Energy Efficient Real-time Task Scheduling on CPU-GPU Hybrid Clusters

Xinxin Mei*, Xiaowen Chu*†, Hai Liu‡, Yiu-Wing Leung*, Zongpeng Li§¶
*Department of Computer Science, Hong Kong Baptist University, Hong Kong
†HKBU Institute of Research and Continuing Education, Shenzhen, China
‡Department of Computing, Hang Seng Management College, Hong Kong
§Department of Computer Science, University of Calgary, Canada
¶State Key Laboratory of Software Engineering, Wuhan University, China
Email: {xxmei, chxw, ywleung}@comp.hkbu.edu.hk, hliu@hsmc.edu.hk, zongpeng@ucalgary.ca

***Abstract*—Conserving the energy consumption of large data centers is of critical significance, where a few percent in consumption reduction translates into millions-dollar savings. This work studies energy conservation on emerging CPU-GPU hybrid clusters through dynamic voltage and frequency scaling (DVFS). We aim at minimizing the total energy consumption of processing a sequence of real-time tasks under deadline constraints. We compute the appropriate voltage/frequency setting for each task through mathematical optimization, and assign multiple tasks to the cluster with heuristic scheduling algorithms. In performance evaluation driven by real-world power measurement traces, our scheduling algorithm shows comparable energy savings to the theoretical upper bound. With a GPU scaling interval where analytically at most 38% of energy can be saved, we record 30-36% of energy savings. Our results are applicable to energy management on modern heterogeneous clusters. In particular, our model stresses the nonlinear relationship between task execution time and processor speed for GPU-accelerated applications, for more accurately capturing real-world GPU energy consumption.**

I. Introduction

Energy consumption is now a major subject of study in ICT [1][2][3]. The cost to power modern data centers during their lifetime surpasses that to manufacture them. E.g., the GPU-accelerated DeepMind computing center, best known for defeating professional human players in the strategy board game of Go, was acquired for about 600 million dollars, while its annual electricity bill is pegged at 150 million [3][4]. Even saving a few percentage of the energy consumed in such computing centers can bring tremendous financial gains.

Graphical processing units (GPUs) have become prevalent accelerators in modern data centers and supercomputers, due to their high computational power. Typically, a CPU can offload its computing tasks to an associated GPU to shorten the running time. In the TOP-500 supercomputer list [5] as of June, 2016, 94 are equipped with accelerators, and 69 out of them are equipped with GPUs. GPUs are also more energy efficient than traditional multi-core CPUs [6]. In the Green-500 List that ranks supercomputers by performance per Watt, 6 out of the top 10 most energy-efficient supercomputers are accelerated with GPUs [7].

Though the hybrid CPU-GPU clusters can achieve higher energy efficiency, their energy consumption is still very high. E.g., a single DGX-1 server from Nvidia consumes up to 3,200 Watts, 75% of which come from its 8 GPUs. To power a large-scale cluster remains a major item of expense for data centers, and energy efficiency of GPU-accelerated clusters is an important direction of research due to the complicated relationship between the runtime performance and power consumption [8][9]. Despite the growing need of energy conservation on GPU-accelerated hybrid clusters, GPU energy management techniques only start to witness developments.

Two commonly used techniques for saving energy in data centers are dynamic voltage and frequency scaling (DVFS) and dynamic resource sleep (DRS). DVFS refers to the capability of adjusting the voltage and frequency of a processor dynamically, while DRS puts idle servers into deep sleep states (or simply turning them off) to conserve energy [2][10]. However, simply transplanting CPU DVFS strategies onto GPU platforms could be ineffective [11][12], mainly due to the following two reasons. First, most existing works on CPU DVFS only consider scaling the CPU voltage or frequency alone, while existing works have shown that the GPU core voltage, GPU core frequency, and GPU memory frequency are the major factors that affect the dynamic GPU power [13][14]. Second, the execution time on a CPU is typically inversely proportional to the processor frequency, which is not always true on a GPU [15]. Many GPU-accelerated applications are memory-bounded and their performance are not only related to GPU processor frequency, but also GPU memory frequency. Hence the tradeoff between the application execution time and its average power consumption on GPUS becomes more complicated.

This work studies energy conservation on CPU-GPU clusters by applying GPU-specific DVFS and DRS. Our major objective is to minimize the total energy consumption of executing a sequence of real-time tasks (such as deep learning training tasks, big data analytics, weather prediction, etc.), while guaranteeing the task deadlines. This requires both appropriate GPU voltage/frequency configuration and task scheduling. Such problem is of practical significance in the resource management of data centers [16]. To tackle the energy minimization problem, we need to accurately understand the GPU performance model and power model. Towards this end, we consider three scaling variables that have significant impact on task execution time and power consumption: GPU core voltage, GPU core frequency, and GPU memory frequency. We first introduce our GPU power model and performance model, which capture the nonlinear relationship between task execution time and the GPU core/memory frequency. We then apply optimization techniques to compute the optimal voltage and frequency setting in terms of minimizing the energy consumption for a single task with and without deadline constraints. Finally, we design an effective heuristic scheduling algorithm to pack multiple tasks to the cluster following the principle of DRS. One challenge in this scheduling problem is to achieve a good balance between dynamic energy consumption (which prefers low voltage/frequency and long execution time) and static energy consumption (which prefers high voltage/frequency and low execution time). Since the optimal DVFS settings obtained in our first step do not consider static energy, we introduce another variable named *readjustment factor* to allow a non-optimal voltage/frequency setting for better task packing and hence less static energy consumption. Our major contributions in this work can be summarized as follows.

1) To the best of our knowledge, this work presents the first analytical GPU-specific DVFS model, and the optimization solution for a single task's DVFS;
2) We design a novel scheduling algorithm with the following features: (i) it exploits GPU DVFS to conserve energy consumption without violating task deadlines; (ii) it effectively packs a set of tasks on a number of servers to reduce static energy consumption; (iii) it intelligently adjusts the DVFS setting for better energy savings.
3) We conduct real GPU experiments on a set of benchmark applications to understand how much energy can be saved by GPU DVFS. We then design simulations based on our experimental data to evaluate the effectiveness of our scheduling algorithm. Our simulation results show that as much as 36% of the energy consumption can be saved.

The rest of the paper is organized as follows. Section II overviews related work. Section III describes our GPU power and performance models, and formulates the energy optimization problem. Section IV presents our optimization techniques and scheduling algorithm. Section V presents simulation results. Finally, we conclude the paper in Section VI.

II. Related Work

A rich body of theoretical studies model processor power consumption with a single variable, the processor speed, which can be controlled by varying the processor's voltage and frequency. Yao *et al.* studied task scheduling on a single processor. They proved that for the offline problem, the optimal speed during task processing is a constant [17]. Aydin *et al.* and Albers *et al.* proved that the offline multiprocessor scheduling problem to minimize energy consumption while meeting the task deadline is NP-hard [18][19]. Aydin *et al.* also proved that when the workload is evenly distributed among the multiple processors, energy can be minimized [18]. Hong *et al.* proved that for a multiprocessor system, there is no online optimal scheduler [20]. Irani *et al.* studied speed scaling along with DRS analytically [21]. They found that the algorithms with DRS perform similarly to those without DRS.

Gharaibeh *et al.* verified that a CPU-GPU hybrid cluster can achieve better performance and energy efficiency than a typical CPU cluster, for extremely large real-world graphic applications [6]. Liu *et al.* integrated CPU DVFS and GPU task migration on a heterogeneous cluster [22]. In their model, a task can be divided into a CPU-subtask and a GPU-subtask, and the execution of the two subtasks is asynchronous. The CPU voltage is scaled for better CPU-GPU load-balance. Liu *et al.* studied power-efficient online scheduling algorithms on CPU-GPU heterogeneous clusters [23]. In their model, the task is allowed to execute on either one CPU processor or one GPU processor. They examined earliest-deadline-first (EDF) and first-fit (FF) heuristic scheduling algorithms. They conducted experiments on a CPU-GPU platform, but because of the difficulty to measure GPU power consumption of different voltage/frequency states, they calculated the data instead.

Our work differs from existing ones in both power and performance models. The above literature all assumed that the processor execution speed is linearly proportional to the processor voltage/frequency (despite the findings in [24]), and the energy consumption is monotonically increasing in the scaling interval. Following these assumptions, the appropriate voltage or frequency level is determined by the processor workload [25]. In contrast, our energy function can be non-monotonic in the voltage/frequency scaling interval, and the optimal voltage/frequency is more related to task properties than processor workload. Furthermore, most existing work ignore the impact of scaling the memory frequency on power consumption and job execution time. We explicitly consider GPU memory frequency as one major factor in reducing the overall energy consumption.

III. System modeling and Problem Formulation

A. System Modeling

1) GPU DVFS Power and Performance Modeling: We start by presenting the GPU power and performance models that consider DVFS. We model the GPU runtime power as a function of the core/memory voltage/frequency ($\mathscr{P}$). A typical GPU card includes a many-core GPU module and an associated GPU memory module, and the voltage and frequency of GPU cores and GPU memory are independent and can be controlled separately. Eq. (1) shows the runtime power function, where P^{G0} is the summation of the power consumption unrelated to the GPU voltage/frequency scaling; V^{Gc}, f^{Gc}, f^{Gm} denote the GPU core voltage, GPU core frequency, and GPU memory frequency respectively. γ and c^G are constant coefficients that depend on the hardware and the application characteristics, indicating the sensitivity to memory frequency scaling and the core voltage/frequency scaling respectively [26]. In this work, the parameters in

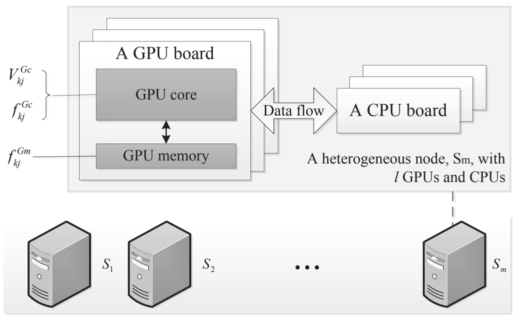

Fig. 1. Our studied CPU-GPU cluster with m servers.

the power modeling of an application are derived from its measured average runtime power consumption.

$$\mathscr{P}(V^{Gc}, f^{Gc}, f^{Gm}) = P^{G0} + \gamma f^{Gm} + c^G (V^{Gc})^2 f^{Gc} \quad (1)$$

Performance modeling of GPU DVFS is rather complex [24][27]. E.g., some recent work relies on machine learning and advanced statistical models [28]. In this work, we seek a first-order mathematical model with concise form to simplify the subsequent analysis of task scheduling. We formulate the performance function ($\mathscr{T}$) of a GPU-accelerated application as shown in Eq. (2) [13], where D represents the component that is sensitive to GPU frequency scaling, and t^0 represents the other component in task execution time. δ is a constant factor that indicates the sensitivity of this application to GPU core frequency scaling. We can always adjust the value of D and δ to model the sensitivity to GPU memory frequency scaling as $1-\delta$.

$$\mathscr{T}(V^{Gc}, f^{Gm}, f^{Gm}) = D(\frac{\delta}{f^{Gc}} + \frac{1-\delta}{f^{Gm}}) + t^0 \quad (2)$$

It is notable that f^{Gc} and V^{Gc} are correlated. For a fixed V^{Gc}, the maximum allowed core frequency ($f^{Gc}_{\max}$) is determined by V^{Gc}. We use $f^{Gc}_{\max} = g_1(V^{Gc})$ to denote this relationship, which has been shown to be sublinear in [29].

With the above GPU DVFS power and performance model, the GPU energy (E^G) consumed to process one task is the product of the runtime power and the execution time.

2) CPU-GPU Cluster Modeling: Fig. 1 shows the model of CPU-GPU cluster considered in this work. In the cluster, there are m servers, each with multiple pairs of CPU-GPU. In this work, we assume that the cluster has only one type of CPU/GPU, but different servers may have different number of pairs of CPU-GPU. We also assume that each task can be assigned to only one CPU-GPU pair, and one CPU-GPU pair can only execute one task at a moment.

A CPU-GPU pair can be in one of three states: runtime (or busy), idle, and off. A runtime CPU-GPU pair consumes both dynamic and static power; an idle CPU-GPU pair consumes only relatively low static power; and a turned-off CPU-GPU pair consumes no power. A CPU-GPU pair can be turned off by shutting down the server, which can only happen if there is no job assigned to any of its CPU-GPU pair. However, there is considerable energy cost from the turning on/off operations. We use Δ to denote the average energy overhead to turn on/off

TABLE I
SUMMARY OF IMPORTANT NOTATIONS

Symbol	Description
$\mathbf{J}$	The whole task set.
J_i	The i-th task in the task set.
a_i	The arrival time of J_i. a_i is a unit number.
d_i	The required deadline of J_i. We assume $d \geq t^\star + a_i$.
γ_i, P_i^0	Parameters related to the runtime power of J_i.
D_i, t_i^0, δ_i	Parameters related to the performance of J_i.
$P_i^\star$	The default runtime power of J_i on one CPU-GPU pair.
$t_i^\star$	The default execution time of J_i on one CPU-GPU pair.
S_j	The j-th server.
l_j	The number of CPU-GPU pairs on S_j.
Δ	The energy overhead of turning on a CPU-GPU pair.
ρ	The threshold for turning off a server.
κ_i	The time when the cluster begins executing J_i.
μ_i	The time when the cluster finishes executing J_i.
T	The time slot. T is a unit number.
$\mathbf{J}(T)$	The task set arrives at T, which has $n(T)$ tasks.
$n(T)$	The number of tasks in $\mathbf{J}(T)$.
$M(T)$	The number of occupied servers at T. $M(T) < m$.
N^{OFF}	The number of offline tasks. $N^{OFF} = n(0)$.
N^{ON}	The number of online tasks. $N^{ON} = \sum_{T \neq 0} n(T)$.

a single CPU-GPU pair. If any CPU-GPU pair is busy, the other CPU-GPU pairs on the same server without workload have to remain in the idle state.

Since the power consumption of a single CPU core is much less than that of a GPU, it is simplified as a constant in our model, i.e., we include the average CPU runtime power into P^{G0} in Eq. (1) for each GPU-CPU pair. Naturally the CPU will be kept active if the associated GPU is active, which means that the GPU and CPU share the same execution time to process a task. Under these conditions, the runtime energy consumption (E_J) of a CPU-GPU pair to process one single task can be reformulated as Eq. (3).

$$\begin{aligned} E_J = (P^{G0} + \gamma f^{Gm} + c^G (V^{Gc})^2 f^{Gc}) \times \\ (D(\frac{\delta}{f^{Gc}} + \frac{1-\delta}{f^{Gm}}) + t^0) \end{aligned} \quad (3)$$

B. Problem Formulation

We formulate our online task scheduling problem considering GPU DVFS as follows. For ease of reference, the major mathematical notations are summarized in Table I.

Our CPU-GPU energy optimization problem arises from the following system setting:

1) A CPU-GPU hybrid cluster that consists of m servers, $\mathbf{S} = \{S_1, S_2, ..., S_m\}$, and the j-th server S_j, has l_j CPU-GPU pairs;
2) A task set of n independent and non-preemptive tasks $\mathbf{J} = \{J_1, J_2, ..., J_n\}$ arriving over time, where the i-th task J_i is represented by a tuple $J_i : \{a_i, d_i, \mathscr{P}_i, \mathscr{T}_i\}$, where a_i denotes the arrival time and d_i denotes the task deadline.

Our objective is to minimize the total energy, E^{total}, while satisfying the task deadline constraints:

$$\begin{aligned} \min. \quad & E^{total} \\ \text{s.t.} \quad & \mu_i \leq d_i, \ \forall i. \end{aligned} \quad (4)$$

where μ_i denotes the time that job J_i finishes.

For every J_i in the task set, we need to compute its GPU voltage/frequency configuration as $\{V_i^{Gc}, f_i^{Gc}, f_i^{Gm}\}$, and its mapping $\{\kappa_i, \phi(J_i)\}$ where κ_i denotes the time the cluster begins to execute J_i according to the scheduling algorithm, and $\phi(J_i)$ denotes the assignment of J_i. $\phi(J_i) = S_{kj}^{J_i} = 1$ indicates that J_i is mapped onto the k-th CPU-GPU pair on the j-th server.

The total energy consumption, E^{total}, can be decomposed into three parts: E^{run}, E^{idle} and $E^{overhead}$, as Eq. (5) shows. E^{run} denotes the energy consumption to process all the tasks, i.e., $E^{run} = \sum_{i=1}^{n} E_{J_i} = \sum_{i=1}^{n} P_{J_i}(\mu_i - \kappa_i)$. E^{idle} denotes the idle system energy. It equals the summation of the idle energy of all the CPU-GPU pairs, $E^{idle} = P^{idle}\sum_{j=1}^{m}\sum_{k=1}^{l_j}\eta_{kj}$, where η_{kj} is the total idle period of the k-th CPU-GPU pair on the j-th server. $E^{overhead}$ denotes the overhead to turn on/off the servers. $E^{overhead} = \omega\Delta$, where ω is the total number of the turn-on behaviours in the cluster (counted based on the unit of a CPU-GPU pair). E^{run} is closely related to the GPU voltage/frequency setting while the others are more relevant to the scheduling algorithm.

$$\begin{aligned} E^{total} &= E^{run} + E^{idle} + E^{overhead} \\ &= \sum_{i=1}^{n} P_{J_i}(\mu_i - \kappa_i) + P^{idle}\sum_{j=1}^{m}\sum_{k=1}^{l_j}\eta_{kj} + \omega\Delta \end{aligned} \tag{5}$$

Empirically, when a dynamic turning off mechanism is involved, E^{run} should be the majority of E^{total}. We elaborate in the next section that for a single task, there exists an optimal solution in the DVFS scaling interval to minimize E^{run}.

IV. DVFS Energy Minimization for CPU-GPU Hybrid Clusters

A. Solution for a Single Task

As a first step, we consider the following sub-problem: *for a single task, given its power and performance models, what is the optimal voltage/frequency setting that minimizes the runtime energy regardless of its deadline?*

Eq. (6) shows the mathematical formulation of the problem. Notice that V^{Gc} and f^{Gc} are correlated variables, and f^{Gc} is upper bounded by a function of V^{Gc}, denoted by $g_1(V^{Gc})$.

$$\begin{aligned} \arg\min E_J &= \arg\min\{(P^{G0} + \gamma f^{Gm} + c^G(V^{Gc})^2 f^{Gc}) \\ &\quad \times (D(\frac{\delta}{f^{Gc}} + \frac{1-\delta}{f^{Gm}}) + t^0)\} \\ s.t.\ \ & V_{\min}^{Gc} \le V^{Gc} \le V_{\max}^{Gc},\quad f_{\min}^{Gm} \le f^{Gm} \le f_{\max}^{Gm}, \\ & f_{\min}^{Gc} \le f^{Gc} \le g_1(V^{Gc}) \end{aligned} \tag{6}$$

As the memory frequency f^{Gm} is independent of V^{Gc} and f^{Gc}, we can analyze core scaling and memory frequency scaling separately. We first consider GPU core voltage and frequency scaling. Given a fixed memory frequency f_o^{Gm}, the solution to Eq. (6) satisfies Theorem 1.

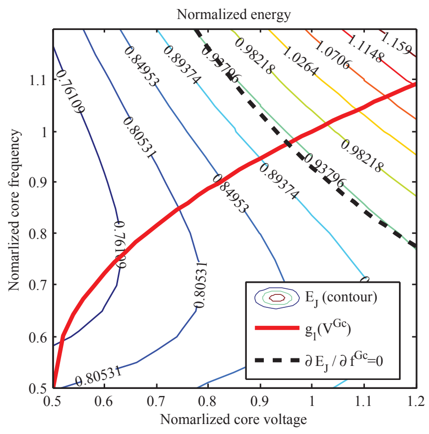

Fig. 2. When memory frequency is fixed, the minimum energy depends on the core voltage only. The data are obtained with $P = 100 + 50f^{Gm} + 150{V^{Gc}}^2 f^{Gc}; t = 25(0.5/f^{Gc} + 0.5/f^{Gm}) + 5; g_1(V^{Gc}) = \sqrt{(V^{Gc} - 0.5)/2} + 0.5$ and $f_o^{Gm} = f_{\max}^{Gm} = 1.2$. Note that although we use a specific function for demonstration, the finding holds for other general functions of our GPU DVFS modeling scheme.

Theorem 1. With a fixed memory frequency, the runtime energy of a task is minimum when the GPU core frequency is maximum corresponding to the GPU core voltage, i.e.,

$$E_{J\min}(f_o^{Gm}) = \arg\min_{V^{Gc}} E_J(V^{Gc}, g_1(V^{Gc}), f_o^{Gm}).$$

Theorem 1 transforms a two-variable optimization problem into a single-variable optimization problem. It implies that when we scale the GPU core alone to conserve energy, we only need to find an appropriate core voltage and set the core frequency to the largest allowed value.

Proof. We obtain the first-order partial derivatives as: $\frac{\partial E_J}{\partial V^{Gc}} = 2V^{Gc}c^G f^{Gc}(t^0 + D\delta + D(1-\delta)/f_o^{Gm})$ and $\frac{\partial E_J}{\partial f^{Gc}} = c^G {V^{Gc}}^2(t_0 + D(1-\delta)/f_o^{Gm}) - D\delta(P^0 + \gamma f_o^{Gm})/{f^{Gc}}^2$. Because $\frac{\partial E_J}{\partial V^{Gc}} > 0$, E_J cannot attain its minimum on the interior of the domain, and E_J is a monotonically increasing function of V^{Gc}. The minimum is on the boundary of $g_1(V^{Gc})$. f^{Gc} can be eliminated such that finding the minimum of E_J is only related to V^{Gc}.

We also give a graphical proof of Theorem 1. In Fig. 2, we plot the contour curves of E_J, $g_1(V^{Gc})$ and $\partial E_J/\partial f^{Gc} = 0$ together. As the figure shows, the optimal solution is along the red curve of $g_1(V^{Gc})$, where $g_1(V^{Gc})$ is tangent to the contour curve of $E_J = E_{J\min}$. $\square$

We then consider GPU memory frequency scaling alone. If the core voltage and frequency settings are fixed as V_o^{Gc} and f_o^{Gc}, we can easily compute the optimal memory frequency by setting $dE_J/df^{Gc} = 0$. We denote $f_\xi^{Gm} = \sqrt{(P^0 + c{V_o^{Gc}}^2 f_o^{Gc})D(1-\delta)/(\gamma(t^0 + D\delta/f_o^{Gc}))}$, that the optimal memory frequency equals: i)$f_{\min}^{Gm}$ ($f_\xi^{Gm} < f_{\min}^{Gm}$); ii)$f_\xi^{Gm}$ ($f_{\min}^{Gm} < f_\xi^{Gm} < f_{\max}^{Gm}$); iii)$f_{\max}^{Gm}$ ($f_\xi^{Gm} > f_{\max}^{Gm}$).

Based on the above analysis, the original three-variable problem is transformed into a two-variable optimization problem. Reducing the problem dimension is vital to speeding up the computation.

We now move on to the problem with task deadline considered: *what is the optimal voltage/frequency setting for a task without violating the given deadline?*

We denote the previous optimal solution of Eq. (6) as $\{f^{\hat{Gc}}, f^{\hat{Gm}}, V^{\hat{Gc}}\}$. We refer to the execution time obtained by substituting $\{f^{\hat{Gc}}, f^{\hat{Gm}}, V^{\hat{Gc}}\}$ into Eq. (2) as the optimal execution time ($\hat{t}$), and the one without GPU DVFS as the default execution time. The optimal execution time is possibly longer than the default execution time ($t^\star$).

Definition 1 (Task priority)**.** We define the task priority according to its optimal execution time. If $d - a < \hat{t}$, the task is deadline-prior; otherwise the task is energy-prior.

Apparently, if a task is deadline-prior, we cannot simply apply the original solution of Eq. (6) because scaling down the frequency too much may violate the deadline constraint. For a deadline-prior task, we may need to scale up the voltage/frequency compared to the original optimal setting.

The updated voltage/frequency setting for a deadline-prior task makes the updated execution time ($\hat{t}'$) equal to its allowed time period, i.e., $\hat{t}' = d - a$. We prove that the optimal solution of Eq. (6) is on the boundary of the domain. Intuitively, for a deadline-prior task, the additional constraint $f^{Gm} \geq \frac{D(1-\delta)}{d-t^0-\frac{D\delta}{f^{Gc}}}$ shrinks the domain. The updated solution $\{{f^{\hat{Gc}}}', {f^{\hat{Gm}}}', {V^{\hat{Gc}}}', \hat{t}'\}$ is defined by both $f^{Gc} = \frac{D\delta}{d-t^0-\frac{D(1-\delta)}{f^{Gm}}}$ and $f^{Gc} = g_1(V^{Gc})$. So for the deadline-prior task, $E_{J\min} = \arg\min_{f^{Gm}} E_J(V_o^{Gc}, f_o^{Gc}, f^{Gm})$, where $t(f_o^{Gc}, f^{Gm}) = d - a$ and $f_o^{Gc} = g_1(V_o^{Gc})$. This is a single-variable optimization problem and can be solved quickly.

B. Solution for Multiple Tasks

If we apply the derived optimal solution for each deadline-prior and energy-prior task in the task set, we obtain a fixed computed task length $\hat{t}/\hat{t}'$, and a minimized E^{run}. In this section, we consider the problem: *given the optimal computed task length of each task, how to schedule a number of tasks on the CPU-GPU cluster?* In the following, we propose our solution, the *EDL readjustment algorithm*, as shown in Algorithms 1 & 2.

As a typical strategy to handle online job scheduling, we divide time into equal time slots, and schedule newly arrived tasks in a time slot as a batch. The duration of a time slot should be significantly shorter than the average job execution time. The system is initiated with a set of offline tasks, which arrive at time slot $T = 0$; and the online tasks arrive at different time slots $T \neq 0$. The set of tasks arriving at time slot T is denoted by $\mathbf{J}(T)$. At the beginning of each time slot, we sort the newly arrived tasks in deadline-increasing order and assign them sequentially. This is referred to as earliest-deadline-first (EDF) scheduling, which is proved to be optimal in terms of feasibility [30].

The task mapping follows a simple principle that always tries to assign the task with the derived optimal task length to the CPU-GPU pair with the lightest workload. The objective is mainly minimizing E^{run}. We define another parameter, θ, to strike a better balance between the two conflicting objectives: minimizing E^{run} and minimizing E^{idle}.

Definition 2 (Task deferral threshold)**.** Given $\hat{t_i}$ as the optimal execution time with minimized runtime energy of J_i, instead of fixing the task execution time as $\hat{t_i}$, we allow it to vary in the interval of $[\theta\hat{t_i}, \hat{t_i}]$, $0 < \theta \leq 1$ by readjusting the frequency setting, in order to further reduce the total energy.

θ describes how much we can sacrifice the runtime energy for a shorter make-span and less occupied servers. It apllies proper voltage/frequency readjustments during the process of task scheduling. When a θ-readjustment is applied, we allow the non-optimal voltage/frequency setting for the energy-prior task in order to make usage of the currently alive servers with idle CPU-GPU pair(s). This behaviour transfers a number of energy-prior tasks into deadline-prior tasks. By default, $\theta = 1$ and no readjustment is allowed. By varying the value of θ, we actually control the maximum allowed portion of such transformation we can make in a task set. Because θ is designed to further reduce the idle energy, intuitively $\theta < 1$ is effective only when $l > 1$ and the idle energy is non-negligible.

Algorithm 1 Online EDL scheduling framework

Input: $\mathbf{J}$, $\mathbf{S}$, θ.
Output: $M(t)$, the corresponding runtime power state of the $M(t)$ occupied servers, $\{f_i^{Gc}, V_i^{Gc}, f_i^{Gm}, \kappa_i, \mu_i\}$ and the mapping of J_i,$\forall i$.

```
1: Execute Algorithm 2 (the EDL $\theta$-readjustment algorithm) at $T = 0$;
2: for all $T > 0$ do
3:     Process the tasks leaving at current time slot;
4:     Turn off the idle servers when appropriate;
5:     if there are arriving tasks then
6:         Assign the tasks to the server according to Algorithm 2,
           and turn on the servers if needed;
7:     end if
8: end for
```

Algorithm 1 shows the framework of our online scheduling algorithm. At $T = 0$, we process the initial set of tasks. Line 1 would output $M(0)$ occupied servers, and the task mapping solution for all the initial tasks.

Our online scheduling has three major components: processing leaving tasks, turning off the servers, and assigning the newly arrived tasks. We describe these components below.

Processing leaving tasks. At each time slot, we identify the set of tasks with $\lceil \mu_i \rceil = T$. We set the corresponding CPU-GPU pairs to idle during the time period of (μ_i, T). If a CPU-GPU pair still has tasks to process, we assign the next task to it at time slot T.

Turning off the servers. After processing the departured tasks, we dynamically turn off the servers using the DRS technique. We do not turn off the server immediately when there is no task to execute on it on the next time slot. Instead, we turn it off after all the CPU-GPU pairs on this server have been idle for at least a period of ρ. This strategy avoids frequent turn-on energy overhead in the case of job arrivals in the near future, at the price of slightly increased idle energy consumption.

Algorithm 2 The EDL θ-readjustment upon task arrival

```
Input: $T$, $M'(T)$, $\mathbf{J}(T)$, $n(T)$, $\theta$, $l$.
Output: the voltage/frequency setting and the mapping of $\mathbf{J}(T)$, $M(T)$.
    // $M'(T)$: the number of occupied servers after turning off the servers in the previous part
1:  for all tasks in $\mathbf{J}(T)$ do
2:      Find the optimal voltage/frequency setting with out missing the deadline for each task;
3:      $\{J_1, ..., J_r, ..., J_{n(T)}\} \longleftarrow$ sort the tasks according to the computed optimal length in EDF order;
4:  end for
5:  for $r = 1$ to $n(T)$ do
6:      $\mu_{SPT} \longleftarrow \min\{\mu_1, ..., \mu_{M'(T)*l}\}$;
7:          // Find the CPU-GPU pair, $S_{SPT}$, with the shortest processing time
8:      if $d_r - \max(T, \mu_{SPT}) \geq \hat{t_r}$ then
9:          Assign $J_r$ to $S_{SPT}$;
10:     else
11:         $t_\theta \longleftarrow \max\{\theta\hat{t_r}, t_{r\min}\}$;
12:           // $t_{r\min}$: the minimum execution time of $J_r$
13:         if $d_r - \max(T, \mu_{SPT}) \geq t_\theta$ then
14:             $\{\hat{V_i^{Gc}}', \hat{f_i^{Gc}}', \hat{f_i^{Gm}}'\} \longleftarrow \hat{t_i}'$=$d_r - \max(T, \mu_{SPT})$;
                  // $\theta$-DVFS is allowed for $J_r$, reconfigure $J_r$
15:             Assign $J_r$ to $S_{SPT}$;
16:         else
17:             Assign $J_r$ to a new CPU-GPU pair;
18:             Set the other CPU-GPU pairs on this server to idle;
19:             $M'(T) \longleftarrow M'(T) + 1$;
20:         end if
21:     end if
22: end for
23: $M(T) \longleftarrow M'(T)$;
```

Assigning the newly arrived tasks. Algorithm 2 shows our assignment strategy for task set $\mathbf{J}(T)$. We divide the solution into two phases. In the first phase (lines 1-4), we compute the optimal voltage/frequency setting that minimizes the runtime energy for each task. With this setting, we obtain a fixed task length of each task. Then in the second phase (lines 5-23), we pack the tasks to servers according to the obtained task lengths and the task deadlines. We always try to assign a task to the CPU-GPU pair with the lightest workload (lines 6-9). Note that in line 6 we need to find the larger value of μ_{SPT} and T, in the case that the CPU-GPU pair has been idle. If the task cannot fit into the selected pair, we check if a voltage/frequency readjustment is possible by setting its task length equal to the remaining time before the deadline (line 14). In line 18, if the task cannot fit into any active CPU-GPU pairs even with the readjustment, we assign it to a new CPU-GPU pair. We turn on the server containing this CPU-GPU pair and set its other CPU-GPU pairs to idle state.

The complexity of this algorithm is $n(\log n + \Phi + m)$, where Φ denotes the complexity of solving the optimization problem in the previous section.

Algorithm 3 The bin-packing scheduling algorithm

```
Input: $\mathbf{J}$, $\mathbf{S}$.
Output: $M(t)$, the voltage/frequency setting and the mapping of $\mathbf{J}$.
1:  for all offline tasks do
2:      $\{J_1, ..., J_r, ..., J_{N_{OFF}}\} \longleftarrow$ sort the offline tasks in earliest-deadline-first order;
3:      for $r = 1$ to $N^{OFF}$ do
4:          Compute the optimal $\{\hat{f_r^{Gc}}, \hat{f_r^{Gm}}, \hat{V_r^{Gc}}, \hat{t_r}\}$ for $J_r$, and the optimal task utilization $\hat{u_r}$;
5:          Assign $J_r$ to the CPU-GPU pairs according to the worst-fit heuristic, where the utilization of a CPU-GPU pair is no larger than 1 [30];
6:      end for
7:  end for
8:  for all $T > 0$ do
9:      Processing the tasks leaving at current time slot;
10:     Turn off the idle servers when appropriate;
11:     if $\mathbf{J}(T) \neq \emptyset$ then
12:         Sort $\mathbf{J}(T)$ in EDF order;
13:         for $r = 1$ to $n(T)$ do
14:             Compute the optimal $\{\hat{f_r^{Gc}}, \hat{f_r^{Gm}}, \hat{V_r^{Gc}}, \hat{t_r}\}$ for $J_r$;
15:             Assign $J_r$ to the CPU-GPU pairs according to the first-fit heuristic, following the criteria in [23], and turn on the servers when needed;
16:         end for
17:     end if
18: end for
```

V. Performance Evaluation

In order to assess the effectiveness of GPU DVFS and our scheduling algorithm on energy conservation, we first conduct real experiments by a commercial power meter to measure the real power consumption and record the execution time for a set of benchmark applications under different DVFS settings on an Nvidia Fermi GPU [29][14]. We then conduct simulations based on the gathered data sets. We also compare our EDL algorithm to a classical bin-packing heuristic algorithm described in Algorithm 3, which is developed in [23]. We modify their algorithm to fit our system model.

A. Simulation Configuration

1) The GPU Scaling Interval: Literally the range of s-calable GPU voltage and frequency varies among different GPU products. Without lose of generality, we compute the normalized values of f^{Gc}, V^{Gc} and f^{Gm} based on the factory default values, instead of the absolute values.

In our experiments, we measure the GPU core scaling with 9 data samples. For each fixed V^{Gc}, we gradually scale up f^{Gc} until the GPU board becomes unstable to get the corresponding $f^{Gc}_{\max}$. We fit the $f^{Gc}_{\max} = g_1(V^{Gc})$ relationship according to the measurement data as $g_1(x) = \sqrt{(x - 0.5)/2} + 0.5$. On our real GPU platform, the scaling interval is: $V^{Gc} \in [0.85, 1.05]$, $f^{Gc} \in [0.85, g_1(V^{Gc})]$, $f^{Gc} \in [0.5, 1.1]$. However, in the simulation we define a wider analytically scaling interval to be: $f^{Gm} \in [0.5, 1.2]$, $V^{Gc} \in [0.5, 1.2]$, and $f^{Gc}(V^{Gc}) \in [0.5, g_1(V^{Gc})]$ where $f^{Gc}_{\max} \approx 1.09$. The GPU voltage/frequency in the interval is continuously adjustable. In this analytical interval, the power consumption $\mathscr{P}$ is strictly convex.

2) Cluster Configuration: On our real CPU-GPU platform, it has $P^{idle} = 85$ Watts, and $(V^{Gc}, f^{Gc}, f^{Gm}) = (1, 1, 1)$ indicating the corresponding GPU configuration of (1.05 V, 995 MHz, 2100 MHz). We use the data for each simulated CPU-GPU pair.

We choose $\rho = \lfloor \Delta / P^{idle} \rfloor$, which is derived from the case that the task arriving at the next time slot would occupy the same server and each server has a single CPU-GPU pair ($P^{idle}\rho \leq \Delta$). We set $\Delta = 200$ Watts and $P^{idle} = 85$ Watts to have $\rho = 2$. Note that there might be other substitutions for ρ which provides better energy conserving performance, but since this paper focuses on DVFS technique, we stay with a simple strategy in the setting of ρ.

In addition, we assume there are at maximum 2048 CPU-GPU pairs, and every 1/2/4/8/16 CPU-GPU pairs are grouped into a server, i.e., $\sum_{j=1}^{m} l_j = 2048, l_j = 1/2/4/8/16$.

3) Task Set Generator: We measure the average runtime power and the execution time with 9 V^{Gc}/f^{Gc} samples and 5 f^{Gm} samples, of 9 GPU benchmark applications. We fit the task power and performance parameters ($\mathscr{P}$ and $\mathscr{T}$) based on the measurement data. Our simulated task set is a mixture of these 9 benchmark tasks, that each time we randomly pick out one task from the task set. We generate the task utilization of a single task according to the uniform distribution in $(0, 1)$, thus the expected average task utilization, $\bar{u_i} = 0.5$. We use the default execution time and the task utilization to derive the task deadline, i.e., $d_i = a_i + t_i^{\star}/u_i$. We quantize the workload of the task set by the task set utilization ($U_{\mathbf{J}}$), which is defined as the summation of the task utilizations over the product of the number of processors and the expected average task utilization, i.e., $U_{\mathbf{J}} = \frac{\sum_{i=1}^{n} u_i}{\bar{u_i} \sum_{j=1}^{m} l_j}$. We assign the initial offline task set utilization and the online task set utilization as $U_{\mathbf{JOFF}}$ and $U_{\mathbf{JON}}$ separately, which contain N^{OFF} and N^{ON} tasks. In this work, $U_{\mathbf{JOFF}} = 0.4$ and $U_{\mathbf{JON}} = 1.6$.

We simulate the task arrival in one day and choose the basic time unit as one minute, i.e., $T \in [1, 1440]$. We generate the number of arriving tasks at each time slot, $n(T), T \in [1, 1440]$ according to the Poisson distribution and refine it until $\sum_{T=1}^{1440} n(T) = N^{ON}$. At each time slot, we pick the $(\sum_{o=1}^{T-1} n(T) + 1)$-th to $\sum_{o=1}^{T} n(T)$-th task from the online task set to construct the current arrival tasks, of which $a_i = T$.

For each l and θ setting, we generate 1000 groups of the above task sets. We compute the average E^{idle}, $E^{overhead}$ and E^{run} separately.

B. DVFS Effect on a Single Task

In this subsection, we present the experimental results and simulation results of GPU DVFS on a single task. According to our measurements, scaling down the core voltage and applying the corresponding maximum allowed core frequency can significantly reduce the energy consumption. The memory frequency scaling influences the energy consumption mostly on the execution time, and different applications have different optimal memory frequency settings. Fig. 3 shows the derived optimal voltage/frequency setting and the corresponding energy saving of our 9 benchmark applications. Legend 'Wide' stands for our simulated scaling interval, and 'Narrow' stands for the realistic scaling interval. For both intervals, the optimal core voltage/frequency is relatively low, close to the allowed lowest setting. The optimal memory frequency varies, depending on the application characteristics. The derived optimal voltage/frequency settings coincide with our measurements.

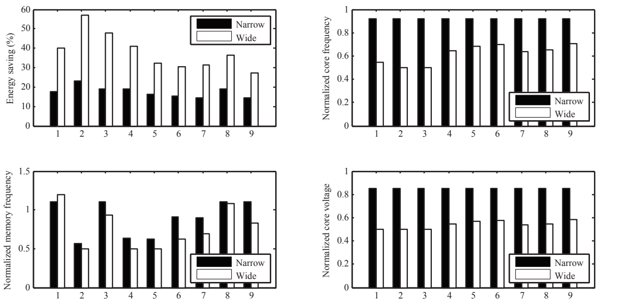

Fig. 3. The energy saving and the optimal voltage/frequency setting of the 9 benchmark applications. The x-axis stands for the task indices.

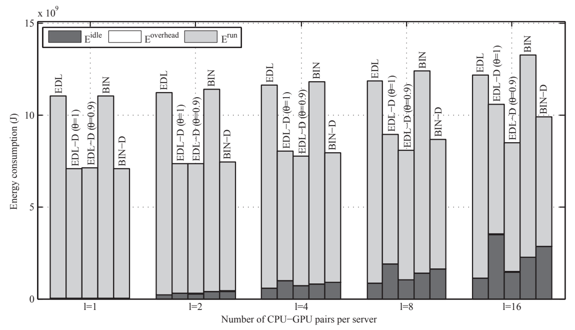

Fig. 4. Decomposition of the total energy consumption. In the figure, 'EDL' and 'BIN' denote our EDL readjustment algorithm and the bin-packing algorithm without GPU DVFS, while 'EDL-D' and 'BIN-D' denote the algorithms with GPU DVFS.

C. EDL Baseline Performance

As proven in [20], there is no optimal solution for our online task scheduling problem. In this work, we refer to the performance of the task scheduling algorithms without GPU DVFS as the baseline performance. In this subsection, we execute the EDL algorithm without runtime readjustment, i.e., $\theta = 1$.

We show the total energy decomposition in Fig.4, where the two highest bars denote the baseline energy consumption. The EDL algorithm leads to less energy consumption for all the l configurations. The runtime energy consumption is independent of l or the scheduling algorithm, with a constant value of 11.03 GJ. The overhead energy is marginal in the whole energy portfolio, varying from 8.32 to 8.78 MJ, and it slightly decreases as l increases. The idle energy consumption changes to the server configuration significantly, varying from 0.03 to 1.13 GJ.

The larger idle energy consumption is mainly caused by those idle CPU-GPU pairs that cannot be turned off even if no active task has been assigned to them. When $l = 1$, each CPU-GPU pair is idle for at most ρ after task processing, while when $l = 16$, the idle period of a CPU-GPU pair is overall much longer, that the CPU-GPU pairs on one server are idle for at

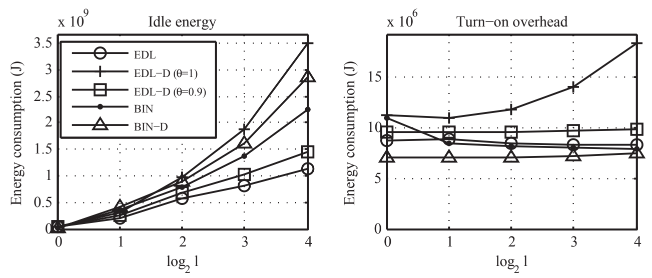

Fig. 5. Comparison between the energy consumption of the non-DVFS and DVFS scheduling algorithms.

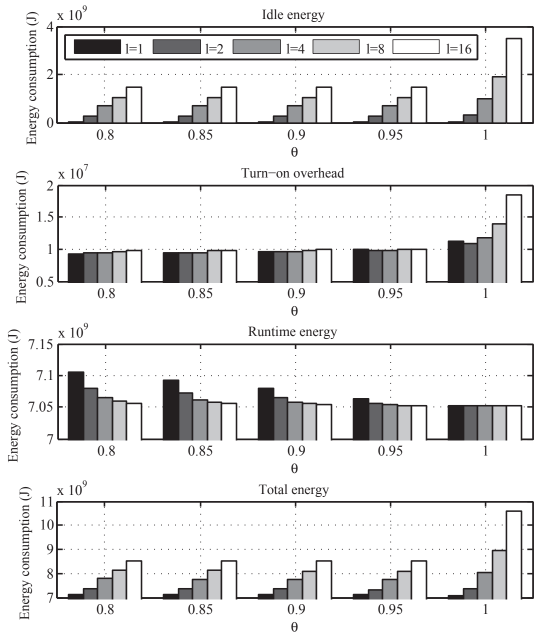

Fig. 6. The energy consumption with runtime readjustments.

least about ρ. Intuitively the more load balanced at runtime, the less idle system energy is. When we examine the runtime task mapping status, the bin-packing first-fit algorithm usually ends in a few CPU-GPU pairs of much heavier workload than the other CPU-GPU pairs. This would cause more idle system energy when l is large.

D. EDL DVFS Performance

We conduct DVFS experiments with $\theta = 1$ and $\theta = 0.9$ firstly, and then discuss the readjustment with other values of θ. For each group of experiments, we use the same offline and online task sets as those of the baseline simulation.

Fig. 4 shows the DVFS energy consumption with three lower bars. The runtime energy consumption of the DVFS algorithms is still a constant. It reduces from 11.03 GJ to 7.05 GJ; about 36.8% of runtime energy is saved with GPU DVFS. When $l = 1$, the three algorithms have similar energy consumption, about 7.08-7.12 GJ, where the bin-packing DVFS algorithm is slightly better. For other values of l, the EDL readjustment ($\theta = 0.9$) DVFS algorithm has the least energy consumption, followed by the bin-packing DVFS algorithm and the EDL DVFS ($\theta = 1$) algorithm. When $l = 16$, the total energy consumption of the three algorithms are 8.51, 9.92, 10.57 GJ respectively.

We further compare the idle energy and the turn-on overhead in Fig. 5. The DVFS algorithms lead to increases of idle system energy, especially for the EDL DVFS algorithm without runtime adjustment. If runtime θ-readjustment is applied, the idle energy is effectively controlled. When $l = 16$, the idle energy of the EDL non-DVFS, DVFS without readjustment and DVFS θ-readjustment algorithms are 1.13, 1.89, 1.45 GJ, respectively. For the bin-packing algorithm, the difference between the DVFS and non-DVFS in idle energy consumption is relatively small, as 0.22 GJ when $l = 16$. The turn-on overhead is still marginal in the whole energy portfolio. In general, the bin-packing algorithm is more effective in controlling the turn-on overhead, which means that the newly arriving tasks are more likely to be assigned to current busy servers, while it is the opposite for the EDL DVFS algorithm without readjustment.

To summarize, the EDL algorithm has better performance in the energy conservation in both the baseline and the DVFS simulation, but a runtime readjustment is needed when a server has many CPU-GPU pairs. A better balance between the runtime energy and the idle energy & turn-on overhead is more necessary when GPU DVFS is applied.

E. Effectiveness of the EDL Readjustment

In previous section, we have confirmed that the θ-readjustment is effective in controlling the idle energy when GPU DVFS is applied. We now discuss the impact of θ on the effectiveness of readjustment strategy.

We conduct the EDL DVFS θ-readjustment algorithm with five different values of θ. We plot the average idle energy, turn-on overhead, runtime energy and the total energy in Fig. 6. It is clear from our experimental results that smaller θ will result in slightly larger runtime energy consumption but less idle energy and turn-on energy. With $\theta \neq 1$, we consume less total energy, less idle energy and less turn-on overhead, especially for large l. For example, when $l = 16$, applying $\theta = 0.95$ reduces the total energy consumption from 10.58 GJ to 8.51 GJ, and reduces the idle energy from 3.5 GJ to 1.45 GJ. It is notable that when $\theta \neq 1$, the total energy, the idle energy, and the turn-on overhead do not vary much to θ for the same l. Much more energy is consumed when $\theta = 1$, therefore a runtime readjustment is quite necessary. For all the experiments, $\theta = 0.95$ ends in the minimum total energy consumption only except $l = 8$.

Fig. 7 shows the energy reduction compared to the baseline total energy consumption of the EDL algorithm of all the θ configurations. Theoretically the energy reduction is upper bounded by the average runtime energy reduction of the set of benchmark applications, i.e., 38% in our case. With appropriate θ, our online EDL algorithm can conserve 30-36% of energy, very close to the theoretical upper limit. But as l becomes larger, the energy reduction gradually decreases. Besides, the energy conservation of larger l depends more on

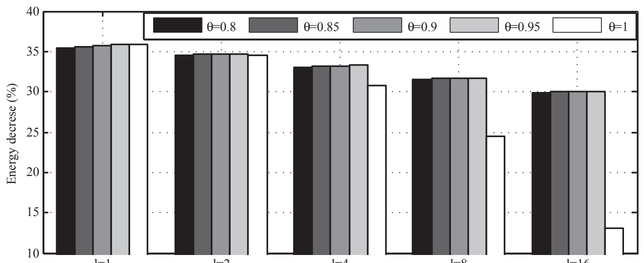

Fig. 7. The energy reduction compared to the baseline energy consumption.

the θ-readjustment. The selection of parameter θ depends on the ratio of the runtime energy over the idle energy.

VI. Conclusions and Future Work

In this paper, we study the energy conserving on CPU-GPU hybrid clusters. We propose the GPU-specific DVFS power and performance models, and derive the appropriate GPU voltage/frequency setting through mathematical optimization. We also design a heuristic scheduling algorithm to assign multiple tasks to the cluster, which uses the runtime DVFS readjustment to make a good balance between the dynamic energy consumption and static energy consumption. We find that for the online arriving tasks, the static energy is non-negligible, that a better balance of the dynamic energy and the static energy is quite necessary. Our algorithm has better energy saving than the traditional bin-packing solution.

In this work, we make a number of assumptions in the problem formulation to simplify the problem, such as homogeneity of CPUs and GPUs. We leave a more practical formulation and solution for our future work. It is also interesting to consider the case that a single task can occupy multiple GPUs.

Acknowledgement

This work is partially supported by HKBU Grant FRG2/14-15/059 and Shenzhen Basic Research Grant SCI-2015-SZTIC-002.

References

[1] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of ACM ISCA'07*, June 2007.

[2] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," in *Proceedings of ACM ASPLOS'09*, March 2009, pp. 205–216.

[3] J. Clark, "Google cuts its giant electricity bill with DeepMind-powered AI," [Online] http://www.bloomberg.com/news/articles/2016-07-19/google-cuts-its-giant-electricity-bill-with-deepmind-powered-ai.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, and H. Meuer, "TOP500," [Online] http://www.top500.org.

[6] A. Gharaibeh, E. Santos-Neto, L. B. Costa, and M. Ripeanu, "The energy case for graph processing on hybrid CPU and GPU systems," in *Proceedings of the 3rd Workshop on Irregular Applications: Architectures and Algorithms*, November 2013, pp. 2:1–2:8.

[7] W.-C. Feng and T. Scogland, "The Green500 list, June, 2016," [Online] http://www.green500.org/lists/green201606.

[8] X. Mei, K. Zhao, C. Liu, and X. Chu, "Benchmarking the memory hierarchy of modern GPUs," in *Proceedings of the IFIP 11th International Conference on Network and Parallel Computing*, 2014, pp. 144–156.

[9] X. Mei and X. Chu, "Dissecting GPU memory hierarchy through microbenchmarking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 72–86, Jan 2017.

[10] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proceedings of IEEE/ACM MACRO*, 2006, pp. 347–358.

[11] 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 *Proceedings of IEEE ICPP'13*, 2013, pp. 826–833.

[12] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, and S. Kato, "Power and performance analysis of GPU-accelerated systems," in *Proceedings of USENIX HotPower'12*, Berkeley, CA, USA, 2012.

[13] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, "Power and performance characterization and modeling of GPU-accelerated systems," in *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS)*, May 2014, pp. 113–122.

[14] X. Mei, Q. Wang, and X. Chu, "A survey and measurement study of GPU DVFS on energy conservation," *Digital Communications and Networks*, December 2016.

[15] D. H. Kim, C. Imes, and H. Hoffmann, "Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics," in *Proceedings of IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 2015, pp. 78–85.

[16] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[17] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, Oct 1995, pp. 374–382.

[18] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2003.

[19] S. Albers, F. Müller, and S. Schmelzer, "Speed scaling on parallel processors," in *Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 2007, pp. 289–298.

[20] K. S. Hong and J. Y.-T. Leung, "Online scheduling of real-time tasks," in *Proceedings of the 9th Real-Time Systems Symposium*. IEEE, 1988, pp. 244–250.

[21] S. Irani, S. Shukla, and R. Gupta, "Algorithms for power savings," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 4, 2007.

[22] W. Liu, Z. Du, Y. Xiao, D. A. Bader, and C. Xu, "A waterfall model to achieve energy efficient tasks mapping for large scale GPU clusters," in *Proceeding of the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, May 2011, pp. 82–92.

[23] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, and X. Lin, "Power-efficient time-sensitive mapping in heterogeneous systems," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2012, pp. 23–32.

[24] R. Nath and D. Tullsen, "The CRISP performance model for dynamic voltage and frequency scaling in a GPGPU," in *Proceedings of ACM MICRO'15*, December 2015, pp. 281–293.

[25] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, December 2001, pp. 95–105.

[26] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proceedings of ACM Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2010, pp. 280–289.

[27] Q. Wang and X. Chu, "GPGPU performance estimation with core and memory frequency scaling," *arXiv preprint arXiv:1701.05308*, 2017.

[28] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 564–576.

[29] X. Mei, L. S. Yung, K. Zhao, and X. Chu, "A measurement study of GPU DVFS on energy conservation," in *Proceedings of USENIX HotPower'13*, 2013.

[30] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.