Parallel Computing 83 (2019) 93–117

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

DVFS-aware application classification to improve GPGPUs energy efficiency

João Guerreiro*, Aleksandar Ilic, Nuno Roma, Pedro Tomás

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal, Rua Alves Redol, 9, Lisboa 1000-029, Portugal

ARTICLE INFO

Article history:
Received 30 November 2016
Revised 20 August 2017
Accepted 1 February 2018
Available online 9 February 2018

Keywords:
GPGPU
Application classification
DVFS
Optimal frequency
Energy savings

ABSTRACT

The increasing importance of GPUs as high-performance accelerators and the power and energy constraints of computing systems, make it fundamental to develop techniques for energy efficiency maximization of GPGPU applications. Among several potential techniques, dynamic voltage and frequency scaling (DVFS) stands out as one of the most promising approaches. Hence, novel DVFS-aware performance and power classification models are herein proposed that correlate application characteristics and GPU architecture features. In particular, by analysing the utilization of graphics and memory components at a single voltage and frequency levels, the proposed classification methodologies are able to predict the impact of DVFS on GPGPU applications execution time and power and energy consumption. The accuracy of the proposed approach is validated on two modern NVIDIA GPUs from the Maxwell and Pascal generations, by relying on 35 benchmarks from the Rodinia, Polybench, Parboil, SHOC and CUDA SDK suites. Experimental results show that the proposed approach can typically predict the optimal operating frequencies of graphics and memory subsystems, attaining up to 36% energy savings (average of 16%), which correspond to an average deviation of 0.74% regarding the optimal case. Moreover, when considering a maximum performance penalty of 10%, up to 26% energy savings are still attained.

1. Introduction

General purpose accelerators have already gained a firm presence in most modern high-performance computing (HPC) systems. In particular, the Graphics Processing Units (GPUs), are commonly used to increase the resulting system performance when executing applications from many commercial and scientific domains. This can be easily observed at the most recent version of the TOP500 list (June 2017): 88 of these systems are equipped with accelerators, where 72 of them use GPUs. However, such an established adoption of GPUs intensifies the importance to find reliable mechanisms that ensure the maximum efficiency of the computing system, both in terms of performance and (most importantly) energy consumption. Accordingly, significant research efforts are being put forth in the investigation of Dynamic Voltage and Frequency Scaling (DVFS) techniques (one of the most used power management strategies), due to the inherent potential for significant power and energy savings in many of the computer system components [1–5].

Studying the effects of DVFS on the resulting energy-efficiency of computing systems requires analysing its impact on different applications, as general-purpose applications can largely vary in the way they use the computational and memory

* Corresponding author.
E-mail address: joaoguerreiro@inesc-id.pt (J. Guerreiro).

https://doi.org/10.1016/j.parco.2018.02.001
0167-8191/

resources of the devices where they are executing [6,7]. While some applications perform a large number of computational operations for each loaded data (more *compute-intensive*), other applications may perform very few (more *I/O-* or *memory-intensive*). Although the resulting performance of the former type of applications is more likely to scale proportionally with the frequency of the cores (highest frequency $\equiv$ best performance), this behaviour is not guaranteed for the latter set of applications. Additionally, most standard applications fall somewhere between these extremes, increasing the characterization complexity. However, knowing the type of application executing can lead to interesting opportunities for energy savings. For instance, certain applications can be executed at lower frequency levels with negligible performance drop-off. Identifying these classes of applications can lead to significant energy savings, since lower operating frequency leads to lower power consumption. Hence, to conduct this type of analysis, it is fundamental to adopt methodologies that allow a proper classification of the application workloads.

Some previous studies on workload characterization in the GPU-domain used Principal Component Analysis (PCA) and hierarchical clustering [7–9], which make the understanding of each resulting class harder (from the computing architecture perspective) and do not necessarily result in an accurate energy-aware classification. Other works depend on GPU simulators and performance counters that are non-existent in real hardware devices, rendering these approaches impossible to replicate in real systems. Furthermore, most existing GPU simulators are based on the NVIDIA Tesla and Fermi microarchitectures, which have already been followed by Kepler (2013), Maxwell (2014) and Pascal (2016). A different approach towards DVFS consists in the development of accurate performance and power models that allow predicting the GPU behaviour under different voltage and frequency scenarios [10–21]. However, while detailed performance and power models may ultimately produce more accurate results, the herein presented work shows that application classification is a more convenient and viable approach not only to identify remarkable energy-savings opportunities, but also to achieve near-optimal results in terms of energy savings.

Accordingly, the main contribution of the presented research is to provide a new classification methodology for GPGPU applications, allowing an easy identification of which applications can benefit from DVFS, in terms of energy savings. To that goal, separate methodologies to classify GPU applications are proposed, focusing on the effects of DVFS on their performance and power consumption, based on how the applications exploit the different GPU resources. The classifiers are trained offline, using a collection of synthetic benchmarks, and can be used to classify any application using hardware performance events gathered during its execution on a single operating frequency. The resulting class is able to characterize how the frequency scaling will affect the application execution time or power consumption, for all the remaining operating frequencies.

The proposed methodology was validated using a set of 35 applications from different relevant benchmark suites (Parboil [22], Rodinia [23], SHOC [24], Polybench [25] and CUDA SDK [26]), on two modern GPU devices: GTX Titan X and Titan Xp, from the Maxwell and Pascal microarchitectures, respectively. The experimental results show that the proposed methodologies are able to accurately and consistently classify the considered GPU applications in terms of their behaviour in performance, power and energy consumption. The proposed classification methodology allows finding a near-optimal pair of operating frequencies (core and memory), which can result on average energy-savings of 16% (20%), and on peak energy-savings that are as high as 36% (32%) on the Maxwell GPU (Pascal GPU). Additionally, in situations where a decrease in the processing performance is not allowed or highly discouraged, the proposed methodology is still able to find real opportunities for energy-efficiency with a limited performance trade-off (*e.g.*, $< 10\%$), resulting in average energy-savings of 9% on the Maxwell GPU (16% on Pascal), although in some classes such savings can even be as high as 22% with only a 0.2% performance trade-off. Accordingly, the most significant contributions of this paper are the following:

- Analysis of the impact of DVFS on the performance and power consumption of different types of GPU benchmarks on real hardware;
- Novel application-classification scheme based on GPU performance and power metrics, able to characterize the impact of DVFS on the execution of different types of applications, for a wide range of GPU operating frequencies, validated with standard applications on real GPU devices;
- Application of the proposed classification methodologies to optimize: (i) the consumed energy; (ii) the energy *v.s.* performance trade-off; and (iii) the exploitable energy-savings ranges.

The rest of this paper is organized as follows. In Section 2, the prevalent GPU architectures are briefly discussed, as well as the impact of DVFS on the performance and power consumption of different applications. Section 3 presents the proposed methodologies to classify GPU applications into classes with similar characteristics. Section 4 validates the proposed methodologies using a set of 35 standard applications. Finally, Section 5 applies the proposed methodologies to find the best operating frequencies for the different applications, in order to maximize the energy-savings. Section 6 compares the proposed methodology with the current state of the art, and finally Section 7 concludes the manuscript.

2. Analysis of the DVFS impact on GPGPU applications

Similarly to other computing systems, the architecture of current GPU devices allows for the different components to be clocked at distinct and independent frequencies. Fig. 1 presents a simplified representation of a modern GPU device,

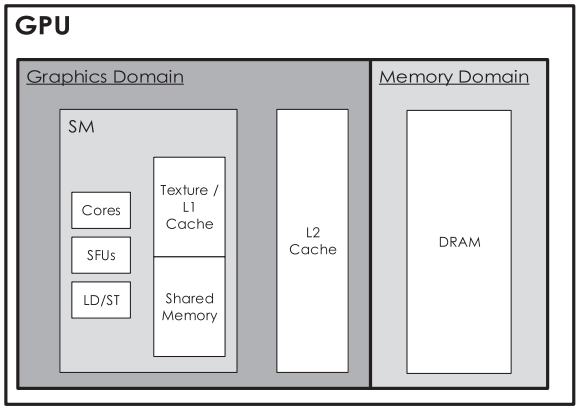


Fig. 1. Existing frequency domains in modern GPU devices (*e.g.* NVIDIA Kepler, Maxwell and Pascal GPUs).

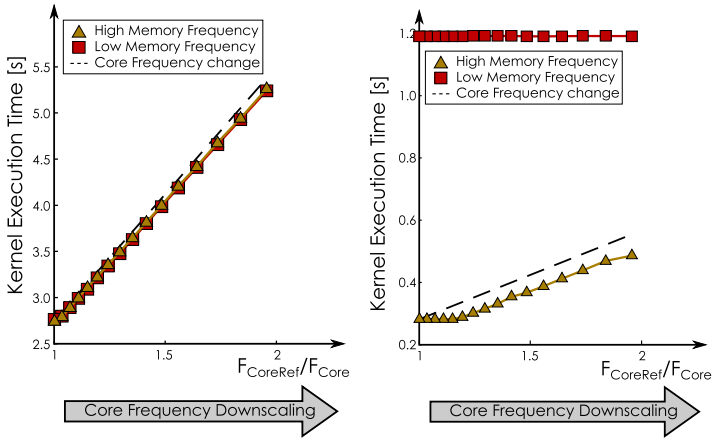
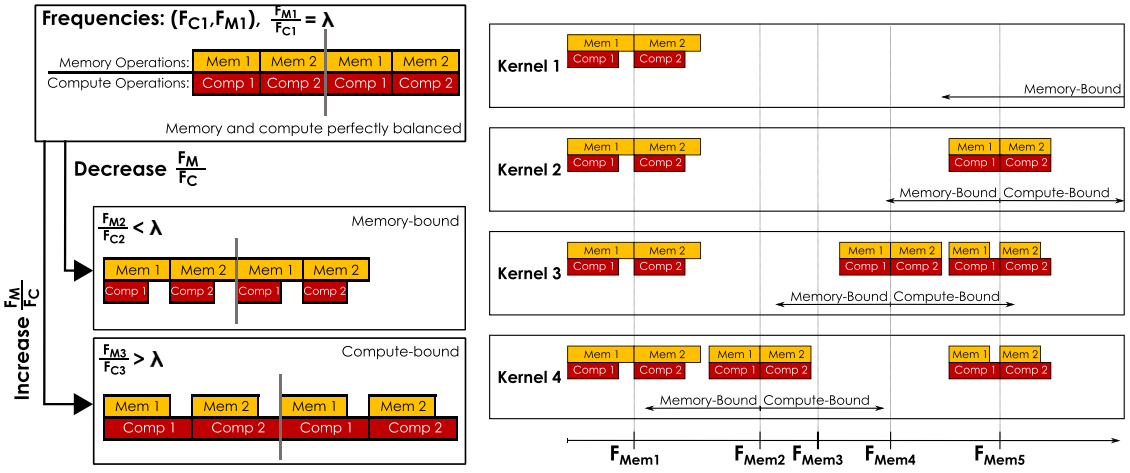


Fig. 2. DVFS impact on two distinct Rodinia applications on a NVIDIA GTX Titan X, where $F_{CoreRef} = 1164MHz$.

containing two distinct frequency domains: the Graphics (or Core) domain and the Memory domain.[1] The Graphics domain includes both the streaming-multiprocessors (SMs) and the L2 cache, while the Memory domain includes only the device main memory, *i.e.* DRAM memory. Scaling the frequency of each domain can have different results on an application execution, largely depending on the considered application characteristics [27]. While it can be expected that a decrease of the core frequency (F_{Core}) and voltage (V_{Core}) will cause the kernel[2] execution time to increase ($T \propto \frac{1}{F_{Core}}$) and the resulting power consumption to decrease ($P_{dynamic} \propto {V_{Core}}^2 F$ and $P_{static} \propto V e^{\gamma V_{Core}}$), the application performance and power consumption over different frequencies are highly dependent on the way the application exploits each of the GPU subsystems. In fact, it has been shown that accurately predicting the impact of DVFS in the execution time or power consumption often requires the usage of complex predictive models [28,29]. Accordingly, it is important to understand the effects of DVFS on both the execution time (t) and power consumption (P), in order to be able to extract meaningful conclusions relative to the behaviour of the resulting energy consumption (E) over different frequencies ($E = P \times t$).

To illustrate the referred problem, Fig. 2 presents one example with two applications that have their execution time affected very differently when the core and memory frequencies are scaled, namely *Hotspot* (from Rodinia) and *Blackscholes* (from CUDA SDK). For *Hotspot* (see Fig. 2a), the kernel execution time always scales inversely with the core frequency (F_{Core}). However, for the *Blackscholes* case (see Fig. 2b), it is possible to maintain the overall kernel execution time while scaling down the core frequency, as long as the memory keeps operating at the higher frequency. It can also be observed that the effects of scaling the memory frequency in the execution time are also very different in the two applications. While for the *Blackscholes* (Fig. 2b) there is a significant increase in the execution time when the memory frequency is decreased, the performance of *Hotspot* (Fig. 2a) is not affected by the same change.

[1] This setup is currently used by several NVIDIA GPU microarchitectures, such as Kepler, Maxwell and Pascal.
[2] Kernel: routine to be executed in a massively parallel fashion on a GPU device by multiple threads.



(a) Overlap of memory and compute operations on three frequency states, with $\frac{F_{M3}}{F_{C3}} > \frac{F_{M1}}{F_{C1}} > \frac{F_{M2}}{F_{C2}}$.

(b) Performance classification changing with the memory frequency.

Fig. 3. Examples of DVFS impact on overlapped instructions. The instructions pairs (**Mem1**, **Comp1**) and (**Mem2**, **Comp2**) require full synchronization.

Accordingly, this manuscript proposes two separate methodologies to classify any application according to the impact of DVFS in the resulting execution time and power consumption. Finally, by combining the two approaches it is possible to infer how the energy-consumption will change when the frequency of the cores and of the memory are scaled. Sections 2.1 and 2.2 describe the DVFS impact on the applications execution time and power consumption, respectively.

2.1. DVFS impact on the applications performance

The impact of DVFS on an application execution time is a complex problem that requires deep understanding of the GPU architecture. In particular, one of the GPU main design goals concerns the use of multiple groups of parallel threads (*warps* in NVIDIA nomenclature) to hide instruction latency. However, in many applications it is not always possible to hide the instruction latency with other warps. Therefore, when analysing the applications performance, from the perspective of their bottlenecks and limiting factors, most works tend to consider two main types of applications [3,30,31]: (1) *compute-bound*, where the execution time is mainly determined by the performance of the processing components; and (2) *memory-bound*, where the execution time mainly depends on the bandwidth and latency of the memory hierarchy when satisfying memory access requests. Accordingly, the adopted setup in terms of the core and memory operating frequencies (F_{Core} and F_{Mem}) will result in different performance *versus* power patterns if a certain kernel is more memory-bound or more compute-bound.

Moreover, while one kernel may be compute-bound at a certain operating frequency state, it may become memory-bound at different core and/or memory frequency states. To illustrate such condition, Fig. 3a presents the relative weight variation of the memory and compute operations of one given kernel for three different scenarios. At frequency state (F_{C1}, F_{M1}), the execution of both **Mem1** and **Comp1** instructions occur at the same time and both finish their execution at the same instant. In this example, the subsequent instructions **Mem2** and **Comp2** require full synchronization but since both instructions finish at the same time, the latency of the threads waiting to be issued is fully hidden by the threads currently executing. However, if the core frequency is increased to a higher value (F_{C2}) and/or the memory frequency is decreased to a lower value (F_{M2}) such that $\frac{F_{M2}}{F_{C2}} < \frac{F_{M1}}{F_{C1}}$, there will be a time interval where only the **Mem** instructions are executing on the GPU, meaning that there are not enough threads executing **Comp** instruction that can hide the latency of the threads waiting on pending memory operations. Hence, at frequency state (F_{C2}, F_{M2}) the application is considered to be memory-bound, since its performance bottleneck depends on the latency of the memory operations. If, on the contrary, the operating frequencies were set to state (F_{C3}, F_{M3}), such that $\frac{F_{M3}}{F_{C3}} > \frac{F_{M1}}{F_{C1}}$, the execution of the **Comp** instructions would be longer than the **Mem** instructions, meaning the performance is limited by the compute instructions, thus resulting in a compute-bound classification.

As a result, the commonly used binary classification (*compute* or *memory-bound*) may not be valid for all combinations of frequency levels. In the remainder of this work an application is considered memory-bounded at frequency F_{Mem_i} if with that memory frequency, there is at least one core frequency (within the range allowed by the device) where the application performance is limited by the accesses to the DRAM. On the other hand, if with memory frequency set to F_{Mem_i} the application is never bounded by DRAM accesses (for all core frequencies), the application is considered to be compute-bounded at frequency F_{Mem_i}.

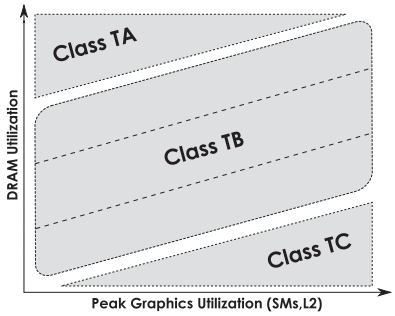


Fig. 4. DVFS-aware performance classes depending on the DRAM and Graphics utilization.

In the two extreme scenarios, where the memory throughput is extremely low compared with the core ($\frac{F_{Mem}}{F_{Core}} \rightarrow 0$) or extremely high ($\frac{F_{Mem}}{F_{Core}} \rightarrow \infty$) it is trivial to classify any given application in the *memory-bound* and *compute-bound* classes, respectively. Accordingly, as the ratio $\frac{F_{Mem}}{F_{Core}}$ is decreased from ∞ (for example by fixing F_{Core} constant and decreasing F_{Mem}), more and more applications initially classified as *compute-bound* will start becoming *memory-bound* at each memory frequency level. Fig. 3b presents one example of such a scenario, where the performance behaviour of four different kernels change as the memory frequency varies. It is important to stress that since all possible values of $\frac{F_{Mem}}{F_{Core}}$ can be achieved by fixing one of the values constant and scaling the other, for the sake of simplification the value of F_{Core} is considered constant, without loss of generality. When the memory frequency is scaled down between any two frequency levels (*e.g.* from $\mathbf{F_{Mem5}}$ to $\mathbf{F_{Mem3}}$), the amount of time required to satisfy all DRAM accesses will increase, and therefore, one of four possible scenarios will occur:

1. **Kernel 1:** The application was already *memory-bound* at the highest memory frequency ($\mathbf{F_{Mem5}}$), so it will remain *memory-bound* at any other lower memory frequency;
2. **Kernel 2:** The application was well balanced as both the **Mem** and **Comp** instructions start and finish at the same time; since a decrease of the memory frequency will mostly affect the **Mem** instructions, the application will become *memory-bound* at $\mathbf{F_{Mem3}}$;
3. **Kernel 3:** The application was *compute-bound*, but as soon as the memory frequency reduces to values lower than $\mathbf{F_{Mem4}}$ the performance of the application starts being limited by the DRAM bandwidth, therefore being *memory-bound* at frequency $\mathbf{F_{Mem3}}$;
4. **Kernel 4:** The application was *compute-bound* at $\mathbf{F_{Mem5}}$ and it remains *compute-bound* at the lower memory frequency $\mathbf{F_{Mem3}}$.

Hence, considering that one of the objectives of this work is to provide a DVFS-aware classification for the resulting performance of GPU applications, this classification must be able to characterize how the execution time of each application changes when core and memory DVFS is applied. Therefore, the proposed methodology to characterize the impact of DVFS in the applications performance will have to depend on the memory frequency levels of the GPU device.

Since GPU devices do not provide any performance counters that immediately define what type of application is running and how its execution is affected by frequency scaling, it is important to choose the relevant counters that can be used to indirectly infer this information. The impact of DVFS in the execution time of an application depends mostly on how it utilizes the GPU resources. In particular, since GPU applications are usually able to exploit the inherent parallelism of the device, the applications execution time is a result of the overlap of the several instructions executed by the multiple warps. Since the memory and compute instructions are generally overlapped during the execution, the impact of DVFS on an application performance will be dependent on the utilization of both the Memory and Graphics resources. Additionally, since different instructions can be executed on distinct functional units of the Graphics domain (single precision, double precision, special function, load/stores, etc.), the utilization of the Graphics domain resources will be mostly related with the component with the highest utilization of that domain. Hence, depending on the utilization ratio of the Memory and Graphics resources($\frac{DRAMUtil.}{GraphicsUtil.}$), different classes of applications can be considered. By taking into consideration which of the two GPU domains is more dominant, such classes can be grouped in three main levels (see also Fig. 4):

1. Applications that are *memory-bound* at the highest allowed memory frequency level (F_{Mem_high}), and therefore at all other lower memory frequency levels if they exist (see Class **TA** in Fig. 4). These applications have a very high DRAM utilization and their execution time is highly affected by changes in the memory frequency. Additionally, as a consequence of the compute instructions being often stalled due to memory dependencies, they are also characterized by a low utilization of the major graphics components (shared memory, floating-point units, etc.).
2. Applications that are compute-bound at F_{Mem_low}, and therefore at all other higher memory frequency levels. The execution of these applications is not significantly affected by changes in the memory frequency, but it is highly dependent on

Table 1
Power consumption for matrixMulCUBLAS kernels with different iterations.

	15 iters	1500 iters
(3505, 975)	226 W	239 W
(810, 595)	116 W	120 W

the core operating frequency. Such applications (see Class **TC** in Fig. 4) are characterized by a high graphics utilization and a low memory utilization.

3. Applications that scale with both memory and core frequencies (see Class **TB** in Fig. 4). If multiple memory frequency levels are available, it is possible to subdivide this class into multiple classes, which will include the applications that are compute-bound at F_{Mem_high} and change to memory-bound at the lower memory frequency levels. By considering one distinct class for each additional memory frequency level, it will allow the distinction between applications that become memory-bound at different memory frequency levels, resulting in better accuracy in the characterization.

Section 3 will detail further how the boundaries between the classes can be obtained, with an example of the utilization of the proposed approach on a real GPU device.

2.2. DVFS impact on the applications power consumption

As it was previously seen, the execution of instructions on different components of the GPU is often partially or fully overlapped. However, the power consumption of the several different components cannot be hidden or masqueraded, and must be always combined together in order to obtain the total power consumption of the GPU.

When considering the instantaneous power consumption (P_{Total}) [18] of a CMOS circuit, one must take into account both its dynamic ($P_{Dynamic}$) and static (P_{Static}) fractions, as $P_{Total}=P_{Dynamic}+P_{Static}$. Additionally, when considering both voltage and frequency scaling, these two fractions of the total power consumption have different scaling behaviours, as $P_{Dynamic} \propto V^2 F$ [32] and $P_{Static} \propto V \hat{I}_{leak}$ [33], where F denotes the operating frequency, V the chip supply voltage and $\hat{I}_{leak}$ is the normalized leakage current for a single transistor, dependent on the threshold voltage (V_{th}) and on the temperature. These formulations relate to the instantaneous power consumption of each separate device component. In a device with more than a single frequency domain (*e.g.* GPU) the total power consumption can be expressed as:

$$P_{GPU} = P_{Graphics}(F_{core}, V_{core}) + P_{Memory}(F_{mem}, V_{mem}), \quad (1)$$

where $P_{Graphics}$ and P_{Memory} represent the power consumption of the graphics and memory domains, respectively. In fact, each of these parts can be further decomposed into the several parcels of the power that is consumed by each of the internal components of that domain (processing cores, LD/ST units, shared memories, L2 cache, etc). However, since device manufacturers do not fully disclose the design of each architecture, several of the parameters that are required for an accurate power modelling are usually unknown. Additionally, it is not easy to measure or infer the instantaneous static/dynamic power parcels in current GPU devices. In fact, most modern GPUs only provide one single counter to report the instantaneous power consumption of the whole GPU board, combining information from multiple contributing domains (Memory and Graphics) in an undisclosed manner, making it a very hard task to distinguish all the separate effects contributing to that value. Accordingly, this work will not focus on characterizing each individual parcel of the power consumption of a given application, but rather the average total, at the device level, during the whole application execution.

Finally, to analyze DVFS impact, this work focuses on the relative difference in the GPU power consumption between the different frequency levels. The proposed methodology does not take into account the influence of temperature in the GPU power consumption. While taking it into consideration could potentially be beneficial when creating a power model of the architecture (since the value of leakage current $\hat{I}_{leak}$ quadratically increases with temperature [18]), it is less significant when classifying the impact of DVFS on the power consumption (*i.e.*, the relative change in power consumption).

As an example, Fig. 5 presents the power consumption and temperature of the GTX Titan X GPU during the execution of two variations of the same application, on two frequency configurations: ($F_{Mem} = 3505$ MHz, $F_{Core} = 975$ MHz) and ($F_{Mem} = 810$ MHz, $F_{Core} = 595$ MHz). The two applications execute the same matrixMulCUBLAS kernel, repeated a different number of times (15 and 1500), thus leading to different GPU active times. It can be seen that, as expected, the application running for a longer period of time causes the GPU temperature to increase, which in turn increases the GPU power consumption. However, although there is a power consumption increase due to temperature variations (up to 13W at the highest frequency setting, as shown in Table 1), when looking at the relative changes in the power consumption, the impact of the temperature is almost non-existent (see Fig. 6), which means the two applications can be classified similarly.

Regarding the effects of DVFS, Fig. 7 presents an example of the overlap (in time) of the memory and computational instructions of two ideal applications, and how the total execution time of each application is affected by core frequency scaling (in both examples, the memory voltage and frequency levels are fixed to a constant value). The application with higher DRAM utilization (see Fig. 7a) has its execution mostly dominated by the memory accesses, specifically to DRAM. Since in this example only the F_{Core} is changing (decreasing), the instantaneous power consumption remains constant for

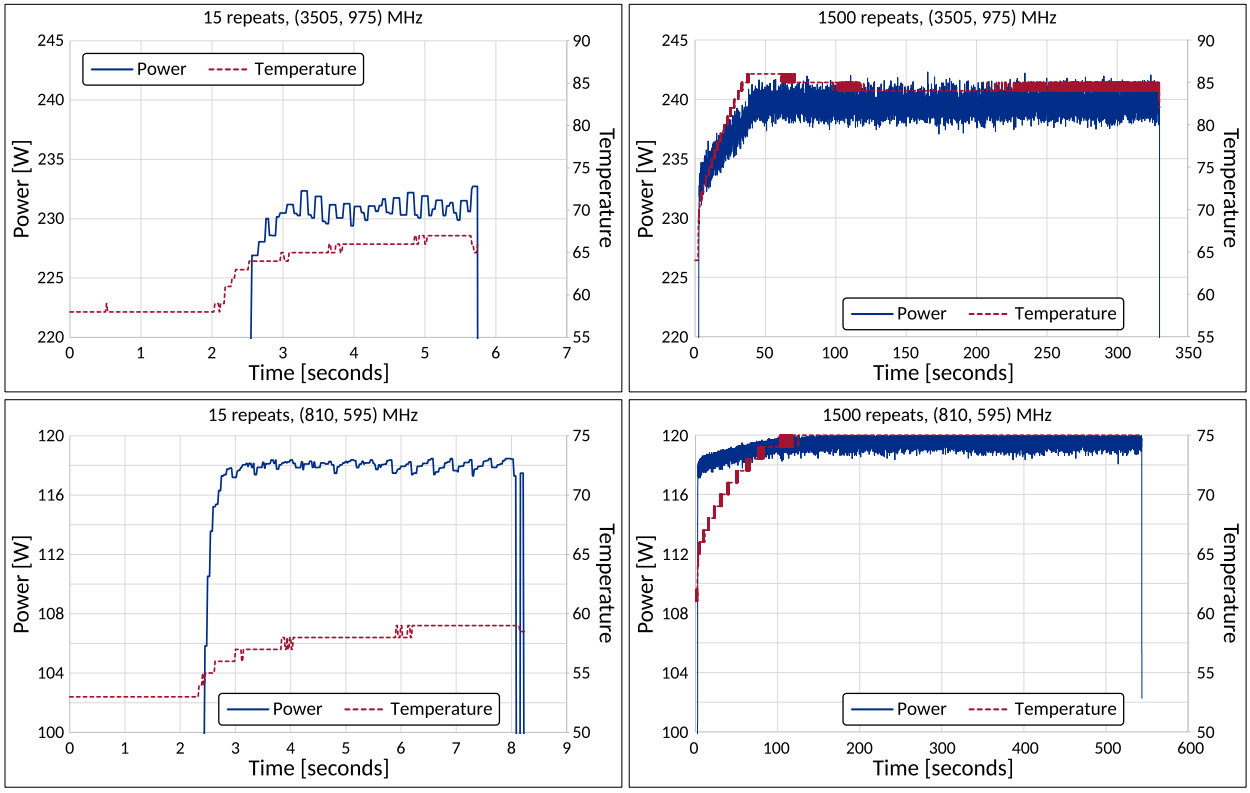


Fig. 5. GPU power consumption and temperature obtained during the execution of the *matrixMulCUBLAS* kernel with square matrices of size 8192 with two different durations (with 15 and 1500 repetitions of the kernel, i.e. with 6 and 330 seconds, respectively). Here are presented the results for two distinct GPU configurations on the Titan X GPU: (F_{Mem} = 3505 MHz, F_{Core} = 975 MHz) and (F_{Mem} = 810 MHz, F_{Core} = 595 MHz).

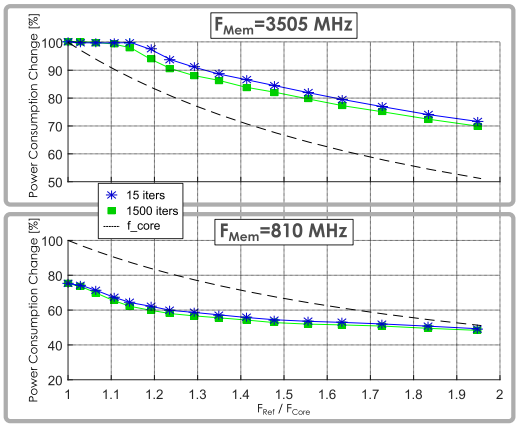
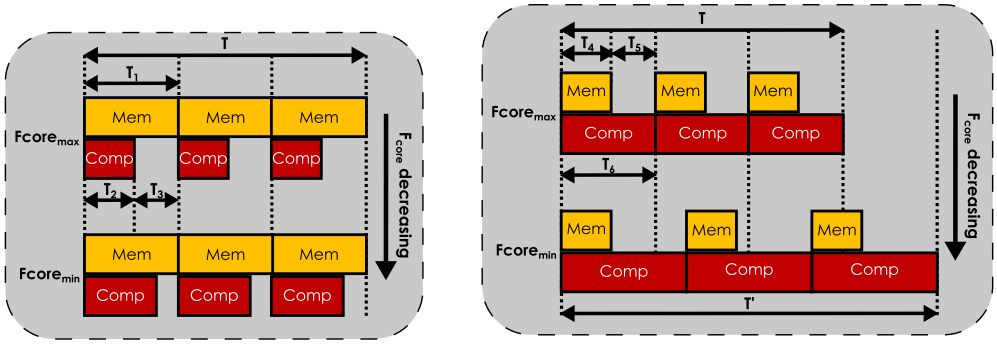


Fig. 6. Power consumption for matrixMulCUBLAS kernels with different number of iterations, on NVIDIA's GTX Titan X.

all components whose utilization is not changed by the core frequency downscaling, which includes both the static and dynamic power consumption of the components in the memory domain. Furthermore, since the total execution time (T) does not change, the average power consumption of these components will also remain constant. On the contrary, the graphics domain will be highly affected by the core frequency downscaling, resulting in an increase of the execution time of the compute instructions (T_2). In fact, by taking into account the previous considerations relative to the instantaneous power, it can be expected that the power consumption will downscale on the components utilized by the compute instructions. By combining all the effects (F_{Core} and V_{Core} downscaling), the average power consumption is expected to decrease with $F_{Core}V_{Core}{}^2$.

On the other hand, the application with high SM utilization (see Fig. 7b) presents an increase of its total execution time (from T to T') when the core frequency is decreased. Similarly to the previous case, the instantaneous power consumption of



(a) High DRAM utilization. (b) High SMs utilization.

Fig. 7. Expected impact of DVFS on the execution time of two ideal applications (in both cases F_{Mem} is constant).

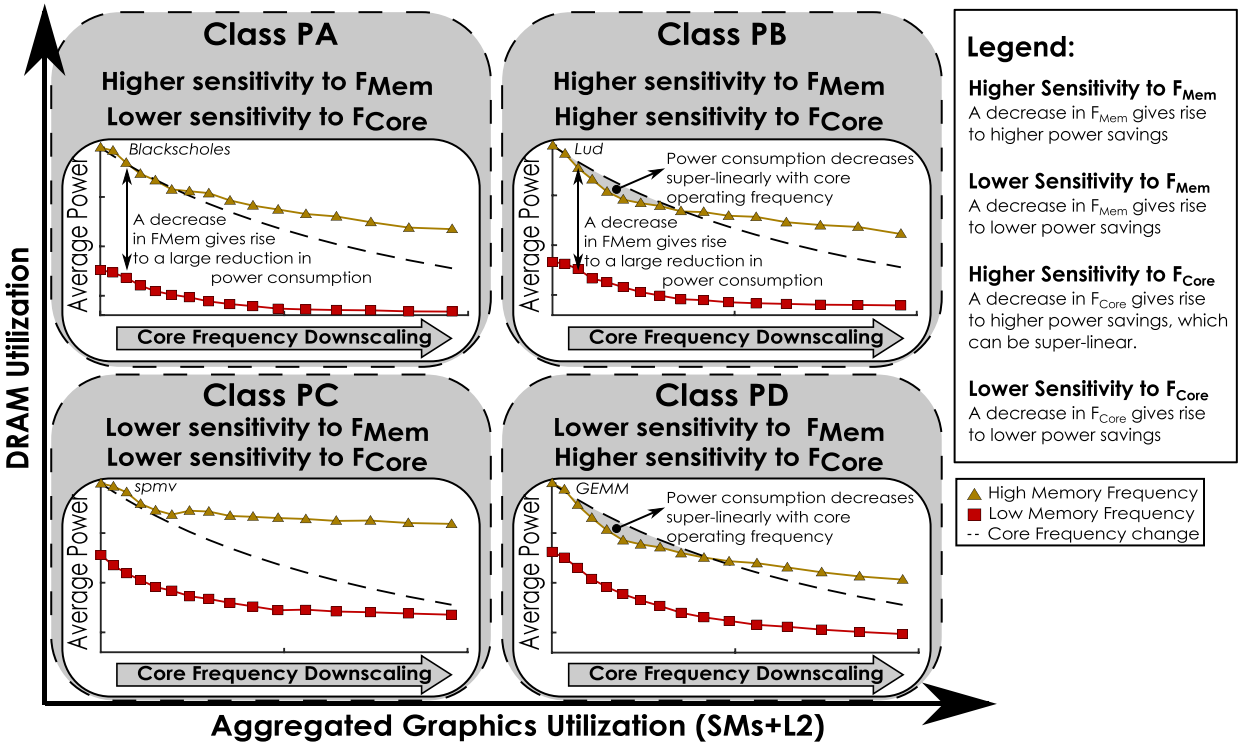


Fig. 8. DVFS-aware power classes depending on the DRAM and Graphics utilization. The example power consumption curves were obtained on NVIDIA's GTX Titan X, for the *Blackscholes, Lud, spmv* and *GEMM* benchmarks.

the SM components will downscale with $F_{Core}V_{Core}{}^2$. However, since the performance bottleneck is now associated with the compute instructions, the observed increase of the total execution time will increase their contribution to the total power consumption of the device. In other words, the contribution of the memory components to the power consumption of the whole application will decrease, since the same number of memory operations is being executed, but scattered for a longer period of time, as the memory components now spend more time idling than before (T_4 decreases and T_5 increases). This will result in a greater decrease of the average power consumption than the previous $F_{Core}V_{Core}{}^2$.

A similar approach could be applied to analyse the impact of memory frequency scaling in the average power consumption of different applications, also resulting in two distinct behaviours depending on how the GPU resources are being utilized. Accordingly, depending on the combined resource utilization of the two GPU domains, it is possible to identify the classes presented in Fig. 8. The definition of each proposed class, depending on how their power consumption changes with the frequency of the cores and memory, is:

1. Class **PA:** higher sensibility to memory frequency changes, lower sensibility to core frequency changes;
2. Class **PB:** higher sensibility to memory frequency changes, higher sensibility to core frequency changes;

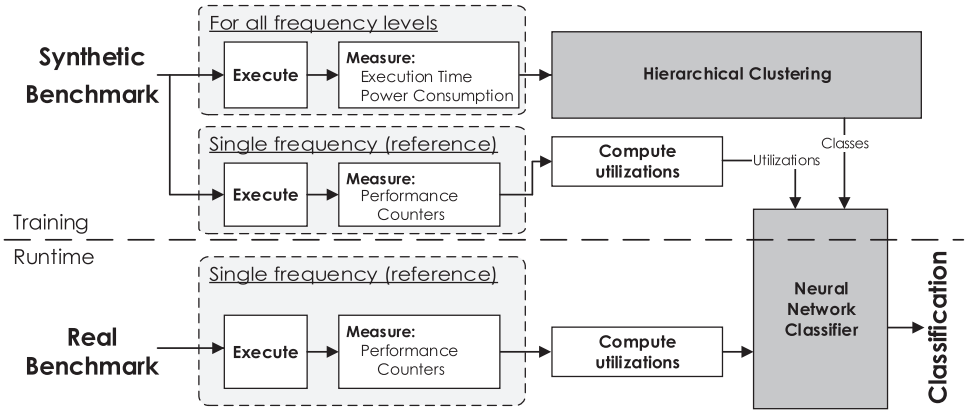


Fig. 9. Overview of the proposed procedure to classify the DVFS impact on the performance/power consumption of GPU applications.

3. Class **PC:** lower sensibility to memory frequency changes, lower sensibility to core frequency changes;
4. Class **PD:** lower sensibility to memory frequency changes, higher sensibility to core frequency changes.

Unlike the performance characterization, the number of identified classes in this characterization does not need to scale with the number of memory frequency levels available in the GPU device. The only special case is when there is only one memory frequency available (rare case in modern GPU devices), in which there is no need to characterize applications regarding the impact of memory frequency downscaling on their executions, therefore only being possible to identify two classes (Classes **PA** and **PB**).

By taking into consideration the set of observations laid out in this section, the following section applies the proposed generic performance and power classification to a real GPU device, and derives a methodology to obtain the boundaries of the performance and power classes (Section 3.2).

3. Application characterization procedure

The description of the conceived methodology to obtain the performance and power-aware characterizations of any given application will be conducted by using a representative example using a NVIDIA GTX Titan X GPU. The selection of this particular GPU device to test the impact of DVFS on the applications execution arises not only from its greater offer of different core frequency levels (43 non-idle levels) and memory (DRAM) frequency levels (3 non-idle levels), but also because it provides a convenient interface to measure the power consumption at runtime.

Fig. 9 presents an overview of the proposed procedure to classify the DVFS impact on the performance (or power-consumption) of GPU applications. To train the classifiers a collection of synthetic benchmarks is used, with their execution time and power consumption measured at all allowed operating frequency levels. In order to characterize how each application is stressing the GPU components, a group of hardware performance events is also measured at a single (reference) operating frequency. Based on the gathered metrics, an hierarchical clustering algorithm is used to define the class of each synthetic benchmark, used later in the supervised training of the neural-network classifier. Once the classifier is trained, it allows the characterization of that given architecture and how DVFS impacts the execution of different types of applications. The following sections further detail each of the steps of the procedure presented in Fig. 9.

3.1. Synthetic benchmarking and profiling

In order to classify any given application it is first necessary to characterize the adopted GPU device, more specifically how DVFS impacts the execution time and power consumption of different types of applications. This is usually done through the execution and profiling of a controlled set of synthetic benchmarks designed for this specific purpose. Accordingly, and since the DVFS impact on the application execution is related to how the resources of the Graphics and Memory domains are utilized, different benchmark applications need to be created, with varying levels of utilization of the two GPU domains.

Fig. 10 presents the basic structure of one of the developed synthetic GPU kernels. By executing multiple kernels with distinct values of the NUM_ITERS parameter, different combinations with different ratios between the number of memory accesses and the amount of computations can be tested. Each synthetic benchmark was executed at all supported frequency levels, during which the execution time and power consumption were accurately measured, in order to evaluate how each application is affected by frequency scaling. The utilization of the several GPU resources by the different kernels was also quantified, by measuring several hardware performance events during the execution of each kernel. However, unlike the execution time and power consumption, these values are measured at a single frequency level. Since the highest operating

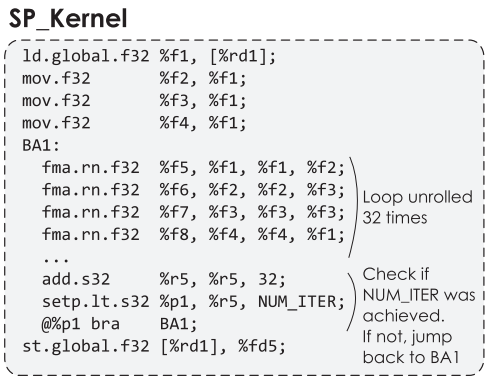


Fig. 10. Example PTX code from one of the developed synthetic kernels.

Table 2
Metrics used in the DVFS-aware performance and power classification methodologies.

Metric name	Description	Used in	
		Performance	Power
$DRAM_{Transactions}$	Number of reads and writes to DRAM memory	YES	YES
$L2_{Transactions}$	Number of reads and writes to L2 cache	YES	YES
$Shared_{Transactions}$	Number of reads and writes to shared-memory	YES	YES
FU_{Single}	Utilization† level of the single-precision function units	YES	YES
FU_{Double}	Utilization† level of the double-precision function units	YES	YES
$FU_{Special}$	Utilization† level of the special function units	YES	YES
$FU_{Texture}$	Utilization† level of the texture function units	NO	YES
Registers	Number of registers used per thread	NO	YES
Occupancy	Ratio of active warps on an SM with the maximum number supported by the SM	NO	YES

† Utilization: Ratio of the experimentally achieved throughput of each unit with respect to the theoretical peak.

frequency level is the one that usually provides the best-performance, the performance events were measured at this level (on GTX Titan X: $F_{Mem} = 3505$ MHz and $F_{Core} = 1164$ MHz).

The impact of DVFS in the performance and power-consumption of applications is in both cases dependent on how the GPU components are being utilized. However, the components most relevant and how to take them into account may differ between the two classifications. While in the performance classification the interesting metrics refer to the utilization of the most predominant components, *i.e.* the ones with higher probability to be limiting the performance, in the power classification the considered metrics are the aggregate utilization of all components, since the device power consumption is the sum of the power consumption of each individual component. Additionally, the components with the greatest influence on the performance may also differ from those with the highest influence on the power consumption (see [19,20,34]), i.e. the performance metrics that will be utilized for the two classification methodologies may be different.

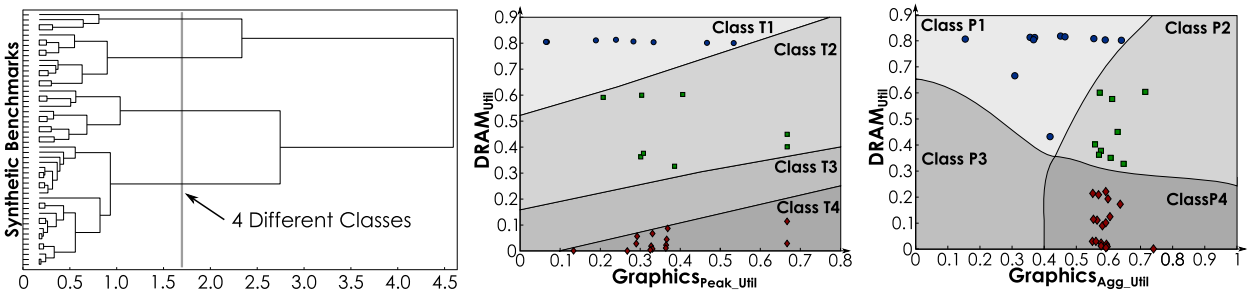
The set of performance counters that were used to quantify the utilization of the two GPU domains in the NVIDIA GTX Titan X GPU are presented in Table 2. The number of registers used per thread can be obtained at compile time by using the *nvcc* compiler [35], while the remaining performance counters can be measured using the nvidia profiler (*nvprof*) [36] during the execution of the kernels. The DRAM bandwidth is computed as follows:

$$DRAM_{Bandwidth} = \frac{DRAM_{Transactions} \times Transaction\_width}{Execution\_time}, \tag{2}$$

where $DRAM_{Transactions}$ and Execution_time are measured by the profiler, while Transaction_width is a characteristic of the GPU microarchitecture. The shared-memory and L2 bandwidths are computed similarly. The quantification of the DRAM and Graphic domains utilization that are used in the proposed DVFS-aware performance classification are computed as follows:

$$DRAM_{Util} = \frac{DRAM_{Bandwidth}}{DRAM_{Peak\_Bandwidth}} \tag{3}$$

$$Graphics_{Peak\_Util} = \max_{All\_components} \{\delta_i \cdot Utilization_i\} \tag{4}$$



(a) Hierarchical clustering for the power characterization.

(b) Decision boundaries for the performance-aware NN classifier.

(c) Decision boundaries for the power-aware NN classifier.

Fig. 11. Performance and Power characterization of synthetic GPU benchmarks on NVIDIA's GTX TITAN X.

For the DVFS-aware power classification, the Graphics domain utilization is computed as follows:

$$\text{Graphics}_{\text{Agg\_Util}} = \frac{1}{\sum_i \omega_i} \sum_{i}^{\text{All\_components}} \omega_i \cdot \text{Utilization}_i \tag{5}$$

The values of the coefficients δ_i and ω_i, in Eqs. (4) and (5), are architecture specific and correspond to the weight of the component i to the overall performance (or power consumption) of the Graphics domain. The determination of these parameters can be done through proper modelling of the architecture, as previously suggested in [34] (valid for GPUs from the Fermi microarchitecture). Nevertheless, for the sake of simplification, the rest of this manuscript will assume $\delta_i = \omega_i = 1$.

3.2. Defining the class boundaries

After the execution and profiling of all the synthetic kernels, it is possible to group the benchmarks according to the DVFS impact on their execution. In order to separate them into clusters with similar characteristics, a clustering approach is used. Although multiple algorithms can be used in this context, techniques leading to spherical clusters were avoided (*e.g.*, K-Means). The hierarchical clustering technique was selected, since it directly specifies a hierarchy of groups and therefore simplifies the selection of the optimal number of clusters.

The set of features used by this clustering procedure are the resulting changes in the execution time (or power consumption), caused by memory and core frequency scaling, *i.e.* the value of the increase (or decrease) of the execution time (or power consumption) in each frequency level, relative to the one achieved at the reference frequency configuration (on GTX Titan X: $F_{Mem} = 3505$ MHz and $F_{Core} = 1164$ MHz). During this hierarchical clustering procedure, the euclidean distance (*distance metric*) is used to quantify the similarity between applications and the Ward's criteria (*linkage criteria*) is used for cluster merging, since it minimizes the within-cluster variance. The considered cut in the tree was performed in order to obtain the desired number of classes. As previously stated, in order to accurately identify the benchmarks that transition from compute-bound to memory-bound at each memory frequency level, the total number of performance classes is related with the number of memory frequencies available in the GPU device. On this particular GPU device, the performance classification will consider four classes of applications, namely two extreme classes **TA** and **TC** (hereafter referred to as **T1** and **T4**) and two middle classes **T2** and **T3**, corresponding to two subdivisions of **TB** in Fig. 4), associated with the two lower memory operating frequencies. For the power classification the suggested set of classes (4) will be considered (**P1**-**P4**). Fig. 11a presents the result of the hierarchical clustering using the power consumption features.

Finally, for each classification methodology (performance or power), after all the synthetic benchmarks have been assigned to one of the four clusters, a classifier must be trained to allow the classification of new (unseen) applications. Although multiple algorithms exist in the literature, neural networks (NN) are herein adopted due to their recognized capacity to model complex non-linear problems. In order to validate the classifier the set of synthetic benchmarks is randomly divided into two subsets: training-set and validation set. Hence, a neural networks topology with two fully-connected hidden layers was devised, where the sigmoid function is used as the activation function on all neurons. The performance network has layer sizes 40 and 10, while the power network has layer sizes 20 and 40. The input nodes are then fed with the values of the Graphics and Memory resource utilizations, with the neural network being trained to identify the classes assigned during the hierarchical clustering stage.

Figs. 11b and c depict the set of synthetic benchmarks used for the performance and power classifications, respectively, as well as the obtained classes and their respective boundaries. It can be seen that, for the performance classification the DVFS impact on the performance is solely dependent on the ratio $\frac{\text{DRAMUtil.}}{\text{GraphicsUtil.}}$, resulting in linear boundaries, while for the power classification this relationship is not linear.

It is important to stress that given the small sizes of the neural-networks, the time required to train is relatively small. In fact, the biggest effort in the training phase of our classification methods is in the execution of all the synthetic benchmarks

on the available frequency configurations. Nonetheless, this is a step that is performed once (offline) and therefore will not have any impact in the application execution time. During application execution, it is only necessary to run the neural network for inference, corresponding, on average, to less than a second on an Intel i7 4500U processor.

3.3. Extension to other architectures

While the proposed methodology mainly considers a NVIDIA GPU from the Maxwell microarchitecture, it is possible to make some considerations regarding its extension to other architectures. The proposed approach assumes that the GPU device has two independent frequency domains, which is the case for all recent GPU devices (both NVIDIA and AMD). Since the execution of the synthetic benchmarks tries to cover a considerable range of possible combinations of resource utilization from the two GPU domains, it is possible to adapt the executed set to the considered GPU device. Additionally, as it was previously mentioned, the suggested number of performance classes depends on the available number of memory frequency levels on the considered device (see Section 2.1). On the other hand, the number of suggested power classes, for any GPU with more than a single memory frequency levels is four (see Section 2.2). However, if future GPUs continue increasing the number of allowed memory and core frequency levels, it is expectable that the number of classes should be increased as well. When considering a different number of classes than those considered in this section, convenient changes should also be applied to the hierarchical clustering stage, specifically, in the cut-off point of the dendrogram tree, in order to obtain the desired number of clusters.

Accordingly, given the generality of the proposed methodologies, it is clear that they can be straightforwardly extended to other GPUs. In fact, given the current evolution trend of NVIDIA GPUs, it is expected that the proposed approach will only keep getting more interesting, since every new generation has been introducing larger ranges of allowed frequencies for both the core and memory domains.

4. Validation of the classification algorithms

To evaluate the proposed methodology, several CUDA-based application benchmarks from the Parboil [22], Rodinia [23], SHOC [24], Polybench [25] and CUDA SDK [26] suites (see Table 3) were executed on two GPU devices from different NVIDIA microarchitectures: GTX Titan X (Maxwell microarchitecture) and Titan Xp (Pascal microarchitecture). The Maxwell GPU (Pascal GPU) provides a user-level interface to scale the core operating frequency between different levels, within the 595-1164MHz (582-1911MHz) range, and the DRAM frequency in 3 (2) non-idle levels: 810, 3300 and 3505 MHz (5705 and 4705 MHz). Additionally, at the lowest memory frequency, *i.e.* 810 MHz, the GTX Titan X GPU allows to further downscale the core frequency into 21 additional levels, down to 135 MHz. In both GPUs, the default frequency setup corresponds to a dynamically managed frequency state, denoted as Auto-boost [37], used to boost the applications performance, by increasing GPU core and memory frequencies when sufficient power and thermal headroom is available. Notwithstanding, to apply the proposed performance and power classification model, it is herein assumed that the GPU operates at the highest user-controlled core and memory frequency levels (i.e., $F_{Mem} = 3505$ MHz and $F_{Core} = 1164$ MHz for the Maxwell GPU and $F_{Mem} = 5705$ MHz and $F_{Core} = 1911$ MHz for the Pascal GPU). Hence, each benchmark is only executed at such reference frequency level, where the performance counters values were measured using the NVIDIA Profiler [36]. Accordingly, based on: (i) the gathered values of the performance counters; (ii) the proposed classification scheme (see Fig. 9); and (iii) the classifiers trained using the synthetic benchmarks (see Figs. 11b and c), each application was subsequently classified in one performance class and one power class.

Finally, to validate and evaluate the attained classifications, each application was also executed at all user-controlled memory and core frequency levels, including with the activation of the Auto-boost feature. At each frequency configuration, the execution time of applications was accurately measured using the NVIDIA Profiler [36] and the GPU power consumption was obtained using the NVML [38] library, which is a C-based API that allows amongst other things, monitoring the state of the GPU. In some NVIDIA GPU devices, as is the case for both the considered devices, it is possible to use NVML to get the current power draw of the device. Through experimental testing, it was determined that the refresh rate of the values obtained using NVML is about 100ms. Accordingly, a GPU power measuring tool was implemented, which samples the GPU power draw every 25 ms. The kernels from the applications with smaller execution times were repeated until each kernel was executed for at least 1s, in order to increase the accuracy of the measured samples. The power consumption of each kernel was computed as the average of all gathered samples. For applications with multiple kernels, the total power consumption was obtained by averaging the consumption of each kernel weighted with the relative execution time of each kernel. In order to guarantee the integrity of the measurements taken, all applications (both synthetic and real) are executed 10 times. The values presented correspond to the average results over all the executed runs.

Sections 4.1 and 4.2 present the results of the proposed performance and power classification methodologies, respectively, while Section 4.3 presents the results of combining the two approaches in order to obtain a DVFS-aware energy classification of GPU applications.

Table 3
Summary of the considered application benchmarks.

Application	Suite	Input
Blackscholes	CUDA SDK	Default
conjugateGradientUM	CUDA SDK	N=1048577600
matrixMulCUBLAS	CUDA SDK	A(6720,5120)xB(5120,3520)
simpleCUFFT	CUDA SDK	signal_size=50000000
CUTCP	Parboil	Large
Histogram	Parboil	Large
LBM	Parboil	Long
2MM	Polybench	Default
3DCONV	Polybench	Default
3MM	Polybench	Default
CORR	Polybench	Default
COVAR	Polybench	Default
FDTD-2D	Polybench	Default
GEMM	Polybench	2048×2048
GRAMSCHM	Polybench	Default
SYRK	Polybench	Default
Backprop	Rodinia	655360
CFD	Rodinia	missile.domn.0.2M
Gaussian	Rodinia	2048×2048
Hotspot	Rodinia	1024, 2, 10,000
K-Means	Rodinia	3000000_34f.txt
LUD	Rodinia	8192×8192
ParticleFilter_Float	Rodinia	-x 256 -y 256 -z 80 -np 50,000
ParticleFilter_Naive	Rodinia	-x 256 -y 256 -z 80 -np 50,000
Srad_V1	Rodinia	4096×4096
Srad_V2	Rodinia	4096×4096
Streamcluster	Rodinia	Default
BFS	SHOC	-passes 100 -size 4
FFT	SHOC	-passes 100 -size 4
MD5Hash	SHOC	-passes 5 -size 4
Reduction	SHOC	-iter 1000 -passes 1 -size 4
S3D	SHOC	-passes 100 -size 4
Sort	SHOC	-passes 100 -size 4
SPMV	SHOC	-iter 400 -passes 1 -size 2
Stencil2d	SHOC	-passes 1 -num-iters 100 -size 4

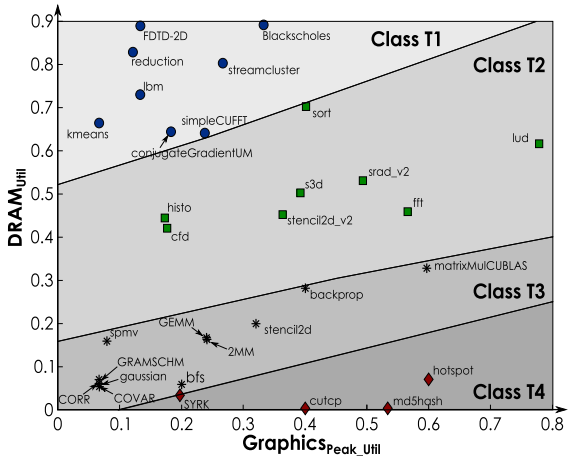
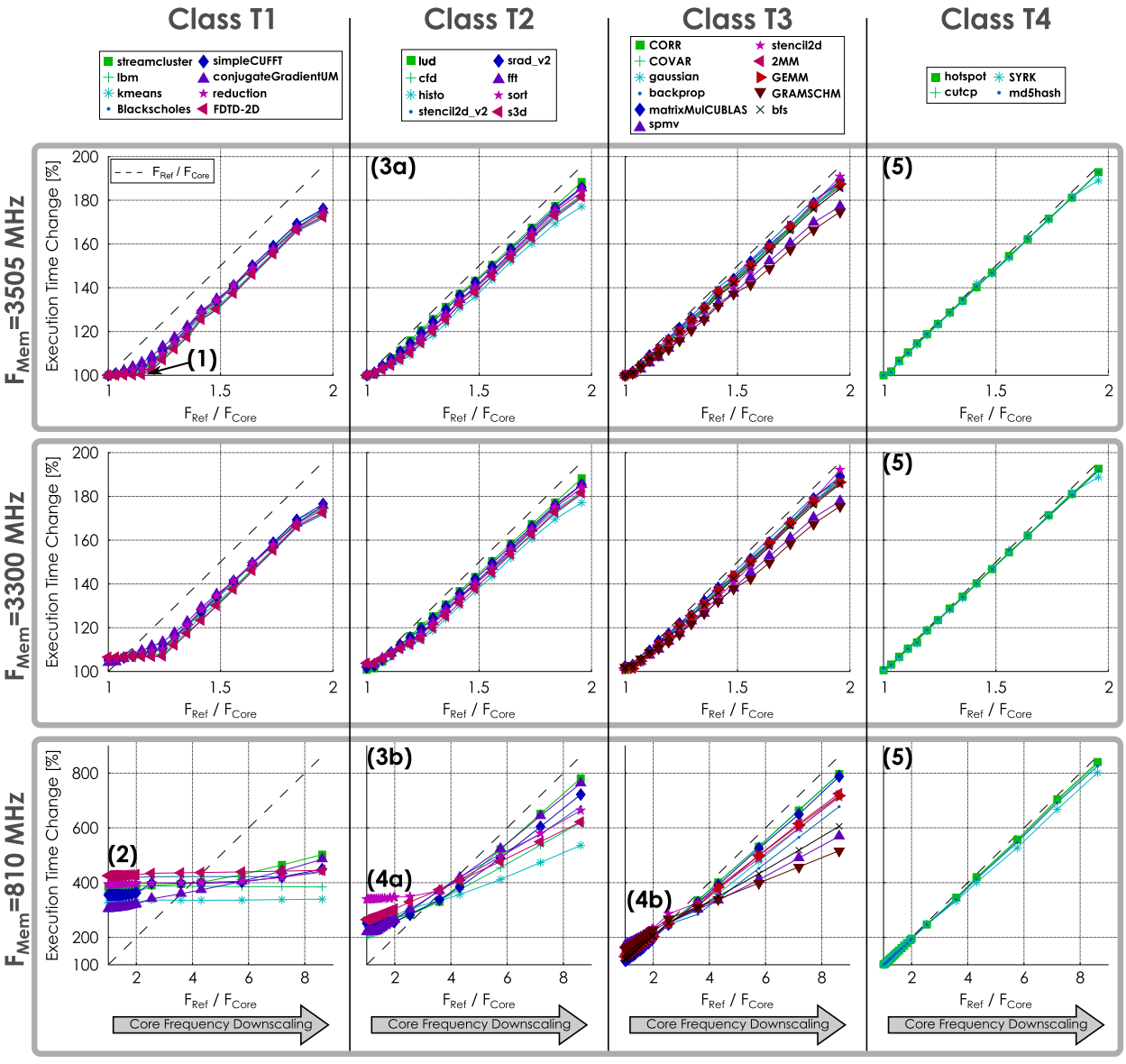


Fig. 12. DVFS-aware performance classification of the tested applications depending on the utilization of the DRAM and Graphics domains, on GTX Titan X (Maxwell) with $F_{Mem} = 3505$ MHz and $F_{Core} = 1164$ MHz.

4.1. Performance classification

Fig. 12 presents the computed values for the $DRAM_{Util}$ and $Graphics_{Peak\_Util}$ for all tested applications and how they are classified in the GTX Titan X GPU device according to the previously trained classifier (see Section 3). Additionally, Fig. 13 presents how the execution time of the applications of each resulting class is affected by the core and memory frequency downscaling, with the presented values being normalized to the execution time of each application at the reference frequency state. To simplify the analysis of the results, a dashed line was added to the graphs in order to show the expected



(1) From core frequencies 1164 MHz (FRef / FCore = 1) to 1013 MHz (FRef / FCore = 1.15) the execution time scales non-linearly with the core frequency.

(2) At memory frequency 810 MHz there is a significant drop-off in the performance of the applications (300%-500%).

(3) At memory frequency 3505 MHz **(3a)** the applications are compute-bound (performance scales with the core frequency), while at memory frequency 810 MHz **(3b)**, the applications performance does not scale directly with the core frequency.

(4) Applications from Class **T3 (4b)** are less sensible to changes in the memory frequency than applications from Class **T2 (4a)**, but still have a non-negligible performance drop-off when the memory frequency is set to 810 MHz (>10%).

(5) The execution time of all applications always scales with the core frequency. At memory frequency 810 MHz the performance drop-off is lower than 1%.

Fig. 13. DVFS-Aware Performance classes on GTX Titan X (Maxwell), with F_{Ref}=1164 MHz. The execution times are normalized to the value at F_{Mem} = 3505 MHz and F_{Core} = 1164 MHz.

execution time variation for a perfect *compute-bound* application, where $T \propto \frac{1}{F_{Core}}$. Finally, a set of markers **(1–5)** was were also added to the figure, to highlight the main takeaways from the results.

By analysing the results presented in these graphs, it can be concluded that although this device has three memory frequency levels, they are not uniformly separated. In fact, the two highest memory frequency levels (3505 and 3300 MHz) are so close to each other (only 6% different) that the performance results of all applications did not significantly change be-

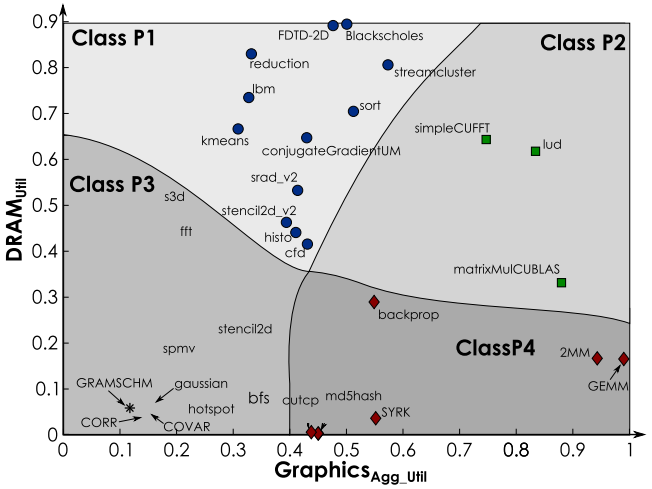


Fig. 14. DVFS-aware performance classification of the tested applications depending on the utilization of the DRAM and Graphics domains, on GTX Titan X (Maxwell) with $F_{Mem} = 3505$ MHz and $F_{Core} = 1164$ MHz.

tween these two levels. Notwithstanding, as there are more significant differences in the power domain, these two memory levels will be exploited in Section 5 to attain energy savings.

On the other hand, when analysing the classification results, a set of important observations are attained, which clearly justifies the partitions of the applications in these four classes. In particular, by observing marker **(1)**, it can be seen that when $F_{Mem} = 3505$ MHz, for the highest core frequencies (above 1013 MHz, i.e. $F_{Ref}/F_{Core} < 1.15$), the applications in class **T1** have a variation of their execution time that is lower than the core frequency variation. Hence, at this memory frequency, these applications are considered *memory-bound*. However, at $F_{Core} = 1013$ MHz ($F_{Ref}/F_{Core} = 1.15$), these applications start having their execution time scaling at the same rate as the core frequency (see Fig. 3a). When the memory frequency is scaled down to 810 MHz (see **(2)** in Fig. 13), there is a significant drop-off in the performance of these applications, which is expected since they have very high DRAM utilization (see **T1** in Fig. 12). Also, as expected at this lowest memory frequency, the range of core frequencies where these applications have negligible performance drop-off is increased, since it means that at a lowest memory frequency the core frequency needs to be further decreased for the Graphics components to become the performance bottleneck.

Class **T2** composes the applications that are *compute-bound* at $F_{Mem} = 3505$ MHz (see **(3a)** in Fig. 13), since their execution time always scales with F_{Core}). However, these applications are memory-bound at $F_{Mem} = 810$ MHz (see **(3b)** in Fig. 13), since for high core frequencies the execution time does not scale with F_{Core}. Additionally, the execution time of these applications also significantly increases when the memory frequency is decreased to the lowest level. However, in a lesser extent than the applications from cluster **T1** (compare **(2)** and **(3b)** in Fig. 13), which is consistent with their smaller utilization of the DRAM resources (see **T2** Fig. 12).

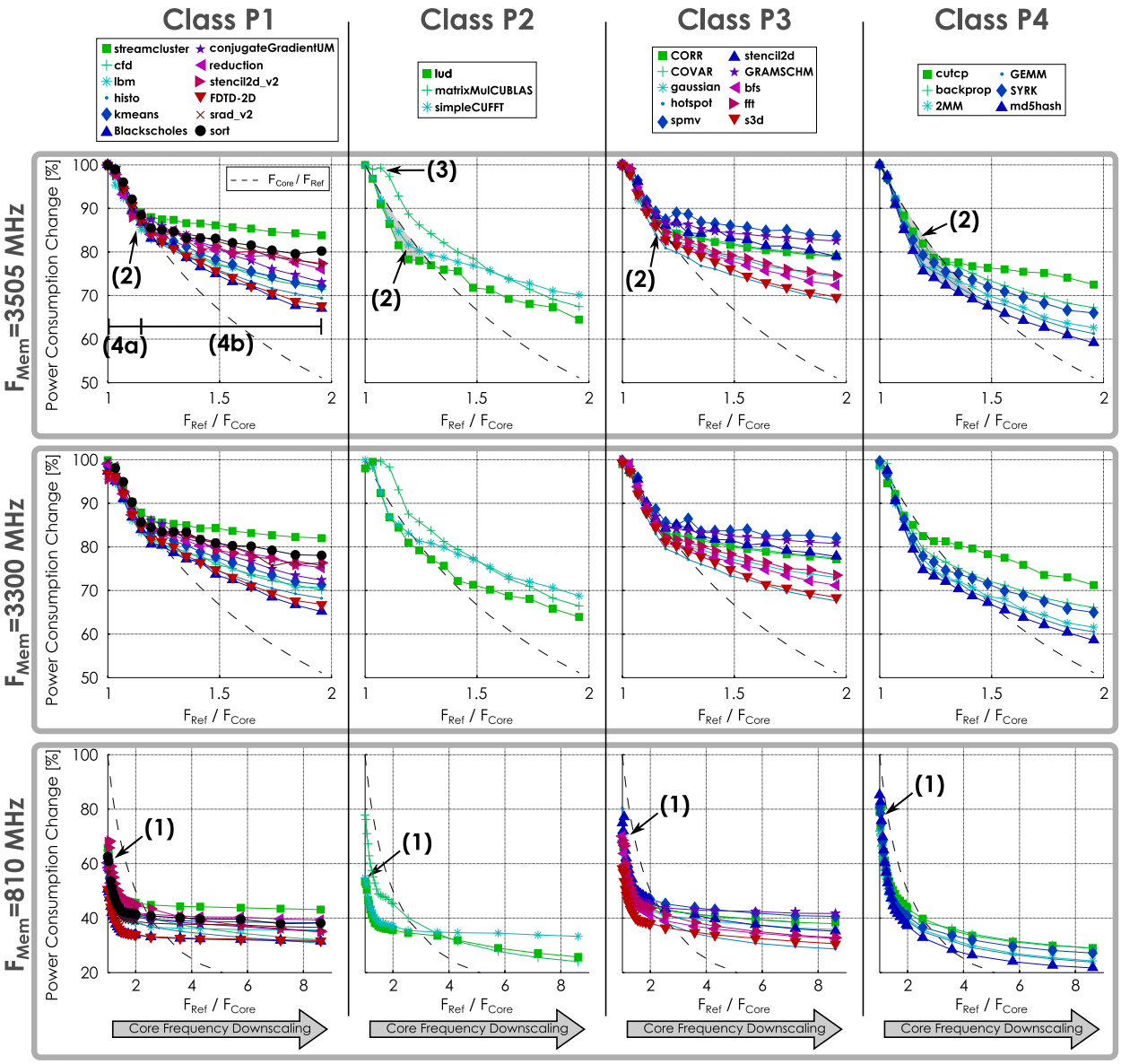
Furthermore, Class **T3** includes the applications that have their execution time scaling with both core and memory frequencies. However, while the same could be potentially said about the applications in the **T2** class, the applications in **T3** always have their execution times scale with the core frequency (even at $F_{Mem} = 810$), which is not the case for the applications in **T2** (compare **(4a)** and **(4b)** in Fig. 13). The behaviour of many of the **T3** applications is a result of their low occupancy of the GPU resources, resulting in a sequential execution of many instructions.

Finally, class **T4** comprises all the applications that can be considered *compute-bound* at all memory frequency levels (see the markers **(5)** in Fig. 13), as their execution time always scales inversely with the core frequency and never scales with the memory frequency (less than 1% performance change when the memory frequency is changed from 3505 MHz to 810 MHz).

Again, it is important to stress that this methodology allows the classification of GPU applications into classes that characterize their performance at all frequency levels, by using the information obtained from their execution at a single core frequency in a real hardware device.

4.2. Power-aware classification

Fig. 14 presents the utilization level of several components of the Graphics and Memory domains for the tested applications and how they are classified according to the proposed power-aware classifier (for the Titan X GPU device). The impact of DVFS in the power consumption of the applications of each class is presented in Fig. 15. As it was the case for the performance characterization, all values are normalized to the power consumption at the reference frequencies. The dashed line shows the core frequency variation at each level, i.e. applications that follow the line have power consumption scaling



(1) At memory frequency 810 MHz, applications from classes **P1** and **P2** have power-savings between 30% to 50%, while applications from classes **P3** and **P4** have power-savings between 10% to 30%.
(2) At memory frequency 3505 MHz, in the frequency range FRef/FCore <1.15, classes **P1**and **P3** have linear or sub-linear variation of the power consumption, while applications from classes **P2** and **P4** have super-linear variations.
(3) matrixMulCUBLAS application is hitting the device power cap, the power-consumption curve
(4) At memory frequency 3505 MHz, the power-consumption variation of applicaitons can be divided in two parts - **(4a)** and **(4b)** - with distinct slopes.

Fig. 15. DVFS-Aware Power classification on GTX Titan X (Maxwell), with F_{Ref}=1164 MHz. The power consumptions are normalized to the value at F_{Mem} = 3505 MHz and F_{Core} = 1164 MHz.

linearly with F_{Core}, while applications with a power consumption variation below this line have super-linear scalability with F_{Core}.

Each of the resulting classes displays different degrees of sensitivity to variations in the core and memory frequencies. In particular, when comparing the applications relative power consumption and when F_{Mem} decreases from 3505 MHz to 810 MHz (at $F_{Core} = F_{Ref}$) - compare the markers **(1)** in Fig. 15 - it can be perceived that applications from classes **P1** and **P2** have an higher degree of sensitivity to memory frequency changes, since they achieve power consumptions in the range from 70% to 50% of the reference consumption (corresponding to power-savings between 30% and 50%). On the other hand,

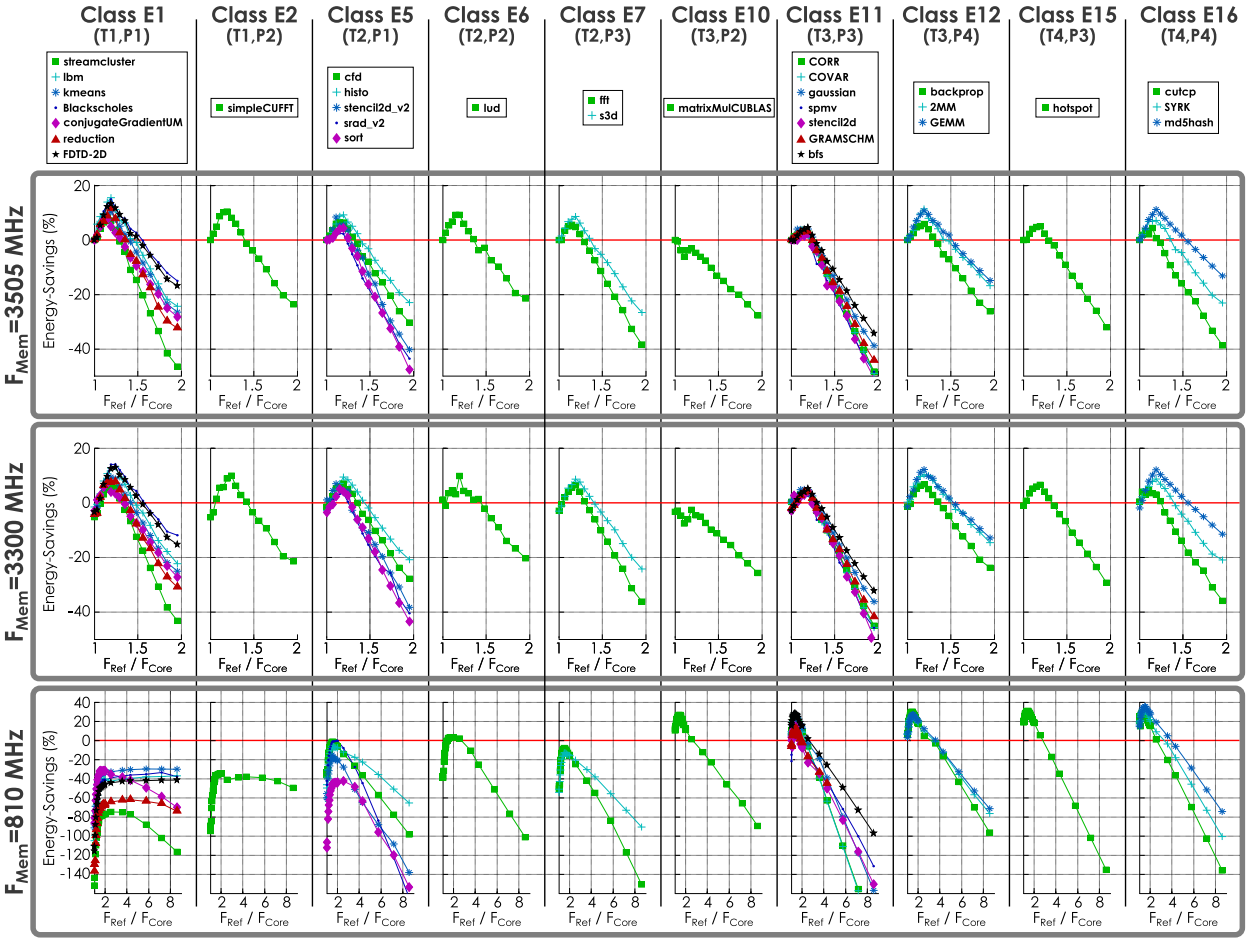


Fig. 16. DVFS-Aware energy classification on GTX Titan X (Maxwell).

applications from classes **P3** and **P4** display a lower degree of sensitivity to memory frequency changes, since they only achieve power-savings between 10% and 30%, when the memory frequency is decreased between the same values.

When analysing the applications power consumption sensitivity to core frequency variations, it can be concluded that, applications from classes **P1** and **P3** display a lower degree of sensitivity than applications from classes **P2** and **P4**. In particular, by comparing the applications power consumption variation at $F_{Mem} = 3505$ MHz with the dashed line (see the markers **(2)** in Fig. 15), it can be observed that the applications from classes **P1** and **P3** present a linear or sub-linear scaling with the dashed-line in the range $F_{Ref}/F_{Core} \in [1;\ 1.25]$. On the other hand, for the same core frequency range, applications from classes **P2** and **P4** show a super-linear power consumption drop-off.

The exception to such trend is presented in Class **P2**, for the *matrixMulCUBLAS* benchmark, which does not fit the profile of the remaining benchmarks (see the marker **(3)** in Fig. 15). Upon further inspection, it was observed that, during the execution of this kernel at the higher frequency levels, the power consumption of the GPU reaches very high levels (close to the device power cap of 275 W), thus suggesting the effect is due to internal GPU power control mechanisms. Since the values presented in Fig. 15 are normalized to the maximum power consumption, which for the *matrixMulCUBLAS* benchmark is limited by the device capabilities, the results will be skewed.

4.3. Energy-aware classification

Since, on the GTX Titan X GPU, the proposed methodologies give rise to four performance classes and four power classifications for this specific GPU device, it can be defined a set of 16 energy classes. However, it is observed that the set of tested benchmarks are classified into only 10 out of those 16 classes. Since both performance- and power-aware classifications are considering the resource utilization of the Graphics and Memory domains, even though they use different ways to combine the utilizations of the several components, many of the combinations between performance and power classes are very hard to achieve. The results of the obtained 10 energy-aware classes are presented in Fig. 16, where it is presented the energy-savings of all applications for all available frequency levels. The energy-saving (R) at frequency state (F_{Core}, F_{Mem}) is

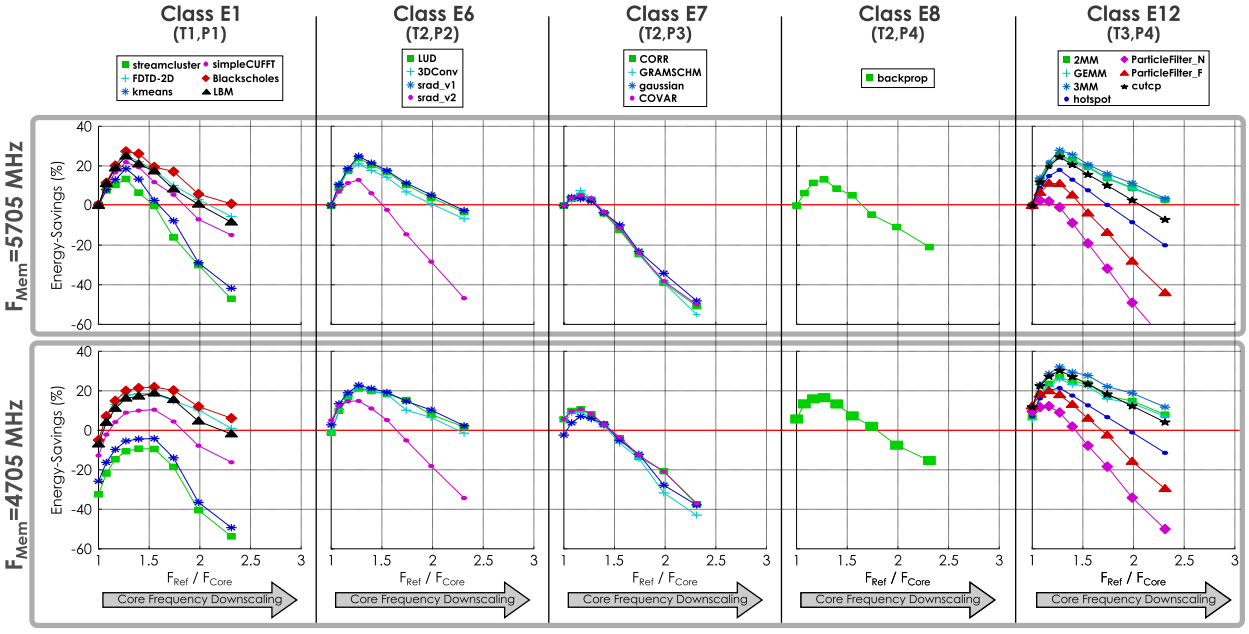


Fig. 17. DVFS-Aware energy classification on Titan Xp (Pascal).

computed in the following way:

$$R(F_{Core}, F_{Mem}) = \frac{E_{ref} - E(F_{Core}, F_{Mem})}{E_{ref}}, \tag{6}$$

where E_{ref} is the energy consumption at the reference frequencies ($F_{Mem} = 3505$ MHz ,$F_{Core} = 1164$ MHz).

Fig. 17 presents the obtained energy-aware classes of applications on the Titan Xp GPU (Pascal microarchitecture). Since in this device there are only two memory frequency levels, the proposed classification methodology will result in three performance classes. Combining these classes with the four power classes will ultimately result in 12 energy classes, some of which will again be unoccupied. The larger range of operating core frequencies available in this GPU device (compared with Maxwell GPU), results in a larger range of core frequencies where the applications with very high DRAM utilization (Class **E1**) decrease power consumption without losing performance: from 1911 MHz down to 1404 MHz (*vs.* 1164-1013 MHz on Maxwell). On the other hand, the lack of a low memory frequency level results in less interesting energy-savings opportunities for applications with high graphics utilization and low DRAM utilization (Class **E12**).

As it can be seen, by combining the results of the two proposed classification methodologies, a set of classes that successfully group together applications with similar energy consumption curves can be obtained. The result of this classification can be used in many different ways to maximize the energy-efficiency of computational systems, some of which are proposed in Section 5.

5. Application-aware DVFS

By providing a systematic mechanism to characterize the impact of DVFS on the energy-consumption of any GPU application, the proposed DVFS-aware classification methodologies create many interesting opportunities to improve the energy-efficiency of HPC systems. In fact, since the impact of DVFS on the energy consumption combines the performance and power classifications of the application, by considering the average trend among all applications in the class, it is possible to predict how the energy consumption of each application will change with the frequency scaling of each GPU domain. Sections 5.1 and 5.2 address two different ways to use these classification methodologies, namely to choose the best operating frequencies (core and memory), in order to solely maximize energy-savings or to try to maximize energy-savings without decreasing the performance more than a defined threshold. Section 5.3 addresses another possible usage of these classification methodologies, that combines the predicted frequency ranges with the energy-savings, in order to select the most convenient operating frequency for a scenario where multiple distinct applications are simultaneously running on the GPU device.

5.1. Optimal frequency for maximizing energy-savings

Just as the impact of DVFS in the energy consumption of applications within the same class is very similar between them, it can also be observed that, all applications of the same class have the same (or very similar) optimal operating

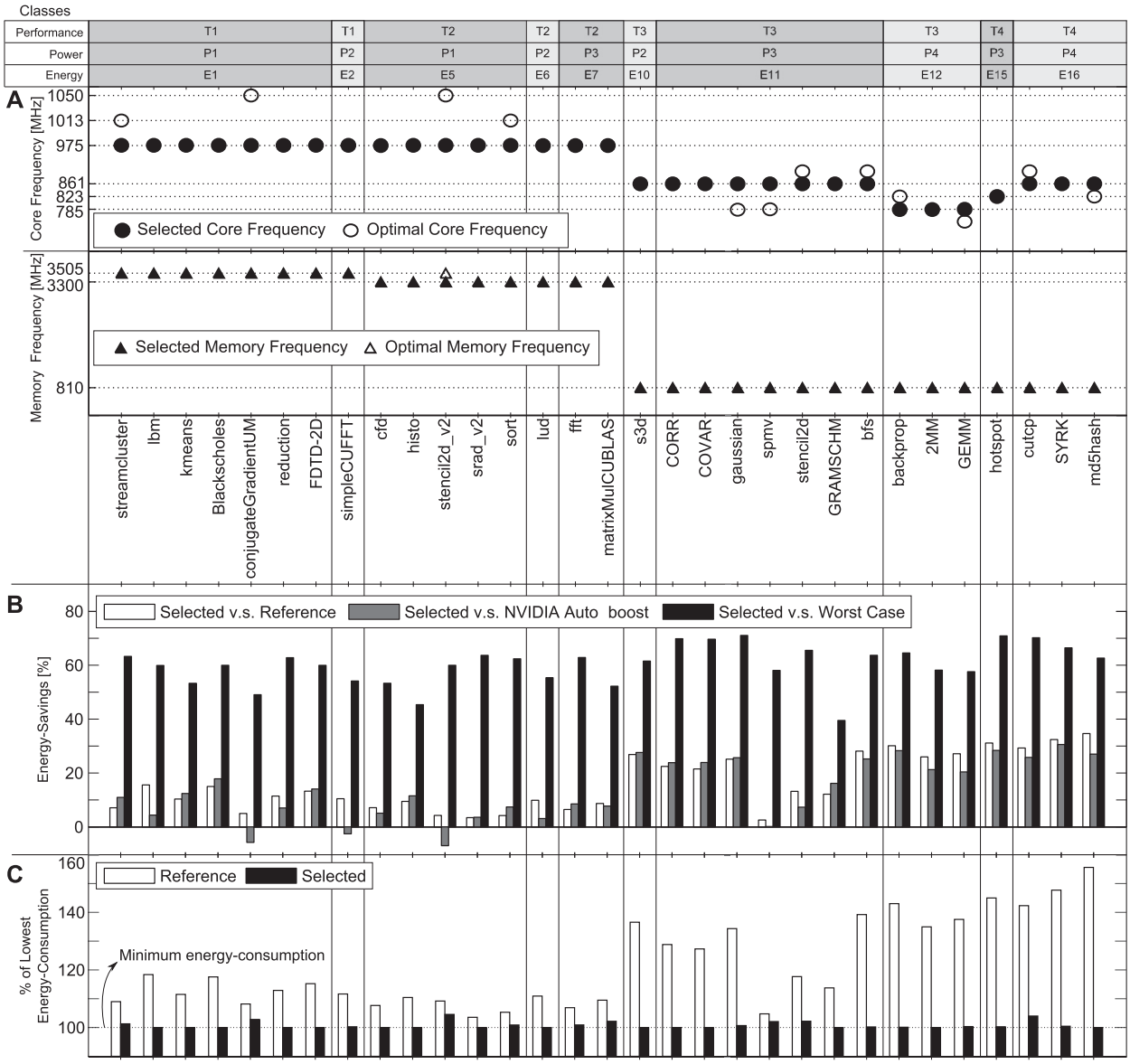


Fig. 18. Achieved energy-savings for the selected and optimal frequency levels, on GTX Titan X (Maxwell).

frequencies (memory and core frequencies that minimize the energy consumption on that device). Hence, by considering the average of the optimal frequencies from the synthetic benchmarks of each class (referred in this section as selected frequencies), it is possible to attain significant energy-savings. Accordingly, Fig. 18A depicts the optimal core and memory frequencies for each application executed on GTX Titan X, as well as the selected per-class core and memory frequencies.

As expected, applications with very high core utilizations (classes **E10, E11, E12, E15** and **E16**) are able to decrease the memory frequency to the minimum value, while applications with high DRAM utilization (classes **E1** and **E2**) require the memory frequency to be set to the highest value.

On the other hand, it is interesting to note that contrary to what one might expect, the applications from the class **E1** (High DRAM / Low Graphics) have higher optimal core frequencies than the applications from class **E16** (Low DRAM / High Graphics). The former group of applications are able to downscale the core frequency (with $F_{Mem} = 3505$ MHz) in order to achieve a lower energy consumption, while the applications from class **E16** are able to decrease the core frequencies even further (with $F_{Mem} = 810$ MHz), which is a result of the interaction between the two power components from Eq. 1. However, since these applications (class **E16**) are very compute-intensive they will have an high performance drop-off when running at the middle core frequency levels, which will be further analysed in Section 5.2.

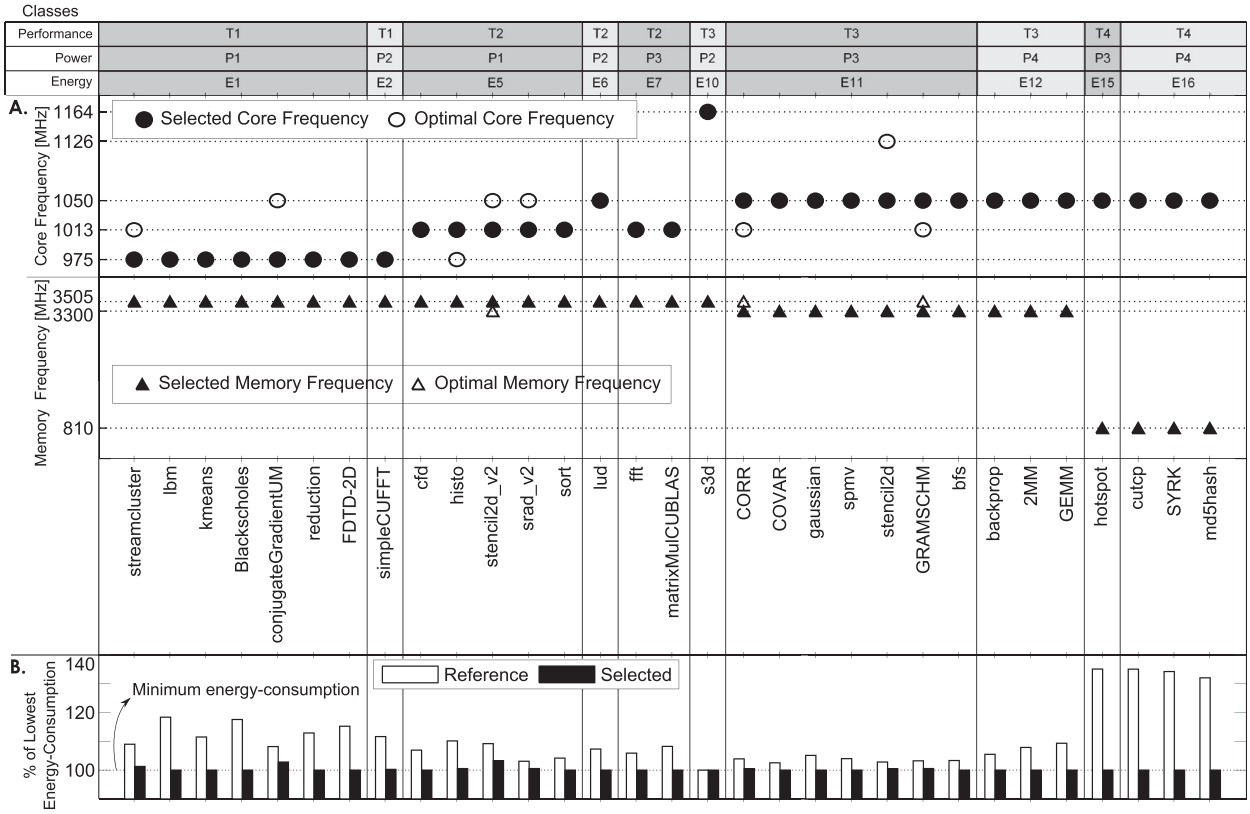


Fig. 19. Achieved energy-savings for the selected and optimal frequency levels, when the maximum allowed performance drop-off is of 10%, on GTX Titan X (Maxwell).

Fig. 18B presents the energy-savings obtained with the selected frequency levels in relation to: (i) the reference frequency; (ii) to the NVIDIA Auto-boost; and (iii) to the worst case, i.e. the highest energy-consumption of each application. As it can be seen, all applications can achieve lower energy consumptions than the ones obtained using the reference frequency (frequency state with highest performance), and only three applications would have lower energy consumption using NVIDIA's Auto-Boost. On average, using the selected pair of frequencies would wield 16% energy-savings compared with the reference, and 13% in relation to the Auto-Boost. Additionally, some applications (classes **E12, E15** and **E16**) can even achieve greater energy-savings, prompted by the fact that they can decrease the memory frequency to 23% of its reference value, making them achieve close to 36% energy-savings.

Although the proposed procedure to choose the selected frequencies does not guarantee optimal energy-savings, i.e. the selected frequency is not always equal to the optimal one, from the results presented in Fig. 18C, it can be seen that the difference between optimal configuration (horizontal line at 100%, representing the lowest energy consumption) and the one obtained using the selected frequencies is very small, with an average of 0.74% difference between the selected and the optimal energy consumptions.

5.2. Optimal frequency for energy v.s. performance trade-off

Another interesting situation to consider is the maximization of the applications energy-efficiency without sacrificing their performance. Considering a situation where at most a 10% drop-off in performance is allowed, and comparing it to the performance of the reference frequency state, the results presented in Fig. 19 can be obtained, for the GTX Titan X GPU. Again, by choosing the newly selected frequencies, all applications are able to achieve lower energy consumptions than the one at the reference state. Additionally, all obtained energy-savings using the selected frequency levels are at most 3% distant from the optimal energy-savings, with average energy-savings of 9% *versus* the reference and 7% *versus* the Auto-boost setup.

It is interesting to note that the applications with the greater gains are those with either a very high DRAM utilization and low Graphics utilization (class **E1**), or the applications in the other extreme (classes **E15** and **E16**). As it was previously seen, this happens mainly in the first type of applications (class **E1**), since at the highest memory frequency it is possible to downscale the core frequency while achieving very negligible performance drop-off. These types of applications can save up to 16% with less than 10% increase in their execution time. On the other hand, applications from classes **E15** and **E16** are able to run at the lowest memory frequency with negligible performance drop-off. For example, if just the memory

Table 4
Summary of energy-savings opportunities enabled by the proposed classification methodologies on GTX Titan X (Maxwell). The presented values correspond to the average results of all applications in that class on the selected frequency configuration comparing with the reference frequency configuration (F_{Mem} = 3505 MHz and F_{Core} = 1164 MHz).

GTX Titan X	Energy classes	E1	E2	E5	E6	E7	E10	E11	E12	E15	E16	ALL
Section 5.1	Energy-savings	11%	10%	6%	10%	6%	27%	18%	28%	31%	32%	**16%**
	Performance drop-off	4%	6%	11%	16%	13%	38%	66%	59%	38%	35%	**31%**
Section 5.2	Energy-savings	11%	10%	5%	7%	7%	0%	3%	26%	25%	25%	**9%**
	Performance drop-off	3%	6%	8%	8%	8%	0%	8%	9%	9%	10%	**7%**

Table 5
Summary of energy-savings opportunities enabled by the proposed classification methodologies on Titan Xp (Pascal). Reference frequency configuration: F_{Mem} = 5705 MHz and F_{Core} = 1911 MHz.

Titan Xp	Energy classes	E1	E6	E7	E8	E12	ALL
Section 5.1	Energy-savings	22%	20%	9%	17%	23%	**20%**
	Performance drop-off	0%	25%	18%	28%	27%	**18%**
Section 5.2	Energy-savings	22%	16%	4%	13%	18%	**16%**
	Performance drop-off	0%	8%	7%	8%	8%	**6%**

frequency was reduced to 810 MHz, while keeping the core frequency at 1164 MHz, all of the applications from these two classes would achieve near 20% energy-savings with an increase of their execution time of less than 1%. In the case where a 10% performance drop-off is allowed, they can achieve up to 26% energy-savings.

Table 4 summarizes the results of Sections 5.1 and 5.2, presenting the average energy-savings and performance trade-off of the applications from each energy class on the GTX Titan X GPU (Maxwell), when comparing the results at the selected frequency configuration with the ones obtained at the highest performance frequency (reference) configuration. Table 5 presents equivalent results for the Titan Xp GPU (Pascal). The Pascal GPU device has a larger range of core and a smaller range of memory operating frequencies compared to the ones in the considered Maxwell GPU device. This results in the observed larger energy-savings for the applications with high DRAM utilization (compare class **E1** on Pascal with classes **E1** and **E2** on Maxwell) and in the lower energy-savings for the applications with high Graphics and low DRAM utilizations (compare class **E12** on Pascal with classes **E12, E15** and **E16** on Maxwell).

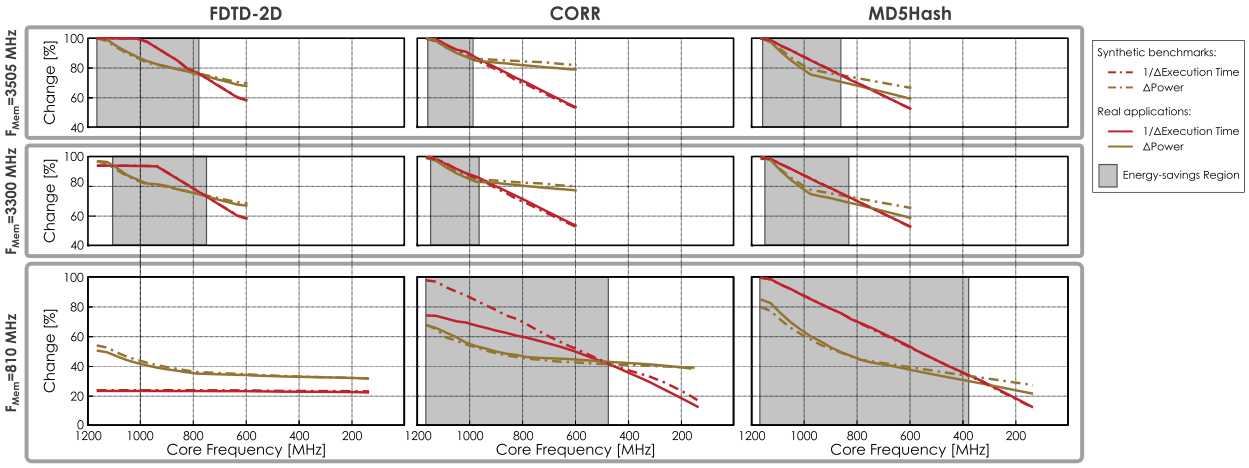
5.3. Energy savings ranges

In certain situations it may be more important to identify the range of setups corresponding to near optimal results (or in this case, the range of operating frequencies that result in energy-savings comparing with the reference state), rather than the optimal conditions for running a certain application. This can be especially useful in a situation where multiple distinct applications are to be executed in parallel, or even in a situation where a single application requires calling multiple heterogeneous kernels. In such a scenario, it is possible for the optimal frequencies to be different for each kernel. Hence, fixing the operating frequency at the optimal values for one of the kernels could actually result in a residual (or null) energy-saving when considering the remaining kernels. Furthermore, it is not always feasible to be constantly changing the operating frequency to the optimal values each time a new instance of the kernels are executed.

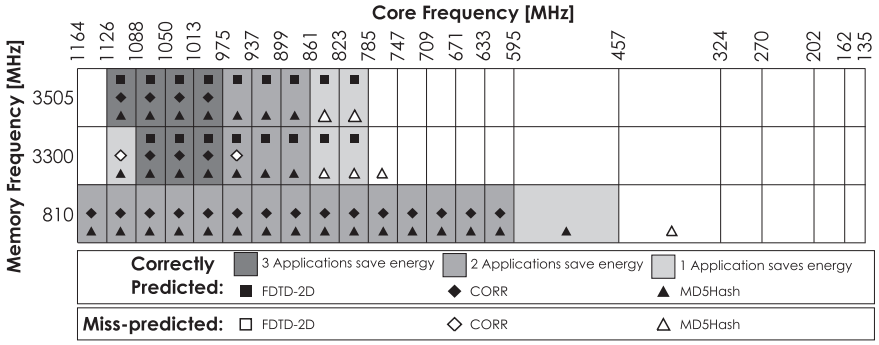
One solution for this specific problem is to find the operating frequency ranges where energy-savings are possible for each individual kernel and combining the collected ranges, in order to identify the core and memory frequency ranges that can achieve energy-savings for all the considered applications. Fig. 20a presents the execution time and power consumption changes of three applications (*FDTD-2D, CORR* and *MD5Hash*) from three distinct energy classes (classes **E1, E11** and **E16**, respectively). It is also depicted the average variation of the execution times and power consumption from the synthetic benchmarks of each class. The fact that they are from different energy classes means they most likely have different optimal frequencies. The objective is thus to find a range of frequencies where all three applications are able to save energy, when compared to the reference state (if such a range exists).

Using the execution time and power consumption variation curves of the synthetic benchmarks, depicted in Fig. 20a, it is possible to identify the energy-savings ranges, as they correspond to the frequency levels where the observed power variation is higher than the performance variation, i.e. the power decrease is more significant than the increase in the execution time. This can be seen in Fig. 20a by identifying the frequencies where the power variation curve is below the execution time variation curve.

With the frequency ranges that are identified for each application, it is possible to plot the histogram presented in Fig. 20b, where the intersection of the three energy-savings ranges is depicted. This figure also presents the difference to the actual energy-saving frequency ranges of each application. It can be seen that, while there are some miss-predicted frequency levels in the edges of the ranges, using the synthetic benchmarks still allows the identification of which frequency ranges enable all three applications to achieve a lower energy-consumption than the one achieved at the reference level. The



(a) Execution time and power consumption variations and the average variations of the synthetic benchmarks of their respective classes, for different values of core and memory frequencies.



(b) Histogram of the energy-savings ranges.

Fig. 20. Energy-aware analysis of three distinct applications (*FDTD-2D*, *CORR* and *MD5Hash*) on a GTX Titan X.

optimal range for a set of multiple applications may not even include the optimal frequencies of each individual application, which is exactly the case for this particular set of applications, since the optimal frequency for the *MD5Hash* benchmark is $F_{Mem} = 810$ MHz, while the *FDTD-2D* benchmark (and all applications from class **E1**) does not save energy at the lowest memory frequency.

6. Related work

Over the past few years, there has been a significant effort in the research community to study the effects of DVFS in GPGPU systems. Several techniques have been proposed, which can be divided in three main subjects: (i) works that study the impact of DVFS in the execution of applications; (ii) works that study classification techniques for GPU applications; and (iii) works that propose runtime models for the prediction of performance and/or power-consumption of GPU applications.

Regarding the effects of DVFS on distinct applications, Jiao et al. [39] studied the impact of core and memory frequency scaling on three applications with different characteristics on a GTX 280 GPU (Tesla microarchitecture). The authors observed that the impact of frequency scaling on the performance and power consumption was dependent on the applications characteristics, since some were more sensitive than others to the scaling of each frequency domain. Ma et al. [3] proposed an energy management framework for CPU-GPU heterogeneous systems, able to distribute the workload between the two systems, and to perform dynamic core and memory frequency scaling of the GPU. Results on a NVIDIA GeForce 8800 GPU (Tesla), allowed achieving about 6% of system (CPU+GPU) and 14.5% of GPU energy-savings. Ge et al. [4] also applied DVFS on a GPU-accelerated system with a Tesla K20c GPU (Kepler). The authors observed that the effects of DVFS on GPU are vastly different than those on the CPU, since the highest GPU frequencies always resulted in the best energy-efficiency (for their set of three tested applications). Mei et al. [2] studied the effects of scaling the voltage and frequency of the cores, as well as the scaling of the memory frequency, on the energy-efficiency of different applications on a GTX 560Ti GPU (Fermi). The authors observed an average of 20% reduction of energy consumption and concluded that the optimal setting (core voltage, core frequency and memory frequency) is dependent on the application characteristics. More recently, the authors

applied the same approach on a GTX 980 GPU (Maxwell microarchitecture) [27] with similar results, concluding that finding the optimal setting is a challenge that needs to be addressed, since significant energy-savings can be achieved by applying DVFS techniques. Sethia et al. [40] designed a dynamic runtime system to optimize the GPU kernel launch parameters and core and memory frequencies, depending on the application characteristics. The authors classified applications into three classes: *compute-, memory-* and *cache-intensive*, based on the GPUWattch characterization, achieving 15% energy-savings.

The majority of previous works on workload characterization on HPC systems (mainly CPUs) involve the combination of Principal Component Analysis (PCA) and hierarchical clustering [41–44]. As a consequence, some previous studies on workload characterization in the GPU-domain have tried to exploit similar approaches. In particular, Kerr et al. [8] characterized PTX workloads using a GPU simulator with the purpose of optimizing the applications. Che et al. [7] performed a diversity analysis of the Rodinia benchmark suite, using a GTX 480 GPU (Fermi). In the same trend, Adhinarayanan et al. [9] also provided an automated framework for characterizing and subsetting GPU workloads, by also relying on PCA and hierarchical clustering. However, while this approach has the advantage of reducing the dimensionality of the problem, it makes the understanding of each resulting class harder (from the computing architecture perspective) and does not necessarily result in an accurate energy-aware classification.

In our previous work [45], a similar methodology to the one used in the herein manuscript was proposed for DVFS-aware classification of GPGPU Applications. However, the work focused on a NVIDIA Tesla K40c GPU from the Kepler microarchitecture, a GPU with a much smaller range of allowed frequencies than the herein considered GTX Titan X and Titan Xp, with only four core frequency levels and a single memory level. For this reason, the maximum energy-savings that were achieved were only 8%. However, even with the small range of frequencies available, it is still possible to identify benefits in performing both performance and power characterization of GPU applications, as energy-savings opportunities are still available.

A different alternative approach towards DVFS consists in the development of accurate performance and/or power models that allow predicting the GPU behaviour under different voltage and frequency scenarios. GPU performance models are usually developed based on GPU pipeline analysis [10,13,14,18], trying to capture the execution characteristics of GPGPU applications. As an example, Nath et al. [14] developed a runtime analytical performance model able to predict the changes in performance when the frequency is scaled with an average accuracy of 4%. However, they require the addition of logic to the GPU scoreboard, making it infeasible to replicate in a real GPU device. Another common approach uses statistical methods and GPU performance counters [12,15], which while usually simpler to apply on real hardware, usually result in large prediction errors. Regarding the research on GPU DVFS runtime power modelling, one common approach relies on empirical methods, which require a break-up of GPU micro-architectures, and usually requiring analyzing the kernel binary code [18,34]. Moreover, these approaches are often product-specific and difficult to port to different devices. An alternative approach is using statistical methods, which rely on the measurement of hardware counters, used to create the runtime power model by either regression [46,47] or machine learning approaches [13,15]. While easier to implement, the regression based methods fail to capture the inherent non-linearity of modern GPU devices, resulting in large prediction errors (from 15% to 23.5% in [47]), while the neural-network solution better suits the complicated data dependencies, resulting however in higher complexity output models, with still non-insignificant prediction errors (10% in [15]).

Accordingly, while detailed performance and power models may ultimately produce more accurate results, the herein presented work shows that application classification, into a small number of DVFS-aware classes, is a rather convenient and viable approach not only to identify remarkable energy-savings opportunities, but also to achieve near-optimal results in terms of energy savings.

7. Conclusions

This work proposes a new methodology to classify GPU applications based on the resulting effects of DVFS on their execution time and power consumption. Although existing classification techniques are not targeted for this specific goals, often resulting in many wrongly classified applications when performance and power consumption are considered, the experimental results obtained with the proposed methodology demonstrate the benefits of this kind of approach. The proposed classification scheme allows application characterization on any modern GPU device, regarding the DVFS impact on its execution. It is based on a preliminary profiling of a set of synthetic benchmarks, followed by a training phase of a classifier. Once the classifier is trained, it is possible to classify any GPU application into a specific class that characterizes the variation of its performance (or power consumption) in the presence of frequency scaling (Core or Memory), by using the information obtained from the execution of each application at a single frequency state. The performance and power classes allow the definition of distinct energy-aware classes of applications that present a similar behaviour in the presence of DVFS. By using these energy-aware classes it was possible to define the optimal pair of operating frequencies (core and memory), resulting on the GTX Titan X GPU (Titan Xp GPU) on average energy-savings of 16% (20%), corresponding to a 0.74% (0.4%) deviation from the optimal, and in certain applications achieving energy-savings as high as 36% (32%). Additionally, the proposed methodologies also allowed to identify DVFS settings that can obtain up to 22% energy-savings at a cost of only 0.2% of performance loss. The analysis of the developed work allows us to conclude that even better opportunities for maximizing the applications energy-efficiency could be exploited if GPU manufacturers offered more liberty to choose the operating frequency of the different device domains (specially in the memory domain), since with more allowed frequencies available

between the maximum and minimum levels, higher energy-savings could be achieved for applications with heterogeneous usage of the device resources.

Acknowledgment

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), under grant SFRH/BD/101457/2014 and project UID/CEC/50021/2013.

References

[1] S. Herbert, D. Marculescu, Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, Low Power Electronics and Design (ISLPED), IEEE, 2007.
[2] X. Mei, L.S. Yung, K. Zhao, X. Chu, A measurement study of GPU DVFS on energy conservation, in: Proceedings of the Workshop on Power-Aware Computing and Systems, ACM, 2013.
[3] K. Ma, X. Li, W. Chen, C. Zhang, X. Wang, Greengpu: a holistic approach to energy efficiency in gpu-cpu heterogeneous architectures, in: 41st International Conference on Parallel Processing (ICPP), IEEE, 2012.
[4] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, Z. Zong, Effects of dynamic voltage and frequency scaling on a k20 gpu, in: International Conference on Parallel Processing, IEEE, 2013.
[5] G.D. Costa, J.M. Pierson, Dvfs governor for hpc: Higher, faster, greener, in: 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), IEEE, 2015.
[6] S. Zhuravlev, S. Blagodurov, A. Fedorova, Addressing shared resource contention in multicore processors via scheduling, ACM SIGARCH Computer Architecture News, ACM, 2010.
[7] S. Che, J.W. Sheaffer, M. Boyer, L.G. Szafaryn, L. Wang, K. Skadron, A characterization of the rodinia benchmark suite with comparison to contemporary CMP workloads, International Symposium on Workload Characterization (IISWC), IEEE, 2010.
[8] A. Kerr, G. Diamos, S. Yalamanchili, A characterization and analysis of ptx kernels, International Symposium on Workload Characterization (IISWC), IEEE, 2009.
[9] V. Adhinarayanan, W.c. Feng, An automated framework for characterizing and subsetting GPGPU workloads, International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2016.
[10] S. Hong, H. Kim, An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness, ACM SIGARCH Computer Architecture News, ACM, 2009.
[11] S.S. Baghsorkhi, M. Delahaye, S.J. Patel, W.D. Gropp, W.m.W. Hwu, An adaptive performance modeling tool for gpu architectures, ACM Sigplan Notices, Vol. 45, ACM, 2010.
[12] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, S. Kato, Power and performance analysis of GPU-accelerated systems, Presented as part of the 2012 Workshop on Power-Aware Computing and Systems, 2012.
[13] S. Song, C. Su, B. Rountree, K.W. Cameron, A simplified and accurate model of power-performance efficiency on emergent gpu architectures, International Symposium on Parallel & Distributed Processing (IPDPS), IEEE, 2013.
[14] R. Nath, D. Tullsen, The CRISP performance model for dynamic voltage and frequency scaling in a GPGPU, in: Proceedings of the 48th International Symposium on Microarchitecture, ACM, 2015.
[15] G. Wu, J.L. Greathouse, A. Lyashevsky, N. Jayasena, D. Chiou, GPGPU performance and power estimation using machine learning, 21st International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2015.
[16] N. Ardalani, C. Lestourgeon, K. Sankaralingam, X. Zhu, Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance, in: Proceedings of the 48th International Symposium on Microarchitecture, ACM, 2015.
[17] X. Ma, M. Dong, L. Zhong, Z. Deng, Statistical power consumption analysis and modeling for GPU-based computing, Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems (HotPower), 2009.
[18] S. Hong, H. Kim, An integrated GPU power and performance model, ACM SIGARCH Computer Architecture News, ACM, 2010.
[19] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, S. Matsuoka, Statistical power modeling of GPU kernels using performance counters, in: International Green Computing Conference (IGCC), IEEE, 2010.
[20] J. Chen, B. Li, Y. Zhang, L. Peng, J.k. Peir, Statistical GPU power analysis using tree-based methods, in: International Green Computing Conference (IGCC), IEEE, 2011.
[21] M. Rhu, M. Sullivan, J. Leng, M. Erez, A locality-aware memory hierarchy for energy-efficient gpu architectures, in: Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, ACM, 2013.
[22] J.A. Stratton, C. Rodrigues, I.J. Sung, N. Obeid, L.W. Chang, N. Anssari, G.D. Liu, W.W. Hwu, Parboil: a revised benchmark suite for scientific and commercial throughput computing, Center for Reliable and High-Performance Computing 127.
[23] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, S.H. Lee, K. Skadron, Rodinia: a benchmark suite for heterogeneous computing, International Symposium on Workload Characterization (IISWC), IEEE, 2009.
[24] A. Danalis, G. Marin, C. McCurdy, J.S. Meredith, P.C. Roth, K. Spafford, V. Tipparaju, J.S. Vetter, The scalable heterogeneous computing (shoc) benchmark suite, in: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, ACM, 2010.
[25] L.N. Pouchet, Polybench: the polyhedral benchmark suite, URL: http://www.cs.ucla.edu/~pouchet/software/polybench/.
[26] NVIDIA, GPU Computing SDK, 2016, Available: http://developer.nvidia.com/gpu-computing-sdk.
[27] X. Mei, Q. Wang, X. Chu, A survey and measurement study of gpu dvfs on energy conservation, Digital Commun. Netw. (2017).
[28] G. Keramidas, V. Spiliopoulos, S. Kaxiras, Interval-based models for run-time DVFS orchestration in superscalar processors, in: Proceedings of the 7th ACM international conference on Computing frontiers, ACM, 2010.
[29] B. Rountree, D.K. Lowenthal, M. Schulz, B.R. De Supinski, Practical performance prediction under dynamic voltage frequency scaling, in: Green Computing Conference and Workshops (IGCC), IEEE, 2011.
[30] O. Kayıran, A. Jog, M.T. Kandemir, C.R. Das, Neither more nor less: optimizing thread-level parallelism for gpgpus, in: Proceedings of the 22nd international conference on Parallel architectures and compilation techniques (PACT), IEEE Press, 2013.
[31] E. Konstantinidis, Y. Cotronis, A practical performance model for compute and memory bound gpu kernels, in: 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), IEEE, 2015.
[32] R. Gonzalez, B.M. Gordon, M.A. Horowitz, Supply and threshold voltage scaling for low power cmos, IEEE J. Solid-State Circuits (1997).
[33] J.A. Butts, G.S. Sohi, A static power model for architects, in: Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture, ACM, 2000.
[34] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N.S. Kim, T.M. Aamodt, V.J. Reddi, GPUWattch: enabling energy optimizations in GPGPUs, ACM SIGARCH Computer Architecture News, ACM, 2013.
[35] NVIDIA CUDA Compiler Driver NVCC, 2016, Available: http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc (v8.0. 2016.).
[36] NVIDIA, CUDA Profiler Users Guide, 2016, Available: http://docs.nvidia.com/cuda/profiler-users-guide/ (v8.0. 2016).
[37] NVIDIA, increase performance with GPU boost and k80 autoboost., 2014, Available: https://devblogs.nvidia.com/parallelforall/increase-performance-gpu-boost-k80-autoboost/.

[38] NVIDIA, NVML API Reference Guide, 2015, Available: http://docs.nvidia.com/deploy/nvml-api/index.html (vR352. 2015).
[39] Y. Jiao, H. Lin, P. Balaji, W.c. Feng, Power and performance characterization of computational kernels on the gpu, in: International Conference on Green Computing and Communications (GreenCom), 2010 IEEE/ACM and International Conference on Cyber, Physical and Social Computing (CPSCom), IEEE, 2010.
[40] A. Sethia, S. Mahlke, Equalizer: dynamic tuning of gpu resources for efficient execution, in: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2014.
[41] J.J. Yi, R. Sendag, L. Eeckhout, A. Joshi, D.J. Lilja, L.K. John, Evaluating benchmark subsetting approaches, International Symposium on Workload Characterization (IISWC), IEEE, 2006.
[42] K. Hoste, L. Eeckhout, Comparing benchmarks using key microarchitecture-independent characteristics, IEEE International Symposium on Workload Characterization, IEEE, 2006.
[43] K. Hoste, L. Eeckhout, Microarchitecture-Independent Workload Characterization, IEEE Micro, 2007.
[44] Z. Jia, J. Zhan, L. Wang, R. Han, S.A. McKee, Q. Yang, C. Luo, J. Li, Characterizing and subsetting big data workloads, IEEE International Symposium on Workload Characterization (IISWC), IEEE, 2014.
[45] J. Guerreiro, A. Ilic, N. Roma, P. Tomás, Performance and power-aware classification for frequency scaling of GPGPU applications, Proceeding of the Workshop for Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar), 2016.
[46] A. Karami, S.A. Mirsoleimani, F. Khunjush, A statistical performance prediction model for opencl kernels on nvidia gpus, The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013), IEEE, 2013.
[47] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, M. Peres, Power and performance characterization and modeling of gpu-accelerated systems, 28th International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2014.