MEPHESTO: Modeling Energy-Performance in Heterogeneous SoCs and Their Trade-Offs

Mohammad Alaul Haque Monil
University of Oregon
mmonil@cs.uoregon.edu

Mehmet E. Belviranli
Colorado School of Mines
belviranli@mines.edu

Seyong Lee
Oak Ridge National Laboratory
lees2@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory
vetter@computer.org

Allen D. Malony
University of Oregon
malony@cs.uoregon.edu

ABSTRACT

Integrated shared memory heterogeneous architectures are pervasive because they satisfy the diverse needs of mobile, autonomous, and edge computing platforms. Although specialized processing units (PUs) that share a unified system memory improve performance and energy efficiency by reducing data movement, they also increase contention for this memory since the PUs interact with each other. Prior work has investigated performance degradation due to memory contention, but few have studied the relationship of power and energy to memory contention. Moreover, a comprehensive solution that models memory contention for kernel placement on contemporary heterogeneous systems on chip (SoCs) in response to energy and performance has been largely unaddressed.

This paper presents MEPHESTO, a novel and holistic approach for managing this balance. The authors characterize applications and PUs in terms of two memory contention factors—time factors and power factors—to achieve the desired trade-off between energy and performance for collocated kernel execution on heterogeneous systems. The authors believe that this investigation is the first to combine all of these factors and present a simple knob-based approach that expresses the target trade-off. The approach is evaluated on a diverse integrated shared memory heterogeneous system with a CPU, GPU, and programmable vision accelerator. By using an empirical model for memory contention that provides up to 92% accuracy, the kernel collocation approach can provide a near-optimal ordering and placement based on the user-defined, energy-performance trade-off parameter. Moreover, the dynamic programming-based heuristics provide up to 30% better energy or 20% performance benefits when compared with the greedy approaches commonly employed by previous studies.

PACT '20, October 3–7, 2020, Virtual Event, GA, USA

ACM ISBN 978-1-4503-8075-1/20/10…$15.00
https://doi.org/10.1145/3410463.3414671

CCS CONCEPTS

• **Computer systems organization → Multicore architectures**; **Heterogeneous (hybrid) systems**; **System on a chip**.

KEYWORDS

Memory contention; Heterogeneous systems; Energy-performance trade-off; System on a Chip

ACM Reference Format:
Mohammad Alaul Haque Monil, Mehmet E. Belviranli, Seyong Lee, Jeffrey S. Vetter, and Allen D. Malony. 2020. MEPHESTO: Modeling Energy-Performance in Heterogeneous SoCs and Their Trade-Offs. In *Proceedings of the 2020 International Conference on Parallel Architectures and Compilation Techniques (PACT '20), October 3–7, 2020, Virtual Event, GA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3410463.3414671

1 INTRODUCTION

Heterogeneous systems are a popular solution for overcoming the temperature barrier when designing processing units (PUs) with high computation capabilities [28, 32]. Although powerful CPUs once dominated computing, GPUs and multi/many-core CPUs are now the go-to solution for high-performance systems [23]. Following this trend, special-purpose hardware for emerging domains—such as tensor PUs, bionic processors, and vision accelerators—have become a commodity in data centers, mobile devices, and autonomous platforms. Moreover, chip manufacturers are embedding a variety of PUs that serve different types of computing needs on a single die in the form of integrated shared memory heterogeneous systems (iSMHS) [37]. Although Intel's Ivy Bridge [11] and AMD's Fusion [3, 31] architectures were among the early systems that combined two compute capable PUs (i.e., CPUs and GPUs) under the same memory subsystem, later generations of integrated heterogeneous systems—such as NVIDIA's Tegra Xavier, Apple's A12 Bionic chip, and Qualcomm's Snapdragon 855 system on chip (SoC)—have brought the degree of heterogeneity within the same chip to extreme ends. In such systems, dozens of PUs with diverse instruction set architectures work together to accelerate kernels that belong to emerging application domains.

One of the most prominent features of iSMHS is that all PUs can directly access the system memory, alleviating additional data transfer costs between the CPU and device memory [7]. The optimal use of these systems heavily relies on collocating tasks simultaneously in different PUs while using the system memory as an

intermediate medium for inter-PU data communication. For example, an iSMHS for an autonomous car must simultaneously run image and video processing, inference for object detection, and other decision-making continuously in a pipeline-like execution scheme.

Collocated kernel execution on an iSMHS will likely cause contention on the shared memory bus, and the resulting interference could negatively affect perceived bandwidth (BW) on collocated kernels. Several studies [4, 7, 15, 37] focused on identifying the memory access patterns of collocated kernels in CPU+GPU iSMHS and suggested smart scheduling mechanisms to minimize contention effects, mostly via ad hoc approaches. However, these approaches do not provide a systematic solution for systems with different heterogeneity characteristics and an arbitrary number of PUs.

Apart from performance, memory contention also has a considerable impact on chip-level power consumption. Several studies [1, 6, 8, 13, 17, 20, 28] focused on iSMHS energy characteristics and relied on greedy algorithms or machine learning-based approaches to control the power consumption via dynamic voltage and frequency scaling (DVFS) and thermal design power (TDP) based power caps. However, the impact of contention on energy usage of PU and the chip is not addressed, and these studies do not build an analytical approach that establishes a direct relationship between energy and memory contention.

The ubiquitous deployment of iSMHS in environments in which the use-case priorities can dramatically vary makes the kernel collocation problem more challenging. Since the objectives are dynamic and constrained by the throughput and power budget needs, meeting them is crucial for optimizing overall utilization of iSMHS. For example, in an autonomous driving scenario, the system software should collocate kernels in the most performance-maximized manner when approaching an intersection with multiple objects to track. On the other hand, while cruising on a highway at a stable speed for which processing throughput needs are lower, kernels can be scheduled to minimize total energy usage. Ideally, such an objective-aware kernel collocation could be achieved with a simple parameter that controls the energy-performance trade-off (EPTO).

This paper presents MEPHESTO, which proposes a holistic approach for controlling the EPTO in the collocated kernel execution on iSMHS. A per-PU kernel operational intensity [12, 33] was used to approximate the effects of contention on performance and energy along with the kernel collocation algorithm to intelligently find near-optimal collocation that satisfies the provided EPTO objective. The authors believe that this is the first effort to propose a generic formulation for memory contention in iSMHS that considers factors that directly correlate to energy and performance during kernel collocation.

The paper makes the following contributions:

- Integrated performance and energy behavior representation are introduced based on time factors and power factors, which are nonlinear functions of the ratios between standalone and collocated execution measurements of a PU in a given iSMHS.
- A novel empirical model was built to estimate the energy and performance of a set of collocated kernels on an arbitrary number of PUs while considering the variation caused by memory contention.

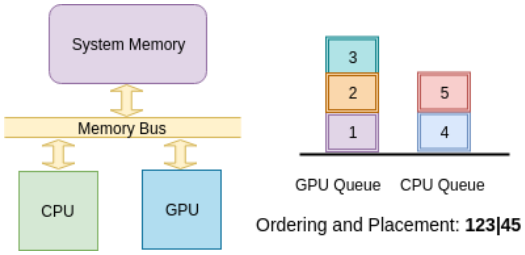


Figure 1: A logical representation of iSMHS with CPU and GPU and kernel queues for O&P: *123|45*.

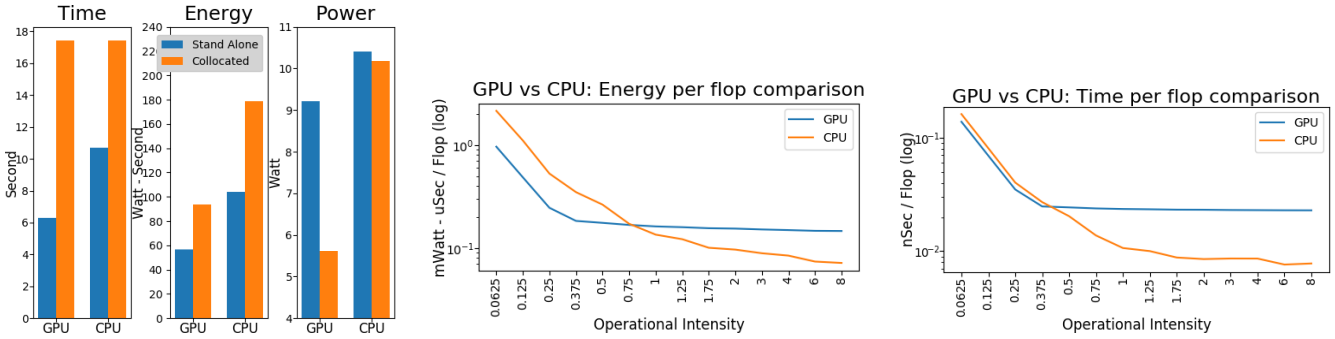
- A collocation algorithm was designed that takes the target EPTO as a user-defined input parameter and employs a novel heuristic to reach a near-optimal ordering and placement (O&P) of a given set of kernels on a target set of PUs.
- The feasibility of MEPHESTO was empirically evaluated by collocating a collection of scientific kernels across three heterogeneous PUs of NVIDIA's Tegra Xavier platform: CPU, GPU, and programmable vision accelerator (PVA). The proposed scheduling algorithm was demonstrated to be able to find near-optimal O&P with a reasonable (on an average 10%) modeling error rate. It also provides up to 30% improvement over a greedy approach.

2 UNDERSTANDING THE EFFECTS OF COLLOCATED EXECUTION ON ISMHS

This section explores the energy and performance implications of collocated kernel execution on an iSMHS similar to that portrayed in Fig. 1. In this example, the CPU and GPU are connected to a shared memory, and the GPU does not have a private memory. There could also be other heterogeneous PUs, such as PVAs and deep learning accelerators (DLAs), connected to the same memory subsystem; however, they are excluded in this specific case for simplicity. In this system, the collocation of five ready-to-execute kernels with the O&P configuration of *123|45*, in which kernels 1, 2, and 3 will be orderly executed on the GPU and kernels 4 and 5 will be placed for CPU execution. Kernels placed on different queues could execute in a collocated manner and result in contention on the memory bus.

2.1 Contention vs. Energy and Performance

To observe the effects of the collocated execution of two maximum memory BW-demanding kernels on performance and energy, the authors ran the STREAM benchmark [21] on the ARM Carmel CPU and Volta GPU of NVIDIA's Xavier platform concurrently. The amount of data that the CPU and GPU were able to process during execution was recorded, and the total execution time and power consumed by the CPU and GPU were measured at 50 ms intervals. The runs are repeated for the same amount of data in standalone mode, and the execution time, total energy consumption, and average power consumption are reported in Fig. 2a. Significant increases in execution time and energy consumption were observed for both kernels while running simultaneously on the CPU and GPU. On the other hand, average power consumption decreased for collocation, particularly from the GPU perspective. Although the authors ensured that CPU and GPU utilization was



(a) Effect of memory contention on time, energy, and power for collocated/standalone execution of the STREAM benchmark on CPU and GPU.

(b) Per-flop energy consumption when kernels with different operational intensities are run in a standalone mode.

(c) Per-flop execution time when kernels with different operational intensities are run in a standalone mode.

Figure 2: Energy and performance behavior under different scenarios.

100% all the time and that the system did not throttle CPU, GPU, or memory frequencies, the BW utilization dropped to almost half of the standalone value during collocated execution. This observation resulted in the realization that the impact of contention on time and energy is different for CPUs and GPUs and further motivated the establishment of a PU-centric contention model for energy and performance (i.e., execution time).

2.2 Need for Characterizing Kernels and PUs

Since contention depends on the amount of memory access requests generated by the kernel, the kernels running on each PU must be characterized in terms of their memory access requests to identify the level of contention they might incur. Also, contention only occurs when the processor cannot find the requested data in the cache and data must be brought from the system memory. For this reason, this work identified kernels based on their frequency of system memory accesses by using an operational intensity metric, which is the ratio of the total number of flops and total bytes read/write (R/W) between last-level cache (LLC) and system memory, as defined by the Roofline model [33]. The lesser the operational intensity, the more likely it is to have contention. Moreover, contention also depends on the physical BW of the system and the amount of cache attached to the processors. For this reason, PU behavior must also be characterized by cross-running different kernels with different operational intensities in both PUs simultaneously to observe the level of contention.

2.2.1 Justification for Using Operational Intensity. Operational intensity depends on traffic between LLC and memory, which represents the common contention point for different PUs in iSMHS. Other factors affect the performance of collocation, such as the memory access patterns of the kernels and caching optimizations, but operational intensity can indirectly include those effects since traffic between LLC and memory reflects those effects. In this way, operational intensity provides a simplified and unified metric that presents a fair trade-off between modeling complexity and prediction accuracy.

2.3 Ordering and Placement

Figure 2a shows that collocating kernels can significantly impact energy and performance. As a result, kernel ordering is important for energy and performance. The example given in Fig. 1 implies that kernel 1 will be collocated with kernel 4 and that kernel 2 will be collocated with kernel 5. This work strives to determine the best O&P for a given set of kernels. One systematic way to approach this problem is to start by running synthetic kernels standalone on the CPU and GPU with different operational intensities to observe how much time and energy is spent for each flop. The Empirical Roofline Toolkit [19] was modified to produce various intensities, and the energy and performance behaviors are shown in Figs. 2b and 2c, respectively. The results show that with greater operational intensity, the CPU becomes more favorable than the GPU since it spends less energy and time per flop.

Another observation at the intersection point of the CPU and GPU curve is that if the operational intensity that identifies a specific kernel falls to the left of the intersection, it is more suited for GPU, and vice versa. Thus, if the operational intensity of a kernel being run is known, an accurate placement decision can be made as to where the kernel should run. However, the intersection points might be different for energy and performance behavior, making the placement decision more complicated. More importantly, Figs. 2b and 2c do not consider collocation or contention. Therefore, a combined solution is needed that will incorporate the

Table 1: Roofline kernels used to understand the outcomes of different O&P variations.

Kernel name	Flop per array index	DRAM R/W byte per array index	Operational intensity	Total flop	GPU (standalone)		CPU (standalone)	
					Flop/s	*Avg. power (Watt)*	*Flop/s*	*Avg. power (Watt)*
1	1	16	0.0625	273.8 G	7.2 G	6.9	6.2 G	13.3
2	6	16	0.375	273.8 G	40.3 G	7.3	36.9 G	12.8
3	12	16	0.75	273.8 G	41. 9G	7.0	72.6 G	12.4
4	20	16	1.25	273.8 G	42.7 G	6.8	99.8 G	12.1
5	48	16	3.0	273.8 G	43.3 G	6.5	115.9 G	10.3

outcomes of Figs. 2a, 2b, and 2c together. The confluence of O&P for a given set of kernels must be considered.

2.4 Kernel Collocation for Varying Energy and Performance

To demonstrate how various O&P configurations affect energy consumption and performance differently, the Empirical Roofline Toolkit was used to generate five kernels, whose details are given in Table 1, with varying operational intensity ratios. These kernels operate in a read-compute-write fashion. The first 8 bytes of data (i.e., double-precision floating point) are read from an array, a series of additions and multiplications is performed, and the data are written back to the same memory location. In this way, data are ensured to be fully read from and written back to the DRAM. The second and third columns of Table 1 show the number of floating-point operations and the total amount of bytes R/W from DRAM for every array index the kernels process, respectively. Based on these values, the operational intensities per kernel were calculated. Through profiling, flops per second and the average power data for each kernel in the standalone mode were generated for CPU and GPU.

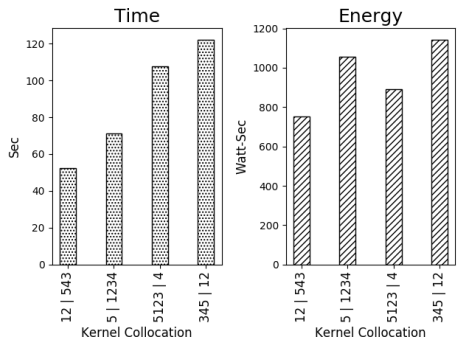


Figure 3: Different total energy and performance behaviors for various collocation combinations for the same kernels.

Four different O&P configurations were used, and the execution time and energy consumption are shown in Fig. 3. The results show that different O&P configurations lead to different time/energy profiles. Although the common strategy in the related literature is to collocate compute-intensive kernels with memory-intensive kernels to improve overall performance, the optimal O&P strategy might vary when a trade-off is being sought between energy and performance. This situation raises a few questions: How can the cut-off point for the compute-memory intensity be defined? What happens if the number of memory-intensive kernels is more than the compute-intensive kernels, or vice versa? How can trade-off control be established between energy and performance? As shown in the following sections, greedy algorithms commonly employed by the related studies are not sufficient to address all these considerations.

This research revolves around addressing the four motivations presented in this section. The remainder of the paper presents an empirical model for defining memory contention by considering the kernel and processors, defines optimal O&P, and presents the kernel collocation algorithm for obtaining a desired EPTO.

3 AN EMPIRICAL MODEL FOR MEMORY CONTENTION

This section presents a core component of MEPHESTO, an empirical model that characterizes kernels and PUs with respect to memory contention. The impact of memory contention is defined in terms of energy consumption and performance (i.e., execution time). For a given O&P of kernels, this model predicts the execution time and energy consumption.

Table 2: The notations used by the model.

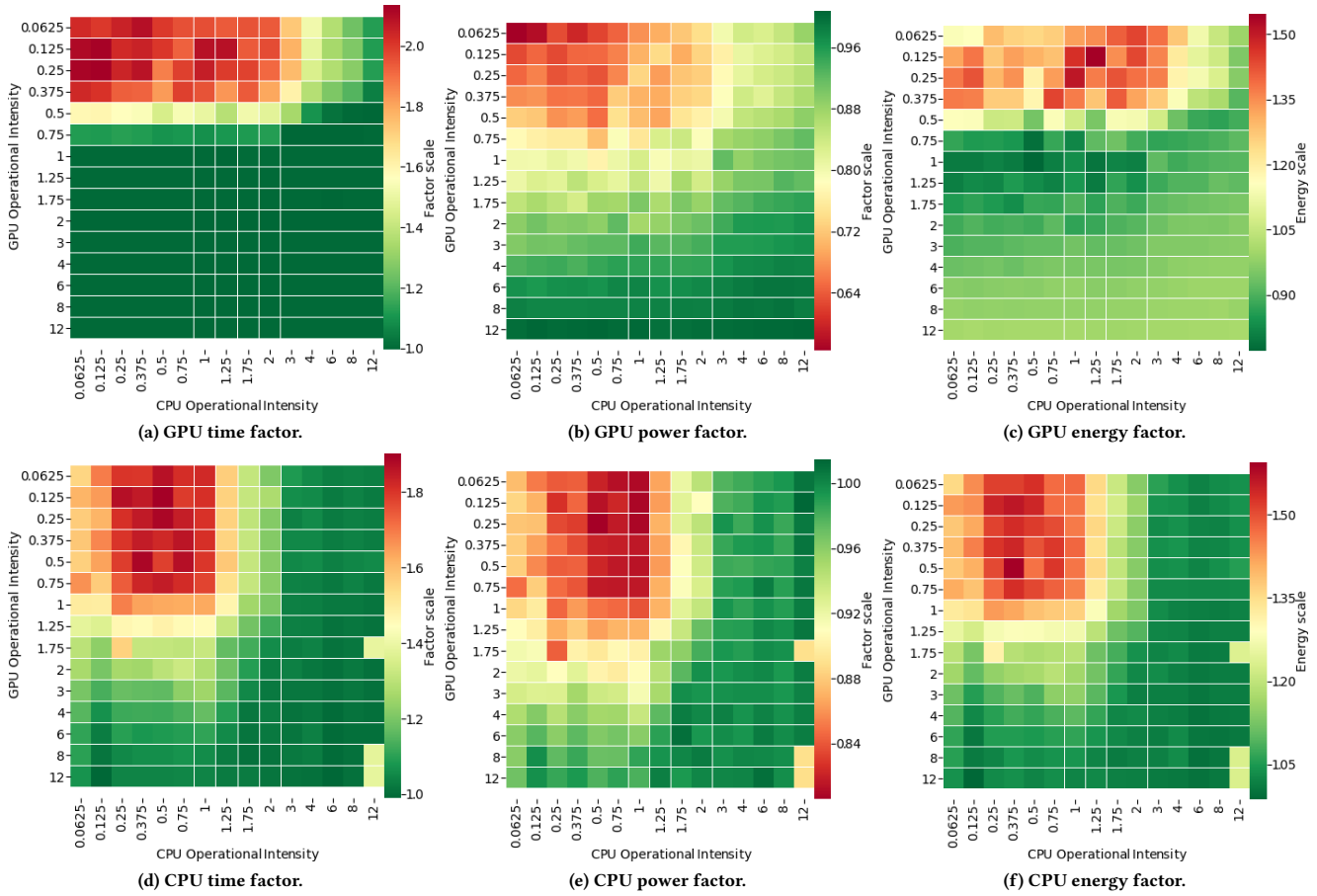
Notation	Explanation
K_i	ith kernel from n kernels $K = \{K_1, K_2, ..., K_n\}$
P_j	jth PU from m PUs $P = \{P_1, P_2, ..., P_m\}$
F_{K_i}	Number of flops in kernel K_i
$T_{P_j}^{K_i}$	Standalone execution time of K_i on PU P_j
$Pw_{P_j}^{K_i}$	Standalone power of PU P_j for K_i
OI_{P_j}	Operational intensity of the kernel on P_j
TF_{P_j}	Time factor of P_j
PF_{P_j}	Power factor of P_j
$TC_{P_j}^{K_i}$	Collocated execution time of K_i on PU P_j
$E_{P_j}^{K_i}$	Energy consumption of K_i on PU P_j
C	Total number of possible O&P
ρ	PU wise queues (i.e., O&P)
τ_c	Execution time of current O&P
ε_c	Energy consumption of current O&P
$S(V)$	Resultant kernel collocation of V kernels
ω	Weight for a given O&P

3.1 Definitions

All symbols and terms used in this model are introduced in this section and are also presented in Table 2. An *O&P* is a set of n kernels $K = \{K_1, K_2, ..., K_n\}$ in which a kernel is an uninterrupted computation that is ready to be executed on any of the available PUs in the system. The *kernel collocator* finds the placement of every K_i on a set of m heterogeneous PUs, which is represented by $P = \{P_1, P_2, ..., P_m\}$.

$$K_i = [F_{K_i}, OI_{K_i}, \forall_{P_j \in P}\{T_{P_j}^{K_i}, Pw_{P_j}^{K_i}\}]. \quad (1)$$

A kernel K_i is represented by using three terms, as shown in Eq. (1). The first term, F_{K_i}, is the number of floating point operations of that kernel. The second term, OI_{K_i}, is the operational intensity. When a kernel, K_i, is placed on a processor, P_j, the standalone execution time and average power consumption are represented by $T_{P_j}^{K_i}$ and $Pw_{P_j}^{K_i}$, respectively. So, the third term represents the pair of standalone execution time and average power consumption for each PU. The objective is to determine these values at compile time and also by partially profiling the kernels at run time. However, when collocated, each K_i exhibits different slowdown in execution time and consumes different average power based on their compute and memory intensity. As suggested previously, in Fig. 2, there is a factor for execution time, TF_{P_j}, and a factor for average power, PF_{P_j}, for a given PU P_j. These factors are the ratio of their collocated to standalone values. Thus, PU is represented as $P_j = [TF_{P_j}, PF_{P_j}]$.



(a) GPU time factor. (b) GPU power factor. (c) GPU energy factor.

(d) CPU time factor. (e) CPU power factor. (f) CPU energy factor.

Figure 4: Collocation factors: collocated/standalone values of time, power, and energy for different operational intensities.

3.2 Characterization of Memory Contention

To define and characterize memory contention, three factors must be determined: (1) how kernels, K, are characterized; (2) how PUs, P, are characterized, and (3) how time factor, TF_{P_j}, and power factor, PF_{P_j}, are formulated. To determine the first factor, operational intensity must be considered as a measure to characterize a kernel's compute or memory intensity. The Empirical Roofline Toolkit [19, 33] was modified to generate kernels with different operational intensities, and their execution was observed. Roofline kernels are designed in a read-compute-write fashion. For this reason, a fixed number of bytes are exchanged between the cache and system memory; as a result, the operational intensities of those kernels do not vary across different PUs with different cache hierarchy. To determine the second factor, a range of kernels was collocated with different operational intensities in PUs of iSMHS. A kernel was kept running in one processor, and the impact on another was observed. While doing this, execution time and power consumption were recorded. From these values, energy consumption was calculated. To capture the impact in a structured way, a ratio of collocated to standalone time and power was made, which are termed *time factor* and *power factor*, respectively. This addresses the third concern.

This PU characterization is a one-time effort for any new system. In this case, the authors were interested in NVIDIA Xavier's CPU and GPU. Figure 4 plots the results. In these heat maps, the x-axis is the operational intensity of kernels that are running on the CPU, and the y-axis is for the GPU. Time and power are captured for both the CPU and GPU, from which energy consumption was calculated. When operational intensity is low (i.e., high memory intensity), the impact of contention is visible in both the CPU and GPU. The time factor goes up to $2.2x$ for the GPU (Fig. 4a) and $1.8x$ for the CPU when contention is present (Fig. 4d). Similar behavior is observed for energy consumption (Fig. 4c). The opposite is observed for the power factor; $0.6x$ is observed for the GPU (Fig. 4b), and $0.8x$ is observed for the CPU (Fig. 4b). Interestingly, after a certain operational intensity occurs, the impact of contention vanishes because the available system BW is enough for the request, and there is no bus contention.

After capturing the impact of different kernels, the time factor, TF_{P_j}, and power factor, PF_{P_j}, can be represented as a function of the operational intensity of the kernel of the current PU, OI_{P_j}, and collocated PU, $OI_{P_{col}}$, as in Eq. (2). A fifth order multivariate polynomial regression curve was generated by using the CxxPolyFit [16] tool that supports up to the ninth order for TF_{P_j}, and PF_{P_j}, in which

the independent variables are OIp_j and OIP_{col}. Using a lower order provides faster evaluation with low accuracy, whereas higher order (e.g., fifth order or higher) provides more accuracy but takes more time to evaluate. Fifth order was used due to a reasonable balance between the evaluation time and accuracy, as recommended by CxxPolyFit [16].

$$TF_{P_j} or PF_{P_j} = f(OI_{P_j}, OI_{P_{col}}). \quad (2)$$

3.3 Collocation Estimator Algorithm

The objective of the collocation estimator algorithm is to take an O&P of variable length kernels and predict the execution time and energy consumption while considering memory contention. To calculate the total execution time, τ_c, for the current O&P, c, the collocated execution time must be estimated from time factor and standalone execution time. For the total energy consumption, ε_c, collocated average power must be estimated from collocated power factor and standalone average power. Moreover, the length (i.e., the time span of execution) of a kernel must be considered since one kernel can be collocated with multiple shorter kernels during its lifetime.

The collocation estimator algorithm given in Algorithm 1 was designed by considering the aforementioned objectives. This algorithm takes an O&P of kernels, c, scheduled on PUs, P. Every PU has its own queue $\rho = \{\rho_1, \rho_2, ..., \rho_m\}$ in which kernels are stored in such an order so that the kernel in the head of the queue will be scheduled first to the corresponding PU. The algorithm determines the overall execution time, τ_c, and total energy, ε_c, incrementally. In the beginning, at [Line 3], τ_c, and ε_c are initialized. Then, at [Line 4], a loop is started, which continues until all queues are empty or all kernels are scheduled. In [Lines 5–7], the queue item at the head is popped from each ρ_j and stored as K_{P_j}. At [Lines 9–10], the time factor, TF_{P_j}, and power factor, PF_{P_j}, are calculated by using Eq. (2) for all nonempty K_{P_j}. Collocated time, TC_{P_j}, is then calculated by multiplying the standalone execution time and collocation time factor at [Line 11]. At [Line 13], the processor with the smallest kernel, P_{min}, is determined, and at [Line 14], the minimum time is added to the total time, τ_c. The minimum execution time was taken because the other kernels in other processors will now have a different kernel as collocated since the minimum one has finished its execution. At [Line 15], energy is calculated by considering the minimum time and collocated average power. Collocated average power is determined by multiplying standalone average power, $Pw_{P_j}^{K_i}$, with the power factor, PF_{P_j}. Since one kernel has finished its execution, the remaining part of the longer kernels must be calculated. For this reason, at [Lines 16–18], the remaining part of collocated time, TC_{P_j}, is factored back to standalone time, $TCremaining_{P_j}$, and pushed to the corresponding queue, ρ_j. These leftover kernels are considered to be just like a new kernel in the next iterations, and this occurs until every queue is empty. The total execution time, τ_c, and energy, ε_c, are determined when every queue is empty. Algorithm 1 finds τ_c and ε_c in *O(nm)* time, where n is the number kernels, and m is the number of processors.

Algorithm 1 Collocation Estimator

1: **Input:** PU wise kernel queue $\rho = \{\rho_1, \rho_2 ... \rho_m\}$
2: **Output:** Execution time, τ_c, and energy consumption, ε_c
3: Initialize: $\tau_c \leftarrow 0 \& \varepsilon_c \leftarrow 0$
4: **while** $\exists_{\rho_j \epsilon \rho} Size(\rho_j) > 0$ **do**
5: **for** each $\rho_j \epsilon \rho$ where $Size(\rho_j) > 0$ **do**
6: $K_{P_j} = \rho_j.POP()$
7: **end for**
8: **for** each $K_{P_j} \neq NULL$ **do**
9: $TF_{P_j} = time\_factor(OI_{P_j}, OI_{P_{col}})$
10: $PF_{P_j} = power\_factor(OI_{P_j}, OI_{P_{col}})$
11: $TC_{P_j} = T_{P_j}^{K_i} * TF_{P_j}$
12: **end for**
13: $P_{min} = \min_{\forall P_j}[TC_{P_j}]$
14: τ_c += $TC_{P_{min}}$
15: ε_c += $\Sigma_{\forall P_j}[Pw_{P_j}^{K_i} * PF_{P_j} * TC_{P_{min}}]$
16: **for** each $TC_{P_j} > TC_{min}$ **do**
17: $TCremaining_{P_j} = [TC_{P_j} - TC_{min}]/TF_{P_j}$
18: $\rho_j.PUSH(TCremaining_{P_j})$
19: **end for**
20: **end while**

4 DEFINING OPTIMAL ORDERING AND PLACEMENT

This section discusses the cost function design to define an optimal O&P based on a given trade-off target. For this reason, all possible combinations of O&P must be considered for all kernels, $K = \{K_1, K_2, ..., K_n\}$ in all PUs, $P = \{P_1, P_2, ..., P_m\}$. Let C represent all the possible ways that n kernels can be ordered and placed on m processors. The execution time of all possible O&P is denoted as $\tau = \{\tau_1, \tau_2, ..., \tau_c\}$, and energy is denoted as $\varepsilon = \{\varepsilon_1, \varepsilon_2, ..., \varepsilon_c\}$, where minimum and maximum execution times are represented by τ_{min} and τ_{max}, respectively. The minimum and the maximum energy consumption for all O&P are represented by ε_{min} and ε_{max}. For example, for five jobs in two processors, there are 482 possible O&P and thus 482 pairs of energy and time. Since there are two parameters—energy and performance—a reference point is needed to define the optimal O&P. This reference point is called the *EPTO parameter*.

The EPTO parameter is represented as a pair of energy performance in the following format—*(performance, energy)*—where the value of performance or energy can be 0–100. If EPTO is set to (0,100), then minimizing execution time is given the highest priority. If EPTO is set to (100,0), then minimizing energy consumption is given the highest priority. If EPTO is set to (30,70), then 70% priority is given to minimize execution time and 30% priority is given to minimize energy consumption.

To achieve this functionality, energy and time pairs of every O&P were converted to a range from 0 to 100. Then, every O&P becomes a point at which energy and time can vary from 0 to 100, where 0 represents the minimum time or energy and 100 represents the maximum. In this way, the energy-time pair of all O&P can be plotted in a 100×100 plot in which EPTO also becomes another point. Then, the distance from EPTO to every O&P is measured. The lower the distance, the higher the weight assigned, and at the end, the O&P with the highest weight is selected. This is achieved by a cost function expressed in Eq. (3). Based on the value of EPTO, τ_c, and ε_c, the weight of every O&P is calculated. A set of C weight is expressed as $\omega = \{\omega_1, \omega_2, ..., \omega_c\}$. The O&P of kernels, $S(J)$, were

selected where the weight is the highest, and this is the optimal O&P for the kernels and defined EPTO.

$$\omega_c = 1/dist\left[\left\{\frac{\tau_c - \tau_{min}}{\tau_{max} - \tau_{min}} * 100, \frac{\varepsilon_c - \varepsilon_{min}}{\varepsilon_{max} - \varepsilon_{min}} * 100\right\}, \{EPTO\}\right]. \tag{3}$$

5 KERNEL COLLOCATION STRATEGY

This section formulates a heuristics based on dynamic programming (DP) for MEPHESTO. By using the DP-based heuristics, a kernel collocation algorithm is designed, followed by a discussion of the complexity analysis of the approach.

5.1 Dynamic Programming Formulation

An exhaustive search throughout all the kernel O&Ps guarantees an optimal solution but is computationally expensive. For this reason, a DP approach was formulated to reach a near-optimal solution and reduce the complexity [2]. The solution is built from a smaller set and recursively builds the bigger ones by selecting maximum weighted subsolutions. At each step, a new placement for one kernel, K_j, is found, which maximizes the weight for the current O&P. V is considered a varying set of kernels, where $V \subseteq K$. Kernel O&P is represented as *S(V)*, which provides the processor wise queue information $\rho = \{\rho_1, \rho_2, ..., \rho_m\}$. In Eq. (4), *S(V)* is built recursively.

$$\begin{cases} S(\{K_1, K_2\}) = Max[Collocate(\{\emptyset\}, \{K_1, K_2\}, \rho_j)], & \text{if } |V| = 2 \\ S(V) \quad\quad = Max[Collocate(S(V - \{K_i\}), \{K_i\}, \rho_j)], & \text{if } |V| > 2 \end{cases} \tag{4}$$

Here, the function $Collocate(S(V), K_i, \rho_j)$ represents the collocation estimator algorithm given at Algorithm 1. As the collocation estimator algorithm takes a specific O&P as an input, the parameters constitute the processor wise queue ρ (i.e., the O&P). The parameters are given as follows:

$S(V)$ is the current O&P.
K_i is a new kernel that will be added to $S(V)$.
ρ_j is the processor queue into which K_i will be added.

The *Collocate* function provides the execution time and energy consumption for that O&P. The *MAX* operation then considers the placement of K_i in all processors, P, and selects the placement where the weight is the maximum based on the cost function and EPTO. The base case of Eq. (4) determines the placement with maximum weight for two kernels since it takes (at least) two kernels to collocate. For example, there are four O&Ps in a scenario with two kernels and two PUs. The base case determines the best O&P from these four O&Ps by using the cost function. For a scenario with three kernels and two PUs, the second case of Eq. (4) is used. In this case, one kernel is separated and placed in all the PUs along with the best O&P of the remaining two kernels, which are derived from the base case. Again, the cost function and EPTO are applied to determine the best O&P of the three kernels. In this way, the total set gets bigger by applying the cost function while DP eliminates unnecessary combinations.

Algorithm 2 DP-Based Kernel Collocation

```
1: Input: n Kernels, K = {K_1, K_2, ..., K_n},
          m Processors, P = {P_1, P_2, ..., P_m}, and EPTO
2: Output: O&P with MAX weight, S(V).
3: for i = 2 to n do
4:     if i == 2 then
5:         Calculate all possible base cases.
6:         continue
7:     end if
8:     for each V ∈ K where |V| = i do
9:         S(V) ← {∅}
10:        for each K_i ∈ V do
11:            Max_weight ← 0
12:            V_partial = V − {K_i}
13:            for each P_j ∈ P do
14:                S(V_Pj) = Collocate(S(V_partial), {K_i}, ρ_j)]
15:                Get time τ_c and energy ε_c
16:                Update τ_max, τ_min, ε_max, ε_min
17:            end for
18:            for each P_j ∈ P do
19:                ω(V_Pj) = Calculate_weight(τ_c, ε_c, EPTO)
20:                if ω(V_Pj) > Max_weight then
21:                    Set Max_weight = ω(V_Pj)
22:                    Update S(V) = S(V_Pj)
23:                end if
24:            end for
25:        end for
26:    end for
27: end for
```

5.2 DP-Based Kernel Collocation Algorithm

The objective of the DP-based kernel collocation algorithm is to determine a near-optimal O&P based on a defined EPTO. Algorithm 2 provides a simplistic pseudo-code that implements Eq. (4). This algorithm takes three inputs: (1) list of kernels, (2) list of PUs, and (3) EPTO. The output of the algorithm is the near-optimal O&P, which is denoted as $S(V)$. This starts with finding the O&P for minimal subset V (i.e., the base case where $|V| = 2$) by finding the maximum weight based on EPTO and the cost function. The algorithm then increases the size of V by one new kernel while reusing the saved maximum weighted (i.e., best) O&Ps from past iterations. The algorithm finds the best O&P, which is processor wise queues, $\rho_{max\,weight} = \{\rho_1, \rho_2, ..., \rho_m\}$. At [Lines 1–2], input and output are defined. At [Line 3], the algorithm iterates through the smallest ($|V| = 2$) to the largest subset size $|V| = n$. In [Lines 4–7], the base case of Eq. (4) is calculated by considering two kernels and m PUs. In [Line 8], the algorithm iterates over every possible subset V of K, where $|V| = i$. [Line 9] initializes $S(V)$. In [Line 10], every K_i is considered from the current set V. In [Line 12], K_i is separated, and a partial set $V\_partial$ is formed. The best O&P for this partial set is already calculated in the previous iteration. At [Line 13], every processor, P_j, is considered for a potential placement for kernel K_i. At [Line 14], K_i is added to the O&P of $S(V\_partial)$, *Collocation Estimator* algorithm (i.e., Algorithm 1) is called, and the result is stored at $S(V_{P_j})$. In [Line 15], execution time, τ_c, and energy consumption, ε_c, are updated, which are the output of Algorithm 1. In [Line 16], τ_{min}, τ_{max}, ε_{min}, and ε_{max} are updated. In this way, time and energy are calculated for all the possible subsets that are built on top of the best O&P of previous iterations. Now, there is a set

of execution time and energy consumption. At [Line 19], the cost function in Eq. (3) is invoked by using the $Calculate\_weight()$ function that uses energy, time, EPTO, and all the minimum-maximum values. In [Lines 21–22], the algorithm checks whether the current O&P provides maximum weight. If it does, Max_weight and $S(V)$ are updated. When the algorithm finishes its iterations, $S(V)$ contains the desired near-optimal O&P for the given inputs.

5.3 Complexity

As mentioned previously, Algorithm 1 has a complexity of $O(nm)$. The outer loop of Algorithm 2 at [Line 3] is iterated $(n-1)$ times, and the selection of subsets V with size i results in the loop at [Line 8] and the innermost loop to be iterated $\sum_{i=2}^{n} \binom{n}{i}$ and $\sum_{i=2}^{n} i\binom{n}{i}$ times, respectively, resulting in a complexity of $O(n^2 2^{n-1} m^2)$. However, a brute force search over all the combinations will reach a complexity of $O(nm^2 n!)$. The DP solution is faster but might not always yield the optimal O&P. The performance of the DP-based strategy is evaluated in the next section.

6 EXPERIMENTAL SETUP

These experiments were conducted on NVIDIA's Tegra Xavier SoC development platform. For parallel kernel execution on the CPU, the OpenMP programming model was used. All the CPU executions refer to the multithreaded execution that uses all the available cores. For GPU and PVA execution, the CUDA and OpenCV programming models are used, respectively. The Xavier platform gives users the ability to measure power consumption for the CPU, GPU, and PVA separately. The *tegra_parser* tool [24] was used to measure PU-wise power consumption. To measure the number of flops and memory R/W bytes from the LLC to the system memory (i.e., operational intensities) for the kernels, NVIDIA's proprietary profiling tool *nvprof* was used. Since the ARM Carmel CPU of Xavier does not yet have the required counters to calculate operational intensity, the values reported by *nvprof* were used. This approach led to a reasonable approximation for the CPU execution, which is further explained in Section 7.2.

For GPU and CPU characterization and execution, scientific kernels from the Rodinia benchmark suite [5] and synthetic kernels from the Roofline toolkit [33] were used to demonstrate the effectiveness of MEPHESTO. For PVA, applications from the OpenCV [29] benchmark suite, which is bundled with NVIDIA's Vision Programming Interface (VPI) software development kit [25], were used. The data corresponding to the characteristics of these kernels is presented in Tables 3 and 4. While experimenting, no power cap was set in the device. Power caps in Xavier limit the maximum frequency in PUs and thus change the kernel behavior; DVFS picks frequency from a defined set of frequency for a power cap. Changing power caps requires the kernels in the affected PUs to be reprofiled to generate data for Tables 3 and 4. By using these kernels, the proposed DP-based scheduling was compared with three other scheduling approaches: (1) optimal scheduling for a specific EPTO, (2) random scheduling, and (3) a greedy scheduling approach commonly used by the related work [37].

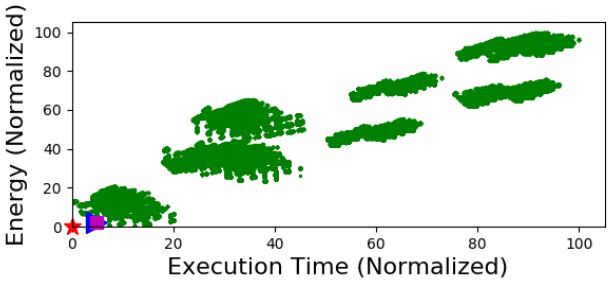


Figure 5: Result for benchmark kernels for EPTO (0, 0): near-optimal O&P selection for all combinations. The red star is EPTO, the blue triangle is the optimal solution, and the magenta square is the near-optimal solution by DP. The absolute minimum-maximum value pairs for energy and time are 0.72–3.3 kWatt-seconds and 45.5–255.5 seconds.

7 EXPERIMENTAL RESULTS

This section evaluates the efficacy of MEPHESTO in six steps: (1) the model and algorithms are shown to result in a near-optimal solution in a CPU+GPU collocated execution scenario, (2) the prediction accuracy of the model is evaluated, (3) different EPTO goals are evaluated, (4) a comparison with a greedy algorithm is presented, (5) the experiments are extended to include concurrent execution on CPU+GPU+PVA, and (6) the overhead associated with MEPHESTO is discussed.

7.1 Kernel Collocation Using Scientific Kernels

The first experiment was a high-level feasibility study to demonstrate how the proposed collocation technique can find a near-optimal solution among hundreds of thousands of possible O&P combinations. In this experiment, the EPTO goal is set as (0,0), which indicates that energy and performance should both be optimized. Although this is an unrealistic goal for the targeted platforms, this experiment is used to argue why the EPTO should be treated as a trade-off knob rather than an "optimize-all" target between energy and performance with more realistic goals such as (100,0), (0,100), and (50,50).

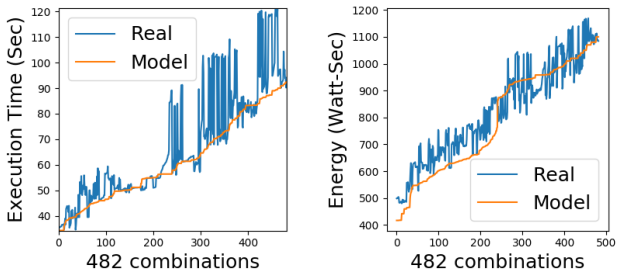
The proposed algorithm was evaluated by using the eight benchmark kernels listed in Table 3. The LLC-to-DRAM R/W bytes and the number of flops reported in the table are generated by using the counters provided by NVIDIA. The execution time and average power data for each kernel are collected in standalone mode for the CPU and GPU. There are 282, 241 possible ways in which eight kernels can be collocated (i.e., 282, 241 O&Ps on two PUs). To apply the EPTO of (0,0), the energy consumption and execution time of each O&P were normalized to a value between 0 and 100. Then, all the possible O&Ps were plotted in Fig. 5. The green circles represent all the possible O&P (random scheduling). The EPTO (0, 0) was marked with a red star, the optimal solution with a blue triangle, and the DP-based solution with a magenta square. The optimal O&P and the near-optimal solution found by the algorithm are closely located. Hence, the DP-based strategy is capable of selecting a reasonable near-optimal solution.

A trade-off between the execution time and the energy consumption is necessary when the inverse relationship between them is

Table 3: Benchmark kernels.

Sl.	Benchmark name	Benchmark source	Total DRAM R/W byte	Total flop	Operational intensity	GPU (standalone)		CPU (standalone)		Favorable processor
						Flop/s	*Avg. Power*	*Flop/s*	*Avg. Power*	
1	cfd	Rodinia	210 G	518 G	2.46	31.0 G	8.3	22.5 G	9.5	Both
2	$srad1$	Rodinia	449 G	441 G	0.88	29.8 G	8.3	87.1 G	9.0	CPU
3	$srad2$	Rodinia	663 G	1.3 T	2.0	77.1 G	7.7	21.8 G	10.6	GPU
4	$particlefinder$	Rodinia	6 G	33 G	5.2	2.8 G	7.9	2.3 G	17.3	Both
5	$stream-triad$	Roofline	918 G	115 G	0.125	22.6 G	9.1	19.4 G	15.3	Both
6	$heartwall$	Rodinia	221 G	1.1 T	5.27	113.8 G	15.6	9.3 G	9.4	GPU
7	nw	Rodinia	91 G	12 G	0.14	1.2 G	5.5	2.4 G	12.1	Both
8	lud	Rodinia	18 G	74 G	3.93	1.3 G	12.1	18.9 G	10.6	CPU

observed. However, in Fig. 5, the inverse relation is absent; hence, the need for a trade-off seems unnecessary (i.e., there is no need for EPTO) because some kernels are better suited for one processor. Running a kernel on the ill-suited processor leads to a nonoptimal result (i.e., higher execution time and a higher level of energy consumption). This explains why there are multiple clusters in Fig. 5. Section 7.3 further delves into the most close-to-optimal cluster, which is the leftmost-bottom cluster and shows the effects of different EPTO values on the success of the proposed scheduling technique.



(a) Execution time comparison. (b) Energy consumption comparison.

Figure 6: Model accuracy.

7.2 Accuracy of the Empirical Model

The proximity of the O&P generated by the algorithm to the optimal O&P relies on the accurate estimation of collocated execution times and energy consumption for each kernel produced by the model. To further evaluate the prediction accuracy of the models, the authors focused on a subset of five kernels in Table 3. The authors observed how the modeled energy and execution time match with the actual execution for all the combinations possible with five kernels, which is 482. Figures 6a and 6b depict the execution time and energy consumption for all 482 combinations by comparing the estimations from the model and the actual execution. In these figures, the x-axis shows the specific O&P combination. The model energy consumption and execution time values, which are denoted by the orange lines, are sorted, resulting in a smooth curve. The real execution time and energy consumption corresponding to the specific O&P for every data point are denoted by blue lines. The analysis shows that the model estimate is on par with the actual values for the execution time and energy consumption. Relative accuracy for an O&P is computed by the formula, $accuracy = [1 - Absolute(Real - Model)/Real]$*100, and then averaged for all. The average model accuracy for execution time and energy consumption is 88.4 and 92%, respectively.

7.3 Experiments with Different EPTO Goals

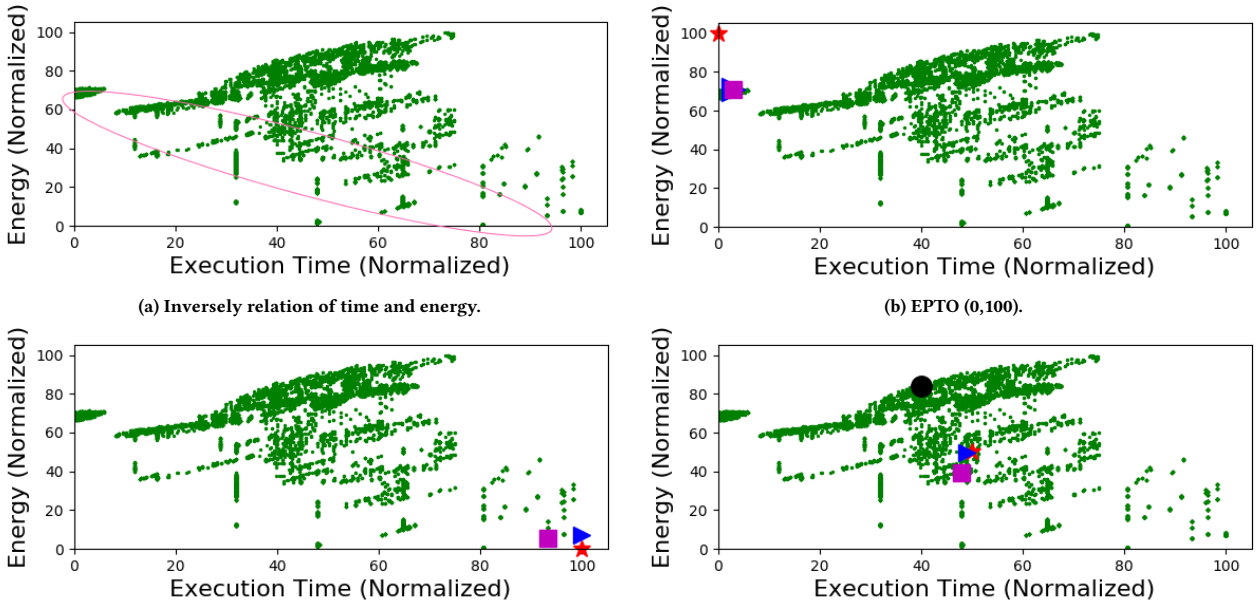
The experiment presented in Fig. 5 includes many possible O&P combinations that result in unpractical high-energy consumption and execution times. To better demonstrate the scale at which various EPTO goals can be used to achieve the desired trade-off, the amount of O&P combination space was reduced by identifying the kernels that are more suitable to run on CPUs or GPUs and fixing them to the corresponding PU.

A kernel is considered to be more suited for a specific processor if the ratio of the execution time is at least two times faster while taking average power into consideration. For example, a kernel has a $Flops/s$ value of $t1$ and $t2$ in PU1 and PU2 and the $Avg.Power$ of $pw1$ and $pw2$ in PU1 and PU2. If $t1/t2 > 2$ (at least two times faster) and $pw1/pw2 < 2$, then the kernel is suitable for PU1. In the same way, the authors determined whether the kernel is suitable for PU2. If the kernel does not satisfy the condition for any processor, then the kernel is considered favorable by both processors, and its placement in all these processors is considered.

Based on the $Flops/s$ and the $Avg.Power$ values reported in Table 3, the CPU-friendly kernels were identified as $srad1$ and lud, and the GPU-friendly kernels were identified as $srad2$ and $heartwall$ (last column of the table). On the other hand, the cfd, $particlefinder$, $stream-triad$, and nw kernels can be run on any processor since no PU always favors their execution.

After the fixed affinities are set, badly performing O&Ps are eliminated, and the inverse relationship between the execution time and energy consumption is revealed in Fig. 7a. The circled diagonal region demonstrates that there is no best solution that optimizes energy and performance, and there is a clear need for different EPTO goals. Figures 7b, 7d, and 7c show the optimal O&P combination (triangle), the near-optimal combination found by the algorithm (square), and the solution found by a greedy algorithm (circle) for different EPTO goals (shown by stars) of (0, 100), (50, 50), and (100, 0), respectively. These figures demonstrate that the DP-based scheduling can select the near-optimal O&P to achieve the desired trade-off between energy consumption and execution time.

EPTO provides significant control over kernel collocation decisions. If the user wants to pick an O&P without having a desired



(a) Inversely relation of time and energy.

(b) EPTO (0,100).

(c) EPTO (100,0).

(d) EPTO (50,50).

Figure 7: Demonstration of the need for EPTO. The red star is EPTO, the blue triangle is the optimal solution,the magenta square is the near-optimal solution by DP, and the black circle is the greedy solution. The *x*- and *y*-axes are normalized. The absolute minimum-maximum value pairs for energy and execution time are 0.72–1.2 kWatt-seconds and 45.5–77.4 seconds.

trade-off in mind, EPTO (0, 0) can be chosen. EPTO (0, 0) will always provide an O&P that is close to the diagonal line shown in Fig. 7a, but no specific trade-off is guaranteed. If the user wants the fastest execution time or the least energy consumption, EPTO (0, 100) and EPTO (100, 0), respectively, will guarantee that trade-off. Other EPTOs, such as EPTO (50, 50), also guarantee the desired trade-off. This strategy provides the run time system with more control over the device and, if necessary, empowers the run time system to dynamically choose different EPTO values based on the device's energy consumption priority. Moreover, this algorithm works irrespective of the system power cap, which in turn provides an extra level of control.

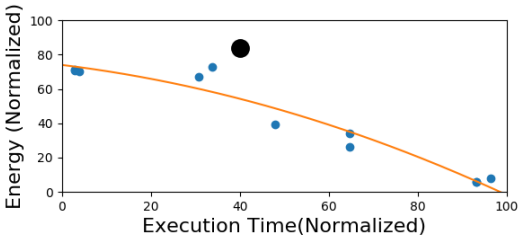
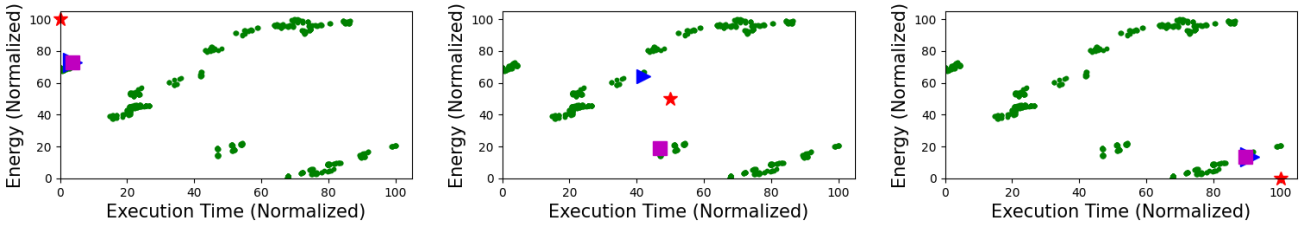


Figure 8: Comparison with greedy scheduling. Blue dots are EPTO points, and the black circle is the greedy solution.

7.4 Comparison with Greedy Algorithms

The authors believe that a study that consults execution time and energy consumption to mitigate memory contention has never been performed. For this reason, the closest work in which the greedy algorithm is only dependent on the execution time [37] was chosen for comparison. The greedy algorithm [37] starts by scheduling the longest GPU-friendly kernel in GPU and selects a CPU-friendly kernel to collocate. This strategy of collocation excludes lower operational intensity for reducing memory contention. This is a classic scenario that stems from the fact that a compute-intensive kernel should be collocated with a memory-intensive kernel. When one kernel in one processor finishes its execution, the next PU-friendly kernel to that processor is chosen. When all PU-friendly kernels are scheduled, the neutral kernels are chosen.

Following this strategy, an O&P of 3675|2841 is chosen by the greedy solution for which the numbers represent the kernels from Table 3. This O&P is compared in Fig. 7d by using a black circle. The positioning of the black circle reveals that there are many better solutions available in terms of the energy consumption and the execution time. Figure 8 presents a better comparison between the greedy approach and the proposed DP-based algorithm. The blue points are DP-based solutions for different EPTO points of 0–100, 10–90, 20–80, ..., 100–0, and a trade-off line is also drawn based on the positioning of different levels of EPTO. The greedy solution is observed to be significantly far from the trade-off line. The best case for execution time—EPTO (0, 100)—provides a scheduling in which execution time is 46.5 seconds for eight kernels, whereas the scheduling picked by the greedy solution provides an execution time of 58.3 seconds, which is 11.8 seconds more (i.e., 20% savings by the DP-based approach). On the other hand, the best case for energy consumption—EPTO(100, 0)—provides the total energy consumption of 0.75 kWatt-seconds, whereas the greedy solution shows the energy consumption of 1.1 kWatt-seconds, which translates to 32% energy savings. Although the DP-based solution is more computationally expensive than the greedy algorithm, it can save more execution time. For example, the DP-based solution takes 1.1 seconds to find the scheduling for eight kernels but can save 11.8



(a) EPTO (0,100). (b) EPTO (50,50). (c) EPTO (100,0).

Figure 9: Experiments with three processors: CPU, GPU, and PVA. The red star is the EPTO, the blue triangle is the optimal solution, and the magenta square is the near-optimal solution picked by DP. The x- and y-axes are normalized. The absolute minimum-maximum value pairs for energy and execution time are 0.47–0.78 kWatt-seconds and 34.01–58.47 seconds.

Table 4: OpenCV kernels for PVA.

Benchmark name	Benchmark source	Total DRAM R/W byte	Total flop	Operational intensity	PVA (standalone)		
					Execution time (sec)	*Flop/s*	*Avg. Power (Watt)*
klt_tracker	VPI.0.1	26 G	72 G	2.41	2.41	30.0 G	1.24
convolve_2D	VPI.0.1	7 G	70 G	5.35	5.35	13.2 G	1.39
timing	VPI.0.1	15 G	122 G	7.87	7.87	15.5 G	1.39

seconds. Moreover, the DP-based approach provides the means to achieve the desired EPTO.

7.5 Three-PU Scenario: CPU, GPU, and PVA

To demonstrate that MEPHESTO can work for diversely heterogeneous systems, the experiments were extended to cover three different PUs: CPU, GPU, and PVA. In this experiment, a different subset of eight kernels was used: five top kernels from Table 3 for execution on CPU and GPU and three OpenCV kernels from the NVIDIA VPI samples from Table 4 for execution on PVA. Among the five kernels from Table 3, *srad*1 is CPU-friendly, *srad*2 is GPU-friendly, and remaining three are considered for both CPU and GPU. For all eight kernels in which one kernel favors CPU, one favors GPU, and three favor PVA, there are 720 O&Ps. The optimal and near-optimal O&P selections are plotted in Fig. 9. For two different EPTO targets—(0, 100) and (100, 0)—the DP-based algorithm was able to select near-optimal solutions in Figs. 9a and 9c, respectively. However, EPTO(50, 50) in Fig. 9b shows an interesting case where optimal and the DP-based solution are far away from each other. While the distance between the optimal point (denoted by a triangle) and the EPTO point (denoted by a star) is shorter, the DP-based solution was able to locate a more energy efficient solution. This is mainly due to the fact that the Euclidean-distance-based method of finding the optimal reference point (denoted by a triangle) relies on the absolute distances and ignores whether the optimal point uses more or less energy/execution-time. However, the DP-based approach was able to pick a solution from one of the closest clusters from the EPTO point. This experiment demonstrates that the proposed algorithm can also achieve the desired trade-off for three PU diversely heterogeneous systems.

7.6 Overhead Analysis of DP-Based Search

The overhead of the proposed DP-based solution in MEPHESTO for varying numbers of kernels to be ordered and placed is shown in Fig. 10. For eight kernels, the algorithm finds the near-optimal solution in 1,061 milliseconds, and this time corresponds to only 1.9% (mentioned in the x-axis) of the total execution time compared with the minimum execution time. On the other hand, since the overhead increases exponentially, for larger kernel counts and shorter kernel execution times, the algorithm should be complemented with a windowing technique similar to the one proposed in Belviranli et al. [2]. Although this technique limits the benefits that can be obtained from considering all potential O&P possibilities, it is a simple and effective approach for controlling the increasing overhead. Moreover, the authors foresee that a multithreaded implementation of their DP-based algorithm will help reduce the overhead.

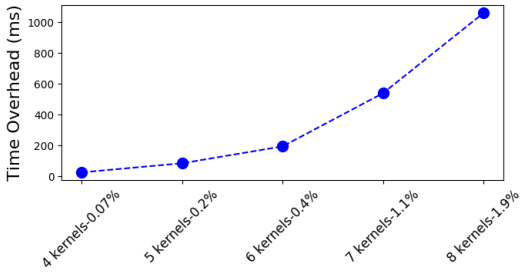


Figure 10: Overhead analysis.

8 RELATED WORK

8.1 Memory Contention Studies

This section reports two studies that are similar to this work. The first study was done by Zhu et al. [36, 37] in which the authors studied co-scheduling on an integrated CPU-GPU system and considered a power cap. They devised a greedy algorithm that addressed

memory contention from degradation in the execution time perspective while selecting frequency for power capping. However, they did not consider the impact of memory contention on power or energy. Moreover, the greedy algorithm does not provide any trade-off opportunity. The second study was done by Lee et al. [15] in which the authors designed a strategy to dynamically predict the slowdown due to memory contention. However, this study only considered execution time. Compared with these works, the strategy in this paper defines memory contention from both the energy and time perspectives while achieving the desired trade-off. Other works [7, 9, 22] studied memory contention and stalling in heterogeneous systems with shared LLC. Pan et al. [26] designed an LLC management strategy for better performance. Cavicchioli et al. [4] studied different SoCs and fused CPU-GPU devices to characterize memory contention. Hill et al. [10] extended the Roofline model for mobile SoCs to address memory contention from the perspective of PU BW usage. These studies mainly focused on performance and did not consider the impact of memory contention on power or energy consumption.

8.2 Kernel Collocation in CPU-GPU Systems

Kernel collocation in a CPU-GPU environment is also another well-studied area. Kaleem et al. [13] studied scheduling in integrated heterogeneous systems in which an online profile was used for load balancing between CPU and GPU. Panneerselvam et al. [28] devised a task placement strategy in a CPU-GPU system that achieves application-specific performance goals. Zhu et al. [37] designed a greedy algorithm with post-local refinement for memory contention-aware kernel collocation. Cho et al. [6] devised an on-the-fly strategy to partition irregular workloads in integrated CPU-GPU systems without considering energy consumption. Zhang et al. [34, 35] designed a decision tree-based model to determine the impact of kernel collocation on different applications in integrated CPU-GPU systems. Pandit et al. [27] designed a dynamic work distribution that considered the data transfer need of kernels in OpenCL run time. Liu et al. [18] designed a scheduling policy for tree traversal algorithms in which CPU and GPU transfer information to make a decision. Although there are more studies in the literature that investigated kernel collocation under memory contention, to the best of the authors' knowledge, there are no schemes that consider kernel collocation with the intention of addressing energy and performance simultaneously while considering the effects of memory contention on both factors. Moreover, almost all existing work focuses only on CPU/GPU-based systems, whereas this method works for more diverse heterogeneous systems, such as SoCs consisting of CPU, GPU, and PVA.

8.3 Energy-Aware Algorithm Studies

Barik et al. [1] introduced a black-box approach for finding energy-aware scheduling by characterizing applications. Ma et al. [20] designed GreenGPU, which dynamically throttles the frequency of GPU and memory. Zhu et al. [37] dynamically finds the appropriate frequency for applications to keep the execution under a power cap. Komoda et al. [14] also studied power capping by using DVFS to find near-optimal frequency settings for CPU-GPU. Intel introduced a power capping mechanism RAPL (running average power limit) in CPUs [30]. Liu et al. [17] designed an energy-aware kernel mapping strategy in a heterogeneous system in which PUs are assigned different frequencies by using DVFS. Unlike these studies, as mentioned previously, the method in this paper considers finding a collocation mapping that can lead to user-defined energy-performance balance while considering contention.

9 CONCLUSION AND FUTURE WORK

This study presents MEPHESTO, which defines memory contention in an integrated shared memory heterogeneous system in terms of energy and performance. MEPHESTO presents an empirical model to estimate a kernel collocation scenario for multiple kernels and devises a strategy to reach a desired energy-performance balance. Based on experiments, this strategy can predict execution time and energy consumption with an acceptable error rate and find a near-optimal solution that outperforms a greedy approach. Moreover, experiments demonstrated the efficacy of MEPHESTO by yielding near-optimal solutions for more than two processors. The authors plan to integrate their kernel collocation mechanism into a run time system.

ACKNOWLEDGMENTS

This research was supported in part by the following sources: Defense Advanced Research Projects Agency (DARPA) Microsystems Technology Office (MTO) Domain-Specific System-on-Chip Program, the US Department of Energy (DOE) Advanced Scientific Computing Research (ASCR) program, and by an appointment to the Oak Ridge National Laboratory ASTRO Program, sponsored by DOE and administered by the Oak Ridge Institute for Science and Education.

This manuscript has been co-authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.

REFERENCES

[1] Rajkishore Barik, Naila Farooqui, Brian T Lewis, Chunling Hu, and Tatiana Shpeisman. 2016. A black-box approach to energy-aware scheduling on integrated CPU-GPU systems. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*. ACM, 70–81.

[2] Mehmet E Belviranli, Farzad Khorasani, Laxmi N Bhuyan, and Rajiv Gupta. 2016. CuMAS: Data transfer aware multi-application scheduling for shared GPUs. In *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 31.

[3] Alexander Branover, Denis Foley, and Maurice Steinman. 2012. Amd fusion apu: Llano. *Ieee Micro* 32, 2 (2012), 28–37.

[4] Roberto Cavicchioli, Nicola Capodieci, and Marko Bertogna. 2017. Memory interference characterization between CPU cores and integrated GPUs in mixed-criticality platforms. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 1–10.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 44–54.

[6] Younghyun Cho, Florian Negele, Seohong Park, Bernhard Egger, and Thomas R Gross. 2018. On-the-fly workload partitioning for integrated CPU/GPU architectures.. In *PACT*. 21–1.

[7] Marvin Damschen, Frank Mueller, and Jörg Henkel. 2018. Co-Scheduling on Fused CPU-GPU Architectures With Shared Last Level Caches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2337–2347.

[8] Gaurav Dhiman and Tajana Simunic Rosing. 2007. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Proceedings of the 2007 international symposium on Low power electronics and design*. ACM, 207–212.

[9] Víctor García, Juan Gomez-Luna, Thomas Grass, Alejandro Rico, Eduard Ayguade, and Antonio J Pena. 2016. Evaluating the effect of last-level cache sharing on integrated GPU-CPU systems with heterogeneous applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.

[10] Mark Hill and Vijay Janapa Reddi. 2019. Gables: A Roofline Model for Mobile SoCs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 317–330.

[11] Y Jiang, X Shen, J Chen, and R Tripathi. 2011. The complexity and approximation of optimal job co-scheduling on chip multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 22, 7 (2011).

[12] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12.

[13] Rashid Kaleem, Rajkishore Barik, Tatiana Shpeisman, Chunling Hu, Brian T Lewis, and Keshav Pingali. 2014. Adaptive heterogeneous scheduling for integrated GPUs. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*. IEEE, 151–162.

[14] Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. 2013. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 349–356.

[15] Shin-Ying Lee and Carole-Jean Wu. 2017. Performance characterization, prediction, and optimization for heterogeneous systems with multi-level memory interference. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 43–53.

[16] Leek. 2020. CxxPolyFit: A simple library for producing multidimensional polynomial fits for C++. https://github.com/LLNL/CxxPolyFit

[17] Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. 2012. Power-efficient time-sensitive mapping in heterogeneous systems. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 23–32.

[18] Jianqiao Liu, Nikhil Hegde, and Milind Kulkarni. 2016. Hybrid CPU-GPU scheduling and execution of tree traversals. In *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 2.

[19] Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J Ligocki, Matthew J Cordery, Nicholas J Wright, Mary W Hall, and Leonid Oliker. 2014. Roofline model toolkit: A practical tool for architectural and program analysis. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 129–148.

[20] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *2012 41st International Conference on Parallel Processing*. IEEE, 48–57.

[21] J. D. McCalpin. 2002. Stream Benchmarks.

[22] Vineeth Mekkat, Anup Holey, Pen-Chung Yew, and Antonia Zhai. 2013. Managing shared last-level cache in a heterogeneous multicore processor. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*. IEEE Press, 225–234.

[23] Sparsh Mittal and Jeffrey S Vetter. 2015. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 69.

[24] Mohammad Alaul Haque Monil. 2020. Tegra parser: A tool to parse power consumption for NVIDIA tegra devices. https://github.com/monil01/tegra_parser/tree/master/c_parser

[25] NVIDIA. 2020. NVIDIA® Vision Programming Interface (VPI). https://docs.nvidia.com/vpi/index.html

[26] Abhisek Pan and Vijay S Pai. 2015. Runtime-driven shared last-level cache management for task-parallel programs. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[27] Prasanna Pandit and R Govindarajan. 2014. Fluidic kernels: Cooperative execution of opencl programs on multiple heterogeneous devices. In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization*. ACM, 273.

[28] Sankaralingam Panneerselvam and Michael Swift. 2016. Rinnegan: Efficient resource use in heterogeneous architectures. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*. ACM, 373–386.

[29] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. 2012. Real-time computer vision with OpenCV. *Commun. ACM* 55, 6 (2012), 61–69.

[30] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. 2012. Power-management architecture of the intel microarchitecture code-named sandy bridge. *Ieee micro* 32, 2 (2012), 20–27.

[31] K. L. Spafford, J. S. Meredith, S. Lee, D. Li, P. C. Roth, and J. S. Vetter. 2012. The tradeoffs of fused memory hierarchies in heterogeneous computing architectures. In *Proceedings of the 9th conference on Computing Frontiers*. ACM, 103–112. https://doi.org/10.1145/2212908.2212924

[32] J. S. Vetter, R. Brightwell, M. Gokhale, P. McCormick, R. Ross, J. Shalf, K. Antypas, D. Donofrio, T. Humble, C. Schuman, B. Van Essen, S. Yoo, A. Aiken, D. Bernholdt, S. Byna, K. Cameron, F. Cappello, B. Chapman, A. Chien, M. Hall, R. Hartman-Baker, Z. Lan, M. Lang, J. Leidel, S. Li, R. Lucas, J. Mellor-Crummey, P. Peltz Jr., T. Peterka, M. Strout, and J. Wilke. 2018. *Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity*. Technical Report. USDOE Office of Science (SC) (United States). https://doi.org/10.2172/1473756

[33] Samuel Williams, Andrew Waterman, and David Patterson. 2009. *Roofline: An insightful visual performance model for floating-point programs and multicore architectures*. Technical Report. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[34] Feng Zhang, Bo Wu, Jidong Zhai, Bingsheng He, and Wenguang Chen. 2017. FinePar: Irregularity-aware fine-grained workload partitioning on integrated architectures. In *Proceedings of the 2017 International Symposium on Code Generation and Optimization*. IEEE Press, 27–38.

[35] Feng Zhang, Jidong Zhai, Bingsheng He, Shuhao Zhang, and Wenguang Chen. 2016. Understanding co-running behaviors on integrated CPU/GPU architectures. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (2016), 905–918.

[36] Qi Zhu, Bo Wu, Xipeng Shen, Li Shen, and Zhiying Wang. 2014. Understanding co-run degradations on integrated heterogeneous processors. In *International Workshop on Languages and Compilers for Parallel Computing*. Springer, 82–97.

[37] Qi Zhu, Bo Wu, Xipeng Shen, Li Shen, and Zhiying Wang. 2017. Co-run scheduling with power cap on integrated cpu-gpu systems. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 967–977.