Platform-Agnostic Learning-Based Scheduling

Andreas Prodromou[1], Ashish Venkat[2], and Dean M. Tullsen[1]
[1] University of California, San Diego
{aprodrom,tullsen}@ucsd.edu
[2] University of Virginia
venkat@virginia.edu

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. Finally, we incorporate our findings in ML-based schedulers and evaluate their sensitivity to the underlying system's level of heterogeneity. We show our schedulers to perform within 2-11% of an oracular scheduler across a variety of underlying heterogeneous-ISA multicore systems without modification.

1 Introduction

Modern multicore architectures employ cores that are increasingly heterogeneous in terms of their microarchitectural characteristics. These architectures, dubbed single-ISA (Instruction Set Architecture) heterogeneous multicores [7, 8], improve the throughput and efficiency of mixed workloads, by catering to their diverse execution characteristics, such as variability in instruction-level parallelism, cache access patterns, branch behavior, etc. Heterogeneous-ISA multicore architectures [4, 17, 2, 16, 1, 10, 15] further exploit an additional degree of freedom by allowing multiple co-packaged ISAs. These architectures benefit from a phenomenon called *ISA affinity* – the inherent preference of an application code region to efficiently execute on a particular ISA [17, 4]. By synergistically exploiting ISA affinity and microarchitectural heterogeneity, these architectures provide significant ($> 30\%$) additional gains in performance and energy efficiency.

This increasing trend in on-chip heterogeneity has only further highlighted the critical need for intelligent workload scheduling, since the potential performance and power benefits of these architectures arrive from smart job-to-core assignment. State-of-the-art scheduling mechanisms rely on predictive models to make active performance predictions for any code region in execution, and further leverage such predictions to make appropriate job allocation decisions [13, 3, 18]. This work advances state-of-the-art scheduling mechanisms by developing learning-based schedulers that are flexible enough to operate efficiently, regardless of the degree of heterogeneity that the underlying hardware exhibits – including both microarchitectural heterogeneity and ISA-heterogeneity.

On a single-ISA heterogeneous multicore, the performance difference of a code region can be vastly different across different cores depending upon their

microarchitectural diversity. ISA heterogeneity further exacerbates the difficulty of performance prediction due to the more complex set of parameters that potentially affect performance. This paper deals with the challenging problem of scheduling mixed workloads on general-purpose heterogeneous-ISA architectures that employ fully disjoint ISAs (ARM's Thumb, x86-64, and Alpha) as proposed by Venkat and Tullsen [17].

This paper examines a variety of machine learning techniques to perform cross-core and cross-platform 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. These ML-based predictors are trained on an extensive simulation-based dataset [17] to capture the interaction between an application's execution characteristics and the architectural and microarchitectural traits of the underlying hardware, and further project the execution of any given code region on arbitrary processor hardware. The dataset includes 72 SPEC simpoint workloads, each simulated on 600 different cores executing multiple ISAs. To the best of our knowledge, this is the most extensive heterogeneous-ISA, cycle-accurate, simulation-based dataset in the literature.

The contributions of this work are:

1. Design of accurate ML-based cross-core and cross-platform performance predictors, which are to the best of our knowledge, the first predictors to effectively cross general-purpose ISA boundaries.
2. In-depth characterization of ML-based performance predictors.
3. Demonstration of an effective ML-based heterogeneous-ISA job scheduler.

2 Motivation

Cross-Platform Performance Prediction. Job schedulers benefit from accurate performance predictions. On a complex system, 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 (sacrificing performance) 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. For predictive schedulers, the prediction error should have a smaller impact on realizing efficient schedules on less diverse systems. This paper proposes and formalizes techniques to quantitatively estimate the **Ease-of-Scheduling** (EoS) on any given heterogeneous multicore, and further showcase it as an objective function that represents an unbiased way of selecting heterogeneous-ISA multicore benchmark systems.

We find that a system's scheduling difficulty level (EoS), a performance predictor's accuracy, and a scheduler's efficiency (one that internally uses said predictor) are intertwined in often unpredictable ways. For example, less accurate predictors do not guarantee worse schedulers, and more accurate predictors do not guarantee better ones. We address this ambiguity by first identifying multicore systems that cover a wide spectrum of heterogeneity, and then focusing on

training accurate per-core performance predictors. We finally aggregate our arsenal of tools (diverse systems, a variety of predictors, and schedulers) to measure the efficiency, overheads, and performance of our learning-based schedulers.

Limitations in Prior Work. This paper also identifies and addresses some common limitations in prior prediction-based scheduling proposals. First, multicore systems under test are often predefined and do not change throughout each proposal, resulting in inflexible solutions with low potential adaptability. Second, a closely related and frequently-observed drawback is the use of small datasets on which predictors are trained and evaluated. Note that large datasets are difficult to construct, 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.

In this work, we address these drawbacks. First, our EoS-based system selection allows us to demonstrate the adaptability of our schedulers. Second, we use the largest available dataset that fits our needs. Third, we use three different machine learning algorithms as predictors. Fourth, we never hand-select features for training – instead, we allow our machine learning models to (brute-force) explore the dataset during training and decide which features they should be using to optimize their prediction accuracy. We show that our methodology results in predictive models that generalize and adapt to a variety of underlying systems without modification.

3 Related Work

Single-ISA heterogeneous architectures are proposed by Kumar, et al. [7], with processor manufacturers currently offering heterogeneous products [5, 6, 14]. Scheduling and resource management for these architectures has been extensively studied, as well (examples include [13, 3, 12] and more).

The primary focus of this work, *general-purpose* heterogeneous-ISA multicore architectures [4, 17, 15], include microarchitecturally heterogeneous cores that implement different ISAs and have been demonstrated to result in significant added benefits compared to its single-ISA counterpart. The ISAs we use in this study are Thumb, Alpha, and x86-64. Recently, the composite-ISA architecture [15] has demonstrated how a heterogeneous-ISA CMP can be based on feature-diverse variations of a single ISA, significantly alleviating concerns regarding their commercial viability. This research would apply

to composite-ISA architectures as well. Beyond performance and energy benefits, exploiting heterogeneous-ISA architectures has been shown to improve security against code-based attacks, such as Return-Oriented Programming, by frequently switching ISAs [16]. Systems combining CPUs and GPUs (or accelerators) in the same package are also examples of heterogeneous-ISA architectures [11], however they are beyond the scope of this work.

Barabalace, et al. [2, 1] propose Popcorn Linux, an OS that facilitates running and migrating applications on general-purpose heterogeneous-ISA systems. Our work assumes similar support. Popcorn Linux includes a basic scheduler that relies on application instrumentation to generate a mapping of functions to cores. LACross [18] is a cross-platform scheduler that uses the LASSO linear regression algorithm to predict a phase's performance and power. LACross is shown to provide efficient resource management on two heterogeneous machines.

4 Dataset Overview

We use a Gem5 and McPAT simulation-based dataset of 43200 samples, consisting of 72 workloads simulated on 600 cores (200 microarchitecturally-diverse cores for each ISA - Thumb, Alpha and x86-64) [17]. Participating features can be separated into two main categories: (1) workload profiling features, and (2) core microarchitectural specifications. In this section we provide more details regarding the dataset we use.

4.1 Workloads and Core configurations

Each workload in our dataset is a Simpoint phase of 100 million (Alpha) instructions. Table 1 presents a description of the 600 cores in this dataset. Further details on how configuration points were chosen by the creators of this dataset can be found in [17].

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 1: Description of core configuration points. Adapted from the design space exploration presented in [17].

4.2 Dataset Partitions

Dynamic workload features can vary according to the core they execute on; for example, instruction count will be constant within an ISA, but different across ISAs, while cache misses will vary by core features and ISA. This section of the

dataset contains profiling measurements obtained via simulations. Specifically, each workload is characterized by 30 features, which are presented in Table 2a.

In addition to dynamic workload features, cores in our dataset are also characterized by 18 static features that describe their microarchitecture. These features are typically available from the manufacturer. Table 2b presents more details.

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 misspredict rate
D_PROC_PWR, D_CORE_PWR	Dynamic processor and core power consumption (McPat)

(a) Dynamically-collected features

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)

(b) Statically-collected features

Table 2: Dataset features: Workload profiling features (a) are collected from all 600 cores. CPU-microarchitectural features (b) describe each core.

4.3 Data Splitting Into Training and Test Sets

Due to the properties of this dataset, splitting it into a training and test set is a challenge. If not careful, we can leak information between the two sets, often resulting in unrealistically accurate predictors. We find that if a workload A runs on CPUs 1 and 2, both samples must reside in the same set. Otherwise, the trained predictor has 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 training and test, since phases may share code and certainly share the dataset.

We address these issues by employing a Leave-One-Group-Out (LOGO) cross-validation methodology. Following the LOGO strategy, one group is assigned as the test set, while all other groups form the training set. Training and testing are performed in a loop, with each group assigned as the test set at least once. We define each benchmark, with all its phases and including runs on all cores as a LOGO group.

5 Experimental Methodology

The scope of this work is twofold. First, we seek to explore a variety of ML-based predictors, each under numerous combinations based on their internal configuration parameters. Second, we want to study the efficacy of these predictors in the context of job scheduling.

5.1 Predictor Evaluation Methodology

In this work, we study three ML-based performance predictors:

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 3: Configuration flags description (ML Suite)

- **Ridge Regression (RR)**, a variant of Linear Regression that penalizes large coefficients, providing increased protection against overfitting.
- **Decision Trees (DT)** algorithms that generate a binary decision tree during training, with each decision node predicated on exactly one input feature.
- **Random Forests (RF)** that generate a collection of decision trees during training. When queried for a prediction, an average of the decision tree responses is returned.

Our trained models accept two inputs: (a) application-specific features (obtained via profiling on a "reference" core), and (b) target core-specific features. They output a performance prediction for the given application-core pair measured in *Alpha Instructions-Per-Cycle* (αIPC). When dealing with multiple ISAs, IPC is not a fair metric since the number of dynamically executed instructions can vary drastically across ISAs. We use Alpha as our base for comparison, allowing us to fairly track execution progress. Due to the use of a reference core, our predictors operate in a one-to-many fashion and are capable of extrapolating runtime performance, from the reference core to any target core in our dataset.

Workloads are characterized using 30 features in our dataset. It is possible that not all these features are necessary for accurate predictions. Furthermore, highly-correlated features or features at different scales can also have a negative impact, requiring appropriate data whitening and data scaling, respectively. Our predictor configuration methodology defines 10 boolean flags (Table 3). Each flag controls the amount of information available during training. For example, the "keepL1" flag defines whether the input to the ML model will include L1 cache information (size, number of misses, etc). The "RefCore" flag defines which of our 600 cores will be used as the reference (profiling) core. To reduce the number of experiments, we allow the RefCore flag to take two values: It can either use a mid-range Alpha core (core #300), or a mid-range x84-64 core (#500).

We perform a brute force exploration over the space generated from our flags. Consequently, we explore 1024 different configurations of the RR model, and 5120 versions of the DT and RF models ($\times 5$ since we also explore 5 different values of tree depth). Finally, we identify the most accurate predictor of each algorithm. Throughout our methodology, we avoid pre-selecting and hardcoding the features of our exploration. Instead, our ML models explore all possible configurations and "decide" which features lead to the most accurate predictions.

5.2 Scheduler Evaluation Methodology

Workload Execution Protocol. We enforce a workload execution protocol to ensure fairness during our experiments. For each system-scheduler combination we study, we execute 200 randomly chosen workloads from our dataset. A new workload appears as soon as a previous workload completes its execution, such that at any given time, the number of in-flight workloads equals the number of cores in the underlying system. For larger experiments (e.g. Ease-of-Scheduling), we draw a list of 200 workloads before the experiment begins and we use the same list for every sub-experiment. This way we can ensure fairness in comparing systems and schedulers.

Choosing Benchmark CMP Systems. To train robust performance predictors that can adapt to varying hardware heterogeneity, it is important to choose diverse benchmark multicore systems; not only in terms of microarchitecture, but also with respect to the resulting scheduling difficulty. We can expect that systems with similar cores will be easier to schedule than one with more diverse cores. For this work, we select three 4-core, three 8-core and three 16-core heterogeneous-ISA CMP benchmark systems. Specifically, we select an easy, a medium, and a hard system, that exhibit varying degrees of scheduling difficulty.

We devise an experiment, called "Ease of Scheduling", to rate systems in terms of their scheduling difficulty. EoS relies on the fact that a random scheduler will perform closer to an optimal scheduler (that provides the best performance) on easier systems (e.g., a homogeneous multicore system). As the system's diversity grows, a random scheduler's efficiency will reduce compared to that of the optimal scheduler.

Using our dataset, we randomly draw heterogeneous-ISA multicore systems. Specifically, we draw 2500 4-, 8- and 16-core systems for a total of 7500 systems. On each system, we apply our workload execution protocol and we use a random scheduler. We repeat the experiment 300 times to minimize noise from random decisions. Finally, we compare the average performance against that of the optimal scheduler. Figure 1 shows our results. Each system is characterized by its EoS rating (Y-axis) and its per-core performance in αIPC (X-axis).

Ease-of-Scheduling is an unbiased selection mechanism, since we cannot influence the ranking of systems. With benchmark systems ranging from the easiest 4-core to the most difficult 16-core system, we (1) efficiently cover a wide range of underlying systems, and (2) can demonstrate that our predictive schedulers adapt well to a variety of underlying systems.

Our results also verify our intuition: Easier systems tend to have more balanced cores in terms of performance, while more difficult systems typically have a small number of high-performance cores and a large number of low-performance cores. We omit presenting the microarchitectural configuration of each multicore benchmark system (total of 84 cores), due to space restrictions. However, we note that some of the microarchitectural traits of each system can be inferred from their EoS rating.

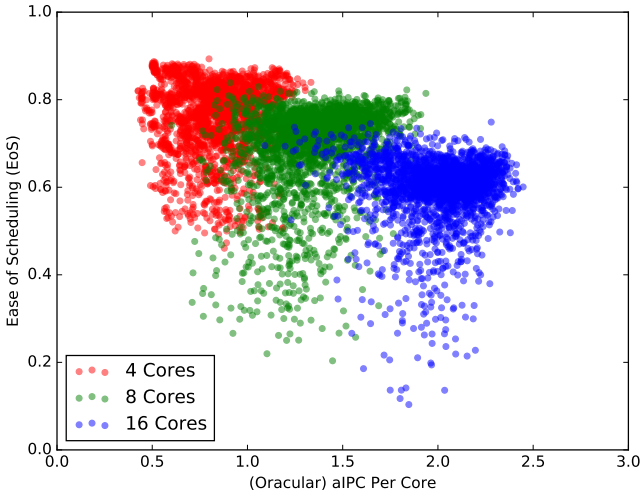

Fig. 1: Randomly-drawn heterogeneous-ISA systems rated in terms of scheduling difficulty. From each cluster, the highest (easiest), median, and lowest (hardest) points are chosen as benchmark systems.

6 Results

This section presents the results of our study. We first evaluate our ML-based performance predictors as standalone modules. We then examine schedulers that incorporate these schedulers.

6.1 Performance Predictors

ML Space Exploration: Training Accurate Performance Predictors. We present the results of our exploration over the three Machine Learning (ML) algorithms we use and over the design space generated by our configuration flags, as described in Section 5.1. After running all possible configurations, we identify the most accurate trained model from each algorithm.

Table 4 presents the prediction accuracy comparison between the three winners. Since we want our performance predictors to be as close to ground truth as possible, we chose to focus on two metrics – *Mean Absolute Error (MAE)*, the average distance of predictions from reality, measured in αIPC , and *Standard Deviation (STD)*, the distribution of prediction errors from each predictor. Using MAE and STD we are essentially describing a bell curve that characterizes prediction error for each predictor. Finally, we report accuracy on both the previously unseen test set, as well as the known training set. Since we intend to use these predictors within schedulers, it is safe to assume that a scheduler will occasionally be faced with a known workload, in which case the training set accuracy is relevant.

When dealing with unseen test set queries, we measure roughly equal accuracy across our winner models. When queried with known data, our linear (RR) model's accuracy remains around the same level as with its test set queries. On the other hand, tree-based predictors report very low MAE and STD for training set queries. Unlike linear models, tree-based predictors have more capacity to

Model	Test Set		Train Set	
	MAE	STD	MAE	STD
RR	0.23	0.3	0.21	0.26
DT	0.22	0.35	0.07	0.04
RF	0.19	0.31	0.03	0.04

Table 4: Accuracy comparison of three predictors (MAE measured in αIPC)

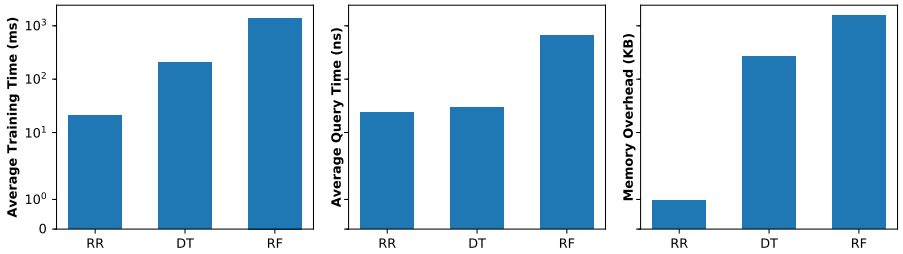

Fig. 2: Predictor algorithm overhead comparison: Training time (left), Query time (middle), Memory overhead (right).Y axes shared and in log scale.

characterize larger datasets and adapt to non-linear relationships. We later show how this improved accuracy leads to significantly more efficient scheduling.

Predictor Overhead Evaluation. This section presents an overhead comparison for our three models. We perform the following experiments on a system with an i7-3770K processor running at 2.9GHz and 16 GB of memory. We use the algorithm implementations from sklearn [9].

First, we vary the dataset size and train each model 100 times to report average training overhead. Training times for a dataset of 30k entries are 21ms, 203ms, and 1.4s for RR, DT, and RF respectively (Fig. 2 – left). Training overhead has little significance in choosing a model, since it happens infrequently. For extremely large datasets however, RF's overhead might become prohibitive.

We measure query overhead (Fig. 2 – middle), by asking each trained model for 55k predictions and report the average time per prediction. Overhead per query for RR, DT, and RF is 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.

Finally, we measure the memory overhead of each model (Fig. 2 – right). 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. The RR model needs 960 bits, the DT model 276kB, and the RF model 1.5MB. Overall, RF is expensive compared to the others. Although it does provide high accuracy, the predictions do not necessarily translate to a significant gain in scheduler efficiency, as observed later.

6.2 ML-based Schedulers

Performance Comparison of ML-based Schedulers. This section presents insights derived in the exploratory part of this work. Specifically, we link each of

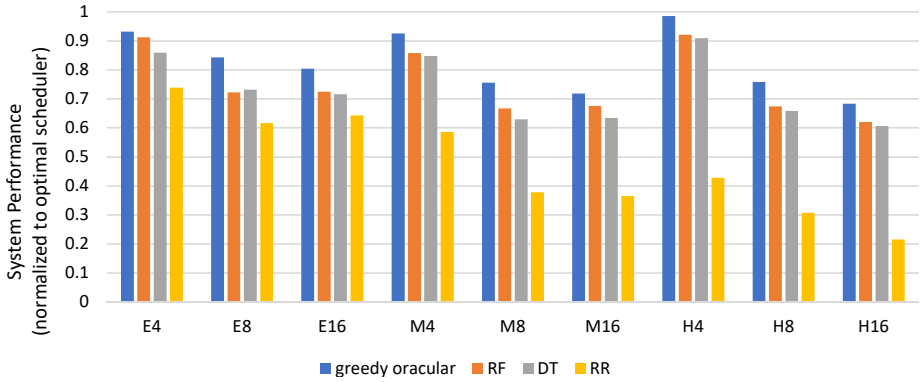

Fig. 3: Scheduler performance comparison on Heterogeneous-ISA systems.

our three predictive models to a scheduler and report its efficiency. Each scheduler receives an $N \times N$ prediction matrix from its predictor and then decides on the appropriate schedule such that it maximizes the system's overall performance (sum of αIPC from all N core-workload pairs in the chosen schedule). We compare our schedulers against a greedy (no knowledge of future workloads), oracular (zero prediction error) scheduler.

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 our decision can result in reduced predictor accuracy, 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 and 50% previously seen workloads, 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).

Figure 3 presents our results, normalized to the optimal schedule (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 easy 8-core (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 due to the reduced training set size. 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 from the reduced training dataset. Furthermore, tree-based prediction accuracy on the training set is excellent, which provides a significant advantage over RR in this experiment.

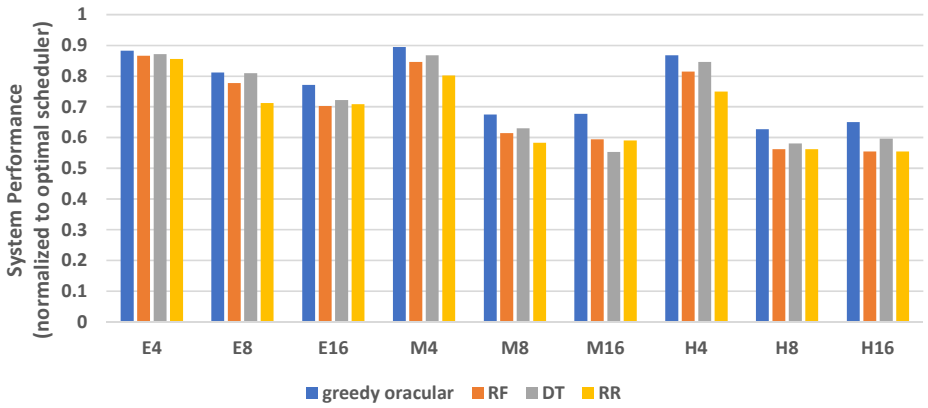

Fig. 4: Scheduler performance comparison on Single-ISA heterogeneous systems.

Performance Comparison on Single-ISA Heterogeneous Systems. To demonstrate the adaptability of our schedulers to less complex systems, we also examine single-ISA heterogeneous systems. We repeat the EoS-based system selection presented in Section 5.2. However, this time we enforce our systems to only use Alpha cores. We must note that our predictors have not been re-trained between the two experiments. Figure 4 presents our results.

We first observe that, unlike with heterogeneous-ISA systems, the DT scheduler has an advantage over RF. The added complexity of the RF predictor does not appear to be as beneficial in the single-ISA case. We can also observe that our linear predictor now reports comparable efficiency as RF and DT (1.5-4.8%). This happens due to the reduced scheduling difficulty on single-ISA systems (higher EoS values). Our best scheduler (DT) performs within 0.2-12.5% across all systems compared to the oracular scheduler (4% on average).

Scheduler Sensitivity to Underlying System. From Figure 3, we observe that our tree-based (DT, RF) schedulers are affected by the transition from 4 to 8 cores, but their performance remains almost intact when we move from 8 to 16 cores. In comparison, the oracular scheduler is affected by both transitions ($4 \rightarrow 8$, $4 \rightarrow 16$), albeit with a smaller impact. These trends are also observed when our schedulers accompany single-ISA systems (Figure 4). Our results show that tree-based schedulers were able to mostly overcome (1) the increase in scheduling difficulty between 8 and 16-core systems, and (2) the variety of underlying ISAs.

7 Conclusions

With the increased adoption of heterogeneity in modern systems, predictive models invariably enjoy attention from the scientific community for applications in scheduling, power management, resource management, and resource allocation. In this work, we first present an exhaustive exploration and analysis of the abilities of ML models to act as cross-core cross-platform performance predictors. We then use these predictors to implement schedulers that adapt well to the underlying architectural and microarchitectural heterogeneity without modification. Our best schedulers are capable of operating within 2-11% of the efficiency of an oracular scheduler.

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, NSF/Intel Foundational Microarchitecture Research Grant CCF-1823444, and a gift from Huawei.

References

1. Barbalace, A., Lyerly, R., Jelesnianski, C., Carno, A., Chuang, H.r., Ravindran, B.: Breaking the boundaries in heterogeneous-isa datacenters. In: ASPLOS (2017)
2. Barbalace, A., Sadini, M., Ansary, S., Jelesnianski, C., Ravichandran, A., Kendir, C., Murray, A., Ravindran, B.: Popcorn: Bridging the programmability gap in heterogeneous-isa platforms. In: ECCS. p. 29. ACM (2015)
3. Craeynest, K.V., Jaleel, A., Eeckhout, L., Narvaez, P., Emer, J.: Scheduling heterogeneous multi-cores through performance impact estimation (pie). In: ISCA. pp. 213–224 (June 2012). https://doi.org/10.1109/ISCA.2012.6237019
4. DeVuyst, M., Venkat, A., Tullsen, D.M.: Execution migration in a heterogeneous-isa chip multiprocessor. In: ASPLOS XVII (2012)
5. Greenhalgh, P.: Big. little processing with arm cortex-a15 & cortex-a7. ARM White paper pp. 1–8 (2011)
6. Kahle, J.: The cell processor architecture. In: MICRO. p. 3. IEEE Computer Society (2005)
7. Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P., Tullsen, D.M.: Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In: MICRO (2003)
8. Kumar, R., Tullsen, D.M., Ranganathan, P., Jouppi, N.P., Farkas, K.I.: Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In: ISCA (2004)
9. scikit learn: Scikit-learn python library. http://scikit-learn.org/stable/
10. Lim, K., Balkind, J., Wentzlaff, D.: Juxtapiton: Enabling heterogeneous-isa research with risc-v and sparc fpga soft-cores. arXiv preprint arXiv:1811.08091 (2018)
11. Mittal, S., Vetter, J.S.: A survey of cpu-gpu heterogeneous computing techniques. ACM Computing Surveys (CSUR) **47**(4), 69 (2015)
12. Somu Muthukaruppan, T., Pathania, A., Mitra, T.: Price theory based power management for heterogeneous multi-cores. In: ASPLOS '14. ACM (2014)
13. Torng, C., Wang, M., Batten, C.: Asymmetry-aware work-stealing runtimes. In: ISCA 2016 (2016). https://doi.org/10.1109/ISCA.2016.14
14. Variable, S.: A multi-core cpu architecture for low power and high performance. Whitepaper-http://www. nvidia. com (2011)
15. Venkat, A., Basavaraj, H., Tullsen, D.M.: Composite-isa cores: Enabling multi-isa heterogeneity using a single isa. In: HPCA 2019 (2019)
16. Venkat, A., Shamasunder, S., Shacham, H., Tullsen, D.M.: Hipstr: Heterogeneous-isa program state relocation. In: ASLPOS (2016)
17. Venkat, A., Tullsen, D.M.: Harnessing ISA diversity: Design of a heterogeneous-isa chip multiprocessor. In: ISCA (2014)
18. Zheng, X., John, L.K., Gerstlauer, A.: Accurate phase-level cross-platform power and performance estimation. In: DAC (2016)