electronics

MDPI

Article

Improving GPU Performance with a Power-Aware Streaming Multiprocessor Allocation Methodology

Zois-Gerasimos Tasoulas * and Iraklis Anagnostopoulos

Department of Electrical and Computer Engineering, Southern Illinois University, Carbondale, IL 62901, USA; iraklis.anagno@siu.edu

* Correspondence: zoisgerasimos.tasoulas@siu.edu

Received: 15 October 2019; Accepted: 27 November 2019; Published: 1 December 2019

Abstract: Graphics processing units (GPUs) are extensively used as accelerators across multiple application domains, ranging from general purpose applications to neural networks, and cryptocurrency mining. The initial utilization paradigm for GPUs was one application accessing all the resources of the GPU. In recent years, time sharing is broadly used among applications of a GPU, nevertheless, spatial sharing is not fully explored. When concurrent applications share the computational resources of a GPU, performance can be improved by eliminating idle resources. Additionally, the incorporation of GPUs in embedded and mobile devices increases the demand for power efficient computation due to battery limitations. In this article, we present an allocation methodology for streaming multiprocessors (SMs). The presented methodology works for two concurrent applications on a GPU and determines an allocation scheme that will provide power efficient application execution, combined with improved GPU performance. Experimental results show that the developed methodology yields higher throughput while achieving improved power efficiency, compared to other SM power-aware and performance-aware policies. If the presented methodology is adopted, it will lead to higher performance of applications that are concurrently executing on a GPU. This will lead to a faster and more efficient acceleration of execution, even for devices with restrained energy sources.

Keywords: GPU; streaming multiprocessor; performance; power; allocation; spatial multitasking

1. Introduction

Application demands for computational resources continuously rise. To meet these demands, software engineers used to take advantage of the improvement on hardware technology, e.g., smaller transistor dimensions, higher clock frequencies, and chips with numerous processing cores. Unfortunately, high transistor density and shrinking transistor dimensions have reached a point where improvement in performance cannot be further achieved. For instance, heat dissipation, power consumption, and material degradation are problems that impede further performance gains from newer hardware generations [1,2]. To increase performance for parallel applications, system engineers have turned to the usage of hardware accelerators. Graphics processing units (GPUs) have been extensively utilized as accelerators, providing significant improvements in performance. GPUs were initially developed to accelerate graphics rendering. Since their first introduction, the necessary tools and frameworks have been developed, allowing programmers to leverage the computational power of GPUs in various application domains. Some examples of areas where GPUs are being used are in the acceleration of neural networks, autonomous cars, and cryptocurrency mining.

GPUs consist of *streaming multiprocessors* (SMs) which in their turn consist of streaming processors (SPs). SPs are the computational units that execute application threads. To achieve high performance, GPUs operate under the single instruction multiple data (SIMD) paradigm. Under this paradigm, all the

SPs of an SM either execute the same instruction or some SPs can be idle. From the application point of view, applications are divided into computational kernels. Threads of a kernel are organized into warps and warps are mapped on the SPs of an SM. The initial GPU programming frameworks introduced *temporal multitasking* for GPU applications. Temporal multitasking meant that multiple kernels of the same or different applications can be launched to the GPU. Even though multiple kernels are launched, only threads of one kernel are active at a specific moment. The rest of the launched kernels remain idle at an SM, waiting for the executing kernel to finish or halt, due to requests for memory data or user input. Temporal multitasking is well researched and is a standard for the majority of commercial GPUs. An additional developed technique for GPUs is *spatial multitasking*. With this technique, more than one kernels share the computational resources of a GPU, simultaneously. That means that multiple kernels can be active at the same time, with each kernel possessing a certain number of SMs. Applications have diverse computational needs, some applications can be computationally intensive while others can be memory-bounded. If all the SMs of a GPU are allocated by one application, GPU resources can be underutilized as a result of the limited needs of an application. For this reason, spatial multitasking can prove a valuable mechanism in order to achieve high utilization of the SMs and as a result yield high performance for a GPU.

A plethora of research works have explored the area of improving performance for GPUs. The authors of [3–5] develop allocation methodologies to improve GPU throughput without considering spatial multitasking among the applications. Furthermore, in [6–9] authors propose methodologies to improve performance by considering the scenario of concurrent applications. Different approaches are followed in [10–12], where architectural and hardware support is needed to achieve improved performance. The aforementioned works improve performance for GPUs but do not consider in their methodology the power consumption of the GPU, or the effect that higher performance will have on average power consumption.

The issue of power consumption is acquiring an important role in GPU usage [13]. On the one hand, GPUs are being introduced as accelerators to mobile and embedded devices, such as smart phones. These devices are powered by sources with limited capabilities. As a result, it is of high importance that GPUs function in a power efficient way in order to be able to boost the performance of devices with constrained power resources. On the other hand, GPU power efficiency is important for sites and projects of large scale. For example, on a GPU farm that mines continuously for cryptocurrencies, access to power resources is not restricted. Nevertheless, unless GPUs help to accelerate computation in a power efficient way, the usage of GPUs will become inefficient and alternative accelerators will need to be explored [14,15]. Additionally, excessive power consumption affects reliability. According to [16], increasing temperature by 15 °C can cause increased failure rates by up to 2×.

Numerous research articles explore power efficiency for GPUs, and propose ways to reduce power consumption. In [17], the authors present a method to improve power efficiency, based on fusing GPU kernels. The method relies on combining data independent kernels from multiple applications and executing them together. The results present improved performance and energy reduction when the kernel fusion method is used. Nevertheless, the experimental results consider only two applications thus, the method needs to be tested against more combinations of applications, to verify that it can provide power efficiency for a variety of applications. Authors in [18] present a technique that optimizes power and aging in general purpose GPUs (GPGPUs). Although in terms of power, the developed technique achieves improvements, it lacks two aspects. It does not focus on improving performance, for the majority of the benchmarks, the technique demonstrates similar or worse performance than the default execution technique. In addition, the developed technique in [18], does not leverage spatial multitasking. Allowing two or more applications to run together can yield further improvements both in terms of power and performance. In [19], authors present a predictive model for GPU applications. The presented model predicts the execution time and calculates power consumption. Based on the predicted execution time and calculated power consumption, the

model decides the optimal number of running cores. The experimental results show that the presented method does not improve power efficiency for every application. Additionally, multiple concurrent applications were not considered during the execution scenario. Finally, in [20], solutions to improve power efficiency for GPUs are presented. The presented solutions, however, require hardware support, and as a result they are difficult to incorporate into existing systems. Moreover, authors in [20] do not consider concurrent GPU applications in their execution scenario.

In this paper we present a methodology to allocate SMs for GPU applications. The methodology aims at improving GPU throughput while at the same time it provides power efficiency. To achieve these goals, the proposed methodology incorporates the following features:

- Given a queue of applications to execute on the GPU, the methodology decides the optimal way of pairing the applications, in order to minimize intra-application slow-down.
- The applications of the pairs are executed concurrently on the GPU, sharing resources.
- Based on profiling information, the methodology decides the appropriate number of SMs that should be allocated by each application. The decision is based on power efficiency and improved GPU throughput.

The methodology is evaluated on benchmarks that traverse graphs, launch pattern-recognition tasks, execute neural networks, apply fast Fourier transforms, and render images. These are tasks commonly executed on mobile devices, either by the operating system or user applications. Using different scenarios and different combinations of applications from the aforementioned categories, the developed methodology proves that it can offer improved performance combined with power efficiency for realistic usage scenarios on mobile devices.

The rest of the article is organized as follows. In Section 2 we present, in detail, the developed power-efficient methodology. In Section 3 we present and discuss the experimental results obtained, comparing the developed methodology with state-of-the-art performance and power efficient allocation methodologies. Section 4 provides a discussion on the obtained experimental results. Finally, Section 5 concludes the article.

2. Materials and Methods

In this section, we present the motivation to develop this methodology, together with the aspects of GPU execution that we leverage to achieve improved performance and power efficiency. Additionally, the allocation methodology is presented in detail in this section. The methodology is divided in two parts, the application profiling part, that takes place only once, and is necessary in order to collect the appropriate information for each application. The second part is the run-time part that decides how to pair applications together, and determines the number of SMs that need to be allocated by each application.

The goal of the presented methodology is to improve GPU throughput (T) while providing power efficiency. As T we define $\frac{Ins_{tot}}{Cyc_{tot}}$, where Ins_{tot} is the total number of instructions executed on the GPU, and Cyc_{tot} are the total cycles that the GPU needed to execute the Ins_{tot} instructions.

2.1. Motivation

The number of SMs that are available to an application affects the IPC (intructions per cycle) and the power consumption of the application [9,18]. As shown in [7], processes can be classified into categories according to their behavior. Specifically, applications can be categorized into 4 classes: *compute intensive*, *memory intensive*, *cache intensive*, and *memory-cache intensive*. Applications from different categories utilize resources differently, for example, compute intensive applications benefit from many SMs being available to the application. On the contrary, memory intensive applications depend more on bandwidth availability, as they have to fetch and store big amounts of data from and to the memory. As a consequence, the number of available SMs does not significantly determine the performance of memory intensive applications. An additional important observation made in [9,18] is

that certain applications will drop their IPC, if the system provides more SMs to the application, above a certain threshold. As a result, there is an incentive to restrain the available SMs to an application, in order to achieve greater performance. This incentive can prove useful when we execute more than one applications concurrently.

Spatial multitasking is a way to efficiently utilize GPU resources [6,21]. Memory intensive applications or memory intensive kernels that belong to compute intensive applications usually do not utilize all of the available SMs because they mainly load and store data. As a consequence, SMs remain idle and the computational cores of the GPU are underutilized, resulting in low IPC. If SMs are spatially shared among multiple applications, the computational resources of the GPU can be continuously used, eliminating the underutilization. Unfortunately, as demonstrated in [4,6,7,9], combining applications together is not trivial. Applications competing for the same resources cause slow-down, which will lead to lower GPU throughput and lower performance than executing applications on their own, even if the hardware is underutilized in the single application scenario. For that reason a key part of our methodology is the pairing of applications before execution. To achieve the best results in terms of application pairing we utilize the pairing methodology presented in [7]. The first part of our methodology is collecting the necessary information per application, in order to be able to characterize it.

In terms of power efficiency exploration, we conducted experiments using the Rodinia [22] benchmarks as applications. We executed 13 single applications for various configurations of SMs and plot the power efficiency per applications in Figure 1. To execute the applications, we used the GPGPU-Sim simulator [23] and used configurations consisting of 5 up to 60 SMs. As power efficiency we define $\frac{IPC}{Average\ power}$, where IPC is the application's IPC and $Average\ power$ is the application's average power in $Watts$. In Figure 1 we can see various behaviors of applications. To elaborate, we distinguish the following groups of applications according to their power efficiency behavior. The first group, where applications like `GUPS` and `BLK` belong, contains applications that their power efficiency remains the same or demonstrates minimal changes through the different SM configurations. The second group is formed by applications that improve their power efficiency when more SMs are available to them, example applications of this group are `HS` and `BP`. The third and last group of applications is formed by applications that either drop their power efficiency as more SMs are available to them, e.g., `LUD`, or applications that have mixed and unpredictable behavior as the number of available SMs changes, e.g., `SPMV`. Based on the observations made from Figure 1, we conclude that knowing an application's behavior helps allocate the number of SMs that will yield increased power efficiency. When two applications are co-executing, it can be beneficial in terms of power efficiency to reduce the available SMs of one application and offer them to the other. Instead of dividing SMs equally among two applications, the best performance and power-efficiency results come by considering the individual behavior of each application and adjusting accordingly the number of allocated SMs per application.

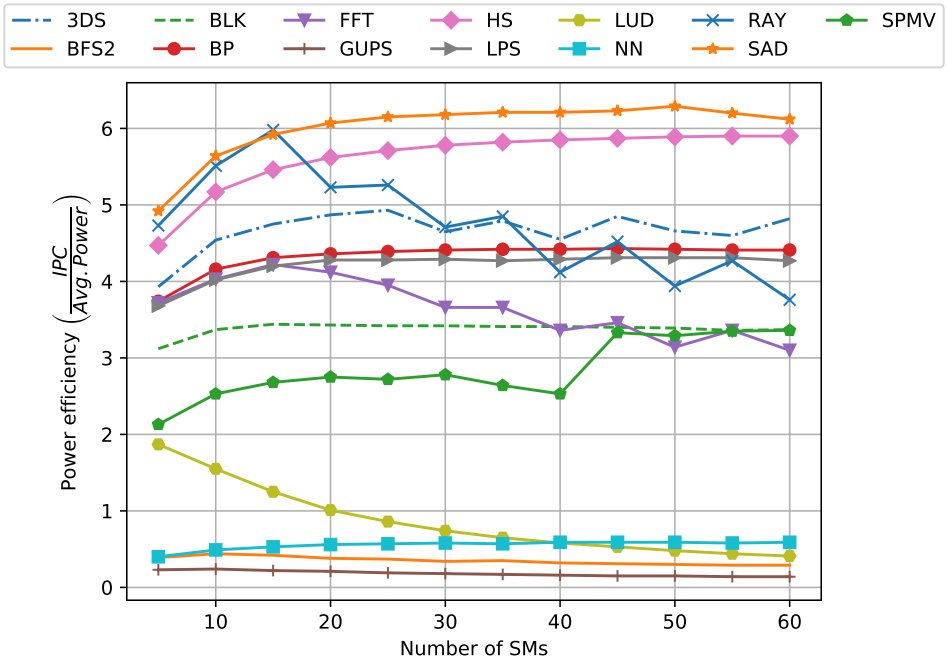


Figure 1. Power efficiency per application for different streaming multiprocessor (SM) configurations.

2.2. Collecting Application Information

The initial part of the proposed methodology consists of information collection about the applications. Considering a group of applications that will be executed in pairs, the methodology needs to collect information per application. The information is collected by executing applications individually, and is essential in order to decide the pairing that will minimize slow-down. The profiling stage happens before execution starts and the application information is collected only once. This information can be reused in the future and even shared between users of same GPU models, thus it does not introduce a significant overhead in the usage of the developed methodology.

The goal of this stage is to collect the necessary information that will allow application characterization, according to [7], and provide power consumption profiling for different SM configurations. With the collected information, applications are categorized in the following four classes: *memory intensive* (M), *memory-cache intensive* (MC), *cache intensive* (C), and *computation intensive* (A).

To classify the applications, we execute them individually on the GPU, using different configurations of SMs, starting form 5 up to 60 SMs, with a step of 5 SMs. The GPU setting we used has a total of 60 SMs, but for the scenario of two co-executing applications, we determine that no application can execute on more than 55 SMs, in order to avoid resource starvation problems. The information we collect per application is stored in an array data structure, and it is formulated as a tuple. Given an application A, the stored tuple for A is: $A(n,\ opt,\ IPC,\ MB,\ L_2 \rightarrow L_1,\ MCR,\ pow,\ powEf)$. The variable notation stands for:

- n, the number of SMs for the current configuration,
- opt, the number of SMs for the configuration that yields optimal power efficiency for A,
- IPC, the *IPC* of A for n SMs,
- MB, the *Memory Bandwidth* of A for n SMs,
- $L_2 \rightarrow L_1$, the *level* 2 *to level* 1 *cache memory bandwidth* of A for n SMs,
- MCR, the *Memory to Computational instructions Ratio* of A for n SMs,
- pow, the *average power consumption* of A for n SMs, and
- $powEf$, the *power Efficiency* of A for n SMs.

The characterization of an application to a class is based on the numbers collected during the execution of the application on 60 SMs. Apart from application characterization, the collected information is used during the run-time phase in order to decide the number of SMs to allocate for each application.

2.3. SM Allocation Policy

The run-time part of the developed methodology aims at improving performance and achieving power efficiency by combining application IPC with average power consumption on the decisions it makes. In order to achieve its goals, the run-time part executes the following tasks:

1. it pairs applications,
2. it partitions SMs between the two executing applications, and
3. it allocates the appropriate number of SMs for each application, clock-gating SMs if a surplus exists.

2.3.1. Pairing Applications

Given a queue of incoming applications to be executed on the GPU, before execution starts, it is essential to decide which applications will be executed together. As stated previously, concurrent application execution can yield higher performance, as hardware can be utilized efficiently. Nevertheless, the decision of which applications to execute concurrently is not trivial as two applications might compete for the same resources, causing a performance degradation. In order for the run-time system to choose the best application matching, the information collected off-line (Section 2.2) is utilized, together with the ILP methodology, presented in [7].

The run-time system begins by pairing applications together in order to minimize slow-down. From the applications that exist in a queue, the proposed methodology matches together the applications that will result in minimum overall slow-down. The next step is to send the pair with the lowest slow-down for execution. In case more applications arrive to a queue while a pair is executing, the execution will not be halted. At the time that an executing pair finishes its execution, in case more applications have arrived, the run-time system will recalculate the optimal matching of applications and will send the pair with the lowest slow-down to be executed on the GPU.

2.3.2. Partitioning SMs

Partitioning the SMs between the two executing applications is equally important as matching application to co-execute. The number of SMs that are available to an application affects significantly the performance and the power efficiency of an application, as shown in Figure 1. In this step, the two applications that will be executed together are sent to the GPU and the host system (CPU) has to decide how many SMs each application will receive.

The SM partitioning algorithm is described concisely in Algorithm 1 and presented here. Given two applications, $App1$ and $App2$, a GPU with κ_{tot} total SMs, this step of the methodology has to decide the value of two integers, κ_{App1} and κ_{App2}, representing the number of SMs that will be available to each application. The partitioning algorithm implements the following logic:

- If $opt_{App1} + opt_{App2} \leq \kappa_{tot}$, then both applications will receive the number of SMs that yields the best power efficiency results for each application. In case there is a surplus of SMs, the remaining SMs will be clock-gated in the next step of the methodology, allocation of SMs (Subsection 2.3.3). By satisfying the needs of the applications in this case, we ensure that each application will have available the resources to achieve high performance combined with power efficiency.
- In case $opt_{App1} + opt_{App2} > \kappa_{tot}$, one of the applications has to retreat on the number of SMs that it is requesting. If $opt = \kappa_{tot}$ for an application and a given GPU, this application has to first request fewer SMs, as we will have two concurrent application executing. We experimentally chose to set the step of reducing SMs to 5. For example, for a GPU with 60 SMs, if an application demonstrates

optimal power efficiency for 60 SMs, it will initially drop the SMs it requests to 55. If after one or both the applications have reduced their requests but still $\kappa_{App1} + \kappa_{App2} > \kappa_{tot}$, one application has to continue reducing the number of SMs it requests. To achieve high GPU throughput and power efficiency, we chose to favor the application with the highest power efficiency. To elaborate, if we assume that $powEf_{App1} > powEf_{App2}$ for κ_{App1} and κ_{App2} SMs respectively, we chose to favor $App1$ over $App2$. We need to clarify that, at this point of the algorithm, $\kappa_{App1}/\kappa_{App2}$ can have values of opt_{App1}/opt_{App2} or lower, since opt_{App1} and/or opt_{App2} might be equal to the total number of SMs on the GPU. To determine the final partitioning of SMs, in the example we use, $App2$ will continue reducing the SMs it requests, by a step of 5, until $\kappa_{App1} + \kappa_{App2} \leq \kappa_{tot}$. When the aforementioned inequality becomes true, the κ_{App1} and κ_{App2} numbers are propagated to the next phase of the run-time mechanism, SM allocation.

Algorithm 1 SM allocation policy

procedure PARTITIONING SMS(App1, App2, κ_{tot}, ProfInfo)
    $\kappa_{App1} = 0$
    $\kappa_{App2} = 0$
    $case = 0$
    if $powEf_{opt_{App1}} > powEf_{opt_{App2}}$ **then**
        $case = 1$
    else
        $case = 2$
    if $opt_{App1} + opt_{App2} \leq \kappa_{tot}$ **then**
        $\kappa_{App1} = opt_{App1}$
        $\kappa_{App2} = opt_{App2}$
    else
        if $opt_{App1} == \kappa_{tot}$ **then**
            $\kappa_{App1} = \kappa_{tot} - 5$
        if $opt_{App2} == \kappa_{tot}$ **then**
            $\kappa_{App2} = \kappa_{tot} - 5$
    while $\kappa_{App1} + \kappa_{App2} > \kappa_{tot}$ **do**
        if $case == 1$ **then**
            $\kappa_{App2} - = 5$
        else
            $\kappa_{App1} - = 5$
    return κ_{App1}, κ_{App2}

procedure ALLOCATING SMS(κ_{App1}, κ_{App2})
    if $\kappa_{App1} + \kappa_{App2} < \kappa_{tot}$ **then**
        `Clock gate SMs` $\kappa_{App1} + \kappa_{App2}$ *to* $\kappa_{tot} - 1$
    `Allocate SMs` 0 *to* $\kappa_{App1} - 1$ `to application App1`
    `Allocate SMs` κ_{App1} *to* $\kappa_{App1} + \kappa_{App2} - 1$ `to application App2`

2.3.3. Allocating SMs

This stage of the methodology accomplishes two tasks, given the numbers κ_{App1} and κ_{App2} from the partitioning step, it directs SMs to be clock-gated if there is a surplus after the partition, and it allocates the appropriate number of SMs per application. When the execution process reaches this stage, the host system (CPU) allocates the first κ_{App1} SMs to $App1$ (SMs 0 to $\kappa_{App1} - 1$) and the next κ_{App2} SMs to $App2$ (SMs κ_{App1} to $\kappa_{App1} + \kappa_{App2} - 1$). In case there are remaining SMs that are not allocated by either application, the allocation stage sends a signal to these SMs to clock-gate them.

That way, the surplus of SMs will remain inactive during the execution of a specific application pair. With this technique we can achieve further power gains as we lower the static power consumed by SMs that do not contribute in the execution of a specific pair of applications. The decisions made by the SM allocating stage can also be seen in Algorithm 1.

3. Results

To evaluate the effectiveness of the methodology we developed, we conducted extended experiments and present the results in this section. For the experiments we used a modified version GPGPU-Sim simulator [23] that allows concurrent application execution. The simulator allows fast prototyping and we can modify various parts of the GPU system to implement our methodology. GPGPU-Sim is a cycle accurate simulator that can execute CUDA C code and allows users to modify several architectural characteristics of a GPU. The applications we used in our experiments were taken from the Rodinia benchmarks [22]. To obtain power measurements for the applications of the experiment we used GPUWattch [24], which is a simulator that integrates with GPGPU-Sim and provides power consumption profiles for the simulations run on GPGPU-Sim. We provide details about the GPU configuration we used during the experiments in Tables 1 and 2. We evaluated the developed methodology using two GPU configurations, one follows the NVIDIA Fermi micro-architecture and the other the NVIDIA Pascal micro-architecture. Apart from that, the developed methodology is applicable to other micro-architectures too, since the only GPU specific information it needs is the total number of SMs.

Table 1. Graphics processing unit (GPU) configuration for the Fermi micro-architecture.

Fermi GPU Micro-Architecture			
Number of SMs	60	Core frequency	700 MHz
Warps per SM	48	Blocks per SM	8
Shared Memory	48 kB	L1 Data cache	16 kB per SM
L1 Instr. cache	2 kB per SM	L2 cache	768 kB
Warp scheduler	GTO [25]		

Table 2. GPU configuration for the Pascal micro-architecture.

Pascal GPU Micro-Architecture			
Number of SMs	60	Core frequency	1417 MHz
Warps per SM	64	Blocks per SM	32
Shared Memory \L1 cahce	64 kB per SM	L2 cache	4096 kB
Warp scheduler	GTO [25]		

To evaluate the developed methodology we measure four performance metrics, GPU throughput, average power consumption, power efficiency $\left(\frac{Throughput}{Average\ power}\right)$, and total energy consumption. We use nine queues of applications in order to account for all the different application behaviors as well as evaluate the developed methodology over diverse scenarios of workloads. Specifically, we use:

- an MC-oriented workload queue, consisting mainly of memory-cache intensive applications,
- an M-oriented workload queue, consisting mainly of memory intensive applications,
- a C-oriented workload queue, consisting mainly of cache intensive applications,
- an A-oriented workload queue, consisting mainly of computationally intensive applications,
- an equally constructed workload queue, where all the classes of applications are represented equally, and
- four workload queues that consist of random mixes of Rodinia benchmarks.

In order for a queue to be characterized as oriented towards a specific class, at least 60% of the applications comprising the queue need to belong to the specific class. For the execution scenario, we considered that if an application of a pair finishes execution, it is re-spawned until the slower application finishes execution.

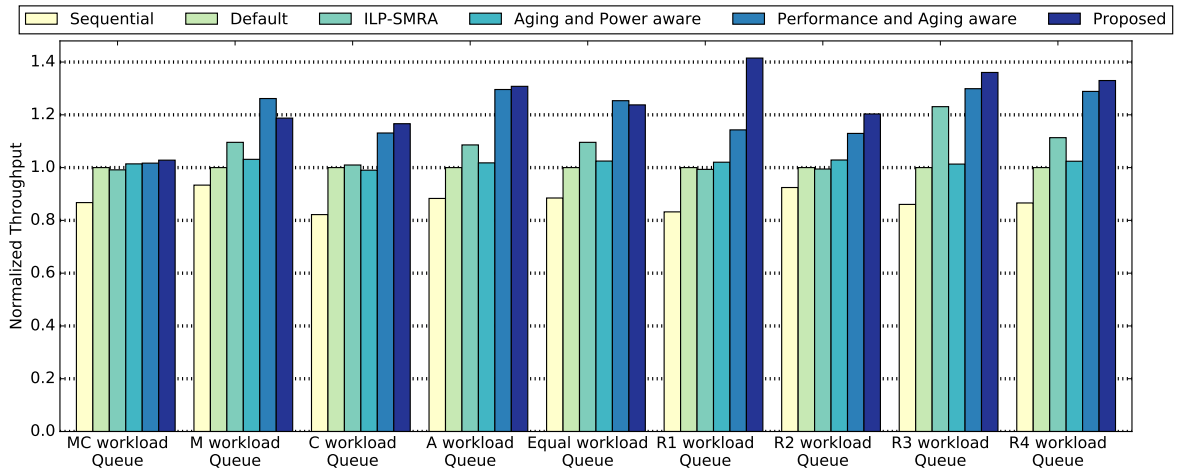
The developed methodology is compared against five different approaches.

1. *Sequential*: During this approach, applications are executed individually on the GPU and all the SMs of the GPU are available for each application. This is not a methodology that utilizes concurrent execution of applications, nevertheless it offers an opportunity to evaluate further the benefits and challenges that concurrent application execution creates.
2. *Default*: With this methodology, applications are paired in a first-come first-served (FCFS) way. The SMs are divided equally between the two co-executing applications. This approach does not consider the performance or power behavior of applications, and as a result does not apply any optimization during execution. We use this methodology as a base-line comparison for our experiments.
3. *ILP-SMRA*: this approach is presented in [7]. It is a methodology that focuses on improving performance by pairing applications together in a way that minimizes slowdown. Additionally, this methodology tracks performance during execution. If the individual IPC of an application suffers from low performance or an application seems to not utilize its available SMs, ILP-SMRA undertakes the dynamic reallocation of SMs, in order to improve overall performance.
4. *Aging and Power aware*: this is a modified version of the methodology presented in [18]. The original methodology works for a single executing application and aims at improving power consumption and limiting aging degradation. To adjust this methodology for concurrent applications, we pair applications according to their arrival time in the queue. In other words, the applications are paired in a FCFS way. We choose to divide SMs equally between the two co-executing applications. For the specific GPU set-up, each application has 30 SMs available. After profiling the applications, each application uses the number of SMs that yields the minimum execution time (information extracted during profiling). In case some SMs remain unused by one or both applications, these SMs are clock-gated during execution. For example, if we need to execute application A and application B concurrently, and assuming $opt_A = 20$ and $opt_B = 40$, application A will get 20 SMs, application B will get 30 SMs (SMs are distributed equally between the applications), and 10 SMs will be clock-gated.
5. *Performance and Aging aware*: this methodology is presented in [9]. It uses ILP to pair applications together and profiles applications before execution. It allocates SMs at the kernel level, thus the number of SMs that an application uses might change during the different kernels of the application. This methodology focuses on improving performance and balancing aging among SMs. Nevertheless, power efficiency is not a metric that this methodology takes into consideration.

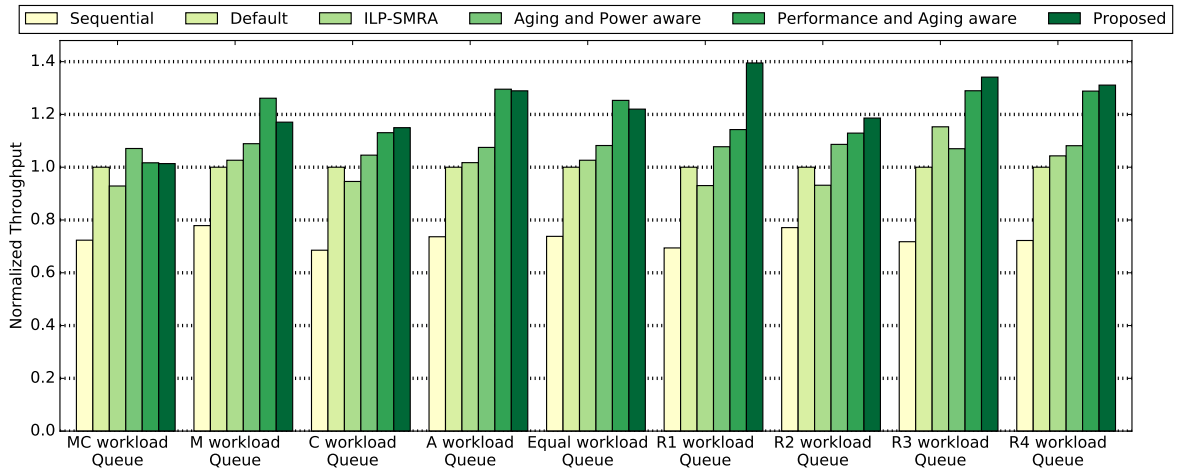
At this point we would like to clarify that our approach is not lacking applicability even though it is not aging-aware. Our target systems are systems that will benefit from improved performance but have limited power resources. At the same time, according to [26], certain modern mobile devices, such as smart phones, have a lifespan of less than 3 years. As a result, aging is not a major factor for these devices because by the time aging effects will start affecting performance, due to user practices, the devices will most likely be withdrawn from usage.

In Figure 2 we present the experimental results for GPU throughput. In Figure 2a we present the results for the Fermi micro-architecture. Additionally, in Figure 2b we present the results for the Pascal micro-architecture. We compare the six methodologies on nine application queues. The general comment we make is that the proposed methodology outperforms the rest of the methodologies in seven out of the nine queues for the Fermi micro-architecture and for five out of the nine queues for the Pascal micro-architecture. The Performance and Aging aware methodology achieves better performance than the proposed methodology for the M and Equal workloads for the Fermi micro-architecture. The Aging and Power aware methodology achieves higher throughput for

the MC workload, and the Performance and Aging aware achieves higher performance for the M, A and Equal workloads for the Pascal micro-architecture. The Sequential approach demonstrates low throughput, even lower than the Default approach. This behavior of low throughput explains the incentive to explore concurrent execution of applications. On average, the proposed methodology outperforms the default methodology by 25% and the ILP-SMRA methodology by 18% for the Fermi micro-architecture. The great difference with the default methodology is explained by the fact that the default methodology pairs applications together under the FCFS principle and does not consider the individual performance characteristics of each application in order to apply mechanisms that will improve throughput. Furthermore, the proposed methodology achieves better performance results than the ILP-SMRA methodology. Even though both methodologies use ILP to pair applications, the proposed methodology outperforms ILP-SMRA for the following reasons. ILP-SMRA commences execution by dividing the SMs equally among the applications. During run-time, it monitors performance and adjusts SMs per application. Following this approach, valuable time and performance can be lost from the start of execution until SMRA triggers a rearrangement of SMs. Additionally, for a number of SMs to be transfered from one application to the other, all scheduled blocks on the SM have to finish execution before blocks from the other application can be scheduled to this SM. This can lead to underutilization of SMs, resulting in lower throughput than the proposed methodology.



(**a**) Fermi architecture



(**b**) Pascal architecture

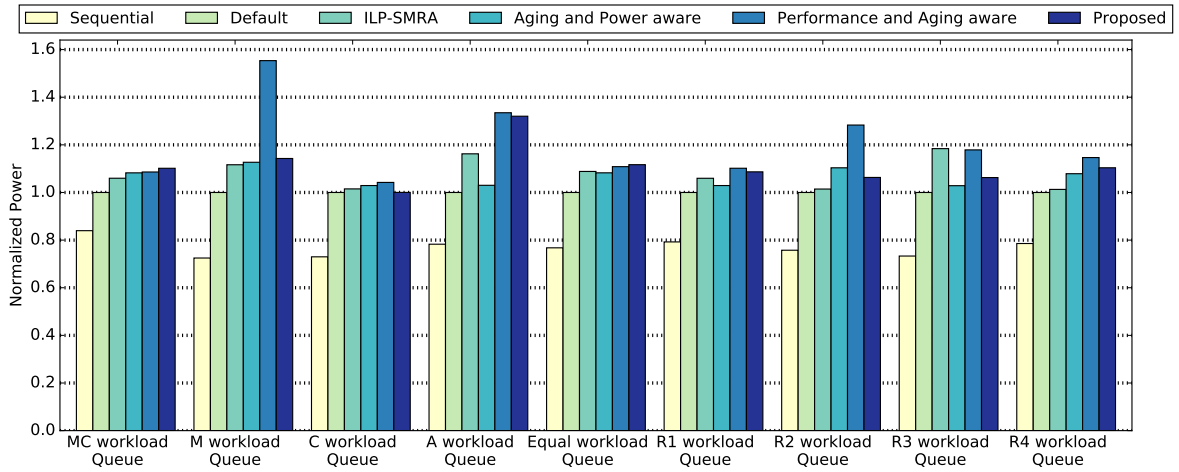
Figure 2. Normalized throughput for nine queues, comparing the developed methodology with existing approaches.

The proposed methodology achieves higher throughput compared to the Aging and Power aware methodology by 16% on average for the Pascal micro-architecture. This is anticipated as the latter does not have an immediate goal of improving throughput and focuses mainly on power and aging improvements. We observe that for the *MC*, *R*1, and *R*2 workloads, the Aging and Power aware methodology achieves higher performance than the ILP-SMRA methodology. This result might seem unexpected as the latter methodology is performance-oriented but the Aging and Power aware methodology in not. The results for these three queues can be explained by benchmarks that individually might fail to achieve improved performance for a specific methodology. As a result, the appearance of such benchmarks in a workload queue can cause a performance oriented methodology to under perform, compared to a methodology that is not performance oriented. Lastly, compared to the Performance and Aging aware methodology, the proposed methodology achieves a 3% higher throughput on average for the Pascal micro-architecture. We receive this improved performance on average because the Performance and Aging aware methodology applies a 10% performance margin loss, in order to make more allocation combinations available, explore them, and achieve balanced aging among the SMs. Due to the fact that the proposed methodology does not have to apply a 10% IPC loss per application, it utilizes more efficient SM configurations and results in higher GPU throughput.

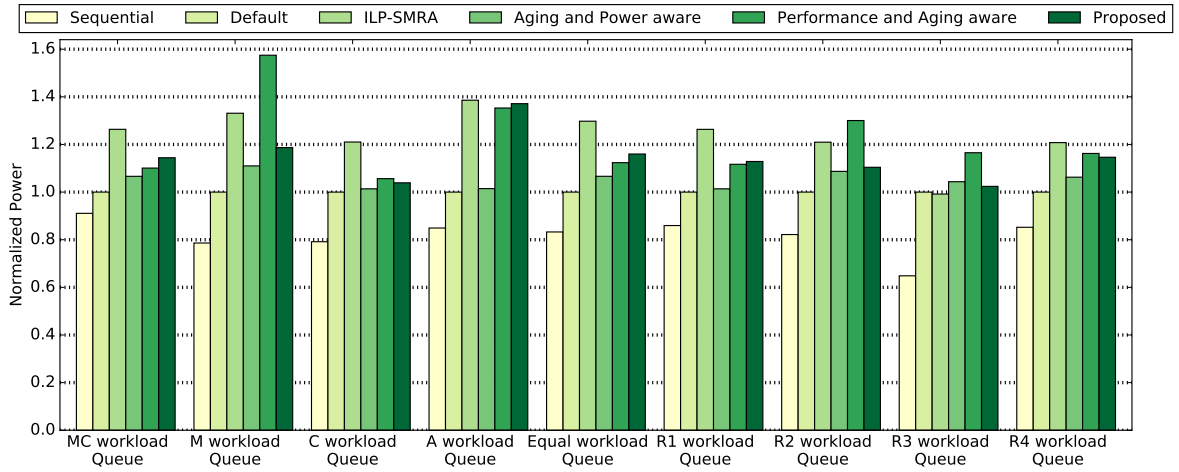
Figure 3a,b depicts the average power consumption per queue, for the six compared methodologies, for the Fermi and Pascal micro-architectures, respectively. To begin with, we notice that the Sequential approach consumes less average power that the other methodologies. This is expected by the fact that the Sequential approach achieves lower throughput. As a consequence, the same amount of work that the co-executing methodologies execute, is completed by the Sequential approach in a longer time period. That is equivalent to more idle cycles, thus, a lower power consumption on average. We observe that the proposed methodology consumes 11% higher power on average, compared to the default methodology for the Fermi micro-architecture. Although average power consumption might be higher than the default approach for certain workloads, we need to keep in mind that performance is always better for the proposed methodology, thus the increased power consumption is a trade-off. Compared to the ILP-SMRA methodology, the proposed methodology consumes on average 10% lower power per queue for the Pascal micro-architecture.

Continuing with our analysis, compared to the Aging and Power aware methodology, the proposed methodology demonstrates a 4.5% higher power consumption on average for the Fermi micro-architecture. As the former methodology is oriented towards aging reduction and lowering power consumption, it achieves better results in terms of power consumption, compared to the proposed methodology. The lower power consumption of the Aging and Power aware methodology can be explained by the 8% performance drop margin that is applied during the SM configuration selection. According to that margin, configurations that yield $Optimal\ IPC$ down to $0.92 \cdot Optimal\ IPC$ are explored before deciding how many SMs to assign to an application. As it will become clear by the next metric, power efficiency, the aforementioned SM allocation policy does not yield the optimal results when both performance and power consumption are important for a system. An additional comment that we can make about the Aging and Power aware methodology is that for certain queues it demonstrates higher average power consumption than the ILP-SMRA methodology. The latter is not a methodology that optimizes power consumption. This behavior can be explained by the fact that for specific benchmarks, methodologies can fail to accomplish their goals. If a queue consists of these outlier benchmarks, then the overall behavior of a methodology for the specific queue will not align with the goals of the methodology. Furthermore, the version of the Aging and Power aware methodology, that we used during the experiments, is a modified version of the original one [18]. As a consequence, for specific workload queues it may perform worse towards its goals, in this case lowering power consumption. Looking at the results for average power consumption for the Performance and Aging aware methodology, the proposed methodology achieves a 7% lower power consumption on average for the Pascal micro-architecture. This is anticipated as the former methodology does not take

power consumption as a parameter when it decides the SM configuration for concurrent applications. As a consequence, it performs worse that the proposed methodology in terms of power consumption.



(**a**) Fermi architecture

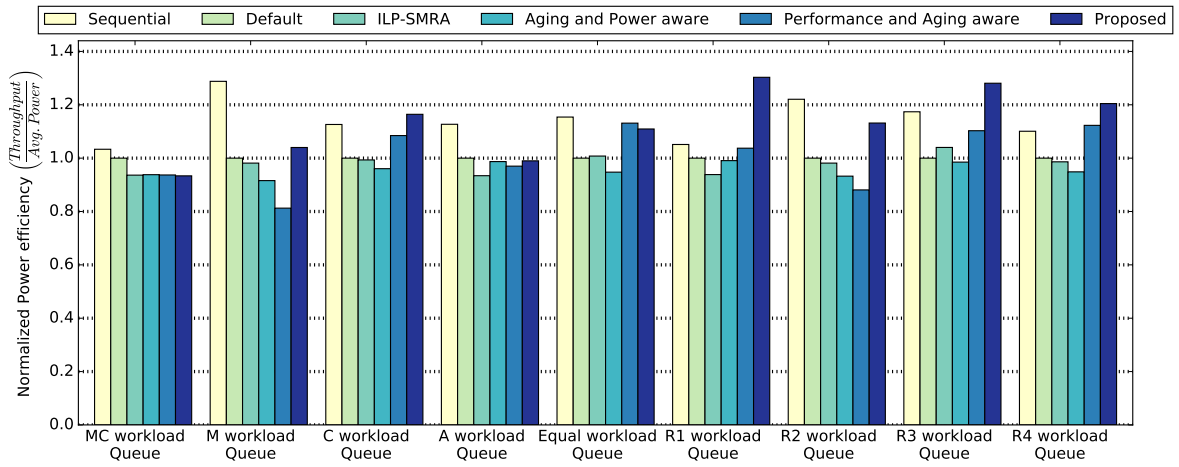


(**b**) Pascal architecture

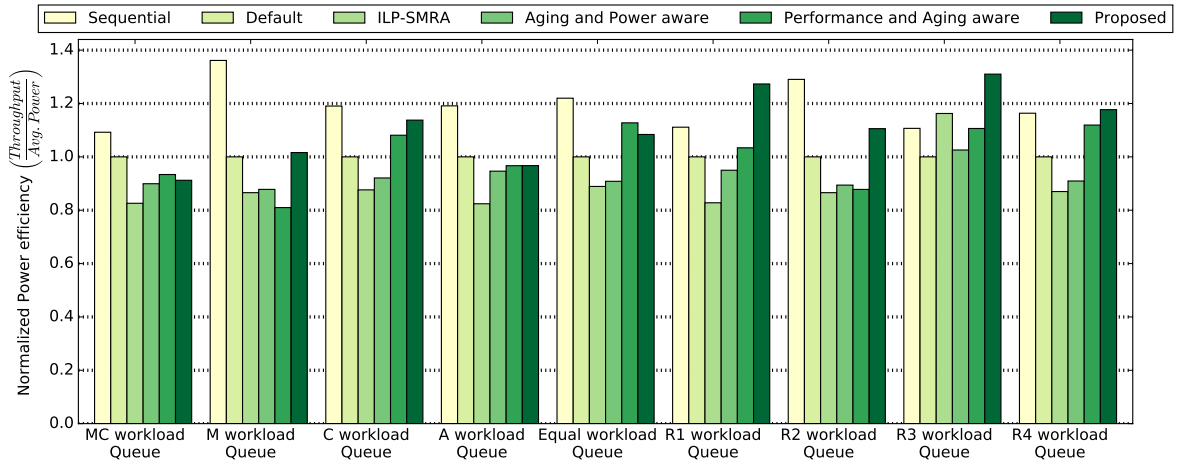
Figure 3. Normalized average power for nine queues, comparing the developed methodology with existing approaches.

In Figure 4 we present the power efficiency per queue for the six methodologies. Using this metric, the correlation between performance and power consumption becomes clear. The proposed methodology is a solution for systems where performance is important but power sources are restricted. It bridges the gap between solutions that aim only at improving performance, disregarding the power consequences that can appear, and on the other side of the spectrum, solutions that aim at lowering power consumption, while negatively affecting performance. Observing Figure 4 we notice that the proposed methodology achieves the best power efficiency for seven out of the nine workload queues, among the concurrent execution approaches for the Fermi micro-architecture. Additionally the proposed methodology demonstrates the best power efficiency for six out of the nine workload queues for the Pascal micro-architecture, considering only the concurrent execution approaches. The Sequential approach outperforms the rest of the approaches for five and six queues out of the nine queues, for the Fermi and Pascal micro-architectures respectively. Nevertheless, this is a consequence of the low average power consumption that Sequential execution achieves. Low power consumption though comes at the cost of low throughput thus, high power-efficiency for the Sequential approach is attached to the trade-off of low performance. Compared with the Default methodology, the proposed methodology demonstrates an 11% higher power efficiency on average for

the Pascal micro-architecture. Compared with the ILP-SMRA methodology, the proposed methodology achieves a 16% higher power efficiency on average for the Fermi micro-architecture. This behavior is a result of the high performance that the proposed methodology achieves, compared to the ILP-SMRA methodology. The high performance combined with the small power difference, on average 3% higher power consumption for the proposed methodology, leads to significantly better power efficiency for the proposed methodology.



(**a**) Fermi architecture

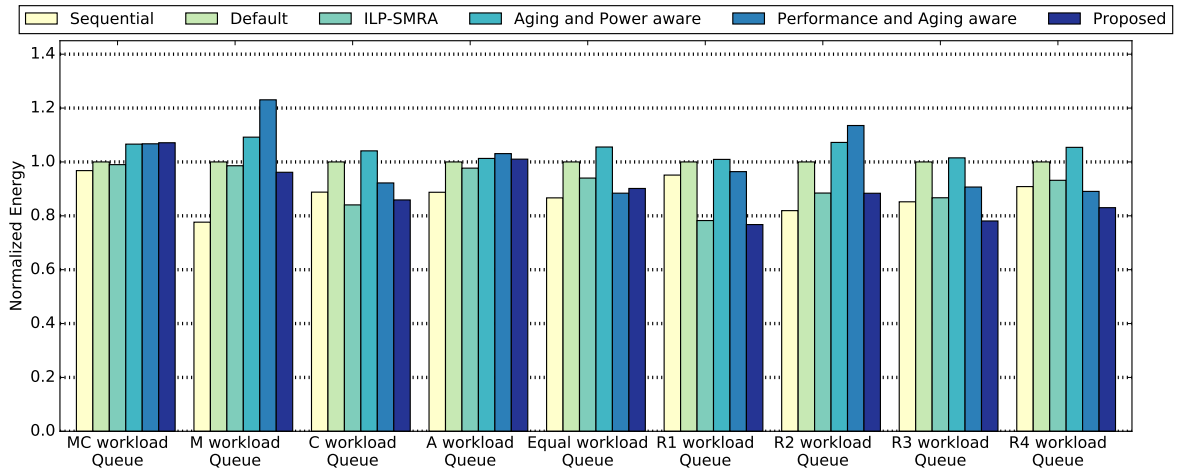


(**b**) Pascal architecture

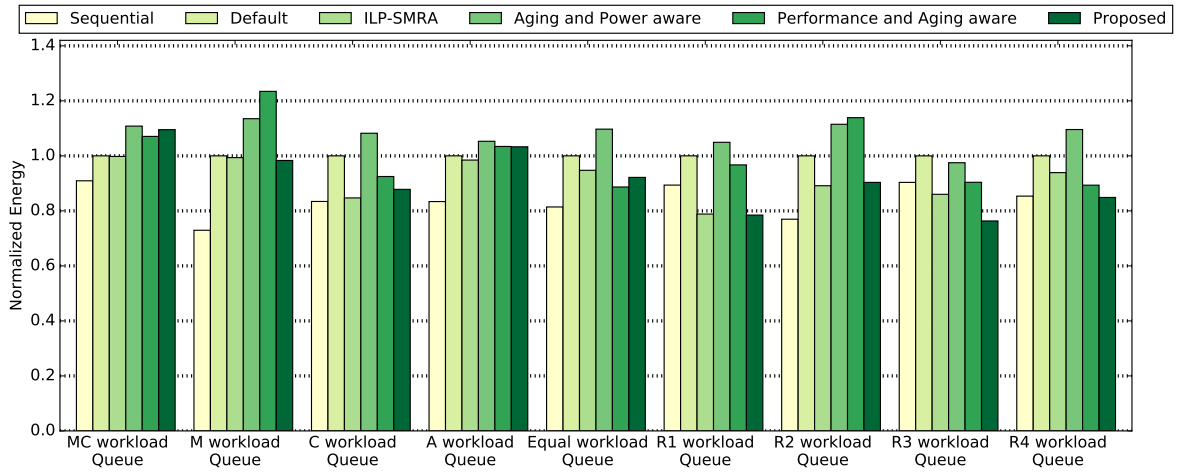
Figure 4. Normalized power efficiency for nine queues, comparing the developed methodology with existing approaches.

Comparing the proposed methodology with the Aging and Power aware methodology in terms of power efficiency, the proposed methodology achieves an 18% higher power efficiency on average for the Pascal micro-architecture. Even though Aging and Power aware methodology lowers, on average, power consumption, it does not maintain high performance. Thus, the power efficiency results are quite low, compared with the proposed methodology. Lastly, compared to the Performance and Aging aware methodology, the proposed methodology achieves a 12% higher power efficiency for the Fermi micro-architecture. The allocation decision during the proposed algorithm takes into consideration both performance and power characteristics for the applications. As a result, the proposed methodology achieves better power performance than the Aging and Power aware methodology. The improved results are also a consequence of the lack of an IPC reduction margin for the proposed methodology, compared to a 10% reduction margin for the Aging and Power aware methodology.

Finally, in Figure 5 we present the total energy consumption per queue for the six methodologies. The Sequential methodology achieves the lower energy consumption for the majority of the workloads. This is expected as this methodology demonstrates low average power as shown in Figure 3. Focusing on the methodologies for concurrent application execution, the Proposed methodology achieves the lowest energy consumption on average. Specifically, compared with the default methodology, the Proposed methodology demonstrates on average, a 9% lower energy consumption for the Pascal micro-architecture. Compared with the ILP-SMRA methodology, for the Pascal micro-architecture, it achieves an 8% lower energy consumption on average. If we compare the Proposed methodology with the Aging and Power aware methodology, we observe that the Proposed methodology achieves a 14% lower energy per queue, on average for the Fermi micro-architecture. Respectively, comparing the Proposed methodology with the Performance and Aging aware methodology, the Proposed methodology achieves an 11% lower energy on average for the Fermi micro-architecture.



(**a**) Fermi architecture



(**b**) Pascal architecture

Figure 5. Normalized total energy consumption for nine queues, comparing the developed methodology with existing approaches.

4. Discussion

In the previous section, Section 3, we compare the proposed methodology with five other relevant methodologies, and present the experimental results. The metrics we use for the evaluation are throughput, average power, power efficiency, and total energy. In this section we discuss in details how the proposed methodology compares to the other methodologies used during the experiments, and why we choose to focus on the power efficiency metric.

First, comparing the proposed methodology with the sequential approach, we observe that the proposed methodology achieves higher throughput for all the workloads. In terms of power, the sequential approach demonstrates lower average power consumption for all the workloads, for both GPU micro-architectures. The results for power efficiency are mixed, with the proposed methodology achieving higher power efficiency for four and three workload queues, for the Fermi and Pascal micro-architectures, respectively. Finally, in terms of total energy, the proposed methodology consumes less energy for four queues for the Fermi micro-architecture. These results can be explained by the fact that for the sequential approach, applications are not executed concurrently. As a result, performance is low due to resources remaining idle. On the other hand, low performance means idle components thus, average power is low and total energy for certain queues is also low.

Commenting on the comparison between the proposed methodology and the default methodology, in terms of performance, the proposed methodology achieves always higher throughput for both micro-architectures. In terms of power, the proposed methodology demonstrates higher average power consumption for all the workloads. Additionally, for the majority of the workloads for both micro-architectures, the proposed methodology achieves higher power efficiency as well. Similarly, the proposed methodology achieves lower total energy consumption for most of the workloads, compared to the default methodology. This behavior can be explained by the following facts. First, the proposed methodology utilizes the ILP technique to create application pairs. This is a first step that improves performance, compared to the FCFS matching of the applications during the default methodology. Furthermore, the proposed methodology uses the profiling information when allocating SMs for the co-executing applications. By utilizing this information, the proposed methodology achieves high performance, as more SMs are allocated to the application that will yield higher throughput. These two attributes of the proposed methodology lead to higher performance, compared to the default methodology. Consequently, high performance causes high average power consumption. Overall, even if average power is increased, the total energy and the power efficiency is better on average for the proposed methodology, compared to the default.

Comparing the proposed methodology with the ILP-SMRA methodology, in terms of throughput, we observe that the proposed methodology outperforms ILP-SMRA for all the workloads. Furthermore, the proposed methodology consumes higher average power for the majority of the workloads but demonstrates higher power efficiency for most of them, for the Fermi micro-architecture. On the other hand, for the Pascal micro-architecture, the proposed methodology demonstrates always lower average power consumption and higher power efficiency. In terms of energy, for the majority of the workloads, the proposed methodology consumes less total energy for both micro-architectures. The improved performance of the proposed methodology, higher throughput combined with higher power efficiency, can be explained by the allocation policy that it follows. The ILP-SMRA initially divides SMs equally between the co-executing applications and reallocates SMs during run-time, if it is needed. The initial equal division of SMs might cause valuable performance loss because the reallocation mechanism can take numerous cycles before it decides to transfer SMs from one application to the other. Additionally, the run-time reallocation causes loss of performance. This happens because in order to launch thread blocks from a different application to an SM, all thread blocks of the previous application on that specific SM should complete their execution first. This causes wasted cycles because new thread blocks should wait before they are scheduled to the newly allocated SMs.

Furthermore, comparing the proposed methodology with the aging and power aware methodology, we observe that the proposed methodology outperforms the later in terms of throughput

for all the workloads for the Fermi micro-architecture. Additionally, the proposed methodology outperforms the aging and power aware methodology for eight out of the nine workloads, for the Pascal micro-architecture. In terms of power, the aging and power aware methodology consumes less average power for the majority of the workloads but the proposed methodology achieves higher power efficiency, for both of the micro-architectures. Additionally, the proposed methodology demonstrates lower energy consumption for the majority of the queues. The proposed methodology performs better than the aging and power aware methodology because it leverages two techniques. One is ILP, by using this technique to form the pairs of applications, slow-down caused by conflicting demands for resources is minimized. The second technique is using the profiling information during the allocation of SMs. The aging and power aware methodology starts by dividing SMs equally between the applications. If an application achieves optimal performance with fewer SMs than the available to it, the SMs that are not used are clock-gated. On the other hand, the proposed methodology uses the profiling information in order to make the most efficient use of the SMs. No SMs will be clock-gated unless both applications achieve their optimal throughput with less than the total number of SMs, combined. In that way, the proposed methodology achieves higher throughput and higher power efficiency.

Finally, comparing the proposed methodology with the performance and aging aware approach, the former achieves higher throughput for the majority of the workloads for both micro-architectures. Additionally, the proposed methodology achieves lower average power for the majority of the workloads as well as higher power efficiency and less energy, for both Fermi and Pascal micro-architectures. The reason that the proposed methodology outperforms on average the performance and aging methodology is the fact that it favors the SM needs of the application with higher optimal power efficiency. That way, both overall power and performance are improved. Additionally, unlike the performance and aging methodology, the proposed approach does not apply a performance drop range for the application with the higher optimal power efficiency. In case the performance and aging methodology cannot satisfy the SM needs of the applications of a pair, it will reduce SMs for both applications up to 20% compared to the optimal IPC of each application.

The novelty of the proposed methodology is the combination of improved performance and power efficiency for co-executing applications on a GPU. We introduce the power efficiency metric in order to demonstrate that the developed methodology improves throughput but at the same time is power aware. Previous works on GPUs have focused on either improved performance or improved power consumption. To highlight our contribution and the applicability of the developed methodology, we use power efficiency as a central metric in our experiments. Similar metrics have been used in other works that develop power-efficient solutions, for example the IPC/W metric in [20].

5. Conclusions

In this article we present a power efficient methodology to allocate SMs for concurrent executing applications on a GPU. The developed methodology is based on collected information about the behavior of the application that will be executed. Based on this information, the methodology decides how many SMs should be allocated by each of the executing applications in order to achieve high performance, coupled with low power consumption. The proposed methodology can be useful for systems that use GPUs to accelerate software execution but present limitations due to their power sources. Compared to state of the art approaches, the proposed methodology can improve performance by up to 25% on average. At the same time, it can improve power efficiency by up to 16% on average, meaning that improved performance is not connected to excessive power consumption. The developed methodology is generic and the same approach can be applied to GPUs with less than 60 SMs. By adopting the proposed methodology, next generation devices will benefit with processing data in a more efficient way without risking excessive power consumption that can lead to reduced battery-life and heat dissipation issues.

Author Contributions: Conceptualization, Z.-G.T. and I.A.; data curation, Z.-G.T.; formal analysis, Z.-G.T. and I.A.; funding acquisition, I.A.; investigation, Z.-G.T.; methodology, Z.-G.T. and I.A.; project administration, I.A.; resources, I.A.; software, Z.-G.T.; supervision, I.A.; validation, Z.-G.T. and I.A.; visualization, Z.-G.T.; writing—original draft, Z.-G.T.; writing—review and editing, I.A.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPU	Central Processing Unit
FCFS	First Come First Served
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
ILP	Integer Linear Programming
IPC	Instructions Per Cycle
L_1	Level 1 cache memory
L_2	Level 2 cache memory
SIMD	Single Instruction Multiple Data
SM	Streaming Multiprocessor
SMRA	Streaming Multiprocessor ReAllocation
SP	Streaming Processor

References

1. Huang, L.; Yuan, F.; Xu, Q. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms. In Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, 20–24 April 2009; European Design and Automation Association: Leuven, Belgium, 2009; pp. 51–56.
2. Huang, L.; Xu, Q. Agesim: A simulation framework for evaluating the lifetime reliability of processor-based socs. In Proceedings of the Conference on Design, Automation and Test in Europe, Dresden, Germany, 8–12 March 2010; European Design and Automation Association: Leuven, Belgium, 2010; pp. 51–56.
3. Lee, M.; Song, S.; Moon, J.; Kim, J.; Seo, W.; Cho, Y.; Ryu, S. Improving GPGPU resource utilization through alternative thread block scheduling. In Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), Orlando, FL, USA, 15–19 February 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 260–271.
4. Tasoulas, Z.G.; Anagnostopoulos, I. Optimizing Performance of GPU Applications with SM Activity Divergence Minimization. In Proceedings of the 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Bordeaux, France, 9–12 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 621–624.
5. Tasoulas, Z.G.; Anagnostopoulos, I. Kernel-Based Resource Allocation for Improving GPU Throughput While Minimizing the Activity Divergence of SMs. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2019**. [CrossRef]
6. Adriaens, J.T.; Compton, K.; Kim, N.S.; Schulte, M.J. The case for GPGPU spatial multitasking. In Proceedings of the IEEE International Symposium on High-Performance Comp Architecture, New Orleans, LA, USA, 25–29 February 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–12.
7. Punyala, S.R.; Marinakis, T.; Komaee, A.; Anagnostopoulos, I. Throughput optimization and resource allocation on gpus under multi-application execution. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 73–78.
8. Tasoulas, Z.G.; Guss, R.; Anagnostopoulos, I. Performance-based and aging-aware resource allocation for concurrent gpu applications. In Proceedings of the 2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Chicago, IL, USA, 8–10 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
9. Tasoulas, Z.G.; Anagnostopoulos, I. Performance and Aging Aware Resource Allocation for Concurrent GPU Applications Under Process Variation. *IEEE Trans. Nanotechnol.* **2019**, *18*, 717–727. [CrossRef]

10. Oh, Y.; Yoon, M.K.; Song, W.J.; Ro, W.W. FineReg: Fine-Grained Register File Management for Augmenting GPU Throughput. In Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 20–24 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 364–376.
11. Kloosterman, J.; Beaumont, J.; Jamshidi, D.A.; Bailey, J.; Mudge, T.; Mahlke, S. Regless: Just-in-time operand staging for GPUs. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, Boston, MA, USA, 14–17 October 2017; ACM: New York, NY, USA, 2017; pp. 151–164.
12. Khorasani, F.; Esfeden, H.A.; Farmahini-Farahani, A.; Jayasena, N.; Sarkar, V. Regmutex: Inter-warp gpu register time-sharing. In Proceedings of the 45th Annual International Symposium on Computer Architecture, Los Angeles, CA, USA, 2–6 June 2018; IEEE Press: Piscataway, NJ, USA, 2018; pp. 816–828.
13. Mittal, S.; Vetter, J.S. A survey of methods for analyzing and improving GPU energy efficiency. *ACM Comput. Surv. (CSUR)* **2015**, *47*, 19. [CrossRef]
14. Betkaoui, B.; Thomas, D.B.; Luk, W. Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing. In Proceedings of the 2010 International Conference on Field-Programmable Technology, Beijing, China, 8–10 December 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 94–101.
15. Jiao, Y.; Lin, H.; Balaji, P.; Feng, W.c. Power and performance characterization of computational kernels on the gpu. In Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing, Hangzhou, China, 18–20 December 2010; IEEE Computer Society: Washington, DC, USA, 2010; pp. 221–228.
16. Anderson, D.; Dykes, J.; Riedel, E. More Than an Interface-SCSI vs. ATA. *FAST* **2003**, *2*, 3.
17. Wang, G.; Lin, Y.; Yi, W. Kernel fusion: An effective method for better power efficiency on multithreaded GPU. In Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing, Hangzhou, China, 18–20 December 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 344–350.
18. Chen, X.; Wang, Y.; Liang, Y.; Xie, Y.; Yang, H. Run-time technique for simultaneous aging and power optimization in GPGPUs. In Proceedings of the 51st Annual Design Automation Conference, San Francisco, CA, USA, 1–5 June 2014; ACM: New York, NY, USA, 2014; pp. 1–6.
19. Hong, S.; Kim, H. An integrated GPU power and performance model. In Proceedings of the ACM SIGARCH Computer Architecture News, Saint-Malo, Franc, 19–23 June 2010; ACM: New York, NY, USA, 2010; Volume 38, pp. 280–289.
20. Gilani, S.Z.; Kim, N.S.; Schulte, M.J. Power-efficient computing for compute-intensive GPGPU applications. In Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), Shenzhen, China, 23–27 February 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 330–341.
21. Wang, Z.; Yang, J.; Melhem, R.; Childers, B.; Zhang, Y.; Guo, M. Simultaneous multikernel: Fine-grained sharing of gpus. *IEEE Comput. Archit. Lett.* **2015**, *15*, 113–116. [CrossRef]
22. Che, S.; Boyer, M.; Meng, J.; Tarjan, D.; Sheaffer, J.W.; Lee, S.H.; Skadron, K. Rodinia: A benchmark suite for heterogeneous computing. In Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA, 4–6 October 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 44–54.
23. Bakhoda, A.; Yuan, G.L.; Fung, W.W.; Wong, H.; Aamodt, T.M. Analyzing CUDA workloads using a detailed GPU simulator. In Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, USA, 26–28 April 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 163–174.
24. Leng, J.; Hetherington, T.; ElTantawy, A.; Gilani, S.; Kim, N.S.; Aamodt, T.M.; Reddi, V.J. GPUWattch: enabling energy optimizations in GPGPUs. In Proceedings of the ACM SIGARCH Computer Architecture News, Tel-Aviv, Israel, 23–27 June 2013; ACM: New York, NY, USA, 2013; Volume 41, pp. 487–498.
25. Rogers, T.G.; O'Connor, M.; Aamodt, T.M. Cache-conscious wavefront scheduling. In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, Vancouver, BC, Canada, 1–5 December 2012; IEEE Computer Society: Washington, DC, USA, 2012; pp. 72–83.
26. Tröger, N.; Wieser, H.; Hübner, R. Smartphones are replaced more frequently than T-shirts. *Gerechtigkeit Muss Sein Vienna* **2017**; pp. 1–20.