Deciphering Predictive Schedulers for Heterogeneous-ISA Multicore Architectures

Andreas Prodromou
aprodromou@eng.ucsd.edu
UC San Diego

Ashish Venkat
venkat@virginia.edu
University of Virginia

Dean M. Tullsen
tullsen@eng.ucsd.edu
UC San Diego

Abstract

Heterogeneous architectures have become increasingly common. From co-packaging small and large cores, to GPUs alongside CPUs, to general-purpose heterogeneous-ISA architectures with cores implementing different ISAs. As diversity of execution cores grows, predictive models become of paramount importance for scheduling and resource allocation. In this paper, we investigate the capabilities of performance predictors in a heterogeneous-ISA setting, as well as the predictors' effects on scheduler quality. We follow an unbiased feature selection methodology to identify the optimal set of features for this task, instead of pre-selecting features before training. We propose metrics that bridge the gap between traditional prediction accuracy metrics and a scheduler's performance. We further present our evaluation methodology, which was meticulously designed with this study in mind, and finally, we incorporate our findings in ML-based schedulers and evaluate their sensitivity to the underlying system's level of heterogeneity.

ACM Reference format:
Andreas Prodromou, Ashish Venkat, and Dean M. Tullsen. 2019. Deciphering Predictive Schedulers for Heterogeneous-ISA Multicore Architectures. In *Proceedings of The 10th International Workshop on Programming Models and Applications for Multicores and Manycores, Washington, DC, USA, February 17, 2019 (PMAM'19),* 10 pages.
https://doi.org/10.1145/3303084.3309492

1 Introduction

Heterogeneous multicores employ cores that can vary in microarchitecture and even instruction set architectures, on the same processor chip. These architectures allow a running application to dynamically migrate execution to the most suitable core, reacting to phase changes and changes to the dynamic execution environment, and further maximizing performance and energy efficiency. While single-ISA heterogeneous multicores [1–4, 19, 21, 25, 26] offer cores that vary in their microarchitectural configurations to cater to the diverse execution characteristics (e.g., instruction-level parallelism, cache access patterns, branch behavior, etc.) of mixed workloads, heterogeneous-ISA architectures [5, 6, 9, 11, 18, 27, 34, 40, 41, 43], capture an additional dimension of heterogeneity – ISA affinity – the inherent preference of an application code region to execute on a particular ISA. Multiple studies have shown that exploiting ISA affinity coult result in significant (>30%) additional gains in performance and energy efficiency.

However, as the amount of on-chip heterogeneity increases, intelligent scheduling becomes more crucial, since the potential performance and power benefits arrive from smart job allocation. In fact, subpar job allocation can waste resources and power. Proactive scheduling mechanisms proposed in the literature rely on predictive models to make active performance predictions for any code region in execution, that can further assist in making appropriate job allocation decisions [17, 38, 45]. The quality of these mechanisms is often considered tied to their prediction accuracy.

State-of-the-art schedulers employ either analytical [14, 17, 32] or statistical [8, 13, 45] (for example, machine learning) predictive models that strive to make accurate performance predictions, and thereby enable near-optimal scheduling decisions. In both cases, predictions are derived using a mathematical equation (or model) whose inputs are typically associated with weights that signify their importance. Analytical models typically employ linear equations, similar to those used by linear regression models. However, these models rely heavily on domain expertise to identify correlations between input features (e.g., instruction mix) and the prediction target (e.g., performance), and consequently determine the appropriate set of feature weights. These models are typically confined to use only a handful of features due to the exponentially increasing complexity of identifying such correlations. Statistical models on the other hand, are trained in software using sophisticated machine learning algorithms that can not only handle far more input features, but can further adapt to non-linear correlations.

Recent technological advances have increased the availability and popularity of machine learning algorithms and tools, leading to an increased number of ML-based computer architecture proposals [8, 13, 44, 45] in recent years. Researchers have demonstrated that predictive models can be deployed and used in modern computation environments, from heterogeneous CMPs to datacenters [14, 32]. However, scheduling for *general-purpose* heterogeneous-ISA CMPs has not been extensively studied, despite significant opportunity for greater performance and energy efficiency. This is in part because, as core diversity increases, the problem of cross-core performance prediction and subsequently the problem of optimal job allocation grows more complex. In a homogeneous multicore architecture, a running thread is expected to have similar performance when it executes on a different core. However, in a single-ISA heterogeneous multicore, the performance difference of the same code region can be vastly different on different cores depending upon the microarchitectural diversity. ISA heterogeneity adds an extra degree of freedom, resulting in a more complex set of parameters that could now potentially affect performance.

In this work, we find through quantitative evaluation that schedulers present, in some cases, significant tolerance towards prediction error. In particular, we identify the processor's level of diversity to be the most influential factor that dictates the level of prediction error tolerance. We seek to quantify and better understand this phenomenon by developing three key metrics: Alpha-IPC (αIPC),

PMAM'19 , February 17, 2019, Washington, DC, USA

ACM ISBN 978-1-4503-6290-0/19/02…$15.00
https://doi.org/10.1145/3303084.3309492

Ease-of-Scheduling (EoS), and Expected Scheduler Efficiency (ESE). EoS and ESE are statistical metrics that allow us to deal with our practically infinite set of heterogeneous(-ISA) multicore systems our database can generate, while αIPC allows us to fairly compare runtime progress of workloads compiled for different ISAs.

We utilize the simulation-based dataset developed by Venkat and Tullsen. [43]. This dataset containts simulation results of 72 workloads, each one simulated on 600 different heterogeneous-ISA cores. To the best of our knowledge, this is the most extensive heterogeneous-ISA, cycle-accurate, simulation-based database in the literature. In this paper, we apply a variety of machine learning techniques to the problem of cross-core performance prediction, where the target core potentially differs in terms of its CPU microarchitectural traits, cache organizations, ISA, inorder/out-of-order execution semantics, vector support, and floating point support. The inputs to our predictors consist of application-specific features (obtained via profiling on a "reference" core), and target core-specific features. The output of the model is a performance prediction for the application and target core pair.

Our goal is to not just make effective predictions, but to gain particular insights about the salient features of predictive ML models, the feature correlations (program characteristics, microarchitectural measurements) that are picked up and amplified by the best ML models, and those that are less important. We do this by deciphering (reverse-engineering) our most accurate predictive models. This approach allows us to probe a fundamental question – to what extent can machine learning models understand, analyze, and project the execution of code on arbitrary processor hardware?

Due to the vast number of possible CMP combinations in our database, we present a meticulously developed methodology utilizing statistical analyses and a variety of evaluation frameworks that allow us to derive insights and statistical expectations regarding prediction accuracy and scheduler efficiency.

The contributions of this work are:

- Establishment of three novel metrics for (1) tracking cross-ISA program throughput, (2) quantifying a system's heterogeneity level, and (3) translating prediction accuracy metrics to expected system's performance.
- Development of accurate ML-based cross-core performance predictors, which are to the best of our knowledge, the first predictors to cross general-purpose ISA boundaries.
- In-depth characterization and evaluation of ML-based performance predictors, and presentation of insights extracted from deciphering trained ML models that make generally accurate performance predictions.
- Demonstration of an effective ML-based heterogeneous-ISA job scheduler.

2 Background and Related Work

This work lies in the intersection of machine learning and computer architecture: We seek to quantify the influence of predictive models and their accuracy, on the final product; a scheduler for heterogeneous-ISA architecture.

We explore three ML algorithms: (a) ridge regression [22] (RR) which is a variant of linear regression [33], (b) decision trees [31] (DT) that are capable of exploiting non-linear relationships by creating a binary decision tree with one condition on a single feature in each node, (c) random forests [12] (RF) that create a collection of decision trees during training and when queried for a prediction, return the average response from all trees. To identify the correlations picked up by our best models, we use two commonly used exploration techniques: linear coefficient analysis, and feature impurity analysis [30]. We further develop a new analysis for categorical microarchitectural features, named "Reverse Path Purity". We describe these techniques in more detail later in the paper.

Single-ISA heterogeneous architectures have been proposed by Kumar et al. [25]. Besides a large body of research proposals, we already see processor manufacturers offering heterogeneous products [15, 19, 24, 39]. There is a large body of research on scheduling and resource management on single-ISA heterogeneous CMPs (such as [17, 37, 38] and more). General-purpose Heterogeneous-ISA multicore architectures [18, 43] include microarchitecturally heterogeneous cores that implement different ISAs. Barabalace, et al. [9, 10] propose Popcorn Linux, an operating system that allows running shared memory applications on a general-purpose heterogeneous-ISA system and further assists in execution migration between the cores. Our work assumes similar software support. Furthermore, exploiting heterogeneous-ISA architectures has been shown to improve security against code-based attacks, such as Return-Oriented Programming, by frequently switching ISAs [42].

Systems combining CPUs and GPUs (or accelerators) in the same package are also examples of heterogeneous-ISA architectures. Our work focuses on systems with general-purpose CPU cores implementing ARM's Thumb ISA[28], Alpha [16] and Intel's x86-64 [23]. Accelerator-based systems (including GPUs) are out of the scope of this work, since the interaction between software and hardware varies greatly between CPUs and GPUs. However, various prediction-based mechanisms have been proposed for such systems [29], often utilizing ML algorithms: Ardalani et al, propose XAPP [7], an ensemble prediction mechanism, which given CPU source code can predict whether this code would benefit from porting to a particular GPU architecture. Hayashi et al. propose a mechanism based on Support Vector Machines (SVM), which predicts if a given code block should run on a CPU or a GPU for better performance [20].

LACross [45] is a cross-platform scheduler that uses the LASSO linear regression algorithm to predict a phase's performance and power consumption. LACross is evaluated on two heterogeneous machines and enables efficient resource management. Chen et al. [13] utilize Principal Component Analysis for dimensionality reduction and clustering of similar applications for scheduling purposes.

Scheduling on general-purpose heterogeneous-ISA CMP architectures has not been investigated to the same extent as single-ISA or CPU-GPU heterogeneous architectures. Popcorn Linux implements a scheduler that relies on application instrumentation and generates a mapping of application's functions to cores [10].

3 Motivation

Schedulers clearly benefit from accurate performance predictions. Depending on the system under test, trial and error scheduling on a complex system (many threads, diverse cores) is (1) unlikely to find the optimal schedule and (2) likely to sample many poor schedules before finding good ones. A scheduler that can predict the performance of any given application on any core can intelligently distribute jobs for maximal throughput without the overhead of sampling all possible permutations.

Intuitively, we expect that a system's level of heterogeneity can affect the difficulty of the scheduling problem. In other words, it should be easier to create good schedulers for systems with lower diversity. In case of predictive schedulers (ML-based or otherwise), prediction error should have a smaller impact on realizing efficient schedules on less diverse systems. Although intuitive, to the best of our knowledge, the impact of heterogeneity with respect to the scheduling problem has not been quantified in the literature. Our **Ease-of-Scheduling** (EoS) metric has been developed in response to this ambiguity.

Regardless of the underlying system, all prediction units can be characterized by their prediction accuracy, using the mean and standard deviation of their prediction error. However, depending on the amount of underlying heterogeneity, these metrics do not necessarily provide an accurate image as to the expected performance of the derived scheduler that uses this unit. Our **Expected Scheduler Efficiency** (ESE) metric statistically quantifies how accurate a performance predictor needs to be in order for the scheduler to lead to high performance, given a system with some arbitrary scheduling difficulty (a system with a given EoS).

Finally, we use *αIPC* to maintain fair performance comparison between workloads compiled for different ISAs. Since the number of dynamic instructions varies between ISAs, we measure execution progress in *Alpha* Instructions Per Cycle. Our compilation framework allows us to map regions of executables to their corresponding regions in an Alpha executable, thereby enabling this metric.

3.1 Limitations in Prior Work

We identify some common limitations in prediction-based scheduling proposals we aim to address with our methodology: Systems under test are often predefined and do not change throughout each proposal, without discussing if and how the proposed scheduler adapts to other systems. A closely related, frequently-observed drawback is the use of small datasets on which these predictors are trained and evaluated. This is due to the difficulty in obtaining large datasets, particularly if the data is accumulated via simulations (slow). In studies where profiling is used, the hardware cannot be significantly varied, so a large dataset would require tens of thousands of benchmarks.

Several prior studies select one machine learning algorithm, often a linear model, which according to our findings is not necessarily the best option for most cases. Discussion of other available algorithms and how well they perform in the presented use cases is often omitted. Aside from the limited exploration of available algorithms, researchers often pre-select the input features of their models. This selection tends to reflect the collective knowledge of this field and includes variables that have been identified in the past to track dynamic characteristics. However, with modern execution environments we are able to collect significantly more measurements, especially if simulations are used. Furthermore, even though feature selection often requires significant skill and time investment, some ML algorithms are particularly good at filtering features and internally discarding those that have no impact on the final prediction. Finally, ML-based studies that show high prediction accuracy do not explore the machine's "reasoning" when making decisions. We seek to understand what these models consider important when predicting an application's performance on a core.

In this work, we address drawbacks commonly found in prior work: To address the issue of small datasets we simulate 72 workloads on 600 different cores each, for a total database size of 43200 entries. The cores we include span 3 different ISAs: Thumb, Alpha, and Intel's x86. The large heterogeneous-ISA core collection is shown to result in predictive models that generalize and adapt to a variety of underlying systems without modification. We explore thousands of configurations of each ML algorithm (different optimizations, input sets etc) and identify the optimal models. For the majority of this work, we do not attempt to pre-select the desired features, instead we allow the ML models to use them all as they see fit. Once our models are trained and report acceptable prediction accuracy, we seek to understand how these accurate predictions are achieved. We use an in-house scheduler evaluation framework to describe the relation between ML metrics (such as mean error, standard deviation etc) and the efficiency of ML-based schedulers. We demonstrate that when we use ML models as the predictive unit of a scheduler, isolated ML metrics do not necessarily reflect the scheduler's quality.

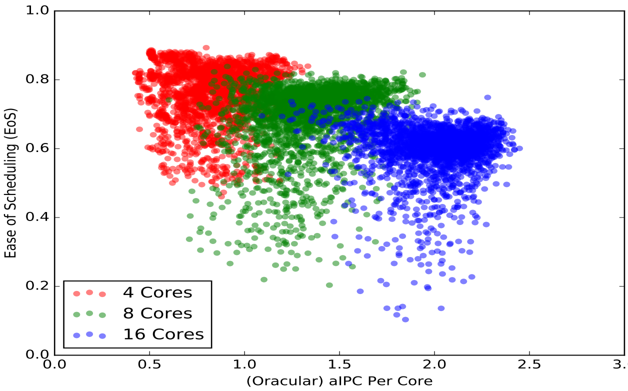


Figure 1. Ease of Scheduling analysis

3.2 Ease of Scheduling (EoS)

We measure EoS by evaluating a system that uses a *random* scheduler. Workloads are run to completion and when execution completes, we compare the obtained mean *αIPC* against the mean *αIPC* the same system reports using a "monte-carlo optimal scheduler". The closer the random scheduler scores compared to optimal, the easier it is to schedule for the system under test. Higher scores mean easier systems.

To minimize noise caused by random decisions, we execute each experiment 300 times, with 200 workloads scheduled per execution and report mean performance. The workloads are randomly chosen *before the experiment begins and remain constant throughout the entire experiment,* for all 300 runs on each of the 7500 systems under test. We ignore cross-core migration overhead since our predictive models don't yet consider cross-ISA binary translation and state transformation costs, which can significantly vary with different application characteristics and different target ISAs. More details on our experimental methodology are presented later in Section 4.

With 200 workloads, the problem of finding a globally optimal schedule is practically infeasible. The number of possible schedules for a 4-core system is 270 digits long. We define optimal performance for each system, as the maximum *observed* performance from all our EoS and ESE experiments. Combined, we compare almost 14 thousand scheduling options on all systems, while scheduling the same 200 workloads appearing in the same order. Throughout the experiment, we use a random scheduler, as well as schedulers based on predictors of varied statistical accuracy, in terms of their Mean Absolute Error and Standard Deviation.

Figure 1 presents our EoS analysis on 7500 randomly chosen heterogeneous-ISA multicore systems. More accurately, we present

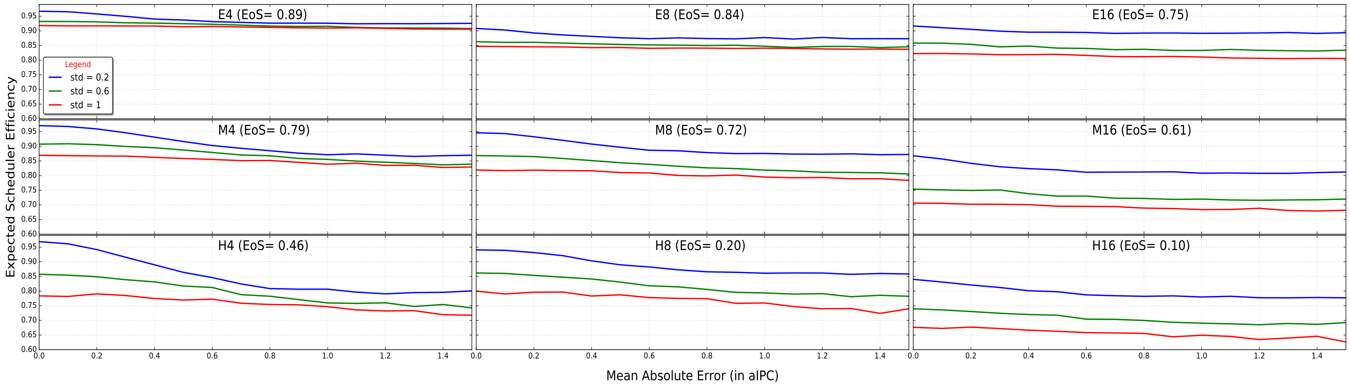


Figure 2. Scheduler efficiency on Easy (E), Medium (M) and Hard (H) 4-, 8-, and 16-core systems.

results on 2500 randomly chosen 4-, 8-, and 16-core systems. Our results highlight the impact of various factors on a system's EoS. First, we observe that the number of cores appears to have an impact, regardless of the system's level of heterogeneity, with each cluster hitting a lower EoS ceiling than the previous one. We also observe significant intra-cluster diversity for all clusters. Our results show that the most difficult systems to schedule resemble a Cell-like heterogeneous-ISA architecture. For example, the most difficult (lowest EoS) 8-core system, contains one high-end x86 core, two medium cores (1 Alpha, 1 x86) and five low-performance, microarchitecturally heterogeneous Thumb cores. Such systems appear to be very unforgiving towards prediction error since there are statistically, limited options that provide optimal or near-optimal performance, each time the scheduler is invoked to make a decision. The easiest systems tend to have more balanced (in terms of performance) cores. Our easiest 16 core system for example, has three high-end x86 cores, while all others are medium performance Alpha, Thumb, or x86.

We use EoS to select benchmark systems for the remainder of this paper. Due to space restrictions, we cannot present all architectural details for each core in the system, unless we limit our study to a very small number of systems. In this work, we consider benchmark systems, characterized by their EoS instead of their architectural traits. Specifically, we use three 4-core, three 8-core, and three 16-core benchmark systems. For each category we use the easiest, medium, and most difficult system. With our benchmark systems ranging from maximum to minimum EoS, and from 4 to 16 cores, we cover a wide range of heterogeneous-ISA multicore systems.

3.3 Expected Scheduler Efficiency (ESE)

ESE is defined as the ratio of average system αIPC over the average αIPC of an optimal scheduler, given a predictor characterized by some mean prediction error and standard deviation (of prediction error) values. We find that more accurate predictions do not necessarily translate into more efficient schedulers. Instead, the system's EoS can have a significant impact, as well as the predictor's accuracy uniformity (standard deviation of prediction error).

In this experiment, we evaluate a typical predictor-based scheduler on each of our 9 benchmark systems. We perform a sweep over possible predictive units by varying Mean Absolute Error (MAE) values from 0 (very accurate) to 1.5 (high prediction error) αIPC and we use three different standard deviation (STD) values (low, medium, high). Prediction error is randomly drawn from a normal distribution characterized by MAE and STD, and applied on the real (oracularly obtained from cycle-accurate simulations) αIPC values. The assumed scheduler receives predictions and decides on a *greedily optimal schedule* that maximizes overall αIPC . We negate absolute error values (since we use MAE), based on the outcome of a fair coin flip before we add it to true performance. Once again, we perform 300 runs with 200 workloads scheduled in each run to measure average αIPC for each data point.

Figure 2 presents our ESE evaluation results. We label each of our benchmark systems using 'E' (for Easy), 'M' (Medium), and 'H' (Hard), followed by the number of cores. For example, "M16" is a 16-core system of medium scheduling difficulty as identified by our EoS analysis. We also include the EoS value of each system in our results. We first observe that standard deviation demonstrates equal or even higher impact on ESE than MAE. Increased system difficulty further exacerbates this effect. For example, the H4 system's ESE drops by up to 17% within the MAE range, but by 19% within the STD range. We further observe that even in the presence of predictors with very high accuracy and uniformity, the system itself dictates how efficient the final scheduler can be, with statistical drops over optimal scheduling of 8%, 13%, and 16% for our 16-core (E,M,H respectively) benchmark systems. On the other hand, the easier a system is, the more immune it becomes to prediction error, with ESE lines not varying much as MAE increases.

Lastly, we find that all our systems demonstrate a significant error tolerance slack: We observe no more than a 2% ESE drop for MAE values under 0.3 and STD=0.2 across all systems. Furthermore, as STD increases, MAE slack also increases, since the predictor is closer to random and eventually plateaus to a low expected efficiency. Beyond the system's EoS level, it is worth noting that even with 100% accurate predictions, a greedy scheduler (locally optimal decisions, no knowledge of the future) cannot guarantee optimal scheduling.

In conclusion, based on our results, and assuming full and extended system utilization, we argue that ESE analyses can unveil tradeoffs during the design phase of future heterogeneous-ISA CMP systems. It is possible that system peak performance can be traded off in order to ensure more uniformity during execution, as well as cost savings on scheduler development. Both EoS and ESE metrics are computationally easy and cost effective exploratory tools.

4 Experimental Framework

This study requires a diverse set of evaluation frameworks. First, to train and evaluate machine learning models, we develop an in-house ML suite which allows us to explore a variety of ML models in depth. Second, to assess the capabilities of trained ML models

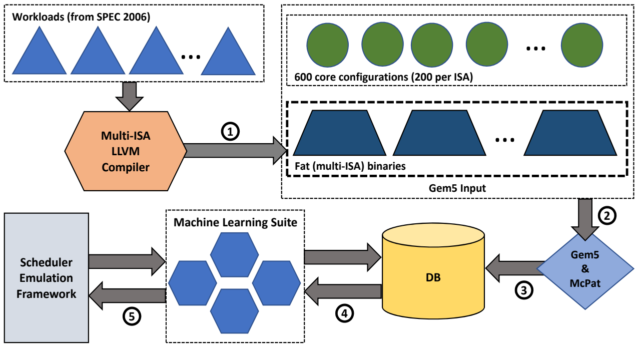


Figure 3. Experimental Framework overview

when used within schedulers, we implement a runtime scheduler framework that makes greedily optimal decisions periodically.

We use the simulation-based dataset collected during the design-space exploration published by Venkat et al. [43]. Figure 3 presents a high-level overview of our experimental framework, including the toolchain flow for creating the dataset. Our frameworks allow us to efficiently explore the interaction between ML models and multi-ISA software and hardware.

4.1 Machine Learning Suite

The Machine Learning aspect of this work is a multi-dimensional problem. On one hand, we seek to explore the behavior of various ML algorithms, with each having its own set of parameters that could potentially improve its accuracy if set at proper values. On the other hand, we work with a significantly large set of features, on which we need to have fine grain control across experiments. For example, we want to have the ability to study how each ML algorithm reacts if we stop providing cache-related features (capacity, number of hits and misses, etc), branch misprediction statistics, etc. Our ML suite supports all our requirements and ensures a fair comparison across all the ML models explored in this work. At its core, our ML suite utilizes the sklearn Python library [36].

The ML suite supports a variety of boolean flags for enabling and disabling feature groups (e.g. cache information) at the model's input, as well as ML optimizations (data scaling and whitening). Table 1 presents an overview of all available flags. Besides control over the input and optimizations, we also have the ability to select which ML algorithm will be used and what kind of training will be performed (i.e. how our database will be split into train and test sets - discussed in Section 6).

Once the parameters are populated with the desired values, our suite includes functions to train, test and score each model. Due to the multiple control options in our study, each machine learning algorithm we explore can have 1024 different configurations (10 boolean flags). Tree-based algorithms have 5120 flavors because of the tree depth variable.

4.2 Scheduler Framework

We developed an event-driven scheduler framework, in order to compare predictors, when used within schedulers. When studying ML-based schedulers, our framework is directly linked to the ML suite presented earlier. The model-under-test is first restored and trained. Once it is ready, our scheduler queries the ML suite to retrieve predictions when necessary. Non-ML-based schedulers such as oracular and random, are implemented directly in our framework.

Flag	Description
Scale	Performs data scaling
Whiten	Performs data whitening
RefCore	Reference core selection (1-600)
KeepRAW	Keeps features with raw values (e.g. INT_R)
KeepPower	Keeps profiled dynamic power features
KeepL1	Keeps L1 cache features
KeepL2	Keeps L2 cache features
Target	Sets the prediction target (typically, aIPC)
KeepRefTarget	Keeps the profiled target (aIPC) value
KeepFI	Keeps Fetch/Issue rate measurements
TreeDepth	Used with tree-based algorithms

Table 1. Configuration flags description (ML Suite)

To compare predictors, we first configure the target system by selecting cores from our database using their unique identifier. We then select and if necessary, train the predictor, before scheduling begins. We designed our scheduler to mimic an execution scenario where workloads appear one at a time as soon as an in-flight workload finishes. In other words, at any given time, the number of in-flight workloads equals the number of cores in our system. Our scheduler's goal is to greedily (no knowledge of future workloads) schedule these workloads in order to achieve the maximum sum of αIPC across all cores, according to the αIPC predictions it receives. Once a schedule has been identified, workloads are migrated to their new cores and resume execution.

In this work, we assume that all our workloads have been profiled on one reference core, which is not necessarily part of the underlying system. Our ML models are trained to provide "one-to-all" predictions. In other words they extrapolate application behavior on the reference core, to the target core. During evaluation we provide our models with a choice of two medium-performance reference cores, one Alpha and one x86, and we allow them to select one to be used in future experiments. Our methodology can be adapted to use one or more of the system's cores as reference, as well as train more specialized predictors for each pair of underlying cores. We choose to use one predefined reference core for two reasons: (1) we can better study the generality of our models in a one-to-all prediction setting, and (2) training more specialized, pairwise predictors would require severely reducing our dataset (to only include two cores instead of 600).

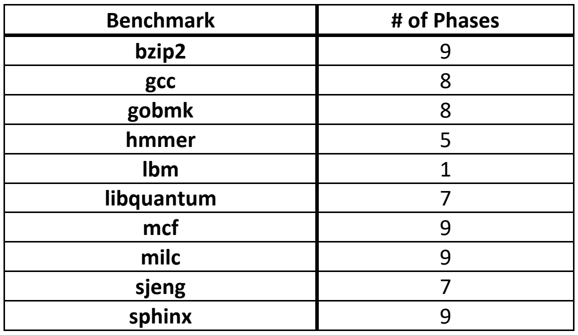
5 Database Description and Data Optimizations

This section describes the database we use throughout this work and our data splitting methodology. Table 2 lists all the workloads (benchmark phases – 100M instructions each) in our database. Table 3 presents the configurations of the cores in our dataset. A justification is presented on why these particular configurations are chosen out of a vast set of possible – but often unrealistic – microarchitectural options, in the original proposal [43].

Each database entry consists of 48 features. Of those, 30 are workload characteristics obtained via gem5-based profiling. The remaining 18 characterize cores and are statically available features. We refer to these two sections of our database as the "workload" and "core" sections.

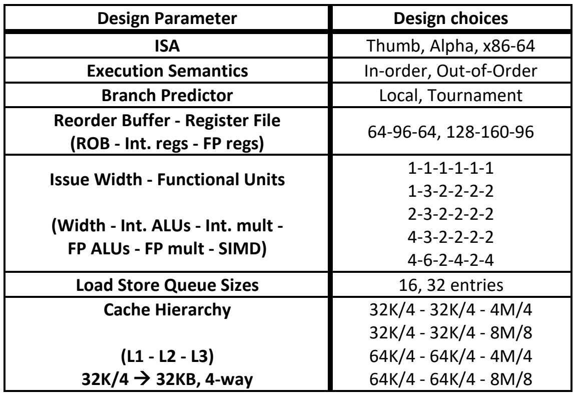
5.1 Workload and Core Sections:

Dynamic workload features can vary according to the core they execute on; for example, instruction count will be constant within



Benchmark	# of Phases
bzip2	9
gcc	8
gobmk	8
hmmer	5
lbm	1
libquantum	7
mcf	9
milc	9
sjeng	7
sphinx	9

Table 2. Breakdown of workloads in our database.



Design Parameter	Design choices
ISA	Thumb, Alpha, x86-64
Execution Semantics	In-order, Out-of-Order
Branch Predictor	Local, Tournament
Reorder Buffer - Register File (ROB - Int. regs - FP regs)	64-96-64, 128-160-96
Issue Width - Functional Units (Width - Int. ALUs - Int. mult - FP ALUs - FP mult - SIMD)	1-1-1-1-1-1 1-3-2-2-2-2 2-3-2-2-2-2 4-3-2-2-2-2 4-6-2-4-2-4
Load Store Queue Sizes	16, 32 entries
Cache Hierarchy (L1 - L2 - L3) 32K/4 → 32KB, 4-way	32K/4 - 32K/4 - 4M/4 32K/4 - 32K/4 - 8M/8 64K/4 - 64K/4 - 4M/4 64K/4 - 64K/4 - 8M/8

Table 3. Description of core configuration points.

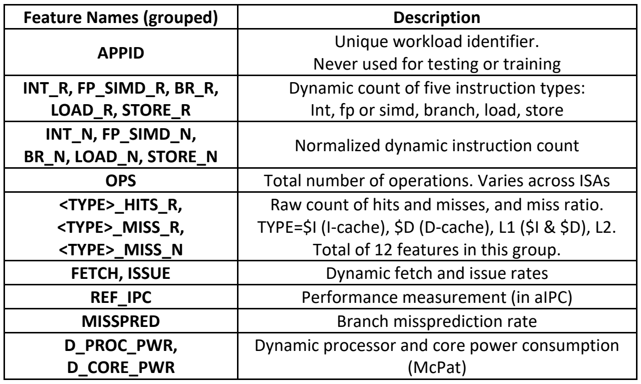
an ISA, but different across ISAs, while cache misses will vary by core features and ISA. Table 4 presents a breakdown of these features, grouped for brevity.

We characterize each core using 18 features. Each feature is collected from the core's microarchitectural specifications and is available from the manufacturer. Table 5 presents all the core features in our database. Some of our features are categorical, without an intrinsic order, for example a core's ISA. In other words, they cannot be ordered from best to worse, smallest to largest etc. Such features are a problem when used with certain ML models. "1-hot encoding" is a commonly used technique that provides a simple solution to this problem. Ordinal features such as cache sizes do not present this problem.

5.2 Data Splitting

Our decision to use benchmark phases as workloads complicates how we split our database into train and test sets for correct evaluation. If we follow the commonly used ratio split method (database entries randomly assigned to either train or test set), we introduce information leakage between the two sets: if some workload A, runs on CPUs 1 and 2, these two database entries should not be split across train and test sets, because the trained predictor would have full information about workload A when making predictions. Similarly, workload A and workload B, on any pair of CPUs, where A and B are different phases of the same benchmark, should also not be split across train and test, because the phases may actually share some code, and certainly share the dataset.

We use our ML suite to study both these problems: When using random data split, we observe excellent prediction accuracy (0.006 MAE) over the (supposedly) unseen test set, which is of course a consequence of this particular splitting approach and not a representative result. When we assign all 9 phases of the *mcf* benchmark as our test set (all other benchmarks in train set), we measure 0.48 MAE, which is quite high. Moving just one mcf phase from test to train, MAE immediately drops to 0.16.



Feature Names (grouped)	Description
APPID	Unique workload identifier. Never used for testing or training
INT_R, FP_SIMD_R, BR_R, LOAD_R, STORE_R	Dynamic count of five instruction types: Int, fp or simd, branch, load, store
INT_N, FP_SIMD_N, BR_N, LOAD_N, STORE_N	Normalized dynamic instruction count
OPS	Total number of operations. Varies across ISAs
<TYPE>_HITS_R, <TYPE>_MISS_R, <TYPE>_MISS_N	Raw count of hits and misses, and miss ratio. TYPE=$I (I-cache), $D (D-cache), L1 ($I & $D), L2. Total of 12 features in this group.
FETCH, ISSUE	Dynamic fetch and issue rates
REF_IPC	Performance measurement (in aIPC)
MISSPRED	Branch missprediction rate
D_PROC_PWR, D_CORE_PWR	Dynamic processor and core power consumption (McPat)

Table 4. Dynamically-collected feature groups in our database.

A better way to split data in such cases is to use a cross validation technique named Leave-One-Group-Out (LOGO) [35]. Following the LOGO strategy, one group is assigned as the test set, while all other groups form the train set. For correct evaluation under this technique, training and testing are performed in a loop, with each group assigned as the test set at least once.

To solve the information leakage problem caused by our workload definition and increase the generality of our evaluation, we define LOGO groups as entire benchmarks, with all their phases, including runs on all core types. Training and testing is performed over all groups for each ML model and we measure MAE across all iterations. We use single-benchmark LOGO for this exploratory phase of our work.

6 Deciphering ML Models

6.1 ML Model comparison

We perform a brute-force exploration over the space generated by our configuration flags (Table 1), to identify the best variation for each of our three algorithms.

We rank the top models based on their prediction accuracy (Mean Absolute Error) over the test set. To study the impact of the reference core's ISA, we use a mid-range Alpha as well as its equivalent x86 core as reference cores (an equivalent core is one where the microarchitecture features are constant, only ISA-specific differences). Our test sets still contain cores from all 3 ISAs, so regardless of the reference core chosen, all our models make αIPC predictions for all other cores in our database. Our ML framework allows us to pick any of the 600 cores as the reference core.

Table 6 presents our evaluation results as well as the configuration of each model. we present MAE, and STD results of each model when queried on their train set entries (all previously seen data), in addition to showing results when queried with the test set. Without loss of generality, we can expect any scheduler to occasionally be confronted with the same applications, possibly even similar inputs – querying with train set entries gives an upper bound to the quality of the predictions when a new workload closely matches a previously seen one.

On the configuration side, the top models have some interesting disagreements. First, an Alpha reference core is preferred by the

Feature Names	Description
CPUID	Unique core identifier. Never used for testing or training.
ISA (THUMB, ALPHA, X86)	Core's ISA. Categorical feature.
SEMANTICS (IO, OOO)	In-order or Out-of-Order. Categorical feature.
BR_PRED (LOCAL, TMNT)	Core's branch predictor, local or tournament. Categorical feature.
WIDTH	Core's width
ROBSIZE	Reorder buffer size
INTREGS, FPREGS	Number of int and fp registers
LSQSIZE	Load-Store Queue size
L1SIZE, L2SIZE	Cache sizes for L1 (kB) and L2 (LLC, MB)
INTALUS, FPALUS	Number of int and fp ALUs
INTMULTDIV, FPMULTDIV	Number of int and fp multiply-divide units
SIMD	Number of SIMD units
AREA	Core's area (McPat)
PEAKPOWER	Core's peak power (McPat)

Table 5. Statically-collected core features in our database.

RR model, while the tree-based models prefer the x86 core. For comparison, the equivalent RR model that uses the x86 reference core, reports 5.4x higher prediction error over the test set and the most accurate RR model that uses the x86 reference core shows 13% lower prediction accuracy compared to the top RR model. Second, our three models also disagree on whether to keep dynamic power consumption measurements, with only the RF model utilizing this information. Third, the RR model scales its input features and drops all raw measurements (very high values) in an attempt to keep all its feature values roughly in the same range which is a common optimization for linear models. Finally, the only features kept by the RR model in addition to normalized dynamic instruction breakdown (cannot be turned off by any model) are L1 cache and reference target features (profiled αIPC on reference core). We later explore which of these features our models consider important and how much weight they are assigned on the final prediction.

Interestingly, the DT model chooses to ignore L1 cache features and the αIPC measurement from the reference core and instead works with raw values, measured fetch and issue rates, and L2 cache features. No optimizations are applied since the DT algorithm shines at internally dealing with messy data. The RF model keeps the highest number of features, but ignores all cache information. It also chooses to whiten its features. This is because keeping correlated features reduces the variance between the forest's trees and as a result reduces prediction accuracy. For comparison, the equivalent RF model without whitening reports a 6% higher MAE.

All three of our models enjoy a very large training dataset, due to the LOGO splitting technique. Such a dataset is not unreasonable in a production-level scheduler, given periodical re-training with newly observed workloads. However, we later (Section 8) show that the RR model is much more sensitive to reduction of the train set size, whereas tree-based models remain within error tolerance.

6.2 Linear Coefficients (LC) Analysis

The linear coefficients analysis can be applied to linear models and gives us an understanding of how it internally utilizes its input variables. During training, Ridge (linear) Regression algorithms generate an equation of the form: $\overline{\alpha IPC} = \sum_{i=1}^{n} c_i f_i$ where n is the number of features, and c_i is the coefficient associated with the feature f_i. Features with higher *absolute* coefficient values have more impact on the final prediction than those with lower coefficient values. The linear coefficient analysis simply sorts each feature based on its significance. However, features with extremely high measurement values compared to the rest of the input can still have significant impact even with lower coefficients. Since our top RR model applies the scaling optimization, we know that all our features have the same mean value of zero, and standard deviation of one. This allows us to simply sort the coefficients themselves in order to understand each feature's significance.

Our results presented in Figure 4(a) reveal that this linear model assigns extremely high weight on the hardware characteristics of the target core. We only present features that have normalized weight of at least 1%. Interestingly, this model assigns significant weight on the type of branch predictor of the target core (LOCAL or TMNT). It also assigns some weight to the target's ISA, as well as its number of functional units, and execution semantics (IO or OOO). When we look at the collection of hardware features that are considered important, it appears that this model is attempting to quantify the computational capacity of its target core. Such an approach is reasonable since this model is only allowed to draw a straight αIPC prediction line. Its best bet is to predict performance based on the overall computational capacity offered by the core. If instead it tried to predict based on dynamic workload measurements or even a combination of the two, the relationship is likely to be non-linear. We notice that even though the profiled αIPC from the reference core is part of this model's input (refer to Table 6), it is assigned a weight less than 1% – in other words it is an insignificant part of the prediction equation derived by our most accurate RR model. Finally, we observe that this model picks one dynamic (profiled) feature and assigns it a small weight – the number of dynamic *instruction* cache misses, which confirms the fact that instruction cache misses, although infrequent, can have a substantial impact on overall performance.

6.3 Mean Decrease of Impurity (MDI) Analysis

Feature impurity analysis is performed on tree-based models. We apply it on our top random forest, since a collection of decision trees gives us a broader view of the feature space. During tree construction, these algorithms create decision nodes with each node being a "less than X" condition on a single feature, where X is typically a real number.

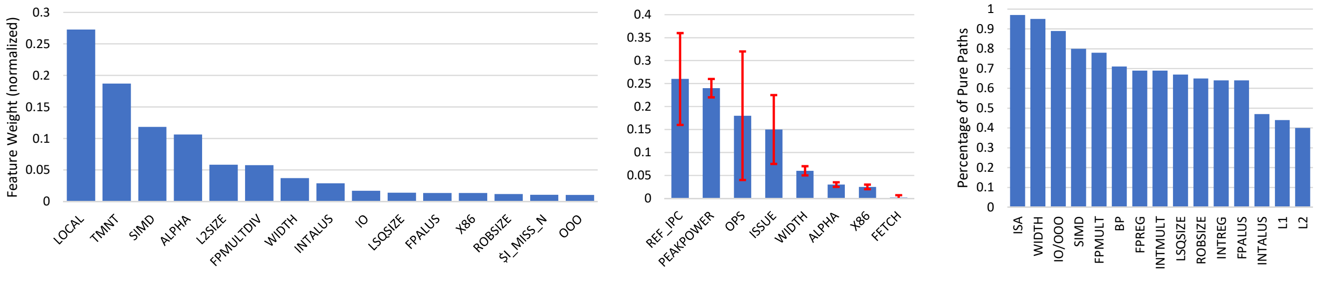
Similarly to linear models gradually adjusting feature weights to minimize prediction error, decision node conditions are chosen such that they minimize the "impurity" of the two subtrees generated by that node. Since we are using *regression* models (not classifiers), the impurity measurement is defined as the variance between all final responses (predictions) of each subtree. During training, we can compute how much each feature decreases the weighted impurity in a tree (e.g. how narrow it makes the subtree's prediction range). Since our RF model performs data whitening, we avoid a shortcoming of this analysis, where the presence of highly correlated features gives the appearance of lower importance (because correlated features can be used interchangeably at decision nodes).

Figure 4(b) presents our impurity analysis results. Similarly to Figure 4(a), we only plot the features with at least 1% of normalized importance. Since we study a *collection* of predictors, we are also able to plot standard deviation lines showing how each feature is ranked amongst all trees.

The RF model assigns high significance to fewer features than the RR model (8 vs 15). We also observe that this collection of features seems more reasonable for performance predictions. For

Model	Test Set		Train Set		Configuration of best model after full space exploration										
	MAE	STD	MAE	STD	Scale	Whiten	Ref Core	Keep RAW	Keep Power	Keep L1	Keep L2	Keep Ref. Target	Keep FI	Tree Depth	#Trees
RR	0.23	0.3	0.21	0.26	✓	✗	Alpha	✗	✗	✓	✗	✓	✗	--	--
DT	0.22	0.35	0.07	0.04	✗	✗	x86	✓	✗	✗	✓	✗	✓	12	--
RF	0.19	0.31	0.03	0.04	✗	✓	x86	✓	✓	✗	✗	✓	✓	11	10

Table 6. Comparison of top ML Models



(a) LC Analysis

(b) MDI Analysis

(c) RPP Analysis

Figure 4. Normalized feature importance from three different analysis techniques. Refer to tables 4 and 5 for a description of our keywords. LC analysis performed on top RR model. MDI performed on top RF model. RPP performed on near-oracular DT predictor.

example, if we focus on their average importance, the profiled αIPC is considered the most influential feature, followed very closely by the target core's static peak power (not to be confused with the workload's profiled dynamic power consumption).

Peak power is a very good proxy for the peak performance of the target core – power consumption is a major factor for any commercial core produced, so manufacturers are typically not willing to increase their product's peak power consumption unless it offers significant gains in performance.

Interestingly, we observe significant difference in the amount of deviation in the top two features. While all trees consider peak power equally important (predict based on target core), the importance of the profiled αIPC varies significantly amongst trees. This partially explains the surprising decision of the linear model presented earlier to not use any of these measurements. Even though this feature combination is meaningful and according to our results successful, their relationship towards the prediction target is non linear (we verify this phenomenon by visualizing all our database entries). Random forests on the other hand are able to exploit this non-linear relationship by having only a small set of trees using profiled αIPC while the remaining trees use other measures to minimize the overall prediction error, and at the same time use the peak power feature in all trees. Clearly, this feature combination is very strong given the excellent prediction accuracy of this model on the train set.

We further observe an architectural feature overlap between the RF and RR models – the target core's instruction width and ISA (all with relatively low importance but also short deviation lines). Notice that when a decision tree path asks the two ISA questions sequentially, it is able to differentiate between all three of our ISAs, unlike the linear model which ends up with less information. Instruction width and ISA are among the top features in our best DT model as well (MDI analysis results not shown), showing a strong consensus among all types of predictors on the usefulness of these variables. With the ISA variable playing such an important role, it is clear that prediction techniques focused on single-ISA heterogeneous datasets cannot predict accurately when applications cross ISAs. To the best of our knowledge, we are the first to design predictors that remain accurate across general-purpose ISAs.

We finally observe that this model also focuses on profiled features such as the profiled number of ops (very high deviation), issue, and fetch rates. It stands to reason that fetch and issue rates hint towards the workload's parallelism levels which eventually affect its performance. The amplitude of this effect depends on the target core's characteristics, which are also queried by our RF model.

6.4 Reverse Path Purity (RPP) Analysis

Consider a decision tree model which has near-perfect prediction accuracy. This model is essentially a collection of paths that start at the root node and end on leaf (decision) nodes. Each path asks a sequence of questions that "filter" the input query. Intuitively, if we observe that some *categorical* feature X which has N possible values gets cleanly separated amongst the tree's paths, then we can argue that the given model deemed important to always "know" the exact value of X before making a prediction. In this experiment we refer to tree paths as pure, if and only if they can be traversed by exactly one value of some categorical feature. Paths that are shared by two or more values are referred to as impure.

It is challenging to measure path purity by simply traversing all paths and reading their questions, because some features can be extrapolated by questions on other features. For example, our tree can ask "Is the target ISA Thumb?", or it can ask "Is there floating point support on the target core?". We devise this experiment to overcome the possibly unpredictable combination of questions that can lead to a single feature value, and it allows us to measure path purity for all features that have a finite number of values.

We begin by creating a near-perfect predictor. We achieve this by training our best DT model using the entirety of our dataset (everything is in the train set). This leads to minimal MAE and STD (both are measured below 0.02). Of course the actual accuracy measurements are irrelevant to this study, but we now know that we have a predictor that successfully converted our entire dataset to 2117 (measured) distinct paths of up to 12 questions each.

In order to measure path purity, we first pick the categorical feature we seek to study. For this walkthrough example, let's assume we pick the ISA feature. Then, we query this model for a prediction for each row of our dataset. We maintain a map between leaf node ID and a count of different ISA values that reached it; by definition exactly one path can reach one leaf node. Those nodes that only counted a single ISA value are pure, while all others (with 2 or 3 values) are not. Clearly, if the model internally fully separates ISAs to different paths, it is of paramount importance (to the model) to "know" which ISA it's predicting for, leading us to the conclusion that ISA is an important feature. If on the other hand, some other feature is shared in most paths, then – regardless of the features implicit weight – this accurate model cares less about extracting the exact value, thus the feature is of lower importance.

We plot our findings in Figure 4(c). We see that a near-perfect model separates the target core's ISA and width almost entirely (97% and 95% of the paths are pure respectively). Execution semantics (inorder/OoO) follow with 89% path purity. The majority of our core-describing features lies between 80% and 60%, while cache sizes and the number of integer ALUs are below 50% (the majority of their paths are impure). This experiment proves that ISA, width, and execution semantics are of paramount importance when predicting performance for heterogeneous-ISA systems. This finding is in agreement with our insights derived in the previous two experiments, however we now have a clear image of just how important these features are. The reason previous experiments do not show higher weights on some of the features (e.g., the ISA) is because they can often be extrapolated via other features.

7 Predictor Overhead Comparison

This section presents an overhead comparison for our three models. We perform the following experiments on a system with an i7 core running at 2.9GHz and 16 GB of memory. We use the algorithm implementations found in sklearn [36].

First, we vary the dataset size and train each model 100 times to report average training overhead. All algorithms are measured to have linear algorithmic complexities for training, however RR has the lowest slope of 1.4, followed by DT (14.3) and RF (92.7). Training times for a dataset of 30k entries are 21ms, 203ms and 1.4s for RR, DT, and RF respectively. Training overhead has little significance in choosing a model, since it happens infrequently. For extremely large datasets however, RF's overhead might become prohibitive.

To measure query overhead, we ask each trained model for 55k predictions and report the average time per prediction. Query overheads for RR, DT and RF are 24ns, 29ns, and 667ns respectively. Query overhead is an important metric, since predictions can be part of the critical path, especially in the context of job scheduling. RR and DT are comparable, however RF is significantly slower.

Our deciphering methodologies presented in Section 6 identify which dynamic measurement each model requires. Dynamic measurements require HW support, specifically on-chip counters. We find that the RR model requires one counter (# instruction cache misses) and the DT model needs 4 counters (# ops, # L2 cache misses, # load instructions and # of branch instructions). Finally, the RF model requires 5 counters (profiled αIPC , issue and fetch rates, # ops and # loads). We note that most modern cores already feature these performance counters.

Finally, we measure the memory overhead of each model. Linear models only need to store their coefficients' values. Tree-based models must store a condition value and a feature identifier for each node. We assume that all condition and coefficient values require 64 bits and the number of features defines the number of bits necessary to represent them. We finally assume that our models only keep the features with high (> 1%) importance. The RR model needs 960 bits, the DT model 276kB, and the RF model 1.5MB. Overall, RF is very expensive compared to the others. Although it does provide high accuracy, the predictions do not necessarily translate to a significant gain in scheduler efficiency. Decision trees are larger than RR, but not prohibitively large, and just slightly slower in predictions/training. They are also more accurate and a little more efficient as schedulers.

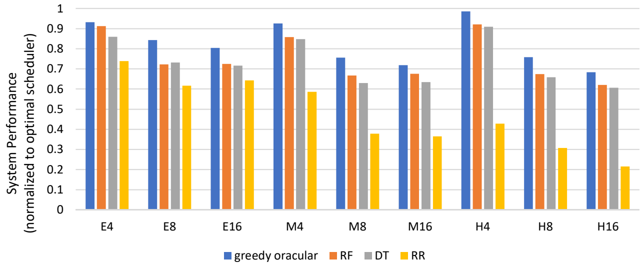


Figure 5. Scheduler performance comparison.

8 Scheduler Evaluation

This section materializes our insights derived in the exploratory part of this work. Specifically, we train our most accurate RR, DT, and RF models using only the features we identify as important in Section 6. Each model is then linked to a scheduler that receives a prediction matrix and derives a greedily optimal schedule to maximize performance. Using this scheduler framework, we run 200 workloads to completion on each of our 9 benchmark multicore systems identified in Section 3.2. We compare our schedulers against a greedy oracular scheduler that consistently makes 100% accurate performance "predictions" on every probe.

We no longer use the LOGO data splitting approach to measure the overall scheduler efficiency. Instead, we increase the size of the test set and reduce the size of the training set, by assigning four randomly chosen benchmarks (instead of one) to be our test set. While this could potentially result in reduced predictor quality in terms of MAE and STD, it enables a larger selection of previously unseen workloads allowing us to explore more realistic computation environments.

For this experiment, we randomly select input workloads that create a mix of 50% unseen (from test set) and 50% previously seen workloads (from train set), resembling a typical IaaS (Infrastructure-as-a-Service) environment (e.g., Amazon's EC2) – some users use EC2 instances to run the same application every time (such as a web server), while others execute a more diverse mix of workloads (software development and testing).

8.1 Performance Analysis

Figure 5 presents our results normalized to the average performance under the optimal scheduler that emits a schedule resulting in the maximum-achievable overall throughput. We first observe that the RF-based scheduler has an advantage compared to the other ML-based schedulers. However, the much cheaper DT-based scheduler scores very close, across all systems. Compared to RF, DT reports a 2.8% average performance reduction (6.2% max reduction), while it outperforms RF on the E8 system by 1.2%. We further observe that our RF scheduler is within 2.2-11.2% (7.5% on average) from an oracular scheduler, and DT within 1.6-16.8% (10% average).

Our RR-based scheduler shows significantly reduced performance. The reduced training set size affects our top RR predictor's accuracy significantly enough to move it past the error tolerance zones we identify in our ESE analysis. While the (test set) accuracy of all our models drops due to the reduced training data set, RR is affected the most, with its MAE doubling and STD increasing by almost 3x. For comparison, tree-based models only experience 6% MAE and STD degradation. Furthermore, tree-based prediction accuracy on the training set is slightly better (by 1%), which provides a significant advantage over RR in this experiment. RR remains roughly within the same levels of accuracy on the training set.

8.2 Sensitivity to Underlying System

Figure 5 allows us to study the sensitivity of each scheduler as the underlying difficulty increases. We observe that our tree-based schedulers are affected by the transition from 4 to 8 cores at a higher degree compared to the oracular scheduler, but their performance remains almost intact by the transition from 8 to 16 cores. The oracular scheduler is affected by both transitions, however with a smaller impact. The RR scheduler is affected by all transitions as well except on the easy systems. Our results show the tree-based schedulers to provide roughly equal performance in all 8 and 16 core systems, essentially overcoming the increase of scheduling difficulty.

9 Conclusions

With heterogeneity becoming more and more available in modern systems, predictive models enjoy increased attention from the scientific community for applications in scheduling, power management, resource management, resource allocation and more. In this work we first present an exhaustive exploration and analysis of the abilities of ML models to act as performance predictors. Our results reveal that accuracy metrics typically used in ML works do not necessarily translate to computer architecture applications such as scheduling in a predictable fashion. Furthermore, we reverse-engineer trained ML models and conclude on which measurements are necessary for accurate predictions. We finally materialize our insights in three ML-based schedulers for heterogeneous-ISA systems and demonstrate their effectiveness on systems of varying scheduling difficulty.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful insights. This research was supported in part by NSF Grant CNS-1652925 and NSF/Intel Foundational Microarchitecture Research Grant CCF-1823444.

References

[1] The Benefits of Multiple CPU Cores in Mobile Devices. Tech. rep., NVidia, 2010.

[2] Variable SMP âĂŞ A Multi-Core CPU Architecture for Low Power and High Performance. Tech. rep., NVidia, 2011.

[3] Qualcomm snapdragon 810 processor: The ultimate connected mobile computing processor. Tech. rep., Qualcomm, 2014.

[4] Apple 2017: The iphone x (ten) announced.

[5] Akram, A. A study on the impact of instruction set architectures on processor performance.

[6] Akram, A., and Sawalha, L. The impact of isas on performance. In *WDD'14*.

[7] Ardalani, N., Lestourgeon, C., Sankaralingam, K., and Zhu, X. Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance. In *MICRO'48* (2015), ACM, pp. 725–737.

[8] Baldini, I., Fink, S. J., and Altman, E. Predicting gpu performance from cpu runs using machine learning. In *SBAC-PAD* (2014), IEEE, pp. 254–261.

[9] Barbalace, A., Lyerly, R., Jelesnianski, C., Carno, A., Chuang, H.-r., and Ravindran, B. Breaking the boundaries in heterogeneous-isa datacenters. In *ASPLOS* (2017).

[10] Barbalace, A., Sadini, M., Ansary, S., Jelesnianski, C., Ravichandran, A., Kendir, C., Murray, A., and Ravindran, B. Popcorn: Bridging the programmability gap in heterogeneous-isa platforms. In *ECCS* (2015), ACM, p. 29.

[11] Bhat, S. K., Saya, A., Rawat, H. K., Barbalace, A., and Ravindran, B. Harnessing energy efficiency of heterogeneous-isa platforms. *ACM SIGOPS Operating Systems Review 49*, 2 (2016), 65–69.

[12] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[13] Chen, J., Nayyar, N., and John, L. K. Mapping of applications to heterogeneous multi-cores based on micro-architecture independent characteristics. In *Third Workshop on Unique Chips and Systems, ISPASS2007. April 2007* (2017).

[14] Cheng, D., Jiang, C., and Zhou, X. Heterogeneity-aware workload placement and migration in distributed sustainable datacenters. In *IPDPS* (2014).

[15] Chitlur, N., Srinivasa, G., Hahn, S., Gupta, P. K., Reddy, D., Koufaty, D., Brett, P., Prabhakaran, A., Zhao, L., Ijih, N., et al. Quickia: Exploring heterogeneous architectures on real prototypes. In *HPCA* (2012), IEEE.

[16] Corporation, D. E. Alpha architecture reference manual.

[17] Craeynest, K. V., Jaleel, A., Eeckhout, L., Narvaez, P., and Emer, J. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *ISCA* (June 2012), pp. 213–224.

[18] DeVuyst, M., Venkat, A., and Tullsen, D. M. Execution migration in a heterogeneous-isa chip multiprocessor. In *ASPLOS XVII* (2012).

[19] Greenhalgh, P. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White paper* (2011), 1–8.

[20] Hayashi, A., Ishizaki, K., Koblents, G., and Sarkar, V. Machine-learning-based performance heuristics for runtime cpu/gpu selection. In *PPPJ* (2015).

[21] Hill, M., and Marty, M. Amdahl's Law in the Multicore Era. *Computer* (July 2008).

[22] Hoerl, A. E., and Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics 12*, 1 (1970), 55–67.

[23] Intel. Intel 64 and ia-32 architectures software developerâĂŹs manual.

[24] Kahle, J. The cell processor architecture. In *MICRO* (2005), IEEE Computer Society, p. 3.

[25] Kumar, R., Farkas, K. I., Jouppi, N. P., Ranganathan, P., and Tullsen, D. M. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO* (2003).

[26] Kumar, R., Tullsen, D. M., Ranganathan, P., Jouppi, N. P., and Farkas, K. I. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In *ISCA* (2004).

[27] Lee, W., Sunwoo, D., Emmons, C. D., Gerstlauer, A., and John, L. K. Exploring heterogeneous-isa core architectures for high-performance and energy-efficient mobile socs. In *GLSVLSI* (2017).

[28] Limited, A. Arm7/tdmi technical reference manual.

[29] Mittal, S., and Vetter, J. S. A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR) 47*, 4 (2015), 69.

[30] Montgomery, D. C., Peck, E. A., and Vining, G. G. *Introduction to linear regression analysis*, vol. 821. John Wiley & Sons, 2012.

[31] Murthy, S. K. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data mining and knowledge discovery 2*, 4 (1998), 345–389.

[32] Nathuji, R., Isci, C., and Gorbatov, E. Exploiting platform heterogeneity for power efficient data centers. In *ICAC* (2007).

[33] Neter, J., Kutner, M. H., Nachtsheim, C. J., and Wasserman, W. *Applied linear statistical models*, vol. 4. Irwin Chicago, 1996.

[34] Nider, J., and Rapoport, M. Cross-isa container migration. In *SYSTOR* (2016).

[35] Refaeilzadeh, P., Tang, L., and Liu, H. Cross-validation. In *Encyclopedia of database systems*. Springer, 2009, pp. 532–538.

[36] scikit learn. Scikit-learn python library. http://scikit-learn.org/stable/.

[37] Somu Muthukaruppan, T., Pathania, A., and Mitra, T. Price theory based power management for heterogeneous multi-cores. In *ASPLOS '14* (2014), ACM.

[38] Torng, C., Wang, M., and Batten, C. Asymmetry-aware work-stealing runtimes. In *ISCA 2016* (2016).

[39] Variable, S. A multi-core cpu architecture for low power and high performance. *Whitepaper-http://www. nvidia. com* (2011).

[40] Venkat, A. *Breaking the ISA Barrier in Modern Computing*. PhD thesis, UC San Diego, 2018.

[41] Venkat, A., Basavaraj, H., and Tullsen, D. M. Composite-isa cores: Enabling multi-isa heterogeneity using a single isa. In *HPCA 2019* (2019).

[42] Venkat, A., Shamasunder, S., Shacham, H., and Tullsen, D. M. Hipstr: Heterogeneous-isa program state relocation. In *ASLPOS* (2016).

[43] Venkat, A., and Tullsen, D. M. Harnessing ISA diversity: Design of a heterogeneous-isa chip multiprocessor. In *ISCA* (2014).

[44] Wu, G., Greathouse, J. L., Lyashevsky, A., Jayasena, N., and Chiou, D. Gpgpu performance and power estimation using machine learning. In *HPCA* (2015).

[45] Zheng, X., John, L. K., and Gerstlauer, A. Accurate phase-level cross-platform power and performance estimation. In *DAC* (2016).