Software Support for Heterogeneous Computing

Siqi Wang
School of Computing
National University of Singapore
Singapore
wangsq@comp.nus.edu.sg

Alok Prakash
School of Computer Science and Engineering
Nanyang Technological University
Singapore
alok@ntu.edu.sg

Tulika Mitra
School of Computing
National University of Singapore
Singapore
tulika@comp.nus.edu.sg

***Abstract*—Heterogeneous computing, materialized in the form of multiprocessor system-on-chips (MPSoC) comprising of various processing elements such as general-purpose cores with differing characteristics, GPUs, DSPs, non-programmable accelerators, and reconfigurable computing, are expected to dominate the current and the future consumer device landscape. The heterogeneity enables a computational kernel with specific requirements to be paired with the processing element(s) ideally suited to perform that computation, leading to substantially improved performance and energy-efficiency. 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. We present compiler time and runtime techniques to unleash the full potential of heterogeneous multi-cores towards high-performance energy-efficient computing on consumer devices.**

***Keywords*-Heterogeneous computing, scheduler, compiler, power/thermal management**

I. Introduction

Current and emerging consumer devices include *mobile application processors* for computation. State-of-the-art mobile application processors are the pinnacles of the system-on-chip (SoC) movement where multiple different functionalities are integrated on a single chip. Examples of current-generation commercial mobile processors include Qualcomm Snapdragon 845 [1], Huawei Hisilicon Kirin 970 [2], Samsung Exynos 9810 [3], and Apple A11 Bionic [4].

The common characteristic of all current- and next-generation mobile processors is the presence of heterogeneous computing [5], which refers to systems that use more than one kind of processing core on the same chip. The diversity can come from two sources. The first is *performance heterogeneity*, where cores with the same instruction-set architecture but different power-performance characteristics co-habit on the same die. The second is *functional heterogeneity*, where cores with very different functional characteristics such as general-purpose CPU, programmable GPU, special-purpose accelerators, and Field-Programmable Gate Arrays (FPGAs) appear on the same die. Recently we have witnessed the introduction of commercial SoCs with both performance and functional heterogeneity in the consumer devices. The modern mobile processor SoCs typically comprise of (a) multiple clusters of general-purpose CPU cores with varying power-performance characteristics, for example, ARM big.LITTLE architecture [6] where high-performance, high-power cores appear alongside low-performance, low-power cores catering to different workloads, (b) general-purpose Graphics Processing Unit (GPU) cores such as NVIDIA Kepler [7], ARM Mali [8], or Qualcomm Adreno [9], (c) Digital Signal Processor (DSP) cores such as Qualcomm Hexagon [10], and (d) a large collection of accelerators, such as Neural Network Processing Unit (NPU), Image Signal Processor (ISP) etc. in Hisilicon Kirin 970 [2], Neural Engine, motion coprocessor, image processor etc. in Apple A11 Bionic [4] as shown in Figure 1. Apart from the architectural and micro-architectural differences, these multi-cores offer additional design points in the power-performance trade-off curve through *dynamic voltage-frequency scaling (DVFS)* of the cores.

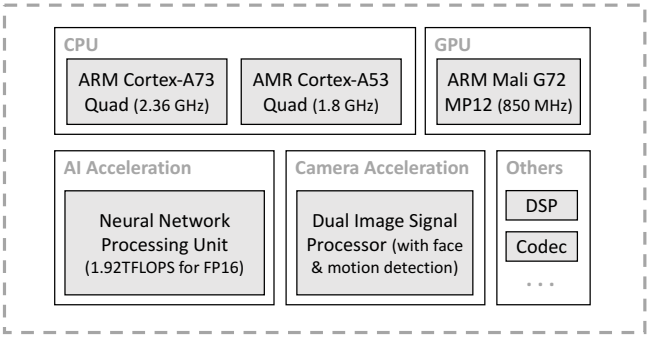


Figure 1: Huawei Hisilicon Kirin970 MPSoC [2]

Clearly, the contemporary mobile processors are complex yet power-efficient systems featuring heterogeneous computing that have the potential to deliver high-performance if all the available resources can be harnessed by the software. At any point, we need to use the cores that are most power efficient for the current computing need without negatively impacting the performance. For example, in a smartphone, the low-power small cores can take care of simple tasks — such as email client, web browsing — saving energy, while the complex cores have to be switched on for compute-intensive tasks — such as 3D gaming, browsing flash-based websites — sacrificing energy. The GPU is responsible for graphics as well as general-purpose applications with abundant parallelism such as image processing. The FPGAs

can accelerate computations with substantial parallelism and irregular control flow. However, the lack of software support puts substantial burden on the programmer.

In particular, given an application specification in high-level programming language, it is challenging to map the application on a heterogeneous multi-core. The mapping involves (a) selecting appropriate core type for each compute-intensive kernel, (b) translating and optimizing the kernel specification in a high-level language to an implementation suitable for the selected core type, (c) partitioning the application kernels or the data among multiple core types for co-execution with improved performance or energy-efficiency. The mapping process can be performed statically at compile time or adapted at runtime. The runtime support requires voltage-frequency settings for the different cores such that the aggregate power consumption of the chip does not exceed the power budget and still enables the application to achieve the best possible performance [11]. Unfortunately, the emergence of heterogeneous multi-cores is not well matched with equal advancements in software support for such complex systems. The complexity of the system with different cores supporting different programming languages and/or Application Programming Interface (API) makes software development on embedded mobile processors a challenging and sometime daunting endeavor. Thus the software deployed on embedded mobile processors routinely reaches only a fraction of the expected performance promised by the cutting-edge hardware in mobile SoCs.

In this paper, we present several techniques that have been proposed to build the software support for heterogeneous computing in consumer devices. As will be shown in the following sections, software support is necessary both at compile time as well as runtime to sufficiently exploit the diverse set of processing cores on modern mobile SoCs. In the rest of the paper, Section II discusses several compile-time strategies, while Section III presents state-of-the-art runtime power and thermal management efforts followed by a comprehensive runtime approach for thermal management on mobile SoCs. Section IV concludes this paper with a brief overview of outstanding issues for future work.

II. Software Support at Compile-Time

The diverse set of heterogeneous processing cores enable designers to achieve decent performance within a stringent power/thermal budget requirement. However, the delicate choice of a certain core type or a combination of several cores for a given application is, most of the time, not obvious. Additionally, the reference code for an application is usually specified in a high-level single-threaded programming language such as C. It requires knowledge of the different cores as well as time and effort to redevelop the application in the accelerator-specific languages. If the performance of the application on different cores can be made available at an early stage, application developers can make an informed decision regarding which core or combinations to use and they can then concentrate on platform-specific languages and optimizations.

In the following subsections, we present several tools that can help designers to make such compile time choices of the execution configuration. Lin-Analyzer [12], MPSeeker [13] and CGPredict [14] represent cross-platform tools that predict the performance of an application on an FPGA and a GPU respectively, from high level C code, that ease the effort and time consuming re-development process for the accelerator-specific languages. An OpenCL partitioning [15] work takes this one step further by accurately predicting co-execution performance from individual execution performances on a CPU-GPU heterogeneous platform.

A. State-of-the-art Techniques

High-level synthesis (HLS) tools [16], which abstract the programming effort above register-transfer level (RTL), are commonly used in application development on FPGAs. However, invoking HLS induces significant design space exploration (DSE) time overheads [12]. In order to rapidly explore the design space to find the execution time achievable on a certain FPGA, the estimation of FPGA performance and area requirement is essential. Starting from high-level specifications (in C/C++), [17] [18] exploit the fine-grain parallelism by accelerating the kernel on single PE. Static analysis are often used, which suffer from inherently conservative dependence analysis. In contrast, Lin-Analyzer [12] and MPSeeker [13] work with dynamic traces, while MPSeeker exploits both coarse and fine-grained parallelism.

On the other hand, modern GPUs present a highly multi-threaded architecture that enables concurrent execution of thousands of threads, which makes the prediction of performance from a single-threaded code a challenging problem. In addition, with more architectural improvements to boost the GPU performance, legacy analytical performance models [19] [20] are not compatible with up-to-date GPU architectures especially because of the complex memory hierarchy. Machine learning models [21] require extensive profiling and delicate choice of the training benchmarks. In addition, the performance of the GPU can be improved largely by platform specific optimizations. Such possibilities are not transparent in the machine learning models.

B. FPGA Performance Estimation

MPSeeker [13] (which builds on Lin-Analyzer [12]) is a high-level analysis framework that considers both fine- and coarse-grained parallelism on FPGAs to estimate accelerator performance and resource requirement from sequential C/C++ code without invoking an HLS tool. The framework is open source and available from https://github.com/zhguanw/lin-analyzer

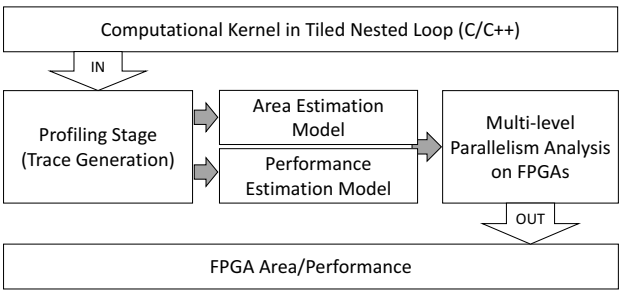


Figure 2: MPSeeker: An Automatic Design Space Exploration framework with Multi-level Parallelism for FPGA

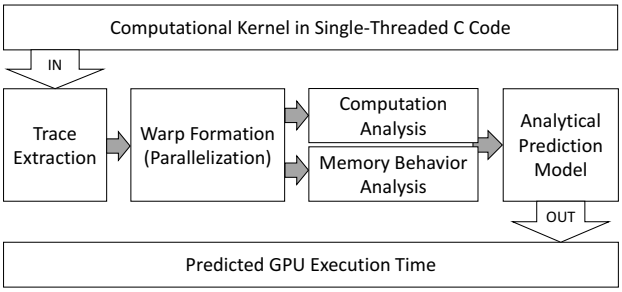


Figure 3: CGPredict: An Analytical framework for Performance Estimation from Single-threaded C Code for GPU

As shown in Figure 2, MPSeeker takes a high-level specification (C/C++) of an algorithm in the form of nested loops, available pragmas for optimization and resource constraints as inputs. A dynamic trace is generated in the *Profiling Stage*, leveraging the intermediate representation of Low-Level Virtual Machine (LLVM IR) [22] and IR instrumentation. The *Performance Estimation Model* estimates the kernel computation cycles through dynamic data dependence graph (DDDG) scheduling [12] as well as data communication cost. A machine learning technique called Gradient Boosted Machine (GBM) is employed to predict the resource usage in *Area Estimation Model*. The DSE step is finally performed by estimating both performance and area for each pragma combination to recommend the best configuration in *Multi-level Parallelism Analysis*.

MPSeeker achieves less than 5.2% error in performance prediction compared to Vivado HLS and on average 15% error in resource estimation. It recommends the same or closely similar combination of pragma settings in minutes instead of hours or sometimes even days taken by exhaustive HLS-based techniques. In addition, MPSeeker identifies bottlenecks of different FPGA implementations when applying diverse optimizations, assists designers in evaluating different architectural options in the context of high-level synthesis and better understand the performance impact of different accelerator design choices.

C. GPU Performance Estimation

CGPredict [14] is an analytical framework to estimate the performance of a computational kernel on an embedded GPU architecture from unoptimized, single-threaded C code.

CGPredict builds the performance model from a dynamic execution trace of the sequential kernel. The trace is modified to expose the available thread-level parallelism that can be potentially exploited by the GPU. At the same time, the memory access trace is analyzed against a performance model of the memory hierarchy that captures the interaction between the cache, the DRAM memory, and the inherent memory latency hiding capability of the GPU through zero-cost context switching of the threads when necessary. As shown in Figure 3, CGPredict takes a computational kernel in the form of single-threaded C code as input similar to MPSeeker and generates its execution trace through a *Trace Extraction* phase. The trace obtained is a serial trace that captures the execution information of the application. To emulate the behavior of a GPU, a *Warp Formation* phase is introduced to transform the single-threaded trace into its multi-threaded equivalent. CGPredict then extracts computation (in the form of compute instructions) and memory access information. Compute instructions are mapped to CUDA PTX ISA [23] to predict the number of GPU instructions, and thus compute cycles in *Computation Analysis* stage. To predict GPU memory cycles, CGPredict takes the memory access information and analyzes its access patterns and cache behavior in *Memory Behavior Analysis* stage. The results from the two analysis stages complete the execution characteristics required from the kernel for performance prediction. Lastly, together with the architectural parameters obtained by micro-benchmarking, an *Analytical Prediction Model* is engaged to predict the final execution performance using the computation and memory execution characteristics.

CGPredict can estimate the performance from C code with 9% estimation error compared to the performance of the corresponding native CUDA code on an embedded NVIDIA Kepler GPU averaged across a number of kernels. As CGPredict is based on analytical modeling, it can provide insights regarding the characteristics of the kernel and the GPU that influence performance, including coalescing of memory accesses or shared memory usage. These insights offer opportunities for the programmers to understand the intrinsic strengths and weakness of the architecture in the context of a particular kernel that can facilitate further code optimizations.

D. Choice of Accelerator

With the help of MPSeeker [13] and CGPredict [14], we can predict the performance of a computational kernel in C code on the FPGA and GPU platform respectively, and therefore assist the designers in selecting the appropriate accelerator that gives the best performance. Here we show a set of experiments with five benchmarks. DCT1D and MM

Table I: Accelerator Choice between GPU and FPGA

Benchmark Name	Input Size	GPU time (ms)		FPGA time (ms)	
		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**

are taken from [24], while DERICHE1, GEMVER1 and MVT are taken from Polybench [25]. These benchmarks are mainly matrix calculations implemented with multiple nested loops. To verify with the actual performance of the kernels on the accelerators, equivalent CUDA code are implemented manually. The verifications are carried out on the NVIDIA embedded Kelper GPU on Jetson TK1 development board [7] (852MHz) and Xilinx ZC702 embedded FPGA [24] (100MHz).

Table I shows that CGPredict together with MPSeeker can suggest the correct accelerator (GPU or FPGA) for each application. The choice among different accelerators is a complex confluence of the considerations of application characteristics and architecture specifications.

For benchmark MM, GEMVER1 and DERICHE1, GPU is a better choice than FPGA. For the accelerators used in the experiment, GPU has much higher frequency (852MHz) and memory bandwidth (17GB/s) compared to FPGA (100MHz, 4GB/s). The better processing capability makes GPU a better choice for these three benchmarks. In addition, the three benchmarks are analyzed to have coalesced memory access pattern, which significantly reduces memory transactions of GPU implementations and improves performance.

For MVT and DCT1D, FPGA is a better choice compared to the GPU. Both MVT and DCT1D have uncoalesced memory access patterns, which cause the GPU to suffer from extensive memory transactions. Different from the GPU implementations, the FPGA accelerator first loads input data of several tiles into its local memory and start the computation. Therefore memory access patterns do not have large impact on FPGA performance, as access latency of FPGA local memory is quite small. It should be noted that GPU performance could be improved by several optimizations such as data layout transformation, loop tiling with shared memory and vectorization. However, the reference CUDA code that we are comparing against do not include such optimizations and hence we refrain from using them.

In general, with the compile time performance prediction tools, the designers are able to know the better choice of core type for a certain computational kernel. With the analytical nature of the models, designers can further optimize the implementation to achieve better performance.

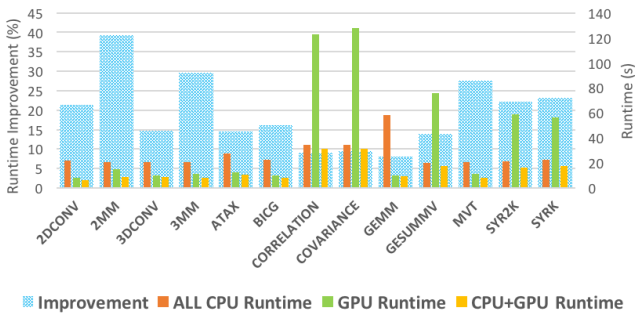


Figure 4: Benefit of Concurrent Execution of CPU and GPU

E. Partitioned Co-execution

With the help of cross-platform languages, such as OpenCL [26], an application, without any modification, can be executed on multiple platforms including CPUs, GPUs, FPGAs and others. While some applications benefit from the execution on GPUs, others may be more suited for FPGA execution, or just on CPUs without any accelerator. On top of functional heterogeneity as discussed in the previous sections, we can additionally exploit the performance heterogeneity available on modern MPSoCs, such as the ARM big.LITTLE platform [26]. A co-execution on all available components of the MPSoC can therefore engage all the cores and achieve better execution performance.

We present such an approach in [15] that statically partitions the application to run across CPU and GPU clusters on mobile application processors in conjunction with appropriate voltage-frequency setting for each core cluster (small core, big core and GPU), in order to maximize the power-performance trade-off of the application. As shown in Figure 4, among all the benchmarks, the improvement in runtime by executing CPU and GPU concurrently can be as high as 39.4% (2MM) and on average 19.2%. With the high margin of execution improvement, the designers can have more knobs to adjust in order to achieve better performance under certain constraints. This work in addition shows that designers can achieve the best performance-energy trade-off (using the metric of interest ED^2) with appropriate voltage-frequency settings.

III. Runtime Support for Dynamic Power and Thermal Management

In the previous section, we discussed several works that enable designers to predict the performance and in some cases even power consumption, of the various kernels in an application on different computation units such as GPU, FPGA and big.LITTLE CPU cores. These early estimates allow system designers to select the right processing core for each of the kernels in an application to achieve high performance in an energy-efficient manner on the heterogeneous platform.

On the other hand, a runtime manager has also been proposed in the existing literature to leverage the Dynamic Voltage Frequency Scaling (DVFS) features of the various processing cores in a heterogeneous platform to further reduce power consumption during runtime. This is especially useful in consumer devices such as smartphones and smartwatches that are constantly held or worn by users. While solutions that target energy-efficient execution ensure long battery life for portable battery-operated mobile devices, techniques to ensure power-efficient execution are important to ascertain tolerable temperature while using these devices.

A. State-of-the-art Techniques

The ever rising demand for higher performance from mobile devices has forced designers and manufacturers to augment increasing number and types of computation units. This inevitably leads to higher power consumption in such devices that needs to be carefully managed to ensure power efficient execution and tolerable temperature while being used [27]. This problem is further exacerbated when the thermal behavior of the system must be maintained during runtime while ensuring high user experience.

Dynamic voltage frequency scaling (DVFS) is a popular technique that has been used extensively for power management. Several rule based governors are included in operating systems for mobile devices to manage, according to various criteria (e.g., CPU usage), the frequency of CPU, GPU, etc. [28]. For GPUs, the authors in [29] introduced a QoS-aware DVFS technique for energy-efficient execution while achieving a higher QoS satisfaction rate than the default on-demand DVFS policy. However, these governors do not directly focus on thermal management. For CPU power and thermal management, we have earlier proposed control-theoretic [30] and price-theory based approaches [31] for task migration and DVFS as well as approximation-aware scheduling and DVFS [32] across ARM big.LITTLE clusters. The task migration requires cross-core performance estimation [33] to perform cost-benefit analysis of the migration.

Mobile gaming is one of the most frequently used applications on mobile devices. They are also one of the most power hungry applications, which has led numerous researchers to explore various power management techniques for mobile gaming. Authors in [34] categorized several 3D mobile games based on their performance and power characteristics. In [35] and [36], we studied several mobile games and proposed a DVFS-based power management approach with options for power-performance tradeoff. However, these works did not focus on thermal management.

The authors in [37] and more recently in [38] target dynamic thermal management on mobile devices while considering the temperature coupling between the processor and the battery. However, these works do not consider the interaction between the CPU and the integrated GPU that directly impacts the application performance, for on-chip thermal management. The authors in [39] focused on thermal management for mutli-core processors while considering the thermal coupling between adjacent cores, however, they did not consider heterogeneous processing cores like the ones found in mobile devices.

A recent work by Singla et al. [40] proposed a predictive approach for thermal management of CPU and GPU in mobile platforms. The main idea in this paper is to proactively turn off the high power consuming Cortex A15 cores, one by one, if a thermal throttling is predicted by their algorithm. The A7 cores and GPU is throttled next, if turning off the A15 cores does not sufficiently reduce the overall temperature. While their results show significant reduction in on-chip temperature variance, this comes at the cost of reduced performance especially for gaming workloads that require both CPU and GPU. In contrast, we propose a PID-controller based cooperative CPU-GPU thermal management technique in [41] that significantly reduces the on-chip temperature variance and achieves better performance than the default Linux governor while maintaining similar peak temperature. The next subsection briefly presents this work.

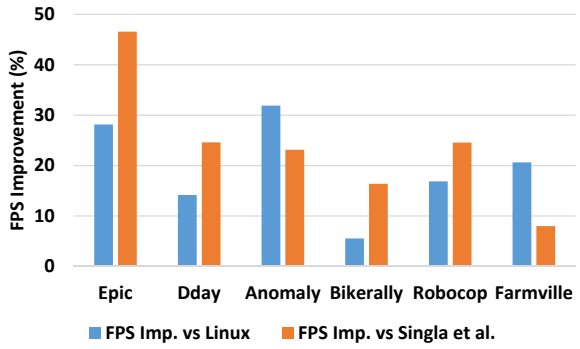
B. Cooperative Dynamic Thermal Management

In this work, we used the Arndale development platform [42] with a Samsung Exynos 5250 heterogeneous MPSoC. This SoC consists of a dual core ARM Cortex-A15 processor and a high performance Mali T604 quad-core GPU. In the first step, we made two observations from the thermal profile of CPU and GPU by running an intensive mobile game that stressed both the CPU and GPU on this platform.

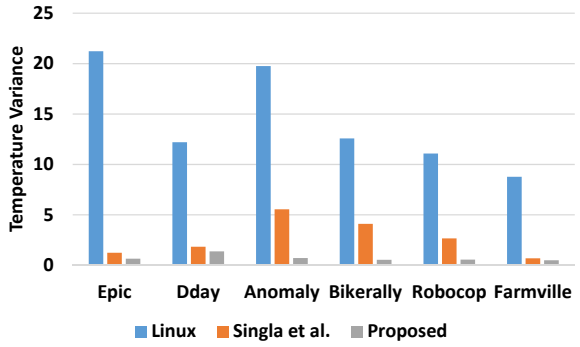
As the temperature of the SoC rises during the gameplay, the default Linux thermal management unit (TMU) monitors the CPU temperature and throttles the CPU frequency once the temperatures rises to a pre-defined threshold (for example, 70°C in this platform). The GPU frequency is throttled at a later 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.

Secondly, different from 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 known as the *thermal coupling* phenomenon that makes it difficult to predict the thermal behavior of one processing core in isolation [43].

Considering these two observations, we proposed a control theory based cooperative CPU-GPU Dynamic Thermal Management (DTM) technique [41]. The proposed technique attempts to maintain the overall chip temperature at or close to a pre-defined threshold by estimating and



(a) Performance improvement of cooperative solution versus other approaches



(b) Temperature Variance Reduction

Figure 5: Improvement in FPS and Temperature Variance.

allocating the available thermal headroom individually to both CPU and GPU based on their utilization values. Appropriate temperature models were derived to predict the CPU and GPU temperatures at a target frequency (to maintain performance), while also considering the temperature of the other processing core. The predicted temperature values were fed to a "Temperature Allocator" block that calculates the reference temperatures for both CPU and GPU. These reference temperatures were then sent to individual PID-controllers to obtain the frequency of CPU and GPU that achieved the desired performance without violating the thermal constraints. The resulting temperature of both CPU and GPU were measured after a fixed period and the entire process was repeated.

We implemented the proposed DTM algorithm as well as the technique proposed by Singla et al. [40] on a real mobile development platform, the Arndale development board and used it to control the frequency settings of CPU and GPU while running several gaming workloads. The resulting performance (frames per second metric for gaming workloads) and the on-chip temperature were observed. As seen from figure 5(a) and 5(b), the proposed algorithm not only reduced the on-chip temperature variance but also achieved higher performance than the default Linux TMU and the approach in [40].

IV. Conclusion and Future Outlook

In this paper, we presented our recent efforts to improve the software support for heterogeneous computing in mobile SoCs. The compile time strategies help to estimate the performance of application kernels on different processing cores such as big.LITTLE CPU, GPU and FPGA. Such tools enable system designers to make early stage decision on executing the kernel on the most suitable processing core in an effort to achieve best performance while also ensuring high energy-efficiency. On the other hand, the runtime strategies are mainly useful for dynamic power and thermal management in such devices without sacrificing performance.

The research efforts presented in this paper are certainly concrete steps in the right direction. However, more work needs to be done in future to make heterogeneous computing truly seamless for application programmers. While the compile time strategies discussed in Section II can be used to predict performance on the various processing cores, an integrated compiler support is needed to compile and map different kernels in any application on a given heterogeneous processing platform that consists of performance-heterogeneous CPU cores and functionally heterogeneous cores such as GPU and FPGA. Support for other processing cores such as DSP and ISP, etc. also needs to be added to this integrated compilation framework to make a truly seamless application mapping process. Additionally, for applications with highly input-dependent behavior, a dynamic solution to partition and schedule the workload on the heterogeneous cores can be more appropriate compared to the static partitioning strategy for co-execution discussed in this paper. As an example, we employ runtime work stealing approach to distribute workloads for the Convolutional Neural Networks inference engines to the CPU cores with vector processing units (ARM Neon) and processing elements in FPGAs for improved load balancing [44].

Finally, while the works presented here mostly consider gaming applications for runtime power and thermal management, GPGPU applications such as image processing, machine learning applications using deep neural networks, etc. also need to be considered for power and thermal efficient execution. Such applications are increasingly being ported to mobile devices and demand both high performance and power efficiency from these platforms. Apart from DVFS based solutions, runtime task migration strategies also need to be developed to ensure that no single processing core is over-worked while other cores are idle and thereby improve the overall system reliability [45].

V. Acknowledgments

This work was partially funded by Singapore Ministry of Education Academic Research Fund Tier 2 MOE2015-T2-2-088.

REFERENCES

[1] "Qualcomm Snapdragon 845," https://goo.gl/3eDr47.
[2] "Huawei Hisilicon Kirin 970," https://goo.gl/zXnjfD.
[3] "Samsung Exynos 9810," http://www.samsung.com/exynos/.
[4] "Apple A11 Bionic," https://en.wikichip.org/wiki/apple/ax/a11.
[5] T. Mitra, "Heterogeneous multi-core architectures," *Information and Media Technologies*, 2015.
[6] "Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM big.LITTLE Technology," http://goo.gl/UVAXVS.
[7] NVIDIA, "NVIDIA Tegra K1: A New Era in Mobile Computing," *White Paper*, 2014.
[8] "ARM Mali-T600 Series GPU OpenCL, Version 2.0, Developer Guide," http://goo.gl/R0FKs8.
[9] "Qualcomm Adreno," https://goo.gl/FU8rts.
[10] L. Codrescu, W. Anderson, S. Venkumanhanti, M. Zeng, E. Plondke, C. Koob, A. Ingle, C. Tabony, and R. Maule, "Hexagon DSP: An Architecture Optimized for Mobile Multimedia and Communications," *IEEE Micro*, 2014.
[11] M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel, "Dark silicon as a challenge for hardware/software co-design," in *CODES+ISSS*, 2014.
[12] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-Analyzer: A high-level performance analysis tool for FPGA-based accelerators," in *DAC*, 2016.
[13] 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 *DATE*, 2017.
[14] S. Wang, G. Zhong, and T. Mitra, "CGPredict: Embedded GPU Performance Estimation from Single-Threaded Applications," *ACM Transactions on Embedded Computing Systems (TECS) - Special Issue CODES+ISSS*, 2017.
[15] A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra, "Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms," in *ICCD*, 2015.
[16] Xilinx Inc., "Xilinx vivado high-level synthesis," 2014, https://goo.gl/2nvniX.
[17] X. Gao, J. Wickerson, and G. A. Constantinides, "Automatically Optimizing the Latency, Area, and Accuracy of C Programs for High-Level Synthesis," in *FPGA*, 2016.
[18] P. Li, P. Zhang, L.-N. Pouchet, and J. Cong, "Resource-Aware Throughput Optimization for High-Level Synthesis," in *FPGA*, 2015.
[19] S. Hong and H. Kim, "An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness," in *ISCA*, 2009.
[20] A. K. Parakh, M. Balakrishnan, and K. Paul, "Performance Estimation of GPUs with Cache," in *IPDPSW*, 2012.
[21] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance," in *MICRO*, 2015.
[22] C. Lattner and V. Adve, "LLVM: a Compilation Framework for Lifelong Program Analysis Transformation," in *CGO*, 2004.
[23] NVIDIA, "Parallel Thread Execution ISA Version 5.0," http://docs.nvidia.com/cuda/parallel-thread-execution.
[24] Xilinx Inc., "ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC," https://goo.gl/bAhHQY.
[25] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a High-level Language Targeted to GPU Codes," in *InPar*, 2012.
[26] Khronos Group, "OpenCL: The open standard for parallel programming of heterogeneous systems." https://www.khronos.org/opencl/.
[27] K. Sekar, "Power and Thermal Challenges in Mobile Devices," in *MobiCom*, 2013.
[28] "Android Governors," http://goo.gl/8j1Eqo.
[29] D. You and K.-S. Chung, "Quality of service-aware dynamic voltage and frequency scaling for embedded GPUs," *IEEE Computer Architecture Letters*, 2014.
[30] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *DAC*, 2013.
[31] T. S. Muthukaruppan, A. Pathania, and T. Mitra, "Price theory based power management for heterogeneous multi-cores," *ACM SIGPLAN Notices*, 2014.
[32] C. Tan, T. S. Muthukaruppan, T. Mitra, and L. Ju, "Approximation-aware scheduling on heterogeneous multi-core architectures," in *ASP-DAC*, 2015.
[33] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *CASES*, 2013.
[34] X. Ma, Z. Deng, M. Dong, and L. Zhong, "Characterizing the Performance and Power Consumption of 3D Mobile Games," *Computer*, 2013.
[35] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU power management for 3D mobile games," in *DAC*, 2014.
[36] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs," in *DAC*, 2015.
[37] Q. Xie, J. Kim, Y. Wang, D. Shin, N. Chang, and M. Pedram, "Dynamic Thermal Management in Mobile Devices Considering the Thermal Coupling Between Battery and Application Processor," in *ICCAD*, 2013.
[38] O. Sahin et al and A. K. Coskun, "On the Impacts of Greedy Thermal Management in Mobile Devices," *Embedded Systems Letters*, 2015.
[39] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "Thermal and Energy Management of High-Performance Multicores: Distributed and Self-Calibrating Model-Predictive Controller," *TPDS*, 2013.
[40] G. Singla et al, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *DATE*, 2015.
[41] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving Mobile Gaming Performance Through Cooperative CPU-GPU Thermal Management," in *DAC*, 2016.
[42] "Arndale Board 5250," http://goo.gl/1ZCSNX.
[43] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili, "Cooperative Boosting: Needy Versus Greedy Power Management," in *ISCA*, 2013.
[44] G. Zhong, A. Dubey, T. Cheng, and T. Mitra, "Synergy: A HW/SW Framework for High Throughput CNNs on Embedded Heterogeneous SoC," *arXiv preprint arXiv:1804.00706*, 2018.
[45] C. Bolchini, M. Carminati, T. Mitra, and T. S. Muthukaruppan, "Combined on-line lifetime-energy optimization for asymmetric multicores," in *DFT*, 2016.