Fast IPC Estimation for Performance Projections Using Proxy Suites and Decision Trees

Kanishka Lahiri and Subhash Kunnoth
Advanced Micro Devices, Bangalore

Abstract—**Accurate IPC estimates are critical for generating performance projections of key workloads on future designs. However, the need to respond to projections requests in a timely manner in the face of rapidly evolving applications and software stacks and tight schedule constraints, often preclude design teams from executing detailed workload analysis, sampling and simulation flows for such purposes.**

We address this problem by taking advantage of the large amount of data that performance modeling teams commonly generate as part of architectural studies across thousands of workload scenarios. We propose two methods for exploiting these datasets: one that builds proxy suites, and another that builds decision-tree based classifiers. Both methods can generate IPC estimates for a target workload without collecting new workload samples, or running a single additional simulation. We discuss our experience using these techniques to estimate the IPC of numerous commercial workloads on four industrial x86 processor designs. The resulting IPC estimates were on average, within 2% of those obtained via measurements or detailed cycle-accurate simulations Importantly, using these methods, we were able to generate IPC estimates for a target workload in a matter of hours to 1-2 days, compared to several weeks using conventional approaches.

I. Introduction

As high-performance general-purpose processors go through various stages of the design cycle (concept, high-level design, detailed design, tape-out and post-silicon bring up), a key requirement is to track (or project) the performance of the design across a set of workloads. These projections are used to raise flags when performance targets are at risk of not being met, help position the product in the market, and enable early customer engagement. Also, projections play a key role during post-silicon performance tuning of the design, where they serve as reference performance targets. Since general-purpose processors exhibit significant variation in performance depending on workload characteristics, it is critical that workloads used for performance projections be both diverse and contemporaneous, to effectively safeguard the design team from post-silicon surprises.

Design teams commonly use a large variety of workloads to drive microarchitectural feature evaluation using detailed, cycle-accurate simulators. Each workload is represented by a suite of simulator inputs, where each input is a fixed-length instruction trace that represents a unique phase of workload behavior. While these suites are often used to project performance of the design, they could fail to meet contemporaneity requirements. This is because generating high-quality traces for a workload is a substantial amount of work, and can take weeks or months, depending on workload complexity. This makes it impractical to rely solely on traces to track the performance of the design in the face of rapidly evolving applications, and software stacks. In addition, product marketing teams may request performance projections on specific workloads that are of particular interest from a business standpoint that may not be part of existing trace suites. The expected turn-around time for generating such answers typically preclude tracing and simulation.

For conventional CPU and memory bound workloads, the projections problem consists of estimating (i) the average IPC (Instructions Per Cycle) and (ii) the effective clock frequency at which the workload will run on the target design. In this work, we focus on the first problem, *i.e.,* estimating a workload's IPC on a target design, but without the luxury of having corresponding simulator inputs.

A. Contributions and Paper Overview

Over a period of time, processor design groups accumulate a large number of trace suites that represent a variety of workloads which can be simulated on cycle-accurate performance models. For a new workload that is of interest from a projections standpoint, we pose the following two questions: "Does this workload have behaviors that we have seen before?" and "Do we need to trace this workload in order to estimate it's IPC?" In this paper, we present two systematic approaches that address these questions and demonstrate that indeed, for the purposes of IPC estimation, careful analysis of existing datasets can prove highly advantageous. The specific contributions we make are the following:

- We motivate the importance of per-workload performance projections using results from cycle-accurate models of two designs, and explain why conventional approaches to workload sampling and simulation can sometimes be impractical.
- We present a flow that analyzes existing simulation data using similarity analysis to automatically generate a "proxy suite" that can be used to estimate IPC (and other micro-architectural statistics) for a workload of interest.
- We describe a more general flow that analyzes existing simulation data to construct classifier-based models

that predict IPC based on easily measurable workload characteristics.

Both these methods have been deployed in an industrial performance modeling environment for commercial x86 processors. We show that in an substantial number of cases, we were able to generate IPC estimates that were within 2% of those obtained either via measurement or via cycle-accurate simulation. These methods are extremely easy to deploy and use, and have computational needs that are trivial compared to simulation. In our experience, generating IPC estimates for a new workload on a target architecture requires a few hours to 1-2 days of work, most of which is consumed in setting up and running the workload on a baseline system.

In the next section we discuss related work. In Section III, we motivate the need for per-workload IPC projections. In Section IV we describe why conventional approaches can be difficult to use in a real-world projections context. In Section V we describe the various learning techniques we used for our work. In Section VI we describe the proxy suite methodology in detail. In Section VII we describe the more general flow that uses decision-tree based classifiers. Section VIII compares and contrasts the two methods. Section IX summarizes our learnings, and discusses avenues of future research.

II. Related Work

The problem of how to sample workloads for architecture design has been extensively studied, and contributions such as [1], [2] have greatly increased our understanding of how similarity of phase behaviors can be effectively exploited. Variations of these methods are widely used in practice. Our work is among a few research efforts that take the observation around similarity a step further, by exploiting similarity in phase behaviors *across* workloads, not just within them. This idea was first explored in [3] in which the authors exploited similarity across constituent programs and data sets to efficiently forecast performance of an entire benchmark suite. Our proxy suite approach goes even further by including traces from *any* previously traced workload. Also, we focus on workload-specific, rather than suite-level performance, and describe our success (and failures) in deploying such methods in a production environment.

In [4] the authors show the value of using previous knowledge of the link between workload characteristics and performance to determine the performance of a new workload, and emphasize the importance of data transformations such as Principal Components Analysis [14] and genetic mutations. These techniques can be (and are) directly applied in our flow too. However the authors do not consider phase behavior or micro-architecture specific workload characteristics. Instead they correlate program-level, architecture-independent workload characteristics with performance. This is reasonable given their goal was to estimate performance of a fixed program across highly diverse systems with potentially different ISAs, processor architecture, and machine configuration, spanning a huge performance space.

Our decision-tree based approach has similarities with [5], [6], [7], [8]. The focus of these efforts is to develop flows that enable design space exploration of many alternative *architectural* configurations. Our problem is more constrained, which allows us to generate performance estimates on the target architecture without any additional simulations, which is not the case in many of these approaches. In fact, our focus is on a problem somewhat opposite to a lot of prior work. In our case, the space of *workloads* is potentially large, but the space of microarchitecture definitions we care about are very limited. So understanding and accounting for dynamic program behaviors, and taking advantage of microarchitecture-specific statistics, are critical for achieving high accuracy.

Recent work ([9], [10]) has shown how to determine pair-wise relative performance ranking in a huge design space, rather than precisely quantify the absolute performance of a workload on a target architecture. This is different from us, where we aim at estimating actual IPC values for a limited set of design points (only 4 in our paper) for which detailed performance simulations of other workloads already exist.

In [11] the authors characterize workload behavior by correlating different types of resource stalls with IPC, and collect this data on a variety of architectures. For a new workload they use similarity analysis to predict performance across the set of architectures of interest. The focus of their work to group programs that have similar performance bottlenecks rather than actually predict workload performance.

A key feature of our work is that the proposed methods are very easy to deploy since they are built on datasets that are *already available with any processor design team*. In recent work, the applicability of learning-based modeling methods (including decision-trees) were recently studied in the context of power modeling [12], highlighting the usefulness of such techniques.

III. Variations in IPC Uplift

When faced with a projections task on a new workload, one may be tempted to use an average IPC uplift based on the large number of performance simulations that performance teams habitually run. The risk of such an approach is illustrated in Figure 1. The graph shows the IPC uplift across over a thousand workload scenarios generated using internal cycle-accurate performance models for two high-performance x86 processor designs. These scenarios cover numerous applications across client and server domains. We observe that the range of uplift is large: while some workloads actually lose a small amount of IPC, others gain as much as 45%. Hence, using a flat IPC uplift when projecting performance for a workload of interest can clearly result in large errors.

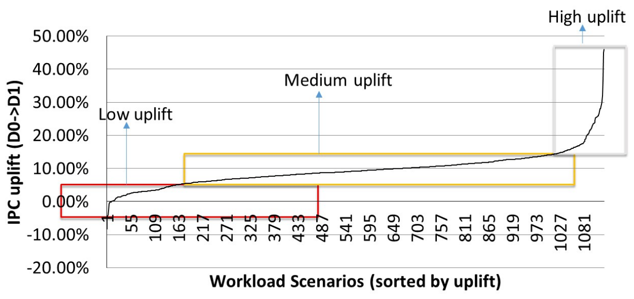

Figure 1: IPC uplift of design D_1 over D_0 for different workloads (Due to confidentiality requirements, we provide limited information about these designs in Section VII)

Such uplift curves are typical for almost any pair of designs. This is not unexpected, since different workloads can be bottlenecked by different parts of the micro-architecture, whereas a new design may only target a subset of features for improvement or re-design. For example, if the target design has significantly lower L2 cache access latencies, a workload whose working set is L1 contained will not see any IPC gains on account of that one feature. On the other hand, a workload that has very high branch counts will likely see a substantial gain from the provisioning of more resources in the branch predictor. Reality is of course more complex (and often counter-intuitive), since features interact with each other and cause the final uplift to be determined by what new bottlenecks manifest themselves once the primary one is alleviated. This is a major reason why cycle-accurate models form an indispensable part of performance analysis methodology for complex processor designs.

IV. The Workload Sampling Bottleneck

The most widely used methodology for architectural performance analysis is based on detailed simulation of a target architecture using a cycle-accurate performance model. The inputs to these models are either dynamic instruction traces (captured from functional simulators or from actual hardware) or static images of system state (including CPU registers, memory and disk) captured via functional simulators. In our work, we mostly use instruction traces, but the methods we propose operate on performance model *output*, and hence do not make any assumptions about the type of input used to stimulate the performance models.

Irrespective of the type of input, the problem of workload sampling is the same. The goal is to collect a sampled set of inputs at different points during the execution of a workload such that the following condition is met. When each point is simulated for a short interval (typically a few million instructions), the average performance observed across all the samples should be a good representation of the expected performance of the original (un-sampled) workload on the target architecture. Since the target architecture is unavailable, these samples are validated by comparing (i) statistics obtained from hardware performance counters while running the workload of interest on an existing hardware platform against (ii) those obtained by simulating the sampled inputs on a well-correlated performance model that models the same hardware platform.

From a projections standpoint, it is critical to prove that the samples can be used to estimate changes in workload IPC under different architectures or architectural configurations. With this in mind, correlation studies like the one above are conducted using different frequencies, or by selectively enabling certain features such as prefetchers and caches, or by studying performance across different hardware platforms. If across these studies, simulated IPC deltas are similar to measured IPC deltas, then we gain a measure of confidence that the samples are a good representation of workload behaviors. However, all this must be achieved while keeping the number of samples as small as possible, in order to minimize computational requirements of performance simulation. As one might appreciate, all of this is a substantial amount of work. Even for well-behaved workloads like SPEC CPU2006 [13], generating a good set of simulator inputs can take months, making such a flow infeasible to execute in real-time for each new projections request.

At the same time, the benefits of such a rigorous approach to workload sampling is not under dispute. Since many micro-architectural features individually offer small gains in performance, any conclusion about uplift of a target workload rests on the assumption that its inputs are well correlated against silicon. So it is essential that workload sampling using the above methodology be an ongoing process to ensure that micro-architecture features are developed keeping in mind the characteristics of modern software. Secondly, as shown in this work, the availability of a large set of well-correlated workload samples collected over several years (even decades) of processor design can prove invaluable [1].

V. Learning Tools

The datasets we consider in this work are either obtained by simulating a large number of inputs using a cycle-accurate performance model, or by collecting hardware performance counter data from silicon. In each case, the columns represent different architectural statistics (*e.g.,* L2 accesses, branch mispredicts), most commonly expressed as an average PKI (Per Kilo Instructions) over a certain interval. Each row represents a small interval of time in the case of hardware data, and a simulator input, like an

[1]Infrequent changes to the instruction set, and backward compatibility requirements ensure that traces once collected, remain simulatable for many years

individual instruction trace, in case of data generated from a performance model.

Principal Component Analysis: The above datasets could potentially have a large number of columns, since the number of statistics that are measurable from hardware and the performance models can run into the hundreds. We restrict our focus to only those statistics for which there is well-documented correspondence between model statistics and hardware performance counters. We find that this covers a lot of important events across all major parts of the micro-architecture. Even so, the number of variables in the dataset can still have an impact on the efficiency and accuracy of learning techniques. Additionally, since PKI statistics from different parts of the architecture often display high degrees of correlation (*e.g.,* L1 cache miss rates and L2 access rates) which can lead to redundancy in the dataset. We use Principal Component Analysis (PCA) to address this issue by transforming the dataset from an original set of N-dimensions to M-dimensions ($M < N$), while ensuring that the resulting variables are uncorrelated. This is known to yield better results from downstream analysis techniques [14].

Clustering: A common problem encountered in data analysis is the assignment of similar observations into clusters, where each observation is represented by an N-dimensional point, and each cluster is represented by its geometric centroid (or the cluster member that is closest to it) [15]. Desirable solutions are ones where intra-cluster distances are minimized, while inter-cluster distances are maximized. In this work, we used public-domain, python-based implementations of well-known techniques such as K-means and hierarchical clustering [16].

Classification: Classification is the problem of identifying which of a set of finite, non-overlapping classes a new data point belongs to, based on a set of measured or observed attributes. In our work, we use a popular classification system called C5.0 [17]. There are a few reasons we chose to use classifiers in general, and C5.0 in particular. First, we were keen on producing a human-readable model that can be analyzed by architects and modeling engineers to develop insights about how workload characteristics influence IPC. Decision trees and rule-based classifiers provide this advantage. Second, for the problems we describe later in the paper, we are often only interested in what category a workload falls into rather than the precise real value of a target metric. For example, answering whether a workload will gain "a lot" (say, more than 50%) or "only a little" (say 5-10%) on a target architecture is sometimes more important than knowing whether the precise gain will be +5% or +7%). This makes the use of classifiers attractive.

Nevertheless, from the perspective of evaluating such methods, we did not stop at merely assessing the ability to classify correctly. We also used the classifier to predict real values (*i.e.,* IPC) by quantizing the target range of IPC uplifts into discrete (but not necessarily uniform) intervals, and then using the mid-point of a predicted class to produce a real number. Since the classes in all of our problems are intervals on the real axis, they are inherently ordered; hence all misclassifications are not equally undesirable. C5.0 provides mechanisms to weigh the penalty of different misclassifications differently, which helps reduce the occurrences of flagrant misclassifications.

VI. Proxy Suites for IPC Estimation

In this section, we describe a flow for generating proxy suites for a new workload, and discuss our experience using it to generate IPC estimates for various commercial workloads on a target design.

A. Methodology

The steps for generating proxy suites are illustrated in Figure 2. The following paragraphs describe each step in detail.

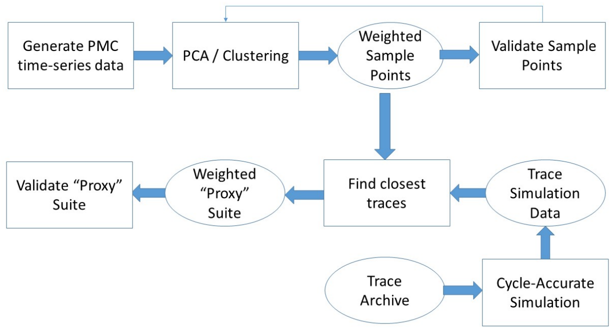

Figure 2: Methodology for generating proxy suites

1) Generate PMC data: We first collect a time-series of various hardware performance monitoring counters (PMCs) while running the workload of interest on a baseline design. This PMC dataset includes functional statistics such as counts of branches, loads and stores, floating point operations as well as micro-architecture dependent statistics such as IPC, cache misses, branch mispredicts, memory bandwidth utilization, *etc*. In our flow, we use about 30 such statistics to cover as much of the design as possible. We configure an internal tool to report these statistics once every few tens of milliseconds to capture fine-grained changes in workload phase behaviors while minimally interfering with workload performance.

2) PCA, Clustering and Validating Samples: The above dataset is subjected to Principal Component Analysis followed by clustering, the output of which is a small number of transformed PMC points (centroids) which adequately represent the characteristics of the entire workload. Each centroid is characterized by a weight that indicates the fraction of the workload that it represents. A variety of solutions are generated during this step by running different algorithms (*e.g.,* K-means, hierarchical) or by setting different parameters within a specific algorithm. For each solution, we perform a reverse transformation from the PCA space back to the original space. Since the original PCA transformation is lossy, in general centroids will not map back to a point that actually exists in the original dataset. Hence we pick a representative sample in the original space that is closest to the re-mapped centroid. After doing this for each centroid, we collect all such points and compute the workload's IPC (considering the centroid weights) and compare it against measured IPC. The solution that produces the lowest IPC error is selected.

3) Cycle-Accurate Simulation: In this step we build a dataset from the results of simulating a large number of instruction traces on a cycle-accurate performance model that models the same baseline design on which the PMC data was collected. While this is a resource intensive step, it needs to be performed only once, and can then be used as a dataset against which numerous workloads can be analyzed with little effort. In fact, these are the same traces that the processor modeling team uses for feature exploration, and hence, simulation results on the baseline architecture may already be available. Each trace simulation yields a vector of statistics that correspond to the PMCs collected above.

4) Find Closest Traces: For each sample point generated via the PCA and clustering step, we simply select the trace who's simulation results have functional and micro-architectural statistics that are closest in terms of N-dimensional distance to the sample point. The result is a suite of traces that has one trace per unique workload phase, with weights that represent the fraction of execution time spent in that phase.

5) Validate Proxy Suite: This is done in several ways. Analyzing the Euclidean distance between the traces and the sample points provides a quick indication of the quality of the solution, and poor solutions are discarded based on a threshold. For solutions that meet the distance criterion, additional validation is recommended. The workload of interest is run on hardware at a few different configurations (*e.g.,* different clock frequencies, or with certain features disabled, or altogether different hardware platforms) to obtain a variety of IPC results. The proxy suite is then simulated on corresponding performance models. The IPC deltas obtained via simulation are compared with those obtained on silicon, and if the difference is small, a valid proxy suite has been found. Note, when validation fails, it by itself is an interesting result. It essentially tells us that the workload has unique behaviors that are not captured by previously traced workloads.

B. Experimental Results

1) Setup: We present case studies of three workloads on which we applied the proxy suite methodology. For each workload, PMC data was collected on a desktop PC based on an AMD A10-5800K APU, equipped with 8GB DDR3, running Windows 8.1. The simulation dataset was constructed from existing runs of an internal cycle-accurate performance model run over more than a 1000 traces representing a wide variety of client and server workloads. The simulation data included results from modeling the baseline design as well as other target designs.

2) Case Study 1: WinRAR [18] is a file archiving and compressing utility for Windows and is commonly used as a processor performance test. When WinRAR 4.2 launched, it came with major changes in the compression algorithms leading to higher performance and compression ratios. Though we had previously characterized and traced WinRAR 3.9, a detailed PMC analysis showed sufficient differences between the versions to justify considering it for tracing. Instead of tracing it, we instead generated a proxy suite using the above methodology. The flow automatically selected a set of traces from several other workloads. The contribution of these traces (by weight) in the new WinRAR 4.2 proxy suite is illustrated in Table I(a).

Constituents of WinRAR proxy suite	Contribution (weight %)
SysMark07	52%
SPECCPU2006	32%
7zip, WinZip, PCMark8	15%

(a)

WinRAR version	Method	% IPC uplift (D0 to D1)
WinRAR 4.2	Hardware Measurement	+1.91%
WinRAR proxy suite	Cycle-accurate models	+2.05%
WinRAR 3.9 suite	Cycle-accurate models	-11.70%

(b)

Table I: Proxy suite results for WinRAR 4.2 (a) constituent traces (b) IPC uplift validation

Interestingly, the methodology did not pick *any* traces from WinRAR 3.9 even though data from those trace simulations were part of the dataset. Instead, the method choose a combination of SPEC CPU2006 [13] and legacy SysMark 2007 [19] traces. Table I(b) shows the result of our validation study, which compares the IPC uplift across two processor designs as obtained from the actual hardware versus the two sets of traces. The results show that WinRAR uplifts generated using the proxy suite (+2.05%) track hardware uplifts very closely (+1.91%) whereas using older WinRAR 3.9 traces leads to completely wrong uplift estimates. Using our methodology, we were able to generate a much higher confidence projection for WinRAR than if we simply chose to use average IPC uplift, or worse still, if we used the traces from the older version.

3) Case Study 2: A revision in a transcoding application prompted us to run the proxy methodology over iTunes v10.6 [20]. The results obtained were different from our experience with WinRAR. The automatically generated proxy suite consisted of a large number of traces from the previous version of iTunes (v9.0). Though qualitative statements in the release notes of the software suggested otherwise, the proxy flow demonstrated that from a micro-architecture and IPC perspective, not much had changed. 97% of the weight of the proxy suite of iTunes v10.6 was contributed by iTunes v9.0 traces, and the remaining 3% came from Sysmark07 traces.

4) Case Study 3: TrueCrypt [21] is a now discontinued open-source encryption package that remains popular as a performance benchmark. Applying the proxy methodology flow to TrueCrypt v7.1 yielded interesting results. The flow did not pick existing traces from TrueCrypt v7.0, but instead picked traces from a few other workloads including PCMark [22]. The validation exercise yielded poor results. Upon further analysis we root caused this to the prevalence of AES instructions in the new version of TrueCrypt, a workload property that was not well represented in our trace suites at the time (PCMark being one of the few exceptions, which explained its inclusion). As a result of this flow, we increased the priority of tracing TrueCrypt.

In summary, over a period of 2 years during which this methodology was deployed, we estimate that it saved us several man months of tracing work, and provided an effective avenue for quickly generating IPC uplift estimates for new versions of old workloads, as well as several new workloads. In current practice, applying proxy methods is a mandatory step before we embark on a tracing project. It has also proved itself to be an essential tool to dictate tracing priorities for incoming workloads.

VII. Classifier-Based IPC Estimation

The encouraging results obtained via the proxy workloads methodology suggest that there is a wealth of information relating performance uplift to workload characteristics in existing simulation datasets. A natural question to ask is, whether one can automatically learn this dependence using supervised learning techniques. To this end, we developed a method that uses decision-tree based classifiers to estimate IPC uplift. This methodology broadly consists of 2 phases. In Phase 1, a decision-tree is constructed that expresses IPC uplift in terms of performance statistics that can easily be gathered from hardware performance monitoring counters. In Phase 2, the model is deployed for the particular workload of interest.

Note that, the decision to build a classifier that predicts IPC uplift instead of absolute IPC was a conscious one. The intuition was that actual IPC is highly dependent on both the micro-architecture and workload characteristics, making it a harder learning problem to solve. For a fixed pair of designs, the IPC uplift obtained is entirely a function of workload characteristics (Figure 1) and hence, potentially yields simpler and more accurate models.

A. Phase 1: Model Development

1) Dataset Construction: The dataset used for this step is constructed from simulations of thousands of pre-existing traces across pairs of design configurations. These consist of a base configuration (D_0, one for which silicon is already available), and a set of target configurations that we are interested in projecting performance for (D_1, D_2, $\ldots D_n$). The number of target configurations are usually a small handful, spanning a few different product definitions based on the same design, or a set of designs that represent a design team's road-map.

We construct the following dataset from cycle accurate simulations of D_0 and D_n. Each row in the dataset corresponds to an independent simulation input, and consists of selected functional and micro-architectural statistics that are known to be measurable from hardware performance counters in D_0. From the simulations, for each row, we compute the IPC uplift of D_n over D_0. To facilitate the use of classifiers, we replace the actual value of uplift with a class label that represents a range of IPC uplift values which contains it.

2) Model Construction: For each dataset so constructed (one for each target design or configuration), we generate a classifier-based model using a randomly chosen subset of rows for training the model. In our work, we used C5.0 [17] to generate decision-trees that predicts which IPC uplift class a given workload belongs to. A portion of an example three-class model is illustrated in Figure 3. The snippet shows decisions made near the leaf level of the tree, where the model is concluding that if a workload has a high L1 DC miss rate (on a base architecture), and a high frequency of x87 instructions it is likely to see low IPC uplift ($<5\%$) on the target design. This shows that the model has recognized that between these two designs x87 performance has not improved much, and that workloads that are dominated by such low-performance code are unlikely to see much gain. For other combinations of statistics in the snippet shown, the model classifies the workload as a medium-uplift workload.

If simulation datasets exist, building the model takes minutes. If simulation data is not available for specific design configurations that are of interest from a projections standpoint (say particular memory configurations), running the simulations on existing inputs can take several days or weeks depending on the number of inputs. For high quality projections, it is important to periodically rebuild the model with the latest simulation data, since over time,

```
L1DReq > 359.59:
:...LSEvent <= 69.03:
    :...x87 > 261.45: IPCUplift <= 5
    :   x87 <= 261.45:
    :   :...uops <= 965.29: IPCUplift <= 5
    :       uops > 965.29:
    :       :...SSE <= 95.52: 5 < IPCUplift <= 15
    :           SSE > 95.52:
    :           :...DRAMBandwidth <= 84.86: 5 < IPCUplift <= 15
    :               DRAMBandwidth > 84.86:
    :               :...L2MissICFill > 0.09: 5 < IPCUplift <= 15
    :                   L2MissICFill <= 0.09:
```

Figure 3: Snipped output of a 3-class decision-tree for identifying IPC uplift classes

	IPC error (%)	Quant. error (%)	Misclass. error (%)	Quant. contr (%)	Misclass. contr (%)	Misclass. Rate (%)
D0-D1	1.94	0.32	2.62	4.97	94.60	69.92
D0-D2	4.50	0.88	5.99	5.67	94.34	70.95
D0-D3	7.78	1.01	10.35	3.57	96.46	72.49
D0-D4	7.75	1.35	9.32	3.44	96.53	80.27

Table II: IPC estimation errors versus cycle-accurate simulation across 500+ workloads using decision trees

IPC characteristics of the target design change due to feature changes, and bug fixes.

B. Phase 2: Model Deployment

In this phase, we run the workload of interest on the base architecture (for which silicon is available) and collect a vector of Performance Monitoring Counter (PMC) values that correspond to the parameters that were used to build the decision-tree in Phase 1. This vector is provided to the classifier model as an input, which in turn generates a IPC uplift class. To estimate IPC of the workload on the target design, we choose the center of the predicted class as the estimated IPC uplift. This along with the measured IPC on the base design gives the final IPC estimate for the target design.

Note that, the run time of the actual classification is negligible compared to the effort in collecting good PMC data for the workload of interest. Depending on the complexity of setup and running the workload, this may take anywhere from a few hours to a few days. Note, this only needs to be done once per workload as long as the base architecture from which uplifts are being predicted remains the same. Once the PMC data is available, generating IPC estimates takes seconds.

A more fine-grained approach could be used, where instead of collecting one vector that captures average PMC behavior over the entire workload, one could collect PMC samples and cluster them using techniques described in Section VI. Each sample would then be input to the classifier model to generate IPC estimates, which then via appropriate weighting and averaging mechanisms could be used to generate a final IPC estimate.

C. Experimental Results

1) Setup: We generated and evaluated models to predict IPC for 4 high-performance x86 processor designs D_1 through D_4, based on PMC data collected on a base design D_0. D_1 and D_2 are designs that based on the same microarchitecture as D_0, all of which are commercial x86 designs aimed at mainstream client desktops and laptops. D_3 is a completely different micro-architecture aimed at low-power mobile devices, and D_4 is a futuristic design in early stages of definition.

We leveraged existing cycle-accurate simulation data targeting these designs across 1100 instruction traces to construct the dataset. We used half the dataset for training the model and rest for evaluation. The classes were determined by clustering the entire set of IPC uplifts into 20 clusters. For the evaluation set, we compare the predicted IPCs with those obtained from cycle-accurate simulations of the same traces. Note, this allowed more comprehensive validation of the modeling methodology than using silicon measurements, since we have cycle-level models for various target architectures for which silicon does not yet exist.

2) Model Error Evaluation: Table II shows the average error in estimating IPC for over 500 traces for each target design (compared against cycle accurate simulation). Additionally, it shows the impact that two sources of error have on the quality of the results. Quantization error occurs because even if the classification is correct, the uplift is assumed to be the mid-point of the class. Misclassification error is simply due to incorrect classifications. There are several interesting things to note from these results.

- IPC errors are *very low*. In fact, they are *highly comparable* to error rates typically associated with rigorous tracing and cycle-accurate modeling methodology. The average error in predicting workload IPC on D_1 was less than 2%. This is a key finding, since the need for high accuracy is highest when projecting performance for the immediate successor to an existing processor design.
- As the target design gets increasingly dissimilar from the base design the errors start to increase (although they well within bounds that determine usefulness of the approach). This is not surprising. The problem of learning an uplift function across highly dissimilar architectures is inherently harder because we are limited by the set of performance counters that are available on

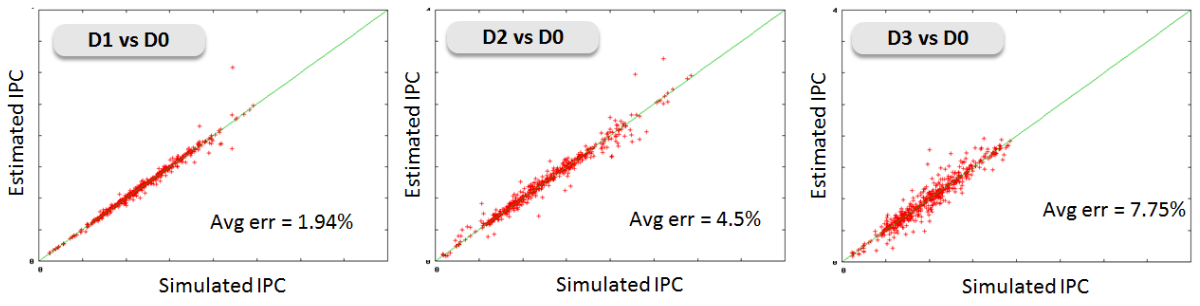

Figure 4: IPC prediction accuracy for 3 target architectures across over 500 workloads

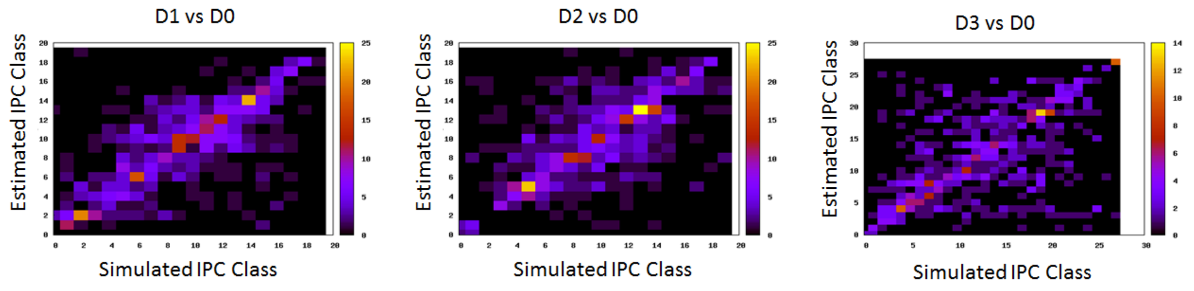

Figure 5: Confusion matrix for 3 target architectures across over 500 workloads

the base architecture.

- We see that about 95% of the error is due to misclassifications and that misclassification rate (the probability the classifier picked the wrong class) is about 60-70%. The fact that we get very good IPC accuracy in spite of this is analyzed further below.

Figure 4 presents scatter plots corresponding to the average data presented in Table II (for lack of space we present results for only 3 designs). While the axis labels have been suppressed (for reasons of product confidentiality) the trend is illustrative of the point mentioned above. The IPC miscorrelation increases slightly going from D_1 through D_3, both in terms of number of outliers as well as spread around the ideal line The scatter plots also show that the classifier is able to predict IPCs accurately across a wide range of workloads spanning both high and low IPC behaviors.

To understand misclassifications better, for each model and associated results, we generated a *confusion matrix*. These are illustrated in Figure 5. Each cell in the matrix is a colour-coded indication of the number of test cases in the evaluation dataset that fell into that cell. By analyzing these matrices, we can tell that first, indeed, misclassifications are primarily responsible for the errors seen in the scatter plot. More crucially, for D_1 and D_2, it shows that misclassifications, when they occur, are more likely to occur between neighboring or nearby classes. This is a non-obvious result, since classifiers are not necessarily discerning about misclassifications during training: all misclassifications are

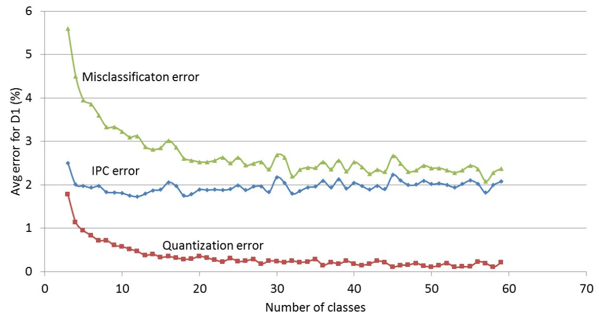

Figure 6: Effect of number of classes on error rates

considered equally undesirable. In fact, although C5.0 allows a user to weigh misclassifications differently, enabling it did not substantially change our results. The errors for D_3 are worse. This was expected, because D_0 and D_3 are completely different micro-architectures.

3) Impact of Class Granularity: A key aspect is in this approach is choosing the number and sizes of classes. In our flow we analyzed the input data and ran single-dimensional clustering on the IPC uplift values to generate uplift classes. We preferred this approach to using uniformly spaced classes, since the latter could potentially result in many classes being under-represented in the training set.

For estimating the IPC of D_1, we varied the number

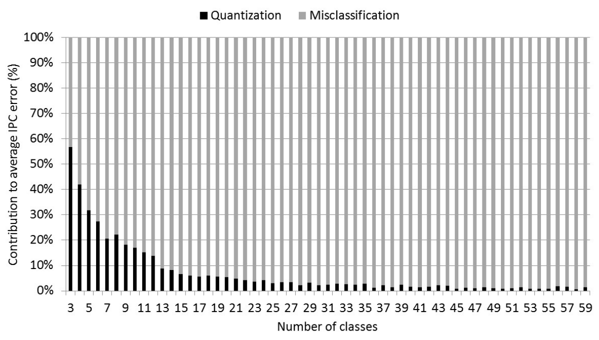

Figure 7: Contribution of quantization and misclassification to overall error

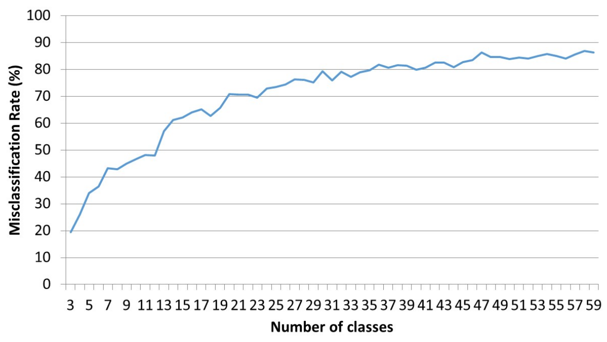

Figure 8: Effect of number of classes on the misclassifications

of classes from 3 to 60 and evaluated decision-tree based classifiers for each. Figure 6 shows the variation in different errors with increasing number of classes. Quantization and misclassification errors are the average errors for correctly and incorrectly classified examples respectively. We see that when the number of classes is very small (say, 3) the error rates are high for both types of error, since with large class sizes, even neighboring class misclassifications result in large errors, along with large quantization errors. Figure 7 shows how increasing the number of classes increases the contribution from misclassifications, but reduces the contribution from quantization. Figure 8 captures misclassification rates, showing that due to the inherent ordering of our classes, we can afford misclassification rates to be as high as 60-70% while providing high quality IPC estimates. Based on these results, we determined that 20 classes is an appropriate number for our purposes. At this granularity, the contribution from quantization is quite low, so one can focus attention on reducing the error due to misclassification. At the same time, we keep the model from becoming excessively complex.

4) Impact of Training Set: We evaluated how sensitive the classifier is to the precise set of training examples used. To do this, for each target training dataset size, we created 100 datasets containing points picked at random. In each case, the remaining points were used to assess misclassification rates. For this study, we used a 3-class decision tree. Figure 9(a) shows that misclassifications decrease as the size of the training set increases. To avoid over-fitting the model, we used a 50% size in our work. Figure 9(b) shows the impact of random selection on misclassification rate at a certain size. The results show that while there is variability in the quality of the model depending on the subset is chosen, there is a high chance that a small number of experiments will yield a good quality model. Out of 100 models generated using 50% of the original data, only 3 had misclassification rates of over 20%.

VIII. Proxy Suites vs Decision Trees

A major advantage of the proxy suites approach over decision-trees is that it actually generates the full set of micro-architectural statistics that are computed by the performance model, which can then be used to identify performance bottlenecks, and drive what-if analysis. While a reading of the path traversed by the decision-tree can provide some intuition about why IPC falls in the range that it picks, it is not nearly as informative.

The other advantage the proxy methodology provides is that negative results can be used to drive tracing priorities (*i.e.,* higher priority should be assigned to important workloads for which good proxies cannot be determined). Over time, this leads to an archive of traces with lower redundancy, resulting in lower demands on storage and computational requirements.

The significant advantage of the classifier-based method is, it can potentially generate a high quality performance estimate even if the overall workload is substantially different from previously collected traces. This is because the model learns how IPC depends on individual workload characteristics, rather than blindly proxying a particular phase behavior with an existing trace. It is also faster to deploy than the proxy method.

When using these methods, modeling teams need to exercise caution, as they need to when interpreting results from any model, even so-called cycle-accurate ones. The proxy suite methodology described earlier provides enough evidence as to whether the generated suite is good enough to

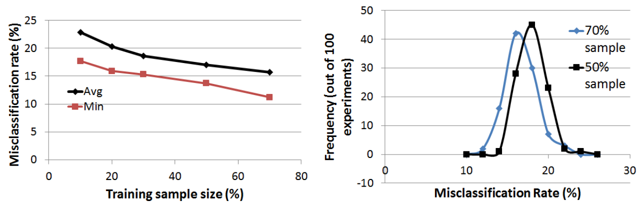

Figure 9: Impact of (a) size and (b) choice of training set

use in a mainstream projections flow, so due attention needs to be paid to the validation results. If the validation is weak, then the decision-tree based flow is a good fallback. A useful feature of the latter flow is the association of a confidence level with each prediction. Based on this information, and additional intuition about the changes in the architecture and workload characteristics, a user can decide how much credibility to assign to the result.

Overall, we were pleasantly surprised at the level of accuracy that was achieved using these methods. Both methods have been automated and are part of a slew of tools that are used internally for workload analysis and performance modeling. Using these flows, generating and validating a proxy suite is about 2-3 days worth of work (not including workload set up time), while generating IPC estimates from the classifier-based model is less than 1 day's work.

IX. Conclusion

In this paper we presented two techniques that can be used to generate performance estimates of new workloads for a target design, by applying data analysis methods to the wealth of simulation results that design teams generate as part of microarchitecture development. Returning to the two questions posed in Section I, it is clear from our experience with proxy suites that "new" workloads often demonstrate sensitivities to micro-architectural changes that are similiar to previously analyzed workloads (or phases thereof). In such cases one can successfully "simulate" a new workload using old traces. Our results, along with those obtained via the more general decision-tree based approach, provides strong evidence that IPC of a new workload can be reliably estimated without resorting to fresh tracing and cycle-accurate simulation, *provided one has access to large enough datasets*. Our IPC estimates were on average within 2% of those obtained via measurement or cycle-accurate simulation. Both techniques were tested and deployed in an industrial setting, and have resulted in savings of many man-months of engineering effort.

Over time, we expect datasets will only grow in size and richness due to increasing number and variety of hardware performance counters, workloads, and processing elements on a single chip. Projecting full-chip performance and power for heterogeneous designs by taking proper advantage of relevant datasets is a challenging area of research with potentially high impact.

Acknowledgements: The authors would like to acknowledge Francesco Spadini and Ravi Bhargava for numerous discussions, and the reviewers for their feedback.

References

[1] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and Exploiting Program Phases," *IEEE Micro*, vol. 23, pp. 84–93, Nov. 2003.

[2] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," in *Proc. Int. Symposium on Computer Architecture (ISCA)*, pp. 84–97, 2003.

[3] L. Eeckhout, J. Sampson, and B. Calder, "Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation," in *Proc. IEEE Int. Symposium on Workload Characterization (IISWC)*, 2005.

[4] K. Hoste, A. Phansalkar, Eekhout.L, A. Georges, L. John, and K. Bosschere, "Performance prediction based on inherent program similarity," in *Int. Conf. on Parallel Architectures and Compilation Techniques*, 2006.

[5] S. Khan, Xekalakis.P, Cavazos.J, and Cintra.M, "Using predictive modeling for cross-program design space exploration in multicore systems," in *Int. Conf. on Parallel Architectures and Compilation Techniques*, 2006.

[6] Dubach.C, T. Jones, and M. Boyle, "Microarchitectural design space exploration using an architecture-centric approach," in *Proc. IEEE/ACM Int. Symposium on Microarchitecture (MICRO)*, 2007.

[7] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A Predictive Performance Model for Superscalar Processors," in *Proc. IEEE/ACM Int. Symposium on Microarchitecture (MICRO)*, pp. 161–170, 2006.

[8] T. S. Karkhanis and J. E. Smith, "A First-Order Superscalar Processor Model," in *Proc. Int. Symposium on Computer Architecture (ISCA)*, pp. 338–, 2004.

[9] T. Chen, Q. Guo, Tang.K, O. Temam, Z. Xu, Z.-H. Zhou, and Y. Chen, "ArchRanker: A Ranking Approach to Design Space Exploration," in *Proc. Int. Symposium on Computer Architecture (ISCA)*, 2014.

[10] B. Piccart, A. Georges, H. Blockeel, and L. Eeckhout, "Ranking Commercial Machines through Data Transposition," in *Proc. IEEE Int. Symposium on Workload Characterization (IISWC)*, 2011.

[11] R. Cammarota, A. Kejariwal, P. Alberto, S. Panigrahi, A. Veidenbaum, and Nicolau.A, "Pruning hardware evaluation space via correlation-driven application similarity analysis," in *Computing Frontiers*, 2011.

[12] D. Lee, T. Kim, K. Han, Y. Hoskote, L. K. John, and A. Gerstlauer, "Learning-Based Power Modeling of System-Level Black-Box IPs," in *Proc. Int. Conf. Computer-Aided Design*, pp. 847–853, 2015.

[13] "SPEC CPU2006." https://www.spec.org/cpu2006/.

[14] "Principal Component Analysis." https://en.wikipedia.org/wiki/Principal_component_analysis.

[15] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, 1990.

[16] "Scientific Computing Tools for Python." http://www.scipy.org.

[17] "Data Mining Tools See5 and C5.0." https://www.rulequest.com/see5-info.html.

[18] "WinRAR." http://www.rarlab.com/.

[19] "BAPCo." https://bapco.com/.

[20] "iTunes: How to convert a song to a different file format." https://support.apple.com/en-in/HT204310.

[21] "TrueCrypt." http://truecrypt.sourceforge.net/.

[22] "PCMark." http://www.futuremark.com/benchmarks/pcmark.