A Runtime Resource Management Policy for OpenCL Workloads on Heterogeneous Multicores

Daniele Angioletti[b], Francesco Bertani[c], Cristiana Bolchini[a], Francesco Cerizzi[b], Antonio Miele[a]
Politecnico di Milano – Dip. Elettronica, Informazione e Bioingegneria – Italy
[a] *{first_name.last_name}@polimi.it* [b] *{first_name.last_name}@mail.polimi.it* [c] *francesco2.bertani@mail.polimi.it*

***Abstract*—Nowadays, runtime workload distribution and resource tuning for heterogeneous multicores running multiple OpenCL applications is still an open quest. This paper proposes an adaptive policy capable at identifying an optimal working point for an unknown multiprogrammed OpenCL workload without using any design-time application profiling or analysis. The approach compared against a design-time optimization strategy demonstrates to be effective in converging to an solution guaranteeing required performance while minimizing power consumption and maximum temperature; it achieves on average values 0.085 W (5.15%) and 0.83°C (1.47%) worse than the static optimal solution.**

***Index Terms*—OpenCL, Hetenrogeneous System Architectures, Runtime Resource Management.**

I. Introduction

Nowadays, system architectures are heterogeneous, integrating various kinds of processing units, such as asymmetric multicore CPUs, GPUs and HW modules. Two easily accessible commercial examples are the Samsung Exynos 5422 chip, integrating an ARM big.LITTLE multicore CPU and an ARM Mali GPU, and the Nvidia Tegra, featuring a similar multicore and an Nvidia GPU. These architectures are particularly attractive because of the availability of a variety of operating points, allowing one to finely tune the achievable performance and power/energy consumption. This represents an opportunity to adapt and optimize workload execution, also based on the peculiarities of each application.

In this scenario, OpenCL enables the programmability of the heterogeneous resources by means of a single program implementation to exploit data parallelism. However, runtime resource management remains an open challenge, because OpenCL requires the designer to explicitly select and configure the specific processing unit where to execute an application.

Indeed, in those scenarios where such architectures can be efficiently exploited, this constitutes a relevant limitation when considering that 1) the workload is highly variable and not necessarily known in advance, 2) each application may have different Quality of Service (QoS) requirements (in terms of a target throughput) and its execution on different processing units may lead to different performance, 3) there are many knobs to be tuned (e.g., Dynamic Voltage and Frequency Scaling (DVFS) and level of parallelization), and, finally, 4) devices may present system-level requirements and constraints, in terms of power budget and temperature limits, especially in mobile and embedded scenarios. Thus, a coordinated tuning and control of both system architecture and application is paramount to optimizing workload execution and resource usage.

In this paper, we propose an adaptive runtime resource management policy minimizing power consumption and chip temperature for architectures featuring a big.LITTLE CPU and a GPU, executing a multiprogrammed OpenCL workload with QoS application-level constraints. The main contributions are:

- a purely runtime adaptive strategy, not requiring any design-time application profiling;
- support for multiprogrammed OpenCL workload;
- coordinated use of application mapping, CPU quota, level of parallelization, and CPU/GPU frequency tuning;
- targeting the fulfillment of QoS requirements while minimizing system power consumption and temperature peak.

Experimental results show that the runtime solution based on the proposed policy has comparable performance with respect to a design-time approach, without the need of in-advance knowledge and the time/cost for performing the design space exploration and storing/accessing the pre-computed optimal solution. More precisely, the approach rapidly converges and fulfills the workload throughput requirements (less than 3.25% of performance violations) with power consumption and maximum temperatures close to those of the optimal configuration identified at design time, achieving on average values 0.085 W and 0.83°C penalty. As such, the approach is interesting in situations where a design-time solution cannot be computed because of the unpredictability of the application scenario or the too many possible alternatives.

The paper is organized as follows. Section II discussed the related work, and Section III introduces the background on the target system. Section IV presents an empirical analysis of the effect of various knobs, leading to the definition of the runtime resource management policy later detailed in Section V. Section VI proposes the experimental validation of the policy, and Section VII draws the conclusions.

II. Related Work

One of the first approaches for the runtime resource management in heterogeneous multicores has been defined in [1]; the controller adapts the execution of single-threaded applications on big.LITTLE architecture to optimize power consumption while satisfying application level performance requirements. Subsequent approaches proposed more advanced controllers managing a single multi-threaded application [2], or multiple ones in the same scenario [3]. None of such approaches

978-3-9819263-2-3/DATE19/©2019 EDAA

considers thermal issues or GPU acceleration. Other approaches consider coordinated CPU-GPU power or thermal management [4], [5], but they consider a single multi-threaded application whose functionalities are already mapped on CPU or GPU; thus only DVFS tuning is actuated to guarantee the required performance. A final group of approaches adopts OpenCL to enable application mapping of the same kernel on CPU and GPU. However, due to the many actuation knobs that define too complex an optimization problem, most of the approaches act at design time (e.g. [6], [7]), or at runtime but considering a single application (e.g. [8]). An existing solution similar to our work is the one proposed in [9]. Differently from our approach, it performs a time-consuming design time profiling of each application to be executed to identify the optimal execution point in terms of power consumption. Such information is used at runtime to optimize overall power consumption while executing a workload composed of an unknown mix of such applications. Later the approach has been enhanced also to consider temperature [10].

Indeed, none of the these solutions tackles runtime resource management in a coordinated manner, without any design-time information gathering, the gap we aim at filling.

III. Background

System Architecture. The solution we propose can be exploited on any heterogeneous architecture, featuring asymmetric CPUs and GPUs and running Linux Operating System (OS). Here we adopted a popular mobile computing board, the Odroid XU3 [11] featuring a Samsung Exynos 5422 chip. It integrates an ARM big.LITTLE multicore with four cores per cluster, and an ARM Mali GPU. CPU frequency can be tuned at cluster level between 200MHz and 2GHz on the big core and from 200MHz to 1.4GHz on the LITTLE. The chip is provided with four power sensors connected to the big, LITTLE, GPU and memory units, respectively, and temperature sensors for each big core and for the GPU.

Target Applications. We envision a variable workload composed of streaming parallel applications (e.g., video processing ones) having a QoS requirement in terms of a minimum throughput. Applications are characterized by a computationally intensive kernel, repeatedly executed on input data (e.g., video frames). The adopted OpenCL benchmark suite is the PolyBench for GPU processing, modified to 1) iteratively execute the computational kernel, and 2) perform a discovery and setup of available OpenCL devices and to enable at each iteration the choice of the actual processing unit.

OpenCL Runtime. On the Odroid XU3, OpenCL execution is supported by the vendor for the Mali GPU only. An open-source runtime, Portable OpenCL (PoCL) library [12], has been installed for CPU execution, and concurrent discovery of runtimes has been enabled with an OpenCL ICD Loader.

Runtime Resource Controller. We implemented a controller similar to the one proposed in [13]. It consists of a process running in user-space, capable of 1) accessing all HW sensors and knobs through Linux interface, 2) controlling process allocation by means of the *cgroups* mechanism, and 3) measuring applications' performance in terms of throughput with the HeartBeat mechanism. The HeartBeat interface is used by the controller to set the OpenCL device to be used by each running application. Similarly to other OpenCL runtimes for CPU, PoCL sees ARM big.LITTLE CPU as a single uniform device, ignoring the asymmetric multicore configuration and thus leading to performance inefficiencies. The adopted controller exploits *cgroups* to mimic the availability of two separate devices, the big and the LITTLE clusters.

IV. Architecture Characterization

This section reports the results of the systematic analysis of the effects of knobs on performance, power consumption and temperature. It is the basis for characterizing the specific architecture power/performance models, and for deriving the knowledge for the runtime resource management policy, in defining a priority in using knobs to achieve the desired behavior.

A. Power and Performance Models

We adopted well-known estimation models and we empirically computed the characterizing parameters from the experimental campaigns executed with the reference board.

Power model. Borrowing from [4], [14], power consumption P_i of core i can be represented as a function of the frequency level f_i (and corresponding voltage V_i) and utilization U_i due to the running applications:

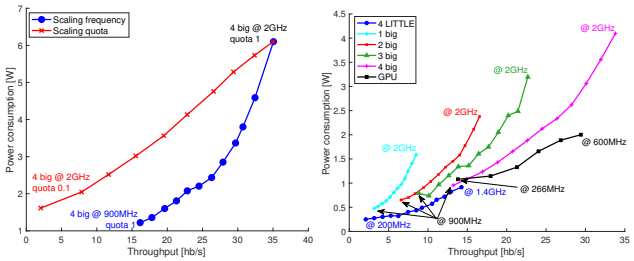
$$P_i = a[f_i] \cdot U_i + b[f_i] \tag{1}$$

a and b are two constants empirically derived for each kind of core (big, LITTLE and GPU) and for each frequency f_i, which is the same for the whole cluster. Finally, U_i is directly provided by the OS, and the assigned CPU quota is directly proportional to the utilization. Indeed, as shown in [4], [14], an application-agnostic power model (considering utilization only) offers a reasonable accuracy to properly take decisions in the proposed control loop.

Performance model. Performance models have been developed to estimate throughput due to configuration variations only on the CPU. Throughput (th_j) of an application j (or in general its performance) running on a single cluster i (either big or LITTLE) has an almost linear relationship with the CPU quota Q_j assigned to the application and cluster's frequency level f_i, whereas there is a sublinear relationship with the parallelization level $\#t$ (in number of assigned CPU cores), as shown in [4], [13]. Furthermore, as in [3], it is reasonable to estimate an average performance ratio between big and LITTLE clusters at the same baseline frequency, namely $r_{b \to L}$. A generic value of such parameter is computed at design time on a characteristic workload; then, it may be also tuned at runtime for each application to have more accurate estimations based on throughput measures.

Therefore, given the current configuration $curr$, we can estimate the performance in a new configuration as:

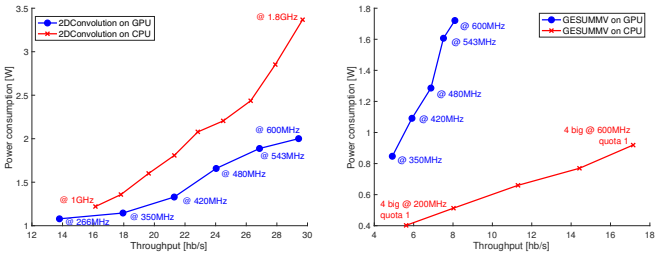
$$th_{new} = r_{b \to L}^{\gamma} \cdot \frac{Q_{new}}{Q_{curr}} \cdot \left(\frac{f_{new}}{f_{curr}} \right) \cdot \left(\frac{\#t_{new}}{\#t_{curr}} \right)^{\beta} \cdot th_{curr} \tag{2}$$



(a) Frequency vs. CPU quota effects, using the big core cluster

(b) Frequency and use of different resources (big, LITTLE, GPU)

Figure 1. big.LITTLE running 2DCONV with different settings



(a) 2DCONV

(b) GESUMMV

Figure 2. Frequency and use of big/GPU effects

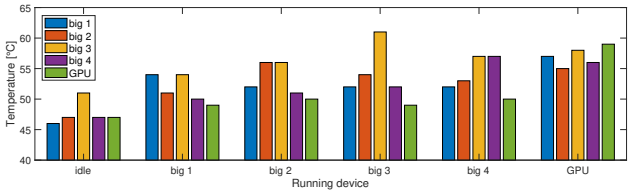


Figure 3. Maximum device temperature due to execution on another device

where $\beta \in (0;1)$ takes into consideration the overhead for parallel execution over multiple cores in the same cluster, and γ is 1 when there is a migration from the big to the LITTLE cluster, -1 from the LITTLE to the big one, and 0 otherwise.

B. Knobs Effects

Figures 1-2-3 show the measured throughput, power consumption and temperature in the different configurations, and how they vary (linearly or not) when we change knobs settings, as discussed in the following.

DVFS vs. CPU quota. Effects of DVFS and CPU quota on performance and power consumption have been already investigated [1]; we here recap the results in the scenario of OpenCL applications. Figure 1(a) shows the throughput (in heartbeat/s.) and the corresponding power consumption (sum of all power sensors) of a 2DCONV application running on four cores of the big cluster when varying the frequency level (from 2GHz down to 1GHz) using a 100% CPU quota, and, dually, when varying the CPU quota (from 100% down to 10%). As clearly shown, varying the frequency allows one to minimize power consumption while maintaining the same performance trend; indeed, reducing the static power consumption is much more effective than acting on the dynamic also with parallel applications.

Parallelization vs. DVFS. Similarly, we analyzed the change of the frequency level against the number of threads spawned on different cores (from 4 down to 1). Figure 1(b) reports the results for 2DCONV; each series of data represents a configuration with a different number of big cores scaling in frequency from 600MHz to 1.2GHz with a 100MHz step. The graph shows that the power consumption slope with respect to the throughput is much steeper in configurations using fewer cores; moreover achieving a certain throughput, the configurations with more cores exhibit a lower power consumption. This is the direct consequence of the fact that the same throughput can be obtained at lower frequencies by using more cores in parallel. Thus, to achieve a certain throughput level, maximizing the parallelism level to select the minimum frequency level is beneficial for power consumption. Finally, by comparing results in Figures 1(a) and 1(b), we can also conclude that to obtain the same throughput it is much more effective in terms of power consumption to scale parallelism level than the quota.

LITTLE vs. big. Experimental results confirmed that as long as the LITTLE cores allow to reach the required throughput, their use leads to lower power consumption and lower temperatures, therefore they are always preferable to the big ones.

CPU vs. GPU. Similarly to the LITTLE cluster, the GPU can provide performance similar to the big cluster when running at high frequency levels. However, as shown in Figure 2, different applications (i.e., 2DCONV and GESUMM) exhibit a different performance/power trade-off on big cluster and GPU: while the former performs better on the big cluster, the latter shows a lower power consumption when running on the GPU and achieving the same throughput. Therefore we cannot define a-priori the best power-efficient allocation between GPU and big cluster, but an online profiling is required.

Temperature analysis. Finally, we performed a systematic analysis of the chip temperatures; to speed-up the process we built a thermal model based on the classical RC thermal network for steady-state temperature by using linear regression (as in [5]). We report in Figure 3 a representative subset of the results consisting of the measured temperatures (four big cores and GPU) when each single unit is working; we noticed that big cores are not distributed symmetrically in the chip thus there is a sensible effect on the reached maximum temperatures due to self-heating and thermal interference among cores. We explored this asymmetry and concluded that when considering the same operating conditions (in terms of resource utilization and frequency), an ordering of the incurred maximum temperature can be identified in the big cluster: *big1 - big4 - big2 - big3*. Moreover this ordering is independent of the fact that the GPU is working or idle. Given this empirical knowledge, the runtime resource management policy can directly exploit this ordering in selecting the resources when aimed at minimizing the maximum temperature, rather than using the derived thermal model.

Final Remarks. The analysis has been used to derive model

parameters a, b and γ in Eq. (1) and (2). The second outcome is a so-called *knobs priority order*, that is the definition of an empirical preference in exploiting the available knobs to achieve the required throughput while minimizing power consumption: 1) LITTLE/big/GPU mapping, 2) DVFS, 3) parallelism, and 4) CPU quota. Online application profiling is necessary to decide whether to use CPU or GPU, and from the thermal point of view, an opportunistic selection of the big cores has to be performed to limit the maximum temperature.

V. The Proposed Runtime Management Policy

Based on the discussed observations, we designed a runtime resource management policy aimed at:

- dynamically mapping parallel applications guaranteeing the each desired throughput;
- converging at run-time to the optimal mapping w.r.t. the minimization of power and thermal impact;
- predicting near-to-optimal execution configuration to minimize the convergence time;
- periodically adjusting the adopted configuration based on feedback loops and responding to performance oscillations and other events (such as application termination).

The policy awakens with a predefined control period and its workflow is organized in four main phases, called *profile*, *decide*, *refine*, and *idle*. No design-time characterization of the executed applications is required and the management relies on the following additional assumptions: to provide best performance applications are executed in isolation, as in [9], and big and LITTLE clusters are considered as separate devices, preventing heterogeneous mappings, as in [8].

Profile. An initial online profiling is executed on every new application entering the system. To avoid perturbations on the already-running workload, a single idle LITTLE core is used to run the application for few control cycles to measure the throughput th_{curr} at frequency f. If no LITTLE core is available, a big one is used. The operation is repeated on the GPU to compute th_{curr_GPU}. Since isolation cannot be achieved when the GPU is already in use, concurrent execution is performed, thus slightly affecting the performance of the running application for few cycles.

Decide. It is performed on any newly profiled application a_i to predict a close-to-optimal execution point based on the estimation models and the *knobs priority order* discussed in Section IV. Algorithm 1 describes the decision process. Initially (Lines 1-11), the performance estimation model is used to identify on each $cluster$ the first operating point guaranteeing performance requirement th_{target} and minimizing power consumption. Configurations are scanned by iterating frequencies f from the current one (not to perturb running applications) to the highest one, and for each f by increasing parallelism w.r.t. the available cores c; as discussed in Section IV, this approach favors parallelization vs. DVFS to minimize power consumption. The process stops at the first compatible configuration, representing the most power efficient one. Similarly, the GPU is analyzed (Lines 13-17), iterating only on the frequency.

Algorithm 1 Decide phase

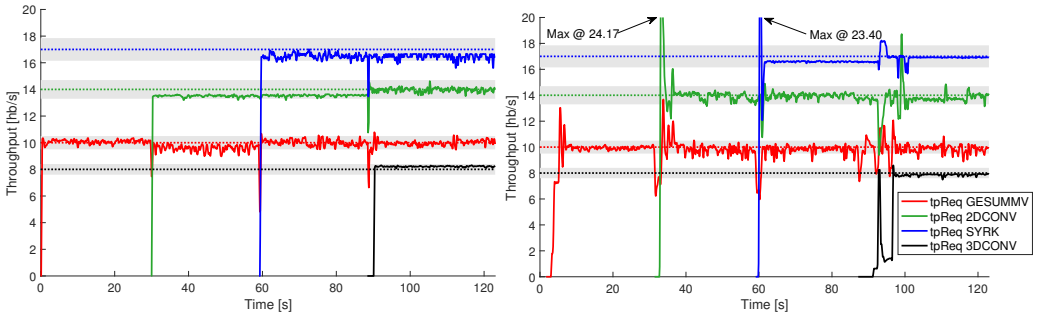
```
1: for all a_i ∈ Apps_profiled do
2:     mappings ← ∅
3:     for all cluster ∈ {big; LITTLE} do
4:         if cluster.#idleCores ≠ 0 then
5:             found ← false
6:             for (f ← cluster.f_curr to cluster.f_max) ∧ !found do
7:                 for (c ← 1 to cluster.#idleCores) ∧ !found do
8:                     th_% ← (a_i.th_curr / a_i.th_target) · r^γ_{b→L} · c^β · (f / cluster.f_curr)
9:                     if th_% ≥ 1 then
10:                        mappings ← mappings ∪ {(cluster, f, c)}
11:                        found ← true
12:    found ← false
13:    for (f ← GPU.f_curr to GPU.f_max) ∧ !found do
14:        th_%_GPU ← (a_i.th_curr_GPU / a_i.th_target) · (f / GPU.f_curr)
15:        if th_%_GPU ≥ 1 then
16:            mappings ← mappings ∪ {(GPU, f, 1)}
17:            found ← true
18:    found ← false
19:    for (m ∈ mappings) ∧ !found do
20:        if m.cluster = LITTLE then
21:            found ← true
22:    if found then
23:        applyMapping(a_i, m)
24:    else
25:        m ← selectMinimumPower(mappings)
26:        if m.cluster = big then
27:            m.big_cores_to_use ←thermalModelCoreSelection(m.c)
28:            applyMapping(a_i, m)
29:        else if m.cluster = GPU then
30:            applyMapping(a_i, m)
31:        else
32:            rejectApp(a_i)
```

Algorithm 2 Refine phase

```
1: for all a_i ∈ Apps_running do
2:     th_ref ← a_i.th_curr / a_i.th_target
3:     if (th_ref < 1 − ε) then
4:         needMoreResources ← needMoreResources ∪ {a_i}
5:     else if (th_ref > 1 + ε) then
6:         needLessResources ← needLessResources ∪ {a_i}
7: for all a_i ∈ needLessResources do
8:     c ← a_i.#cores
9:     min#Cores ← ⌈(c^β · a_i.th_target / a_i.th_curr)^(1/β)⌉
10:    if min#Cores = c then
11:        a_i.quota ← a_i.quota · a_i.th_target / a_i.th_curr
12:    else if a.cluster.#freeCores > 0 then
13:        a_i.cluster.f ← app_i.cluster.f − FREQ_STEP
14:    else
15:        a_i.#cores = a_i.#cores − 1
16:        a_i.big_cores_to_use ←thermalModelCoreSelection(a_i.#cores)
17: for all a_i ∈ needMoreResources do
18:    c ← a_i.#cores
19:    if a_i.quota < c · 100% then
20:        a_i.quota ← a_i.quota · a_i.th_target / a_i.th_curr
21:        if a_i.quota > c · 100% then
22:            a_i.quota ← c · 100%
23:    else if a.cluster.#freeCores > 0 then
24:        a_i.#cores = a_i.#cores + 1
25:        a_i.big_cores_to_use ←thermalModelCoreSelection(a_i.#cores)
26:    else
27:        a_i.cluster.f ← app_i.cluster.f + FREQ_STEP
28:        if a_i.cluster.f > a_i.cluster.f_max then
29:            a_i.cluster.f = a_i.cluster.f_max
30:    applyMapping(a_i)
```

Among the identified configurations (up to three), the LITTLE one is usually preferred, being the most effective w.r.t. temperature (Lines 18-23). If more than one configuration is available, power consumption is estimated to decide between big cores and GPU, and in the former case a subset of big cores is selected based on the thermal model to limit maximum temperature. Finally, if no mapping configuration is identified, the application is killed notifying the user of the lack of resources to achieve the required QoS (Lines 24-32).

Refine. This phase performs iterative adjustments of the decisions taken on the basis of pure estimations, by exploiting a feedback loop from the system. Differently from the *decide*



(a) Design-time strategy (b) Proposed approach

Figure 4. Experimental results: throughput

phase that works only on the subset of newly arrived applications, the *refine* phase is executed on the overall workload. The goal is to reduce estimation errors and improve the performance estimation models, by tailoring them on each running application; in particular, since the effect of parallelization may vary for each application, the corresponding sublinear estimation model has to be refined online. Finally, it is necessary to mitigate the effects of new applications on the already running ones.

Algorithm 2 presents the *refine* workflow. The first step detects applications that are 1) over-performing and 2) under-performing w.r.t. the given requirement; this condition is based on the comparison of the ratio between the current throughput and target counterpart with a specified tolerance threshold ε (Lines 1-6). Then, for each application of the first set, the policy tries to perform a reduction of the provided resources, working on parallelization level, CPU quota and DVFS (Lines 7-16); in particular we can reduce the frequency when there is at least an idle core in the cluster, which shows resource over-provisioning. Released resources can be used for increasing the assignment to under-performing applications (Lines 17-30), following the same priority order. A similar behavior is adopted for applications running on the GPU, being the DVFS the only usable knob to cope with over-provisioned or under-provisioned applications. To perform iterative adjustments, the phase is invoked multiple times in subsequent control periods, until no further actions are possible and applications' requirements are satisfied. Then, it evolves to the *idle* phase.

Idle. When the system configuration is eventually balanced or there is no running application, the manager becomes idle, until a new event occurs, that is when an application enters the system, the manager evolves either to the *profile* or *decide* phase, or when an application terminates or a throughput variation is detected, the manager evolves to the *refine* phase.

VI. Experimental Results

The proposed policy has been implemented and evaluated with the experimental setup discussed in Section III. We compared the approach against a static strategy mimicking the approaches proposed in [9], [10], the most recent related work addressing our considered scenario. The strategy assumes that all applications are known and can be profiled at design-time; it performs a design-time exploration to identify a set of optimal configuration points for each application achieving the required throughput by means of different number of cores and frequency levels (for the sake of fairness kernel partitioning is not considered). At runtime, for each application entering/exiting the system, the Cartesian product of configuration sets of the running applications is explored to identify the optimal solution for the overall workload and CPU quota is dynamically adapted. This strategy minimizes maximum temperature and power consumption.

A workload has been generated by considering `GESUMMV`, `2DCONV`, `SYRK` and `3DCONV` applications from Polybench benchmark suite. These applications present different performance characteristics: `2DCONV` is more suited to run on the GPU cluster with respect to `GESUMMV`, as shown in Figure 2; `SYRK` reaches high throughput easier than `3DCONV` that manifests a slow throughput convergence due to its heavy workload. Applications are annotated with different throughput requirements and arrival times. Finally, a tolerance threshold $\varepsilon = 5\%$ is used for the throughput request.

Experimental results are reported in Figures 4, 5 and 6 showing applications' throughput, power consumption and maximum temperature, respectively. As shown in the first figure, the proposed policy is able to quickly converge after the *profile* phase to a configuration achieving the requested QoS, taking in 6 control cycles, in the worst case when running 4 applications, i.e., $6s$. Then, it continuously refines settings with a fine-grained tuning to limit possible disturbances, fulfilling QoS requirements for almost all the time, as reported in Table I; the percentage of violations is approximately 3.25% worse than the static strategy. The table also reports the average throughput and the related standard deviation for each application, showing almost similar results to the static approach. We may conclude that the proposed solution is a viable replacement of the time-consuming profiling-based static strategy. The policy minimizes power consumption and maximum temperature achieving on average values 0.085 W (5.15%) and 0.83°C (1.47%) worse than the static optimal solution; a more detailed analysis dividing the overall experiment time in periods is reported in Table II. This represent a relevant result when considering that we are not using any design-time information on the workload. Both the percentage of violations and the slightly lower optimal working points show the effectiveness of the defined policy, considering no design-

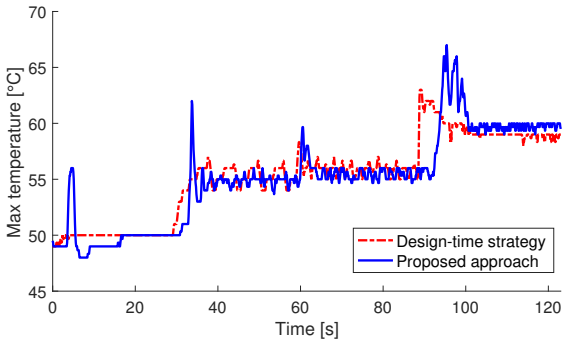


Figure 5. Experimental results: maximum temperature

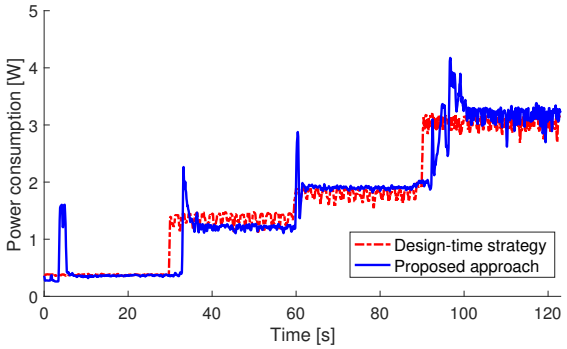


Figure 6. Experimental results: power consumption

time information is used. Experimental results are reported in Figures 4, 5 and 6 reporting applications' throughput, power consumption and maximum temperature, respectively.

A unique negative result is reported for 3DCONV. 3DCONV, quite performance demanding, enters the system when an already heavy workload is running. The first decision taken (at time $95s$) is to map it on the big cluster and causes the application to under-perform. At the same time SYRK is outperforming on the GPU. Therefore, their mapping is dynamically swapped and within 4 cycles they reach the desired throughput without affecting the remaining applications. However, the effect of such overloading is that 3DCONV considerably violates the QoS requirement, and maximum temperature and power are not optimal in the last time interval.

VII. Conclusions

This paper proposes an adaptive policy for the runtime workload distribution and resource tuning for heterogeneous multicores running multiple OpenCL applications. The policy identifies an optimal working point for an unknown multi-programmed OpenCL workload without using any design-time application profiling or analysis. The approach compared against a design-time optimization strategy is effective in converging to a solution guaranteeing the required performance while minimizing power consumption and maximum temperature. Future work will consider kernel partitioning among heterogeneous devices.

Table I
Percentage of time each application satisfied the QoS after initial profiling and corresponding average throughput and standard deviation

Appl.	Req. Throughput (hb/s)	Design-time strategy % satisfy	Design-time strategy Throughput avg./std.dev.	Proposed Approach % satisfy	Proposed Approach Throughput avg./std.dev.
GESUMMV	10	91%	9.86(±0.61)	91%	9.79(±0.55)
2DCONV	14	98%	13.63(±0.77)	92%	13.84(±0.72)
SYRK	17	94%	16.35(±1.33)	96%	16.75(±0.36)
3DCONV	8	95%	7.77(±1.85)	86%	7.16(±1.92)

Table II
Mean difference between design-time and runtime strategy

Time interval	Power consumption	Maximum temperature
0s-30s	0.01 W (2.94%)	0.73°C (1.45%)
31s-61s	0.16 W (11.68%)	0.44°C (0.79%)
62s-92s	0.03 W (1.41%)	0.04°C (0.08%)
93s-123s	0.14 W (4.57%)	2.11°C (3.56%)

References

[1] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era," in *Proc. Design Automation Conf.*, 2013, pp. 174:1–174:9.

[2] E. D. Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini, "Workload-aware power optimization strategy for asymmetric multiprocessors," in *Proc. Design, Automation Test in Europe Conf.*, 2016, pp. 531–534.

[3] F. Gaspar, L. Taniça, P. Tomás, A. Ilic, and L. Sousa, "A Framework for Application-Guided Task Management on Heterogeneous Embedded Systems," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 42:1–42:25, Dec. 2015.

[4] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU power management for 3D mobile games," in *Proc. Design Automation Conference*, 2014, pp. 1–6.

[5] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving Mobile Gaming Performance Through Cooperative CPU-GPU Thermal Management," in *Proc. Design Automation Conf.*, 2016, pp. 47:1–47:6.

[6] A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra, "Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms," in *Proc. Int. Conf. Computer Design*, 2015, pp. 208–215.

[7] E. Paone, F. Robino, G. Palermo, V. Zaccaria, I. Sander, and C. Silvano, "Customization of OpenCL Applications for Efficient Task Mapping Under Heterogeneous Platform Constraints," in *Proc. Conf. Design, Automation & Test in Europe*, 2015, pp. 736–741.

[8] C. Bolchini, S. Cherubin, G. C. Durelli, S. Libutti, A. Miele, and M. D. Santambrogio, "A Runtime Controller for OpenCL Applications on Heterogeneous System Architectures," *SIGBED Rev.*, vol. 15, no. 1, pp. 29–35, Mar. 2018.

[9] A. K. Singh, A. Prakash, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi, "Energy-Efficient Run-Time Mapping and Thread Partitioning of Concurrent OpenCL Applications on CPU-GPU MPSoCs," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 147:1–147:22, Sep. 2017.

[10] E. W. Wachter, G. V. Merrett, B. M. Al-Hashimi, and A. K. Singh, "Reliable Mapping and Partitioning of Performance-constrained openCL Applications on CPU-GPU MPSoCs," in *Proc. Symp. Embedded Systems for Real-Time Multimedia*, 2017, pp. 78–83.

[11] Hardkernel co., "Odroid XU3," http://www.hardkernel.com.

[12] P. Jääskeläinen, C. S. de La Lama, E. Schnetter, K. Raiskila, J. Takala, and H. Berg, "pocl: A Performance-Portable OpenCL Implementation," *Int. Journal of Parallel Programming*, vol. 43, no. 5, pp. 752–785, 2015.

[13] A. Kanduri, A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, and N. Dutt, "Approximation-aware Coordinated Power/Performance Management for Heterogeneous Multi-cores," in *Proc. Design Automation Conf.*, 2018, pp. 68:1–68:6.

[14] H. Rexha, S. Holmbacka, and S. Lafond, "Core Level Utilization for Achieving Energy Efficiency in Heterogeneous Systems," in *Proc. Int. Conf. Parallel, Distributed and Network-based Processing*, 2017, pp. 401–407.