Concurrent and Orthogonal Software Power Meters for Accurate Runtime Energy Profiling of Parallel Hybrid Programs on Heterogeneous Hybrid Servers

Hafiz Adnan Niaz, *Member, IEEE*, Ravi Reddy Manumachu, *Member, IEEE*, and Alexey Lastovetsky, *Member, IEEE*

***Abstract*—Energy predictive models employing performance events have emerged as a promising alternative to other mainstream methods for developing software power meters used in runtime energy profiling of applications. These models are cost-effective and provide a highly accurate means of measuring the energy consumption of applications during execution. Recently, software power meters have been proposed to profile the dynamic energy consumption of data transfers between CPU and GPU in heterogeneous hybrid platforms, thereby effectively addressing the gap between software power meters that measure computations and those that measure data transfers. However, the state-of-the-art software power meters lack fundamental properties essential for achieving accurate runtime energy profiling of parallel hybrid programs on heterogeneous hybrid servers. Two critical properties are *concurrency* and *orthogonality*. In this work, we define these essential properties and propose a methodology for developing concurrent and orthogonal platform-level software power meters capable of accurate runtime energy profiling of parallel hybrid programs on heterogeneous hybrid servers. We apply this methodology to develop software power meters for three heterogeneous hybrid servers that consist of Intel multicore CPUs and Nvidia GPUs from different generations. Furthermore, we demonstrate the accuracy and efficiency of the proposed software power meters by using them to estimate the dynamic energy consumption of computation and communication activities in three parallel hybrid programs. Our results show that the average prediction error for dynamic energy consumption by these software power meters is just 2.5% across our servers.**

***Index Terms*—Software power meters, energy efficiency, energy profiling, energy modelling, parallel computing, heterogeneous computing, heterogeneous platforms.**

I. Introduction

HETEROGENEOUS hybrid servers with accelerators are ubiquitous in high-end digital platforms such as high-performance computing (HPC) clusters, data centres, and clouds. Runtime energy profiling of parallel hybrid programs on such servers is a challenge since it requires an accurate and efficient measurement of the energy consumed by different parallel computation and communication activities of the program. A parallel hybrid program running on a heterogeneous hybrid server involves the concurrent execution of both computation and communication activities. In this work, we primarily focus on developing software power meters that accurately profile the dynamic energy consumption of a parallel hybrid program. This refers to the energy consumed by the server due to the program's execution. In the following section, we will clarify the terminology used in this study, which includes terms such as heterogeneous hybrid server, parallel hybrid program, dynamic energy consumption, and total energy consumption.

There are three mainstream approaches for energy profiling: (a) System-level physical measurements using external power meters, (b) Measurements using on-chip power sensors, and (c) Energy predictive models.

The first approach considered the *ground-truth* is the most accurate nodal energy measurement method [1]. However, there are many challenges when employing it to measure the energy consumption of applications. First, this approach gives the node's total instantaneous power consumption, which is usually the power reading provided by the external power meter operating at the maximum sampling rate. Therefore, obtaining the energy consumption solely by the application is not straightforward and challenging due to persistent background noise arising from periodical OS, disk, network book-keeping tasks and fans. In addition, the power meter has a manufacturer-reported accuracy of measurement. The WattsUp Pro power meter employed in this work has a reported accuracy of $\pm$ 1.5% of the actual value, whereas the ANSI standard WT310 Yokogawa boasts an accuracy of $\pm$ 0.1%.

To eliminate any potential interference (noise) from activities unrelated to an application execution, one must employ statistical averaging (for example, using the Student's t-test). However, Fahad et al. [1] demonstrate that this statistical procedure is highly time-consuming mainly due to slow convergence to a sample mean meeting a required precision (or standard deviation of the mean) of 2.5%, which represents consistency in repeated measurements. A higher precision requirement will further delay the convergence. Second, the approach is unsuitable for runtime energy profiling in platforms lacking power meter hardware and software infrastructure. To summarize, the *ground-truth*

Received 16 June 2025; revised 2 October 2025; accepted 22 November 2025. Date of publication 26 November 2025; date of current version 11 December 2025. This work was supported by TaighdeÉireann - Research Ireland under Grant 20/FFP-P/8683. Recommended for acceptance by D. Li. *(Corresponding author: Ravi Reddy Manumachu.)*

The authors are with the School of Computer Science, University College Dublin (UCD), Belfield, D04 V1W8 Dublin, Ireland (e-mail: hafiz.niaz@ucdconnect.ie; ravi.manumachu@ucd.ie; alexey.lastovetsky@ucd.ie).

This article has supplementary downloadable material available at https://doi.org/10.1109/TPDS.2025.3637511, provided by the authors.

Digital Object Identifier 10.1109/TPDS.2025.3637511

approach provides a statistical mean of energy consumption of an application that is expensive to obtain.

Unlike the *ground-truth* method, on-chip power sensors enable online energy measurement through vendor-specific programmatic interfaces. However, this approach too has several disadvantages. First, there is a lack of comprehensive professional documentation describing how power or energy consumption is measured. For example, it is unclear whether the on-chip power sensors are based on software energy predictive models or electronic hardware models (employing current estimates and input voltage). Hence, on-chip sensors function as closed boxes, making it difficult to relate the energy consumption of the computing elements containing the sensors with the overall energy consumption of an application.

Second, on-chip power sensors for Intel multicore CPUs have been shown to be inaccurate when profiling CPU computations and the data transfers between the multicore CPU and GPU in heterogeneous hybrid servers [1], [2]. In contrast, the on-chip power sensors for Nvidia GPUs, which utilize the Nvidia Management Library (NVML), have improved significantly in accuracy for profiling GPU computations, particularly in the latest processor generations (for example, starting from the A40 GPU processor generation). However, it is important to note that NVML does not provide energy consumption data for data transfers between the host CPU and GPU. Ultimately, the time required to experimentally determine the energy consumption of an activity (whether computations or data transfers) is roughly equivalent to that of the ground-truth method, as both involve costly statistical averaging using the Student's t-test to determine the accurate ground-truth sample mean.

The third approach involves using software energy predictive models that employ performance events, which are measurable runtime software and hardware activities as predictor variables. This method has emerged as a promising alternative for developing software power meters for runtime energy profiling compared to other mainstream methods. There are two main reasons for this. First, if implemented correctly, its runtime accuracy can surpass that of other approaches. In fact, a single runtime measurement obtained from other methods (such as power meters and on-chip sensors) can significantly differ from the accurate but expensive statistical ground-truth mean. For instance, the average prediction error of these other methods used on our dedicated HPC servers, after three repetitions, ranged from 30% to 40%.

A software power meter that uses a linear energy predictive model with reliable performance events, defined as those events that yield consistent counts across runs, regardless of environmental noise, will produce the same ground-truth mean value even when environmental noise is present. This indicates that the performance event-based method can achieve the same level of accuracy as the ground-truth method when there is no noise. Additionally, it is likely to be more accurate in noisy conditions because it provides measurements that would align with the ground-truth method under ideal, noise-free circumstances.

In addition, a software power meter is cost-effective since the reliable performance events employed in accurate linear energy predictive models exhibit the same counts from run to run and do not need statistical averaging. Therefore, the time to obtain an experimental data point using this approach is significantly less than the other approaches.

To summarize, energy predictive models employing reliable performance events is the most cost-effective and potentially the most accurate method for the measurement of energy consumption of applications. We now survey prominent research works in energy predictive modelling of computations and data transfers for CPUs and GPUs based on performance events before explaining why they are inadequate for accurate runtime energy profiling of parallel hybrid programs on heterogeneous hybrid servers.

Research works [2], [3], [4], [5], [6], [7], [8], [9], [10], [11] propose energy predictive models for CPU processors based on performance events and resource utilization. Research works [2], [11] consider factors crucial to the accuracy of linear energy predictive models on modern multicore CPUs, which include *additivity* of performance events [12] and *consistency test* containing selection criteria for model variables, intercept, and coefficients [13].

Research works [14], [15], [16], [17], [18], [19] propose energy predictive models for computations on GPU processors based on Nvidia CUDA Profiling Tools Interface tool (CUPTI) events and metrics. Niaz et al. [2] is the first work that comprehensively studies the energy consumption of data transfer between a host CPU and a GPU accelerator on heterogeneous hybrid platforms using the three mainstream energy measurement methods. They propose a methodology that selects a small subset of performance events that can be obtained in one application run and effectively captures all the energy consumption activities during a data transfer between a host CPU and a GPU accelerator. Furthermore, they design accurate and reliable platform-level linear energy predictive models employing disjoint sets of performance events to estimate the dynamic energy of computations and data transfers during parallel hybrid program execution. For the computations involving the GPU processors, on-chip sensors are employed since they are found to be accurate for Nvidia A40 GPUs employed in their experiments.

However, the proposed software energy predictive models are not suitable for employment as software power meters for accurate runtime energy profiling of parallel hybrid programs running on heterogeneous hybrid servers. This is because the software power meters lack some fundamental properties, with two critical ones being *concurrency* and *orthogonality*. Although the software power meters based on the software energy predictive models proposed in [2] are concurrent and disjoint, they do not possess orthogonality.

We will describe the fundamental properties including the two critical ones, concurrency and orthogonality, as well as the motivation and intuition behind the properties in Section III. We will use the term software power meter designed to measure an activity (computation or data transfer) to denote a linear energy predictive model based on performance events. Therefore, the term *software power meter* refers specifically to a performance event-based software power meter.

In brief, concurrency of software power meters refers to their capability to simultaneously collect their respective sets of performance events capturing accurately and reliably the concurrent execution of the sensed computation and communication activities. This means that software power meters can gather all performance events at once during a single application run. For software power meters to function concurrently, sufficient resources (such as registers) must be available to store all events simultaneously. Furthermore, the performance event sets of these power meters do not have to be disjoint; in theory, two concurrent power meters can share an event and read it independently at different points during an application run.

On the other hand, orthogonality pertains to a pair of software power meters measuring two independently powered activities and means that the power meters can accurately and reliably measure the energy of these activities whether they happen serially or in parallel. For two software power meters to be considered orthogonal, they must only sense and measure their own activities and remain unaffected by the activities monitored by their counterpart. An essential condition for achieving orthogonality is that the sets of performance events of the software power meters must be disjoint.

We will now further elaborate on the reasons why the state-of-the-art models are unsuitable. First, energy profiling and optimization of computations have dominated energy research because of the prevailing belief that the energy consumption of data transfer is negligible compared to that of computations. However, this notion has been proven false, as demonstrated in Niaz et al. [2]. Consequently, much of the existing work has focused on developing energy predictive models that estimate the energy consumption of only the CPU or GPU computational activities.

Theoretically, CPU and GPU energy predictive models can serve as software power meters for estimating the energy consumption of concurrent executions of CPU and GPU computations, as they utilize disjoint sets of performance events. The CPU models rely on performance events that track the CPU's on-core activities, while GPU performance events are derived from CUPTI events that monitor the activities of various hardware components within the GPU. To the best of our knowledge, there is only one study by [20] that proposes an energy model based on a neural network trained on both CPU and GPU performance events to estimate the energy consumption of parallel applications on heterogeneous mobile platforms. However, these models may not provide accurate estimates for runtime energy profiling of parallel hybrid programs, which involve the concurrent execution of computation and communication activities. This inaccuracy arises because computations and data transfers use shared CPU resources, both on-core and off-core. As a result, CPU models may overestimate the energy consumption of CPU computations in scenarios where performance events capture energy-consuming activities related not just to CPU computations but also to simultaneous CPU computations and data transfers between the CPU and GPU.

Second, the research work [2] uses a methodology that identifies disjoint sets of performance events for three activities: CPU computations, data transfers from the CPU to a GPU, and data transfers from a GPU to the CPU. These sets of performance events are then used to create accurate linear energy predictive models for each activity. Although the sets are concurrent and disjoint, they are not orthogonal; this is due to the fact that the methodology does not account for the simultaneous execution of computation and communication activities. As a result, the performance event sets may not accurately reflect runtime energy profiling for concurrent computation and communication activities in parallel hybrid programs.

To summarize, state-of-the-art research can be divided into two main categories: a) Studies that propose energy predictive models for CPUs and GPUs. These models can function as practical software power meters for estimating the energy consumption of concurrent execution in CPU and GPU computational activities, but not for communication activities; and b) Studies that utilize a methodology to construct accurate energy predictive models for serial computations and data transfers, while overlooking the critical property of orthogonality in their design and implementation.

In this work, we define key properties, such as concurrency and orthogonality, of software power meters that are essential for accurately profiling runtime energy consumption of parallel hybrid programs on heterogeneous hybrid servers. These power meters are ideal for dynamically optimizing parallel hybrid programs on heterogeneous hybrid servers with respect to both performance and energy. We then present a methodology for developing concurrent and orthogonal software power meters that provide accurate runtime estimation of energy consumption associated with independently powered parallel computation and communication activities in parallel hybrid programs. These power meters implement platform-level linear energy predictive models that employ performance events provided by the Likwid package [21]. Likwid organizes performance events into groups that are customized for specific processor models. Detailed descriptions of the performance events and their groupings can be found in the supplemental materials. Additionally, our methodology incorporates an efficient method for selecting performance events based on their natural groupings, as derived from the processor architecture identified by Likwid, as well as through an additivity test and a strong positive correlation with energy consumption.

We then employ the methodology to develop concurrent and orthogonal software power meters for three heterogeneous hybrid servers that feature Intel multicore CPUs and Nvidia GPUs from different generations. For each server, this methodology finds power meters for computations taking place in the multicore CPU, computations in a GPU, and data transfers between the CPU and GPU (one power meter for each direction). A crucial aspect of this methodology's success in finding concurrent and orthogonal software power meters for both computations and data transfers is that modern processor architectures provide numerous redundant performance events that capture both on-core and off-core activities. While the data transfer power meters include performance events associated with off-core chip traffic, the power meters for computations primarily utilize core-local performance events belonging to the Performance Monitoring Counter (PMC) group.

We demonstrate the accuracy and efficiency of our proposed software power meters through runtime energy profiling of three parallel hybrid programs: matrix addition, matrix multiplication, and 2D fast Fourier transform, all tested on the three servers. In addition to their fundamental properties of concurrency and orthogonality, our proposed software power meters exhibit several advantageous features.

The average prediction error for total dynamic energy consumption on our servers is only 2.5%. However, this percentage is affected by the higher average prediction error of 5% from software power meters used for GPU computations. These meters rely on power readings from the GPUs' on-chip power sensors.

We would like to highlight that our power meters can accurately monitor all three hybrid programs because they are designed for platform-level use rather than being tailored to individual applications. As a result, no modifications to the power meters are necessary to capture energy consumption activities specific to any application. It's worth noting that optimizing the runtime of parallel hybrid programs using our proposed software power meters is beyond the scope of this work.

The main contributions of this work are:

- Formulation of fundamental properties, such as concurrency and orthogonality, of software power meters essential for accurate runtime dynamic energy profiling of parallel hybrid programs running on heterogeneous hybrid servers.
- Methodology to construct concurrent and orthogonal platform-level software power meters that enable accurate runtime estimation of dynamic energy consumption associated with parallel computation and communication activities in hybrid programs.
- Development of concurrent and orthogonal software energy power meters for three heterogeneous hybrid servers that consist of Intel multicore CPUs and Nvidia GPUs from different generations.
- Experiments demonstrating the high accuracy and efficiency of these power meters in profiling runtime energy consumption for three parallel hybrid programs: matrix addition, matrix multiplication, and 2D fast Fourier transform, across the three heterogeneous hybrid servers.

The structure of the rest of the paper is organized as follows: Section II defines key terms related to energy consumption in computing and the statistical confidence used in our experiments. Section III describes the concurrency and orthogonality properties of software power meters. Section IV details the methodology of construction of concurrent and orthogonal software power meters. Section V presents our proposed software power meter API. Section VI provides a detailed analysis of the results for three parallel hybrid applications. Section VII reviews related research. Finally, Section VIII concludes the paper.

II. Energy-Efficiency Terminology and Statistical Confidence in Experiments

We define a heterogeneous hybrid server as a platform comprising a multicore CPU and one or more accelerators. The computing devices in such platforms are the CPU and the accelerators. In this work, we mean a CPU and an accelerator when we refer to "a pair of communicating computing devices". Some platforms have direct links between the accelerators. These platforms are out of the scope of this work.

A parallel hybrid program running on a heterogeneous hybrid server comprises several software components (kernels) executing in parallel. There is a one-to-one mapping between the components and computing devices of the hybrid platform. Executing an accelerator component involves a dedicated CPU core, which runs a hosting thread, along with the accelerator itself that executes the accelerator code. The hosting thread running on the CPU core manages and coordinates the execution of a kernel on an accelerator. The execution of the accelerator component includes three key steps: data transfer between the CPU and accelerator memory, computations performed by the accelerator code, and data transfer back from the accelerator memory to the CPU. In contrast, executing a CPU component solely involves the CPU cores running the multithreaded CPU code.

An activity in a software component pertains to either computations within the component or the data transfer between a pair of communicating computing devices. Therefore, at any given moment during its execution, the parallel hybrid program can engage in a single activity - such as computation on the CPU, computation on one of the accelerators, or data transfer between a pair of computing devices - or it may perform multiple such activities in parallel.

Consider a program executing only one activity. To calculate the dynamic energy E_D of the activity, the following formula is used: $E_D = E_T - (P_S \times T_E)$. In this formula, P_S represents the static power consumption of the platform, E_T is the total energy consumption of the platform during the activity, and T_E is the duration of the activity in seconds.

The execution time of all activities is measured on the CPU side. The execution time of a data transfer between a pair of computing devices is measured on the CPU side using the CPU processor clock. The process involves the following steps: querying the processor clock to record the start time, executing the data transfer, querying the processor clock again to record the end time, and then calculating the difference between the start and end times. The execution time of a computation activity is measured in a similar fashion. The execution time is reported in seconds (Sec).

The dynamic energy consumption of an activity is also measured on the CPU side. The measurement steps differ for methods using a software power meter and ground-truth. We describe the measurement step sequence in the ground-truth method using the HCLWattsUp energy measurement API [22]. The sequence includes starting the energy meter using HCLWattsUp's *start()* function, executing the activity, stopping the energy meter using HCLWattsUp's *stop()* function, and then querying the energy meter for the total energy consumption of the activity using HCLWattsUp's *energy()* function. The dynamic energy consumption (E_D) is obtained using the formula above. All the steps in the sequence are invoked from the CPU side.

For a software power meter, the measurement step sequence includes starting the power meter, executing the activity, stopping the power meter, and then querying the power meter for dynamic energy consumption. The units used in this work for power and energy consumption are Watts (W) and Joules (J).

The prediction error of a software power meter is calculated as follows: Prediction Error (%) $= (|Actual - Estimated|)/Actual \times 100$ where $Actual$ is the value obtained using the ground-truth method and $Estimated$ is the value given by the software power meter. To calculate the relative percentage difference between two values, we use the formula: $(|e_2 - e_1|)/e_1 \times 100$ where e_1 is the baseline.

In all experiments utilizing the ground-truth method, we ensure the reliability of our measurements by repeating the experiment until the sample mean falls within a 95% confidence interval and achieves a precision (or standard deviation of the mean) of 2.5%. To accomplish this, we employ Student's t-test, which assumes that the individual observations are independent and that their population follows a normal distribution. We verify the validity of these assumptions using Shapiro-Wilk Test.

Finally, we use the standard error (SE) to indicate the variability in the data. For example, we employ the standard error of the sample mean to illustrate the uncertainty in the estimate of the sample mean.

III. Concurrency and Orthogonality of Software Power Meters

We will begin by defining the concept of independently powered activities within a parallel hybrid program. This definition is crucial for understanding the intuition behind concurrent and orthogonal software power meters. Next, we will outline the properties or characteristics of these power meters. Additionally, in this work, we will focus exclusively on GPU-based hybrid servers, so for simplicity, we will use the term "GPU" instead of the more general term "accelerator" from this point forward.

A parallel hybrid program running on a heterogeneous hybrid server at any moment of time comprises one or several simultaneously running activities. The types of activities include: (a) Computations on the CPU cores, excluding the cores that are dedicated to host GPUs and connected to them via PCIe links, (b) Computations on a GPU, and (c) Data transfer (or communication) between the CPU and a GPU in either forward or reverse direction.

We define a pair of activities as being independently powered if the total dynamic energy consumption of the pair equals the sum of the dynamic energy consumption of each individual activity, regardless of whether the activities are running simultaneously. Furthermore, a set of activities is considered independently powered if each pair of activities within that set is independently powered.

To clarify the concept of independently powered activities, we examine pairs of such activities in the execution of a parallel hybrid program on the heterogeneous hybrid server used in our experiments, Skylake P100 GPU server, which consists of a single-socket Intel multicore CPU and an Nvidia P100 GPU. In this setup, CPU and GPU computation activities are independently powered. Additionally, a communication activity between a host CPU core and the GPU and CPU computation activity are also independently powered. It is important to note that the host CPU core is not shared between these two activities, allowing us to treat them as independently powered.

However, parallel communication activities between the CPU and GPU - whether in forward or reverse directions, or both - are not independently powered, as they share the same power source. Now, consider the execution of a parallel hybrid program on a heterogeneous hybrid server equipped with a dual-socket multicore CPU and two GPUs: the first GPU has affinity to the first CPU socket, while the second GPU has affinity to the second CPU socket. In this scenario, a pair of communication activities involving different GPUs will be independently powered, regardless of the direction of data transfer.

Now consider a class of parallel hybrid programs where all activities in any program in this class are independently powered on a given platform. Our goal is to develop a set of software power meters, with one dedicated to each activity, that would correctly and accurately measure the dynamic energy consumed by these activities in any program within this class. The power meters will function correctly if and only if the following properties are satisfied:

1) All power meters must be able to run in parallel.
2) If a specific activity does not occur during the measured time interval, the power meter designed to measure this activity will return zero (or negligible) dynamic energy consumption.
3) If a specific activity is running during a given time interval, the power meter designed to measure this activity must accurately report the dynamic energy consumed by this activity during that time interval.

If these properties are satisfied, the power meters will provide accurate measurements of dynamic energy consumption for each activity during any time interval of the program's execution. Furthermore, the sum of the measured values will equal the total dynamic energy consumed by the program within that time interval.

In this work, we propose software power meters that are implemented as linear dynamic energy predictive models, using normalized performance events as predictor variables. Next, we formulate certain properties that these power meters should have in relation to the performance events. Software power meters that meet these properties will also satisfy properties (1) to (3), ensuring their correctness.

The properties are outlined as follows:

a) If only one activity is running during the execution of a program, the power meter designed to measure this activity will return the accurate dynamic energy consumed by this activity. Moreover, each performance event of the power meter will have the same (or almost the same) count for different runs of the program.
b) All performance events used by the power meters can be collected in parallel.
c) Power meters do not share performance events, meaning each power meter has a unique set of performance events used as predictors.
d) If a specific activity does not occur within the measured time interval, all performance events of the power meter

designed to measure this activity will have zero (or negligible) counts.

e) Consider two programs, each running a different activity, A and B. In this case, each performance event of the power meters designed to measure these activities will return the same (or almost the same) count, regardless of the order in which the programs are executed.

These properties are formulated to be directly testable through experiments. A set of software power meters satisfying property (b) is called *concurrent*. A set of power meters fulfilling (c), (d) and (e) is termed *orthogonal*.

In the following section, we present a detailed experimental methodology aimed at constructing software power meters that meet these properties. This methodology includes a fast selection procedure for the performance event sets used in the power meters.

We then employ the methodology to develop power meters for three heterogeneous hybrid servers: a server having one single-socket multicore CPU and one GPU, a server with a dual-socket multicore CPU and one GPU, while the third server is equipped with one single-socket multicore CPU and two identical GPUs.

IV. Methodology to Construct Concurrent and Orthogonal Software Power Meters

This section outlines our methodology for constructing concurrent and orthogonal software power meters designed for accurate runtime profiling of parallel hybrid programs on a heterogeneous server, which comprises one multicore CPU and two Nvidia GPUs.

We implement software power meters to measure the dynamic energy consumption of CPU computation and communication activities between CPU and GPU using linear energy predictive models. These models utilize Likwid performance events [21] as model variables. Niaz et al. [2] note that no CUPTI events or metrics capture data transfer activity on the GPU side. This finding highlights the passive role of GPUs in data transfer between CPUs and GPUs, indicating that the GPU's share of energy-consuming activities is insignificant compared to the CPU. Furthermore, our power meters for GPU computational activities rely on the NVML API for the on-chip sensors of the GPU. We have determined that these on-chip sensors provide accurate and reliable data for GPU generations starting from A40 GPU. Additionally, through comprehensive experiments on our Intel Icelake A40 server, we have confirmed that our two GPU software power meters for the two A40 GPUs are both concurrent and orthogonal.

We begin by discussing the challenging task of shortlisting Likwid performance events. Likwid is a well-known mainstream tool to obtain performance event counts of multicore CPU processors. It provides a certain number of core-local and socket-wide counters, denoted as c, and offers e performance events. The values of c and e vary between different processor architectures, with e typically being much larger than c. Moreover, both c and e tend to increase with each new processor generation. The counters are further divided into two groups: core-local and socket-wide, with each group capable of storing counts for its corresponding performance events.

However, the large number of available performance events makes it practically infeasible to collect all of them during runtime. Consequently, identifying a small subset of performance events that effectively captures all energy consumption activities during application execution is a significant challenge. While statistical methods, such as correlation and Principal Component Analysis (PCA), might seem like suitable solutions, relying solely on these techniques proves ineffective for generating accurate energy predictive models.

An effective technique for selecting performance events is the additivity test proposed by [12]. Additivity is based on the experimental observation that the total energy consumption when executing two applications in sequence is equal to the sum of the energy consumptions for each application run individually. Consequently, the additivity criterion follows a straightforward and intuitive rule: the performance event count for the serial execution of two applications should equal the sum of the counts observed during the individual executions of each application. The authors [12] point out that the inaccuracy of energy models based on performance events arises because many commonly used core-local performance events are not additive on modern multicore processors. Therefore, the accuracy of performance monitoring counter (PMC)-based energy models can be enhanced by excluding non-additive performance events.

Furthermore, a significant source of inaccuracy in these models is the violation of fundamental energy conservation laws. This includes issues such as having a non-zero intercept in dynamic energy models, negative coefficients in additive terms, and the non-linearity present in some advanced models, including machine learning (ML) models [12], [13]. The theory of energy predictive models in computing [13] formalizes properties of energy predictive models based on performance event counts, which are derived from the fundamental law of energy conservation. The practical implications of this theory for improving the prediction accuracy of linear energy predictive models are consolidated in a consistency test. This test includes criteria for selecting model variables, intercepts, and coefficients to develop accurate and reliable linear energy predictive models.

Thus, to enhance the accuracy of dynamic energy models that utilize performance events, it is essential to eliminate non-additive performance events from the models and ensure adherence to basic energy conservation laws.

The methodology employs a fast selection procedure that integrates a natural grouping of performance events based on processor architecture, additivity, the theory of energy predictive models, and high positive correlation. This process is much quicker than an approach that evaluates all possible pair-wise correlations, which is impractical given the large number of performance events. This procedure aims to select concurrent and orthogonal sets of performance events.

A. Fast Selection Procedure for Performance Event Sets

The inputs to the procedure are sets of applications, the base additivity error, $\mathbb{A}, 0 \leq \mathbb{A} \leq 100$, the base positive correlation,

$\rho, 0 \leq \rho \leq 100$, the statistical precision (standard deviation of the mean), $\epsilon, 0 \leq \epsilon \leq 100$, and the threshold, $\tau, 0 \leq \tau \leq 1$. The application sets include:

- B_{C_1} comprising k base applications involving CPU computations.
- B_{H2D_1} and B_{H2D_2} comprising l_1 and l_2 base applications, respectively, involving data transfers of different sizes from the host CPU to GPU_1 and the host CPU to GPU_2, respectively.
- B_{D2H_1} and B_{D2H_2} comprising m_1 and m_2 base applications, respectively, executing data transfers from GPU_1 to CPU and from GPU_2 to CPU, respectively.
- n compound applications.
- h carefully designed synthetic parallel hybrid benchmarks comprising three CPU threads that lead the execution of CPU and GPU computations and are responsible for data transfers between the host CPU and GPUs.

A *compound application* is defined as the sequential execution of two base applications. If the base applications are denoted as A and B, we represent their compound application as AB. The total number of compound applications, n, is derived from a random selection of base application pairs. Compound applications encompass both CPU computations and data transfers. However, each individual base application consists of either a CPU computation or a data transfer activity, but not both.

The parallel hybrid benchmarks used in the procedure are representative of the class of parallel hybrid programs, where all activities within the program are independently powered. In each parallel hybrid benchmark, all simultaneously executed activities are always independently powered. Specifically, data transfers between the host CPU and GPUs occur simultaneously in one direction. However, if multiple data transfers use the same CPU-GPU pair, those transfers happen sequentially. For instance, a data transfer from the host CPU to a GPU is followed by a data transfer from that same GPU back to the host CPU.

The procedure does not utilize Pearson correlation coefficient to assess the correlation between two performance events or to evaluate the correlation of a performance event with dynamic energy consumption. Instead, it examines the functional relationship between a performance event and dynamic energy (or a pair of events) to determine if a strong positive correlation exists. A correlation is considered not significantly positive if there is notable non-linearity in the functional relationship. We use the same parameter ρ when checking the correlation between two performance events as well as between a performance event and dynamic energy consumption.

The procedure outputs concurrent and orthogonal sets of performance events for CPU computations and data transfers. According to the definition of concurrency, a single application run is enough to gather the concurrent sets of performance events. However, if the sets of performance events are not concurrent, multiple application runs will be necessary. As a result, the procedure provides a positive integer that indicates the number of application runs needed to collect all the performance event sets.

The procedure has four stages. In the first stage, it selects all highly additive (with an additivity error $\leq \mathbb{A}$) and highly positively correlated performance events with dynamic energy ($\geq \rho$). This stage relies solely on the input compound applications. The main steps of this stage are detailed in the supplemental, Section IV.

The second stage partitions the output from the first stage into five sets: $\{P_{C_1}, P_{H2D_1}, P_{H2D_2}, P_{D2H_1}, P_{D2H_2}\}$. The set P_{C_1} contains events attributed to CPU computation activity. The sets P_{H2D_1} and P_{H2D_2} include events specific to data transfer activities from the host CPU to GPU_1 and GPU_2, respectively. The sets P_{D2H_1} and P_{D2H_2} contain events related to data transfer from GPU_1 to the CPU and from GPU_2 to the CPU, respectively.

This stage uses input application sets that comprise the base applications: $\{B_{C_1}, B_{H2D_1}, B_{H2D_2}, B_{D2H_1}, B_{D2H_2}\}$. The main steps of this stage are outlined below:

- The procedure begins by categorizing the shortlisted performance events from stage one into performance monitoring counter groups provided by Likwid. These groups are labeled as $g_1, g_2, \ldots, g_G$. We will describe the specific Likwid groups for the hybrid servers used in this study in the next section. Let group $g_i, i \in \{1, \ldots, G\}$ contain c_i performance events, $\{e_{i1}, e_{i2} \cdots, e_{ic_i}\}$. The analysis for each group $g_i, i \in \{1, \ldots, G\}$ proceeds as follows:
  - For each event $e_{ij}, i \in \{1, \ldots, G\}, j \in \{1, \ldots, c_i\}$ and each base application b from the sets, $\{B_{C_1}, B_{H2D_1}, B_{H2D_2}, B_{D2H_1}, B_{D2H_2}\}$, the rate of the event count, $r(e_{ij}, b)$, is calculated as follows: $r(e_{ij}, b) = \frac{count(e_{ij}, b)}{t(b)}$. In this formula, *count*(x, y) returns the normalized count of event x during the execution of the application y, and $t(y)$ is the execution time of application y.
  - Let τ be the specified threshold. Then for each event $e_{ij}, i \in \{1, \ldots, G\}, j \in \{1, \ldots, c_i\}$ and for each activity $p \in \{C_1, H2D_1, H2D_2, D2H_1, D2H_2\}$, we define the following criteria for inclusion of e_{ij} in P_p:
    1) The minimum rate of the event count of e_{ij} across all base applications involving activity p must be greater than τ:

    $$\min_{b \in B_p} r(e_{ij}, b) > \tau$$

    2) The highest rate (the maximum of the maximum rates) of the event count of e_{ij} across all base applications involving activities other than p must be less than or equal to τ:

    $$\max_{q \neq p} \left\{ \max_{b \in B_q} r(e_{ij}, b) \right\} \leq \tau$$

    If both conditions (1) and (2) are satisfied, then e_{ij} is included in P_p.
  - If e_{ij} is included in P_p or if all activities have been considered, we proceed to the next event. If not, we move on to the next activity.

The inputs to the third stage consist of h synthetic parallel hybrid benchmarks and the five performance event sets, $\{P_{C_1}, P_{H2D_1}, P_{H2D_2}, P_{D2H_1}, P_{D2H_2}\}$, obtained from the second stage. The objective of this stage is to analyze the impact of interference caused by independently powered activities running in parallel on these performance events while also eliminating events that exhibit cross-contamination. The stage follows these main steps:

- For each hybrid program $h_k, k \in \{1, \ldots, h\}$ and for each event set, $P_p, p \in \{C_1, H2D_1, H2D_2, D2H_1, D2H_2\}$:
  - The procedure analyzes the counts of all events in the event set, P_p. For each performance event, it calculates the relative percentage difference between the normalized event counts obtained for the hybrid program execution and those from the stand-alone execution of the base program used in the hybrid to implement the p-th activity. If this relative percentage difference exceeds the specified tolerance level, ϵ, the performance event is eliminated from consideration. This test removes from P_p all performance events, whose counts are affected by parallel execution of other activities. If such events were to be used as predictors in linear dynamic energy models employed by software power meters, they would violate the orthogonality property (e) formulated in Section III.

The five performance event sets produced in the third stage, $\{P_{C_1}, P_{H2D_1}, P_{H2D_2}, P_{D2H_1}, P_{D2H_2}\}$, are orthogonal for the h synthetic parallel hybrid benchmarks utilized in this stage. We consider these benchmarks to be representative of the class of parallel hybrid programs where all simultaneously executed activities are independently powered.

In the final stage, these five orthogonal performance event sets are used as input. The objective of this stage is to make the sets concurrent by eliminating redundant events in each set that duplicate the contributions of other events in the linear dynamic energy model employing the set. The process consists of the following steps:

- The procedure first checks if there are enough performance monitoring counters to store the shortlisted performance events from the five sets. If there are sufficient counters, the procedure terminates and returns the performance event sets, along with the indication that one application run is sufficient to obtain all the performance events.
- For each performance event set, $P_p, p \in \{C_1, H2D_1, H2D_2, D2H_1, D2H_2\}$, the procedure examines the Likwid performance monitoring counter groups associated with the performance events in the set. It then performs intra-group and inter-group sub-steps, which are detailed in [2]. Specifically, these two sub-steps involve analysis of the Likwid performance monitoring counter groups within a performance event set. The analysis utilizes all event records, including the performance event counts and the dynamic energy consumption data generated in the previous stages.
  - In the intra-group step, the procedure prunes the performance events within each group based on their correlation with one another and their dynamic energy consumption.
  - In the inter-group step, the focus shifts to pruning performance events across different groups by examining pairwise correlations. This step aims to identify a minimal set of performance events that are highly positively correlated with dynamic energy, while not being highly correlated with each other. This approach captures separate independent contributions to dynamic energy that can be obtained from a single application run.
- If the performance event sets are not concurrent, the procedure increases the correlation (ρ) in increments of 1% (up to the maximum allowed by the user-specified statistical precision, ϵ). For each increment, it removes all highly additive performance events (with an additivity error $\leq \mathbb{A}$) that have a positive correlation (at least $\geq \rho$). If the resulting performance event sets are concurrent, the procedure exits.
- If the performance event sets remain non-concurrent, the procedure reduces the additivity error ($\mathbb{A}$) in increments of 1% (again, up to the maximum allowed by the user-specified statistical precision, ϵ). For each decrement, it removes all performance events with an additivity error greater than $\mathbb{A}$.

If the procedure fails to identify the desired performance event sets after these steps, it will determine the number of application runs needed to obtain all the shortlisted performance events. It will then return this number along with the performance event sets.

B. Scalability of the Methodology

The methodology's scalability pertains to its capacity to identify concurrent and orthogonal performance event sets as the number of physical CPU sockets and GPUs increases. Three factors influence scalability: the number of registers available for core-local and socket-wide performance monitoring counters, as well as the physical topology of the server, which is determined by two key components: the number of physical CPU sockets and GPUs.

We will first examine how the three factors influence scalability by using our Intel Icelake A40 server as an example. This server features a single-socket multicore CPU and two A40 GPUs. Likwid categorizes performance events on this platform into eleven performance monitoring counter groups, specifically {CBOX,..., WBOX}. There are four core-local and seven socket-wide performance monitoring counter groups. Each group is assigned a specific number of registers that allow for the simultaneous collection of performance event counts coresponding to that group during a single application run.

For this server platform, Likwid provides four registers to store performance event counts related to the PCIe socket-wide counter group, known as PBOX. In contrast, it offers 120 registers for the Last Level Cache (LLC) socket-wide counter group, referred to as CBOX. Additionally, in the core-local performance monitoring counter groups, performance event counts are recorded for each core, whereas in the socket-wide performance monitoring counter groups, the counts are aggregated for the entire socket.

Based on our methodology applied to three hybrid server platforms, we find that a CPU computation software power meter uses performance events sourced from core-local Performance Monitoring Counter (PMC) groups. In contrast, a data transfer software power meter relies on performance events from each socket-wide performance monitoring counter group. This difference is due to the fact that data transfer power meters incorporate performance events from socket-wide performance monitoring counter groups, which capture off-core chip traffic.

So, the four data transfer power meters (which monitor communication between the host CPU and the two A40 GPUs in both directions) designed for the Intel Icelake A40 server will exhaust all the registers allocated to the PBOX performance monitoring counter group because this server is a single-socket system. This issue means that our methodology does not scale efficiently if a single-socket CPU server has more than two GPUs. Specifically, if a single-socket host CPU connects to an additional GPU, it will require two more events (one for each data transfer direction) from the PBOX group. In total, this would necessitate six counters from the PBOX group; however, this server only has four available. As a result, the power meters will be unable to maintain the desired concurrency property.

Consider the scenario where a server has a ratio of the number of physical CPU sockets to the number of GPUs that is greater than or equal to 0.5. In this case, our methodology will scale effectively because the number of registers available for a socket-wide Performance Monitoring Counter (PMC) group will be the product of the number of sockets and the number of registers in that group. Each register provides a separate socket-wide count.

For example, take a dual-socket multicore CPU server that hosts four GPUs. In this setup, the PBOX performance monitoring counter group will provide eight socket-wide counts (four registers, with each register offering two socket-wide counts). These eight counts can be utilized by the eight power meters dedicated to the data transfers between the CPU and GPUs, where there are two data transfers for each CPU-GPU pair. Therefore, the scalability of our methodology is not a concern for hybrid servers with a favorable ratio of CPU sockets to GPUs.

TABLE I
CONCURRENT AND ORTHOGONAL PERFORMANCE EVENT SETS FOR CPU COMPUTATIONS AND DATA TRANSFER DIRECTIONS IN THE ICELAKE A40 GPU SERVER CONTAINING A SINGLE-SOCKET ICELAKE MULTICORE CPU AND TWO NVIDIA A40 GPUS

CPU Computation	
Groups	**Performance Events**
CBOX	READ_NO_CREDITS_ANY
PMC	FP_ARITH_INST_RETIRED_SCALAR_DOUBLE
M2M	DISTRESS_ASSERTED_VERT
TCBOX	REQ_FROM_PCIE_CMPL_REQ_OWN
UBOX	M2U_MISC1_RXC_CYCLES_NE_CBO_NCB
WBOX	FREQ_TRANS_CYCLES
Data Transfer CPU $\Longrightarrow$ A40 GPU1	
Groups	**Performance Events**
CBOX	DIRECT_GO_OPC_EXTCMP
PMC	FP_ARITH_INST_RETIRED_SCALAR_SINGLE
M2M	TXR_HORZ_ADS_USED_AD_UNCRD
PBOX	IIO_CREDITS_ACQUIRED_DRS_ANY
TCBOX	INBOUND_ARB_REQ_WR
Data Transfer A40 GPU1 $\Longrightarrow$ CPU	
Groups	**Performance Events**
CBOX	RXC_ISMQ0_REJECT_BL_RSP_VN0
PMC	MEM_LOAD_RETIRED_L3_HIT
M2M	VERT_RING_AK_IN_USE_DN_EVEN
PBOX	TXC_INSERTS_AD_ANY
TCBOX	NUM_REQ_OF_CPU_BY_TGT_MEM
Data Transfer CPU $\Longrightarrow$ A40 GPU2	
Groups	**Performance Events**
CBOX	REQUESTS_INVITOE
PMC	MEM_LOAD_L3_HIT_RETIRED_XSNP_NONE
M2M	PREFCAM_DROP_REASONS_CH0_STOP_B2B
PBOX	RXC_INSERTS_ALL
TCBOX	INBOUND_ARB_REQ_FINAL_RD_WR
Data Transfer A40 GPU2 $\Longrightarrow$ CPU	
Groups	**Performance Events**
CBOX	RXC_ISMQ1_REJECT_ANY0
PMC	OFFCORE_REQUESTS_DEMAND_DATA_RD
M2M	RXR_OCCUPANCY_BL_UNCRD
PBOX	IIO_CREDITS_USED_DRS_1
TCBOX	INBOUND_ARB_REQ_REQ_OWN

C. Applying the Methodology to Select Performance Event Sets for Three Heterogeneous Hybrid Servers

Table I shows the concurrent and orthogonal performance event sets for the heterogeneous hybrid server, Icelake A40 GPU server, featuring a single-socket Intel multicore CPU and two Nvidia A40 GPUs. The parameters input to the methodology are: $\mathbb{A} = 5\%$, $\rho = 95\%$, $\epsilon = 2.5\%$, and $\tau = 0.025$.

Section V.A in the supplemental lists the sets of applications. Section V.B contains the concurrent and orthogonal performance event sets for the two hybrid servers, *Haswell k40c GPU server* and *Skylake P100 GPU server*.

D. Software Power Meters for Computation and Communication Activities

We develop software power meters using linear dynamic energy predictive models that leverage normalized performance events as predictor variables. Based on the practical applications of the theory of energy predictive models in computing [13], our models maintain a zero intercept and non-negative regression coefficients. We utilize the Huber Regressor from the Python's Scikit-learn package to construct these models. The Huber Regressor is a robust regression algorithm that is less sensitive to outliers compared to ordinary least squares (OLS) linear regression.

To train and test these models, we split the datasets into 70% for training and 30% for testing. These records are obtained by executing the applications that are used to select the performance event sets.

The software power meters for the Icelake A40 GPU server are outlined in Table II. The average prediction errors of these power meters are as follows: 1% for CPU computations and 2% for the four data transfer software power meters between the host CPU and the two A40 GPU servers. The software power meters for GPU computations utilize NVML and their average prediction error is 5%.

In Section V.A of the supplemental material, you can find the set of applications used for training and testing. Section V.C provides details on the software power meters (linear energy predictive models) and the average prediction errors for the other two servers: the Haswell k40c GPU server and the Skylake P100 GPU server.

TABLE II
SOFTWARE POWER METERS IMPLEMENTING THE LINEAR DYNAMIC ENERGY PREDICTIVE MODELS FOR ICELAKE A40 GPU SERVER

Model Description	Linear Regression Model
CPU computations	6.73E-05 × FP_ARITH_INST_RETIRED_SCALAR_DOUBLE + 6.32E-07 × READ_NO_CREDITS_ANY + 2.23E-07 × DISTRESS_ASSERTED_VERT + 4.863E-05 × REQ_FROM_PCIE_CMPL_REQ_OWN + 0.00E00 × M2U_MISC1_RXC_CYCLES_NE_CBO_NCB + 3.24E-07 × FREQ_TRANS_CYCLES
CPU → A40 GPU1	4.60E-07 × DIRECT_GO_OPC_EXTCMP + 8.14E-08 × FP_ARITH_INST_RETIRED_SCALAR_SINGLE + 1.28E-08 × TXR_HORZ_ADS_USED_AD_UNCRD + 6.40E-11 × IIO_CREDITS_ACQUIRED_DRS_ANY + 7.13E-07 × INBOUND_ARB_REQ_WR
A40 GPU1 → CPU	2.24E-08 × RXC_ISMQ0_REJECT_BL_RSP_VN0 + 3.93E-07 × MEM_LOAD_RETIRED_L3_HIT + 8.16E-08 × VERT_RING_AK_IN_USE_DN_EVEN + 3.79E-12 × TXC_INSERTS_AD_ANY + 4.66E-07 × NUM_REQ_OF_CPU_BY_TGT_MEM
CPU → A40 GPU2	3.10E-07 × REQUESTS_INVITOE + 1.69E-08 × MEM_LOAD_L3_HIT_RETIRED_XSNP_NONE + 4.76E-08 × PREFCAM_DROP_REASONS_CH0_STOP_B2B + 1.69E-11 × RXC_INSERTS_ALL + 3.93E-07 × INBOUND_ARB_REQ_FINAL_RD_WR
A40 GPU2 → CPU	3.24E-08 × RXC_ISMQ1_REJECT_ANY0 + 1.11E-08 × OFFCORE_REQUESTS_DEMAND_DATA_RD + 4.66E-08 × RXR_OCCUPANCY_BL_UNCRD + 1.66E-11 × IIO_CREDITS_USED_DRS_1 + 5.41E-07 × INBOUND_ARB_REQ_REQ_OWN

V. SOFTWARE POWER METER API

We propose a software power meter API that enables the runtime integration of software power meters. These power meters implement our proposed linear energy predictive models to estimate the energy consumption of both computation and communication activities.

The API function *hcl_create_power_cmeter()* creates a software power meter for computations on a device represented by a 2-tuple, {*dtype*, *devno*}, where *dtype* represents the device type (CPU or GPU) and *devno* signifies the device number within a class of devices given by the device type. Upon successful creation of the power meter, the function returns a handle to the power meter in the variable, *pmeter*. The argument *peventnames* is an array of size *npe*, which contains the names of the performance events. The argument, *modelf*, is a user-defined model function that takes as input an array of performance event values and returns the dynamic energy consumption of the computations.

```
typedef double ( *hcl_energy_model_func )(
  int npe, const double* peventarray);
int hcl_create_power_cmeter(
  hcl_device_type dtype, int devno,
  int npe, const char** peventnames,
  const hcl_energy_model_func* modelf,
  hcl_power_meter* pmeter
);
```

The API function *hcl_create_power_dmeter()* is used to create a software power meter for monitoring data transfer between CPU and GPU. The direction of the data transfer is specified by the *direction* parameter, which accepts two values: *FORWARD* and *BACKWARD*. *FORWARD* indicates data transfers from the host CPU to the GPUs, while *BACKWARD* indicates transfers from the GPUs back to the host CPU. Additionally, the *modelf* argument is a user-defined function that calculates the dynamic energy consumption of the data transfers based on performance event values. Upon successful creation of the power meter, the output argument *pmeter* provides a handle for accessing the power meter. It is important to note that only one power meter can be used for each direction to estimate the dynamic energy consumption of data transfers occurring in that direction within a specific code region.

```
int hcl_create_power_dmeter(
  int direction, int npe,
  const char** peventnames,
  const hcl_energy_model_func* modelf,
  hcl_power_meter* pmeter
);
```

The API function *hcl_start_power_meters()* starts a set of power meters of size *nmeters* represented by an array *meterids* for energy estimation.

```
int hcl_start_power_meters(int nmeters,
  hcl_power_meter* meterids);
```

Conversely, the API function *hcl_stop_power_meters()* halts the operation of the meters represented by *meterids*. Additionally, the output array *energyestimates* contains the dynamic energy estimates where the array element $energyestimates[i]$ contains the estimate for meter i. Two more arrays, $times$ and $avgpowerestimates$, contain the times and average dynamic energy estimates, respectively.

```
int hcl_stop_power_meters(int nmeters,
  hcl_power_meter* meterids,
  double *times, double *avgpowerestimates,
  double *energyestimates);
```

The API functions, *hcl_start_power_meters* and *hcl_stop_power_meters*, provide the flexibility to selectively start and stop one or more power meters based on the given set of power meter IDs. To estimate the energy consumption of a specific code region, the API function *hcl_start_power_meters()* should be called at the beginning of the region, and the API function *hcl_stop_power_meters()* should be called at the end. These API functions require an array of meter IDs that represent the power meters associated with the activities involved in executing the code region.

The API function *hcl_free_power_meters()* below releases the resources associated with the set of meters given by *meterids*.

All the API functions mentioned above are not thread-safe, which means they must be called by a single CPU thread. However, the functions for creating power meters (*hcl_create_power_cmeter* and *hcl_create_power_dmeter*) and

```
int hcl_free_power_meters(int nmeters,
  hcl_power_meter* meterids);
```

for destroying them (*hcl_free_power_meters*) can be called by different CPU threads. Additionally, the start and stop API functions, *hcl_start_power_meters* and *hcl_stop_power_meters*, may also be called by different CPU threads.

A. Instrumentation of a Parallel Hybrid Matrix Multiplication Application Using Software Power Meters

We demonstrate the functionality of our software power meter API through a parallel hybrid matrix multiplication application running on the Icelake A40 GPU server.

There are several methods to instrument parallel hybrid applications with our software power meters. In the most general case, we can measure specific activities during designated phases of the application. In this illustration, we present an approach that selectively starts and stops each power meter for individual activities, such as computation or data transfer. This method allows us to identify the sections of code or activities that consume the most energy. These sections then become targets for energy optimization, although exploring that aspect is beyond the scope of this work.

Fig. 3(b) illustrates the parallel hybrid matrix multiplication application (HDGEMM), which computes the matrix product C += $A \times B$ of two dense square matrices, A and B, each of size $N \times N$.

The application consists of three software components: the CPU, A40 GPU_1, and A40 GPU_2. All components share the matrix B. The matrices A and C are horizontally partitioned, with each software component assigned several contiguous rows of A and C, as specified by input parameters to the application. The CPU is responsible for N_1 rows of A and C. The A40 GPU_1 component is assigned N_2 rows of A and C, while the A40 GPU_2 component takes on the remaining rows, specifically $N - N_1 - N_2$ of A and C. Each software component i (where $i \in \{1, 2, 3\}$) computes the matrix product C_i += $A_i \times B$.

The application consists of three main stages. The first stage involves transferring data: A_2, B, and C_2 from the host CPU to A40 GPU_1, and A_3, B, and C_3 from the host CPU to A40 GPU_2. The second stage is dedicated to performing local computations within the software components. Finally, the third stage entails transferring the resulting matrices, C_2 and C_3, from A40 GPU_2 and A40 GPU_1 back to the host CPU.

Figs. 1 and 2 illustrate the use of our proposed software power meter API functions to estimate the energy consumption associated with computations and data transfers.

The application-specific routines - *cpudgemm()*, *gpudgemm1()*, and *gpudgemm2()* - are responsible for device-specific computations. Additionally, the routines *gpudt1()* and *gpudt2()* handle data transfers between the host CPU and A40 GPU_1, as well as between the host CPU and A40 GPU_2.

```
1  #include <edm.h>
2  int main(int argc, char *argv[]) {
3      double *A, *B, *C;
4      enum metertype { CPU = 0, GPU1, GPU2, H2D1, D2H1,
           H2D2, D2H2 };
5      int NS = 7, nevents= 6;
6      char* cpuevents[] = {
7         ”CBOX:READ_NO_CREDITS_ANY”,
8         ”PMC:FP_ARITH_INST_RETIRED_SCALAR_DOUBLE”,
9         ”M2M:DISTRESS_ASSERTED_VERT”,
10        ”TCBOX:REQ_FROM_PCIE_CMPL_REQ_OWN”,
11        ”UBOX:M2U_MISC1_RXC_CYCLES_NE_CBO_NCB”,
12        ”WBOX:FREQ_TRANS_CYCLES” };
13     char* h2d1events[] = {
14        ”CBOX:DIRECT_GO_OPC_EXTCMP”,
15        ”PMC:FP_ARITH_INST_RETIRED_SCALAR_SINGLE”,
16        ”M2M:TXR_HORZ_ADS_USED_AD_UNCRD”,
17        ”PBOX:IIO_CREDITS_ACQUIRED_DRS_ANY”,
18        ”TCBOX:INBOUND_ARB_REQ_WR”,
19        ”IBOX:CACHE_TOTAL_OCCUPANCY_ANY”
20     };
21     char* d2h1events[] = {
22        ”CBOX:RXC_ISMQ0_REJECT_BL_RSP_VN0”,
23        ”PMC:MEM_LOAD_RETIRED_L3_HIT”,
24        ”M2M:VERT_RING_AK_IN_USE_DN_EVEN”,
25        ”PBOX:TXC_INSERTS_AD_ANY”,
26        ”TCBOX:NUM_REQ_OF_CPU_BY_TGT_MEM”,
27        ”IBOX:MISC0_FAST_REQ”
28     };
29     // Similar event arrays for the
30     // data transfers between CPU and GPU2
31     hcl_power_meter meterids[NS];
32     hcl_create_power_cmeter(HCL_CPU, 0,
33        nevents, cpuevents, cpumodelf, &meterids[CPU]);
34     hcl_create_power_cmeter(HCL_GPU, 0,
35        0, NULL, hcl_nvml_modelf, &meterids[GPU1]);
36     hcl_create_power_cmeter(HCL_GPU, 1,
37        0, NULL, hcl_nvml_modelf, &meterids[GPU2]);
38     hcl_create_power_dmeter(FORWARD, nevents, h2d1events,
39        h2d1modelf, &meterids[H2D1]);
40     hcl_create_power_dmeter(BACKWARD, nevents, d2h1events,
41        d2h1modelf, &meterids[D2H1]);
42     hcl_create_power_dmeter(FORWARD, nevents, d2h2events,
43        h2d2modelf, &meterids[H2D2]);
44     hcl_create_power_dmeter(BACKWARD, nevents, d2h2events,
45        d2h2modelf, &meterids[D2H2]);
46 }
```

Fig. 1. Illustration of the software power meter API in a parallel hybrid matrix multiplication application executing on a server comprising a Intel multicore CPU and two Nvidia A40 GPUs. Seven energy power meters are deployed in this application. Three power meters for computations on the CPU, A40 GPU_1, and A40 GPU_2 ($meterids[CPU]$, $meterids[GPU1]$, and $meterids[GPU2]$). Two for data transfers between the host CPU and the A40 GPU_1 and the A40 GPU_1 and the host CPU. Finally, two for data transfers between the host CPU and the A40 GPU_2 and the A40 GPU_2 and the host CPU.

We first describe the code shown in Fig. 1. Lines 4-28 contain arrays that list the relevant performance events for computations on the multicore CPU and for data transfers between the host CPU and the A40 GPU devices.

In line 31, the array declaration containing the power meter identifiers is defined. Lines 32-37 include API function calls that create software power meters for computations on the CPU and two GPUs using the function call, *hcl_create_power_cmeter*. The power meter identified as $meterids[CPU]$ measures the energy consumption of computations on the CPU. Meanwhile, the power meters $meterids[GPU1]$ and $meterids[GPU2]$ estimate the energy consumption for computations on A40 GPU_1 and A40 GPU_2, respectively.

```
1  #include <edm.h>
2  int main(int argc, char *argv[]) {
3      double times[NS], avgpowers[NS], energyestimates[NS];
4  #pragma omp parallel sections num_threads(3)
5  {
6  #pragma omp section
7      {
8          hcl_start_power_meters(1, &meterids[CPU]);
9          cpudgemm(N1, N, N, A, B, C);
10         hcl_stop_power_meters(1, &meterids[CPU],
               &energyestimates[CPU]);
11     }
12 #pragma omp section
13     {
14         hcl_start_power_meters(1, &meterids[H2D1]);
15         gpudt1(N2, N, N, &A[N1*N], B, &C[N1*N]);
16         hcl_stop_power_meters(1, &meterids[H2D1],
               &times[H2D1], &avgpowers[H2D1],
               &energyestimates[H2D1]);
17         hcl_start_power_meters(1, &meterids[GPU1]);
18         hcl_start_power_meters(1, &meterids[D2H1]);
19         gpudgemm1(N2, N, N);
20         hcl_stop_power_meters(1, &meterids[D2H1],
               &times[D2H1], &avgpowers[D2H1],
               &energyestimates[D2H1]);
21         hcl_stop_power_meters(1, &meterids[GPU1],
               &times[GPU1], &avgpowers[GPU1],
               &energyestimates[GPU1]);
22     }
23 #pragma omp section
24     {
25         hcl_start_power_meters(1, &meterids[H2D2]);
26         gpudt2(N1+N2, N, N, &A[(N1+N2)*N], B,
               &C[(N1+N2)*N]);
27         hcl_stop_power_meters(1, &meterids[H2D2],
               &times[H2D2], &avgpowers[H2D2],
               &energyestimates[H2D2]);
28         hcl_start_power_meters(1, &meterids[GPU2]);
29         hcl_start_power_meters(1, &meterids[D2H2]);
30         gpudgemm2(N1+N2, N, N);
31         hcl_stop_power_meters(1, &meterids[D2H2],
               &times[D2H2], &avgpowers[D2H2],
               &energyestimates[D2H2]);
32         hcl_stop_power_meters(1, &meterids[GPU2],
               &times[GPU2], &avgpowers[GPU2],
               &energyestimates[GPU2]);
33     }
34 }
35     hcl_free_power_meters(NS, meterids);
36     exit(EXIT_SUCCESS);
37 }
```

Fig. 2. The matrix multiplication application contains three OpenMP threads that execute the software components (kernels) in parallel. The software power meters are selectively started and stopped for individual computation and communication activities.

Lines 38-45 detail the API function calls that create data transfer power meters using the function call, *hcl_create_power_dmeter*. The power meters $meterids[H2D1]$ and $meterids[D2H1]$ estimate the energy consumption for data transfers between the host CPU and A40 GPU_1. Similarly, $meterids[H2D2]$ and $meterids[D2H2]$ estimate the energy consumption for data transfers between the host CPU and A40 GPU_2. It's important to note that the power meters for data transfers are distinct for the two directions: from the host CPU to the GPU and from the GPU back to the host CPU.

The model function $cpumodelf$ estimates the energy consumption of computations based on the performance events recorded in the array, *cpuevents*. The pre-defined function hcl_nvml_modelf utilizes Nvidia's NVML interface to estimate the energy consumed during computations on the A40 GPU. Additionally, the functions $h2d1modelf$ and $d2h1modelf$ estimate the energy consumption associated with data transfers between the host CPU and A40 GPU_1, using performance events captured in the arrays *h2d1events* and *d2h1events*, respectively. There are also two corresponding functions for data transfers between the host CPU and A40 GPU_2.

Fig. 2 illustrates the remainder of the code. Lines 3-4 show the declaration of the arrays that will store all the times, average power and dynamic energy estimates. Line 4 initiates a parallel OpenMP region, which consists of three threads executing the software components concurrently. In this line, we start the energy power meter for CPU computations using the API function call, *hcl_start_power_meters*. Line 9 includes the execution of the CPU software component. Finally, Line 10 stops the power meter and retrieves the dynamic energy estimate of the CPU computations, storing it in the array element *energyestimates[CPU]*.

In Line 14, we start the power meter for the data transfer from the host CPU to A40 GPU_1. Line 15 triggers the data transfer operations. In Line 16, we stop the power meter and obtain the dynamic energy estimate of the CPU computations for the array element, *energyestimates[H2D1]*. Line 19 includes computations performed on A40 GPU_1 and transfers the results back to the host CPU. It is important to note that the routine *gpudgemm1()* calls a CUDA library function that combines both the computations and the data transfer in a single invocation. Consequently, we start two power meters in Lines 17-18: one for the computations and the other for the data transfer from A40 GPU_1 to the host CPU. In Lines 20-21, we stop the power meters and retrieve the dynamic energy estimates in *energyestimates[GPU1]* and *energyestimates[D2H1]*.

Lines 23-33 present the code for the execution of the A40 GPU_2 software component. In Line 35, we release all the resources associated with the power meters using the API function *hcl_free_power_meters*. The total dynamic energy consumption of the program is calculated as the sum of the dynamic energy estimates stored in the array *energyestimates*.

VI. Experimental Results

We demonstrate the accuracy of our proposed software power meters on three heterogeneous hybrid servers using three highly optimized parallel hybrid scientific applications which are matrix addition (HDGEADD), matrix multiplication (HDGEMM), and 2D fast Fourier transform (HFFT2D). The specifications of these three servers are given in the supplemental (Section II).

While we validate our approach with three regular applications, it is important to note that our platform-level software power meters can accurately estimate the energy consumption of parallel hybrid programs, irrespective of whether these programs are regular or irregular in nature. The proposed software power meters utilize performance events selected through a methodology that incorporates a diverse range of applications, which includes both regular and irregular types. Additionally, these software power meters are based on linear energy predictive models that have been robustly trained using a wide variety of applications. Details regarding the benchmark suites employed for this methodology and model training can be found in the supplemental materials.

In this section, we present results for the Icelake A40 GPU server. This server contains a single-socket Intel Icelake multicore CPU and two Nvidia A40 GPUs. The gcc, cuda, and Intel OneAPI versions on this server are 12.3.0, 12.0 and 2023.0.0 Build 20221201. Results for the other two server platforms are also included in the supplemental materials. For all the servers, the multicore CPUs employ the default CPU power governor, *ondemand*, which dynamically adjusts CPU frequency based on CPU utilization.

Each hybrid application running on the Icelake A40 GPU server consists of three software components: CPU, A40 GPU_1, and A40 GPU_2. The CPU_1 component utilizes 22 out of the 24 available CPU cores. The A40 GPU_1 component represents the first Nvidia A40 GPU, with a dedicated PCI-E link connecting a host CPU core to this GPU. Similarly, A40 GPU_2 represents the second Nvidia A40 GPU, also connected via a dedicated PCI-E link to another host CPU core.

In each hybrid application, the data transfers between host CPU cores and GPUs in the same direction occur simultaneously. These activities are independently powered. However, data transfers that occur between the same pair of a host CPU core and GPU in opposite directions are not independently powered and do not run in parallel. Specifically, a data transfer from the GPU to the CPU always logically follows a data transfer from the CPU to the GPU. Therefore, in all of our applications, whenever activities occur in parallel, they are all independently powered.

Additionally, every hybrid application is equipped with our proposed software power meters to monitor computations and data transfers using the software power meter API. Seven energy power meters are integrated into each application: three power meters monitor computations in the CPU, A40 GPU_1, and A40 GPU_2 components, while four power meters track data transfers between the host CPU and the A40 GPU components.

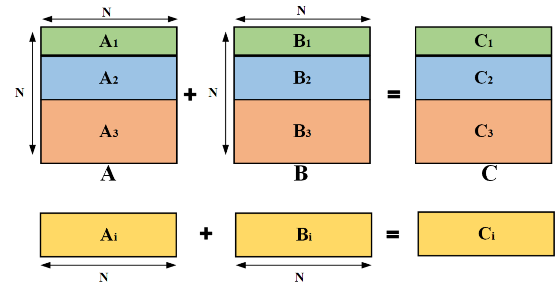
The experiments are repeated for a data point obtained using the ground-truth method until the sample mean falls within the 95% confidence interval, achieving a precision (or standard deviation of the mean) of 2.5%. To accomplish this, we use Student's t-test, assuming that the individual observations are independent and that the population follows a normal distribution. We validate these assumptions using Shapiro-Wilk Test.

For data points obtained from software power meters, we perform three repetitions and calculate the average prediction error, which is the average of the prediction errors from the three runs. Finally, we report the overall average prediction error of a software power meter, calculated by averaging the average prediction errors obtained across all data points.

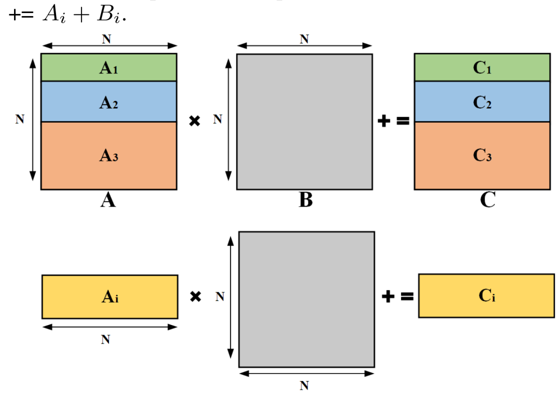
A. Parallel Hybrid Applications: Description

Matrix Addition (HDGEADD): The parallel hybrid matrix addition application, referred to as HDGEADD, computes the matrix addition (C += $A + B$) for two dense square matrices, A and B, both of size $N \times N$. Fig. 3(a) illustrates the application. The matrices A, B and C are horizontally partitioned so that each software component is assigned several contiguous rows of A, B and C provided as input parameters to the application.

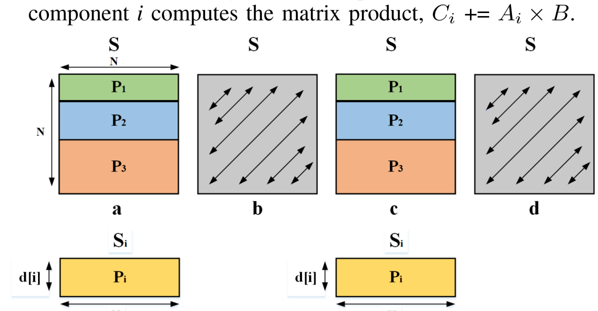
Each software component i computes the matrix addition, C_i += $A_i + B_i$. The application comprises three main stages:



(a) Parallel matrix addition application (HDGEADD) computing the matrix addition (C += $A + B$) of two dense square matrices A and B of size $N \times N$. The application comprises three software components executed in parallel where the component i computes the matrix addition, C_i += $A_i + B_i$.



(b) Matrix partitioning between the three software components in parallel hybrid matrix multiplication application (HDGEMM) computing the matrix product ($C+ = A \times B$) of two dense square matrices A and B of size $N \times N$. The matrix B is shared by all the components. Each software component i computes the matrix product, C_i += $A_i \times B$.



(c) Parallel hybrid 2D fast Fourier Transform application (HFFT2D) computing the 2D-FFT of a signal matrix S of size $N \times N$. The matrix S is partitioned such that each software component is assigned several contiguous rows of S. The 2D FFT is accomplished in four steps: (a) Computing N 1D FFTs. (b) Transpose of the signal matrix. Steps (c) and (d) repeat steps (a) and (b), respectively.

Fig. 3. Description of the three parallel hybrid applications.

1) *Data Transfer Stage:* In the first stage, the matrices A_2, B_2, and C_2 are transferred from the host CPU to A40 GPU_1, while A_3, B_3, and C_3 are transferred to A40 GPU_2.
2) *Computation Stage:* The second stage involves local computations in the software components. For the component that utilizes the multicore CPU, the computations are carried out using the Intel Math Kernel Library (MKL) routine (DGEADD). In the components that use the A40 GPU, the computations are performed using the CUBLAS library routine.
3) *Result Transfer Stage:* The third stage includes transferring the result matrices C_2 and C_3 from A40 GPU_2 and A40 GPU_1 back to the host CPU.

Matrix Multiplication (HDGEMM): The parallel hybrid matrix multiplication application (HDGEMM) computes the matrix product ($C \mathrel{+}= A \times B$) for two dense square matrices A and B, each of size $N \times N$. Fig. 3(b) illustrates the application. Further details can be found in Section V-A. The computations in the CPU software component are performed using the Intel MKL DGEMM library routine, while the computations in the software components that involve the A40 GPUs are performed using the CUBLAS DGEMM library routine.

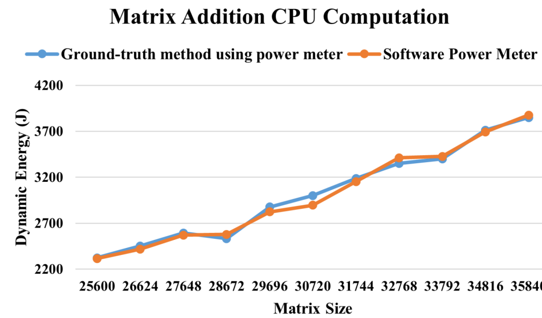
2D Fast Fourier Transform (HFFT2D): The parallel hybrid 2D fast Fourier Transform application (HFFT2D) computes the 2D-FFT of a signal matrix S of size $N \times N$. Fig. 3(c) illustrates the application. The application comprises three software components executed in parallel (one CPU component, one GPU_1 component, and one GPU_2 component). The matrix S is partitioned so that each component receives several contiguous rows of S provided as input.

The 2D FFT is accomplished in four steps: (1) Computing 1D FFTs on N rows of the signal matrix N; (2) Transposing the signal matrix; (3) Repeating step (1); and (4) Repeating step (2).

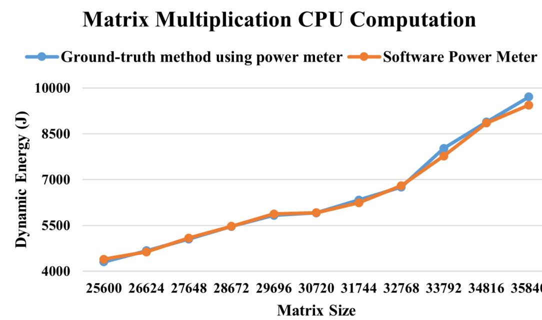
The application consists of eight stages. The first stage involves transfering data P_2 from the host CPU to A40 GPU_1 and P_3 from the host CPU to A40 GPU_2. The second stage involves local computations in the software components. The computations in the software component involving the multicore CPU are performed using the Intel MKL FFT library routine. The computations in the software components involving the A40 GPUs are performed using the CUFFT library routine. The third stage entails transfering the result matrices P_2 and P_3 from A40 GPU_2 and A40 GPU_1 to the host CPU. The fourth stage involves the transposing the matrix S on the CPU. These four stages are repeated. It is important to note that we do not consider the energy consumption of the matrix transposition on the CPU, as it is negligible.

B. Results and Discussion

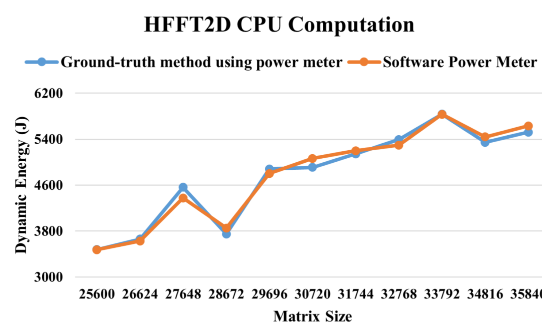
Fig. 4 illustrates the energy predictions by the software power meter for CPU computations for the three applications. The x-axis represents the matrix size N, ranging from 25600×25600 to 35840×35840, while the y-axis indicates the dynamic energy consumption in joules. Each plot consists of two curves: the blue curve represents the actual dynamic energy measured using ground-truth data from external power meters, and the orange curve illustrates the predicted dynamic energy consumption by the CPU computation software power meter. The average prediction error is 2.5%. Additionally, the CPU computation software power meter reports negligible energy consumption during phases when the CPU computation activities are absent.



(a)



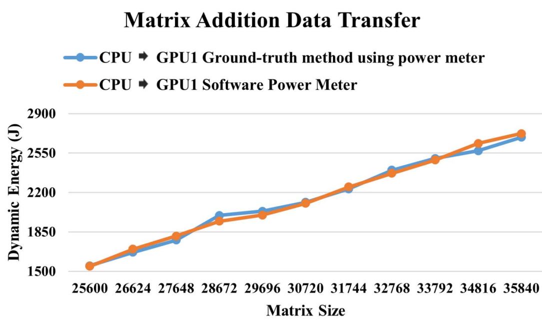
(b)



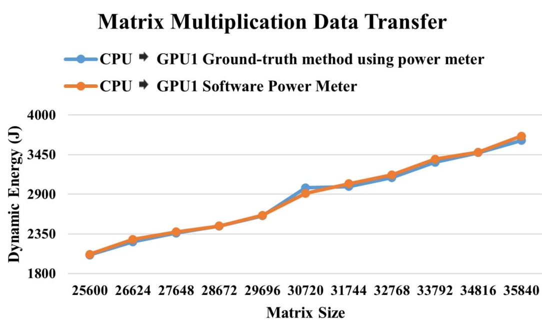
(c)

Fig. 4. The plots compare the dynamic energy consumptions of CPU computation component estimated by the CPU software power meter (indicated in orange) and the ground truth measurement method (shown in blue) for the three parallel hybrid programs executed on the Icelake A40 GPU server. The orange data points represent averages from three repetitions, with a standard error of $\pm 1\%$ of the average. The blue points are derived using Student's t-test, with a standard error of $\pm 2.5\%$ of the mean. These standard errors apply to all the orange and blue data points shown in this work. Note that we have not included error bars in the plots, as they are quite small.

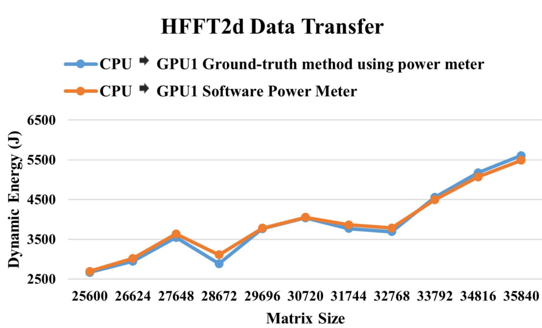
Fig. 5 illustrates the energy predictions by the software power meters for the data transfers from the host CPU to A40 GPU_1 for the three applications. Each plot again features two curves: the blue curve shows the actual dynamic energy measured using ground-truth data from external power meters, while the orange curve displays the predicted energy consumption by the data transfer software power meter. The average prediction error



(a)

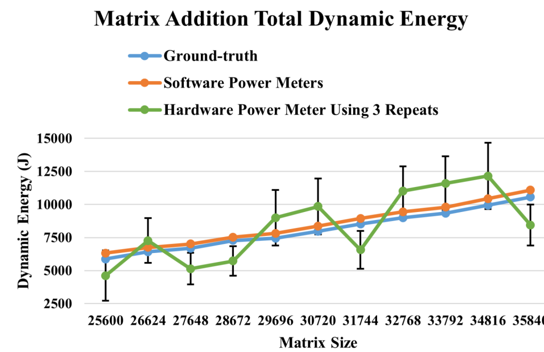


(b)

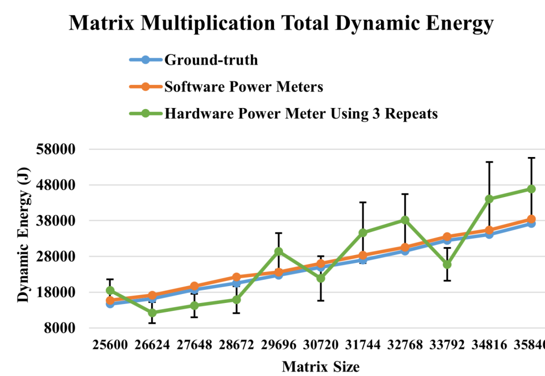


(c)

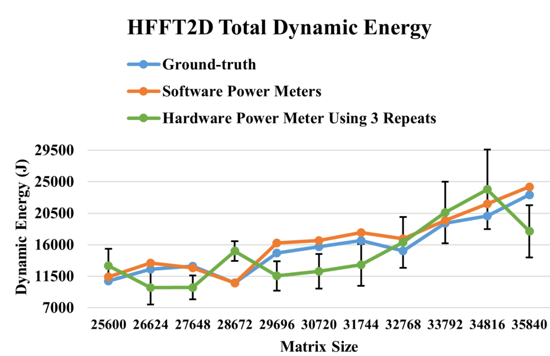
Fig. 5. The plots compare the dynamic energy consumption of data transfers from the CPU to A40 GPU_1 estimated by the data transfer software power meters (indicated in orange) and the ground truth measurement method (shown in blue) for the three parallel hybrid applications executed on the Icelake A40 GPU server.



(a)



(b)



(c)

Fig. 6. The plots demonstrate the total dynamic energy consumption recorded using three different approaches, which include software power meters (indicated in orange), the ground truth measurement method (shown in blue), and a method utilizing hardware power meters that employs the average of three repetitions for each data point (represented in green). Each green point is shown by a vertical error bar with the average value in the middle. The standard error of the green points ranges from ±16% to ±25% of the average.

remains at 2.5%. Moreover, the software power meter for a specific data transfer activity records negligible dynamic energy consumption for phases when this activity is absent.

Fig. 6 illustrates the total dynamic energy consumption estimated by our software power meters, represented in orange. This is compared to the measurements obtained through the ground truth method, shown in blue, and an approach that uses the average of three repetitions with hardware power meters across the three parallel hybrid programs. The total dynamic energy consumption estimated by the software power meters is calculated by summing the dynamic energy estimates from all the power meters. The average prediction error when using hardware power meters with three repetitions is 25% compared to the ground truth measurements. In contrast, the average prediction errors of the software power meters against the ground truth for the three applications - HDGEADD, HDGEMM, and HFFT2D - are 3%, 3%, and 4%, respectively. However, these percentages are affected by the higher average prediction error of 5% from software power meters used for GPU computations. These meters rely on power readings from the GPUs' on-chip power sensors.

In Section VIII.A of the supplemental material, we present the experimental results that demonstrate the energy predictions made by our software power meter for computations on the A40 GPU_1 used in three different applications executed on the

Icelake A40 GPU server. The average prediction error for these computations is 5%. These results also apply to computations on the A40 GPU_2. Additionally, in Section VIII.B of the supplemental material, we provide energy predictions from the software power meters regarding data transfers between the host CPU and the A40 GPU units in both directions for the three applications. Here, the average prediction error for dynamic energy consumption is 2.5%.

In Section VIII.C, we present experimental results showing energy predictions made by our software power meters for two other hybrid servers: the Haswell K40c GPU server and the Skylake P100 GPU server. For the Haswell K40c GPU server, the observed average prediction errors for total dynamic energy consumption are 3%, 4%, and 4%. In contrast, the errors for the Skylake P100 GPU server are 2%, 3%, and 4%.

Furthermore, our experiments show that using current energy predictive models proposed in [2] to estimate the energy consumption of computations on multicore CPU processors, as well as the energy used for data transfers between a CPU and GPU, can lead to considerable prediction errors. When applied to our parallel hybrid programs that perform these activities simultaneously, these errors can reach as high as 50%.

The experimental results confirm that our proposed software power meters are concurrent and orthogonal across three different hybrid CPU-GPU server platforms. Additionally, the power meters demonstrate high prediction accuracy. Importantly, for each platform, the same power meters can effectively profile the dynamic energy consumption of all three hybrid programs, as they are designed to be platform-level rather than application-specific. Thus, no modifications to the power meters are necessary to capture any missing application-specific energy-consuming activities.

VII. Literature Review

This section presents a brief overview of the research literature related to our work.

A. Energy Predictive Models for Communications

Gamell et al. [23] propose constant bandwidth energy models for the network and main memory. The energy model estimates the energy consumption of m bytes between a pair of processes via a communication link with a peak theoretical bandwidth of B bytes as $(m/B) \times P_{tdp} \times t$, where P_{tdp} is the theoretical peak power (specified in the datasheet) of the communication link and t is the time of the data transfer. Khaleghzadeh et al. [24] consider the energy cost of communications between two heterogeneous hybrid servers connected by a Gigabit Ethernet network switch, which consumes a constant power irrespective of the amount of data transferred. Malik et al. [25] propose an application-specific constant bandwidth model predicting the energy of data transfer between a CPU and GPU for a heterogeneous hybrid matrix multiplication application.

Moreover, existing energy models of data transfer between computing devices are simplistic functions of bandwidth and peak power of the communication link between the computing devices, which fail to capture the complexity of real-world scenarios [23], [25]. Niaz et al. [2] comprehensively study the energy consumption of data transfer between a host CPU and a GPU accelerator on heterogeneous hybrid servers using the three mainstream energy measurement methods: (a) System-level physical measurements based on external power meters (ground-truth), (b) Measurements using on-chip power sensors, and (c) Energy predictive models. The authors focus on the third approach and introduce a novel methodology for selecting a minimal subset of performance events that effectively represent all energy consumption activities during data transfers. Using these selected events for computations and communications, they develop accurate linear energy predictive models to reliably estimate energy consumption of concurrent computation and communication activities in three parallel hybrid applications.

B. On-Chip Energy Sensors

Hackenberg et al. [26] report systematic errors in RAPL energy counters and find that it is inclined towards certain types of workload and can give poor power predictions for others. Fahad et al. [1] present a comparative study of on-chip sensors against the ground truth. They demonstrate that using Intel RAPL in energy optimization may lead to significant energy losses. Experiments with a DGEMM matrix-multiplication hybrid application have shown that using RAPL for building energy profiles for energy optimization methods results in energy losses ranging from 37% to 84% (depending on matrix sizes) in comparison with the accurate ground-truth profiles.

Research works ([1], [27]) report poor accuracy and anomalies with sensors in older generations of Nvidia GPUs (Nvidia Tesla K20c and K40c). The latest Nvidia GPUs (for example, the A40 GPU processor generation and later) provide accurate on-chip power sensors whose readings can be obtained programmatically using the Nvidia Management Library (NVML) interface. However, these power sensors do not capture any data transfer activity on the GPU side.

C. Energy Predictive Models for CPU Computations

Software-based energy prediction models rely on various runtime performance-related variables for energy prediction. Among these, performance events and resource utilization serve as key model variables, primarily because of their strong positive high correlation with energy consumption. The research works such as [28], [29], [30], [31], and [32] focus on resource utilization like CPU, memory, disk, and network usage. On the other hand, works like [3], [4], [5], [6], [7], [8], [9], [10], and [11] are centered around performance events. However, performance events are often preferred over utilization variables due to their good prediction accuracy.

Economou et al. [33] developed a linear power prediction model that uses CPU, disk, and network utilization metrics, along with a performance event containing the off-chip memory access counts. Rivoire et al. [34] evaluated five real-time power models for full systems. Four of these models relied on resource utilization, while the fifth was based on the approach proposed by [33]. Their findings showed that the performance event-based model was the most accurate because it accounted for the key factors influencing dynamic power consumption in the system. Khokhriakhov et al. [35] introduced a qualitative linear dynamic energy model that uses CPU utilization and performance events from the PMC performance monitoring counter group to analyze and explain energy nonproportionality observed in their multicore CPU platforms.

D. Energy Predictive Models for GPU Computations

Energy predictive modelling for Nvidia GPUs exhibits limited attention compared to the plethora of research for CPU processors. One primary reason is that older generations of Nvidia GPUs (preceding Nvidia A40) are poorly instrumented for runtime energy modelling. Based on our experiments on these GPUs, many key events and metrics overflow for large problem sizes due to restrictive 32-bit integers being dedicated to storing the values. Hence, they were unsuitable for energy predictive modelling of parallel hybrid applications with large problem sizes. However, the latest generation GPUs, such as Nvidia A40, provide better energy instrumentation support to facilitate accurate runtime modelling of energy consumption.

Nvidia GPU processors provide CUPTI events and metrics, which are similar to performance events for multicore CPUs, that are employed in energy predictive models for GPU computations [14], [15], [16]. Abe et al. [36] propose energy models employing performance events for different generations of Nvidia GPUs. Adhinarayanan et al. [37] propose application-specific online statistical regression power models employing performance events.

Guerreiro et al. [38] propose linear energy predictive models for core and memory domains that utilize utilization variables obtained from CUPTI events and metrics on GPU devices. They report an average prediction error ranging from 2.5% to 8% for Tesla K40c GPUs. However, it is important to note that key events overflowed for the problem sizes used in the HPC workloads analyzed in this work with this particular GPU. Wang et al. [39] use CUPTI events and metrics in their machine learning energy predictive model, which is employed in the dynamic optimization of machine learning training workloads for performance and energy efficiency. They report an average prediction error of approximately 5% for their model. In contrast, our proposed software power meter for GPU computations employs NVML, as we determined it to be accurate within $\pm$ 5% of the actual values. Looking ahead, we plan to explore the development of more accurate performance event-based models for GPU computations in our future work.

VIII. Conclusion

Heterogeneous hybrid servers equipped with accelerators are commonly found in high-end digital platforms, such as high-performance computing (HPC) clusters, data centers, and cloud environments. However, conducting runtime energy profiling for parallel hybrid programs on these servers presents a challenge, as it requires accurate and efficient measurement of the dynamic energy consumed by different parallel computation and communication activities of the program.

Among the three main approaches for energy profiling - system-level physical measurements using external power meters, on-chip power sensors, and energy predictive models based on performance events - it is the third approach that shows the most promise. This is due to its cost-effectiveness and better runtime accuracy compared to the other two methods.

Nonetheless, existing energy predictive models that utilize performance events often lack two critical properties for effective runtime energy profiling of parallel hybrid programs on heterogeneous hybrid servers: *concurrency* and *orthogonality*.

In our work, we have defined these two essential properties and presented the motivation for addressing them. We proposed a methodology to develop platform-level software power meters that are both concurrent and orthogonal, enabling accurate runtime energy profiling of parallel hybrid programs on heterogeneous hybrid servers. We applied this methodology to create software power meters for three heterogeneous hybrid servers that combine Intel multicore CPUs and Nvidia GPUs from different generations.

Finally, we demonstrated the high accuracy and efficiency of our power meters by using them to estimate the dynamic energy consumption of both computation and communication activities in three parallel hybrid programs running on the three heterogeneous hybrid servers. The software implementation of software power meter API is available at [40].

Acknowledgment

The authors thank Jakub Kerlin from the School of Computer Science for developing the software power meter API implementation.

References

[1] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, “A comparative study of methods for measurement of energy of computing,” *Energies*, vol. 12, no. 11, 2019, Art. no. 2204.

[2] H. A. Niaz, R. R. Manumachu, and A. Lastovetsky, “Accurate and reliable energy measurement and modelling of data transfer between CPU and GPU in parallel applications on heterogeneous hybrid platforms,” *IEEE Trans. Comput.*, vol. 74, no. 3, pp. 1011–1024, Mar. 2025.

[3] F. Bellosa, “The benefits of event: Driven energy accounting in power-sensitive systems,” in *Proc. 9th Workshop ACM SIGOPS Eur. Workshop: Beyond PC: New Challenges Operating Syst.*, 2000, pp. 37–42.

[4] C. Isci and M. Martonosi, “Runtime power monitoring in high-end processors: Methodology and empirical data,” in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2003, pp. 93–104.

[5] T. Li and L. K. John, “Run-time modeling and estimation of operating system power consumption,” *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 160–171, Jun. 2003.

[6] B. C. Lee and D. M. Brooks, “Accurate and efficient regression modeling for microarchitectural performance and power prediction,” *SIGARCH Comput. Archit. News*, vol. 34, no. 5, pp. 185–194, Oct. 2006.

[7] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, “Decomposable and responsive power models for multicore processors using performance counters,” in *Proc. 24th ACM Int. Conf. Supercomputing*, 2010, pp. 147–158.

[8] W. L. Bircher and L. K. John, “Complete system power estimation using processor performance events,” *IEEE Trans. Comput.*, vol. 61, no. 4, pp. 563–577, Apr. 2012.

[9] R. Basmadjian and H. D. Meer, “Evaluating and modeling power consumption of multi-core processors,” in *Proc. 3rd Int. Conf. Future Energy Syst.: Where Energy, Comput. Commun. Meet (e-Energy)*, May 2012, pp. 1–10.

[10] J. Haj-Yihia, A. Yasin, Y. B. Asher, and A. Mendelson, “Fine-grain power breakdown of modern out-of-order cores and its implications on Skylake-based systems,” *ACM Trans. Archit. Code Optim.*, vol. 13, no. 4, pp. 1–25, 2016.

[11] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, “Improving the accuracy of energy predictive models for multicore CPUs by combining utilization and performance events model variables,” *J. Parallel Distrib. Comput.*, vol. 151, pp. 38–51, 2021.

[12] A. Shahid, M. Fahad, R. Reddy, and A. Lastovetsky, “Additivity: A selection criterion for performance events for reliable energy predictive modeling,” *Supercomputing Front. Innov.*, vol. 4, no. 4, pp. 50–65, Dec. 2017.

[13] A. Shahid, M. Fahad, R. R. Manumachu, and A. L. Lastovetsky, “Energy predictive models of computing: Theory, practical implications and experimental analysis on multicore processors,” *IEEE Access*, vol. 9, pp. 63149–63172, 2021.

[14] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, “Statistical power modeling of GPU kernels using performance counters,” in *Proc. Int. Green Comput. Conf. Workshops*, 2010, pp. 115–122.

[15] Y. Zhang, Y. Hu, B. Li, and L. Peng, “Performance and power analysis of ATI GPU: A statistical approach,” in *Proc. IEEE 6th Int. Conf. Netw., Architecture, Storage*, 2011, vol. 11, pp. 149–158.

[16] S. Song, C. Su, B. Rountree, and K. W. Cameron, “A simplified and accurate model of power-performance efficiency on emergent GPU architectures,” in *Proc. IEEE 27th Int. Parallel Distrib. Process. Symp.*, 2013, pp. 673–686.

[17] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, “GPGPU performance and power estimation using machine learning,” in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 564–576.

[18] Q. Wang and X. Chu, “GPGPU power estimation with core and memory frequency scaling,” *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 2, pp. 73–78, Oct. 2017.

[19] J. Nunez-Yanez, K. Nikov, K. Eder, and M. Hosseinabady, “Run-time power modelling in embedded GPUs with dynamic voltage and frequency scaling,” in *Proc. 9th Workshop Des. Tools Architectures Multicore Embedded Comput. Platforms*, New York, NY, USA:Association for Computing Machinery, 2020, pp. 1–6.

[20] N. Mammeri, M. Neu, S. Lal, and B. Juurlink, “Performance counters based power modeling of mobile GPUs using deep learning,” in *Proc. 2019 Int. Conf. High Perform. Comput. Simul.*, 2019, pp. 193–200.

[21] J. Treibig, G. Hager, and G. Wellein, “Likwid: A lightweight performance-oriented tool suite for x86 multicore environments,” in *Proc. 39th IEEE Int. Conf. Parallel Process. Workshops*, 2010, pp. 207–216.

[22] M. Fahad and R. R. Manumachu, “HCLWattsUp: Energy API using system-level physical power measurements provided by power meters,” 2022. [Online]. Available: https://csgitlab.ucd.ie/manumachu/hclwattsup

[23] M. Gamell et al., “Exploring power behaviors and trade-offs of in-situ data analytics,” in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, New York, NY, USA:Assoc. Comput. Machinery, 2013, pp. 1–12.

[24] H. Khaleghzadeh, M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, “Bi-objective optimization of data-parallel applications on heterogeneous HPC platforms for performance and energy through workload distribution,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 543–560, Mar. 2021.

[25] T. Malik and A. Lastovetsky, “Towards optimal matrix partitioning for data parallel computing on a hybrid heterogeneous server,” *IEEE Access*, vol. 9, pp. 17229–17244, 2021.

[26] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, “Power measurement techniques on standard compute nodes: A quantitative comparison,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2013, pp. 194–204.

[27] M. Burtscher, I. Zecena, and Z. Zong, “Measuring GPU power with the K20 built-in sensor,” in *Proc. Workshop Gen. Purpose Process. Using GPUs*, 2014, pp. 2828–2836.

[28] M.B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, “Predictive system shutdown and other architectural techniques for energy efficient programmable computation,” *IEEE Trans. Very Large Scale Integration Syst.*, vol. 4, no. 1, pp. 42–55, Mar. 1996.

[29] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, “The design and use of SimplePower: A cycle-accurate energy estimation tool,” in *Proc. 37th Annu. Des. Automat. Conf.*, 2000, pp. 340–345.

[30] T. Heath, B. Diniz, B. Horizonte, E. V. Carrera, and R. Bianchini, “Energy conservation in heterogeneous server clusters,” in *Proc. 10th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2005, pp. 186–195.

[31] X. Fan, W.-D. Weber, and L. A. Barroso, “Power provisioning for a warehouse-sized computer,” in *Proc. 34th Annu. Int. Symp. Comput. Archit.*, 2007, pp. 13–23.

[32] A. Kansal and F. Zhao, “Fine-grained energy profiling for power-aware application design,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, pp. 26–31, Aug. 2008.

[33] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, “Full-system power analysis and modeling for server environments,” in *Proc. Workshop Model., Benchmarking, Simul.*, 2006, pp. 70–77.

[34] S. Rivoire, P. Ranganathan, and C. Kozyrakis, “A comparison of high-level full-system power models,” in *Proc. Conf. Power Aware Comput. Syst.*, 2008.

[35] S. Khokhriakov, R. R. Manumachu, and A. Lastovetsky, “Multicore processor computing is not energy proportional: An opportunity for bi-objective optimization for energy and performance,” *Appl. Energy*, vol. 268, 2020, Art. no. 114957.

[36] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, “Power and performance characterization and modeling of GPU-accelerated systems,” in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2014, pp. 113–122.

[37] V. Adhinarayanan, B. Subramaniam, and W.-c. Feng, “Online power estimation of graphics processing units,” in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, 2016, pp. 245–254.

[38] J. A. Guerreiro, A. Ilic, N. Roma, and P. Tomás, “Modeling and decoupling the GPU power consumption for cross-domain DVFS,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2494–2506, Nov. 2019.

[39] F. Wang, W. Zhang, S. Lai, M. Hao, and Z. Wang, “Dynamic GPU energy optimization for machine learning training workloads,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2943–2954, Nov. 2022.

[40] J. Kerlin and R. R. Manumachu, “Software power meter API for parallel hybrid programs on heterogeneous hybrid servers,” 2025. [Online]. Available: https://csgitlab.ucd.ie/manumachu/openh/-/tree/master/softwarepowermeterapi?reftype=heads

Hafiz Adnan Niaz (Member, IEEE) received the BSc degree in computer systems engineering from UCET-IUB and the MS degree in computer engineering from the NUST College of Electrical and Mechanical Engineering (CEME-NUST), Islamabad Pakistan. He is currently working toward the PhD degree from the School of Computer Science, University College Dublin (UCD) Ireland. His research interests include parallel computing, high-performance computing, and energy-efficient computing.

Ravi Reddy Manumachu (Member, IEEE) received the BTech degree from I.I.T, Madras in 1997 and the PhD degree from the School of Computer Science, University College Dublin (UCD) in 2005. He is currently assistant professor in the School of Computer Science with UCD. His main research interests include high performance heterogeneous computing and energy-efficient computing.

Alexey Lastovetsky (Member, IEEE) received the PhD degree from the Moscow Aviation Institute in 1986, and the Doctor of Science degree from the Russian Academy of Sciences in 1997. He is currently associate professor in the School of Computer Science at University College Dublin (UCD). He is also the founding director of the UCD Heterogeneous Computing Laboratory. He authored the monographs Parallel computing on heterogeneous networks (Wiley, 2003) and High performance heterogeneous computing (Wiley, 2009). His main research interests include heterogeneous parallel computing and energy-efficient parallel computing. He is an inaugural ScholarGPS Highly Ranked Scholar - Lifetime (ranked within the top 0.05% of all scholars in Parallel Computing).