# Mobile Application Processors: Techniques for Software Power-Performance Optimization

**Alok Prakash**
Nanyang Technological University

**Tulika Mitra**
National University of Singapore

**Siqi Wang**
National University of Singapore

***Abstract*—The latest and upcoming mobile application processors, embedded in a myriad of consumer devices, are typically implemented as heterogeneous multiprocessor system-on-chips, comprised of various processing engines such as general-purpose cores with differing characteristics, GPUs, DSPs, nonprogrammable accelerators, and reconfigurable computing devices. The heterogeneity allows application kernels with specific requirements to be executed on processing core(s) ideally suited to perform that computation, resulting in a significantly improved performance and energy-efficiency. However, while heterogeneous computing is an attractive proposition in theory, considerable software support at all levels is essential to fully realize its promises. The system software needs to orchestrate the different on-chip compute resources in a synergistic manner with minimal engagement from the application developers, as well as fulfill the stringent power-thermal constraint in mobile computing. In this article, we present compiler time and runtime techniques to unleash the full potential of heterogeneous multiprocessors toward high-performance energy-efficient computing on consumer devices.**

*Digital Object Identifier 10.1109/MCE.2020.2969171*
*Date of current version 8 June 2020.*

**Mobile application processors** are the brains of consumer devices, fueling the tremendous growth in their capabilities over the past decade. The thirst for achieving higher performance at a low power budget, especially for mobile computing, has been a key driver of the significant advancements made in the VLSI domain that has led to the integration of multitude of sophisticated processing cores into a single chip. Huawei's HiSilicon Kirin 980, Apple's A12 Bionic as well as the upcoming Samsung's Exynos 9820, Qualcomm's Snapdragon 855, and Mediatek's Helio X30 SoCs are examples of such heterogeneous, high performance, low power multiprocessor system on chips (MPSoCs).

## Mobile Application Processors: Landscape

One of the key characteristics in all the latest and upcoming MPSoCs is the use of a variety of dedicated processing cores for specific tasks such as embedded graphics processing units (GPU) for graphics applications, digital signal processors (DSP) for signal processing tasks, neural processing units (NPU) for the slew of new and upcoming machine learning and AI applications, etc.[1] The heterogeneity in these MPSoCs can be broadly classified into two types: *performance heterogeneity* and *functional heterogeneity*. Performance heterogeneity refers to the use of two or more types of cores that offer the same functionality by supporting the same instruction set architecture (ISA), but differ in their power-performance characteristics. ARM's traditional big.LITTLE and the more recent DynamIQ cluster architectures as well as Mediatek's tricluster designs are examples of such architectures that combine CPUs offering high performance at higher power consumption together with lower performance, but more power-efficient cores. In such a setup, the high performance cores are typically used only for demanding applications such as games and content heavy web browsing, while the power efficient cores are used for all other mundane or background tasks. Figure 1 shows the block diagram of Huawei's HiSilicon Kirin 980 MPSoC that employs ARM's DynamIQ architecture by combining higher performance ARM Cortex A76 CPUs with the power-efficient ARM Cortex A55 CPUs.

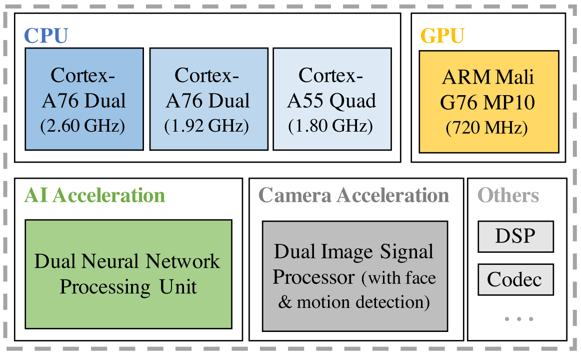
CPU
Cortex-A76 Dual (2.60 GHz)
Cortex-A76 Dual (1.92 GHz)
Cortex-A55 Quad (1.80 GHz)
GPU
ARM Mali G76 MP10 (720 MHz)
AI Acceleration
Dual Neural Network Processing Unit
Camera Acceleration
Dual Image Signal Processor (with face & motion detection)
Others
DSP
Codec
. . .

**Figure 1.** Huawei HiSilicon Kirin 980 MPSoC.

On the other hand, functional heterogeneity refers to the use of *specialized processing cores for specific tasks* to allow applications from a certain class to execute most efficiently on these cores. For example, GPUs are primarily used for graphics intensive applications such as gaming with high fine-grained parallelism, while FPGAs can effectively exploit both fine and coarse grained parallelism and are especially suitable for applications with irregular control flow. Modern SoCs, similar to one shown in Figure 1, integrate a myriad of such dedicated functionally heterogeneous processing engines such as GPU, ISP, DSP, etc., and even FPGAs to improve both performance and power efficiency. A recent trend over the past couple of years is to also integrate a dedicated neural processing engine to specifically accelerate machine learning applications on these mobile application processors.

## Software Power-Performance Optimization: Challenges

It is clearly evident that the wide assortment of processing cores, both general purpose and specialized, used in modern SoCs necessitates software developers to take special care to *map* their applications effectively on such platforms. Unfortunately, the growth in the software ecosystem has not kept pace with the rapid growth in the hardware capabilities of these MPSoCs. The *mapping* process involves 1) the selection of the most appropriate processing core for each segment of the code, 2) translating and optimizing the code segment from the high-level language to the one appropriate for the selected core, and finally 3), partitioning the key computational kernels in the code segment across multiple processing cores to further exploit parallelism for performance and energy efficiency. The mapping process is typically performed statically at compile-time and adapted dynamically at runtime for power and energy efficiency to provide good user experience and long battery life. The less than stellar growth in the capacity of batteries[2] used in mobile devices puts additional pressure on system designers to make the most efficient use of the underlying hardware to ensure long battery life. However, the complex nature of the mapping process, as described above, makes it a daunting task to fully exploit the benefits of the advancements made in the hardware domain.

Another important aspect in mobile computing that is often overlooked is that while energy efficiency is important for portable devices to ensure long battery life, power-efficient execution is also critical to ensure that the devices do not overheat at any point. With the advent of numerous wearable devices that are often directly in contact with our skin, the importance of a power-efficient execution cannot be overstated. As will be discussed later in this article, this creates further challenges, while also presenting unique opportunities, during application mapping on modern heterogeneous MPSoCs.

This article builds upon our earlier work[1] and discusses several state-of-the-art compile-time and runtime approaches to ease the adoption of heterogeneous computing in consumer electronic devices. In particular, we discuss 1) recent approaches to obtain early performance estimates on various accelerators to aid the selection of the most appropriate accelerator for a given application, 2) latest techniques for concurrently using multiple heterogeneous resources on an MPSoC for higher power-performance efficiency, and finally 3), we focus on specific techniques for efficient mapping of popular applications such as three-dimensional (3-D) gaming and machine learning on heterogeneous mobile MPSoCs.

## SELECTING THE MOST APPROPRIATE ACCELERATOR

The diverse set of heterogeneous processing cores present on modern SoCs enables applications to execute with limited resources and stringent power/thermal budget of mobile computing which are previously impossible. While different processing cores present different performance gains for different types of applications, it is usually difficult to make the delicate choice of the most appropriate accelerator to attain maximum benefits. The difficulties lie in the knowledge required to utilize all accelerators on board as well as timely redevelopment effort in accelerator specific languages (CUDA for GPU, Verilog for FPGA, etc.) to obtain actual performance benefits of applications on accelerators. If the performance of applications on different accelerators are made available at an early stage, application developers will then be able to make an informed decision in selecting the most appropriate accelerator to use and then concentrate on further platform-specific languages and optimizations. Cross-platform performance prediction tools are proposed to ease the process. We present several state-of-the-art frameworks (Lin-Analyzer,[3] MPSeeker,[4] CGPredict[5]) and illustrate how they are used to achieve a choice between GPU and FPGA for different applications at early stage.

### Cross-Platform Prediction Tools for Accelerators

Dedicated accelerators continues to grow in architectural complexity to sustain the increasing demand for performance and power efficiency. Traditionally, cycle-accurate simulators and high-level synthesis (HLS) tools are used to predict the performance in design time at the cost of slow runtime, especially with significant number of design points in design space exploration (DSE). Cross-platform prediction tools that run much faster, on the other hand, present potential of a much faster and efficient DSE.[3]

In the domain of FPGA, MPSeeker[4] presents a high-level analysis framework that estimates the FPGA performance and resource requirement from sequential C/C++ code without invoking a commercial HLS tool. As shown in Figure 2, MPSeeker takes a high-level specification (C/C++) of an application in the form of tiled nested loops, resource constraints and optimization pragmas as inputs. The execution of the application is captured through a dynamic trace, which

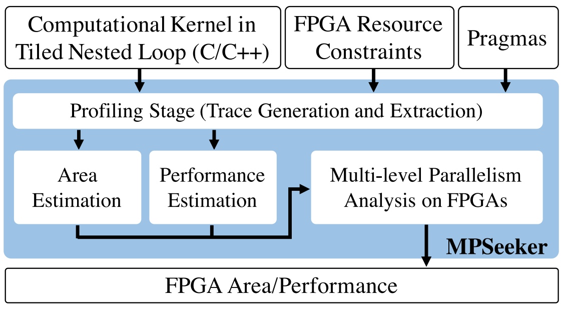
Computational Kernel in Tiled Nested Loop (C/C++)
FPGA Resource Constraints
Pragmas
Profiling Stage (Trace Generation and Extraction)
Area Estimation
Performance Estimation
Multi-level Parallelism Analysis on FPGAs
MPSeeker
FPGA Area/Performance

**Figure 2.** MPSeeker: An automatic DSE framework with multilevel parallelism for FPGA.[4]

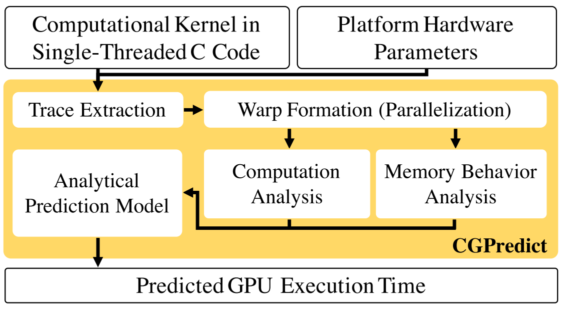
Computational Kernel in Single-Threaded C Code
Platform Hardware Parameters
Trace Extraction
Warp Formation (Parallelization)
Analytical Prediction Model
Computation Analysis
Memory Behavior Analysis
CGPredict
Predicted GPU Execution Time

**Figure 3.** CGPredict: An analytical framework for performance estimation from single-threaded C Code for GPU.[5]

is generated by profiling the intermediate representation of low-level virtual machine (LLVM IR). The *area estimation* is predicted through a machine learning technique implementing a gradient boosted machine. The kernel computation cycles (*performance estimation*) is estimated through dynamic data dependence graph scheduling[3] together with data communication cost. With DSE performed by estimating both performance and area for each pragma combination, MPSeeker is able to recommends the best configuration in *multilevel parallelism analysis*. While achieving high prediction accuracy, MPSeeker recommends the same optimal design point with more than 500X DSE speed-up.

In contrast to FPGAs, GPUs enabling concurrent execution of thousands of fine-grained threads through the highly multithreaded architecture, present more challenges in predicting the performance from a single-threaded code. CGPredict[5] proposes an analytical framework to estimate the performance of a computational kernel on an embedded GPU architecture from unoptimized, single-threaded or sequential C code. As shown in Figure 3, CGPredict analyze a dynamic execution trace similar to MPSeeker. The single-threaded trace is transformed through a *Warp Formation* into its multithreaded equivalent that can be potentially exploited by the GPU. With the parallelized trace, the compute instructions are mapped to CUDA PTX ISA to predict the compute cycles in *computation analysis*, whereas the memory access trace is analyzed to capture the interaction between cache and DRAM in *memory behavior analysis*. Along with the architectural parameters obtained by microbenchmarking, the compute and memory information is analyzed through an *analytical prediction model* that captures the inherent memory latency hiding capability of GPU to obtain the final execution performance. In addition, as CGPredict performs analytical modeling, the characteristics of the kernel on GPU architecture (coalescing of memory accesses, shared memory usage, etc.) that influence performance are made transparent. This information provides insights for the programmers to understand the intrinsic strengths and weakness of the architecture for a particular kernel that can facilitate further code optimizations.

## Selecting the Accelerator

With the Lin-Analyzer[3](MPSeeker[4]) and CGPredict,[5] the execution performance of a computational kernel on FPGA and GPU, respectively, can be predicted from its C code implementation, thus easing effort of redevelopment for the designers in selecting appropriate accelerators. A set of experiments are carried out to demonstrate the effectiveness of the frameworks. The benchmarks are mainly matrix calculations implemented with multiple nested loops. Equivalent CUDA code are implemented manually to verify actual GPU performance.[5] Experiments were performed on the NVIDIA embedded Kelper GPU on Jetson TK1 development board (852 MHz) and Xilinx ZC702 embedded FPGA (100 MHz).

Table 1 shows that with CGPredict and MPSeeker, the correct accelerator choice (in bold font) can be made for each application. For some benchmarks (MM, GEMVER1, DERICHE1), GPU is more preferable than FPGA, since the GPU platform in the experiments provides higher processing capability (852 MHz, b/w: 17 GB/s) compared to the FPGA platform (100 MHz, b/w:

**Table 1. Comparison of estimated and actual performance of benchmarks on GPU and FPGA.**

| Benchmark | Input | GPU time (ms) | | FPGA time (ms) | |
|---|---|---|---|---|---|
| Name | Size | Estimate | Actual | Estimated | Actual |
| MM | 1024 | **242.51** | **250.27** | 1180 | 1450 |
| MVT | 2048 | 48.31 | 42.37 | **9.09** | **10.41** |
| GEMVER1 | 2048 | **2.61** | **4.57** | 16.55 | 19.81 |
| DERICHE1 | 1024 | **0.95** | **1.53** | 2.99 | 3.37 |
| DCT1D | 1024 | 2697.75 | 2685.36 | **636.47** | **650.8** |

4 GB/s). Additionally, the three benchmarks exhibit coalesced memory access pattern, which significantly reduces memory transactions and improves performance for GPU. FPGA is the preferred choice for MVT and DCT1D, as both application present uncoalesced memory access patterns, causing GPU to suffer from extensive memory transactions. On the other hand, FPGA loads input data of several tiles into its local memory before the execution starts improving, otherwise slow nonlocal memory accesses. It is noteworthy that GPU performance could be improved by optimization techniques such as data layout transformation, loop tiling with shared memory and vectorization, as suggested by the uncoalesced memory access pattern.

In general, with the aforementioned compile time cross-platform performance prediction tools, the designers are able to make an informed decision on which accelerator to use for a computational kernel. With the analytical analysis, designers can further perform hardware specific optimization to achieve even better performance.

## CPU AND ACCELERATOR COEXECUTION

We presented several techniques in the "Selecting the Most Appropriate Accelerator" section that can be effectively relied upon to rapidly estimate the performance and even power consumption of application kernels on multicore CPUs as well as various accelerators like GPU and FPGA. The early, typically at compile-time, estimates enable system designers to select the most appropriate accelerator to achieve both performance and power efficiency on platforms with a diverse set of accelerators.

At the same time, the various processing elements (PE), including the accelerators, on a heterogeneous platform can also be exploited simultaneously to achieve even higher power-performance efficiency. Various compile-time and runtime strategies, targeting both general purpose as well as specific applications, have been proposed recently to maximize the power-performance efficiency. This section discusses the latest state-of-the-art techniques proposed in this direction.

### General Purpose Applications

The various computational kernels in a general purpose application can be mapped on the multicore CPUs and even the accompanying accelerator such as a GPU that are widely available in latest mobile application processors. Application tasks can be mapped on the big.LITTLE multicore CPUs and migrated at runtime from the *big* to the *LITTLE* cores, and vice versa, to achieve either higher performance or power efficiency. However, prior to task migration, performance must be estimated for the target core to allow a cost-benefit analysis. An alternative widely used strategy for power-efficient runtime execution is dynamic voltage frequency scaling (DVFS). This technique is frequently implemented in consumer devices such as hand-held smartphones or tablets and wearable smartwatches. As mentioned earlier in the "Mobile Application Processors: Landscape" section, many existing techniques focus on energy-efficient execution to extend the battery life of battery-operated systems. On the other hand, power-efficient

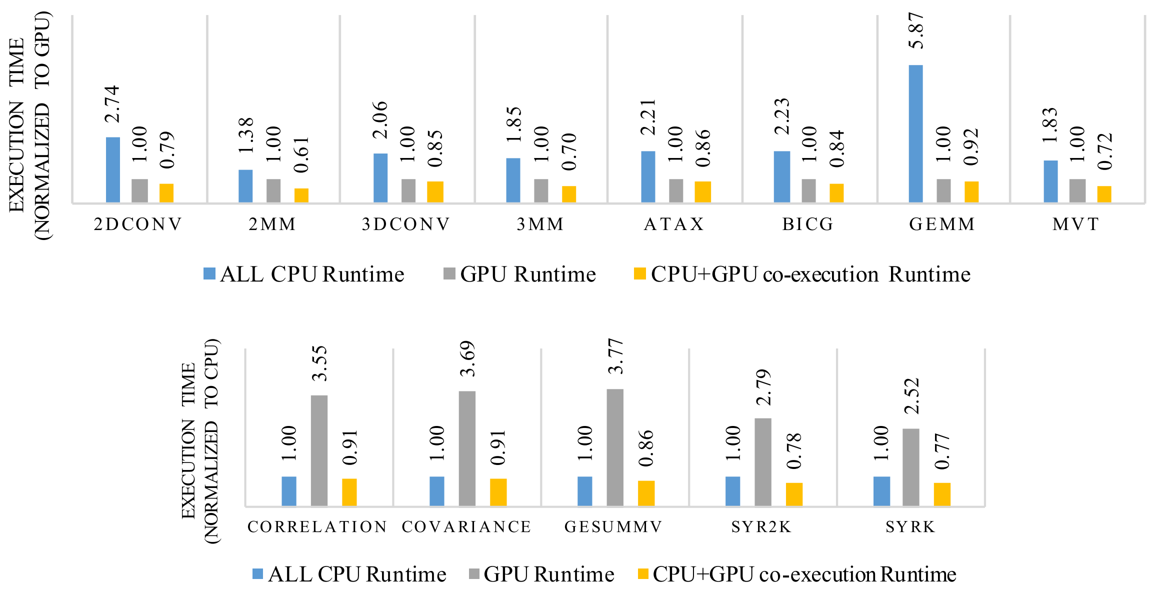

**Figure 4.** Benefit of coexecution of CPU and GPU.

execution is also an important concern to ensure that device temperatures remain tolerable for users holding or wearing them. However, this is a challenging problem especially due to the contradictory requirements of maintaining high user experience while simultaneously focusing on a power-efficient execution. Mobile operating systems typically include several rule-based DVFS governors that manipulate frequency based on runtime characteristics such CPU and GPU utilization. However, these governors primarily focus on power management and largely ignore thermal management. Several strategies such as control-theoretic, price-theory, and even approximation-aware scheduling have been used in conjunction with DVFS for effective power- and thermal-aware mapping of applications on the ARM big.LITTLE CPU cores.

Taking one step further, the concurrent coexecution of a single computational kernel on multiple PEs or accelerators, in tandem with the multicore CPUs, exhibits great potential in achieving additional performance and power efficiency. It has been observed that some applications are better suited for acceleration on GPUs while others execute more efficiently on the CPU. Even so, executing the application simultaneously on all available processing cores (big and LITTLE CPU cores and the GPU), as proposed by Prakash *et al.*,[6] together with suitable DVFS settings for all these cores, leads to the most energy-efficient execution. Experimental results from the work by Prakash *et al.*,[6] shown in Figure 4, confirm that the improvement in runtime by executing concurrently on CPU and GPU can be as high as 39.4% (2MM) and on average 19.2%, compared to the stand-alone executions. The large improvement in execution time allows designers to have more flexibility in tuning the various DVFS settings to achieve higher performance under certain constraints. This work also showed that designers can achieve the best performance-energy tradeoff (using their metric $ED^2$) with appropriate voltage-frequency settings.

However, the coexecution of multiple power hungry PEs expedites the accumulation of heat and the chip temperature may repeatedly exceed the temperature threshold. Operating systems typically include thermal management measures, such as throttling of frequency to reduce power consumption drastically or cutoff power completely to protect the chip from failure. Therefore, the benefit of coexecution can be compromised by the throttling of frequency by the underlying OS. OPTiC is proposed by Wang *et al.*,[7] to anticipate such thermal behaviors and predicts the best configuration of the multiple PEs (CPU, GPU). The thermal behavior of a computational kernel on a PE is captured through a thermal throttling model that predicts the occurrence of OS throttling and performance under such thermal condition. The power and performance of the CPU and GPU at all frequencies are predicted through analytical modeling from profile runs. From the individual PE performances, the allocation of workload and the coexecution performance are predicted through a coexecution model that considers the effect of thermal frequency throttling and the resource contention. The framework then goes through all possible frequency settings and predicts the performance to determine the optimal configurations. While the performance of the application is largely affected by thermal conditions, OPTiC is able to predict the configuration that presents on average 13.8% performance improvement over standalone execution. OPTiC further shows great temperature control over real-life applications. With the configuration predicted by OPTiC, the chip exhibits a much cooler temperature when compared to the Linux frequency governors.

### Application Specific Optimizations

Apart from developing strategies to improve the power-performance efficiency of general purpose applications, several authors have also proposed specific techniques to leverage the underlying hardware for widely used applications such as three-dimensional (3-D) gaming and machine learning.

**3-D Gaming Applications** Millions of smartphone users play a large variety of 3-D games on their mobile phones. The drastic growth in the capabilities of the mobile application processors over the past decade has greatly benefited these games, allowing them to produce ever realistic graphics in high resolution and highly imaginative and interesting gameplays. However, the concurrent use of several processing cores needed to play these games also results in high power consumption during gameplay, reducing battery life and increasing device temperature. Several researchers have proposed novel techniques for power management of gaming applications, including DVFS-based approaches that provide options to trade power consumption with gaming performance.

Temperature coupling between the processor and battery has also been considered for dynamic thermal management (DTM) on mobile devices. Predictive approaches for thermal management of both the CPU as well the GPU has been studied by Singla *et al.*,[8] where the authors propose to first turn OFF the power-hungry A15 cores, one at a time, if their algorithm predicts a thermal throttling. If this does not bring the temperature down sufficiently, the frequency of A7 CPU cores and GPU are reduced next. This approach leads to a notable reduction in the variance of on-chip temperature, however, comes at the cost of reduced performance especially for 3-D gaming workloads that exhibit a complex dependence on both CPU and GPU.

During a gameplay, it was observed that as the temperature of the SoC rises, the default Linux thermal management unit (TMU) monitors the CPU temperature and throttles the CPU frequency once the temperatures rises to a predefined threshold (for example, 70°C). The GPU frequency is throttled at a later stage by its device driver and is not coordinated with the CPU. This allows the SoC temperature to continue to increase even after the CPU frequency is throttled. The SoC only cools down after both the CPU and GPU frequencies are throttled significantly. The net result of this *uncoordinated* DVFS is a significant oscillation in the on-chip temperature that may even create reliability issues. Additionally, unlike power consumption, because of the close proximity of CPU and GPU, even a modest increase in the temperature of one directly impacts the temperature of the other. This is called the *thermal coupling* phenomenon that makes it difficult to predict the thermal behavior of one processing core in isolation.

Hence, we proposed a control-theory-based cooperative CPU–GPU DTM technique by Prakash *et al.*[9] and implemented it, as well as the technique proposed by Singla *et al.*[8] on a real mobile development platform, the Arndale development board and used it to control the frequency settings of CPU and GPU while playing several 3-D games. The Arndale development platform runs on a Samsung Exynos 5250 heterogeneous MPSoC that consists of a dual core ARM Cortex-A15 processor and a high performance Mali T604 quad-core GPU. The resulting performance (frames per second metric for gaming workloads) and the on-chip temperature were observed. Figure 5(a) and (b) shows that the proposed algorithm not only reduced the on-chip temperature variance, but also achieved

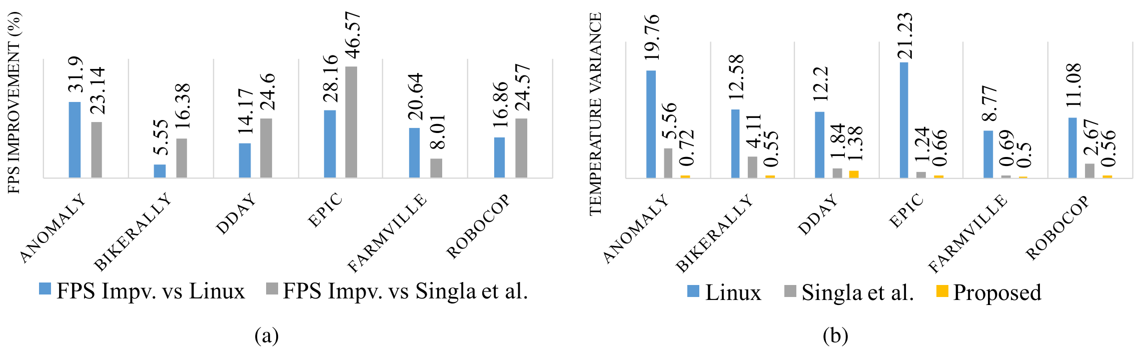

**Figure 5.** Results achieved by Prakash et al.[9] (a) Improvement in performance (FPS) and (b) temperature variance reduction.

higher performance than the default Linux TMU and the approach by Singla *et al.*[8]

**Machine Learning Applications** Even though mobile platforms are not powerful enough for "training" of deep neural networks (NN), they are being increasingly deployed for "inference" using previously trained models to enrich user experiences in many mobile applications. For example, speech recognition enabled by recurrent neural networks (RNNs) and continuous vision enabled by convolutional neural networks (CNNs) that extract high-level semantic information from real-time video streams is used in many applications, such as object recognition and augmented reality games. There has been tremendous progress to port NN applications onto mobile devices. The effort started with cloud-assisted approaches[10] where only the acquisition of information (e.g., images) happened on the mobile devices bypassing their hardware limitations. But, these approaches are limited by the network connection and may even lead to breach of data privacy. The recent advancements in embedded SoC architectures, including more powerful general purpose cores and dedicated neural processing engines, enable mobile devices to perform NN inference. Applications such as complex gesture recognition using machine learning are being attempted on latest mobile phones.[11] However, the enormous computation and memory requirements are still unresolved challenges. Researchers are approaching this challenge from two complementary directions: introducing new dedicated hardware accelerators to speed up NN applications, e.g., commercialized Google TPU, Huawei NPU, etc., and system-level effort in exploiting existing architectures for NN applications.

New accelerators have led to substantial performance speed-ups and power savings. The Huawei Kirin 980 chip, featuring dual NPU, is claimed to have achieved peak processing of 1.92 TFLOPS (for FP16) per NPU, which is 25X the performance of a CPU with 50X the energy efficiency. However, these accelerators are heavily customized to the special characteristics of current NN applications. With the fast evolving NN structures, new architectures are expected to emerge on consumer devices in the near future.

On the other hand, system-level effort in enabling deep learning on existing architecture exhibits generality and can be widely applied in current consumer devices. Highly tuned and optimized libraries, such as *ARM Compute Library Tencent NCNN*, are created to facilitate efficient CNN implementation on the *ARM* cortex core architecture, which is prevalent in mobile devices. These libraries are optimized at assembly level with multithreading on multicores and acceleration through *ARM NEON* vectorization technology, facilitating the developers with highly efficient NN implementation on mobile devices. However, most of the implementations are ignorant to heterogeneities present in the system, causing the coexecution on multiple PEs (e.g., big and little CPU clusters) to have worse performance. In addition, GPU, as a common accelerator in most mobile devices, demonstrates excellent performance in NN applications. Hence, frameworks have been proposed to enable CNN on existing platforms with collaborative execution of CPU and GPU. One of the most recent effort, CGOOD,[12] based on the CNN configurations, hardware specifications and optimization requirements, generates C code and GPU code (CUDA or OpenCL) that runs on respective mobile platforms and achieves up to 25X improvement. It should be noted that the actual performance of applications are, however, largely dependent on optimizations. For example, OpenCL implementations for GPU is observed to provide two times worse performance compared to CPU in object detection with the OpenCV DNN module.

## CONCLUSION AND FUTURE OUTLOOK

In this article, we presented several state-of-the-art techniques for optimizing power and performance of software on latest mobile application processors. To aid the selection of the most appropriate accelerator for application kernels, techniques for estimating their performance on GPU and FPGAs were discussed. This was followed by a survey of the latest strategies proposed to simultaneously exploit multiple accelerators of the SoC. While compile-time techniques help in achieving high performance-energy efficiency, runtime strategies are employed primarily for dynamic power and thermal management in such devices without sacrificing performance.

The novel strategies discussed in this article have indeed made great strides toward making the most use of the underlying cutting-edge hardware in latest mobile application processors. However, significant hurdles still need to be overcome before application programmers can effortlessly and effectively exploit heterogeneous MPSoCs. For example, an integrated compiler support is paramount for the compilation and seamless mapping of application kernels on a given heterogeneous processing platform that consists of performance-heterogeneous CPU cores as well as functionally heterogeneous cores such as GPU and FPGA. At the same time, other accelerators such as DSPs and ISPs also need to be supported by this integrated compiler to make the entire application mapping process on hardware an entirely transparent step to software developers. In addition, for applications with highly input-dependent behavior, a dynamic strategy for workload partitioning and scheduling on the heterogeneous cores may prove to be more suitable than the static solutions discussed in this article. For example, a runtime work stealing approach has been adopted in Zhong *et al.*[13] to distribute the workloads for the CNNs inference engines on the CPU cores with vector processing units (ARM Neon) and FPGA-based accelerators for improved load balancing.

Finally, while runtime power and thermal management for gaming applications has received considerable attention in the past, GPGPU applications such as machine learning and image processing still need to be thoroughly investigated for power and thermal efficient execution. Such applications are increasingly being ported to mobile devices, demanding both high performance and power efficiency. Other than DVFS-based solutions, runtime task migration strategies should also be developed to ensure an equitable distribution of workloads across all processing cores, without over-stressing one or a subset of cores, to improve the overall system reliability.[14]

## ACKNOWLEDGMENT

This article is an extended version of our earlier work “Software Support for Heterogeneous Computing,” published in the 2018 IEEE Computer Society Annual Symposium on VLSI.[1] This work was supported in part by Singapore Ministry of Education Academic Research Fund Tier 2 MOE2015-T2-2-088 and in part by the National Research Foundation Singapore under its Campus for Research Excellence and Technological Enterprise program with Technical University of Munich, TUMCREATE.

## REFERENCES

1. S. Wang, A. Prakash, and T. Mitra, “Software support for heterogeneous computing,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2018, pp. 756–762.
2. T. Coughlin and IEEE Consumer Electronics Society Future Directions Committee, “A Moore’s law for mobile energy: Improving upon conventional batteries and energy sources for mobile devices,” *IEEE Consum. Electron. Mag.*, vol. 4, no. 1, pp. 74–82, Jan. 2015.
3. G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, “Lin-analyzer: A high-level performance analysis tool for FPGA-based accelerators,” in *Proc. 53rd Annu. Des. Autom. Conf.*, 2016, pp. 1–6. [Online]. Available: https://github.com/zhguanw/lin-analyzer

4. G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, “Design Space exploration of FPGA-based accelerators with multi-level parallelism,” in *Proc. IEEE Des., Autom. Test Europe Conf. Exhib.*, 2017, pp. 1141–1146.
5. S. Wang, G. Zhong, and T. Mitra, “CGPredict: Embedded GPU performance estimation from single-threaded applications,” *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 146:1–146:22, 2017.
6. A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra, “Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms,” in *Proc. 33rd IEEE Int. Conf. Comput. Des.*, 2015, pp. 208–215.
7. S. Wang, G. Ananthanarayanan, and T. Mitra, “OPTiC: Optimizing collaborative CPU–GPU computing on mobile devices with thermal constraints,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 3, pp. 393–406, Mar. 2019.
8. G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, “Predictive dynamic thermal and power management for heterogeneous mobile platforms,” in *Proc. Design, Autom. Test Europe Conf. Exhib.*, 2015, pp. 960–965.
9. A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, “Improving mobile gaming performance through cooperative CPU-GPU thermal management,” in *Proc. 53rd Annu. Des. Autom. Conf.*, 2016, pp. 47:1–47:6.
10. S. K. Sharma and X. Wang, “Live data analytics with collaborative edge and cloud processing in wireless IoT networks,” *IEEE Access*, vol. 5, pp. 4621–4635, 2017.
11. M. Panella and R. Altilio, “A smartphone-based application using machine learning for gesture recognition: Using feature extraction and template matching via Hu image moments to recognize gestures,” *IEEE Consum. Electron. Mag.*, vol. 8, no. 1, pp. 25–29, Jan. 2019.
12. D. Kang, E. Kim, I. Bae, B. Egger, and S. Ha, “C-good: C-code generation framework for optimized on-device deep learning,” in *Proc. Int. Conf. Comput.-Aided Des.*, 2018, pp. 105:1–105:8.
13. G. Zhong, A. Dubey, C. Tan, and T. Mitra, “Synergy: An HW/SW framework for high throughput CNNs on embedded heterogeneous SoC,” *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 2, pp. 13:1–13:23, 2019.
14. C. Bolchini, M. Carminati, T. Mitra, and T. S. Muthukaruppan, “Combined on-line lifetime-energy optimization for asymmetric multicores,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst.*, 2016, pp. 35–40.

**Alok Prakash** is currently a Senior Research Fellow with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. Contact him at alok@ntu.edu.sg.

**Siqi Wang** is currently a Research Assistant and working toward the Ph.D. degree with the School of Computing, National University of Singapore, Singapore. Contact her at wangsq@comp.nus.edu.sg.

**Tulika Mitra** is currently a Professor with the School of Computing, National University of Singapore, Singapore. Contact her at tulika@comp.nus.edu.sg.