IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 18, 2019, accepted October 30, 2019, date of publication November 4, 2019, date of current version November 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2951218

GPU Static Modeling Using PTX and Deep Structured Learning

JOÃO GUERREIRO, (Student Member, IEEE), ALEKSANDAR ILIC, (Member, IEEE), NUNO ROMA, (Senior Member, IEEE), AND PEDRO TOMÁS, (Senior Member, IEEE)
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, 1000-029 Lisbon, Portugal

Corresponding author: João Guerreiro (joao.guerreiro@inesc-id.pt)

This work was supported in part by the national funds through Fundação para a Ciência e a Tecnologia (FCT), under Grant SFRH/BD/101457/2014, and in part by the projects UID/CEC/50021/2019, PTDC/EEI-HAC/30485/2017, and PTDC/CCI-COM/31901/2017.

ABSTRACT In the quest for exascale computing, energy-efficiency is a fundamental goal in high-performance computing systems, typically achieved via dynamic voltage and frequency scaling (DVFS). However, this type of mechanism relies on having accurate methods of predicting the performance and power/energy consumption of such systems. Unlike previous works in the literature, this research focuses on creating novel GPU predictive models that do not require run-time information from the applications. The proposed models, implemented using recurrent neural networks, take into account the sequence of GPU assembly instructions (PTX) and can accurately predict changes in the execution time, power and energy consumption of applications when the frequencies of different GPU domains (core and memory) are scaled. Validated with 24 applications on GPUs from different NVIDIA microarchitectures (Turing, Volta, Pascal and Maxwell), the proposed models attain a significant accuracy. Particularly, the obtained power consumption scaling model provides an average error rate of 7.9% (Tesla T4), 6.7% (Titan V), 5.9% (Titan Xp) and 5.4% (GTX Titan X), which is comparable to *state-of-the-art* run-time counter-based models. When using the models to select the minimum-energy frequency configuration, significant energy savings can be attained: 8.0% (Tesla T4), 6.0% (Titan V), 29.0% (Titan Xp) and 11.5% (GTX Titan X).

INDEX TERMS GPU, DVFS, modeling, scaling-factors, energy savings.

I. INTRODUCTION

Over the past decade, the high-performance computing (HPC) area has observed a noticeable upsurge in the utilization of accelerators, more specifically graphics processing units (GPUs). The energy efficiency of these devices can have a large impact on the total cost of large-scale computer clusters. As an example, the Summit supercomputer (number one system of June'2019 Top500 list [1]), uses a total of 27 648 NVIDIA Volta GPUs to achieve a peak performance of almost 200 petaflops. For that, it requires a power supply of 13 million watts, which corresponds to an estimated cost of 17 million dollars per year (on power supply alone) [2]. The magnitude of such values highlights the importance of effective mechanisms to maximize the energy efficiency of these systems, as a mere 5% decrease in the energy consumption could generate savings of around 1 million dollars.

One example of such mechanisms is the dynamic voltage and frequency scaling (DVFS), which allows placing devices into lower performance/power states. When carefully applied to match the needs of the executing applications, DVFS can lead to significant power and energy savings, sometimes with minimum impact on performance [3], [4]. A recent study showed that using DVFS techniques in GPUs executing deep neural networks applications can provide energy savings up to 23% during training and 26% during inference phases [5].

However, an efficient use of energy management techniques, such as DVFS, requires accurate models that can predict how the energy consumption changes with the GPU operating frequencies (and voltages). This type of modeling is often done by separately modeling the performance and the power consumption of the GPU, focusing on how each one separately scales with DVFS [6], [7]. On the other hand, several previous works have shown that the performance/power

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

behavior of GPU applications considerably vary with the application characteristics [8], [9], which makes these predictive models to require some information from the application to provide accurate predictions.

When performing DVFS management, the run-time utilization of the GPU components[1] can be used in the implementation of predictive models, significantly reducing the overall search space of available voltage-frequency (V-F) configurations. However, to obtain those utilizations levels, most of the existing GPU modeling approaches require (at least) one execution of the application. An alternative and highly promising approach consists in providing predictions of the DVFS impact on the application behavior, prior to its execution. This alternative relies on using the GPU assembly of the kernels[2] [10]–[12] (described in the NVIDIA PTX ISA [13]), which can be obtained at compile-time. Although this approach is expected to yield less accurate results (when compared with *state-of-the-art* run-time models), it allows the first execution of an application to be done at a close to the optimal V-F configuration. Additionally, new usage scenarios occur from this type of static modeling, such as allowing programmers to easily evaluate how changes in the source code can affect the DVFS behavior of applications.

Accordingly, the goal of the herein proposed work is to provide accurate predictions on how the GPU execution time, power and energy consumptions of applications scale when DVFS is applied, without requiring their execution. To that end, the proposed methodology uses the PTX assembly code given by the compiler. However, unlike previous works that simply rely on general code statistics, such as the histogram of instructions in the PTX code [12], [14], the proposed approach takes a step further and considers the specific sequence of kernel instructions, to improve the prediction accuracy. To model how the pattern of instructions stresses the GPU components, thus contributing to different performance, power and energy scalings, a deep neural network is used. In particular, the proposed research leverages the recent advances in deep neural networks, particularly in the field of natural language processing (NLP), by using a recurrent encoder architecture, based on Long Short-Term Memory (LSTM) blocks.

The proposed models were extensively validated on four different GPU devices (Tesla T4, Titan V, Titan Xp and GTX Titan X) from the four most recent NVIDIA GPU microarchitectures (Turing, Volta, Pascal and Maxwell). To assess the accuracy of the trained models, a collection of 24 benchmarks (not used in model training) was considered. These benchmarks were selected from five commonly used suites (CUDA SDK [15], Parboil [16], Polybench [17], Rodinia [18] and SHOC [19]). The obtained results show that the proposed models are able to provide accurate predictions. In particular, the power consumption scaling model provides average errors of 7.9% (Tesla T4), 6.7% (Titan V), 5.9% (Titan Xp) and 5.4% (GTX Titan X), which is comparable to the accuracy achieved by run-time counter-based models. Furthermore, when the proposed energy scaling model is used to select the minimum-energy V-F, it allows achieving considerable energy savings of 8.0% (Tesla T4), 6.0% (Titan V), 29.0% (Titan Xp) and 11.5% (GTX Titan X).

Accordingly, the most significant contributions of this paper are the following:

- Novel deep learning network, implemented using recurrent neural blocks (LSTMs), which takes the sequence of PTX assembly code of a GPU kernel and encodes it into a latent space representation that characterizes how the kernel utilizes de GPU components.
- Three new GPU predictive models: *i)* performance scaling, *ii)* power consumption scaling, and *iii)* energy consumption scaling, which can predict how the executing time, power consumption and energy consumption of an application changes for different V-F configurations, based solely on the application PTX assembly code (*i.e.*, no execution of the application is required).
- Validation of the proposed models with 24 applications (not used during training), by comparing the predicted performance/power/energy scaling-factors with the measured values on four different GPU devices (including one from the most recent NVIDIA Turing microarchitecture). The models can be used in different ways, out of which two are analysed, namely, for finding the minimum energy V-F configuration and for finding the set of Pareto-optimal configurations.

The complete source code of this framework is publicly available online at `https://github.com/hpc-ulisboa/gpuPTXModel`.

II. BACKGROUND AND MOTIVATION

A. PARALLEL THREAD EXECUTION (PTX) LANGUAGE

The NVIDIA Parallel Thread Execution (PTX) [13] is an intermediate assembly language for NVIDIA GPUs, where the program has not yet been fully assembled into the device-specific code. NVIDIA provides the PTX instruction set architecture (ISA) for the different generations of GPU devices [13] and the *nvcc* compiler can be used to obtain the PTX code (in plain-text) from CUDA programs.

This work leverages the PTX code as an effective way of characterizing a given GPU application. Its adoption (in favor of higher level CUDA code) was decided because it is more specific to the GPU hardware, allowing a better modeling of the device. From the PTX code of an application, it is generally possible to make the connection between each instruction and the GPU resource that is exercised during its execution. Hence, the proposed approach uses the PTX code to infer a information similar to the one obtained from hardware counters (as used in previous GPU modeling works).

[1]Utilization: the ratio of the amount of time the unit is active over the total kernel execution time.

[2]Kernel: routine to be executed in a massively parallel fashion on a GPU, by multiple threads.

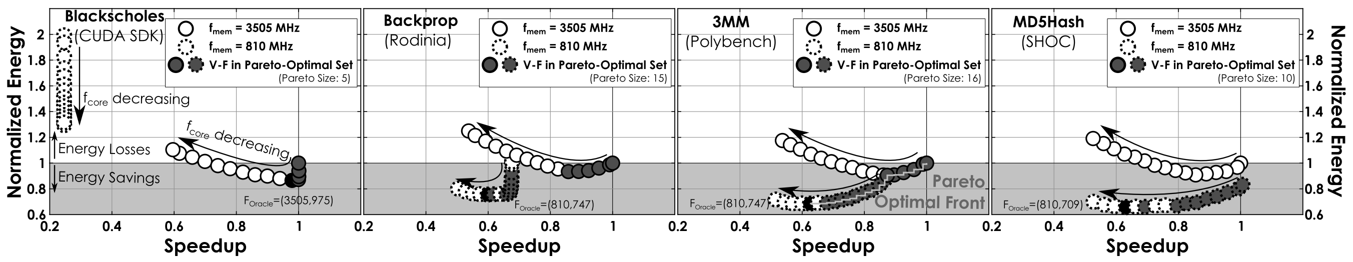


FIGURE 1. **DVFS impact on the energy scaling and speedup of four applications executed on the GTX Titan X. Each point corresponds to a specific V-F. Values are normalized to the values obtained at the highest V-F (maximum performance).**

B. DVFS IMPACT ON APPLICATION BEHAVIOR

Modern GPU devices have two independent frequency domains: the *core* (or graphics) *domain*, clocked at f_{core}, and the *memory domain*, clocked at f_{mem}. Each frequency domain is associated with a specific voltage level. The existence of these independent domains allows adapting their voltage and frequency to the specific requirements of the executing application — this is called **DVFS**. Depending on how a given application exercises the GPU resources, DVFS can have vastly different effects its performance/power consumption, which can in some cases result in considerable energy savings [3], [4], [20]–[22]. However, finding the best voltage-frequency (V-F) levels for a given application is not a trivial task [4], [9], [23]. To properly apply DVFS without harming the execution of an application, one must be able to accurately predict how the changes to V-F can affect the application behavior (execution time, power and energy consumption).

Fig. 1 presents an example of the execution of four applications on the GTX Titan X GPU (NVIDIA Maxwell microarchitecture). Each subplot presents the variation of the application energy and performance normalized to the values at the highest V-F configuration, which in this case is $\mathbf{F_{max}} = (f_{mem} = 3505$ MHz, $f_{core} = 1164$ MHz). Each point corresponds to a different V-F, out of the considered 32 (2 memory levels, 3505 MHz and 810 MHz, and 16 core levels in the range [595, 1164] MHz).

From this figure, it can be seen that the unique application characteristics (used algorithm, data types, operations, size of the input data, dimensions of the grid of threads, *etc.*), lead to vastly different behaviors. For example, in the *MD5Hash* benchmark, decreasing the memory frequency from 3505 MHz to 810 MHz has a negligible impact on performance (speedup does not change between the different f_{mem} levels), indicating that this application is not memory-intensive. On the other hand, the *BlackScholes* benchmark has a significant drop in speedup when the memory frequency is changed to the lowest level, indicating that this application is very memory-intensive. In fact, for the *BlackScholes*, once the memory is set to the lowest frequency, any changes in f_{core}, within the range allowed by the device, does not lead to any further changes in speedup.

These examples also confirm that it is not trivial to find the best V-F configuration. On one hand, the V-F that leads to the minimum energy significantly differs from application to application. On the other hand, the performance degradation associated with that V-F level can also be highly application dependent. When considering the *BlackScholes* and *MD5Hash* benchmarks, it can be seen that the minimum-energy configurations ($\mathbf{F_{Oracle}}$) are (3505,975) and (810,709), for the two applications respectively. In the case of the *BlackScholes* benchmark, this leads to energy savings of 13.5% (*vs.* $\mathbf{F_{Max}}$) at a cost of only 2.2% drop in performance. However, for the *MD5Hash* benchmark, using its corresponding $\mathbf{F_{Oracle}}$ leads to much higher energy savings (34.2%), at a much higher performance cost (37% performance drop-off).

Considering that sometimes such performance drop-offs cannot be tolerated, looking for the minimum-energy V-F may not always be the best option. An alternative approach, as suggested by Fan *et. al.* [12], is to consider a multi-objective optimization problem, with a set of Pareto-optimal solutions. In other words, one could search for the V-F configurations that maximize the speedup and minimize the normalized energy, *i.e.*, the configurations that are not dominated by any other configuration. In this case, not being dominated in performance means that for the same energy, there are no frequencies that lead to higher performance levels (higher speedup). On the other hand, not being dominated in energy means that for the same speedup, there are no frequencies that lead to a lower normalized energy. The set of Pareto-optimal V-Fs for a given application can be found by iterating between all available configurations and seeing if it is dominated by any other configuration.

Fig. 1 shows the Pareto-optimal set for the four considered example applications. As one might expect, not only do the configurations in each Pareto-optimal set depend on the application, but also the size of the set can vastly differ. The most memory intensive benchmark (*BlackScholes*) has five V-F configurations in the Pareto-optimal set (all with $f_{mem} = 3505$ MHz), while the most compute intensive one (*MD5Hash*) has 10 Pareto-optimal V-Fs (all with $f_{mem} = 810$ MHz). Between these two extremes are the *Backprop* and *3MM* benchmarks, with 15 and 16 Pareto-optimal V-Fs, respectively (split across the two available memory levels).

These observations highlight the importance of accurate DVFS-aware performance/power/energy models, since no matter the goal (*e.g.*, finding minimum-energy V-F *vs.* Pareto-optimal V-F set), it is imperative to be able to characterize how these three metrics (performance/power/energy) change with the V-F of the GPU domains.

C. RELATED WORK

There have been many research works that have addressed the goal of improving the energy efficiency of GPU devices [21], [24]–[32]. In particular, the work of Hong and Kim [33] was one of the first to accurately characterize the performance of GPU applications. They also proposed a power model for a GTX280 GPU (Tesla microarchitecture) based on an analysis of both the binary PTX and the device pipeline at run-time [34], attaining highly accurate predictions. However, the trained model could not be replicated at different core or memory configurations. Leng *et al.* [35] integrated Hong's power model inside the GPGPU-Sim [36] simulator, creating the GPUWattch tool, which can estimate the cycle-level GPU power consumption during application execution (with support for the Tesla and Fermi microarchitectures). Furthermore, cycle-level simulators are too slow to be applied in run-time and predict the optimal V-Fs.

Nath and Tullsen [37] developed a run-time analytical performance model able to predict the performance changes with GPU DVFS, with an average prediction error of 4%. However, the proposed approach requires the addition of extra logic to the GPU scoreboard, making it infeasible to be replicated on real hardware. Alternatively, statistical models can be developed using performance counters already available on the GPUs, as it was done in the work by Wu *et al.* [38], which studied how the performance and power consumption of AMD GPUs scale with the core and memory frequencies. The proposed approach groups applications based on their performance/power *scaling-factors*. Properly trained neural network classifiers are used to characterize new applications, by predicting which scaling-factor better represents an application. The proposed approach achieves average prediction errors of 15% (performance) and 10% (power) on the considered AMD Radeon HD 7970 GPU.

Guerreiro *et al.* [7] proposed a DVFS-aware GPU power consumption model, which predicts the GPU power consumption for any V-F configuration, by using performance counters gathered at a single configuration. To estimate the model of each GPU device, the authors devised a suite of publicly available microbenchmarks. The model was validated on three GPUs, achieving average errors of 7% (Titan Xp), 6% (GTX Titan X) and 12% (Tesla K40c). This model was later extended, by focusing on its different use-cases [39] (*e.g.*, using the proposed model to predict only the *scaling-factors* of the GPU power). This largely improved the predictions accuracy, leading to average errors of 3.5% (Titan Xp), 4.5% (GTX Titan X) and 2.4% (Tesla K40c).

By following a similar approach Wang *et al.* [6] proposed a DVFS-aware GPU performance model. The authors estimated the GPU architecture parameters using a collection of microbenchmarks and a group a performance counters, measured during their execution. Validated across a wide range of core and memory frequencies, on a Maxwell GPU, the model attains an average prediction error of 4.83%.

More recently, some works have started to tackle the topic of GPU static analysis, specifically regarding modeling based on the PTX code of a kernel. Arafa *et al.* [11] presented a static GPU performance model. The authors converted the PTX code to a list of tasks with known modeled behavior (through microbenchmarking done by Andersch *et al.* [40]), achieving prediction errors within 10% of the measured performance. However, the work does not consider DVFS.

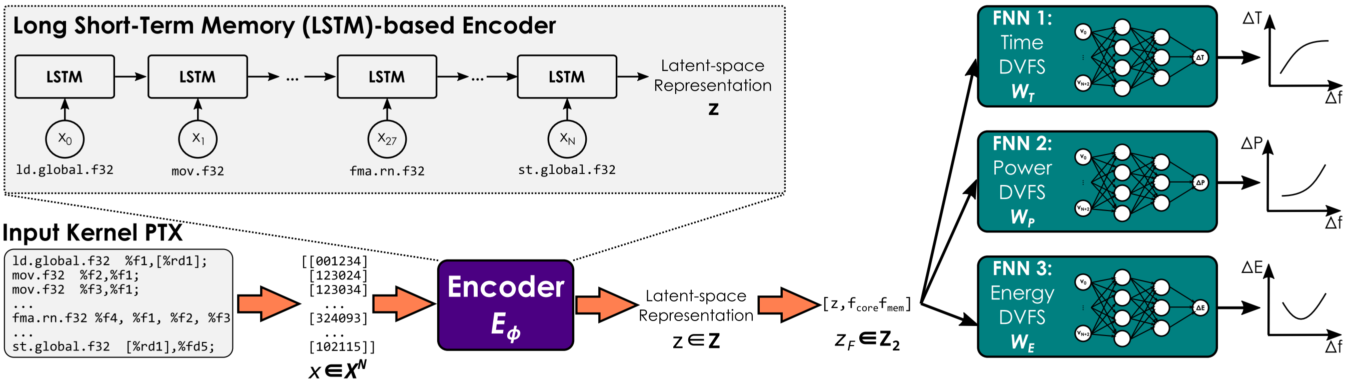
Other static models have been proposed that consider DVFS. In particular, Alavani *et al.* [10] presented a way to predict the execution time of an application prior to its execution, with an average prediction error of 26.9% on a Tesla K20 GPU (Kepler). On the other hand, Fan *et al.* [12] developed DVFS-aware static models for performance and energy of GPU devices. The two models are trained based on a static vector of 10 features, where each component represents the count of a type of instructions. As previously mentioned, these authors suggest that the best V-F configuration for a given application, should not be looked for in the minimum-energy points, but in a set of Pareto-optimal V-Fs. Validated on a GTX Titan X (Maxwell), with 12 benchmarks, the proposed models can predict most of the frequencies in the Pareto-optimal sets, and are able to predict the minimum-energy V-F for two applications (out of 12).

Arunkumar *et al.* [14] addressed the topic of multi-module GPUs. In their work, the authors propose an instruction-based energy estimation framework, which is able to modulate its corresponding energy-per-instruction value, for the different types of PTX instructions. Similarly to what is herein done, the authors also have to instrument the CUDA code of considered applications, in order to obtain PTX code that represents the number of executed instructions.

As it can be seen, the majority of the research that has been performed on GPU modeling requires at least one execution of the application to obtain predictions. On the other hand, the works that adopt static modeling techniques only consider the count of each type of instructions, without any consideration to the sequence/order they are executed in. Furthermore, these works have not been properly validated on recent GPU architectures (the work targeting the most recent GPU focuses on a NVIDIA Maxwell GPU, which is from early 2015). In contrast, the herein presented work proposes a different approach to predict the scaling behavior of the performance, power and energy consumption of an application before its execution. Using a recurrent neural network (with LSTM blocks), this new approach considers the sequence of PTX instructions. This approach is validated on multiple GPUs across four different NVIDIA microarchitectures, including the most recent Turing generation.

III. PTX-BASED MODELING

The proposed model is based on the rationale that both the GPU performance and power consumption depend on which



```
ld.global.f32  %f1,[%rd1];
mov.f32  %f2,%f1;
mov.f32  %f3,%f1;
...
fma.rn.f32 %f4, %f1, %f2, %f3
...
st.global.f32  [%rd1],%fd5;
```

```
[[001234]
 [123024]
 [123034]
   ...
 [324093]
   ...
 [102115]]
```

FIGURE 2. **Diagram of the proposed PTX-based characterization models.**

GPU components are utilized during the execution of applications [9]. The utilization of a specific component is not only dependent on the total number of instructions executed by the component and its ratio over the other types of instructions, but also on the order these instructions are executed on. Hence, the proposed modeling framework takes into account not only the different types of instructions of a given application, but also their corresponding sequence. The goal of this framework is to output simultaneously the scaling-factors for three different metrics (execution time, power and energy) at the different V-F configurations (*vs.* the selected reference configuration).

A. DEEP STRUCTURED STATIC MODELING

The modeling methodology proposed in this work, presented in Fig. 2, can be divided into two main learning blocks: *i)* a recurrent neural network (RNN), and *ii)* three output fully connected feedforward neural networks (FNNs). To represent each instruction, an embedding step is used, which encodes not only the PTX instruction, but also the PTX instruction modifiers, the number of operands used, the existence of operand dependencies and the type of dependency, namely if the operand was produced by a previous memory or compute instruction. Hence, the RNN works as an LSTM-based encoder, taking as input the sequence of embeddings (x) of a kernel and providing as output a latent space vector that encodes that kernel (z) and, particularly, the way it utilizes the GPU components. The second learning block is comprised of three separate, fully connected, FNNs. These networks take as input the latent space representation and are trained to provide, at their outputs, the scaling-factors of the execution time, power consumption or energy consumption for the different V-F configurations.

Formally, a kernel code is represented as a sequence of N instruction embeddings $x = \{x_1, x_2, \ldots, x_N\} \in X^N$, where $x_i \in X$, with X denoting the space of possible instruction embeddings. This sequence is given as input to an LSTM-based encoder $E_\phi : X^N \rightarrow Z$, where $Z = \mathbb{R}^L$ is the latent space and L denotes its dimensionality, extracting a representation $z = E_\phi(x)$. The latent space representation is then appended with the considered frequency configuration $F = (\mathrm{f_{mem}}, \mathrm{f_{core}})$, such that $z_F = [z, F]$, where $z_F \in Z_2$ ($Z_2 = \mathbb{R}^{L+2}$). Finally, the frequency latent space representation is used by the three separate fully connected FNNs $W_{\{T,P,E\}} : \mathbb{R}^{L+2} \rightarrow \mathbb{R}$ to predict the scaling factors of the execution time, power consumption and energy consumption. As an example, and looking at the prediction given by the first output FNN, *i.e.*, the execution time scaling-factor, the whole network is given by:

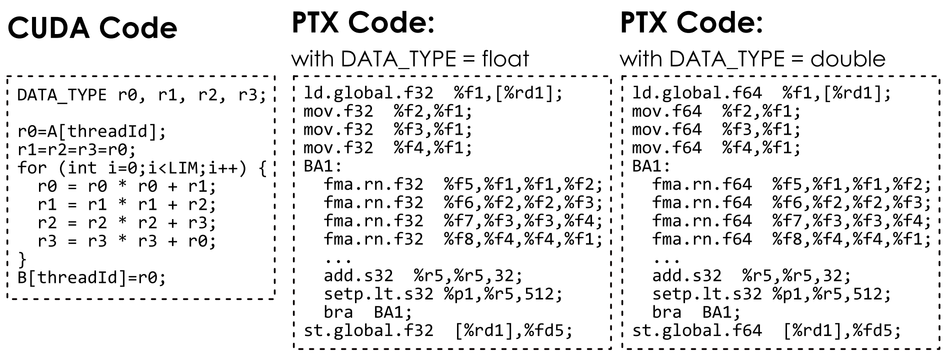
$$\widehat{\Delta T} = W_T(z_F) = W_T(E_\phi(x)). \tag{1}$$

B. EMBEDDINGS

In order to feed the RNN layer with the sequence of PTX instructions, first they have to be encoded into an appropriate format. To that end, an embeddings stage is proposed, which takes into account not only each specific instruction, but also the context of past instructions. In particular, the embeddings takes into account the following information from each specific instruction:

- **instruction name**, from the known list of instructions defined in the PTX ISA (*e.g.*, `ld`, `add`, `fma`, `bra`, *etc.*).
- **state space specifier**, also defined in the PTX ISA, usually associated with memory instructions and corresponding to a specific storage area (*e.g.*, `.local`, `.global`, `.shared`, *etc.*).
- **data type specifier**, which specifies both the considered basic type and the size of the destination operand (*e.g.*, `.u8`, `.f32`, `.f64`, *etc.*).
- **number of operands**, corresponding to the number of register operands used by the instruction (input + destination).

All this relevant information allows the model to clearly identify the GPU components that are used during the execution of each instruction. Additionally, the embedding of each instruction also takes into account the following information based on past instructions, which are relevant to identify



CUDA Code

```
DATA_TYPE r0, r1, r2, r3;

r0=A[threadId];
r1=r2=r3=r0;
for (int i=0;i<LIM;i++) {
  r0 = r0 * r0 + r1;
  r1 = r1 * r1 + r2;
  r2 = r2 * r2 + r3;
  r3 = r3 * r3 + r0;
}
B[threadId]=r0;
```

PTX Code:
with DATA_TYPE = float

```
ld.global.f32  %f1,[%rd1];
mov.f32  %f2,%f1;
mov.f32  %f3,%f1;
mov.f32  %f4,%f1;
BA1:
  fma.rn.f32  %f5,%f1,%f1,%f2;
  fma.rn.f32  %f6,%f2,%f2,%f3;
  fma.rn.f32  %f7,%f3,%f3,%f4;
  fma.rn.f32  %f8,%f4,%f4,%f1;
  ...
  add.s32  %r5,%r5,32;
  setp.lt.s32 %p1,%r5,512;
  bra  BA1;
st.global.f32  [%rd1],%fd5;
```

PTX Code:
with DATA_TYPE = double

```
ld.global.f64  %f1,[%rd1];
mov.f64  %f2,%f1;
mov.f64  %f3,%f1;
mov.f64  %f4,%f1;
BA1:
  fma.rn.f64  %f5,%f1,%f1,%f2;
  fma.rn.f64  %f6,%f2,%f2,%f3;
  fma.rn.f64  %f7,%f3,%f3,%f4;
  fma.rn.f64  %f8,%f4,%f4,%f1;
  ...
  add.s32  %r5,%r5,32;
  setp.lt.s32 %p1,%r5,512;
  bra  BA1;
st.global.f64  [%rd1],%fd5;
```

FIGURE 3. CUDA source code example and the corresponding PTX codes depending on the data type.

dependencies scenarios that can impact both kernel performance and the average power consumption:

- **closest input operand dependency**, corresponding to the closest previous instruction that had as a destination register one of the registers used as input (0 if there are no dependencies).
- **dependency type**, corresponding to the type of dependency (if there is any), namely if it is a dependency to a memory access or computational instruction.

C. TRAINING METHODOLOGY

To train the whole network, a specific procedure is proposed that allows training the three output FNNs at the same time. To this end, the set of training applications is first separated into different batches, organized by kernel length (*e.g.*, batches of 8 applications). At each training epoch, each batch is used to train only one of the output networks. For example, the first batch is propagated forward only through the execution time network (**Encoder** + **FNN1** in Fig. 2). The considered optimization loss is the mean absolute error (MAE) between the predicted values and the measured ones. Then, based on the obtained errors, backpropagation is used to update the weights of both the **Encoder** and **FNN1**. Afterwards, the next batch is propagated forward only through the power consumption network (**Encoder** + **FNN2**), followed again by backpropagation of the errors. This process is repeated for all batches (in each training epoch), interleaving them between the three output FNNs, which are therefore being updated one at a time, while the encoder is always being updated. Finally, the training procedure implements a mechanism that ensures that, across different epochs, the same batch of applications is not always propagated forward to the same FNN.

One particular feature of the proposed training procedure is the fact that, by allowing the three output FNNs to be trained semi-simultaneously, it allows the encoder LSTM to have information on the three target metrics (execution time, power consumption and energy consumption). An alternative approach would be to fully train an encoder + FNN for each of the output metrics or even focus on a single output FNN at each epoch. However, by interleaving the training of the FNNs to each smaller batch of applications, the proposed training procedure tries to ensure that the LSTM encoder is able to learn and generate a more robust latent space, capable of describing how the different GPU components are stressed. Experimental validation confirmed that the proposed approach provides better accuracy than if each network was fully trained separately.

D. MICROBENCHMARKING THE GPU

To model the usage of the different GPU components, a group of publicly available microbenchmarks was used, namely the ones proposed by Guerreiro *et al.* in [7][3] and Arunkumar in [14][4]. The considered benchmarks were carefully selected not only to contain most of the PTX instructions defined in the PTX ISA, but also to include a wide variety of code patterns (different instruction mixes, GPU components utilizations, arithmetic intensities, memory access patterns, *etc.*). Overall, 145 microbenchmarks were used.

Fig. 3 presents an example of a considered benchmark, illustrating its CUDA source and two corresponding PTX codes depending on the defined data type. Specifically, it presents two examples, corresponding to the cases when `DATA_TYPE` is single precision (`float`) and double precision (`double`). It can be seen that the same CUDA instruction `r0 = r0 * r0 + r1` can be converted into different PTX instructions. If `r0` and `r1` are of type `float`, the corresponding PTX instruction is `fma.rn.f32 %f5,%f1,%f1,%f2`, while if they are of type `double` the corresponding instruction is `fma.rn.f64 %f5,%f1,%f1,%f2`. Hence, depending on the data type, the same instruction is issued to different computational units (32-bit floating-point or 64-bit floating-point, respectively).

Similarly, load (or store) instructions (*e.g.* `r0 = A[threadId]`) can also map to different PTX instructions depending on the data source (or destination). In Fig. 3, data is loaded and stored back to global memory, hence

[3]`https://github.com/hpc-ulisboa/gpupowermodel`

[4]`https://github.com/akhilarunkumar/GPUJoule_release`

the `.global` modifier in the PTX `ld` and `st` instructions. However, if, for example, the data had been written to shared memory, they would use instead the `.shared` modifier. Since the shared and DRAM memories have very distinct characteristics (different latencies and peak power consumptions) and are even clocked at different operating frequencies, this modifier is crucial to the proposed model and is therefore taken into account in the proposed embeddings.

As an example of how the proposed embeddings can be extracted from a PTX instruction, looking at the `fma.rn.f32 %f5,%f1,%f1,%f2` instruction (from the `float` example presented in Fig. 3). The information considered in the embeddings is: *i)* **instruction name** (`fma`); *ii)* **state space specifier** (none); *ii)* **instruction type specifier** (`.f32`); *iii)* **number of operands** (3 input and 1 destination); *iv)* **closest input operand dependency** (`%f2` was written 3 instructions before); *v)* **dependency type** (`%f2` was changed by a memory instruction).

As it can be seen it is generally possible to infer from the PTX code of an application which GPU resources will be exercised during its execution. However, it is important to stress that, unlike previous approaches, the proposed strategy does not rely on any information obtained from the applications execution. Nevertheless, it is also important to mention that there are some inherent limitations to using the PTX code. For example, the *nvcc* compiler performs several code optimizations before creating the lower-level code. One of such optimizations is the unrolling of `for` loops a specific number of times (usually 32, provided that the size of the loop is greater than 32). After those 32 repetitions of the main loop instructions, a branch instruction is placed to redirect the program execution back to the beginning of the loop (`bra BA1` in Fig. 3). The number of times that this jump is taken depends on the limit of the `for` loop (the value of `LIM` in Fig. 3). This means that two applications with the same kernel code, but different values of `LIM`, for example, `LIM=64` and `LIM=2048`, can have the same optimized PTX code, despite the actual number of executed instructions being rather different. To further complicate matters, the number of times the loop is cycled through can be dependent on the data size, which can even be defined only at run-time. For this reason, all the considered microbenchmark kernels (and the applications later used to test the trained models) have their loops manually unrolled, ensuring that the sequence of instructions in the PTX code is the same as the sequence of executed instructions.

Another potential limitation inherent to any static analysis approach regards the global memory accesses. For example, even though load (or store) operations can have the `.global` modifier, it is unknown where exactly the data is coming from. Depending on the data access pattern, the same instruction can result in data transfers from different hierarchy levels (*e.g.*, L1 cache, L2 cache or main DRAM). Since these memory elements have very different characteristics, they can have completely distinct impacts on both performance and power consumption, despite the initial PTX instruction

TABLE 1. Summarized description of the used GPUs.

		Tesla T4	Titan V	Titan Xp	GTX Titan X
Base architecture		Turing	Volta	Pascal	Maxwell
Compute capability		7.5	7.0	6.1	5.2
Memory frequencies (MHz)		5001	850	{4705, 5705}	{810, 3505}
Core freq. range (MHz)		[300:1590]	[135:1402]	[582:1911]	[595:1164]
Default mem. frequency		5001	850	5705	3505
Default core frequency		585	1200	1404	975
Number of SMs		40	80	30	24
Per SM	**CUDA cores** (SP/INT units)	64	64	128	128
	DP units	2	32	4	4
	SF units	32	32	32	32
	Shared memory (bytes)	up to 96K	up to 96K	96K	96K
L2-cache size (bytes)		4M	4.5M	3M	3M
Global memory size (bytes)		16G	12G	12G	12G
Memory bus width (bits)		256	3072	384	384
Memory bus type		GDDR6	HMB2	GDDR5X	GDDR5
TDP (W)		70	250	250	250

being the same. It is left for future work to infer more information on the access patterns of the PTX kernel, in order to improve the kernel characterization and the resulting performance/power/energy prediction accuracy. A compromising approach could be to combine application information obtained statically (PTX) and dynamically (hardware counters, *e.g.*, cache miss ratios), to further improve modeling accuracy. As an example, this could be achieved by combining the latent-space representation with the counter-based characteristics (in a similar way to what is herein done to obtain z_F), which could then be provided to the output FNNs.

IV. EXPERIMENTAL RESULTS

A. EXPERIMENTAL SETUP

The proposed models were validated on four GPUs from the most recent NVIDIA microarchitectures, namely the Turing, Volta, Pascal and Maxwell family of GPUs (summarized in Table 1). Experiments with the Volta, Pascal and Maxwell GPUs were performed on a Linux CentOS 7.5 server, with CUDA 10.0 and NVIDIA driver v410, while the most recent Turing GPU was tested using Google's Cloud Platform [41], using a Debian GNU/Linux 9 server, also with CUDA 10.0 and NVIDIA driver v410.

In order to obtain the PTX source code of each application, the *nvcc* compiler was used with the `-ptx` flag. Additionally, to adjust the version of the PTX ISA to the different generations of the devices, the flag `-gencode=arch=compute_70,code=compute_70` was also used (in this case for the Titan V, which has a compute capability of 7.0). NVIDIA's NVML [42] library was used to change the GPU operating frequencies, as well as to measure the GPU power consumption. The power consumption of each kernel was computed as the average of all samples gathered during the application execution. Each kernel execution was repeated whenever necessary, in order to achieve an execution time of at least 1 second at the fastest GPU configuration ($\mathbf{F_{max}}$, *i.e.*, highest core and memory frequencies), given that some GPU power sensors have a low sampling frequency [7].

To obtain all the required data points to train the proposed models, the set of microbenchmarks (see Section III-D) was executed on each GPU device at the available V-F configurations. During the execution of each application, the execution time and average power consumption were measured. The proposed models predict the scaling-factors of the time/power/energy in relation to a reference level, which was defined as the maximum allowed frequency ($\mathbf{F_{max}}$) of each GPU device (see Table 1). In order to guarantee the accuracy of the presented results, all applications were executed multiple times and the median value was recorded.

Finally, the accuracy of the estimated models was confirmed using 24 benchmarks from a set of widely used benchmark suites, as presented in Table 2. The testing benchmarks were not used to train the models. Fig. 4 presents the mixture of instructions of the considered testing applications (as read from the PTX code), from which the variety of instructions across the different benchmarks can be confirmed.

TABLE 2. Standard benchmarks used for model validation.

Suite	Application Name
CUDA SDK [15]	*Blackscholes*
Parboil [16]	*MRI-Gridding*
Polybench [17]	*2MM, 2DCONV, 3MM, ATAX, BICG, CORR, COVAR, FDTD-2D, GEMM, GESUMMV, GRAMSCHM, MVT, SYRK*
Rodinia [18]	*Backprop, Hotspot*
SHOC [19]	*FFT, MD5Hash, S3D, S3D_double, Sort, Stencil2D, Stencil2D_double*

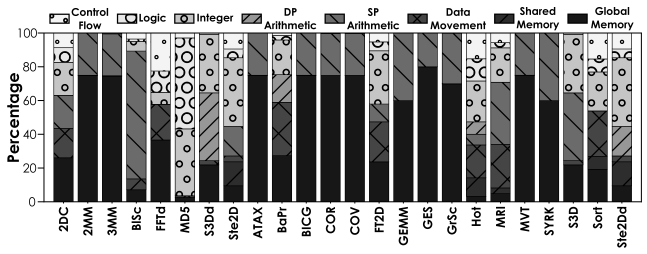


FIGURE 4. Different types of instructions for the testing benchmarks, on the GTX Titan X.

B. MACHINE LEARNING SETUP

The implemented tool, used to obtain the results herein presented, is provided online and is open-source.[5] The machine learning models were implemented using PyTorch (v1.2.0), namely using the `torch.nn.Linear` and `torch.nn.LSTM` functions. To find the best network topology, *i.e.* during the hyperparameter optimization, 90% of the microbenchmark set was used for training and the remaining 10% for validation of each obtained model. The accuracy of each model was determined by the MAE between the predicted value and the measured one (oracle), by using the PyTorch `SmoothL1Loss` function. To maximize the usefulness/ease-of-use of the models, the network topology was optimized for one GPU device (GTX Titan X) and then used for the other GPUs.

The best found hyperparameter configuration for the RNN encoder results in LSTM blocks with 2 layers (unidirectional) of size 50, with a learning rate of 0.005 and using the Adam optimization algorithm. The considered batch size comprises 8 applications. The output FNNs have 2 hidden layers (with sizes 100 and 70), with a learning rate of 0.01 and also using the Adam optimization algorithm. After each hidden layer, an activation function is applied (ReLU).

C. STATIC MODELS ACCURACY

The obtained results show that the accuracy of the predictions obtained using the proposed static models is comparable to the accuracy achieved by the best *state-of-the-art* run-time models. Specifically, the power scaling model predictions, which can be compared with *state-of-the-art* models such as the ones presented in [28], [38], [39], are presented in Fig. 5, across the four considered GPU devices. Each point corresponds to a value of the predicted scaling-factor *vs.* its measured value (oracle), and different points represent different applications and/or V-F configurations. For example, in the GTX Titan X plot, the testing set is composed of 24 applications $\times$ 2 f_{mem} levels $\times$ 16 f_{core} levels $=$ 768 datapoints. In this case, the model is capable of accurately predicting the power consumption scaling-factors, on a frequency range of up to 4.3$\times$ in memory frequency and 2$\times$ in core frequency, with a mean absolute error (MAE) of 5.35%. It is important to restate that these DVFS predictions are made prior to any execution of benchmarks, and are based solely on the PTX kernel code, *i.e.* without using any run-time information. The power scaling model results in similarly accurate results in the three other GPU devices, with MAE of 7.85% (Tesla T4), 6.68% (Titan V) and 5.86% (Titan Xp).

When compared with other *state-of-the-art* power models, in particular with the approach presented in [39], it is observed that the GPU power scaling-factors model of such proposal, based on performance counters, achieves a MAE of 3.54% and 4.55% for the Titan Xp and GTX Titan X, respectively. Even though the approach herein proposed can be slightly less accurate than *state-of-the-art* power consumption counter-based models, it should be stressed that very accurate results can still be obtained based only on information obtained at compile-time (without any application execution).

Fig. 6 presents the overall accuracy of the three output models for all considered GPU devices, by displaying the cumulative error distribution of each model across the testing benchmarks for all V-F configurations. The obtained results show that the power models are (on average) the ones that produce the best predictions, while the execution time models are the less accurate ones. Still, across the four GPUs,

[5]`https://github.com/hpc-ulisboa/gpuPTXModel`

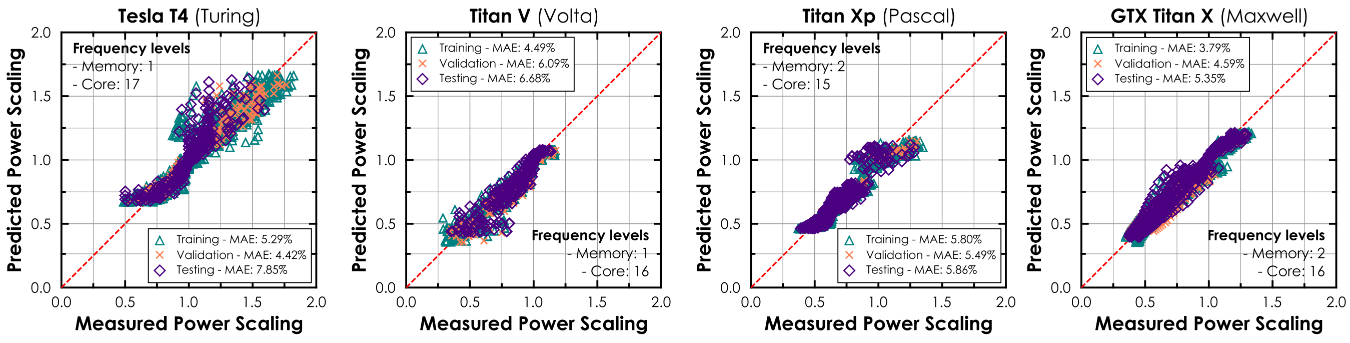
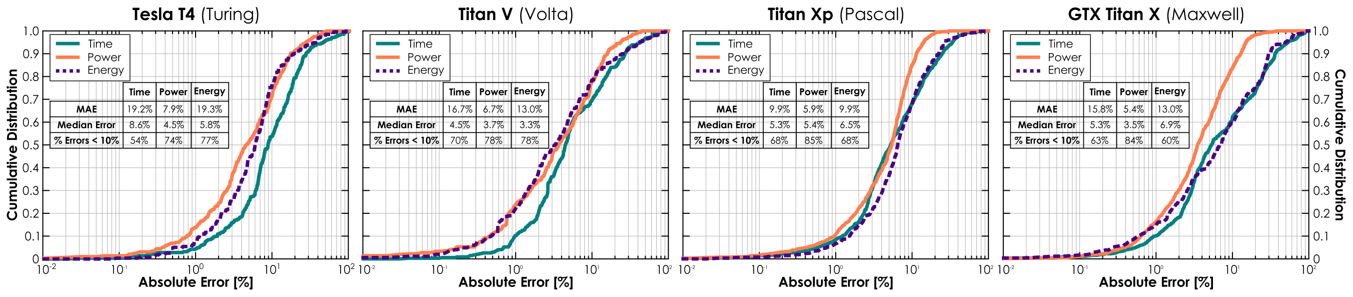


FIGURE 5. Results of the DVFS-aware power scaling model on different GPUs (Number of benchmarks: 130 for training, 15 for validation, 24 for testing).



Tesla T4 (Turing)

	Time	Power	Energy
MAE	19.2%	7.9%	19.3%
Median Error	8.6%	4.5%	5.8%
% Errors < 10%	54%	74%	77%

Titan V (Volta)

	Time	Power	Energy
MAE	16.7%	6.7%	13.0%
Median Error	4.5%	3.7%	3.3%
% Errors < 10%	70%	78%	78%

Titan Xp (Pascal)

	Time	Power	Energy
MAE	9.9%	5.9%	9.9%
Median Error	5.3%	5.4%	6.5%
% Errors < 10%	68%	85%	68%

GTX Titan X (Maxwell)

	Time	Power	Energy
MAE	15.8%	5.4%	13.0%
Median Error	5.3%	3.5%	6.9%
% Errors < 10%	63%	84%	60%

FIGURE 6. Cumulative distribution of the prediction errors on the testing benchmarks, across all V-F configurations.

54% (Tesla T4), 70% (Titan V), 68% (Titan Xp) and 63% (GTX Titan X) of the predicted time scaling-factors have an absolute error below 10%. The percentage of predicted values with an error below 10% for the power model are 74% (Tesla T4), 78% (Titan V), 85% (Titan Xp) and 84% (GTX Titan X), while for the energy model are 77% (Tesla T4), 78% (Titan V), 68% (Titan Xp) and 60% (GTX Titan X).

The accuracy of the 12 estimated models are summarized in Table 3, where it can be seen that the MAE of the energy scaling models are 19.3% (Tesla T4), 13.0% (Titan V), 9.9% (Titan Xp) and 13.0 % (GTX Titan X).

D. PARETO-OPTIMAL SOLUTIONS

The main use-case of predictive models, such as the ones herein proposed, is to perform the DVFS management to maximize the energy efficiency of the computing system. Considering a multi-objective optimization problem with a set of Pareto-optimal solutions, similar to the one that was proposed in [12], this technique can be a useful approach to find the best V-F configurations for different applications. Fig. 7 shows the measured and predicted values of the normalized energy and speedup for 14 different testing applications (not used in training), when considering the GTX Titan X GPU. Each plot not only presents the values of the normalized energy in function of the speedup (measured and predicted), but also which V-F configurations are in the Pareto-optimal sets (measured and predicted) and their respective sizes. Here, it is important to note that not all the considered GPU devices allow a similar flexibility in choosing the V-F configuration. Since the GTX Titan X is the GPU device that allows the larger variation of the memory frequency, it is an interesting situation to analyze.

In Fig. 7, applications were organized from the most memory-intensive (*eg.*, *BlackScholes*, *FDTD-2D*, *S3D_double*) to the most compute-intensive (*eg.*, *ATAX*, *MRI-Gridding*, *MD5Hash*). The obtained results show that the estimated models for the GTX Titan X can predict how the decrease in F_{mem}, from 3505 MHz to 810 MHz, affects the speedup of the more memory-intensive applications. However, as the DRAM intensity starts decreasing, resulting in more transactions coming directly from the L2-cache (*eg.*, *FFT_double*, *2DConvolution*, *Backprop*), the model is gradually not as accurate in describing how the decrease in F_{mem} affects the energy and speedup of applications. This relates to the previously mentioned fact that there is no way to infer from a PTX instruction where a data transaction is coming from (see Section III-D). Finally, for the more compute-intensive applications, the model successfully predicts that decreasing F_{mem} has a small impact on speedup, leading to large benefits in energy consumption.

E. ATTAINED ENERGY SAVINGS

The energy scaling predictions can also be evaluated based on how well they can predict the minimum-energy V-F configuration. Fig. 8 shows the results of using the energy scaling model to find the minimum V-F configuration on the GTX Titan X GPU. On the top of Fig. 8 are presented, for each testing application, the values of the frequency levels associated with the measured minimum energy ($\mathbf{F_{Oracle}}$) and the predicted minimum energy ($\mathbf{F_{Pred.}}$). It can be seen that $\mathbf{F_{Pred.}}$ does not always match $\mathbf{F_{Oracle}}$, which means the proposed model does not guarantee optimal energy savings. However, the results presented at the bottom of Fig. 8 show that, for

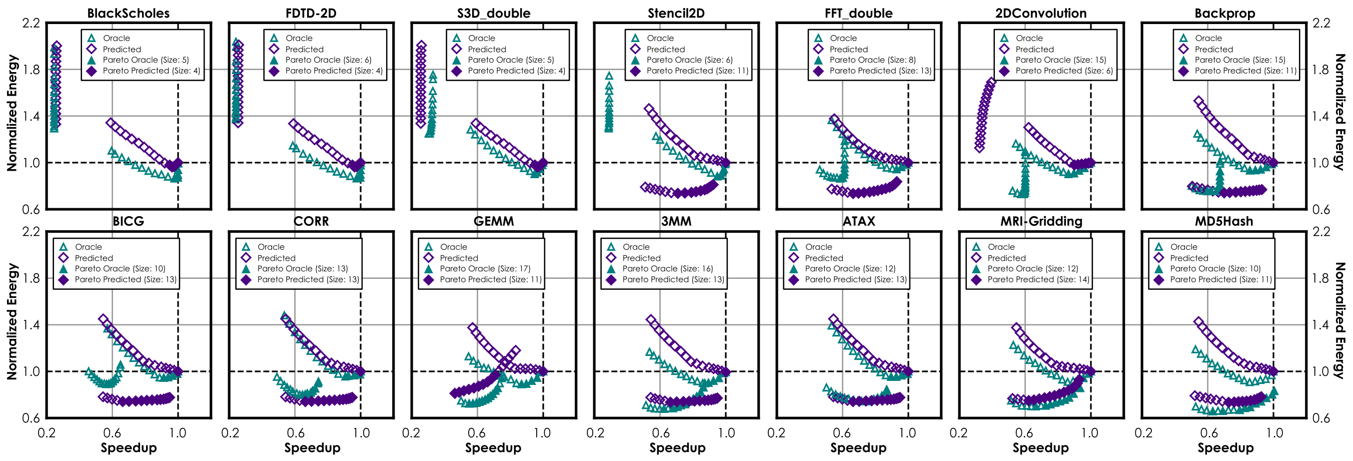


FIGURE 7. Measured and predicted DVFS impact on the energy scaling and speedup of 14 testing applications, on the GTX Titan X. Each point corresponds to a specific V-F configuration. Values are normalized to the values at the highest V-F.

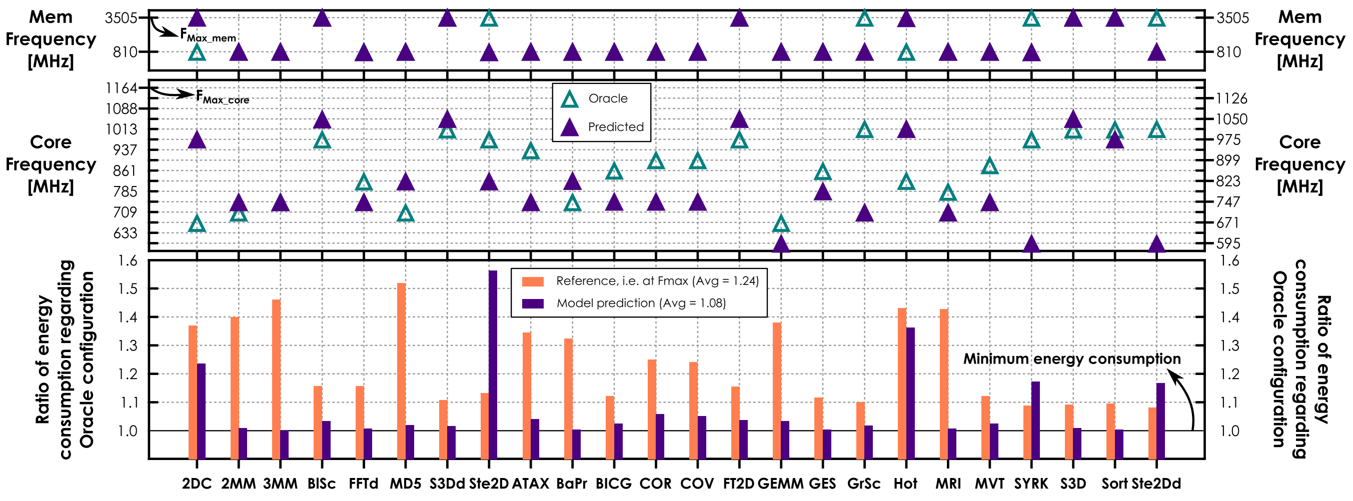


FIGURE 8. Results of the proposed models to predict the minimum energy configuration on the GTX Titan X.

many applications, the difference between using $\mathbf{F}_{\mathbf{Pred.}}$ and $\mathbf{F}_{\mathbf{Oracle}}$ (horizontal line at 1.0, representing the lowest energy consumption) is negligible. In reality, across the 24 testing benchmarks, the usage of $\mathbf{F}_{\mathbf{Pred.}}$ results in an average energy consumption only 8% higher than the optimal, while using the maximum performance configuration ($\mathbf{F}_{\mathbf{Max}}$) leads to (on average) an energy consumption of 24% higher than the optimal. Looking at the energy savings obtained at $\mathbf{F}_{\mathbf{Pred.}}$ (see Table 3), when compared to the energy at $\mathbf{F}_{\mathbf{Max}}$, the proposed energy model allows achieving average savings of 8.0% (Tesla T4), 6.0% (Titan V), 29.0% (Titan Xp) and 11.5% (GTX Titan X).

Overall, the presented research, whose results are summarized in Table 3, represents an important step forward in the field of GPU modeling, by allowing very accurate compile-time predictions of the scaling behavior of the execution time, power consumption and energy consumption. This is a significant improvement over previous DVFS-aware static models, like the one presented in [10], where the performance model had a MAE of 26.9%. Furthermore, this novel approach can be useful in other scenarios than the most commonly considered case of DVFS management, such as in allowing programmers to easily evaluate how changes in the source code can affect the DVFS behavior of applications.

TABLE 3. Summary of the obtained results for the proposed models on the testing benchmarks.

	Model Type	Tesla T4	Titan V	Titan Xp	GTX Titan X
MAE	Time DVFS	19.2%	16.7%	9.9%	15.8%
	Power DVFS	7.9%	6.7%	5.9%	5.4%
	Energy DVFS	19.3%	13.0%	9.9%	13.0%
Median	Time DVFS	8.6%	4.5%	5.3%	5.3%
	Power DVFS	4.5%	3.7%	5.4%	3.5%
	Energy DVFS	5.8%	3.3%	6.5%	6.9%
Energy Savings (*vs.* $\mathbf{F}_{\mathbf{Max}}$)	Best (at $\mathbf{F}_{\mathbf{Oracle}}$)	16.8%	9.1%	32.6%	18.1%
	Model (at $\mathbf{F}_{\mathbf{Pred.}}$)	**8.0%**	**6.0%**	**29.0%**	**11.5%**
	(% of Oracle)	(47.6%)	(65.9%)	(89.0%)	(63.3%)

V. CONCLUSION

This work presented a novel approach to model the GPU performance, power and energy. In particular, the proposed approach can be used to predict how the frequency of GPU domains will affect the execution time, power and energy of applications, before they are actually executed, *i.e.* at compile-time. To model the GPU, a suite of 145 microbenchmarks was used, carefully selected to exercise the different GPU components. The proposed procedure results in three output models that take into account the sequence of low-level assembly (PTX) instructions of any unseen kernel to predict its corresponding scaling behavior. Validated on four different GPU devices from distinct microarchitectures (Turing, Volta, Pascal and Maxwell), the models achieve rather accurate results. In particular, the power scaling one, which is able to accurately predict the DVFS impact on the power consumption of applications prior to their execution, offers a mean absolute error of 10.2% (Tesla T4), 6.7% (Titan V), 5.9% (Titan Xp) and 5.4% (GTX Titan X). Using the obtained models to select the minimum-energy frequency configuration allows achieving (on average) energy savings of 8.0% (Tesla T4), 6.0% (Titan V), 29.0% (Titan Xp) and 11.5% (GTX Titan X).

REFERENCES

[1] *TOP500, Supercomputer Site*. Accessed: Nov. 2019. [Online]. Available: https://www.top500.org

[2] NVIDIA and Oak Ridge National Laboratory. (2018). *Summit GPU Supercomputer Enables Smarter Science*. [Online]. Available: https://devblogs.nvidia.com/summit-gpu-supercomputer-enables-smarter-science/

[3] X. Mei, L. S. Yung, K. Zhao, and X. Chu, "A measurement study of GPU DVFS on energy conservation," in *Proc. Workshop Power Aware Comput. Syst. (HotPower)*, 2013, Art. no. 10.

[4] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "DVFS-aware application classification to improve GPGPUs energy efficiency," *Parallel Comput.*, vol. 83, pp. 93–117, Apr. 2019.

[5] Z. Tang, Y. Wang, Q. Wang, and X. Chu, "The impact of GPU DVFS on the energy and performance of deep learning: An empirical study," in *Proc. Int. Conf. Future Energy Syst. (e-Energy)*, 2019, pp. 315–325.

[6] Q. Wang and X. Chu, "GPGPU performance estimation with core and memory frequency scaling," in *Proc. Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2018, pp. 417–424.

[7] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "GPGPU power modeling for multi-domain voltage-frequency scaling," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 789–800.

[8] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2010, pp. 129–142.

[9] X. Mei, Q. Wang, and X. Chu, "A survey and measurement study of GPU DVFS on energy conservation," *Digit. Commun. Netw.*, vol. 3, no. 2, pp. 89–100, 2017.

[10] G. Alavani, K. Varma, and S. Sarkar, "Predicting execution time of CUDA kernel using static analysis," in *Proc. Int. Conf. Parallel Distrib. Process. Appl., Ubiquitous Comput. Commun., Big Data Cloud Comput., Social Comput. Netw., Sustain. Comput. Commun. (ISPA/IUCC/BDCloud*, Dec. 2018, pp. 948–955.

[11] Y. Arafa, A.-H. A. Badawy, G. Chennupati, N. Santhi, and S. Eidenbenz, "PPT-GPU: Scalable GPU performance modeling," *IEEE Comput. Archit. Lett.*, vol. 18, no. 1, pp. 55–58, Jan./Jun. 2019.

[12] K. Fan, B. Cosenza, and B. Juurlink, "Predictable GPUs frequency scaling for energy and performance," in *Proc. 48th Int. Conf. Parallel Process. (ICPP)*, 2019, Art. no. 52.

[13] NVIDIA. (2019). *Parallel Thread Execution ISA Version 6.4*. [Online]. Available: https://docs.nvidia.com/cuda/parallel-thread-execution/

[14] A. Arunkumar, E. Bolotin, D. Nellans, and C.-J. Wu, "Understanding the future of energy efficiency in multi-module GPUs," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 519–532.

[15] NVIDIA. (2019). *GPU Computing SDK*. [Online]. Available: https://developer.nvidia.com/cuda-code-samples

[16] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, D. Geng, W.-M. Liu, and W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center Reliable High-Perform. Comput.*, vol. 127, Mar. 2012.

[17] L.-N. Pouchet. (2012). *Polybench: The Polyhedral Benchmark Suite*. [Online]. Available: http://www.cs.ucla.edu/~pouchet/software/polybench/

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 44–54.

[19] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suite," in *Proc. 3rd Workshop Gen.-Purpose Comput. Graph. Process. Units (GPGPU)*, 2010, pp. 63–74.

[20] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, Aug. 2007, pp. 38–43.

[21] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang, "GreenGPU: A holistic approach to energy efficiency in GPU-CPU heterogeneous architectures," in *Proc. Int. Conf. Parallel Process. (ICPP)*, Sep. 2012, pp. 48–57.

[22] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, "Effects of dynamic voltage and frequency scaling on a K20 GPU," in *Proc. Int. Conf. Parallel Process. (ICPP)*, Oct. 2013, pp. 826–833.

[23] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding GPU power: A survey of profiling, modeling, and simulation methods," *ACM Comput. Surv.*, vol. 49, no. 3, 2016, Art. no. 41.

[24] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Proc. Int. Green Comput. Conf. Workshops (IGCC)*, Aug. 2010, pp. 115–122.

[25] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU microarchitecture through microbenchmarking," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2010, pp. 235–246.

[26] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in *Proc. 27th Int. Symp. Parallel Distrib. Process. (IPDPS)*, May 2013, pp. 673–686.

[27] A. Sethia and S. Mahlke, "Equalizer: Dynamic tuning of GPU resources for efficient execution," in *Proc. 47th Annu. Int. Symp. Microarchitecture (MICRO)*, Dec. 2014, pp. 647–658.

[28] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, "Power and performance characterization and modeling of GPU-accelerated systems," in *Proc. Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2014, pp. 113–122.

[29] V. Adhinarayanan, B. Subramaniam, and W.-C. Feng, "Online power estimation of graphics processing units," in *Proc. 16th Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, May 2016, pp. 245–254.

[30] X. Mei and X. Chu, "Dissecting GPU memory hierarchy through microbenchmarking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 72–86, Jan. 2017.

[31] G. Chennupati, N. Santhi, S. Eidenbenz, and S. Thulasidasan, "An analytical memory hierarchy model for performance prediction," in *Proc. Winter Simulation Conf. (WSC)*, Dec. 2017, pp. 908–919.

[32] A. Lopes, F. Pratas, L. Sousa, and A. Ilic, "Exploring GPU performance, power and energy-efficiency bounds with cache-aware roofline modeling," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2017, pp. 259–268.

[33] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proc. 36th Int. Symp. Comput. Archit. (ISCA)*, 2009, pp. 152–163.

[34] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2010, pp. 280–289.

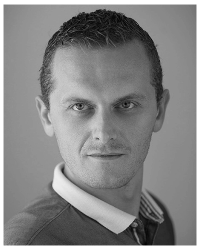
[35] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2013, pp. 487–498.

[36] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2009, pp. 163–174.

[37] R. Nath and D. Tullsen, "The CRISP performance model for dynamic voltage and frequency scaling in a GPGPU," in *Proc. 48th Int. Symp. Microarchitecture (MICRO)*, Dec. 2015, pp. 281–293.

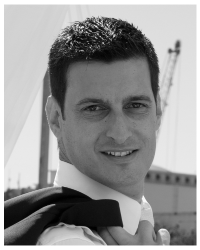
[38] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *Proc. Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 564–576.
[39] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "Modeling and decoupling the GPU power consumption for cross-domain DVFS," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2494–2506, Nov. 2019.
[40] M. Andersch, J. Lucas, M. Álvarez-Mesa, and B. Juurlink, "Analyzing GPGPU pipeline latency," in *Proc. Summer School Adv. Comput. Archit. Compilation High-Perform. Embedded Syst. (ACACES)*, 2014.
[41] Google. (2019). *NVIDIA Tesla T4 GPUs Now Available in Beta.* [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/nvidia-tesla-t4-gpus-now-available-in-beta
[42] NVIDIA. (2019). *NVML API Reference Guide vR418*. [Online]. Available: http://docs.nvidia.com/deploy/nvml-api

JOÃO GUERREIRO received the M.Sc. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa, Lisbon, Portugal, in 2014. He is currently pursuing the Ph.D. degree with the Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), where he is also a Junior Researcher. His research interests include parallel computing, graphics processors, and energy-efficiency.



ALEKSANDAR ILIC received the Ph.D. degree in electrical and computer engineering from Instituto Superior Tecnico (IST), Universidade de Lisboa, Portugal, in 2014. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, IST, and a Senior Researcher of the Signal Processing Systems Group (SiPS), Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). He has contributed to more than 40 articles to international journals and conferences and has served in the organization of several international scientific events. His research interests include high-performance and energy-efficient computing and modeling on parallel heterogeneous systems.

NUNO ROMA received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 2008. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, IST, and a Senior Researcher of the Signal Processing Systems Group (SiPS), Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research interests include computer architectures, specialized and dedicated structures for digital signal processing, energy-aware computing, parallel processing, and high-performance computing systems. He contributed to more than 100 manuscripts to journals and international conferences and has served as a Guest Editor of the *Springer Journal of Real-Time Image Processing* (JRTIP) and of the *EURASIP Journal on Embedded Systems* (JES). He has also acted as the organizing chair of several workshops and special sessions. He has a consolidated experience on funded research projects leadership (principal investigator). He is a member of several research Networks of Excellence (NoE), including HiPEAC (European Network of Excellence on High Performance and Embedded Architecture and Compilation). He is a Senior Member of the IEEE Circuits and Systems Society and of the ACM.



PEDRO TOMÁS (S'04–M'09–SM'18) received the five-year licentiate, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the Instituto Superior Tecnico (IST), Technical University of Lisbon, Portugal, in 2003, 2006, and 2009, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering (DEEC), IST, and a Senior Researcher with the Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research activities include computer microarchitectures, specialized computational structures, and high-performance computing. He is also interested in artificial intelligence models and algorithms. He is a member of the IEEE Computer Society and has contributed to more than 60 articles to international peer-reviewed journals and conferences.