Performance and Energy Modeling for Cooperative Hybrid Computing

Rong Ge and Xizhou Feng
Dept. of Mathematics, Statistics and Computer Science
Marquette University
Milwaukee, WI
{*rong.ge,xizhou.feng*}*@marquette.edu*

Martin Burtscher and Ziliang Zong
Department of Computer Science
Texas State University
San Marcos, TX
{*burtscher, ziliang*}*@txstate.edu*

***Abstract*—Accelerator-based heterogeneous systems can provide high performance and energy efficiency, both of which are key design goals in high performance computing. To fully realize the potential of heterogeneous architectures, software must optimally exploit the hosts' and accelerators' processing *and* power-saving capabilities. Yet, previous studies mainly focus on using hosts and accelerators to boost application performance. Power-saving features to improve the energy efficiency of parallel programs, such as Dynamic Voltage and Frequency Scaling (DVFS), remain largely unexplored.**

Recognizing that energy efficiency is a different objective than performance and should therefore be independently pursued, we study how to judiciously distribute computation between hosts and accelerators for energy optimization. We further explore energy-saving scheduling in combination with computation distribution for even larger gains. Moreover, we present *PEACH*, an analytical model for Performance and Energy Aware Cooperative Hybrid computing. With just a few system- and application-dependent parameters, *PEACH* accurately captures the performance and energy impact of computation distribution and energy-saving scheduling to quickly identify the optimal coupled strategy for achieving the best performance or the lowest energy consumption. *PEACH* thus eliminates the need for extensive profiling and measurement. Experimental results from two GPU-accelerated heterogeneous systems show that *PEACH* predicts the performance and energy of the studied codes with less than 3% error and successfully identifies the optimal strategy for a given objective.

***Keywords*-Energy-Efficient Computing, Heterogeneous Computing, Performance and Energy Modeling**

I. Introduction

To overcome the performance and power limitations of conventional general-purpose microprocessors, many high-performance systems employ accelerator-based heterogeneous architectures. On the most recent TOP500 list, four of the top ten supercomputers include either Xeon Phi or GPU accelerators. Of these heterogeneous systems, each node integrates at least two types of computational engines—multi-core-based host processing units and many-core-based accelerating units—that can work collaboratively to increase computational performance and energy efficiency.

Adapting parallel computation models to heterogeneous architectures presents many challenges, one of which is how to orchestrate computations between host processing units and accelerating units. Many heterogeneous applications use an offload computation model, which breaks the program execution into phases and offloads highly parallel and compute-intensive phases to accelerators. While this model is effective for certain codes, it suffers from two main drawbacks. First, data transfers between hosts and accelerators incur time overhead, which may offset the performance gain from using accelerators. Second, host processing units may be left idle or underutilized during the off-loading phase but still consume a significant amount of power.

The cooperative hybrid computation model employs all processing units in the hosts and accelerators to concurrently execute compute-intensive phases. It has better resource utilization and may deliver higher performance by aggregating the processing capabilities of all available cores and by reducing data transfers [15, 16].

While improving performance is important, maximizing energy efficiency is also critical. Power is a key constraint at many levels in high-performance computing (HPC) systems. Improving energy efficiency not only reduces energy cost but also allows for greater performance given the same power budget. Previous studies neither sufficiently address the energy-efficiency aspect nor fully exploit the power-aware capabilities available on the latest heterogeneous systems, e.g., those with DVFS-capable Nvidia K20 GPUs.

In this paper, we present models and methods for performance- and energy-efficient heterogeneous computing. Specifically, we focus on GPU-accelerated systems and describe *PEACH*—a model for **P**erformance and **E**nergy **A**ware **C**ooperative **H**ybrid computing. *PEACH* explores two strategies: computation distribution that splits the application workload between the hosts and the accelerators, and energy-saving scheduling that adapts the host and accelerator speeds to the demand of the workload. These two strategies can be used individually or collectively. *PEACH* provides analytical models that capture the dependencies between performance measures and hardware/software parameters.

We evaluate the strategies and models on three benchmarks and two power-aware heterogeneous systems. Both systems use Intel Sandy Bridge processors as host processing units. One system is accelerated with an Nvidia Tesla C2075 GPU and the other with a Tesla K20 GPU. We show that, for compute-intensive applications such as matrix multiplication, the energy-optimal computation distribution is different from the performance-optimal computation distribution when the default CPU and GPU speeds are used; the former saves 25% energy with a 22% performance degradation compared to the latter, which delivers the maximum performance among all possible configurations. Coupling energy-saving scheduling with computation distribution can improve energy efficiency. For example, the energy-optimal coupled strategy can save 33% energy with only a 13% performance reduction compared to the strategy that delivers the maximum performance. Our main findings are as follows.

- Intelligently distributing computation over both host and accelerator processing units can achieve better performance and/or energy efficiency than CPU-only or GPU-only execution for compute-intensive applications.
- Computation distributions that are optimized for application performance differ from those optimized for energy efficiency. For matrix multiplication on one of our systems, the performance-optimal computation distribution is [22% CPU : 78% GPU] whereas the energy-optimal distribution is [0% CPU : 100% GPU] when the host and accelerator run at their default speeds.
- Coupling energy-saving scheduling with computation distribution can further improve performance and energy efficiency. For instance, the energy-optimal coupled strategy runs 9% faster and saves 8% more energy than its counterpart that uses computation distribution only.
- The *PEACH* model is easy to use and accurately captures the performance and energy effects of computation distribution and energy-saving scheduling. The prediction error is within 3% on our benchmarks.
- With just a few system- and application-dependent parameters, *PEACH* quickly identifies the best individual and coupled strategies for optimizing performance or energy without the need for extensive measurement.
- The *PEACH* models and the experimental results indicate that the performance and energy-efficiency gains are platform and benchmark dependent, suggesting that efficient designs require system- and program-specific adaptation.

This paper is organized as follows. Sections II and III present the machine abstraction and the *PEACH* models. Section IV illustrates model usages. Section V introduces the experimental platforms. Sections VI and VII present the model evaluations and applications. Section VIII discusses related work. Section IX draws conclusions.

II. Model Abstraction

A. Hardware Abstraction

Many modern accelerator-based heterogeneous systems can be abstracted as power-aware cooperative hybrid computers. Such machines typically possess a tree-like hierarchical structure, where a non-leaf component comprises multiple lower-level components. Components at certain levels are power scalable, meaning that they support multiple performance/power states through features like DVFS.

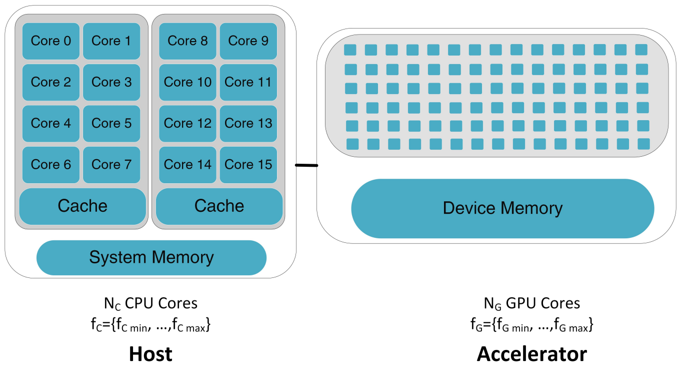


Figure 1: A heterogeneous system with a CPU-based host and a GPU-based accelerator. The CPU and GPU cores can operate at multiple frequencies.

This abstraction applies to many heterogeneous systems, including those accelerated with GPUs or Xeon Phis. For practical reasons, we concentrate on GPUs in this paper. Figure 1 shows an example of such a system, which consists of a host with multiple multi-core CPUs and a GPU accelerator with many processing cores. To minimize complexity, our model entities are at the node level, consist of a GPU and a host, and disregard the cache hierarchy. If, for algorithmic or workload reasons, a computation does not use all available cores on either the host or the accelerator, we discount that entity's effective computing capacity correspondingly. We assume that DVFS is only available at the host and GPU level such that all CPU cores and all GPU cores run at the same frequency as their siblings.

B. Workload Abstraction

We use the Phased Hierarchical Task Graph (PHTG) [3] to model programs running on a GPU-based heterogeneous system. In this model, program execution is divided into phases where each phase comprises multiple nested parallel tasks. We assume that the same workload can run on GPUs and CPUs and that tasks within phases with sufficient concurrency will be distributed across the host and accelerator.

As both the PHGT program abstraction and the machine abstraction use a similar hierarchical structure, the task of mapping a workload to a heterogeneous system is greatly simplified. With these two abstractions, we can formulate the problem of performance- and energy-aware cooperative hybrid computing as follows:

Given a workload W, an accelerator-based heterogeneous machine M, and a user-specified efficiency metric η, find a tuple (f_C, f_G, α) that maximizes η.

Here, α denotes the portion of the workload that is distributed to the accelerator, f_C is the CPU speed, f_G is the GPU speed, and the efficiency metric η can either be performance or energy efficiency or a combination thereof.

III. The *PEACH* Model

The *PEACH* model captures the performance, energy, and efficiency effects of computation distribution, CPU DVFS, and GPU DVFS for cooperative hybrid computation. For quick reference, we list the model parameters in Table I.

A. Modeling Base Performance

We consider execution phases that comprise a sufficiently large number of subtasks that can be executed in parallel. Such an execution phase can be either a distinct phase or a combination of multiple consecutive distinct phases. Let W be the total number of subtasks to be executed. We partition the subtasks into two groups: group G, which contains the fraction α of the subtasks, and group C, which contains the remaining subtasks. Group G is assigned to the GPU accelerators and group C to the host CPUs. This process creates the following computation distribution:

$$W = W_C + W_G \tag{1}$$

$$W_G = \alpha \cdot W \tag{2}$$

$$W_C = (1 - \alpha) \cdot W \tag{3}$$

Here, α denotes the portion of the workload that is distributed to the GPUs and $0 \leq \alpha \leq 1$, and W_G and W_C denote the amount of workload distributed to GPUs and CPUs, respectively.

For a given workload W, we use the compute rate R to summarize the combined computational capability provided by the host CPUs and the accelerator GPUs. R describes the number of subtasks of workload W that the host and the accelerator complete in one second. It is a function of the hardware configuration and workload characteristics. Note that different workloads may define their units of computation (i.e., subtasks) differently. While R can vary with the type of workload, it is relatively constant for a fixed type of workload as long as there is sufficient computation to saturate the processing units.

Let R, R_C, and R_G be the compute rates of the hybrid system, the CPUs, and the GPUs. Let T, T_C, and T_G be the execution times of the hybrid system, the CPUs, and the GPUs to complete their assigned workload portions. Thus,

$$T_C = W_C / R_C \tag{4}$$

$$T_G = W_G / R_G \tag{5}$$

$$T = max\{T_C,\ T_G + T_{OH}\} \tag{6}$$

We note two main points for Eq. (6). First, offloading tasks onto the accelerator incurs an overhead time (T_{OH}). T_{OH} includes the time to transfer data and to launch the kernel. If $T_{OH} \ll T_G$, ignoring T_{OH} does not affect the model's accuracy. Second, when distributing the workload across CPUs and GPUs, the overall execution time is the longer of the CPU execution time and the sum of GPU execution time and the offloading overhead.

Using Eqs. (1)-(6), we can calculate the compute rate of cooperative hybrid computing

$$R = \quad \frac{W}{T} = \frac{1}{max(\frac{(1-\alpha)}{R_C}, \frac{\alpha}{R_G} + \frac{T_{OH}}{W})} \tag{7}$$

Eq. (7) can be simplified by ignoring T_{OH} when $T_{OH} \ll T_G$

$$R = \quad \frac{W}{T} = \frac{1}{max(\frac{(1-\alpha)}{R_C}, \frac{\alpha}{R_G})} \tag{8}$$

B. Modeling Base Energy

We break down the system power into the sum of the power of three components as follows.

$$P = \sum_{i \in (C,G,M)} P_i \tag{9}$$

C represents the CPU, G the GPU, and M the host memory as well as other parts of the host.

Each component power P_i can be further broken down into a base power $P0_i$ and a dynamic power Pd_i, where the former is independent of the workload and the latter is due to the execution of the workload W. Because each component's base power is workload independent and fixed for a given speed of the processing units, the individual base powers can be merged into a single system base power

$$P0 = \sum_{i \in (C,G,M)} P0_i \tag{10}$$

As shown in Figure 2, during application execution power is consumed in four possible ways : (1) the system consumes base power P_0 over the entire execution time T, (2) the CPU consumes dynamic power Pd_C during the CPU execution time T_C, (3) the GPU consumes dynamic power Pd_G during the GPU execution time T_G, and (4) the CPU draws power P_{OH} in addition to $P0_C$ for hosting the GPU execution if the GPU portion takes longer than the CPU portion, in other words, if $T_G > T_C$. The system energy is the sum of these four components.

$$E = T \cdot P0 + T_C \cdot Pd_C + T_G \cdot Pd_G + (T_G - T_C) \cdot P_{OH} \tag{11}$$

Since all variables in Eq. (11) can be directly measured or derived, the system energy can be computed using Eq. (11).

C. Modeling Energy Efficiency

We adopt the typical approach used in energy-efficiency studies and define the energy efficiency, denoted by η, as the workload to energy ratio, i.e., $\frac{W}{E}$ with a unit of OPs/joule, or, equivalently, the compute rate to power ratio $\frac{R_{avg}}{P_{avg}}$ expressed in GFLOPS/watt or OPs/watt.

In *PEACH*, the effective energy efficiency η of the hybrid system can be derived from Eqs. 4 and 11 as follows.

$$\eta = \frac{W}{E} = \frac{1}{P0 \cdot \max(\frac{1-\alpha}{R_C}, \frac{\alpha}{R_G}) + \frac{(1-\alpha)Pd_C}{R_C} + \frac{\alpha Pd_G}{R_G} + P_{OH} \cdot \max(\frac{\alpha}{R_G} - \frac{1-\alpha}{R_C}, 0)} \tag{12}$$

Table I: Parameters used in the *PEACH* models

Parameter	Description	Affiliation	Notation	Affected by
W	Workload			
α	% of W on accelerators	Host CPUs	$1-\alpha$	
		Accelerators	α	
N	Concurrency	Host CPUs	N_C	
		Accelerators	N_G	
f	Frequency	Host CPUs	f_C	
		Accelerators	f_G	
R	Compute rate	Host CPUs	R_C	W, f_C, N_C
		Accelerators	R_G	W, f_G, N_G
$P0$	Base power	Host CPUs	$P0_C$	f_C, N_C
		Host memory + other	$P0_M$	
		Accelerator device	$P0_G$	f_G, N_G
Pd	Activity power	Host	Pd_C	W, f_C, N_C
		Host memory + other	Pd_M	W, f_C, N_C
		Accelerator device	Pd_G	W, f_G, N_G

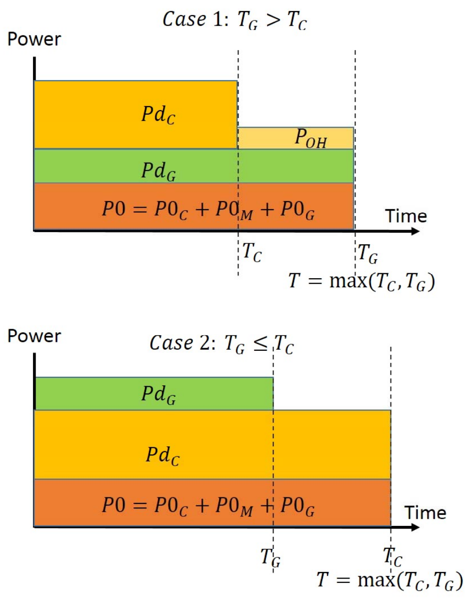


Figure 2: Power consumption during application execution if T_{OH} is negligible.

For simplicity, Eq. 12 assumes the offloading time overhead T_{OH} to be negligible compared to T_G. Note that we include T_{OH} in our case studies for workloads and distributions where it is comparable to T_G. Eq. 12 shows that the system-level energy efficiency is determined by multiple factors, including the computation distribution α, the application performance and energy characteristics on CPUs and GPUs, and the system/component base power.

We define the CPU energy efficiency η_C and the GPU energy efficiency η_G similarly. These metrics apply to the respective component and exclude the base power, which is already included in the system-wide base power. Thus, $\eta_C = \frac{R_C}{Pd_C}$ for CPUs and $\eta_G = \frac{R_G}{Pd_G}$ for GPUs.

D. Integrating DVFS into the Model

DVFS is an effective power saving technology. DVFS-capable components support a set of performance/power states, each with an associated frequency/voltage pair. Higher performance states normally run at higher frequencies and have higher compute rates but draw more power.

DVFS directly affects the *PEACH* model parameters. Specifically, under CPU DVFS, the model parameters R_C, $P0_C$, and Pd_C become functions of the CPU frequency. The same applies to DVFS-capable GPUs such as the Tesla K20. For example, the K20c supports six power/performance states and can freely switch between them during execution. With GPU DVFS, the model parameters R_G, $P0_G$, and Pd_G become functions of the GPU frequency.

By modeling parameters such as the computing rate and the power as functions of the CPU or GPU frequency, the performance, energy, and energy efficiency in *PEACH* become functions of α, f_C, and f_G. The formulas remain as shown above except that parameters like R_C, R_G, Pd_C, and Pd_G change depending on the chosen device frequency.

IV. Model Usage

A. Identifying the Optimal Workload Distribution

We first focus on situations where the CPU and GPU run at their default speeds without any energy saving scheduling. In this case, the *PEACH* models can be employed to identify the optimal computation distribution for a user-selected metric, e.g., performance, energy, energy efficiency, or a combination thereof. Once the distribution α is determined, the performance and energy efficiency measures can be computed directly. This is a significant extension over previous cooperative hybrid computing studies.

1) Finding the Best-Performance Distribution: With GPU-accelerated cooperative hybrid computing, the best performance is achieved when the CPU and GPU executions fully overlap, in other words, $T_C = T_G$ or $\frac{(1-\alpha)\cdot W}{R_C} = \frac{\alpha \cdot W}{R_G}$. Thus, the best-performance distribution α_{perf} is

$$\alpha_{perf} = \frac{R_G}{R_C + R_G} \tag{13}$$

Using the α_{perf} distribution, the application reaches its highest performance $R = R_C + R_G$, finishes in the shortest time $\frac{W}{R_C+R_G}$, and yields a system energy efficiency of

$$\eta = \frac{R_C + R_G}{P_0 + Pd_C + Pd_G} \tag{14}$$

2) Finding the Best-Energy Distribution: For a given workload, the least-energy distribution α_{eff} is the same as the most energy-efficient distribution. α_{eff} can be computed as the minimum of the following energy function.

$$\min_{\alpha_{eff} \in [0,100]} \quad P0 \cdot \max(\tfrac{1-\alpha}{R_C}, \tfrac{\alpha}{R_G}) + \tfrac{(1-\alpha) \cdot Pd_C}{R_C} + \tfrac{\alpha \cdot Pd_G}{R_G} + P_{OH} \cdot \max(\tfrac{\alpha}{R_G} - \tfrac{1-\alpha}{R_C}, 0) \tag{15}$$

We stress that α_{perf} and α_{eff} are not necessarily equal. If they are different, the best-performance distribution consumes more power and energy than the best-energy distribution for the same amount of work.

B. Identifying the Best Coupled Frequency and Distribution

The CPUs and GPUs of power-aware heterogeneous systems can run at multiple speeds. If there are L_C CPU speeds and L_G GPU speeds available, then there are a total of $L_C \cdot L_G$ possible (f_C, f_G) pairs. For each pair, there exists a best-performance distribution and a best-energy distribution.

We can use *PEACH* to identify the best coupled frequency and distribution, denoted by the tuple (f_C^*, f_G^*, α^*), either to maximize application performance or to maximize energy efficiency, as outlined in the following procedure.

1) For each valid pair (f_C, f_G), use *PEACH* to find $R_{max}(f_C, f_G)$ and the corresponding (f_C, f_G, α_{perf}).
2) Find the maximum value out of all $R_{max}(f_C, f_G)$ and report the corresponding $(f_C^*, f_G^*, \alpha_{perf}^*)$.
3) For each valid pair (f_C, f_G), use *PEACH* to find $\eta_{max}(f_C, f_G)$ and the corresponding (f_C, f_G, α_{eff}).
4) Find the maximum value out of all $\eta_{max}(f_C, f_G)$ and report the corresponding $(f_C^*, f_G^*, \alpha_{eff}^*)$.

V. Experimental Platform and Environment

A. Experimental Platform

All experiments are conducted on the two GPU-accelerated heterogeneous systems described in Table II. Both systems use dual eight-core Xeon Sandy Bridge E5-2670 processors on the hosts. System I is accelerated with a Tesla K20c Kepler-based GPU and System II with a Tesla C2075 Fermi-based GPU. Though each E5-2670 core supports two threads, we disabled this hyperthreading feature to simplify the DVFS control and the performance analysis.

The CPU cores support 16 DVFS states, ranging from 1.2 GHz to 2.6 GHz in 0.1 GHz increments and, additionally, 2.601 GHz. The nominal frequency is 2.6 GHz. The 2.601 GHz state represents TurboBoost mode with an actual frequency of up to 3.3 GHz. The K20c's cores and memory are also DVFS capable. Table III summarizes the available states. The default state is 705 MHz core speed and 2600 MHz memory speed. Note that the core and memory speeds must be set together. DVFS is not supported by the C2075 card. The `cpufreq` interface is used to perform CPU DVFS scheduling, and the `nvidia-smi` utility is used to perform GPU DVFS scheduling.

Table II: Experimental platforms. System I: a 16-core host + a K20 GPU; System II: a 16-core host + a C2075 GPU.

	Host	**K20**	**C2075**
Architecture	Intel E5-2670	Nvidia GK110	Nvidia C2075
#Cores	16	2496	448
Default Freq.	2.6 GHz	705 MHz	1.15 GHz
DVFS	Yes	Yes	No
Mem. Size	32 GB	5 GB	6 GB
Threading API	OpenMP	CUDA	CUDA
Peak Perf.	332.8 GFLOPS	1.17 TFLOPS	515 GFLOPS
Mem. BW	51.2 GB/s	208 GB/s	144 GB/s
Compiler	gcc 4.4.6	nvcc 5.5	nvcc 5.5
OS & Driver	CentOS 6.1	Driver 331.20	Driver 331.20

Table III: K20c supported memory/core frequency pairs

Core freq. (MHz)	758	705	666	640	614	324
Mem. freq. (MHz)	2600	2600	2600	2600	2600	324

B. Benchmark Selection

We use the three benchmarks described in Table IV for our experimental evaluation. They can all distribute the computation to GPUs and CPUs according to a user-specified distribution ratio. In each program, the CPU portion is parallelized with OpenMP, and the GPU portion is based on CUDA SDK code or other published implementations. For example, MatrixMultiply uses the OpenBLAS implementation [2] on the CPU and a CUDA implementation on the GPU. The TSP code stems from the ILCS Framework [4].

C. Performance, Power, and Energy Profiling

We directly collect all performance and power data. Performance results, including the execution time and derived computation rates like GFLOPS and the task completion rate, are obtained by instrumenting the programs. Power data are collected using the eTune power-performance profiling framework [7]. Multiple streams of power samples are recorded, including the power consumed by the entire system, the two CPU sockets, the two memory controllers, and the GPU card. eTune provides scripts to control the profiling, i.e., start, stop, annotation, and logging. These power data are synchronized with the application execution and with each other via timestamps.

Specifically, we sampled power streams from the following sources. System power data are sampled by two external WattsUp power meters that are plugged in between the computer power supplies and a power outlet. The CPU power data stem from embedded power sensors on the Sandy Bridge processors and are gathered via the Running Average Power Limit (RAPL) interface [10]. GPU power data come from the embedded power sensor on the K20c card via Nvidia's System Management Interface (`nvidia-smi`). In this study, we use a sampling interval length of one second.

Table IV: Applications and Benchmarks under Study

Benchmark	Description	HPC/HTC	Characterization	Origin
MatrixMultiply	Dense matrix multiplication	HPC	Floating-point, compute-intensive	CUDA SDK [1]
TSP	Traveling salesman problem	HTC	Integer, compute-intensive	ILCS [4]
MatrixTranspose	Dense matrix transpose	HPC	Double precision floating-point, memory-intensive	CUDA SDK

To boost the accuracy, we repeat all computations multiple times such that each program takes several minutes to run.

VI. Model Validation and Results

We validate the *PEACH* base performance and energy models on several benchmarks and system architectures. First, we focus on the performance and energy effects of the computation distribution by fixing the CPU and GPU speeds at their default values. MatrixMultiply serves as benchmark in this section. Unless stated otherwise, the matrix size is 12800×12800, which saturates the K20c GPU [8].

A. Parameter Measurement

In *PEACH*, the base performance models require the parameters R_C, R_G, and T_{OH}. The energy models require the additional parameters $P0_C$, $P0_G$, $P0_M$, Pd_C, Pd_G, and P_{OH}. In all experiments, we transfer data to/from the GPU once and execute MatrixMultiply on the CPUs and GPUs several times to increase the running time for improved measurement accuracy and to minimize the effects of the off-loading overhead T_{OH}. When running the CPUs and GPUs at fixed speeds, $P0_C$, $P0_G$, and $P0_M$ are system-dependent parameters, i.e., they are constant for a given system. R_C, R_G, Pd_C, Pd_G, and P_{OH} are dependent on the workload and the system. All of these parameters can be obtained or derived from the collected measurement data. For instance, Table V shows the measured model parameters for MatrixMultiply on our two hybrid systems.

There are several noteworthy observations to be made from these measurements. First, the base power consumption of the K20c and the C2075 is higher than the bare hardware installation power, which is 16 watts and 25 watts, respectively, according to the product specifications. This is because we measure the power of the GPU cards after they have been initialized and readied for execution. Second, the measured CPU and GPU compute rates are reasonably close to peak performance. Third, the K20c has a much higher energy efficiency than both the C2075 and the Xeon E5-2670 processors at the component level.

B. Model Accuracy

We apply the *PEACH* models to predict the performance and the energy efficiency with various computation distributions and compare them against actual measurements in Figure 3. In general, the model's predictions match the measurements very well for both performance and energy efficiency and on both systems, especially when the execution of the CPU portion takes longer than that of the GPU portion. The prediction error is less than 3% if the hosting power P_{OH} is taken into account. Ignoring the hosting power increases the error to 15% for energy efficiency.

Table VI: Predicted best-performance and best-energy distribution [CPU% : GPU%] for MatrixMultiply

	System I		System II	
	Pred.	Meas.	Pred.	Meas.
Best-Performance	22 : 78	22 : 78	49 : 51	49 : 51
Best-Energy	0 : 100	0 : 100	49 : 51	49 : 51

C. Identifying the Optimal Distributions

We use Eqs. 13 and 15 to predict the optimal computation distributions for maximizing the performance or energy efficiency, respectively. Table VI shows the predictions and the actual measurements for both systems. On System I, $\alpha_{perf} = 78\%$ and $\alpha_{eff} = 100\%$. On system II, $\alpha_{perf} = \alpha_{eff} = 51\%$. On both systems and for both objectives, *PEACH* predicts the correct optimal distributions.

VII. Model Application

In this section, we demonstrate the model usage for identifying the best coupled strategies when employing computation distribution and energy-saving scheduling. We first present the performance and energy effects of CPU and GPU DVFS technology and then show detailed results of model applications. All results are for System I.

A. Effects of DVFS Technology

As explained in Section III-D, the *PEACH* parameters are functions of the CPU speed f_C and the GPU speed f_G when DVFS is used. To obtain these functions, we run MatrixMultiply 100% on the CPU and then 100% on the GPU. Figure 4 presents the resulting model parameters for different CPU and GPU speeds. Several interesting observations can be made from these experiments.

- For MatrixMultiply, the CPU compute rate R_C increases with the CPU speed f_C close to linearly: $R_C = 111 \cdot f_C(GHz) + 6.1$ with a coefficient of determination of 99.9% by linear regression.
- When the system is idle, the CPU power consumption $P0_C$ and the remaining components' power $P0_M$ are constant relative to the CPU speed f_C. In addition, $P0_C < P0_M$. However, when the system executes MatrixMultiply, the CPU and the remaining host components draw the additional power Pd_C and Pd_M, respectively. The CPUs draw more active power than the other components, that is, $Pd_C > Pd_M$. Statistical linear regression analysis shows a strong linear relationship between the host's dynamic power and the CPU speed, i.e., $Pd_C + Pd_M = 124 \cdot f_C - 100.23$ with a coefficient of determination of 98%.

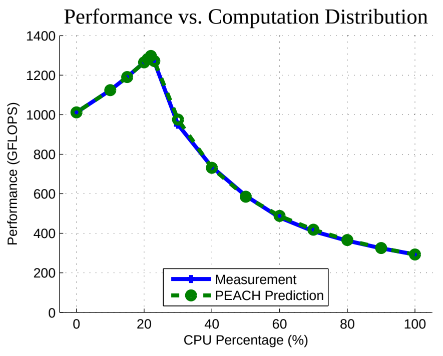
Table V: Measured model parameters

(a) System-dependent model parameters

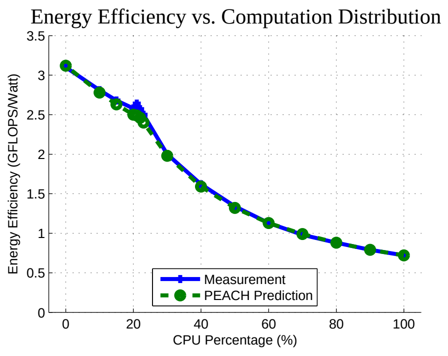
	Host		Accelerator
	$P0_C$(W)	$P0_M$(W)	$P0_G$(W)
Sys. I	42.4	76.7	46.6
Sys. II	42.4	76.7	82.3

(b) System- and application-dependent model parameters for MatrixMultiply

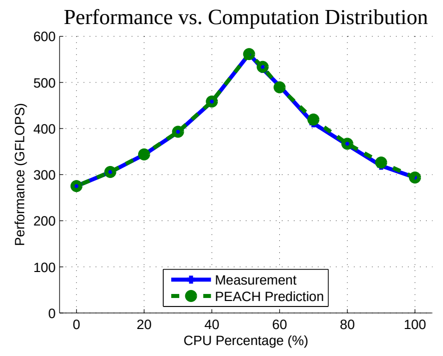
	Host		Accelerator		P_{OH}(W)
	R_C(GFLOPS)	Pd_C(W)	R_G(GFLOPS)	Pd_G(W)	
Sys. I	293	239.4	1052.4	128.6	30
Sys. II	293	239.4	302.53	117.8	28



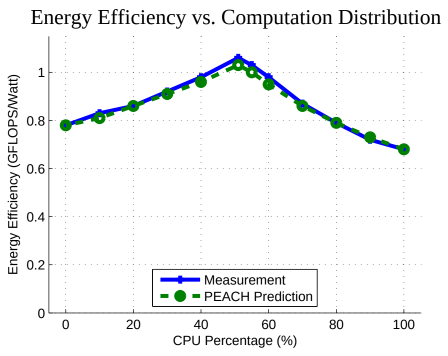
(a) System I: 16 CPUs + K20c GPU



(b) System I: 16 CPUs + K20c GPU



(c) System II: 16 CPUs + C2075 GPU



(d) System II: 16 CPUs + C2075 GPU

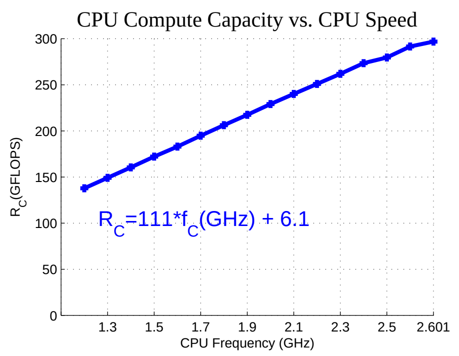
Figure 3: Model prediction vs. actual measurement of performance and energy efficiency on MatrixMultiply with various computation distributions

- The GPU compute rate R_G increases with the GPU speed f_G. Statistical regression analysis on four GPU speeds excluding the highest and lowest speeds shows $R_G = 1.495 \cdot f_G(MHz) - 1.78$ with a perfect correlation. The highest GPU speed of 758 MHz does not always follow this function because it often draws too much power, which causes the power management to automatically lower the clock speed. The exception at 324 MHz is expected as this is the only setting with a lower memory speed.
- The GPU base power $P0_G$ is constant relative to the GPU speed f_G (again with the exception of $f_G = 324\,MHz$). It is 47 watts once the GPU has been initialized. The GPU active power Pd_G linearly increases with f_G: $Pd_G = 0.337 \cdot f_G(MHz) - 109$ with a coefficient of determination of 99.7%.

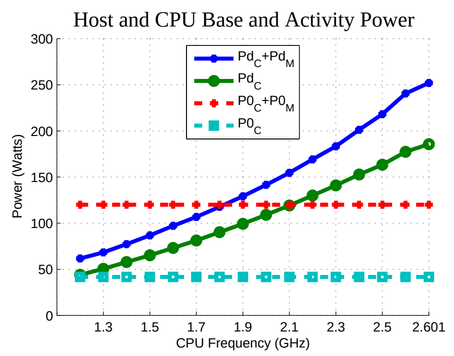
B. Coupling Distribution with Power-Aware Scheduling

We plug the profiled model parameters presented in section VII-A into Eqs. 8 and 12 to predict the performance and energy efficiency of MatrixMultiply under various coupled computation distributions and DVFS schedules. For simplicity, we use a constant $P_{OH} = 30$ in the model even though the overhead decreases with CPU speed from 30 watts at 2.6 GHz to 10 watts at 1.2 GHz on System I. Figure 5 shows the model predictions and measurements for three CPU speeds and $f_G = 705\,MHz$. Overall, the model predictions match almost perfectly with the measurements for distributions with a large percentage of CPU execution. The maximum discrepancy is within 3% and happens when the GPU portion takes longer to execute than the CPU portion. The model's predicted energy efficiency also matches the measurements well with a maximum error of 3%.

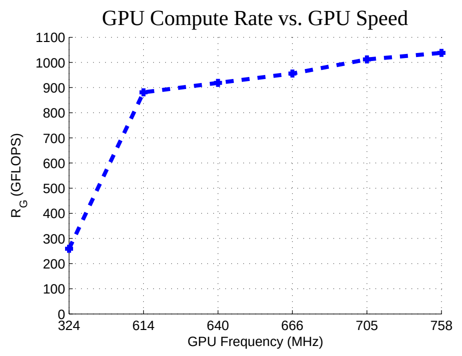
We now use the model to predict the best achievable performance and energy efficiency on System I. Table VII presents the results. For each combination of CPU and GPU speed, the first row lists α_{perf}%: (R_{max}, η), and the second row lists α_{eff}%: (R, η_{max}). The table excludes the highest and lowest GPU speeds because the former is unstable and



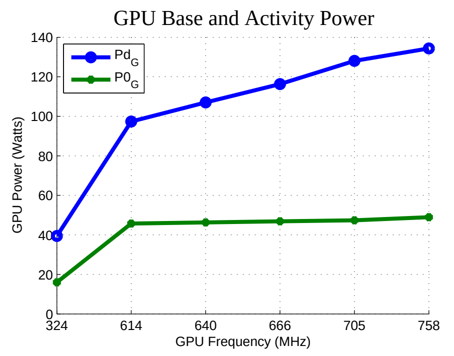
(a) R_C vs. f_C



(b) Host and CPU power vs. f_C

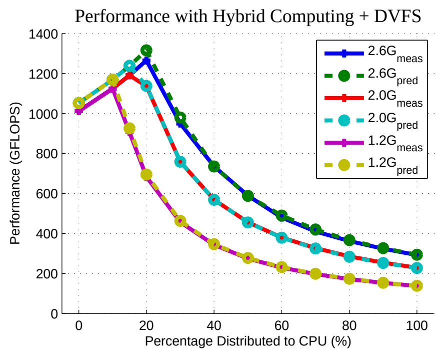


(c) R_G vs. f_G

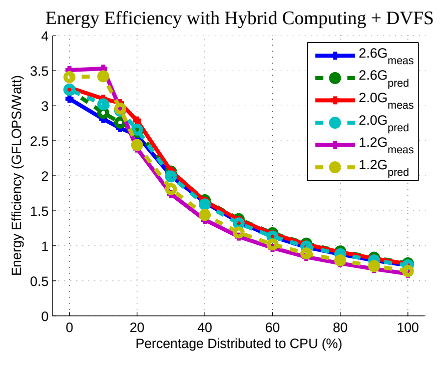


(d) GPU power vs. f_G

Figure 4: Measured model parameters and their variations with DVFS on MatrixMultiply



(a) 16 CPUs + K20c GPU



(b) 16 CPUs + K20c GPU

Figure 5: The measured and predicted effect of coupled adaptive mapping and DVFS scheduling on MatrixMultiply

the latter delivers much lower performance. There are several interesting observations to be made from these results.

First, for a given pair of CPU and GPU speeds, the best-performance distribution is distinct from the best-energy distribution except for low CPU speeds. Second, energy-efficient strategies offload a larger percentage of the computation to the GPU, leading to energy savings of up to 25%. Third, for a given CPU speed, running the GPU at a higher speed while offloading more computation results in both better performance and better energy efficiency. In other words, the GPU should always run at 705 MHz if there is offloaded computation. Fourth, both performance and energy efficiency vary with CPU speed when the GPU runs at 705 MHz. As the CPU speed decreases, a larger percentage of the computation must be offloaded to the GPU to reach the best achievable performance at that CPU speed. Note that the trend for energy efficiency is different. As the CPU speed decreases, the best achievable energy efficiency stays the same down to $f_C = 1.6$ GHz because 100% of the computation is offloaded. Once f_C drops to 1.4 GHz and lower, the energy efficiency improves when running some of the computation on the CPU.

Overall, the coupled strategy achieves the best performance of 1,336 GFLOPS with a configuration of [22% CPU: 78% GPU] at $f_C = 2.6\,GHz$ and $f_G = 705\,MHz$. The best energy efficiency of 3.42 GFLOPS/Watt is achieved with a configuration of [10% CPU: 90% GPU] at $f_C = 1.2\,GHz$

and $f_G = 705\,MHz$. The latter saves 32.6% energy and runs 12.5% slower than the former.

C. Model Prediction for Other Applications

Table VIII presents the model predictions and measurements for TSP, MatrixTranpose, and MatrixMultiply with a smaller matrix size of 3200×3200. Overall, the predictions match the measurements well for these applications.

We make the following observations. First, for compute intensive applications such as TSP and MatrixMultiply, when only using cooperative hybrid computing, the performance-oriented computation distribution incurs a higher increase in energy consumption than in performance compared to the energy-oriented computation distribution. Second, for compute intensive applications, coupling hybrid computing and DVFS scheduling delivers better energy efficiency and possibly better performance than hybrid computing without DVFS scheduling. Third, memory intensive applications such as MatrixTranpose differ significantly from compute intensive applications in two aspects. (1) The GPU execution rate R_G does not change with the GPU speed f_G. (2) The ratio of the GPU memory bandwidth to the CPU offloading bandwidth (PCI x16) is about 35:1, which is why the overhead time T_{OH} must be taken into account when designing hybrid computing algorithms. As a result, when only employing hybrid computing, both performance- and energy-oriented strategies choose not to offload work. Fourth, for memory intensive applications, a coupled strategy can achieve higher energy efficiency and performance than its counterpart that uses computation distribution only.

VIII. Related Work

Prior research in energy-efficient homogeneous multicore computing has primarily focused on reducing power and saving energy with little or no performance impact. One strategy adjusts software concurrency over the execution and thus controls the number of participating cores and the resulting power consumption [6, 14]. The other strategy leverages power-aware DVFS processors and scales down their performance states when demand is low [20, 22].

Heterogeneous computing has mainly focused on improving performance [5, 18]. By using CPUs and accelerators together, hybrid computing can deliver higher performance than CPU-only or GPU-only execution [23]. Nevertheless, hybrid computing algorithms and systems do not typically take energy efficiency into account. For example, Qilin [16] and Merge [15] solely consider performance when distributing computation among CPU cores and accelerators.

More recently, power- and energy-aware heterogeneous computing has begun to draw attention from researchers. Most of the existing work in this domain attempts to understand and model the power and energy consumption of heterogeneous computing systems; research in GPU power management is still sparse. Studies in [8, 11] experimentally investigate the power and energy behavior of applications on prototypes of GPU-accelerated systems. Other efforts analytically model GPU power with architecture-level instructions [9, 13, 19], hardware performance events [17, 21], or algorithm parameters [11].

Komoda et al [12] exploit DVFS and task mapping to power cap the GPU accelerated systems. *PEACH* is different from the empirical performance and power models in [12] in multiple aspects. First, *PEACH* is a general model for studying performance, energy, and their tradeoff and is suitable for studies with various objectives, while theirs is for studying the achievable performance for a given maximum power. Second, *PEACH* models the compute rate and energy consumption over execution times, while theirs is constrained by the power cap at time instances. Third, *PEACH* provides analytical formulae to directly calculate the resultant performance and energy for given DVFS settings and task mappings, while theirs indicates that time and power are functions of DVFS settings and task mapping.

IX. Conclusions and Future Work

Power-aware, cooperative hybrid computing is a new direction for energy efficient HPC. While heterogeneous systems provide a larger exploration space to increase performance, a performance gain may demand more energy consumption. We develop the PEACH model to quantify the relations between hardware and software parameters and performance/energy measures in cooperative hybrid computing. Experimental results demonstrate that the PEACH model is accurate and can guide the design of parallel algorithms and system schedulers.

In the future, we will extend this work to include MIC-based heterogeneous systems as well as more irregular applications. We are also planning to build systems and frameworks to automate the use of *PEACH* in performance- and energy-aware cooperative hybrid computing.

Acknowledgement

This work is supported in part by the NSF under Grants No. CNS-1305382, CNS-1305359, CNS-1212535, DUE-1141022, and CNS-1217231 as well as by donations from Nvidia Corporation.

References

[1] CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit.

[2] OpenBLAS. http://xianyi.github.io/OpenBLAS/.

[3] F. Blagojevic, X. Feng, K. W. Cameron, and D. S. Nikolopoulos. Modeling Multi-Grain Parallelism on Heterogeneous Multi-core Processors: A Case Study of the Cell BE. In *Proc. of the 2008 International Conference on High-Performance Embedded Architectures and Compilers*, 2008.

[4] M. Burtscher and H. Rabeti. A Scalable Heterogeneous Parallelization Framework for Iterative Local Searches. In *27th IEEE International Parallel & Distributed Processing Symposium (IPDPS2013)*, May 2013.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE International Symposium on Workload Characterization (IISWC2009)*, pages 44–54. IEEE, 2009.

Table VII: Predicted MatrixMultiply performance and energy efficiency on System I. Each cell has the format "$\alpha\%$: (R GFLOPS, EFF GFLOPS/Watt)". The upper row reflects the best performance and the lower row the best energy efficiency.

	705 MHz	666 MHz	640 MHz	614 MHz
2.6 GHz	**78: (1336, 2.58)**	78: (1275, 2.56)	76: (1225, 2.47)	76: (1206, 2.48)
	100: (1052, 3.23)	100: (994, 3.18)	100: (955, 3.15)	100: (916, 3.11)
2.4 GHz	80: (1316, 2.69)	80: (1243, 2.66)	78: (1225, 2.60)	78: (1175, 2.58)
	100: (1052, 3.23)	100: (994, 3.18)	100: (955, 3.15)	100: (916, 3.11)
2.2 GHz	82: (1284, 2.80)	80: (1243, 2.72)	80: (1194, 2.70)	80: (1145, 2.68)
	100: (1052, 3.23)	100: (994, 3.18)	100: (955, 3.15)	100: (916, 3.11)
2.0 GHz	82: (1264, 2.85)	82: (1212, 2.84)	82: (1165, 2.82)	80: (1138, 2.75)
	100: (1052, 3.23)	100: (994, 3.18)	100: (955, 3.15)	100: (916, 3.11)
1.8 GHz	84: (1253, 3.00)	84: (1184, 2.96)	82: (1141, 2.88)	82: (1118, 2.89)
	100: (1052, 3.23)	100: (994, 3.18)	100: (955, 3.15)	100: (916, 3.11)
1.6 GHz	86: (1224, 3.13)	86: (1156, 3.09)	84: (1137, 3.05)	84: (1091, 3.02)
	100: (1052, 3.23)	100: (994, 3.18)	100: (955, 3.15)	100: (916, 3.11)
1.4 GHz	88: (1196, 3.27)	86: (1151, 3.23)	86: (1111, 3.20)	86: (1066, 3.17)
	88: (1196, 3.27)	86: (1151, 3.23)	86: (1111, 3.20)	86: (1066, 3.17)
1.2 GHz	90: (1169, 3.42)	88: (1130, 3.41)	88: (1086, 3.37)	88: (1041, 3.33)
	90: (1169, 3.42)	88: (1130, 3.41)	88: (1086, 3.37)	88: (1041, 3.33)

Table VIII: Model predictions for TSP, MatrixTranspose, and MatrixMultiply

Strategy	Objective	Application	TSP	Transpose (10240x10240)	MatrixMultiply (3200x3200)
			(f_C, f_G, $\alpha\%$): (Climbs/s, Climbs/watt)	(f_C, f_G, $\alpha\%$): (GB/s, GB/s/watt)	(f_C, f_G, $\alpha\%$): (GFLOPS, GFLOPS/watt)
Distribution	Performance	Meas.	(2.6, 705, 95): (50,041, 143)	(2.6, 705, 0): (13.07, 0.0441)	(2.6, 705, 79): (1238, 3.56)
		Pred.	(2.6, 705, 95): (50,137, 140)	(2.6, 705, 0): (13.09, 0.0432)	(2.6, 705, 79): (1332, 3.57)
	Energy	Meas.	(2.6, 705, 100): (47,562, 170)	(2.6, 705, 0): (13.07, 0.0441)	(2.6, 705, 100): (1052, 2.29)
		Pred.	(2.6, 705, 100): (47,298, 164)	(2.6, 705, 0): (13.09, 0.0432)	(2.6, 705, 100): (1052, 2.29)
Distribution + DVFS	Performance	Meas.	(2.6, 705, 95): (50,041, 143)	(2.5, 705, 10): (14.52, 0.0508)	(2.6, 705, 79): (1238, 3.56)
		Pred.	(2.6, 705, 95): (50,137, 140)	(2.5, 705, 10): (14.62, 0.0496)	(2.6, 705, 79): (1332, 3.57)
	Energy	Meas.	(1.2, 705, 100): (47,339, 184)	(2.4, 324, 10): (13.44, 0.0522)	(1.2, 705, 89): (1183, 3.85)
		Pred.	(1.2, 705, 100): (47,695, 176)	(2.5, 705, 10): (14.52, 0.0518)	(1.2, 705, 89): (1042, 3.83)

[6] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 157–166, New York, NY, USA, 2006. ACM.

[7] R. Ge, X. Feng, T. Wirtz, Z. Zong, and Z. Chen. eTune: A Power Analysis Framework for Data-Intensive Computing. In *The Workshop on Power-Aware Systems and Architectures in conjunction with International Conference on Parallel Processing*, 2012.

[8] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. In *42nd International Conference on Parallel processing Workshops (ICPPW)*, 2013.

[9] S. Hong and H. Kim. An integrated gpu power and performance model. *ACM SIGARCH Computer Architecture News*, 38(3):280–289, 2010.

[10] Intel. Volume 3B: System Programming Guide, Part 2. *Intel 64 and IA-32 Architectures Software Developers Manual*, June 2013.

[11] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson. Power Aware Computing on GPUs. In *Symposium on Application Accelerators in High Performance Computing*, pages 64–73. IEEE, 2012.

[12] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura. Power Capping of CPU-GPU Heterogeneous Systems through Coordinating DVFS and Task Mapping. In *31st International Conference on Computer Design*, pages 349–356, Oct 2013.

[13] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. GPUWattch: Enabling Energy Optimizations in GPGPUs. In *ISCA*, volume 40, 2013.

[14] D. Li, B. de Supinski, M. Schulz, D. Nikolopoulos, and K. Cameron. Strategies for Energy Efficient Resource Management of Hybrid Programming Models. *IEEE Transaction on Parallel and Distributed Systems*, 24(1), January 2013.

[15] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng. Merge: a Programming Model for Heterogeneous Multi-Core Systems. *SIGPLAN Not.*, 43(3):287–296, March 2008.

[16] C.-K. Luk, S. Hong, and H. Kim. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, pages 45–55, 2009.

[17] X. Ma, M. Dong, L. Zhong, and Z. Deng. Statistical Power Consumption Analysis and Modeling for GPU-Based Computing. In *Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems (HotPower)*, 2009.

[18] J. Nickolls and W. Dally. The GPU Computing Era. *Micro, IEEE*, 30(2):56–69, 2010.

[19] J. Pool, A. Lastra, and M. Singh. An Energy Model for Graphics Processing Units. In *IEEE International Conference on Computer Design (ICCD)*, pages 409–416. IEEE, 2010.

[20] B. Rountree, D. Lownenthal, B. de Supinski, M. Schulz, V. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 460–469. ACM, 2009.

[21] S. Song, C. Su, B. Rountree, and K. W. Cameron. A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures. In *27th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2013.

[22] Q. Wu, V. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D. W. Clark. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *the 38th IEEE/ACM International Symposium on Microarchitecture*, pages 271–282, Barcelona, Spain, 2005.

[23] C. Yang, F. Wang, Y. Du, J. Chen, J. Liu, H. Yi, and K. Lu. Adaptive optimization for petascale heterogeneous cpu/gpu computing. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 19–28, 2010.