Power-Efficient Time-Sensitive Mapping in Heterogeneous Systems*

Cong Liu
University of North Carolina at Chapel Hill
Dept. of Computer Science
cong@cs.unc.edu

Jian Li
IBM Austin Research Laboratory
jianli@us.ibm.com

Wei Huang
IBM Austin Research Laboratory
huangwe@us.ibm.com

Juan Rubio
IBM Austin Research Laboratory
rubioj@us.ibm.com

Evan Speight
IBM Austin Research Laboratory
speight@us.ibm.com

Felix Xiaozhu Lin
Rice University
Dept. of Computer Science
xzl@rice.edu

ABSTRACT

Heterogeneous systems that contain multiple types of resources, such as CPUs and GPUs, are becoming increasingly popular thanks to the potential of achieving high performance and energy efficiency. In such systems, the problem of data mapping and communication for time-sensitive applications while reducing power and energy consumption is more challenging, since applications may have varied data management and computing patterns on different types of resources. In this paper, we propose power-aware mapping techniques for CPU/GPU heterogeneous system that are able to meet applications' timing requirements while reducing power and energy consumption by applying DVFS on both CPUs and GPUs. We have implemented the proposed techniques in a real CPU/GPU heterogeneous system. Experimental results with several data analytics workloads show that compared to performance-driven mapping, our power-efficient mapping techniques can often achieve a reduction of more than 20% in power and energy consumption.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures; C.4 [**Performance of Systems**]: *design studies, modeling techniques*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*scheduling*

General Terms

Algorithms, Management, Measurement, Performance, Design, Theory, Experimentation

Keywords

Multi-Core Architectures, Heterogeneous CPU/GPU Multiprocessors, Power-Efficient Scheduling

*This work was performed while Cong Liu and Felix Xiaozhu Lin were interns at IBM Austin Research Laboratory.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PACT'12, September 19–23, 2012, Minneapolis, Minnesota, USA.
Copyright 2012 ACM 978-1-4503-1182-3/12/09 ...$15.00.

1. INTRODUCTION

Given the need to achieve higher performance without driving up power consumption, energy usage and heat dissipation, most chip manufacturers have shifted to multicore architectures. An important subcategory of such architectures are those that are heterogeneous in design. By integrating specific-purpose processing elements with general-purpose computing elements, such architectures can provide high performance and power efficiency [17, 21]. Heterogeneous architectures have been widely adopted in various computing domains, ranging from embedded systems to high performance computing systems.

One of the most prominent early examples of a heterogeneous multicore architecture is the IBM/Sony Cell® architecture, which consists of a Power processor and eight synergistic processors on the same chip. More recently, Graphic Processing Units (GPUs) have seen wide-spread use in heterogeneous systems. GPUs are able to execute data-parallel applications efficiently, due to their highly multi-threaded architecture and high-bandwidth memory. Major chip vendors have already released products that integrate CPUs and GPUs on the same chip, such as AMD's Fusion® APU chip.

By providing heterogeneous processing elements with different performance and energy characteristics in the same system, heterogeneous architectures are expected to provide more flexibility for better energy efficiency[21, 17]. Indeed, prior work [17] has shown that heterogeneous systems are able to reduce power and energy consumption compared to homogeneous systems. However, without power-aware application and data mapping methods that explicitly consider heterogeneity, such advantages cannot be fully exploited. Such methods are of interest for other reasons as well. For

example, excessive power consumption may cause system unreliability (due to power constraints and local hot spots).

In work on homogeneous systems, much work has been directed at techniques that reduce power and energy consumption by applying dynamic voltage/frequency scaling (DVFS) techniques to lower CPUs' voltages/frequencies. With the growing prevalence of such heterogeneous architectures, there is a need to evolve such techniques so they can be applied in systems with different computing resources. One important category of applications that could benefit from such techniques is those that are time-sensitive; examples include data analytics, stock trading, avionics, real-time scoring of bank transactions, live video processing, etc. Such applications range from desktop-level systems to systems that require significant computational capacity. As an example of the latter, systems processing time-sensitive business transactions have been implemented by Azul Systems on top of the highly-parallel Vega3 platform, which consists of CPUs and GPUs with up to 864 cores [22]. In this paper, such time-sensitive applications are our focus.

The driving problem that motivates our research is that of determining how to run analytics workloads[1] efficiently in terms of timing guarantees and power/energy efficiency on heterogeneous architectures. This can be very beneficial to systems design in the era of "Big Data" analytics [14]. "Big Data" analytics thrives to monetize insights in both structured and unstructured data generated from all sorts of sources [2], and make businesses more agile. Ongoing research [24] has shown that running such analytics workloads on GPUs often provides much better performance with respect to response times and power/energy efficiency. Analytics workloads often have strict response time requirements in order to make business decisions in realtime. As an example, IBM recently released a new multicore-based stream computing platform called IBM InfoSphere Streams System®, which enables massive amounts of data from "living entities" to be analyzed in real time, delivering fast, accurate insights to enable smarter business decision-making. Yet another example is IBM InfoSphere BigInsights® that brings the power of Hadoop to the enterprise for peta-scale Big Data analytics.

Motivated by the above observations, in this paper, we address the problem of devising power-efficient mappings of time-sensitive applications onto CPU/GPU heterogeneous systems. This problem can be divided into two subproblems. First, since applications may have different performance characteristics when executed on CPUs than GPUs, efficient mapping techniques are needed to provide performance guarantees[3] while considering heterogeneity. It has been shown that the general problem of mapping applications with deadlines onto heterogeneous systems consisting of at least two types of resources (e.g., CPUs and GPUs) is NP-hard in the strong sense [5]. Thus, it is important to design efficient mapping heuristics that can provide a measurable guarantee (such as a speed-up factor, as shown in Sec. 4) to meet applications' deadlines. Secondly, in order to reduce power and energy consumption, power-efficient techniques such as DVFS are needed that can be applied on both CPUs and GPUs. Our proposed strategy is to first map applications onto CPUs and GPUs, and to then apply DVFS on each CPU and GPU.

More specifically, our mapping techniques consider the characteristics of both resources and applications to make mapping decisions. An application is always mapped to a resource that can execute it with the fastest speed while meeting its deadline. We propose mapping techniques that can be applied both initially and when applications dynamically arrive at run-time. Additionally, by leveraging the fact that at run-time some applications typically exhibit actual execution times that are lower than their worst-case execution time, we propose and apply techniques to dynamically and aggressively reduce voltages/frequencies on CPUs and GPUs. Note that our strategy should generally apply to other heterogeneous systems containing two different types of computing resources, such as systems with CPUs and FPGAs, CPU with on-chip accelerators, etc. Our overall strategy is based on several components:

1. An offline scheme for mapping the initial set of applications and computing the appropriate voltage/frequency to minimize power and energy consumption while meeting all deadlines (Sec. 4).

2. An online scheme for dynamically mapping applications arriving at run-time and re-evaluating the voltage/frequency settings (Sec. 5).

3. A speculative voltage/frequency scaling scheme that aggressively reduces voltage/frequency by exploiting average-case execution time information (Sec. 6).

We evaluated the above strategy by implementing it on a real CPU/GPU heterogeneous system and running several analytics workloads, such as Non-Negative Matrix Factorization (NMF)[4] [18] implemented using OpenCL [8]. Experimental results using real power measurements demonstrate that our proposed algorithms can achieve much better power and energy efficiency than performance-driven mapping algorithms that always map applications to resources with the fastest response time. Our experiments show that power and energy consumption can be further reduced by exploiting earlier-than-worst-case completions and by aggressively reducing voltages/frequencies by exploiting average-case execution time information.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 presents our system model. Sections 4-6 describe the proposed power-efficient mapping techniques in detail. Section 7 discusses our implementation methodology and experimental results. Section 8 concludes.

2. RELATED WORK

Kumar et al. [17] were among the first work to demonstrate that heterogeneous architectures have advantages over

[1] The term "application" used in this paper refers to such analytics workloads. In the evaluation section (Sec. 7), we implement several such analytics workloads to evaluate the performance.

[2] Everyday, 2.5 quintillion bytes of data are created [14], coming from many sources: from sensors used to gather climate information, posts to social media sites, digital pictures and videos posted online, transaction records of online purchases, and cell phone GPS signals to name a few.

[3] In this paper, the required performance guarantee is meeting applications' deadlines.

[4] NMF has been used in many data analyzing problems such as text mining, spectral data analysis, and scalable Internet distance prediction [18].

homogeneous ones in terms of throughput and power efficiency. In work on throughput-oriented systems, Luk et al. [21] proposed an adaptive mapping technique to map applications onto a heterogeneous system consisting of CPUs and GPUs. This mapping algorithm has been implemented in an experimental system called Qilin for programming CPUs and GPUs. Evaluation results show that compared to manual mapping (e.g., manually done by the programmer), their adaptive mapping performs close in both execution time and energy consumption, but can better adapt to changes in input sizes, and hardware and software configurations. In [4], Augonnet et al. proposed an efficient mapping technique that overlaps communication with computation to support application graphs in heterogeneous systems consisting of CPUs and GPUs. Moreover, several [13, 15] propose scheduling techniques in heterogeneous systems in order to improve performance and/or energy efficiency, at either user-level and operating system kernel-level. Also much work (see [20] for a survey) has been done on DVFS-based power-aware scheduling techniques for homogeneous systems.

In real-time and embedded systems, much work has been done on power- and energy-efficient scheduling on DVFS-enabled platforms, including uniprocessor and multiprocessor systems. An overview of such work can be found in [6]. However, none of this prior work targets heterogeneous systems consisting of CPUs and GPUs. In contrast to our work, where power/energy measurements were obtained in a real heterogeneous system by running real workloads, in most prior work, a simulation-based evaluation approach was used.

3. SYSTEM MODEL

We consider the problem of mapping n independent applications $J = \{J_1, J_2, J_3, ..., J_n\}$ arriving at time 0, and a number of dynamically arriving applications, onto m processors $\{P_1, P_2, ..., P_m\}$. Each processor is either a CPU or a GPU. Each application J_i has an arrival time r_i, a deadline d_i, and a worst-case execution time of e_i^c (resp., e_i^g) if executed on a CPU (resp., GPU) under the maximum voltage/frequency. The execution time and deadline can range from milliseconds to hours, depending on the specific application. We can obtain the worst-case execution time of an application by either using a worst-case execution time analysis tool [12] or by sampling runs.[5] Any application J_i must complete its execution by time d_i. We require that for any application J_i, $e_i^c + r_i \leq d_i$ and $e_i^g + r_i \leq d_i$ hold; otherwise, deadlines cannot be met. Our techniques can also be extended to be applied to more generalized real-time execution models. A summary of important notations is given in Table 1.

Power model. Prior work has shown that the power/energy consumption of a processor is a monotonically increasing superlinear function of the supply voltage [23]. The dominant components of power consumption in widely adopted CMOS technology are active and idle power dissipation. Similar to previous work [7, 11, 16], we can approximate the power consumption p_{total} of an on-chip system by the following:

[5]Sampling techniques have been commonly used to obtain worst-case execution times in prior related work (see [12] for a survey).

Table 1: Notation Summary.

n	number of applications arriving at time 0
m	number of processors
J_i	i^{th} application
P_i	i^{th} processor (either a CPU or a GPU)
r_i	arrival time of J_i
d_i	deadline of J_i
e_i^c	worst-case execution time of J_i executed on a CPU under its maximum voltage
e_i^g	worst-case execution time of J_i executed on a GPU under its maximum voltage
v_k	k^{th} voltage level on a processor
v_{max_i}	maximum voltage level on P_i

$$p_{total} = p_{idle} + \sum_{i=1}^{m} \lambda_i \cdot f_i^3.$$

In this expression, p_{idle} denotes the idle power dissipation, which is assumed to include the base power consumption of the CPU and the power consumption of all other components. Additionally, each processor P_i also consumes power $\lambda_i \cdot f_i^3$ that depends on the processor's operating frequency f_i. (Note that frequency is proportional to voltage [23] with a small offset.) Thus, the active energy consumed by processor P_i during a time interval with length t is given by $\lambda_i \cdot f_i^3 \cdot t$, where λ_i is a coefficient term that includes power loss due to power supply inefficiency. Furthermore, the execution time of an application is inversely proportional to voltage and frequency [11, 23]. As seen from this model, DVFS is able to reduce power/energy consumption by reducing the voltage/frequency.

We assume that both CPUs and GPUs have a fixed number of voltage steps, which is often the case in practice. Moreover, most modern processors specify voltage/frequency pairs (referred to as a *P-state*) instead of separate voltage and frequency values. Thus, for conciseness, we use voltage in this paper to represent the voltage/frequency pair. Let $\{v_1, v_2, ..., 1.0\}$ ($v_j \leq v_{j+1} \leq 1.0$) denote the normalized voltage levels of processor P_i (the maximum normalized voltage level on any processor is 1.0). Each workload J_i has a worst-case execution time e_i^c on a CPU under the maximum voltage level, or e_i^g on a GPU under the maximum voltage level. Since the execution time of an application is inversely proportional to voltage, the execution time of J_i executed on any CPU (respectively, GPU) P_i under a specific voltage level v_k is thus given by $\frac{e_i^c}{v_k}$ (respectively, $\frac{e_i^g}{v_k}$).

Preemptive vs. non-preemptive execution. Preemptive execution is allowed on CPUs. However, on GPUs, executions are often non-preemptive [10]. That is, once an application starts execution on a GPU, it cannot be preempted by other applications until its completion. We thus assume that executions are preemptive on CPUs but non-preemptive on GPUs.

Earliest-deadline-first. As shown in Sec. 4, we use earliest-deadline-first (EDF) to order applications on each processor. Under EDF, applications are prioritized by their deadlines, and any ties are broken by application ID.

4. STATIC MAPPING

In this section, we present the offline mapping algorithm for the initial set of applications arriving at time 0, assuming that each application's worst-case execution time is known. In later sections, we show how to adapt to the dynamic case where workloads arrive at run-time.

The algorithm contains an assignment phase, a load-balancing phase, and a voltage scaling phase. The assignment phase assigns applications to processors (i.e., CPUs and GPUs), the load-balancing phase balances loads among processors, and the voltage scaling phase adjusts the voltage/frequency on each processor. The goal is to reduce power and energy consumption (i.e., adjust each processor's voltage/frequency to the lowest possible level) while meeting all deadlines. A *feasible* mapping is one that meets all deadlines.

Assignment. The problem of assigning applications to heterogeneous systems containing at least two types of processors while meeting all deadlines has been shown to be NP-hard in the strong sense [5]. An assignment algorithm for such systems named FF-3C , recently proposed by Andersson et. al [3], provides provable quantitive performance guarantee on meeting deadlines. FF-3C was designed for assigning real-time *periodic* [6]applications onto two different types of processors. As shown in [3], FF-3C has a speed-competitive ratio of 2, which implies that if any feasible mapping algorithm exists for a set of applications over A, then FF-3C can provide a feasible mapping for the same set of applications over a computing platform A' each of whose processors has at least twice the speed of the corresponding processor in A. A low speed-competitive ratio of 2 indicates high performance of the FF-3C algorithm.

Our proposed assignment phase is designed based upon FF-3C. Compared to the original FF-3C algorithm, the only modification is to order applications in a certain way before making the assignment decisions (as shown later). The intuition behind the ordering process is to increase the likelihood that applications that run faster on a specific type of processor are mapped onto processors of that type. The pseudo-code of the assignment algorithm is shown in Fig. 1. Before describing the algorithm, we give several necessary definitions.

Definition 1. *The load of an application J_i executed on CPU (resp., GPU) under the maximum voltage is given by* $load_i^c = \frac{e_i^c}{d_i - r_i}$ *(resp.,* $load_i^g = \frac{e_i^g}{d_i - r_i}$*). An application J_i is termed to be heavy on a CPU (resp., a GPU) if* $load_i^c > 1/2$ *(resp.,* $load_i^g > 1/2$*).*

Definition 2. *The heterogeneity ratio of an application J_i is defined to be* $H_i = max(\frac{e_i^c}{e_i^g}, \frac{e_i^g}{e_i^c})$*. For any application J_i,* $\frac{e_i^c}{e_i^g} > 1$ *implies that J_i is more suitable to be executed on a GPU than on a CPU. The favorite processor type of an application J_i is a GPU if* $\frac{e_i^c}{e_i^g} > 1$ *and a CPU otherwise.*

[6]A periodic application generates a potentially infinite sequence of jobs with deadlines. Jobs arrive exactly p time units apart where p is often called period. Note that the periodic task model generalizes the application model used in this paper. This is because each application under our model can be deemed as a periodic task that only releases one job.

ASSIGNMENT

```
1  Sort applications in J^h and J^nh by
   largest-heterogeneity-ratio-first
2  for each application J_k in J^h do
3    Assign J_k to its favorite processors by first-fit
4  for each application J_k in J^nh do
5    Assign J_k to its favorite processors by first-fit
6  for each remaining application J_k in J^nh do
7    Assign J_k to its non-favorite processors by first-fit
8  for each processor P_i do
9    Sort applications assigned to P_i by EDF
```

Figure 1: The psuedo-code of Algorithm *Assignment*.

Definition 3. *Let $J_1, J_2, ..., J_{q_i}$ denote applications that are assigned to processor P_i, ordered by EDF (i.e., $d_k \leq d_{k+1}$) that are not completed at time t. Then, the processor load on CPU (respectively, GPU) P_i at time t is given by* $load(P_i, t) = max_{k=1}^{q_i}\Big(\frac{\sum_{o=1}^{k} \overline{e}_o^c}{d_k - t}\Big)$ *(resp.,* $load(P_i) = max_{k=1}^{q_i}\Big(\frac{\sum_{o=1}^{k} \overline{e}_o^g}{d_k - t}\Big)$*), where $\overline{e}_o^c$ ($\overline{e}_o^g$) denotes the remaining execution time of J_o at time t on CPU (resp., GPU). At time 0,* $load(P_i, 0) = max_{k=1}^{q_i}\Big(\frac{\sum_{o=1}^{k} e_o^c}{d_k}\Big)$ *(resp.,* $load(P_i) = max_{k=1}^{q_i}\Big(\frac{\sum_{o=1}^{k} e_o^g}{d_k}\Big)$*) if P_i is a CPU (resp., GPU).*

Example Consider assigning two applications J_1 and J_2 onto a CPU P_1, where $(e_1^c, d_1 = 1, 4)$ and $(e_2^c, d_2 = 3, 12)$. Thus, we have $load(P_1, 0) = max\big(1/4, (1+3)/12\big) = 1/3$.

Now we describe the pseudo-code of the assignment algorithm. Note that the *first-fit* policy used in this algorithm represents the classical first-fit bin-packing policy, which assigns an application to the first processor that can accommodate it. In our problem context, a processor can accommodate an application if and only if the application's deadline can be met (which can be checked by Theorem 1 shown later).

Algorithm description. Algorithm *Assignment* assigns applications to processors. First, applications are divided into two sets: a set J^h in which applications are heavy on their non-favorite processors and a set J^{nh} in which applications are non-heavy on both CPUs and GPUs. Then, within each set, applications are sorted by largest-heterogeneity-ratio-first (Line 1). This is to give applications with a larger heterogeneity ratio a greater chance to be assigned to their favorite processors. Then applications in J^h and J^{nh} in order are assigned to their favorite type of processor by first-fit (Lines 2-5). Any remaining applications in J^{nh} are assigned to their non-favorite type of processor by first fit (Lines 6-7). Finally applications on each processor are ordered by EDF (Lines 8-9). Note that if an application cannot be assigned to a processor (while meeting its deadline) in Lines 3, 5, and 7, then the algorithm fails to produce a feasible assignment.

So far we have not yet discussed the criteria for successfully mapping an application to a processor to meet its deadline. To provide such a criteria, we apply a well-known result on scheduling independent applications with deadlines on a uniprocessor under EDF, as stated in the following theorem.

Theorem 1. *[19] For executing applications arriving at time 0 on CPU (or GPU) P_i in EDF order, all deadlines are guaranteed to be met if $load(P_i, 0) \leq 1$, where $load(P_i, 0)$ is the processor load at time 0 as defined in Def. 3.*

Time complexity. It can be shown that Algorithm *Assignment* has a time complexity of $O(n^2 \cdot m)$, where n is the number of applications and m is the number of processors.[7]

Performance guarantee provided by the assignment algorithm. It has been shown in [3] that FF-3C has a speed competitive ratio of 2. There are only two differences between our assignment algorithm and FF-3C. First, we sort applications within sets J^h and J^{nh} by largest-heterogeneity-first where FF-3C does not assume any specific ordering among tasks. Second, we target independent applications with deadlines where FF-3C targets periodic applications with deadlines. Due to the facts that the order of applications used in our algorithm can be viewed as a specific case of all possible orders supported by FF-3C and the periodic application model generalizes the independent application model, our mapping algorithm also yields a speed competitive ratio of 2. (Note that the load-balancing and the voltage/frequency scaling schemes as presented next do not invalidate this speed competitive ratio result since they are executed after the assignment process.)

Example Consider assigning a set of six applications onto a heterogeneous system with a CPU P_1 and a GPU P_2. The parameters of these applications are: $J_1(6, 2, 10)$, $J_2(2, 1, 5)$, $J_3(4, 3, 15)$, $J_4(6, 1, 8)$, $J_5(3, 4, 12)$, $J_6(1, 3, 4)$, where $J_i(a, b, c)$ denotes that $e_i^c = a$, $e_i^g = b$, and $d_i = c$. The heterogeneity ratios of these applications can be calculated by Def. 2 as follows: $H_1 = 3$, $H_2 = 2$, $H_3 = 4/3$, $H_4 = 6$, $H_5 = 4/3$, and $H_6 = 3$. By the definitions of J_h and J_{nh}, in this example, $J_h = \{J_1, J_4, J_6\}$ and $J_{nh} = \{J_2, J_3, J_5\}$. By Algorithm *Assignment* shown in Fig. 1, applications J_5 and J_6 are assigned to CPU P_1, and the other four applications are assigned to GPU P_2. All deadlines are guaranteed to be met since under this assignment, by Def. 3, $load(P_1) = 1/3 < 1$ and $load(P_2) = 7/15 < 1$. (This example system will be used throughput this paper.)

Load-balancing. After mapping applications onto processors, it may be the case that some processors have much heavier loads than others. This is undesirable from the perspective of reducing the overall system power and energy consumption because the whole system may need to run longer due to the imbalanced load. This may cause other system components such as disks and memory to consume more energy. Thus, we propose to balance loads among processors after the initial mapping to avoid such problems. We set up a threshold denoted thr ($0 \leq thr \leq 1$) to trigger the load-balancing process. That is, if the difference between the maximum demand (as defined below) and the average demand among processors becomes larger than thr, then load-balancing is performed. The pseudo-code of the algorithm is shown in Fig. 2.

[7]Note that in order to apply the mapping algorithm in a large-scale systems with many nodes and achieve minimal mapping overheads, the algorithm can be loop unrolled and parallelized, and hence introduces negligible overhead.

LOAD-BALANCING

```
1  while demand_max > (1 + thr) · demand_avg do
2    Order applications on P_max
         by shortest-exeuction-time-first
3    for each application J_k on P_max in order do
4      if e_k < dem_max − demand_min and
           J_k can be assigned to P_min then
5        Remove J_k from P_max and assign it to P_min
6    Update demand_avg, demand_max, P_max
7    if demand_avg, demand_max, and P_max
         remain unchanged then
8      Terminate
```

Figure 2: The psuedo-code of Algorithm *Load-Balancing*.

Definition 4. *The demand of CPU (resp., GPU) P_i is defined to be $demand(P_i) = \sum_{k=1}^{q_i} e_k^c$ (resp., $\sum_{k=1}^{q_i} e_i^g$). Let $demand_{max}$, $demand_{min}$, and $demand_{avg}$ denote the maximum, the minimum, and the average demand among processors respectively. Let P_{max} (resp. P_{min}) denote the processor that has the maximum (resp. minimum) demand.*

Algorithm description. If the condition shown on Line 1 of the pseudo-code of the algorithm in Fig. 2 holds, then the loads on some processors exceed the average load by more than thr. Lines 2-4 balance loads between P_{max} and P_{min}. Specifically, Lines 2-4 try to re-assign some application that is originally assigned to P_{max} to P_{min} while still meeting all deadlines (Line 4). Lines 5-7 provide a termination condition if $demand_{max}$, $demand_{avg}$, and P_{max} remain unchanged, which implies that the algorithm cannot further balance the load. Note that a small value of thr can lead to more balanced loads but at the expense of higher complexity.[8]

Example Consider the example assignment shown in Example 2. For this example, $demand(P_1) = 4$, $demand_{max} = demand(P_2) = 7$, $demand_{avg} = 5.5$. Let $thr = 20\%$ for this example. By Algorithm *Load-Balancing*, since J_2 has the smallest execution cost and can be assigned to P_1, J_2 is removed from P_2 and assigned to P_1. This leads to more balanced loads between P_1 and P_2, where $demand(P_1) = demand(P_2) = demand_{max} = demand_{avg} = 6$.

Voltage/frequency scaling. After the assignment and the load-balancing phase, we adjust the voltage level on each processor in order to reduce power and energy consumption. The following theorem computes the voltage level that should be set at time 0 in order to achieve the best energy-efficiency.

Theorem 2. *At time 0, the optimal voltage level on processor P_i (either a CPU or a GPU) to minimize power and energy consumption while meeting all the deadlines is constant and equal to $max(v_1, v(load(P_i, 0)))$, where $load(P_i, 0)$*

[8]Since both the assignment and the load-balancing can be executed offline, in practice it might be desirable to have a small value of thr in order to obtain a more balanced mapping.

is the processor load of P_i and $v(load(P_i,0))$ denotes the lowest voltage level of P_i that is no less than $load(P_i,0)$.

PROOF. Since the power-voltage relationship is a monotonic function (as discussed in Sec. 3), we can reduce the power and energy consumption by maintaining a fixed voltage level while fully utilizing the processor to the maximum extent. Thus, power and energy consumption can be reduced to the maximum extent under the lowest voltage level under which deadlines can be guaranteed. Thus, to prove the theorem, it is sufficient to prove that $max(v_1, v(load(P_i,0)))$ is the lowest voltage level under which all deadlines can be guaranteed. Depending on the relationship between v_1 and $v(load(P_i,0))$, we have two cases.

Case 1: $v(load(P_i,0)) \leq v_1$**.** In this case, $max(v_1, v(load(P_i,0))) = v_1$, which is the lowest possible voltage level on P_i. For conciseness, assume that P_i is a CPU (the argument holds for the case where P_i is a GPU as well). As discussed in Sec. 3, under v_1, for any application J_o, its execution time is given by $\frac{e_o^c}{v_1}$. By Def. 3 and the definition of $v(load(P_i,0))$, we have

$$v_1 \geq v(load(P_i,0)) \geq load(P_i,0). \quad (1)$$

Thus, for P_i operated under voltage level v_1, its processor load can be re-calculated by $max_{k=1}^{q_i}\Big(\frac{\sum_{o=1}^{k} e_o^c/v_1}{d_k}\Big) = load(P_i,0)/v_1 \overset{\{\text{by (1)}\}}{\leq} 1$. Therefore, by Theorem 1, v_1 is the lowest voltage level under which all deadlines are met.

Case 2: $v(load(P_i,0)) > v_1$**.** In this case, $max(v_1, v(load(P_i,0))) = v(load(P_i,0))$, which from the statement of the theorem is the lowest voltage level of P_i that is no less than $load(P_i,0)$. That is,

$$v(load(P_i,0)) \geq load(P_i,0) \quad (2)$$

holds. Under $v(load(P_i,0))$, for any application J_o, its execution time is given by $\frac{e_o^c}{v(load(P_i,0))}$. Thus, for P_i operated under voltage level $v(load(P_i,0))$, its processor load can be re-calculated by $max_{k=1}^{q_i}\Big(\frac{\sum_{o=1}^{k} e_o^c/v(load(P_i,0))}{d_k}\Big) = load(P_i,0)/v(load(P_i,0)) \overset{\{\text{by (2)}\}}{\leq} 1$. Thus, under voltage level $v(load(P_i,0))$, all deadlines are met. Now we prove that $v(load(P_i,0))$ is the lowest voltage level that guarantees to meet all deadlines. For P_i operated under any voltage level $v' < load(P_i,0)$, the execution time of any application J_o is given by $\frac{e_o^c}{v'}$. Therefore, the processor load on P_i under v' becomes $v(load(P_i,0)) = max_{k=1}^{q_i}\Big(\frac{\sum_{o=1}^{k} e_o^c/v'}{d_k}\Big) = load(P_i,0)/v' > 1$, which by Theorem 1 implies that deadlines may be missed under v'. □

Example Again, reconsider the system given in Example 4. Assume that both CPUs and GPUs have three voltage/frequency steps: $v_1 = 0.5$, $v_2 = 0.8$, and $v_{max} = 1$. By Theorem 2, since $load(P_1,0) = 3/5$ and $load(P_2,0) = 6/15$, we have $v(load(P_1,0)) = v_2 = 0.8$ and $v(load(P_2,0)) = v_1 = 0.5$. Thus, P_1 can be operated under a voltage of v_2 and P_2 can be operated under a voltage of v_1. All applications can still meet their deadlines after reducing voltages.

DYNAMIC-MAPPING

```
1  Assign J_k to its favorite processors by first-fit
2  if J_k cannot be assigned then
3    for each processor P_i of J_k's favorite type do
4      Order applications on P_i by
         lowest-heterogeneity-ratio-first
5      for each application J_h on P_i in order do
6        for each processor P_a other than P_i do
7          if J_h can be assigned to P_a and J_k can be
             assigned to P_i after removing J_h from P_i then
8            Assign J_h to P_a and assign J_k to P_i
9            Order applications on P_a and P_i by EDF
10   if J_k still cannot be assigned to any processor then
11     for each processor P_i of J_k's non-favorite type do
12       Assign J_k to P_i
13       Order applications on P_i by EDF
14 for each processor P_i whose application queue
     is changed do
15   Re-adjust P_i's voltage to be max(v_1, v(load(P_i, t)))
16 if an early completion is detected on processor P_i
     at time t then
17   Re-adjust P_i's voltage to be max(v_1, v(load(P_i, t)))
```

Figure 3: The pseudo-code of Algorithm *Dynamic-Mapping*.

5. DYNAMIC MAPPING WITH BUDGET RECLAMATION

In practice, it is often the case that applications arrive at run-time. Thus, in this section, we propose an algorithm based upon the static mapping algorithm to handle the dynamic arrival case. In practice, applications may often complete earlier than under the worst case. Thus, if an application completes earlier, then our algorithm takes advantage of processing capacity made available and adjusts the voltage on that application's processor on-the-fly to enable more power and energy savings. Finally, if an incoming application with a large heterogeneity ratio cannot be assigned to its favorite processor, then our algorithm swaps this application with some application with a small heterogeneity ratio that has been assigned to one of the incoming application's favorite processors.

Under the dynamic mapping algorithm, the voltage level will be re-adjusted in two circumstances: (i) when a new application arrives, and (ii) an early completion is detected. The pseudo-code of the algorithm is shown in Fig. 3. Assume that an application J_k arrives at time t.

Algorithm description. Algorithm *Dynamic-Mapping* assigns an application arriving at run-time to a processor and re-adjusts the processor's voltage accordingly. When an application J_k arrives at run-time, J_k is assigned to some processor of its favorite type (Line 1). If none of J_k's favorite processors can accommodate it, then the algorithm checks whether there exists an application J_h queued[9] on one of J_k's favorite processors P_i such that J_k can be assigned to P_i after removing J_h from P_i, and J_h can be assigned to some other processor P_a (Lines 2-7). If J_k still cannot be assigned to one of its favorite type of processors, then J_k is assigned to one of its non-favorite type of processors (Lines 8-10). Finally, if any processor's queue has been changed or an early completion is detected, then the voltage/frequency of this

[9]Note that J_h must be queued and is not executing

processor is re-adjusted (Lines 11-14). Note that the algorithm fails if the application cannot be assigned in Lines 1, 8, and 12. Also note that since compilation at run-time, which is needed in order to dynamically map application, may cost significant power and energy consumption, one optimization step is to pre-compile all types of applications that arrive at run-time.

As discussed in Sec. 3, executions on GPUs are non-preemptive. Thus, we are not able to apply Theorem 1 as a criteria for successful assignment since it assumes preemptive EDF and targets applications that arrive at time 0. The following theorem provides such a criteria for the dynamic arrival case. Consider the case where an application J_k comes at time t at run-time and is mapped to processor P_i by Algorithm *Dynamic-Mapping*. Let J_x denote the application that is executing or just completes at t. Let e'_x denote the remaining execution time of J_x after t.

Theorem 3. *An application J_k arriving at time t at run-time can be successfully assigned to processor P_i if $load(P_i, t) \leq 1$ and $load(P_i, t + e'_x) \leq 1$.*

PROOF. If J_x completes at t or J_k has a later deadline than J_x, then by Theorem 1, $load(P_i, t) \leq 1$ is a sufficient condition for J_k to be successfully assigned to P_i because in these cases no preemption may occur after t. The other case is that J_x is executing at t but has a deadline later than J_k. Under this case, J_k has an earlier deadline than J_x but cannot preempt J_x. Thus, we need to check the processor load again when J_x completes. When J_x completes at time $t+e'_x$, applications queued on P_i including J_k can be executed in the EDF order and no preemption may occur after $t + e'_x$. That is, $load(P_i, t + e'_x) \leq 1$ provides a sufficient condition for J_k to be successfully assigned to P_i. □

Time complexity. The time complexity of Algorithm *Dynamic-Mapping* is dominated by the three loops in Lines 3-7, which have a complexity of $O(m^2 \cdot n^2)$.

Example Consider the same system as presented in Examples 4-4. Now assume that an application J_7 arrives at time 4 at run-time, whose $e^c_7 = 4$, $e^g_7 = 6$, and $d_7 = 11$. By Def. 2, CPU is the favorite type of processor of J_7. By Theorem 1, it cannot be assigned to P_1. By Algorithm *Dynamic-Mapping*, J_5 is removed from P_1 and assigned to P_2, and J_7 can be assigned to P_1. After the assignment, $load(P_1, 4) = 4/7$ and $load(P_2, 4) = 0.79$. Thus, by Theorem 2, both P_1 and P_2 can be operated with a voltage of v_2.

6. AGGRESSIVE VOLTAGE REDUCTION

Although the static and dynamic mapping algorithms provide sound assignment and voltage scaling schemes, they all assume the worst-case workloads. This assumption is conservative and pessimistic since the run-time execution time is often shorter than the worst-case. In this section, we propose an aggressive voltage reduction scheme by speculating that workloads will most likely present an average-case computational demand that is lower than the worst-case demand.

The mapping scheme is the same as the algorithm presented in previous sections, except that voltages are set assuming average-case execution times instead of the worst-case.[10] Since we assume average-case execution times, we may need to adjust the voltage/frequency when an early or late completion is detected. We apply similar schemes as used in Algorithm *Dynamic-Mapping* for the adjustment. Specifically, when an early or late completion is detected on processor P_i at time t, $load(P_i, t)$ is re-computed and P_i's voltage is adjusted to be $max(v_1, v(load(P_i, t)))$. Note that some applications may miss their deadlines if late completions occur. Thus, the aggressive voltage reduction algorithm may reduce power and energy consumption at the cost of missing deadlines.

Example For the example application system as presented in Example 4, assume that the average execution time for all six applications is 0.5 time unit less than the corresponding worst-case execution time. Then, by the above aggressive voltage reduction algorithm, both P_1 and P_2 can be operated under a voltage of v_3 since by using the average-case execution times, $load(P_1, 0) = 0.3$ and $load(P_2, 0) = 0.4$. This further reduces the voltage compared to the case where the static or the dynamic mapping algorithm is used.

In order to balance the tradeoff between performance and energy efficiency, we introduce a parameter K for each processor to determine the aggressiveness level. On any processor, K can be set to be $\frac{v_{avg}}{v_{worst}} \leq K \leq 1$, where v_{avg} is the voltage level calculated by assuming average-case execution times and v_{worst} is the voltage level calculated by assuming worst-case execution times.[11] For applications whose run-time execution times are often consistent, it is preferable to set $K = \frac{v_{avg}}{v_{worst}}$ in order to achieve maximal power and energy efficiency. On the other hand, for applications whose run-time execution times often vary, it is preferable to set K to be a larger value than $\frac{v_{avg}}{v_{worst}}$ in order to avoid late completions that may cause deadline misses.

7. EVALUATION

In this section, we present the implementation methodology and experimental results used to evaluate the effectiveness of our proposed algorithms.

7.1 Methodology

We implemented the proposed algorithms in a real heterogeneous desktop computer consisting of a CPU and a GPU. The hardware specification is given in Table 2. The benchmarks used in the experiments are listed in Table 3. Each benchmark set consists of the same corresponding application but with multiple input sizes. For the GPU implementations of the benchmarks, we used the ones provided by IBM or in the AMD OpenCL Software Development Kit (SDK) [9] if available.

Regarding the software architecture of the implemented experimental system, we provide an API to programmers for writing GPU-parallelizable operations at the application level. The API is built on top of C/C++ so that it can be easily adopted. Beneath the API layer is the system layer,

[10] The average-case information may not need to be accurate, as we shall discuss later.

[11] Note that since K can be considered as a run-time optimization parameter, we do not require accurate information on a workload's average-case execution time. If the actual run-time execution time is more or less than the assumed average-case execution time, then we can adjust K and the voltage/frequency accordingly.

Table 2: Experimental Hardware Specification.

	CPU	GPU
Architecture	Intel Xeon 5160	AMD Radeon HD 5770
Voltage/Frequency levels	3GHz (1.5v) / 2GHz (1.2v)	850MHz(1.25v) / 500MHz(1v) / 148MHz(0.95v)
TDP	80W	108W
OS	64-bit Linux Ubuntu lucid	

Table 3: Benchmarks.

Benchmarks	Description	Source
Non-negative Matrix Factorization (NMF)	OpenCL implementation of NMF	IBM
MatrixMultiplication (MM)	Dense matrix multiplication	AMD OpenCL SDK
MatrixTranspose (MT)	Dense matrix Transpose	AMD OpenCL SDK
MatrixOperation (MO)	Mixed operations on dense matrix	AMD OpenCL SDK

which consists of a compiler and a scheduler. The compiler intercepts OpenCL function calls at compile, replacing them with functionally equivalent calls into its heterogeneous scheduling runtime. It transforms an input OpenCL program to use all OpenCL devices available. Our proposed scheduling algorithms are implemented in the scheduler that schedules applications to run on the CPU or GPU.

We compared the static mapping scheme, the dynamic mapping scheme with budget reclamation, and the aggressive voltage reduction scheme with each other. Moreover, we compared these three proposed schemes with a performance-driven mapping algorithm, which is called earliest-response-time-first (ERF). ERF maps an application onto the resource that provides the shortest response time. In all experimental results presented below, we denote our static mapping, dynamic mapping with budget reclamation, aggressive voltage reduction schemes, and the ERF scheme as "Static," "Dynamic," "Aggressive," and "ERF" respectively.

Our evaluation metrics are the power and energy consumption. We measured the entire system power (including the CPU, GPU, memory, and disks) using the WT210 digital power meter [25]. Also we measured the wall-clock execution time for each experiment. Then the total energy consumption can be obtained by computing the product of power and execution time.

Generate application parameters. In our experiments, the benchmark sets were generated as follows. For each experiment, we randomly generated a number of benchmarks with different input matrix files. To obtain the execution time of a benchmark, we ran the benchmark repeatedly for thirty minutes and recorded the maximum execution time among these runs as the measured execution time. The worst-case execution time (WCET) of each benchmark was then calculated based upon the corresponding measured execution time (MET): $WCET = 120\% \cdot MET$. Per-benchmark loads were uniformly distributed in [0.001, 0.1]. Benchmark deadlines were then calculated from worst-case execution times and benchmark loads. For each experiment, the cap on the sum of all benchmarks' loads was varied by three values: 1 (light system load), 1.4 (medium system load), and 1.8 (heavy system load). Each benchmark set was generated by creating benchmarks until the sum of all benchmarks' loads exceeded the corresponding system load cap, and by then reducing the last benchmark's load so that the sum of all benchmarks' loads equalled the system load cap.

Applying DVFS on CPUs and GPUs. To enable voltage/frequency scaling on the CPU, we applied the CPUFreq kernel infrastructure [2] that implements CPU voltage/frequency scaling. It enables the operating system to scale the CPU voltage/frequency up or down in order to save power/energy. Most modern operating systems including Linux Ubuntu Lucid support CPUFreq.

Unfortunately, voltage/frequency scaling on GPUs is not easy. The major difficulty is that there is no OpenCL-supported ATI GPU driver that supports voltage/frequency scaling. The official ATI GPU driver supports OpenCL, but does not support voltage/frequency scaling. On the other hand, the open-source ATI GPU driver called RadeonDriver [1] that comes with Ubuntu supports voltage/frequency scaling via the Kernel Mode Setting (KMS). However, RadeonDriver does not support OpenCL. Therefore, we applied a modeling approach to obtain the GPU power values under different voltage levels, as explained below.

By using the RadeonDriver, we are able to obtain the GPU power when running non-OpenCL applications under different voltage/frequency levels. Let $P(1, A)$ (respectively, $P(v_i, A)$) denote the GPU power when running non-OpenCL application A under the maximum voltage level (respectively, the i^{th} voltage level v_i). Moreover, by using the official ATI GPU driver, we are able to obtain GPU power when running OpenCL applications under the maximum voltage/frequency. Let $P(v_{max}, A_{CL})$ denote the GPU power when running OpenCL application A_{CL} under the maximum voltage/frequency. Then, we can calculate the GPU power when running A_{CL} under the i^{th} voltage/frequency level, denoted $P(v_i, A_{CL})$, as follows.

$$P(v_i, A_{CL}) = \frac{P(v_{max}, A_{CL})}{P(v_{max}, A)} \cdot P(v_i, A).$$

7.2 Results

In this section, we present results on power and energy consumption that compare different algorithms.

Power consumption. Figs. 4 (a)-(c) show the power consumption results under light, medium, and heavy system loads. Each box in each figure plots the normalized power consumption under ERF, Static, Dynamic, and Aggressive for running the four benchmarks. As seen, our proposed algorithms can achieve a much better power efficiency compared to ERF in all scenarios. For example, under the light load, our algorithms can reduce the power consumption for around 20%. Another observation is that under the light system load, Static, Dynamic, and Aggressive achieve the same power consumption. This is due to the fact that when the system load is light, Static (as well as Dynamic and Aggressive) can reduce the voltage on CPUs and GPUs to the lowest level. While under medium and heavy system loads, it is seen that Aggressive and Dynamic can reduce more power consumption compared to static. This is because Aggressive and Dynamic can utilize the budget due to the difference between the actual execution time and the worst-case execution time to further reduce the voltage/frequency.

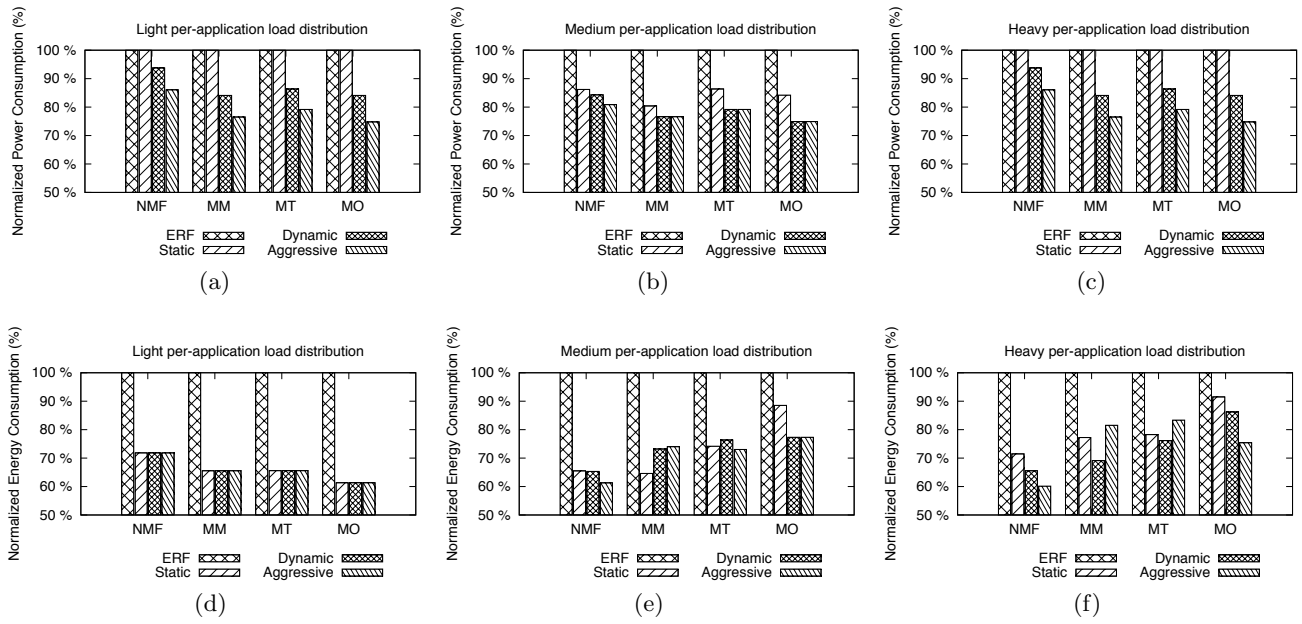

Figure 4: Power and energy consumption results. In the first (resp., second and third) column of graphs, light (resp., medium and heavy) loads are assumed. The first (resp., second) row shows the power (resp., energy) consumption results.

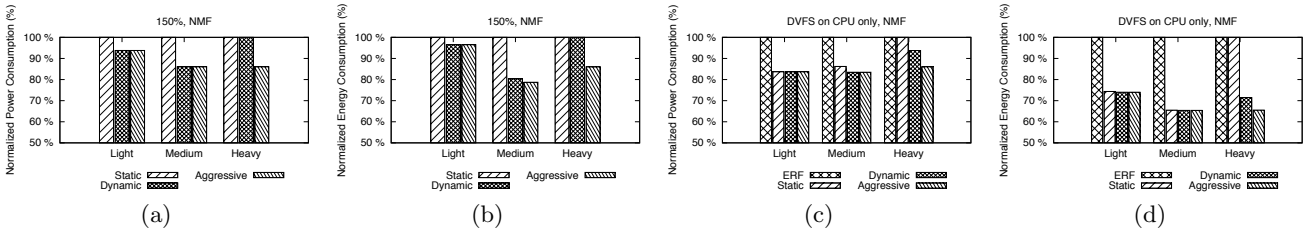

Figure 5: (a)-(b). Results when the worst-case execution times equal 150% of the measured execution times. (a) and (b) show the power and energy consumption results. (c)-(d). Results with CPU DVFS only. (c) and (d) show the power and energy consumption results.

Energy consumption. Figs. 4 (d)-(f) show the energy consumption results under light, medium, and heavy system loads. Each box in each figure plots the normalized energy consumption under ERF, Static, Dynamic, and Aggressive for running the four benchmarks. It can be observed that Static, Dynamic, and Aggressive can achieve a much better energy efficiency compared to ERF in all scenarios. For example, under the light system load, Static, Dynamic, and Aggressive can reduce the energy consumption for more than 27%. In certain scenarios, such improvement can reach up to 40% (e.g., when running NMF under the heavy per-application load distribution using Aggressive). An interesting observation is that under medium and heavy system loads, Aggressive consumes more energy than Static and Dynamic for certain benchmarks such as MatrixMultiply. This is because when the voltage is reduced, the execution time increases, which causes an increase on the energy consumed by the whole system (including memory and disks).

More conservative worst-case execution times. As seen in Fig. 4, in most cases, Aggressive achieves power and energy consumption results that are close to those of Static and Dynamic. In this experiment set, we want to evaluate the effectiveness of Aggressive when the ratio of the worst-case execution time over the measured execution time increases. Intuitively, Aggressive is expected to reduce more power/energy consumption when this ratio increases, since in this case Aggressive is able to reduce the voltage to the lowest possible level by assuming the average-case execution time. Thus, in this experiment set, we set the worst-case execution times to equal 150% of the measured execution times. Fig. 5 (a) and (b) show the power and energy consumption under Static, Dynamic, and Aggressive for running the NMF benchmark.[12] As seen, compared to the case shown in Fig. 4, Aggressive is able to reduce power and energy consumption more. For example, under the heavy system load, Aggressive can reduce 13% more power and energy consumption compared to Static and Dynamic. From these results, we claim that for benchmarks whose worst-case execution times are often longer than the average-case, Aggressive can achieve better energy efficiency.

DVFS on CPUs only. Since we can only apply a modeling approach to obtain the GPU power under different voltage levels, it is important to know whether the proposed

[12] Note that ERF is not included in this experiment set since our focus is to evaluate whether Aggressive can achieve better performance by exploiting average-case execution time information.

algorithm can still save power/energy by applying DVFS on CPUs only. Thus, we conducted an additional set of experiments, in which voltages on CPUs are still scaled according to the proposed algorithm, but GPUs are always set to run under the maximum voltage level (i.e., the DVFS step of the algorithms is not executed on GPUs). Fig. 5 (c) and (d) show the power and energy under ERF, Static, Dynamic, and Aggressive for running the NMF benchmark. As seen, our proposed algorithms reduce the power and energy consumption by a considerable amount compared to ERF in all scenarios. For example, under the light load, our algorithms can reduce 16% more power consumption and 25% more energy consumption.

8. CONCLUSION

In this paper, we proposed power-efficient time-sensitive mapping techniques for heterogeneous systems consisting of CPUs and GPUs. These techniques were implemented in a real CPU/GPU heterogeneous system. Experimental results demonstrate the effectiveness of the proposed techniques in reducing both power and energy consumption.

Several interesting avenues for further work exist. It would be interesting to consider heterogeneous systems consisting of more than two types of processors (e.g., a system containing CPUs, GPUs, and FPGAs). Moreover, contrary to our current strategy where each GPU and CPU has its own scheduler and application queue, designing a centralized scheduler that captures the information on both CPUs and GPUs at run-time may be able to make better mapping decisions in terms of reducing power and energy consumption.

9. REFERENCES

[1] AMD GPU Open-Source Drivers. www.help.ubuntu.com/community/RadeonDriver, 2011.

[2] CPUFreq technology. www.wiki.archlinux.org/index.php, 2011.

[3] B. Andersson, G. Raravi, and K. Bletsas. Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. In *Proc. of the 31st Real-Time Sys. Symp.*, pp. 239-248, 2010.

[4] C. Augonnet, J. Clet-Ortega, S. Thibault, and R. Namyst. Data-aware task scheduling on multi-accelerator based platforms. In *Proc. of the 16th international conference on parallel and distributed systems*, 2010.

[5] S. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In *Proc. of the 33rd International Conference on Parallel Processing*, pp. 467-474, 2004.

[6] J.J. Chen and C. F. Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *Proc. of the 13th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 28-38, 2007.

[7] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *Proc. of the 2005 ACM SIGMETRICS International conference on Measurement and modeling of computer systems*, pp. 303-314, 2005.

[8] Khronos Corporation. The OpenCL Language. www.khronos.org/opencl, 2011.

[9] Advanced Micro Devices. AMD APP SDK. www.developer.amd.com/sdks/AMDAPPSDK/, 2011.

[10] G. Elliott and J. Anderson. Real-World Constraints of GPUs in Real-Time Systems. In *Proceedings of the First International Workshop on Cyber-Physical Systems, Networks, and Applications*, pp. 48-54, 2011.

[11] M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. of the 2nd Workshop on Power Aware Computing Sys.*, 2002.

[12] R. Wilhelm et al. The worst-case execution-time problem-overview of methods and survey of tools. In *ACM Transactions on Embedded Computing Systems*, Vol. 7, Issue 3, 2008.

[13] S. Ghiasi, T. Keller, and F. Rawson. Scheduling for heterogeneous processors in server systems. In *Proc. of the 2nd Conference on Computing Frontiers*, pp. 199-210, 2005.

[14] IBM. Big data analytics. http://www.ibm.com/software/data/infosphere/bigdata-analytics.html, 2011.

[15] V. Jimenez, L. Vilanova, I. Gelado, M. Gil, G. Fursin, and N. Navarro. Predictive runtime code scheduling for heterogeneous architectures. In *Proc. of the 4th International Conference on High Performance Embedded Architectures and Compilers*, pp. 19-33, 2009.

[16] Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztián Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, December 2003.

[17] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan. Heterogeneous Chip Multiprocessors. In *IEEE Computer*, pp. 32-38, 2005.

[18] D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Proc. of the Intl. Conf. on Neural Info. Processing Sys.*, pp. 556-562, 2000.

[19] J. Liu. Real-time systems. In *Prentice Hall*, 2000.

[20] Y. Liu and H. Zhu. A survey of the reserach on power management techniques for high-performance systems. In *Journal of Software - Practice and Experience*, Vol. 40, Issue 11, 2010.

[21] C. Luk, S. Hong, and H. Kim. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Proc. of the 42nd International Symp. on Microarchitecture*, pp. 45-55, 2009.

[22] S. Pak. Azul Systems extends leadership in business critical Java applications performance with the new Vega series. www.azulsystems.com/press/052008vega3.htm, May 2008.

[23] J. Rabaey. Digital integrated circuits. In *Prentice Hall*, 1996.

[24] R. Wu, B. Zhang, and M. Hsu. GPU-Accelerated Large Scale Analytics. In *HP Labotories Technical Report*, HPL-2009-38, 2009.

[25] Yokogawa. Wt210 digital power meter. www.yokogawa.com/tm/Bu/WT210, 2011.