Learning-Based Workload Phase Classification and Prediction Using Performance Monitoring Counters

Erika S. Alcorta and Andreas Gerstlauer
Electrical and Computer Engineering, The University of Texas at Austin
{esalcort, gerstl}@utexas.edu

***Abstract*—Predicting coarse-grain variations in workload behavior during execution is essential for dynamic resource optimization of processor systems. Researchers have proposed various methods to first classify workloads into phases and then learn their long-term phase behavior to predict and anticipate phase changes. Early studies on phase prediction proposed table-based phase predictors. More recently, simple learning-based techniques such as decision trees have been explored. However, more recent advances in machine learning have not been applied to phase prediction so far. Furthermore, existing phase predictors have been studied only in connection with specific phase classifiers even though there is a wide range of classification methods. Early work in phase classification proposed various clustering methods that required access to source code. Some later studies used performance monitoring counters, but they only evaluated classifiers for specific contexts such as thermal modeling.**

In this work, we perform a comprehensive study of source-oblivious phase classification and prediction methods using hardware counters. We adapt classification techniques that were used with different inputs in the past and compare them to state-of-the-art hardware counter based classifiers. We further evaluate the accuracy of various phase predictors when coupled with different phase classifiers and evaluate a range of advanced machine learning techniques, including SVMs and LSTMs for workload phase prediction. We apply classification and prediction approaches to SPEC workloads running on an Intel Core-i9 platform. Results show that a two-level kmeans clustering combined with SVM-based phase change prediction provides the best tradeoff between accuracy and long-term stability. Additionally, the SVM predictor reduces the average prediction error by 80% when compared to a table-based predictor.

***Index Terms*—workload phase prediction, phase classification**

I. Introduction

Predicting program behaviors has been a crucial step for runtime optimizations such as dynamic cache reconfiguration [14] or power and thermal management [12]. Predictions allow runtime systems to make proactive decisions instead of reactive ones. It has been shown that proactive decisions yield better optimization results [1]. However, proactive approaches are challenging because they require predicting the future.

Programs are known to go through phases that typically show repetitive patterns. Researchers have proposed predicting phase patterns to achieve proactive runtime optimizations [3], [7], [9], [11], [15]. This requires two main tasks: (1) identifying the phases, i.e., phase classification, and (2) learning their patterns to anticipate changes, i.e., phase prediction.

Phase classification requires selecting a set of inputs that can characterize phases and a clustering method that groups similar inputs together. The inputs proposed in prior studies are very diverse. Some of them focus on defining similar blocks of code while others measure the system's workload with performance metrics. Finding similar blocks of code typically requires accessing the source code [2] or augmenting the hardware [16], which limits their applicability. By contrast, performance metrics can be obtained transparently by sampling the hardware counters available in most modern processors. Prior work has proposed various methods to classify these hardware counters into phases [3], [11], [18]. However, there is no comprehensive study to evaluate the best counter-based clustering method. In this work, we adapt phase classification methods previously used with other inputs to support hardware counters and compare them to existing counter-based solutions.

Classified phases are used by phase prediction methods to learn patterns between them and foretell future phase behavior. Early work proposed methods that predicted the phase ID of the next sample interval [6]. Later studies realized that many phases last for multiple consecutive intervals and proposed other prediction strategies. A recent approach predicts the phase IDs of a window of several upcoming samples [2], showing that decision trees are superior to different variations of maximum likelihood predictions. An alternative strategy consists of using run-length encoding to compress phase information and predict both the duration of a phase and the ID of the next phase, where previous work proposed using table-based predictors for this purpose [14]. To the best of our knowledge, no existing work has, however, investigated application of more advanced machine learning approaches in the context of either window- or change-based phase prediction. Furthermore, the accuracy of phase prediction can be strongly influenced by the quality of phase classification. However, existing phase predictors have only been studied in limited and specific combinations with phase classifiers.

In this work, we study hardware counter-based phase classification and prediction tasks both in isolation and in different combinations. We first compare several classification techniques using hardware counters and characterize the phases that they output. We then evaluate the performance of multiple predictors against different classifiers and compare the performance of traditional table-based predictors and decision trees to more advanced machine learning techniques, specifically support vector machines (SVM) and long-short term memory (LSTM). Finally, we assess whether the classification technique influences their accuracy and discuss the best-performing pairs.

In summary, the contributions of this paper are as follows:

1) We perform a comprehensive study of different clustering algorithms to assess their performance in classifying

978-1-6654-3166-8/21/$31.00 ©2021 IEEE

TABLE I
SUMMARY OF EXISTING PHASE CLASSIFIERS.

Ref	Inputs	Granularity	Classifier
[13]	Basic block frequency vectors	100M instructions	Random linear projection and k-means clustering
[6]	Hardware counters	100M instructions	Static value ranges
[3], [11]	Hardware counters	100 ms	Principal component analysis and k-means clustering
[4]	Frequency vectors of branch instruction pointers	100M instructions	Leader-Follower clustering with centroid learning
[16]	Instruction type frequency vectors	10M instructions	Two-level past footprints table
[5]	Frequency vectors of hashed EIP samples	100M instructions	Phase history table
[14]	Signature of instruction counts between branches	10M and 50M instructions	Signature tables and similarity threshold
[18]	Hardware counters	W * 1 ms	Two-level k-means clustering
[2]	Frequency vector of executed branches	10,000 branches	Gaussian Mixture Model clustering

TABLE II
SUMMARY OF EXISTING PHASE PREDICTORS.

Ref	Predicts	Inputs	Predictor	Classifier
[6]	Next sample	Previous samples	Global history table	[6]
[17]	Next sample	Previous sub-phases	Precursor vectors and distance threshold	[18]
[14]	Next sample, next phase, phase duration	Previous samples	Multilevel phase history table	[14]
[12]	Phase duration	Previous durations	Linear adaptive filter	[16]
[2]	Next k samples	Branch frequency vectors	Decision tree	[2]

program phases with hardware counters. Our study includes phase classification techniques that, to the best of our knowledge, have not previously been adopted for clustering hardware counters into phases.

2) We evaluate the outcomes of phase classifiers in combination with multiple phase predictors to find the pair that results in the most accurate phase predictions. This evaluation includes advanced machine learning-based prediction techniques that had not been investigated for phase prediction before.
3) We perform studies with large-scale workloads from the SPEC benchmark suite running on a single core of a state-of-the-art desktop-class Intel Core i9-9900k workstation. Results show that the best classification and prediction combination is a two-level kmeans clustering approach combined with an SVM to predict phase ID changes and another SVM to predict phase duration.

The remainder of this paper is organized as follows: Section II discusses related work. Sections III and IV describe details of the evaluated classifiers and predictors, respectively. Section V presents results. Finally, in Section VI, we discuss our conclusions and future work.

II. RELATED WORK

In this section, we discuss prior work related to phase classification and prediction. We summarize our survey of various representative phase classifiers and predictors in Table I and Table II, respectively.

The set of inputs and granularity that researchers have used to define a phase is very diverse. Examples of these are shown in the second and third column of Table I. Some researchers aim at finding similar blocks of code [2], [4], [5], [8], [10], [13], [14], [16], while others focus on execution time workload metrics [3], [6], [18]. Previous work has observed that most performance metrics are highly correlated with the executed code [4]; therefore, the resulting classified phases are very similar. Finding similar blocks of code typically requires either access to the source code [2], [13], [14] or augmenting the hardware [8], [10], [16]. To avoid this limitation, we focus on using hardware counters as the input data to the phase classifiers that we evaluate in this work.

A list of various classification methods used in prior work is shown in the last column of Table I. Some studies have looked at methods that can perform clustering in an incremental and online fashion [4], [5], [10], [14], [16] while others propose iterative clustering methods [2], [3], [13], [18]. Incremental approaches have not been used with hardware counters in the past. Their online implementation is cheaper than iterative approaches, and they do not require knowing the number of phases that a workload has. We apply the technique from [4] to classification with hardware counters. In [6], a predefined static set of value ranges are used to bin workloads based on their memory boundedness. The ranges are relevant to dynamic power management (DPM) as their target use case, and require manual definition. A hierarchical, multilevel phase approach is presented in [14] and [18]. In [14] two separate table-based classifiers are used with different granularities to define fine and coarse phases. In [18] k-means is used to generate one sub-phase per sample and the sub-phases are then grouped into fixed-size windows to define phases with a second instance of k-means. This approach may generate more stable phases, but it can also lose precision when detecting phase changes. Out of these, we include the approach from [18] in our evaluation.

Prior research has followed multiple strategies to predict upcoming phases at runtime. We survey various predictors in Table II. Traditionally, table-based predictors like global history table (GHT) [6] and Markov chain tables [5], [8], [14] were used. Most of these predictors foretell the phase of the following interval. Many researchers have noticed that the same phase may last multiple intervals. Consequently, previous work has also proposed either predicting the duration of phases [5], [12], [14] and the phase ID of the next change [5], [14] or predicting multiple future intervals instead of just one [2]. Predicting the exact duration of a phase is a challenging task. Chang et al. [5] approximated the duration

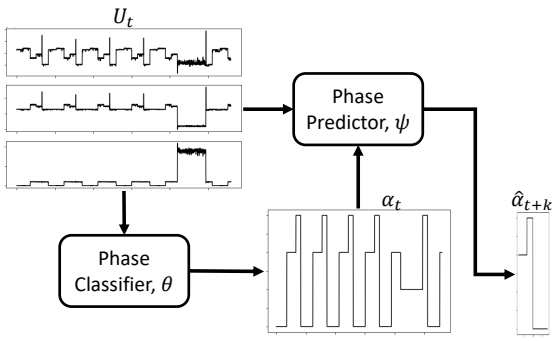

Fig. 1. Overview of phase classification and prediction.

of phases into ranges, limiting the precision of a prediction of when a change will happen. Srinivasan et al. [12] proposed to use a linear adaptive filter that learns and predicts the duration of phases online. Other researchers have proposed predicting the phase of a window of upcoming sample intervals instead. For example, in [2] a decision tree predictor is trained offline to predict the phase labels of the following 30 intervals with high accuracy. Each phase predictor has been evaluated with a single classification scheme. In this work, we look at how different classifiers impact their accuracies. Additionally, we propose other machine learning-based prediction techniques that have not been evaluated for these tasks in the past.

III. Phase Classification

Fig. 1 shows an overview of phase classification and prediction using performance monitoring counters. Phase classification is the first step that uses samples of a trace of hardware counters and determines their phase classification. Formally, we refer to the trace of hardware counter samples as $U \in \mathbb{R}^{T \times M}$, where T is the number of observations of M hardware counters. We use U_t to denote the vector of hardware counters observations at time t. Additionally, we refer to any phase classifier, Θ, as a function that maps each sample U_t in the input trace to a series of discrete phase IDs α_t.

In the following, we describe the different phase classifiers that we evaluate in this work. We adopt some classifiers shown in Table I and modify the inputs of those that do not use hardware counters.

a) Table-based: Several classifiers use a table to incrementally cluster samples as they arrive in a streaming fashion. They are variations of the leader-follower clustering algorithm. This method facilitates an online implementation and eliminates the need to provide the number of clusters as input. It requires a distance metric, a threshold, and an updating policy as input parameters. Some variations limit the maximum number of clusters and use eviction policies such as LRU to add new clusters. We did not find this necessary for our workloads; instead, we can control the threshold to limit the number of clusters. As suggested by prior work, we implement this algorithm using Manhattan distance and sweep different thresholds to find an optimized threshold value. In the rest of the paper we refer to this classifier as *table*.

b) Two-level k-means: This method uses the same approach as in [18]. The workloads are classified into sub-phases and phases. The granularity of sub-phases is one sample, while the granularity of a phase consists of a fixed size number of sub-phases, W. We used the same maximum number of sub-phases as in [18], $N_1 = 10$. We sweep different values of W to select an optimized value that minimizes the average CPI corrected coefficient of variation (CCoV). Finally, we need to select a number of phases per benchmark. We execute k-means multiple times, incrementing the number of clusters until the CPI CCoV is no longer decreased by more than 5%. In the rest of this paper we refer to this classifier as *2kmeans*

c) PCA with k-means: This method first reduces the dimensionality of the input space with principal component analysis (PCA). Then, the k-means clustering algorithm is applied to the reduced space. We follow the same approach that we use to select the number of phases of *2kmeans*. In the rest of this paper we refer to this classifier as *pcakmeans*

d) Gaussian Mixture Model: This is an iterative clustering algorithm that finds its clusters in a similar way as k-means does. It assumes that combining different Gaussian models creates the data points in a cluster. Therefore, in addition to considering the cluster means, it also considers their variances when assigning samples to clusters. This algorithm requires the number of clusters as an input. We follow the same approach that we used in other iterative algorithms. We refer to this classification method as *gmm* in the rest of this paper.

e) Manual Oracle Classifier: We inspected the workloads manually and assigned phase labels based on our observations while prioritizing long-term phase stability over short-term variations. We use it as an oracle to compare against other classifiers and to evaluate phase predictors accuracy. We call it *manual* in the rest of this paper.

IV. Phase Prediction

As shown in Fig. 1, a phase predictor takes the history of phased IDs at the output of the phase classifier or a sub-trace of past hardware counter samples to predict the phase ID of one or more future samples. We denote these predictions as $\hat{\alpha}_{t+k}$, $k \geq 1$. Different phase prediction strategies use as input either a window of the collected samples U_t, or a window of classified phases, α_t. We separate the phase prediction strategies into two groups: window phase prediction and phase change prediction. This section presents the predictors that we propose for each strategy.

A. Window-Based Phase Prediction

The goal of window-based prediction is to accurately determine the phase ID of the immediately upcoming sample window of size k. These predictors can either look at the history of phase IDs or look at the history of pre-classification data, in our case, hardware counters. Some workload phases last for very long periods, and using the history of phase IDs may not be helpful information to the predictor. Therefore, all our window predictors use the trace of hardware counters as inputs. The assumption is that the predictors will find relationships between the counters' short-term variations and the upcoming phase IDs. Note that while predictions do not

directly utilize the output of phase classifiers, phase classification ground truth is needed to train the predictors. Formally, the window-based phase prediction problem is defined as: $(\hat{\alpha}_{t+1}, ..., \hat{\alpha}_{t+k}) = \Psi\big((U_{t-h+1}, ..., U_t)\big)$, where Ψ is the phase prediction model, h is the input window size, and $\hat{\alpha}_{t+i}$ denotes the predicted phase IDs i steps into the future at time t.

For this task, we propose using LSTMs and SVMs. LSTMs are known for their robust capabilities of handling time-dependent data. Our models use a single-layer LSTM to encode the history of hardware counters, followed by a fully connected layer that classifies the phase ID of the upcoming k data samples using one one-hot encoded vector per sample. For the SVM, we transform the input trace into a vector and train k SVM models. We compare this LSTM to another learning-based structure used in previous work, the decision tree (DT).

B. Phase Change Prediction

The same phase ID may last multiple consecutive intervals. Phase change prediction focuses on learning and predicting phase transitions. To do so, the series α_t is transformed using run-length encoding to generate a new series, C_j. The new series consists of pairs $C_j = (\alpha_j, d_j)$ of phase IDs α_j and their duration d_j. At each detected phase transition, j, we use separate models to predict the ID of the next phase, $\hat{\alpha}_{j+1}$ and how far into the future the transition to this next phase will happen, i.e., the duration of the current phase α_j, $\hat{d}_j$

1) Next Phase Prediction: These predictors look at the history of different phases as a sequence α_j to predict the ID of the next phase $\hat{\alpha}_{j+1}$. For workloads with uniform phase patterns that repeat without variations, table-based predictors can trivially solve this problem. However, when the pattern is not entirely uniform, more robust predictors that can generalize the input data can yield better results. For this reason, we propose to apply support vector machines (SVM), whose design objective is to generalize unseen data rather than just fitting the training set, as well as an LSTM network, due to its ability to predict sequences, for this task. Both models take a history of one-hot encoded phase IDs as inputs. We compare these approaches to a global history table (GHT) based prediction technique as baseline.

2) Phase Duration Prediction: Phase duration prediction aims at accurately determining how long a phase will last. We frame the duration prediction as a regression problem. We consider two types of inputs. The first is to use the duration history of past phase α_i with the same phase ID α_j, $(d_i | i < j, \alpha_i = \alpha_j)$. The second is using the history of phase IDs and their duration, $C_i, i < j$. The best choice depends on the predictor and the workload. In general, we find that the relationships between durations of the same phase are linear; therefore, linear predictors are better suited for these inputs. Some workloads may have relationships between the duration of different phases, and in general, these inter-phase relationships may not be linear. We propose to use a linear SVM and a multi-layer perceptron (MLP). Previous work proposed a linear filter for duration predictions. Since we train the SVM and MLP models offline, we compare the performance of SVM and MLP to a linear regression (LR) model. Additionally, we implement a baseline which predicts the duration to be the same as the previous execution of phase ID α_j, i.e., last value (LV) prediction.

V. Experiments and Results

We evaluate our approach on a desktop-class Intel Core i9-9900K running Debian 9.13. We execute workloads on one core and collect data every 10ms using Intel's EMON tool. The Intel hardware can sample 4 fixed counters and up to 4 variable counters simultaneously. We select the four variable counters to characterize the workload by its memory boundedness (L2 misses and main memory read accesses), control flow predictability (mispredicted branch instructions), and operation mix (floating-point operations).

To generate the workload traces, we used the SPEC CPU 2017 benchmark suite. We selected a subset of workloads that show distinctive phases with several transitions throughout their execution to allow sufficient data for training and validation. The workloads that we used and their corresponding total number of samples are *cactuBSSN* with 149,597, *fotonik3d* with 91,977, *mcf* with 47,834, *nab* with 234,753, *pop2* with 167,548, and *wrf* with 296,786. We used the *ref* inputs given by the benchmark suite. We split the traces into 70% of samples for training and 30% for testing and comparison of predictors. The training set is further split into 30% of the samples used as a validation set to tune hyperparameters.

A. Phase Classification

Phase classifiers require selecting hyperparameters to optimize their output. The *table* classifier takes the distance threshold as an external input, and the iterative classifiers take the number of clusters. To determine the threshold of the table classifier, we evaluated the trade-off between the CPI CCoV and the distinct number of phases. We selected a threshold value of 1 that achieves the best tradeoff. To obtain an optimized number of clusters for iterative classifiers, we followed the approach described in Section III. This resulted in a different optimized number of phases for each benchmark and classifier summarized in Table III.

To evaluate the phase classifiers, we use the corrected coefficient of variation (CCoV) metric introduced in [4]. Additionally, we use the average phase duration $\bar{d}_j$ in number of samples as a measure of long-term stability. Table III also shows the CCoV metric for each classifier and benchmark. We observe that *table* can achieve a CCoV that is similar to iterative classifiers *gmm* and *pcakmeans* at the expense of requiring a larger number of phases. Iterative classifiers, therefore, yield a more efficient classification. However, they trade off the easy online implementation that *table* provides. Notice that *2kmeans* yields the highest CCoV on average with *cactuBSSN* having the highest CCoV overall. This benchmark has short regions where the CPI is significantly higher than the rest of the trace. When the fixed-size windows of *2kmeans* miss these transitions, it assigns them to surrounding contiguous regions, causing the increase in CCoV.

TABLE III
SUMMARY OF PHASE CLASSIFICATION RESULTS.

	manual			2kmeans			gmm			pcakmeans			table		
	CCoV	d_j	# ph.	CCoV	d_j	# ph.	CCoV	d_j	# ph.	CCoV	d_j	# ph.	CCoV	d_j	# ph.
cactuBSSN	0.17	111.0	7	4.72	105.5	5	0.17	76.9	4	0.25	72.4	4	0.22	35.96	9
fotonik3d	0.34	24.12	6	0.4	24.1	3	0.26	13.63	3	0.23	13.94	4	0.25	6.56	10
mcf	0.19	430.0	7	0.24	67.63	3	0.18	11.4	3	0.16	8.17	4	0.14	9.84	11
nab	0.03	358.7	5	0.03	41.8	7	0.02	215.3	5	0.01	133.7	6	0.12	236.7	10
pop2	0.05	42.26	4	0.08	10.14	4	0.04	24.06	3	0.06	1.36	5	0.08	5.32	12
wrf	0.36	16.07	3	0.26	6.17	5	0.24	3.08	3	0.27	3.29	3	0.18	1.27	14
AVERAGE	0.19	163.7	5.33	0.95	42.55	4.5	0.15	57.39	3.5	0.16	38.8	4.33	0.16	49.27	11.0

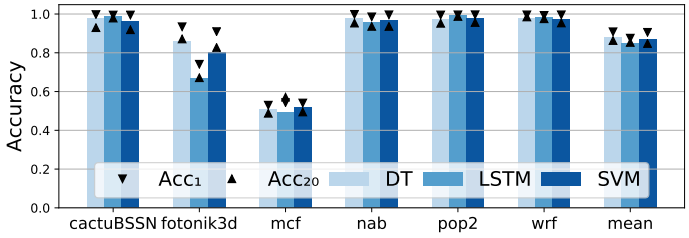

Fig. 2. Accuracy of 20-step window predictors trained with manual classifier.

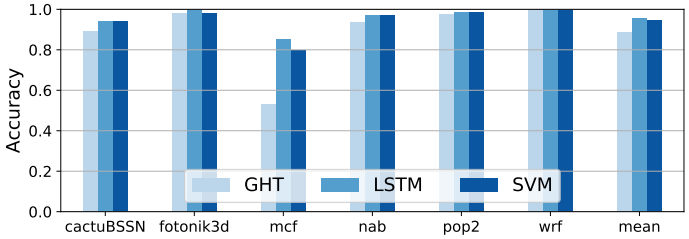

Fig. 3. Accuracy of phase change predictors trained with manual classifier.

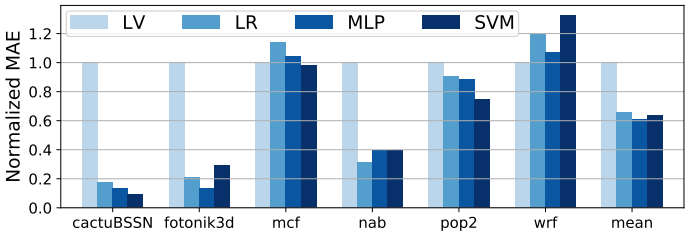

Fig. 4. Normalized MAE of duration predictors with manual classifier.

The best performing classifier in terms of average CCoV is *gmm*, closely followed by *pcakmeans* and *table*. They are all performing better than *manual* in terms of CCoV. However, *manual* trades off variability for long-term stability. This is noticeable in the average duration of phases, where *manual* is 2.8x longer than the average duration of *gmm*.

B. Phase Prediction

To evaluate phase predictors, we use accuracy for window and next phase predictors and mean absolute error (MAE) for duration predictors. We average the accuracy across all outputs of window predictors as a comparison metric. We also measured the inference time of a software implementation for each model to evaluate their runtime complexities.

1) Window Prediction: We studied the models' hyperparameter space using our validation sets and determined that they performed best with an input window size of $h = 20$ steps. Additionally, we find that a tree depth of 8 for DT, a single-layer LSTM with 128 neurons, and a linear SVM return the best validation accuracy. Fig. 2 shows the average accuracy of window-based predictors across the test set when using phases obtained from a manual classifier. All three models have very similar accuracies across benchmarks, except for *fotonik3d*. This trace has some minor variations towards the end. They cause the LSTM model to predict false phase changes, while DT and SVM are more robust to these variations. The poorest performing benchmark is *mcf*. The workload behavior changes continuously throughout its execution, which makes the workload generally very hard to predict. None of the models were able to learn such changes. The fastest model is DT with an average inference time of 1.8ms, followed by SVM with 2.2 ms, and lastly, LSTM with 3.8ms.

2) Phase Change Prediction: We evaluated GHT, LSTM, and SVM models for phase change prediction. We determined that a history of 10 phases as input yields the most accurate results in the validation sets. We also studied hyperparameter spaces for LSTM and SVM. A single-layer LSTM with 40 cells and a polynomial kernel for SVM return the highest average validation accuracy. Fig. 3 displays the test set results. The higher accuracy results of LSTM and SVM showcase how learning-based techniques generalize historical data. The most significant gap in the accuracy of GHT and learning-based techniques is shown by *mcf*. Its phase sequence suffers some changes over time, causing the models' accuracy to be significantly lower than in other benchmarks. By contrast, *wrf*'s phase sequence remains unchanged throughout its execution, resulting in 100% accuracy for all models. Inference times and accuracy have opposite trends. The fastest model is GHT (0.05ms), followed by SVM (0.07ms), and LSTM (1.03ms).

3) Phase Duration Prediction: We evaluate three models for duration prediction: LR, MLP, and SVM. With the validation set, we determined that the best input history for each model is different. For LR, we use the duration of 5 previous executions of the target phase; for SVM, the duration of the 5 previous phases of any type and the duration of the prior execution of the target phase; and for MLP, the duration of 3 previous executions of the target phase and the duration of the last phase. We used a LV predictor as baseline. To visualize the results in the same scale for all benchmarks, we plot the MAE of each predictor normalized against LV MAE for the test set in Fig. 4. The MAE of LV is 12.29, 14.26, 101.04, 11.56, 0.76, and 0.47 for *cactuBSSN*, *fotonik3d*, *mcf*, *nab*, *pop2*, and *wrf*, respectively. All models improved the prediction accuracy over LV. On average, MLP is the best performing model, closely followed by SVM. In *mcf*, the duration of phases gets

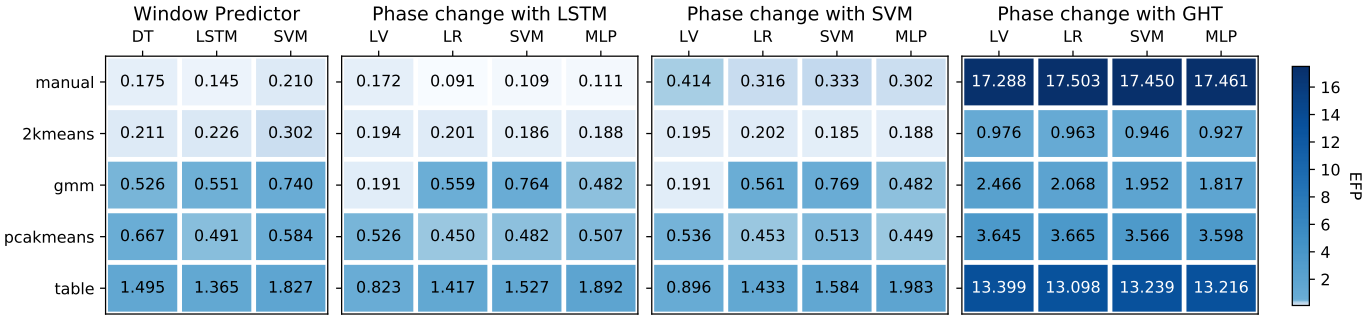

Fig. 5. EFP of all classifiers and predictors.

significantly shorter over time, and the models struggled to improve over LV. LR is the fastest with 1.1ms inference time, followed by SVM and MLP with 1.3ms and 1.6ms.

C. Combined Classification and Prediction

To compare combinations of classifiers and predictors, we introduce error-frequency product (EFP) as a metric that penalizes both prediction inaccuracies and classification overfitting:

$$EFP = 100 \cdot \frac{|\frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)|}{\bar{d}_j}, \quad (1)$$

where N is the number of predicted samples and 100 is a scale factor. We select CPI as our variable of interest, y. To determine the error, we use a vector $(v_1, ..., v_c)$, where c is the corresponding total number of phases and each element v_i represents the average CPI that characterizes phase i. At each timestep t, we transform the prediction of phase $\hat{\alpha}_t$ to a prediction $\hat{y}_t = v_{\hat{\alpha}_t}$ and measure the error $y_t - \hat{y}_t$. We normalize average errors by the average phase duration $\bar{d}_j$. We define EFP as the product of the average error and the average transition frequency $1/\bar{d}_j$.

The EFP of all combinations of classifiers and predictors is shown as a heatmap in Fig. 5. For window predictors, the columns correspond to the models that we studied. For phase change predictors, we evaluated all combinations of next phase and duration prediction. The individual column labels correspond to the duration predictor and the shared titles correspond to the next phase predictors. Without considering the *manual* classifier, the best combination of classification and prediction that reduces the EFP is *2kmeans* with an SVM for next phase and duration prediction. *2kmeans* resulted is the best classifier for 13 out of 15 predictors, performing sometimes even better than *manual*. The most remarkable difference between *2kmeans* and the other classifiers is its two-level clustering approach which considers a window of samples instead of a single sample to define phases. Out of all the classifiers, we observe that *table* tends to have the worst EFP values. When comparing window and phase change predictors, a phase change predictor is always best across all classifiers. For the manual classifier, the LSTM model for next phase prediction shows strong superiority over the SVM model. However, such a clear trend is not exhibited by other classifiers. Clearly, GHT shows the highest EFP values for any classifier and duration prediction. This shows that learning-based predictors are more suitable for workload prediction.

VI. Summary and Conclusions

In this paper, we performed a comprehensive study of hardware counter-based workload phase classification and prediction techniques. We showed that learning-based techniques result in more accurate results than traditional table-based methods. Our isolated studies concluded that GMM was the best classifier in terms of minimizing both CPI variations between phases and phase transitions. However, when we studied classification in combination with phase prediction, results showed that a two-level kmeans clustering approach is the best option for 87% of the predictors that we studied. Future work includes evaluating multi-core settings and applying phase prediction to various use cases.

Acknowledgments

This work was supported by NSF grant CCF-1763848.

References

[1] C. Ababei and M. Moghaddam. A survey of prediction and classification techniques in multicore processor systems. *IEEE TPDS*, 2018.
[2] M.-C. Chiu and E. Moss. Run-time program-specific phase prediction for Python programs. In *ManLang*, 2018.
[3] R. Cochran and S. Reda. Consistent runtime thermal prediction and control through workload phase detection. In *DAC*, 2010.
[4] A. Sembrant et al. Efficient software-based online phase classification. In *IEEE IISWC*, 2011.
[5] C.-H. Chang et al. Sampling-based phase classification and prediction for multi-threaded program execution on multi-core architectures. In *ICPP*, 2013.
[6] C. Isci et al. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *MICRO*, 2006.
[7] E. Alcorta et al. Phase-aware cpu workload forecasting. In *SAMOS*, 2021.
[8] J. Lau et al. Transition phase classification and prediction. In *HPCA*, 2005.
[9] M. Rapp et al. Neural network-based performance prediction for task migration on s-nuca many-cores. *IEEE TC*, 2020.
[10] M. T. Cruz et al. Unsupervised variable-grained online phase clustering for heterogeneous/morphable processors. In *HPCS*, 2016.
[11] R. Khanna et al. Phase-aware predictive thermal modeling for proactive load-balancing of compute clusters. In *ICEAC*, 2012.
[12] S. Srinivasan et al. Program phase duration prediction and its application to fine-grain power management. In *ISVLSI*, 2013.
[13] T. Sherwood et al. Automatically characterizing large scale program behavior. In *ASPLOS*, 2002.
[14] W. Zhang et al. Multilevel phase analysis. *ACM TECS*, 2015.
[15] Y. Kim et al. P^4: Phase-based power/performance prediction of heterogeneous systems via neural networks. In *ICCAD*, 2017.
[16] O. Khan and S. Kundu. Microvisor: A runtime architecture for thermal management in chip multiprocessors. In *Transactions on HIPEAC*. 2011.
[17] S. Khoshbakht and N. Dimopoulos. Execution phase prediction based on phase precursors and locality. In *E2SC*, 2017.
[18] S. Khoshbakht and N. Dimopoulos. A new approach to detecting execution phases using performance monitoring counters. In *ARCS*, 2017.