PeaPaw: Performance and Energy-Aware Partitioning of Workload on Heterogeneous Platforms

LI TANG, University of Notre Dame
RICHARD F. BARRETT and JEANINE COOK, Sandia National Laboratories
X. SHARON HU, University of Notre Dame

Performance and energy are two major concerns for application development on heterogeneous platforms. It is challenging for application developers to fully exploit the performance/energy potential of heterogeneous platforms. One reason is the lack of reliable prediction of the system's performance/energy before application implementation. Another reason is that a heterogeneous platform presents a large design space for workload partitioning between different processors. To reduce such development cost, this article proposes a framework, PeaPaw, to assist application developers to identify a workload partition (WP) that has high potential leading to high performance or energy efficiency before actual implementation. The PeaPaw framework includes both analytical performance/energy models and two sets of workload partitioning guidelines. Based on the design goal, application developers can obtain a workload partitioning guideline from PeaPaw for a given platform and follow it to design one or multiple WPs for a given workload. Then PeaPaw can be used to estimate the performance/energy of the designed WPs, and the WP with the best estimated performance/energy can be selected for actual implementation. To demonstrate the effectiveness of PeaPaw, we have conducted three case studies. Results from these case studies show that PeaPaw can faithfully estimate the performance/energy relationships of WPs and provide effective workload partitioning guidelines.

CCS Concepts: • **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Theory of computation** → *Models of computation*; • **Computing methodologies** → Parallel computing methodologies;

Additional Key Words and Phrases: Heterogeneous platforms, workload partitioning, performance modeling, energy modeling

ACM Reference Format:
Li Tang, Richard F. Barrett, Jeanine Cook, and X. Sharon Hu. 2017. PeaPaw: Performance and energy-aware partitioning of workload on heterogeneous platforms. ACM Trans. Des. Autom. Electron. Syst. 22, 3, Article 41 (March 2017), 26 pages.
DOI: http://dx.doi.org/10.1145/2999540

1. INTRODUCTION

Heterogeneous platforms provide application developers with more than one kind of processor and have the potential to offer higher performance and energy efficiency

We thank Sandia National Laboratories for funding this work. Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
Authors' addresses: L. Tang and X. S. Hu, Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556; emails: {ltang, shu}@nd.edu; R. F. Barrett, Information Systems Analysis Center, Sandia National Laboratories, Albuquerque, NM 87185; email: rfbarre@sandia.gov; J. Cook, Center for Computing Research, Sandia National Laboratories, Albuquerque, NM 87185; email: jeacook@sandia.gov.

DOI: http://dx.doi.org/10.1145/2999540

[Chen and Singh 2012; Rofouei et al. 2008] than homogeneous platforms. However, it is rather costly for applications to actually achieve the high performance and energy efficiency promised by such a platform [Hsu et al. 2005; Liu and Luk 2012; Brodtkorb et al. 2010]. A typical application development process for improving performance/energy consists of three steps: design, implementation, and evaluation. The major issue with this method is that the process often needs to be iterated in order to achieve a superior implementation [Tichy 1982]. For heterogeneous platforms, this issue is more challenging than for conventional uniprocessor or homogeneous multiprocessor platforms. A heterogeneous platform has different processors that deliver distinct performance/energy characteristics, and this processor heterogeneity increases the design space for partitioning the workload between different processors [Angiolini et al. 2006; Chung et al. 2010; Koufaty et al. 2010]. For example, some studies have shown that partitioning the code [Daga and Nutter 2012; Sodsong et al. 2014] instead of the input dataset [Zhong et al. 2012; Grasso et al. 2013] can better exploit the performance/energy potential of heterogeneous platforms. Such consideration is not necessary for uniprocessor or homogeneous multiprocessor platforms. With the extremely large design space for heterogeneous platforms, the development process often needs to be repeated more times to achieve the promised performance/energy.

This article aims to help application developers reduce the development cost while exploiting the performance/energy potential of heterogeneous platforms. Most studies [Cong and Yuan 2012; Grewe and OBoyle 2011; Goraczko et al. 2008; Gregg et al. 2011; Paterna et al. 2012] of improving perfromance/energy of heterogeneous platforms focus on task scheduling, which is not helpful for saving the application development cost in design time. While, we focus on the design cycle (i.e., workload partitioning) of the application development process and target central processing unit (CPU)+graphics processing unit (GPU) heterogeneous platforms. Our basic idea is to assist application developers to partition the workload of a single application and quickly identify a workload partition (WP) with high performance/energy potential (based on estimation) before actual implementation. Then, application developers can focus on implementing the suggested WP and its optimization. This modified development process can significantly reduce the development cost by avoiding repeated development cycles.

To achieve this goal, we develop a framework for Performance/Energy Aware PArtitioning of Workload on heterogeneous platforms (PeaPaw). Workload here refers to the amount of computation (measured by flops) and memory traffic (measured by bytes) to be executed, which can be partitioned via dividing code or dataset. PeaPaw specifically provides two types of help on suggesting a WP to application developers. One type of help is a set of performance-oriented and energy-oriented guidelines that application developers can follow to partition a given workload for better performance and energy, respectively. Most of these guidelines only provide a general direction, so application developers may design one or multiple WPs for a given workload and a design goal (performance or energy). To help application developers select a WP among different ones, PeaPaw also provides help through performance/energy estimation of WPs on heterogeneous platforms.

To evaluate the effectiveness of PeaPaw, we have conducted three detailed case studies. One is based on a simple synthetic application that is composed of vector operations. Another is based on a linear algebra application that performs matrix transpose and matrix multiplication. The third one is based on the data assembly (DA) stage in a representative finite element method (FEM) code (i.e., miniFE [Heroux et al. 2009]). For each case study, we have designed four different WPs. We then implemented these designed WPs and measured their performance and energy on real CPU+GPU platforms. By comparing the measured performance and energy data with the estimated ones, we show that PeaPaw can provide reliable prediction for performance/energy comparisons

among WPs. We also use the measured data to indicate that the workload partitioning guidelines can efficiently help design WPs with better performance or energy.

Our PeaPaw builds on two main contributions to help reduce the application development cost for exploiting performance/energy potential on heterogeneous platforms. First, we have developed a high-level performance/energy modeling scheme. The modeling scheme extends the performance/energy roofline models [Williams et al. 2009; Choi et al. 2013] to estimate the performance/energy potential of WPs on heterogeneous platforms. We consider the performance/energy impacts of an idle processor in different WP execution models. We also introduce three parameters that can represent all the WPs in the full design space. The second contribution is the derivation of the performance-oriented and energy-oriented workload partitioning guidelines. The guidelines can assist application developers to determine whether to partition a given workload and how to do so if needed. We have also proposed a scheme that can help classify heterogeneous platforms and connect them with specific workload partitioning guidelines.

The article is organized as follows. We first provide a brief discussion about workload partitioning and the performance/energy roofline models in Section 2. We next introduce the PeaPaw framework in Section 3 and describe its performance/energy models in Section 4. We then present the performance-oriented and energy-oriented workload partitioning guidelines in Section 5. Finally, in Sections 6 and 7, we show the PeaPaw framework validation and conclude the article.

2. BACKGROUND

In this section, we first give a brief discussion about the workload partitioning concept. We then summarize different WP execution models on heterogeneous platforms. Last, we review the original performance/energy roofline models for uniprocessor platforms.

2.1. Workload Partitioning

Workload partitioning aims to partition a given workload (measured by flops for computation and by bytes for memory access) via dividing code or dataset among the processors of a given multiprocessor platform (e.g., Tang et al. [2013], Ma et al. [2012], and Sodsong et al. [2014]). For homogeneous multiprocessor platforms, the general criterion [Huang and Feng 2009] of workload partitioning is to evenly partition the input dataset among the processors since dataset partitioning (DP) can achieve load balancing easily and each processor behaves identically for a particular algorithm. For heterogeneous platforms, not only DP but also code partitioning (CP) should be considered. Judicious code partitioning between distinct processors has been shown to be able to achieve better performance/energy than partitioning the dataset only [Tang et al. 2013; Ge et al. 2014]. This is because the CPU and GPU may have different execution efficiency for the same code. Hence, the design space of workload partitioning for heterogeneous platforms can be much larger than that of homogeneous multiprocessor platforms[1].

To study workload partitioning, in this article, a workload is assumed to be a combination of an algorithm description (i.e., pseudo code) and a dataset description (i.e., data structures). After partitioning, a WP is depicted by the pseudo code and percentage of the total dataset size assigned to every processor. Figure 1 shows a simple example of using CP and DP to partition a given workload among CPU and GPU. The given pseudo code is a loop composed of an indirect memory access and an element in a vector raised to the power 10. The given input dataset consists of three vectors with size of

[1]The workload here is at the algorithm level and differs from the actual workload, which is also dependent on particular implementation, compilation, runtime, and architecture.

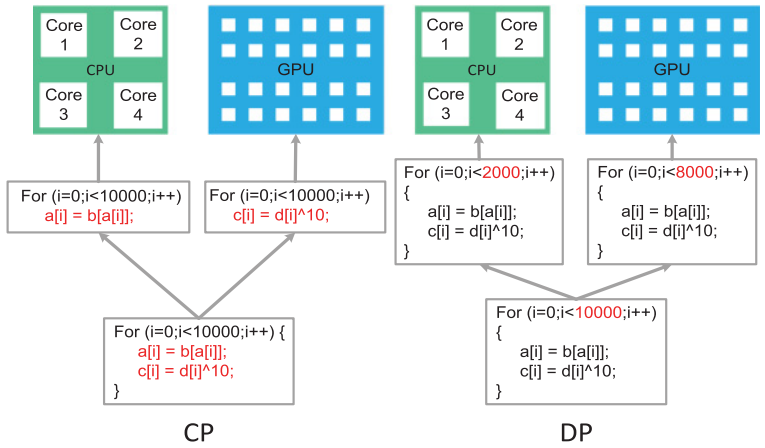


Fig. 1. Examples of CP and DP.

10,000: a, b, and d. The CP-based WP maps the indirect memory access and power in all iterations onto the CPU and GPU, respectively. The DP-based WP maps 20% and 80% of the iterations (i.e., dataset size) onto the CPU and GPU, respectively.

2.2. Execution Models

Here we summarize the execution models for scheduling WP execution. In this article, we consider the following general scenario of using CPU+GPU platforms: CPU/GPU communication before kernel execution, kernel execution on GPU and/or CPU, and CPU/GPU communication before kernel execution. We focus on the kernel execution part andhighly parallelizable kernels[2].

We classify the WP execution models into four categories: CPU-only (CO), GPU-only (GO), DP, and CP. CO and GO are the extreme cases of workload partitioning where one processor is responsible for the whole given workload. Since DP does not incur CPU/GPU data dependency, it partitions workload into the CPU and GPU workloads by dividing the input dataset into two subsets. Then DP can run its CPU and GPU workloads concurrently. We assume the input dataset can be divided finely, and then there is always a perfect DP-based WP (p-DP) that has identical CPU and GPU execution time. Depending on whether CP causes CPU/GPU data dependency, we further classify CP to CP-independency and CP-dependency (the examples are shown in Figure 1). CP-independency partitions workload into the CPU and GPU workloads by dividing the code into two subsets and running them without CPU/GPU communication concurrently. Thus, the system execution time of DP-independency and CP-independency is dominated by the processor with longer execution time. To allow CPU/GPU concurrent execution for CP-dependency, a simple software pipelining is used. CP-dependency divides the code into two subsets and also divides the input dataset into small subsets, and then the CPU and GPU communicate after processing each subset of data instead of the whole dataset. We assume the CPU/GPU communication is negligible in CP-dependency and the number of datasets is large. Then CP-dependent can overlap most of the execution time of the CPU and GPU stages. Thus, the system execution time of CP-dependency is also dominated by the processor with a longer execution time.

[2]DP does not incur CPU/GPU data dependency.

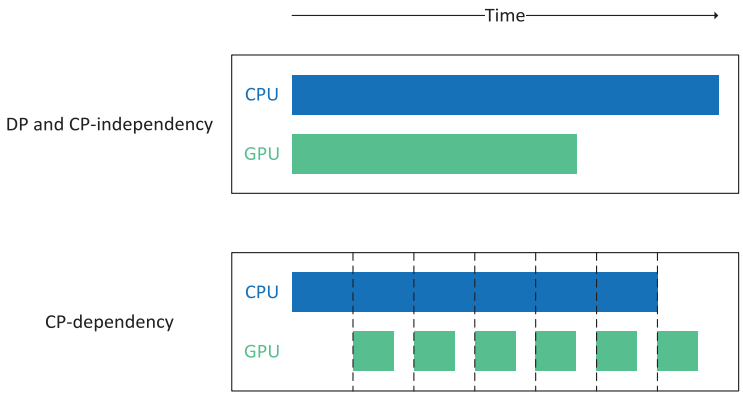


Fig. 2. Execution models of DP and CP.

2.3. Roofline Models

Since one of our tasks is to develop a performance/energy modeling scheme, we briefly review the original roofline models of performance/energy and explain why they are insufficient for direct use in PeaPaw. Williams et al. introduced the first roofline model in Williams et al. [2009] to estimate the performance upper bound for a given uniprocessor platform based on the processor's theoretical peak computation performance and achievable peak memory bandwidth. The model defines the operational intensity of a given workload as the ratio of the number of flops to the number of accessed bytes in the workload. Let I represent the operational intensity, and then

$$I = \frac{N_f}{N_b}, \tag{1}$$

where N_f and N_b are the total numbers of flops and bytes in workload. We assume that the input dataset is large and then N_b is positive. Then I is either zero or positive. Hence, for a particular uniprocessor platform, the performance upper bound defined as flops/s is calculated by

$$P = \frac{N_f}{t_b N_b \cdot \max\{I \cdot \frac{t_f}{t_b}, 1\}} = \frac{1}{t_b \cdot \max\{\frac{t_f}{t_b}, I^{-1}\}}, \tag{2}$$

where t_f and t_b are the time of executing one flop and moving one byte on that uniprocessor platform, respectively. Since t_f and t_b are constant for a given uniprocessor platform, Equation (2) indicates that the processor performance is memory bounded if $I \times \frac{t_b}{t_f}$ is smaller than I and vice versa.

Choi et al. proposed a roofline model [Choi et al. 2013] of energy to estimate the processor energy of running a workload on uniprocessor platforms. The model decomposes processor energy to dynamic and static energy and assumes that the dynamic energy consumed for one flop and one byte are constant for a given uniprocessor platform. Then the processor energy is computed as

$$\begin{aligned} E &= E_{dynamic} + E_{static} \\ &= e_f N_f + e_b N_b + p_{static} \cdot \max\{t_f N_f, t_b N_b\}, \end{aligned} \tag{3}$$

where e_f and e_b are the dynamic energy per flop and byte, respectively, and p_{static} is the processor static power.

The performance/energy roofline models and some extensions [Ilic et al. 2014; Nugteren and Corporaal 2012] are all introduced for uniprocessor platforms and can be readily applied to homogeneous multiprocessor platforms. However, extending the models to heterogeneous multiprocessor platforms does not seem to be straightforward. For example, one way of using the roofline models for heterogeneous platforms is to aggregate the estimated performance (i.e., GFlops) or energy of CPU and GPU. There are two problems with this approach. First, the simple performance aggregation cannot capture the WP execution where one processor has much idle time (i.e., wasted performance and idle energy). Second, the condition that CPU and GPU run workloads with different operational intensities is not considered in the simple aggregation. In PeaPaw, we introduce more complex models for performance/energy estimation in order to handle the unique features of heterogeneous platforms. Yet our models are still simple enough to be built easily and used without too much overhead.

3. PEAPAW FRAMEWORK

In this section, we first present an overview and the workflow of the PeaPaw framework. Then, we introduce the workload and machine parameters of PeaPaw. Last, the limitations and scalability of PeaPaw will be discussed.

3.1. Overview of PeaPaw

The objective of PeaPaw is to provide guidelines on workload partitioning and performance/energy estimation of particular WPs on heterogeneous platforms. The guidelines indicate the ideal WP that can achieve the best performance or energy on the given platform.

Figure 3 illustrates the overall workflow of PeaPaw. The input to the PeaPaw framework is a set of machine parameters and a workload description. A workload is described at the algorithm level and includes both pseudo code and dataset description. The machine parameters for describing the performance/energy characteristics of a heterogeneous platform will be discussed in Section 3.2. The output of the PeaPaw framework is a description of the WP that has the highest estimated performance or energy efficiency among the designed WPs.

The workflow of PeaPaw consists of four main stages: (i) identification of P-Paw or E-Paw, (ii) workload partitioning, (iii) WP abstraction, and (iv) performance/energy estimation and comparison. Stage (ii) relies on application developers, while the other three stages are the tools provided by PeaPaw. Stage (i) uses the given platform's machine parameters to classify it into a performance-oriented WP (P-Paw) category or an energy-oriented WP (E-Paw) if the design goal is energy. Each category corresponds to a workload partitioning guideline. By following the guideline from stage (i), in stage (ii), application developers partition the given workload between the CPU and the GPU. This stage may result in a few different WPs. Note that, using the guideline from stage (i), application developers can readily eliminate many WP alternatives. These WPs are then abstracted by using WP parameters in stage (iii). In stage (iv), the abstracted WP and machine parameters are used to estimate the performance or energy of the WPs designed in stage (ii), and the one with the highest performance or energy efficiency is selected as the final WP.

3.2. PeaPaw Parameters

The fundamental idea of PeaPaw is using parameters to abstract the given platform and designed WPs. We summarize all the machine and workload parameters in Tables I and II.

Machine parameters describe the performance and energy characteristics of the given CPU+GPU platform. To capture the computation and memory performance of

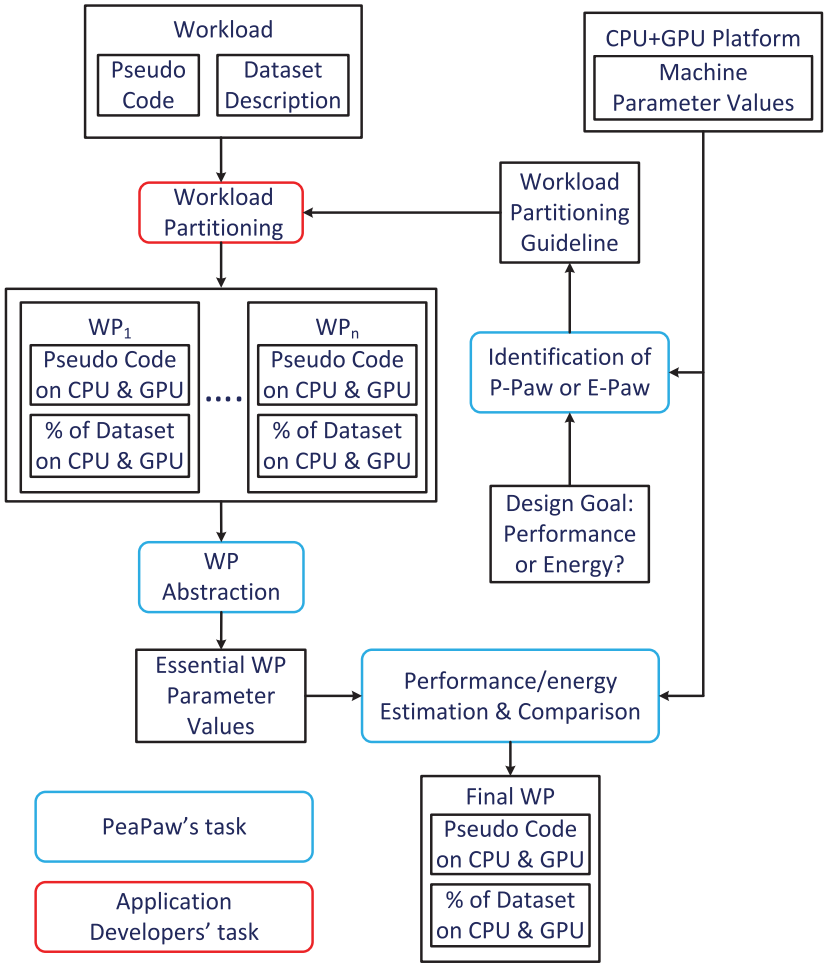


Fig. 3. Overview of the PeaPaw framework.

Table I. Workload Parameters

Parameter	Description
N_{fC}	Number of flops in the CPU workload
N_{bC}	Number of bytes in the CPU workload
I_C	Operational intensity of the CPU workload
N_{fG}	Number of flops in the GPU workload
N_{bG}	number of bytes in the GPU workload
I_G	Operational intensity of the GPU workload
N_f	Number of flops in the given workload
N_b	Number of bytes in the given workload
I	Operational intensity of the given workload

Table II. Machine Parameters

t_{fC}	Time per flop of CPU
t_{bC}	Time per byte of CPU
e_{fC}	Energy per flop of CPU
e_{bC}	Energy per byte of CPU
p_{sC}	Static power of CPU
t_{fG}	Time per flop of GPU
t_{bG}	Time per byte of GPU
e_{fG}	Energy per flop of GPU
e_{bG}	Energy per byte of GPU
p_{sG}	Static power of GPU

a processor, similarly to Williams et al. [2009], we use time per flop and time per byte movement and denote them by using t_{fC} and t_{bC} for CPU (t_{fG} and t_{bG} for GPU), respectively. We do not consider cache effects in PeaPaw. The simplicity is acceptable for applications with large working dataset (i.e., much larger than last level cache) and few repeated data accesses.

To represent the energy characteristics of a processor, we use a simple power model as follows:

$$p = p_{dynamic} + p_{static}, \tag{4}$$

where p, $p_{dynamic}$, and p_{static} represent the total, dynamic, and static processor power, respectively. The static power is constant for a given processor and is denoted by p_{sC} and p_{sG} for CPU and GPU, respectively. The dynamic power is dependent on the executed workload and can be further decomposed into power consumed by computation and data movement. We consider two processor power states and assume that p is equal to p_{static} when processor is idle. To abstract these dynamic power characteristics, we follow the energy roofline model [Choi et al. 2013]. We use energy per flop (denoted as e_{fC} and e_{fG} for CPU and GPU, with unit pJ/flop) and energy per byte movement (denoted as e_{bC} and e_{bG} for CPU and GPU, with the unit pJ/byte).

In total, we use a set of 10 machine parameters (i.e., $\{t_{fC}, t_{fG}, t_{bC}, t_{bG}, e_{fC}, e_{fG}, e_{bC}, e_{bG}, p_{sC}, p_{sG}\}$) to describe a CPU+GPU platform's performance and energy characteristics. For a given platform, the values of these machine parameters are constant. The values of t_{fC} and $t_{fG}$are set as

$$t_{fC} = \frac{1}{PeakPerf_{CPU}} \quad \text{and} \quad t_{fG} = \frac{1}{PeakPerf_{GPU}}, \tag{5}$$

where the peak performance for CPU and GPU is the product of number of cores, basecore frequency, and issue width[3]. Then t_{fC} or t_{fG} can be computed as

$$t_f = \frac{1}{\#\_cores \times base\_core\_frequency \times issue\_width}. \tag{6}$$

If the processor single instruction multiple data (SIMD) engines are used, then developers can capture this feature by changing the value of $issue\_width$. To get the values of t_{bC} and t_{bG}, we ran a collection of microbenchmarks and measure the peak memory bandwidth. t_{bC} and t_{bG} are then set to the inverse of the respecting bandwidth value. For the energy parameter values, we follow the approach used in Choi et al. [2013]. More detailed examples of obtaining the machine parameter values are given in Section 6.

[3]Since the dual-issue units are not used in our experiments, we let the issue width be 1 for peak performance calculation in Equation (6).

WP parameters depict the designed WPs from the workload partitioning stage. A direct way to represent a WP designed in the *workload partitioning* stage is to use four abstracted WP parameters (i.e., $\{N_{fC}, N_{bC}, N_{fG}, N_{bG}\}$). Since PeaPaw does not require code implementation, the WP parameter values can be obtained by applying static analysis on the algorithm or pseudo code. Specially, we count each useful (i.e., doing useful work in algorithm) computation operation (e.g, addition) as one flop and each single-precision memory access in an allocated array as four bytes. Section 6 will give a more detailed example on how to perform the static analysis for this purpose. To reduce the effort of using PeaPaw to help workload partitioning, we make the WP parameters independent on the architectural impacts (e.g., memory coalescing and SIMD). In addition, this simplicity is acceptable since PeaPaw aims to help developers find relative better WP and relative accuracy is more important than absolute accuracy for PeaPaw. However, if high development cost is allowed, then these architectural impacts can be captured when doing static analysis. Without considering architectural impacts, one property of workload partitioning is that the total numbers of flops and bytes for a given workload is fixed (assume that processor communication is negligible). That is, for any WP_i, we have

$$N_f = N_{fC} + N_{fG} \quad \textit{and} \quad N_b = N_{bC} + N_{bG}. \tag{7}$$

Using this property, we reduce the parameters used to describe a designed WP to three parameters, that is, I, I_C, and I_G, which represent the operational intensities of the entire, CPU, and GPU workloads, respectively. We will elaborate on the derivation of I, I_C, and I_G in Section 4.

3.3. Discussion

The goal of PeaPaw is to help developers exploit performance/energy potential of heterogeneous platforms for a given application before implementation. This goal differentiates PeaPaw from traditional studies (e.g., Paterna et al. [2012] and Grasso et al. [2013]) of online task scheduling on heterogeneous platforms in two aspects. First, PeaPaw works in design time and does not require the workload implementation. Second, PeaPaw focuses on code development for a single given application. The behavior of tasks that form the given application's workload is unknown before workload partitioning. Therefore, most task scheduling studies cannot be readily applied for guiding workload partitioning before implementation.

To make PeaPaw lightweight, PeaPaw only targets heterogeneous platforms having one CPU and one GPU. For heterogeneous platforms having more than one CPU and one CPU, PeaPaw could also be applied if the design goal is performance. The idea is to divide processors into two subgroups and each subgroup of processors is viewed as one single processor. When multiple processors are grouped, their peak computation performance and memory bandwidth are aggregated for extracting the computation and memory performance values. This process continues until the entire workload is distributed among the given processors. If the design goal is energy, then PeaPaw can only handle heterogeneous platform with the same CPUs and the same GPUs. In this case, all the identical CPUs and all the identical GPUs are grouped into their respective subgroups. For each group, processors' computation performance, memory bandwidth, and static power are aggregated for extracting parameter values. After workload has been partitioned into the two subgroups, each subgroup's workload is evenly assigned to all the processors within the subgroup. In its current form, PeaPaw is not able to handle heterogenous platforms using field-programmable gate array (FPGA) or application-specific processors since they cannot be easily abstracted by using time per flop and time per byte. This is left for future work.

Using PeaPaw to guide workload partitioning requires additional effort for acquiring the PeaPaw parameter values, but it should reduce the cost of workload partitioning. The cost of workload partitioning is dependent on the complexity of the given workload and platform. The workload complexity (e.g., code size) impacts the number of WPs that follow the guideline. The platform complexity (e.g., number of processors) incurs more iterations of PeaPaw usage if it is applied. However, using PeaPaw for complicated workloads and platforms should still reduce the code development effort than without using PeaPaw.

4. PERFORMANCE AND ENERGY ESTIMATION

A major task in PeaPaw is to estimate the performance or energy of a given WP. As discussed earlier, it is impractical to profile or measure these metrics for a large number of WPs. A high-level performance/energy model can be extremely helpful for such estimation work. In this section, we present the details of the PeaPaw performance/energy models and how they are used for performance/energy estimation of WPs.

4.1. Essential WP Parameters

As discussed in Section 3.2, one way of characterizing WPs is to use four WP parameters, that is, N_{fC}, N_{bC}, N_{fG}, and N_{bG}. However, it is more convenient to use only three parameters, that is, I, I_C, and I_G based on Equation (7). We refer to I, I_C, and I_G as *essential WP parameters*. Using the essential WP parameters has two specific advantages. First, if using these parameters to derive guidelines, the guidelines would be independent of the dataset size. Second, the essential WP parameters use one less parameter, which helps simplify the workload partitioning guidelines and performance/energy data visualization.

All DP-based WPs have the same presentation: $I = I_C = I_G$, and by using the essential WP parameters, we use p-DP to represent all DP-based WPs. If the design goal is to optimize performance, then it is clear that p-DP achieves the best estimated performance among all DP-based WPs since both processors would be kept busy, which leads to the shortest execution time. If, on the other hand, reducing energy is the goal, then p-DP and two extreme DP-based WPs (i.e., CO and GO) are the only three WP alternatives to be the most energy efficient DP-based WPs. In our models, we separate the CO (i.e., $I_C = I$ and $I_G = 0$) and GO (i.e., $I_C = 0$ and $I_G = I$) WPs from the $I = I_C = I_G$ case. Thus, we can safely adopt p-DP as the best WP for the $I = I_C = I_G$ case.

Now we discuss the presentation of the CP-based WPs by using the essential WP parameters. Theoretically, using CP can obtain WPs with $I = I_C = I_G$. We view these WPs as equivalent to the DP-based WPs, so we keep using p-DP to represent them. The following equations show that a WP does not satisfy $I = I_C = I_G$ and can be represented by using a particular set of the essential WP parameters. According to the definition of operational intensity, we can rewrite Equation (7) as follows:

$$N_b I = N_{bC} I_C + N_{bG} I_G. \tag{8}$$

Since N_{bG} is equal to $N_b - N_{bC}$, we can extend Equation (8) to

$$N_b I = N_{bC} I_C + (N_b - N_{bC}) \cdot I_G. \tag{9}$$

By incorporating the essential WP parameters, $\{I, I_C, I_G\}$, we can solve Equation (9) to get the ratio of N_{bC} to $N_b$as

$$\frac{N_{bC}}{N_b} = \frac{I - I_G}{I_C - I_G}. \tag{10}$$

By replacing N_{bC} by $N_b - N_{bG}$ in Equation (9), we have

$$\frac{N_{bG}}{N_b} = \frac{I - I_C}{I_G - I_C}. \tag{11}$$

Similarly, we can get

$$\frac{N_{fG}}{N_b} = I_G \cdot \frac{I - I_C}{I_G - I_C} \tag{12}$$

and

$$\frac{N_{fC}}{N_b} = I_C \cdot \frac{I - I_G}{I_C - I_G}. \tag{13}$$

It can be seen from Equations (10)–(13) that, for a given workload (i.e., N_b is fixed), a WP (i.e., N_{fC}, N_{bC}, N_{fG}, and N_{bG}) can be expressed by using a unique set of the essential WP parameters.

4.2. Performance Estimation

Similarly to the roofline model in Williams et al. [2009], we define performance of a WP as the average number of flops executed per second. We first estimate the total system execution time. Since all the WP execution models introduced in Section 2.2 can overlap the CPU and GPU execution, the total system execution time is dominated by the processor that has longer execution time, that is

$$T_{sys} = \max\{T_{CPU}, T_{GPU}\}. \tag{14}$$

Based on the performance roofline model that assumes a processor can overlap its computation and memory access time, the CPU execution time can be computed as

$$T_{CPU} = \max\{t_{fC}N_{fC}, t_{bC}N_{bC}\}. \tag{15}$$

Similarly, the GPU execution time is

$$T_{GPU} = \max\{t_{fG}N_{fG}, t_{bG}N_{bG}\}. \tag{16}$$

By replacing T_{CPU} and T_{GPU} in Equation (14) with Equations (15) and (16), we have

$$\begin{aligned} T_{sys} &= \max\{\max\{t_{fC}N_{fC}, t_{bC}N_{bC}\}, \max\{t_{fG}N_{fG}, t_{bG}N_{bG}\}\} \\ &= \max\{t_{fC}N_{fC}, t_{bC}N_{bC}, t_{fG}N_{fG}, t_{bG}N_{bG}\}, \end{aligned} \tag{17}$$

and then

$$P_{sys} = \frac{N_f}{T_{sys}} = \left(\max\left\{\frac{N_{fC}}{N_f} \cdot t_{fC}, \frac{N_{bC}}{N_f} \cdot t_{bC}, \frac{N_{fG}}{N_f} \cdot t_{fG}, \frac{N_{bG}}{N_f} \cdot t_{bG}\right\}\right)^{-1} \tag{18}$$

represents the system performance.

As stated in Section 4.1, we want to define P_{sys} as a function of the essential WP parameters, $\{I, I_C, I_G\}$. For ease of derivation, we consider two cases separately: (i) $I = I_C = I_G$, and (ii) otherwise. The $I = I_C = I_G$ case corresponds to p-DP. We have pointed out in Section 2.3 that modeling the performance of p-DP can simply aggregate the CPU and GPU performance by using the original roofline model of performance. Based on Equation (2), we get the performance of p-DP as

$$P_{sys} = \frac{1}{t_{bC} \cdot \max\{\frac{t_{fC}}{t_{bC}}, I^{-1}\}} + \frac{1}{t_{bG} \cdot \max\{\frac{t_{fG}}{t_{bG}}, I^{-1}\}}. \tag{19}$$

For case (ii), by using the essential WP parameters (i.e., Equations (10)–(13)), we can express Equation (18) as

$$P_{sys} = \left(\max\left\{t_{fC} \cdot \frac{I_C \cdot (I - I_G)}{I \cdot (I_C - I_G)}, t_{bC} \cdot \frac{I - I_G}{I \cdot (I_C - I_G)}, t_{fG} \cdot \frac{I_G \cdot (I - I_C)}{I \cdot (I_G - I_C)}, t_{bG} \cdot \frac{I - I_C}{I \cdot (I_G - I_C)}\right\}\right)^{-1}. \tag{20}$$

4.3. Energy Estimation

We define the energy efficiency of WP as the average number of flops per joule. The energy efficiency targets the CPU and GPU in the given platform. To estimate the energy efficiency, we follow the energy roofline model [Choi et al. 2013] to decompose the processor energy into dynamic and static energy, that is,

$$E = E_{dynamic} + E_{static}, \tag{21}$$

where $E_{dynamic}$ and E_{static} represent the dynamic and static energy consumed by executing WP on the given platform. By multiplying the system static power (i.e., the total of CPU and GPU static power, $p_{sC} + p_{sG}$) with the total system execution time (Equation (17)), we can get the system static energy

$$E_{static} = (p_{sC} + p_{sG}) \cdot \max\{t_{fC}N_{fC}, t_{bC}N_{bC}, t_{fG}N_{fG}, t_{bG}N_{bG}\}. \tag{22}$$

For each processor, its dynamic energy consists of computation and memory access dynamic energy. Since PeaPaw assumes that the dynamic energy usage of each operation is constant for a given processor, we can model the system dynamic energy as

$$\begin{aligned} E_{dynamic} &= E_{CPU\_dynamic} + E_{GPU\_dynamic} \\ &= e_{fC}N_{fC} + e_{bC}N_{bC} + e_{fG}N_{fG} + e_{bG}N_{bG}, \end{aligned} \tag{23}$$

where e_{fC} and e_{bC} (respectively, e_{fG} and e_{bG}) represent the energy per flop and byte of CPU (respectively, GPU).

Now, we can get the total system energy as

$$\begin{aligned} E_{total} = (p_{sC} + p_{sG}) &\cdot \max\{t_{fC}N_{fC}, t_{bC}N_{bC}, t_{fG}N_{fG}, t_{bG}N_{bG}\} \\ &+ e_{fC}N_{fC} + e_{bC}N_{bC} + e_{fG}N_{fG} + e_{bG}N_{bG}. \end{aligned} \tag{24}$$

By dividing the total number of flops in the given workload by the total system energy, we get the energy efficiency estimation of WP as

$$\begin{aligned} EE &= \frac{N_f}{E_{total}} \\ &= \left(e_{fC} \cdot \frac{N_{fC}}{N_f} + e_{bC} \cdot \frac{N_{bC}}{N_f} + e_{fG} \cdot \frac{N_{fG}}{N_f} + e_{bG} \cdot \frac{N_{bG}}{N_f}\right. \\ &\quad \left. + (p_{sC} + p_{sG}) \cdot \max\left\{t_{fC} \cdot \frac{N_{fC}}{N_f}, t_{bC} \cdot \frac{N_{bC}}{N_f}, t_{fG} \cdot \frac{N_{fG}}{N_f}, t_{bG} \cdot \frac{N_{bG}}{N_f}\right\}\right)^{-1}. \end{aligned} \tag{25}$$

By using Equations (10)–(13) and the energy estimation in Equation (25), we have an energy efficiency expression by using the essential WP parameters

$$\begin{aligned} EE = \left(e_{fC} \cdot \frac{I - I_G}{I_C - I_G} + e_{bC} \cdot \frac{I - I_G}{I \cdot (I_C - I_G)} + e_{fG} \cdot \frac{I - I_C}{I_G - I_C} + e_{bG} \cdot \frac{I - I_C}{I \cdot (I_G - I_C)}\right. \\ \left. + (p_{sC} + p_{sG}) \cdot \max\left\{t_{fC} \cdot \frac{I - I_G}{I_C - I_G}, t_{bC} \cdot \frac{I - I_G}{I \cdot (I_C - I_G)}, t_{fG} \cdot \frac{I - I_C}{I_G - I_C}, t_{bG} \cdot \frac{I - I_C}{I \cdot (I_G - I_C)}\right\}\right)^{-1}. \end{aligned} \tag{26}$$

5. WORKLOAD PARTITIONING GUIDELINES

PeaPaw provides a set of guidelines for partitioning workloads that can significantly reduce the design space of heterogeneous platforms. With the guidance from PeaPaw, the large design space for workload partitioning on heterogeneous platforms can be reduced. Based on the design goal (performance or energy) of workload partitioning, we introduce two sets of workload-independent guidelines: performance-oriented and energy-oriented. The key idea of developing these guidelines is to classify CPU+GPU platforms into different categories. Depending on the machine parameters, a platform can be classified into one of three P-Paw (respectively, eight E-Paw) categories for performance-oriented (respectively, energy-oriented) partitioning. Each P-Paw category is associated with a particular performance-oriented guideline, and each E-Paw category is associated with a particular energy-oriented guideline.

5.1. Performance-Oriented Guidelines

We classify heterogeneous platforms according to performance by identifying whether the CPU and GPU have similar performance characteristics. To measure the processor performance characteristics, we propose using machine balance, introduced in Callahan et al. [1988]. Machine balance is the ratio of computation performance to memory performance and can be computed as

$$BL_C = \frac{{t_{fC}}^{-1}}{{t_{bC}}^{-1}} = \frac{t_{bC}}{t_{fC}} \qquad \text{and} \qquad BL_G = \frac{{t_{fG}}^{-1}}{{t_{bG}}^{-1}} = \frac{t_{bG}}{t_{fG}}. \tag{27}$$

Since t_{bC}, t_{fC}, t_{bG}, and t_{fG} are always positive, BL_C and BL_G are positive ratios.

By comparing BL_C with BL_G, we can classify CPU+GPU platforms into three P-Paw categories: `CPU_DP-GPU_DP`, `CPU_COMP-GPU_MEM`, and `CPU_MEM-GPU_COMP`. If BL_C is equal to BL_G, then we refer to the P-Paw category as `CPU_DP-GPU_DP` since the two processors have the same machine balance and behave identically for the same workload. The `CPU_COMP-GPU_MEM` and `CPU_MEM-GPU_COMP` categories consist of platforms for which the CPU is more efficient than the GPU in terms of computation and memory access, respectively.

For the platforms in the `CPU_DP-GPU_DP` category, we can show that using p-DP to achieve balanced CPU and GPU workload gets the best estimated performance. For `CPU_COMP-GPU_MEM` and `CPU_MEM-GPU_COMP`, it is desirable to partition the workload into a portion with higher operational intensity and a portion with lower operational intensity and assign them to different processors. Note that the guidelines for `CPU_COMP-GPU_MEM` and `CPU_MEM-GPU_COMP` are more general than the one for `CPU_DP-GPU_DP`, since they cover more CPU+GPU platforms. The three P-Paw categories and their associated guidelines are summarized as follows.

—`CPU_DP-GPU_DP`
  —*Identifier*: $BL_C = BL_G$.
  —*Guideline*: Partition the given workload's dataset to achieve p-DP.
—`CPU_COMP-GPU_MEM`
  —*Identifier*: $BL_C > BL_G$.
  —*Guideline*: Partition the given workload's code into a portion with higher operational intensity and a portion with lower operational intensity and assign them to the CPU and GPU, respectively.
—`CPU_MEM-GPU_COMP`
  —*Identifier*: $BL_C < BL_G$.
  —*Guideline*: Partition the given workload's code into a portion with higher operational intensity and a portion with lower operational intensity and assign them to the GPU and CPU, respectively.

Observation 5.1 below states the optimality of guideline associated with the CPU_DP-GPU_DP category.

Observation 5.1. *When CPU and GPU have the same machine balance, that is, $BL_C = BL_G$, p-DP achieves the highest estimated performance.*

The correctness of this observation can be shown as follows. Specifically, we prove that Equation (17) has the lowest value when $\max\{t_{fC}N_{fC}, t_{bC}N_{bC}\} = \max\{t_{fG}N_{fG}, t_{bG}N_{bG}\}$ (i.e., p-DP) and $\frac{t_{bC}}{t_{fC}} = \frac{t_{bG}}{t_{fG}}$ (i.e., $BL_C = BL_G$). Since $\frac{t_{bC}}{t_{fC}} = \frac{t_{bG}}{t_{fG}}$, CPU and GPU are either both computation bounded or memory bounded. Namely, $\max\{t_{fC}N_{fC}, t_{bC}N_{bC}\} = \max\{t_{fG}N_{fG}, t_{bG}N_{bG}\}$ can be simplified to $t_{fC}N_{fC} = t_{fG}N_{fG}$ or $t_{bC}N_{bC} = t_{bG}N_{bG}$. Then Equation (17) has the value of $t_{fC}N_{fC}$ (or $t_{fG}N_{fG}$) or $t_{bC}N_{bC}$ (or $t_{bG}N_{bG}$). Suppose that another WP, say, WP^-, derived from p-DP by moving N_{fM} flops and N_{bM} bytes from the CPU workload to the GPU workload, has higher estimated performance than p-DP. Then, the execution time of WP^- obtained by following Equations (14)–(17) has the new value of $t_{bG} \cdot (N_{bG} + N_{bM})$ or $t_{fG} \cdot (N_{fG} + N_{fM})$, which are obviously larger than $t_{bG}N_{bG}$ or $t_{bC}N_{bC}$. Hence, the supposition, that is, WP^- has higher estimated performance than p-DP, is false.

The guidelines associated with the CPU_COMP-GPU_MEM and CPU_MEM-GPU_COMP categories only provide a general direction for workload partitioning. More detailed guidelines should include the information of the given workload and require more categories. To simplify the PeaPaw guidelines, we only consider workload-independent guidelines in this article and leave workload-dependent guidelines for our future work.

5.2. Energy-Oriented Guidelines

Best-performance WPs do not necessarily lead to best-energy WPs. For example, a best-performance WP uses both CPU and GPU, but the best-energy WP uses CPU only when the GPU is extremely energy inefficient. Thus, we introduce a set of energy-oriented guidelines when the goal of partitioning the workload is energy and examine these guidelines in this subsection.

Similarly to the performance-oriented case, we want to classify heterogeneous platforms into different E-Paw categories. To do so, we identify whether dynamic or static energy dominates the system energy for a platform since a processor still consumes energy (i.e., static energy) when it is idle. To identify the dominant energy for a given platform, we introduce two gradient energy metrics for quantifying the system energy variation of migrating one flop and one byte in WP and denote them by ΔGP_f and ΔGP_b, respectively. ΔGP_f and ΔGP_b can be computed as

$$\Delta GP_f = |e_{fC} - e_{fG}| - (p_{sC} + p_{sG}) \cdot t_{fG} \tag{28}$$

and

$$\Delta GP_b = |e_{bC} - e_{bG}| - (p_{sC} + p_{sG}) \cdot t_{bG}. \tag{29}$$

ΔGP_f and ΔGP_b are dependent only on the hardware platform and have units of joules.

To help understand the role of gradient energy in E-Paw, assume that a workload is fully mapped onto the GPU of a given platform and the platform has $e_{fG} < e_{fC}$ and $t_{fG}N_f > t_{bG}N_b$. Then moving one flop from the GPU to the CPU would increase the total dynamic energy by $e_{fC} - e_{fG}$ while decreasing the total static energy by $(p_{sC} + p_{sG}) \cdot t_{fG}$. The computation gradient energy, ΔGP_f, captures the net energy increase due to moving one flop from CPU to GPU ($e_{fC} < e_{fG}$) or GPU to CPU ($e_{fC} \geq e_{fG}$). The memory gradient energy can be interpreted in the same way.

We use ΔGP_f and ΔGP_b to classify a given platform into eight E-Paw categories: `CPU-only`, `GPU-only`, `CPU_COMP-GPU_MEM`, `CPU_MEM-GPU_COMP`, `Race-to-halt`, `CPU_COMP-GPU_COMP`, `CPU_MEM-GPU_MEM`, and `Workload-dependent`. The `CPU-only` and `GPU-only` categories are essentially the same, since they both indicate that the system energy is dominated by both the computation and memory dynamic energy. `CPU_COMP-GPU_MEM` and `CPU_MEM-GPU_COMP` refer to platforms where CPU is more energy efficient than GPU for computation and memory access, respectively. Similarly, `CPU_COMP-GPU_COMP` and `CPU_MEM-GPU_MEM` refer to platforms whose system energy is dominated by dynamic energy of computation and memory access, respectively. `Race-to-halt` includes platforms whose system energy is dominated by static energy. For the `Workload-dependent` category, it includes all other platforms that are not in the other seven E-Paw categories.

For the `CPU-only` and `GPU-only` categories, mapping the given workload onto the processor with lower dynamic energy per flop and byte can achieve better energy efficiency. For `CPU_COMP-GPU_MEM` (respectively, `CPU_MEM-GPU_COMP`), the given workload's code should be partitioned into a portion with higher operational intensity and a portion with lower operational intensity and assign them to the CPU and GPU (respectively, GPU and CPU), respectively. For `Race-to-halt`, the system energy is determined by the execution time, so its guideline is to achieve the best performance, that is, follow the performance-oriented guidelines. `CPU_COMP-GPU_COMP` and `CPU_MEM-GPU_MEM` indicate that low energy usage requires even partitioning of the computation and memory access, respectively. Take `CPU_COMP-GPU_COMP`, for example, all the memory accesses in the target workload should be offloaded onto the memory energy efficient processor, and the computation should be evenly distributed between the two processors. Finding the most energy-efficient WP for the platforms in the `Workload-dependent` category is more complicated and requires profiling the workload. We summarize the E-Paw categories and their associated guidelines below.

—`CPU-only`
  —*Identifier*: $\Delta GP_f > 0$, $\Delta GP_b > 0$, $e_{fC} < e_{fG}$ and $e_{bC} < e_{bG}$.
  —*Guideline*: Run the given workload on CPU.
—`GPU-only`
  —*Identifier*: $\Delta GP_f > 0$, $\Delta GP_b > 0$, $e_{fC} > e_{fG}$ and $e_{bC} > e_{bG}$.
  —*Guideline*: Run the given workload on GPU.
—`CPU_COMP-GPU_MEM`
  —*Identifier*: $\Delta GP_f > 0$, $\Delta GP_b > 0$, $e_{fC} < e_{fG}$ and $e_{bC} > e_{bG}$.
  —*Guideline*: Partition the given workload's code into a portion with higher operational intensity and a portion with lower operational intensity and assign them to the CPU and GPU, respectively.
—`CPU_MEM-GPU_COMP`
  —*Identifier*: $\Delta GP_f > 0$, $\Delta GP_b > 0$, $e_{fC} > e_{fG}$ and $e_{bC} < e_{bG}$.
  —*Guideline*: Partition the given workload's code into a portion with higher operational intensity and a portion with lower operational intensity and assign them to the GPU and CPU, respectively.
—`Race-to-halt`
  —*Identifier*: $\Delta GP_f + \Delta GP_b < 0$.
  —*Guideline*: Follow the performance-oriented guidelines.
—`CPU_COMP-GPU_COMP`
  —*Identifier*: $\Delta GP_f > 0$ and $\Delta GP_b < 0$.
  —*Guideline*: Evenly distribute the workload's computation to the CPU and GPU. Distribute the workload's memory access to the CPU only when $e_{bC} < e_{bG}$ or the GPU only when $e_{bC} > e_{bG}$.

Table III. Processor Specifications

Processor	Type	# of cores	Base core clock	Issue width
i7 2600K	CPU	4	3400MHz	8
i3 2100T	CPU	2	2500MHz	8
GTX Titan	GPU	2688	837MHz	2
GTX 750	GPU	512	1020MHz	2

—`CPU_MEM-GPU_MEM`
—*Identifier*: $\Delta GP_f < 0$ and $\Delta GP_b > 0$.
—*Guideline*: Evenly distribute the workload's memory access to the CPU and GPU. Distribute the workload's computation to the CPU only when $e_{fC} < e_{fG}$ or the GPU only when $e_{fC} > e_{fG}$.

Observation 5.2 below states the optimality of guideline associated with the `GPU-only` category.

Observation 5.2. *When system energy is dominated by dynamic energy and GPU is more dynamic energy efficient than CPU in terms of computation and memory access, using GPU only achieves the lowest energy.*

To see why Observation 5.2 is valid, we can show that any WP other than `GPU-only` would result in higher total energy. According to Equation (24), the energy of the `GPU-only` WP satisfies

$$E_{GPU\_only} = e_{fG}N_f + e_{bG}N_b + (p_{sC} + p_{sG}) \cdot \max\{t_{fG}N_f, t_{bG}N_b\}. \tag{30}$$

Now, considering a WP^- using both CPU and GPU, we can use Equation (24) to obtain the new energy usage. Then the energy difference between this WP^- and `GPU-only` is

$$\begin{aligned}\Delta e = {} & (e_{fC} - e_{fG}) \cdot N_{fC} + (e_{bC} - e_{bG}) \cdot N_{bC} + (p_{sC} + p_{sG}) \cdot \\ & (\max\{t_{fC}N_{fC}, t_{bC}N_{bC}, t_{fG}N_{fG}, t_{bG}N_{bG}\} - \max\{t_{fC}N_f, t_{bC}N_b\}).\end{aligned} \tag{31}$$

Combining the conditions that $e_{fC} > e_{fG}$, $e_{bC} > e_{bG}$, $\Delta GP_f > (p_{sC} + p_{sG})$, and $\Delta GP_b > (p_{sC} + p_{sG})$ with Equation (7), we get that Δe is positive. Hence, Observation 5.2 is correct.

Similar observations can be found in the `CPU-only`, `CPU_COMP-GPU_MEM`, and `CPU_MEM-GPU_COMP` categories. The `Race-to-halt`, `CPU_COMP-GPU_COMP`, and `CPU_MEM-GPU_MEM` categories provide a general direction-to-partition workload for better energy efficiency.

6. VALIDATION OF PEAPAW

In this section, we first present our system setup and the abstracted machine parameters. We then use three case studies to validate the PeaPaw performance/energy models and workload partitioning guidelines.

6.1. System Setup

We use two CPUs and two GPUs to build four different heterogeneous platforms. The two CPUs are high-performance Intel i7 2600K and low-power Intel i3 2100T. The two GPUs are high-performance NVIDIA GeForce GTX Titan and low-power NVIDIA GeForce GTX 750. The hardware details are summarized in Table III. All four platforms are equipped with dual-channel 16GB DDR3 1666MHz memory and CORSAIR CX600 power supply unit.

The operating system is CentOS 7 64-bit. The GCC 4.7 compiler is used to produce optimized CPU binaries at the -O2 optimization level. The NVCC compiler of CUDA 7.5 is used to generate the kernel binaries on the GPUs.

Table IV. Machine Parameter Values

Processor	t_f (ps)	t_b (ps)	e_f (pJ)	e_b (pJ)	p_s (W)
i7 2600K	9.5	65.9	118	462	26.8
i3 2100T	25	73	135	581	9.7
GTX Titan	0.4	4.2	57	187	64.1
GTX 750	1.9	14.8	78	169	16.4

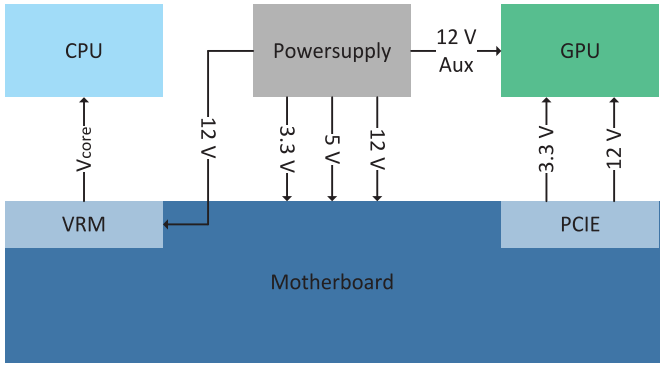


Fig. 4. Schematic of the power supply for general discrete CPU+GPU platforms.

To measure the energy data, we use the power measurement setup in Tang et al. [2013]. The setup is composed of one data acquisition unit for data logging and several current clamps for capturing current changes of CPU and GPU during execution. Each power supply lane of CPU and GPU is monitored by a dedicated current clamp. Figure 4 illustrates an example of the CPU and GPU power supply, where the CPU is power by a dedicated 12V through a voltage regulator module and the GPU card is powered by two lanes (i.e., 3.3V and 12V) through the PCIE interface and one auxiliary 12V lane. The power supply voltages of the monitored lanes are constant during execution. To obtain the energy, we also insert timers in program to measure the execution time of kernels and synchronize with the current data. We use the current, voltage and time to compute the CPU and GPU energy and add them to represent the total system energy. The CPU energy includes the energy consumed by the CPU processor and its power module. The GPU energy is the energy consumed by the whole GPU card. Note that this setup is for validating the PeaPaw performance/energy estimation. It is not needed in actual usage of PeaPaw.

6.2. Machine Parameters

To obtain the performance parameter values for the four heterogeneous platforms, we compute processors' theoretical peak computation performance (measured by Flops/s) and measure their achievable peak memory bandwidth (measured by Bytes/s) by using two popular microbenchmarks, the STREAM benchmark [McCalpin 1995] (i.e., for CPU) and the scalable heterogeneous computing (SHOC) benchmark suite [Danalis et al. 2010] (i.e., for GPU). For t_{fC} and t_{fG}, we apply Equation (6). To estimate t_{bC} and t_{bG}, we measure the peak memory bandwidth of CPUs and GPUs. Then t_{bC} and t_{bG} are set to the inverse of the corresponding memory bandwidth values.

To obtain the energy parameter values, we follow the approach used in Choi et al. [2013]. For each processor, we first design and run a kernel with changeable numbers of flops and bytes. Then, we change the numbers of flops and bytes in that designed kernel and measure the execution time and processor energy usage for each run. We then use linear regression on the obtained data and Equation (3) for getting the values of e_{fC},

Table V. P-Paw Categories

Platform	BL_C	BL_G	P-Paw category
i7+Titan	0.9	10.3	CPU_MEM-GPU_COMP
i7+750	0.9	7.8	CPU_MEM-GPU_COMP
i3+Titan	0.4	10.3	CPU_MEM-GPU_COMP
i3+750	0.4	7.8	CPU_MEM-GPU_COMP

Table VI. E-Paw Categories

Platform	ΔGP_f	ΔGP_b	$p_{sC}+ p_{sG}$	E-Paw category
i7+Titan	311.5	0.65	89.2	CPU_MEM-GPU_MEM
i7+750	64.7	0.85	29.4	Race-to-halt
i3+Titan	244	0.75	78.28	GPU-only
i3+750	50.5	0.93	18.48	GPU-only

e_{bC}, e_{fG}, and e_{bG}. All the machine parameter values of our platforms are summarized in Table IV.

6.3. Case studies

In this subsection, we describe the three case studies: a synthetic application (SA), a linear algebra application (LA), and the data assembly stage (DA) in miniFE. We also discuss the details of the designed WPs for each case study: CO (i.e., CPU-only), GO (i.e., GPU-only), DP (i.e., p-DP), and CP.

6.3.1. Case Study 1: Synthetic Application. We engineered a synthetic benchmark, SA, that has a simple CP-based WP that can fully utilize both CPU and GPU. SA consists of a compute-intensive portion with a memory-intensive portion. The pseudo code of SA is shown below.

```
#define SIZE 6400000
float a[SIZE], b[SIZE], c[SIZE], d[SIZE], e[32][SIZE];
for (i=0;i<SIZE;i++) {
   a[i] = b[i] + c[i];
   for (j=0;j<32;j++) d[i] += pow(e[j][i],63);
}
```

We use OpenMP and CUDA to implement the CPU and GPU code, respectively. For CO, we evenly distribute all the loop iterations to available CPU threads. For GO, each loop iteration is assigned to one GPU thread. For DP, we divide the loop iterations into two parts. One runs on CPU while the other one runs on GPU. We manually tune the number of iterations on each processor such that CPU and GPU can start and complete their workload at the same time. For CP, we put all the loop iterations of vector addition on CPU and the loop iterations of power function on GPU.

To get the essential WP parameter values of each WP, we conduct static code inspection of the four WPs and manually calculate I, I_C, and I_G since the code is very simple. We count each addition and multiplication as one flop and each data access of one data item in an array as 4 bytes. Then a vector addition contributes 1 flop and 12 bytes, and one iteration of the j loop generates 2,048 flops and 256 bytes. We use these values to compute the essential WP parameter values and the results are {7.6, 7.6, 0}, {7.6, 0, 7.6}, {7.6, 7.6, 7.6}, and {7.6, 0.1, 8} for CO, GO, DP, and CP, respectively.

6.3.2. Case Study 2: Linear Algebra Application. Similarly to SA, we developed another two-function application, LA, that is more complicated than SA. LA is a combination of two data-independent phases: matrix multiplication and matrix transpose. There is no computation in the matrix transpose phase. Thus, the matrix multiplication phase is relatively more computation intensive. The input to LA are three matrices: `A`, `B`, and `D`. The output of LA are two matrices: `C` and `E`. The matrix multiplication reads and multiplies matrices `A` and `B` and then store the results in matrix `C`. All these three matrices are 1024×1024 matrices. The matrix transpose phase reads and transposes matrix `D` and then store the results in matrix `E`. The sizes of matrices `D` and `E` are both $8,192 \times 8,192$.

We use the strategy similar to SA to design and implement the four WPs. CO evenly distributes the workload to all available CPU threads. GO maps multiplication and comparison of one data item in the matrix onto one GPU thread. DP evenly divides the workload and maps it to CPU and GPU. Since the matrix multiplication and matrix transpose phases are data independent, CP runs the matrix transpose and matrix multiplication phases on CPU and GPU, respectively. The essential WP parameter values are {0.24, 0.24, 0}, {0.24, 0, 0.24}, {0.24, 0.24, 0.24}, and {0.24, 0, 0.25} for CO, GO, DP, and CP, respectively.

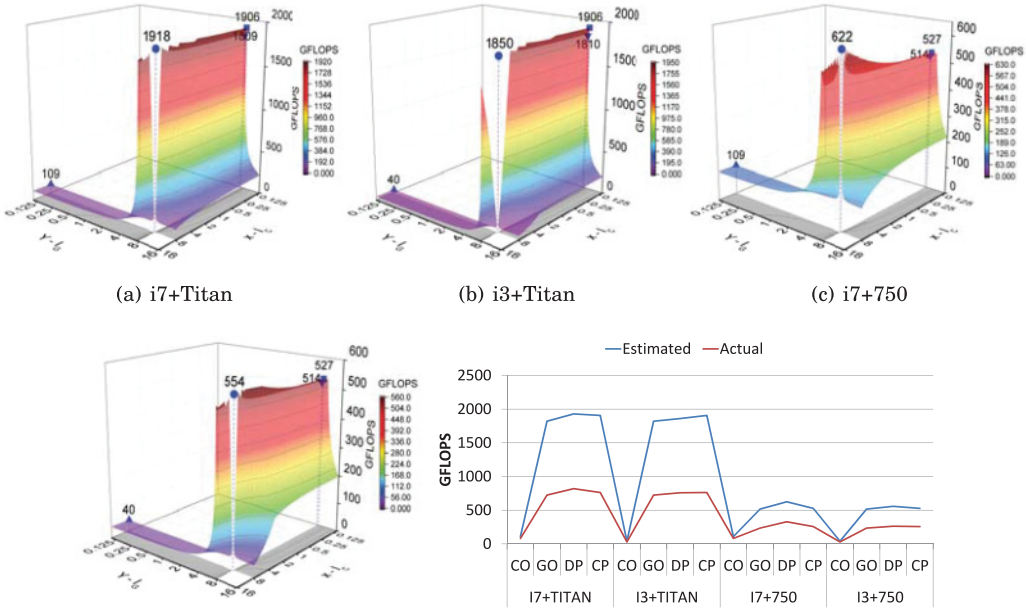
6.3.3. Case Study 3: DA in miniFE. To make the validation more practical, we selected a relatively large and complex program, a stage of miniFE, as our third case study. As discussed in Barrett et al. [2014], miniFE is a proxy FEM application that could be used to predict the performance trends of real-world FEM applications. MiniFE is composed of two major stages: cg-solver and DA. DA has about 1,200 lines of code and is responsible for generating a system of equations including a large sparse global matrix. There are two separate functions in DA: (i) stiffness matrix computation (STC) and (ii) stiffness matrix assembly (SMA). The main body of STC is a three-level nested loop of intensive multiplication. SMA is mainly responsible for searching the positions in the global matrix for each data item in the local stiffness matrices by using binary search. More details of DA in miniFE can be found in Tang et al. [2013].

We follow the ideas in Tang et al. [2013] and select `Omp` and `SmemDA` as the DA designs on CPU and GPU (i.e., CO and GO), respectively. DP evenly divides the workload and maps them to CPU and GPU. CP maps the STC and SMA functions onto GPU and CPU, respectively. The essential WP parameter values are {4.4, 4.4, 0}, {4.4, 0, 4.4}, {4.4, 4.4, 4.4}, and {4.4, 1.1, 4.9} for CO, GO, DP, and CP, respectively.

6.4. PeaPaw Validation

In this subsection, we validate the performance/energy estimation approach as well as the guidelines in the PeaPaw framework. For each case study, we implement the four designed WPs and further measure their actual performance/energy on our CPU+GPU platforms. By comparing the measured data with the estimated results, we show that PeaPaw can faithfully estimate the performance/energy relationships of WPs and provide correct guideline for achieving desirable WPs.

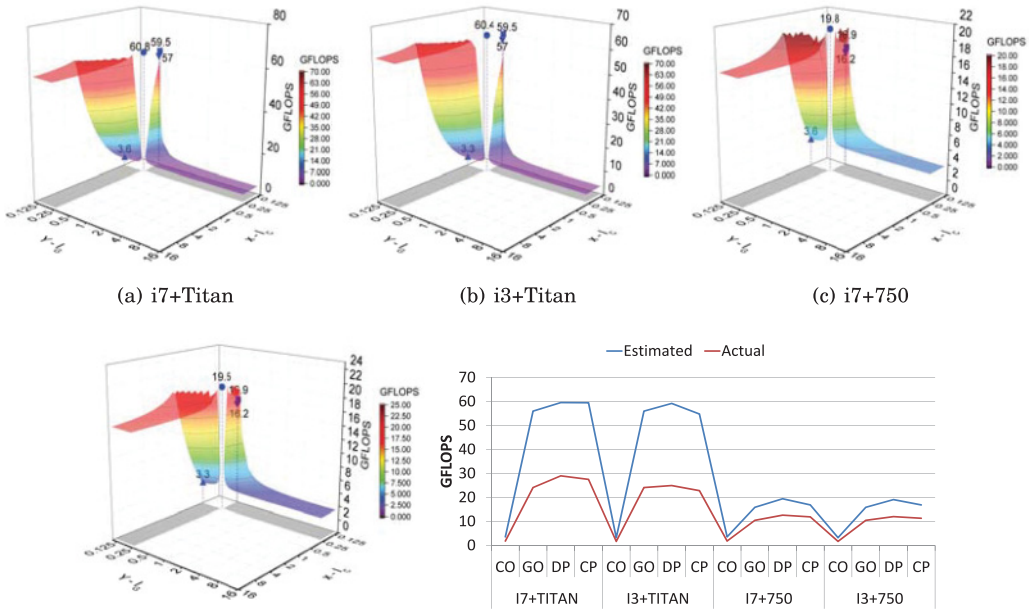
6.4.1. Performance Results. We first use Equations (19) and (20) to estimate the performance of all the designed WPs on our selected heterogeneous platforms, that is, i7+GTX750, i7+Titan, i3+GTX750, and i3+Titan. The machine and essential WP parameter values are given in Table IV and Section 6.3, respectively. Figures 5, 6, and 7 depict the estimated performance for SA, LA, and DA running on the four heterogeneous platforms, respectively. In all the figures, the x-, y- and z-axis represent I_C and I_G, and the estimated performance, respectively. Both the x- and y-axis use logarithmic scale with base of 2. The graphs in these figures share some common features. Specifically, there are two separate surfaces representing the estimated performance of WPs.



(a) i7+Titan (b) i3+Titan (c) i7+750

(d) i3+750 (e) Estimated and actual performance

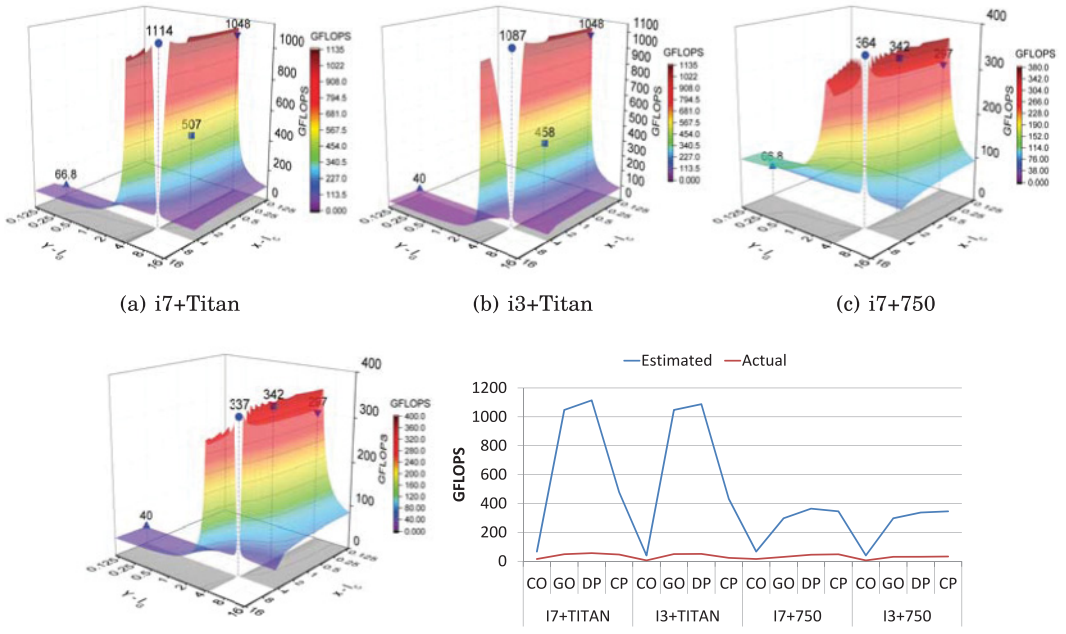
Fig. 5. Estimated and actual performance of SA's WPs on four different platforms.



(a) i7+Titan (b) i3+Titan (c) i7+750

(d) i3+750 (e) Estimated and actual performance

Fig. 6. Estimated and actual performance of LA's WPs on four different platforms.



(a) i7+Titan (b) i3+Titan (c) i7+750

(d) i3+750 (e) Estimated and actual performance

Fig. 7. Estimated and actual performance of DA's WPs on four different platforms.

These surfaces correspond to WPs in two distinct regions of combinations of I_C and I_G, which are shaded in gray in the x-y plane. The other two regions contain no WPs since the I_C and I_G values cannot be both larger or smaller than the given workload's operational intensity. Each graph also includes four points (i.e., triangle, inverted triangle, round, and square) corresponding to the four designed WPs: CO, GO, DP, and CP. As discussed in Section 4, the estimated performance of DP is the aggregation of the performance values of CO and GO.

We next study the absolute accuracy of the estimated performance. Based on the performance plots, the estimated performance can be obtained by checking the corresponding combination of I_C and I_G values. Figures 5(e), 6(e), and 7(e) summarize the measured actual performance (shown as bars) together with the corresponding estimated performance (shown as curves) of SA, LA, and DA. It can be seen that our performance model results in more accurate estimated performance for SA and LA than DA. Specially, all 16 combinations of WPs and platforms achieve 60%, 47%, and 11% of the estimated performance on average for SA, LA and DA, respectively. The reason for DA's low performance modeling accuracy is that DA has very complicated computation and memory behaviors, and these cannot be precisely captured by the simple PeaPaw modeling. On the other hand, for each individual case study, the accuracy of CO is higher than the other WPs since CPU is more capable in overlapping its computation and memory execution. For all the four platforms, CO can achieve 74%, 52%, and 19% of the estimated performance on average for SA, LA, and DA. DA has much lower estimation accuracy since it has large data transfer overhead and its memory access patterns are complicated.

The estimation error mainly comes from the PeaPaw performance/energy models and inaccuracy in the obtained parameter values. For the PeaPaw models, they lack the consideration of CPU/GPU data communication and processor pipeline stalls. More

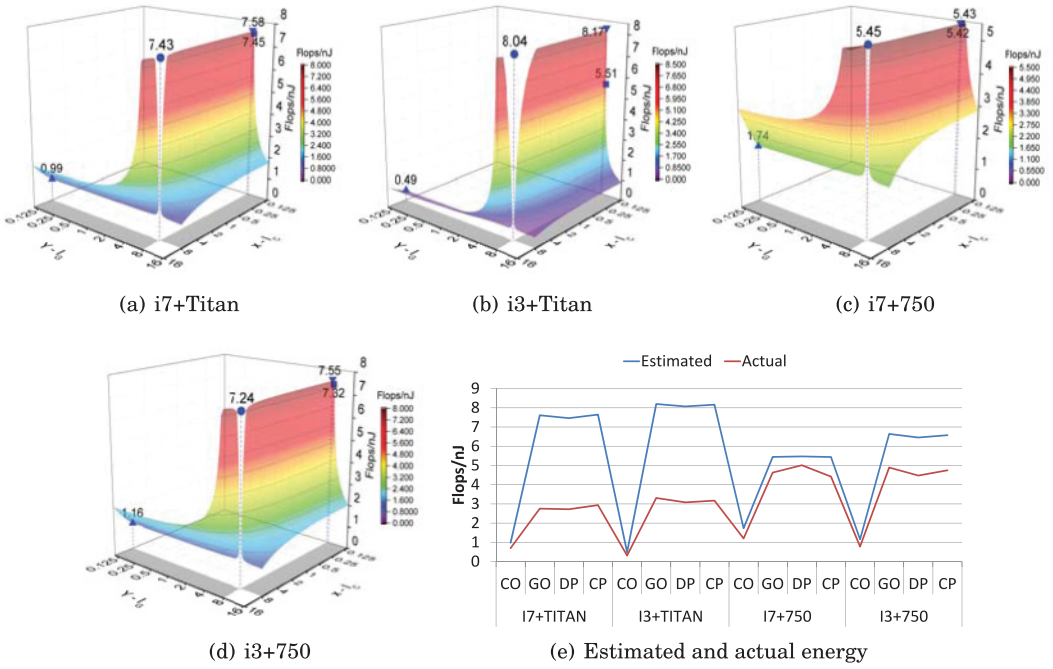
accurate parameter values better reflect the actual execution. Specially, the accuracy can be measured by how much the obtained parameter values differ from the values measured with performance counters. The accuracy of the obtained parameter values can be improved by considering architectural features (e.g., memory coalescing and SIMD) when doing static analysis. Although this consideration improves the performance/energy estimation accuracy of WPs after obtaining WPs, it does not increase the number of workload partitioning guidelines or make the guidelines difficult to be followed.

We now examine the fidelity, which is the relative accuracy of the estimated performance. Since PeaPaw focuses on helping find relatively better WP and equally correlates different design factors, relative accuracy is more important than absolute accuracy for PeaPaw. By fidelity, we mean the correct prediction of the performance relationship among different WPs (i.e., the relative ordering of the WPs in terms of performance). As Figure 5(e) shows, for SA, the actual performance trends completely agree with the estimated performance trends. For example, CP has 1% and 16% lower estimated performance and 7% and 22% higher actual performance than DP on i7+Titan and i7+750, respectively. Also, the estimated performance of CP is 3% higher than DP, and the actual performance value of CP is 2% higher than DP on i3+Titan. Figures 6(e) and 7(e) also confirm the high fidelity of the PeaPaw performance estimation. For LA, CP has 1%, 8%, 15%, and 13% lower estimated performance and 5%, 9%, 7%, and 6% lower actual performance than DP on i7+Titan, i3+Titan, i7+750, and i3+750. Also, GO performs better than CO, and DP performs better than GO on all the platforms. We can observe identical trends on DA. Thus, the estimated performance of WPs can reliably reflect the relative magnitude of the corresponding actual performance.

6.4.2. Energy Results. Similar to validation for performance estimation, we use Equation (26) to estimate the energy efficiency of all combinations of WPs and platforms. The performance and energy parameter values required in the energy estimation are given in Table IV. Figures 8, 9, and 10 illustrate the estimated energy efficiency values (picojoule per flop). First, similarly to the PeaPaw performance estimation, we can observe that energy estimation is relatively more accurate for simpler case studies and CO. For example, all 16 combinations of WPs and platforms can achieve 47%, 40%, and 13% of the estimated energy efficiency on average for SA, LA, and DA, respectively. Also, for all four platforms, CO can achieve 52%, 50%, and 20% of the estimated energy efficiency on average for SA, LA, and DA, respectively. Second, we can see that DA has lower average energy estimation accuracy (achieves 13% of the estimated energy efficiency) than SA and LA (61% and 39%). This is because DA is more complicated and harder to be automatically optimized than SA and LA, which means its essential WP parameter values obtained by code inspection are more inaccurate.

We now examine the fidelity of the estimated energy efficiency. As Figures 8(e), 9(e), and 10(e) show, the actual energy trends completely agree with the estimated energy trends. For SA, CP has 2%, 1%, and 2% higher estimated energy efficiency and 8%, 3%, and 6% higher actual energy efficiency than DP on i7+Titan, i3+Titan, and i3+750, respectively. On i7+750, CP has 1% lower estimated energy efficiency and 12% lower actual energy efficiency than DP. Across different case studies, for i3+750, we can also observe that GO has the lowest energy usage among the four WPs for SA, LA, and DA. Thus, the estimated energy can reliably reflect the relative magnitude of the corresponding actual energy.

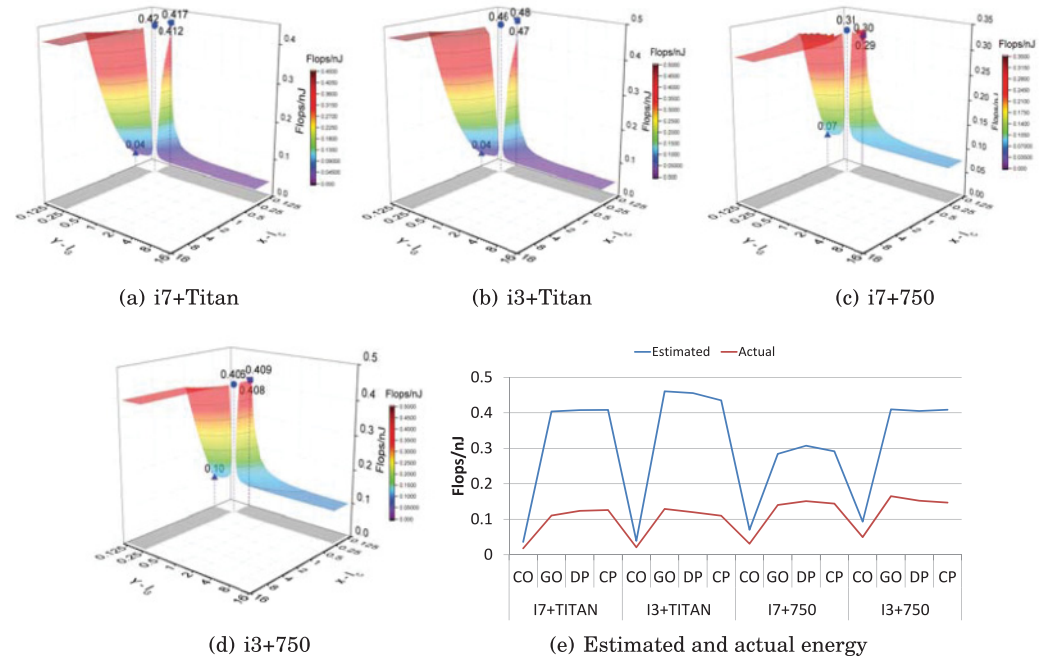
6.4.3. Guideline Validation. In this subsection, we validate whether the proposed workload partitioning guidelines can identify the right WP for better performance or energy.



(a) i7+Titan (b) i3+Titan (c) i7+750

(d) i3+750 (e) Estimated and actual energy

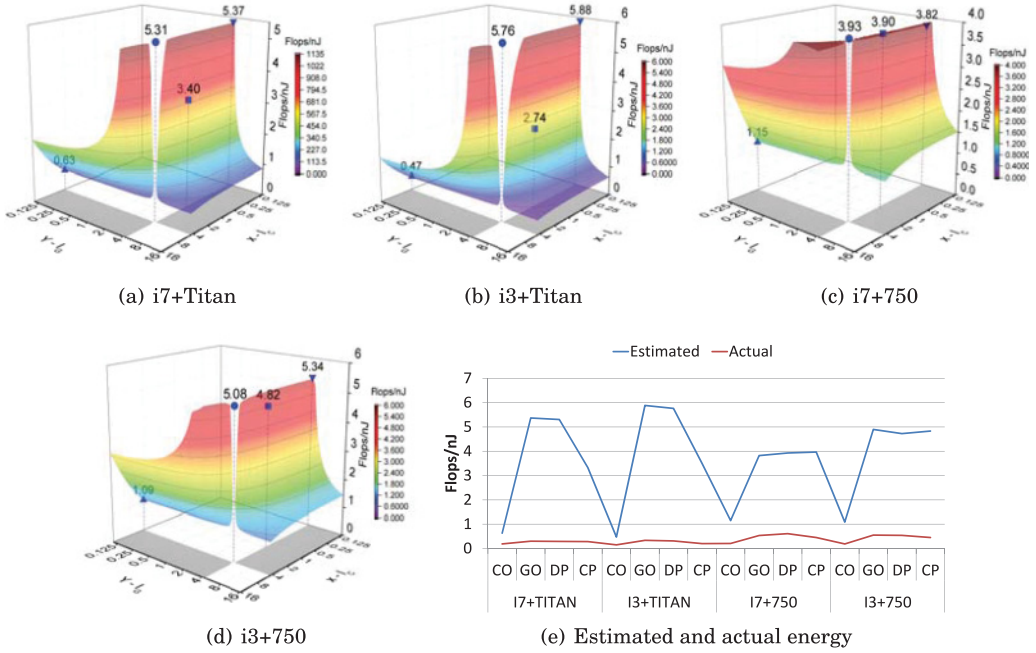
Fig. 8. Estimated and actual energy of SA's WPs on four different platforms.



(a) i7+Titan (b) i3+Titan (c) i7+750

(d) i3+750 (e) Estimated and actual energy

Fig. 9. Estimated and actual energy of LA's WPs on four different platforms.



(a) i7+Titan (b) i3+Titan (c) i7+750

(d) i3+750 (e) Estimated and actual energy

Fig. 10. Estimated and actual energy of DA's WPs on four different platforms.

First, we validate the effectiveness of the performance-oriented guidelines by examining if the suggest WP could achieve the best actual performance. The P-Paw and E-Paw categories of our platforms are shown in Tables V and VI, respectively. Our platforms fail to include all the P-Paw and E-Paw categories due to the reason that current GPUs provide much higher machine balance and energy efficient than current CPUs. According to the PeaPaw guidelines, all four platforms should use CP and map the more computation-intensive code onto the CPU for better performance. We have followed this guideline for designing CP for the three case studies. However, as Figures 5(e), 6(e), and 7(e) show, among the 12 combinations of case studies and platforms, CP achieves better performance than CO, GO, and DP only for SA on i3+Titan and DA on i3+750. The reason is that the performance-oriented guidelines are workload independent and the guideline associated with the `CPU_MEM-GPU_COMP` category only provides a general direction for workload partitioning. In other words, designing one CP does not necessarily lead to the best-performance WP. For example, CP has lower performance than DP on all four platforms for LA, since LA has much lower I than SA and DA and all our CPUs and GPUs are already memory-bounded in terms of performance.

We now validate the effectiveness of the energy-oriented guidelines. For both i3+Titan and i3+750, the guidelines suggest using GO for the given workload. It can be seen from Figures 8, 9, and 10 that GO has the highest energy efficiency among the four WPs for all SA, LA, and DA on both i3+Titan and i3+750. For i7+750, DP has higher energy efficiency than the other WPs since DP achieves the best performance for SA, LA, and DA. This proves the selected guideline of `Race-to-halt` for i7+750. For i7+Titan, the guideline suggests balancing the memory operations on the CPU and GPU. This guideline is similar with the performance-oriented ones, which only provide general direction for CP. Thus, we can see that the CP has the highest energy efficiency for DA on i7+Titan and SA on i7+Titan, i3+Titan, and i3+750.

7. CONCLUSION AND FUTURE WORK

In this article, we propose the PeaPaw framework to help application developers partition workload on CPU+GPU platforms and identify WPs with better performance or energy efficiency. In PeaPaw, we have extended the roofline models of performance/energy to estimate the performance and energy efficiency of WPs on heterogeneous platforms. We also derived two sets of workload partitioning guidelines based on the PeaPaw models. To help select appropriate guideline for a given platform and design goal, we propose the P-PAW and E-PAW categories for classifying heterogeneous platforms. By applying the PeaPaw framework, we have conducted three case studies and designed and implemented four WPs for each case study. The actual performance/energy results of the implemented WPs validate the effectiveness of the PeaPaw performance/energy modeling and workload partitioning guidelines.

For future work, we plan to increase the accuracy of the performance/energy estimation for WPs. The major limitation of our current PeaPaw models is that the data communication between CPU and GPU is not considered. Thus, new parameters that represent the CPU/GPU communication performance and energy will be added to the new models.

REFERENCES

Federico Angiolini, Jianjiang Ceng, Rainer Leupers, Federico Ferrari, Cesare Ferri, and Luca Benini. 2006. An integrated open framework for heterogeneous MPSoC design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*. European Design and Automation Association, 1145–1150.

R. F. Barrett, P. S. Crozier, D. W. Doerfler, M. A. Heroux, P. T. Lin, H. K. Thornquist, T. G. Trucano, and C. T. Vaughan. 2014. Assessing the validity of the role of mini-applications in predicting key performance characteristics of scientific and engineering applications. *J. Parallel Distrib. Comput.* 75 (2014).

Andre R. Brodtkorb, Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik, and Olaf O. Storaasli. 2010. State-of-the-art in heterogeneous computing. *Sci. Program.* 18, 1 (2010), 1–33.

David Callahan, John Cocke, and Ken Kennedy. 1988. Estimating interlock and improving balance for pipelined architectures. *J. Parallel Distrib. Comput.* 5, 4 (1988), 334–358.

Doris Chen and Deshanand Singh. 2012. Invited paper: Using OpenCL to evaluate the efficiency of CPUS, GPUS and FPGAS for information filtering. In *Proceedings of the 2012 22nd International Conference on Field Programmable Logic and Applications (FPL'12)*. IEEE, 5–12.

Jee Whan Choi, Daniel Bedard, Robert Fowler, and Richard Vuduc. 2013. A roofline model of energy. In *IPDPS-27*. IEEE, 661–672.

Eric S. Chung, Peter A. Milder, James C. Hoe, and Ken Mai. 2010. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 225–236.

Jason Cong and Bo Yuan. 2012. Energy-efficient scheduling on heterogeneous multi-core architectures. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*. ACM, 345–350.

Mayank Daga and Mark Nutter. 2012. Exploiting coarse-grained parallelism in b+ tree searches on an apu. In *SCC*. IEEE, 240–247.

Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *GPGPU-3*. ACM, 63–74.

Rong Ge, Xizhou Feng, Martin Burtscher, and Ziliang Zong. 2014. PEACH: A model for performance and energy aware cooperative hybrid computing. In *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 24.

Michel Goraczko, Jie Liu, Dimitrios Lymberopoulos, Slobodan Matic, Bodhi Priyantha, and Feng Zhao. 2008. Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *Proceedings of the 45th Annual Design Automation Conference*. ACM, 191–196.

Ivan Grasso, Klaus Kofler, Biagio Cosenza, and Thomas Fahringer. 2013. Automatic problem size sensitive task partitioning on heterogeneous parallel systems. In *ACM SIGPLAN Not.* Vol. 48. ACM, 281–282.

Chris Gregg, Michael Boyer, Kim Hazelwood, and Kevin Skadron. 2011. Dynamic heterogeneous scheduling decisions using historical runtime data. In *Workshop on Applications for Multi- and Many-Core Processors (A4MMC)*.

Dominik Grewe and Michael F. P. OBoyle. 2011. A static task partitioning approach for heterogeneous systems using OpenCL. In *Compiler Construction*. Springer, 286–305.

Michael A. Heroux, Douglas W. Doerfler, Paul S. Crozier, James M. Willenbring, H. Carter Edwards, Alan Williams, Mahesh Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich. 2009. *Improving performance via mini-applications*. Tech. Rep. SAND2009-5574, Sandia National Laboratories.

C.-H. Hsu, Wu-chun Feng, and Jeremy S. Archuleta. 2005. Towards efficient supercomputing: A quest for the right metric. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005.*. IEEE, 8–pp.

S. Huang and W. Feng. 2009. Energy-efficient cluster computing via accurate workload characterization. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 68–75.

Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. 2014. Cache-aware roofline model: Upgrading the loft. *Comput. Arch. Lett.* 13, 1 (2014), 21–24.

David Koufaty, Dheeraj Reddy, and Scott Hahn. 2010. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European Conference on Computer Systems*. ACM, 125–138.

Qiang Liu and Wayne Luk. 2012. Heterogeneous systems for energy efficient scientific computing. In *Reconfigurable Computing: Architectures, Tools and Applications*. Springer, 64–75.

Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *ICPP-41*. IEEE, 48–57.

John D. McCalpin. 1995. Memory bandwidth and machine balance in current high performance computers. *IEEE Technical Committee on Computer Architecture Newsletter* (1995), 19–25.

Cedric Nugteren and Henk Corporaal. 2012. The boat hull model: Adapting the roofline model to enable performance prediction for parallel computing. In *ACM Sigplan Notices*, Vol. 47. ACM, 291–292.

Francesco Paterna, Andrea Acquaviva, Alberto Caprara, Francesco Papariello, Giuseppe Desoli, and Luca Benini. 2012. Variability-aware task allocation for energy-efficient quality of service provisioning in embedded streaming multimedia applications. *IEEE Trans. Comput.* 61, 7 (2012), 939–953.

Mahsan Rofouei, Thanos Stathopoulos, Sebi Ryffel, William Kaiser, and Majid Sarrafzadeh. 2008. Energy-aware high performance computing with graphic processing units. In *Proceedings of the Workshop on Power Aware Computing and System*.

Wasuwee Sodsong, Jingun Hong, Seongwook Chung, Yeongkyu Lim, Shin-Dug Kim, and Bernd Burgstaller. 2014. Dynamic partitioning-based JPEG decompression on heterogeneous multicore architectures. In *PMAM*. ACM, 80.

Li Tang, X. Sharon Hu, Danny Z. Chen, Michael Niemier, Richard F. Barrett, Simon D. Hammond, and Genie Hsieh. 2013. GPU acceleration of data assembly in finite element methods and its energy implications. In *ASAP-24*. IEEE, 321–328.

Walter F. Tichy. 1982. Design, implementation, and evaluation of a revision control system. In *Proceedings of the 6th International Conference on Software Engineering*. IEEE Computer Society Press, 58–67.

Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

Ziming Zhong, Vladimir Rychkov, and Alexey Lastovetsky. 2012. Data partitioning on heterogeneous multicore and multi-GPU systems using functional performance models of data-parallel applications. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER'12)*. IEEE, 191–199.

Received May 2016; revised August 2016; accepted September 2016