



Article

Accurate Energy and Performance Prediction for Frequency-Scaled GPU Kernels

Kaijie Fan ^{1,*} , Biagio Cosenza ²  and Ben Juurlink ¹

¹ Faculty of Electrical Engineering and Computer Science, Technische Universität Berlin, Einsteinufer 17-6.0G, 10587 Berlin, Germany; b.juurlink@tu-berlin.de

² Department of Computer Science, University of Salerno, 84084 Fisciano (Salerno), Italy; bcosenza@unisa.it

* Correspondence: kaijie.fan@campus.tu-berlin.de

Received: 6 March 2020; Accepted: 21 April 2020; Published: 27 April 2020



Abstract: Energy optimization is an increasingly important aspect of today's high-performance computing applications. In particular, dynamic voltage and frequency scaling (DVFS) has become a widely adopted solution to balance performance and energy consumption, and hardware vendors provide management libraries that allow the programmer to change both memory and core frequencies manually to minimize energy consumption while maximizing performance. This article focuses on modeling the energy consumption and speedup of GPU applications while using different frequency configurations. The task is not straightforward, because of the large set of possible and uniformly distributed configurations and because of the multi-objective nature of the problem, which minimizes energy consumption and maximizes performance. This article proposes a machine learning-based method to predict the best core and memory frequency configurations on GPUs for an input OpenCL kernel. The method is based on two models for speedup and normalized energy predictions over the default frequency configuration. Those are later combined into a multi-objective approach that predicts a Pareto-set of frequency configurations. Results show that our approach is very accurate at predicting extrema and the Pareto set, and finds frequency configurations that dominate the default configuration in either energy or performance.

Keywords: frequency scaling; energy efficiency; GPU; modeling

1. Introduction

Power consumption is a major concern of modern computing platforms, from small-scale embedded systems to large-scale compute clusters. For instance, next-generation computing systems will need to perform 10^{18} (a billion billion) calculations per second on a power budget of 20 MW. Meeting this target will require major improvements in energy efficiency across the whole stack—starting at the hardware and memory subsystem and interconnecting up to the software level.

Dynamic voltage and frequency scaling (DVFS), where the voltage and frequency of the processor are scaled according to the requirements of the running application, is one of the most promising power management strategies. Thanks to different energy management libraries, such as *Running Average Power Limit* (RAPL) [1] on Intel processors and the *NVIDIA Management Library* (NVML) [2] on modern *graphics processing units* (GPUs), it is possible to both measure the energy consumption of a task (kernel) and dynamically change the frequency during program execution. However, while power management for CPUs has been widely researched over a number of years, GPUs are a relatively new area of study in this field. Hence, further investigation could deliver rich results for high performance with lower power consumption. By using NVML, a programmer is able to tune core and memory frequencies for a specific application, and different applications may show different energy consumption and performance depending on the selected frequency setting: for instance, a compute-bounded

application will benefit of memory frequency down-scaling with reduced energy consumption at the same performance. Nevertheless, it is not easy to manually perform such tuning because of the large space of available frequency configurations. For example, an NVIDIA GTX Titan X supports a total number of 219 possible configurations, spanning four memory frequencies and 85 core frequencies (note that not all memory-core combinations are supported; e.g., it is not possible to have both maximal core and memory frequency). Sampling such a large space is not viable option for many applications; thus, this work focuses on the design and implementation of a predictive approach, which aims at minimizing both the energy-per-task and the running time, e.g., by solving a multi-objective optimization problem.

Predicting the best frequency configurations is difficult. We have already seen that the large number of settings makes it impractical to perform an exhaustive search, and that the tuning space is multi-objective. An important aspect to consider is the fact that energy and performance with different frequency settings highly depend on the application. Figure 1 shows the Pareto sets (e.g., speedup on the x axis and and normalized energy consumption on the y axis) of three applications: k-NN, Kmeans and Blackscholes. Clearly, the distribution of the different memory settings highly depends on the kernel: memory-bounded applications are more sensitive to increases of memory frequency, while compute-bounded ones are mainly affected by core frequency increases. This aspect is discussed in detail in Section 5.2.

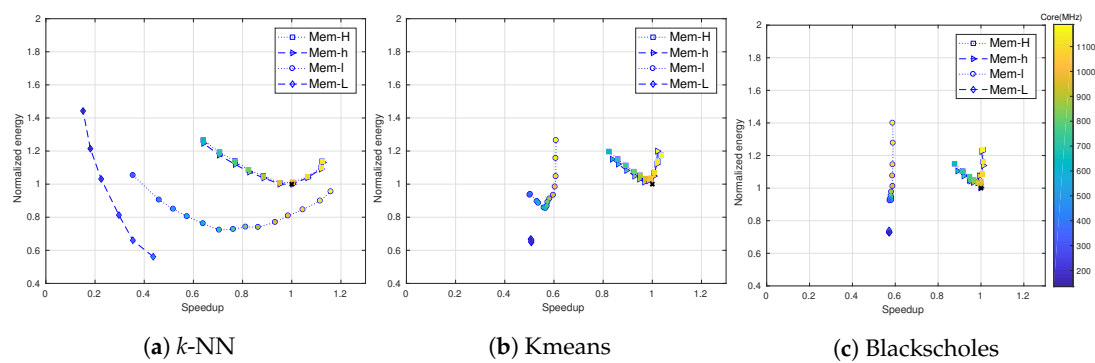


Figure 1. Speedup and normalized energy for three selected benchmarks and different frequency configurations. Toward the bottom (energy) and right-hand side (speedup) is best.

Moreover, this predictive modeling problem presents two additional challenges. First, the set of available configurations is not uniform. Figure 2 shows the available combination of memory and core frequencies on an NVIDIA Titan X. It is evident that some parts of the potential frequency domain are very dense, while in others, many configurations are missing, due to limitations on NVML (related to hardware limitations). The second problem is related to the input-size of the kernel: in a very small kernel, the energy and performance are slightly different than for a larger one. This aspect has never been investigated before; thus it is important to have a quantitative study of this aspect.

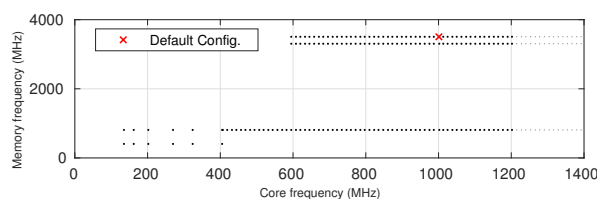


Figure 2. Supported combinations of memory and core frequencies.

Contributions This article expands on our previous work [3], whose contributions were:

- An analysis of the Pareto optimality (performance versus energy consumption) of GPU applications on a multi-domain frequency scaling setting on an NVIDIA Titan X.

- A modeling approach based on static code features that predicts core and memory frequency configurations, which are Pareto-optimal with respect to energy and performance. The model was built on 106 synthetic micro-benchmarks. It predicts normalized energy and speedup with specific methods, and then derives a Pareto set of configurations out of the individual models.
- An experimental evaluation of the proposed models on an NVIDIA Titan X, and a comparison against the default static settings.

In addition, we expand prior work with the following new contributions:

- A novel methodology for handling an imbalanced dataset based on the SMOTE algorithm, which further improves the modeling for critical cases; e.g., low memory frequency configurations.
- An analysis of the impacts of various input sizes on energy and performance.
- An extended analysis of the energy and performance application characterization on a set of twelve applications.

The rest of this article is organized as follows. The related work is introduced in Section 2. Section 3 provides an overview of the modeling approach, discussing features, training data and the machine learning methodology for performance and energy. Section 4 focuses on the handling of an imbalanced dataset. An extensive experimental part is presented in Section 5, which includes oversampling evaluation, application characterization, input size analysis and final evaluation of the predicted values. The article ends with conclusions in Section 6.

2. Related Work

Energy-performance modeling has received great attention from the research community. Mei et al. [4], in particular, wrote a survey and measurement study of GPU DVFS. Calore et al. [5] evaluated the use of DVFS on two high-end processors in the HPC system and estimated the benefits obtainable tuning CPUs and GPUs clock frequencies. Ge et al. [6] applied fine-grained GPU core frequency and coarse-grained GPU memory frequency on a Kepler K20c GPU. Tiwari et al. [7] proposed DVFS strategy be used both intra-node and inter-node to reduce power consumption by the CPU. Vysocky et al. [8] introduced tools, and applied them to CPU to save energy by tuning hardware parameters during the runtime.

Much work focuses on modeling one single objective, either energy or performance. In terms of energy efficiency, a number of optimization techniques have been recently proposed [9–13]. Among them, Hamano et al. [9] proposed a task scheduling scheme that optimizes the overall energy consumption of the system. Lopes et al. [10] proposed a model that relies on extensive GPU micro-benchmarking using a cache-aware roofline model. Song et al. [12] proposed *Throttle CTA Scheduling* (TCS), which reduces the number of active cores to improve energy-efficiency for memory-bound workloads.

In the domain of performance, many evaluation methodologies based on different architectures have been proposed [14–17]. Approaches [18,19] to predict performance by taking DVFS into consideration have been discussed. Kofler et al. [20] and Ge et al. [21] proposed a machine learning approach based on *artificial neural networks* (ANN) that automatically performs heterogeneous task partitioning. Bhattacharyya et al. [22] improved performance by combining static and dynamic analyses. Mesmay et al. [23] converted online adaptive libraries into offline by automatically generating heuristics. ϵ -PAL [24] proposed a machine learning iterative adaptive approach that samples the design space to predict a set of Pareto-optimal solutions with a granularity regulated by a parameter ϵ .

Here we discuss most important related work and Table 1 shows the comparison. Grewe et al. [25] used machine learning for a purely static approach, which, however, only predicted task partitioning. Steen et al. [26] presented a micro-architecture independent profiler based on a mechanistic performance model for a CPU. However, they did not take frequency scale into consideration, which, as already described in Section 1, plays a heavy role on energy and performance behaving.

Abe et al. [27], Guerreiro et al. [28,29] and Wu et al. [30] proposed performance and energy models by taking frequency scaling into consideration. Among them, Abe et al. [27] estimated the models by using performance counters but did not consider the non-linear scaling effects of the voltage. Guerreiro et al. [29] made more improvements: they not only presented the approach of gathering performance events by micro-benchmarks in detail, but also predicted how the GPU voltage scales. Wu et al. [30] studied the performance and energy models of an AMD GPU by using K-means clustering.

Table 1. Comparison against the state-of-the-art.

Paper	Static	Pareto-Optimal	Frequency Scaling	Machine Learning
Grewe et al. [25]	✓	×	×	✓
Steen et al. [26]	×	✓	×	×
Abe et al. [27]	×	×	✓	×
Guerreiro et al. [29]	×	×	✓	✓
Wu et al. [30]	×	×	✓	✓
Our work	✓	✓	✓	✓

Nevertheless, all of these approaches gathered the hardware performance counters (features) while running a kernel. In contrast, our work focuses on features that can be extracted statically, which can be used to estimate the speedup and normalized energy consumption models of a new kernel without running it. Furthermore, we figure out the Pareto-optimal solutions of memory-core frequency configurations of the new kernel. To the best of our knowledge, our work is the first to predict Pareto-optimal frequency configurations on a GPU using static models.

3. Background

Our approach to model the energy consumption and speedup of a input kernel is based on machine learning methodology, applied to a feature describing the kernel code and the frequency setting. This Section provides an overview of the method with a description of the different phases (training with oversampling and prediction), followed by a description of the feature representation; the synthetic training data; the predictive modeling approach for speedup and energy; and the final step to derive the predicted Pareto set.

3.1. Overview

The methodology proposed by our work is based on typical two-phase modeling with supervised learning: in a first training phase the model is built; later, when a new input code is provided, a prediction phase infers the *best* configurations. Figures 3 and 4 illustrate the workflow of this work, respectively for the training and prediction phases.

The goal of the training phase is to build two separate models for speedup and normalized energy. To do that, a set of OpenCL micro-benchmarks are provided for training (1). For each code in the micro-benchmark, a set of static features is extracted (2) and stored in a static feature dataset. Successively, each micro-benchmark is executed with various frequency configurations (3). The obtained energy measurements, together with normalized frequency configurations and the static features, are oversampled and used to train the normalized energy model (5). The same technology is applied on the speedup model training (6).

In the prediction phase, a new OpenCL code is provided as input to the framework. First, its static code features are extracted (1). The static features (2) and the frequency configurations (3) are combined together to form a set of feature vectors, each corresponding to a specific frequency setting. For each configuration, the previously trained models ((4) and (6)) are used to predict its normalized energy consumption (5) and speedup (7). Once the predictions for all memory configurations are available (8),

the dominant points are calculated and returned as predicted a Pareto set (9). Note that the feature oversampling technique does not affect the prediction phase, but is only applied to features in the training data.

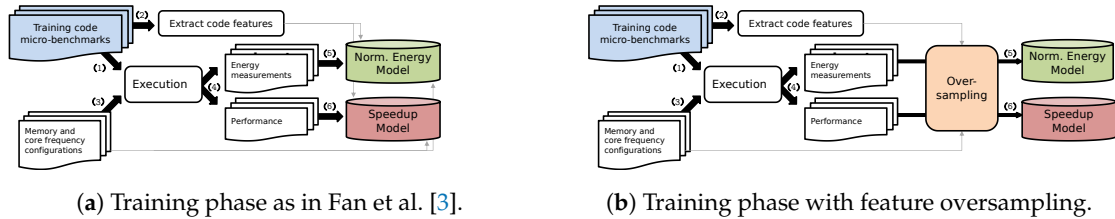


Figure 3. Training phase.

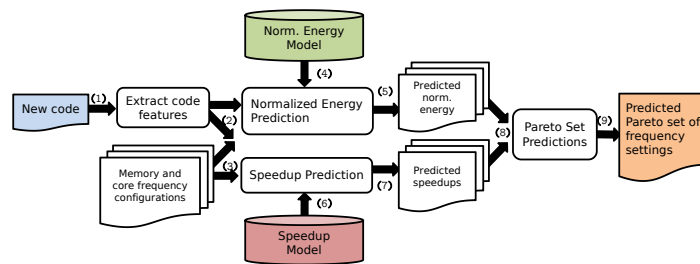


Figure 4. Prediction phase.

3.2. Features

To build an accurate predictive model, we define a set of numerical code features extracted from OpenCL kernels, which are then conveniently encoded into a feature vector. The feature representation used by our work is inspired by Guerreiro et al. [29], where features are designed to reflect the modular design and structure of the GPU’s architecture, which allow them to easily decompose the power consumption in multiple architectural components [31]. These ten features represent the number of integer and floating point operations; memory access on either global or local memory; and special functions, such as trigonometric ones.

Formally, a code is represented by the static feature vector

$$\vec{k} = (k_{int_add}, k_{int_mul}, k_{int_div}, k_{int_bw}, k_{float_add}, k_{float_mul}, k_{float_div}, k_{sf}, k_{gl_access}, k_{loc_access})$$

where each component represents a specific instruction type; e.g., integer bitwise (k_{int_bw}) or special functions (k_{sf}) instructions, or memory access to either global (k_{gl_access}) and local (k_{loc_access}) memory.

Frequency configurations are also represented as features: the vector $\vec{f} = (f_{core}, f_{mem})$, where f_{core} is the core frequency and f_{mem} is the memory frequency. The frequency values, which lie in the intervals [135, 1189] (core, NVIDIA GTX Titan X) and [405, 3505] (memory, NVIDIA GTX Titan X), are both linearly mapped into the interval [0, 1].

The vector $\vec{w} = (\vec{k}, \vec{f})$ represents the features associated with the execution of a kernel \vec{k} and frequency setting \vec{f} . Our final goal is to predict, for an input kernel \vec{k} , a subset of frequency configurations that is Pareto-optimal.

Instead of encoding the total number of instructions of a given type, each feature component is normalized over the total number of instructions. Such a normalization step allows us to have all features mapped in the same range, so that each feature contributes approximately proportionately to the model, and as a result, codes with the same arithmetic intensity, but different number of total instructions will have the same feature representation.

With respect to related work [20,25] in which features were extracted from the AST (abstract syntax tree), we prefer to use an LLVM IR (a low-level intermediate representation used by the LLVM

compiler framework) approach rather than the AST because it is more portable, as it can be easily adapted to other compilation infrastructure, e.g., that based on SPIR-V (Vulkan, OpenCL), and because it is a more accurate representation, as LLVM IR may have undergone to several optimizations which are not captured in the AST.

Being based on a static representation, our modeling approach does not consider the input size of a kernel. However, we have experimentally evaluated this aspect and found that the energy and performance behaviors are similar for any input size larger than 262,144. Details can be found in Section 5.3.

3.3. Training Data

Instead of using as training data the existing test benchmarks, we used a different and separate set of synthetic training codes specifically designed for the purpose. In related work, synthetic test benchmarks have been proposed for generic OpenCL code, e.g., using deep learning-based code synthesis [32], or in domain-specific context, such as stencil computations [33].

Our approach is a combination of pattern-based and domain-specific synthetic code generation, and was carefully designed around the feature representation [29]. Code benchmarks are generated by pattern: each pattern covers a specific feature, and generates a number codes with different instruction intensities (as a consequence, each pattern is designed to stress a particular component of the GPUs). For instance, the pattern `b-int-add` includes nine codes with a variable number of integer addition instructions, from 2^0 to 2^8 . This training code design enables a good coverage of (the static part of) the feature space. Additionally, a set of training benchmarks corresponding to a mix of all used features is also taken into account. Overall, we generated 106 micro-benchmarks.

The training data are represented by each code executed with a given frequency setting. Each code has been executed with a subset of 40 carefully sampled frequency settings, leading to a training size of 4240 samples. It is important to remark that, for a given micro-benchmark, it takes 20 min to test 40 frequency settings and 70 min to test all the 177 frequency settings, thereby making difficult the exhaustive search of all configurations.

To improve the accuracy of the model on a very specific memory configuration, we introduced an oversampling technique that adds new points to the training data. This approach is discussed in detail in Section 4.

3.4. Predictive Modeling

The final goal of this work is to predict which GPUs frequency configurations are *best* suited for an input OpenCL kernel. A frequency setting is a combination of a core frequency and memory frequency. For each setting, we are interested in both execution time (in ms) and energy consumption (in Joules). In this multi-objective context, there is no single *best* configuration, but a set of Pareto-optimal values, each exposing a different trade-off between energy and performance. This Section explains how our work is capable of predicting a Pareto set of frequency settings for an input OpenCL code.

Our approach is based on three key aspects. First, our predictive model uses machine learning: it is built during a training phase, and later reused on a new code for inference. Second, the multi-objective model is split into two single-objective problems, which are addressed with two more specific methods. Third, a final step derives a set of (multi-objective) configurations out of the two (single-objective) models.

Due to the different behaviors of speedup and normalized energy, we tested different kinds of regression models, including OLS (ordinary least squares linear regression), LASSO (least absolute shrinkage and selection operator) and SVR (support vector regression) for speedup modeling, and polynomial regression and SVR for normalized energy modeling. Because of the more accurate results, in this section we only report about SVR with different kernels.

In general, given a training dataset $(\vec{w}_1, y_1), \dots, (\vec{w}_n, y_n)$, where \vec{w}_i is a feature vector and y_i the observed value (e.g., either speedup or energy), the SVR model is represented by the following function:

$$f(\vec{w}) = \sum_{i=1}^n (\alpha_i - \alpha_i^*) K(\vec{w}, \vec{w}_i) + b \quad (1)$$

where b refers to the bias, and α_i and α_i^* are Lagrange multipliers that are obtained by solving the associated dual optimization problem [34]. $K(\vec{w}_i, \vec{w}_j)$ is the kernel function, which will be specified later in this section.

3.4.1. Speedup Prediction

The first model focuses on the performance of the code with different frequency settings. To have a more accurate predictive model, we focus on modeling normalized performance values; i.e., speedup over a baseline configuration using a default memory setting, instead of raw performance.

We analyzed a large set of codes, including the twelve test benchmarks and the 106 micro-kernels used for training. Based on the analysis insights, while keeping constant input code and memory frequency, we sought to increase the speedup linearly with the core frequency. For this reason, we used SVR with linear kernel for speedup prediction.

Formally, given a set of n kernel executions in the training set T , we define a training sample of input–output pairs $(\vec{w}_1, s_1), \dots, (\vec{w}_n, s_n)$, where $\vec{w}_i \in T$, and each kernel execution of \vec{w}_i is associated to its measured speedup s_i . Therefore, the kernel function in (1) is defined as $K(\vec{w}_i, \vec{w}_j) = \vec{w}_i \cdot \vec{w}_j$. Additionally, the C and ϵ parameters [34] are set to 1000 and 0.1.

After the training, coefficients α_i, α_i^* and b represent the model, which is later used to predict the speedup of a new kernel execution \vec{w} comprised of a new input code \vec{k} and a frequency setting \vec{f} .

3.4.2. Normalized Energy Model

A second model is used for energy prediction. As we did for performance, we focus on predicting per-kernel normalized energy values instead of directly modeling energy or power.

We observed how normalized energy behaves on a large number of codes. However, in this case the relation is not linear: while keeping constant both input code and memory frequency, normalized energy shows a parabolic behavior with increasing core frequency. After this point, the increase on core frequency does not compensate for the increase on power, leading to an overall decrease of energy per task. Because of that, we modeled the normalized energy with a non-linear regression approach; after testing different ones, we selected *radial basis function* (RBF) for the kernel.

Formally, given a set of n kernel executions in the training set T , we define a training sample of input–output pairs $(\vec{w}_1, e_1), \dots, (\vec{w}_n, e_n)$, where $\vec{w}_i \in T$, and each input \vec{w}_i is associated to its normalized energy value e_i . Therefore, the kernel function in (1) is defined as $K(\vec{w}_i, \vec{w}_j) = \exp(-\gamma \|\vec{w}_i - \vec{w}_j\|^2)$ with parameters $\gamma = 0.1$, $C = 1000$ and $\epsilon = 0.1$.

After the training, the model is represented by the coefficients α_i, α_i^* and b , which are later used to predict the normalized energy of a new kernel execution \vec{w} .

3.4.3. Deriving the Pareto Set

The final calculation of the Pareto-optimal solution is a straightforward application of multi-objective theory. We briefly recall here the most important concepts.

The general idea of Pareto dominance implies that one point dominates another point if it is better toward one objective and in the others is not worse. In our bi-objective problem, we have two goals, speedup and normalized energy, which need to be maximized and minimized, respectively. Given two kernel executions \vec{w}_i and \vec{w}_j , corresponding to (s_i, e_i) and (s_j, e_j) , \vec{w}_i dominates \vec{w}_j (denoted by $\vec{w}_i \prec \vec{w}_j$) if we have one of the following cases: (1) $s_i \geq s_j$ and $e_i < e_j$; or (2) $s_i > s_j$ and $e_i \leq e_j$.

A kernel execution \bar{w}^* is Pareto-optimal if there is no other kernel execution \bar{w}' such that $\bar{w}' \prec \bar{w}^*$. A Pareto-optimal set P^* is the set of Pareto-optimal kernel execution. A Pareto-optimal front is the set of points that constitutes the surface of the space dominated by Pareto-optimal set P^* .

Once we have the two predictions for each point (i.e., kernel execution) of the space, we can easily derive the Pareto set P' by using Algorithm 1.

In our case, this simple algorithm is enough to process all the kernel executions associated with a new input kernel. However, faster algorithms with lower asymptotic complexity are available [35].

Algorithm 1 Simple Pareto set calculation.

```

1: Predictions  $\leftarrow \{(s_1, e_1), \dots, (s_m, e_m)\}$ 
2:  $P' \leftarrow \emptyset$  ▷ Output Pareto set
3: Dominated  $\leftarrow \emptyset$  ▷ Set of dominated points
4: while Predictions  $\neq \emptyset$  do
5:   candidate  $\leftarrow$  Predictions.pop()
6:   for point  $\in$  Predictions do
7:     if candidate  $\prec$  point then
8:       Predictions  $\leftarrow$  Predictions  $\setminus$  {candidate}
9:       Dominated  $\leftarrow$  Dominated  $\cup$  {candidate}
10:    end if
11:    if point  $\prec$  candidate then
12:      Dominated  $\leftarrow$  Dominated  $\cup$  {point}
13:    else
14:       $P' \leftarrow P' \cup$  {candidate} ▷ We have found a point in the frontier
15:    end if
16:  end for
17: end while

```

4. Modeling Imbalanced Dataset

The problem of modeling performance and energy with different frequency configurations has an additional problem on NVIDIA Titan X GPUs: the set of available frequency configurations provided by the NVML tool is not evenly distributed, and therefore, is highly imbalanced. Imbalanced data is a problem common to many machine learning algorithms, and different solutions have been studied in the literature. In our predictive approach, the imbalanced distribution of the training data affects, particularly, some low-memory frequency configurations. In this section, we introduce how the frequency domain is distributed, and discuss how to solve the imbalanced data problem.

4.1. Frequency Domain and Test Setting

Our work is based on the ability of setting up memory and core frequencies, and on getting an accurate measurement of the energy consumption of a task execution. For the experimental evaluation of our approach, we relied on the capabilities provided by the NVML [2] library. It supports a number of functions to check which frequencies are supported (`nvmlDeviceGetSupportedMemoryClocks()`), to set the core and memory frequency (`nvmlDeviceSetApplicationsClocks()`), and to get the power consumption of the GPUs (`nvmlDeviceGetPowerUsage()`).

It is important to remark that different NVIDIA GPUs may have very different tunable configurations. For example, the NVIDIA Titan X provides four tunable memory frequencies (labeled for simplicity mem-L, l, h and H to represent 405, 810, 3304 and 3505 MHz) while the NVIDIA Tesla P100 only supports one. In addition, we experimentally noticed that some of the configurations marked as supported by NVML are not available, because the setting function does not actually change the frequencies. Figure 5 shows those frequency configurations on an NVIDIA Titan X (Figure 5a) and a Tesla P100 (Figure 5b). The black points represent the actual *available* memory-core configurations.

On Titan X, while setting to a core frequency higher than 1202 MHz for mem-L, h, H, the core frequency is actually set to 1202 MHz. The gray points indicate those configurations indicated as supported by NVML but that actually correspond to the core frequency of 1202 MHz.

As our goal was to statically model how core and memory frequency behave with different applications, we disabled any dynamic frequency feature (auto-boost): all experiments have been performed at a manually-defined memory setting. The red cross represents the default frequency configuration while not using dynamic scaling.

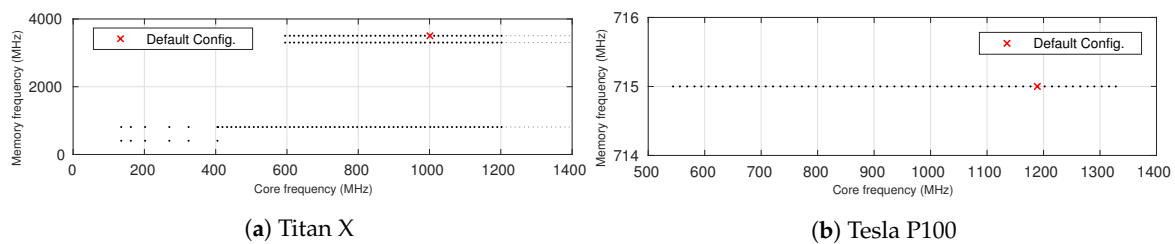


Figure 5. Supported combinations of memory and core frequencies as in Fan et al. [3].

An important issue of modeling these frequency configurations is that they are not evenly spread over the frequency domain; instead, different core frequencies are available for each memory frequency. In particular, the lowest memory configuration (mem-L) only supports six core frequencies, while mem-L has 71, and both mem-h and mem-H have 50.

Because of the larger space of possible memory configurations, our work is more interesting on the Titan X. The methodology introduced by this work is portable, and all tests presented in this work have been performed on both NVIDIA GTX Titan X and NVIDIA Tesla P100. However, we mainly focus on the most interesting Titan X scenario and all graphics refer to such architecture unless explicitly mentioned.

The main target architecture is equipped with the Titan X GPUs based on Maxwell architecture, supporting Compute capability 5.2, with default frequencies of 3505 MHz (memory) and 1001 MHz (core), OpenCL version 1.2 and driver version 352.63. The OS was Linux CentOS 14.

The per-kernel energy consumption is computed out of the power measurements; e.g., the average of sampled power values times the execution time. NVML provides power measurements at a frequency of 62.5 Hz, which may affect the accuracy of our power measurements if a benchmark runs for too short a time. Therefore, the applications have been executed multiple times, to make sure that the execution time was long enough to reach a statistically consistent power value.

4.2. Imbalance of Available Frequency Configurations

As discussed in Section 4.1, there are a total of 177 supported frequency configurations on NVIDIA GTX Titan X. However, it is clear the number of supported core frequencies on each memory frequency is not same (as shown in Figure 5a); the second lowest memory (mem-L) has 71 core frequencies supported, while the lowest memory (mem-L) only supports six.

To make the samples more balanced, and also to make less difficult the exhaustive search of all supported configurations, we carefully under-sampled core frequencies on the mem-H, mem-h and mem-L. Over all, we used 40 frequency configurations to train and make prediction in Fan [3]. As shown in Figure 6a, the red cross represents the sampled configurations used in Fan [3]. Since the mem-L only has six core frequencies available, we chose all the six points rather than under-sampling. However, the imbalanced data problem still exists.

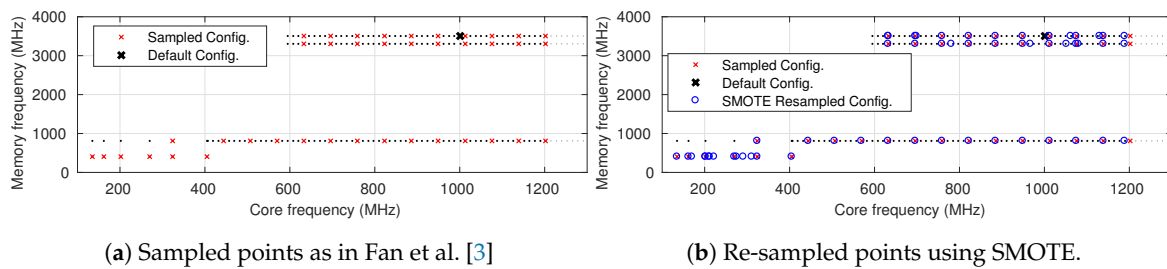


Figure 6. Comparison of sampled frequency configurations.

One way to handle the problem is to generate new samples in the minority class (mem-L), which is named oversampling and used widely in the machine learning to compensate for an imbalanced dataset. There are three popular techniques to oversample minority classes: (i) random sampling with replacement, (ii) the adaptive synthetic (ADASYN) sampling method and (iii) the synthetic minority oversampling technique (SMOTE) [36]. Figure 6b shows the oversampled points (blue circle) using the most common technique SMOTE. After oversampling, there are 14 points on the lowest memory frequency, and they are used in the training phase (see Figure 3b) to improve the prediction accuracy.

5. Experimental Evaluation

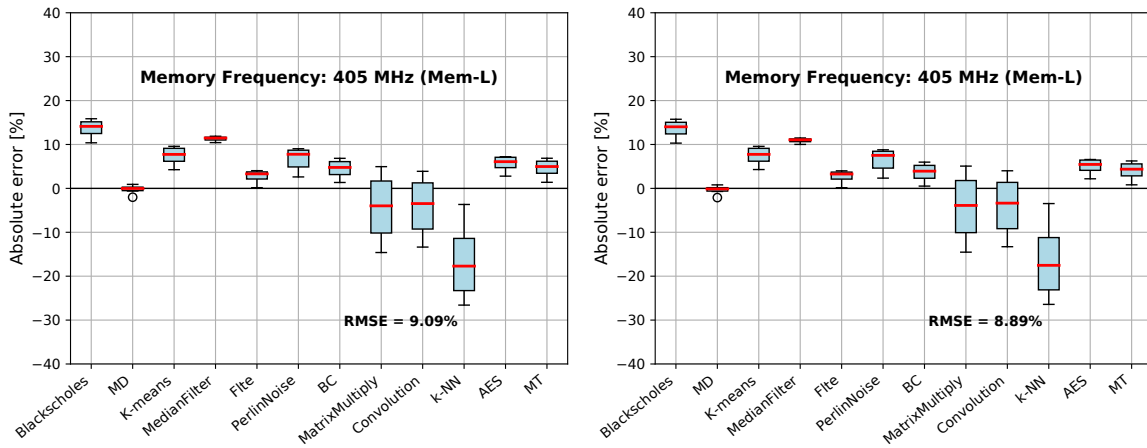
This Section presents and discusses the results of our study. The evaluation consists of an improved result using oversampling (Section 5.1); an analysis of energy and performance characterization (Section 5.2), followed by input-dependent analysis (Section 5.3); and an error analysis of our prediction model for speedup and energy efficiency (Section 5.4). It concludes with the evaluation of the predicted set of Pareto solutions (Section 5.5).

5.1. Experimental Evaluation of Oversampling

This section evaluates the accuracy of our speedup and normalized energy predictions using oversampling technique SMOTE. The modeling approach used for this evaluation is the one described in Section 3.4 based on linear SVR, RBF SVR and trained on micro-benchmarks. For each application, we trained the speedup and normalized energy models with all the oversampled frequency configurations, predicted the values and then calculated the error after actually running that configuration.

Figures 7a and 8a show the minimum, median and maximum error (%), and the error distribution of the 25 and 75 percentiles, for the speedup and normalized energy, respectively, for only one memory frequency (mem-L). We do not show the errors on the other three memory frequencies because SMOTE technique mostly affects the minority class (mem-L configuration). Therefore, we only show the prediction accuracy improvement on mem-L .

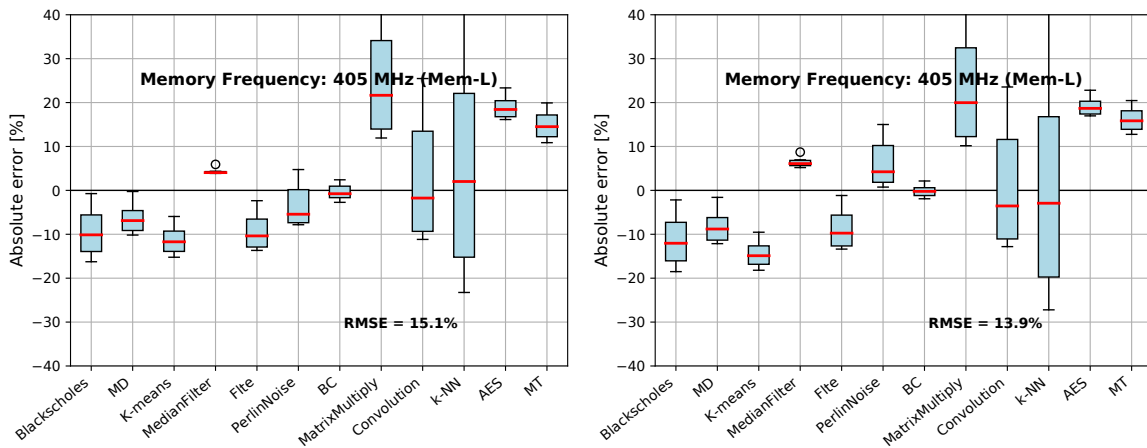
Figures 7b and 8b show the prediction error using SMOTE. Compared to the results under lowest memory frequency in Fan [3], the accuracy of speedup prediction and normalized energy prediction had only modest improvements of 0.1% and 1.0%, respectively.



(a) Prediction error (Fan et al. [3]).

(b) Prediction error (SMOTE).

Figure 7. Speedup prediction error under lowest memory frequency.



(a) Prediction error (Fan et al. [3]).

(b) Prediction error (SMOTE).

Figure 8. Prediction error of normalized energy under lowest memory frequency.

5.2. Application Characterization Analysis

In Figure 9, we analyzed the behavior of twelve test benchmarks in terms of both speedup and (normalized) energy consumption. For each code, we show speedup (x axis) and normalized energy (y axis) with different frequency configurations; the reference baseline for both correspond to the energy and performance value of the default frequency configuration. Generally, the applications show two main patterns (see top and other codes in Figure 9), i.e., memory- vs. compute-dominated kernels, which correspond to the different sensitivity to core and memory frequency changes.

Speedup In terms of speedup, k -NN shows a high variance with respect of the core frequency: for mem-H and mem-h, speedup goes from 0.62 up to 1.12, which means that it can double the performance by only changing the core frequency; for the mem-l the difference is even larger. The limited data for mem-L suggest a similar behavior. At the other extreme, blackscholes and MT show very little speedup difference while increasing the core frequency: all configurations are clustered to the same speedup for mem-L and l, while in mem-h and H the difference is minimal (from 0.89 to 1). Other applications behave within those two extreme codes.

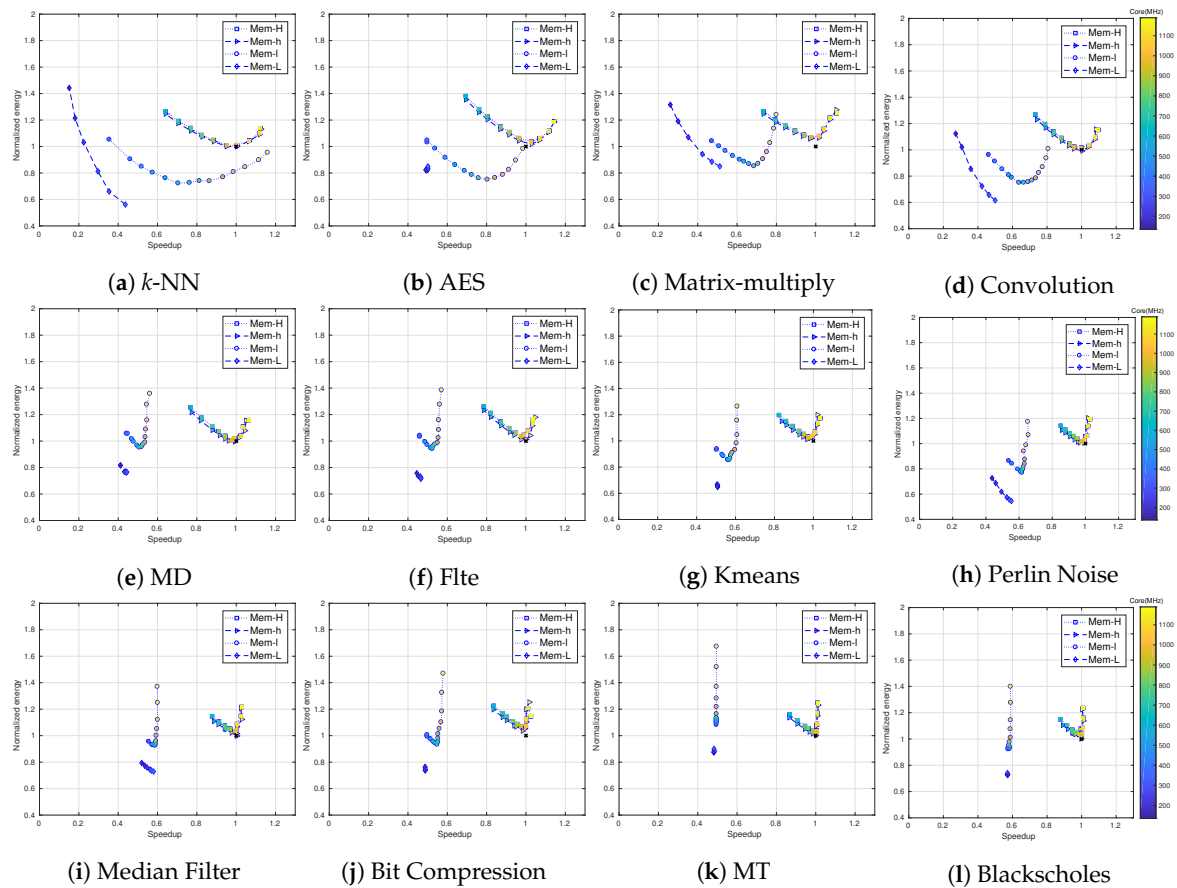


Figure 9. Speedup and normalized energy for twelve selected benchmarks and different frequency configurations as in Fan et al. [3]. Bottom (energy)-right (speedup) is better.

Normalized energy As previously mentioned, normalized energy often exhibits a parabolic distribution with a minimum. With respect to core frequency, it varies within smaller intervals. For the highest memory frequencies, it goes up to 1.4 for the first four codes, and up to 1.2 for the others. Again, the lowest configuration present very different behaviors: on *k*-NN, energy-per-task may be double the baseline, up to 0.8; in blackscholes, on the other hand, mem-L shows the same normalized energy for all the core frequencies.

High vs. low memory frequencies There is a big difference between high (mem-H and h) and low (mem-l and L) frequency configurations. Mem-H and h behave in a very similar way, with regard to both speedup and normalized energy. Both mem-l and mem-L have behavior that is much harder to predict. Mem-l behaves like the highest memory frequency at a lower normalized energy for the first four codes; however, on the other four codes, the configurations collapse to a line. The mem-L is even more erratic: in some codes, all points collapse to a very small area, practically a point. This is a problem for modeling: lowest memory configurations are much harder to model because their behavior is very erratic. In addition, because the supported configurations are not evenly distributed, we also have less points to base our analysis.

Pareto optimality In general, we can see two different patterns (this also extend to the other test benchmarks). In terms of Pareto optimality, most of the dominant points are mem-h and H. However, lower memory settings may as well contribute to the Pareto-set with configurations; in *k*-NN, for instance, mem-l has a configuration that is as fast the highest ones, but with 20% less energy consumption.

The default configuration is often a very good one. However, there are other dominant solutions that cannot be selected by using the default configuration.

5.3. Input-Size Analysis

Our previous work [3] was built on static code features, and did not take different input sizes into consideration. In fact, changing problem size results in a significant effect on the performance [20]. While in our case, it is more important to understand that the Pareto optimal solutions are likely to change with different input sizes. We analyze the statement presenting a case study with two applications: Matrix Multiply and MT (Mersenne Twister). These two applications have been chosen to represent very different behaviors, but the insights apply to all the tested applications. The applications have been executed with different problem sizes and the results are shown in Figures 10 and 11.

Matrix Multiply Application We tested the four memory settings mentioned above, labeled for simplicity L, l, h and H, each with all supported core frequencies. The default setting (mem-H and core at 1001 MHz) is at the intersection of the green lines. In terms of Matrix Multiply (Figure 10), speedup (the left column) benefits greatly from core scaling. On the other hand, (normalized) energy consumption behaves differently. In the middle column of Figure 10, for three out of four memory configurations, normalized energy is similar to a parabolic function with a minimum point: while increasing the core frequency, first the energy decreases as the computational time is reduced; but then, the higher frequencies have an impact on energy in a way that it does not compensate for the improvement on speedup. The lowest memory configuration (mem-L) seems to show a similar behavior; however, we do not have data at higher core frequencies to validate it (core frequencies larger than 405 MHz are not supported for mem-L; details in Figure 5a).

However, the speedup and normalized energy are slightly different between small and large input sizes. For the smaller problem size (the top in Figure 10), the rate of speedup increases with core frequency scaling and also the curvature of normalized energy are lower than the other three larger input sizes. While for the other three input sizes, the speedup and normalized energy consumption do not change a lot as the input size increases from 262144 to 1048576, respectively.

The right column in Figure 10 shows both energy and performance. As they behave differently, there is no single optimal configuration. In fact, this is a multi-objective optimization problem, with a set of Pareto-optimal solutions. It is important to note that for the small size (8192), the default configuration (black cross) is Pareto-optimal, while it is not for the other three input sizes.

Mersenne Twister Application In contrast to the behavior of Matrix Multiply, Mersenne Twister behaves differently, not only regarding the speedup and normalized energy consumption for the same input size, but also the effect of different sizes. For the speedup, increasing the core frequency does not improve performance, while selecting the highest memory frequency (mem-H) does, as shown in the left column of Figure 11. This behavior is justified by the larger number of memory operations. The energy consumption of Mersenne Twister behaves similar to Matrix Multiply, while the increase of energy consumption with higher core frequencies is larger.

In terms of input size, the speedup is decreased with the increasing input size, especially for mem-l and mem-L. The normalized energy consumption is better in the small input size (8192) than the other sizes. Mapping the observations to the bi-objective problem (the right column if Figure 11), it is clear to find that the mem-l solutions are not Pareto-optimal for large input size, which illustrates that the Pareto-optimal solutions mainly exist with higher memory frequency configurations for the large input size of a memory-bounded application.

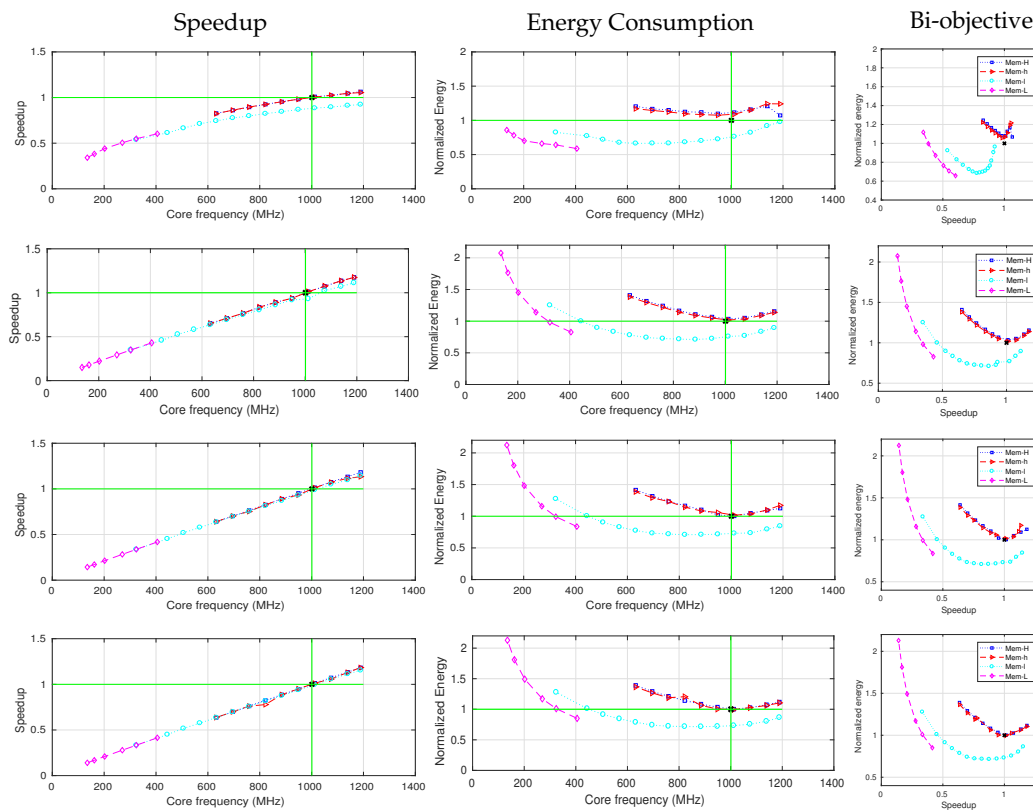


Figure 10. Matrix multiply results with different memory and core frequency in different input sizes (8192, 262,144, 524,288 and 1,048,576 from top to bottom).

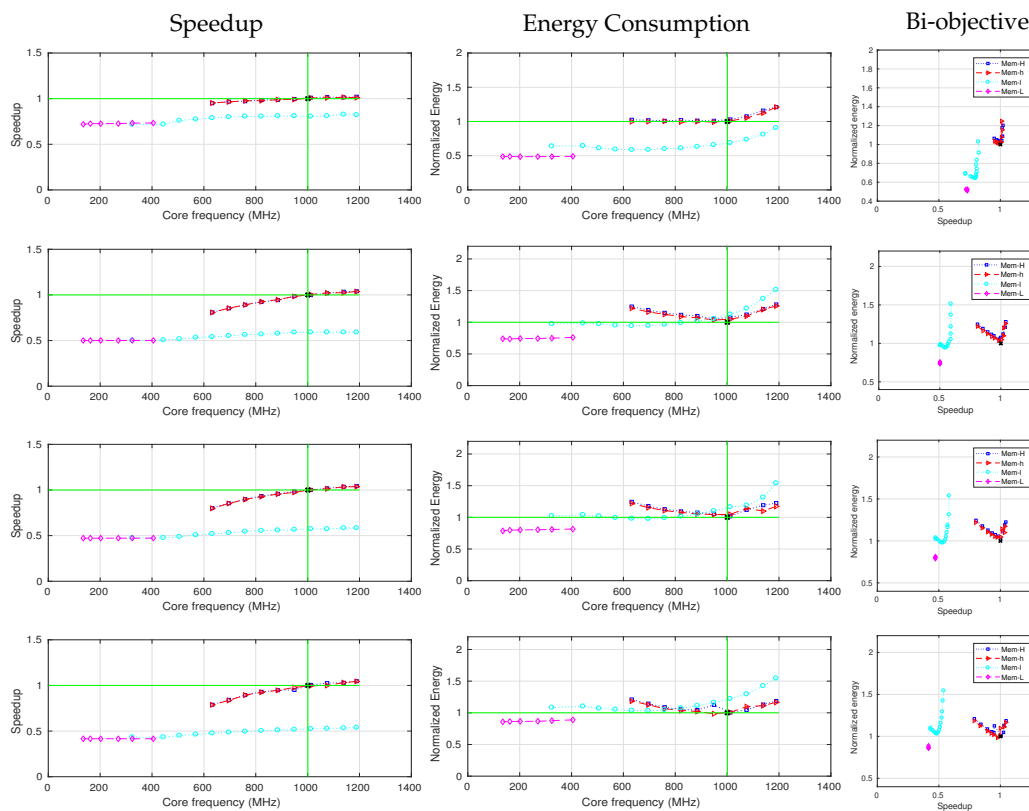


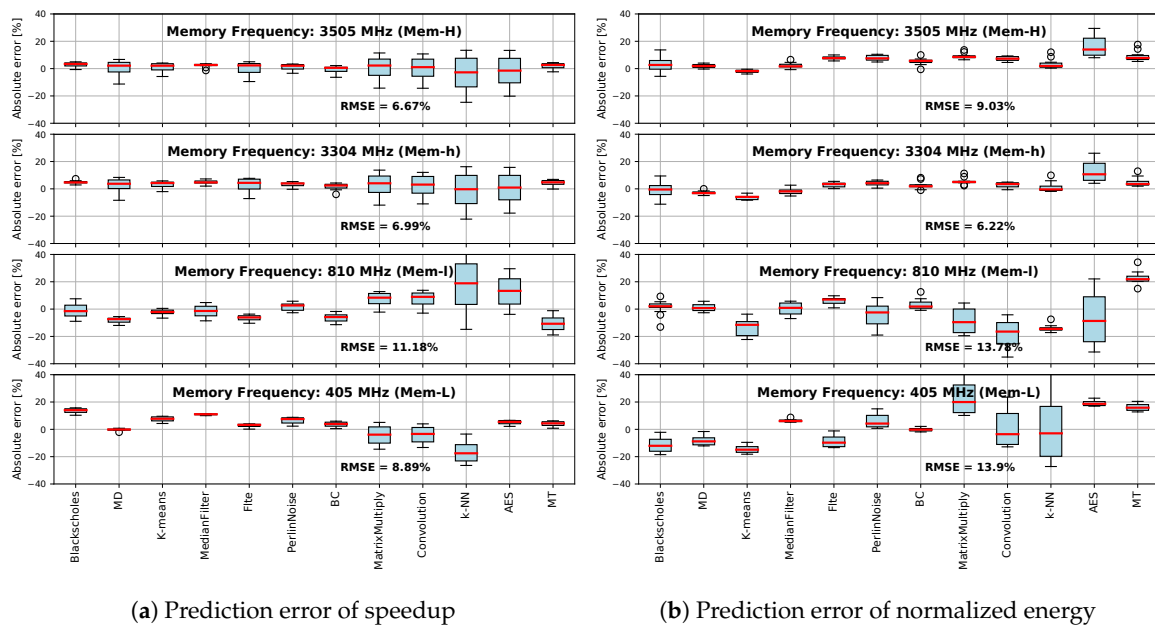
Figure 11. Mersenne Twister results with different memory and core frequency in different input size (8192, 262,144, 524,288 and 1,048,576 from top to bottom).

5.4. Accuracy of Speedup and Normalized Energy Predictions

We discussed the prediction error of speedup and normalized energy under the lowest memory frequency in Section 5.1. In this section we use the same analysis method to analyze the accuracy of predictions with all sampled frequency configurations. The predictions are obtained by resampled training data with SMOTE technique (see in Section 3.1).

In Figure 12a, the speedup error analysis shows that the error is dependent on the memory frequency. The error for the highest memory frequencies is quite low. It is usually within the 5% and goes over the 10% only for few outliers. The error here is also evenly distributed (over and under approximations are similar). Figure 12b shows the normalized energy prediction error by memory frequency and program. High memory frequency predictions are accurate. However, the relatively small error for the two highest-frequency configurations is not evenly distributed as for speedup, and it is also application dependent. For instance, the AES code is always over-approximated.

With respect to [3], the use of SMOTE reduced the speedup prediction error for memory frequency 405 MHz (from 9.1% to 8.9%) and normalized energy prediction error for memory frequency 405 MHz (from 15.1% to 13.9%).



(a) Prediction error of speedup (b) Prediction error of normalized energy

Figure 12. Accuracy of speedup and normalized energy predictions with SMOTE.

5.5. Accuracy of the Predicted Pareto Set

Once the two models have predicted the speedup and normalized energy for all frequency configurations, Algorithm 1 is used to calculate the predicted Pareto set. The accuracy analysis of the Pareto set is not trivial because our predicted set may include points that, in actual measured performance, are not dominant each other. In general, a better Pareto approximation is a set of solutions that, in terms of speedup and normalized energy, is the closest possible to the real Pareto-optimal one, which in our case has been evaluated on a subset of sampled configurations.

Lowest memory configuration Because of technical limitations of NVML, the memory configuration mem-L only supports six core configurations, up to only 405 MHz; therefore it covers only a limited part of the core-frequency domain. This leads to a lower accuracy of normalized energy prediction (Figure 12b). In addition, the Pareto analysis shows that the last point is usually dominant to the others, and it contributes to the overall set of Pareto points in 11 out of 12 codes, as shown in Figure 13 (the six mem-L points are in green, the last point is blue when dominant).

We used a simple heuristics to cover up with this issue: we used the predictive modeling approach on the other three memory configurations, and added the last of the mem-L configuration in the Pareto set. This simple solution is accurate for all but one code: AES.

Pareto frontier accuracy Figure 13 provides an overview of the Pareto set predicted by our method and the real ones, over a collection of twelve test benchmarks. The gray points represent the measured speedup and normalized energy of all the sampled frequency configurations (mem-H, mem-h and mem-l), except for mem-L, which are in green because they are not modeled with our predictive approach. The default configuration is marked with a black cross. The blue line represent the *real* Pareto front P^* , while the red crosses represent our predicted Pareto set P' (we did not connect these points because they are not necessarily dominant each other).

Coverage difference Table 2 shows different metrics that evaluate the accuracy of our predicted Pareto set. A measure that is frequently used in multi-objective optimization is the *hypervolume* (HV) indicator [37], which measures the volume of an approximation set with respect of a reference point in terms of the dominated area. In our case, we are interested on the *coverage difference* between two sets (e.g., the real Pareto set P^* and the approximation set P'). Therefore, we use the *binary hypervolume* metric [38], which is defined by:

$$D(P^*, P') = HV(P^* + P') - HV(P') \quad (2)$$

Because we maximize on speedup and minimize on normalized energy consumption, we select (0.0, 2.0) as the reference point. In addition, we also indicate the cardinality of both predicted and optimal Pareto set.

The twelve test benchmarks in Figure 13 are sorted by coverage difference. PerlIn Noise is the code with the nearest distance to the optimal Pareto set: the 12 predicted points are very close to the 10 optimal ones, and the overall coverage distance is minimal (0.0059). Overall, the Pareto predictions for the first six codes are very accurate (≤ 0.0208). Five more codes have some visible mispredictions which, however, translate to a not so large error (≤ 0.0362). k -NN is the worst code because of lowest accuracy of speedup prediction, which shows in Figure 12a.

Accuracy on Extrema We additionally evaluated the accuracy of our predictive approach on finding the extreme configurations; e.g., the two dominant points that have, respectively, minimum energy consumption and maximum speedup. Again, we removed from this analysis the mem-L configurations, whose accuracy was discussed above. The rationale behind this evaluation is that the accuracy on the Pareto predictions may not reflect the accuracy on these extreme points. As shown in Table 2, the point with maximum speedup is predicted exactly in 7 out of 12 cases, and the error is small. In case of the point with minimum energy, we have larger mispredictions in general; in particular two codes, AES and MT, have a very large error. This reflects the single-objective accuracy observed before, where the accuracy of speedup is generally higher than the accuracy of energy. The high error on all our analysis with the MT code is mainly due to the fact that lower memory configurations collapses to a point (mem-L) and a line (mem-l), a behavior that is not showed by other codes.

Predictive modeling in a multi-objective optimization scenario is challenging because few mispredicted points may impact the whole prediction, as they may dominate other solutions with a good approximation. Moreover, errors are not all equals: overestimation on speedup, as well as underestimation on energy, are much worse than the opposite, as they may introduce wrong dominant solutions. Despite that, our predictive approach is able to deliver good approximations in ten out of twelve test benchmarks.

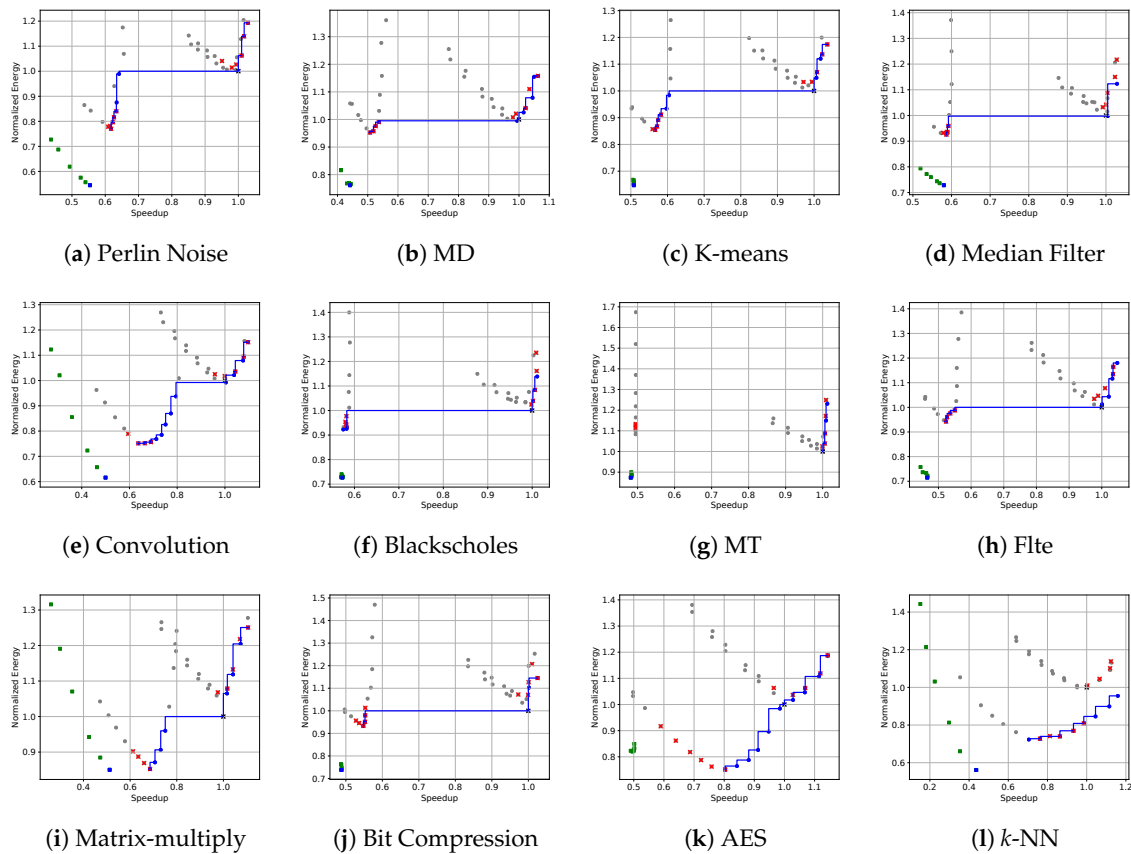


Figure 13. Accuracy of the predicted Pareto front. Measured solutions are shown for all configurations, while the other data points are based only on the highest frequency configurations. Bottom (energy)-right (speedup) is better.

Table 2. Evaluation of predicted Pareto fronts.

Benchmark	$D(P^*, P')$	#Points		Extrema Point Distance	
		$ P' $	$ P^* $	Max Speedup	Min Energy
PerlinNoise	0.0059	12	10	(0.0, 0.0)	(0.009, 0.008)
MD	0.0075	9	11	(0.0, 0.0)	(0.0, 0.0)
K-means	0.0155	10	12	(0.0, 0.0)	(0.007, 0.003)
MedianFilter	0.0162	11	6	(0.001, 0.094)	(0.008, 0.006)
Convolution	0.0197	10	14	(0.0, 0.0)	(0.042, 0.038)
Blackscholes	0.0208	9	7	(0.002, 0.097)	(0.007, 0.016)
MT	0.0272	10	6	(0.003, 0.018)	(0.505, 0.114)
Flte	0.0279	9	11	(0.012, 0.016)	(0.0, 0.0)
MatrixMultiply	0.0286	9	10	(0.0, 0.0)	(0.073, 0.050)
BitCompression	0.0316	11	6	(0.0, 0.0)	(0.020, 0.023)
AES	0.0362	11	14	(0.0, 0.0)	(0.214, 0.165)
k-NN	0.0660	9	8	(0.036, 0.183)	(0.057, 0.004)

6. Conclusions

This paper introduces a modeling approach aimed at predicting the best memory and core frequency settings for an OpenCL application on GPUs. The proposed methodology is based on a two-phase machine learning approach. To handle the imbalanced dataset in the training phase, SMOTE algorithm is introduced. After that the model built with oversampling data is used to predict the best frequency configurations of a new input kernel.

The modeling approach is designed to address both energy and performance in a multi-objective context. Different models are build to predict the normalized energy and the speedup. Successively, these models are used together to derive a set of Pareto-optimal solutions. Results on an NVIDIA Titan X show that it is possible to accurately predict a set of good memory configurations that are better than the default predefined one.

In the future, we believe that novel modeling approaches are required, given the rising interest in multi-objective problems involving energy efficiency, approximate computing and space optimization.

Author Contributions: Conceptualization, K.F.; Methodology, B.C. and K.F.; Software, K.F.; Validation, K.F.; Formal analysis, K.F.; Investigation, K.F.; Resources, B.J.; Data curation, K.F.; Writing—original draft preparation, K.F.; Writing—review and editing, B.C.; Visualization, K.F.; Supervision, B.J.; Project administration, B.C. and B.J.; Funding acquisition, B.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been partially funded by the DFG project CELERITY (CO 1544/1-1, project number 360291326) and by the China Scholarship Council.

Acknowledgments: We want to acknowledge the anonymous reviewers for their valuable insights.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Intel. RAPL (Running Average Power Limit) Power Meter. Available online: <https://01.org/rapl-power-meter> (accessed on 24 April 2020).
2. NVIDIA. NVIDIA Management Library (NVML). Available online: <https://developer.nvidia.com/nvidia-management-library-nvml> (accessed on 24 April 2020).
3. Fan, K.; Cosenza, B.; Juurlink, B.H.H. Predictable GPUs Frequency Scaling for Energy and Performance. In Proceedings of the 48th International Conference on Parallel Processing, ICPP, Kyoto, Japan, 5–8 August 2019; pp. 52:1–52:10.
4. Mei, X.; Yung, L.S.; Zhao, K.; Chu, X. A Measurement Study of GPU DVFS on Energy Conservation. In Proceedings of the Workshop on Power-Aware Computing and Systems, Berkeley, CA, USA, 3–6 November 2013; pp. 10:1–10:5.
5. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccion, R. Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e4143. [[CrossRef](#)]
6. Ge, R.; Vogt, R.; Majumder, J.; Alam, A.; Burtcher, M.; Zong, Z. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. In Proceedings of the 42nd International Conference on Parallel Processing, ICPP, Lyon, France, 1–4 October 2013.
7. Tiwari, A.; Laurenzano, M.; Peraza, J.; Carrington, L.; Snavely, A. Green Queue: Customized Large-Scale Clock Frequency Scaling. In Proceedings of the International Conference on Cloud and Green Computing, CGC, Xiangtan, China, 1–3 November 2012.
8. Vysocky, O.; Beseda, M.; Riha, L.; Zapletal, J.; Lysaght, M.; Kannan, V. MERIC and RADAR Generator: Tools for Energy Evaluation and Runtime Tuning of HPC Applications. In Proceedings of the High Performance Computing in Science and Engineering—Third International Conference, HPCSE, Karolinka, Czech Republic, 22–25 May 2017; Revised Selected Papers. pp. 144–159.
9. Hamano, T.; Endo, T.; Matsuoka, S. Power-aware dynamic task scheduling for heterogeneous accelerated clusters. In Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS, Rome, Italy, 23–29 May 2009; pp. 1–8.
10. Lopes, A.; Pratas, F.; Sousa, L.; Ilic, A. Exploring GPU performance, power and energy-efficiency bounds with Cache-aware Roofline Modeling. In Proceedings of the 2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, Santa Rosa, CA, USA, 24–25 April 2017; pp. 259–268.
11. Ma, K.; Li, X.; Chen, W.; Zhang, C.; Wang, X. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In Proceedings of the 41st International Conference on Parallel Processing, ICPP, Pittsburgh, PA, USA, 10–13 September 2012; pp. 48–57.

12. Song, S.; Lee, M.; Kim, J.; Seo, W.; Cho, Y.; Ryu, S. Energy-efficient scheduling for memory-intensive GPGPU workloads. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Dresden, Germany, 24–28 March 2014; pp. 1–6.
13. Choi, J.; Vuduc, R.W. Analyzing the Energy Efficiency of the Fast Multipole Method Using a DVFS-Aware Energy Model. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops, Chicago, IL, USA, 23–27 May 2016; pp. 79–88.
14. Lee, J.H.; Nigania, N.; Kim, H.; Patel, K.; Kim, H. OpenCL Performance Evaluation on Modern Multicore CPUs. *Sci. Program.* **2015**, *2015*. [[CrossRef](#)]
15. Shen, J.; Fang, J.; Sips, H.J.; Varbanescu, A.L. An application-centric evaluation of OpenCL on multi-core CPUs. *Parallel Comput.* **2013**, *39*, 834–850. [[CrossRef](#)]
16. Harris, G.; Panangadan, A.V.; Prasanna, V.K. GPU-Accelerated Parameter Optimization for Classification Rule Learning. In Proceedings of the International Florida Artificial Intelligence Research Society Conference, FLAIRS, Key Largo, FL, USA, 16–18 May 2016; pp. 436–441.
17. Pohl, A.; Cosenza, B.; Juurlink, B.H.H. Portable Cost Modeling for Auto-Vectorizers. In Proceedings of the 27th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2019, Rennes, France, 21–25 October 2019; pp. 359–369.
18. Panneerselvam, S.; Swift, M.M. Rinnegan: Efficient Resource Use in Heterogeneous Architectures. In Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT, Haifa, Israel, 11–15 September 2016.
19. Wang, Q.; Chu, X. GPGPU Performance Estimation with Core and Memory Frequency Scaling. In Proceedings of the 24th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2018, Singapore, 11–13 December 2018; pp. 417–424. doi:10.1109/PADSW.2018.8645000. [[CrossRef](#)]
20. Kofler, K.; Grasso, I.; Cosenza, B.; Fahringer, T. An automatic input-sensitive approach for heterogeneous task partitioning. In Proceedings of the International Conference on Supercomputing, ICS'13, Eugene, OR, USA, 10–14 June 2013; pp. 149–160.
21. Ge, R.; Feng, X.; Cameron, K.W. Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems. In Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS, Rome, Italy, 25–29 May 2009; pp. 1–8.
22. Bhattacharyya, A.; Kwasniewski, G.; Hoefler, T. Using Compiler Techniques to Improve Automatic Performance Modeling. In Proceedings of the International Conference on Parallel Architecture and Compilation, San Francisco, CA, USA, 18–21 October 2015.
23. De Mesmay, F.; Voronenko, Y.; Püschel, M. Offline library adaptation using automatically generated heuristics. In Proceedings of the 24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS, Atlanta, GA, USA, 19–23 April 2010.
24. Zuluaga, M.; Krause, A.; Püschel, M. e-PAL: An Active Learning Approach to the Multi-Objective Optimization Problem. *J. Mach. Learn. Res.* **2016**, *17*, 104:1–104:32.
25. Grewe, D.; O'Boyle, M.F.P. A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL. In Proceedings of the 20th International Conference on Compiler Construction, CC, Saarbrücken, Germany, 26 March–3 April 2011; pp. 286–305.
26. Den Steen, S.V.; Eyerman, S.; Pestel, S.D.; Mechri, M.; Carlson, T.E.; Black-Schaffer, D.; Hagersten, E.; Eeckhout, L. Analytical Processor Performance and Power Modeling Using Micro-Architecture Independent Characteristics. *IEEE Trans. Comput.* **2016**, *65*, 3537–3551. [[CrossRef](#)]
27. Abe, Y.; Sasaki, H.; Kato, S.; Inoue, K.; Eda, H.; Peres, M. Power and Performance Characterization and Modeling of GPU-Accelerated Systems. In Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, 19–23 May 2014; pp. 113–122.
28. Guerreiro, J.; Ilic, A.; Roma, N.; Tomás, P. GPU Static Modeling using PTX and Deep Structured Learning. *IEEE Access* **2019**, *7*, 159150–159161. [[CrossRef](#)]
29. Guerreiro, J.; Ilic, A.; Roma, N.; Tomas, P. GPGPU Power Modelling for Multi-Domain Voltage-Frequency Scaling. In Proceedings of the 24th IEEE International Symposium on High-Performance Computing Architecture, HPCA, Vienna, Austria, 24–28 February 2018.
30. Wu, G.Y.; Greathouse, J.L.; Lyashevsky, A.; Jayasena, N.; Chiou, D. GPGPU performance and power estimation using machine learning. In Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, 2 October 2015; pp. 564–576.

31. Isci, C.; Martonosi, M. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36), San Diego, CA, USA, 5 December 2003.
32. Cummins, C.; Petoumenos, P.; Wang, Z.; Leather, H. Synthesizing benchmarks for predictive modeling. In Proceedings of the International Symposium on Code Generation and Optimization, CGO, Austin, TX, USA, 4–8 February 2017; pp. 86–99.
33. Cosenza, B.; Durillo, J.J.; Ermon, S.; Juurlink, B.H.H. Autotuning Stencil Computations with Structural Ordinal Regression Learning. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium, IPDPS, Orlando, FL, USA, 29 May–2 June 2017; pp. 287–296.
34. Smola, A.J.; Schölkopf, B. A tutorial on support vector regression. *Stat. Comput.* **2004**, *14*, 199–222. [[CrossRef](#)]
35. Li, B.; Li, J.; Tang, K.; Yao, X. Many-Objective Evolutionary Algorithms: A Survey. *ACM Comput. Surv.* **2015**, *48*, 13:1–13:35. [[CrossRef](#)]
36. Lemaître, G.; Nogueira, F.; Aridas, C.K. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *J. Mach. Learn. Res.* **2017**, *18*, 1–5.
37. Zitzler, E.; Thiele, L.; Laumanns, M.; Fonseca, C.M.; da Fonseca, V.G. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *Trans. Evol. Comp.* **2003**, *7*, 117–132. [[CrossRef](#)]
38. Zitzler, E. Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Ph.D. Thesis, University of Zurich, Zürich, Switzerland, 1999.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).