Accurate Phase-Level Cross-Platform Power and Performance Estimation

Xinnian Zheng, Lizy K. John, Andreas Gerstlauer
The University of Texas at Austin, Austin, TX, USA
xzheng1@utexas.edu, {ljohn, gerstl}@ece.utexas.edu

ABSTRACT

Fast and accurate performance and power prediction is a key challenge in co-development of hardware and software. Traditional analytical or simulation-based approaches are often too inaccurate or slow. In this work, we propose LACross, a novel learning-based, analytical cross-platform prediction framework that provides fast and accurate estimation of time-varying software performance and power consumption on a target hardware platform. We employ a fine-grained phase-based approach, where the learning algorithm synthesizes analytical proxy models that predict the performance and power of the workload in each program phase from performance statistics obtained through hardware counter measurements on the host. Our learning approach relies on a one-time training phase using a target reference model or real hardware. We applied our approach to 35 benchmarks from SPEC 2006, SD-VBS and MiBench. Results show on average over 97% prediction accuracy for predicting both fine-grain performance and power traces at speeds of over 500 MIPS.

1 Introduction

One of the core challenges in system designs is the necessity for fast and accurate prediction of both performance and power consumption of real world applications and benchmarks. Applications often exhibit significant power and performance variations, where estimation of time-varying performance and power traces can provide useful information for optimization of both hardware and software.

Traditionally, hardware models are presented to software developers in the form of cycle-accurate or functional virtual platforms and instruction set simulators (ISSs) [3, 15]. Even though such models can very accurately estimate the software performance, simulation speeds are typically very slow, which severely limits the amount of exploration that software or hardware developers can perform. By contrast, analytical models of processors are fast, but typically not as accurate. Moreover, performance or power models constructed via analytical techniques are often exclusively targeted at design space exploration for existing hardware and a given set of benchmarks. Such models require execution traces or statistics obtained by running the actual workload in question either on a partial ISS model [14] or on a physical realization of a close micro-architecture variant [13]. By contrast, applications can run significantly faster on real machines. Intuitively, there exists a latent relationship between execution time of a program on different machines. An interesting question is thus whether a few example runs on the slow, detailed simulator and the corresponding runs on some other real hardware can give insight into the correlation between the two [22].

In this paper, we propose a novel learning-based, analytical framework for such cross-platform prediction, *LACross*, that is capable of fast and accurate estimation of both performance and power at fine temporal granularities. It is known that programs tend to exhibit more homogeneous behavior at the individual phase level [20]. This motivates us to pursue an approach that performs performance and power prediction at the granularity of program phases. With a proper choice of phases, temporal variations in large scale program behavior can be accurately captured. This can provide hardware and software developers with fine-grain information about the location of performance and power hotspots of the program as it would execute on a target system. Hardware developers can use such architecture proxies trained on small micro-benchmarks to evaluate time-varying behavior of large, real-world applications not otherwise possible when only slow pre-silicon models are available in early development stages. Similarly, learning models trained on real implementations or other reference models can be given to external software developers without requiring access to target hardware or exposing architecture internals.

The rest of the paper is organized as follows: after an overview of our approach in Section 2, Section 3 surveys related work. Section 4 describes the formulation of our performance and power prediction problem. Section 5 then discusses the experimental setup, and Section 6 presents results of our prediction framework. Finally, Section 7 concludes with a summary of the key contributions and results.

2 Overview

An overview of the proposed LACross approach is shown in Figure 1. The learning-based formulation of the performance and power prediction problem consists of two stages: a training stage and a prediction stage. During the training stage, a set of sample programs ("training set") are executed both on the host machine ("host") and a reference target model ("target"). The reference model could be either a simulator or real physical hardware, such as a development board. The target and the host do not necessarily have to be of similar architectures. In fact, as our results will show, it is possible to achieve accurate prediction between targets and hosts that are of vastly different micro-architectures and instruction set architectures (ISA).

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2897977

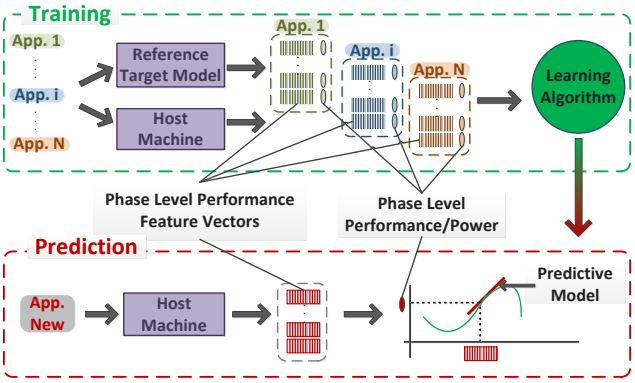


Figure 1: LACross overview.

For each workload we obtain, at phase level, various hardware performance features from the host as well as reference performance and power from the target. Our goal is to extract the latent relationship between the host and target. We formulate this problem into a statistical learning setting, and derive prediction models for both performance and power on the target. During the prediction stage, a new application is executed only on the host. A set of performance features is obtained at phase level and used as inputs to the prediction model in order to produce an estimate of the performance and power on the target.

3 Related Work

Traditional simulation-based approaches estimate performance of a program by executing it on cycle-accurate or cycle-approximate ISSs [3, 7, 15]. The main drawback is speed, as throughput of most ISS is on the order of several hundred KIPS to several MIPS. Hardware-assisted acceleration [8] and source-level, host-compiled and transaction level modeling (TLM) techniques [5] have recently been proposed for improving simulation speed while trying to maintain accuracy close to an ISS. Throughput of these higher-level approaches is often 200-500 MIPS when including cache simulation, while accuracy is often above 90%. However, they typically involve static or dynamic back-annotation of source code or hardware models with simulated target performance estimates. With the effect of compiler optimization and out-of-order hardware execution, back-annotation becomes difficult to track, which inherently limits its accuracy.

Early analytical models [19, 21] focused on the evaluation and study of microarchitectural variations on pipeline and instruction-level parallelism. More recently, statistical and regression-based methodologies have emerged. Bircher *et al.* introduced techniques using linear regression for predicting power consumption from performance counters [4]. McCullough *et al.* [16] survey counter based power modeling techniques and show their limitations in modeling complex systems. Lee and Brooks proposed a predictive modeling and spatial sampling method [13] for efficient microarchitecture design space exploration. Joseph *et al.* [11] also utilized regression-based approaches to construct processor performance models. Similar ideas were also introduced by Ipek *et al.* [10] using artificial neural networks instead of regression. However, all of these approaches try to obtain statistical performance models for some target architecture of interest from measurements performed on the same base architecture. By contrast, we aim to provide performance and power prediction by establishing analytical models that correlate two distinct architectures. Our previous work [22] had similar goals. However, prior work was limited to predicting performance of whole programs only, where errors of more than 40% were shown for small embedded benchmarks on simple in-order targets. By contrast, the approach in this paper supports estimation of both power and performance at fine temporal granularities with more than 95% accuracy when targeting complex benchmarks and architectures.

4 Problem Formulation

Many possible granularities for defining program phases have been proposed [9, 20]. In our approach, we define the program phases in units of dynamic basic blocks, and we study the effect of different granularities on prediction.

We apply a variant of a LASSO linear regression to our performance and power prediction problem with two key differences. We impose extra constraints on the model parameters and we perform linear regression on a phase-specific basis. We formulate this as a phase-level simplex-constrained quadratic programming (SCQP) problem. For each workload, we obtain feature vectors consisting of selected hardware performance counter measurements for every program phase. We make the assumption that the model in each phase follows a linear relationship with its features. We then apply SCQP to obtain a linear predictive model specific for each program phase by correlating the host performance features with target performance and power.

Formally, let m be the total number of program phases in the training set. Define $X = (x_1, \cdots, x_m)$, and $Y = (y_1, \cdots, y_m)$, where $x_j \in \mathbb{R}^d$ denotes the performance feature vector obtained from the host for program phase j and $y_j \in \mathbb{R}$ its corresponding cycles or power from the reference simulator or hardware. Similarly, for a given test program with n program phases, let $V = (v_1, \cdots, v_n)$ and $\Theta = (\theta_1, \cdots, \theta_n)$ denote the sets of performance feature vectors $v_i \in \mathbb{R}^d$ at phase i and unknowns $\theta_i \in \mathbb{R}^d$ corresponding to parameters of the linear hyperplane at each v_i that need to be determined. The predicted performance or power at phase i is then computed as $v_i^T \theta_i$. The overall prediction for the entire program can be computed as $tr(V^T \Theta)$.

In each phase i, SCQP determines the corresponding θ_i by solving the following optimization problem,

$$\begin{aligned} \underset{\theta_i}{\text{minimize}} \quad & \frac{1}{2n} \|X^T \theta_i - Y\|^2_{D(v_i)} \\ \text{subject to} \quad & \|\theta_i\|_1 \leq T \text{ and } \theta_i \geq 0, \end{aligned} \tag{4.1}$$

where T is a tuning parameter and $\|p\|_A$ denotes the matrix induced norm of vector p by matrix A (i.e $\|p\|_A = \sqrt{p^T A p}$). $D(v_i) \in \mathbb{R}^{m \times m}$ is a scaling matrix with diagonal entries $D(v_i)_{jj} = \mathbf{1}\{\|x_j - v_i\|_2 \leq \epsilon\}$ and $\mathbf{1}\{\}$ being the standard indicator function, where ϵ is a tuning parameter.

Intuitively, the quadratic objective function in (4.1) aims to minimize the prediction error in each program phase by considering only the feature vectors in the training set that are close to v_i as determined by a l_2-distance threshold of ϵ. Since feature vectors that are close to each other are more likely to exhibit similar performance patterns across different architectures, we impose the distance constraint such that only relevant feature vectors in the training set are considered when forming the prediction model for each phase. The l_1-norm constraint on the hyperplane parameter

Table 1: Performance counters profiled on the host.

L1 Total Cache Misses	L1 Total Cache Accesses
L2 Total Cache Misses	L2 Total Cache Accesses
L3 Total Cache Misses	L3 Total Cache Accesses
TLB Loads	Cycle Stalled
Unconditional Branches	Conditional Branches
Branch Misses	Floating Point Operations
Instructions	Cycles

θ_i restricts solutions to be small in order to avoid overfitting, and the positivity constraint states that all the performance features on the host should contribute positively to the performance or power on the target. These type of constraints are commonly known as simplex constraints.

The SCQP problem (4.1) does not have an analytical solution. In fact, it can be transformed to a particular type of convex optimization problem for which the objective function is a quadratic plus a convex but non-smooth function (the l_1-norm constraint). The solution can be computed efficiently by first-order iterative algorithms, such as proximal gradient methods [18].

Notice that in the SCQP, we need to choose two tuning parameters ϵ and T. We employ a standard technique known as cross-validation [12] to determine their values. In particular, we randomly choose a subset of the original training data set and divide it into a training subset and a test subset. We train using only data from the training subset, and we use data from the test subset to compute an average prediction error percentage. We iteratively repeat this process applying different values for T and ϵ until the cross-validation error is less than 5%.

During prediction, the constructed models for all unique per-phase feature vectors θ_i are cached, such that (4.1) does not need to be solved repeatedly for the same weight matrix $D(v_i)$. Two feature vectors v_j and v_k are defined to be unique iff $\|v_j - v_k\|_\infty \geq L$, where the threshold L is empirically chosen to be 200. A threshold of 200 is found to be enough for filtering out the inherent noise in processor performance counter based phase measurements.

5 Prediction Infrastructure

In the following section, we describe the implementation of our training, measurement and prediction framework.

5.1 Training Set

We use a similar training setup as in [22] for explicit comparison of prediction accuracy. Our training set consists of 157 diverse and representative sample programs from the ACM-ICPC (International Collegiate Programming Contest) database. We use original programs and inputs. In prior work, the size of the training set was artificially increased to improve coverage by creating 100 random inputs for each program. This is not necessary in our case. Profiling programs at phase granularity provides sufficient amount of training data, and no addition of artificial and possibly unrepresentative data is necessary. With our new phase-based approach, a small training set with low training overhead is sufficient to achieve high accuracies.

5.2 Profiling and Measurement Setup

We perform profiling of the training set on a host platform with an Intel Core i7-920 processor and 24GB of memory. To demonstrate the effectiveness of our approach on state-of-the-art mobile and embedded target platforms, we employ physical hardware references as targets for training and prediction. We use the ODROID-U3 [1] (U3) development board to obtain reference performance measurements (cycle counts). The U3 board uses the Samsung Exynos 4412 SoC as its hardware platform. The Exynos 4412 SoC contains a homogeneous quad-core ARM Cortex-A9 processor with 32 KB L1 instruction and data cache. For reference power measurements, we use the ODROID-XU3 [2] (XU3) development board. It uses the Samsung Exynos 5422 SoC as the hardware platform. The Exynos 5422 SoC consists of a heterogeneous big.LITTLE CPU arrangement, which combines a Cortex-A15 (A15) and a Cortex-A7 (A7) processor cluster. The XU3 development board integrates an on-chip TI INA231 current sensor for power measurements of all eight cores, but the CPU hardware is restricted to not allow any performance counter measurements. Thus, two different boards serve as performance and power references due to hardware limitations associated with each board.

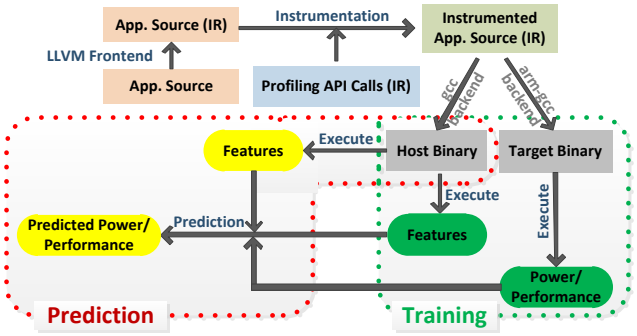


Figure 2: Prediction framework.

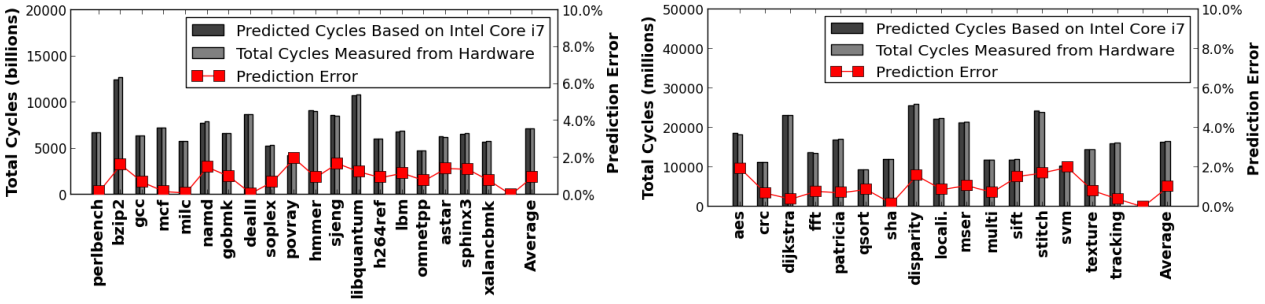
For our study, we are mainly interested in predicting performance and power for single-core workloads. Thus, all programs are restricted to run on one core till completion, which minimizes measurement noise due to core migration. On the XU3 board, we restrict the workloads to run solely on one of the A15 processors with DVFS disabled.

We use the PAPI toolset [17] for collecting a total of 14 hardware performance counters, shown in Table 1, on the Intel host. For power measurement on the target, we developed a custom API that interacts directly with the onboard TI INA231 power sensor.

5.3 Prediction Framework

Figure 2 shows the tool flow used in our LACross framework for fine-grain phase-level profiling and prediction. We utilize the LLVM compiler infrastructure to instrument profiling API calls at IR basic block level of each program during compilation. The application sources are first compiled into LLVM IR and then instrumented and linked against the profiling API. The instrumented LLVM IR tracks the number of dynamic basic blocks executed by the program in order to log various counter and power measurements at the end of each program phase. Instrumentation at the IR level thereby guarantees that features and reference performance and power obtained for each phase correspond to the same execution on both the host and the target.

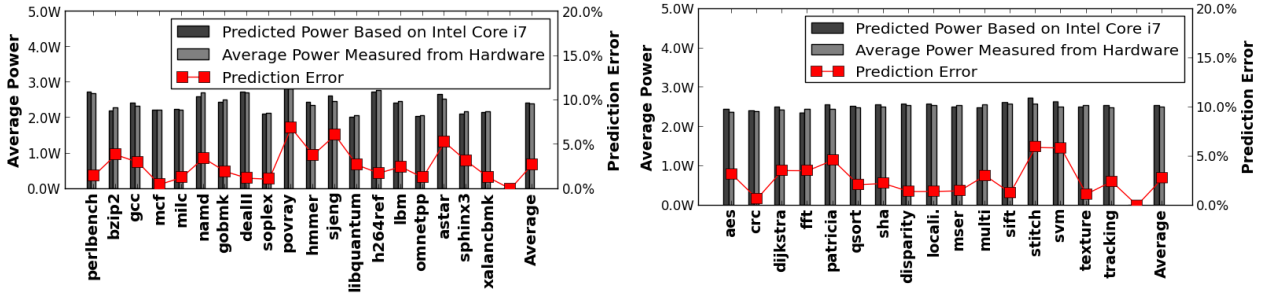
During training, the instrumented IR is cross-compiled into host and target binaries. During prediction, we obtain the performance features on the host for each program phase, and we use previously collected training data to solve Eq. (4.1) and predict per-phase performance and power. We use MATLAB 2013a as the main computation environment.



(a) 19 SPEC CPU programs.

(b) 16 Mibench and SD-VBS programs.

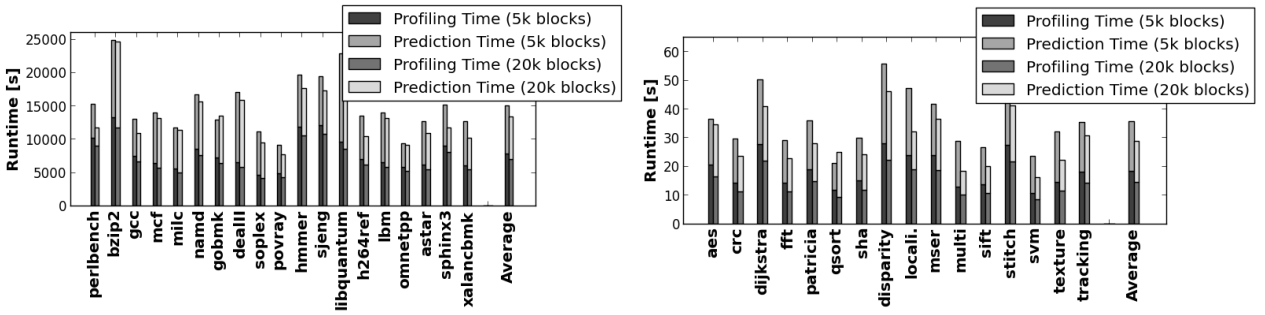
Figure 3: Predicted target cycles and prediction accuracy of 35 benchmarks (phase granularity = 5,000 blocks).



(a) 19 SPEC CPU programs.

(b) 16 Mibench and SD-VBS programs.

Figure 4: Predicted target power and prediction accuracy of 35 benchmarks (phase granularity = 20,000 blocks).



(a) 19 SPEC CPU programs.

(b) 16 Mibench and SD-VBS programs.

Figure 5: Runtime of 35 benchmarks.

6 Experiments and Results

To show the effectiveness of the proposed LACross framework, we test it with 35 selected benchmark programs from three standard benchmark suites that are not encountered in the training set. For comparison with [22], we use the same seven programs (**aes, crc, dijkstra, fft, patricia, qsort, sha**) from the MiBench suite and nine programs (**disparity, localization, mser, multi_ncut, sift, stitch, svm, texture_synthesis, tracking**) from the San Diego Vision Benchmark Suite (SD-BVS). Furthermore, we apply our approach to the prediction of 19 programs (**perlbench, bzip2, gcc, mcf, milc, namd, gobmk, dealII, soplex, povray, hmmer, sjeng, libquantum, h264ref, lbm, omnetpp, astar, sphinx3, xalancbmk**) from SPEC CPU 2006. We chose the 19 programs from SPEC implemented in C/C++ as we use the C/C++ interface provided in PAPI to instrument counter profiling calls. We use the "large", "fullhd" and "ref" input sets for programs from MiBench, SD-VBS and SPEC CPU 2006, respectively.

These 35 programs are first profiled on the Intel host to obtain hardware performance feature vectors at program phase level using the PAPI toolset. We then conduct performance and power predictions using previously trained models for the U3 and XU3 boards, respectively. In order to study the effect of phase granularity on the prediction accuracy and speed, we perform our experiments at different phase sizes. Note that for power prediction, due to the hardware constraint on the sampling period of the TI INA 231 current sensor on the XU3 development board, we found a phase granularity of approximately 20,000 basic blocks to be the fastest the sensor hardware could support.

6.1 Overall Prediction Accuracy and Speed

Figure 3 shows the accuracy of predicting whole program performance for the 35 test programs at a phase granularity of 5,000 basic blocks. Predicted cycles are very close to the actual cycle measurements obtained on physical hardware. The worst case prediction error is around 2%, with average errors less than 1%. Note that the accuracy we refer to here is the percentage prediction accuracy of whole program performance. By comparison, an average error of more than 5% and errors as large as 40% are reported in [22] for SD-

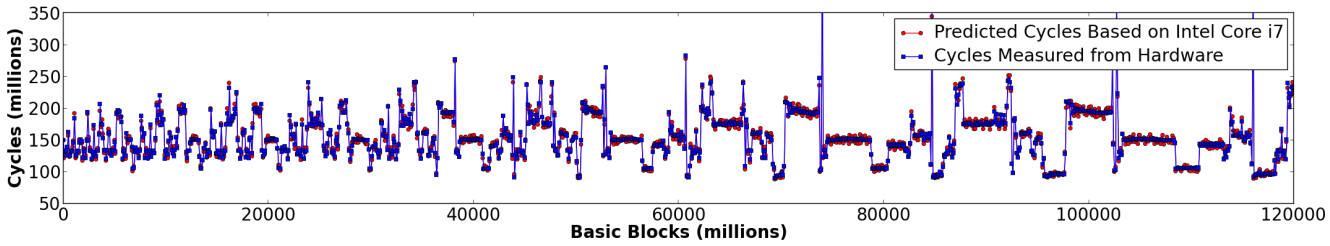


Figure 6: Fine-grained performance behavior of **dealII** on U3 target. (Prediction vs. Actual)

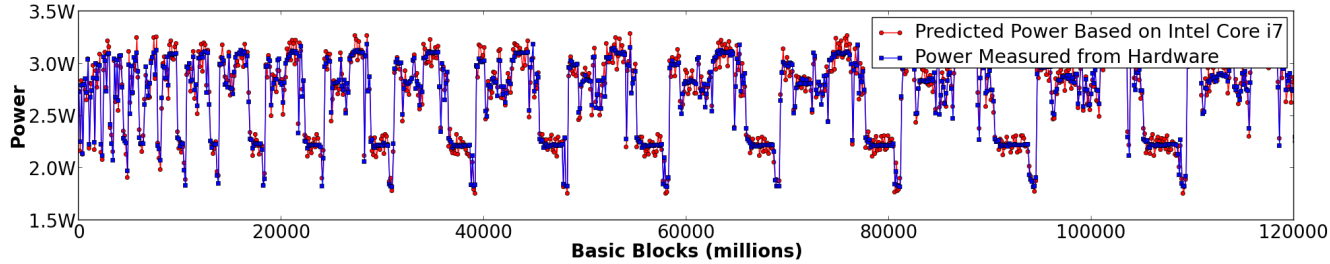


Figure 7: Fine-grained power behavior of **dealII** on XU3 target. (Prediction vs. Actual)

VBS and MiBench benchmarks under a larger training set.

Figure 4 similarly shows the overall program-wise power prediction accuracy of the 35 programs from the test set, profiled at a phase granularity of 20,000 basic blocks. The average error for predicting average power over whole programs is about 2.5%, while the worst-case prediction error is less than 10%.

The total runtime of each test program, as shown in Figure 5, consists of the profiling time and the prediction time. The profiling time is the time it takes to collect various counters on the host. Hardware limitations on the simultaneous usage of counters necessiate 5 separate runs of the program on the Intel host to obtain the 14 features. The prediction time measures the total duration of solving the optimization problem (4.1) for each phase of the program. Solving time is governed by the dimension of data matrix X and of the neighborhood defined by distance threshold ϵ in the problem formulation (4.1). For the same phase granularity, comparing runtimes of the SPEC programs (Fig. 5b) with the MiBench and SD-VBS programs (Fig. 5a), we see that as programs execute longer, the number of dynamic phases grows proportionally, which results in more time spent in solving the optimization problem (4.1). As phase granularity increases from 5,000 blocks to 20,000 blocks, the average total runtime generally decreases due to a decrease in both the profiling and prediction time. As the sampling granularity of program phases becomes larger, the total number of dynamic phases decreases accordingly, and thus fewer variables θ need to be solved for each program. Note that prediction complexities vary across applications due to differences in the convergence speed of solving (4.1) numerically. At the same time, the sampling of the performance counters via the PAPI toolset also incurs performance overheads [6]. Hence, as the number of dynamic phases decreases, the overhead of profiling also becomes smaller.

We demonstrate an example of fine-grained dynamic power and performance tracing. Figure 6 and 7, show the dynamic behavior of executing the **dealII** benchmark on the predicted and real targets. Here, we use a phase granularity of 20,000 basic blocks. Results show that the prediction tracks accurately against performance and power measurements obtained from the hardware.

6.2 Phase Granularity Tradeoffs

As indicated above, the choice of phase granularity will influence prediction accuracy and speed. A finer phase granularity can potentially increase training set coverage and thus improve accuracy. Finer phase granularity, however, also requires more frequent profiling and prediction. We further study the accuracy-speed tradeoff with respect to different choices of phase granularity.

To measure simulation speed, we use the total number of dynamic instructions in a target program divided by the total time it takes to obtain the prediction on the host (i.e, profiling of the performance feature vectors plus the time spent on solving the SCQP (4.1) for all phases of the program). To measure accuracy, we use the mean absolute percentage error (MAPE) between the prediction and the actual measurement across all phases and programs.

As shown in Figure 8, the overall accuracy and speed for predicting workload performance varies significantly with respect to the choice of phase granularity. At a granularity of 5,000 basic blocks, per-phase prediction accuracy for performance is about 92%. Note that due to averaging effects, when aggregating all the phases to obtain the overall performance of an entire program, as shown in Section 6.1 and Figures 3 and 4, accuracy is considerably higher.

At a granularity of 500 blocks, phase-level predictions are about 95% accurate as compared to real hardware measurements. As the phase granularity gradually increases from 500 to 20,000 basic blocks, the prediction accuracy only experiences a minor decreasing trend. The diminishing returns at finer and finer granularities are likely due to the instability of performance counter measurements at very small sampling periods. Such noisy measurements can deviate the SCQP from the nominal target function. When the phase granularity grows beyond 50,000 basic blocks, however, the accuracy drops drastically due to the lack of coverage in the training data. This is consistent with prior work [22], where large errors are seen when performing prediction at the whole program level despite using a much larger training set. At the same time, the simulation speed improves proportionally with a decrease in phase granularity. As such, there is an optimal tradeoff between speed and accuracy at medium granularities.

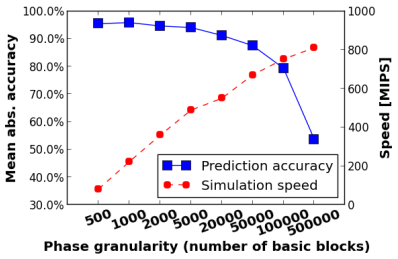


Figure 8: Speed and accuracy tradeoff (performance prediction).

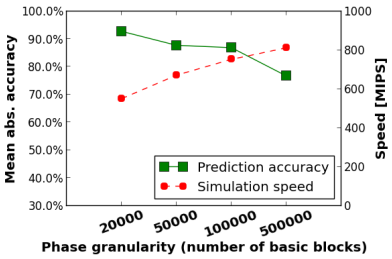


Figure 9: Speed and accuracy tradeoff (power prediction).

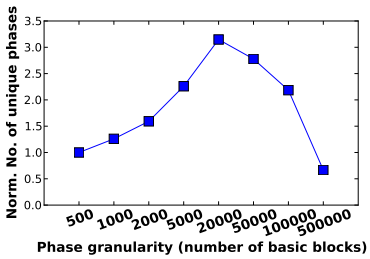


Figure 10: Normalized total number of unique phases.

For power prediction (Figure 9), a similar tradeoff is observed. Our method, however, is still accurate (over 90% phase-wise accuracy compared to real hardware measurements) and fast (over 500 MIPS). Again, due to error cancellation effects, at the same phase granularity, accuracy of predicting average power for a whole program is better than the average absolute per-phase prediction accuracy.

6.3 Program Phase Homogeneity

Prediction accuracy correlates closely with training set coverage. However, it is not clear whether better training set coverage at finer granularities is a result of the homogeneous nature of performance patterns in smaller program phases or alternatively simply from having more training data when sampling more frequently. To help clarify this question, we show in Figure 10 the total number of unique dynamic program phases across all training set programs as a function of the phase granularity. As the phase granularity increases, the total number of unique phases increases. This indicates that more diverse performance patterns emerge as increasingly larger chunks of a program are encapsulated in one phase. Conversely, as program phases become more fine-grained, even though the total number of phases increases, the total number of unique phases decreases. This indicates that the improvement in the coverage of the training set at smaller granularities is due to an increase in homogeneity in the performance patterns as phases becomes smaller.

Note that as granularity continues to increase, the total number of unique phases would also be expected to continue to increase. However, since all programs complete their execution in a finite amount of time, the number of dynamic phases is finite. Thus, the total number of unique phases eventually decreases when the phase granularity becomes too large. This agrees with the downward trend observed in Figure 10 for phase granularities greater than 20,000.

7 Summary and Conclusions

This paper proposes LACross, a unified learning-based framework for fast and accurate phase-level cross-platform prediction of performance and power of a workload running on a target machine. The key idea is the simple observation that performance and power consumption of an application running on two different platforms are correlated, even if the two platforms are of vastly different architectures. We further show that constructing prediction models at program phase level helps to resolve training set coverage issues and thus increases accuracy. With proper choice of phase granularity, the prediction achieves over 97% accuracy at speeds over 500 MIPS for both performance and power. This is orders of magnitude faster than traditional cycle-accurate simulation, with the drawback that it requires application source code to be available. Predictions based on an x86 host can thereby run at almost the same speeds as executions on physical ARM hardware. With better hardware counter support in the hosts, prediction speeds could be even higher.

Acknowledgments

This work is supported by SRC grant 2012-HJ-2317. We would also like to thank the anonymous reviewers for their helpful suggestions to improve the paper.

8 References

[1] ODROID U3 Development Board. http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275.

[2] ODROID XU3 Development Board. http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127.

[3] N. Binkert et al. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[4] W. Bircher et al. Runtime identification of microprocessor energy saving opportunities. In *ISLPED*, 2005.

[5] O. Bringmann et al. The next generation of virtual prototyping: Ultra-fast yet accurate simulation of HW/SW systems. In *DATE*, 2015.

[6] S. Browne et al. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, 2000.

[7] T. E. Carlson et al. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC*, 2011.

[8] D. Chiou et al. FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators. In *MICRO*, 2007.

[9] M. Huang et al. A framework for dynamic energy efficiency and temperature management. In *MICRO*, 2000.

[10] E. Ipek and S. A. Mckee. Efficiently exploring architectural design spaces via predictive modeling. In *ASPLOS*, 2006.

[11] P. J. Joseph. A predictive performance model for superscalar processors. In *MICRO*, 2006.

[12] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, 1995.

[13] B. C. Lee et al. CPR: Composable performance regression for scalable multiprocessor models, 2008.

[14] S. Li et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.

[15] P. S. Magnusson et al. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, 2002.

[16] J. C. McCullough et al. Evaluating the effectiveness of model-based power characterization. In *USENIX*, 2011.

[17] P. J. Mucci et al. PAPI: A portable interface to hardware performance counters. In *DoD HPCMP*, 1999.

[18] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.

[19] D. B. Noonburg and J. P. Shen. Theoretical modeling of superscalar processor performance. In *MICRO*, 1994.

[20] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *PACT*, 2001.

[21] D. J. Sorin et al. Analytic evaluation of shared-memory systems with ILP processors. In *ISCA*, 1998.

[22] X. Zheng et al. Learning-based analytical cross-platform performance prediction. In *SAMOS*, 2015.