Adaptive Power Shifting for Power-Constrained Heterogeneous Systems

Cristobal Ortega, Lluc Alvarez, Alper Buyuktosunoglu, Ramon Bertran, Todd Rosedahl, Pradip Bose, Miquel Moreto

Abstract—The number and heterogeneity of compute devices, even within a single compute node, has been steadily on the rise. Since all systems must operate under a power cap, the number of discrete devices that can run simultaneously at their highest frequency is limited by the globally-imposed power cap. Current systems incorporate a centralized power management unit that statically controls the distribution of power among the devices within the node. However, such static distribution policies are unaware of the dynamic utilization profile across the devices, which leads to power allocations that end up degrading system throughput performance. The problem is particularly acute in the presence of heterogeneity since type-specific performance-boost capabilities cannot be leveraged via utilization-agnostic static power allocations.

This paper proposes Adaptive Power Shifting for multi-accelerator heterogeneous systems (APS), a technique that leverages system utilization information to dynamically allocate and re-distribute power budgets across multiple discrete devices. Democratizing the power allocation based on dynamic needs results in dramatic speedup over a need-agnostic static allocation. We use APS in a real OpenPOWER compute node with 2 CPUs and 4 GPUs to demonstrate the value of on-demand, equitable power allocations. Overall, the proposed solution increases performance with respect to two state-of-the-art techniques by up to 14.9% and 13.8%.

Index Terms—Heterogeneous systems, Power-constrained systems, Accelerator system design, Power-efficient architectures.

1 Introduction

In recent years, data centers have started incorporating multiple diverse accelerators per node to provide efficient performance growth through specialization. However, complex heterogeneous systems with multiple discrete accelerators cannot afford to fully power all the devices simultaneously [1], [2]. To overcome this limitation, modern systems include mechanisms to adjust the power budget of the devices, allowing them to safely operate under a given power cap. The power cap of a system can be set for a variety of reasons within a data center, including safe operation modes when power delivery and thermal limits have been reached, and economical reasons.

Efficiently distributing power in heterogeneous systems with multiple discrete accelerators and varying power caps is a challenging problem. The growing amount of accelerators requires power distribution algorithms to be simple and scalable to ensure fast response times. Moreover, the latency of communicating control signals between discrete devices is usually very long, so minimizing device communation is a must. On top of this, the proliferation of different accelerators (GPUs, FPGAs, ASICs, etc.) manufactured by different vendors imposes power management solutions to use generic knobs present in all kind of devices, such as Dynamic Voltage Frequency Scaling (DVFS). The nature of heterogeneous workloads further complicates the problem, as their power demands can change due to program phases [3], accelerators being active or idle in different phases [4], and multiple programs running in different devices.

Previous works have studied how to efficiently distribute power under a limited power budget in different scenarios, as summarized in Table 1. Most of these works propose power distribution techniques for a single chip. The most common technique is to apply per-core DVFS to maximize the overall performance [5], [6], [7]. Also, some works combine DVFS with policies to control thread scheduling [8], core allocation [9], [10], power gating [11], and SMT levels [12]. Similarly, power and thermal management of heterogeneous chips with a single integrated GPU, such as Intel's Dynamic Power and Thermal Framework (DPTF) [13] or AMD's Bidirectional Application Power Management (BAPM) [14], use a centralized hardware controller inside the chip that considers power and thermal constraints. All these techniques are not directly applicable to systems with multiple discrete devices since single chips do not have information about the requeriments and the power budgets of the discrete devices. Furthermore, they use on-chip hardware controllers with adhoc interfaces, they rely on power knobs that are not present in all types of devices, they do not suffer from long communication latency between devices, and they consider on-chip power and thermal constraints that do not apply to systems with discrete devices. Other works distribute power between CPUs and a single GPU at application level, as summarized in Table 2. Many of these works intercept calls to the GPU kernels and shift power where the computation takes place [15], [16], [17], [18], [19], [20], [21]. However, this approach does not work in systems with multiple accelerators running multiprogrammed workloads. In such sceneraio, a centralized solution is needed to coordinate the

- *C. Ortega, L. Alvarez and M. Moreto are with the Barcelona Supercomputing Center and the Universitat Politecnica de Catalunya. E-mail: name.surname@bsc.es*
- *R. Bertran, A. Buyuktosunoglu and P. Bose are with the IBM T.J. Watson Research Center. T. Rosedahl is with IBM Systems. E-mail: {rbertra, alperb, rosedahl, pbose}@us.ibm.com*

TABLE 1: Previos work on distributing power among components in different scenarios.

Scope of work	Target workloads	Work
Core level	Mixed CPU workloads	[5], [8], [11], [26], [27]
Heterogeneous Chip	Single CPU-GPU workloads	[15], [16], [17], [18], [19], [20], [21]
Homogeneous System	Mixed CPU workloads	[6], [7], [9], [10], [12], [28]
Heterogeneous System	Multiple Mixed CPU workloads and CPU-GPU workloads	This work, [24], [25]

TABLE 2: Comparison with the state of the art

Related work	Works for single CPU-GPU workloads	Works for multiple CPU-GPU workloads	Power cap in Prediction Failure
[15], [16], [17], [18], [19], [20], [21]	✓	✗	✗
[24], [25]	✓	✓	✗
This work	✓	✓	✓

power consumption of multiple devices executing multiple workloads. For these reasons, high-performance systems with discrete accelerators do not incorporate any of the aforementioned solutions.

Industrial solutions to distribute power in heterogeneous systems with multiple discrete devices, such as IBM On Chip Controller (OCC) [22] or AMD System Management Unit (SMU) [23], use conservative heuristics to set individual power budgets for all the discrete devices present in the system. A centralized controller is responsible for managing the power consumption of the devices connected to it. When a power cap is enforced, the controller ensures that the system power consumption is below the power cap by dividing the power cap among the devices and informing them of their individual power budget. Then, individual devices are allowed to independently adjust their frequencies to honor the specified power budget while maximizing the performance or power efficiency of the device. This decoupled scheme loses opportunities for optimization, as the centralized controller sets the power budgets of the devices without considering their utilization, their contribution to the overall system performance, or their power headroom. Tangram [24] proposes a hierarchical organization of robust controllers that consider the throughput of the devices to distribute power among them. However, Tangram uses a complex algorithm with a long search phase and it requires a substantial amount of long latency communication between controllers, which compromises response time and scalability. Market solutions can also be used in heterogeneous systems [25], although no prior work has evaluated them in a setup with multiple discrete accelerators.

This paper presents *Adaptive Power Shifting for heterogeneous systems (APS)*, a technique that maximizes the performance of power-constrained heterogeneous systems by leveraging system information to distribute power among all the devices. APS advocates for a hardware/software power distribution mechanism that carefully balances efficiency and fairness by leveraging dynamic load information. In contrast with previous works, APS is agnostic of the devices (CPU, GPU, or other accelerators), it uses a simple and scalable heuristic that requires minimal communication between devices, it works for single applications and multiprogrammed workloads, and it can be implemented in current systems without any hardware modification, as we demonstrate with the deployment on a real OpenPOWER system with 2 CPUs and 4 GPUs. APS uses device utilization as the metric for optimization, which is very well suited for heterogeneous systems with devices with different power and performance characteristics, and it is also a very accurate proxy for power consumption. At execution time, APS monitors the power consumption and the utilization of all the devices in the system and dynamically shifts the power between them. To maximize performance, APS enables devices that are more utilized to have a higher power budget than devices that are less utilized. APS also captures changes in the power cap and quickly re-assigns the power budgets of the devices to minimize the time the system is over the specified power cap and to reduce the time a device is assigned an underperforming power budget. This paper makes the following contributions:

- We demonstrate that current solutions in modern systems can lead to over or under provisioning the power budget of individual devices in a power-constrained heterogeneous system. Current solutions cannot respond to the changing power requirements of heterogeneous workloads that use multiple devices in different program phases. Thus, a dynamic, efficient, and scalable power distribution technique is needed to maximize performance under various different power demands within a node composed of multiple discrete heterogeneous devices.
- We propose APS, a technique that distributes the power among the devices of a power-constrained heterogeneous system based on the utilization of each device. APS includes mechanisms to maximize the performance of the system while honoring a power cap and quickly and efficiently react to changes in the power cap.
- We implement and evaluate APS to manage the power distribution on an OpenPOWER system with 2 CPUs and 4 GPUs. When running under different power caps, APS improves performance over static power distributions up to 15.9% for single CPU-GPU applications and up to 20.5% for multiprogrammed workloads. APS also improves performance up to 14.9% over Tangram [24] and up to 13.8% over a state-of-the-art market solution [25].

This paper is organized as follows: Section 2 provides the required background and discusses the related work, Section 3 motivates this work, Section 4 presents APS, Section 5 describes the experimental setup, Section 6 evaluates APS, and Section 7 remarks the conclusions of this work.

2 Background and Related Work

2.1 Power Capping

Dynamic Voltage and Frequency Scaling (DVFS) provides a mechanism to adjust the voltage and the frequency of a chip. DVFS exposes different combinations of voltage and frequency in which the hardware can safely operate. DVFS is included in virtually all system making it the most used hardware knob to control the power consumption of a device. In this work, we use DVFS to manage the power consumption of the devices to make APS generic and directly applicable to any system, regardless of the type of accelerators it includes.

To ensure heterogeneous systems do not exceed their power cap, designers estimate the peak power consumption of the individual devices and add them up to get the peak power consumption of the system [26], [29]. However, modern computing systems are often restricted to consume less power than their peak for a variety of reasons, including safe

TABLE 3: Static power distributions for multiple devices in a system with 2 CPUs and 4 GPUs.

Power Cap (W)	Fairness		CPU Prio.		GPU Prio.	
	CPUs	GPUs	CPUs	GPUs	CPUs	GPUs
1000	166×2	166×4	200×2	150×4	75×2	212×4
900	150×2	150×4	200×2	125×4	75×2	187×4
800	133×2	133×4	200×2	100×4	75×2	162×4
700	116×2	116×4	200×2	75×4	75×2	137×4
600	100×2	100×4	200×2	50×4	75×2	112×4

operation modes when power delivery and thermal limits are reached, and cost reasons in servers and data centers. When these situations arise, a power cap is introduced in the system and the available power is distributed among the devices. In addition, globally assigning a power cap in the system can also be useful to address point-wise heat levels, specially if combined with capable cooling systems and safe operation modes.

Current industrial systems with multiple devices use *static* power distribution schemes, i.e., for a given power cap, they set a fixed power budget for every device regardless of the characteristics of the workloads running on the devices. For example, the IBM OCC controller of the OpenPOWER architecture provides a mechanism to adjust the power budgets of a group of CPUs and the GPUs when a power cap is introduced. As shown in previous research [22], this mechanism can be used to implement three static power distributions: (i) Fairness equally distributes the system power cap among all the devices in the system; (ii) CPU Priority (CPUprio) distributes the system power cap between the CPUs and the GPUs with different weights in favor of the CPUs; and (iii) GPU Priority (GPUprio) distributes the system power cap between the CPUs and the GPUs with different weights in favor of the GPUs.

Table 3 shows the power budgets of the devices for each policy in a heterogeneous system with 2 CPUs and 4 GPUs and power caps of 1000, 900, 800, 700, and 600W. The static policies shown in Table 3 set the power budget of the devices to fixed values, without considering their utilization, which can lead to underperforming power distributions. For instance, if a workload is CPU intensive and the power distribution is not CPUprio, CPUs run at a low frequency, leading to an overall slowdown. Moreover, if an application has CPU and GPU phases, static power distributions are unable to boost specific accelerators at the appropriate time.

Once the controllers have assigned power budgets to all the devices, these can adjust their running frequency inside a range of available frequencies that honor their power budget. The most used CPU governor in Linux is the *ondemand* governor, which periodically calculates the CPU utilization (non-idle cycles) and sets a corresponding frequency [30]. Notice that *ondemand* governor does not have power information, failing to deliver the best frequency when a power cap is imposed, and it does not consider possible interactions with other devices that need a higher power budget. Similarly, GPUs have a *Maximum Efficiency mode* [31] that sets maximum frequency when a workload is running and it lowers the frequency otherwise. This decoupled scheme loses opportunities for optimization, as the power distribution among the devices does not consider their utilization, limiting the range of available frequencies for the devices to maximize performance or efficiency.

2.2 Related Work

To overcome the limitations of static distributions, there has been considerable work in the literature that proposes dynamic solutions for different types of systems and scenarios.

2.2.1 Homogeneous Chips

Previous research on power distribution targets homogeneous multicore processors. Isci et al. [5] propose to monitor the performance of the cores and apply DVFS to maximize the overall performance under a power cap. Winter et al. [8] present different scheduling and power management algorithms on a 256-core architecture. Adileh et al. [32], [33] schedule applications into out-of-order or in-order cores based on their performance and power consumption. Some works propose to coordinate the frequencies of the CPU and the memory to maximize performance [26], [34], [35], coordinate DVFS with other techniques such as power gating [11], core allocation [9], [10], and SMT levels [12], save energy [27] under a power cap, and manage power through a resource controller based on market solutions [6] and machine learning models [7].

APS targets a system with multiple heterogeneous discrete devices and changing power caps, which makes the problem more challenging due to the complexity of such systems and the characteristics of the workloads that seldomly use the accelerators. APS can be coupled with the aforementioned techniques to optimize the performance within the specified power budget for the CPUs.

2.2.2 Heterogeneous Systems with a Single GPU

Some research work has studied how to distribute power in heterogeneous systems with only one accelerator. These solutions are tailored to a single CPU-GPU application, and their algorithms distribute power only when compute kernels start and finish their execution on the accelerator.

Majumdar et al. [17] predicts the best hardware configuration for GPU computation kernels at runtime to improve energy-efficiency by tracking recent execution history. Jiao et al. [18] coordinate the execution of concurrent kernels and DVFS to improve the energy efficiency of a system with a single GPU. Bailey et al. [15] coordinate the DVFS of a power capped system with a single GPU executing a single CPU-GPU application. Similarly, Komoda et al. [16] coordinate DVFS and task mapping in a system with a single GPU.

The main drawback of these solutions is that they do not apply to multiprogrammed workloads, since taking decisions at system level based on the combination of profile information of single applications leads to underperforming configurations. To overcome this limitation, APS uses sampling to continuously monitor the requirements of the devices, independently of the application, so it is not limited to monitoring activity and power distribution at the granularity of compute kernels. This allows APS to immediately react to changes in the power cap and to quickly set an optimal power distribution. None of the aforementioned solutions considers reacting to changes in the power cap.

Sampling-based techniques are also used by Paul et al. to distribute the power in a heterogeneous chip with an integrated GPU to improve energy efficiency [20] and performance [19]. However, their proposal only configures the

power states of the CPUs among 4 possible preset values, and they rely on the controller to indirectly re-adjust the frequency of the GPU based on the power state of the CPU and the thermal coupling of both devices. This previous work is not applicable to discrete GPUs because they do not have thermal nor performance coupling effects with the CPU, as they are in different chips.

2.2.3 Heterogeneous Systems with Multiple GPUs

Tangram [24] proposes to distribute power on a heterogeneous system with multiple discrete devices by means of a hierarchical organization of robust controllers with a common scalable interface. The authors demonstrate Tangram in a system with 2 CPUs and 1 GPU running a single CPU-GPU application. This approach requires a significant amount of long latency communication between the devices and, as shown in Section 6.3, it uses a Nelder-Mead search algorithm that suffers from long search periods and tends to determine unbalanced power distributions in some multiprogrammed workloads. APS works for mixed CPU-GPU applications, it uses a fast algorithm that scales to multiple devices, and can be easily implemented in existing large-scale systems without hardware modifications.

Market solutions have also been proposed to distribute power in heterogeneous systems [25], although they have never been deeply studied in a scenario with multiple discrete accelerators. Like APS, market solutions are algorithms that solve dynamic optimization problems, and they apply allocations for forward-facing needs based on rear-facing data. One of the key aspects of market solutions is the bidding mechanism. Market solutions that have true bids and support a Walrasian Equilibrium are provably efficient but, when the bids are based on heuristics, market solutions can be less effective. Unfortunately, bidding mechanisms for distributing power in heterogeneous systems with multiple accelerators have never been studied in the literature. In Section 6.3 we show that APS outperforms a market solution that uses heuristic bids based on the TDP of the devices.

3 Motivation

To highlight the challenges of distributing power consumption under a power cap, we run a CPU-bound workload and a GPU-bound workload on an OpenPOWER system with 2 CPUs and 4 GPUs. The CPU-bound workload is composed of a DAXPY kernel in the CPUs and a Speckle Reducing Anisotropic Diffusion (SRAD) kernel [36] in the GPUs. The GPU-bound workload is Tensorflow [37] training an Inception v3 neural network, which uses the GPUs for the main computation and the CPUs for the data transfers, updating the weights, and offloading work to the GPUs.

Figure 1 compares the default power distribution of the OpenPOWER architecture (Default[1]), the three static power distribution techniques explained in Section 2.1 (Fairness, CPUprio, and GPUprio), and APS in a system with a power cap of 800W and different numbers of GPUs used. The y-axis shows the weighted speedup with respect to the Fairness technique and same number of GPUs, and the x-axis shows the average power consumption of the main computation phase normalized to the system power cap of 800W.

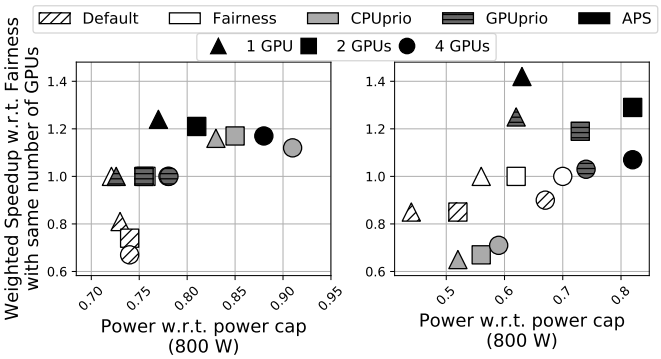


Fig. 1: Execution of DAXPY in the CPUs and SRAD in the GPUs (left), and Tensorflow training an Inception v3 neural network (right).

The left plot in Figure 1 shows the results for the CPU-bound workload running a DAXPY kernel in the CPUs and a SRAD kernel in the GPUs. The default power distribution degrades the performance with respect to Fairness because it drastically reduces the frequency of the CPUs. CPUprio significantly improves the weighted speedup with respect to Fairness by 16.5%, 16.7%, and 12.0% for 1, 2, and 4 GPUs, respectively. Reducing the power assigned to the GPUs does not degrade the performance of SRAD, while the extra power in the CPUs boosts the performance of DAXPY.

Finally, APS improves the weighted speedup with respect to CPUprio by 6.3%, 4.8%, and, 3.8% for 1, 2, and 4 GPUs, respectively. APS shifts power to the GPUs when the CPUs enter an idle state, boosting GPU applications and improving the overall performance. Also, APS reduces the average power consumption with respect to CPUprio (between 2.8% and 5.1%) because it lowers the frequency of the GPUs when they are in a low utilization phase.

The right chart in Figure 1 shows the results for the GPU-bound workload running Tensorflow. CPUprio degrades performance up to 34.6% with respect to Fairness because the computation in the GPUs is drastically slowed down, affecting the overall performance even if the data transfers to the GPUs and the weight updates in the CPUs are done at the maximum possible performance. The default power distribution lowers the frequency of the GPUs more than Fairness but not as much as CPUprio, resulting in a slowdown of 15.0%. In contrast, GPUprio achieves speedups of up to 24.7% (with 1 GPU) by accelerating the computation phase in the GPUs.

APS achieves the best performance for this workload by dynamically shifting power between the CPUs and the GPUs. In the phases that perform data transfers and weights update, APS shifts power to the CPUs while, in the compute phases, the power is shifted to the GPUs. As a result, APS improves the weighted speedup with respect to GPUprio by 17.0%, 10.2%, and a 3.9% for 1, 2, and 4 GPUs, respectively.

In conclusion, APS outperforms static techniques due to a better utilization of the available power. Static distributions cannot respond to the changing power requirements of heterogeneous applications that use different devices

[1]There is no public information on the default behavior of the OpenPOWER architecture. Based on the experimental results, the default behavior sets very conservative power budgets to all the devices.

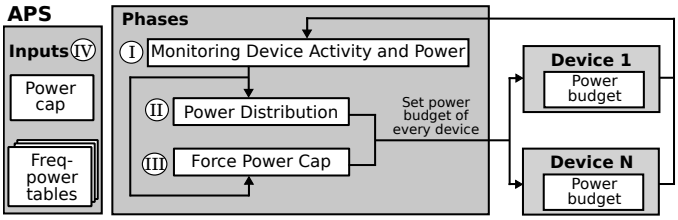


Fig. 2: Overview of a system using APS.

in program phases. Thus, an intelligent dynamic power distribution specially targeted to heterogeneous systems with multiple discrete accelerators is needed to maximize the system power-performance efficiency. To this end, APS provides a hardware/software power distribution mechanism [12], [38], [39] that carefully balances efficiency and fairness by leveraging dynamic load information, increasing the fidelity of the power distributions [40]. Thanks to these properties, APS outperforms other dynamic techniques such as Tangram [24] and market-based allocation [25]

4 Adaptive Power Shifting for Multi-Accelerator Heterogeneous Systems

APS maximizes the performance of power-constrained heterogeneous systems by shifting power among devices based on their utilization. Devices with high power budgets and low utilization cannot fully use their power budget, thus APS shifts power to devices with a higher utilization.

4.1 Overview

Figure 2 shows a system using APS. APS is independent of the underlying devices and of the type of workload, which can be composed of any number of applications using any combination of devices during their execution.

APS constantly monitors the power consumption and utilization of all the devices in the *Monitoring Device Activity and Power* phase (Ⓘ in Figure 2). These readings are done through the standard interfaces of each device. As explained in Section 5, in our setup we use perf for the CPU utilization, in-band OCC readings for the CPU power consumption, and NVML for the utilization and the power consumption of the GPUs. After gathering multiple samples of the power consumption and utilization, the algorithm triggers a *Power Distribution* phase to distribute the available power among the devices based on their utilization (Ⓘ in Figure 2). If the total power consumption surpasses the power cap, APS goes to a *Force Power Cap* phase to meet the power cap as soon as possible (Ⓘ in Figure 2). The power cap is an input to APS set by the administrator or higher-level power manager (e.g. rack or cluster level). If no power cap is specified, it is set to the node Thermal Design Power (TDP).

The frequency-power (*freq_power*) tables are key elements of the design of APS (Ⓘ in Figure 2). APS uses a pre-generated freq_power table for every device type in the system. The freq_power table of each device contains the maximum observed power consumption for all the frequencies, and is generated offline by running a stressmark. In our system we use as stressmarks a DAXPY for the CPUs and a DGEMM for the GPUs. With the freq_power tables APS can quickly set a frequency level that honors the power budget assigned to a device and avoid iterative searches for a budget compliant frequency. These freq_power tables are stored in software, but they could be implemented in hardware since they are small (one entry per DVFS level).

```
1 avgSamples = N
2 numSamples = 0
3 while True do:
4   TotalUtil = 0.0
5   TotalPower = 0.0
6   for all devices in the system:
7     freq[device]  ← read device frequency
8     util[device]  ← read device utilization
9     power[device] ← read device power
10    TotalUtil  += util[device]
11    TotalPower += power[device]
12  ++numSamples
13  if numSamples == (avgSamples-1):
14    PowerDistribution(freq,util,power,TotalUtil)
15    numSamples = 0
16  else if TotalPower > PowerCap:
17    ForcePowerCap(freq,power,TotalPower);
```

Listing 1: Algorithm of the monitoring phase.

Note that the design of APS is open to using other hardware knobs other than DVFS to adjust the power budgets of the devices. The only requirement to use other hardware knobs is to create a proxy that correlates the configuration values of a knob with the power consumption of the device under that configuration, similarly to the freq_power tables. To combine multiple knobs [41], APS would require a table that contains every combination of the values of the knobs and their power, as well as a mechanism that decides the most convenient configuration when the same power can be achieved with different combinations of the knobs. This scenario with multiple knobs is out of the scope of this work.

APS does not require any modifications in the source code of the applications and it does not instrument the offloading of the GPU kernels nor any application-level interaction between the CPUs and GPUs. Instead, the interaction between the devices is modeled after their utilization and power consumption, which is measured at system level.

4.2 APS Phases

APS relies on 3 phases, as shown in Figure 2: Ⓘ monitor the system, Ⓘ distribute the available power, and Ⓘ force the power cap when the total system power exceeds it.

4.2.1 Monitoring Device Activity and Power

The monitoring phase is responsible of monitoring the system status and triggering the Power Distribution and Force Power Cap phases. The system status includes power consumption and utilization of all the devices, which is computed as the percentage of cycles that a device has been executing a process during the sampling period. The sampling period in our experimental setup is 200ms, as discussed in Section 5.

The monitoring phase iterates indefinitely over the algorithm shown in Listing 1. After the initialization, the algorithm gathers the current frequency, the utilization and the power consumption of all the devices (lines 7 to 9 in Listing 1). The frequencies, the utilization and the power consumption of the devices are stored in three vectors, `freq`, `util` and `power`, which contain one element per device. In addition, the accumulated utility and power consumption of all the devices is calculated (lines 10 and

11) on `TotalUtil` and `CurrentPower`, respectively. When a number of samples has been taken (5 in our setup), the monitoring algorithm triggers the Power Distribution phase (lines 13 to 15). When the total system power exceeds the power cap, the monitoring algorithm triggers the Force Power Cap phase (lines 16 and 17) to reduce the power budget of every device so that the power cap is respected.

4.2.2 Power Distribution

The Power Distribution phase is responsible of setting the power budgets of all the devices in the system according to their utilization. The algorithm that implements this phase is shown in Listing 2. The input to the algorithm is the data gathered by the monitoring algorithm and the total system power cap (*PowerCap*).

The Power Distribution algorithm starts by computing the relative utilization *RelativeUtil* of every device with respect to the total utilization of the system (lines 2 and 3). To do so, it divides the utilization of each device by the accumulated utilizations of all the devices *TotalUtil*. For instance, if a system with 6 devices has a *TotalUtil* of 400% and one of the devices has an utilization of 80%, its relative utilization is $0.8/4.0 = 0.2$. Then, the algorithm enters a loop that iterates until there is no power headroom. To control this condition, the *PowerHeadroom* is the difference between the total system power cap and the sum of the power consumption of all the devices in the system, and the *minPowerStep* is the minimum possible increase in power consumption caused by the minimum possible increase of the frequency of any of the devices of the system. The *minPowerStep* avoids re-triggering the outer loop constantly. In our setup *minPowerStep* is 5 Watts, corresponding to the minimum possible increase of 70 Hz in the CPU frequency. Thus, when the *PowerHeadroom* is lower than 5 Watts, it is impossible to assign the *PowerHeadroom* to any device and the Power Distribution phase finishes.

The outermost loop starts by setting the *CurrentPower* to zero (line 5), which is a variable that is going to accumulate the power consumed by all the devices running at a certain frequency. Then, the algorithm checks if all the devices are running at their maximum frequencies (lines 6 and 7). If all the devices are already running at their maximum frequencies, the Power Distribution phase finishes (line 8). Otherwise, the innermost loop iterates over all the devices (line 9). For each device, its *TargetPower* (line 10) is calculated as $RelativeUtil[device] \times PowerCap$. The *TargetPower* of a device represents the maximum power the device can consume, based on the relative utility of the device in the system and the power cap. For instance, for a power cap of 1000W, a device with a relative utilization of 20% will have a *TargetPower* of $0.2 \times 1000W = 200W$. Note that, if two different devices have the same relative utilization, they will get the same portion of the power cap as *TargetPower*. Then the *StressmarkPower* of the device is obtained by consulting the freq_power table (line 11). The *StressmarkPower* represents the power consumption of the device running the stressmark at the current frequency. Then the algorithm calculates the *RatioDevice* of the device (line 12) as $StressmarkPower/power[device]$. The *RatioDevice* represents how much power an application consumes running at a certain frequency compared to the stressmark running at the same frequency. This is done because the freq_power tables store frequency and power values generated with stressmarks that are CPU and GPU intensive, but most workloads have different frequency-power characteristics than the stressmark. For instance, if an IO-intensive application has a power consumption of 75W running at a certain frequency and the stressmark consumes 150W running at the same frequency, the *RatioDevice* is $150W/75W = 2$. The goal of the *RatioDevice* is to scale up the *TargetPower* of the device before looking for its target frequency in the freq_power table, so the optimal frequency for that device is found in a single step, honoring the power budget and minimizing the power headroom without the need to iteratively refine the power budget. To do so, the algorithm calculates the *PowerBudget* as $TargetPower \times RatioDevice$ (line 13). This *PowerBudget* is searched in the freq_power table for that device type (line 14), and APS selects the highest frequency that satisfies the *PowerBudget*. Finally, the *TargetPower* of the device is accumulated in the *CurrentPower* (line 15). Once the innermost loop has calculated the frequencies of all the devices, the *PowerHeadroom* is calculated as $PowerCap - CurrentPower$ (line 16).

```
1 Require: freq[], util[], power[], TotalUtil, PowerCap
2 for all devices in the system:
3   RelativeUtil[device] = util[device] / TotalUtil
4 while PowerHeadroom > 0 and PowerHeadroom > minPowerStep:
5   CurrentPower = 0
6   MaxFreqDevices = devices with freq == maxFreq
7   N = Number of devices in MaxFreqDevices
8   if N == Number of devices in the system: break
9   for all devices in the system:
10    TargetPower = RelativeUtil[device] × PowerCap
11    StressmarkPower = freq_power_table[freq[device]]
12    RatioDevice = stressmarkPower / power[device]
13    PowerBudget = TargetPower × RatioDevice
14    freq[device] = freq_power_table[PowerBudget]
15    CurrentPower = CurrentPower + TargetPower
16  PowerHeadroom = PowerCap − CurrentPower
17 apply_frequencies()
```

Listing 2: Algorithm of the Power Distribution phase.

At the end of the algorithm, when all the frequencies of all the devices have been calculated and there is no power headroom available, the new frequencies are applied to all the devices via the DVFS controller (line 17). The new frequency of a device limits its power consumption to its calculated power budget until the monitoring phase triggers again the Power Distribution phase.

Note that this algorithm could be implemented as a reinforcement learning problem where agents or devices take actions by increasing or reducing their running frequencies, and the reward is based on how much power headroom is left or what is their individual throughput. The viability of this approach depends mostly on the trade-offs between the quality of the decisions and the complexity, which can lead to large overheads compared to simpler heuristics. We leave the study of such an approach for future work.

4.2.3 Force Power Cap

This phase forces the power cap when the total power consumption exceeds it. The goal of this phase is to reduce the power consumption as soon as possible to minimize the time over the specified power cap. For this reason, the Force Power Cap phase quickly reduces the power consumption equally from all the devices, and APS relies on the next

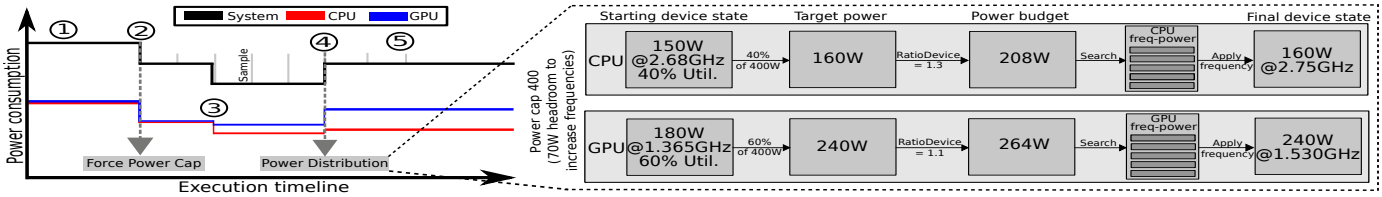


Fig. 3: Synthetic example of APS distributing power in a system with 1 CPU and 1 GPU.

```
1 Require: freq[], power[], CurrentPower, PowerCap
2 while CurrentPower > PowerCap:
3   NonMinFreqDevices = devices with freq > minFreq
4   N = Number of devices in NonMinFreqDevices
5   if N == 0: break
6   PowerToReduce = (CurrentPower - PowerCap)
7   DevicePowerToReduce = (PowerToReduce / N)
8   CurrentPower = 0
9   for device in NonMinFreqDevices:
10    PowerBudget = power[device] - DevicePowerToReduce
11    freq[device] = freq_power_table[PowerBudget]
12    CurrentPower += PowerBudget
13  MinFreqDevices = devices with freq == minFreq
14  CurrentPower += sumPower(MinFreqDevices)
15 apply_frequencies()
```

Listing 3: Algorithm of the Force Power Cap phase.

Power Distribution phase to optimize the power budgets of the devices under the new power cap. The algorithm that implements the Force Power Cap phase is shown in Listing 3. Note that this phase can be triggered at any moment of the execution, as the power cap can be exceeded at any time due to applications entering a phase with higher power consumption, responses to thermal or energetic emergencies, etc. Therefore, distributing the power in this phase can lead to measurements not representative of the workloads running in the devices (the Power Distribution phase measures 5 samples before distributing the power).

The Force Power Cap algorithm reduces the power budget of the devices that are not running at their minimum frequency until the sum of the power of all devices is less than or equal to the power cap. First, the algorithm selects the devices that are not running at their minimum frequency (lines 3 to 4 in Listing 3) and, if all of the devices are running at their minimum frequencies, the algorithm finishes (line 5). Otherwise, APS calculates the amount of power that needs to be reduced, *PowerToReduce*, as $CurrentPower - PowerCap$ (line 6). The *PowerToReduce* is equally divided among these devices by calculating the *DevicePowerToReduce* in (line 7). Then, the *CurrentPower* is set to zero (line 8), and the algorithm enters a loop that reduces the power of all the devices that are not running at their minimum frequency (line 9).

For each device that is not running at its minimum frequency, its new power budget (*PowerBudget* in line 10) is computed by subtracting the *DevicePowerToReduce* from the current power consumption of the device. Then, the device is assigned a new frequency by looking up its new power budget in the freq_power table of that device type and selecting the highest frequency that satisfies it (line 11), and the *PowerBudget* of the device is accumulated on the *CurrentPower* (line 12), which controls the total power of the system. After the innermost loop, the power consumption of the devices that are running at their minimum frequency is also accumulated on *CurrentPower* (lines 13 and 14).

Finally, the frequencies assigned to all the devices are applied (line 15). The new frequencies of all devices ensure that the total power consumption honors the power cap or that all the devices are running at their lowest frequency. If the power behavior changes and exceeds the power cap, the monitoring phase is in charge of triggering this phase again.

APS can use this algorithm thanks to the information of the freq_power tables. Previous work [24] need to lower the frequencies of all devices to their minimum possible frequency level since their mechanism is not aware of the power consumption of every frequency level.

4.3 Algorithm Walkthrough for a CPU+GPU

Figure 3 shows an example of APS managing a system with 1 CPU and 1 GPU (red and blue lines, respectively). The total power consumption is represented by the sum of both devices (black line). Initially (①), each device is running a workload and no power cap is specified. At ②, a power cap of 400W is introduced. APS invokes the Force Power Cap phase to reduce the power consumption of each device. As a result, the CPU and GPU are set to a lower frequency and their power consumption decreases.

The applications running in the CPU and the GPU enter a phase with a lower power consumption at ③. At ④ APS invokes the Power Distribution phase to set the power budget of the CPU and GPU based on their utilization from the last 5 samples. Since the total system power consumption is lower than the power cap, APS has power headroom to increase the frequencies of the CPU and GPU. The right part of Figure 3 shows how this process is done with an example.

At ④, the total system power consumption is 330W, and the CPU and GPU consume 150W and 180W with an utilization of 40% and 60%, respectively. Thus, APS has a power headroom of 70W to boost the performance. The first step is to calculate the *Target power*, that represents the maximum power budget for a device within a power cap. In this example, the target power for the CPU and GPU is 160W and 240W, respectively.

Next, APS calculates the *power budget* of every device. To do so, APS relies on a freq_power table that is generated with a stressmark for that device. Therefore, if an application has a lower power consumption than the stressmark at a given frequency and APS applies this same frequency, the device will not fully use its power budget. In the example, the power consumption of the CPU stressmark running at 2.68GHz is 200W, and the GPU running the stressmark at 1365MHz consumes 200W. Consequently, the RatioDevice for the CPU is $200/150 = 1.3$, and for the GPU the RatioDevice is $200/180 = 1.1$.

In the next step APS looks for the highest frequency that honors the *power budget* in the freq_power table of each device. The power values searched in the freq_power tables of the CPU and the GPU are 208W and 264W, respectively.

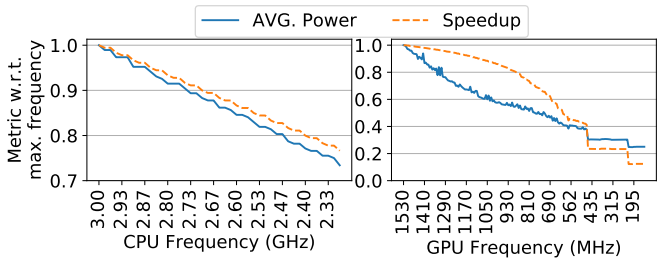


Fig. 4: Average power consumption and speedup when running a stressmark in a CPU (left) and GPU (right). CPU and GPU stressmarks are DAXPY and DGEMM, respectively.

Note that these power values are higher than the *Target power* of each device, but neither the CPU nor the GPU will consume that much power because the workloads they are executing have a lower power consumption than the stressmarks, as reflected by a *RatioDevice* higher than 1.

Finally, after applying the new frequencies in the CPU and the GPU, the power consumption of both devices is the same as their *Target power* (note that it could be a bit lower), and the available power is used much more efficiently than in the initial state at ④. After the Power Distribution phase, the power consumption of the devices does not change. Therefore, from ⑤ until the end of the execution, APS keeps monitoring the system but does not change the frequencies.

5 Experimental Setup

We evaluate APS on an OpenPOWER system (PowerNV 8335-GTH) with 2 CPUs and 4 GPUs distributed in 2 sockets. Each socket has an IBM POWER9 processor [42] that runs by default at 3.00GHz and 2 NVIDIA Volta V100 GPUs [31] that run by default at 1.53GHz.

When all devices are running at their higest frequency, the system can have a power consumption of up to 1500W. Therefore, we use power caps that range between 66% and 40% of the peak power consumption, which represent multiple realistic scenarios within a system.

We implement APS as a process in the system. This process uses a single thread that always runs on core 0. The process measures execution time and utilization with perf [43] and does in-band power consumption readings from Linux to the OCC [22] for the CPUs and with the NVIDIA Management Library (NVML) [44] for the GPUs. The APS sampling rate is limited by the overheads of the measurement tools. Reading the power consumption and utilization of the CPUs and the GPUs takes on average 50ms and 105ms, respectively, and can be affected by the load of the system. Therefore, we set a sampling time of 200ms, which allows APS to recognize irregular behaviors.

Note that NVML measures the utilization of the GPU as a whole, but it does not allow to measure the utilization of each Streaming Multiprocessor (SM) of the GPU individually. Other tools such as libcupti or dcgmi allow reading per-SM utilization, but doing it in APS would introduce larger overheads and result in larger sampling times. APS could suffer from inaccurate utilization readings when GPU applications present a very unbalanced utilization across SMs. In our setup the GPU benchmarks use all the SMs, so this situation never happens. Similarly, the CPU utilization does not consider busy waiting synchronization mechanisms, which are not used in our CPU benchmarks.

TABLE 4: List of CPU (left) and GPU (right) benchmarks.

Name	Suite	Power	Name	Suite	Power
DAXPY	-	High	Tensorflow	-	High
BT	NAS	High	DGEMM	-	High
EP	NAS	High	Particlefilter	Rodinia	High
FT	NAS	High	Heartwall	Rodinia	Low
MG	NAS	High	Kmeans	Rodinia	Low
UA	NAS	High	Myocyte	Rodinia	Low
CG	NAS	Low	Srad (v1)	Rodinia	Low
LU	NAS	Low	Srad (v2)	Rodinia	Low
SP	NAS	Low	Quicksilver	CORAL	Low
Kripke	CORAL	Low	QMCPack	CORAL	Low
graph500	CORAL	Low	Lulesh	CORAL	Low

5.1 DVFS Capabilities

The CPUs in our system have 43 frequency levels (from 3.00GHz to 2.30GHz). APS adjusts the frequency of each CPU separately. When the frequency of a CPU is adjusted, all the cores of that CPU adjust their frequencies accordingly. APS does not control the frequencies of the cores individually. Lowering the CPU frequency has a linear impact on power and performance for a DAXPY stressmark as shown in Figure 4 (left). The GPUs can be set to run from 1.53GHz to 135MHz through the NVML, and the frequency of each GPU is set separately. We only use frequencies higher than 500MHz, since lower frequencies have a significant negative impact on performance, as shown in Figure 4 (right). To ensure that the physical power management of the system does not intervene in our experiments, we set an unrealistically high power cap at the hardware level that is never reached. Note that the single-threaded process that implements APS is affected by the changes CPU to the frequency, but we do not observe any performance implications because the execution time of the APS algorithm is very short.

5.2 Benchmarks

Table 4 shows the CPU and the GPU benchmarks used in our evaluation in the left and right columns, respectively, and their average power consumption (high/low). The benchmarks belong to the NPB [45], CORAL [46], and Rodinia [36] suites. We also train an Inception v3 neural network [47] with the ImageNet data set [48] in Tensorflow [37], and use a DAXPY kernel for the CPU, and a DGEMM kernel for the GPU. The execution time of all benchmarks is 200s to 375s running in isolation with no power cap.

5.3 Baselines

We compare APS against 3 static power distributions (Fairness, CPUprio, GPUprio) and a profile-based approach. We do not include the default OCC of the OpenPOWER system because, as seen in Section 3, it is very conservative and it is outperformed by all the other approaches.

For the Fairness, CPUprio, and GPUprio power distributions we set the power budgets of all the devices as specified

in Table 3. The power budget of the devices is constantly monitored and, if needed, their frequencies are corrected to honor their power budgets. If a device has available power headroom we increase its running frequency.

The profile-based static approach (Static Prof) finds the best power distribution for mixed workloads using offline profiling of the workloads and computing a Mixed-Integer Linear Programming (MILP) model. We measure offline the maximum power consumption and execution time of every benchmark individually for all the frequencies in our system, and then we apply the MILP model shown in Model 1 to select the frequencies of the devices that minimize the overall slowdown of all the benchmarks. At the beginning of the execution, Static Prof sets the frequencies of every device to the frequencies obtained by the model. Note that this model could achieve better results if it used percentiles instead of the maximum power consumption and execution time of the benchmarks, or if it used more features.

$$\begin{array}{ll} \text{minimize} & \sum I_{df} \times Slowdown_{df} \\ \text{subject to} & \sum_{i=0}^{6} I_{df} = 1 \\ & \sum I_{df} \times Power_{df} \leq \text{System power cap} \\ & I_{df} = \left\{ \begin{array}{c} 1 \text{ if device } d \text{ runs at frequency } f \\ 0 \text{ Otherwise} \end{array} \right\} \end{array}$$

Model 1: Mixed-Integer Linear Programming to set the frequency of the devices when using Static Prof. $Power_{df}$ is the maximum power measured in an offline profiling when the device d runs at frequency f. $Slowdown_{df}$ is the slowdown when the device d runs at frequency f with respect to the device d running at its highest frequency.

6 Evaluation

This section evaluates APS in a system running a single heterogeneous application (Section 6.1) and multiprogrammed workloads (Section 6.2). Finally, Section 6.3 compares APS with multiple state-of-the-art techniques.

6.1 Single CPU-GPU Applications

We evaluate APS with Tensorflow training an Inception v3 neural network using all the CPUs and GPUs in the system. The GPUs perform the main computation phase, but the CPU performance is relevant for (1) updating the weights of the neural network, (2) transferring data between CPUs and GPUs, and (3) doing I/O operations while GPUs are idle.

We experiment with Fairness, CPUprio, GPUprio, Static Prof, and APS with power caps of 1000, 900, 800, 700, and 600W. Results are shown in Figure 5. The y-axis shows the speedup with respect to Fairness and different power caps are represented in the x-axis. When no power cap is applied, this experiment consumes up to 1660W (1090W on average). Fairness degrades performance with respect to no power cap by 14.4%, 18.3%, 25.5%, 33.4%, and 48.1% for power caps of 1000, 900, 800, 700, and 600W, respectively.

Figure 6 shows a timeline of the power consumed by the CPUs and the GPUs running Tensorflow under Fairness with a power cap of 900W. The timeline shows a stable power consumption by the CPUs of between 59W and 75W in the whole execution. In contrast, the power consumption of the GPUs varies widely over time, with high and low power spikes of 50W to 150W. These power

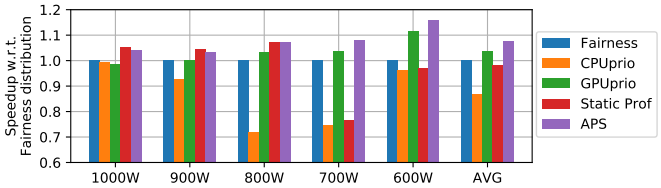


Fig. 5: Speedup of Tensorflow training an Inception v3 network with different power distributions and power caps.

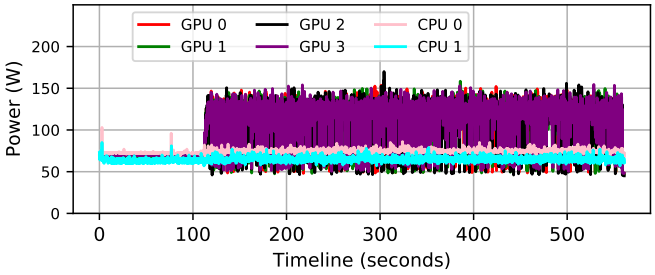


Fig. 6: Power consumption over time of the all the devices running Tensorflow under a Fairness power distribution.

spikes are caused by the characteristics of the workload in different program phases, since the power budgets and the frequencies of the GPUs are constant under Fairness. The average duration of the GPU phases with low and high power consumption are 404ms and 470ms, respectively.

Figure 5 shows that APS and GPUprio outperform Fairness. This workload relies on GPUs for the main computation, so the performance increases when the power distribution favors the GPUs. APS can respond to this behavior dynamically based on the power consumption and the utilization of the devices. Note that, the lower the power cap, the higher the speedup for APS and GPUprio. This happens because, as shown in Figure 4 in the GPU power-performance curve, the power consumption drops faster than the performance. Thus, the performance improvements of APS and GPUprio with respect to Fairness and CPUprio are higher as we lower the power cap. With a power cap of 600W, APS and GPUprio outperform Fairness by 15.9% and 11.5%, respectively. These speedups come from using higher frequencies for the GPUs, and the extra 4.4% of APS is due to running the CPU phases at a higher frequency. APS and GPUprio run the GPUs at a higher frequency than Fairness and APS further improves performance by running the CPUs at a higher frequency for the data transfers, the weight updates, and the I/O phases. Notice that the data transfers and the execution of the GPU kernels are overlapped, so there are no obvious program phases during the execution. However, the power consumption of the GPUs varies during the execution (as shown in Figure 6), which is exploited by APS to efficiently distribute power.

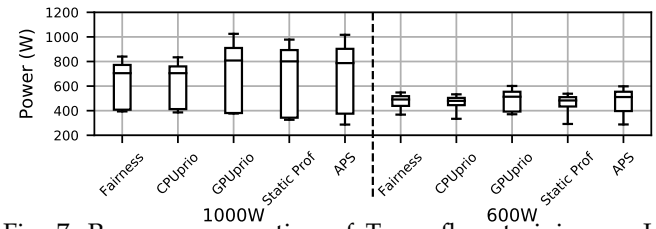


Fig. 7: Power consumption of Tensorflow training an Inception v3 network with different power distributions and power caps. Boxes represent from the 25th to the 75th percentiles and whiskers from the 5th to the 95th percentiles.

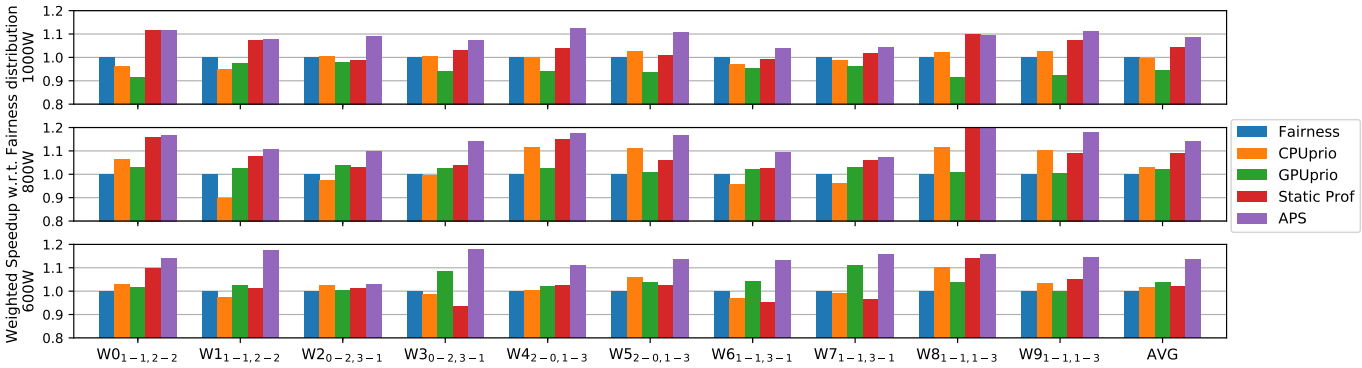


Fig. 8: Weighted speedup of mixed workloads (2 CPU and 4 GPU applications) running under different power caps.

CPUprio assigns power budgets favoring the CPUs. With a power cap of 700W, Fairness runs GPUs at 900MHz while CPUprio sets their frequency to 643MHz. This causes large performance differences because the GPUs power-performance trade-off is much better at 900MHz than at 643MHz, as shown in Figure 4. For high and low power caps the performance differences are smaller. For 600W, Fairness sets a very low power budget for the GPUs, which results in similar frequencies to the ones used by CPUprio. For 1000W and 900W there is enough power for CPUprio to set a high GPU frequency similar than Fairness.

Static Prof achieves a performance improvement with respect to Fairness of 5.2%, 4.3%, and 7.3% for power caps of 1000W, 900W, and 800W, respectively. For lower power caps of 700W and 600W, Static Prof has a performance degradation with respect to Fairness of 23.4% and 2.9%, respectively. When setting a power cap of 700W, Static Prof lowers the frequency of the GPUs to 690MHz, much less than the 900MHz set by Fairness, so the performance is greatly degraded. For 600W, the frequency of the GPUs is similar in Static Prof and Fairness, 645MHz and 723MHz, respectively.

In terms of power consumption, GPUprio, Static Prof, and APS have a similar power consumption for a power cap of 1000W. Yet, as we lower the power cap to 600W, GPUprio consumes similar power than APS without providing performance benefits due to the CPUs running at a lower frequency, while Static Prof fails to use the available power headroom due to the high power phases of Tensorflow.

Figure 7 shows the power consumption of the experiments with power caps of 1000W and 600W. The figure shows power consumption in the y-axis for all the static and our dynamic power distributions in the x-axis. The boxes represent from the 75^{th} to the 25^{th} percentile, the band inside the boxes represent the median power, and the whiskers represent from the 95^{th} to the 5^{th} percentile. It can be observed that, with a power cap of 1000W, GPUprio, Static Prof, and APS have a similar power consumption. Yet, as we lower the power cap to 600W, GPUprio consumes similar power than APS without providing performance benefits due to the CPUs running at a lower frequency, while Static Prof fails to use the available power headroom due to the high power phases of Tensorflow.

6.2 Mixed Workloads (CPU-GPU Applications)

This section evaluates APS in a power capped system running mixed workloads composed of CPU and GPU programs. The mixed workloads are composed of 2 CPU benchmarks and 4 GPU benchmarks randomly selected from Table 4. To refer to these mixed workloads, we use the following nomencalture:

$$WorkloadNumber_{CPUHigh-CPULow,GPUHigh-GPULow}$$

where *CPUHigh* and *GPUHigh* are the number of high-power consumption benchmarks (for CPU and GPU, respectively) and *CPULow* and *GPULow* are the number of low-power consumption benchmarks (for CPU and GPU, respectively). The devices start running their assigned benchmarks at the same time and benchmarks run until completion.

Figure 8 reports the weighted speedup for different workloads under 3 power caps (1000, 800, and 600W) using the Fairness, CPUprio, GPUprio, Static Prof, and APS power distributions. The x-axis shows 2 random workloads for 5 scenarios ($CPU_{High} - CPU_{Low}, GPU_{High} - GPU_{Low}$): (1) 1-1, 2-2; (2) 0-2, 3-1; (3) 2-0,1-3; (4) 1-1,3-1; (5) 1-1, 1-3. Figure 9 shows the average power consumption and power headroom for the mixed workloads with respect to the power cap. When no power cap is applied, these mixed workloads have a power consumption of up to 1614W (867.5W on average). On average, Fairness degrades performance with respect no power cap by 10.1%, 19.6%, and 30.1% for power caps of 1000, 800, and 600W, respectively.

Figure 8 shows that APS achieves the best performance across all the workloads and power caps, achieving average weighted speedups over Fairness of 8.8%, 13.9%, and 13.6% for power caps of 1000, 800, and 600W, respectively. The best power distribution heavily depends on the characteristics of each workload, so the static approaches fail at improving performance for all workloads, while APS consistently achieves speedups in all cases by intelligently shifting the power to the devices that better utilize it. For this reason, Figure 9 shows that APS increases the average power consumption over Fairness by 6.8%, 5.9%, and 6.3% for power caps of 1000, 800, and 600W, respectively, while respecting the power cap. Notice that the available power headroom also depends on the workload power consumption behavior. On average, the time over the power cap for APS is 1.2%, 3.1%, 4.9% for power caps of 1000, 800, and 600W, respectively.

CPUprio outperforms Fairness in several workloads. In particular, workloads 4, 5, 8 and 9 are running 3 low-power GPU benchmarks and at least 1 high-power CPU benchmark, so shifting power to CPUs improves performance. The opposite situation happens in workloads 1, 6, and 7, so the performance of CPUprio is worse than Fairness. As

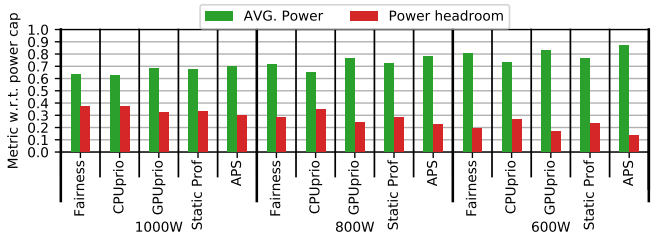


Fig. 9: Average power consumption and power headroom for different power distributions under different power caps when running different mixed workloads.

seen in Figure 9, CPUprio is the approach that uses the least power, with power headrooms of 36.6%, 36.5%, and 29.6% for power caps of 1000, 800, and 600W, respectively.

GPUprio achieves similar or worse performance than Fairness with a power cap of 1000W. This is because CPUs run at the lowest frequency and the GPUs at the highest frequency, and the slowdowns in the CPU benchmarks outweigh the speedups in the GPU benchmarks. With lower power caps, GPUprio achieves better performance than Fairness due to the power-performance curve of GPUs (shown in Figure 4), specially in workloads where 3 GPUs are running high-power benchmarks (workloads 1, 3, 6, and 7). Consequently, the average power consumption of GPUprio over Fairness increases as the power decreases, as shown in Figure 9.

Static Prof achieves average performance improvements of 4.3%, 8.8%, and 2.1% with respect to Fairness for power caps of 1000W, 800W, and 600W. For a power cap of 600W, Static Prof degrades performance for workloads 3, 6, and 7 due to GPUs running high-power benchmarks. In higher power caps, this behavior is less noticeable due to the power-performance curve of the GPUs (as shown in Figure 4). Static Prof is a conservative approach in workloads with changing power behaviors, which can present a large power headroom in some application phases. As we can observe in Figure 9, Static Prof uses 3.9% less power on average than Fairness for a power cap of 600W.

6.3 Comparison with State-of-the-Art

This section compares APS with two state-of-the-art techniques: a market-based solution [25] and Tangram [24].

We implement a market solution based on [6], which is aimed to distribute multiple resources (e.g. cache, off-chip bandwidth, and power) among multiple CPUs. We adapt the algorithm to distribute power between CPUs and GPUs. In our market solution (*Market*), agents (e.g. CPUs and GPUs) bid for power based on their expected power utility. Power utility is modeled based on the fact that the length of the compute phase tends to scale linearly with the processor frequency. Agents measure their compute and memory phases from the last interval to compute their power utility. Then, a centralized resource arbiter collects bids, adjusts the price of the resource based on the bids and the power availability, and publishes the new price. Agents bid again based on the new price and their utility. We set the budget for bidding for the agents as their maximum TDP. In our setup, GPUs have a higher TDP and higher budget for bidding than CPUs. If the power surpasses the power cap, all the devices reduce their power budget equally. Note that this implementation, as proposed in the original paper [6], does not take advantage of real-time power measurements, limiting the ability of the bidders to bid accurately. Using real-time measurements in Market is left for future work.

We also compare APS with Tangram [24], which manages the power budgets of the devices in heterogeneous systems to minimize the overall Energy-Delay Product (EDP). Tangram has two operating modes: (1) preconfiguration mode and (2) enhancement mode. The preconfiguration mode is used when the computation only uses one device type, and it applies a static power distribution for all the devices. We extend the static power distributions to fit our larger-scale experimental framework. The enhancement mode is used when all the devices in the system are active. This mode applies a Nelder-Mead search to find the power distribution that minimizes EDP. The Nelder-Mead search is leveraged by throughput, which is read by collecting performance counters in the CPUs and the GPUs. As described in [24], if the power goes above the power limit, the devices are set to their lowest frequencies and the Nelder-Mead search is restarted. Also, there is 5% probability to restart the search. Since the power is already limited to a given power cap, we set Tangram to maximize overall throughput. Note that, although Tangram is proposed as a hardware technique, in the original manuscript [24] it is evaluated using a software implementation. For our comparison with APS, we use a software implementation of Tangram that has a single Nelder-Mead search.

Note that systems under APS, Tangram and Market (as well as under other state-of-the-art solutions) can surpass the power cap. This happens if, at the moment the power consumption is stable and maximized under a power cap, a device enters a program phase with a higher power consumption, or if the power cap is decreased due to external circumstances such as power or thermal emergencies. For this reason, the three solutions include a control mechanism that ensures the power cap is always respected.

Figure 10 shows the average weighted speedup of Market and Tangram with respect to APS for the mixed workloads evaluated in Section 6.2 with power caps of 1000W, 800W and 600W. The whiskers represent the range of speedups achieved by Market and Tangram in all the workload mixes.

Results show that, compared to APS, Market presents average slowdowns of 3.5%, 2.9% and 7.5% for power caps of 1000W, 800W and 600W, respectively. In addition, the whiskers show that Market does not improve performance in any workload mix with any power cap compared to APS. Market performs worse than APS in the workload mixes that benefit from assigning more power to the CPUs (workloads 4 and 5) because, in our implementation where the agents have a bidding budget of their maximum TDP, GPUs have a higher bidding budget, so the algorithm is biased to power distributions that favor GPUs while CPUs starve for power. As a consequence, we observe large imbalances in the power budgets of the GPUs and the CPUs, which end up causing performance inefficiencies in the CPU-intensive mixed workloads because the speedups in the GPU workloads are overweighted by the slowdowns in the CPU workloads. In addition, we also observe that

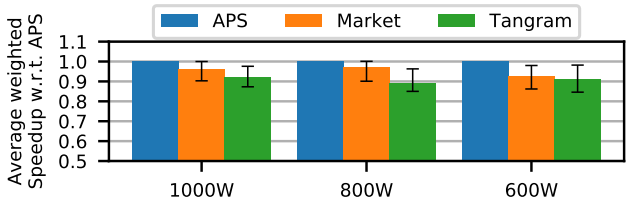


Fig. 10: Average weighted speedup of Market and Tangram with respect to APS for mixed workloads (from Figure 8) with different power caps. Whiskers represent the range of speedups achieved in all the workload mixes.

Market introduces performance penalties when GPUs bid intensively for power (workloads 6 and 7). Since agents only consider power utility but not the current power consumption of the devices, they bid for power independently of low or high power phases, which are common in GPU workloads. This can lead to power distributions that do not honor the system power cap, forcing an emergency power reduction in all the devices to lower the consumption below the power cap. All together, we observe that Market is not able to correctly balance the power budgets of the CPUs and the GPUs in some cases. This problem is not caused by the fundamental idea behind the market approaches in general nor by the algorithm that makes the decisions. Instead, the main problem of Market is the implementation of the bidding capacities of the devices. In our implementation the bidding capacity of a device is its maximum TDP, so there is a big disparity in the bidding power of the CPUs and the GPUs. In general, bids in a market mechanism should reflect the actual value that the bidding process obtains from using the resource. Thus, if bids are set heuristically, the Market algorithm is subject to the vagaries of that heuristic. The problem of finding an optimal bidding mechanism for systems with CPUs and GPUs has never been addressed because Market algorithms have never been studied in this context. This research direction is left for future work.

Figure 10 also shows that Tangram offers less performance than APS, with average slowdowns of 7.2%, 9.5% and 8.5% under power caps of 1000W, 800W and 600W, respectively. As before, all the workloads mixes perform worse with Tangram than with APS. We observe that the main reason for this performance disparity is that, in some workload mixes, Tangram suffers from slow reactions to changing power behaviors. The Nelder-Mead search is a complex algorithm with a long search phase, which compromises response time and scalability for a large number of devices. Therefore, in workload mixes where power consumption varies considerably along the execution (workloads 4, 6, 7, and 9), the Nelder-Mead search misses opportunities to boost the overall performance. Another important reason for the performance differences are the unbalanced power distributions that Tangram uses for some workload mixes (workloads 0, 1, 2, 3, and 8), which are caused by the nature of the Nelder-Mead search. The algorithm assigns more power to the devices with a higher throughput so, when it assigns more power to the GPU with the highest throughput, its throughput is repeatedly augmented at every iteration of the Nelder-Mead search, while the rest of devices are never assigned more power. In some phases of the execution we see that this behavior creates big imbalances in the power budgets of the devices. As a result, an important speedup is achieved for the single application running on the GPU with the highest throughput, but the performance does not change in the rest of applications running in the other devices. In contrast, APS distributes the power in a much more balanced way, so the speedup achieved by the application running on the GPU with the highest throughput is not as large as in Tangram, but it is compensated by the speedups obtained in the rest of applications, resulting in an average weighted speedup larger than Tangram.

On average, Tangram performs slightly worse than Market for all the power caps. However, in many workload mixes they achieve very similar performance, as shown by the whiskers. In some workload mixes Tangram is not as effective as Market because of the slower Nelder-Mead search. In other cases (workloads 4 and 9) the reason is the imbalance between the power budgets of the CPUs and the GPUs, which happens in both Market and Tangram, but it is more extreme in Tangram. Tangram assigns as much power budget as possible to the GPUs with maximum throughput and does not increase the power budget of the CPUs. In Market the GPUs have a higher bidding budget than the CPUs, so a big portion of the power is shifted to the GPUs, but the CPUs still get a small portion of the power headroom, so the distribution is not as extremely unbalanced as in Tangram.

6.4 APS Design Discussion

This section shows the importance of the freq_power tables, the RatioDevice, and the Force Power Cap phase of APS. As explained in Section 4.1, the freq_power table of a device contains the maximum power consumption for all the frequencies, and is generated offline by running a stressmark.

To demonstrate the benefits of the proposed design, we compare APS against 4 alternative implementations: *APS-NoScalePower (APS-NSP)* does not scale the current power consumption, so the RatioDevice is always 1. *APS-SpecificBenchmarkTables (APS-SBT)* uses a freq_power table for every benchmark instead of a freq_power table generated with a stressmark. The freq_power tables are generated offline by running all the benchmarks on their devices at every available frequency. *APS-NoForcePowerCap-Min (APS-NFPC-Min)* uses a constant RatioDevice of 1 and, when a power cap is introduced, reduces the frequencies of all the devices to their minimum. *APS-NoForcePowerCap-Max (APS-NFPC-Max)* uses a constant RatioDevice of 1 and, when a power cap is introduced, it does not change the frequencies of the devices. The last two variants use the Power Distribution phase to iteratively adjust the frequencies of the devices until their power consumption is equal to their power budget and the system power cap is met.

Figure 11 shows the average weighted speedup with respect to Fairness for the mixed workloads evaluated in Section 6.2 with a power cap of 800W. Results show that APS outperforms all the alternative designs except APS-SBT. Compared to APS, APS-NSP, APS-NFPC-Min and APS-NFPC-Max present performance losses of 1.5%, 4.6% and 2.2%, respectively. These slowdowns happen because these alternative implementations do not scale the power of the

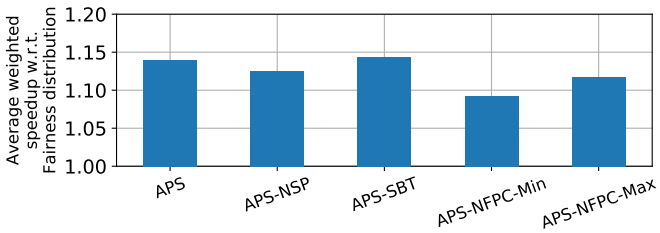


Fig. 11: Average weighted speedup of different APS variants with respect to Fairness for mixed workloads (from Figure 8) with a power cap of 800W.

running applications to the one observed with the stressmark, so more iterations of the Power Distribution phase are required to optimally distribute the power. The performance differences between APS-SBT and APS are below 1% because the specific freq_power tables per benchmark achieve the same goal as the one generated by the stressmark. However, generating the freq_power tables per benchmark requires a huge amount of executions to profile all the applications running on their devices at all the frequencies, which can be avoided by combining the RatioDevice and a single freq_power table per device generated with a stressmark. We adopt this approach to gain this simplicity.

7 Conclusions

Current systems include multiple discrete devices per node. Unfortunately, power constraints limit the number of devices that can operate simultaneously at their highest frequency. To efficiently distribute the power in such systems, dynamic power management techniques that consider device utilization are needed. This work presents APS, a mechanism that leverages system utilization to distribute the available power among multiple discrete devices in power-constrained multi-accelerator heterogeneous systems. APS dynamically adjusts the power budget of the devices based on their utilization, allowing highly-utilized devices to have a higher power budget than low-utilized devices.

We evaluate APS on an OpenPOWER system with 2 CPUs and 4 GPUs. APS outperforms the Fairness static power distribution, with speedups of up to 15.9% in single CPU-GPU applications and weighted speedups of up to 20.5% in multiprogrammed workloads. APS also outperforms Tangram and Market for multiprogrammed workloads, improving the weighted speedup up to 14.9% and 13.8%, respectively, with a simpler implementation.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Innovation (contract PID2019-107255GB-C21/AEI/10.13039/501100011033), by the Generalitat de Catalunya (contract 2017-SGR-1328), and by the IBM/BSC Deep Learning Center initiative.

Ll. Alvarez has been supported in part by the Spanish Ministry of Economy, Industry and Competitiveness under the Juan de la Cierva Formacion fellowship No. FJCI-2016-30984. M. Moreto has been supported in part by the Spanish Ministry of Economy, Industry and Competitiveness under Ramon y Cajal fellowship No. RYC-2016-21104.

References

[1] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.
[2] N. Hardavellas *et al.*, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. 4, pp. 6–15, 2011.
[3] R. Murphy *et al.*, "On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implications," *IEEE Transactions on Computers*, vol. 56, no. 7, pp. 937–945, 2007.
[4] M. Arora *et al.*, "Redefining the Role of the CPU in the Era of CPU-GPU Integration," *IEEE Micro*, vol. 32, no. 6, pp. 4–16, 2012.
[5] C. Isci *et al.*, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006, pp. 347–358.
[6] X. Wang *et al.*, "XChange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 113–125.
[7] R. Bitirgen *et al.*, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 318–329.
[8] J. A. Winter *et al.*, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2010, pp. 29–39.
[9] R. Cochran *et al.*, "Pack & Cap: Adaptive DVFS and thread packing under power caps," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 175–185.
[10] S. Reda *et al.*, "Adaptive Power Capping for Servers with Multithreaded Workloads," *IEEE Micro*, vol. 32, no. 5, pp. 64–75, 2012.
[11] A. Vega *et al.*, "Crank It Up or Dial It Down: Coordinated Multiprocessor Frequency and Folding Control," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46, 2013, pp. 210–221.
[12] H. Zhang *et al.*, "Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16, 2016, pp. 545–559.
[13] V. Srinivasan *et al.*, "An innovative approach to dynamic platform and thermal management for mobile platforms," in *2010 International Conference on Energy Aware Computing*, 2010, pp. 1–4.
[14] "Advanced Micro Devices, BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 10h-1FFh Processors." http://developer.amd.com/resources/developer-guides-manuals/.
[15] P. E. Bailey *et al.*, "Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems," in *2014 43rd International Conference on Parallel Processing*, 2014, pp. 371–380.
[16] T. Komoda *et al.*, "Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 349–356.
[17] A. Majumdar *et al.*, "Dynamic gpgpu power management using adaptive model predictive control," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 613–624.
[18] Q. Jiao *et al.*, "Improving gpgpu energy-efficiency through concurrent kernel execution and dvfs," in *2015 International Symposium on Code Generation and Optimization (CGO)*, 2015, pp. 1–11.
[19] I. Paul *et al.*, "Cooperative boosting: Needy versus greedy power management," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013, pp. 285–296.
[20] I. Paul *et al.*, "Coordinated energy management in heterogeneous processors," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.
[21] H. Wang *et al.*, "Workload and power budget partitioning for single-chip heterogeneous processors," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 401–410.
[22] T. Rosedahl *et al.*, "Power/Performance Controlling Techniques in OpenPOWER," in *High Performance Computing*, 2017, pp. 275–289.
[23] T. Burd *et al.*, ""zeppelin": An soc for multichip architectures," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 133–143, 2019.

[24] R. P. Pothukuchi *et al.*, "Tangram: Integrated control of heterogeneous computers," in *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 384–398.
[25] M. Guevara *et al.*, "Navigating heterogeneous processors with market mechanisms," in *IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 95–106.
[26] W. Felter *et al.*, "A performance-conserving approach for reducing peak power consumption in server systems," in *Proceedings of the 19th Annual International Conference on Supercomputing*, 2005, pp. 293–302.
[27] Q. Deng *et al.*, "CoScale: Coordinating CPU and Memory System DVFS in Server Systems," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 143–154.
[28] H. Ribic *et al.*, "AEQUITAS: Coordinated Energy Management Across Parallel Applications," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16, 2016, pp. 4:1–4:12.
[29] E. Rotem *et al.*, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.
[30] R. Ayoub *et al.*, "Os-level power minimization under tight performance constraints in general purpose systems," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 321–326.
[31] "NVIDIA TESLA V100 GPU ARCHITECTURE," https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.
[32] A. Adileh *et al.*, "Maximizing heterogeneous processor performance under power constraints," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 3, Sep. 2016.
[33] A. Adileh *et al.*, "Mind the power holes: Sifting operating points in power-limited heterogeneous multicores," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 56–59, 2017.
[34] N. C. Nachiappan *et al.*, "Domain knowledge based energy management in handhelds," in *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 150–160.
[35] Y. Liu *et al.*, "Fastcap: An efficient and fair algorithm for power capping in many-core systems," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 57–68.
[36] S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.
[37] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.
[38] M. Guevara *et al.*, "Strategies for anticipating risk in heterogeneous system design," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 154–164.
[39] R. P. Pothukuchi *et al.*, "Yukta: Multilayer resource controllers to maximize efficiency," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 505–518.
[40] X. Wang *et al.*, "Rebudget: Trading off efficiency vs. fairness in market-based multicore resource allocation via runtime budget reassignment," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2016, Atlanta, GA, USA, April 2-6, 2016*, T. Conte *et al.*, Eds. ACM, 2016, pp. 19–32. [Online]. Available: https://doi.org/10.1145/2872362.2872382
[41] K. Rao *et al.*, "Application-specific performance-aware energy optimization on android mobile devices," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 169–180.
[42] S. K. Sadasivam *et al.*, "IBM Power9 Processor Architecture," *IEEE Micro*, vol. 37, no. 2, pp. 40–51, 2017.
[43] A. C. de Melo, "The new linux perf tools," 2010.
[44] "NVIDIA Management Library (NVML)," https://docs.nvidia.com/deploy/nvml-api/index.html.
[45] H.-Q. Jin *et al.*, "The OpenMP implementation of NAS parallel benchmarks and its performance," 1999.
[46] "CORAL Benchmark Codes." [Online]. Available: https://asc.llnl.gov/CORAL-benchmarks/
[47] C. Szegedy *et al.*, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.
[48] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, 2015.

AUTHOR BIOGRAPHIES

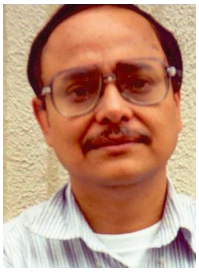
Cristobal Ortega is a PhD. Student at the Universitat Politècnica de Catalunya (UPC) and the Barcelona Supercomputing Center (BSC). He received his M.S. degree in 2016 and graduated in Engineering in computer science in 2014 from the Universitat Politecnica de Catalunya (UPC).

Lluc Alvarez is a researcher at the Barcelona Supercomputing Center (BSC) and a lecturer at the Universitat Politècnica de Catalunya (UPC). He received his B.Sc. degree from the Universitat de les Illes Balears (UIB) in 2006 and his M.Sc. and Ph.D. degrees from the UPC in 2009 and 2015.

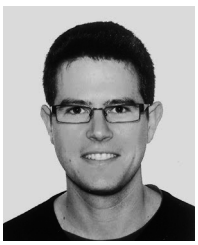
Alper Buyuktosunoglu received PhD degree in electrical and computer engineering from University of Rochester. Currently, he is a Principal Research Staff Member in Efficient and Resilient Systems department at IBM T. J. Watson Research Center. He has been involved in research and development work in support of IBM Power Systems and IBM z Systems in the areas of computer architecture and robust power management. He has over 100 patents, has received several IBM-internal awards, has published over 100 papers, and has served on various conference technical program committees in these areas. He also has served on the editorial board of IEEE MICRO. Dr. Buyuktosunoglu is a member of the IBM Academy of Technology, Hall-of-fame of MICRO, Hall-of-fame of HPCA, an IBM Master Inventor and an IEEE Fellow.

Ramon Bertran received the Ph.D. degree in computer architecture from the Polytechnic University of Catalonia, Barcelona, Spain, in 2014. He is currently a Research Staff Member with the Efficient and Resilient Systems Research Group, IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He is involved in research and development work on power-aware computer systems.

Todd Rosedahl IBM Chief Power/Thermal/Energy Management Engineer on POWER. Todd has worked on power, thermal, and energy management for his entire 25yr career at IBM and has over 20 related patents.



Pradip Bose is a Distinguished Research Staff Member and manager of the Efficient and Resilient Systems Research Group. His research and supervisory responsibilities are currently focused on energy-efficient and reliable systems and domain-specific accelerators, with relevance to cloud computing and artificial intelligence.



Miquel Moretó is a Ramón y Cajal fellow at the Universitat Politècnica de Catalunya (UPC) and an associate researcher at the Barcelona Supercomputing Center (BSC). Prior to joining UPC, he was a Fulbright post-doctoral fellow at the International Computer Science Institute (ICSI), Berkeley, USA. He received the Ph.D. degree from UPC in 2010.