The Case for Exploiting Underutilized Resources in Heterogeneous Mobile Architectures

Chen-Ying Hsieh, Ardalan Amiri Sani, and Nikil Dutt
Department of Computer Science, University of California, Irvine, USA
email: {chenyinh, ardalan, dutt}@uci.edu

***Abstract*—Heterogeneous architectures are ubiquitous in mobile platforms, with mobile SoCs typically integrating multiple processors along with accelerators such as GPUs (for data-parallel kernels) and DSPs (for signal processing kernels). This strict partitioning of application execution on heterogeneous compute resources often results in underutilization of resources such as DSPs. We present a case study executing a mix of popular data-parallel workloads such as convolutional neural networks (CNNs), computer vision filters and graphics rendering kernels on mobile devices, and show that both performance and energy consumption of mobile platforms can be improved by synergistically deploying these underutilized compute resources. Our experiments on a mobile Snapdragon 835 platform under both single and multiple application scenarios executing the aforementioned workloads demonstrates average performance and energy improvements of 15-46% and 18-80%, respectively, by synergistically deploying all available compute resources, especially the underutilized DSP.**

***Index Terms*—heterogeneous architectures, mobile systems, performance and energy, embedded software**

I. Introduction

Modern mobile platforms are being increasingly used for a variety of applications with differing computational demands, such as video, gaming, virtual/augmented reality, and computer vision applications. Consequently mobile platforms deploy Multiprocessor SoCs (MPSoCs) that integrate multiple heterogeneous CPUs together with accelerators such as GPUs, DSPs, neural processing units (NPUs) and other hardware IP blocks into a single chip (Table I) to meet the performance and quality requirements of emerging applications. These accelerators are shipped with software toolchains allowing application developers to exploit domain-specific acceleration of task kernels. For instance, contemporary mobile GPUs are usually shipped with support for applications written in OpenCL and CUDA, and excel in accelerating data-parallel kernels typically found in computer vision and gaming applications. Similarly, mobile DSPs come with SDK toolchains supporting application development (e.g., OpenCL for TI's Tesla DSP and C++ Qualcomm's Hexagon DSP SDK) and acceleration for signal processing applications.

Vendor	SoC	CPU	GPU	Other IPs
Qualcomm	Snapdragon	HMP	Adreno	Hexagon DSP
TI	OMAP	HMP	PowerVR	Tesla DSP
NVIDIA	Tegra	HMP	NVIDIA	-
Samsung	Exynos	HMP	Mali	Neural Processor
Apple	A series	HMP	Apple	Neural Processor

TABLE I: Contemporary Mobile SoCs

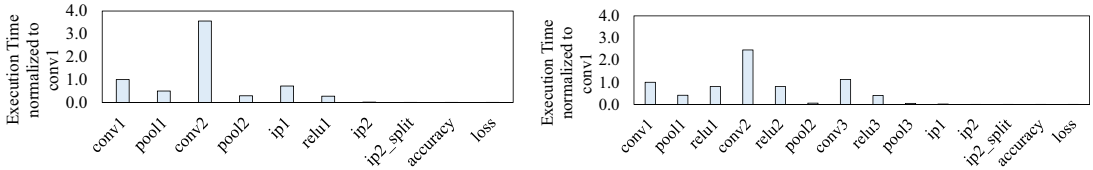
To improve performance, a developer typically partitions an application into task kernels to be executed on compute units and accelerators (e.g., CPU, GPU, DSP) that correspondingly promise a boost in performance. For instance, a convolutional neural network (CNN) application with multiple layers can be partitioned into data-parallel tasks for each layer and mapped onto GPUs for boosting performance. Intuitively, this strict partitioning of tasks to execute them on the highest-performing compute units should result in overall better performance. However, mobile platforms often face resource contention when executing multiple applications, saturating these high-performing compute units. In such scenarios – contrary to intuition – offloading of computational pressure to other underutilized and seemingly under-performing compute units (e.g., DSPs) can actually result in overall improvements in performance and energy. Indeed, in our experimental case study, we observed an average improvement of 15-46% in performance and 18-80% in energy when executing multiple CNNs, computer vision and graphics applications on a mobile Snapdragon 835 platform by utilizing idle resources such as DSPs and considering all available resources holistically.

To estimate the underutilization of accelerators, we surveyed 175 top-ranked Android applications from popular categories such as games, video/audio players, social media and photography. Our study of these applications show that approximately 15% of them use the GPU, while only one utilizes the DSP. These low utilization numbers motivate the opportunity to achieve better performance and energy for mobile applications by holistically exploiting underutilized resources.

The main contribution of our paper is a case study of executing multiple applications (e.g., CNN, computer vision and graphics) on a mobile Snapdragon 835 platform, and demonstrating the ability to exploit underutilized heterogeneous compute resources for concomitant improvements in performance and energy. In particular:

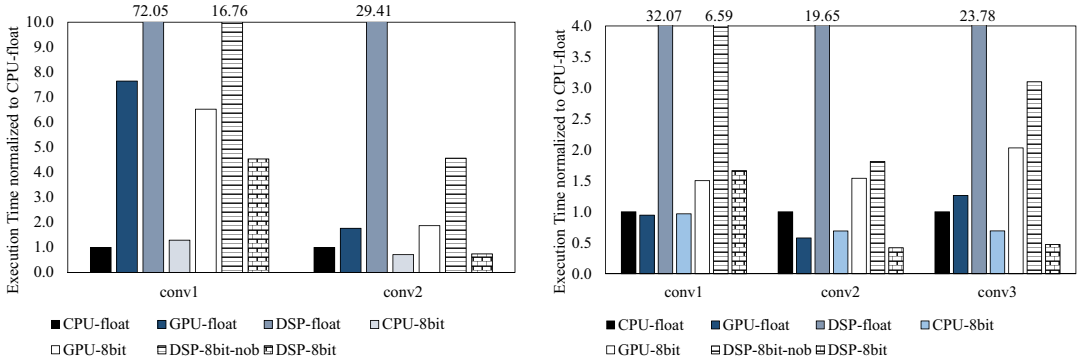
1) We demonstrate the ability to exploit underutilized mobile platform resources such as DSPs for holistic performance and energy improvement
2) For single, multiple, and mixed-workload applications, we observed between 15-45% improvements in performance, and between 18-80% improvements in energy over the conventional mapping strategy that assigns tasks to the highest-performing compute units (GPUs).

978-3-9819263-2-3/DATE19/©2019 EDAA



(a) MNIST (b) CIFAR10

Fig. 1: Performance breakdown of CNN



(a) MNIST (b) CIFAR10

Fig. 2: Performance of Convolutional Layers

II. Experimental Case Study

A. Experimental Setup

Experiment	Description
CPU-float, CPU-8bit	Run the original or quantized version on the CPU
GPU-float, GPU-8bit	Run the original or quantized version on the GPU
DSP-float	Run the original version on the DSP
DSP-8bit	Run the quantized version on the DSP w/ batch processing
DSP-8bit-nob	DSP-8bit w/o batch processing
Hetero	Layers or stages are statically configured to run on highest-performing compute unit
Hetero-noGPU	Like Hetero but avoid using GPU

TABLE II: Keywords used in Experiments

Platform: We use a Snapdragon 835 development board with the Android 6 operating system (which uses the Linux 4.4.63 kernel). The board's SoC integrates custom CPUs with big-LITTLE configurations that conform to ARM's ISA. It also integrates a GPU with unified shaders, all capable of running compute and graphics workloads. The 835 board has two Hexagon DSPs: a cellular modem DSP dedicated to signal processing, and a compute DSP for audio, sensor, and general purpose processing. We target exploiting the compute DSP since it is typically idle. We use the Trepn profiler [1] to measure the power and energy consumption.

Applications: For the CNN applications, we select two Caffe CNNs: *lenet-5* and *cuda-convnet* using datasets MNIST and CIFAR10, respectively. MNIST represents a lightweight network with a few layers and low memory footprint whereas CIFAR10 has more layers and high memory footprint. We also implemented a quantized version of Caffe, which supports quantized matrix multiplication using 8-bit fixed-point for convolutional and fully-connected layers. The other layers still perform floating-point computation. The experiments include floating-point and fixed-point versions of CNN models running on CPU, GPU and DSP. For the CED application, we modified Chai CED [2] to support all heterogeneous compute resources for each stage.

Table II summarizes the different experiments by executing the above applications on various compute units (CPU, GPU, DSP, and heterogeneous – including all compute units). In addition to the original floating-point version of CNNs, we also deploy 8-bit quantized versions to exploit the DSP effectively. The row *DSP-8-bit* represents a single function call for batch processing of 100 images to amortize the communication overhead, whereas the row *DSP-8bit-nob* represents no batch processing, i.e., separate function calls for each image.

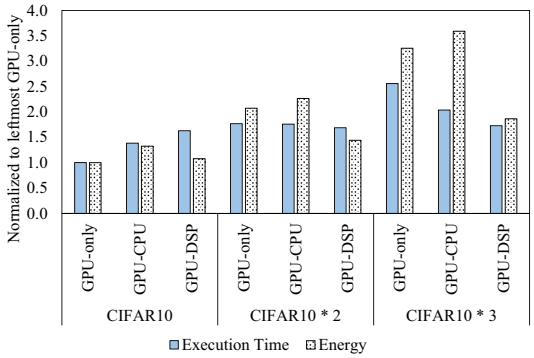


Fig. 3: Performance of executing multiple CIFAR10 instances on different compute units

B. Opportunities for Exploiting Underutilized Resources

Figure 2 presents the performance of the convolutional layers of MNIST and CIFAR10. Since the Hexagon DSP is fixed-point optimized, the quantized version (*DSP-8bit*) of the conventional layers are able to outperform some of the other

versions. Therefore – following intuition – the performance of a single application can be boosted by allocating the workload to the corresponding highest-performing compute unit. *However – counterintuitively – we may be able to exploit seemingly slower compute units to gain overall performance and energy improvements.* Figure 3 illustrates this scenario, showing the execution time of running one to three instances of CIFAR10 in parallel. When executing only one CIFAR10 instance, the *GPU-only* version yields the best result compared to *GPU-CPU* and *GPU-DSP* versions (as expected). However, when we execute multiple instances of CIFAR10 (i.e., panels showing *CIFAR10*2* and *CIFAR10*3*), we observe that offloading to the other seemingly inferior compute units (e.g., CPU & DSP) yields overall better performance. Indeed, when executing 3 instances of CIFAR10 (*CIFAR10*3*), we see that the performance of *GPU-CPU* and *GPU-DSP* significantly outperform the *GPU-only* version, since the GPU is saturated. This simple example motivates the opportunity to exploit underutilized resources such as DSPs as outlined in Sections II-C and II-D.

C. Optimization for Single Application Class

Intuitively, the performance and energy consumption of an application (e.g., CNN) can be improved by partitioning and executing on specific accelerators (e.g., GPUs). But frameworks such as Tensorflow and Caffe run the CNN model on the same GPU, saturating that compute unit while missing the opportunity to improve performance and energy consumption by exploiting other underutilized compute units (e.g., CPU and DSP). Therefore, we partition the neural network at the layer level so each layer can be executed as a task running on a different compute unit to exploit heterogeneity. Figure 4a, 4b shows the execution time, average power and energy consumption of running different versions of MNIST and CIFAR10. For MNIST, *conv2* runs on DSP and the others run on CPU. For CIFAR10, *conv2*, and *conv3* run on DSP, and the others run on GPU. Although *DSP-8bit* has better performance over convolution layers in general as shown in Figure 2, it performs worse due to the floating-point computation in other layers such as the Pooling and ReLu layers. For all quantized models, the accuracy drops 1.4% on average. *Hetero* represents the results of utilizing diverse compute units to gain performance and energy improvements. Indeed, the *Hetero* results show a 15.6% performance boost and a 25.4% energy saving on average compared to *CPU-float* and *GPU-float* (which respectively perform best for MNIST and CIFAR10). We observed similar patterns for the CED experiments as shown in Figure 4c: the *Hetero* configuration for CED executing four stages on the DSP, GPU, CPU, CPU respectively, produced the best performance and energy consumption results, demonstrating the potential to effectively exploit underutilized heterogeneous resources.

Figure 4d shows the results of running multiple CIFAR10 instances. The results are grouped by CPU, GPU and heterogeneous resources and the values are normalized to *CPU-8bit*. For *CPU-8bit*, the performance is scalable but the power and energy consumption increases drastically with more instances because more cores are exercised. The performance of *GPU-8bit* downgrades along with the increase of instances because they contend for the GPU. *Hetero* shows more stability than the others due to the distribution of the workload over all compute resources. We also simulate the scenario when the GPU is saturated by rendering high-quality graphics. We use the GPU Performance Analyzer benchmark to produce a high quality graphics workload. As Figure 4e shows, the performance of GPU-float and *Hetero* decreased significantly because the GPU is fully-saturated by the above-mentioned graphics workload. *Hetero-noGPU* is statically configured to offload the *conv2*, *conv3* and *relu* layers to DSP while the other layers run on CPU. As *Hetero-noGPU* specifically avoided using the GPU, its performance and energy consumption outperforms the others.

D. Optimization for Multiple Application Classes

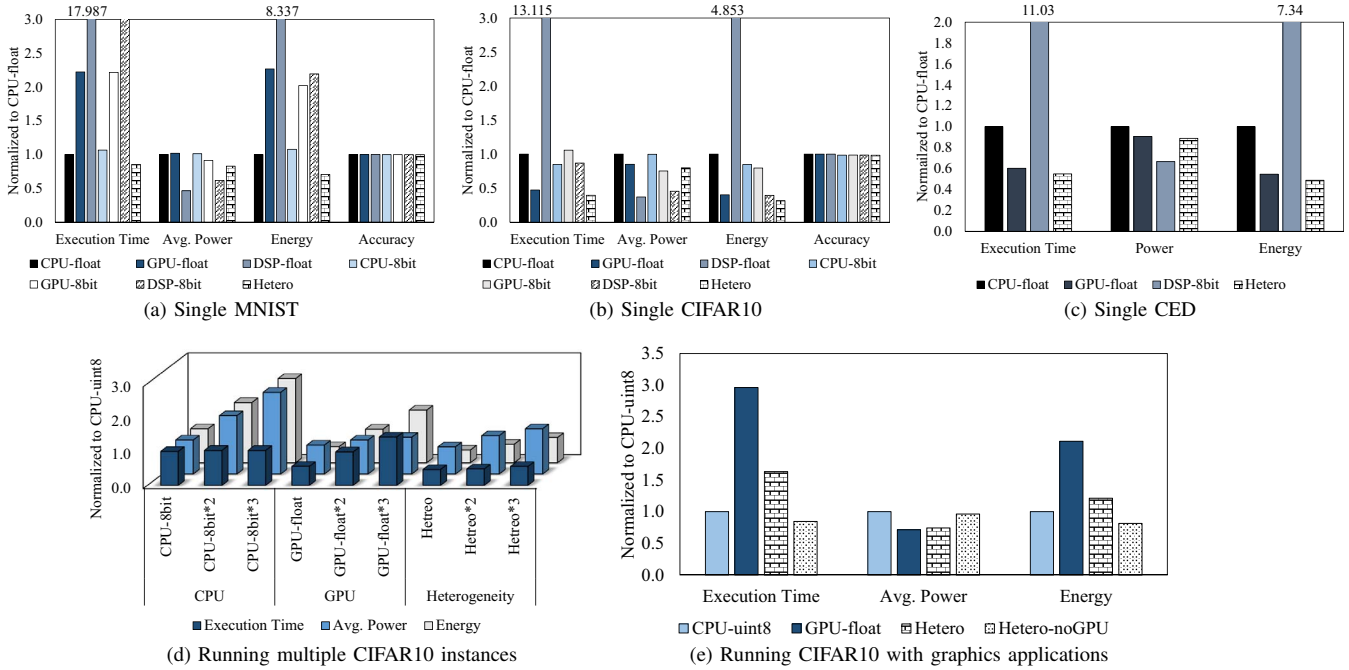
When executing multiple application classes on a system, both the task partitioning and the exploitation of heterogeneous resources help for better distribution of workload, which in turn leads to better performance and energy consumption.

Figure 5 presents the results of running different combinations of canny edge detector (CED), CIFAR10, and the graphics application. *CPU/CPU* means CED runs the CPU and CIFAR10 also runs on the CPU. The other terms in the figure follow the same convention. Execution time is from when we execute all the applications in parallel to when the last application terminates. We make the following observations:

- By exploiting all heterogeneous (including underutilized) resources efficiently, we can achieve better results: the fully heterogeneous *Comb5* outperforms GPU-only *Comb3* by 32% for performance and 29% for energy consumption.
- Underutilized resources may not be beneficial when compute units are not saturated. Consider CPU/GPU *Comb2* versus GPU-only *Comb3*: although *Comb3* executes both applications on the GPU, it is not saturated and thus still yields better results than *Comb2* that distributes utilization among heterogeneous resources.
- The Quality of service (QoS) for each application must also be considered. For instance, although the no-GPU *Comb7* has overall better performance than GPU-saturated *Comb6*, the graphics application suffers a QoS-loss of 8.9% in frame rate when switching from *Comb6* to *Comb7*, showing that we must also respect each application's QoS requirement during task mapping.

III. Related Work

Heterogeneous architectures have been researched from diverse perspectives including performance, power and energy management for a wide spectrum of application domains, including CPU-GPU collaboration [3] [4] [5] [6] [7]. and CPU-DSP collaboration [8]. There is a large body of work on accelerating convolutional neural networks (CNNs) by offloading to hardware accelerators [9] [10] [11].



(a) Single MNIST (b) Single CIFAR10 (c) Single CED

(d) Running multiple CIFAR10 instances (e) Running CIFAR10 with graphics applications

Fig. 4: Performance, power, and energy consumption for single or multiple CNNs/CED with different task mapping

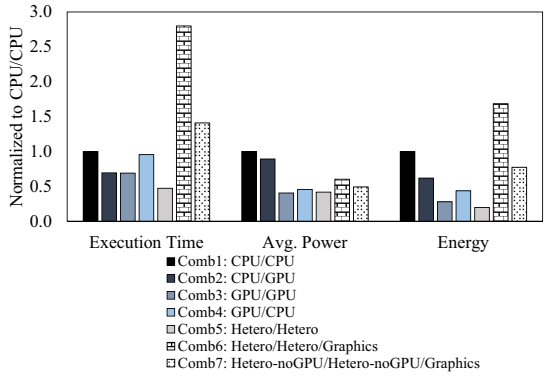


Fig. 5: Scenarios of running multiple applications

Our work differs from the above in multiple ways. First, we focus on mobile energy-constrained heterogeneous platforms and aim to holistically deploy execution of applications that can execute across multiple heterogeneous compute resources including CPU, GPU and DSP (as opposed to only one or a pair of resources). Second, we experimentally demonstrate simultaneous improvements in performance and energy through task mapping that efficiently exploits underutilized heterogeneous resources (e.g., the DSP on the mobile Snapdragon 835 platform).

IV. Conclusion

In this paper, we presented a case study using CNN, computer vision, and graphics applications to demonstrate the ability to exploit available yet underutilized heterogeneous resources to improve both performance and energy in mobile devices. We illustrated the potential for exploiting the DSP for such applications as this accelerator can be an efficient compute alternative that is generally underutilized. We examined different application scenarios to show the benefit of having higher heterogeneity in SoCs. For single and multiple application scenarios executing mixed workloads, we observed an average performance and energy consumption improvement of 15-46% and 18-80%, respectively, by synergistically deploying all available compute resources, especially the underutilized DSP. The performance and energy consumption turn out to be further improved when all the available compute resources are considered for the computation.

References

[1] T. Profiler, "Qualcomm."
[2] J. Gmez-Luna *et al.*, "Chai: Collaborative heterogeneous applications for integrated-architectures," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.
[3] J. Lee *et al.*, "Orchestrating multiple data-parallel kernels on multiple devices," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*.
[4] P. Pandit *et al.*, "Fluidic kernels: Cooperative execution of opencl programs on multiple heterogeneous devices," ser. CGO '14.
[5] A. Prakash *et al.*, "Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*.
[6] J.-G. Park *et al.*, "Synergistic cpu-gpu frequency capping for energy-efficient mobile games," *ACM Trans. Embed. Comput. Syst.*, 2017.
[7] C.-Y. Hsieh *et al.*, "Memcop: memory-aware co-operative power management governor for mobile games," *Design Automation for Embedded Systems*, Mar 2018.
[8] K. Chandramohan *et al.*, "Partitioning data-parallel programs for heterogeneous mpsocs: Time and energy design space exploration," ser. LCTES '14.
[9] U. Aydonat *et al.*, "An opencl™deep learning accelerator on arria 10," ser. FPGA '17.
[10] G. Hegde *et al.*, "Caffepresso: An optimized library for deep learning on embedded accelerator-based platforms," in *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*.
[11] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.