Adaptation in Heterogeneous Mobile SoCs

Amit Kumar Singh
University of Essex
United Kingdom
a.k.singh@essex.ac.uk

Geoff V. Merrett
University of Southampton
United Kingdom
gvm@ecs.soton.ac.uk

Bashir M. Al-Hashimi
University of Southampton
United Kingdom
bmah@ecs.soton.ac.uk

Motivation for Heterogeneous Mobile SoCs

The reliance of embedded computing systems is increasing to heterogeneous mobile SoCs consisting of different types of cores such as CPU and GPU. An example includes the Samsung Exynos 5422 SoC containing a CPU with four ARM cortex-A15 (big) and four ARM Cortex-A7 (LITTLE) cores, and a GPU with 6 ARM Mali-T628 cores [1]. These architectures provide opportunities to exploit distinct features of CPU and GPU cores to meet performance and energy consumption requirements. Further, the cores in these SoCs support dynamic voltage and frequency scaling (DVFS), which can be used to reduce dynamic power consumption ($P \propto V^2 f$). This helps to reduce energy consumption if the extra time taken to run the workload at a lower voltage and frequency can be accounted by sufficient reduction in the power consumption.

Open Computing Language (OpenCL) [3] provides an opportunity to write applications that can execute across heterogeneous cores including CPUs and GPUs [6, 8–10]. However, since CPU and GPU cores typically handle task/thread and data level parallelism, respectively, the performance and energy consumption of an application varies when executed onto only CPU, only GPU, or both CPU and GPU cores [17]. The variation depends upon the kind of parallelism dominating the application. Existing studies have shown significant reduction in execution time and energy consumption when an application simultaneously exploits both the CPU and GPU cores with an appropriate partitioning of threads between them [6].

Related Research and Their Shortcomings

For concurrently executing applications in a heterogeneous mobile SoC, e.g., image and MP3 decoding with respective frames per second (fps) requirements, energy-efficient run-time management of applications on SoC resources is desired. Several efforts have been made towards this [4, 7, 11, 13, 15]. However, they consider cores having the same instruction set architecture (ISA), e.g., big and/or LITTLE cores. Usually, they employ Pthreads, which cannot be used to exploit cores of different ISAs such as CPU and GPU. As mentioned earlier, the OpenCL programming model overcomes such limitations, but most of the existing works for embedded systems utilize either the CPU or GPU [12, 14], and there are limited efforts to exploit both at the same time [5, 6]. In [6], only a single application at a time is considered. Concurrent applications are considered in [5], but the mapping (defined as the number of used cores, their types (e.g., big, LITTLE) and operating frequencies) and thread-partitioning (defined as the fraction of threads to be executed on CPU cores) for each application remain fixed, which is identified before starting applications' execution. This indicates no adaptation of the mapping and thread-partitioning during execution. Therefore, due to the lack of adaptation, the freed cores by an application cannot be used by already running applications or to start execution of a waiting application. This could help in improving resource utilisation and/or energy efficiency. Further, for CPU-GPU SoC, the adaptation needs to consider collaboration between CPU and GPU cores processing capabilities for energy efficiency, which is missing. An adaptive approach for CPU-GPU SoC is recently presented [18], but it considers only a single application at a time and only CPU or GPU for application execution.

ArmSummit-2018, 2018,
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Motivational Case Study: Adaptation in CPU-GPU Mobile SoC

Fig. 1 illustrates the mapping and thread-partitioning process of applications SYR2K and SYRK from Polybench benchmark [16] on the Samsung Exynos 5422 SoC when employing **non-adaptive** (e.g., [5, 6]) and our proposed **adaptive** approaches. At the beginning, SYR2K is started on big (B) and GPU (G) cores and SYRK on LITTLE (L) cores. Since embedded GPU drivers do not support spatially isolated and time multiplexed execution of multiple applications, SYR2K uses all the GPU cores till its completion. Thereafter, SYRK starts using the GPU cores. Both the applications are started with appropriate thread-partitioning between CPU and GPU cores by taking CPU and GPU processing capabilities into account, i.e., by having proper collaboration between CPU and GPU. It can be seen in Fig. 1 that the **non-adaptive** approach leaves cores unused when SYR2K completes execution. This has resulted in a higher execution time of **50 seconds** for SYRK and a total energy consumption of **85 joule (J)** for both the applications. In contrast, upon completion of SYR2K, the **adaptive** approach performs adaptation (through collaboration between CPU and GPU cores), i.e. remapping SYRK by taking the freed cores into account, and execution is started with a new thread-partitioning that enqueues more threads to CPU cores as it has increased processing capability. This leads to early completion of SYRK at **42 seconds**. As a result of reduced execution time, total energy consumption is reduced to **68 J**.

Adaptation is also beneficial to support a new application when no resource is available by adapting running applications to lower number of cores. The running application that is over performing can be chosen for this purpose. and the freed resources can be used for the new application. This has resulted in timely start of the new application and early completion as well, which results in lower energy consumption.

In summary, adaptation (remapping and repartitioning) at application arrival and completion brings several benefits such as the exploitation of freed cores, addition of a new application when no core is available and early completion of applications by exploiting freed cores, which also leads to low energy consumption (Power × time).

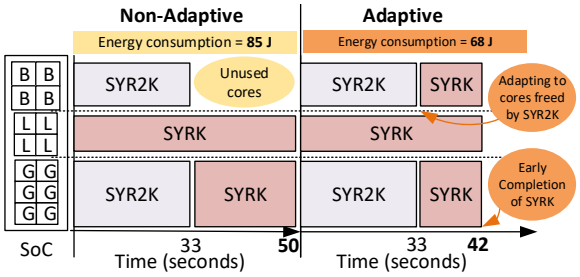

Figure 1: Adaptation at application completion: Non-adaptive vs. adaptive mapping and thread-partitioning on big (B), LITTLE (L) and GPU (G) cores of Samsung Exynos 5422 mobile SoC.

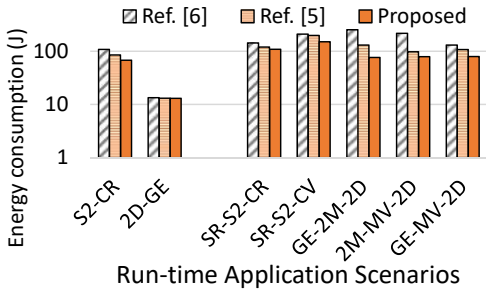

Figure 2: Energy consumption by various approaches for different run-time scenarios: two and three concurrent applications.

Challenges

To perform energy-efficient collaborative adaptation (remapping and thread-repartitioning) of executing applications while satisfying their performance requirements, the following key challenges need to be addressed:

- Since OpenCL enqueues all application threads between CPU and GPU before starting execution, with no track of executed threads over time, online thread-partitioning is not possible. To enable this, threads can be grouped into several equal size chunks, and enqueuing and partitioning done at chunk level. However, it imposes challenges to identify the best number of chunks (each chunk contains the same number of threads) so that the cores are neither starved nor overfed from threads.
- Upon an application completion, identifying the energy saving benefits of adaptation (to higher number of cores) for each executing application by considering timing and energy overheads.
- Upon an application arrival, energy-efficient and performance satisfying adaptation (to lower number of cores) of executing applications to facilitate performance satisfying execution of the arrived application.
- Collaboration between CPU and GPU is required by identifying their processing capabilities at various adaptation points such that balanced execution can be performed.

Contributions

To address the above-mentioned challenges, we have made the following contributions:

(1) An collaborative adaptation approach that performs an energy-efficient mapping and thread-partitioning between CPU and GPU cores for concurrent applications, and applies energy optimizing adaptations (remapping and thread-repartitioning) at application completion and arrival through CPU and GPU collaboration.

(2) To reduce adaptation overhead, for each application, energy-efficient mapping and thread-partitioning when using a different number of CPU core are explored at design-time and stored for using at run-time. Additionally, the best number of chunks for each application is also explored at design-time and execution starts by feeding chunks to CPU and GPU continuously by collaborating on their processing capabilities.

(3) Implementation and experimental validity of the proposed approach on an Odroid-XU3 platform [2] (contains a state-of-the-art mobile SoC prevalent in tablet/smart-phone devices).

The talk will cover details of these contributions and obtained results. A sample result is shown in Figure 2, which indicates that the proposed adaptive approach achieves reduction in energy consumption when compared to closely related approaches in [6] and [5]. Additionally, performance improvements are achieved. To the best of our knowledge, this is the first study on energy-efficient collaborative adaptation for concurrently executing performance constrained applications on CPU and GPU cores of a mobile SoC.

REFERENCES

[1] 2016. Exynos 5 Octa (5422). www.samsung.com/exynos/. (2016).
[2] 2016. Odroid-XU3. http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127. (2016).
[3] 2016. The open standard for parallel programming of heterogeneous systems. https://goo.gl/A9wXRJ. (2016).
[4] A. Aalsaud et al. 2016. Power–Aware Performance Adaptation of Concurrent Applications in Heterogeneous Many-Core Systems. In *ISLPED'16*.
[5] A. K. Singh et al. 2017. Energy-Efficient Run-Time Mapping and Thread Partitioning of Concurrent OpenCL Applications on CPU-GPU MPSoCs. *TECS'17* (2017).
[6] A. Prakash et al. 2015. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *ICCD'15*.
[7] B. Donyanavard et al. 2016. SPARTA: runtime task allocation for energy efficient heterogeneous many-cores. In *CODES+ISSS'16*.
[8] C. K. Luk et al. 2009. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *MICRO'09*.
[9] D. Grewe et al. 2011. A static task partitioning approach for heterogeneous systems using OpenCL. In *CC'11*.
[10] D. Grewe et al. 2013. OpenCL task partitioning in the presence of GPU contention. In *LCPC'13*.
[11] E. D. Sozzo et al. 2016. Workload-aware power optimization strategy for asymmetric multiprocessors. In *DATE'16*.
[12] I. Grasso et al. 2014. Energy efficient hpc on embedded socs: Optimization techniques for mali gpu. In *IPDPS'14*.
[13] J. Ma et al. 2016. An Analytical Framework for Estimating Scale-Out and Scale-Up Power Efficiency of Heterogeneous Manycores. *TC* (2016).
[14] K. Chandramohan et al. 2014. Partitioning data-parallel programs for heterogeneous MPSoCs: time and energy design space exploration. In *LCTES'14*.
[15] K. V. Craeynest et al. 2012. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *ISCA'12*.
[16] S. Grauer-Gray et al. 2012. Auto-tuning a high-level language targeted to GPU codes. In *InPar'12*.
[17] Y. Wen et al. 2014. Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms. In *HiPC'14*.
[18] Ben Taylor et al. 2017. Adaptive optimization for OpenCL programs on embedded heterogeneous systems. In *LCTES'17*.