Workload-aware Power Optimization Strategy for Asymmetric Multiprocessors

E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, C. Bolchini
Politecnico di Milano, Italy
Dipartimento di Elettronica Informazione e Bioingegneria
{emanuele.delsozzo, gianlucacarlo.durelli, antonio.miele, cristiana.bolchini, marco.santambrogio}@polimi.it
ettore.trainiti@mail.polimi.it

***Abstract*—Asymmetric multi-core architectures, such as the ARM big.LITTLE, are emerging as successful solutions for the embedded and mobile markets due to their capabilities to trade-off performance and power consumption. However, both the Heterogeneous Multi-Processing (HMP) scheduler integrated in the commercial products and the previous research approaches are not able to fully exploit such potentiality. We propose a new runtime resource management policy for the big.LITTLE architecture integrated in Linux aimed at optimizing the power consumption while fulfilling performance requirements specified for the running applications. Experimental results show an improvement of the 11% on the performance and at the same time 8% in peak power consumption w.r.t. the current Linux HMP solution.**

I. Introduction

Because of the reaching of the power wall, the trend in microprocessor design has shifted towards parallel architectures, such as multi/many-cores and heterogeneous solutions to boost performance. However, even if a large number of cores enables a better satisfaction of applications' performance demand, power consumption is becoming a relevant issue, especially for mobile systems having a limited battery life.

In the mobile and embedded appliances' scenario, an emerging heterogeneous solution is the asymmetric multi-core architecture; it is composed of multiple units sharing the same Instruction Set Architecture (ISA) but having different micro-architectural implementations, thus offering divers power/performance characteristics. The ARM big.LITTLE [1] is one of such heterogeneous architectures, where a high-performance power-hungry A15 multi-core (called *big*) is coupled with a slower low-power A7 multi-core cluster (called *LITTLE*). As an example, several embedded devices, such as the Samsung Galaxy S5 mobile phone [2] or the Odroid XU3 board [3], are powered by the Samsung Exynos chip [2] implementing a big.LITTLE architecture.

The state-of-the-art solution for the runtime resource management of the big.LITTLE is the Heterogeneous Multi-Processing (HMP) scheduler [4], a patch to the standard scheduler in the Linux kernel which dynamically dispatches threads to big or LITTLE cluster according to their characteristics. However, HMP presents some limitations (as discussed in Section II) and specifically with compute intensive, data parallel applications it is unable to tune the system in order to achieve an energy efficient solution.

Given these motivations, we propose a new runtime resource management policy, enhancing the Linux scheduler, running on ARM big.LITTLE architecture and targeting parallel applications (the source code is available at [5]). This policy exploits Dynamic Voltage and Frequency Scaling (DVFS) and dynamic thread management, to:

- optimize power consumption while guaranteeing performance requirements of the running applications;
- balance the application threads execution among asymmetric cores;
- adapt to varying workloads by leveraging on-line performance measurements;
- improve the state-of-the-art HMP scheduler in both performance and power management.

Experimental results on the Odroid XU3 [3] show an improvement of the 11% on the performance and at the same time 8% in peak power consumption w.r.t. the current Linux HMP solution, resulting in turn in a reduction of 5% of the maximum temperature and of 11% of the board energy consumption.

The paper is organized as follows. Section II discusses the past approaches to highlight their limitations. Section III defines the working context and problem formulation. In Section IV, we introduce the proposed runtime management policy. The experimental validation of the approach is presented in Section V, while Section VI closes the paper.

II. Related Work

Several works in literature investigated run-time resource management in asymmetric multi-cores, in particular focusing on power management ([6, 7]). One of the most recent works [8] demonstrated how, in the case of asymmetric processors, the standard *race-to-idle* solution is not beneficial in terms of neither power dissipation nor energy consumption. As a consequence, in our context characterized by applications expressing a throughput goal running on an asymmetric system, a more advanced run-time resource management is paramount.

One of the major issues of asymmetric multicore architectures consists in finding the right core (or mix of cores) where to allocate the applications threads and how to set their frequencies through DVFS settings. State-of-the-art approaches tackled the problem by partitioning the available cores either

This work was partially funded by the European Commission in the context of the FP7 SAVE project (#610996-SAVE).

by virtually coupling one big and one LITTLE core and then mapping processes to these virtual units (*core-mapping*) [9, 10], or by dividing the cores into a big cluster and a LITTLE one (*cluster-migration*) [11]. However both the approaches have the drawback that they do not exploit all the available cores on the system, which is of utmost importance to obtain better performance in the case of parallel applications. On the other hand, many works focused on the run-time resource management to fulfill application goals while concurrently optimizing energy and power consumption. Among them, the works proposed in [12, 11, 13] develop heuristics and control-theory based approached to obtain energy saving up to 49% w.r.t. the standard system behavior.

HMP is the last version of the scheduler for ARM big.LITTLE proposed by Samsung [4], that is able to exploit big and LITTLE cores at the same time to improve performance and energy efficiency; however, HMP's dispatching is not optimal when executing parallel applications, and moreover the DVFS controller is not yet coupled with the resource assignment process thus leading to sub-optimal power management. Our proposal builds on HMP and has the goal of optimizing the power consumption of the device in a context where applications running on the system have a specified throughput requirement. With respect to all existing approaches, our solution enables the concurrent usage of the big and LITTLE cores for executing applications and directly manages the DVFS actuation to improve energy efficiency.

III. Problem Definition

This section details the context of the work presenting the model for both the supported applications and the target system, and introduces the formulation of the considered resource management problem.

A. Application

The approach we propose is designed for parallel applications with computational-intensive loops, where the user can specify some minimum throughput requirement or goal[1]. Possible examples are audio/video applications, typically run on embedded platforms, which must satisfy a minimum throughput in terms of frame rate to guarantee a good user experience. The reference application model is a cyclic task graph where the kernel node, the computationally intensive part of the code, might be potentially parallelized with the fork-join paradigm, by using state-of-the-art libraries, such as OpenMP [14]. Usually, the considered platforms execute a dynamic and unpredictable workload.

B. System specification

The proposed resource management policy is tailored for the ARM big.LITTLE architecture [1], and, in particular, we considered the Samsung Exynos 5422 chip hosted on the Odroid XU3 board [3], composed of a *big* cluster (suited for high performance), and a *LITTLE* (suited for low power) (Table I). Moreover, the system loads a Linux Operating System (OS) featuring the HMP scheduler and providing all drivers to control DVFS, which can be used to change runtime frequencies with a *per-cluster* granularity.

[1]In the text *requirement* and *goal* are used indiscriminately.

TABLE I. Exynos 5422 specification.

	big	LITTLE
Processor type	Cortex A15	Cortex A7
Cores in cluster	4	4
Minimum frequency	800MHz	800MHz
Maximum frequency	1900MHz	1300MHz

TABLE II. System characterization.

	big	LITTLE
Number in cluster	N_b	N_L
Utilized cores	$n_b \leq N_b$	$n_L \leq N_L$
Available frequencies	F_b	F_L
Running frequency	$f_b \in F_b$	$f_L \in F_L$

We model the system using the variables presented Table II. The combination of such variables determines the configuration c of the system in that instant of time. The entire configurations space C is:

$$C = N_b \times F_b \times N_L \times F_L + N_b \times F_b + N_L \times F_L$$

with cardinality; for our platform, $|C| = 1224$.

C. Problem formulation

The problem we address is the reduction of the power consumption of the considered asymmetric architecture while fulfilling the throughput requirements of each application. The total power consumption of the system is a function of the units currently used in the system in relation to a given configuration $P(c)$, and can be computed as

$$\begin{aligned} P(c) = &(P_{B\ idle}(N_b - n_b, f_b) + P_{B\ utilized}(n_b, f_b)) \\ &+(P_{L\ idle}(N_L - n_L, f_L) + P_{L\ utilized}(n_L, f_L)) \end{aligned}$$

where $P_{\{B|L\}\ idle}$ and $P_{\{B|L\}\ utilized}$ are two functions related to the type of processor, the number of utilized processors n_b and n_L, and their frequency f_b and f_L.

The workload consists of a set of running applications A, each one specifying a given goal (in terms of throughput) G_a. In order for the resource management policy to meet the overall performance requirement, each application must reach a satisfactory throughput T_a such that $T_a > G_a$. The objective of the resource management policy is to find the configuration $\hat{c}$ with the minimum power consumption among all the configurations $\bar{C} \in C$ that are able to satisfy the goals of the applications, that is:

$$\begin{aligned} \hat{c} = &< n_b, f_b, n_L, f_L > \text{ such that} \\ &\hat{c} \in \bar{C} = \{c | T_a > G_a, \forall a \in A\} \text{ and} \\ &P(\hat{c}) = \min P(\bar{c}), \forall \bar{c} \in \bar{C} \end{aligned}$$

IV. The proposed run-time resource management policy

The resource management policy we propose periodically runs as a stand-alone process on top the OS and interacts with the OS and the applications to gather information on the current status of the system (in terms of power consumption and applications' throughput) and actuates on the available knobs (application mapping, number of application's threads, per-cluster DVFS) according to the decision taken. The policy activity can be divided in a set of steps discussed in the following subsections: 1) monitoring of the applications' throughput; 2) prediction of the performance achievable by using a given configuration; 3) fast exploration of the configuration space to find the configuration to enforce; 4) enforcing the selected configuration.

A. Throughput Monitoring

The proposed policy exploits a state-of-the-art solution to acquire high-level information from the application, called Heartbeat [15]. The Heartbeat Application Programming Interface (API) requires the application source code to be minimally instrumented with three specific calls, in order to: 1) register a new starting application (and the possibly-related minimum throughput) to the policy; 2) update the current heartbeat, at the end of each loop iteration; 3) unregister the application that is going to finish. All this information is asynchronously collected by the policy in a processes' table (our Heartbeat is implemented by using a Posix shared memory) and will be exploited in the subsequent decision phases.

B. Performance Estimation

The run-time performance monitoring allows the policy to determine whether the goals are met or not, but does not provide any clue regarding which new configuration has to be set. Therefore, a fast method to estimate how the performance of the system will benefit from the enforcement of a given configuration has been defined. The method relies on the following general assumptions: 1) application performance benefits from a fairly balanced distribution among used cores; 2) application performance generally improves with the number of cores on which computations are parallelized; 3) given two different processor types, it is possible to measure the performance gain when switching from one to the other.

Given these considerations, we can characterize each configuration c in terms of a speedup w.r.t. a selected base configuration, namely $speedup(c)$. Let us assume we know the performance gain K when using a single A15 core at a certain frequency f_{REF_B} with respect to using a single A7 core at frequency f_{REF_L}. By assuming linear scaling and no overhead due to concurrent applications' execution, we can compute the expected speedup of a given configuration with respect to the A7 core (adopted as the base configuration) as:

$$speedup(c) = K \cdot n_b \cdot \frac{f_b}{f_{REF_B}} + n_L \cdot \frac{f_L}{f_{REF_L}}$$

The speedup value is application-dependent, and it is also affected by the number of running applications. Therefore, the policy performs a profiling phase when there is a variation in the currently executed workload. This task is implemented by setting the configuration with only one A7 core, and, then, by setting the one with only one A15 core at known frequencies; K value is computed as the ratio between the collected throughput values of both configurations.

C. Configurations Space Exploration

After performing the profiling phase, the policy is able to associate with each of the configurations an estimation of the power and of the performance speedup with respect to a known reference performance. Based on these profiles, we can organize all the possible configurations in a table and order it with respect to the speedup. The goal of the configurations space exploration is to find within all the possible configurations in the table the one that minimize the power consumption while respecting all the applications' requirements. A sort of binary search is adopted, using two indexes, c_{Low} and c_{High}. c_{Low} points to the first configuration that does not satisfy the performance requirement, while c_{High} is the first configuration found to satisfy the performance goal, which might provide more throughput than strictly required. In between these values, there are configurations allowing to fulfill the performance requirement, and the goal of the exploration phase is to identify a configuration $\hat{c}$ in the $(c_{Low}, c_{High}]$ range, providing the throughput strictly necessary to satisfy the requirement. Finally from the valid configurations (i.e. the one above $\hat{c}$) the policy selects the one with the lowest power consumption.

Note that each time there is a change in the applications' mix, c_{Low}, c_{High}, and the expected speedups are reset to the initial values. Furthermore, since performance predictions are only estimations, at run-time the system can observe the actual speedup, and correct the configuration table accordingly. The policy generally converges with a number of trials that is logarithmic in the number of configurations, being inspired on a binary search. However, should all the performance predictions fail, the process tries all possible configurations.

D. Configuration Enforcement

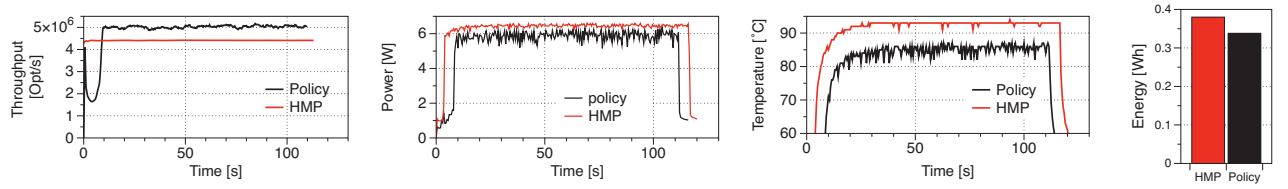
The selected configuration is selected by setting 1) the cluster voltage/frequency level, and 2) the necessary number of parallel threads for each application. The cluster voltage/frequency levels are easily set by means of the *cpufreq*, a Linux utility. Instead the setting of the number of threads is more problematic since the HMP scheduler does not always balance in a fair way threads among heterogeneous cores. Therefore, since we cannot rely on HMP balancing features in heterogeneous configurations, we need to identify the number of threads to be used and their mapping and set them by means of *schedsetaffinity* utility. In more details, we spawn a number of threads which is function of the configuration c used and the gain K. We approximate the gain K measured at run-time with $\frac{Thread_b}{Thread_L}$, which represents the number of threads to be executed on each big and LITTLE core so that the computation terminates about at the same time on all the cores.

V. Experimental Results

This section illustrates the results obtained by the devised policy on the Odroid XU3 [3]. In our tests we used a pricing option application based on the Black & Scholes formula (BS) parallelized by means of OpenMP `parallel for` pragma; if not differently forced (as we do with our solution), this pragma parallelizes on all the available cores. The source code of the policy, implemented in C++ as a userspace library, is available at [5].

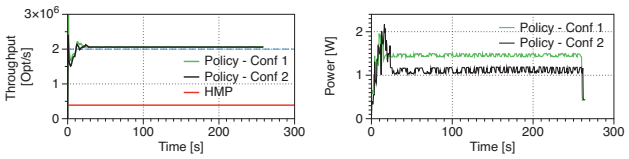
A. Comparison against HMP

In our first experiment we aimed at comparing our solution (without DVFS) against HMP when an application is executed without performance goals, i.e. maximizing its throughput. Figure 1 illustrates the results obtained using our policy and HMP in a run of BS. Our policy obtained a $\sim 11\%$ speedup thanks to a better threads distribution and workload balance across the asymmetric cores, which corresponds also to a $\sim 8\%$ reduction in peak power thanks to a better usage of the A7 cores. These results have a relevant impact on the internal temperatures, which greatly impacts on the device lifetime and on the cooling costs. In fact, the max temperature registered



(a) Throughput (b) Power (c) Temperature (d) Board Energy

Fig. 1. Comparison of our approach against HMP while maximizing performance.



(a) Throughput (b) Power

Fig. 2. Behavior of the proposed solution controlling a single application.

in the experiment is $\sim 7°C$ lower for our policy compared to HMP.As a consequence the total energy consumed by the board (included the cooling fan) measured using an external power monitor is reduced by $\sim 11\%$ (Figure 1(d)).

B. Power Optimization

Figure 2 reports the behavior of the policy when controlling an instance of BS executing with a goal of 2 MOpts/s. We performed two tests one converging to the configuration that satisfies the throughput (*Conf 1*) and another one with the policy optimizing also the power (*Conf 2*). HMP is set to execute with the configuration *Conf 2*. The throughput plot shows that *Conf 1* converges to a configuration that almost exactly satisfies the desired goal, and that *Conf 2* still has the same throughput[2]; both of them converges to the solution in $\sim 30s$. The power plot illustrates that the search for the less power hungry configuration in this case lead to a $\sim 36\%$ power reduction, with respect to focusing only on guaranteeing the performance requirement. Finally, the experiment shows that HMP is not able to balance the threads on the resources used causing a $5\times$ performance reduction. We also measured that our policy (using a period of $500ms$) introduces a 3.5% overhead in the system. This overhead has been measured by comparing the performance of BS when executed with and without our controller.

VI. Conclusion

In this work we presented a workload-aware policy for a better exploitation of heterogeneous resources. Based on the use of available techniques and mechanisms, the proposed run-time resource management approach can achieve relevant power efficiency improvements, without compromising performance. The policy is workload-aware, so that run-time adjustments can be carried out to satisfy user's requirements while optimizing power consumption. Future work is aimed at extending the proposed solution to a more general framework with respect to platforms and application environments.

References

[1] ARM, "big.LITTLE Technology." [Online]. Available: http://www.arm.com/

[2] Samsung, "Samsung website," http://www.samsung.com.

[3] Hardkernel co., "Odroid XU3," www.hardkernel.com/.

[4] K. Yu, D. Han, C. Youn, S. Hwang, and J. Lee, "Power-aware task scheduling for big.LITTLE mobile processor," in *Proc. Int. SoC Design Conf.*, 2013, pp. 208–212.

[5] "Source code of the work available at:," https://bitbucket.org/necst/save-orchestrator-release.

[6] T. Somu 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.

[7] S. Yoo, "An empirical validation of power-performance scaling: DVFS vs. multi-core scaling in big.LITTLE processor," *IEICE Electronics Express*, vol. 12, no. 8, pp. 1–9, 2015.

[8] C. Imes and H. Hoffmann, "Minimizing Energy Under Performance Constraints on Embedded Platforms: Resource Allocation Heuristics for Homogeneous and single-ISA Heterogeneous Multi-cores," *SIGBED Rev.*, vol. 11, no. 4, pp. 49–54, Jan. 2015.

[9] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite Cores: Pushing Heterogeneity Into a Core," in *Proc. Int. Symp. on Microarchitecture*, 2012, pp. 317–328.

[10] M. Kim, K. Kim, J. Geraci, and S. Hong, "Utilization-aware load balancing for the energy efficient operation of the big.LITTLE processor," in *Proc. Design, Automation and Test in Europe Conf.*, 2014, pp. 1–4.

[11] T. 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. 1–9.

[12] F. Gaspar, A. Ilic, P. Tomas, and L. Sousa, "Performance-Aware Task Management and Frequency Scaling in Embedded Systems," in *Int. Symp. on Computer Architecture and High Performance Computing*, 2014, pp. 65–72.

[13] S. Holmback, S. Lafond, and J. Lilius, "Performance Monitor Based Power Management for big.LITTLE Platforms," in *Proc. HIPEAC Workshop on Energy Efficiency with Heterogeneous Computing*, 2015, p. 16.

[14] OpenMP Architecture Review Board, "OpenMP application program interface version 4.0," 2013. [Online]. Available: http://www.openmp.org/

[15] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats for software performance and health," *ACM Sigplan Notices*, vol. 45, no. 5, pp. 347–348, 2010.

[2]The first configuration is $\langle 3, 900, 2, 1200\rangle$ while the second is $\langle 1, 900, 4, 1200\rangle$.