Predictable GPUs Frequency Scaling for Energy and Performance

Kaijie Fan
Technische Universität Berlin
kaijie.fan@campus.tu-berlin.de

Biagio Cosenza
Technische Universität Berlin

Ben Juurlink
Technische Universität Berlin

ABSTRACT

Dynamic voltage and frequency scaling (DVFS) is an important solution to balance performance and energy consumption, and hardware vendors provide management libraries that allow the programmer to change both memory and core frequencies. The possibility to manually set these frequencies is a great opportunity for application tuning, which can focus on the best application-dependent setting. However, this task is not straightforward because of the large set of possible configurations and because of the multi-objective nature of the problem, which minimizes energy consumption and maximizes performance.

This paper proposes a method to predict the best core and memory frequency configurations on GPUs for an input OpenCL kernel. Our modeling approach, based on machine learning, first predicts speedup and normalized energy over the default frequency configuration. Then, it combines the two models into a multi-objective one that predicts a Pareto-set of frequency configurations. The approach uses static code features, is built on a set of carefully designed micro-benchmarks, and can predict the best frequency settings of a new kernel without executing it. Test results show that our modeling approach is very accurate on predicting extrema points and Pareto set for ten out of twelve test benchmarks, and discover frequency configurations that dominate the default configuration in either energy or performance.

CCS CONCEPTS

• **Computer systems organization** → **Parallel architectures**;
• **Hardware** → *Power and energy*.

KEYWORDS

Frequency scaling, Energy efficiency, GPUs, Modeling

ACM Reference Format:
Kaijie Fan, Biagio Cosenza, and Ben Juurlink. 2019. Predictable GPUs Frequency Scaling for Energy and Performance. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337833

ICPP 2019, August 5–8, 2019, Kyoto, Japan

ACM ISBN 978-1-4503-6295-5/19/08…$15.00
https://doi.org/10.1145/3337821.3337833

1 INTRODUCTION

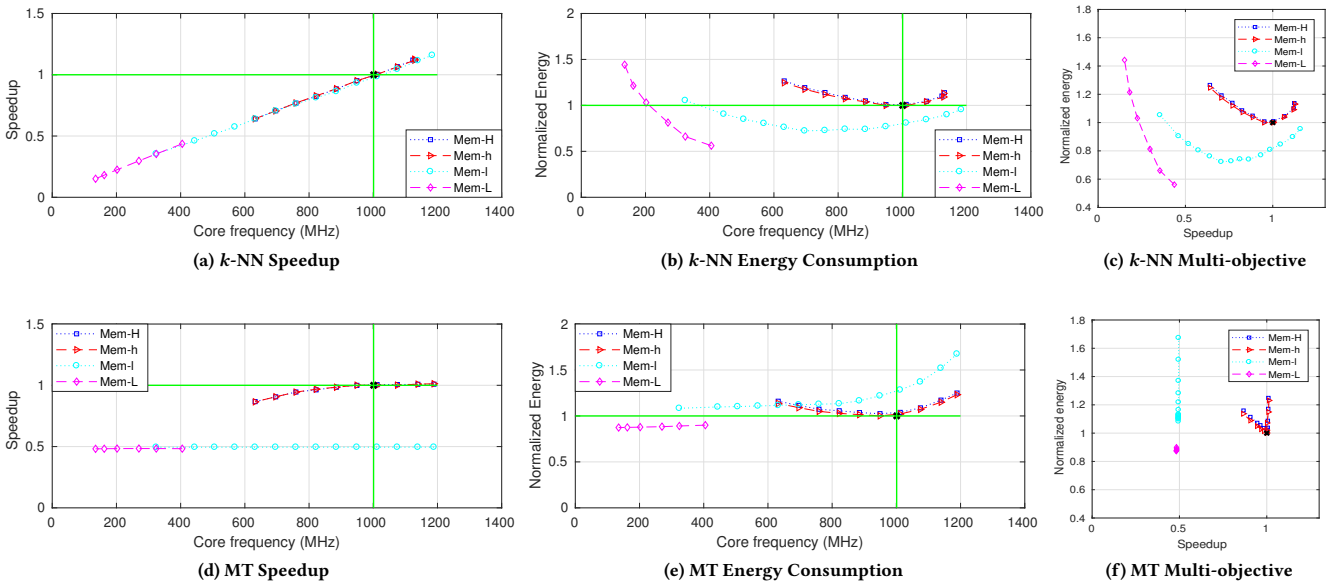
Energy consumption is a major concern of today's computing platforms. The energy consumed by a program affects the cost of large-scale compute clusters as well as the battery life of mobile devices and embedded systems. Modern processors provide a number of solutions to tackle power and energy constraints such as asymmetric cores, power and clock gating, and *dynamic voltage and frequency scaling* (DVFS). The latter, in particular, is very effective in improving energy efficiency until it reaches a voltage and frequency point that is close to the threshold voltage; after that point, the energy efficiency decreases again [15].

In this context, *graphics processing units* (GPUs) represent an interesting scenario as they provide high performance, but they also consume a considerable amount of power. Fortunately, modern GPUs implement DVFS. For instance, the *NVIDIA Management Library* (NVML) [22] provides a way to report the board power draw, power limits, and to dynamically set both core and memory frequencies. Being able to tune core and memory frequencies is very interesting: different applications may show varying energy consumption and performance with different frequency setting; e.g., a memory-bounded application may benefit of core down-scaling with reduced energy consumption at the same performance. However, manually performing such tuning is not easy. For example, the NVIDIA GTX Titan X supports four memory frequencies (405, 810, 3304, and 3505 MHz) and 85 core frequencies (from 135 to 1392 MHz), with a total number of 219 possible configurations (note that not all memory-core combinations are supported; e.g., is not possible to have both maximal core and memory frequency). Moreover, while energy-per-task is the metric to be minimized, we also want our programs to deliver high performance. These two conflicting goals translate into a multi-objective optimization problem where there is no single optimal solution, but instead a set of Pareto-optimal solutions, each exposing different trade-off between energy consumption and performance. This work aims at predicting the best memory and core frequency configurations of an input OpenCL kernel.

1.1 Motivation

Predicting the best frequency configurations is challenging for different aspects. The large number of settings makes it impractical to perform an exhaustive search, in particular if it has to be performed on many applications. At the same time, the tuning space of such bi-objective problems present some challenging properties.

Figure 1 shows speedup and normalized energy consumption for two examples: `k-NN` and `MT` (Mersenne Twister). These two applications have been chosen to represent very different behaviors, but the insights apply to all the tested applications.



(a) *k*-NN Speedup (b) *k*-NN Energy Consumption (c) *k*-NN Multi-objective

(d) MT Speedup (e) MT Energy Consumption (f) MT Multi-objective

Figure 1: Speedup (a,d), normalized energy consumption (b,e) and both (c,f) of two applications with different memory and core frequency.

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 speedup (Fig. 1a), `k-NN` benefits greatly from core scaling. However, this is not true for `MT` (Fig. 1d), where increasing the core frequency does not improve performance, while selecting the highest memory frequency (`mem-H`) does. This behavior is justified by the larger amount of memory operations.

On the other hand, (normalized) energy consumption behaves differently. In `k-NN` (Fig. 1b), for three out of four memory configurations, normalized energy is similar to a parabolic function with a minimum point in [885,987] MHz: 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 Fig. 4a). Moreover, this behavior depends on the kernel: e.g., in `MT` (Fig. 1e), the increase of energy consumption with higher core frequencies is larger than `k-NN`.

Figure 1c and 1f show 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 notice that the baseline default configuration (black cross) may be not Pareto-optimal (Fig. 1c) or be only one of more dominant solutions (Fig. 1f). This paper starts from this observation to derive a multi-objective model that tries to automatically predict Pareto-optimal frequency configurations of a new kernel.

1.2 Contributions

The focus of this research is two-fold. First, we analyzed how normalized energy and speedup behave on GPUs applications. The analysis, based on a large set of codes including real applications and synthetic micro-kernels, aimed at discovering which program properties, in particular static code features, affect the energetic and performance behavior of a kernel.

The insights of this analysis motivated the design of a predictive model that is able to select the *best* (hopefully *Pareto-optimal*) frequency configurations in terms of normalized energy and speedup. The predictive framework is purely based on static information extracted from the input OpenCL kernel. Thus, once the model is built on a set of carefully-designed micro-benchmarks, it is capable to quickly derive the best configurations for any new application.

This work makes the following contributions:

- An analysis of the Pareto optimality (performance versus energy consumption) of GPUs 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 is built on 106 synthetic micro-benchmarks. It predicts normalized energy and speedup with different *ad hoc* methods, and then derives a Pareto set of configurations out of the individual models.
- An experimental evaluation of the proposed models on twelve test benchmarks on an NVIDIA Titan X, and a comparison against the default static settings.

2 RELATED WORK

Energy-performance modeling has received great attention from the research community. Mei *et al.* [21], in particular, wrote a survey and measurement study of GPU DVFS. Ge *et al.* [9] applied fine-grained GPU core frequency and coarse-grained GPU memory frequency on a Kepler K20c GPU. Tiwari *et al.* [27] proposed DVFS strategy both in intra-node and inter-node to reduce power consumption on CPU.

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 [3, 12, 19, 20, 26]. Among them, Hamano *et al.* [12] proposed a task scheduling scheme that optimizes the overall energy consumption of the system. Lopes *et al.* [19] proposed a model that relies on extensive GPU micro-benchmarking using a cache-aware roofline model. Song *et al.* [26] 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 [13, 17, 24]. Approaches [23, 28] to predict performance by taking DVFS into consideration have been discussed. Kofler *et al.* [16] and Ge *et al.* [8] proposed a machine learning approach based on *Artificial Neural Networks* (ANN) that automatically performs heterogeneous task partitioning. Bhattacharyya *et al.* [2] improved performance model by combining static and dynamic analysis. Mesmay *et al.* [6] converted online adaptive libraries into offline by automatically generating heuristics. ϵ-PAL [32] proposes 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.

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

Paper	Static	Pareto-optimal	Frequency Scaling	Machine Learning
Grewe *et al.* [10]	✓	×	×	✓
Steen *et al.* [7]	×	✓	×	×
Abe *et al.* [1]	×	×	✓	×
Guerreiro *et al.* [11]	×	×	✓	✓
Wu *et al.* [29]	×	×	✓	✓
Our work	✓	✓	✓	✓

Here we discuss most important related work and Table 1 shows the comparison. Grewe *et al.* [10] used machine learning for a purely static approach which, however, only predicted task partinioning. Steen *et al.* [7] presented a micro-architecture independent profiler based on a mechanistic performance model on CPU. However, they do not take frequency scale into consideration, which, as already described in Section 1.1, plays a heavy role on energy and performance behaving.

Abe *et al.* [1], Guerreiro *et al.* [11] and Wu *et al.* [29] proposed performance and energy models by taking frequency scaling into consideration. Among them, Abe *et al.* [1] estimated the models by using performance counters but did not consider the non-linear scaling effects of the voltage. Guerreiro *et al.* [11] made more improvement: 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.* [29] studied the performance and energy models of an AMD GPU by using K-means clustering.

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 GPU using static models.

3 METHODOLOGY

Our approach to model the energy consumption and speedup of a input kernel is based on machine learning methodology, applied to a feature representation of the kernel code and of the frequency domain. This Section describes an overview of the method, followed by a description of the feature representation, the synthetic training data, the predictive modeling approach for speedup and energy, and how those are used to derive the predicted Pareto set.

3.1 Overview

The methodology proposed by our work is based on a 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. Fig. 2 and Fig. 3 illustrate the workflow of this work,respectively for the training and prediction phases.

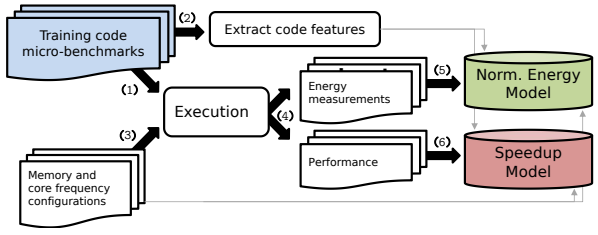


Figure 2: Training phase.

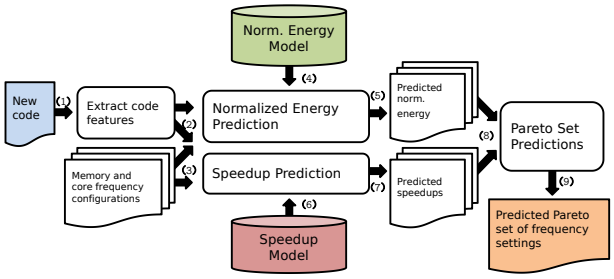


Figure 3: Prediction phase.

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 data set. Successively, each micro-benchmark is executed with various frequency configurations (3). The obtained energy and performance measurements (4), together with frequency configurations and the static features, are stored in the training data set. Once these steps have been accomplished for all micro-benchmarks, the features are normalized and used to train the two models for normalized energy (5) and speedup prediction (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)(6) are used to predict its normalized energy consumptions (5) and speedup (7). Once the predictions for all memory configurations are available (8), the dominant points are calculated and returned as predicted Pareto set (9).

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.* [11], where features are designed to reflect the modular design and structure of the GPUs architecture, which allow them to easily decompose the power consumption in multiple architectural components [14]. 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

$$\begin{aligned}\vec{k} \quad = \quad & (k_{int\_add}, k_{int\_mul}, k_{int\_div}, k_{int\_bw}, \\ & k_{float\_add}, k_{float\_mul}, k_{float\_div}, k_{sf}, \\ & k_{gl\_access}, k_{loc\_access})\end{aligned}$$

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

Frequency configurations are also represented as features: the couple $\vec{f} = (f_{core}, f_{mem})$, 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) and $[405, 3505]$ (memory), 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 allow us to have all features mapped in the same range, so that each feature contributes approximately proportionately to the model; as a result, codes with the same arithmetic intensity but different number of total instructions will have the same feature representation.

With respect of related work [10, 16], where features are extracted from the AST, we instead implemented the feature extraction with an LLVM pass running on the intermediate representation of the kernel.

3.3 Synthetic Training Benchmarks

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 [5], or in domain-specific context such as stencil computations [4].

Our approach is a combination of pattern-based and domain-specific synthetic code generation, and is carefully designed around the feature representation [11]. Code benchmarks are generated by pattern: each pattern covers a specific feature, and generates a number codes with different instruction intensity (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 is 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 minutes to test 40 frequency settings, 70 minutes to test all the 174 frequency settings, making therefore difficult the exhaustive search of all configurations.

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 Joule). 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 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 data set $(\vec{w}_1, y_1), \ldots, (\vec{w}_n, y_n)$, 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, α_i and α_i^* are Lagrange multipliers that are obtained by solving the associated dual optimization problem [25]. $K(\vec{w}_i, \vec{w}_j)$ is the kernel function, which will be specified later in this section.

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 we sought that, while keeping constant input code and memory frequency, the speedup increases linearly with the core frequency (this can bee seen also in the motivational examples in Section 1.1). 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 $(\vec{w}_1, s_1), \cdots, (\vec{w}_n, s_n)$, 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 (2) is defined as $K(\vec{w}_i, \vec{w}_j) = \vec{w}_i \cdot \vec{w}_j$. Additionally, the C and ϵ parameters [25] are set to 1000 and 0.1.

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

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, and with a minimum (see Section 1.1). After this point, the increase on core frequency does not compensate 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 kernel.

Formally, given a set of n kernel executions in the training set T, we define a training sample of input-output pairs $(\vec{w}_1, e_1), \cdots, (\vec{w}_n, e_n)$, 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 (2) 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 $\vec{w}$.

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 in 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:

- $s_i \geq s_j$ and $e_i < e_j$, or
- $s_i > s_j$ and $e_i \leq e_j$.

A kernel execution $\vec{w}^*$ is Pareto-optimal if there is no other kernel execution $\vec{w}'$ such that $\vec{w}' \prec \vec{w}^*$. 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 the following algorithm:

Algorithm 1 Simple Pareto set calculation

```
1:  Predictions ← {(s_1, e_1), ..., (s_m, e_m)}
2:  P' ← ∅                                   ▷ Output Pareto set
3:  Dominated ← ∅                            ▷ Set of dominated points
4:  while Predictions ≠ ∅ do
5:    candidate ← Predictions.pop()
6:    foreach point ∈ Predictions do
7:      if candidate ≺ point then
8:        Predictions ← Predictions \ {candidate}
9:        Dominated ← Dominated ∪ {candidate}
10:     if point ≺ candidate then
11:       Dominated ← Dominate ∪ {point}
12:     else                    ▷ We have found a point in the frontier
13:       P' ← P' ∪ {candidate}
```

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 [18].

4 EXPERIMENTAL EVALUATION

This Section presents and discusses the experimental design and the results of our study. The test setup is discussed in the next section (4.1). The evaluation consists of an analysis of energy and performance characterization (4.2), followed by an error analysis of our prediction model for speedup (4.3) and energy efficiency (4.4). It concludes with the evaluation of the predicted set of Pareto solutions (4.5).

4.1 Frequency Domain and Test Setting

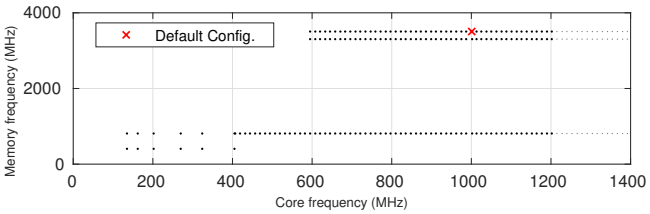
Our work is based on the ability of setting up memory and core frequencies, and on getting an accurate measurement of the energy consumption of a task execution. For the experimental evaluation of our approach, we relied on the capabilities provided by the NVML [22] library. It supports a number of functions 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()`).

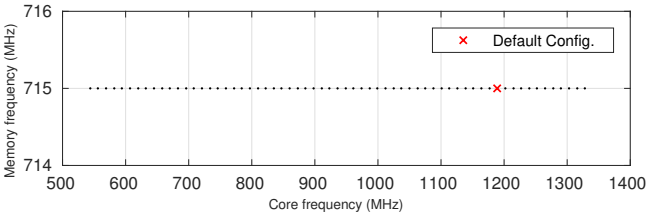
It is important to remark that different NVIDIA GPUs may have very different tunable configurations. For example, the NVIDIA Titan X provides four tunable memory frequencies while the NVIDIA Tesla P100 only supports one. In addition, we experimentally noticed that some of the configurations marked as supported by NVML are not available, because the setting function does not actually change the frequencies.

Fig. 4 shows those frequency configurations on an NVIDIA Titan X (4a) and a Tesla P100 (4b). 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 is 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.



(a) Titan X



(b) Tesla P100

Figure 4: Supported combinations of memory and core frequencies.

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 test presented in this work have been performed on both NVIDIA GTX Titan X. 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 1001MHz (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 a too short time. Therefore, the applications have been executed multiple times, to make sure that the execution time is long enough to reach a statistical consistent power value.

4.2 Application Characterization Analysis

We analyzed the behavior of twelve test benchmarks in terms of both speedup and (normalized) energy consumptions.

In Fig. 5, we show a selection of eight significant applications taken from the twelve test benchmarks. 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.

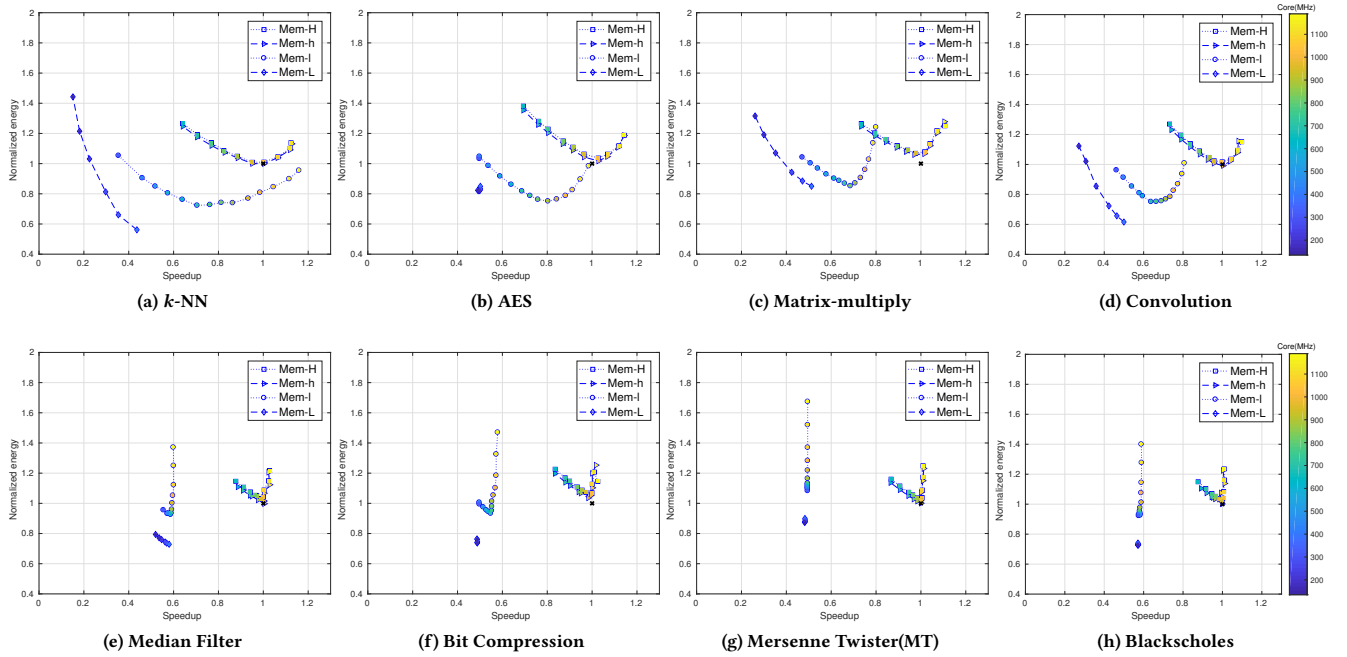
Generally, the applications show two main patterns (see top and bottom codes in Fig. 5), i.e., memory- vs compute-dominated kernels, which correspond to the different sensitivity to core and memory frequency changes.

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, while the limited data for `mem-L` suggest a similar behavior.

At the other extreme, `blackscholes` shows 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.

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 from twice the baseline, up to 0.8; in `blacksholes`, on the other side, `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 of 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



(a) *k*-NN (b) AES (c) Matrix-multiply (d) Convolution

(e) Median Filter (f) Bit Compression (g) Mersenne Twister(MT) (h) Blackscholes

Figure 5: Speedup and normalized energy for eight selected benchmarks and different frequency configurations. Bottom (energy)-right (speedup) is better.

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.

4.3 Accuracy of Speedup Predictions

This section evaluates the accuracy of our speedup predictions. The modeling approach used for this evaluation is the one described in Section 3.4 based on linear SVR and trained on micro-benchmarks. The evaluation is performed on the features extracted from the twelve test benchmarks discussed before. For each application, we predicted the speedup value for all the sampled frequency configurations, and then we calculated the error after actually running that configuration.

To have an accurate analysis of the accuracy, we grouped the errors by memory and core frequency. The box-plots in Fig. 6 shows

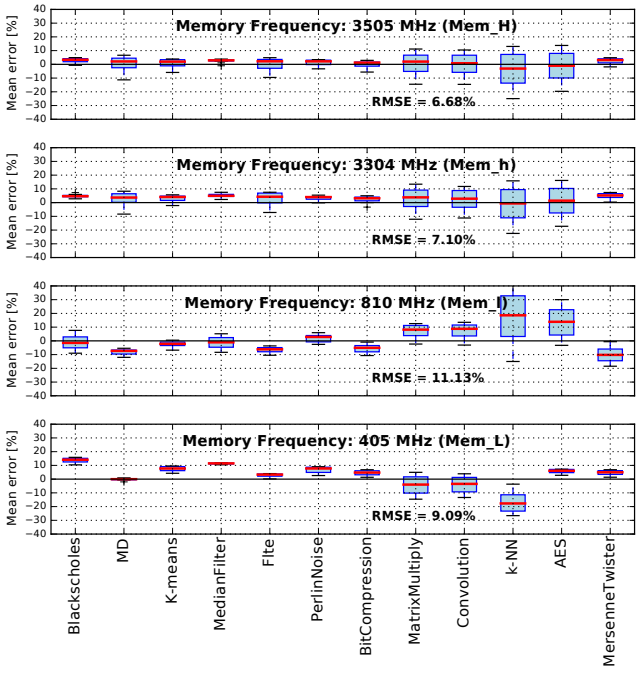


Figure 6: Prediction error of speedup

the minimum, median and maximum error (%), and the error distribution of the 25 and 75 percentile.

The 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).

On the other hand, the two lowest memory frequencies are very hard to predict. Results show a mean error that, in some cases, is higher than 20%. `Mem-L` is mainly under-approximated, while `mem-l` is mainly over-approximated. The error is application-dependent, the reasons for the larger error are mainly two. First, as shown in previous analysis, some applications have a very different sample distribution, clustered at `mem-L`. Second, because of the limited number of supported configurations, we have only six samples for `mem-L` in the training set. Such a small amount of points is not enough for predictive modeling.

4.4 Accuracy of Normalized Energy Predictions

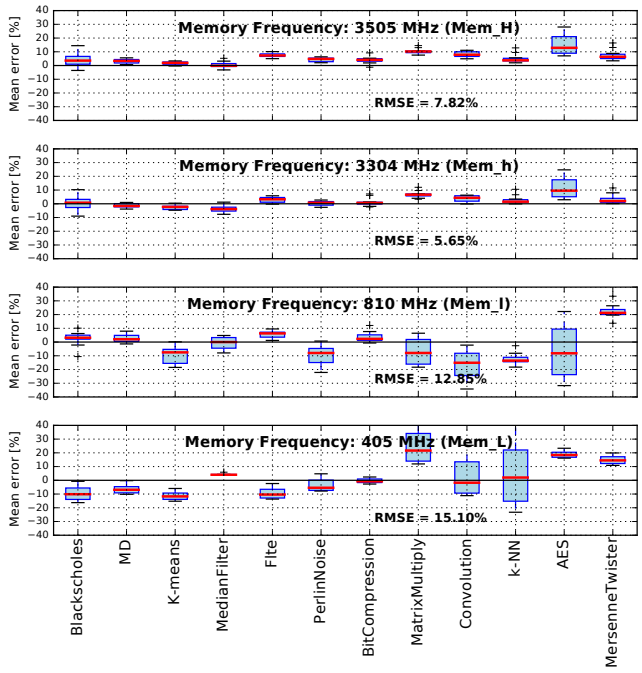


Figure 7: Prediction error of normalized energy

To evaluate the accuracy of the SVR model used for normalized energy predictions, we used the same methodology discussed in the previous section. Fig. 7 shows the 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.

Also this model lacks of accuracy for the two lowest memory configurations. In this case, `mem-l` are mainly under-approximated, while for `mem-L` the error is application-dependent. As for performance, the error analysis indicates that lowest memory configurations have higher error because of their very varying energetic behavior, and because of the limited number of supported configurations for `mem-L`.

4.5 Accuracy of 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 (Fig. 7). 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 Fig. 8 (the six `mem-L` points are in green, the last point is blue when dominant).

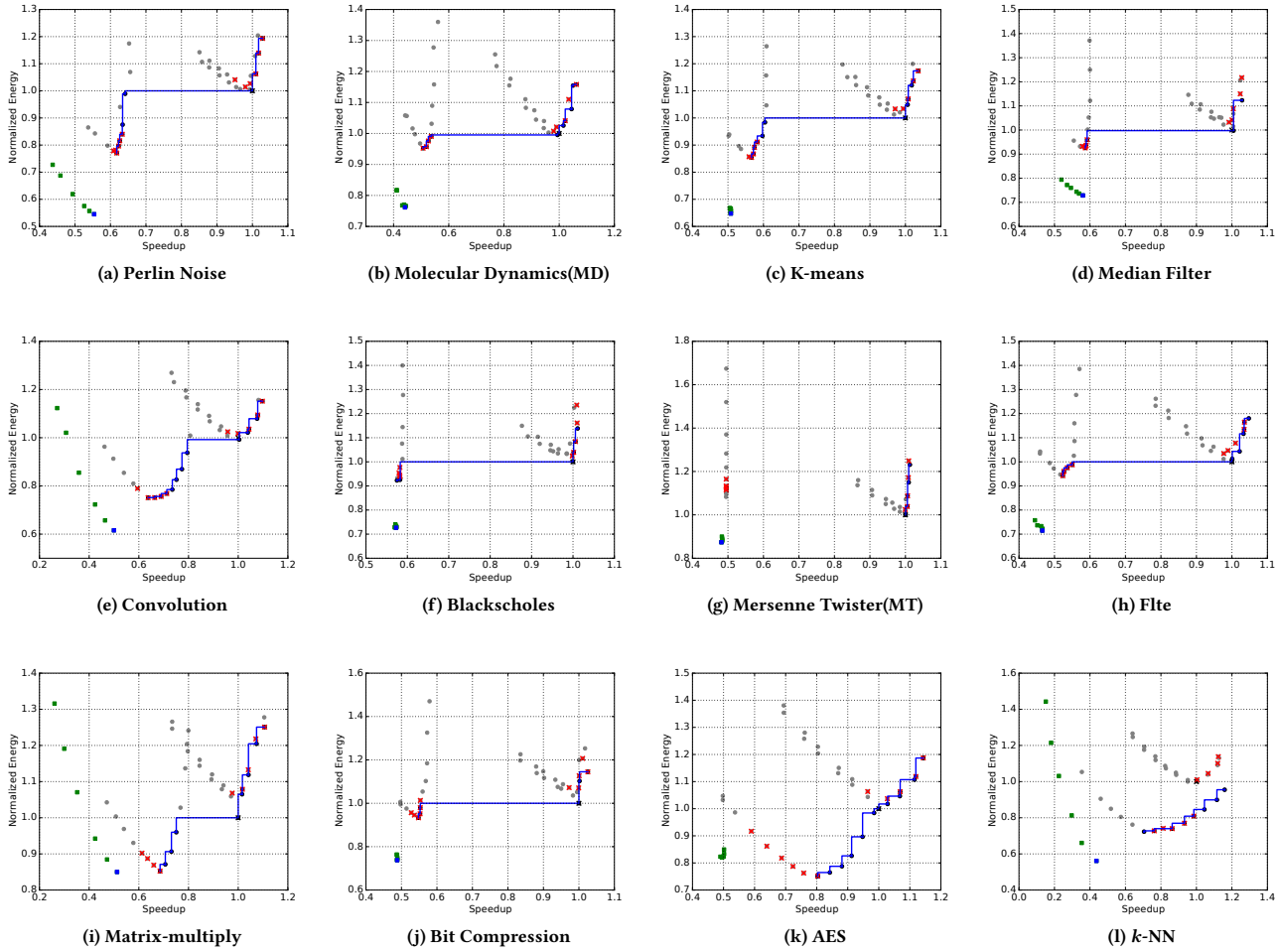
We used a simple heuristics to cover up with this issue: we used the predictive modeling approach on the other three memory configurations, and added the last of the `mem-L` configuration in the Pareto set. This simple solution is accurate for all but one code: `AES`.

Pareto frontier accuracy. Fig. 8 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 represents 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).

Table 2: Evaluation of predicted Pareto fronts

Benchmark	$D(P^*, P')$	#Points $\lvert P'\rvert$	#Points $\lvert P^*\rvert$	Extreme point distance *max speedup*	Extreme point distance *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)

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*



(a) Perlin Noise (b) Molecular Dynamics(MD) (c) K-means (d) Median Filter

(e) Convolution (f) Blackscholes (g) Mersenne Twister(MT) (h) Flte

(i) Matrix-multiply (j) Bit Compression (k) AES (l) *k*-NN

Figure 8: 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.

(HV) indicator [31], 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 [30], 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 Fig. 8 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 Fig. 6.

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 rational 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 reflect 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.

5 CONCLUSION

This paper introduces a modeling approach aimed at predicting the best memory and core frequency settings for an OpenCL application on GPUs. The proposed framework is based on a two-phase machine learning approach: first, the model is built during a training phase performed on a set of synthetic benchmarks; later, the model is used for predicting the best frequency settings 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 raising interest in multi-objective problems involving energy efficiency, approximate computing, and space optimization.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Yuki Abe, Hiroshi Sasaki, Shinpei Kato, Koji Inoue, Masato Edahiro, and Martin Peres. 2014. Power and Performance Characterization and Modeling of GPU-Accelerated Systems. In *IEEE 28th International Parallel and Distributed Processing Symposium*. 113–122.

[2] Arnamoy Bhattacharyya, Grzegorz Kwasniewski, and Torsten Hoefler. 2015. Using Compiler Techniques to Improve Automatic Performance Modeling. In *International Conference on Parallel Architecture and Compilation*.

[3] JeeWhan Choi and Richard W. Vuduc. 2016. Analyzing the Energy Efficiency of the Fast Multipole Method Using a DVFS-Aware Energy Model. In *IEEE International Parallel and Distributed Processing Symposium Workshops*. 79–88.

[4] Biagio Cosenza, Juan J. Durillo, Stefano Ermon, and Ben H. H. Juurlink. 2017. Autotuning Stencil Computations with Structural Ordinal Regression Learning. In *IEEE International Parallel and Distributed Processing Symposium, IPDPS*. 287–296.

[5] Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. 2017. Synthesizing benchmarks for predictive modeling. In *International Symposium on Code Generation and Optimization, CGO*. 86–99.

[6] Frédéric de Mesmay, Yevgen Voronenko, and Markus Püschel. 2010. Offline library adaptation using automatically generated heuristics. In *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS*.

[7] Sam Van den Steen, Stijn Eyerman, Sander De Pestel, Moncef Mechri, Trevor E. Carlson, David Black-Schaffer, Erik Hagersten, and Lieven Eeckhout. 2016. Analytical Processor Performance and Power Modeling Using Micro-Architecture Independent Characteristics. *IEEE Trans. Computers* (2016).

[8] Rong Ge, Xizhou Feng, and Kirk W. Cameron. 2009. Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS*. 1–8.

[9] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. 2013. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. In *42nd International Conference on Parallel Processing, ICPP*.

[10] Dominik Grewe and Michael F. P. O'Boyle. 2011. A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL. In *20th International Conference on Compiler Construction , CC*. 286–305.

[11] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. 2018. GPGPU Power Modelling for Multi-Domain Voltage-Frequency Scaling. In *24th IEEE International Symposium on High-Performance Computing Architecture, HPCA*.

[12] Tomoaki Hamano, Toshio Endo, and Satoshi Matsuoka. 2009. Power-aware dynamic task scheduling for heterogeneous accelerated clusters. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS*. 1–8.

[13] Greg Harris, Anand V. Panangadan, and Viktor K. Prasanna. 2016. GPU-Accelerated Parameter Optimization for Classification Rule Learning. In *International Florida Artificial Intelligence Research Society Conference, FLAIRS*. 436–441.

[14] Canturk Isci and Margaret Martonosi. 2003. 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)*.

[15] Shailendra Jain, Surhud Khare, Satish Yada, V. Ambili, Praveen Salihundam, Shiva Ramani, Sriram Muthukumar, M. Srinivasan, Arun Kumar, Shasi Kumar, Rajaraman Ramanarayanan, Vasantha Erraguntla, Jason Howard, Sriram R. Vangal, Saurabh Dighe, Gregory Ruhl, Paolo A. Aseron, Howard Wilson, Nitin Borkar, Vivek De, and Shekhar Borkar. 2012. A 280mV-to-1.2V wide-operating-range IA-32 processor in 32nm CMOS. In *IEEE International Solid-State Circuits Conference, ISSCC*. 66–68.

[16] Klaus Kofler, Ivan Grasso, Biagio Cosenza, and Thomas Fahringer. 2013. An automatic input-sensitive approach for heterogeneous task partitioning. In *International Conference on Supercomputing, ICS'13*. 149–160.

[17] Joo Hwan Lee, Nimit Nigania, Hyesoon Kim, Kaushik Patel, and Hyojong Kim. 2015. OpenCL Performance Evaluation on Modern Multicore CPUs. *Scientific Programming* (2015).

[18] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. 2015. Many-Objective Evolutionary Algorithms: A Survey. *ACM Comput. Surv.* 48, 1, Article 13 (Sept. 2015), 35 pages.

[19] Andre Lopes, Frederico Pratas, Leonel Sousa, and Aleksandar Ilic. 2017. Exploring GPU performance, power and energy-efficiency bounds with Cache-aware Roofline Modeling. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS*. 259–268.

[20] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In *41st International Conference on Parallel Processing, ICPP*. 48–57.

[21] Xinxin Mei, Ling Sing Yung, Kaiyong Zhao, and Xiaowen Chu. 2013. A Measurement Study of GPU DVFS on Energy Conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*. Article 10, 5 pages.

[22] NVIDIA. [n. d.]. NVIDIA Management Library (NVML). ([n. d.]). https://developer.nvidia.com/nvidia-management-library-nvml

[23] Sankaralingam Panneerselvam and Michael M. Swift. 2016. Rinnegan: Efficient Resource Use in Heterogeneous Architectures. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT*.

[24] Jie Shen, Jianbin Fang, Henk J. Sips, and Ana Lucia Varbanescu. 2013. An application-centric evaluation of OpenCL on multi-core CPUs. *Parallel Comput.* 39, 12 (2013), 834–850.

[25] Alexander J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and Computing* 14, 3 (2004), 199–222.

[26] Seokwoo Song, Minseok Lee, John Kim, Woong Seo, Yeon-Gon Cho, and Soojung Ryu. 2014. Energy-efficient scheduling for memory-intensive GPGPU workloads. In *Design, Automation & Test in Europe Conference & Exhibition*. 1–6.

[27] Ananta Tiwari, Michael Laurenzano, Joshua Peraza, Laura Carrington, and Allan Snavely. 2012. Green Queue: Customized Large-Scale Clock Frequency Scaling. In *International Conference on Cloud and Green Computing, CGC*.

[28] Qiang Wang and Xiaowen Chu. 2018. GPGPU Performance Estimation with Core and Memory Frequency Scaling. In *24th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2018, Singapore, December 11-13, 2018*. 417–424. https://doi.org/10.1109/PADSW.2018.8645000

[29] Gene Y. Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU performance and power estimation using machine learning. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015*. 564–576.

[30] Eckart Zitzler. 1999. *Evolutionary algorithms for multiobjective optimization: methods and applications*. Ph.D. Dissertation. University of Zurich, Zürich, Switzerland.

[31] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. 2003. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *Trans. Evol. Comp* 7, 2 (April 2003), 117–132.

[32] Marcela Zuluaga, Andreas Krause, and Markus Püschel. 2016. e-PAL: An Active Learning Approach to the Multi-Objective Optimization Problem. *Journal of Machine Learning Research* 17 (2016), 104:1–104:32.