Machine Learning-Based Approaches for Energy-Efficiency Prediction and Scheduling in Composite Cores Architectures

Hossein Sayadi, Nisarg Patel, Avesta Sasan, Houman Homayoun
Department of Electrical and Computer Engineering
George Mason University
Fairfax, VA, USA
{hsayadi, npatel33, asasan, hhomayou}@gmu.edu

***Abstract*—Heterogeneous architectures offer divers computing capabilities. Composite Cores Architecture (CCA) is a class of dynamic heterogeneous architectures that empowers the system to build the most appropriate core at run-time for each application by composing cores together to make larger core or decomposing a large core into multiple smaller cores. While CCA provides more flexibility for the running application to find the best run-time configurations to maximize energy-efficiency, due to the interdependence of various tuning parameters such as the core type, run-time voltage and frequency setting, and number of threads, it makes the scheduling more challenging. In this work, we investigate the scheduling challenges of multithreaded applications on CCA architectures. This paper describes a systematic approach to predict the right configurations for running multithreaded workloads on the composite cores architecture. It achieves this by developing a machine learning-based approach to predict core type, voltage and frequency to maximize the energy-efficiency. Our predictor learns offline from an extensive set of training multithreaded workloads. It is then applied to predict the optimal processor configuration at run-time by considering of the multithreaded application's characteristics and the optimization objective. For this purpose, five well-known machine learning models are implemented for energy-efficiency optimization and precisely compared in terms of accuracy and hardware overhead to guide the scheduling decisions in a CCA. The results show that while complex machine learning models such as MultiLayerPerceptron are achieving higher accuracy, after evaluating their implementation overheads, they perform worst in terms of power, accuracy/area and latency as compared to simpler but slightly less accurate regression-based and tree-based classifiers.**

Keywords-energy-efficiency; heterogeneous architecture; composite cores; scheduling; machine learning

I. INTRODUCTION

Heterogeneous multi-cores offer an effective solution to energy-efficient computing. To unlock the potential of the heterogeneity, software applications must adapt to the variety of different processors and make good use of the underlying hardware by executing workloads on the most appropriate core type. By running multithreaded applications on heterogeneous architectures, each thread is able to run on a core that matches its resource needs more closely than one-size-fits-all solution [1]. However, the effectiveness of heterogeneous architectures significantly depends on the scheduling policy and how efficiently we can allocate applications to the most appropriate processing core [3,4,9,27]. Applying ineffective scheduling decisions can lead to performance degradation and excess power consumption in such architecture [1,11,17,25]. Commercially available heterogeneous architectures include Intel Quick IA [6], ARM's big.LITTLE [8], and Nvidia Tegra 3 [7] that integrates a high performance big core with a low power little core on a single chip.

Composite cores architectures can provide further benefits by allowing the system to construct a right core for each running application. Several designs have been proposed that provide some level of dynamic heterogeneity. These proposals include Core Fusion [17], TFlex [18] and Composite Cores [1,11]. In [11,17] the concept of composite cores is proposed where a big core architecture can be dynamically decomposed into a smaller little-core architecture. The authors in [1] adapted the concept of composite cores in 3D by further enabling the core composition and decomposition at a low granularity of processor building blocks such as register file and load and store queue. Their proposed architecture allows multiple smaller cores to be composed together to build a larger core or vice versa, as needed. While composite cores architecture provides more opportunity to construct the right core for the running applications, it is making the scheduling a difficult problem. Previous studies have mainly examined the advantages of using single threaded applications in CCAs [1,2,13,17]. However, running multithreaded applications on a CCA and composing ideal processor architecture for energy-efficiency is a more challenging problem, considering the possible number of cores and threads and type of core micro-architecture. Furthermore, the challenge of how many and what type of core to compose for each multithreaded applications becomes even more complicated considering the impact of tuning parameters on energy-efficiency.

The main challenge for scheduling is to effectively tune system, architecture and application level parameters in CCA when running multithreaded applications. These parameters that are critical to performance and power include core type, voltage/frequency settings and the number of running threads. While there has been number of studies on mapping applications to heterogeneous architectures, no solution has been developed for mapping multithreaded applications into composite cores with its unique architecture. In addition, previous studies on mapping applications to multi-core architectures have focused primarily on 1) homogeneous architectures, 2) static heterogeneous architectures where the number and type of cores are fixed at design time, and 3) configuring individual or a subgroup of tuning parameters at a time, such as application's thread counts [5,9,21,22],

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.28

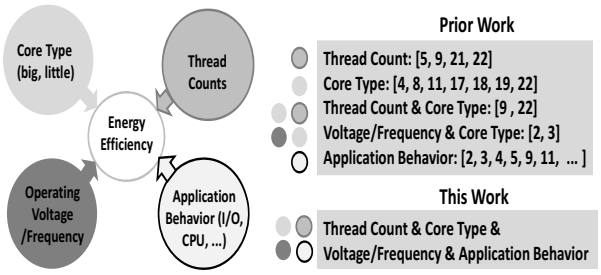


Fig.1. Tuning parameters influencing energy-efficiency in composite cores architecture. Prior work on scheduling using these tuning parameters.

voltage/frequency [2,3], or core type [4,8,11,17,18,19,22] and they have ignored the interplay among all these parameters. This study indicates that these parameters individually, while important, do not make a truly optimum configuration to achieve the best energy-efficiency on a CCA. The best configuration for a multithreaded application can be effectively found, only when these parameters are jointly optimized. Fig. 1 illustrates the tuning parameters influencing in scheduling decision in a CCA. Also, recent prior works as well as the contribution of our work is shown in this figure.

In this paper, through methodical investigation of power and performance, and comprehensive system and micro-architectural level analysis, we first characterize multithreaded applications on CCA to understand the power and performance trade-offs offered by various configuration parameters and to find how the interplay of these parameters affects the energy-efficiency. Our study is focused on a CCA where many little cores (base) can be configured into few big cores (composed) and vice versa. The experimental results support that there is no unique solution for the best configuration for different applications. Given the dispersed pattern of optimum configuration, we develop various machine learning models to predict the energy-efficiency of parallel regions, and guide scheduling and fine-tuning parameters to maximize the energy-efficiency. As behavior of applications changes at run-time, we applied our prediction and tuning method at a fine-grained level of individual parallel region within an application.

We use five well-known machine learning models to build predictors based on the knowledge extracted from an extensive set of hardware performance data which are a good representative of application behavior for each parallel region within the training phase. The models are then used at run-time to predict the optimal processor configuration for each parallel region of a multithreaded workload to maximize the energy-efficiency. We analyze the proposed machine learning-based models in terms of their prediction accuracy, power overhead and implementation cost to understand their cost effectiveness.

II. Motivation and Overview of Our Approach

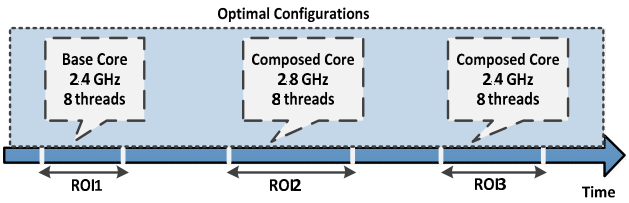


Fig. 2. Optimal configurations in different parallel regions of an application

Multithreaded applications are comprised of number of parallel regions which are separated by serial regions. In this work, we refer to these parallel regions as Region of Interest (ROI). In a homogenous multi-core architecture and for conventional scheduling, all of these regions are processed on the same core type, same voltage and frequency, and number of threads, though not all regions may have the same preferences. Some of these ROIs may benefit from different configurations than the others to obtain the maximized energy-efficiency. As mentioned earlier, the main challenge for scheduling is to effectively tune system, architecture and application level parameters in CCA for the entire application as well as the intermediate parallel regions to achieve maximum energy-efficiency. In this work, similar to [1,2], we are assuming that big cores (composed) are constructed by composing multiple little (base) cores.

Fig. 2 illustrates an overview of optimal configurations for an application selected from SPLASH-2 multithreaded benchmark suite with three parallel regions. In each ROI the best possible configuration for core type, operating frequency and thread counts that results in maximized energy-efficiency is specified. As can be seen, ROI1 needs to be run on the base core with 2.4GHz frequency and 8 threads, whereas for ROI2 in order to achieve the best energy-efficiency, we need to compose two small cores to make a big core. Moreover, the optimal operating frequency increases to 2.8GHz. The ability to accurately predict the optimal application, system and even microarchitecture parameters and adapt them to achieve the maximum energy-efficiency within different parallel regions of an application is the main motivation for this work. We develop several machine learning-based approaches which are able to predict the optimal setting of tuning parameters and change them to suit the specific requirement of each parallel region within the multithreaded application.

Fig. 3 depicts our three-stage approach for predicting the right core type and application configuration when running a multithreaded application on composite cores architecture. Our machine learning-based approach begins from extracting microarchitectural data (referred as feature extraction), from different parallel regions of application to characterize the multithreaded workload. These data (or features) include the hardware performance counter data, which are representative of application behavior at run-time. Next, a machine learning-based predictor (that is built off-line) takes in these features and predicts the best configuration settings for a given parallel region. For this purpose, we have implemented five well-known machine learning algorithms and compare them in terms of accuracy, power and area overhead to find to the most effective learning model which yields in optimized energy-efficiency. Finally, we configure the processor and schedule the application to run on the predicted configuration. In this work, the metric that we use to characterize energy-efficiency is the Energy Delay Product (EDP) which aims to balance performance and power consumption. We construct and

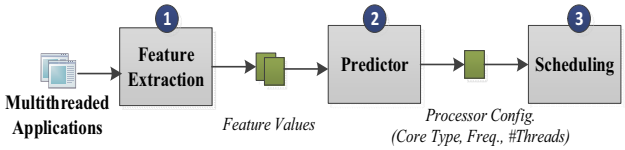


Fig. 3. An overview of our approach for predicting the optimal configuration and scheduling the multithreaded application

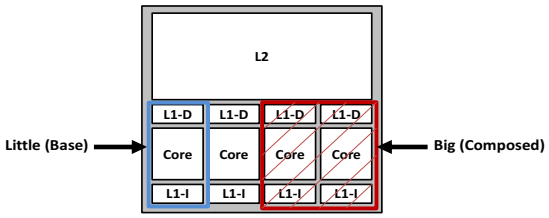


Fig. 4. Conceptual structure of a four-core composite architecture

compare five predictors using the machine learning algorithms described in the section V to guide the scheduling in a CCA.

III. Experimental Setup and Methodology

This section provides the details of our experimental setup. We use Sniper [13] version 6.1, a parallel, high speed and cycle-accurate x86 simulator for multi-core systems. McPAT [15] is integrated with Sniper and is used to obtain power consumption results. We study SPLASH-2 [14] and PARSEC [26] multithreaded benchmark suites for simulation. For architectural simulation, we modeled a heterogeneous composite cores architecture based on the recently proposed work in [17, 18]. Our study is focusing on a CCA where two little cores (base) can be configured into one big core (composed) and vice versa. Fig. 4 provides a conceptual overview of a four-core CCA. In this paper, we investigate two baseline heterogeneous CCAs which consist of multiple base and composed cores: 1) 8base/4comp, and 2) 4base/2comp. It is also important to note that for benchmark simulation we applied the binding (one-thread-per-core) model with #threads == #cores to maximize the performance of multithreaded applications [4, 5].

Table I shows the microarchitectural configuration of base (little) and composed (big) core of CCA in our experiment. We collect performance counters data on each architecture for characterization and drive the scheduling and mapping algorithms. We use these data to extract and evaluate the actual behavior of applications (I/O, CPU or memory intensive) for predicting energy-efficiency and assisting scheduling decision.

IV. Characterization Results

In this section, we evaluate the application performance and energy-efficiency sensitivity to the tuning parameters of operating frequency, number of running threads, and the choice of microarchitectures (base vs. composed) in heterogeneous composite cores architecture. Note that the entire set of benchmark analysis results is quite extensive. Therefore, due to space limitation we only present the results for a limited number of representative benchmarks shown in Fig. 5. This figure depicts the overall performance in terms of execution time (represented as a bar graph) and EDP results (represented as a line graph) for four different multithreaded benchmarks across different core types, frequencies and number of threads.

A. Frequency Sensitivity

For this analysis, all benchmarks were simulated using a baseline composed core running with only a single thread. The operating frequency is swept from 1.6 GHz to 2.8GHz with a step of 400MHz and the voltage is changed between 0.7, 0.8, 0.9, and 1V, respectively. As can be seen in Fig. 5, some benchmarks are very sensitive to changing the frequency. For instance, in *fmm* and *cholesky* reducing the frequency almost linearly reduces the overall performance. Overall, as expected, as the frequency increases, the performance increases accordingly. The EDP results show that a higher frequency

TABLE I. Architectural Specification

Microarch. Parameter	Base	Composed
Issue-Commit width	2	4
INT instruction queue	16 entries	32 entries
FP instruction queue	16 entries	32 entries
Reorder buffer	32 entries	64 entries
Branch penalty	7cyc	14cyc
iL1-dL1 Cache	16KB/4-way/2cyc	32KB/4-way/2cyc
L2 Cache	4MB/8-way/32cyc	4MB/8-way/32cyc

results in higher EDP. Increasing the number of threads interestingly reduces the sensitivity to frequency. In other words, increasing the number of running threads increases the performance gain due to parallelization. Consequently, the overall performance as the number of threads increases is more influenced by the speedup gain as a results of parallelism rather than operating at higher frequency. Moreover, the results show that the base core is more sensitive to frequency scaling than the composed cores. This is also an interesting observation as the composed core has a large pipeline, allowing it to tolerate performance cost due to alterations in access latency to cache subsystem as a result of frequency scaling. Note that changing clock frequency changes the number of cycles it takes for the processor to communicate with the cache.

B. Core Type Sensitivity

In this section, the results are reported for a baseline configuration with a core running a single thread at the highest frequency of 2.8 GHz and operating voltage of 1V. The changing parameter is the core type, which varies between a base (little) core and a composed (big) core architecture. Core type demonstrates constant behavior with regards to EDP. As shown in Fig. 5, there is a clear gap between the big composed and little base cores (in Thread1 and F2.8), with composed core having lower EDP. In these cases, the performance benefits of the composed core outweigh the energy savings of the base core.

C. Thread Count Sensitivity

Finally, each benchmark is simulated with varying the numbers of threads. In this step, each simulation was performed at the same frequency of 2.8 GHz and operating voltage of 1V, when changing the number of threads from 1 to 8. As shown in Fig. 5, increasing the thread counts leads to better performance at the cost of higher power. Moreover, there is a large gap between the EDP values of base and composed core in lower number of threads. In particular, we observe that by increasing the application thread counts, the corresponding gap between different core types diminishes and makes the base core competitive to the composed core in terms of EDP.

D. Joint Analysis (Core Type, Frequency, Thread Count)

To understand the interplay among various tuning parameters and find the optimum configuration for maximizing the energy-efficiency, in this section all permutations of the parameters were simulated. We test four voltage/frequency settings on two core types and execute each multithreaded benchmark with 1 to 4 or 8 threads (depending on the core type) using a native input set, where each thread is assigned to a single core. These results are illustrated in Fig 5. Due to space limitation, we only demonstrate the results for 1, 4 and 8 running threads. Furthermore, the best evaluated execution time and EDP for each application are shown in each figure.

Table II presents the optimal configurations for both examined CCAs, 8base/4comp and 4base/2comp architectures. This table includes benchmarks name, followed by the best core configuration parameters (Core, Freq., #Thread) in terms

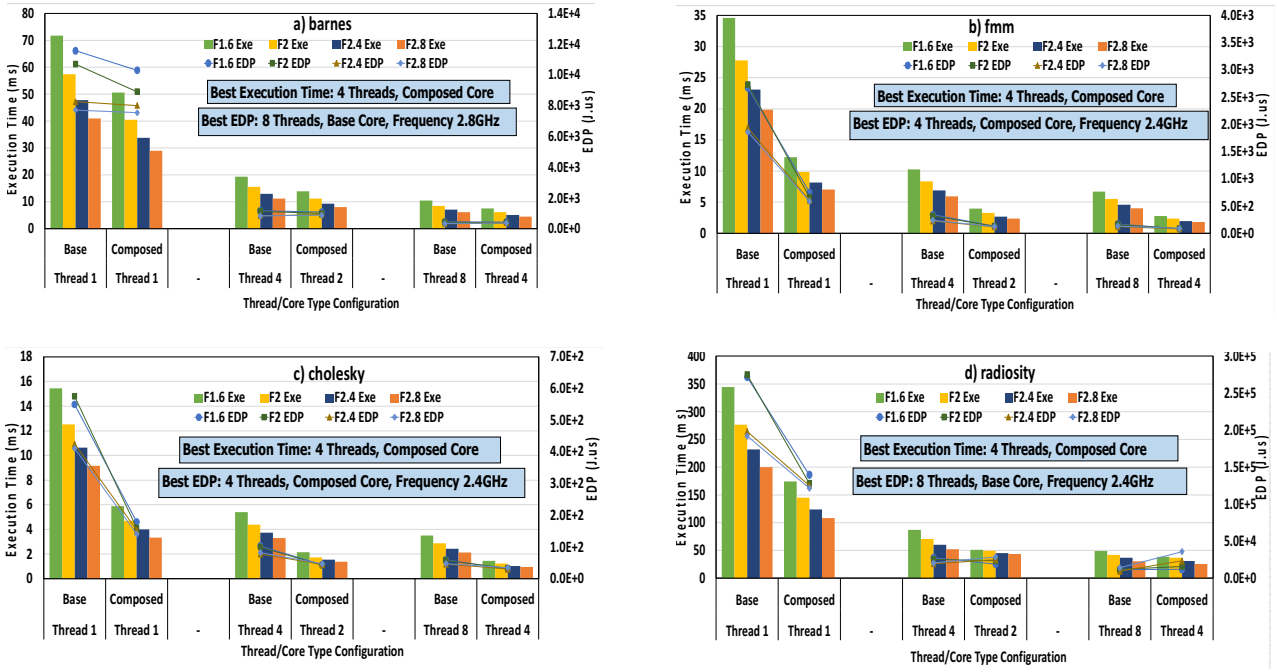


Fig. 5. Execution Time and EDP of a) barnes, b) fmm, c) cholesky, d) radiosity with various Core Types, Threads, Frequencies

of EDP across base and composed cores. We have also calculated the relative EDP variation for each benchmark, which indicates the relative difference between energy-efficiency for the best configuration parameters in base and composed cores. We quantify variation as (best_base – best_comp)/best_base. The variation parameter indicates whether it is justified to compose cores. For this purpose, a variation threshold is defined that decides what type of core architecture should be selected for executing the corresponding multithreaded application more energy-efficiently. This user-defined threshold can be adjusted based on the architecture and available resources as well as the cost of core composition. Note that composing base core to build big composed cores is not free and comes with power as well as core utilization overhead. The core utilization overhead is in fact due to using additional cores to build bigger cores. When cores are composed to build a bigger core, fewer cores will be available for incoming or co-scheduled applications. In this work, we assume a 20% variation threshold. As a result, if the EDP variation between best-base and best-composed architectures is found to be lower than 20%, we use the base core for scheduling instead of composing to avoid power as well as core utilization costs.

As can be seen from Table II, for most studied applications the best running thread counts remain unchanged across different core types. For instance, *barnes* performs with 2.4 GHz and 2.8 GHz on base and composed cores, respectively, while the best number of running threads on both architectures is 8. As shown, the variation has negative value for some cases, which indicates it is more energy as well as core-utilization efficient to run the application on little base core. Therefore, given that the variation value is lower than pre-defined threshold, rather than running the application on costly big composed core, we schedule the multithreaded application onto cost-effective base core. From these observations, we conclude that while we can obtain significant performance gains, power and core utilization costs could be drastic when running application on big composed core. As a result, in those cases we choose the little base core as the optimal core configuration.

E. Parallel Regions Analysis

As explained before, multithreaded applications are composed of a number of parallel sub-regions, which are

TABLE II. OPTIMAL CONFIGURATIONS WITH OPTIMIZATION TARGET OF EDP FOR DIFFERENT ARCHITECTURES

Benchmark	8Base/4Comp					4Base/2Comp				
	Best-base		Best-composed		Var.	Best-base		Best-composed		Var.
	Freq. (GHz)	*#Thread*	*Freq. (GHz)*	*#Thread*	(%)	*Freq. (GHz)*	*#Thread*	*Freq. (GHz)*	*#Thread*	(%)
barnes	2.4	8	2.8	4	-444.8	2.8	4	2.8	2	-475.4
fmm	2.4	8	2.4	4	2.2	2.4	4	2.8	2	-2.9
cholesky	2.4	8	2	4	28	2.4	4	2.8	2	5.8
radix	2.8	8	2.8	4	-138.7	2.8	4	2.8	2	-236.2
radiosity	2.4	8	1.6	4	-102.8	2.4	4	1.6	2	-128.1
raytrace	2.4	5	2	4	-28.9	2.4	4	2.8	2	-152
fft	2	4	2	2	36.3	2	4	2	2	36.3
lu.cont	2.4	8	2.8	4	27.2	2	4	2	2	7.8
blackscholes	2.8	4	2.4	4	83.85	2.8	4	2.4	2	77.08
bodytrack	2	3	2	3	41.23	2	3	2	2	34.1
ferret	2	6	2	4	62.4	2	4	2	2	54.3

TABLE III. OPTIMAL CONFIGURATION IN DIFFERENT PARALLEL REGIONS OF APPLICATIONS FOR EDP OPTIMIZATION

radix			
Region	Core Type	Freq.	#Threads
1	base	2.8	8
2	comp	2.4	8
3	base	2.8	8
4	comp	2.8	7
cholesky			
Region	Core Type	Freq.	#Threads
1	base	2.8	1
2	base	2.4	8
3	comp	2.8	8

fft			
Region	Core Type	Freq.	#Threads
1	base	2.8	8
2	comp	2.8	8
3	comp	2.8	1
4	comp	2.4	8
5	comp	2.8	8
lu.cont			
Region	Core Type	Freq.	#Threads
1	base	2.4	8
2	comp	2.8	8
3	comp	2.4	8

separated by serial regions. Considering the application behavior, not all ROIs may have the same performance and power requirements. To illustrate the improvements offered by composite cores architectures, studied multithreaded benchmarks were modified to monitor the behavior of each parallel region within the application. Simulation markers were placed at different sections of the benchmarks that would be simulated as individual parallel regions. All simulations were then run again using all permutations of the configuration parameters discussed earlier, including voltage/frequency, core type, and thread counts. Due to space limitation in our paper, we only present simulation results from parallel regions of four benchmarks which are reported in Table III. Each table shows the optimal configuration in terms of EDP for each of the ROIs listed for a given benchmark. It can be clearly seen that every ROI within the application does not benefit from the same configuration parameters.

In order to clarify this point, here we look at an example, the radix benchmark, in more depth. Radix benchmark was instrumented with four individual sub-regions. As results show, all four sub-regions have different set of configuration parameters to achieve the best EDP. The core type varies between base and composed, and the frequency varies between 2.8GHz and 2.4GHz. Also, the thread counts changes between 7 and 8. This example demonstrates the importance of using right tuning parameters for best EDP, not only for the entire multithreaded application, but also even for each parallel region within the application. Overall, the results show the importance of concurrent optimization at the application, system and microarchitecture levels at coarse-gained level of application or even fine-grained level of individual parallel regions within the application to maximize the energy-efficiency. The challenge is to develop a technique that automatically determines the best configuration for any given multithreaded applications and optimization goal and perform the tuning at a fine-grained level of individual parallel region within the application. In the next section, we will describe our proposed approach using various machine learning models.

The diversity of optimum configurations across various applications and their parallel ROIs demonstrates that when running a given multithreaded workload on a heterogeneous CCA, depending on the application and energy-efficiency optimization metric, different configuration parameters (Core Type, V/Freq., #Thread) lead to the best energy-efficiency. The configuration also changes at run-time for each parallel region within the application. In other words, experimental results support that there is no unique solution as the best configuration across various parallel regions of an application. This dispersed pattern of optimum results implies the necessity of developing a prediction method to guide scheduling decision of multithreaded applications onto heterogeneous composite cores architecture in order to improve the energy-efficiency.

V. PREDICTIVE MODELING

Recent studies have proposed linear regression modeling [20] to predict the power [10] and performance [3,11,19] of a processor at run-time**.** As mentioned earlier, in this work we implement different machine learning models to estimate the EDP. Table IV shows five machine learning models that we use for predicting the best processor and application configuration to deliver the lowest EDP. These models include least square median, linear regression, Multi-layer Perceptron (an artificial neural network model), and two decision tree techniques namely REPTree and M5Tree. We selected these five classifiers for two reasons. First, they are from three different types of machine learning methods; regression, neural network, and decision tree, covering a diverse range of learning algorithms which are inclusive to model both linear and non-linear problems. Second, the prediction model produced by these learning algorithms is deterministic which is compatible with our numerical target variable, EDP. All of ML classifiers were implemented using WEKA machine learning toolkit [24]. The inputs to our models are a set of features extracted from application profiling at run-time. While these microarchitectural features are accessible through our simulator infrastructure, in a real hardware, they are also accessible through hardware performance counters. These architectural information used for differentiating workloads behavior are listed in Table V. The output of our models is the optimal EDP that corresponds to the type of core to use for running the application, the clock frequencies of the core and thread counts. The goal is to use these models to find the best configuration parameters for individual parallel regions within an application and understand how these parameters should be adapted at run-time in a heterogeneous architecture. Developing and deploying these models include a 3-step process for supervised machine learning as follows: (i) generating training data, (ii) developing a predictive model, (iii) using the predictor at run-time for every parallel region of the application.

A. *Training the Predictor*

Fig. 6 depicts the process of training multithreaded

TABLE IV. MACHINE LEARNING CLASSIFIERS USED FOR PREDICTION

ML Classifier	Learning Type
LinearReg	Regression
LeastSqMed	Regression
MultiLayerPercep	Neural Network
M5Tree	Decision Tree
REPTree	Decision Tree

TABLE V. HARDWARE PERFORMANCE DATA USED FOR TRAINING THE CLASSIFIERS

Category	Hardware performance counter
Memory subsystem	L1 D-cache access, L1 D-cache miss, L1 I-cache access, L1I- cache miss, L2 cache access, L2 cache miss, I-TLB miss, D-TLB miss
Instructions	Integer instruction issue, Integer floating point issue
Branch	Branch instruction, Branch misprediction

applications to build a machine learning classifier for EDP prediction. Training involves finding the best processor and application configuration and extracting feature values for each training workload, reducing the extracted features to the most vital performance counters, and developing a learning model from the training data. It is important to note that the input variables in our classifiers are performance counters extracted from different parallel regions of application, and the output variable is the EDP for a given set of tuning parameters.

Generating Training Data. To derive the prediction model for energy-efficiency, we need to develop a data set to train the prediction model. We applied our ML classifiers on extensive set of SPLASH-2 and PARSEC multithreaded benchmark suits. The studied multithreaded applications represent diverse compute, memory and I/O intensity behavior. We exhaustively execute each training benchmark, with different processor and application configurations and record the configuration with lowest EDP. We have also collected the hardware performance data from 20 parallel sub-regions of these applications and use them to build regression, neural network and decision tree classifiers for predicting the EDP. As mentioned earlier, the extracted features from each of the parallel ROIs appropriately represent the application behavior during these execution phases. For each parallel region within an application, we collect twelve hardware performance counters data listed in Table V on all possible configurations of core types, voltage/frequency operating points and number of threads. We applied machine learning models on the studied benchmarks using these performance counters and profiling information for predicting energy-efficiency. In order to validate each of our classifiers, we applied the percentage split method to divide the dataset into two sets, using 70% (known applications) of the data to train the model and 30% (unknown applications) to simulate and test.

Developing Predictive Models. The features together with the processor configuration are supplied to each learning algorithm. The learning algorithm attempts to find a correlation from the feature values to the optimal configuration and predicts the energy-efficiency that corresponds to each configuration. As described in previous section, our predictors are based on a number of features extracted from the hardware performance counter attributes. One of the key aspects in building an accurate predictor is finding the right features to characterize the input data. We started from twelve performance counters that can be collected from parallel

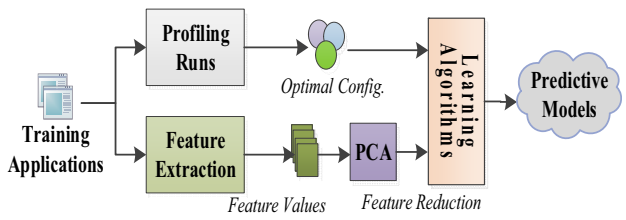


Fig. 6. Training process for machine learning predictive models

regions of each multithreaded application. These features listed in Table V include performance counters representing pipeline front-end, pipeline back-end, cache subsystem, and main memory behaviors and are influential in the performance of standard applications. As shown in Fig. 6, after feature extraction we use Principle Component Analysis (PCA) and correlation analysis on our training set to monitor the most vital micro-architecture parameters to capture application characteristics. By applying the attribute reduction method, we determine the four most related performance counters including L1 D-cache access, L2 cache-access, L2 cache-miss and branch misprediction. These performance counters are included in our model as input parameters.

B. Prediction Phase

Once we build our ML predictor, we deploy it for predicting the energy-efficiency of various configurations. Fig. 7 provides an overview of EDP prediction process and tuning processor and application parameters using the trained machine learning classifiers. This prediction model predicts continuous values representing EDP as a function of performance counter inputs and tuning parameters, which is then used to make the scheduling decisions at run-time. In particular, in this phase we run a multithreaded application with the most aggressive configuration setting where all tuning parameters are set at max (maximum thread counts, highest frequency, and for composed core). It is important to note that this would be the fastest way to collect run-time features of an application, since this is done for most aggressive configuration that corresponds to the highest performance. At run-time, we extract the hardware performance counters by profiling the application for each parallel region. The ML classifier then takes the key performance features and configuration settings as inputs, and outputs the system energy-efficiency for each configuration. The configuration corresponding to the lowest EDP is then used to tune and schedule the application. Thus, at run-time, given an unknown application, the predictor can predict the EDP of all possible configurations based on a single run data. The configuration corresponding to the lowest estimated EDP is then selected for the run. The predictive models by observing run-time behavior of a multithreaded application running with a specific configuration, predicts the right configuration parameters to achieve the maximum energy-efficiency. It is important to note that each predictor can be simply trained for other objectives such as ED^2P optimization.

C. Experimental Results

In this section, we present the evaluation results for our machine learning predictors. We compare these learning techniques in terms of EDP prediction accuracy and hardware implementation cost. As mentioned in previous section, in this work we focus on analyzing two heterogeneous CCA consisting of 8base/4comp, and 4base/2comp cores. In order to perform a comprehensive EDP characterization of studied architectures, we classified all possible configurations (core types and number of threads) into four classes. The first two are *Fully-Base* and *Fully-Composed* configurations are referred to cases in which the best energy-efficiency is achieved with full utilization of the base and composed cores, respectively. In other words, the optimum number of threads is equal to the maximum number of existing base/composed cores. On the other hand, we use *Partially-Base* and *Partially-Composed* configurations when the best number of threads is lower than maximum available cores.

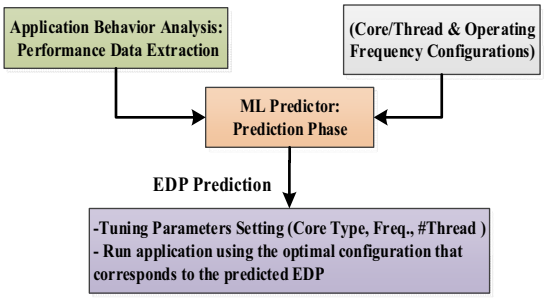
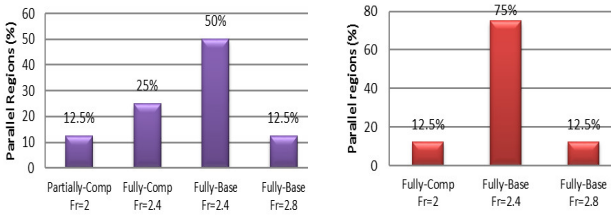


Fig. 7. Energy-efficiency prediction and tuning parameters configuration



a) 8Base/4Comp b) 4Base/2Comp

Fig. 8. The distribution of the optimal configurations for EDP

Optimal Configurations. Fig. 8 shows the distribution of the optimal configurations for two studied composite core architectures. It demonstrates how the distribution of optimal configurations changes across all studied parallel regions (for all studied applications). In both studied architectures, Fully-Base configuration running at a medium frequency of 2.4 GHz yields the lowest EDP for a majority of studied cases. However, composing cores yields the lowest EDP also for a noticeable number of cases; 37.5% in 8Base/4Comp and 12.5% in 4Base/2Comp. From Fig. 8(a), we observe that overall 62.5% of studied cases benefit from Fully-Base configuration which yields the lowest EDP. Moreover, as shown in Fig. 8(b), for all studied cases in 4Base/2Comp architecture, no Partially-Composed or Partially-Base configurations was selected as the optimum configuration; i.e. in this architecture for maximum energy-efficiency, all cores, either base or composed, need to be allocated to the running multithreaded application. Also, the optimal clock frequency found to be lower than the maximum frequency for the majority of studied cases in both architectures (more than 87.5%). This diagram shows the need to adapt the microarchitecture and application settings to different multithreaded applications for energy-efficiency optimization. Overall, the results confirm a large disparity in the optimum configuration across a large range of tuning parameters, highlighting the importance of developing a predictive method. Also, as the number of base cores in the studied architecture increases, the importance of composing cores to make a larger core is highlighted; more than 32% of studied cases in 8Base/4Comp vs. 12.5% in 4Base/2Comp are corresponding to composite cores.

Prediction Accuracy. To evaluate the accuracy of our prediction model, we calculate the value of relative mean absolute error defined as [(estimated value-actual value)/actual value]×100. This metric indicates the relative difference between the predicted and observed maximum EDP. Fig. 9 shows the energy-efficiency accuracy comparison of the machine learning classifiers used for predicting the EDP. As shown, M5Tree achieves close to 94.5% accuracy and outperforms all other classifiers in predicting the energy-efficiency. This tree-based classifier generates a decision list for regression problems using separate-and-conquer process which results in highest EDP accuracy. Next are Perceptron, LinearReg and LeastSqMed predictors, respectively. We implemented a multi-layer perceptron neural network with three layers which is capable of numerical predictions, since neurons are isolated and region approximations can be adjusted independently to each other. Finally, REPTree classifier shows the lowest accuracy as compared to other learning models. REPTree is another fast decision tree learning model, which builds a decision tree using information gain and variance. This model only sorts values for numeric attributes once and missing values are dealt with by splitting the corresponding instances into pieces which negatively impacts the accuracy of predictor in our EDP prediction problem as compared to other models.

Hardware Implementation. In this section, we discuss the hardware implementation of the machine learning classifiers. We use Vivado HLS compiler to develop the HDL implementation of the classifiers and estimate the area, latency and power overhead. When it comes to choosing machine learning classifiers for hardware implementation, accuracy of any algorithm is not the only parameter for decision-making. Area, power and latency overhead of ML classifiers are also key factors in selecting a cost-efficient machine learning classifier. While complex algorithms such as Neural Networks can deliver high accuracy, they will also add significant overhead in terms of hardware implementation. Also given their complexity, they might be slow in finding the right configuration for scheduling. We are interested in analyzing these overheads when implementing these machine-learning algorithms. A ML algorithm with high accuracy, low area, low power consumption, and low latency is the ideal choice for EDP prediction to guide the scheduling.

The latency, power and area results for implemented machine learning algorithms are shown in Table VI. As can be seen, the MultiLayerPerceptron algorithm results in significant area and latency overhead compare to other learning methods. REPtree decision tree is the fastest algorithm compared to others but comes with the lowest accuracy. M5Tree, another decision tree learning predictor, is the most accurate predictor however it comes with significant area overhead. Clearly the results show some trade-off between accuracy, latency, and area overhead. Therefore, it is important to compare classifiers by taking these parameters into account.

The metric of EDP accuracy over area is a fair ratio to compare the studied predictors. This metric essentially indicates which learning algorithm is the most accurate per unit of silicon area. We have shown results of Accuracy/Area in Fig. 10. As can be seen in this figure, REPTree, LinearReg and LeastSqMed classifiers are performing significantly better in terms of accuracy per area compared to highly accurate but

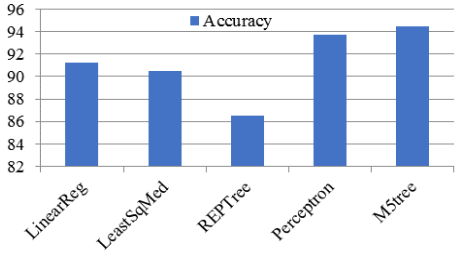


Fig. 9. EDP accuracy comparison of ML predictors

TABLE VI. HARDWARE SYNTHESIS COMPARISON OF ML PREDICTORS

ML Algorithm	Latency (cycles@10ns)	Power (W)	Area (LUTs+FFs+DSPs)
LinearReg	36	0.253	3071
LeastSqMed	46	0.267	3127
MultiLayerPercep.	116	0.52	20955
M5Tree	51	0.287	11120
REPTree	9	0.241	2532

complex MultiLayerPerceptron and M5Tree. In addition, if delay is a constraint, REPTree, LinearReg and LeastSqMed classifiers outperform the more complex MultiLayerPerceptron and M5Tree in terms of latency. This is mainly because REPTree model doesn't involve in complex floating-point operations unlike others and instead, it involves in various conditional evaluations. This helps REPTree to achieve lower power consumption as well. However, this is not the case for every tree-based classifier. M5tree which is also a tree-based classifier with higher power consumption has floating point operations. Out of all classifiers, MultiLayerPerceptron performs worst in terms of power consumption and latency mostly because of complex sigmoid function calculations. Comparing based on Accuracy/Area ratio, the results show that REPTree is outperforming all other learning algorithms.

VI. CONCLUSION

Scheduling multithreaded applications on composite cores architectures is a challenging problem, given various optimization parameters. In this paper, we respond to this challenge by developing a scheduling and tuning solution. The space for tuning configuration parameters in a composite core architecture is large, and our analysis indicates that there is no unique solution for the most energy-efficient configuration for different multithreaded applications, calling for developing a model to predict energy-efficiency for various tuning parameters. In response, we present a systematic approach for energy-efficiency prediction using various machine learning algorithms. We develop five machine learning-based models for estimating energy-efficiency of multithreaded applications in composite cores architecture. Our proposed ML-based approach, takes hardware performance counters information at run-time from a multithreaded application, and it predicts the most energy-efficient configurations based on run-time analysis and sets the number of threads and operating frequency. It also decides whether to compose little cores into big cores. The results show significant EDP prediction accuracy of as high as 94% across studied applications. We also compared these algorithms in terms of accuracy, latency, power and area overhead. Our results show that although using non-linear regression or neural network models such as M5Tree or MultiLayerPerceptron provides more accurate EDP prediction compared to simple regression or decision tree-based models, they significantly increase complexity of the design in terms of power and area overhead.

ACKNOWLEDGMENT

This research is based upon work partially supported by the National Science Foundation under Grant No.1527151.

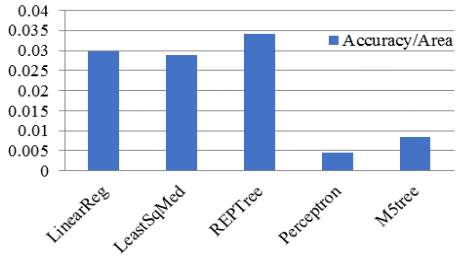


Fig. 10. Accuracy/Area ratio comparison between ML predictors

REFERENCES

[1] H. Homayoun et al., "Dynamically Heterogeneous Cores through 3D Resource Pooling", In *HPCA'12*, pp. 1-12, New Orleans, USA, Feb. 2012.

[2] V. Kontorinis et al., "Enabling Dynamic Heterogeneity Through Core-on-Core Stacking", In *DAC'14*, pp. 1-6, San Francisco, USA, June 2014.

[3] M. K. Tavana et al.,"ElasticCore: Enabling Dynamic Heterogeneity with Joint Core and Voltage/Frequency Scaling", In *DAC'15*, pp. 1-6, San Francisco, USA, June 2015.

[4] K. V. Craeynest et al.,"Scheduling Heterogeneous Multi-cores through Performance Impact Estimation (PIE)", In *ISCA'12*, pp. 213-224, Portland, USA, June 2012.

[5] K. K. Pusukuri et al.,"Thread Reinforcer: Dynamically Determining Number of Threads via OS Level Monitoring", In *IISWC'11*, pp. 116-125, Austin, USA, Nov. 2011.

[6] N. Chitlur et al., "QuickIA: Exploring Heterogeneous Architectures on Real Prototypes", In *HPCA'12*, pp. 1-8, February 2012.

[7] NVidia. The benefits of multiple CPU cores in mobile devices. http://www.nvidia.com/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf, 2010.

[8] P. Greenhalgh, "Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7: Improving Energy Efficiency in High-Performance Mobile Platforms", ARM, Tech. Rep. September 2011.

[9] G. Liu et al., "Dynamic Thread Mapping for High Performance, Power-Efficient Heterogeneous Many-core Systems", In *ICCD'13*, pp. 54-61, Asheville, USA, October 2013.

[10] J. Cong et al., "Energy-efficient Scheduling on Heterogeneous Multi-core Architectures", In *ISLPED'12*, pp. 345-350, 2012.

[11] A. Lukefahr et al., "Composite cores: Pushing heterogeneity into a core", In *MICRO-45*, pp. 317-328, 2012.

[12] B. Kuttanna, Technology insight: Intel silvermont microarchitecture.https://software.intel.com/sites/default/files/managed/bb/2c/02_Intel_Silvermont_Microarchitecture.pdf, May 2013.

[13] T. E. Carlson et al., "An Evaluation of High-Level Mechanistic Core Models," In *TACO*, vol. 11, New York, USA, October 2014.

[14] S. C. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations", In *ISCA'95*, pp. 24-36, New York, USA, May 1995.

[15] S. Li et al.,"McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", In *MICRO'09*, pp. 469-480, December 2009.

[16] Intel. Xeon processor 5500 series: An inteligent approach to IT challenges. 2009.

[17] E. Ipek et al.,"Core Fusion: Accommodating Software Diversity in Chip Multiprocessors," In *ISCA'12*, vol. 35, pp. 186-197, New York, USA, May 2007.

[18] C. Kim et al. "Composable Lightweight Processors", In *MICRO'07*, pp. 381-394, Chicago, USA, December 2007.

[19] B.C. Lee et al., "Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction", In *SIGPLAN Not*. 41, 11 pp. 185–194, Oct. 2006,

[20] R. Koenker, "Quantile Regression", No. 38, Cambridge university press.

[21] R. Kumar et al., "Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance" , In *ISCA'04*, June 2004

[22] M. Becchi et al., "Dynamic thread assignment on heterogeneous multiprocessor architectures", In *ACM CF*, pp. 29–40, 2006.

[23] G. Liu et al., "Procrustes: Power Constrained Performance Improvement Using Extended Maximize-Then-Swap Algorithm", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 34, No. 10, pp.1664–1676, October 2015.

[24] E. Frank et al., "The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[25] D.H. Albonesi et al., "Dynamically tuning processor resources with adaptive processing", In IEEE Computer, Special Issue on Power-Aware Computing. Vol. 36, No. 12, pp. 49–58, December 2003.

[26] C. Bienia et al., "Parsec 2.0: A New Benchmark Suite for Chip Multiprocessors", In *5th Annual Workshop on Modeling Benchmarking and Simulation*, June 2009.

[27] H. Homayoun, "Heterogeneous Chip Multiprocessor Architectures for Big Data Applications", In *ACM CF*, 2016.