New Techniques for Power-Efficient CPU-GPU Processors

by
Kapil Dev

M.S., Rice University, Houston, TX, 2011
B.Tech., MNIT Jaipur, India, 2006

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in School of Engineering at Brown University

PROVIDENCE, RHODE ISLAND

May 2017

This dissertation by Kapil Dev is accepted in its present form
by School of Engineering as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Recommended to the Graduate Council

Date_____ _____

Sherief Reda, Advisor

Date_____ _____

Ruth Iris Bahar, Reader

Date_____ _____

Jacob Rosenstein, Reader

Approved by the Graduate Council

Date_____ _____

Peter M. Weber, Dean of the Graduate School

Vitae

Kapil Dev was born to Smt. Vinod Devi and Sh. Ishwar Singh in a small village, named Dhanasri in Haryana state, in India. He received his undergraduate degree (B. Tech.) in Electronics and Communication Engineering from Malaviya National Institute of Technology (MNIT) Jaipur in 2006. After that he worked as a Design Engineer for Texas Instruments in Bangalore for two years before moving to USA. He earned his Master of Science (MS) degree in Electrical and Computer Engineering from Rice university in 2011 and started his PhD under Prof. Sherief Reda at Brown University in the same year. During his PhD he worked on thermal/power characterization, modeling and management of real CPU-GPU processors. In general, his research interests include exploring architectural designs for CPU, GPU, and FPGA devices and identifying opportunities to improve the performance and energy efficiency of system hardware, ranging from chips to servers to data-centers, under realistic physical constraints such as thermal design power (TDP), energy, and reliability.

kapil_dev@brown.edu
Brown University, RI, USA

Selected Publications:

1. K. Dev, X. Zhan, and S. Reda, "Online Characterization and Mapping of Workloads on CPU-GPU Processors," submitted to IEEE International Symposium on Workload Characterization (IISWC), 2016.

2. K. Dev, S. Reda, I. Paul, W. Huang, and W. Burleson, "Workload-aware Power Gating Design and Run-time Management for Massively Parallel GPGPUs," accepted in IEEE Symposium on Very-Large Scale Integration (ISVLSI), 2016.

3. K. Dev, I. Paul, and W. Huang, "A Framework for Evaluating Promising Power Efficiency Techniques in Future GPUs for HPC," in ACM High Performance Computing Symposium (HPC), 2016.

4. A. Majumdar, G. Wu, K. Dev, J. L. Greathouse, I. Paul, W. Huang, A.-K. Venugopal, L. Piga, C. Freitag, and S. Puthoor, "A Taxonomy of GPGPU Performance Scaling," in IEEE International Symposium on Workload Characterization (IISWC), 2015.

5. K. Dev, G. Woods and S. Reda, "High-throughput TSV Testing and Characterization for 3D Integration Using Thermal Mapping," in Design Automation Conference (DAC), 2013.

6. K. Dev, A. N. Nowroz and S. Reda, "Power Mapping and Modeling of Multi-core Processors," in IEEE International Symposium on Low-Power Electronics and Design (ISLPED), 2013.

Patent:

S. Reda, A. N. Nowroz, and K. Dev, "A Power Mapping and Modeling System for Integrated Circuits," published US Patent, PCT/US 2016/012443 A1, 2013.

Acknowledgements

This thesis would not have been possible without constant support, guidance and inspirations of many grateful individuals. First of all, I want to express my deepest gratitude to my advisor, Prof. Sherief Reda for his immense knowledge, insights, invaluable guidance, and support during my graduate study at Brown University. I would like to thank him for all his thought provoking questions and encouragement throughout my Ph.D. experience.

I am grateful to my dissertation committee members Prof. Iris Bahar and Prof. Jacob Rosenstein for agreeing to serve on the committee and taking time to read and comment on the thesis. Their suggestions and comments were invaluable to this work and helped me in shaping up my final thesis.

I also want to thank AMD Research for giving me an opportunity to do internship for about a year. A significant part of my research on low power design of future general purpose GPUs was conducted at AMD Research Austin, TX. I will always remember my experience of working with great researchers and industry leaders at AMD. I am specially thankful to all of my co-authors and collaborators, including Prof. Wayne Burleson, Dr. Indrani Paul, Dr. Wei Huang, Dr. Joseph Greathouse, Dr. Leonardo Piga, Dr. Yasuko Eckert, Dr. Gene Wu, Dr. Abhinandan Majumdar, A.-K. Venugopal, C. Freitag, and S. Puthoor from AMD.

I would also like to thank my co-authors, Prof. Gary Woods from Rice University, my lab mates Dr. Abdullah Nowroz, Xin Zhan, and of course Prof. Sherief Reda at

Brown. I owe a lot of my research output to their inputs and hard efforts. I also want to thank Patrick Temple and Sriram Jayakumar for their contributions on creating the thermal imaging infrastructure.

I would like to thank my friends and lab mates, Hokchhay Tann, Soheil Hashemi, Reza Azimi, Shuchen Zheng, Ryan Cochran, Kumud Nepal, Onur Ulusel, Marco Donato, and Dimitra Papagiannopoulou. I am happy to share my graduate journey with all of you.

During my PhD studies, I was blessed to have many great friends, including my two wonderful roommates Ravi Kumar and Jay Sheth, team mates from volleyball, and graduate fellows in both engineering and other departments at Brown. I want to thank all of them for making my life fun-filled and memorable at Brown.

Last, but not the least, none of this would have been possible without love, support, encouragement and patience from my parents, sisters, brothers and my entire family. My family provided me good moral and education values throughout my life. Regardless of being from a small village, where institutes for higher education are not easily accessible, their strong belief in education allowed me to move to different cities (both within India and in USA) to get higher education and pursue the PhD program. With the blessings of my parents, I became the first person to pursue a PhD program from my village.

Abstract of "New Techniques for Power-Efficient CPU-GPU Processors" by Kapil Dev, Ph.D., Brown University, May 2017

Power is one of the key challenges for improving the performance of modern CPU-GPU processors. Research efforts are needed at both design-time and run-time of processor to improve its power efficiency (Performance/Watt). To improve the run-time power management, accurate measurement based power models are needed. Further, the power efficiency of CPU-GPU processors for different workloads depends on the type of device they run on and the run-time conditions of the system [e.g., thermal design power (TDP) and existence of other workloads]. So, an online workload characterization and mapping method is needed. Furthermore, for future massively parallel processors, the low power techniques, like power gating (PG) should be evaluated for their potential benefits before going through the cost of implementing them.

This thesis makes the following contributions towards improving the performance and power efficiency of CPU-GPU processors. First, we propose new techniques for post-silicon power mapping and modeling of multi-core processors using infrared imaging and performance counter measurements. Using detailed thermal and power maps, we demonstrate that in contrast to traditional multi-core CPUs heterogeneous processors exhibit higher intertwined behavior for dynamic voltage and frequency scaling (DVFS) and workload scheduling, in terms of their effect on performance, power and temperature. Second, we propose a framework to map workloads on appropriate device of CPU-GPU processors under different static and time-varying workload/system conditions. We implement the scheduler on a real CPU-GPU processor, and using OpenCL benchmarks, we demonstrate up to 24% runtime improvement and 10% energy savings compared to the state-of-the-art scheduling techniques. Third, to improve the performance and power efficiency of future massively parallel GPUs, we provide an integrated solution to manage leakage power by incorporating workload/run-time-awareness into the PG design methodology. On a hypothetical future GPU with 192 compute units, our results show that a PG

granularity of 16 CU per cluster achieves 99% peak run-time performance without the excessive 53% design-time area overhead of per-CU PG. Further, we demonstrate that the incorporation of design-awareness into the run-time power management can maximize the benefits of power gating, and improve the overall power efficiency of future processors by additional 5%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Characterization

Historically, device and technology scaling have helped in improving the performance and *power efficiency* (performance per Watt) of computing devices. In 1965, based on the industry scaling trend at that time, Gordon Moore observed that the number of transistors in an integrated circuits were approximately doubling every 18 months; this empirical trend has remained valid so far and is widely known as Moore's Law [69]. As the transistor gets smaller, it can switch faster while consuming less power. In 1974, to supplement Moore's Law, Dennard *et al.* provided ideal scaling conditions for metal-oxide-semiconductor field-effect transistors (MOSFETs) for achieving simultaneous improvement in transistor density, switching speed and power density [24]. According to the ideal scaling, with the decrease in transistor size, both voltage and current were also decreased in proportion to the length of transistor. Since the power requirement of the chip remained proportional to area, keeping the power density more or less constant. In other words, combined with

Moore's Law, Dennard's scaling implied that power efficiency of circuits would increase at roughly the same rate as transistor density. Hence, the systematic and predictable transistor scaling principles setup a roadmap for semiconductor industry in terms of targets and expectations for coming generations of process technology.

Continuous advances in lithographic techniques and materials have ensured that both Moore's Law and Dennard scaling have been followed by the semiconductor industries for more than three decades. These efforts have even led to multi-core processors providing higher parallel compute capability in the same or smaller chip sizes. However, recently, around 2005 time-frame, voltage scaling has reached its lower limit due to threshold voltage limits and its exponential dependence on sub-threshold leakage power. As a result, reductions in feature size no longer guarantee the performance per watt improvements (power efficiency) [11]. The steady increase in leakage current has not only hurt the power efficiency, but also increased the power density and risk of thermal run-away conditions, beyond the capability of current cooling solutions. As a result, new transistor technologies such as high-dielectrics, metal gates and multiple-gate devices (e.g., FinFETs) have been introduced to keep improving the power efficiency [11]. FinFETs are shown to decrease the leakage power up to $10\times$, however, they also suffer from internal self-heating and thermal issues. Due to device physics, the leakage power could be dominant even in FinFETs because of its exponential dependence on temperature [57, 17, 106]. So, it is essential to manage the leakage power using improved design-time (e.g., power gating) and run-time power management algorithms for improving the power efficiency of highly parallel future processors.

Figure 1.1 shows the evolution of processor architecture over time [94]. In the single-core era, the performance of processors was improved by device scaling and frequency scaling. The dynamic power and complexity of logic were the main bottlenecks in this era. In the post-Dennard era, the clock speed has more or less saturated, so the perfor-

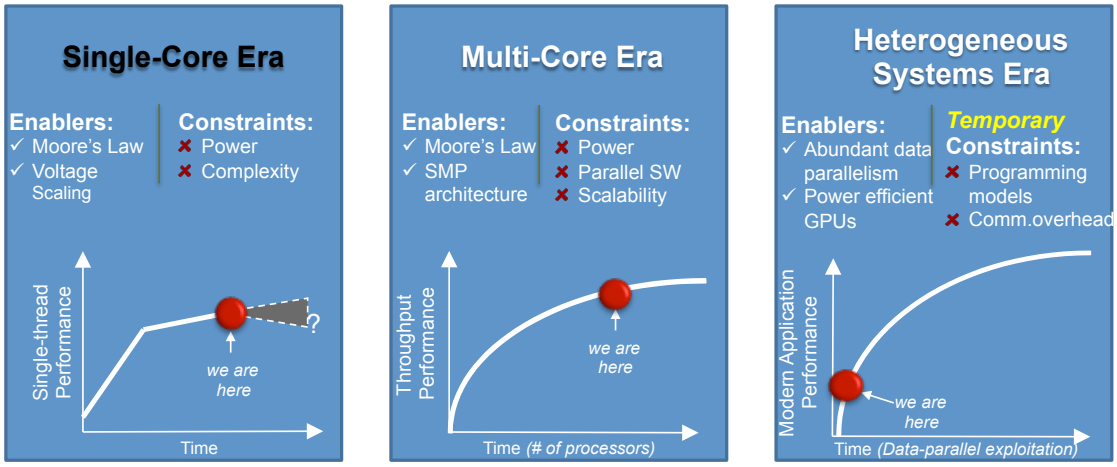

Figure 1.1: Evolution of processor design over time [94].

mance scaling is achieved by improving the power efficiency and energy usage through other techniques, for example use of multi-core CPU-GPU heterogeneous systems. Applications are being parallelized to make effective use of multi-core processors. Typically, different applications and different phases within an application have varying characteristics in terms of amount of parallelism, memory requirement, etc. So, to better meet the varying needs of applications, modern processors are equipped with heterogeneous compute units (e.g., CPUs and GPUs) integrated on the same die. CPU provides better performance for single threaded and highly branch divergent applications, on the other hand, GPU provides better performance and power efficiency for data-parallel applications.

GPUs are available in two forms: discrete cards and integration with CPU cores. While systems with discrete (high-performance and high-power) GPUs are used for getting maximum performance for highly parallel applications, integrated heterogeneous processors offer great balance between performance and power efficiency for a wide range of applications. Both integrated and discrete GPUs have their unique challenges in terms of power efficiency. While integrated GPUs have to share power and thermal budget with the on-die CPU cores, the performance of massively parallel discrete GPUs is limited by leakage power, thermal design power (TDP), and cooling solutions. As emphasized by

Dally, performance scaling depends on how efficiently the TDP is used to perform computations [23]. So, the performance of computing devices in the current heterogeneous era could be defined as follows:

$$\text{Performance}(ops/s) = \text{Power}(W) \times \text{Efficiency}(ops/Joule). \tag{1.1}$$

In other words, power-efficiency (Perf/W) is inversely proportional to energy consumption. Minimizing energy is crucial for both low- and high-TDP devices, so it has become an important metric for across the devices. In this thesis, we come up with techniques that advance the state-of-the-art methods in both experimental and run-time fronts which could improve the power efficiency of current and future processors.

First, to improve the power efficiency of existing processors, one needs to have a setup to make reliable measurements of power and performance when it is running real workloads. The performance could be measured by either in the form of instructions executed per second or by direct measurement of total runtime of applications. On the other hand, measuring power is somewhat challenging. One could use external power meter to measure the total power being used by the processor, but it does not provide information about how much power is being dissipated in different blocks of the processor which is essential for effective power management. For example, in a multi-core processor, one or more cores might be actively running workloads at a time and other cores might be idle, dissipating un-necessary leakage power. Depending on the spatial temperature profile of the chip, leakage power would be different. It is worth mentioning that leakage power does not contribute towards performance, so any technique that reduces leakage power would improve the overall power efficiency of the processor. Post-silicon thermal measurement based power mapping techniques have been proposed to analyze the fine-grained power distribution of the chips [42, 67, 91, 21]. The current techniques typically ignore the thermal profile based leakage power modeling. Also, it is equally important to build block-wise

reliable power models based on real measurements so that power could be estimated in real time for runtime power measurement. Overall, we need a framework that could be used to make measurements on real systems and build reliable power models for different IP blocks of the processor. The prior work does not provide such complete framework. *As one of the contributions of this thesis, we address some of the challenges of the existing techniques and provide a complete framework for post-silicon power mapping of both homogeneous and heterogeneous processors*. The specific contributions of the thesis are listed in section 1.2 of this chapter.

Second, to improve the power efficiency of a heterogeneous system with integrated CPU-GPU devices, it is important that workloads are launched on the appropriate device. Different devices provide different performance and energy for a given workload. Further, as discussed in this thesis, the runtime and energy of workload not only depend on the device but also depend on the runtime conditions of the system. For example, if the processor is running on battery and is in energy saving mode, then a workload can have its best performance on certain device (e.g., GPU), but if the processor is allowed to dissipate higher power (i.e., higher TDP), then the other device (e.g., CPU) could provide higher performance at the cost of higher energy. Similarly, the device that minimizes the performance or energy also depends on the available resources, e.g. number of CPU cores available for scheduling the workload. If some of the cores are being used by other workloads, then the scheduling decision should take that information into account while making appropriate scheduling decisions. Existing techniques either make the device decision statically or do not take dynamically changing system conditions in to account. *Therefore, there is a potential of improving power efficiency of heterogeneous processors by scheduling workloads on appropriate device under time-varying TDP and resource conditions.* In this thesis, we present a framework that provides kernel-level, hardware status-aware runtime/energy minimization scheduling for CPU-GPU processors during run-time.

Third, future systems are likely to incorporate GPUs with hundreds of compute units (CUs) [73]. Emerging trends show that these CUs have to operate under tight power budgets for safe operating temperatures and avoid excessive leakage power or thermal runaway. As a result, not all CUs can always be powered on across all applications due to thermal and power constraints [32]. Further, high-performance computing (HPC) and other workloads show various amounts of parallelism and scalability trends as a function of the number of active CUs. As a result, keeping all CUs active at all times will lead to increased power consumption without necessarily providing performance benefits. Thus, it is necessary to dynamically adjust the number of active CUs, ideally at per-CU granularity, through power gating (PG) mechanisms based on the run-time requirements of workloads. Power gating is a technique in integrated circuit design that significantly reduces leakage power by powering off inactive GPU CUs. However, power gating introduces significant design and verification complexity, and area overheads due to the introduction of header/footer transistors, which if applied liberally in a per-CU manner can either provide no additional value or negate its benefits. Hence, there is a tradeoff between power gating design overheads and its run-time performance and power efficiency benefits. *We argue that it is important that design-time power gating granularity decisions need to be aware of the run-time behavior of the workloads and vice-versa to provide sufficient return on investment (ROI)*. In this thesis, we develop an integrated approach towards addressing power gating challenges in future GPUs.

1.2 Major Contributions of This Thesis

1. **New Techniques for Post-silicon Power Mapping and Modeling of Processors:** In this thesis (chapter 3), we propose new techniques for post-silicon power mapping and modeling of multi-core processors using infrared imaging and performance

counter measurements [25]. We devise a novel, accurate finite-element modeling (FEM) framework to capture the relationship between temperature and power, while compensating for the artifacts introduced from substituting traditional heat removal mechanisms with oil-based infrared-transparent cooling mechanisms. Furthermore, we decompose the per-block power consumption into leakage and dynamic using a novel thermal conditioning method. Using the leakage power models, we develop a method to analyze within-die leakage spatial variations. We also relate the actual power consumption of different blocks to the performance monitoring counter (PMC) measurements using empirical models. Our total estimated power through infrared-based mapping on a quad-core processor achieve very close results with an average absolute error of 1.07 W of the measured power. Further, we use infrared imaging to obtain detailed thermal and power maps of a heterogeneous processor. First, we show that the new parallel programming paradigms (e.g. OpenCL) for CPU-GPU processors create a tighter coupling between the workload and thermal/power management unit or the operating system. We demonstrate that in contrast to traditional multi-core CPUs heterogeneous processors exhibit higher intertwined behavior for dynamic voltage and frequency scaling (DVFS) and workload scheduling, in terms of their effect on performance, power and temperature. Further, by using the floorplan information of the processor to launch a workload on GPU from an appropriate CPU-core, one can reduce both, the peak temperature (by 11 °C) and the leakage power (by 4 W) of the chip. The findings presented in the thesis can be used to improve performance and power efficiency of both multi-core CPU and CPU-GPU heterogenous processors.

2. **New Techniques for Online Characterization and Mapping of Workloads on CPU-GPU Processors:** Modern CPU-GPU processors allow us to run workloads on both CPU and GPU devices simultaneously. In this thesis (chapter 4), we demonstrate that the runtime and energy of a workload not only depend on the type of

device we run it on, but also depend on the run-time conditions, e.g. thermal design power (TDP) budget of the processor and the number of cores available to the workload [29]. Furthermore, even under a static system environment, different parallel kernels within an application can differ in their appropriate scheduling decisions. To exploit these observations, we propose techniques to map workloads on appropriate device under following static and time-varying workload/system conditions: 1) considering each kernel of an application separately, 2) modeling the effect of TDP on workload scheduling, 3) considering the effect of available resources, in particular number of CPU cores on scheduling. To achieve performance and/or energy-efficient scheduling in a dynamic system environment, we characterize each kernel workload online to consider the run-time resource conditions. Further, using learning models that are trained off-line from carefully selected performance counter data, our framework uses a computationally light-weight support vector machine (SVM) to dynamically map individual kernels during run-time on CPU or GPU to minimize total runtime or energy of the system. The scheduler takes into account the time-varying TDP budget and use of one or more CPU-cores by other workloads in to account while making the scheduling decisions. We implement the scheduler on a real CPU-GPU processor, and using OpenCL benchmarks, we demonstrate up to 24% runtime improvement and 10% energy savings compared to the state-of-the-art scheduling techniques.

3. **New Techniques for Implementing Power Gating on Massively Parallel Future GPUs:** Future graphics processing units (GPUs) will likely feature hundreds of compute units (CUs) and be power constrained, which leads to serious challenges to existing power gating methodologies. In this thesis (chapter 5), we propose design-time and run-time techniques to effectively implement power gating in future GPUs [27, 26]. Based on industrial models and measurement facilities, we show that designers must consider run-time parallelism within potential target workloads

while implementing power gating designs. This will lead to improvements in performance and power efficiency while minimizing design overheads. Furthermore, we show that design awareness during run-time power management can optimally leverage power gating with frequency boosting. By scaling measurements from a recent AMD GPU to a potential future 10 nm technology node, we analyze the impact of PG granularity on performance and power efficiency of a broad and representative set of HPC/GPU applications. Our results show that a PG granularity of 16 CU per cluster achieves 99% peak run-time performance without the excessive 53% design-time area overhead of per-CU power gating. We also demonstrate that a run-time power management algorithm that is aware of the PG design granularity leads to up to 18% additional performance under thermal-design power constraints. Moreover, the analysis presented in the paper is applicable to other massively parallel system architectures as well.

The remainder of this thesis is organized as follows. Chapter 2 presents the required background for power modeling in ICs, challenges in both pre-silicon and post-silicon power modeling, and related works on power mapping, workload scheduling and low-power design techniques for CPU-GPU processors. Chapter 3 presents our framework for post-silicon power mapping of both homogeneous multi-core CPU and heterogeneous CPU-GPU processors using infrared emissions. In Chapter 4, we provide the detailed description of proposed online workload characterization and mapping on CPU-GPU processors. The benefits of workload-aware power gating design and design-aware run-time power management algorithm for future massively parallel GPUs are described in chapter 5. Finally, in Chapter 6, we summarize our findings and outline directions for possible research extensions based on this thesis.

Chapter 2

Background

2.1 Basics of Power Consumption

Power consumption of a chip could be broken down in to two components: dynamic and leakage power dissipation. The dynamic power is consumed due to switching activity of transistors and interconnects. It increases with increase in frequency and operating voltage of the circuit. Further, it also depends on the effective load capacitance of logic circuits. Formally, the dynamic power of a circuit is given by

$$P_{dyn} = \frac{1}{2}\alpha C_{eff} V_{dd}^2 f, \tag{2.1}$$

where V_{dd} is the power supply voltage, f is the operating frequency, α is the switching activity factor, and C_{eff} denotes the effective load capacitance of the switching transistors. Typically, any increase in operating frequency of a logic circuit requires corresponding increasing in voltage for faster switching of transistors. So, voltage V and frequency f are the two dominant factors of dynamic power. For the same reason, all modern processors

have built-in dynamic voltage and frequency scaling (DVFS) features in them that control the frequency and voltage of the processor based on different workload conditions. When the workload activity is high, the processor increases the frequency; otherwise, it keeps the frequency and voltage low to save power. The processor also keeps the voltage and frequency below the maximum operating limits so that the power density and the temperature of the chip do not exceed the safe limits for a given cooling solution.

The other component of power consumption in processor is static leakage power. The leakage power is dissipated when the processor is idle and when there is no switching activity in the circuit. The dominant component of leakage power (also called sub-threshold leakage) has exponential dependence on threshold voltage and the temperature. In particular, the sub-threshold leakage current is given by:

$$I_{lkg} = I_0 e^{\frac{-qV_{th}}{nkT}}, \tag{2.2}$$

where, I_0 is a constant that depends on the transistor's geometrical dimensions and process technology, n is a number greater than one, q is the electrical carrier charge, k is the Boltzmann constant, and T is the junction temperature of the transistor. The leakage power ($V_{dd} \times I_{lkg}$) is sensitive to variations in supply voltage, threshold voltage and temperature induced variabilities. Furthermore, the inherent statistical fluctuations in nanoscale manufacturing have increased within-die process variability, which impacts the leakage profile of the die. Aggressive device scaling in sub-100 nm technologies has increased the contribution of leakage power to the total processor power. New transistor technologies (e.g., FinFETs) have been introduced in below 20 nm process node to mitigate the sub-threshold leakage power. However, FinFET devices suffer from self-heating and are prone to thermal runaway due to confinement of the channel, surrounded by silicon dioxide, which happens to have lower thermal conductivity compared to bulk silicon [17]. Further, the International Technology Roadmap for Semiconductors (ITRS) predicted that

the sub threshold leakage ceiling for FinFET will be comparable to planar bulk MOSFETs [57, 106]. Hence, in future massively parallel processors (e.g., GPUs), leakage power can still be a significant contributor if all compute units are left powered on and idle at high temperatures. In summary, it is essential to reduce the leakage power and use the power effectively to maximize the performance and power efficiency of processors.

2.2 Heterogeneous Computing and OpenCL Paradigm

Heterogeneous computing involves the use of different types of processing units for computation. A computation unit can be a general-purpose processing unit (CPU), a graphics processing unit (GPU), or a special-purpose processing unit [e.g., digital signal processor (DSP), field programmable gate array (FPGA), etc.]. In the past, CPUs were used for general purpose applications and GPUs were mainly used for graphics applications. Recently, increasing number of applications are being parallelized to leverage the parallel compute power of GPUs. GPUs are optimized for highly parallel applications. As a result, they are becoming increasingly popular for general purpose applications. Further, with modern applications requiring interactions with various types of sensors and systems (e.g., networks, audios, videos, etc.), applications have different phases optimized for different systems. Thus integration of CPU with other devices, viz. GPU, FPGA, and DSP, has become a reality and hence, we have entered in the heterogeneous computing era.

Programming different devices in a heterogeneous system typically involved using vendor-specific APIs and languages and vice-versa. For example, NVIDIA's CUDA (short for Compute Unified Device Architecture) platform was compatible with GPUs from only NVIDIA [79]. In an effort to establish an open, royalty-free standard for cross-platform, parallel programming of heterogeneous systems, in June 2008, different industries (Apple,

AMD, Intel, NVIDIA, IBM, to name a few) came together to form the Khronos Compute Working Group [75]. Apple submitted the initial proposal to the Khronos Group of its internally developed OpenCL (Open Computing Language) to the Khronos Group. After reviews and approvals from different CPU, GPU, embedded processors, and software companies, the first revision of OpenCL 1.0 was released in December 2008. Since then OpenCL has been maintained and refined by the Khronos Group.

The main benefits of OpenCL framework are two folds. First, it allows users to consider all computational resources, such as multi-core CPUs, GPUs, FPGAs, etc. as peer computational units and correspondingly allocate different levels of memory, taking advantage of the resources available in the system. Hence, it provides a substantial acceleration in parallel processing. Second, OpenCL provides software portability across different vendors. It allows the developers to divide the computing problems into mix of concurrent subsets to run on devices from different vendors without having to rewrite the application. Recently, NVIDIA has also extended its CUDA to support OpenCL. In this thesis, we use benchmarks written in OpenCL to run on CPU-GPU processors.

OpenCL Platform and Execution Models. The OpenCL programming language is based on the ISO C99 specification with some extensions and restrictions. In its platform model, it is assumed that a host is connected to one or more OpenCL devices [3]. Host is typically a CPU and the devices could be GPU, FPGA, DSP, or the CPU itself. Each device may have multiple compute units, each of which have multiple processing elements (PEs). Figure 2.1 depicts the OpenCL platform model pictorially. Further, the execution model of OpenCL comprises two components: kernels and host applications. Functions executed on an OpenCL devices are called “kernels”. They are the basic unit of executable code which can run on one or more PEs of the device depending on the amount of parallel work assigned by the the host application.

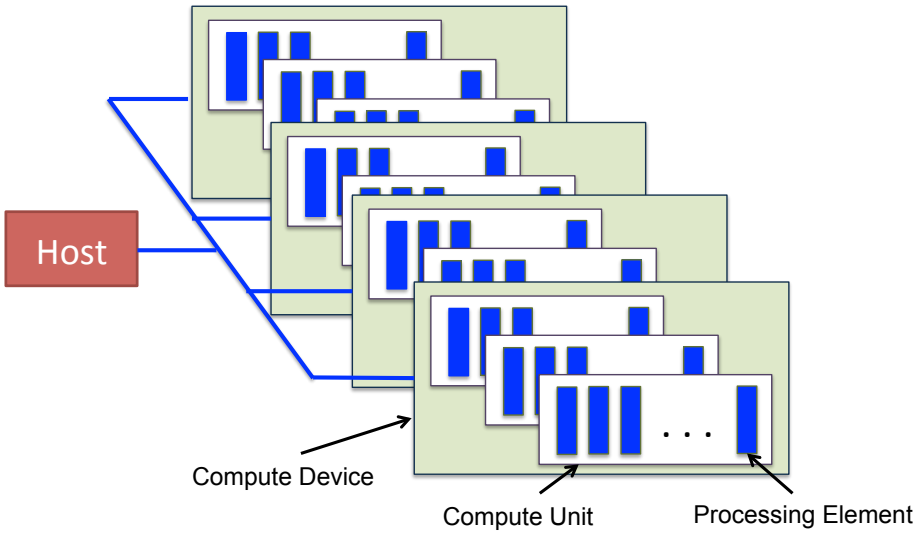

Figure 2.1: OpenCL platform model.

Figure 2.2 shows the execution of an OpenCL application on the OpenCL platform model. The host application is divided into two parts: serial code, which runs only on the host (CPU) and the parallel code corresponding to one or more kernels, which can run on CPU, GPU, or any other OpenCL device. The sequential part of the host program defines devices' context and queues kernel execution instances using command queues. For devices from the same vendor, all devices could be grouped in to single context, but there has to be a separate command queue for each device to launch kernel on a device. Figure 2.2 (b) shows the typical OpenCL execution model with two command queues (one for CPU and other for GPU) in a single context.

Typically, the programmer decides the device for a kernel statically at application development time. There have been few previous works [36, 16, 110, 6] that proposed dynamic scheduling schemes to decide the device during run-time. Both application-level (i.e., same device for all kernels in an application) and kernel-level (based on each kernel's characteristics) scheduling schemes have been proposed. However, none of the previous work considered system physical condition (e.g., TDP) and run-time conditions (e.g., existence of other workloads on CPU) during scheduling decisions. In this thesis (chapter 4),

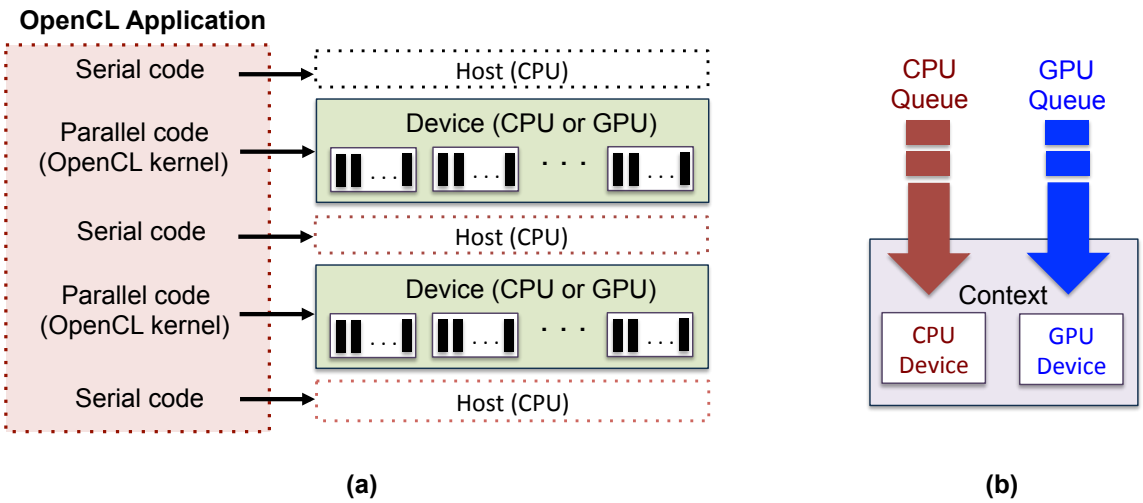

Figure 2.2: A typical OpenCL application launch on devices: a) platform model with an openCL application, b) OpenCL execution model with two command queues (one for each device) in single context.

we propose better scheduling techniques that not only consider the kernels' characteristics, but also take physical and run-time conditions of the system in to account while making scheduling decisions. We demonstrate that our proposed scheduling scheme performs better than both the static and the state-of-the-art scheduling schemes.

Next, we provide the background and related work for post-silicon power mapping of processors, workload scheduling on CPU-GPU processors and low power design of future massively parallel systems.

2.3 Post-Silicon Power Mapping and Modeling

Typically, computer-aided power analysis tools and simulators are used to estimate the power consumption of processors. While these tools are essential to analyze different design tradeoffs, the estimates made by these tools at the design time could deviate significantly from the actual power dissipation of working processors due to number of rea-

sons [38, 70]. Some of the reasons behind this discrepancy are as follows. First, the real processor design have billions of transistors and large input vector space. Since the power dissipation depends on the input pattern being applied, it becomes difficult for the simulators to estimate power for all possible input vector space in current processors. Probabilistic approaches can be used to reduce the size of input vector space, but it could add errors in power estimation due to lack of proper models for spatiotemporal correlation between different signals and internal nodes of the circuit [38]. Similarly, design tools could introduce errors in dynamic power estimation due to errors in coupling capacitance estimation between neighboring wires. Finally, process variations (both intra-die and inter-die) and dynamic thermal profile of chip impact the leakage power of the chip [43]. The pre-silicon tools rely on statistical models to models such variations, which could lead to inaccuracies in power estimates at design time.

In recent years, post-silicon power mapping has emerged as a technique to mitigate the uncertainties in design-time power models and enable effective post-silicon power characterization [42, 67, 91, 21, 92, 71, 93]. Many of these techniques rely on inverting the thermal emissions captured from an operational chip into a power profile. However, this approach faces numerous challenges, such as the need for accurate thermal to power modeling, the need to remove artifacts introduced by the experimental setup, where infrared transparent oil-based heat removal system can lead to incorrect thermal profiles, and leakage variabilities. One of the most important factor in estimating post-silicon power is to have an accurate modeling matrix $\mathbf{R}$ which relates temperature to power. Hamann *et al.* [42] constructed the modeling matrix by using a laser measurements setup that injects individual powers pulses on the actual chip and measures the resultant response. Cochran *et al.* [21] and Nowroz *et al.* [71] used controlled test chips to experimentally find the $\mathbf{R}$-matrix by enabling each block in the test circuits. Both these methods need extensive experimental setup or special circuit design needs. Previous approaches to model $\mathbf{R}$ in

simulation (e.g., [51]) were only done for copper (Cu) spreader with the only objective of speeding thermal simulation runtime, where the model matrix $\mathbf{R}$ is used to substitute lengthy finite-element method (FEM)-based thermal simulations. In contrast to previous methods, we use finite-element method to accurately estimate the modeling matrix which encompasses all physical factors such as, cooling fluid temperature, fluid flow rate, heat transfer coefficients, chip geometry, etc.

Post-silicon infrared imaging requires oil-based cooling system [42, 67]. The thermal analysis based on oil-based system differ from widely used Cu-based heat sink [44]. Attempts to modify the oil-based system to match the Cu-based characteristics were not completely verified as they relied on the measurement of a single thermal sensor [66]. Our method translates the full oil-based thermal map to Cu-based thermal map, which is then used for all of our power analysis. Hence, our approach provides more accurate leakage power modeling. Recent works to estimate within-die leakage variability include analytical methods, empirical models, statistical method [60, 62, 104]. Actual chip leakage trend and values can deviate from these models significantly. Our leakage method accurately estimates leakage variabilities introduced by process variability without the need for any embedded leakage sensors that occupy silicon real estate.

In recent years, there has been a significant work using performance monitoring counters (PMCs) to model power consumption of processors [45, 88, 8, 98, 61, 41]. Performance counters are embedded in the processor to track the usage of different processor blocks. Examples of such events include the number of retired instructions, the number of cache hits, and the number of correctly predicted branches. The general approach of existing techniques is to choose a set of plausible performance counters to model the activity of each structure in the processor and then create empirical models that utilize the activities to estimate the power of each structure and the total power. In almost all existing techniques, the main way to verify the correctness is through the observation of the

total power at chip level. In contrast to previous works, where the PMCs are related and modeled to total chip power or simulated power, we relate actual power of each circuit block as estimated through infrared-based mapping to the runtime PMCs. This gives accurate per-block PMC models and enable us to isolate directly the PMCs responsible for power consumption of each block. The models could be used for effective run-time power management of processors.

Heterogeneous processors with architecturally different devices (CPU and GPU) integrated on the same die have introduced new challenges and opportunities for thermal and power management techniques because of shared thermal/power budgets between these devices. Using detailed thermal and power maps from infra-red imaging, we show that the new parallel programming paradigms (e.g., OpenCL) for CPU-GPU processors create a tighter coupling between the workload and thermal/power management unit or the operating system. Further, in this thesis, we demonstrate that the DVFS and spatial scheduling power management decisions are highly intertwined in terms of performance and power efficiency tradeoffs on a heterogeneous processor.

2.4 Workload Scheduling on Heterogeneous Processors

Heterogeneous systems with integrated CPU and GPU devices are becoming attractive as they provide cost-effective energy-efficient computing. OpenCL has emerged as a widely accepted standard for running the programs across multiple devices which differ in their architecture. For example, the OpenCL programming paradigm allows arbitrary work-distribution between CPU and GPU devices, where the programmer controls the distribution at the application development time. The operating system (OS) together with OpenCL Runtime (also called OpenCL driver) could schedule the application on the cho-

sen device. However, such a static scheme may not lead to an appropriate device selection for all kernels because different kernels may have different preferred devices based on the data size and kernel characteristics [110]. Furthermore, this scheduling decision seldom considers the run-time physical conditions [e.g., thermal design power (TDP), CPU workload conditions], which, as shown in this thesis, could affect the device decision.

Recent years have witnessed multiple research efforts devoted to efficient scheduling schemes for heterogeneous systems [68, 87, 30, 5, 110, 82, 90, 107]. The survey paper by Mittal *et. al.* provides an excellent overview of the state-of-the-art techniques for such systems [68]. We notice that most of the recent works have focused on discrete GPUs. In this thesis (chapter 4), we focus on the integrated GPU systems, where the performance of GPU and CPU could be comparable for many kernels. Prakash *et al.* [90] and Pandit *et al.* [82] proposed dividing each kernel between CPU and GPU devices, which requires careful consideration of data synchronization between the two partitions. In contrast, we focus on scheduling the entire kernel on either CPU or GPU device; so, these works are orthogonal to our work. Diamos *et al.* [30], Augonnet *et al.* [5], and Lee *et al.* [55] propose performance-aware dynamics scheduling solutions for single application cases running on discrete GPU systems. Pienaar *et al.* propose a model-driven runtime solution similar to OpenCL, but their approach requires writing programs using nonstandard constructs for implementing directed acyclic graphs [87]. SnuCL [52] provides an OpenCL framework for heterogeneous clusters, where the scheduling decision is made by the programmer at development time. Aji *et al.* use SnuCL to extend OpenCL APIs (also called MultiCL) with the scheduling related hints [2]. All these efforts are directed towards better scheduling under static system conditions, while our work makes appropriate scheduling decisions under both static and dynamically changing system conditions.

Application-level device contention-aware scheduling schemes based on average historical runtime have also been proposed [37, 36, 16]. However, our scheduler makes the

scheduling decisions at kernel-level leading to higher performance and energy savings than application-level scheduling. In Maestro, data orchestration and tuning is proposed for OpenCL devices using an additional abstraction layer over the existing OpenCL Runtime [100]. In Qilin, adaptive mapping of computations on CPU and GPU devices is implemented to minimize both runtime and energy of system [63]; however, unlike our approach, they require the complete application to be rewritten using custom APIs.

Yuan *et al.* use offline support vector machine based method to classify the kernels for CPU and GPU [110] based on static code structure; this work is similar to ours, but is limited in following ways. First, they use only the workload characteristics obtained at compile-time (except the work-group sizes) as features in their classifier without taking the run-time system conditions (TDP and other workloads on CPU cores) in to consideration. Therefore, their approach could potentially lead to wrong scheduling decisions. Second, their work focuses mainly on performance, however our work takes both performance or energy as an optimization goal and makes the scheduling decisions accordingly. Bailey *et al.* consider scheduling under different TDPs [6]; however, their approach is only applied in an off-line mode as it lacked the capability to switch from CPU to GPU or vice versa during run-time. Hence, in contrast to previous works, our approach decides the appropriate device for each kernel under time-varying TDPs and run-time CPU-load conditions, leading to higher runtime improvements and energy savings compared to the state-of-the-art scheduling techniques.

2.5 Workload-Aware Low-Power Design of Future GPUs

GPUs are being used to improve performance and energy efficiency of many classes of high-performance computing (HPC) applications [13, 34]. Typically, these applications

have high parallelism, however, some kernels have limited parallelism and they do not require all the compute units (CUs) available in a massively parallel GPU. For such kernels, some of the CUs could be power gated to save leakage power without affecting the performance of the kernel. Dynamic voltage and frequency scaling (DVFS), clock-gating, and power-gating are common techniques used to manage power and energy in multi-core and parallel processors [109, 46, 97, 25, 49].

J. Li *et al.* proposed a run-time voltage/frequency and core-scaling scheduling algorithm that minimizes the power consumption of general-purpose chip multi-processors within a performance constraint [59]. J. Lee *et al.* analyzed throughput improvement of power-constrained multi-core processors by using power gating and DVFS techniques [54]. Wang *et al.* proposed workload-partitioning mechanisms between the CPU and GPU to utilize the overall chip power budget to improve throughput [108]. In [84], Paul *et al.* characterized thermal coupling effects between CPU and GPU and proposed a solution to balance thermal and performance-coupling effects dynamically. To minimize the leakage power dissipation in the idle GPU compute units and improve energy efficiency, different architecture-level power gating schemes are proposed in the context of performance requirements of applications [13, 109]. While Majeed *et al.* proposed a PG-aware warp scheduler to improve the benefits of GPU power gating [1], usefulness of core-level power gating for data center has also been investigated [58]. In order to maximize the benefits of power gating, Xu *et al.* proposed prioritization in warp scheduling to group the warps with same divergence behavior together in time to maximize the idleness window for single instruction multiple thread (SIMT) execution lanes [114].

The existing techniques are useful to improve the power efficiency of GPUs with underutilized resources or improve performance under TDP constraints. However, unlike our study, most of previous studies investigated PG opportunities statically by assuming the finest level of power gating at per-CU/core level without considering area overhead

or they did not consider the impact of design-time choices on the run-time performances. In contrast to previous works that considered few CUs [58, 50], our methodologies are geared for hundreds of CUs that will be available at the end of the silicon roadmap.

In the HPC community, there have been studies related to energy-performance-power trade-offs for HPC applications [85, 95]. Laros *et al.* performed large-scale analysis of power and performance requirements for scientific applications based on the static tuning of applications through DVFS, core, and bandwidth scaling [40]. Balaprakash *et al.* described exascale workload characteristics and created a statistical model to extrapolate application characteristics as a function of problem size [7]. Wu *et al.* also look at similar projection approach [112]. They rely on machine-learning classifications to project performance and power at different configurations, but they do not account for leakage power, thermal constraints or technology scaling. Further, Huang *et al.* [41] proposed power model based on run-time proxies for multi-core processors. These power models were used to predict power at both core and chip level to design energy saving run-time polices. All these efforts focused mainly on existing hardware architectures; However, we focus on massively parallel GPU architecture with hundreds of CUs at different PG granularities in the exascale timeframe. In contrast to the previous works, we investigate the effect of design-time power gating granularities coupled with run-time power management on the performance and power efficiency of future massively parallel GPUs under fixed power constraints.

Chapter 3

Post-Silicon Power Mapping and Modeling

3.1 Introduction

In this chapter, we describe a novel framework for post-silicon power mapping and modeling for multicore CPU-GPU processors. As described in the previous chapter, power is a major design challenge for the chip architects due to its limiting nature on the performance of semiconductor-based chips. The design complexity of modern processors coupled with process variability and runtime workloads characteristics make it harder to accurately estimate power consumption during design time [12, 88]. In recent years, post-silicon power mapping based on infrared imaging has emerged as a technique to mitigate the uncertainties in design-time power models [42, 67, 91, 21]. Many of these techniques rely on inverting the thermal emissions captured from an operational chip into a power profile. However, this approach faces numerous challenges, such as the need for accurate

thermal to power modeling, the need to remove artifacts introduced by the experimental setup, where infrared transparent oil-based heat removal system can lead to incorrect thermal profiles, and leakage variabilities. Our proposed framework solves many of the open challenges in this area. In particular, our framework is capable of identifying the dynamic and leakage power consumption of the main blocks of multi-core processors under different workloads, while simultaneously analyzing the impact of process variability on leakage and capturing the relationship between the performance monitoring counters (PMCs) and per-block power consumption.

Further, heterogeneous CPU-GPU processors are becoming mainstream these days due to their good power efficiency for wide range of applications. The new programming paradigms (e.g., OpenCL) for these processors allow arbitrary work-distribution between CPU and GPU devices, where the programmer controls the distribution at the application development time [75]. Due to the shared nature of thermal and power resources and due to application-dependent work distribution between two devices, there are new challenges and opportunities to optimize performance and power efficiency of the CPU-GPU processors [84, 85]. We perform experiments on both CPU-only and CPU-GPU processors.

Modern processors have two main knobs of thermal and power management: dynamic voltage and frequency scaling (DVFS), and scheduling of workloads on different compute units of the chip [33, 31, 89, 19, 20]. DVFS is used to trade performance for keeping temperature and power below their safe limits; similarly, thermal-aware scheduling helps in distributing thermal hot spots across the die. In a traditional multi-core CPU, all cores have the same micro-architecture. Therefore, at a fixed DVFS setting scheduling has little or negligible effect on the performance and power of a workload, especially for a single-threaded workload. On the other hand, as we demonstrate in this chapter, DVFS and spatial scheduling-based power management decisions are highly intertwined in terms of performance and power efficiency tradeoffs on a heterogeneous processor. In addition to

confirming many largely believed behavior in simulation, our experiments highlight multiple implications of CPU-GPU processors on thermal and power management techniques. The major contribution of this chapter are as follows.

1. We propose a numerical technique that uses accurate finite-element modeling (FEM) to translate the measured thermal maps captured from infrared-transparent heat sink systems to corresponding thermal maps of traditional metal and fan sinks, and then inverts the translated thermal maps to power maps. The proposed technique compensates for the thermal artifacts introduced by oil-based setup and can substitute for experimental techniques to match thermal behavior of different sinks [66].

2. We use thermal conditioning to devise spatial leakage variability models. The leakage models enable us to decompose the per-block power consumption into its dynamic and leakage components. Once estimated for a given chip, these leakage models can be used to compute leakage power map for any workload readily from its thermal-map alone, hence simplifying the overall power-mapping process.

3. We collect PMC values while simultaneously performing infrared-based power mapping. The PMC values are correlated with the power maps to identify the PMCs that are directly responsible for the power consumption of each block. Unlike previous works, [88, 9, 8] which had no access to the actual per-block power consumption, we develop per-block mathematical models by relating the measured PMCs to the per-block power consumption as calculated by the infrared power mapping framework. We use the PMC-based models to analyze the transient power consumption of each processor block.

4. We apply our proposed framework on a real quad-core processor to get detailed dynamic and leakage powers for different blocks (e.g., cores, L2-caches, etc.) while executing workloads using multiple SPEC CPU 2006 benchmarks. Proposed PMC-

based models are used to estimate power dissipation in each block of the processor over time. Our power mapping results provide useful insights into the distribution of power in multi-core processors.

5. We also use the framework to obtain detailed thermal and power breakdown of a real CPU-GPU chip as a function of hardware module and workload characteristics. We characterize the effects of task scheduling and DVFS on a heterogeneous processor using the total power consumption of different blocks.

6. We study interactions between workload characteristics, scheduling decisions, and DVFS settings for OpenCL workloads. We observe that the effects of DVFS and scheduling on performance, power, and temperature for OpenCL workloads are highly intertwined. Therefore, DVFS and scheduling must be considered simultaneously to achieve the optimal runtime and energy on CPU-GPU processors.

7. We show that the CPU and GPU devices have different power densities and thermal profiles, which could have multiple implications on the thermal and power management solutions for such processors.

The organization of the chapter is as follows. Section 3.2 describes the proposed framework for post-silicon power mapping and modeling. In Section 3.3, we present our power mapping and modeling results for a quad-core CPU processor. Next, in Section 3.4, we highlight the implications of integrating two architecturally different devices (CPU and GPU) on a single die on their thermal and power management using detailed power mapping experiments. Finally, we summarize the chapter in Section 3.5.

3.2 Proposed Power Mapping and Modeling Framework

Post-silicon power mapping for multi-core processors is the process of reconstructing power dissipation in different hardware blocks from the thermal infrared emissions of the processor during operation and under realistic loading conditions. When a processor runs a workload, it consumes power, which dissipates heat and changes the temperature of the chip. The thermal emissions from the chip can be captured by an infrared imaging system, and processed to reveal the underlying power consumption profile [67, 42].

Post-silicon power mapping involves many challenges at both the experimental and modeling fronts. At the experimental front, it is required to control the speed and temperature of the oil flow on top of the processor to remove the generated heat, while maintaing good optical transparency to the infrared imaging systems. Furthermore, it is important to accurately synchronize all the measurements of the system, including thermal maps, fluid state measurements, total power consumption, and PMC measurements from within the processor. At the processing front, challenges include the need to model the relationship between power consumption and temperature. This process is complicated by the fact that replacing the fan and copper heat-spreader with an infrared-transparent fluid-based heat sink system alters the thermal profile of the die [44]. Compromised thermal characteristics will alter the leakage profile of the processor [62, 104]. Decomposing the total power into leakage and dynamic is a challenging task due to the dependency of leakage on process variability and temperature.

Figure 3.1 gives the framework of the proposed power mapping and modeling method. At the beginning, a one-time design effort per chip-design is conducted to devise accurate FEMs ($\mathbf{R}_{oil}$ and $\mathbf{R}_{cu}$) that relate power to temperature under two heat removal mechanisms (oil-based and Copper/fan-based). During run-time, realistic workloads are applied to the

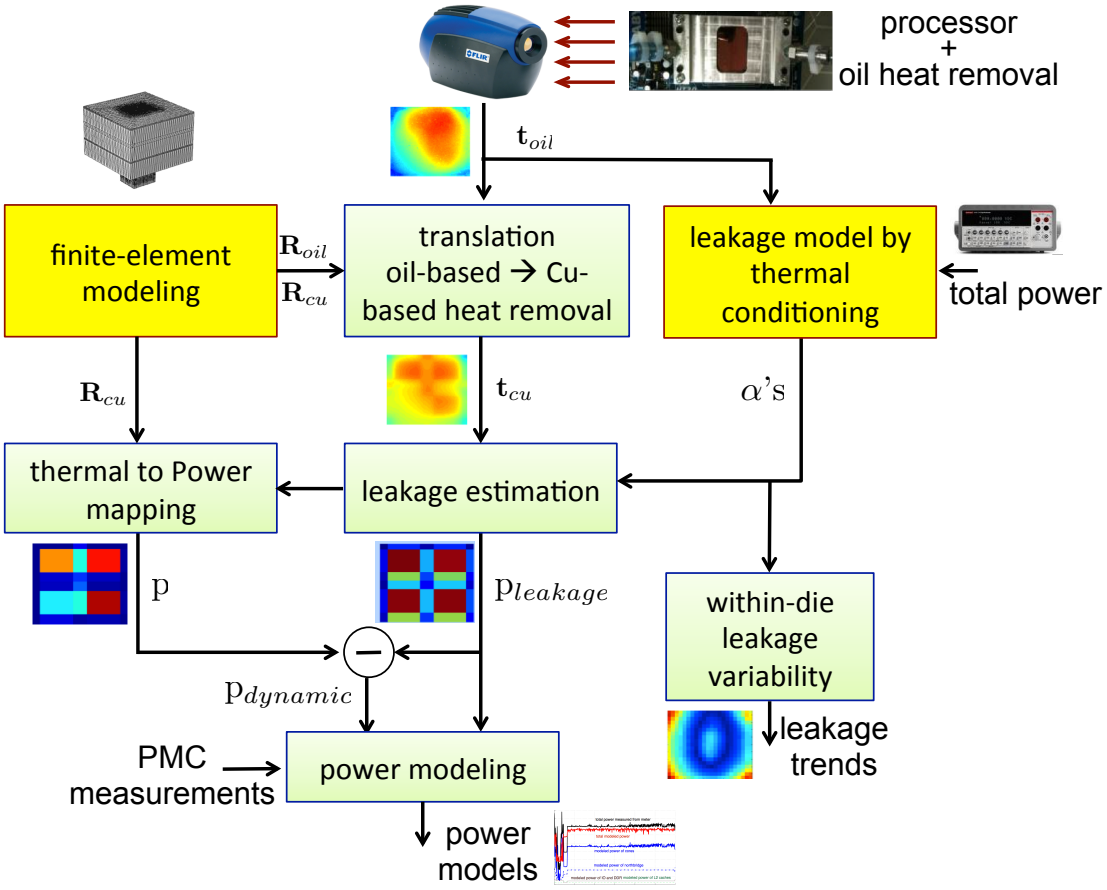

Figure 3.1: Proposed power mapping and modeling framework.

processor and the steady-state or averaged thermal map ($\mathbf{t}_{oil}$) is captured with the infrared camera. Using the devised FEMs, the captured thermal map is then translated to produce a thermal map ($\mathbf{t}_{cu}$) that mimics the case when the oil-based heat sink is replaced by a traditional Copper (Cu) spreader + fan heat removal mechanisms. Thermal conditioning is one-time modeling process that models the leakage power profile as a function of the temperature profile and can be further used to estimate the spatial variability trends. For each measured thermal map $\mathbf{t}_{cu}$, the leakage models are used to estimate the leakage power per block. The thermal map is then numerically processed to yield the per-block power maps, where we use leakage power as lower bound constraint. The total power for each block in the core is separated into dynamic and leakage power. The estimated power for different blocks of the processor is then modeled with runtime performance monitoring

counters and sensor measurements. The PMCs models can be then used to model the transient power consumption or in cases where no infrared imaging system is available. Below sections describe different components of the proposed framework.

3.2.1 Modeling Relationship Between Temperature and Power

Our goal is to model the relationship between power and temperature for a processor. In particular, if $\mathbf{t}$ is a vector that denotes the steady state or averaged thermal map of the processor in response to some power map denoted by $\mathbf{p}$, then our goal is to model the relationship between $\mathbf{p}$ and $\mathbf{t}$. We note that the length of $\mathbf{p}$ is determined by the number of the blocks in the processor's layout and the length of $\mathbf{t}$ is determined by the number of pixels in the thermal image. Our modeling approach consists of the following three steps.

1. We first describe the modeling and simulation of heat transfer in the case of oil-based heat sink. We show that the underlying physics can be described by a linear operator $\mathbf{R}_{oil}$ that maps $\mathbf{p}$ to $\mathbf{t}_{oil}$. This operator is determined empirically by simulation using accurate FEM modeling.

2. We then describe the modeling and simulation of heat transfer with Cu-based heat sink. Here, the underlying physics can also be described by a linear operator $\mathbf{R}_{cu}$ that maps $\mathbf{p}$ to $\mathbf{t}_{cu}$.

3. Thirdly, we describe how to translate a captured thermal image $\mathbf{t}_{oil}$ to make it appear as if it is coming from a Cu-based heat spreader.

Figure 3.2: Infrared-transparent oil-based heat removal system.

1. Modeling Oil-Based Heat Sink.

Modeling a heatsink and getting its thermal-power model matrix ($\mathbf{R}_{oil}$) typically consists of three components: a) setting up the simulation model, b) simulating heat-transfer physics, c) getting model matrix. These steps are described in the following paragraphs.

a) Model Setup. To enable the thermal imaging of the processor while maintaining the cooling efficiency similar to conventional fan-based heat sink system, we designed a special cooling system, as shown in Figure 3.2. The system has a rectangular channel of height 1 mm through which an infrared-transparent mineral oil is flowing from the inlet valve to the outlet valve. Two infrared-transparent windows (one at the top and other one at bottom of channel) are assembled in the system in such a way that they allow midwave infrared waves to pass through part of the channel. In addition to being infrared transparent, the bottom window spreads the heat generated in small processor die over a larger area, which improves heat removal capacity.

When the multi-core processor is switched-on, heat is generated at the active (transistor) layer of the die. The majority of the heat generated inside processor-die flows upwards and is carried away by the fluid-flow after passing through the bottom window. Small portion of the heat also flows through a secondary path towards the bottom side of the die to the motherboard and eventually to ambient. Empirically, we found that in our setup about 10% of the heat flows downward through the secondary path and about 90% of the heat

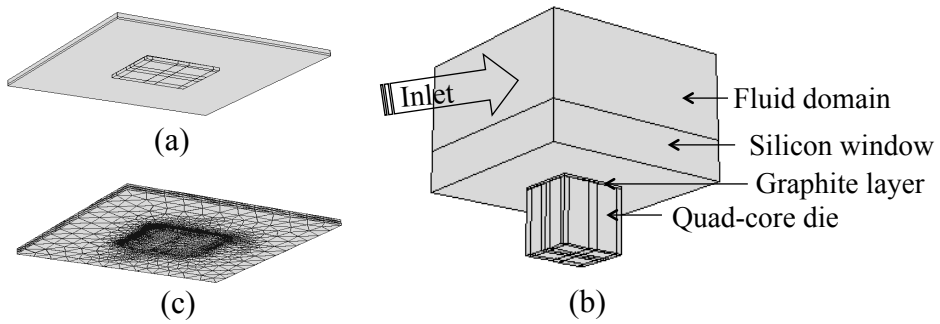

Figure 3.3: Model for oil-based system: (a) model-geometry with actual aspect-ratio; (b) model-geometry in perspective view; (c) meshed model

flows upward in our setup; so, we consider these values in our models. The infrared radiations from the silicon-die pass through the bottom window, fluid, and top window before they could be captured by the infrared camera. To improve the emissivity uniformity of the die, we apply a thin coating of graphite at the back side of the die; graphite has emissivity close to that of a perfect black body radiator.

We modeled the system in COMSOL Multiphysics tool, which is widely used to solve multiple coupled physical phenomena. COMSOL has a finite element analysis (FEM) based solver as its core computational-engine [22]. The geometry of the simulated model is shown in Figure 3.3 (a); Figure 3.3 (b) shows the zoomed picture, so it does not necessarily have same scale as actual scale. The model has following domains: the processor's die, divided into a number of blocks as dictated by the floorplan, a 25 μm graphite-layer, a 2 μm thermal interface material, an infrared-transparent silicon-window and fluid domain.

material	ρ	k	C_p	μ
silicon	2330	148	703	-
graphite	1950	150	710	-
mineral oil	838	0.138	1670	14.246e-3

Table 3.1: Material properties. ρ denotes the density of the material in kg/m^3, k represents the thermal conductivity of the material in W/(m.K), C_p denotes the specific heat capacity of the material at constant pressure in J/(kg.K), and μ represents the dynamic viscosity of the fluid in Pa.s.

We modeled the secondary path of heat removal by specifying a uniform heat removal rate from the bottom of the die. This uniform heat removal abstracts the impact of heat removal carried by the motherboard. The properties of different materials used in our simulation model are reported in Table 3.1.

To solve the modeling problem using finite-element method (FEM), the complete geometry has to be divided into smaller elements in a process known as meshing. Creating a proper mesh is important for two reasons: (1) a properly-sized mesh enables accurate simulation of the required physical phenomena, and (2) it controls the convergence of the numerical solution. For these two reasons, we refined the mesh to appropriate sizes at different interfaces and corners by adding boundary-layers and by choosing the mesh-size individually for each domain. The mesh is refined iteratively until it has significant impact on the final solution. The meshed model is shown in Figure 3.3 (c).

b) Model Simulation. Essentially, we have to simulate two types of physics: *fluid-flow* and *conjugate heat transfer*, simultaneously to obtain the temperature profile for a given power dissipation profile of the processor. We describe these two simulations in detail in the next paragraphs.

In our experimental system, we measured the flow speed, fluid temperature and the fluid pressure using a Proteus Fluid Vision flow meter. The average fluid speed is maintained at 5 m/s using a gear pump, the fluid temperature is maintained at 20 °C using a thermoelectric cooler with a feedback controller that receives its input from the fluid temperature meter, and the flood pressure at the inlet of heat sink is equal to 24 psi. In order to decide the the nature of fluid-flow, we compute the ratio of inertial-force to viscous-force, also called Reynolds number (Re), for the measured flow-speed in our system. For our channel dimensions and fluid flow characteristics, we computed the Re number for the flow as 434.48. Since Re$<$1000, we consider a laminar flow in our model simulations.

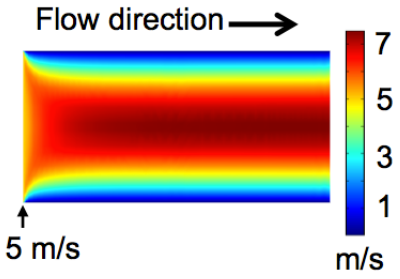

Figure 3.4: Velocity flow profile in the channel of the heat sink.

We assume that fluid-flow is incompressible, which is a reasonable assumption because the fluid is flowing at such a high speed that there does not exist significant temperature gradient in the fluid domain which could potentially change the fluid density.

Internally, the FEM tool solves Navier-Stokes conservation of momentum equation and conservation of mass equation to simulate the laminar flow [96]. We use following boundary conditions during flow-simulation. Since the flow is laminar, we consider no-slip boundary condition at all four walls of the fluid-domain, i.e. the fluid has zero velocity at the boundary. We also consider a uniform normal inflow velocity at the inlet of fluid domain. The simulated velocity profile for the measured flow rate in the heat sink's channel is shown in Figure 3.4.

We have to simulate the heat transfer in both solid and fluid domains. During all our experiments, we wait for the steady-state of the processor before we capture its thermal image. So, we simulate the heat-transfer equation in steady-state, where the heat equation in solid and fluid domains is given by [96]:

$$\rho C_p \mathbf{v}.\nabla T = \nabla.(k\nabla T) + Q \tag{3.1}$$

where, T is the temperature in Kelvin, $\mathbf{v}$ is the velocity field, and Q denotes the heat sources in W/m^3. For heat-transfer physics, we use following boundary conditions during

simulation. It is assumed that all external walls of the system exchange heat with ambience through natural convection process; the typical heat-transfer coefficient (h) for natural heat convection is 5 W/(m^2.K).

In the simulation model, we assume a standard silicon die of 750 μm and that power dissipation happens at the bottom of silicon die. Hence, if a particular block i of the die is dissipating, say, Q_i amount of power per unit area, then, in order to compute the temperature profile, we apply $p_i = Q_i * Block\_Area$ Watts of power to that block and simulate the heat-transfer and fluid-flow equations simultaneously.

c) Model Matrix Operator. While the model setup and simulation under various power profiles is a time-consuming task, the entire system operation can be represented by a modeling matrix, denoted by $\mathbf{R}_{oil}$, which is a *linear operator* that maps the power profile into a thermal map [51, 42]. If $\mathbf{p}$ is a vector that denotes the power map, where the power of each block, p_i, is represented by an element in $\mathbf{p}$, then $\mathbf{R}_{oil}\mathbf{p} = \mathbf{t}_{oil}$. The values of the matrix $\mathbf{R}_{oil}$ are learned through the FEM simulations of the setup, where we apply unit power pulses at each block location, one at a time, and compute the thermal profile at the die-surface for each case. The thermal profile resultant from activating block i corresponds to the i column of $\mathbf{R}_{oil}$. After simulating all blocks, we have the model matrix ($\mathbf{R}_{oil}$) complete. This thermal matrix can be used to relate any power profile and to the temperature profile.

To validate that the power to thermal relationship of the complete system can be modeled using a linear operator, we performed the following experiment. First, we simulated the temperature profile by allocating 1 W of power to the top-left part of a die; the simulated thermal map, $\mathbf{t}_1$, is shown in the first column of Figure 3.5. Next, we applied a unit power to the bottom-right part of the die and obtained the temperature map, $\mathbf{t}_2$, shown in the second column of Figure 3.5. Third, we simulated the temperature profile by assum-

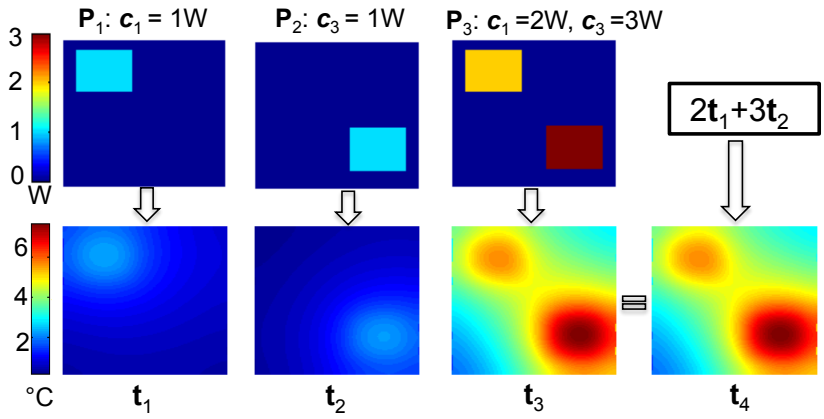

Figure 3.5: Verifying the linear relation between power and temperature for oil-based system. Temperatures are shown as ΔT, difference over fluid-temperature.

ing that the top-left is dissipating 2 W and the bottom-right is dissipating 3 W power. The simulated temperature map, $\mathbf{t}_3$ for this case is shown in the third column of Figure 3.5. If the physics of system can be indeed represented by a linear operator, then the *superposition principle* should hold, and the temperature map simulated in the third case, $\mathbf{t}_3$, should be equal to $2\mathbf{t}_1 + 3\mathbf{t}_2$). The resultant temperature map from superposition is given in the fourth column of Figure 3.5, perfectly matching the results from simulation, confirm the validity of the model.

2. Modeling Copper-Based Heat Sink.

In traditional heat removal systems, a heat spreader, made of copper and relatively larger in size than the processor-die size, is attached on the back-side of the die. In addition, a fan could be installed directly on the top of the heat spreader to increase the heat removal capacity. In our simulation, we model the multi-core processor die and the heat-spreader directly, while heat-removal capabilities of different fans are simulated by varying the heat-transfer coefficient at the top side of metal heat spreader. The model simulated using FEM is shown in Figure 3.6 (a); and, the meshed model is shown in Figure 3.6 (b). Unlike oil-based system, where we had to simulate both flow and heat-transfer physics simul-

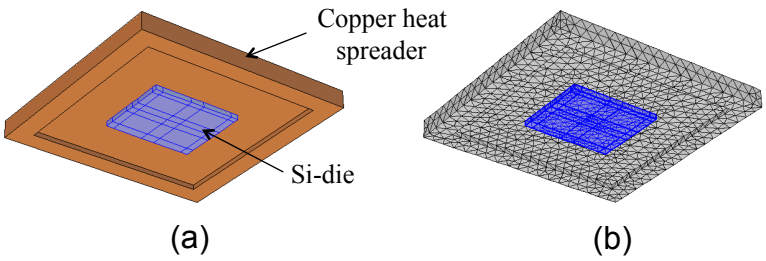

Figure 3.6: Model for Cu/fan-based cooling system (a) Geometry; (b) Meshed model.

taneously, with a metal heat spreader system, we only need to simulate the heat-transfer with appropriate boundary conditions. The dimensions used for the heat spreader in our simulation model are the actual dimensions of the heat spreader that came with our experimental processor. Finally, to compute the modeling matrix ($\mathbf{R}_{cu}$) for the Cu/fan-based system, we simulate the thermal response of the system by applying unit power pulses at each block, one at a time and assemble the column of the model $\mathbf{R}_{cu}$. This step is similar to building $\mathbf{R}_{oil}$ matrix operator, as discussed above.

3. Heat Sink Thermal Translation.

We replaced the conventional fan-cooled copper heat-spreader heat sink system with a special fluid-based heat sink system to capture the thermal images of the processor. The thermal characteristics of the mineral oil and its direction flow changes the temperature profile of the die [44], which has implications on leakage power. That is, if we run same workload on the processor, we get different temperature and leakage profiles for two heat sink systems. Previous work did not model this effect accurately [67, 4] as pointed in the literature [44]. We propose an accurate technique to compute the temperature profile of the die for Cu-based heat sink system from the measured temperature profile for oil-based heat sink system. The proposed technique is as follows. Let's assume that some power profile $\mathbf{p}$ is imposed in the simulation model on the die, then the temperature profile in two cases can be expressed as:

$$\mathbf{R}_{oil}\mathbf{p} = \mathbf{T}_{oil} \tag{3.2}$$

$$\mathbf{R}_{cu}\mathbf{p} = \mathbf{T}_{cu} \tag{3.3}$$

From Equations (3.2) and (3.3), we could write:

$$\mathbf{R}_{cu}^{-1}\mathbf{T}_{cu} = \mathbf{R}_{oil}^{-1}\mathbf{T}_{oil} \Longrightarrow \mathbf{T}_{cu} = \mathbf{R}_{cu}\mathbf{R}_{oil}^{-1}\mathbf{T}_{oil}$$

It is worth mentioning here that the thermal resistance matrices, $\mathbf{R}_{cu}$ and $\mathbf{R}_{oil}$, need not to be square matrices as there are typically many more pixels than blocks in the floor plan. In such cases, we either need to compute pseudoinverse of the matrix or we have to solve following equation to obtain $\mathbf{T}_{cu}$ from $\mathbf{T}_{oil}$:

$$\mathbf{T}_{cu} = \mathbf{R}_{cu}\left(\mathbf{R}_{oil}^{T}\mathbf{R}_{oil}\right)^{-1}\mathbf{R}_{oil}^{T}\mathbf{T}_{oil} \tag{3.4}$$

In order to validate the above technique, we applied a power profile of 40 W to our die model and simulated the temperature profile for oil-based system in COMSOL. The simulated profile for the oil-system is shown in Figure 3.7 (a). Next, we computed the temperature profile for cu-based system in two ways: 1) using the proposed technique, and 2) using COMSOL for reference. As could be seen from Figure 3.7 (b) and Fig-

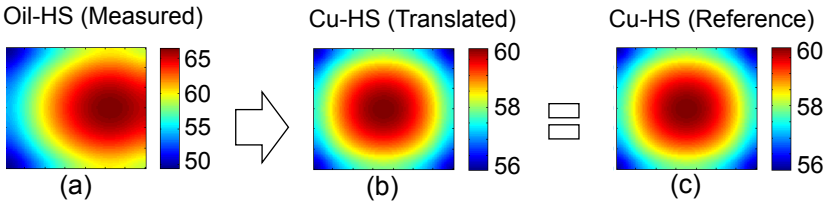

Figure 3.7: (a) Thermal map measured for the oil heatsink (HS) system, (b) thermal map for the Cu heat spreader translated using Equation (3.4), and (c) thermal map simulated directly for Cu heat spreader.

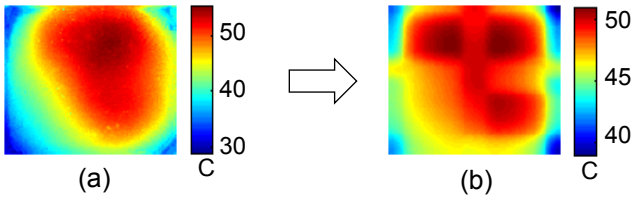

Figure 3.8: (a) Measured thermal map from oil-based cooling system (measured); (b) thermal map of Cu-based cooling system translated using Equation (3.4).

ure 3.7 (c), the two temperature profiles, computed in two ways, for the Cu-system are exactly the same. This confirms that the simulation framework for two systems is correct. To further illustrate the usefulness of the proposed technique, we ran standard benchmark applications on three cores of our experimental quad-core processor (described in details in Section 3.3). The measured thermal map of the processor is given in Figure 3.8 (a), and the translated thermal image for the Cu-based system is given in Figure 3.8 (b). It is clear that the two heat removal mechanisms have different thermal profiles, and our method is capable of translating between the thermal profiles, compensating for the differences.

3.2.2 Thermal to Power Mapping

a. Leakage Modeling

As described in chapter 2, aggressive scaling in sub-100 nm technologies has increased the contribution of leakage power to the total processor power. Leakage also has strong dependency on temperature, and as a result, the thermal profile of the die can vary due to leakage temperature interaction [15]. In this section, we propose a spatial leakage power mapping method based on a novel *thermal conditioning* technique[1]. The sub threshold leakage current, which is the dominant component of leakage power [62, 104], is given by:

[1]The leakage mapping was performed with the help of Abdullah Nowroz.

$$P_{sub} = VA\frac{W}{L}v_T^2(1 - e^{\frac{-V_{DS}}{v_T}})e^{\frac{(V_{GS}-V_{th})}{av_T}}, \tag{3.5}$$

where, P_{sub} is the subthreshold leakage power, V is the supply voltage, A is a technology dependent constant, V_{th} is the threshold voltage, W and L are the device effective channel width and length respectively, v_T is the thermal voltage, V_{DS} and V_{GS} are the drain-to-source voltage and gate-to-source voltage respectively, and a is the subthreshold swing coefficient for the transistor. Although leakage is exponential on temperature, for a given voltage and device and range of typical operation (20 °C – 85 °C), we can use Taylor series expansion to approximate the leakage power near a reference temperature T_{ref}. An expansion that includes up to quadratic terms is given by:

$$P_{sub}(T) = P_{ref} + \alpha_1(T - T_{ref}) + \alpha_2(T - T_{ref})^2, \tag{3.6}$$

where, $P_{sub}(T)$ is the leakage power at temperature T, P_{ref} is the leakage power at the reference temperature T_{ref}, and α_1 and α_2 are constants that depend on the voltage, process variability, and structure of devices. To model the chip's spatial leakage profile, we divide our die area into sufficiently large number of locations, n, such that the leakage power, $P_{sub}(T_i)$, at location i is given by:

$$P_{sub}(T_i) = P_{ref,i} + \alpha_{1,i}(T_i - T_{ref}) + \alpha_{2,i}(T_i - T_{ref})^2 \tag{3.7}$$

where T_i is the average temperature at location i, and $\alpha_{1,i}$ and $\alpha_{2,i}$ are model coefficients for location i. The total leakage power is sum of all the n locations in the chip, which can be written as:

$$P_{leakage} = \sum_i P_{ref,i} + \sum_{i=1}^{n}[\alpha_{1,i}(T_i - T_{ref}) + \alpha_{2,i}(T_i - T_{ref})^2], \tag{3.8}$$

which can be re-arranged as:

$$\begin{aligned} P_{leakage} - \sum_i P_{ref,i} &= \sum_{i=1}^{n} \alpha_{1,i}\Delta T_i + \alpha_{2,i}\Delta T_i^2 \\ \Delta P &= \sum_{i=1}^{n} \alpha_{1,i}\Delta T_i + \alpha_{2,i}\Delta T_i^2, \end{aligned} \tag{3.9}$$

where, $\Delta P = P_{leakage} - \sum_i P_{ref,i}$. In order to learn the model coefficients, we propose a novel *thermal conditioning* method. The idea is to increase the temperature of the chip gradually by increasing the temperature of the oil, while simultaneously recording the thermal images of the die, and measuring the total power consumption of the chip. Throughout the experiment, an application of stable nature is always executing. The increase in total power consumption would purely be due to changes in leakage. Thus, each thermal conditioning experiment provides a thermal image and an incremental total leakage power, which creates an instance of Equation 3.9. For example, the j^{th} thermal conditioning experiment will provide the following equation:

$$\Delta P_j = \sum_{i=1}^{n} \alpha_{1,i}\Delta T_{j,i} + \alpha_{2,i}\Delta T_{j,i}^2. \tag{3.10}$$

For m thermal conditioning experiments, we can assemble the system of equations as:

$$\begin{bmatrix} \Delta T_{1,1} & \Delta T_{1,1}^2 & \cdots & \Delta T_{1,n} & \Delta T_{1,n}^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \Delta T_{m,1} & \Delta T_{m,1}^2 & \cdots & \Delta T_{m,n} & \Delta T_{m,n}^2 \end{bmatrix} \begin{bmatrix} \alpha_{1,1} \\ \alpha_{2,1} \\ \vdots \\ \alpha_{1,n} \\ \alpha_{2,n} \end{bmatrix} = \begin{bmatrix} \Delta P_1 \\ \vdots \\ \Delta P_m \end{bmatrix} \tag{3.11}$$

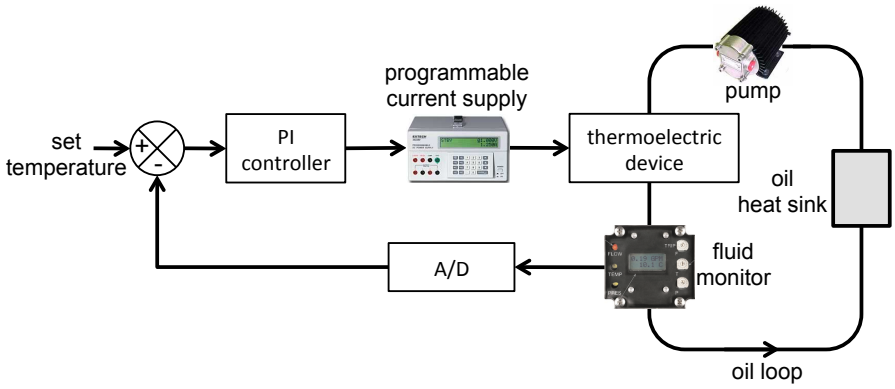

Figure 3.9: Experimental setup for thermal condititioning.

We solve the above system of equations by standard least-square regression technique to find first-order and second-order model coefficients; hence, we get a total of $2n$ coefficients ($\alpha_{1,1} - \alpha_{1,n}$ and $\alpha_{2,1} - \alpha_{2,n}$). To compute the leakage power, P_{ref}, at the reference temperature, we fit a quadratic model of the power measured to the average temperature from the thermal maps of the chip, and extrapolate to get the dynamic power. We estimate the P_{ref} by subtracting the dynamic power from the total power measured at T_{ref}. For a particular chips, these coefficients need to be computed only once, and then for estimating leakage of any thermal profile for the chip.

To implement thermal conditioning in our experimental setup, we use a thermoelectric device and a fluid monitoring device in line with the oil flow as shown in Figure 3.9. By changing the voltage and current of the thermoelectric device, we can either cool or heat the fluid to the desired temperature. Thus, we setup a feedback control system to control the fluid temperature to any desired set temperature point. In the feedback loop, the fluid temperature is compared to the set point and the error is fed to a PI controller, the output of which derives the programmable power supply of the thermoelectric device.

b. Reconstructing Dynamic and Total Power

Reconstructing the underlying power map of the processor from the measured thermal im-

ages is an inverse problem. In the framework used for quad-core processor, we measure the thermal maps for oil-based cooling system ($\mathbf{T}_{oil}$) and reconstruct the power-dissipation in different sub-units of the quad-core die for the Cu-based cooling system. We first compute leakage power in each die-unit from the the equivalent thermal image for the Cu-system and use the leakage power as the lower bound while reconstructing the total power for each unit. In particular, we solve the following optimization problem to reconstruct power map of the die.

$$\begin{aligned} \mathbf{p}^* = \quad & \arg_{\mathbf{p}} \min \|\mathbf{R}_{cu}\mathbf{p} - \mathbf{R}_{cu}\left(\mathbf{R}_{oil}\mathbf{R}_{oil}^T\right)^{-1}\mathbf{R}_{oil}\mathbf{T}_{oil}\|_2, \\ & \text{s. t.} \quad p_i^{leak} \leq p_i. \end{aligned} \tag{3.12}$$

where, $\mathbf{p}^*$ is the reconstructed power-vector, p_i^{leak} denotes the leakage power in the i^{th} die-block, and p_i denotes the power in the i^{th} block of the die. Other terms, $\mathbf{R}_{cu}$, $\mathbf{p}$, $\mathbf{R}_{oil}$, and $\mathbf{T}$, are already defined in the text before. By solving the above optimization problem, we obtain the total power of each block for the die. Finally, we compute the dynamic power of each block by subtracting the leakage power from the reconstructed total power. Using $p_i^{leak} > 0$ constraint helps in ensuring that dynamic power for all blocks is always positive. Hence, we reconstruct all, the dynamic, leakage, and total powers for each block of the processor from its measured temperature image.

3.2.3 Power Modeling Using PMCs

A popular approach for modeling total power is through the use of *performance monitoring counters* (PMCs) [48, 45, 10, 113, 86, 88, 9, 47, 61, 8]. Performance counters are embedded in the processor to track the usage of different processor blocks. Typical approach is to model the power of different blocks using their switching activity and com-

Procedure: PMC-based power modeling procedure
Input: Infrared-based power estimates for each block and associated PMC measurements
Output: Power Models for each block as a function of PMC measurements

For each circuit block i:

a. Identify the PMC measurements that are strongly correlated with power estimates of i.

b. Use least-square estimation to fit a linear model that estimates the power of i as a function of the strongly-correlated PMC measurements.

Figure 3.10: Algorithm to compute PMC-based models.

pare the total estimated power against the total measured power. That is there is no reliable way to verify the estimated block power. In contrast, our infrared-based power mapping technique directly obtains the power consumption of each circuit block under different workload conditions. Thus, we propose to simultaneously collect the measurements of the PMC, while collecting the infrared imaging data. The post-silicon power estimates are then used to derive fitted empirical models that relate the performance counters to the power consumption of each of block. For instance, if m_1, m_2, m_3 are three PMCs correlated to the power estimates, p_i, of block i, then an empirical model, $\hat{p}_i$, can be described as $\hat{p}_i = c_0 + c_1 m_1 + c_2 m_2 + c_3 m_3$, where c_0, c_1, c_2 and c_3 are the model coefficients which have to be determined by fitting the observed power estimates of each block with the PMC measurements on a training set of workloads. The fitting is done using least-square estimation, where it is desired to minimize the modeling error, $(\hat{p}_i - p_i)^2$ over the training data. The main steps of our power modeling procedure are summarized in Figure 3.10.

The fitted PMC models can enable us to substitute the post-silicon power mapping results in situations where infrared imaging is difficult. These cases include, for example, systems deployed in user environments where access to infrared imaging is not easy, or

for high-resolution transient power mapping. Infrared-based transient power mapping is inherently limited because of the low-pass filtering of power variations and the limited sampling rate of infrared cameras [91]. We illustrate the use of PMC-based models for transient power modeling in Section 3.3.

Next, we present the power mapping results for two processors using our proposed framework: 1) a quad-core CPU processor, 2) a heterogeneous CPU-GPU processor. The results for the quad-core processor shows the usefulness of the framework for modeling leakage power, dynamic power and PMC based power models. On the other hand, for CPU-GPU processor, we use the framework mainly for understanding the implications of integrating two architecturally different devices. So, we focus on reconstructing only the total power and temperature of different blocks for the heterogeneous processor.

3.3 Power Mapping of a Multi-core CPU Processor

For power mapping of a homogeneous processor, we used a motherboard fitted with a 45 nm AMD Athlon II X4 610e quad-core processor and 4 GB of memory. The motherboard runs Linux OS with 2.6.10.8 kernel. The floorplan of the processor with 11 different blocks is shown in Figure 3.11. We treat each core as one block, as we could not find public-domain information on the make-up of blocks within each core. The processor has 4 $\times$ 512 KB L2 caches, which are easy to identify in the floorplan. The processor lacks a shared L3 cache. The area in the center is occupied by the northbridge and other miscellaneous components such as the main clock trunks, the thermal sensor, and the built-in thermal throttling and power management circuits. The periphery is composed of the devices for I/O and DDR3 communication. The processor supports four distinct DVFS settings. Except for the first experiment, we set the DVFS to 1.7 GHz. We image the

processor using a mid-wave FLIR 5600 camera with 640× 512 pixel resolution. We also intercept the 12 V supply lines to the processor and measure the current through a shunt resistor connected to an external Agilent 34410A digital multimeter, which enables to log the total power measurements of the processor.

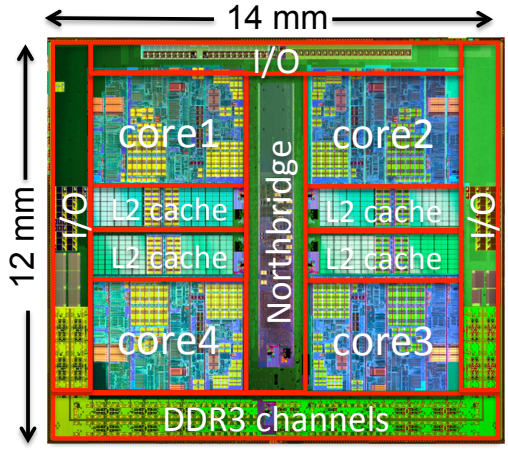

Figure 3.11: Layout AMD Athlon II X4 processor.

Experiment 1. Verification of Modeling Matrices: Given the processor layout and our setup, we first constructed the modeling matrices, $\mathbf{R}_{oil}$ and $\mathbf{R}_{cu}$, that map the power consumption to temperatures across the die in case of oil-based heat removal and Cu-based heat removal respectively. We compute these matrices by using finite-element modeling and simulation techniques described in Section 3.2.1. We verify the accuracy of the $\mathbf{R}_{oil}$ by comparing its modeling results against the images for the thermal system. To verify the accuracy of our modeled matrix $\mathbf{R}_{oil}$, a custom cpu-intensive micro-benchmark is utilized. The quad-core AMD processor has four DVFS settings: 0.8 GHz, 1.7 GHz, 1.9 GHz, and 2.4 GHz. First, we run the custom application on all four cores at 2.4 GHz frequency and capture the steady-state thermal image of the die and measure the total power of the processor. Let $\mathbf{t}_1$ be the resultant thermal image, and p_1 denotes the total measured power. Then, we change the frequency of just core 1 to 0.8 GHz to ensure that the switching activity profile changes only in one core. We again capture a steady-state thermal image, $\mathbf{t}_2$ of the processor and measure total power p_2. Since the activity change was localized

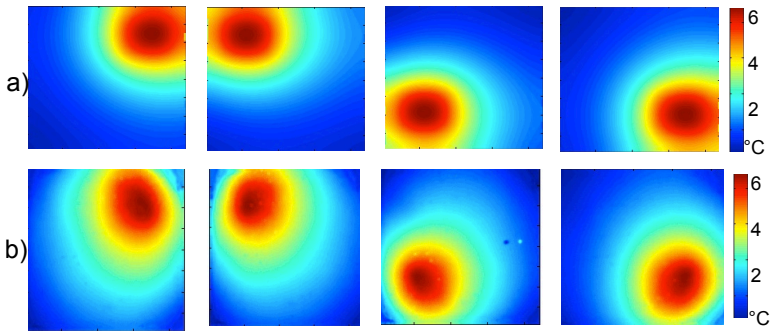

Figure 3.12: Thermal-matrix verification through comparison of impulse-responses of the system (a) simulated; (b) measured.

to only one core, we can expect the difference in power profiles, as denoted by the vector $\delta\mathbf{p}$, between the two cases to be mainly zero everywhere, but equal to $p_1 - p_2$ at the vector position corresponding to core 1. Thus, we can compare the thermal simulation results of $\mathbf{R}_{oil}\delta\mathbf{p}$ against the actual thermal image difference $\mathbf{t}_1 - \mathbf{t}_2$ to verify the accuracy of the $\mathbf{R}_{oil}$ model. The first column Figure 3.12 contrasts the simulation versus the real thermal map, showing the accuracy we obtained. We also repeated the same procedure for the other three cores and include the results in Figure 3.12.

Experiment 2. Demonstration of Power Mapping: The goal of this experiment is to demonstrate the results of power mapping the processor using different number of workloads and different workload characteristics. Our workloads come from widely used SPEC CPU06 benchmark suite. We selected four benchmark applications, which cover both integer point and floating point computations and processor-bound and memory-bound characteristics. These benchmarks are listed in Table 3.2.

	memory bound	**processor bound**
Integer point	*omnetpp*	*hmmer*
Floating point	*soplex*	*gamess*

Table 3.2: Selected SPEC CPU06 benchmarks.

a) Evaluation of the total, dynamic and leakage power maps for various workloads: In order to demonstrate the process of reconstructing power dissipation in different sub-units

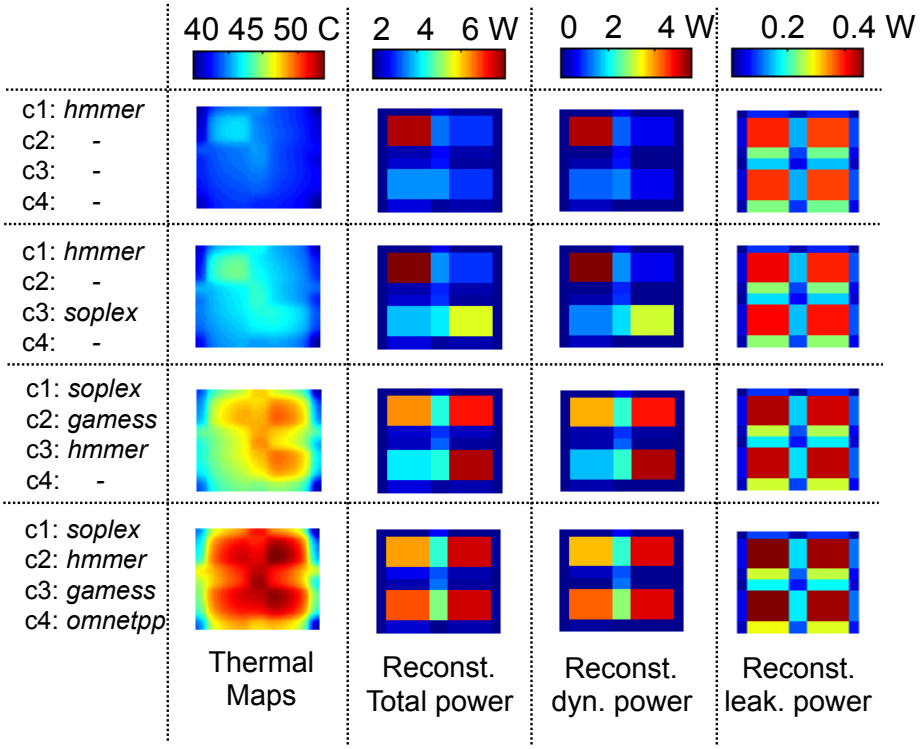

Figure 3.13: Thermal maps, reconstructed total, dynamic, and leakage power maps.

of multi-core processor from the measured thermal images, we ran 30 different cases of workload sets. For each experiment, we captured the steady-state thermal image using an infrared-camera and reconstructed the underlying power maps from the translated thermal maps to the Cu-based spreader. We decomposed the total power maps into dynamic and leakage power dissipation of each block of the processor and analyzed the spatial leakage variability. The reconstructed maps for four sample cases are shown in Figure 3.13. For example, the third-row shows a case, where we ran *soplex*, *gamess*, and *hmmer* benchmarks on cores 1, 2, and 3 respectively. Second column shows the equivalent temperature maps for Cu-system for each workload-case. The third column shows the reconstructed total power dissipation in each block for the four cases. It is clear from the reconstructed power-maps that they agree the intuitive expectation that cores running processor-bound applications (i.e., *hmmer* and *gamess*) are having higher power consumption than the idle cores or cores running memory-bound workloads. Similarly, fourth and fifth column show

core 1	core 2	core 3	core 4	Reconstructed total power (W) for each block											Total power (W)			
				core 1	L2-1	core 2	L2-2	core 3	L2-3	core 4	L2-4	I/O	N. B.	DDR3	dyn	lkg	dyn+lkg	meas
omnetpp	-	-	-	3.91	0.28	1.06	0.23	1.10	0.15	1.61	0.40	1.40	4.23	1.13	11.52	3.97	15.49	16.82
-	*omnetpp*	-	-	1.51	0.23	3.49	0.25	1.18	0.15	1.58	0.28	1.52	4.4	1.10	11.70	3.97	15.68	16.85
-	-	*omnetpp*	-	1.55	0.23	1.18	0.23	3.31	0.15	1.50	0.26	1.67	4.36	1.05	11.52	3.97	15.49	16.97
-	-	-	*omnetpp*	1.59	0.36	1.14	0.23	0.92	0.15	3.95	0.17	1.56	4.15	1.01	11.26	3.96	15.22	16.81
hmmer	-	-	-	5.68	0.23	0.97	0.23	0.99	0.15	1.55	0.40	1.22	4.34	0.97	12.72	4.02	16.73	18.49
soplex	-	-	-	4.07	0.30	1.02	0.23	1.08	0.15	1.60	0.38	1.29	4.3	1.09	11.54	3.97	15.51	17.11
gamess	-	-	-	5.41	0.23	0.94	0.23	0.89	0.15	1.51	0.42	1.21	4.23	0.90	12.13	3.99	16.12	18.23
omnetpp	*omnetpp*	-	-	3.81	0.31	3.19	0.23	1.24	0.15	1.78	0.36	1.29	5.19	1.07	14.55	4.10	18.65	19.60
hmmer	*hmmer*	-	-	5.90	0.25	5.07	0.24	0.96	0.16	1.82	0.41	1.26	5.95	0.97	18.70	4.30	23.00	23.70
soplex	*soplex*	-	-	3.94	0.34	3.30	0.24	1.30	0.16	1.84	0.37	1.29	5.3	1.13	15.08	4.12	19.20	19.82
gamess	*gamess*	-	-	5.60	0.26	4.80	0.24	0.84	0.16	1.68	0.45	1.22	5.72	0.87	17.60	4.24	21.84	23.17
omnetpp	-	*soplex*	-	3.90	0.32	1.11	0.23	3.55	0.16	1.75	0.35	1.28	5.19	1.07	14.80	4.11	18.91	19.86
omnetpp	-	*hmmer*	-	4.05	0.34	0.99	0.24	5.29	0.16	1.73	0.35	1.28	5.48	0.95	16.67	4.20	20.86	21.64
omnetpp	-	*gamess*	-	4.16	0.35	1.04	0.24	5.05	0.16	1.70	0.37	1.28	5.48	0.95	16.58	4.19	20.77	21.57
hmmer	-	*soplex*	-	6.01	0.24	0.98	0.24	3.57	0.16	1.79	0.42	1.24	5.51	1.04	17.00	4.21	21.22	21.72
hmmer	-	*gamess*	-	6.08	0.25	0.88	0.24	4.98	0.16	1.67	0.42	1.25	5.71	0.84	18.20	4.27	22.47	23.33
soplex	-	*gamess*	-	4.38	0.38	1.05	0.24	5.16	0.16	1.77	0.36	1.28	5.6	0.96	17.13	4.22	21.35	21.93
soplex	*soplex*	*soplex*	-	3.86	0.38	3.17	0.32	3.48	0.16	2.00	0.35	1.35	6.01	1.13	17.96	4.26	22.22	22.34
hmmer	*hmmer*	*hmmer*	-	6.28	0.27	5.24	0.26	5.55	0.17	2.09	0.41	1.35	7.53	0.88	25.35	4.68	30.03	28.71
omnetpp	*omnetpp*	*omnetpp*	-	3.91	0.37	3.18	0.30	3.48	0.16	1.98	0.37	1.35	6.05	1.12	18.00	4.26	22.26	22.23
gamess	-	*gamess*	*gamess*	6.13	0.60	0.76	0.25	4.61	0.17	6.51	0.18	1.32	6.71	0.74	23.41	4.57	27.99	27.99
gamess	*gamess*	*gamess*	-	5.98	0.36	5.06	0.36	5.11	0.17	1.87	0.43	1.30	7.22	0.76	24.02	4.60	28.62	28.30
omnetpp	*soplex*	*gamess*	-	4.09	0.40	3.46	0.36	5.52	0.17	2.03	0.36	1.38	6.66	1.02	21.02	4.42	25.45	24.87
omnetpp	*soplex*	*hmmer*	-	4.02	0.39	3.31	0.31	5.65	0.17	2.06	0.36	1.39	6.56	1.05	20.85	4.42	25.27	24.69
soplex	*gamess*	*hmmer*	-	4.33	0.44	5.16	0.35	5.72	0.17	2.11	0.35	1.36	7.16	0.97	23.54	4.57	28.11	27.01
soplex	*soplex*	*soplex*	*soplex*	3.92	0.54	3.10	0.27	3.24	0.16	4.35	0.25	1.40	6.56	1.14	20.53	4.40	24.93	24.40
hmmer	*hmmer*	*hmmer*	*hmmer*	6.37	0.50	5.13	0.28	5.21	0.18	7.34	0.19	1.44	8.88	0.83	31.25	5.10	36.35	33.12
gamess	*gamess*	*gamess*	*gamess*	6.32	0.67	5.16	0.30	4.83	0.18	6.89	0.20	1.39	8.58	0.82	30.32	5.03	35.35	33.21
omnetpp	*omnetpp*	*omnetpp*	*omnetpp*	3.98	0.54	3.17	0.27	3.31	0.16	4.44	0.25	1.41	6.72	1.14	20.97	4.42	25.39	24.61
soplex	*hmmer*	*gamess*	*omnetpp*	4.31	0.60	5.49	0.27	5.49	0.18	4.77	0.25	1.43	8.06	0.91	26.97	4.79	31.77	29.52

Table 3.3: Power-mapping results for 30 test cases. N.B. stands for north bridge block; dyn stands for dynamic; lkg stands for leakage; dyn+lkg is the total power reconstructed from post-silicon in infrared imaging; and meas is the total power measured through the external digital multimeter.

the per-unit reconstructed dynamic power and leakage power for four different workloads. The figures also show that the L2 cache power is mainly dominated by leakage power with a small amount of dynamic power.

The per-block power results for all 30 different workload cases are presented in Table 3.3. We also report the total dynamic power, total leakage power, and the sum of leakage and dynamic power. The results show that the leakage power comprise about 20% of the total power. We also report in the last column the total measured power through the external multimeter after compensating for the total leakage difference between the oil-based sink and the Cu-based sink. We notice that our total estimated power through infrared-based mapping achieve very close results, with an average absolute error of only 1.07 W of the measured power. The differences could be either to modeling inaccuracies or due to the fact that the measured total power also include the power consumed by the off-chip voltage regulators, and thus, it does not represent the net power consumed by the

processor. We have also considered including the total measured power as a constraint to the optimization formulation given in Section 3.2.2; however, the resultant power maps have displayed some counter-intuitive results.

b) Effect of number of applications: To see the impact of increasing number of applications on the power consumption of different blocks, such as cores, caches, northbridge, I/O, DDR3 channels, we run high power application *hmmer* in four different ways. First, we run one instance of *hmmer* on core 1, second, we run two instances of *hmmer* on core 1 and core 2, third we run three instances of *hmmer* on core 1, core 2 and core 3 and last we run four instances of *hmmer* on all four cores. Figure 3.14 shows the trend of power consumption of different blocks in the processor as we increase number of applications. When a core is idle it usually clock gates, and consumes minimum power, but as we increase the number of applications, the total power of the four cores increases proportionally. In contrary, the power consumption from other blocks such as the northbridge, I/O, DDR3 do not change as much depending on the number of workloads, because those blocks do not clock gate and they are always operational.

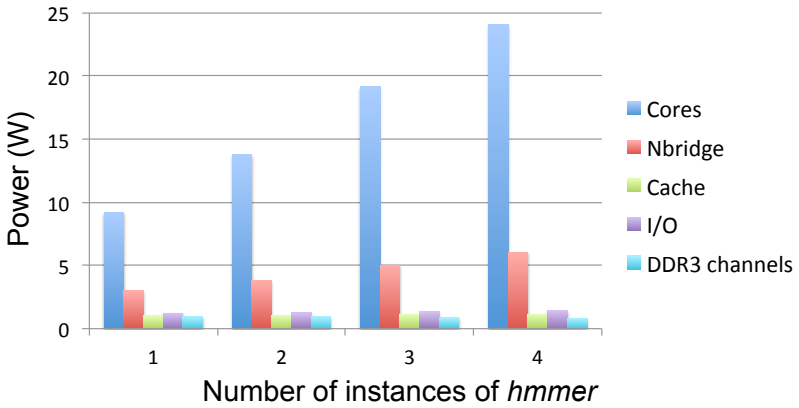

Figure 3.14: Increasing number of instances of *hmmer* in the quad-core processor

c) Total core power consumption over various workloads: To get insight of how the core power consumption varies across different workloads, we plot in Figure 3.15 the percentage of core power to the total power for all 30 test cases. We can see core to total power

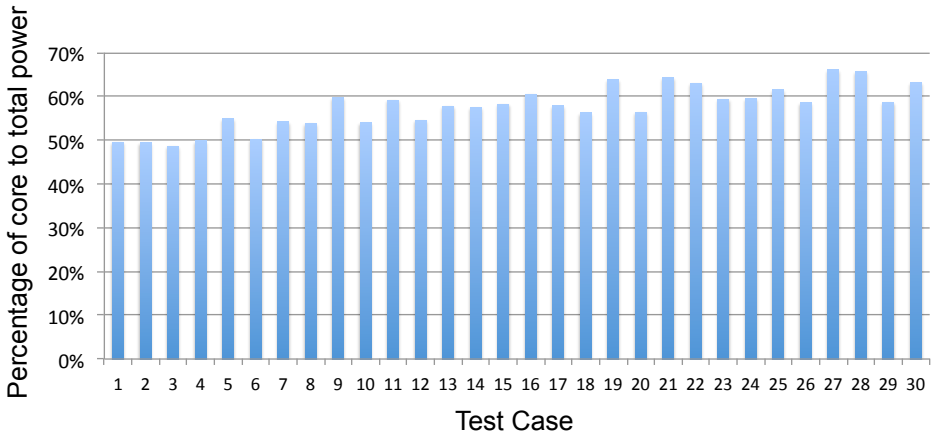

Figure 3.15: Percentage of core power to total power

percentage is high for our higher power test cases, such as, $hmmer$ and $gamess$. As the number of workload increases, the percentage of core to total power varies from 50% to 66% depending on applications' profiles and the number of cores running applications.

Experiment 3. Process Variability and Leakage Power Estimation: To estimate the leakage profile of the AMD quad-core processor, we use the thermal conditioning techniques described in Section 3.2.2, where we increase the chip temperature from 27 °C to 55 °C by increasing the infrared transparent cooling fluid temperature from 18 °C to 45 °C, and measuring the associated changes in power consumption and thermal profiles of the chip using infrared imaging. We divide our chip into small blocks of size about 0.4 mm^2 resulting into approximately 418 first-order and 418 second-order coefficients. In order to maintain the stability of the least square estimation, the maximum number of coefficients i.e. the leakage power resolution is limited by the available number of instances of Equation (3.10). We collected approximately 2000 data points to solve our least square estimation. The total reference leakage power, $\sum p_{ref}$ in Equation (3.9) is estimated by changing the die ambient temperature and using the procedure described in Section 3.2.2.

To uncover the underlying leakage spatial-variability introduced by process variability,

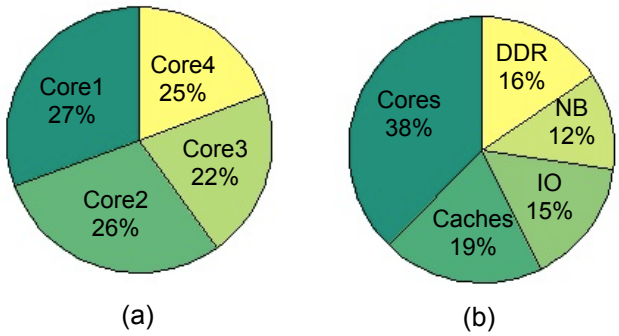

Figure 3.16: a) Percentage leakage power per core with its L2 cache b) percentage leakage power per block type.

we assume constant temperature across the die, and measure the leakage power for each grid location. Figure 3.16.a shows the percentage of leakage power for each core with its L2-cache. Core 1 has approximately 5% more leakage than the lowest power core. This result for instance can be used to bias the operating system scheduler to allocate applications on the lower-leakage cores before the higher-leakage cores. Figure 3.16.b gives the total leakage power distribution among different blocks. There is approximately 10.3% within-die variations among all the blocks. Our results on leakage variability and power-mapping for processors could be used to calibrate design time simulation tools and hence, could be of great use for the architectural community.

Experiment 4. PMC-Based Power Modeling: In our fourth experiment we seek to create empirical models that relate the performance monitoring counters (PMC) to the post-silicon power consumption of each block in the quad-core processor as described in Section 3.2.3. We collect the measurements of 11 PMCs for our quad-core processor using `pfmon` tool. The 11 PMC are listed in Figure 3.17. We compute the correlation coefficient between the measurements of the performance counters and map the power consumption of each block, and we report in Figure 3.17 all the PMC that have strong to good correlation or anti-correlation with power consumption. For example, the number of retired μops (PMC #3), the data cache access (PMC #4), the retired branch instructions

#1	REQUESTS_TO_L2
#2	DISPATCHED_FPU
#3	RETIRED_UOPS
#4	DCACHE_CACHE_ACESSES
#5	L2_CACHE_MISS
#6	MEMORY_REQUESTS
#7	MEMORY_CONTROLLER_REQUESTS
#8	CPU_TO_DRAM_REQUESTS
#9	DRAM_ACCESSES_PAGE
#10	DISPATCH_STALLS
#11	RETIRED_BRANCH_INSTRUCTIONS

block power	correlates	anti correlates
cores	(3, 4, 11, 2)	–
L2 caches	(10, 1, 11, 2, 7)	–
Northbridge + misc	(3, 4, 11, 1, 9, 8, 7, 10, 2)	–
I/O + DDR channels	5	(2, 11, 4, 3)

Figure 3.17: Correlation between performance counters and power consumption of processor blocks.

(PMC #11), the floating point instructions (PMC #2) all provide strong correlation to the power consumption of cores. In case of I/O and DDR channels, the L2 cache misses (PMC #5) provide a strong correlation of power consumption, while PMC #2, #11,#3, #4 provide strong anti-correlation. Notice that these performance counters are strongly correlated with the power consumption of the caches and and cores. That is, when the cores and caches are experiencing high activity, the I/O and DDR channels will experience low activity and vice versa.

Given the measurements of the PMCs and their correlations with the post-silicon power mapping results, we empirically fit a power model for each processor block to its post-silicon estimated power using least-square estimation as described in Section 3.2.3. The input to the power models are the most correlated PMCs as described in the previous paragraph. For instance, we report in Figure 3.18 the power consumption of Core 1 and

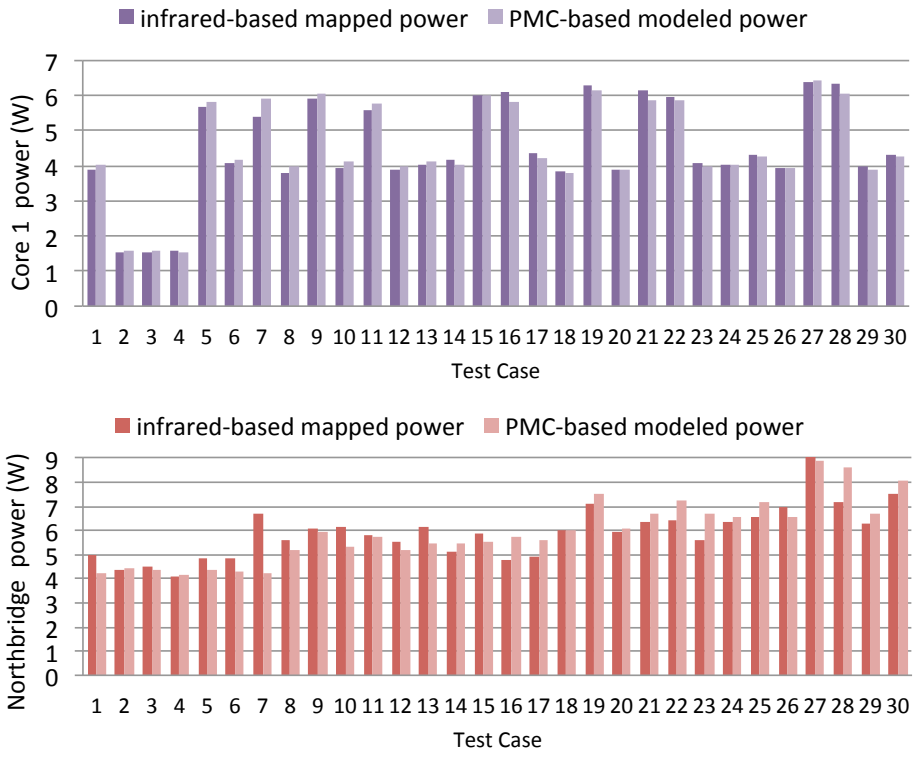

Figure 3.18: Power consumption as estimated by the infrared-based system and the fitted models using the performance counters for the 30 test cases.

the northbridge blocks as estimated by infrared mapping and the fitted PMC models. We notice that the PMC-based fitted models track the power mapping results closely, with a mean absolute error of 2.6% in the case of Core 1 and about 9.2% in case of the north bridge block. To illustrate the use of PMC in transient modeling, we utilize the derived PMC models to estimate the transient power consumption for the different blocks of the processor. Figure 3.19 gives the power consumption for case 28 for the first 120 seconds in execution. We report in blue solid line the sum of power of all cores, the dashed blue line gives the power consumption of the northbridge, while the brown and dashed green lines give the power of IO and L2 caches respectively. Finally, the red line gives the total modeled power and the black line gives the total power form the external multimeter. We note that the PMC-based modeling is able to track the transient response accurately, following the changes in total power consumption.

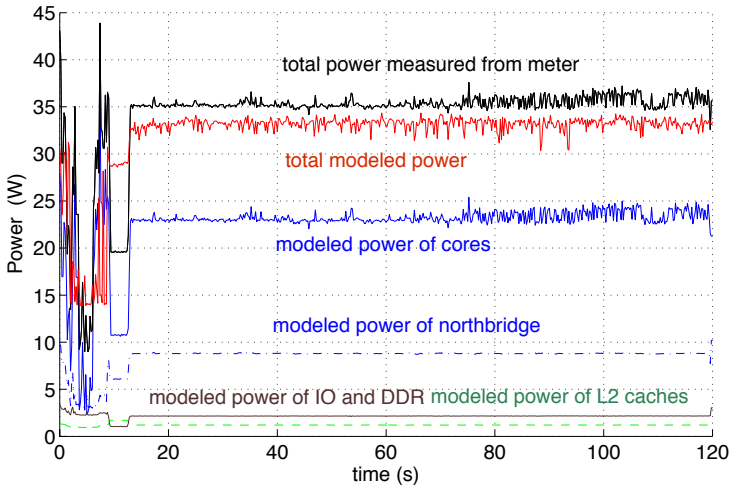

Figure 3.19: Transient power modeling using PMC measurements.

3.4 Power Mapping of a CPU-GPU Processor

CPU-GPU processors have different thermal and power management mechanisms than traditional multi-core CPUs because they have two architecturally dissimilar devices integrated on the same die. In this section, we depict multiple implications of CPU-GPU processors through detailed thermal and power mapping of a real processor.

3.4.1 Experimental Setup

Motherboard Setup. For experiments on a CPU-GPU processor, we use a motherboard with FM2+ socket fitted with an AMD A10-5700 Accelerated Processing Unit (APU) and 8 GB DRAM. The floorplan of the APU is shown in Figure 3.20 [72]. The APU has two x86 modules, containing two CPU cores each and it has 2×2MB L2 caches. The APU has an integrated GPU with 6 single-instruction-multiple-data (SIMD) compute units. The IP blocks surrounding the SIMD units are denoted as GPU auxiliary units because they are active when the SIMD units are active. The area between L2 caches is occupied by unified

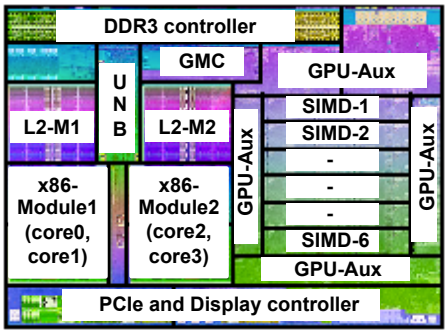

Figure 3.20: Floorplan of the AMD A10-5700 APU.

north bridge (UNB), which acts as an interface between L2 caches and DDR3 controller. The GPU has an additional memory controller, called graphics memory controller (GMC), which is optimized for the graphics related memory requests. In this paper we consider two frequency settings for CPU: 1.4 GHz and 3.0 GHz. The APU and the motherboard used in this study has limited BIOS-level support for frequency scaling in GPU device (only DFS without voltage scaling) and hence, limits the usefulness for the power management. So, we fix the GPU frequency to 800 MHz in our experiments. The latest AMD drivers on newer versions of APUs may support full DVFS on integrated GPU, thus has even more potential than what is shown in this work.

While the basic setup and power mapping framework remains the same when we change the processor and/or the motherboard, the IR-transparent heatsink has to be changed whenever the physical dimensions of the CPU socket and heatsink assembly changes. The A10-5700 APU requires FM2+ socket, so, we rebuilt the IR-transparent heatsink for the CPU-GPU processor. It is worth mentioning that for understanding the implications of CPU-GPU processors, we mainly focussed on the total power of different blocks. Also, we did not have full access to the GPU performance counters, so we did not perform PMC modeling for this processor. In particular, we solve the following constrained least-square-error minimization problem to reconstruct the power map ($\mathbf{p}$) of the die.

$$\mathbf{p}^* = \quad \arg_{\mathbf{p}} \min \|\mathbf{Rp} - \mathbf{T}\|_2, \tag{3.13}$$
$$\text{s. t.} \quad p_i \geq 0.$$

where, $\mathbf{p}^*$ is the reconstructed power-vector, p_i denotes the power in the i^{th} block of the die. By solving the above optimization problem, we obtain the total power of each block for the die. Using, $p_i > 0$ constraint helps in ensuring that reconstructed power for all blocks is always positive.

Benchmarks. We use the following workloads (mainly OpenCL) to cover a wide range of characteristics. First, in order to fully control the workload distribution between CPU and GPU devices, we wrote a simple custom micro-kernel (μKern) in OpenCL that multiplies two vectors of arbitrary size for a given number of times. We use multiple iterations inside the kernel so that the die reaches a stable thermal state, which improves the reproducibility of thermal/power results. The micro-kernel is a *homogeneous* type of workload because once it is launched on the GPU, the CPU is completely idle and vice-versa.

As a representation of real-life CPU-GPU workloads, we selected six OpenCL workloads from publicly available Rodinia benchmark suite [14]. In particular, we selected (CFD) solver from computational fluid dynamics, breadth-first search (BFS) from graph algorithms, Needleman-Wunsch (NW) from bioinformatics, Gaussian Elimination (GE) from linear algebra, stream cluster (SC) from data mining, and particle filter (PF) from the medical imaging domain. Unlike μKern, these benchmarks have multiple kernels, and when a particular iteration of these kernels is running on GPU, CPU could be preparing data for the next kernel launch. Therefore, they are also called as *heterogeneous* benchmarks. Among the selected heterogeneous benchmarks, BFS and PF benchmarks have high branch-divergences, so they perform better on CPU than GPU. Further, when run on GPU with CPU as host, CFD, GE, and PF have low CPU-boundedness, defined as the

proportion of total runtime spent on the CPU device, while others spend large portion of the total runtime on the CPU device.

When an OpenCL benchmark is launched on CPU, it uses all available cores of the CPU. To contrast the OpenCL workloads against CPU workloads and also to show the impact of core-level scheduling decisions on a traditional CPU, we use single-threaded benchmarks from the SPEC's CPU2006 benchmarks-suite [101]. When a SPEC benchmark is launched on CPU, it uses only one out of the four CPU cores.

3.4.2 Results

In this subsection, we present multiple implications of integrated CPU-GPU processors on thermal and power management techniques through experiments on a real processor.

a. Scheduling on Multi-core CPU versus CPU-GPU Processor

In a traditional multi-core CPU, workload scheduling is done at operating system (OS) level using system level commands like *taskset*. However, in CPU-GPU processors, the OpenCL framework provides application-programming interfaces (APIs) to control the compute devices; it requires the programmer to distribute the work among different devices. This inherently different nature of scheduling, combined with architecturally different devices, leads to different thermal and power profiles in CPU-GPU processor than the traditional CPU-only processor.

Figure 3.21 depicts the outcome of these two scheduling choices: termed as purely OS-based and application-based scheduling here, with the help of thermal maps for a SPEC CPU benchmark (`hmmer`) and an OpenCL workload (`NW`). The floorplan of the processor

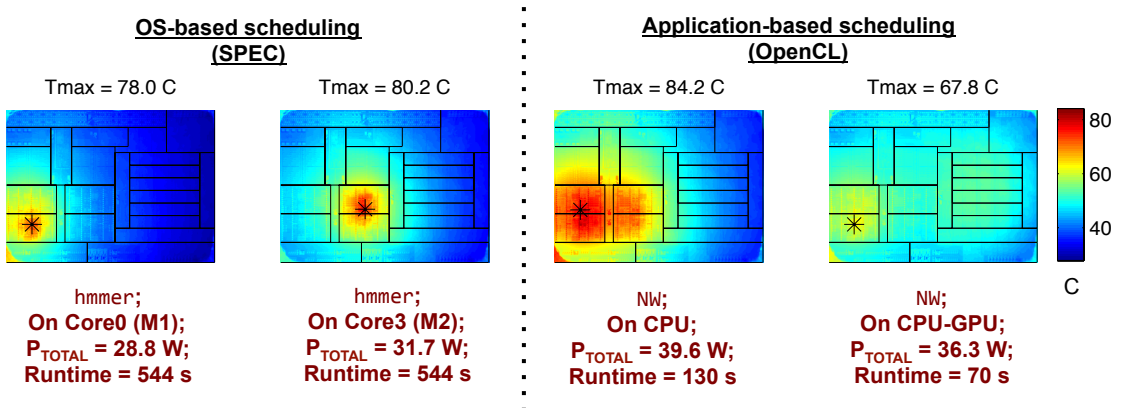

Figure 3.21: Scheduling techniques: OS-based scheduling of a SPEC CPU benchmark (`hmmer`) and application-based scheduling of an OpenCL benchmark (`NW`).

is also overlaid on these thermal maps. The die-shot with floorplan information of the processor was shown earlier in Figure 3.20. From the Figure 3.21, we observe that with the help of OS, the `hmmer` benchmark could be launched on one of the CPU cores as shown in the thermal maps in first two columns. As expected, the thermal hotspots are primarily located in the active cores, e.g. on core0 and core3 in these maps. On the other hand, with the help of OpenCL runtime, the application-based scheduler could launch the kernels of `NW` workload on both CPU and GPU devices. The thermal maps from such scheduling are shown in the 3^{rd} and 4^{th} column in Figure 3.21. We notice that the thermal profiles of OpenCL workload are different. In particular, when the OpenCL kernels are launched on CPU, they uses all four cores of the CPU; so, the thermal hotspots in this case are spread over all cores (as shown in the thermal map in 3^{rd} column). Further, when the kernels are launched on GPU, the hotspots are spread over both CPU and GPU, with CPU being hotter than GPU due to its higher power density. This is also because, in the `NW` benchmark, when the GPU is running a particular iteration of kernels, the CPU is preparing work for the future iterations, keeping both CPU and GPU devices active simultaneously.

Further, from Figure 3.21, we observe that the application-based scheduling on a CPU-

GPU processor leads to larger range of the peak temperature (84.2 °C and 67.8 °C) and total runtime (130 s and 70 s) by running workloads on CPU and GPU devices than the CPU-only scheduling from the OS. Hence, we observe that although both the OS and application-based scheduling affect the thermal hot spot locations, the application-based scheduling inherently takes advantage of the heterogeneity in a CPU-GPU system and achieves a more thermal friendly and power efficient scheduling. In summary, scheduling on a CPU-GPU processor creates tight interaction between the application and the power management unit which has a potential implication as follows.

Implication 1: *A scheduling scheme could be implemented on CPU-GPU processors wherein the thermal/power management unit, OS and application can make the scheduling decisions in cohesion.*

b. Interplay of Scheduling and DVFS

To demonstrate the interactions between DVFS and scheduling on CPU-GPU processors, we analyze the thermal, power and performance characteristics of a heterogeneous OpenCL workload (`CFD`) launched on the AMD APU. Figure 3.22 shows thermal maps and their corresponding power-breakdowns for two scheduling cases (GPU/CPU) and two DVFS settings (1.4/3.0 GHz) for the `CFD` benchmark. The power breakdown for different cases are shown in the pi-charts. As expected, the power in x86 modules is higher when the kernel is launched on CPU, while the power in GPU units is higher when it is launched on GPU. The figure also shows the peak temperature, total power and runtime for each case. We notice that the power, performance, and thermal profiles are different in different cases. In particular, among the four options, two DVFS settings and two scheduling choices, the total power and the peak temperature for the OpenCL `CFD` benchmark vary up to 23.4 W and 40.5 °C, while the performance varies by a factor of 10.5×.

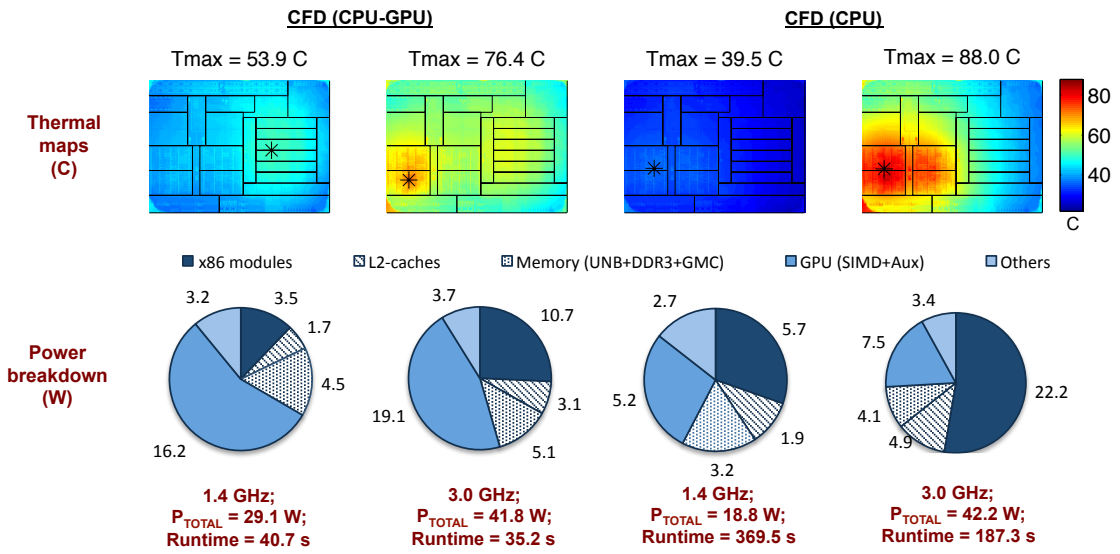

Figure 3.22: Thermal and power maps showing the interplay between DVFS and scheduling for the CFD benchmark. The peak temperature, power and runtime are significantly different for different DVFS and scheduling choices.

Further, DVFS and scheduling, collectively, have strong effect on the location of thermal hotspots. For example, when the CFD kernels are launched on the CPU, as expected, CPU DVFS has negligible impact on the location of thermal hotspots on the die (column 3 and 4 in Figure 3.22). However, when the kernels are launched on GPU, the sequential part of the workload still runs on one of the CPU cores. So, in some cases, for example in the thermal map shown in column 2 of Figure 3.22, the thermal hotspot could be located in the x86 module even though the parallel kernels are running on GPU. This is because the maximum operating frequency of CPU cores is higher than GPU compute unites due to their deeper pipelines and smaller register files than GPU. Also, power has a super linear relationship with the operating frequency/voltage ($\propto fV^2$), therefore, CPU cores typically have higher power density than GPU at higher frequency. Hence, as shown in the thermal maps in Figure 3.22, the location of thermal hotspot for the application-based scheduling on a CPU-GPU processor is dependent on both CPU DVFS and scheduling choices.

The strong intertwined behavior of DVFS and scheduling on performance and power does not exist in traditional multi-core processors. For example, as was shown earlier in

Figure 3.21, when the `hmmer` (SPEC) benchmark is launched on different CPU cores although the location of thermal hot spot shifts (which has ramifications on thermal management), the performance, peak temperature and total power does not change significantly because all four CPU cores have the same micro-architecture. The slight differences in the total power (28.8 W versus 31.7 W) and die temperature (78 °Cversus 80.2 °C) are because of differences in leakage power arising from differences in relative proximity of these cores to the GPU units. We observed similar behavior on other representative SPEC CPU benchmarks, namely `omnetpp` (memory-bound integer-point), `soplex` (memory-bound floating-point), `gamess` (compute-bound floating-point) also. In summary, the power management for regular multi-cores is simpler than heterogeneous processors, because DVFS and scheduling can be considered independently; however, the behavior of DVFS and scheduling is intertwined from a thermal perspective. Weak interactions on power occur because of the thermal coupling on the die. However, DVFS and scheduling techniques have greater impact on performance and thermal/power profiles of CPU-GPU processors for OpenCL workloads, which can be fluidly mapped to the CPU or GPU. We summarize this discussion as the following implication.

Implication 2: *DVFS and scheduling must be considered simultaneously for the best runtime, power, and thermal profiles on CPU-GPU processors.*

c. Workload-Dependent Scheduling and DVFS Choices

Different OpenCL workloads have different characteristics, e.g. branch divergences behavior and the proportions of work distributed between CPU and GPU devices. Therefore, the optimal scheduling and DVFS choice for performance and power/temperature varies across different workloads. Below, we provide the optimal scheduling and DVFS choices for the selected heterogeneous workloads.

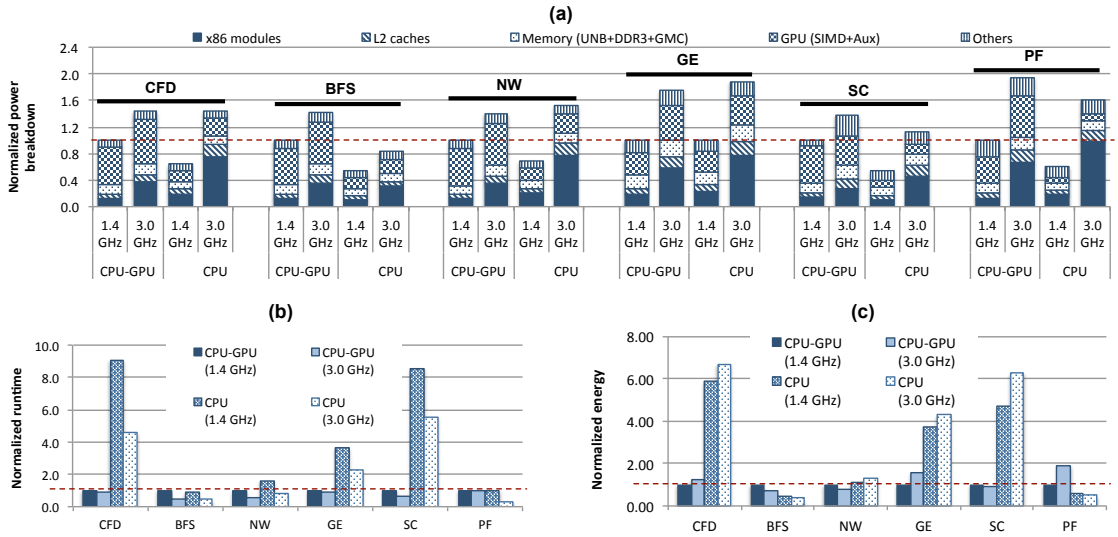

Figure 3.23: Normalized power breakdown (a), runtime (b), and energy (c) for 6 heterogeneous OpenCL benchmarks executed on CPU-GPU and CPU devices at two different CPU DVFS settings (normalization with respect to "CPU-GPU at 1.4 GHz" cases).

Power and Temperature Minimization. In Figure 3.23.a, we show the breakdown of the total power for the selected heterogeneous OpenCL benchmarks under different scheduling and DVFS conditions. The power values are normalized with respect to the total power in "1.4 GHz CPU-GPU" case for each benchmark. As expected, we notice that for all benchmarks the average total power is the lowest when they are launched on CPU at 1.4 GHz. Similarly, although not shown in the figure, for all benchmarks the peak die-temperature is the lowest when they are launched on CPU at 1.4 GHz. This is expected because for the "1.4 GHz, CPU" case, CPU frequency is the lowest and GPU is idle; so, both CPU and GPU dissipate the least amount of power. In all other cases, either CPU will dissipate higher power or both CPU and GPU will dissipate power.

Further, we notice that the irregular benchmarks (e.g., `BFS`) with better power efficiency on CPU, could dissipate unnecessarily high power if run on CPU-GPU. This is also because the current GPUs do not have fine grained (SIMD or CU-level) power gating. So, when a workload with irregular branches is launched on GPU, only a portion of SIMDs would be doing the useful work and others would be idle, dissipating unnecessary leakage

power. The recent ideas related to fine-grained power gating to reduce leakage power in GPUs would be quite useful in such cases [114]. We summarize these observations as the following implication.

Implication 3: *Running workloads on CPU device at the lowest DVFS setting provides minimum power and peak temperature because power gating GPU is more power efficient than keeping both CPU and GPU active at low CPU DVFS.*

Runtime and Energy Minimization. Figure 3.23.b and 3.23.c illustrate the performance and energy results of the selected heterogeneous OpenCL workloads at different scheduling and DVFS settings. The optimal DVFS and scheduling choices for minimizing runtime, energy, along with power, are summarized in Table 3.4. Among them, the energy or runtime results are more interesting. We notice that the optimal scheduling for minimizing energy and runtime are typically the same, but the optimal DVFS settings for minimizing energy and runtime could be different. In other words, we make following two observations from the results shown in Table 3.4.

1. *If a particular scheduling choice minimizes runtime, it also minimizes the energy.* From the Table 3.4, we observe that running `BFS` and `PF` on CPU leads to both optimal energy and runtime; similarly `CFD`, `NW`, `GE`, and `SC` lead to lower energy and runtime when run on GPU with CPU as the host device. This behavior is ob-

Table 3.4: Optimal DVFS and scheduling choices to minimize power, runtime, and energy for the selected heterogeneous OpenCL workloads.

Workload Name	Minimum Power/Temp	Minimum Runtime	Minimum Energy
CFD	1.4 GHz, CPU	3.0 GHz, CPU-GPU	1.4 GHz, CPU-GPU
BFS	1.4 GHz, CPU	3.0 GHz, CPU	3.0 GHz, CPU
NW	1.4 GHz, CPU	3.0 GHz, CPU-GPU	3.0 GHz, CPU-GPU
GE	1.4 GHz, CPU	3.0 GHz, CPU-GPU	1.4 GHz, CPU-GPU
SC	1.4 GHz, CPU	3.0 GHz, CPU-GPU	3.0 GHz, CPU-GPU
PF	1.4 GHz, CPU	3.0 GHz, CPU	3.0 GHz, CPU

served because BFS and PF have high control-divergences, so they are more suited for the CPU architecture; running them on GPU mean both CPU and GPU would consume power, with GPU providing less performance. On the other hand, the other benchmarks have high parallelism, so GPU is more power efficient for them.

2. *The energy of CPU-GPU benchmarks with low CPU-boundedness could be minimized by reducing the CPU frequency.* We observe that the energy of CFD and GE is the lowest at low CPU frequency (1.4 GHz). This is because CFD and GE have low CPU-boundedness, measured by the relative portion of the work executed on CPU when the workload is launched on GPU. So, the performance improvement from increasing the CPU frequency does not compensate for the corresponding increase in power for these benchmarks. On the other hand, NW and SC have the lowest energy at high CPU-frequency (3.0 GHz). This is because NW and SC have high CPU-boundedness, so, increasing the CPU frequency improves their performance significantly. We summarize these results through following implication.

Implication 4: *The optimal DVFS and scheduling choices for minimizing runtime and energy on a CPU-GPU processor are functions of workload characteristics.*

d. Asymmetric Power Density of CPU-GPU Processor

Typically, power dissipation in GPU and CPU devices for the same OpenCL kernel is different due to differences in their architectures and operating frequencies. Further, for the studied heterogeneous processor, GPU occupies larger die-area than the CPU, and therefore, it has lower power density than CPU for the same total power. In this section, we confirm the asymmetric power density of two device experimentally. Although it is difficult to make a circuit block to dissipate certain amount of power in a real processor, the homogeneous μKern, which keeps only one device active at a time, is used to analyze

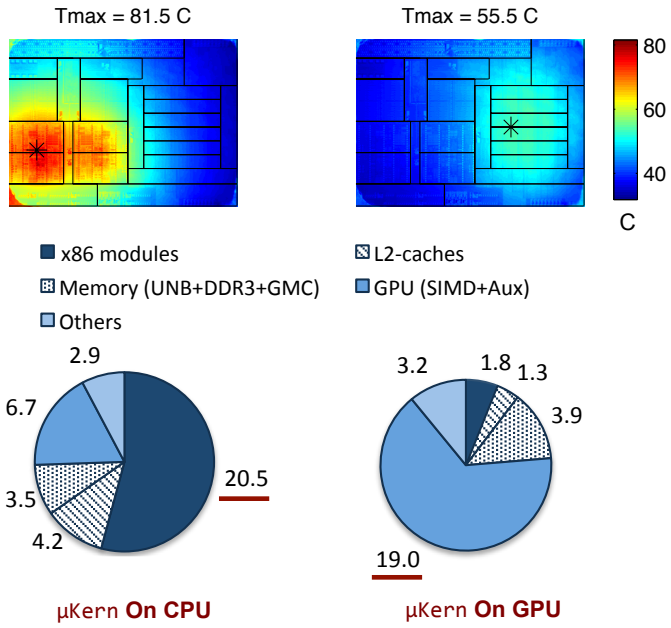

Figure 3.24: Thermal and power maps demonstrating asymmetric power density of CPU and GPU devices. μKern is launched on CPU and GPU devices. For the comparable power on CPU (20.5 W) and GPU (19 W), the peak temperature on CPU is about 26 °C higher than on GPU.

the power densities of the two device. Figure 3.24 shows the thermal and power maps of the die when we launch the μKern on CPU and GPU devices at fixed DVFS (3 GHz). From the pi-charts, we observe that the power consumption of CPU in column-1 (20.5 W) is comparable to the power consumption of GPU in column-2 (19 W). However, from the thermal maps, we notice that the peak temperature in two cases are significantly different; in particular, the peak temperature of CPU is higher than that of GPU by 26 °C . We computed the power density of CPU and GPU in two cases and found that the power density of CPU is $2.2\times$ higher than that of the GPU. Therefore, even for the comparable amount of power, CPU has higher peak temperature than the GPU.

Further, due to higher power density of CPU than GPU, it is possible that the thermal hotspot could be located on CPU even though the OpenCL kernels are launched on GPU. This is because when a kernel is launched on GPU, CPU acts as its host device; so, the CPU could also be active to prepare the work for the next iteration of kernel launch. This

could be confirmed from the thermal map shown earlier in the column-2 of Figure 3.22. We notice that, at 3 GHz DVFS, the peak temperature is located on CPU blocks even though the kernels are launched on GPU. More importantly, we observe that, at higher DVFS, the likelihood of CPU reaching the thermal limit first is higher than that of GPU. Although at low DVFS (e.g., 1^{st} column of Figure 3.22), the hotspot may be located on GPU blocks, but the peak temperature of GPU in that case is lower than the thermal limits. The higher temperature of GPU in this case would lead to higher leakage power, but the peak temperature of GPU will still be below the thermal limit. The asymmetric power density and its effect on thermal profiles of the CPU-GPU processor, as discussed above, could have multiple implications on the thermal and power management of the system. Few of them are as follows.

Implication 5: *Due to lower peak temperature in GPU, it could have fewer number of thermal sensors per unit area than the CPU.*

Implication 6: *The extra thermal slack available on the GPU could be used to improve its performance through frequency-boosting, provided it meets all architectural timings, like register file access time.*

Implication 7: *One could design a localized cooling solution (e.g., thermo electric cooler based) for separate and efficient cooling of CPU and GPU devices on such processors.*

e. Leakage Power-Aware CPU-Side Scheduling

Here, we demonstrate the importance of scheduling the sequential part of an OpenCL CPU-GPU workload on an appropriate core of the CPU. Typically, a CPU-GPU processor has multiple cores on the CPU side. So, when a workload is launched on GPU with CPU as the host device, it could be launched from any of the available CPU cores. Since the

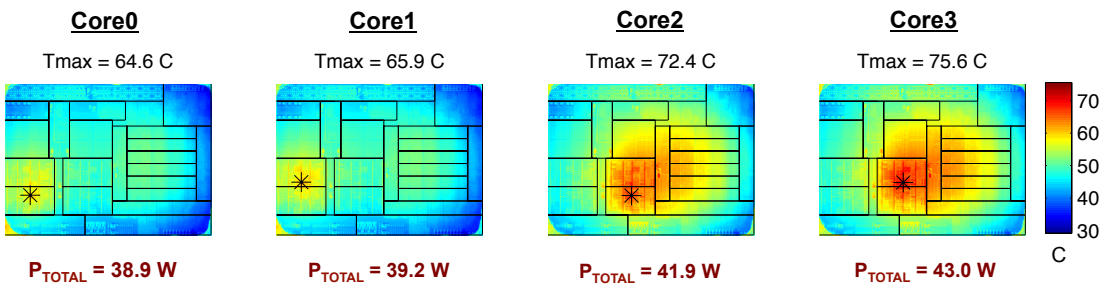

Figure 3.25: Impact of CPU core-affinity when a benchmark (SC) is launched on GPU from different CPU cores at fixed DVFS setting.

cores in x86 module-2 (M2) are in close proximity to GPU than the cores in x86 module-1, the thermal coupling, and therefore, the leakage power is different in each case.

To understand the differences in thermal and power profiles, we launched a heterogeneous benchmark (SC) on GPU from 4 different cores of the CPU. Figure 3.25 shows the thermal maps of the die in all 4 cases. We observe that the thermal and power profiles of the chip is indeed different for different core-affinities. Specifically, the total power when the workload is launched from Core0, 1, 2, and 3 are 38.9 W, 39.2 W, 41.9 W, and 43 W, respectively; while, the corresponding peak die temperature values are 64.6 °C, 65.9 °C, 72.4 °C, and 75.6 °C, respectively. Hence, we notice that the total power and peak temperature of the die is higher (by 4 W and 11 °C, respectively) when the benchmark is launched from Core3 than when it is launched from Core0. This happens because Core3 is in close proximity to GPU; so, there is stronger thermal coupling between Core3 and GPU than between Core0 and GPU. The strong thermal coupling leads to higher temperature and leakage power in both CPU and GPU. So, it is important to launch the kernels from an appropriate CPU core. We encapsulate this observation in the following implication.

Implication 8: *The OS or the CPU-side scheduler should use the floorplan information of the processor to launch a workload on GPU from an appropriate CPU-core to reduce both, the peak temperature and the leakage power of the chip.*

3.5 Summary

In this chapter, we analyzed the power consumption of different blocks of a quad-core CPU processor and a heterogeneous CPU-GPU processor. Our results reveal a number of insights into the make-up and scalability of power consumption in modern processors. We also devised accurate empirical models that estimate the infrared-based per-block power maps using the PMC measurements. We used the PMC models to accurately estimate the transient power consumption of different processor blocks of a multi-core CPU processor.

CPU-GPU processors are becoming mainstream due to their versatility in terms of performance and power tradeoffs. In this chapter, we showed that the integration of two architecturally different devices, along with the OpenCL programming paradigm, create new challenges and opportunities to achieve the optimal performance and power efficiency for such processors. With the help of detailed thermal and power breakdown, we highlighted multiple implications of CPU-GPU processors on their thermal and power management techniques. For the studied CPU-GPU processor, among different frequencies and two devices, the performance could vary up to 10.5×, while the total power and peak temperature vary up to 23.4 W and 40.5 °C , respectively. We showed that DVFS and scheduling must be considered simultaneously for the best runtime, power, and thermal profiles on CPU-GPU processors. In the next chapter, we discuss the workload scheduling on CPU-GPU processors in greater detail.

Chapter 4

Workload Characterization and Mapping on CPU-GPU Processors

4.1 Introduction

Heterogeneous processors, with integrated CPU and GPU devices, offer great balance between performance and power efficiency for a wide range of applications [68]. Furthermore, they eliminate many of the overheads associated with the communication between discrete CPU and GPU devices. CUDA [79] and OpenCL [75] are two prominent parallel computing frameworks that allow the programmer to run kernels/workloads on GPU devices. While CUDA only supports GPU devices from NVIDIA, OpenCL could be used to run kernels on both CPU and GPU devices from multiple vendors. In particular, the latter provides low-level APIs to choose a device. Currently, the programmer decides the device for an application statically at the development time based on profiling results, and the operating system (OS) together with OpenCL Runtime schedules the application

on the chosen device. However, such a static scheme may not lead to an appropriate device selection for all kernels because different kernels may have different preferred devices based on the data size and kernel characteristics [87]. Furthermore, this scheduling decision seldom considers the run-time physical conditions [e.g., thermal design power (TDP) and CPU workload conditions]. Since the TDP can vary depending on the processor model, available battery power, and user preferences for power management, and similarly, CPU load could vary depending on the number of cores occupied by other workloads, the existing schemes may lead to potentially inefficient scheduling under dynamic system conditions [6, 110, 36, 16].

Modern processors incorporate hardware (HW) mechanisms to dynamically enforce desired TDP budgets. However, by their own nature, these mechanisms only leverage "knobs" available to the hardware (e.g., clock gating, cycle skipping and dynamic frequency-voltage settings), and they are not able to control the scheduling of applications. In this chapter, we show that the best runtime and energy scheduling choice for a parallel kernel depends on the TDP of the chip and on the individual kernel characteristics. Thus, when TDP changes during runtime, it can be beneficial to override the static preference and re-map a parallel kernel to a different device (e.g., from GPU to CPU or vice versa). Similarly, under a fixed TDP budget, the suitable device for a kernel depends on number of available cores on CPU; if one or more cores of CPU become busy with other workloads, then the best device could change from CPU to GPU device. To address these challenges, we propose a hardware status-aware scheduler to dynamically map parallel kernels to the best device such that runtime or energy is minimized. Few prior work proposed scheduling at application-level [36, 16]. Others advocate frameworks that do not take into consideration the time-varying system conditions, such as TDP budget and run-time workload conditions [87, 30, 5, 110, 6]. This chapter makes the following contributions to address the issues with existing scheduling methods.

- We develop a workload scheduling framework for CPU-GPU processors that makes the scheduling decisions to minimize runtime or energy by taking run-time conditions (e.g., TDP and number of available CPU cores) into account. Unlike previous works [6, 110], which were either static in nature or mainly used compile-time workload-characteristics, we profile the workloads online and make appropriate device decisions under varying system conditions. We show that such scheduling provides better performance and higher energy savings compared to the state-of-the-art application scheduling.

- We monitor the run-time physical and existing workload conditions by reading model specific registers (MSRs) for chip TDP and the performance counter values. We then use a support-vector machine (SVM) classifier that uses these run-time system conditions along with the workload-specific performance monitoring counters as features to predict the appropriate device on CPU-GPU processors. The SVM classifier is trained using measurements on the target hardware. While the existing schedulers focus on runtime minimization, our scheduler could provide efficient scheduling for both runtime and energy, depending on the user preference.

- We implemented our proposed framework as a computationally light-weight power management tool that extends HW-based TDP enforcement capabilities to include CPU/GPU scheduling. We tested our tool on a real state-of-the-art CPU-GPU processor-based system using OpenCL benchmarks. For the studied benchmarks, we show that the proposed kernel-level scheduling provides up to 40% and 24% better performance than the static developer-based scheduling choice and the state-of-the-art scheduling schemes, respectively. Further, for the studied TDP traces, our scheduler provides up to 10% higher energy savings than the developer-based energy minimization scheduling choice..

The rest of the chapter is organized as follows. We provide the motivation of this work in section 4.2. In section 4.3, we describe the proposed framework for online workload characterization and mapping on CPU-GPU processors in detail. The experimental setup and the runtime and energy improvement results from the proposed scheduling schemes are presented in section 4.4. Finally, we summarize the chapter in Section 4.6.

4.2 Motivation

In this section, we motivate the need for a run-time physical and workload conditions-aware (in particular TDP and CPU load-aware) scheduler for optimizing performance or energy on CPU-GPU processors. First, we demonstrate the impact of TDP on scheduling decisions. Then, we highlight the interplay between workload characteristics, CPU-load and TDP on device decisions where the CPU-load denotes the number of CPU-cores busy running other background workloads. Thirdly, we also show the advantage of kernel-level scheduling instead of application-level scheduling for improving performance and power efficiency of CPU-GPU processors.

a. Need for TDP-Aware Scheduling

Here, we discuss the impact of TDP budget on workload scheduling for CPU-GPU processors. By definition, TDP denotes the maximum power of the chip that can be handled by the cooling system. In recent processors from Intel and AMD, a configurable TDP (cTDP) feature is introduced to handle different usage scenarios, available cooling capacities and desired power consumptions. As the TDP could change dynamically for saving battery life or adapting to the user behavior, we study the effect of changing chip TDP on the scheduling decisions.

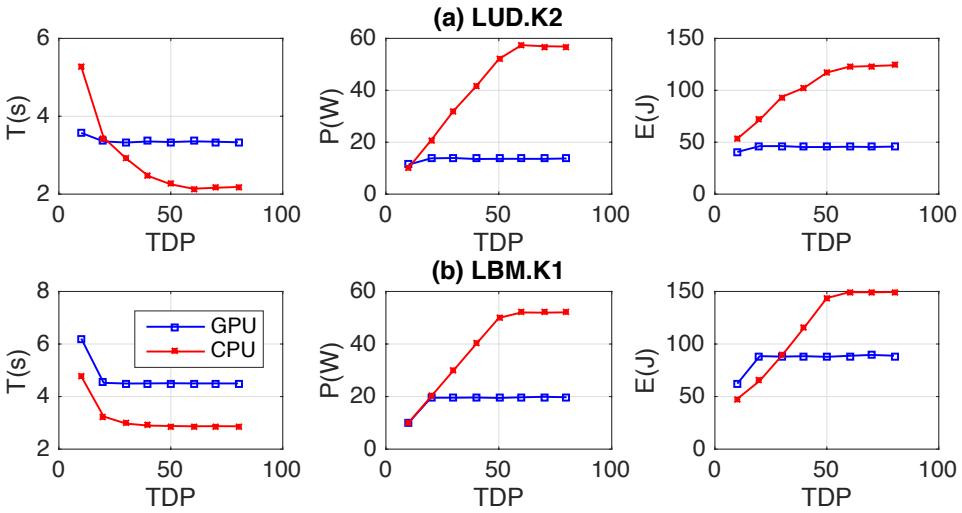

Figure 4.1: Energy, power and runtime versus package TDP for two benchmarks: (a) `CUTCP` and (b) `LBM` on GPU and CPU devices of an Intel Haswell processor.

Figure 4.1 shows runtime, power and energy versus TDP trends for two application kernels (`LUD.K2` and `LBM.K1`) that we observed on a heterogeneous processor. The total power either follows the TDP or saturates based on the device type and the workload characteristics. Runtime and energy trends are more interesting; in particular, we observe two types of trends in runtime and energy. In one trend the runtime/energy of the `LBM.K1/LUD.K2` kernels are always lower on GPU or CPU devices irrespective of the TDP, while in the second trend (runtime for the `LUD.K2` kernel and energy for the `LBM.K1` kernel) the optimal device is a function of its TDP. These trends arise because of the kernel characteristics and the differences in maximum operating frequency and architectures of CPU and GPU devices in a CPU-GPU processor. More specifically, by the nature of its design, CPU is more aggressively pipelined than GPU to reduce latency, and as a result, GPU reaches its maximum frequency before the maximum TDP of the package, while the CPU can increase its frequency (and thereby dissipates higher power) until it reaches the maximum TDP, as shown in the power versus TDP plots. At this point, the CPU consumes large power (due to super-linear relationship between frequency and power) that it is not the most energy-efficient even though it may deliver lower runtime.

On the studied processor, the maximum GPU frequency is 1.25 GHz, while the maximum CPU frequency is 4 GHz. In general, the occurrence of such energy cross-over with respect to TDP is a function of runtime and power behavior of a kernel on two devices. For example, the optimal device for energy could be different at low and high TDPs when a kernel runs faster on CPU than GPU with a speedup less than the power ratio between CPU and GPU devices (e.g, `LBM.K1` in Figure 4.1).

b. Need for CPU Load-Aware Scheduling

Here, we demonstrate that the best scheduling choice not only depends on the TDP budget, but also depends on the run-time conditions on the CPU cores. In particular, we show in Figure 4.2 the scheduling maps that minimize runtime for two kernels (`LUD.K2` and `LBM.K1`) when number of CPU cores available to the kernel are varied from 1 to 4. Here, we vary the number of CPU cores available to the OpenCL kernel to simulate the effect of different CPU load conditions in the system. Further, in this work, we assume that no two workloads run on the same core at a time. This is a reasonable assumption because in a real system a scheduler would always try to run a workload on the free or available cores. Also, by having the OpenCL and non-OpenCL workload run on different cores of CPU, the interference between them is minimized, which makes the scheduling problem tractable. In particular, we demonstrate the following effects through the scheduling maps shown in Figure 4.2.

First, we show that, at a fixed TDP (say 80 W), as the number of CPU cores available for the kernel reduces, the best device for runtime could change from CPU to GPU. This is because the compute throughput of CPU reduces with decrease in number of CPU cores, which in turn increases the kernel runtime on CPU. This is an expected behavior, but it has implications on the scheduling decisions. For example, at 80 W TDP, when all cores of CPU are available, the runtime of `LUD.K2` kernel on GPU and CPU are 3.3 s and 2.2 s,

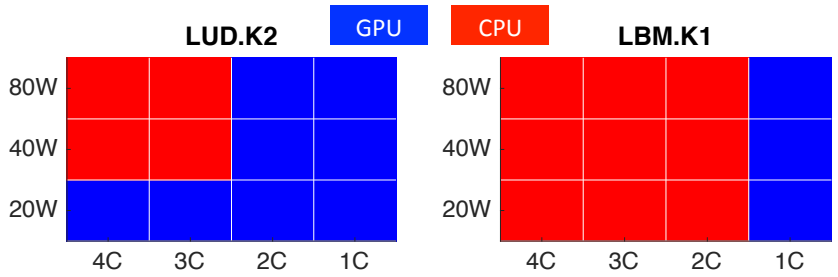

Figure 4.2: Runtime-optimal devices for two kernels (`LUD.K2` and `LBM.K1`) at 3 different TDPs (20, 40, 80 W) and 4 different number of CPU-cores (1C to 4C) for an Intel Haswell processor.

making CPU a favorable device. However, if 3 out of 4 CPU cores become busy (say with SPEC workloads), the kernel runtime on GPU remains the same, but the runtime on CPU becomes 7.7 s, making GPU a favorable device. This example shows that we need a scheduler that takes in to account the current CPU load while making scheduling decisions between CPU and GPU devices.

Further, in Figure 4.2, we also show the interplay among kernel-characteristics, TDP, and the number of CPU cores available to the kernel. From the scheduling maps in Figure 4.2, we notice that the impact of TDP (also shown earlier in Figure 4.1) and the number of available CPU cores on performance is higher for the `LUD.K2` kernel than for `LBM.K1`. Therefore, as the number of the available CPU cores reduce from 4 to 2, the runtime optimal device for `LBM.K1` is still CPU, while it changes to GPU for the `LUD.K2` kernel. Similarly, as the TDP changes, the performance of `LUD.K2` kernel is affected more than that of `LBM.K1` kernel (see Figure 4.1 for exact trends). Hence, we demonstrate that both TDP and existing workload conditions should be considered simultaneously for making best scheduling decisions on a CPU-GPU processor. Its worth mentioning that in this work, we only study the co-location of workloads on CPU cores while keeping the number of execution units (EUs) on GPU constant; this is because the processor used in our experiments does not support changing the number of EUs on the GPU device.

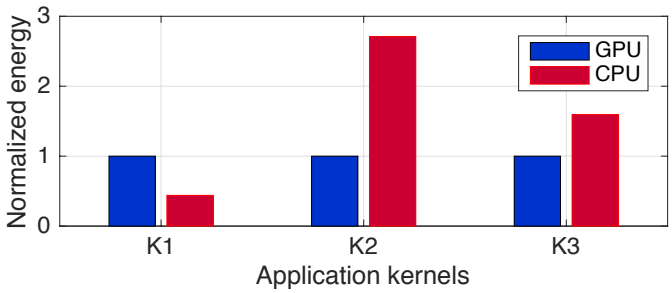

Figure 4.3: Energy of different kernels (K1-K3) of the `LUD` application on CPU and GPU at 60 W TDP.

c. Need for Kernel-Level Scheduling

In Figure 4.3, we show the energy of different kernels of a sample OpenCL application (`LUD` with 3 kernels) on GPU and CPU devices. Here, we observe that different kernels consume different energy on the two devices, In particular, we observe that kernel K1 consumes less energy on CPU, while kernels K2-K3 have lower energy on CPU due to different kernel behaviors. Similar trends could occur in runtime [110]. Therefore, we need a kernel-level scheduler instead of simple application-level scheduler [36, 16] to minimize runtime and energy for better energy efficiency.

Motivated by these observations, we present in the next section a framework that provides kernel-level, hardware status-aware runtime/energy minimization scheduling for CPU-GPU processors during run-time.

4.3 Proposed Methodology

In this section, we describe the proposed framework and the machine learning-based models used to achieve kernel-level run-time hardware status-aware scheduling in detail.

Framework Architecture. The high-level organization of our framework is given in Fig-

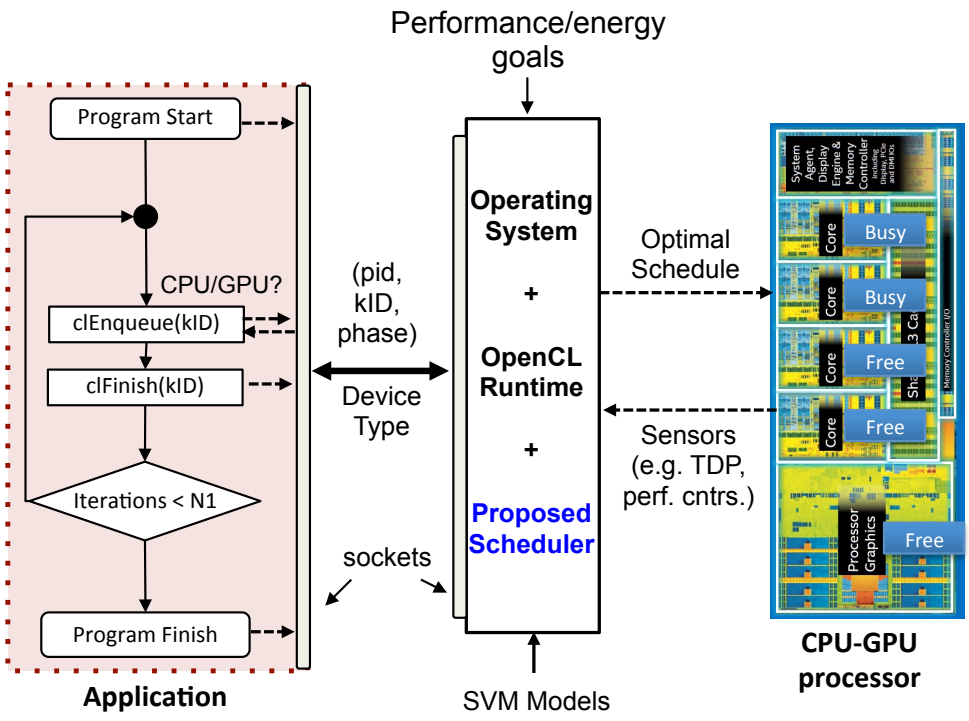

Figure 4.4: Block diagram of the proposed scheduler for CPU-GPU processors.

ure 4.4. The figure shows the OS, OpenCL Runtime and the proposed scheduler as a single unit because they act as an interface between the processor and the application. Custom APIs, independent of actual application, are implemented in C language to exchange information between the application and the scheduler using efficient UNIX sockets. These APIs have negligible overhead on the actual application as demonstrated by our results later in the paper. To make appropriate decision for all kernels of an application, the scheduler keeps track of the current phase for each kernel by exchanging kernel-id (kID), phase (e.g., kernel-enqueue), and process-id (pid) information with the application. The scheduler monitors the hardware performance counters (PMCs) and input these measurements to an *a priori* trained SVM classifier to predict the appropriate device (or class) that minimizes energy or runtime for the kernel. These models for the classifier are trained offline using performance counter measurements collected from different applications executed at multiple TDP values and under different CPU-load conditions, which could change when one or more cores are busy running other workloads. In Figure 4.4, the CPU cores busy

with other workloads are marked as "Busy" and those available for the OpenCL kernel are marked as "Free". During run-time, the scheduler uses the SVM model to classify the kernel in to one of the two classes (GPU or CPU), as described below.

Kernel Classification. In order to make an optimal device-decision, it is sufficient to predict the relative ratio of energy or runtime of each device (CPU vs GPU) for a kernel and therefore, building accurate and potentially complex models to estimate runtime and/or power for the kernel are not needed. Therefore, we use support-vector machine (SVM) based classifiers to predict the optimal device in our scheduler. We also evaluated *k-means* and decision-tree based classifiers in the paper, but SVM provides the most accurate predictions as demonstrated later in the results. We *et al.* also used similar classification approach, based on static code profiling along with work group sizes to make device decisions [110] however, their approach does not include run-time hardware physical (TDP) and CPU-load (number of free/busy cores) conditions in the scheduling process. So, there could be performance loss in a dynamic system environment, as shown in the results section. Further, their classifier is based on the work group size and the static compiler-level kernel information, while we use performance counters, measured on the actual hardware, as the feature vector for the classifier; hence, it captures the kernel behavior more reliably on the target hardware.

We collect a broad set of PMCs available by running multiple OpenCL applications on our experimental system to build reliable SVM models. Table 4.1 shows the list of PMCs we studied as the feature space for the SVM-based classifier. We selected these performance counters as they not only represent the overall kernel characteristics but also enable the differentiation of kernels with respect to suitability on CPU or GPU devices. For example, kernels with higher branch instructions benefit more on CPU, while those with higher LLC misses and resource stalls performance better on GPU. The trained models are stored and used by the scheduler to make optimal decision for different kernels. In

Table 4.1: List of performance counters for the SVM classifier.

INSTR	INSTRUCTIONS_RETIRED
REFCLK	UNHALTED_REFERENCE_CYCLES
CLK	CPU_CLK_UNHALTED
BR	BRANCH_INSTRUCTIONS_RETIRED
BRMISS	BRANCH_MISSES
LLC	LLC_MISSES
L1DL	L1_DCACHE_LOAD_MISSES
L1DS	L1_DCACHE_STORE_MISSES
L2MISS	L2_RQSTS:ALL_DEMAND_MISS
STALL	RESOURCE_STALLS:ANY

particular, we use the following SVM equation to predict the appropriate device ($c < 0$ for GPU, and CPU otherwise) during run-time:

$$c = \sum_{i}^{n} \alpha_i.l_i.\kappa(s_i, x) + b, \tag{4.1}$$

where, n denotes the number of support vectors, b denotes the bias of SVM classifier, x is the test feature, and s_i and l_i denote the i^{th} support vector and its class-label, respectively. Since the feature space is usually not linearly separable, we adopt polynomial kernel function for SVM. So, both bias and support vectors are obtained by training an SVM classifier using 3^{rd} order polynomial basis function denoted as $\kappa(x, y) = (1+x.y)^3$. Although training an SVM model is computationally expensive, the testing is far less expensive than the training. Further, it is worth mentioning that these models need to be built only once, so if the underlying hardware changes, the SVM models could be retrained and the new models could be used during run-time.

Online Kernel Characterization and Mapping Algorithm. Algorithm 1 shows the proposed online methodology/algorithm to characterize and schedule kernels on appropriate device in our scheduling framework. To keep track of the chip TDP and the performance counters (PMCs) values, it is assumed that the scheduler is running continuously throughtout the application time, say as a linux kernel-task. At each iteration i of a kernel (k), the scheduler first checks for the free/available CPU cores (C_i^k) using PMC values of

Algorithm 1: Online characterization and mapping of kernels/workloads on CPU-GPU processors.

Result: $device$ (CPU/GPU)
Start_scheduler(); //Keeps track of chip TDP and PMCs;
for *Every iteration* i *of kernel* k **do**
    Find available CPU cores (C_i^k);
    if $C_i^k == 0$ **then**
        $device$ = GPU;
    else
        Read current chip TDP (P_i^k);
        $device$ = database_query(C_i^k, P_i^k, k);
        if *(device == NULL)* **then**
            $device = SVM\_model(PMC, TDP_i, C_i^k)$;
            Collect performance counters (PMC);
            database_add(C_i^k, P_i^k, k);
        end
    end
    return $device$;
end
Function $SVM\_model(PMC, TDP, C)$);
    x = (PMC, TDP, C); // feature-vector;
    $device$ = $\sum_i^n \alpha_i.l_i.\kappa(s_i, x) + b$;
    return $device$;

different cores. If all cores are occupied by some other workloads, the scheduler selects GPU as the default device, otherwise the scheduler reads the current chip TDP and checks if the same kernel has been profiled before at similar physical and run-time conditions (i.e. for the same number of CPU cores available). If the kernel is seen for the first time and is not profiled at the similar conditions before, then the SVM model is invoked to predict the device with performance counters PMCs, TDP_i and CPU load conditions (C_i^k) as its input. The current device decision is then added to the database for using in the future iterations of the kernel.

To create the device database, we discretize the entire TDP range into steps of 10 W TDP to store the optimal device-map for each kernel at different number of cores and TDP ranges. We only need to store binary values (e.g., 0 for GPU and 1 for CPU), so the database size is reasonably small and is proportionals to the number of kernels in the application. Also, by default, we delete the optimal device-map for all kernels of the

application when it is finished. However, if the same application is expected to be seen again, we have an option of keeping the device-maps for all kernels stored across multiple iterations of the application. Also, its worth mentioning that we build different SVM models for minimizing runtime or energy goals as the device decision for minimizing one is typically different than the other.

4.4 Experimental Setup

We perform our experiments on a real system equipped with an Intel Haswell Core i7-4790K CPU-GPU processor and 16 GB memory. The CPU has 4 cores with 4×256 KB L2 caches and an 8 MB L3 cache. It has an HD graphics 4600 GPU with 20 execution units, each with 16 cores, integrated on the same die. The nominal frequency of CPU cores is 4 GHz, while the GPU cores run up to 1.25 GHz. To show the effectiveness of the proposed scheduler, we have selected 13 OpenCL benchmarks from 3 popular benchmark suites to cover wide range of workload characteristics – Rodinia [14], Parboil [102], and Polybench [35]. The list of benchmark, along with the number of kernels in each benchmark are shown in Table 4.2. In total, there are 24 OpenCL benchmarks. Further, to study

Table 4.2: List of OpenCL benchmarks and their kernels.

Benchmarks	**#Kernels**
Streamcluster (SC)	1 (K1)
Particlefilter (PF)	4 (K2-K5)
LU Decomposition (LUD)	3 (K6-K8)
Distance-cutoff Coulombic potential (CUTCP)	1 (K9)
Lattice-Boltzmann method (LBM)	1 (K10)
Hotspot (HS)	1 (K11)
Heartwall (HW)	1 (K12)
Computational fluid-dynamics (CFD)	3 (K13-K15)
Sparse matrix-vector multiplication (SPMV)	1 (K16)
General Matrix Multiply (SGEMM)	1 (K17)
Stencil (STL)	1 (K18)
Finite-difference time-domain method (FDTD)	3 (K19-K21)
Gram-Schmidt Process (GSCHM)	3 (K22-K24)

the effect of co-runners, we select 4 representative workloads (`hmmer, omnetpp, gamess,` and `soplex`) from SPEC CPU2006 suite [101]. Among them `hmmer` and `gamess` are computationally intensive workloads, while `omnetpp` and `soplex` are memory-bound workloads. In our experiments, we run one or more instances of these workloads on CPU cores to vary the CPU-load.

Further, to build the SVM models for minimizing energy, we measure the energy of each kernel using Intel's running average power limit (RAPL) APIs. We implement the time-varying TDP by setting hardware model-specific registers (MSR) corresponding to package power limit. Further, in this work, we focus mainly on those applications which have at least one of its kernel being launched multiple number of times. This assumption is typically true for the scientific applications as they involve iterative algorithms, wherein certain functions are executed multiple times, with potentially different inputs in each iteration. The runtime/energy of a kernel in its very first iteration could be significantly different than its runtime/energy in the latter iterations due to unknown hardware state and cache warm up during the first iteration. Therefore, we use first two iterations to run on CPU to collect the performance counters and use them in the SVM model to predict optimal device. Finally, for robust training of SVM models, we ran multiple iterations of each kernel to obtain reliable energy and runtime measurements.

4.5 Results

In this section, we provide following set of results from our experiments: a) evaluation of the accuracy of the proposed SVM-based kernel-workload scheduler, b) the impact of TDP and CPU-load conditions on the scheduling decisions for minimizing runtime and energy, c) comparison of our proposed scheduler against two state-of-the-art scheduling

methods, d) demonstration of the effectiveness of our proposed scheduler during run-time on a real system, e) the runtime overheads of the proposed scheduler.

a. Kernel Classification Accuracy

We evaluated our SVM model at different physical and run-time resource availability conditions for different kernels. In particular, we varied the TDP between 10 W to 80 W (in steps of 10 W), number of CPU cores available to OpenCL workload from 4 to 1 (in steps of 1). We evaluated the SVM model under following 4 scenarios based on whether TDP or the number of available cores or both are allowed to change on the system: 1) fixed TDP, fixed number of available cores, 2) fixed TDP, variable number of cores, 3) variable TDP, fixed number of cores 4) both TDP and number of cores variable. We found that our SVM model performs well in all 4 scenarios. Specifically, the maximum inaccuracy for different kernels at different possible conditions in the aforementioned four scenarios are 2.08%, 3.12%, 2.43%, and 2.31%, respectively.

b. Impact of TDP and CPU-Load on the Scheduling Decisions

Here, we discuss the interplay of TDP and CPU-load conditions on scheduling decisions for minimizing runtime and energy of different kernels.

Scheduling for Minimizing Runtime. Figure 4.5 (a) shows device map (blue for GPU and red for CPU) for all 24 kernels that minimizes the kernels' runtime under different TDP and CPU-load conditions. In particular, we show the results for 3 TDPs (20 W, 40 W, and 80 W) and 4 different CPU-load conditions (4CL to 1CL), achieved by pinning the OpenCL workload on different number of CPU cores (4 cores to 1 core). For the results shown in this figure, we do not run any other workload on the remaining CPU-cores; so, the device map here provides insights into the relative performance scaling of different kernels on CPU-cores versus GPU at different TDPs. From the device map in

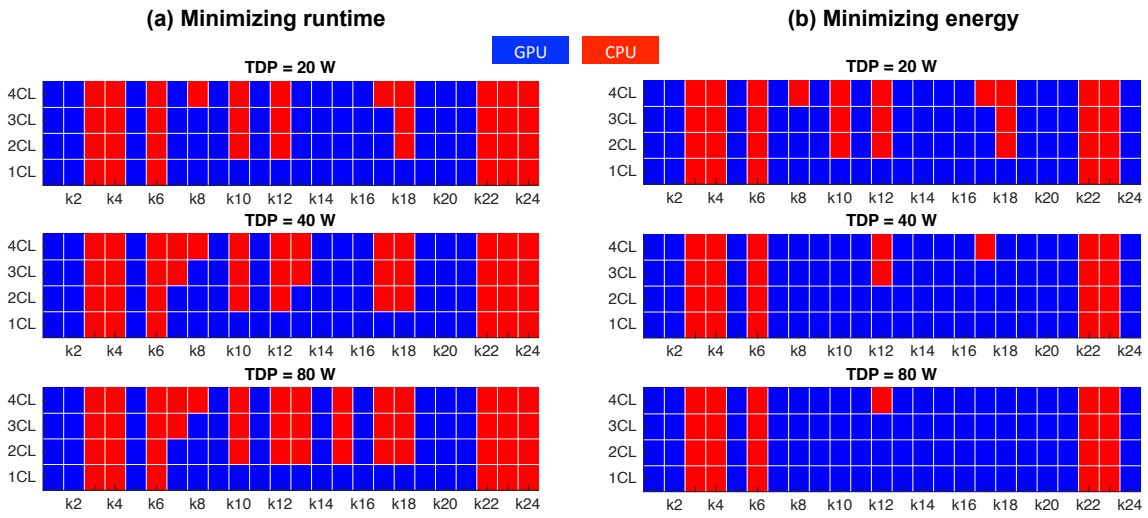

Figure 4.5: Device map for minimizing (a) runtime, (b) energy when executed with different number of cores (without co-runners).

Figure 4.5 (a), we observe that the scheduling decision depends on the CPU-load conditions, requiring the scheduler to be aware of run-time workload conditions while making the scheduling decisions. Specifically, at a fixed TDP, we observe 3 types of trends in scheduling decisions as the number of CPU cores available to the OpenCL workload are varied. They are as follows.

1. Always faster on GPU.
2. Always faster on CPU.
3. CPU faster at higher number of CPU cores and GPU faster at lower number of cores.

Kernels in the first category have lower performance on CPU (even at all 4 cores) than GPU. Ten out of 24 kernels (K1, K2, K5, K9, K11, K14, K16, K19, K20, K21) fall in to this category. These kernels have enough parallelism and low branch divergences, so they run faster on GPU. Similarly, six out of 24 kernels (K3, K4, K6, K22, K23, and K24) run faster on CPU irrespective of number of CPU-cores allocated to them. The GPU compute throughput is significantly lower than CPU for these kernel due to lack of parallelism or

large number of synchronization points or the kernel is too short in duration that sending it to GPU only adds unnecessary overhead in runtime. Among them, kernels K3, K6, and K23 are relatively short kernels (few micro seconds), while others lack parallelism and benefit more from higher CPU frequency. The scheduling for these two categories is relatively simpler because the scheduler does not need to consider the effect of CPU load-conditions while making decisions. So, the existing schedulers could also perform well for such kernels [110].

However, scheduling for the kernels falling into third category (8 out of 24 kernels) is challenging. When given all 4 cores, the performance of these kernels on CPU is either comparable or only up to 4$\times$ better (on our 4-core system) than on GPU. Therefore, at a fixed TDP, as the number of available cores reduces (e.g., from 4 to 1), the CPU performance could become worse than GPU. For making appropriate scheduling for such kernels, the scheduler needs to first find the number of available cores at the time of scheduling and then make the decision accordingly. The SVM model used in the proposed scheduler includes the number of available CPU cores as one of the features, and hence, makes better decision under varying CPU load-conditions than the state-of-the-art schedulers, which are typically oblivious of such run-time conditions [110].

Furthermore, for the kernels in third category the change of performance of kernels on CPU with respect to TDP could also affect the device decisions. Specifically, for the fixed CPU-load (e.g., 3CL case, where 1 core is busy with other workload), the performance of a kernel at low TDP (e.g., 20 W) could be lesser on CPU than on GPU, however, as the TDP is increased (e.g., from 20 W to 80 W), the performance of CPU improves significantly and therefore, CPU could provide better performance than GPU. Hence, we conclude that for effective workload scheduling on a real system with varying physical and run-time conditions, the scheduler should consider system conditions in to account while making the device decisions. The proposed scheduler makes appropriate device

decisions by profiling each kernel at different TDPs and run-time CPU load conditions before making decisions.

Scheduling for Minimizing Energy. In Figure 4.5 (b), we show the device map (blue for GPU and red for CPU) for all 24 kernels that minimizes the kernels' energy under 3 different TDPs (20, 40, 80W) and 4 different CPU-load conditions (4CL to 1CL). Similar to the case for minimizing execution time, as discussed above, the scheduling device that minimizes energy also depends on both TDP and number of CPU cores available.

Specifically, we make the following scheduling-related insights from the device map shown in Figure 4.5 (b). First, we observe that for 5 out of 24 kernels (K8, K10, K12, K17, and K18) the device that minimizes energy is a function of both chip TDP and the number of available CPU cores. In particular, we notice that for these kernels, at lower TDP (20 W), CPU consumes lower energy because both CPU and GPU dissipate full 20 W TDP, but CPU provides better performance than GPU, hence lower energy. However, as the TDP is increased to 80 W, CPU becomes less energy efficient for these kernels. This is because as the TDP is increased the GPU power saturates on our system as it reaches its maximum frequency (1.25 GHz), however the increase in TDP allows CPU to run at higher frequency (up to 4.4 GHz), leading to significantly higher dynamic power ($\propto V^2 f$) without providing proportional increase in performance. So, it is important for the scheduler to use TDP information in the model while making device decisions for minimizing energy. The existing scheduler do not use such information in their model and potentially lead to less energy-efficient scheduling decisions.

Second, for the above-mentioned 5 kernels, the number of available CPU-cores also affects the scheduling decisions. In particular, we observe that as the number of cores reduce from 4 to 1, the performance of kernels on CPU reduces more than the decrease in CPU power (due to leakage power in other cores); so, for these kernels CPU becomes

less energy-efficient than GPU at small number of cores. Furthermore, we observe that although for most kernels GPU tends to be more energy-efficient device than CPU at higher TDP, some kernels (e.g. K3, K4, K6, K23, and K24) have lower energy on CPU even at higher TDP. This happens when kernels are either too short in duration or have significantly higher performance (more than $4\times$) on CPU that GPU at 80W, so they consume lesser energy on CPU than on GPU at all TDPs. The scheduling for such kernels is less challenging and therefore, even the existing schedulers, which do not consider all run-time conditions, could also make correct scheduling decisions for those kernels.

c. Comparison Against the State-of-the-Art

Figure 4.6 shows the comparison of performance for 3 different scheduling methods under different TDP and workload conditions for selected kernels. It also shows the average performance gains/loss for all 24 kernels in the last set of bars. For comparison against the state-of-the-art scheduling methods, we choose to show results on 4 representative physical and run-time conditions in Figure 4.6(a)-(d): 1) OpenCL workload on all 4 cores at 80 W, 2) OpenCL workload on 1 core and SPEC workload on 3 cores at 80 W, 3) OpenCL on all 4 cores at 20 W, and 4) OpenCL on 1 core and SPEC on 3 cores at 20 W. Furthermore, we compare our proposed method against the following two state-of-the-art scheduling methods.

1. `Application-level-user-based (App-level)`: It chooses the device that minimizes the total runtime or energy at application-level, instead of kernel-level [36, 16]. This case could also represent the user/developer's static method of selecting the optimal device. Moreover, this method is both TDP- and CPU load-oblivious because the user typically profiles the program at only one (default or maximum) TDP when no other workload is running on the system.

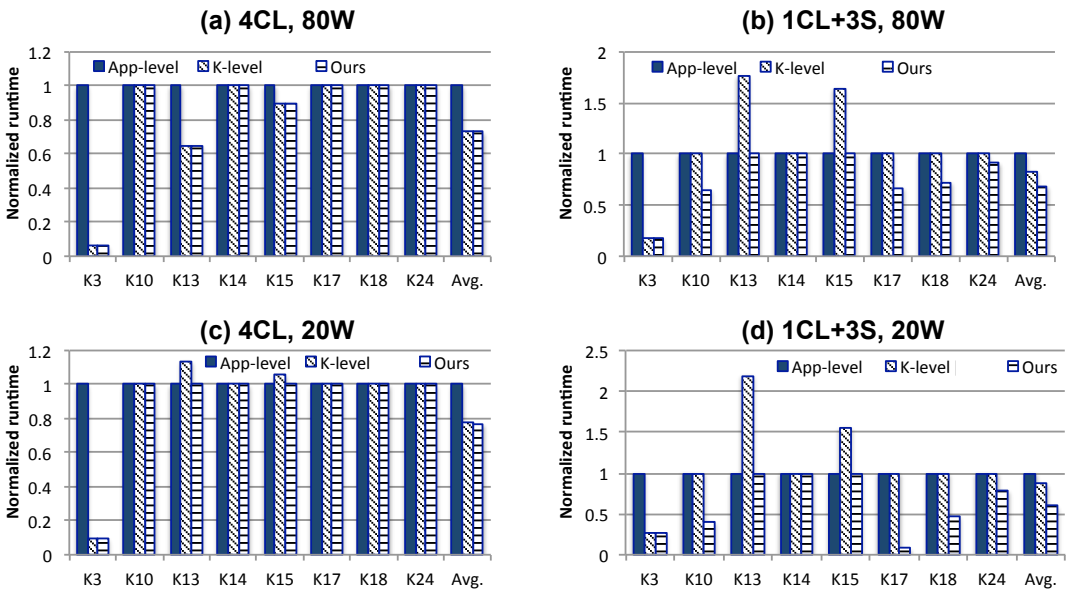

Figure 4.6: Comparison of runtime for `Ours` method against state-of-the-art schedulers (`App-level` [36, 16] and `K-level` [110]) at two TDP and two CPU-load conditions: a) OpenCL on 4 cores at 80W TDP, b) OpenCL on 1 core and SPEC on 3 cores at 80W TDP, c) OpenCL on 4 cores at 20W TDP, d) OpenCL on 1 core and SPEC on 3 cores at 20W TDP. The normalization is done with respect to the `App-level` case.

2. `Kernel-level-static-conditions-based (K-level)`: It chooses the device that minimizes the total runtime at kernel-level assuming fixed TDP and CPU-load conditions [110]. So, this method performs better than static user method, but suffers performance/energy loss when system conditions vary over time.

While both `App-level` and `K-level` methods are assumed to be profiled at fixed 80 W TDP under no-workload conditions, our proposed method, also called `Ours` in Figure 4.6 considers both run-time physical and CPU load-conditions, leading to better overall performance and energy efficiency.

All the results shown in Figure 4.6 are normalized to the `App-level` case, which typically performs worse than both `K-level` and `Ours` case under all 4 representative conditions shown in the figure. This is because different kernels of an application could have their best performance on different devices, so choosing the same device for all ker-

nels does not provide the best performance. From Figure 4.6 (a)-(d), we observe that `Ours` method provides better or equal performance than the other two scheduling methods for all benchmarks and under all four system conditions. However, when some of the CPU-cores are used by other workloads (SPEC benchmarks in our case), our proposed method (`Ours`) provides better performance than both `App-level` and the state-of-the-art `K-level` method.

Under no-workload conditions (i.e., 4CL cases), both `K-level` and `Ours` methods provide similar, but better performance than the `App-level` method. This is because both `K-level` and `Ours` methods make decisions at kernel-level and for some of the selected kernels (e.g., K3 and K13), the scheduling device has huge impact on their performance. For example, K3 kernel is too short in duration that sending them to GPU causes un-necessary runtime overhead. The `App-level` scheduling method wrongly chooses GPU device for these two kernels because they, along with other three kernels K2, K4, and K5, belong to the same application (`particlefilter`); kernels K2 and K5 are dominant in runtime and their performance is better on GPU, so the (`App-level`) method chooses GPU for all four kernels of this application. Further, we notice that when all four cores are available (i.e., 4CL cases), changing the TDP from 80 W to 20 W does not affect the scheduling decision for performance for the selected kernels (although, it does impact the decision for energy); so, both `K-level` and `Ours` methods choose the best device for all kernels.

The performance gains from `Ours` method increase as the TDP changes from 80 W to 20 W and the number of available cores change from 4CL to 1CL, as shown in Figure 4.6 (c)-(d). This is because `Ours` method finds out the current CPU-load on the system before making a scheduling decision, while the other two methods do not take such system information into account. In particular, for the 1CL+3S case, wherein 3 out of 4 cores are running SPEC benchmarks and only 1 CPU-cores is available for the OpenL workload,

`Ours` method provides 40% and 31% better average performance than `App-level` and `K-level` methods, respectively.

Further, as seen in Figure 4.6 (c), its possible that the `K-level` method could lead to worse scheduling than both `App-level` and `K-level` methods because the kernel-level decision made under fixed TDP (80 W) under no-load conditions (4CL) could be different than the decision at `App-level` under the same conditions. Kernels K13 and K15 are two such examples; they run faster on CPU for 4CL case. However, these two kernels (along with K14) belong to the same application `CFD` and, for 4CL case, the entire application runs faster on GPU due to K14 being the dominant kernel. Further, when 3 of 4 cores are used by SPEC workloads and only 1 core is available for the OpenCL workload, all 3 kernels perform better on GPU. So, for these 3 kernels, both `Ours` and `App-level` methods make correct scheduling decision at 1CL+3S condition, but `K-level` method, unaware of the system-conditions makes wrongs device decisions (CPU) for K13 and K15 kernels. It is worth mentioning that the correct device decisions by `App-level` method for these two kernels is purely incidental. On the other hand, the correct-decisions from `Ours` method are not incidental because `Ours` method takes varying system conditions in to account while making device decisions.

d. Demonstration of the Scheduler's Behavior on a Real System

Here, we demonstrate the effectiveness of our scheduler in a time-varying TDP environment under fixed number of cores. To this end, we applied arbitrary time-varying TDP traces to the hardware by writing to the package power limit MSR. Figure 4.7 shows the response of the proposed scheduling method for minimizing the energy for the LUD application with 3 kernels (k1-k3). Similar scheduling behavior is observed for runtime.

From the 3 subplots in Figure 4.7 (a)-(c), we notice that the lower energy device for

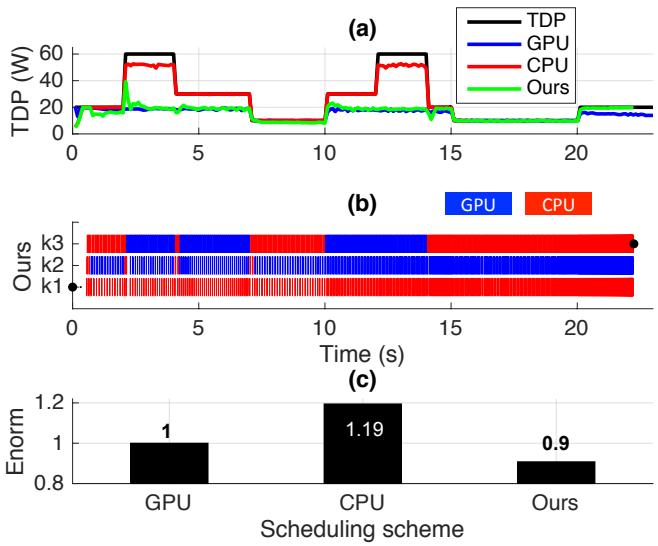

Figure 4.7: Demonstration of TDP-aware kernel-level dynamic scheduling for LUD application with 3 kernels; (a) time-varying TDP and the actual power dissipated under 3 different scheduling schemes: GPU, CPU, and `Ours`; (b) the execution of one or more kernels on different devices over time; (c) shows the normalized energy for 3 different scheduling schemes.

different kernels is different under different TDP values and our scheduler is able to launch each kernel on the appropriate device. As we can see from the Figure, for kernel $k1$ the scheduler selects CPU as the optimal device under all TDPs. This is because each iteration of $k1$ is a relatively short ($< 0.1\ ms$) and therefore, the overhead of launching it on GPU leads to higher energy than if it is run on CPU itself. On the other hand, kernel $k2$ is always scheduled on GPU to minimize the energy. Among all kennels, the lower-energy device for $k3$ kernel is TDP-dependent. Specifically, for $k3$ kernel of the LUD application, CPU is the lower-energy device at lower TDP while GPU minimizes the energy at higher TDP. Our scheduler accurately predicts the energy-optimal devices for each kernel under a time-varying TDP.

We compare the total energy dissipation of LUD application in 3 scheduling cases: GPU, CPU, and `Ours` method. Here, GPU-case is the same as the `App-level` method decision because the developer typically performs the characterization at fixed high TDP

value and for both these benchmarks, GPU provides lower energy at high TDP value. Obviously, the energy savings from `Ours` method will depend on the time-varying TDP-pattern. The maximum benefit will be seen when the TDP changes from low to high or high to low only once, say immediately after the kernels start. In that case, our scheduler will make correct device decision after couple of profiling iterations, however, all other static or TDP-oblivious schemes would keep the device decision fixed to potentially wrong device and hence, they will incur energy losses. Nevertheless, for the selected TDP trace, `Ours` method provides 10% and 24% lower energy than the `App-level` CPU device cases, respectively. Finally, it is worth mentioning that the scheduler keeps the history of optimal device for each kernel under different TDPs to avoid invoking the SVM predictor for the previously seen TDP value. This can be observed for the $k2$ of LUD kernel, wherein after $t > 10\ s$, when the TDP values are repeated, the kernel keeps running on GPU without invoking the SVM predictor. Reuse of the past predictions amortizes the overall overhead of the predictor.

e. Overheads

While the proposed SVM-based classifier provides performance improvements or energy savings through efficient scheduling during run-time, it is important to understand its overhead on the overall runtime of the application. We evaluated the runtime overhead of adding custom APIs and SVM on all applications. The maximum runtime overhead is about 1.9%. Given high potential of performance improvement and energy savings, we believe that this overhead is reasonably acceptable.

4.6 Summary

In this chapter, we presented a scheduling framework that takes in to account the system dynamic conditions, along with the workload characteristics to minimize runtime or energy on CPU-GPU processors. In contrast to previous approaches that either mapped entire applications or did not consider run-time conditions, our fine-grained approach enables scheduling at the kernel-level while considering system conditions during scheduling decisions to fluidly map the kernels between CPU and GPU devices. In a way, our approach complements the built-in hardware capabilities to limit TDP by incorporating the ability to schedule as well. To identify the best mapping for a kernel, we developed a SVM-based classifier that monitors the measurements of the performance counters to profile both the current workload and detects the number of available cores online, and accordingly decides the best device for the kernel to minimize total runtime or energy. We trained the classifier using off-line analysis that determined the best performance counters to use. We fully implemented the scheduler and tested it on a real integrated CPU-GPU system. Our results confirm its superiority as it is able to outperform application-based scheduling and the state-of-the-art scheduling methods by 40% and 31%, respectively. Similarly, our scheduling framework provides up to 10% more energy saving for the selected time-varying TDP pattern than the user-based application-level scheduling scheme.

Chapter 5

Workload-Aware Low Power Design of Future GPUs

5.1 Introduction

Efficient performance and power management are critical for effective operation of modern processors in high performance computing (HPC) systems. HPC scientific applications have strict performance requirements under tight power budgets. Graphics Processing Units (GPUs) are now commonly used in many HPC systems due to their high performance and power efficiency. As of November 2012, four of the top ten and 62 of the top 500 supercomputers on the Top500 list were powered by accelerators [77, 81]. Future petascale and exascale systems are likely to incorporate GPUs with hundreds of compute units (CUs) [73]. Emerging trends show that these CUs have to operate under tight power budgets for safe operating temperatures and avoid excessive leakage power or thermal runaway. As a result, not all CUs can always be powered on across all applications due to

thermal and power constraints [32]. Thus, it is necessary to dynamically adjust the number of active CUs through power-gating (PG) mechanisms based on the run-time needs of applications. However, PG introduces serious design and area overheads, which if applied liberally can negate its benefits. There is a tradeoff between design overheads and run-time performance and power efficiency.

In this chapter, we develop an integrated approach towards addressing power gating challenges in future GPUs by analyzing 1) design-time decisions, where the benefits of fine-grain power gating must be balanced against its overheads, and 2) run-time decisions, where power gating and frequency boosting need to be applied adaptively to control the number of active CUs based on the GPU design, the application needs, and the total power budget. Specifically, the contributions of this chapter are as follows.

- We demonstrate the need for an integrated solution to manage leakage power by incorporating *workload/run-time-awareness* into the PG design methodology that determines the optimal PG granularity, and *design-awareness* into the run-time power management algorithm that finds the optimal number of CU to power gate.
- Using realistic industrial scaling models and actual hardware measurements on an existing GPU, we project run-time parallelism trends of HPC applications to future massively parallel GPUs. We use these trends together with the accurate PG area models to determine the appropriate design choices for PG granularity (i.e., PG cluster size) that improve power efficiency without sacrificing performance and incurring unnecessary design overheads.
- We propose a run-time power management algorithm that utilizes PG design knowledge to shift power from unused CUs towards boosting the frequency of active CUs, thereby leading to better performance and power efficiency of the system working under a strict power cap.

- Compared to per-CU PG, we demonstrate that a workload-aware design of a 16 CU per cluster achieves 99% peak runtime performance without the excessive 53% design area overhead. In addition, we demonstrate that a run-time power management algorithm that is aware of the PG design granularity leads to up to 18% higher performance under thermal-design power (TDP) constraints. Although these results are based upon AMD's Graphics Core Next (GCN) architecture and the particular set of chosen representative applications, the overall methodology can be applied to other GPU architectures and applications as well.

The rest of the chapter is organized as follows. Section 5.2 discusses the motivation and goals of our work. Sections 5.3 discusses the proposed models and methodology at both design-time and run-time. The scaling methodology validation, the run-time algorithm, and the performance and power efficiency results at different PG granularities, along with the key findings are presented in Section 5.4. Finally, we summarize the main conclusions of the chapter in Section 5.5.

5.2 Motivation & Goals

In order to achieve extremely high parallel performance that is required to meet future (e.g. exascale or petascale) computing needs [73], future HPC systems are predicted to operate massively parallel GPUs under tight power and thermal constraints, which poses unique power and performance challenges as discussed in the next paragraphs.

a. Leakage Power in Future Massively Parallel GPUs.

Future GPUs would very likely require a large number of CUs to be packed within a single processor chip. Further, technologies such as 3D integration may also be required

to address the challenges of on-chip communication delay and packaging density [28]. All these result in high power consumption, larger power densities, and hence, higher temperatures across the chip. Although FinFET technology significantly reduces leakage power due to higher threshold voltage in the off-state [57], the FinFET devices suffer from self-heating problems and are prone to thermal runaway due to confinement of the channel, surrounded by silicon dioxide, which happens to have lower thermal conductivity compared to bulk silicon [17]. Further, the International Technology Roadmap for Semiconductors (ITRS) predicted that the sub threshold leakage ceiling for FinFET will be comparable to planar bulk MOSFETs [106, 17]. Hence, in future massively parallel GPUs, leakage power can still be a significant contributor, specially if all CUs are left powered on at high operating temperatures. Therefore, if leakage is not properly dealt with, the tight power budget will throttle both operating frequency and number of active CUs, leading to unacceptable performance.

b. Workloads Demonstrate Diverse Scaling Trends.

For this study, we analyze the US Department of Energy "proxy" and other scientific computing applications for exascale [78, 14] and find out that only a small subset of such applications are embarrassingly parallel. In fact, there is a wide range of diverse characteristics in their usage of hardware resources, in particular, the number of CUs [50]. There is a large degree of load imbalance in these applications due to branch divergence and memory divergence, and therefore opportunities to save power by power gating unused CUs. Figure 5.1 shows the performance of three example HPC application kernels as a function of the number of active CUs (Section 5.3 describes the used methodology). The performance of each kernel is normalized to its own minimum baseline. For example, the performance of `GEMM.sgemmNN` kernel scales almost perfectly with the number of CUs, while the performance of `LULESH.CalcHourglass` kernel saturates after reaching

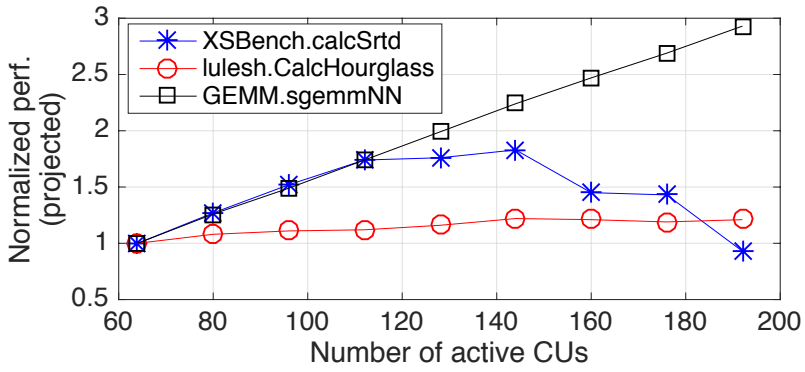

Figure 5.1: Performance scaling of 3 example kernels on a future GPU with 192 CUs.

140 CUs due to saturated memory bandwidth. On the other hand, `XSBench.calcSrtd` kernel has a peak performance at 124 CUs, beyond which significant cache thrashing occurs and performance degrades. Thus, we observe that HPC applications display various parallelism trends due to their diverse compute and memory behavior, requiring a runtime power management system to adaptively control the number of CUs by power gating unused CUs under a fixed TDP.

c. PG Ggranularity is Critical to Performance and Power Cconsumption.

If an HPC application can not leverage all CU units, then these units can be power gated and the power savings from gating can be used to boost frequencies of the remaining active CUs for higher performance under a given power budget. The amount of power savings depends on the *PG granularity*–defined as the minimum number of CUs that can be power gated at once, usually a design-time decision. A finer PG granularity design could provide more power savings, and therefore, higher frequency-boosting than a coarser PG granularity design. However, implementing finer PG granularity would require larger power gating transistors (to support higher frequency boost) and more number of buffers, clamp/isolation cells, etc., resulting in large design area overheads. On the other hand, an overly coarse PG granularity has the advantage of reducing the number of control signals and routing resources, but it can result in excessive leakage power and runtime per-

formance degradation under a fixed TDP, especially for applications that use fewer number of CUs than the exact multiples of the PG granularity. Thus, there is a design-time trade-off to be made, and it is important to make the decision in an application-aware manner. Existing run-time power management systems often have no knowledge of the underlying PG design leading to sub-optimal decisions.

d. Run-time Power Management in Future Massively Parallel GPUs.

When and how often a CU can be power gated are determined by the run-time power management controller based on workload characteristics under a given power envelope. However, the run-time power management controller often has no knowledge of underlying power-gating design and overheads. In fact, to the best of our knowledge, the current state-of-the-art GPU power management algorithms in AMD and NVidia GPUs only consider frequency scaling [74, 80, 103]. They do not perform techniques such as increasing the number of active CUs or boosting the frequency through power gating mechanisms to optimize performance within power constraints. Further, unlike our study, most of the previous studies investigated PG opportunities by assuming the finest level of power gating at per-CU/core level without incorporating the design-time decisions in to run-time power management algorithms [1, 58, 56, 83]. In future massively parallel GPUs, the run-time power management unit should determine the resource needs (#CUs) of an application quickly enough to maximize the performance improvements and energy savings from power gating unused CUs. We argue that scaling the number of active CUs and/or boosting the frequency is needed for future GPU architectures, when operating under a tight power budget, and run-time management needs to be aware of design decisions while turning on and/or power gating CUs to meet application's parallelism demands.

In summary, it is important to address power management solution from design-time to run-time in order to meet the high performance requirements for future GPUs, oper-

ating under tight power budgets. PG methodologies that do not consider workload characteristics during design and design choices during workload execution can lead to poor performance and power efficiency with large design and area overheads. So, our **goal** is to couple both design-time PG granularity sizing and run-time opportunities to maximize performance and power efficiency for future GPUs. In the next section, we propose synergistic PG methodologies at design and run-time to evaluate and implement efficient power management in future GPU systems.

5.3 Proposed Methodology

As has been the recent trend, we assume GPU architecture scaling occurs primarily through increasing the number of parallel compute units (CU) and memory bandwidth. Figure 5.2 shows a future GPU microarchitecture that is similar to a current 28 nm GPU. The specifications of the existing hardware and the future hardware studied in the paper are shown in Table 5.1. We evaluate the performance and power efficiency of a future massively parallel GPGPU architecture with 192 GPU CUs and 2048 GB/s memory bandwidth (BW) at 10 nm technology. The number of CUs were selected to provide reasonable approximations of mainstream GPUs targeted for petascale and exascale systems [73]. However our methodology can be easily generalized to other future architectures with different peak compute throughput and memory bandwidth requirements. We also assume the internal CU architecture (including cache hierarchy) does not change significantly over the period studied. Naturally, this is an unrealistic assumption from a micro-architectural perspective as CU architectures will continue to be refined and improved and cache hierarchy will evolve. However design reuse of the same microarchitecture (with incremental improvements) across multiple generations is a common industry best practice, mainly gaining performance in particular areas due to engineering advances and learnings, but not chang-

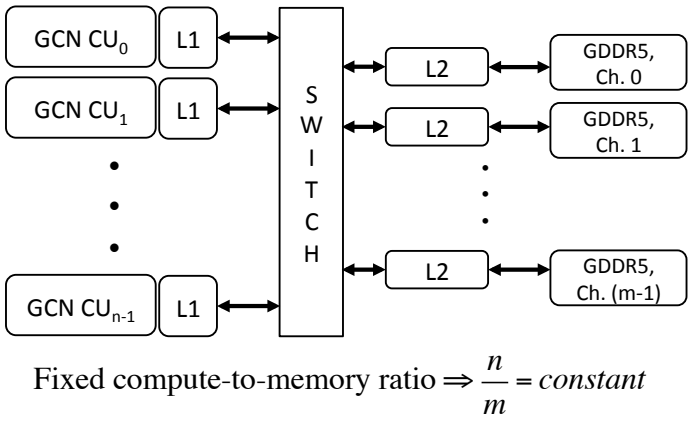

Figure 5.2: Template GPU architecture. The compute throughput and memory bandwidth are proportional to n and m, respectively.

ing greatly at the most fundamental level. Our focus here is to understand the high-order performance, power and energy effects of PG granularity in future architectures.

The overall methodology for power gating of future GPUs is as follows. First, we propose in Subsection 5.3.1 a methodology to scale power and performance measurements on existing GPU devices to future devices with similar micro-architecture. Some practical considerations during the modeling and projection process are discussed in Subsection 5.3.2. Using these projected measurements, we propose in Subsection 5.3.3 a methodology to analyze the impact of different PG granularity choices with respect to the available parallelism in applications and the opportunities for run-time frequency boosting. Next, we propose a run-time power management technique in Subsection 5.3.4 that utilizes the characteristics of workloads as well as PG granularity to maximize performance and power efficiency under a fixed TDP.

Table 5.1: Baseline (existing) and future GPU systems.

Parameter name	**Baseline H/W**	**Future H/W**
# CUs (n)	32	192
Nominal compute frequency (f_0)	1 GHz	1 GHz
Memory bandwidth ($\propto m$)	264 GB/s	2048 GB/s
Technology node	28 nm	10 nm
Nominal voltage	1V	0.7V
TDP	250 W	125, 150, 175 W

5.3.1 Performance and Power Scaling

Here we describe a projection framework for future GPUs. We use a typical 28 nm GPU architecture as the baseline for hardware measurements and projections using our three-step methodology: a) hardware measurements from existing hardware; b) power and performance scaling to future architectures at the same technology node; and c) applying technology scaling including FinFET effects using industrial process models. The overall projection methodology for power estimation is shown in Figure 5.3.

a. Native Hardware Execution

Our baseline hardware consists of an AMD Radeon HD 7970 system [76], which features the AMD Graphics Core Next (GCN) architecture and is paired with 3 GB of GDDR5 memory organized as a set of 12 memory channels, as shown in Figure 5.2. The GPU contains 32 CUs, each has one scalar unit and four 16-wide SIMD vector units, for a total of 2048 ALUs. Each CU contains a single instruction cache, a scalar data cache, a 16 KB L1 data cache and a 64 KB local data share (LDS) or software managed scratchpad. All CUs share a single 768 KB L2 cache. In its highest performing computing configuration, HD 7970 offers a peak throughout of 1 Teraflop double precision floating point FMAC operations and 264 GB/s peak memory bandwidth.

We first measure power and performance of different OpenCL application kernels across a wide range of configurations on the 28 nm HD 7970 system. In particular, we take measurements of performance and power (both dynamic and leakage) across 25 kernels in 13 applications from the exascale computing proxy applications [78] and the Rodinia benchmark suite [14] over 448 hardware configurations. Here, the number of active CUs is adjusted from 4 to 32 over a range of 8×, and the CU frequency is varied from 300 MHz to 1 GHz over a range of 3.3×, in steps of 100 MHz. While scaling CU frequency, volt-

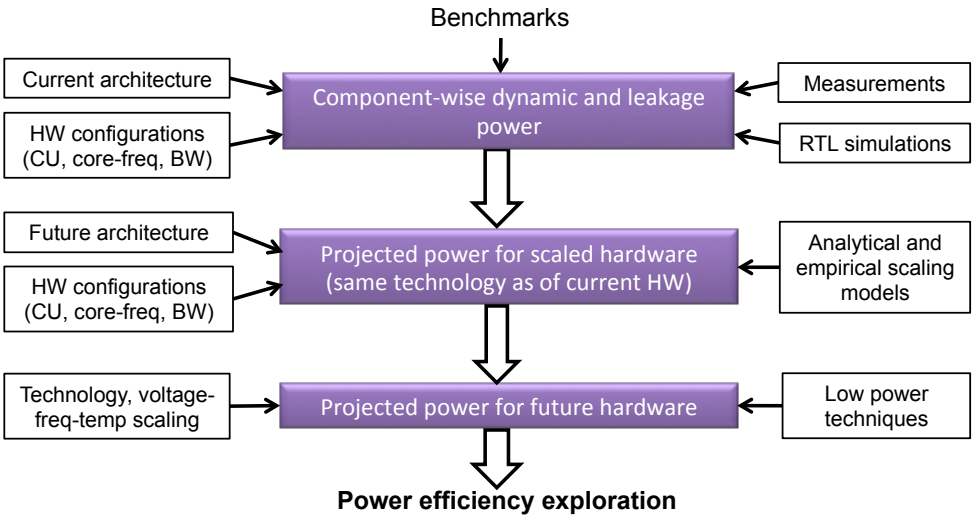

Figure 5.3: The proposed 3-step power projection methodology.

age is also scaled according to the 28 nm GPU voltage-frequency table. Memory BW is varied from 90 GB/s (at 475 MHz bus frequency) to 264 GB/s (at 1375 MHz bus frequency) across a 2.9× range, in steps of 30 GB/s (150 MHz) by changing the memory bus frequency. The power measurements out of the baseline are split into dynamic and leakage power using on-chip real-time hardware power proxies. Power measurements include those that are dissipated in cores' logic, SRAM-based caches, memory controller and interconnect. Using AMD's CodeXL library, we also gather hardware performance events for each application kernel at each configuration that capture bus activity, data flow volume and compute-memory behaviors.

b. Modeling Effects of Scaling the Hardware

The power values gathered from existing GPU must be scaled to estimate the power that the workloads would require on future systems. The two key components of this are scaling to account for change in the numbers of CUs and memory bandwidth, and scaling to future technologies. In this step, we develop analytical scaling models, and hardware measurements-driven and RTL-driven empirical models to scale dynamic, leakage and

interconnect power from the baseline to future GPUs *at the same technology node*. The technology scaling effects are applied in the last step.

Hardware compute-to-memory (CtoM) ratio driven dynamic power projection: We project power and performance from measurements on the current GPU on a per-kernel basis to the future GPU architecture (with different CU count and memory bandwidth) at the same technology node using the same compute-to-memory ratio for both devices. We consider a compute throughput that is proportional to the product of number of CUs and CU frequency, and a memory bandwidth (BW) that is proportional to memory frequency under a fixed number of memory channels. Similar to the Roofline model [111], we use the idea that for designs with the same micro-architecture, performance scales proportional to the scaling of the compute throughput and bandwidth of a GPU, given the same compute-to-memory ratio. Figure 5.4 shows an example of a performance scaling surface for `waxpby` kernel of `miniFE` application with respect to the compute throughput (GFLOPs) and memory BW (GB/s) at a fixed CU frequency. The performance of this kernel on a future GPGPU device with 192 CUs, 1 GHz CU frequency and 2048 GB/s bandwidth is projected using measurements with 12, 16, 20, and 24 CUs on the current GPGPU device. Note that the memory frequency is varied with CUs to keep the compute-to-memory ratio constant. Similar scaling method is used for dynamic power.

We project power and performance for a potential future GPU at all possible hardware design configurations from 64 CUs to 192 CUs in steps of 16 CU, CU frequency from 400 MHz to 1000 GHz in steps of 100 MHz, and main memory bandwidth from 1.6 TB/s to 4 TB/s in steps of 400 GB/s, resulting in a total of 441 distinct hardware configurations. Further, the proportion of SRAM-based cache power on the baseline hardware is computed by an industrial RTL-level tool through the Synopsys PrimeTime PX (PTPX) [105]. The SRAM dynamic power is then scaled as a square-root function of its size (s), as shown in equation 5.1. This is because the SRAM dynamic power depends on the wordline and

bitline lengths, not the cache size. Therefore, we have

$$Dyn_{SRAM} = \sqrt{s} \cdot Dyn_{SRAM\_meas}. \tag{5.1}$$

Area-based leakage power projection: We model the leakage cost for the increased size of the circuits (e.g., cache size) in future GPUs by scaling the leakage power of different on-chip components, specifically CUs, SRAM caches, and MCs, separately, based on the relative increase in their area between the current and future GPUs. Using PTPX tool and floor-plan area assessment of the different components in existing GPUs, we empirically derive the leakage power partition ratio in existing hardware between CUs, caches, MCs, and miscellaneous logic for a power virus application running at worst-case die temperature. We find that a typical leakage power partition ratios in HD 7970 are 50%, 3%, and 47%, respectively. Note that, at a fixed temperature, the partition ratios need to be derived only once using PTPX. We use these partition ratios to distribute the measured leakage power in current hardware among different components and further scale to future GPU based on their area change. For example, if the SRAM capacity in future GPU is

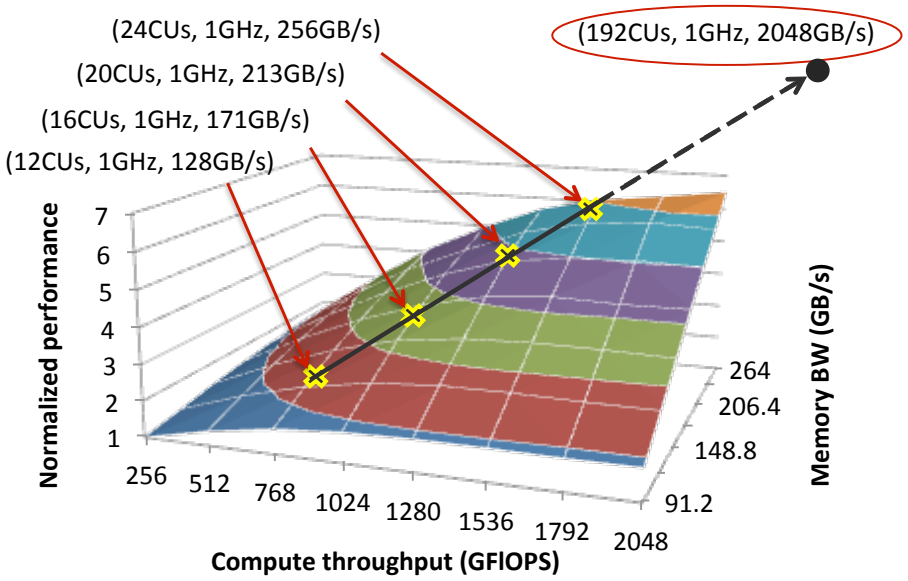

Figure 5.4: Performance scaling surface for `miniFE.waxpby`.

x times more than that in the current GPU, its cache leakage power is also x times more. That is

$$\begin{aligned} Lkg_{logic} &= a \cdot Lkg_{meas\_total} \\ Lkg_{SRAM} &= b \cdot Lkg_{meas\_total} \\ Lkg_{MC\_misc} &= c \cdot Lkg_{meas\_total}, \end{aligned} \tag{5.2}$$

where, a, b, and c ratios are obtained from PTPX simulation of the existing GPU, and $a + b + c = 1.0$. Furthermore, the leakage power in the future GPU is given by:

$$Lkg_{future} = x \cdot Lkg_{logic} + y \cdot Lkg_{SRAM} + z \cdot Lkg_{MC\_misc}, \tag{5.3}$$

where, x, y, and z are all greater than 1; they are the area ratios between future GPU and the current GPU for CU logic, SRAM cache, and memory controller, respectively. Note that the area ratios do not include area shrinkage due to technology scaling nor change, which will be considered in third (c) step, so the area ratios mainly represent an increase in count and capacity of the component (e.g., $x = 6$ for projecting from 32 CU to 192 CU). The leakage power thus derived also assumes fixed voltage (V), frequency (F) and temperature (T) between the existing and future GPUs. The effect of V-F-T changes are accounted for in the final step.

Interconnect power modeling and projection: Power dissipation in the local and global interconnects of the GPU contributes to a significant fraction of the total chip power [64]. While the local interconnects are short wires and are used to transfer data within the IP blocks, global interconnects are relatively long wires and are used for inter tile/IP data transfers or that between the processor and memory. In our framework, the local interconnect power is accounted for in the on-chip hardware power proxies. To consider global buses between CUs and L2 cache, L2 and MCs/DDR PHY transceivers, we use PTPX

power simulation of a power virus application as the reference. We then compute actual bus activity factors of different application kernels from the performance counters measuring data flow volume, peak BW, and access rates. The interconnect power of a kernel is then computed by applying a de-rating factor (DF) proportional to the ratio of bus activities for the kernel and reference application.

$$DF = activity_{kernel}/activity_{power\_virus}, \tag{5.4}$$

where, the activity factor (AF) (e.g., $activity_{kernel}$), depends on the execution time and net data transfer (read+write) between the L2 cache and the memory controller. That is:

$$AF \propto (FetchSize + WriteSize)/exec\_time. \tag{5.5}$$

Finally, we apply an area-based wire-length model to scale the global interconnect power for each application kernel from the current hardware to future hypothetical GPU hardware using relative wire distances. The interconnect power of a kernel (Pic_{kernel}) on future GPU is computed as:

$$Pic_{kernel} = Pic_{power\_virus} * DF * SF, \tag{5.6}$$

where, Pic_{power_virus} denotes the interconnect power of a reference power virus application on current hardware and SF denotes the wire-length scaling factor to account for hardware scaling and change in interconnect topology. For example, for a future GPU with 192 CUs, we multiply the interconnect power of the 32 CU baseline GPU by a scaling factor of 6. This assumes that the global interconnect architecture remains the same for future GPUs and the wire lengths scale up, which may not be true if other network topology is used or 3D stacking is deployed. The wire-length scaling model is a function of the on-chip interconnect topology. Our flexible framework gives us the ability to apply

different optimization techniques and topologies on the interconnect such as including a Network on Chip (NoC) power model simply by changing the wire-length scaling models to reflect changes in the average data movement distance.

c. Technology Scaling and Temperature Impact

Until this step, technology scaling has not been considered. Technology scaling affects area, voltage, and the relative power contributions between transistors and interconnects, as well as the split between dynamic power and leakage power. Different commercial fabs have different technology scaling characteristics such as the voltage-leakage and temperature-leakage dependency. We model temperature scaling for future node by accounting for worst-case junction temperature. In our case, we set it to 105 °C based on real silicon data. In addition, new technologies such as FinFET also introduce disruptive points along the scaling curve. Research-oriented projections from ITRS are not always representative of industrial designs. In this work, we use proprietary industrial process models (from AMD) to scale a hypothetical future GPU design from 28 nm to 10 nm including introduction of FinFET at 14 nm and beyond. Our parameterized scaling model accounts for the different scaling trends for gate and metal capacitances as well as leakage currents with process technologies. To project power at different voltage and frequency points of interest in the future technology node we scale the dynamic power from nominal voltage-frequency point at the future process in proportion to $(V/V_{nom})^2 * (f/f_{nom})$ [13]. Similarly, we super-linearly scale the leakage power with voltage at the different points of interest based on leakage versus voltage relationship at that technology.

5.3.2 Practical Considerations

Here, we list out a few practical considerations we accounted for when developing the model and projections.

Leakage power. Leakage power on the HD 7970 was measured at a constant temperature maintained by a thermal head. Temperature is measured from an on-chip thermal diode. We measured power consumption of a power virus across different frequencies at a fixed voltage. By linearly extrapolating power measurements at different frequencies back to zero frequency, we derive the leakage power at the maintained temperature. By repeating the experiments multiple times at different temperatures, we also extract the relationship between leakage and temperature. Leakage power at future technology nodes is extracted from these hardware measurements scaled by models co-developed by silicon fabrication vendors and AMD for future process technologies. In addition, we observe that runtime silicon hot spots are mostly located in CUs which consume a significant portion of the GPU power. Also performance is often limited by the worst case temperature hotspots in the CUs. Therefore, we make a reasonable assumption of modeling leakage power of the entire GPU at the worst case silicon junction temperature for the future technology. This is because it is infeasible to experimentally account for leakage of individual GPU micro-architectural components due to the lack of very fine-grain temperature monitoring capability on existing hardware.

Power Gating vs. DVFS. Power gating (PG) and DVFS are orthogonal power management techniques. Given the large number of CUs and the increasing portion of leakage power in future technology nodes, we argue that PG should be considered first as an important means of reducing power when there are inactive CUs. DVFS is more practical for adjusting active CU power consumption according to the intensity of the kernel activities.

Different GPU core architecture. We demonstrate our methodology and results based upon AMD's Graphics Core Next (GCN) architecture and assume that future GPU scales out to more number of CUs with a similar architecture. There may be alternative ways, for example, keeping number of CUs relatively constant and scale up individual CUs to make them more complicated. We argue that power gating is still beneficial, as long as there are applications that cannot utilize all available CUs. Therefore, our proposed methodology and management schemes are still valuable.

FinFET technology. In our analysis, the impact of FinFET technology on power is automatically accounted for as FinFET is an integrated part of the underlying process technology models used for this work. For example, the dynamic and leakage power scaling factors from 20 nm to 14 nm are noticeably different from those of 28 nm to 20 nm due to introduction of FinFET at 14 nm. We lumped all technology scaling factors across different nodes to reach the final scaling factors at 10nm, our technology node of interest.

Application algorithms. We assume same workloads are running on the existing GPU and the future GPU. This assumption is not realistic as applications will get optimized and replaced by improved algorithms over time. However, since our focus is to study practical approaches to power gating, we ignore the effects of any application changes and leave projecting power and performance for an unknown workload to future work. Instead we project power and performance of the same workloads with same input sets at different design points of interest in the future GPU. The proposed methodology is also applicable to non-HPC workloads such as graphics.

Next, we describe the workload-aware PG design techniques and design-time aware run-time power management scheme for future GPUs in the following subsections.

5.3.3 Workload-Aware Design-time Analysis

As shown earlier in Figure 5.1, workloads have varying amount of parallelism trends due to their compute and memory requirements. Therefore, not all CUs are needed for getting the best performance of many applications; the unused CUs could be power gated to save power. Leakage power saved from gating unused CUs can be used towards boosting frequency at run-time in a TDP-constrained design. The finer the PG granularity, the higher is the boosting potential. However, beyond a certain granularity, if the maximum current and frequency allowed for a given PG transistor sizing are reached, further frequency boosting becomes impossible. Similarly, with finer granularity, savings in leakage power become much smaller leading to relatively smaller performance improvement and a point of diminishing return. Therefore, we propose a methodology to evaluate leakage power, frequency boosting factor, and area overhead at different PG granularities as a function of the optimal number of CUs needed by an application at run-time and its frequency-boosting potential.

Leakage power analysis. First, we model the effects of PG granularity on total leakage power by considering the optimal number of CUs needed by an application. Let's assume that there are N CUs in a GPU device and that the device has a PG granularity of s. If an application needs n CUs for its optimal performance, and if we denote the average leakage power of one CU by p_1, then the total leakage power, denoted by $P_L\,(n, s)$, of the system when n CUs are active can be expressed as

$$P_L\,(n, s) = p_1 s \left\lceil \frac{n}{s} \right\rceil, \quad n \leq N \text{ and } s \leq N, \tag{5.7}$$

where, $\lceil \cdot \rceil$ denotes the ceiling operator. The ceiling operator used in the above equation provides the number of clusters needed to activate the desired number of CUs (n) for a given PG cluster-size (s). So, even if there is only one CU active in a cluster,

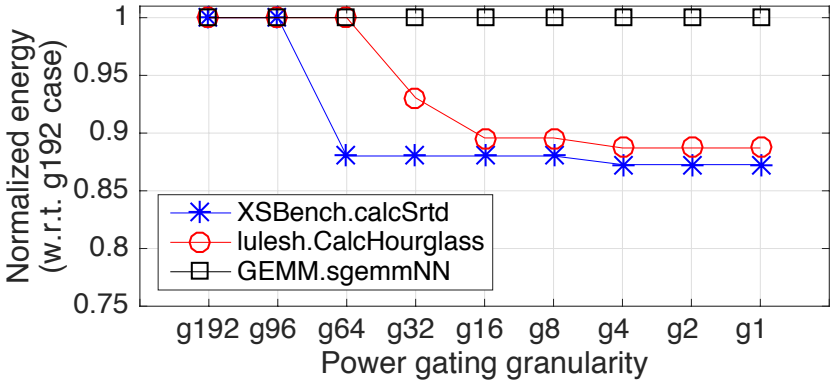

Figure 5.5: Normalized energy of selected kernels at different power gating granularities.

the whole cluster needs to be enabled, with other CUs dissipating idle leakage power. We note that for fixed number of active CUs, the leakage will be different for different PG granularities. Further, different application kernels have different power gating opportunities due to differences in their characteristics in terms of compute intensity, memory intensity, inter-thread conflicts and control divergence. For example, Figure 5.5 shows the normalized energy versus PG granularity for three example kernels, namely, `XSBench.calcSrtd`, `lulesh.CalcHourglass`, and `GEMM.sgemmNN`, obtained through our projection models. We sweep the number of CUs per cluster from 192 (i.e., g192 with no CU-level power gating) to all the way to 1 (i.e., g1 with per-CU power gating) and analyze each kernel's normalized energy with respect to the baseline case of no CU-level power gating. We observe "knee points", in terms of energy vs. granularity, across different kernels. In particular, we see that the energy reduction of these kernels almost flattens out after a granularity of 16. Thus, we seek an optimal PG design that balances the benefits of fine-grain granularity (performance and power efficiency) with the cost (die area overhead) of the system.

Frequency boosting. As shown earlier in the Figure 5.1, some HPC kernels/applications do not require all available CUs to be active for their best performance. The performance of these kernels can be potentially improved by turning off unused CUs and using that

power towards boosting the operating frequency as long as the total power is below the TDP and the die temperature does not exceed the maximum allowed junction temperature. However, increasing the operating frequency requires increasing the operating voltage for the correct functioning of the device, resulting in increases in both the dynamic power and the leakage power of the chip. The amount of boosting depends on the PG granularity of the design, the TDP, and the maximum die temperature of the device. For our analysis, we use the worst-case die temperature to ensure deterministic performance under a fixed cooling solution irrespective of the process and ambient variation across parts. In addition, one has to consider a realistic worst-case scenario for the design time analysis. Finer PG granularity could provide more power savings, and therefore, higher frequency-boosting.

For every kernel, we compute the potential frequency boosting factor and the associated factor of increase in voltage with respect to the nominal voltage and frequency so that the total power at the boosted frequency is below TDP. The increase in frequency will be accompanied by an increase in current in the device. We compute the increase in current as the ratio of increase in power to the increase in voltage. Finally, an increase in the switching current would require a proportional increase in the size of the sleep transistor-size to reduce IR-drop across the transistor. Since larger transistors are needed to support higher frequency, the kernel with the maximum frequency-boost dictates the size of transistors.

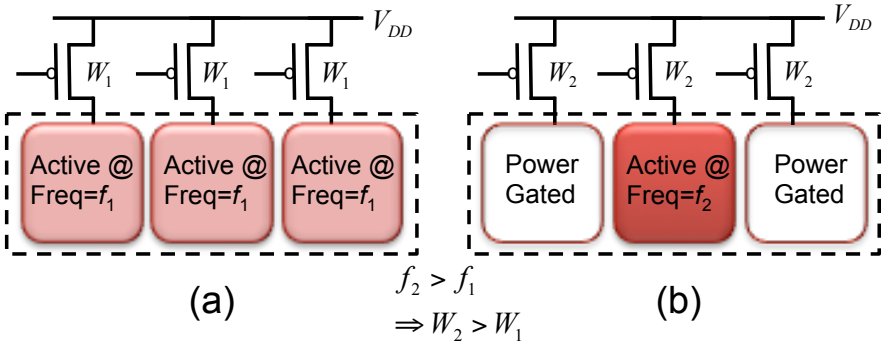

Figure 5.6: Sleep transistor sizing for frequency-boosting.

PG area overhead. Implementing PG requires adding power gates, buffers, clamp cells, I/O buffers, and control logic to the design [53], which increases its total area. Further, the area overhead depends on the granularity at which the power gating is implemented in the design and the amount of frequency boosting that can be allowed under TDP during run-time. The area overhead, A_{ov}, due to PG can be expressed as

$$A_{ov} = A_{gates} + A_{aon} + A_{cntrl}, \tag{5.8}$$

where, A_{gates}, A_{aon} and A_{cntrl} denote the area overhead due to power gates/buffers, always-on (AON) cells, and control logic, respectively. A_{gates} scales with the frequency-boost, which is different for different PG granularity designs and kernels. That is

$$A_{gates} = A_{gates\_f_0} \frac{I_{fboost\_max}}{I_{f_0}}, \tag{5.9}$$

where, $A_{gates\_f_0}$ is the area overhead of power gates at the nominal frequency, f_0, of the design, and I_{fboost_max} denotes the current at the maximum frequency-boosting factor, which is decided by the kernel with the largest power slack with respect to TDP. We use current instead of power for circuit area overhead analysis because the power gating transistor sizes are governed by the current flowing through them, not power. The interplay between the size of PG transistors and potential frequency-boosting is explained pictorially in Figure 5.6, where, the chip is assumed to have three power-gating clusters. When all CUs are active, as shown in Figure 5.6 (a), the maximum operating frequency to keep power below TDP limit is f_1 and corresponding equivalent width of sleep transistors is W_1. However, when only $1/3^{rd}$ of the CUs are active, the frequency of active CUs could be boosted to f_2 without violating the TDP limit; however, this would require the sleep transistor-width to be increased from W_1 to W_2, as shown in Figure 5.6 (b). The silicon area overhead as well as the switching capacitance overhead are obviously different in these two cases. It

is worth mentioning that the area overhead due to PG transistors starts saturating as the maximum frequency is attained.

As the PG cluster-size, s, increases, fewer number of AON cells are needed due to reduced intra-cluster signals. Thus, A_{aon} is modeled as a product of the perimeter of the cluster and the number of PG clusters in the device. For a cluster size s, where its CUs arranged as $s_v \times s_h$ grid, the perimeter of the cluster will be $2\,(s_v + s_h)$, and with N CUs, gated at a granularity of s CUs per cluster, the number of clusters in the chip will be $\frac{N}{s}$. So, A_{aon} is modeled as

$$A_{aon} \propto (s_v + s_h)\,\frac{N}{s}, \quad \text{such that} \quad s = s_v \times s_h. \tag{5.10}$$

Hence, as the number of cluster increases, the overall area due to A_{aon} cells increases. Finally, the area overhead due to PG control logic and associated sleep/wake signals is modeled as a linear function of the number of PG clusters; that is

$$A_{cntrl} \propto \frac{N}{s}. \tag{5.11}$$

Figure 5.7 shows the layout of a compute unit from an industrial GPU design that has power gates inserted in a checker board pattern [53]. In this figure, the snapshot on the right shows different power gating components (i.e., power gates, always ON (AON) cells, and I/O buffers) in the zoomed area marked by a white rectangle on the layout. We use this layout to compute the area overheads from these components used for implementing power gating in the design. The area overhead results are presented in Section 5.4.

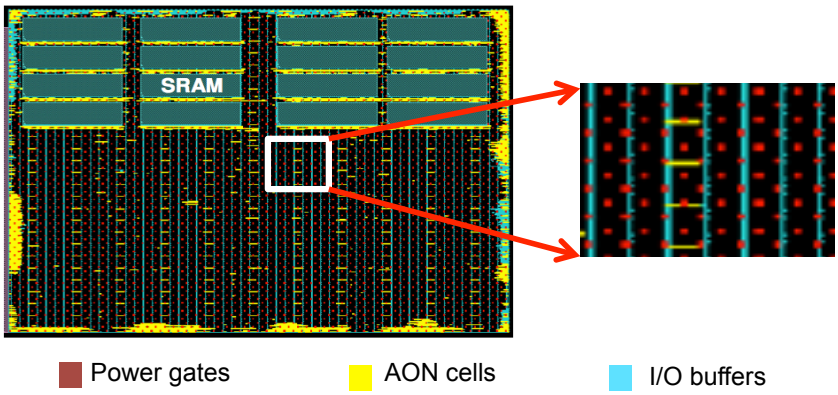

Figure 5.7: Layout of a real GPU compute unit showing power gates, always-on (AON) cells and I/O buffers [53]. The snapshot on the right shows the zoomed area marked by white rectangle on the layout.

5.3.4 Design and Workload-Aware Run-time Management

Our goal is to devise a run-time power management system that can leverage knowledge of the PG granularity to maximize the performance and power benefits of the device for all workloads. To implement an effective run-time system, the first step is to build a predictor to determine the number of CUs that could be power gated dynamically. Different kernels require different number of active CUs to achieve their best performances, as shown earlier in Figure 5.1, so the predictor has to be workload-aware. The second step is to encapsulate the predictor into a run-time power management algorithm that periodically sets the optimal number of CUs required for the kernel's execution. Our goal is to develop a simple and practical predictor that can be implemented efficiently in a run-time algorithm with minimal hardware overhead and complexity.

Performance correlation against hardware counters. We studied the correlation between 20+ hardware performance counters with the performance for all kernels at different CU counts on our baseline hardware, and we found that GPU compute utilization, namely, *VALUBusy*, has a very strong correlation (>0.99) with the performance of an

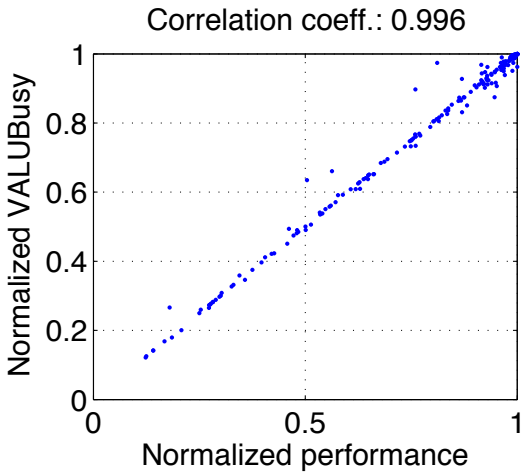

Figure 5.8: Correlation between *VALUBusy* and performance for 25 kernels.

application across all kernels (as shown in Figure 5.8). *VALUBusy*, also denoted as V in Algorithm 2, represents the percentage of GPU-time when vector instructions are being processed, where higher values indicate higher compute units utilization. Further, we scale the *VALUBusy* to the future architecture by using the scaling methodology described in Section 5.3.1, to obtain performance trends of kernels against the number of active CUs.

Power management algorithm. We propose a run-time power-management algorithm that searches for the optimal CU count dynamically using application characteristics and PG granularity information. The algorithm applies a gradient-based analysis towards power gating idle CUs and adjusting the frequency of active CUs for an application kernel under a given TDP. This algorithm can be invoked at any sampling interval (per-kernel, per-application, at fixed intervals within a kernel). In this work, we invoke it at every kernel boundary at every iteration due to current hardware limitations. The proposed algorithm, given in Algorithm 2, has three main components: 1) initialization, 2) gradient computation, and 3) configuration prediction.

1. *Initialization.* During initialization we run the kernel at two different CU configurations and collect *VALUBusy* values (line 1-2 of Algorithm 2). The convergence time of our

Algorithm 2: Gradient-based algorithm to find optimal CU count and frequency for a kernel during run-time.

```
   Data: PG granularity (s), minimum step-size (Δn_0), nominal frequency (f_0)
   Result: Optimal CU count and frequency
1  Initiatlization();
2  k = 2;
3  // Find optimal CU count at nominal frequency (f_0)
4  // VALUBusy denoted as V; Gradient as G
5  while (ΔV > tol) OR (G <= 0) do
6      k = k + 1;
7      n_k=PredictNumCU(n_{k-1}, Δn_0, G);
8      P_k = PredictPower(n_k, f_0, s);
9      if P_k <= TDP then
10         Run kernel at n_k and measure V_k;
11         ΔV = (V_k − V_{k-1})/V_{k-1};
12         G = (V_k − V_{k-1})/(n_k − n_{k-1});
13     else
14         k = k − 1; // TDP exceeded
15         ReduceNumCU();
16     end
17 end
18 // Find optimal operating frequency
19 while P_k < TDP do
20     Boost the operating frequency;
21     //Use PG granularity while varying frequency
22     PG_Granularity_PredictNumCU();
23 end
24 Optimal CU count = n_k at the highest V_k;
```

iterative algorithm depends on the difference in performance between these initial starting points and is a function of the actual optimal CU count of the kernel, which depends on the kernel characteristics. The larger the difference, more iterations are needed to converge. Empirically, we choose 96 and 100 CUs as the starting configurations to balance the convergence rate of the algorithm against the energy savings and/or performance gains, which can happen if the starting configurations have too low CU counts or too high CU counts.

2. *Gradient computation.* Next, we compute the gradient (G) for *VALUBusy* by taking the ratio of the change in the value of *VALUBusy* counter to the change in the active CUs (line 12 of Algorithm 2). We denote the minimum step-size for change in CU count in the system as Δn_0; we can increase or decrease the number of active CUs only in the steps of integer multiple of Δn_0. The algorithm predicts the number of active CUs for the current

iteration (k) of the kernel from the number of active CUs and the gradient of *VALUBusy* (V) counter with respect to the number of CUs in the previous step ($k-1$); this step is denoted as PredictNumCU(.) function in line 7 of Algorithm 2). That is

$$n_k = n_{k-1} + \Delta n_0 * \text{sgn}\,(G) * \{1 + \text{round}\,(C * |G|)\}\,, \tag{5.12}$$

where, sgn$(\cdot)$ denotes the signum function, round(G) rounds the value of G to the closest integer, and the parameter C, found empirically, controls the converges rate of the algorithm. Further, due to adaptive and quantized step-sizes, it is possible to overshoot the maxima point in our proposed algorithm. Therefore, unlike standard gradient-ascent method, we reduce the number of CU count when the gradient is negative. Further, to avoid the algorithm from getting stuck at a fixed CU count [e.g., when round$(G) = 0$], we replace the gradient term in standard gradient-ascent method by $\{1 + \text{round}\,(C * |G|)\}$ term in Equation 5.12. The algorithm runs until the relative change in the optimization function (*VALUBusy*) is smaller than a specified tolerance value (tol). We set tol to 2%.

3. *PG design-aware configuration prediction.* In order to simplify the power/performance scaling models, either CU-count or operating frequency is changed at a time, and not both of them at the same time. Using the gradient-based technique and PG granularity information, we predict the optimal configuration based on the following cases.

Case 1: Activating optimal number of CUs under TDP at nominal frequency. Once the run-time algorithm predicts the optimal CU count for a kernel, it estimates the corresponding power consumption based on the previous power readings using PG granularity size. The algorithm estimates the total power (both dynamic and leakage) to check it against the TDP; this step is denoted as PredictPower(.) function in line 8 of Algorithm 2. For a given PG granularity, the worst-case leakage power of the chip is computed using equation 5.7. The dynamic power at the predicted HW configuration is estimated using linear extrapo-

lation/interpolation of the dynamic power values estimated at the past two iterations of the algorithm. If the predicted power consumption at the optimal CU configuration is more than the TDP, the algorithm reduces the CU count until the power constraint is met.

Case 2: Adjusting CU count and frequency to save leakage power and utilize available power headroom. If there is power slack or headroom available after selecting the optimal number of CUs for an application/kernel, the remaining power is utilized towards boosting the frequency. The amount of frequency boosting is derived by computing similar gradients for *VALUBusy* with respect to change in frequency (similar to line 12 of Algorithm 2). Configuration prediction is done for every kernel at various power gating granularity designs. In order to achieve the best performance from the available power slack, the algorithm varies the number of CUs within immediate integer multiples of the PG granularity and adjusts the frequency to meet the TDP. This step is denoted as PG_Granularity_PredictNumCU(.) in line 22 of Algorithm 2. We choose CU scaling before frequency boosting because for HPC type parallel applications CU scaling provides better power efficiency than frequency scaling.

The proposed management algorithm can be implemented anywhere in the power management firmware, GPU device driver, system software run-time APIs or Operating System (OS) of the future GPU. As an example, this can be achieved by exposing CU power gating control and *VALUBusy* register to the OS and implementing the predictor algorithm in the OS policies. In the absence of future hardware, we use offline analysis to analyze the performance and power efficiency results in this work.

5.4 Evaluation Results

In this section, we present the following results– a) Methodology validation; b) optimal PG granularity across different workloads; c) area overheads of implementing power gating at different granularities; d) evaluation of run-time power management algorithm; e) additional performance gains through frequency boosting; f) effect of TDP on optimal power gating granularity. We present our run-time performance and power efficiency results under different power gating granularity designs using a future massively parallel GPU, as described in Section 5.3 with 192 CUs, 2 TB/sec memory bandwidth, 1 GHz nominal frequency and voltage at 10 nm process technology. Table 5.2 shows the list of all 25 kernels (labelled as K1 to K25), along with their oracle optimal CU count obtained through the offline-analysis.

a. Methodology Validation

We validate our power and performance projection methodology by using measurements at configurations with low CU count to predict the runtime and power at configurations with high CU counts at multiple memory and CU frequencies within the design operat-

Table 5.2: Optimal number of CUs for the studied kernels.

Kernel Name	**Opt. #CUs**	**Kernel Name**	**Opt. #CUs**
CoMD.EAM.advPos (K1)	124	GEMM.sgemmNN (K14)	192
CoMD.EAM.advVel (K2)	104	GEMM.sgemmNT (K15)	104
miniFE.dotprod (K3)	64	XSBench.bitSort (K16)	124
miniFE.waxpby (K4)	192	XSBench.calcSrtd (K17)	124
minife.matvec (K5)	184	XSBench.uGridSrch (K18)	152
MaxFlops.peak (K6)	192	bprop.adjW (K19)	104
lulesh.cacFBHour (K7)	184	bprop.lfwd (K20)	192
lulesh.integStress (K8)	184	cfd (K21)	184
lulesh.calcHourglass (K9)	140	hotspot (K22)	88
graph500.locRedNext (K10)	192	Device memory (K23)	64
graph500.botUpStep (K11)	124	Nbody (K24)	184
graph500.unionClear (K12)	64	kmeans.swap (K25)	80
kmeans.kernel_c (K13)	184		

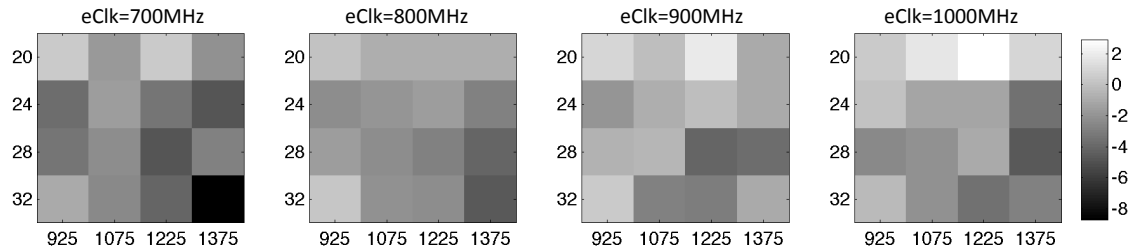

Figure 5.9: Performance model prediction errors (%) for `miniFE.waxpby` on the baseline hardware at memory frequencies: 925-1375 MHz, #CUs: 20-32, CU engine frequencies (eClk): 700-1000 MHz.

ing points of today's GPU. Note that this does not include technology scaling. Figure 5.9 shows the performance prediction errors for `waxpby` kernel for range of CU count (20-32), memory clock (925-1375 MHz) and CU engine clock frequencies (700-1000 MHz). Except for one case (32 CUs, 1375 MHz memory clock and 700 MHz CU frequency) with the prediction error of 8.8%, the maximum absolute errors in predictions for all other configurations are below 5%. The large error for one case is because the otherwise memory-bound `waxpby` kernel behaves like a compute-bound kernel at lower CU frequency [65, 111]. Overall, the average performance and power prediction errors for `miniFE.waxpby` kernel are 2.1% and 1.6%, respectively, which are quite reasonable for an early stage study. Prediction outliers are attributed to hardware noise and small measurement set with the desired compute-to-memory ratio. Further, the average errors for all 25 kernels in predicted execution time and power against the actual measurements at a fixed configuration of 32 CUs, 1375 MHz memory clock and 1 GHz CU cock fre-

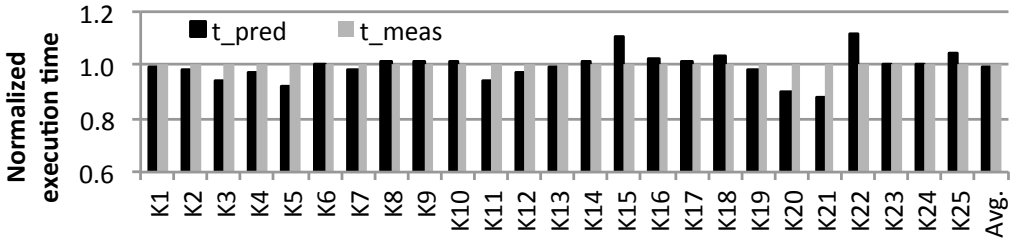

Figure 5.10: Predicted vs. measured normalized execution time at the 32 CU, 1 GHz eClk, and 1375 MHz mClk frequency of HD 7970 for the selected kernels.

quency are 2.0% and 1.1%, respectively. Figure 5.10 shows the predicted (t_pred) and the measured (t_meas) execution times; the results for power and also similar.

b. Optimal PG Granularity

To derive the proper PG granularity, which is a design-time decision, we need to consider the following tradeoff. On one hand, the granularity needs to be finer so that we do not miss power saving (hence, performance boosting) opportunities. On the other hand, based on the area overhead analysis in Section 5.3.3, the granularity needs to be coarser to reduce silicon area overhead and cost. To reach an optimal tradeoff, we first look at the run-time performance and power efficiency of a future GPGPU with different PG granularities at a nominal 1 GHz compute frequency and under a 150 W TDP constraint.

With 150 W TDP constraint and the nominal 1 GHz frequency, Figure 5.11 (a)-(b) show the execution time and energy for the selected kernels at different PG granularities using our design-aware run-time management algorithm, normalized to the baseline case where all CUs power gated together, i.e., g192. Note that power efficiency is inversely proportional to energy, so lower energy in Figure 5.11.b means higher power efficiency. Similarly, performance is inversely proportional to execution time. The figure shows results from selected kernels for seven PG granularities: 1, 8, 16, 32, 64, 96, 192 *CUs per cluster* (denoted as g1, g8, g16, g32, g64, g96, and g192 in Figure 5.11) that can be power gated at the same time. This corresponds to 192, 24, 12, 6, 3, 2, and 1 independent power gating domains. We choose these granularities mainly because they are divisible to the total 192 CUs. For other high-performance parallel processor architectures with different number of CUs, the cluster size choices may vary. However, the proposed decision methodology would still hold.

For most kernels, we see improved performance and power efficiency at finer PG gran-

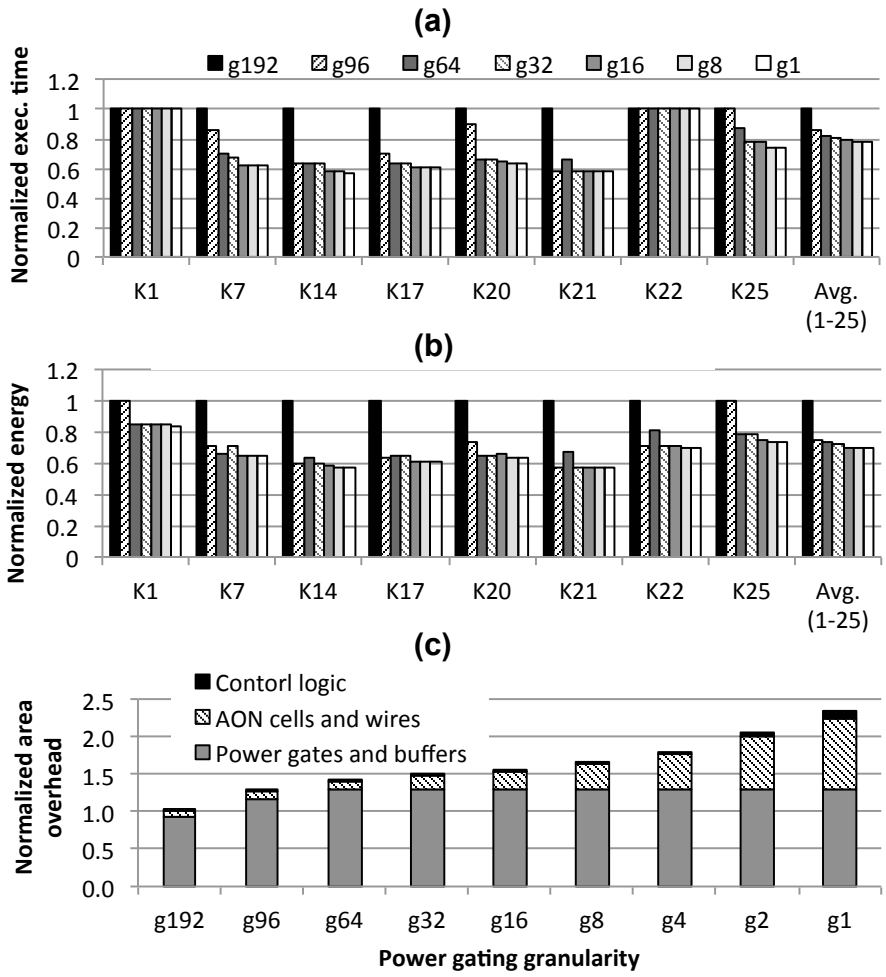

Figure 5.11: a) Execution time, and b) energy of kernels at different PG granularities with TDP = 150 W, c) power gating area overheads at different PG granularities.

ularity with diminishing return beyond g16. The improvement comes from the fact that kernels are TDP limited and the more leakage power saved from finer-grained power gating can be leveraged to activate more CUs, which in general improves performance and power efficiency. In fact, the largest performance improvement of 14% is seen between one PG domain (g192) and two PG domains (g64) for the entire GPU. However, beyond a certain PG granularity, the additional leakage power saving becomes smaller, and hence the diminishing returns in performance and power efficiency with more silicon area overhead. Specifically, there is no significant performance and power efficiency benefit between 16 CU PG granularity and per-CU PG granularity. Two exceptions are the

performance of `CoMD.EAM.advPos` (K1) and `hotspot` (K22), where we see no performance changes across different PG granularities as compared to the baseline because the two kernels do not exceed the 150 W TDP even without power gating and can run at their optimal CUs.

Across 25 kernels we have investigated, we find that 16 CU cluster size (g16) is the finest granularity that is necessary. At g16, we find an average performance and power efficiency improvement of 21% and 30%, respectively, compared to the baseline case g192. The actual cluster size at design time may change as different applications and kernels may frequently run on the future hardware system. As long as the cluster size is decided based on the characteristics of a realistic and broad set of applications and kernels, our methodology is applicable. Hence, we conclude that there is an optimal PG granularity shared by most workloads.

c. PG Area Overhead

We have seen that finer PG granularity coupled with effective run-time algorithm provides better performance and power efficiency for a TDP-constrained design. However, the improvements come at the cost of design and area overhead incurred from implementing fine-grain power gating. Previous results in the literature report a PG area overhead from 5% to 40% of total die area [53, 46, 97]. Without loss of generality, we use relative percentage values in this paper. For the CU design of Figure 5.7, which represents a granularity of 1 CU per cluster (g1), the contributions from power gates, AON cells, and control logic are estimated as 53%, 40%, and 7% respectively based on area estimates from the layout. Using the proposed analysis in Section 5.3.3, we compute the PG area overhead at other granularities for all kernels. Among all kernels studied, `Graph500.unionClear` kernel has the highest frequency boosting potential at run-time and, therefore, determines the worst-case area overhead. Figure 5.11 (c) gives the area overhead at different PG granular-

ities normalized to the 192 CU granularity (g192) case. Compared to g192, where all CUs can be power gated at once, the overhead for g96 and g16 increases to 27% and 54% respectively. The overhead becomes 2.35× for per-CU power gating granularity. Typically, the kernel with low parallelism and low power has the highest frequency-boost potential. The optimal number of CU count for `Graph500.unionClear` kernel is 64, so any granularity finer than 64 CUs has the same power dissipation and power slack, and hence, the same frequency-boost potential. So, beyond the 64 CU granularity, the area overhead due to PG transistors also starts saturating as the maximum frequency is attained. However, the overheads due to AON cells, control logic and wires keep on increasing. Hence, as the granularity is varied from the coarsest (g192) to the finest (g1), the overhead due to clamp cells increases by $13.7\times$, resulting in an overall area increase of $2.35\times$.

By comparing the performance and energy gains against the PG design area overhead at different PG granularities given in Figure 5.11, we observe that per-CU PG provides only about 1% improvement in performance at the cost of 53% increase in the PG area overhead compared to the 16 CU granularity design. However, 16 CU granularity provides 21% improvement in performance and 30% improvement in power efficiency at the cost of only 54% increase in PG area overhead compared to 192 CU (single-cluster) granularity. So, we conclude that 16 CU PG granularity is an optimal design choice for the studied massively parallel GPGPU architecture and per-CU power gating is an overkill. Choosing 16 CU PG granularity over per-CU granularity could reduce the die area overhead from 5-40% [46, 97] to 3-28%.

d. Run-time Power Management Algorithm

Our run-time algorithm, as described in Section 5.3.4, first predicts the optimal CU counts by monitoring *VALUBusy* at nominal frequency. Figure 5.12 shows the predicted CU counts for four kernels: `CoMD.EAM.advPos` (K1), `MaxFlops.peak` (K6), `XSBench.-`

`calcSrtd` (K17), and `XSBench.uGridSrch` (K18), together with their corresponding oracle CU counts, derived from off-line execution time estimates. As we can see, tracking the *VALUBusy* does lead to very accurate optimal CU count prediction.

Compared to the existing work, which uses static analysis to evaluate the benefits of per-core PG in existing hardware with fewer number of processing units, the proposed Algorithm 2 considers PG granularity and performance sensitivity to number of CUs to change the number of active CUs during run-time for a massively parallel architecture. Figure 5.13 shows the convergence behavior of the proposed algorithm for the same four kernels of Figure 5.12. After initialization, the algorithm predicts the number of CUs based on the percentage change in *VALUBusy* and the gradient of *VALUBusy* with respect to CU-count, as shown in Figure 5.13. The algorithm has three kinds of stopping criteria: 1) when it finds the maxima point of *VALUBusy* with respect to CU-count, 2) when the predicted CU-count reaches minimum or maximum number of CUs in the system, and 3) when the relative change in *VALUBusy* is smaller than the specified tolerance value, 2% in our case. In all cases, the algorithm ensures that the total power at the predicted configuration is less than or equal to TDP.

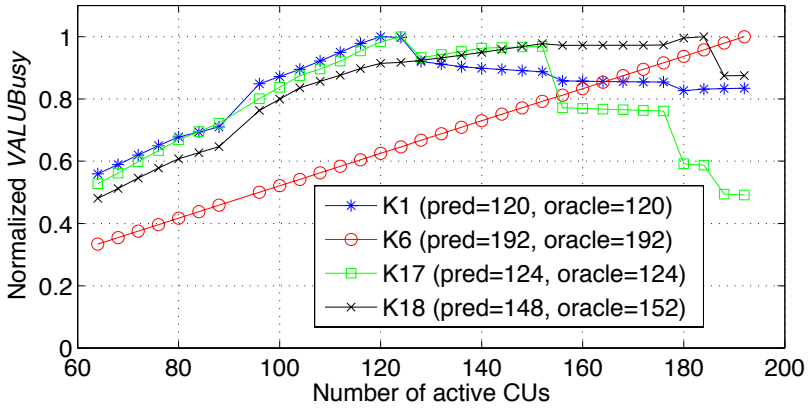

Figure 5.12: Normalized *VALUBusy* across the number of CUs and predicted vs. actual optimal CU-count.

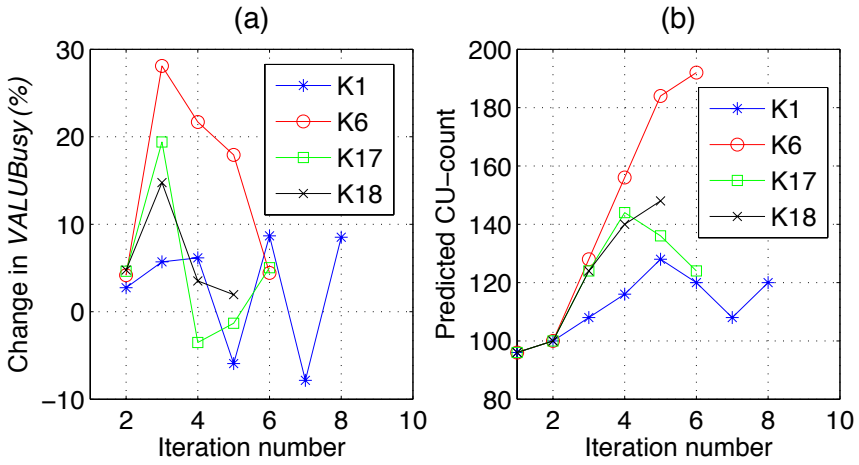

Figure 5.13: Algorithm convergence. (a) % change of *VALUBusy* in two consecutive iterations. (b) progress of predicted optimal CU counts across kernel iterations.

We show all three types of convergence behaviors in Figure 5.13. First, for `CoMD.EAM-.advPos` (K1) and `XSBench.calcSrtd` (K17) kernels, the algorithm overshoots the CU-count prediction and retreats back to the optimal CU-count configuration. For example, in `CoMD.EAM.advPos` (K1), the algorithm predicts 120 CUs in the 6^{th} iteration by decreasing the CU-count from 128 CUs in the 5^{th} iteration. Since, there is an increase in *VALUBusy* at this step, the algorithm reduces the CU-count further to 108 CUs in the 7^{th} iteration based on the gradient value. However, the *VALUBusy* at 108 CUs is smaller than the *VALUBusy* at 120 CUs, so, the algorithm increases the CU-count back to 120 CUs and stops. Second, for `MaxFlops.peak` (K6) kernel, the performance (or *VALUBusy*) keeps increasing with respect to number of active CUs, so the algorithm keeps increasing the CU-count until the maximum number of CUs in the system is reached. Third, for `XSBench.uGridSrch` (K18) kernel, the percentage change in performance is lesser than the tolerance value above 148 CUs, so it stops after 5^{th} iteration. In this case, the algorithm predicts 148 CUs, although it has 1% lower performance than at 152 CUs (the oracle CU-count). For all kernels that we have investigated in this paper, the algorithm finds the optimal CU counts in lesser than 8 iterations. Given the fact that kernels usually go through tens of iterations, and dynamically changing hardware configurations requires only a few microseconds, the proposed algorithm introduces very little runtime overhead.

e. Design-aware Frequency Boosting

The run-time algorithm could improve the performance further for less-than-TDP kernels that are frequency sensitive with relatively flat CU scalability around the optimal CU count. Figure 5.14 (a) shows the performance boost of four kernels that consume less than TDP at their optimal CU count. Among them, `CoMD.EAM.advVel` (K2) and `graph500.botUpStep` (K11) have 11% performance improvement by running at higher frequency. Similarly, Figure 5.14 (b) shows performance improvement for four kernels, `kmeans.kernel_c` (K13), `GEMM.sgemmNT` (K15), `XSBench.bitSort` (K16) and `bprop.adjW` (K19) with our PG design-aware runtime algorithm in a design with 16 CU granularity. The left bars indicate performance for the kernels, where a design-decoupled runtime always enables optimal number of CUs for an application without considering any effects of PG granularity, resulting in additional CUs idle but ungated due to granularity size. These kernels do not see any frequency boosting as they reach TDP limits because of leakage power dissipated by the idle CUs. However, our design-aware run-time algorithm, as indicated by the right bars for these kernels, chooses to turn on CUs that are multiple of PG granularity and utilizes the saved idle leakage power and remaining power headroom to boost frequency leading to up to 18% better performance. It also translates to average 5% additional power-efficiency improvement for the selected kernels.

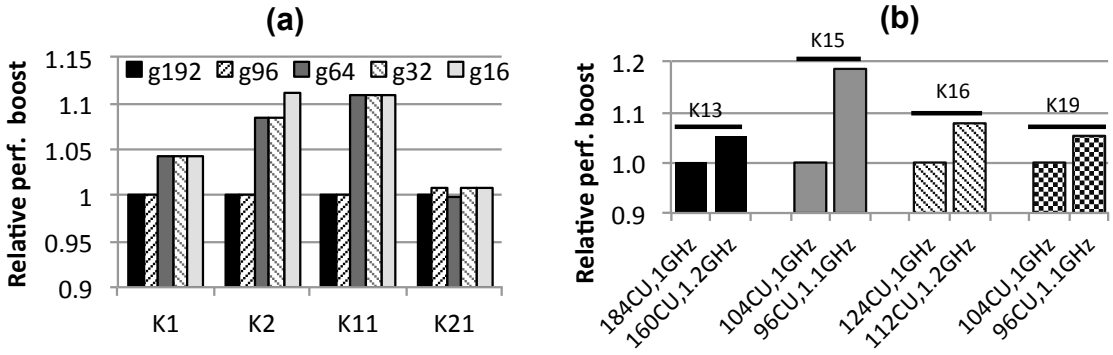

Figure 5.14: Performance boosting by increasing the frequency to use the power slack.

By exposing the PG granularity as a readable register or an API, OS/driver/firmware can easily access such information and make appropriate power dating decisions. Hence, additional performance gains can be achieved by boosting the frequency through a design and workload-aware run-time algorithm.

f. Effect of TDP on Optimal PG Granularity

So far, we have assumed a 150 W TDP during run-time power gating analysis. It is necessary to look at other TDP values and see if the 16 CU cluster decision is still valid. Figure 5.15 shows an example of such analysis for the `miniFE.matvec` (K5) kernel at 125 W, 150 W and 175 W TDP, which is a representative high-power kernel that can reach all TDP levels with different number of active CUs. We can see that execution time reduces as TDP increases as a result of more CUs being active. In all three TDP levels, performance starts to flatten out beyond a power gating granularity of a 32 CU cluster size, and a 16 CU cluster size is good enough for the studied design. Our run-time algorithm is able to adapt to the optimal CU count under different TDP constraints. We observe that design-time PG granularity is mostly independent of TDP.

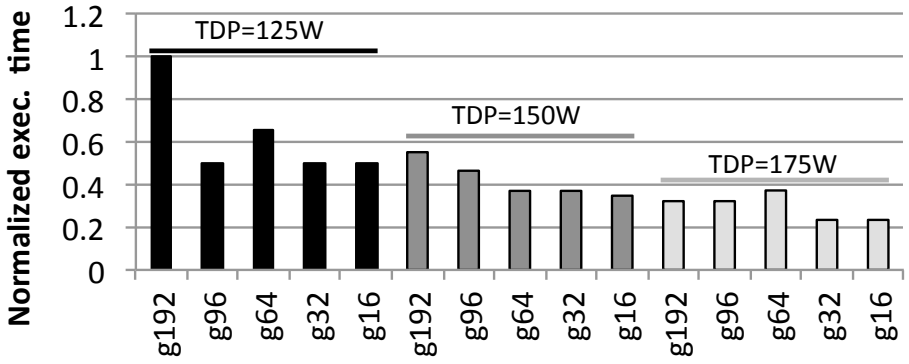

Figure 5.15: Normalized execution time of `miniFE.matvec` kernel (K5) at different PG granularities and three different TDPs.

5.5 Summary

In this chapter, we investigated how to leverage power gating to improve performance and power efficiency for future massively parallel GPU architecture. We showed that the optimal power gating granularity is a design-time architecture knob that governs the maximum gain that can be achieved during runtime. Further, our results demonstrate that finer power gating granularity can result in large area over-heads, whereas a sub-optimal power gating granularity can also result in wastage of leakage power and performance degradation under a fixed TDP for applications that need fewer number of CUs than the power gating granularity. By scaling measurements from a real 28 nm GPU to a hypothetical future GPU with 192 CUs in 10 nm node, we showed that a PG granularity of 16 CU/cluster achieves 99% peak run- time performance without the excessive 53% design-time area overhead of per-CU PG. We also demonstrate that a run-time power management algorithm that is aware of the PG granularity leads to up to 18% additional performance through frequency- boosting under thermal-design power (TDP) constraints. Thus, there is a design-time tradeoff to be made, and it is important to make the decision application-aware to implement an efficient power management in future GPU systems.

Chapter 6

Summary of Dissertation and Potential Future Extensions

In this thesis, we proposed new techniques to improve power efficiency of current and future CPU-GPU processors. To validate our techniques, we used extensive set of experiments on real hardware. To evaluate the proposed techniques on existing hardware, we used a quad-core CPU and an accelerated processing unit (CPU-GPU processor) from AMD. Similarly, for evaluating low-power design technique (in particular, power gating) on future hardware (massively parallel GPU), we first make measurements on an existing GPU by running different workloads and use those measurements to make projections on the future hardware. This chapter summarizes our contributions by highlighting the key results and discussing the potential research extension of the thesis.

6.1 Summary of Results

In chapter 3, we introduced multiple novel techniques that advance the state-of-the-art post-silicon power mapping and modeling. We devised accurate finite-element models that relate power consumption to temperatures, while compensating for the artifacts introduced by using infrared-transpired heat removal techniques. We devised techniques to model leakage power through the use of thermal conditioning. These leakage power models were used to yield fine-resolution leakage power maps and within-die variability trends for multi-core processors. We used an optimization formulation that inverts temperature to power and decomposes this power into its dynamic and leakage components and analyzed the power consumption of different blocks of a quad-core processors under different workload scenarios from the SPEC CPU 2006 benchmarks. Our results revealed a number of insights into the make-up and scalability of power consumption in modern processors. We also devised accurate empirical models that estimate the infrared-based per-block power maps using the PMC measurements. We used the PMC models to accurately estimate the transient power consumption of different processor blocks.

Further, for heterogeneous CPU-GPU processors, we showed that the integration of two architecturally different devices, along with the OpenCL programming paradigm, create new challenges and opportunities to achieve the optimal performance and power efficiency for such processors. With the help of detailed thermal and power breakdown, we demonstrated that choosing the appropriate CPU frequency and hardware device for CPU-GPU workloads are crucial to attain higher power efficiency for these devices. For the studied CPU-GPU processor, among different frequencies and two devices, the performance could vary up to 10.5×, while the total power and peak temperature vary up to 23.4 W and 40.5 °C, respectively. In other words, workload scheduling and DVFS are highly intertwined for these processors and should be decided appropriately.

In chapter 4, we presented a scheduling framework that takes in to account the system dynamic conditions, along with the workload characteristics to minimize runtime or energy on CPU-GPU processors. In contrast to previous approaches that either mapped entire applications or did not consider run- time conditions, our fine-grained approach enables scheduling at the kernel-level while considering system conditions during scheduling decisions to fluidly map the kernels between CPU and GPU devices. In a way, our approach complements the built-in hardware capabilities to limit TDP by incorporating the ability to schedule as well. To identify the best mapping for a kernel, we developed an SVM-based classifier that monitors the measurements of the performance counters to profile both the current workload and detects the number of available cores online, and accordingly decides the best device for the kernel to minimize total runtime or energy. We trained the classifier using off-line analysis that determined the best performance counters to use. We fully implemented the scheduler and tested it on a real integrated CPU-GPU system. Our results confirm its superiority as it is able to outperform application-based scheduling and the state-of-art scheduling methods by 40% and 31%, respectively. Similarly, our scheduling framework provides unto 10% more energy saving for selected time-varying TDP patterns than the user-based application-level scheduling scheme.

Finally, in chapter 5, we investigated in detail how to leverage power gating (a well known power saving technique) to improve performance and power efficiency for future massively parallel exascale and petascale GPU architectures. The analysis is based upon a scalable power projection framework that employs a combination of native hardware-execution, analytical and empirical scaling models, and industry-scale technology models to enable power efficiency optimization of future GPUs. We showed that a simplistic per-CU power gating granularity only incurs significant silicon area overhead without further benefits of run-time performance. Therefore, to achieve better power efficiency without sacrificing performance, we believe the design-time decision on optimal power gating

granularity needs to be aware of applications characteristics on run-time parallelism. This is particularly important to high-performance computing systems with massive amount of hardware parallelism, such as future GPUs in exascale and petascale HPC systems. We show that a PG granularity of 16 CU/cluster for a 192 CU hypothetical future GPU achieves 99% peak runtime performance without the excessive 53% design-time area overhead of per-CU PG. In addition to presenting the analysis methodology, we also demonstrate results with an efficient run-time algorithm that has the knowledge of underlying hardware power gating granularity. Our results show that a run-time power management algorithm that is aware of the PG granularity leads to up to 18% additional performance through frequency-boosting under thermal-design power (TDP) constraints.

6.2 Potential Research Extensions

In this thesis, we introduced multiple techniques to improve the power efficiency of modern processors. The ideas presented in this dissertation could be extending through the following possible research directions.

In this work, while performing the thermal and power mapping of heterogeneous CPU-GPU processors, we kept the GPU frequency fixed at factory settings due to publicly available drivers at the time. On the newer processors, with better power management capabilities, one could study the impact of adaptively changing the operating frequency and the number of compute-units of GPU based on the workload characteristics. Similarly, recently, researchers in both academia and industry are studying devices with different performance and power capabilities to target different market domains. For example, Intel is actively pursuing its goal of integrating reconfigurable computing with the x86 CPU cores [39, 18]. Integration of high power CPU with relatively lower power FPGA on the

same die will certainly add new dimension to the heterogeneous systems. One could apply our proposed power mapping and modeling techniques to understand and design better power management units on these systems. Similarly, our infrared imaging based power mapping setup could be used to study thermal and power issues on mobile processors and SoCs (e.g., Snapdragon from Qualcomm [99] and Tegra series from NVIDIA [80]).

Our online workload characterization and mapping work for CPU-GPU processors could be extended to multiple CPU and GPU devices for systems equipped with multiple such devices. The results shown in the thesis are for a system equipped with processor with one CPU and one GPU integrated on the same die. Multiple CPUs, GPUs, and even FPGA, will allow applications with wide range of kernel characteristics to run on a suitable device in the most power efficient manner. Our proposed scheduling algorithm could be scaled for efficient workload scheduling on such systems.

Finally, in this thesis, we provided a comprehensive analysis of power gating design to save power on future massively parallel GPUs. Due to very high power efficiency demand (about 25$\times$ the existing GPUs) of future exascale and petascale systems [73], multiple low power techniques will be needed to save power under different workload conditions. Some of these techniques are asynchronous logic to save clocking power, data-compression to save interconnect power, SRAM retention to save cache and register power, etc. The benefits of these techniques should be evaluated against their cost and design overheads. For example, we believe that interconnect power could be significant in future systems, so techniques such as 3D die stacking and processing in/near memory would be crucial to further reduce power associated with data movement. Our design framework could be extended to study such different low power techniques for future systems.

Bibliography

[1] M. Abdel-Majeed, D. Wong, and M. Annavaram. Warped Gates: Gating Aware Scheduling and Power Gating for GPGPUs. In *International Symposium on Microarchitecture (MICRO)*, pages 111–122, 2013.

[2] A. M. Aji, A. J. Pena, P. Balaji, and W.-c. Feng. Automatic Command Queue Scheduling for Task-Parallel Workloads in OpenCL. In *IEEE International Conference on Cluster Computing*, pages 42–51, Sept 2015.

[3] AMD. OpenCL Course: Introduction to OpenCL Programming. In *[Online] http://developer.amd.com*.

[4] E. K. Ardestani, F.-J. Mesa-Martinez, and J. Renau. Cooling Solutions for Processor Infrared Thermography. In *IEEE Symposium on Semiconductor Thermal Measurement and Management (SEMI-THERM)*, pages 187 –190, 2010.

[5] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198, February 2011.

[6] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. D. Supinski. Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems. In *International Conference on Parallel Processing*, pages 371–380, 2014.

[7] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris. Abstract: An Exascale Workload Study. In *High Performance Computing, Networking, Storage and Analysis (SCC), SC Companion:*, pages 1463–1464, Nov 2012.

[8] R. Bertran, M. Gonzalez, X. Martorel, N. Navarro, and E. Ayguade. Decomposable and Responsive Power Models for Multicore Processors using Performance Counters. In *International Conference on Supercomputing*, pages 147–158, 2010.

[9] K. Singhand M. Bhadauria and S. A. McKee. Real Time Power Estimation and Thread Scheduling via Performance Counters. In *Proc. Workshop on Design, Architecture, and Simulation of Chip Multi-Processors*, pages 46–55, 2008.

[10] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime Identification of Microprocessor Energy Saving Opportunities. In *International Symposium on Low Power Electronics and Design*, pages 275–280, 2005.

[11] M. Bohr. A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13, Winter 2007.

[12] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *International Symposium on Computer Architecture*, pages 83–94, 2000.

[13] J. M. Cebrín, G. D. Guerrero, and J. M. Garcia. Energy Efficiency Analysis of GPUs. In *International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1014–1022, 2012.

[14] S. Che, M. Boyer, Jiayuan M., D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, 2009.

[15] M. Cho, W. Song, S. Yalamanchili, and S. Mukhopadhyay. Thermal System Identification (TSI): A Methodology for Post-Silicon Characterization and Prediction of the Transient Thermal Field in Multicore Chips. In *Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, pages 118–124, 2012.

[16] H. J. Choi, D. O. Son, S. G. Kang, J. M. Kim, H.-H. Lee, and C. H. Kim. An Efficient Scheduling Scheme Using Estimated Execution Time for Heterogeneous Computing Systems. *The Journal of Supercomputing*, pages 886–902, 2013.

[17] J. H. Choi, A. Bansal, M. Meterelliyoz, J. Murthy, and K. Roy. Leakage Power Dependent Temperature Estimation to Predict Thermal Runaway in FinFET Circuits. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 583–586, Nov 2006.

[18] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In *Design Automation Conference (DAC)*, pages 109:1–109:6, 2016.

[19] R. Cochran, C. Hankendi, A. Coskun, and S. Reda. Identifying the Optimal Energy-Efficient Operating Points of Parallel Workloads. In *ACM/IEEE International Conference on Computer-Aided Design*, pages 608–615, 2011.

[20] R. Cochran, C. Hankendi, A. Coskun, and S. Reda. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *ACM/IEEE International Symposium on Microarchitecture*, pages 175–185, 2011.

[21] R. Cochran, A. Nowroz, and S. Reda. Post-Silicon Power Characterization Using Thermal Infrared Emissions. In *ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 331–336, 2010.

[22] COMSOL. http://www.comsol.com. In *Online*.

[23] W. J. Dally. It's About the Power: An Architect's View of Interconnect. In *Keynote IEEE International Interconnect Technology Conference (IITC)*, 2012.

[24] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[25] K. Dev, A. N. Nowroz, and S. Reda. Power Mapping and Modeling of Multi-Core Processors. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 39–44, 2013.

[26] K. Dev, I. Paul, and W. Huang. A Framework for Evaluating Promising Power Efficiency Techniques in Future GPUs for HPC. In *ACM High Performance Computing (HPC) Symposium*, 2016.

[27] K. Dev, S. Reda, I. Paul, W. Huang, and W. Burleson. Workload-Aware Power Gating Design and Run-time Management for Massively Parallel GPGPUs. In *IEEE Symposium on Very-Large Scale Integration*, 2016.

[28] K. Dev, G. Woods, and S. Reda. High-Throughput TSV Testing and Characterization for 3D Integration Using Thermal Mapping. In *Design Automation Conference (DAC)*, 2013.

[29] K. Dev, X. Zhan, and S. Reda. Online Characterization and Mapping of Workloads on CPU-GPU Processors. In *IEEE International Symposium on Workload Characterization*, submitted, 2016.

[30] G. F. Diamos and S. Yalamanchili. Harmony: An Execution Model and Runtime for Heterogeneous Many Core Systems. In *Proc. of the Intl. Symp. on High Performance Distributed Computing*, pages 197–200, 2008.

[31] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *International Symposium on Computer Architecture (ISCA)*, pages 78–88, 2006.

[32] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *International Symposium on Computer Architecture (ISCA)*, pages 365–376, 2011.

[33] S. Eyerman and L. Eeckhout. Fine-grained DVFS Using On-chip Regulators. *ACM Trans. Arch. Code Opt.*, 8(1):1–24, February 2011.

[34] S. Z. Gilani, N. S. Kim, and M. J. Schulte. Power-Efficient Computing for Compute-Intensive GPGPU Applications. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 330–341, Feb 2013.

[35] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos. Auto-Tuning a High-Level Language Targeted to GPU Codes. In *Innovative Parallel Computing (InPar)*, pages 1–10, May 2012.

[36] C. Gregg, M. Boyer, K. Hazelwood, and K. Skadron. Dynamic Heterogeneous Scheduling Decisions Using Historical Runtime Data. In *Proc. of the 2nd Workshop on Applications for Multi- and Many-Core Processors*, 2011.

[37] C. Gregg, J. S. Brantley, and K. Hazelwood. Contention-Aware Scheduling of Parallel Code for Heterogeneous Systems. In *HotPar'10*, 2010.

[38] P. Gupta, A. B. Kahng, and S. Muddu. Quantifying Error in Dynamic Power Estimation of CMOS Circuits. *Analog Integrated Circuits and Signal Processing*, 42(3):253–264, 2005.

[39] P. K. Gupta. Xeon+FPGA Platform for the Data Center. In *The Fourth Workshop on the Intersections of Computer Architecture and Reconfigurable Logic (CARL), Co-located with ISCA*, 2015.

[40] Laros III J. H., K. T. Pedretti, S. M. Kelly, W. Shu, and C. T. Vaughan. Energy Based Performance Tuning for Large Scale High Performance Computing Systems. In *Proceedings of the 2012 Symposium on High Performance Computing (HPC)*, pages 6:1–6:10, 2012.

[41] Wei H., C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware, and B. Brock. Accurate Fine-Grained Processor Power Proxies. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 224–234, Dec 2012.

[42] H. Hamann, A. Weger, J. Lacey, Z. Hu, and P. Bose. Hotspot-Limited Microprocessors: Direct Temperature and Power Distribution Measurements. *IEEE Journal of Solid-State Circuits*, 42(1):56–65, 2007.

[43] K. R. Heloue, N. Azizi, and F. N. Najm. Full-Chip Model for Leakage Current Estimation Considering Within-Die Correlation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(6):847–887, 2009.

[44] W. Huang, K. Skadron, S. Gurumurthi, R. J. Ribando, and M. R. Stan. Differentiating the Roles of IR Measurement and Simulation for Power and Temperature-Aware Design. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 1–10, 2009.

[45] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *International Symposium on MicroArchitecture*, pages 93–104, 2003.

[46] H. Jiang, M. Marek-Sadowska, and S. R. Nassif. Benefits and Costs of Power-Gating Technique. In *International Conference on Computer Design (ICCD)*, pages 559–566, Oct 2005.

[47] V. Jiménez, F. J. Cazorla, R. Gioiosa, M. Valero, C. Boneti, E. Kursun, C.-Y. Cher, C. Isci, A. Buyuktosunoglu, and P. Bose. Power and Thermal Characterization of POWER6 System. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 7–18, 2010.

[48] R. Joseph and M. Martonosi. Run-Time Power Estimation in High Performance Microprocessors. In *International Symposium on Low Power Electronics and Design*, pages 135–140, 2001.

[49] S. Kaxiras and M. Martonosi. Computer Architecture Techniques for Power-Efficiency. *Synthesis Lectures on Computer Architecture*, 3(1):1–207, 2008.

[50] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das. Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013.

[51] T. Kemper, Y. Zhang, Z. Bian, and A. Shakouri. Ultrafast Temperature Profile Calculation in IC Chips. In *THERMINIC*, pages 133–137, 2006.

[52] J. Kim, S. Seo, J. Lee, J. Nah, G. Jo, and J. Lee. SnuCL: An OpenCL Framework for Heterogeneous CPU/GPU Clusters. In *ACM International Conference on Supercomputing*, pages 341–352, 2012.

[53] S. Kosonocky. Practical Power Gating and Dynamic Voltage/Frequency Scaling. In *HotChips*, 2011.

[54] J. Lee and N. S. Kim. Optimizing Throughput of Power- and Thermal-constrained Multicore Processors Using DVFS and Per-core Power-gating. In *Design Automation Conference (DAC)*, pages 47–50, 2009.

[55] J. Lee, M. Samadi, Y. Park, and S. Mahlke. SKMD: Single Kernel on Multiple Devices for Transparent CPU-GPU Collaboration. *ACM Trans. Comput. Syst.*, 33(3):1–27, August 2015.

[56] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim. Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 111–120, Oct 2011.

[57] W. Lee, Y. Wang, T. Cui, S. Nazarian, and M. Pedram. Dynamic Thermal Management for FinFET-based Circuits Exploiting the Temperature Effect Inversion Phenomenon. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 105–110, 2014.

[58] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power Management of Datacenter Workloads Using Per-Core Power Gating. *IEEE Comp. Arch. Lett.*, 8(2):48–51, July 2009.

[59] J. Li and J. F. Martinez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. In *High-Performance Computer Architecture (HPCA), 2006. The Twelfth International Symposium on*, pages 77–87, Feb 2006.

[60] W. Liao, L. He, and K. M. Lepak. Temperature and Supply Voltage Aware Performance and Power Modeling at Microarchitecture Level. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 24(7):1042–1053, November 2006.

[61] M. Y. Lim, A. Porterfield, and R. Fowler. SoftPower: Fine-Grain Power Estimations Using Performance Counters. In *International Symposium on High Performance Distributed Computing*, pages 308–311, 2010.

[62] Y. Liu, R. P. Dick, L. Shang, and H. Yang. Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '07, pages 1526–1531, 2007.

[63] C.-K. Luk, S. Hong, and H. Kim. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. In *Proc. of the IEEE/ACM Intl. Symp. on Microarchitecture*, pages 45–55, 2009.

[64] N. Magen, A. Kolodny, U. Weiser, and N. Shamir. Interconnect-Power Dissipation in a Microprocessor. In *International Workshop on System Level Interconnect Prediction*, pages 7–13, 2004.

[65] A. Majumdar, G. Wu, K. Dev, J. L. Greathouse, I. Paul, W. Huang, A.-K. Venugopal, L. Piga, C. Freitag, and S. Puthoor. A Taxonomy of GPGPU Performance Scaling. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 118–119, 2015.

[66] F. J. Mesa-Martinez, E. Ardestani, and J. Renau. Characterizing Processor Thermal Behavior. In *Architectural Support for Programming Languages and Operating Systems*, pages 193–204, 2010.

[67] F. J. Mesa-Martinez, M. Brown, J. Nayfach-Battilana, and J. Renau. Power Model Validation Through Thermal Measurements. In *International Symposium on Computer Architecture*, pages 1–10, 2007.

[68] S. Mittal and J. S. Vetter. A Survey of CPU-GPU Heterogeneous Computing Techniques. *ACM Comp. Surv.*, 47(4):1–35, July 2015.

[69] G. E. Moore. Cramming More Components onto Integrated Circuits. In *Electronics*, pages 114–117, 1965.

[70] F. Najm. Power Estimation Techniques for Integrated Circuits. In *International Conference on Computer Aided Design*, pages 492–499, 1995.

[71] A. N. Nowroz and S. Reda. Thermal and Power Characterization of Field-Programmable Gate Arrays. In *Proc. ACM Intl. Symp. on Field Programmable Gate Array*, pages 111–114, 2011.

[72] S. Nussbaum. AMD "Trinity" APU. In *Hot Chips*, 2012.

[73] Online:. https://asc.llnl.gov/exascale/exascale-white.pdf.

[74] Online. https://www.amd.com/documents/powertune_whitepaper_web.pdf.

[75] Online. https://www.khronos.org/opencl/.

[76] Online. http://www.amd.com/documents/gcn_architecture_whitepaper.pdf.

[77] Online:. http://www.green500.org.

[78] Online:. http://www.mantevo.org/applications.php.

[79] Online. http://www.nvidia.com/object/cuda_home_new.html.

[80] Online. http://www.nvidia.com/object/tegra.html.

[81] Online:. http://www.top500.org.

[82] P. Pandit and R. Govindarajan. Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices. In *Proc. of Intl. Symp. on Code Generation and Optimization*, pages 273–283, 2014.

[83] I. Paul, W. Huang, M. Arora, and S. Yalamanchili. Harmonia: Balancing Compute and Memory Power in High-performance GPUs. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 54–65, 2015.

[84] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili. Cooperative Boosting: Needy Versus Greedy Power Management. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 285–296, 2013.

[85] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili. Coordinated Energy Management in Heterogeneous Processors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 59:1–59:12, 2013.

[86] J. Peddersen and S. Parameswaran. CLIPPER: Counter-based Low Impact Processor Power Estimation at Run-time. In *Asia and South Pacific Design Automation Conference*, pages 890–895, 2007.

[87] J. A. Pienaar, A. Raghunathan, and S. Chakradhar. MDR: Performance Model Driven Runtime for Heterogeneous Parallel Platforms. In *Proc. of the Intl. Conference on Supercomputing*, pages 225–234, 2011.

[88] M. Powell, A. Biswas, J. Emer, and S. Mukherjee. CAMP: A Technique to Estimate per-Structure Power at Run-Time Using a Few Simple Parameters. *International Symposium on High Performance Computer Architecture*, pages 289–300, 2009.

[89] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel. Improving Mobile Gaming Performance Through Cooperative CPU-GPU Thermal Management. In *Design Automation Conference (DAC)*, pages 47:1–47:6, 2016.

[90] A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra. Energy-Efficient Execution of Data-Parallel Applications on Heterogeneous Mobile Platforms. In *IEEE International Conference on Computer Design*, pages 208–215, Oct 2015.

[91] Z. Qi, B. H. Meyer, W. Huang, R. J. Ribando, K. Skadron, and M. R. Stan. Temperature-to-Power Mapping. In *International Conference on Computer Design*, pages 384–389, 2010.

[92] S. Reda and A. Nowroz. *Power Modeling and Characterization of Computing Devices: A Survey*. Foundations and Trends in Electronic Design Automation Series. Now Publishers, 2012.

[93] S. Reda, A. N. Nowroz, R. Cochran, and S. Angelevski. Post-silicon power mapping techniques for integrated circuits. *Integration, the VLSI Journal*, 46(1):69 – 79, 2013.

[94] P. Rogers. Heterogenous Systems Architecture Overview. In *Hot Chips*, 2013.

[95] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. De Supinski, and M. Schulz. Bounding Energy Consumption in Large-Scale MPI Programs. In *ACM/IEEE Conference on Supercomputing (SC)*, pages 1–9, Nov 2007.

[96] Y. Shabany. *Heat Transfer: Thermal Management of Electronics*. CRC Press, 2010.

[97] Y. Shin, J. Seomun, K.-M. Choi, and T. Sakurai. Power Gating: Circuits, Design Methodologies, and Best Practice for Standard-cell VLSI Designs. *ACM Trans. Des. Autom. Electron. Syst.*, 15(4):28:1–28:37, October 2010.

[98] K. Singh, M. Bhadauria, and S. A. McKee. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, 2009.

[99] Snapdragon. https://www.qualcomm.com/products/snapdragon. In *Online*.

[100] K. Spafford, J. Meredith, and J. Vetter. Maestro: Data Orchestration and Tuning for OpenCL Devices. In *Proc. of the International Euro-Par Conference on Parallel Processing: Part II*, pages 275–286, 2010.

[101] C. D. Spradling. SPEC CPU2006 Benchmark Tools. *SIGARCH Comput. Archit. News*, 35(1):130–134, March 2007.

[102] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. *UIUC, Tech. Rep. IMPACT-12-01*, March 2012.

[103] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang. PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 445–457, 2014.

[104] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full Chip Leakage Estimation Considering Power Supply and Temperature Variations. In *The International Symposium on Low Power Electronics and Design (ISLPED)*, pages 78–83, 2003.

[105] Synposys-PTPX. http://www.synopsys.com/. In *Online*.

[106] International Technology Roadmap for Semiconductors (ITRS). http://www.itrs.net/reports.html.

[107] A. Vilches, A. Navarro, R. Asenjo, F. Corbera, R. Gran, and M. J. Garzarn. Mapping Streaming Applications on Commodity Multi-CPU and GPU On-Chip Processors. *IEEE Trans. on Parallel and Distributed Systems*, 27(4):1099–1115, April 2016.

[108] H. Wang, V. Sathish, R. Singh, M. J. Schulte, and N. S. Kim. Workload and Power Budget Partitioning for Single-chip Heterogeneous Processors. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, pages 401–410, 2012.

[109] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng. Power Gating Strategies on GPUs. *ACM Trans. Archit. Code Optim.*, 8(3):13:1–13:25, October 2011.

[110] Y. Wen, Z. Wang, and M. F. P. O'Boyle. Smart Multi-Task Scheduling for OpenCL Programs on CPU/GPU Heterogeneous Platforms. In *International Conference on High Performance Computing (HiPC)*, pages 1–10, 2014.

[111] S. Williams, A. Waterman, and D. Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, April 2009.

[112] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou. GPGPU Performance and Power Estimation Using Machine Learning. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 564–576, Feb 2015.

[113] W. Wu, L. Jin, J. Yang, P. Liu, and S. X.-D. Tan. A Systematic Method for Functional Unit Power Estimation in Microprocessors. In *Design Automation Conference*, pages 554–557, 2006.

[114] Q. Xu and M. Annavaram. PATS: Pattern Aware Scheduling and Power Gating for GPGPUs. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 225–236, 2014.