Energy Efficient Job Scheduling with DVFS for CPU-GPU Heterogeneous Systems

Vincent Chau
Department of Computer Science
Hong Kong Baptist University
Hong Kong
vincchau@comp.hkbu.edu.hk

Xiaowen Chu
Department of Computer Science
Hong Kong Baptist University, Hong Kong
HKBU Institute of Research and Continuing Education
Shenzhen, China
chxw@comp.hkbu.edu.hk

Hai Liu
Department of Computing
Hang Seng Management College
Hong Kong
hliu@hsmc.edu.hk

Yiu-Wing Leung
Department of Computer Science
Hong Kong Baptist University
Hong Kong
ywleung@comp.hkbu.edu.hk

ABSTRACT

The past few years have witnessed significant growth in the computational capabilities of GPUs. The race for computing performance makes the uses of many-core accelerators more necessary. However, GPUs consume a significant amount of energy as compared with CPUs. One way to reduce the energy consumption is to scale the speed and/or voltage of the processor. Typically, the faster the processor runs, the faster we finish jobs, but the more power is required by the processor. It is hence important to balance between performance and power consumption.

In this paper, we consider the following scheduling problem. We have a set of jobs to be assigned to different processors. Each job may have different characteristics depending on the type of processor that it is assigned to. The goal is to minimize the total energy consumption. After proving the NP-hardness of this problem, we propose a constant approximation algorithm for the case when processors can scale to any continuous speed. When processors have a set of discrete speeds, we propose a heuristic algorithm and compare with some classical scheduling algorithms experimentally. Then, we extend this heuristic to the online case where jobs arrive over time. Our simulation results show that the proposed heuristic algorithms are effective and can achieve near-optimal performance.

ACM Reference format:
Vincent Chau, Xiaowen Chu, Hai Liu, and Yiu-Wing Leung. 2017. Energy Efficient Job Scheduling with DVFS for CPU-GPU Heterogeneous Systems. In *Proceedings of e-Energy'17, Hong Kong, May 17-19, 2017,* 11 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Power management aims in reducing the energy consumed by computer systems while maintaining a good level of performance. Especially, distributed systems need to satisfy the demands to the detriment of energy consumption. As the demand of computation and data processing is growing exponentially, researchers have moved to high-performance computing platforms. The race for computing performance makes the uses of many-core accelerators (such as GPUs, Intel MICs, and FPGAs) more necessary. E.g., GPU-based cloud computing has been recently introduced by Amazon, Microsoft Azure, Google, IBM, Aliyun, etc. This is because the state-of-the-art GPUs are always around one order of magnitude faster than mainstream CPUs in terms of raw computational power. Furthermore, GPUs are more power-efficient than CPUs in terms of Flops-per-Watt [11, 26]. Indeed, in the TOP500 list of the supercomputers, 94 of them are equipped with GPUs in June 2016. However, GPUs consume significant amount of energy as compared with CPUs. For example, the Titan supercomputer is equipped with 18,688 NVIDIA Tesla K20X GPU cards and consumes about 8.21 million Watts at full load, resulting in about 23 million dollars per year of electricity bill. Hence how to reduce the energy consumption by GPUs becomes a key issue for data centers that host CPU-GPU heterogeneous systems.

Because of the high energy consumption of GPUs, power management techniques are essential for both CPUs and GPUs. While the power management in CPUs has been widely studied, the combination of CPU-GPU has not been fully explored yet.

One traditional way to reduce the energy consumption is to scale the speed and/or voltage of the processor. The faster the processors run, the more power they consume. It becomes possible to run the given jobs with slower speed to reduce the energy usage. This mechanism, used to save energy, is known as the Dynamic Voltage Frequency Scale (DVFS) mechanism where the speed (or frequency) and voltage of the processors can be changed during the execution. The theoretical study of this model was initiated in a seminal paper by Yao, Demers and Shenker in [31]. The processor can execute at most one job at each time. Each job is defined by a processing requirement which corresponds to the number of cycles that a job needs to be completed. If $f(t)$ denotes the speed (or frequency) of the processor at time t, then the total amount of work executed by the processor during an interval of time $[t, t')$ is equal to $\int_t^{t'} f(u)du$. The processor's dynamic power consumption follows the well-known cube-root rule for CMOS devices [15, 16]. It can be considered as $P \propto f^\alpha$ which is proportional to f^α with $\alpha > 1$ [12, 20, 31].

On the other hand, whenever the processor is on active state, it consumes a fixed amount of static energy per unit of time due to transistors leakage.

Our goal is to minimize the total energy consumption, i.e., the sum of *dynamic* and *static* energy. It can be noticed that the faster the processors run, the faster we complete jobs but the more power is required by the processors. We aim to find a tradeoff between the static and the dynamic energy by assigning jobs to appropriate processors.

In this paper, we consider the following scheduling problem for GPU-accelerated data centers. We have a set of jobs to be assigned to different processors, including both CPUs and GPUs. Each job may have different characteristics depending on the type of processor that it is assigned to. Especially, jobs are much more effective when they are assigned to GPUs compared to CPUs. The goal is to minimize the total energy consumption.

1.1 Related Work

In the following, we refer to *homogeneous* environment when jobs have the same processing requirement on each processor while the *heterogeneous* environment is when jobs may have different processing requirement depending on the processor that it is assigned to. The processing requirement of a job corresponds to the number of operations a processor needs to do in order to complete the job.

In the context of CPU-GPU, Chen et al. [13] considered the case of mixed CPU-GPU without DVFS. Different from our problem, they considered that jobs have fixed processing time and their goal is to minimize the makespan, i.e., the time duration up to the moment that the last job finishes. They studied the online case where jobs are revealed over time. They proposed a 3.85-competitive algorithm for this problem and improved the competitive ratio for some special cases.

A series of papers have appeared on the multiprocessor environment with frequency scaling. Albers et al. [3] considered the energy minimization problem on homogeneous processors. They showed that the problem is strongly NP-hard when jobs have common release time and common deadline. Here, each job needs to be scheduled within their own release time and deadline. Greiner et al. [17] proposed a $B_{\lceil\alpha\rceil}$-approximation algorithm for the homogeneous multiprocessor problem where $B_{\lceil\alpha\rceil}$ is the $\lceil\alpha\rceil$-th Bell number. Bampis et al. [9] improved the result by proposing a generalized Bell number, i.e., $B_\alpha = \sum_{k=0}^{\infty} \frac{k^\alpha e^{-1}}{k!}$ and gave a $(1+\varepsilon)B_\alpha$-approximation algorithm for the heterogeneous multiprocessor. Later, Cohen-Addad et al. [14] proved that the energy minimization problem on heterogeneous multiprocessor is APX-hard, i.e., there is no PTAS (Polynomial Time Approximation Scheme) for this problem. Liu et al. [22] considered the case when jobs can migrate between two different core. They showed that their approach can reduce the energy consumption by more than 50% compared with traditional approaches. Ma et al. [23] proposed a holistic energy management framework for GPU-CPU heterogeneous architectures and showed that it achieves 21.04% average energy savings.

Mei et al. [25, 26] investigated the impact of GPU DVFS on energy saving. They showed that scaling the frequency of the GPU can reduce the energy consumption. Note that all these works did not consider the static energy.

Recently, Mei et al. [24] considered the CPU-GPU DVFS problem. They proposed an algorithm to conserve energy consumption without violating the deadlines of the jobs. They showed that as much as 36% of the energy consumption can be saved.

Another related work is to minimize the makespan given an energy budget. Pruhs et al. [29] first gave a $\log(m)$-approximation where m is the number of processors. It was then improved to 2-approximation by Bampis et al. [10].

Some works have been focusing on the load balancing problem. There is a set of jobs to be assigned to a set of processors, and the goal is to minimize the following cost: $\sqrt[\alpha]{\sum_{i=1}^m W_i^\alpha}$ where W_i is the workload of processor i. This problem is also named the L_α-norm problem where $\alpha > 1$ is the dimension of the vector space. This problem is similar to minimizing the energy consumption when jobs have common release time and common deadline. Indeed, the energy consumption can be obtained by making the L_α-norm cost to the power of α then multiply by the common deadline. Awerbuch et al. [6] showed that there exists a constant approximation algorithm for the problem of L_α-norm. Later, Alon et al. [4] proposed an approximation scheme for this problem. Finally, Avidor et al. [5] gave a greedy algorithm that is a $\left(2 - \Theta\left(\frac{\ln \alpha}{\alpha}\right)\right)$-approximation for the L_α-norm under the homogeneous environment. Azar and Epstein [7] proposed a 2-approximation algorithm for the L_α-norm under the heterogeneous environment. Note that this approximation ratio does not depend on the parameter α anymore.

Recent surveys of the area can be found in [1, 2, 8, 11, 25, 27]. We show in Table 1 the major difference between existing works and the current paper.

Table 1: Summary of different related problems

Problem	Results
Minimizing makespan s.t. energy consumption	$\log(m)$-approximation [29] 2-approximation [10]
Minimizing energy consumption s.t. makespan	NP-hardness [3] $(1+\varepsilon)B_\alpha$-approximation [9]
Minimizing total energy consumption with makespan cost	This paper

1.2 Our contributions

Our major contributions in this work can be summarized as follows.

- we prove the NP-hardness of the problem.
- we propose a greedy 2^α-approximation algorithm for the case when processors can scale to any continuous speed.
- we investigate the performance of the greedy algorithm through simulations on real setting and compare with some

classical scheduling algorithms. We show that the proposed algorithm is effective and can achieve near-optimal solution.
- we extend the greedy algorithm to the online case where jobs are revealed only we reach this time. We show that by modifying the input data, we can also obtain a near-optimal solution.

The rest of the paper is organized as follows. In Section 2, we formulate the scheduling problem as an optimization problem and prove its NP-hardness. Then in Section 3, we propose a 2^α-approximation algorithm for the case when processors can scale to any continuous speed. In Section 4, we consider the case when processors only have a set of discrete speeds that can be scaled to. We propose a heuristic algorithm and compare with some classical scheduling algorithms through simulations in terms of energy savings and execution time. Based on this heuristic, we extend it to the online case where jobs arrive over time in Section 5. Finally, we conclude the paper in Section 6.

2 PROBLEM DEFINITION AND NOTATIONS

In data center context, users have requests to submit to the servers. The servers have to deal with them in order to satisfy the users. It can be formalized as a scheduling problem where we have a set of jobs to process by a set of processors. Since servers have different types of processors, the processing time may be affected. Each job j is defined by a processing requirement $p_{i,j}$ if it is assigned to processor i which may be different depending on the type of processor. The processing requirement corresponds to the number of cycles that a job needs to be completed. Moreover, we have a set of processors that can scale their speed. We distinguish two components of power consumption. The first one is the *dynamic* power which is related to the frequency and the voltage, and the second component is the *static* power consumption which depends on the time we keep the processor on active state. That is $P = P_{dynamic} + P_{static}$.

The dynamic power of a processor is modeled as

$$P_{dynamic} = K_{eff} \cdot f \cdot V^2$$

where K_{eff} is the effective switching capacitance of the chip, f is the processing speed of the processor, and V is the voltage [12, 20, 31].

It can be noted that $P_{dynamic}$ is a convex and increasing function of the voltage V. Moreover, the speed is bounded by the chosen voltage. Once the voltage is set, the processor can only choose among a set of available speeds. Therefore, if a job is scheduled with a low speed, we can scale the voltage into a lower state in order to save energy.

To simplify the notation, the dynamic power can be written without loss of generality as $P_{dynamic} = f^\alpha$. If the processor runs at speed f during t units of time, then the dynamic energy consumption is $E_{dynamic} = f^\alpha t$.

It can be noticed that if processors have independent completion time, then assigning a job on a processor does not affect the other processors. Therefore, we can assign jobs greedily to the processor that minimizes the total energy consumption.

More generally, suppose first that jobs are already assigned to processors, and let W_i be the total workload assigned to processor i. For a given time C to process the assigned jobs, each processor needs to run at speed W_i/C to get the minimum dynamic energy consumption on each processor since the dynamic power function is convex.

Thus the total dynamic energy consumption is

$$E_{dynamic} = \sum_{i=1}^{m} \left(\frac{W_i}{C}\right)^\alpha C$$

where m is the number of processors.

On the other hand, we also consider the static energy consumption, i.e., whenever the processor i is on active state, the static energy consumption rate is γ_i per unit of time. Therefore if the processor i runs for t units time, then the static energy consumption is $E_{static} = t\gamma_i$. Moreover, we assume that all processors in our system need to stay on active state together, i.e. a processor cannot go into sleep state (and therefore no static energy is consumed) as long as at least one processor is running jobs. Thus, the total energy consumption is defined as

$$E = \sum_{i=1}^{m} \left(\left(\frac{W_i}{C}\right)^\alpha + \gamma_i\right) C = \left(\sum_{i=1}^{m} W_i^\alpha\right) C^{1-\alpha} + C \sum_{i=1}^{m} \gamma_i.$$

Our goal is to minimize the total energy consumption E. In other words, we want to determine the workload W_i of each processor i that is assigned (therefore the assignment of jobs on each processor) and the time C such that all processors stay on active state.

We can assume without loss of generality that $\gamma = \sum_{i=1}^{m} \gamma_i$. Thus, the objective function becomes

$$\left(\sum_{i=1}^{m} W_i^\alpha\right) C^{1-\alpha} + C\gamma$$

We summarize the different notations in Table 2.

Table 2: Notation Summary

n	number of jobs
m	number of processors
f	speed of the processor
γ_i	static energy rate of processor i
γ	static energy rate of all processors
W_i	total workload that is assigned to processor i
C	processing time
α	parameter of the processor
P	power function of the processor
E	energy consumption
$P_{dynamic}$	dynamic power
$E_{dynamic}$	dynamic energy
P_{static}	static power
E_{static}	static energy

We can notice that the dynamic and static energy are opposite. When the processing time decreases, the dynamic energy increases while the static energy decreases. By the same way, if the processing time increases, the dynamic energy decreases while the static energy increases. As it is a sum of two convex functions, we can find the optimal processing time C.

3 APPROXIMATION ALGORITHMS

In this section, we first show that the scheduling problem is NP-hard. Then we show that there exists a greedy algorithm that achieves an approximation of 2^α.

Theorem 3.1. *The problem of assigning jobs to processors and to minimize the total energy consumption is NP-hard.*

Proof. In order to establish the $\mathcal{NP}$-hardness, we present a reduction from the 3-Partition problem which is strongly $\mathcal{NP}$-hard. In the problem of 3-Partition, we are given a set I of $n = 3k$ items and k bins of size B. Each item $j \in I$ has a value $B/4 \leq v_j \leq B/2$. In the decision of the problem, we ask whether there exists a partition of I into k triplets $S_1, S_2, \ldots, S_k$ such that the sum of the value of the items of each triplet is equal to B.

Given an instance of the 3-Partition problem, we construct an instance of our scheduling problem as follow. We have a set of k identical processors where the power function is $P_{dynamic} = f^\alpha$ with $\alpha = 3$ and $P_{static} = \gamma = 2$. For each item $j, 1 \leq j \leq n$, we introduce a job j with $p_{i,j} = v_j$ for each processor $1 \leq i \leq k$.

We claim that the instance of the 3-Partition problem is feasible if and only if there is a feasible schedule for our problem whose cost is $3Bk$.

Let us assume that the instance of the 3-Partition problem is feasible. Therefore, there exists a partition of I into k triplets $S_1, S_2, \ldots, S_k$ such that the sum of the value of the items of each bin is equal to B. Then we can schedule the corresponding jobs of each triplet S_i on processor i with speed 1. The energy consumption for each processor is $1^3 B$ for the dynamic energy and $2B$ for the static energy and the total energy is equal to $3B$. We can see that the speed of 1 leads to the minimum energy consumption. We recall the energy consumption of one processor with a workload of B: $B^\alpha C^{1-\alpha} + \gamma C$ with $\alpha = 3$ and $\gamma = 2$.

In order to find the value of the processing time C that minimizes the energy consumption, we need to verify that the function is convex.

$$(B^\alpha C^{1-\alpha} + \gamma C)'' = (B^3 C^{-2} + 2C)'' = \left(\frac{-2B^3 C}{C^4} + 2\right)'$$
$$= \left(\frac{-2B^3}{C^3} + 2\right)' = \frac{6B^3 C^2}{C^6} \geq 0$$

The global minimum can be found by solving

$$\frac{-2B^3}{C^3} + 2 = 0 \Leftrightarrow B^3 = C^3 \Leftrightarrow B = C$$

Thus the total energy consumption is $3Bk$.

For the opposite direction of our claim, we assume there exists a feasible schedule of cost $3kB$. Let S_i be the set of jobs that are scheduled on processor i. Clearly, due to the convexity of the power function, if there is a processor with workload $B + \varepsilon$ and another processor $B - \varepsilon$ for any $\varepsilon > 0$, the energy consumption will be higher. Indeed, we have $\frac{(B+\varepsilon)^3 + (B-\varepsilon)^3}{C} > \frac{2B^3}{C}$ for any processing time C. Thus, the jobs in each S_i form a partition of the items into k triplets and therefore form a feasible solution for the 3-Partition problem. □

Theorem 3.2. *If there exists a $\sqrt[\alpha]{\rho}$-approximation for the L_α-norm minimization problem, then there exists a ρ-approximation for our problem.*

Proof. If jobs are already assigned to processors, the only variable here is the makespan C, that is the time in which the last jobs finish. Due to the convexity of the dynamic energy cost, the speed will be scaled such that all jobs finish at time C. We recall that the objective function is $\frac{\sum (W_i)^\alpha}{C^{\alpha-1}} + C\gamma$.

Suppose we know the optimal assignment for the following objective function $\mathcal{W} = \sum (W_i)^\alpha$. In the following, we use W_i as the total workload on processor i of this solution. Similarly, let W'_i be the total assigned workload on processor i by our algorithm and the cost $\mathcal{W}' = \sum (W'_i)^\alpha$.

If there exists a $\sqrt[\alpha]{\rho}$-approximation for the L_α-norm minimization problem, then we have $\sqrt[\alpha]{\mathcal{W}} \leq \sqrt[\alpha]{\mathcal{W}'} \leq \sqrt[\alpha]{\rho}\sqrt[\alpha]{\mathcal{W}}$.

Consequently, we obtain

$$\mathcal{W} \leq \mathcal{W}' \leq \rho \mathcal{W} \tag{1}$$

We now show that this approximation ratio still holds for our problem. Let $C = \operatorname{argmin}_x \left(\mathcal{W} x^{1-\alpha} + x\gamma\right)$ and $C' = \operatorname{argmin}_x \left(\mathcal{W}' x^{1-\alpha} + x\gamma\right)$ be the makespan that minimizes the dynamic energy consumption for the respective assignment.

From the above definition, we can deduce that for any other makespan $x \neq C$, we have $C^{1-\alpha}\mathcal{W} + C\gamma \leq x^{1-\alpha}\mathcal{W} + x\gamma$, in particular for $x = C'$.

$$\mathcal{W} C^{1-\alpha} + C\gamma \leq \mathcal{W} C'^{1-\alpha} + C'\gamma \tag{2}$$
$$\mathcal{W}' C'^{1-\alpha} + C'\gamma \leq \mathcal{W}' C^{1-\alpha} + C\gamma \tag{3}$$

Our goal is to show that

$$\mathcal{W} C^{1-\alpha} + C\gamma \leq \mathcal{W}' C'^{1-\alpha} + C'\gamma \leq \rho(\mathcal{W} C^{1-\alpha} + C\gamma)$$

We have

$$\begin{aligned}
\mathcal{W} C^{1-\alpha} + C\gamma &\leq \mathcal{W} C'^{1-\alpha} + C'\gamma && \text{by (2)}\\
&\leq \mathcal{W}' C'^{1-\alpha} + C'\gamma && \text{by (1)}\\
&\leq \mathcal{W}' C^{1-\alpha} + C\gamma && \text{by (3)}\\
&\leq \rho \mathcal{W} C^{1-\alpha} + C\gamma && \text{by (1)}\\
&\leq \rho(\mathcal{W} C^{1-\alpha} + C\gamma) && \text{because } \rho \geq 1
\end{aligned}$$

We showed that if we have a ρ-approximation for the $\min \sum w_i^\alpha$ problem, then we have the same approximation ratio for our problem. □

We now propose the following Algorithm 1 for assigning jobs to processors. We consider that jobs have the same processing requirement on all processors. We denote p_j the processor requirement of a job j. After assigning jobs to processors, we compute the optimal processing time (makespan) that minimizes the energy consumption according to the assignment.

Algorithm 1 Assignment of jobs

```
1: for each processor $i$ do
2:     Set the load $W_i \leftarrow 0$
3: end for
4: for each job $j$ do
5:     Assign job $j$ on the least loaded processor.
6:     Update $W_i \leftarrow W_i + p_j$
7: end for
8: Compute the optimal makespan $C = \sqrt[\alpha]{\frac{(\alpha-1)\sum_{i=1}^{m} W_i^\alpha}{\gamma}}$ according to the obtained assignment
9: Schedule jobs at speed $W_i/C$ on each processor $i$.
```

Corollary 3.3. *The Algorithm 1 is a $\left(2-\Theta\left(\frac{\ln\alpha}{\alpha}\right)\right)^\alpha$-approximation algorithm when jobs have the same processing requirement on each processor.*

Proof. The part from line 1 to 7 of Algorithm 1 is proposed by Avidor et al. [5]. Given this assignment of jobs, we have a workload of W_i on each processor. It has been proved that this assignment is a $\left(2-\Theta\left(\frac{\ln\alpha}{\alpha}\right)\right)$-approximation for the min $\sqrt[\alpha]{\sum W_i^\alpha}$ problem [5].

More formally, let O_i be the workload of processor i in an optimal solution, then we have

$$\sqrt[\alpha]{\sum_{i=1}^{m} O_i^\alpha} \leq \sqrt[\alpha]{\sum_{i=1}^{m} W_i^\alpha} \leq \left(2-\Theta\left(\frac{\ln\alpha}{\alpha}\right)\right)\sqrt[\alpha]{\sum_{i=1}^{m} O_i^\alpha}$$

By raising the above equation to the power α, we get

$$\sum_{i=1}^{m} O_i^\alpha \leq \sum_{i=1}^{m} W_i^\alpha \leq \left(2-\Theta\left(\frac{\ln\alpha}{\alpha}\right)\right)^\alpha \sum_{i=1}^{m} O_i^\alpha$$

By Theorem 3.2, we conclude that we have

$$\begin{aligned}\sum_{i=1}^{m} O_i^\alpha C^{1-\alpha} + C\gamma &\leq \sum_{i=1}^{m} W_i^\alpha C'^{1-\alpha} + C'\gamma \\ &\leq \left(2-\Theta\left(\frac{\ln\alpha}{\alpha}\right)\right)^\alpha \left(\sum_{i=1}^{m} O_i^\alpha C^{1-\alpha} + C\gamma\right)\end{aligned}$$

where C and C' are the processing time that minimize the cost of the respective equation.

Thus, we have a $\left(2-\Theta\left(\frac{\ln\alpha}{\alpha}\right)\right)^\alpha$-approximation. □

We now need to prove that given the assignment, we can compute the minimum cost schedule in polynomial time.

Property 1. *Given the assignment, the total workload is W_i on each processor i, then $C = argmin_x\left(\sum_{i=1}^{m} W_i^\alpha x^{1-\alpha} + x\gamma\right)$ can be computed in $O(m)$ time.*

Proof. We first show that the function is convex, in particular, we only need to show that the second derivative is positive. In order to simplify the notation, we denote $\mathcal{W} = \sum_i W_i^\alpha$.

$$\begin{aligned}\left(\mathcal{W}x^{1-\alpha} + x\gamma\right)'' &= \mathcal{W}\left(\frac{1}{x^{\alpha-1}}\right)'' + (x\gamma)'' \\ &= \mathcal{W}\left(\frac{-(\alpha-1)x^{\alpha-2}}{x^{2(\alpha-1)}}\right)' = -(\alpha-1)\mathcal{W}\left(\frac{1}{x^\alpha}\right)' \\ &= -(\alpha-1)\mathcal{W}\left(\frac{-\alpha x^{\alpha-1}}{x^{2\alpha}}\right) = \alpha(\alpha-1)\mathcal{W}\left(\frac{1}{x^{\alpha+1}}\right) \geq 0\end{aligned}$$

In order to find the global minimum, we need to find the value of x that verifies

$$0 = \left(\mathcal{W}x^{1-\alpha} + x\gamma\right)' = -(\alpha-1)\mathcal{W}\left(\frac{1}{x^\alpha}\right) + \gamma$$

$$\gamma = \frac{(\alpha-1)\mathcal{W}}{x^\alpha} \Leftrightarrow x = \sqrt[\alpha]{\frac{(\alpha-1)\mathcal{W}}{\gamma}}$$

The optimal makespan of the schedule can be computed as above in $O(m)$ time because of the sum. □

Theorem 3.4. *Algorithm 1 has a complexity of $O(m + n\log m)$ where n is the number of jobs and m the number of processors.*

Proof. By maintaining a sorted list of processors according to their load, we can assign each job to the first processor of the list and insert this processor in the list with binary search. Finally, the initialization takes $O(m)$ time, the assignment takes $O(n\log m)$ time, and the optimal makespan can be computed in $O(m)$ time. □

We now propose an Algorithm 2 for the case when jobs have different processing requirement depending on the type of processor to which the job is assigned. We denote $p_{i,j}$ the processor requirement of a job j if it is assigned to processor i. The variables $y_{i,j}$ correspond to the assignment of jobs: it is equal to 1 if job j is assigned to processor i, 0 otherwise. In the following convex program, we consider fractional assignment, i.e., the variables can be any positive real number.

The constraints (4) ensure that each job is assigned at least once. The constraints (5) ensure that the load on processor i is not less than the assigned workload. After solving this convex program, we need to round the fractional assignment into integral assignment, i.e., the variables $y_{i,j}$ will be 0 or 1 after the rounding. This will give us a feasible schedule. Finally, we calculate the processing time that minimizes the global energy consumption according to the assignment.

Corollary 3.5. *The Algorithm 2 is a 2^α-approximation algorithm when jobs have different processing requirement on each processor.*

Proof. As for Corollary 3.3, we refer to the algorithm proposed by Azar et al. [7] for the assignment of jobs. They proposed a 2-approximation algorithm for minimizing the min $\sqrt[\alpha]{\sum W_i^\alpha}$ problem where W_i is the total workload on processor i. Based on their algorithm, we get a 2^α-approximation algorithm for our problem. □

Even though Algorithm 2 has a constant approximation ratio, it implies to solve a convex program which may be hard in practice.

We propose a simple algorithm for the case when jobs have different processing requirement. The idea is to assign jobs to a processor such that the maximum workload among all processors changes the least.

Algorithm 2 Assignment of jobs when jobs have different processing requirement

```
1: Solve the following Convex Program proposed by Azar et al. in [7]:
2:
```

$$\min \sum_{i=1}^{m} W_i^{\alpha} + \sum_{i=1}^{m}\sum_{j=1}^{n} y_{i,j}p_{i,j}$$

$$\text{s.t.} \sum_{i=1}^{m} y_{i,j} \geq 1 \qquad \forall j = 1,\dots,n \tag{4}$$

$$\sum_{j=1}^{n} y_{i,j}p_{i,j} - W_i \leq 0 \qquad \forall i = 1,\dots,m \tag{5}$$

$$y_{i,j} \geq 0 \qquad \forall i = 1,\dots,m, j = 1,\dots,n \tag{6}$$

```
3: Round the fractional assignment into integral assignment proposed by Shmoys and Tardos in [30].
4: for each processor $i$ do
5:     Set the load $W_i \leftarrow \sum_{j=1}^{n} y_{i,j}p_{i,j}$
6: end for
7: Compute the optimal makespan $C = \sqrt[\alpha]{\frac{(\alpha-1)\sum_{i=1}^{m} W_i^{\alpha}}{\gamma}}$ according to the obtained assignment
8: Schedule jobs at speed $W_i/C$ on each processor $i$.
```

Algorithm 3 Unobtrusive Fit: Assignment of jobs on heterogeneous environment

```
1: for each processor $i$ do
2:     Set the load $W_i \leftarrow 0$
3: end for
4: Set $y_{i,j} \leftarrow 0$ for each processor $i$ and each job $j$.
5: Sort jobs in non-ascending order of workload.
6: for each job $j$ do
7:     //Assign job $j$ that modify the least the maximum workload.
8:     $i^\star = \arg\min_i W_i + p_{i,j}$
9:     Update $W_{i^\star} \leftarrow W_{i^\star} + p_{i^\star,j}$
10:    $y_{i^\star,j} \leftarrow 1$
11: end for
12: Compute the optimal makespan $C = \sqrt[\alpha]{\frac{(\alpha-1)\sum_{i=1}^{m} W_i^{\alpha}}{\gamma}}$ according to the obtained assignment
13: Schedule jobs at speed $W_i/C$ on each processor $i$.
```

Note that in Algorithm 3, we sort jobs in non-ascending order of workload. It can be done only when jobs follow the same order on each processor. If we consider the case when jobs have arbitrary processing requirement, then we cannot get an order of the jobs. As we consider the CPU-GPU environment, we assume that two jobs will keep the same ratio of processing requirement on CPU as well as GPU.

The intuition to sort jobs in this way comes from the fact that if we schedule large jobs first, it will keep small jobs later, and therefore the modification on the maximum workload will be less significant.

4 PERFORMANCE EVALUATION

In the previous section, we show that there exists polynomial time approximation algorithm for our problem. However, as the dynamic power consumption follows the cube-root rule most of time, the approximation ratio is $2^{\alpha} = 8$ which is not acceptable in practice. We compare the quality of the solutions of classical algorithms with the optimal solution. Moreover, we supposed previously that we can scale the frequency of the CPU or GPU at any value, which may not be true in practice. Therefore, we consider that each processing unit has a set of available speeds, and we denote it S_i for a processor i. Notice that the cost may be higher in this case since due of the convexity of the power consumption function, the optimal frequency may not be in the set of available speeds. We compare Unobtrusive Fit (Algorithm 3) with previous algorithms by simulations. In the context of energy saving, we also need to consider the running time of these algorithms.

To compute the optimal solution, we formulate our problem with the following Integer Linear Program (Algorithm 4).

We denote $x_{i,j,s}$ as the workload of job j on processor i running at speed s.

Algorithm 4 Integer Linear Program for the energy minimization scheduling problem.

$$\min \sum_{s\in S_i}\sum_{i}\sum_{j} K_i x_{i,j,s}\left(\frac{sV_i}{s_{base}}\right)^2 + C\gamma$$

$$\text{s.t.} \sum_{i} y_{i,j} = 1 \qquad \forall j \tag{7}$$

$$\sum_{s\in S_i} x_{i,j,s} = p_{i,j}y_{i,j} \qquad \forall i,j \tag{8}$$

$$\sum_{s\in S_i}\sum_{j} \frac{x_{i,j,s}}{s} \leq C \qquad \forall i \tag{9}$$

$$y_{i,j} \in \{0,1\} \qquad \forall i,j \tag{10}$$

$$x_{i,j,s} \geq 0 \qquad \forall i,j,s \tag{11}$$

$$C \geq 0 \tag{12}$$

The constraints (7) ensure that each job is assigned exactly to one processor. The constraints (8) ensure that each job is entirely processed on the assigned processor. Finally, the constraints (9) give us the makespan of the schedule. The objective function is to minimize the summation of dynamic energy and static energy. Moreover, we suppose that by scaling the frequency, we can also change the voltage of the processor as consequence in order to minimize the energy consumption. Indeed, if the voltage cannot be scaled, we can see that the dynamic energy is the same whatever the speed it is executed when the dynamic energy is defined by $K_iV_i^2\sum_{s\in S_i} x_{i,j,s}$ for a given job j on processor i. Then we can simply schedule every job at the maximum speed to reduce the static energy.

Note that in the discrete speed model, we can use the integer linear program in Figure 4. Since the assignment of jobs is given, the binary variables $y_{i,j}$ are fixed, therefore it becomes a linear program where we only need to find the speed execution of jobs and it can be solved efficiently with a solver. We compare the optimal solution with the different schedules obtained by the following algorithms:

- L_2-OPT (Algorithm. 5): the optimal solution of L_2-norm;
- L_2-approx (Algorithm 2): a 2^α-approximation algorithm of our problem;
- MS-approx: a 2-approximation algorithm for the makespan minimization problem when jobs have different processing requirement and proposed by Lenstra et al. [21];
- Unobtrusive Fit (Algorithm 3).

We are interested in the approximation ratio of each algorithm by dividing the obtained cost by the optimal cost.

The L_2-OPT algorithm corresponds to the following integer convex program (Algorithm 5).

Algorithm 5 Integer Convex Program based on the L_2-norm

$$\min \quad \sum_i \left(\sum_j y_{i,j} p_{i,j} \right)^2$$

$$\text{s.t.} \sum_i y_{i,j} = 1 \qquad \forall j \qquad (13)$$

$$y_{i,j} \in \{0,1\} \qquad \forall i,j \qquad (14)$$

Constraints (13) ensure to assign each job to processors. As the constraints (14) ensure that we have integer variables, thus the solving may take time, indeed the integer convex programming is harder than the integer linear programming.

In order to support the relevance of the experiments, we consider a situation where we have a set of Intel CPUs Core E5-2650v3 and a set of Nvidia GPUs Tesla K40. The following table summarizes the features of these the computation units. We suppose that jobs

Table 3: Features of the used environment

	CPU	GPU
Model	E5-2650 v3	K40
Processing Power (GFLOPS)	320	1430-1680
TDP (W)	105	235
Base Frequency (GHz)	2.3	0.8
Available Frequency (GHz)	1.6 ; 1.833 ; 2.067	0.5; 0.6; 0.7
Overclock Frequency (GHz)	2.533; 2.767 ; 3	/
Voltage (V)	1.3	12
Capacitance	2.70131E-06	2.03993E-06

are fully supported by both CPU and GPU. If a job is assigned to a CPU, it can be processed by using all its cores, and as well for GPU. The theoretical computational capacity of the E5-2650 v3 [19] is 320 GFLOPS while the K40 [28] is between 1430 and 1680 GFLOPS in double precision. Therefore, the factor of the Processing Power is within $[4.46; 5.25]$.

The experiments are written in Python and we use Gurobi [18] to solve both linear programs and convex programs. For every case, we generated 40 instances of jobs and took the average of the approximation ratio.

We first compare the different algorithms by changing the parameter $\gamma = \{0.001, 0.01, 0.1, 1\}$ and with different size of instance, i.e., the number of jobs is between 100 and 1000. Moreover, we consider that there is 100 CPUs and 100 GPUs. The different results can be found respectively in Figures 1, 2, 3 and 4. Moreover, the comparison of the running time of different algorithms can be found in Table 4.

Table 4: Running time of the different algorithms (in seconds)

jobs	L_2-OPT	L_2-approx	MS-approx	Unobtrusive Fit
100	1724.27	44.02	12.15	0.04
200	383.11	212.63	28.32	0.08
300	653.57	1068.55	45.29	0.13
400	108.72	1856.96	60.99	0.17
500	171.71	2618.88	79.79	0.21
600	289.22	5628.96	101.15	0.25
700	330.83	7510.32	117.80	0.28
800	422.75	9825.46	137.18	0.32
900	541.94	11495.95	158.76	0.36
1000	764.61	18248.75	185.90	0.39

We can observe that L_2-approx has the worst performance compared to the other algorithms especially when there are few jobs. Moreover, its running time is also the worst when there are more than 300 jobs to assign.

MS-approx has also bad performance but is more competitive than L_2-approx and its running time is much better than L_2-approx. Its approximation ratio is around 2 as we can observe in Figure 4. Even though L_2-OPT has better approximation ratio, it is still an integer convex program, thus the running time can not be bounded in the worst case. However, we observe in the simulation that it takes less time than solving the L_2-approx when there are more jobs (less than 300) but its running time is high when there are few jobs. Finally, Unobtrusive Fit has an approximation ratio less than 1.2 for any number of jobs (between 100 and 1000) and it takes less than 1 second for 1000 jobs instance. On the other hand, we observe that the approximation ratio decreases when there are more jobs for all algorithms.

In the following, we investigate the modification of the input data, i.e., the processing requirement of the jobs. Indeed, we only considered processing requirement of jobs to assign them to processors. We now take into account the parameters of the processors, i.e., the capacitance, the voltage and and the maximum speed.

We will consider four combinations of these parameters:

- VCS: Voltage, Capacitance and maximum Speed
- VC: Voltage and Capacitance
- VS: Voltage and maximum Speed
- CS: Capacitance and maximum Speed

The processing requirements are modified as follows: For every parameter Π, we multiply the initial workload by Π_i/Π_1 where Π_1 is the value of the parameter of the first processor. More formally, we have for the different combination the following calculation for each job j and each processor i:

- VCS: $p_{i,j} \leftarrow p_{i,j} \times \frac{V_i}{V_1} \times \frac{C_i}{C_1} \times \frac{\max S_i}{\max S_1}$

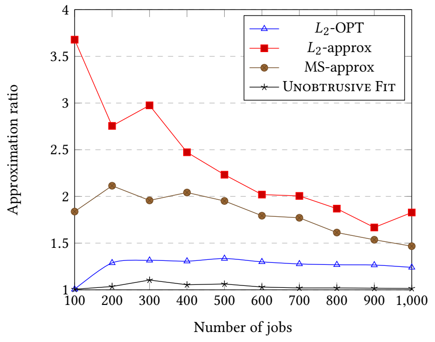

Figure 1: Approximation ratio of different algorithms with 100 CPU and 100 GPU with $\gamma = 0.001$

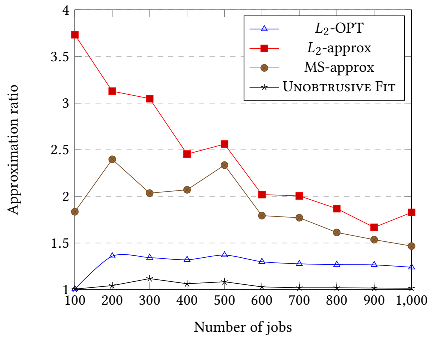

Figure 2: Approximation ratio of different algorithms with 100 CPU and 100 GPU with $\gamma = 0.01$

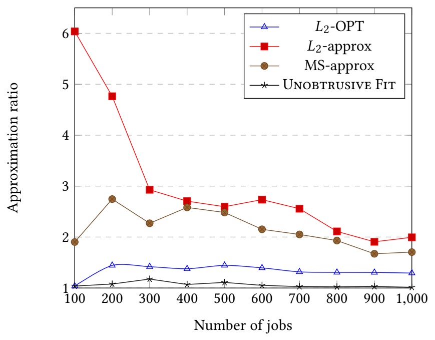

Figure 3: Approximation ratio of different algorithms with 100 CPU and 100 GPU with $\gamma = 0.1$

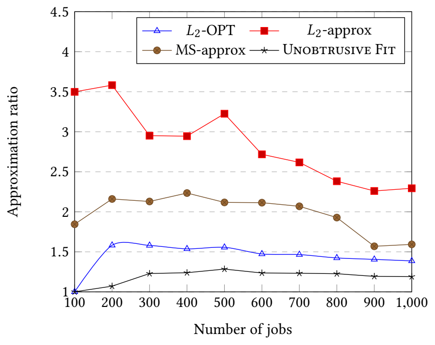

Figure 4: Approximation ratio of different algorithms with 100 CPU and 100 GPU with $\gamma = 1$

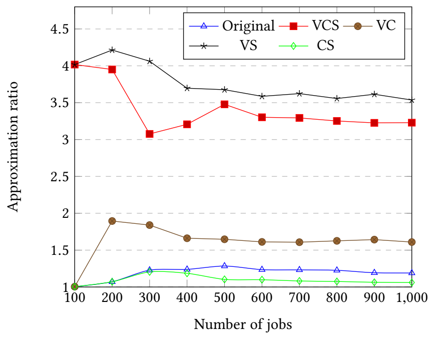

Figure 5: Approximation ratio of Unobtrusive Fit with 100 CPU and 100 GPU and $\gamma = 1$

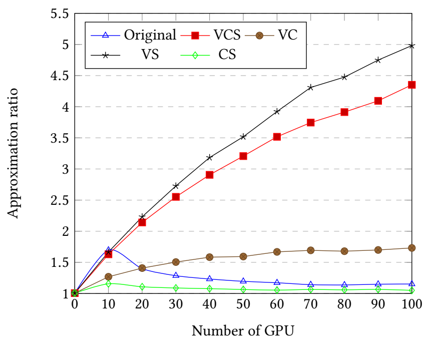

Figure 6: Approximation ratio of Unobtrusive Fit with 500 jobs, 50 CPU and $\gamma = 1$

- VC: $p_{i,j} \leftarrow p_{i,j} \times \frac{V_i}{V_1} \times \frac{C_i}{C_1}$
- VS: $p_{i,j} \leftarrow p_{i,j} \times \frac{V_i}{V_1} \times \frac{\max S_i}{\max S_1}$
- CS: $p_{i,j} \leftarrow p_{i,j} \times \frac{C_i}{C_1} \times \frac{\max S_i}{\max S_1}$

We first compare the different modifications of input on Unobtrusive Fit under the same environment, i.e., there are 100 CPUs and 100 GPUs and we consider that the static energy rate is $\gamma = 1$. After running Unobtrusive Fit on these modifications of input, we get an assignment of jobs on processors and we calculate the energy consumption with initial input. The results can be found in Figure 5.

First, we observe that there are two groups of results. When we modify the processing requirement with VS (Voltage and maximum Speed) or with VCS (Voltage, Capacitance and maximum Speed), the obtained assignments are very bad. They are even worse than the previous algorithm that we compared before. The approximation ratios are between 3 and 4. Secondly, the modification of input according to VC (Voltage and Capacitance) is still worse than without modification and the approximation ratios are less than 2. Finally, the modification of input according to CS (Capacitance and maximum Speed) is better than without modification. Its approximation ratios are also less than 1.2.

We then investigate the performance of Unobtrusive Fit when there are different numbers of GPUs. In particular, we compare the case when there are more or less GPUs than the number of CPUs (see Fig.6). As previously, we observe that when we modify the processing requirement with VS or with VCS rule, the resulting schedules are bad for our problem. The approximation ratio increases when the number of GPUs increases. The approximation ratio is almost 5 when there are 100 GPUs and 50 CPUs. Except when there is less than 20 GPUs in our environment, the resulting schedule without modification in the input is better than the modification of processing requirement according to VC. The modification of input according to CS rule is the best, resulting in an approximation ratio less than 1.16 in all the simulated cases.

5 ONLINE ALGORITHM

We now suppose that jobs arrive over time. Each job j has a release time r_j at which it is revealed. The goal is to minimize the total energy cost, our idea is to exploit the Unobtrusive Fit algorithm whenever jobs arrive in order to assign them to processors. We then compute the optimal schedule according to this assignment of jobs. When another job is released, we update the load of each processor according to what it has been executed until this moment and apply the Unobtrusive Fit algorithm again.

Algorithm 6 Online Unobtrusive Fit: Assignment of jobs on heterogeneous environment

```
1: for each processor $i$ do
2:     Set the load $W_i \leftarrow 0$
3: end for
4: Set $y_{i,j} \leftarrow 0$ for each processor $i$ and each job $j$.
5: while Jobs arrive do
6:     Let $\mathcal{J}$ be the set of jobs that arrive at time $t$.
7:     Use Algorithm 3 (Unobtrusive Fit) with $\mathcal{J}$ and the load
       $W_i$ for each processor $i$
8:     Execute jobs according to the obtained schedule and update
       the remaining load $W_i$ for each processor $i$.
9: end while
```

To compute the optimal solution, we formulate our problem with the following Integer Linear Program (Algorithm. 7). We denote $x_{i,j,s,z}$ as the workload of job j on processor i running at speed s on zone z.

The difference with the previous section is that we have to assign fragment of each job on different zone. We define a zone as an interval of time between two release time. Let $\mathcal{T} = \cup_{j=1}^{n} r_j$ be the set of release time, and denote $t_1 < t_2 < \ldots < t_{|\mathcal{T}|}$ the elements of $\mathcal{T}$. Then the zone z is the following time interval $[t_z, t_{z+1})$ while the length is $\ell(z) = t_{z+1} - t_z$. Note that the last zone is arbitrary large, when $z = |\mathcal{T}|$, we have $t_{z+1} = +\infty$.

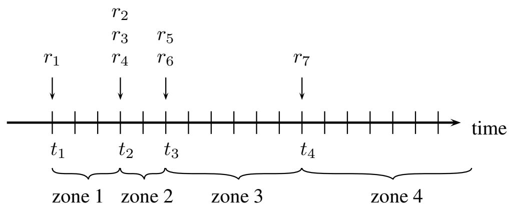

Figure 7: Example of different zones. Job 1 is released at time t_1, jobs 2, 3 and 4 are released at time t_2, jobs 5 and 6 are released at time t_3 and job 7 is released at time t_4.

The constraints (15) ensure that each job is assigned exactly to one processor. The constraints (16) ensure that each job is entirely processed on the assigned processor. The constraint (17) gives us the makespan of the schedule. We can see that this constraint is sufficient for the makespan. Indeed, by considering the last zone, the linear program will assign the maximum amount of workload in the other zones. The constraints (18) ensure that in each zone, the processing time does not exceed the length of the zone. Finally, the constraints (19) ensure that each job cannot be scheduled before it is released. Note that the makespan of the schedule is at least $t_{|\mathcal{T}|}$ since it is a release time. Thus, at least one job needs to be scheduled after this time. The objective is to minimize the total energy.

Algorithm 7 Integer Linear Program for the energy minimization scheduling problem with release time.

$$\min \sum_{t\in\mathcal{T}} \sum_{s\in S_i} \sum_{i} \sum_{j} K_i x_{i,j,s,t} \left(\frac{sV_i}{s_{base}}\right)^2 + C\gamma$$

$$\text{s.t.} \quad \sum_i y_{i,j} = 1 \qquad \forall j \qquad (15)$$

$$\sum_{z=1}^{|\mathcal{T}|} \sum_{s\in S_i} x_{i,j,s,z} = p_{i,j} y_{i,j} \qquad \forall i,j \qquad (16)$$

$$t_{|\mathcal{T}|} + \sum_{s\in S_i} \sum_j \frac{x_{i,j,s,|\mathcal{T}|}}{s} \le C \qquad \forall i \qquad (17)$$

$$\sum_{s\in S_i} \sum_j \frac{x_{i,j,s,z}}{s} \le \ell(z) \qquad \forall i,z \qquad (18)$$

$$x_{i,j,s,z} = 0 \qquad \forall i,j,s,t_z < r_j \qquad (19)$$

$$y_{i,j} \in \{0,1\} \qquad \forall i,j \qquad (20)$$

$$x_{i,j,s,z} \ge 0 \qquad \forall i,j,s,z \qquad (21)$$

$$C \ge 0 \qquad (22)$$

5.1 Experiments setting

We consider there are 50 GPUs and 100 CPUs (core) for different number of jobs : 100, 200, ..., 1000. We compare the offline optimal solution given by the Integer Linear Program (Algorithm 7) with the energy consumption of the online algorithm (Algorithm 6). The performance of the algorithm is given by the ratio of the respective cost. As for the previous part, we take into account the different parameter of each CPU before assigning jobs to processors (VC, VS, CS, VCS).

We first define how to generate the release time of jobs. We consider the following function from [22]:

$$request(x) = 0.0001x^6 + 0.0064x^5 - 0.2251x^4 + 2.9466x^3 - 7.634x^2 - 34.53x + 174.7863$$

The function $request(x)$ represents the request rate according to the time. It starts at midnight and ends at midnight of the next day. We consider that this function is within the interval $x \in [0, 15.2]$ in order to get the same value at $x = 0$ and $x = 15.2$ We mapped this function into the interval $[0, 1440]$ minutes by multiplying the value x by 94.74. The representation of this function can be found in Figure 9.

Then we use the exponential distribution which is given by the following function $F(x) = 1 - e^{-\lambda x}$. The value of λ is the rate of the distribution. When this rate is higher, the probability to get the next event in a short period is higher. Typically, let us suppose we have λ and we pick a random value r in $[0, 1]$ in a uniform distribution, then the next event will happen after x minutes with respect to the equation $1 - e^{-\lambda x} = r$. For example, let $\lambda = 1/10$ and $r = 0.5$, then the next event will happen after 6.93 minutes. This distribution for different value of λ can be found in Figure 8.

We combine this exponential distribution with the request rate function. In the request rate function, the maximum value is 383. Thus, we set $\lambda = \frac{request(\frac{t}{94.74})}{383}$ which is the ratio between the

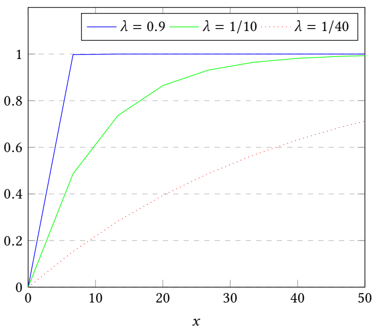

Figure 8: Exponential distribution $F(x) = 1 - e^{-\lambda x}$ for $\lambda = 0.9, 0.1$ and 0.025.

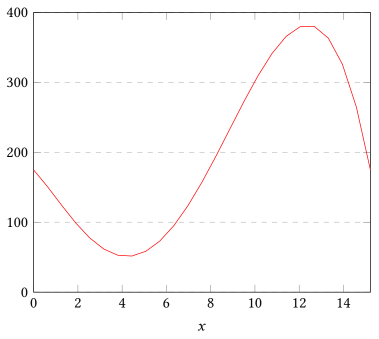

Figure 9: Request rate function with respect to the time x for $x \in [0, 15.2]$ where $request(x) = 0.0001x^6 + 0.0064x^5 - 0.2251x^4 + 2.9466x^3 - 7.634x^2 - 34.53x + 174.7863$

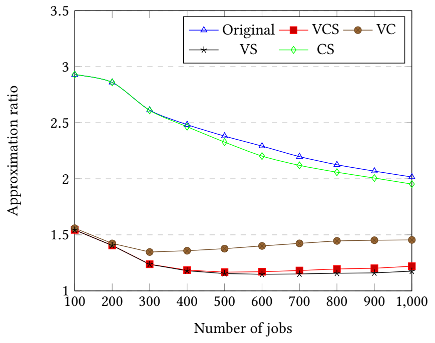

Figure 10: Approximation ratio of Online Unobtrusive Fit with 100 CPU and 50 GPU and $\gamma = 1$

request rate at time t and the maximum value. We then pick a random value in $[0, 1]$ with a uniform distribution. Thus, we aim to solve the following equation: $1 - e^{-\lambda x} = r \Leftrightarrow 1 - r = e^{-\lambda x} \Leftrightarrow \log(1 - r) = -\lambda x \Leftrightarrow x = -\frac{\log(1-r)}{\lambda}$. We then update the time t to $t + x$.

Algorithm 8 Generation of release time of jobs with exponential distribution

```
1: Input: time $t$ in minutes
2: $request(x) = 0.0001x^6 + 0.0064x^5 - 0.2251x^4 + 2.9466x^3 - 7.634x^2 - 34.53x + 174.7863$
3: $\lambda = \frac{request\left(\frac{t \mod 1440}{94.74}\right)}{383}$
4: $r \leftarrow$ random value in $[0, 1]$ with uniform distribution
5: $t' = -\frac{\log(1-r)}{\lambda}$
6: Update $t \leftarrow t + t'$
7: return $t$
```

It can be noticed that if the value of λ is higher, then the next event will happen after a shorter period. As shown in the request rate function, during off-peak hour, the request rate is low, then the next event will happen after a longer period.

According to experiments results, we observe that the results are opposite. Unlike the previous part where there is no release time, the performance of Online Unobtrusive Fit becomes bad when we consider original processing requirement and when we modify the processing requirement according to the Capacitance and the maximum Speed (CS). The approximation ratio is between 2 and 3.

However, the Online Unobtrusive Fit for the other rules (VCS, VC, VS) has a better performance. Their approximation ratios are less than 1.5. Moreover, we observe that for VS and VCS rules, the approximation ratio of Online Unobtrusive Fit is less than 1.2 when there are more than 400 jobs.

6 CONCLUSION

In this paper, we first showed that the problem of assigning jobs on CPU-GPU heterogeneous systems with speed-scaling is $\mathcal{NP}$-hard. We then showed that there exists a constant approximation algorithm when processors can scale into any continuous speed.

Since the approximation is not small, we propose a heuristic algorithm and compare with some classical scheduling algorithms in terms of performance and execution time. However, in practice, processors are given a set of speeds that can be scaled. Our simulation results show that the proposed heuristic algorithm (Unobtrusive Fit) is very effective and can achieve near-optimal performance in all simulated cases. It consumes only 20% more energy compared to the optimal assignment. Especially when we modify the processing requirement of jobs according to the Capacitance and the maximum Speed of the processor, the proposed heuristic returns a schedule with a cost at most 16% more than the optimal schedule.

In a second part, we consider the online setting. Jobs arrive over time and they are discovered only once we reach this time. We adapted the proposed heuristic for this setting. Moreover, we generate the set of release time according to a request rate function and we apply the Unobtrusive Fit algorithm each time a job is discovered. Finally, we find that, unlike the previous part where there is no release time, and when we modify the processing requirement according to the Capacitance and the maximum Speed or when we do not modify, the performance of Online Unobtrusive Fit becomes bad, while the other rules (VCS, VC, VS) consume on average 50% more energy compared to the optimal solution.

Depending on whether there is release time or not, we may use different rule of modification of processing requirement in order to get a better assignment of jobs on heterogeneous environment.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This work is partially supported by research grant HKBU FRG2/14-15/059 and Shenzhen Basic Research Grant SCI-2015-SZTIC-002.

REFERENCES

[1] Susanne Albers. 2010. Energy-efficient algorithms. *Commun. ACM* 53, 5 (2010), 86–96.

[2] Susanne Albers. 2011. Algorithms for Dynamic Speed Scaling. In *STACS (LIPIcs)*, Vol. 9. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 1–11.

[3] Susanne Albers, Fabian Müller, and Swen Schmelzer. 2014. Speed Scaling on Parallel Processors. *Algorithmica* 68, 2 (2014), 404–425.

[4] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. 1997. Approximation Schemes for Scheduling. In *SODA*. ACM/SIAM, 493–500.

[5] Adi Avidor, Yossi Azar, and Jirí Sgall. 2001. Ancient and New Algorithms for Load Balancing in the l_p Norm. *Algorithmica* 29, 3 (2001), 422–441.

[6] Baruch Awerbuch, Yossi Azar, Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. 1995. Load Balancing in the L_p Norm. In *FOCS*. IEEE Computer Society, 383–391.

[7] Yossi Azar and Amir Epstein. 2005. Convex programming for scheduling unrelated parallel machines. In *STOC*. ACM, 331–337.

[8] Evripidis Bampis. 2016. Algorithmic Issues in Energy-Efficient Computation. In *DOOR (Lecture Notes in Computer Science)*, Vol. 9869. Springer, 3–14.

[9] Evripidis Bampis, Alexander V. Kononov, Dimitrios Letsios, Giorgio Lucarelli, and Maxim Sviridenko. 2013. Energy Efficient Scheduling and Routing via Randomized Rounding. In *FSTTCS (LIPIcs)*, Vol. 24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 449–460.

[10] Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli. 2014. A note on multiprocessor speed scaling with precedence constraints. In *SPAA*. ACM, 138–142.

[11] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. 2016. Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods. *ACM Comput. Surv.* 49, 3 (2016), 41:1–41:27.

[12] Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. 2006. Leakage-Aware Energy-Efficient Scheduling of Real-Time Tasks in Multiprocessor Systems. In *IEEE Real Time Technology and Applications Symposium*. IEEE Computer Society, 408–417.

[13] Lin Chen, Deshi Ye, and Guochuan Zhang. 2014. Online Scheduling of mixed CPU-GPU jobs. *Int. J. Found. Comput. Sci.* 25, 6 (2014), 745–762.

[14] Vincent Cohen-Addad, Zhentao Li, Claire Mathieu, and Ioannis Milis. 2014. Energy-Efficient Algorithms for Non-preemptive Speed-Scaling. In *WAOA (Lecture Notes in Computer Science)*, Vol. 8952. Springer, 107–118.

[15] Vincent W. Freeh, Nandini Kappiah, David K. Lowenthal, and Tyler K. Bletsch. 2008. Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. *J. Parallel Distrib. Comput.* 68, 9 (2008), 1175–1185.

[16] Rong Ge, Xizhou Feng, and Kirk W. Cameron. 2005. Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters. In *SC*. IEEE Computer Society, 34.

[17] Gero Greiner, Tim Nonner, and Alexander Souza. 2014. The Bell Is Ringing in Speed-Scaled Multiprocessor Scheduling. *Theory Comput. Syst.* 54, 1 (2014), 24–44.

[18] Inc. Gurobi Optimization. 2015. Gurobi Optimizer Reference Manual. (2015). http://www.gurobi.com

[19] I HPC Systems. 2015. Comparison of Intel processors. (2015). http://www.hpc.co.jp/compare_intel_processors.html

[20] Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. 2007. Algorithms for power savings. *ACM Trans. Algorithms* 3, 4 (2007).

[21] Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. 1990. Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Math. Program.* 46 (1990), 259–271.

[22] Wenjie Liu, Zhihui Du, Yu Xiao, David A. Bader, and Chen Xu. 2011. A Waterfall Model to Achieve Energy Efficient Tasks Mapping for Large Scale GPU Clusters. In *IPDPS Workshops*. IEEE, 82–92.

[23] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In *ICPP*. IEEE Computer Society, 48–57.

[24] Xinxin Mei, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. 2017. Energy Efficient Real-time Task Scheduling on CPU-GPU Hybrid Clusters. In *INFOCOM*. IEEE.

[25] Xinxin Mei, Qiang Wang, and Xiaowen Chu. 2016. A Survey and Measurement Study of GPU DVFS on Energy Conservation. *CoRR* abs/1610.01784 (2016).

[26] Xinxin Mei, Ling Sing Yung, Kaiyong Zhao, and Xiaowen Chu. 2013. A measurement study of GPU DVFS on energy conservation. In *HotPower@SOSP*. ACM, 10:1–10:5.

[27] Sparsh Mittal and Jeffrey S. Vetter. 2014. A Survey of Methods for Analyzing and Improving GPU Energy Efficiency. *ACM Comput. Surv.* 47, 2 (2014), 19:1–19:23.

[28] NVIDIA. 2013. Tesla K40 GPU Acclerator, Board Specification. (2013). http://www.nvidia.com/content/PDF/kepler/Tesla-K40-PCIe-Passive-Board-Spec-BD-06902-001_v05.pdf

[29] Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. 2008. Speed Scaling of Tasks with Precedence Constraints. *Theory Comput. Syst.* 43, 1 (2008), 67–80.

[30] David B. Shmoys and Éva Tardos. 1993. An approximation algorithm for the generalized assignment problem. *Math. Program.* 62 (1993), 461–474.

[31] F. Frances Yao, Alan J. Demers, and Scott Shenker. 1995. A Scheduling Model for Reduced CPU Energy. In *FOCS*. IEEE Computer Society, 374–382.