "CLIM: A cross-level workload-aware timing error prediction model for functional units," *IEEE Trans. Comput.*, vol. 67, no. 6, pp. 771–783, Jun. 2018.

[18] X. Jiao, A. Rahimi, B. Narayanaswamy, H. Fatemi, J. P. de Gyvez, and R. K. Gupta, "Supervised learning based model for predicting variability-induced timing errors," in *Proc. IEEE 13th Int. New Circuits Syst. Conf.*, 2015, pp. 1–4.

[19] J. J. Zhang and S. Garg, "FATE: Fast and accurate timing error prediction framework for low power DNN accelerator design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2018, pp. 1–8.

[20] J. J. Zhang and S. Garg, "BandiTS: Dynamic timing speculation using multi-armed bandit based optimization," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2017, pp. 922–925.

[21] P. Whittle,"Multi-armed bandits and the Gittins index," *J. Roy. Statist. Soc., B (Methodol.)*, vol. 42, no. 2, pp. 143–149, 1980.

[22] B. Khaleghi, S. Salamat, M. Imani, and T. Rosing, "FPGA energy efficiency by leveraging thermal margin," in *Proc. IEEE 37th Int. Conf. Comput. Des.*, 2019, pp. 376–384.

[23] O. Assare and R. Gupta, "Accurate estimation of program error rate for timing-speculative processors," in *Proc. 56th Annu. Des. Autom. Conf.*, 2019, pp. 1–6.

[24] M. De Kruijf, S. Nomura, and K. Sankaralingam, "A unified model for timing speculation: Evaluating the impact of technology scaling, CMOS design style, and fault recovery mechanism," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2010, pp. 487–496.

[25] Univ. of Cambridge, 2010. [Online]. Available: http://www.cl.cam.ac.uk/teaching/0910/ ECAD+Arch/mips.html

[26] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *The J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[27] T. Jia, Y. Wei, R. Joseph, and J. Gu, "An adaptive clock scheme exploiting instruction-based dynamic timing slack for a GPGPU architecture," *IEEE J. Solid-State Circuits*, vol. 55, no. 8, pp. 2259–2269, Aug. 2020.

[28] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, vol. 112. New York, NY, USA: Springer, 2013, Art. no. 18.

[29] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, vol. 26. New York, NY, USA: Springer, 2013.

[30] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, 1995, vol. 14, pp. 1137–1145.

[31] G. Forman and M. Scholz,"Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement," *ACM SIGKDD Explorations Newslett.*, vol. 12, no. 1, pp. 49–57, 2010.

[32] X. Zhang *et al.*, "A clutter suppression method based on SOM-SMOTE random forest," in *Proc. IEEE Radar Conf.*, 2019, pp. 1–4.

[33] S. W. Cheng, "A high-speed magnitude comparator with small transistor count," in *Proc. 10th IEEE Int. Conf. Electron. Circuits Syst.*, 2003, vol. 3, pp. 1168–1171.

[34] S. Nangate, "California (2008). 45nm open cell library," 2008. [Online]. Available: http://www.nangate.com

[35] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int. Symp. Code Gener. Optim.*, 2004, pp. 75–86.

[36] K. Agarwal, D. Sylvester, and D. Blaauw, "Modeling and analysis of crosstalk noise in coupled RLC interconnects," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 5, pp. 892–901, May 2006.

[37] D. Patterson, "50 years of computer architecture: From the mainframe CPU to the domain-specific tpu and the open RISC-V instruction set," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 27–31.

[38] J. Yue *et al.*, "7.5 A 65nm 0.39-to-140.3 TOPS/W 1-to-12b unified neural network processor using block-circulant-enabled transpose-domain acceleration with 8.1× higher TOPS/mm 2 and 6T HBST-TRAM-based 2D data-reuse architecture," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2019, pp. 138–140.

[39] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H. J. Yoo, "7.7 LNPU: A 25.3 TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2019, pp. 142–144.

[40] Z. Esmaeilbeig, S. Khobahi, and M. Soltanalian, "Deep-RLS: A model-inspired deep learning approach to nonlinear PCA," 2020, *arXiv:2011.07458v2*.

[41] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747v1*.

[42] D. C. Liu and J. Nocedal,"On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, no. 1/3, pp. 503–528, 1989.

[43] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017, *arXiv:1412.6980v9*.
[44] P. Asadi and K. Navi,"A new low power 32× 32-bit multiplier," *World Appl. Sci. J.*, vol. 2, no. 4, pp. 341–347, 2007.
[45] M. Hofmann, "Support vector machines-kernels and the kernel trick," *Notes*, vol. 26, no. 3, pp. 1–16, 2006.
[46] J. Mitéran, S. Bouillant, and E. Bourennane, "Classification boundary approximation by using combination of training steps for real-time image segmentation," in *Proc. Int. Workshop Mach. Learn. Data Mining Pattern Recognit.*, 2003, pp. 141–155.
[47] A. Kulkarni, Y. Pino, and T. Mohsenin, "SVM-based real-time hardware Trojan detection for many-core platform," in *Proc. 17th Int. Symp. Quality Electron. Des.*, 2016, pp. 362–367.

Arash Fouman Ajirlou (Student Member, IEEE) received the BSc degree in computer engineering from the University of Tehran, Tehran, Iran, in 2017. He is currently working toward the PhD degree with the Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois, in 2018. He was a research assistant at the School of Electrical and Computer Engineering, University of Tehran, Iran, between 2015 and late 2017. From 2017 to late 2018, he served as the secretary of the Electrical and Computer Engineering committee at Alumni Association of Faculty of Engineering, University of Tehran, Iran. In 2018, prior to starting his PhD, he was a digital designer with the Engineering Department of Ofogh Tajrobe Moj Company, Tehran, Iran. His primary interests include embedded systems and high-performance/low-power computing systems, with an emphasis on machine learning and self governing systems. His current focus is on utilizing machine learning methodologies to enhance processor performance and energy consumption.

Inna Partin-Vaisband (Member, IEEE) received the BSc degree in computer engineering and the MSc degree in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, respectively, in 2006 and 2009, and the PhD degree in electrical engineering from the University of Rochester, Rochester, New York, in 2015. She is currently an assistant professor with the Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois. Between 2003 and 2009, she held a variety of software and hardware R&D positions at Tower Semiconductor Ltd., GConnect Ltd., and IBM Ltd., all in Israel. Her primary interests include the area of high performance integrated circuits and VLSI system design. Her research is currently focused on innovation in the areas of AI hardware and hardware security. Yet another primary focus is on distributed power delivery and locally intelligent power management that facilitates performance scalability in heterogeneous ultra-large scale integrated systems. Special emphasis is placed on developing robust frameworks across levels of design abstraction for complex heterogeneous integrated systems. She is an associate editor of the *Microelectronics Journal* and has served on the technical program and organization committees of various conferences.