Bi-Directional Timing-Power Optimisation on Heterogeneous Multi-Core Architectures

Jing Huang, Renfa Li, Yehua Wei, Jiyao An, Wanli Chang

Abstract—Optimisation of timing performance and power consumption on heterogeneous multi-core architectures is gaining increasing attention. Systems and devices may have varying demands on timing and power, which motivates more flexible optimisation. Along this line, we consider a heterogeneous computing architecture with multiple cores, where each core runs a mixed stream of general and dedicated tasks with a certain scheduling strategy. Employing the queuing model, we first propose a load balancing algorithm, which minimises the average response time of the general tasks whilst guaranteeing the timing requirements of the dedicated tasks. Built upon the above, we propose a bi-directional optimisation algorithm that is able to improve the timing performance under the constraint of power consumption, and reduces the power consumption for the given timing requirement. Extensive numerical experiments illustrate the significance of the proposed algorithms. Implementation on a real platform validates the consistency between the theoretical analysis and the practical results.

Index Terms—Heterogeneous multi-core architectures, timing performance, power consumption, load balancing, bi-directional optimisation, queuing model.

1 Introduction

Majority of the contemporary computing systems employ heterogeneous multi-core architectures, composed of different cores or identical cores with different settings. There are varying demands on timing performance and power consumption, which are often the main design objectives to optimise. For example, a smart phone is expected to push harder towards low-power operation when the battery is close to depletion. An automobile needs to guarantee the timing of certain tasks and minimise the response time of other ones, the extent of the latter depending on the energy storage status. This motivates more flexible optimisation, where there have been very few studies.

In this paper, we consider a heterogeneous multi-core computing architecture. Each core runs a mixed stream of general and dedicated tasks according to its own scheduling strategy. The dedicated tasks are often critical and bounded to one core for affinity. The allocation of the general tasks to cores is a design decision to be explored. Employing the queuing model, our main contributions are as follows:

- We propose a load balancing algorithm under the assumption that the frequencies (speeds) of the cores are known, which minimises the average response time (waiting time plus execution time) of the general tasks whilst ensuring that each type of the dedicated tasks meets its timing requirement.
- Built upon the above, we propose a bi-directional optimisation algorithm to determine the frequencies of the cores. It is able to in one way improve the timing performance under the constraint of power consumption, and in the other way reduces the power consumption for the given timing requirement.

Evaluation is conducted on extensive numerical examples and a real platform.

The rest of this paper is organised as follows. Section 2 reviews the related research. Section 3 describes the models and presents the problem statement. Section 4 introduces the load balancing algorithm (Section 4.1) and the bi-directional optimisation algorithm (Section 4.2). Evaluation is reported in Section 5 and 6. Section 7 makes a conclusion.

2 Related Work

In this section, we review the related research in three areas of performance optimization, power optimization techniques, and joint optimization of performance and power.

Performance Optimization. There is a significantly large body of literature on performance optimization in distributed systems, in which [1] is an excellent collection. Performance can be optimized by load balancing [2]. Many related approaches have been proposed, such as game theory [3], [4] and queuing theory [5]. Here, we focus on queuing-based approaches. When researchers employ queuing theory, the considered system usually has two characteristics, i.e., (1) the number of tasks is large; (2) the arrival times and sizes of tasks are random. There are many studies on queuing-based load balancing, which model the system as a different queueing model, such as M/M/1 [6], M/G/1 [7], M/M/K [2], [8], [9], G/G/K [10], etc. The main target of these studies is minimizing the average response

- *Jing Huang, Renfa Li, and Jiyao An are with the College of Computer Science and Electronic Engineering, and the Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan University, China, 410081. (email: jingh@hnu.edu.cn, lirenfa@hnu.edu.cn, anbobcn@aliyun.com)*
- *Yehua Wei is with the College of Information Science and Engineering, Hunan Normal University, China, 410082. (email: yehuahn@hunnu.edu.cn)*
- *Wanli Chang is with the Department of Computer Science, University of York, UK, YO10 5GH. (email: wanli.chang@york.ac.uk)*

Manuscript received 11 September 2019; revised 07 April 2020, 14 July 2020; accepted 4 August 2020. W. Chang is the corresponding author.

time of tasks. Some studies focus on one type of task [2], [9], while others focus on multiple types of tasks [8], [11], [12], [13]. When a system performs multiple types of tasks, it needs to set priority for each type of tasks so as to distinct the importance from different types of tasks. Such a consideration has been investigated in [7], [8], [11], [12]. However, these studies have more or less not considered the following factors, i.e., system heterogeneity, task priority diversity, and importance of dedicated tasks. For examples, in [7], [8] the system heterogeneity was not fully considered; in [11] the task priority diversity and the importance of dedicated tasks were not mentioned; in [12] the importance of dedicated tasks was ignored. It is clear that although there are many excellent studies on performance optimization, there are some rooms for in-depth research.

Power Optimization Techniques. Power optimization, which is also called energy efficiency in some studies, is a primary concern in the fields of cloud computing and embedded computing [14], and the optimization techniques used in these fields could learn from each other. The goal of power optimization is to reasonably allocate power to a system in a manner that reduces unnecessary energy loss while ensuring performance requirements [15]. Energy-saving techniques of hardware can be applied at different design levels: the *complementary metal-oxide semiconductor (CMOS) level*, *processor-level*, and *interconnection network-level* [14]. For instance, [16] [17] considered the energy efficiency problem at the CMOS level, while [18] and [19] did so at the processor-level and interconnection network-level, respectively. At present, the hardware-assisted software approach is a major strategy for achieving energy efficiency, where dynamic voltage and frequency scaling (DVFS) is one of the most utilized technologies. DVFS has been widely applied both in cloud systems in which performance is the major goal [20], [21], [22], [23] and in embedded systems with strict power constraints [14], [24], [25], [26]. DVFS is often implemented at operating system level and obtains the energy-delay trade-off by adjusting the processor frequency and supply voltage [27] [28].

Joint Optimization of Performance and Power. As energy efficiency has become a primary concern, joint optimization of performance and power is inevitable. Many researchers have focused on this topic from a variety of perspectives. Some researchers study the problem according to the number of executable instructions per watt (IPW). For instance, [29] proposed a runtime optimization approach for dynamic workload to achieve maximized power normalized performance; [30] proposed an adaptive energy minimization approach using regression learning. Fundamental to the approaches of [29] and [30] is that the IPW will be maximum when a system delivers the most energy efficiency, and the advantage of them is that the most energy-efficient frequencies can be found quickly for familiar applications. Some researchers use queueing theory to investigate this topic, and recent studies include [2], [9], [31], [32], and so on. In these studies, the problem of concern usually was treated as a multi-variable and multi-constraint optimization problem. In [2], the application scenario is to assign a stream of tasks onto a multi-processor system, where each processor was modeled as an M/M/K queuing system; and the purpose is to obtain the optimal load balancing and power allocation. Similar problem and queuing model were studied in [32] and [9]. Although, the above studies all considered the performance and power optimization, none of them took into account the mixed-priority. In [31], both the joint optimization and mixed-priority were investigated. However the performance constraints of dedicated tasks were not included. Moreover, as mentioned in [33], both maximum performance and minimizing energy use are of great importance, while only the minimizing of energy use was discussed in [31].

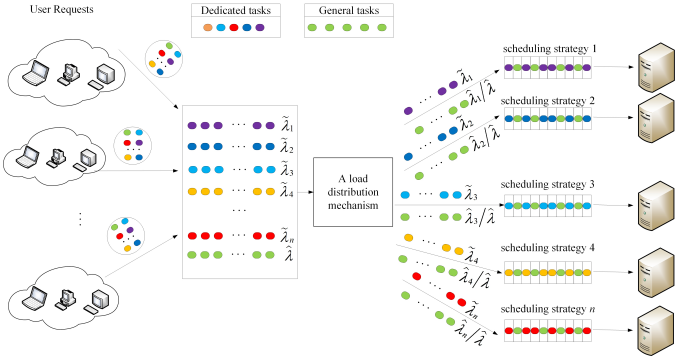

Fig. 1. System structure

3 System Model and Problem Statement

The concerned system model is addressed in this section, which contains three system sub-models, that is, queuing model, energy model, and scheduling model, and then the present optimization problem is stated as two sub-problems.

3.1 Queuing Model

Since the arrival time and size of tasks are random, we use queueing theory to model the system, similar to the studies [2], [6], [7], [34]. The details of our model are described below.

The system is assumed to be consisting of n heterogeneous processors, each processor accepts an independent Poisson stream of dedicated tasks with average arrival rate $\widetilde{\lambda}_i$ $(1 \leq i \leq n)$, i.e., the inter-arrival times are independent and identically distributed (i.i.d.) exponential random variables with mean $1/\widetilde{\lambda}_i$. There is a Poisson stream of general tasks with arrival rate $\widehat{\lambda}$ that requires being split into n substreams, $\widehat{\lambda}_1, \widehat{\lambda}_2, \ldots, \widehat{\lambda}_n$, assigned onto processors $1, 2, \ldots, n$. Consequently, a processor deals with a combined Poisson stream of dedicated and general tasks with arrival rate $\widetilde{\lambda}_i + \widehat{\lambda}_i$. The average task size (the number of instructions) of dedicated tasks and that of general tasks are exponential random variables with mean $\widetilde{r}_i$ and $\widehat{r}$, respectively, which means that the service time of tasks also follows the exponential distribution. According to the above description, each processor is modeled as an $M/M/1$ queuing system. Mathematical notations that will be used in this paper are presented in Table 1, and the system structure is shown in Fig. 1.

TABLE 1
Mathematical notations in this paper

Symbol	Definition
	Inputs
v_i	Processor $i, 1 \leq i \leq n$.
$s_{i1}, s_{i2}, ..., s_{iM_i}$	Adjustable speeds of v_i.
P_i^*	Static power of v_i.
α_i	Power consumption exponent of v_i.
$\overline{T}$	Given performance.
$\overline{P}$	Given power.
$\overline{\widetilde{T}_i}$	Performance constraint on dedicated tasks on v_i.
$\widetilde{\lambda}_i$	Arrival rate of dedicated tasks to v_i.
$\widehat{r}$	Average task size of general tasks.
$\widetilde{r}_i$	Average task size of dedicated tasks on v_i.
	Variables
s_i	s_{ig_i} Speed of processor i.
$\widehat{\lambda}_i$	Arrival rate of general tasks to v_i.
	Intermediate variables
$\widetilde{T}_i$	Average response time of dedicated tasks on v_i.
$\widehat{T}_i$	Average response time of general tasks on v_i.
$\widehat{T}$	$= \frac{\widehat{\lambda}_1}{\widehat{\lambda}}\widehat{T}_1 + \frac{\widehat{\lambda}_2}{\widehat{\lambda}}\widehat{T}_2 + \cdots + \frac{\widehat{\lambda}_n}{\widehat{\lambda}}\widehat{T}_n$.
P	System power.
$\widetilde{x}_i = \widetilde{r}_i/s_i$	Average execution time of dedicated tasks on v_i.
$\widetilde{u}_i = 1/\widetilde{x}_i$	Average service rate of dedicated tasks on v_i.
$\widetilde{m}_i = 2/\widetilde{u}_i^2$	Second moment of dedicated tasks on v_i.
$\widehat{x}_i = \widehat{r}/s_i$	Average execution time of general tasks on v_i.
$\widehat{u}_i = 1/\widehat{x}_i$	Average service rate of general tasks on v_i.
$\widehat{m}_i = 2/\widehat{u}_i^2$	Second moment of general tasks on v_i.
λ_i	$= \widetilde{\lambda}_i + \widehat{\lambda}_i$.
$\widehat{\lambda}$	$= \widehat{\lambda}_1 + \widehat{\lambda}_2 + \cdots + \widehat{\lambda}_n$.
m_i	$= \left(\widetilde{\lambda}_i \big/ \lambda_i\right) \cdot \widetilde{m}_i + \left(\widehat{\lambda}_i \big/ \lambda_i\right) \cdot \widehat{m}_i$.
$\widehat{\rho}_i$	$= \widehat{\lambda}_i \cdot \widehat{x}_i = \widehat{\lambda}_i \widehat{r} \big/ s_i$.
$\widetilde{\rho}_i$	$= \widetilde{\lambda}_i \cdot \widetilde{x}_i = \widetilde{\lambda}_i \widetilde{r}_i \big/ s_i$.
$\rho_i = \widehat{\rho}_i + \widetilde{\rho}_i$	Average percentage of time that v_i is busy.

3.2 Energy Model

The operating frequency f of a processor matches a speed s that can be measured by the number of executed instructions per second. Therefore, the set of available frequencies $\{f_{i1}, f_{i2}, ..., f_{iM_i}\}$ of processor i can be mapped to a set of speeds $\{s_{i1}, s_{i2}, ..., s_{iM_i}\}$. For convenience, we specify $s_{i1} < s_{i2} < \cdots < s_{iM_i}$.

The power dissipation of a processor mainly consists dynamic and static consumption, where dynamic power consumption is the dominant component. The dynamic power consumption can be expressed by $P = kCV^2f$, where k is an activity factor, C the loading capacitance, V the supply voltage, and f the clock frequency. Given that $s \propto f$ and $f \propto V$, then $P \propto s^\alpha$, where α is approximately 3 [35]. For ease of discussion, we model the dynamic power of processor i with speed s_i as $\beta_i {s_i}^{\alpha_i}$, where β_i is a coefficient factor, and the static power as P_i^*. Therefore, the power model of a processor with speed s_i can be formulated as $\beta_i {s_i}^{a_i} + {P_i}^*$. In this paper, two types of energy models are considered, namely, frequency-conversion model and frequency-constant model.

In the *frequency-conversion* model, processor speed will be set to 0 if there is no task to perform, i.e., the processor only consumes basic power ${P_i}^*$ when it is idle. In one unit time, if ρ_i represents the average percentage of time that a processor is busy, then $(1 - \rho_i)$ represents the average percentage of time that a processor is idle. Consequently, the average power consumption of a processor in one unit time can be modeled as

$$\begin{aligned} P_i &= \rho_i(\beta_i s_i^{\alpha_i} + {P_i}^*) + (1 - \rho_i)P_i^* \\ &= \left(\widehat{\lambda}_i\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i\right)\beta_i {s_i}^{\alpha_i - 1} + P_i^*. \end{aligned} \tag{1}$$

In the *frequency-constant* model, processor speed is a constant, whether there are tasks being executing on the processor. Therefore, the average power consumption of a processor in one unit time can be modeled as

$$\begin{aligned} P_i &= \rho_i(\beta_i s_i^{\alpha_i} + {P_i}^*) + (1 - \rho_i)(\beta_i s_i^{\alpha_i} + {P_i}^*) \\ &= \beta_i {s_i}^{a_i} + {P_i}^*. \end{aligned} \tag{2}$$

The above two models have also been used in some studies such as [2], [9], [36]. The *frequency-conversion* model might save more energy than *frequency-constant* model in theory. However, the later is more popular than the former in practical, because frequent changes in voltage and clock frequency will introduce additional consumption and transient errors [37]. Generally, *frequency-conversion* and *frequency-constant* models respectively suit for systems with long and short task arrival intervals.

3.3 Scheduling Model

Scheduling strategies directly affect the response time of tasks, where the response time of a task is the sum of waiting time and execution time. In this paper, three priority strategies are taken into account, the details of which are as follows:

Scheduling strategy 1 (PS 1): all general tasks and dedicated tasks on this processor are scheduled on a FCFS basis, without priority. We identify this discipline as "dedicated tasks without priority." If a processor's scheduling strategy is set to this mode, the average response time of dedicated tasks assigned on this processor, $\widetilde{T}_i$ [38, p. 700], is

$$\begin{aligned} \widetilde{T}_i &= \widetilde{x}_i + \frac{\lambda_i m_i}{2\left(1 - \rho_i\right)} \\ &= \frac{\widetilde{r}_i}{s_i} + \frac{\widehat{\lambda}_i\widehat{r}^2 + \widetilde{\lambda}_i\widetilde{r}_i^2}{s_i\left(s_i - \widehat{\lambda}_i\widehat{r} - \widetilde{\lambda}_i\widetilde{r}_i\right)}, \end{aligned} \tag{3}$$

and the average response time of general tasks, $\widehat{T}_i$ [38, p. 700], is

$$\begin{aligned} \widehat{T}_i &= \widehat{x}_i + \frac{\lambda_i m_i}{2\left(1 - \rho_i\right)} \\ &= \frac{\widehat{r}}{s_i} + \frac{\widehat{\lambda}_i\widehat{r}^2 + \widetilde{\lambda}_i\widetilde{r}_i^2}{s_i\left(s_i - \widehat{\lambda}_i\widehat{r} - \widetilde{\lambda}_i\widetilde{r}_i\right)}. \end{aligned} \tag{4}$$

Scheduling strategy 2 (PS 2): dedicated tasks are always scheduled before general tasks, and all tasks are executed without interruption. We identify this discipline as "prioritized dedicated tasks without pre-emption." If a processor's scheduling strategy is set to this mode, the average response time of dedicated tasks, $\widetilde{T}_i$ [38, p. 702], is

$$\begin{aligned} \widetilde{T}_i &= \widetilde{x}_i + \frac{\lambda_i m_i}{2\left(1 - \widetilde{\rho}_i\right)} \\ &= \frac{\widetilde{r}_i}{s_i} + \frac{\left(\widetilde{\lambda}_i\widetilde{r}_i^2 + \widehat{\lambda}_i\widehat{r}^2\right)}{s_i\left(s_i - \widetilde{\lambda}_i\widetilde{r}_i\right)}, \end{aligned} \tag{5}$$

and the average response time of general tasks, $\widehat{T}_i$ [38, p. 702], is

$$\begin{aligned}\widehat{T}_i &= \widehat{x}_i + \frac{\lambda_i m_i}{2\left(1-\widetilde{\rho}_i\right)\left(1-\rho_i\right)} \\ &= \frac{\widehat{r}}{s_i} + \frac{\widehat{\lambda}_i\widehat{r}^2 + \widetilde{\lambda}_i\widetilde{r}_i^2}{\left(s_i - \widetilde{\lambda}_i\widetilde{r}_i\right)\left(s_i - \widetilde{\lambda}_i\widetilde{r}_i - \widehat{\lambda}_i\widehat{r}\right)}.\end{aligned} \tag{6}$$

Scheduling strategy 3 (PS 3): dedicated tasks are always scheduled before general tasks on this processor, with pre-emption. We term this discipline as "prioritized dedicated tasks with pre-emption." If a processor takes this mode as its scheduling strategy, the corresponding average response time of dedicated tasks, $\widetilde{T}_i$ [38, p. 704], is

$$\begin{aligned}\widetilde{T}_i &= \widetilde{x}_i + \frac{\widetilde{\lambda}_i\widetilde{m}_i}{2\left(1-\widetilde{\rho}_i\right)} \\ &= \frac{\widetilde{r}_i}{s_i} + \frac{\widetilde{\lambda}_i\widetilde{r}_i^2}{s_i\left(s_i - \widetilde{\lambda}_i\widetilde{r}_i\right)},\end{aligned} \tag{7}$$

and the average response time of general tasks, $\widehat{T}_i$ [38, p. 704], is

$$\begin{aligned}\widehat{T}_i &= \frac{1}{\left(1-\widetilde{\rho}_i\right)}\left(\widehat{x}_i + \frac{\lambda_i m_i}{2\left(1-\rho_i\right)}\right) \\ &= \frac{1}{s_i - \widetilde{\lambda}_i\widetilde{r}_i}\left(\widehat{r} + \frac{\widehat{\lambda}_i\widehat{r}^2 + \widetilde{\lambda}_i\widetilde{r}_i^2}{s_i - \widehat{\lambda}_i\widehat{r} - \widetilde{\lambda}_i\widetilde{r}_i}\right).\end{aligned} \tag{8}$$

The scheduling strategy on each processor can be any of the above three (*PS 1*, *PS 2*, *PS 3*). For the convenience, we mark processor i as v_i ($1 \leq i \leq n$), and classify all processors into three groups G1, G2, and G3, where G1, G2, and G3 include all of the processors whose scheduling strategy is *PS 1*, *PS 2*, and *PS 3*, respectively. The formulas Eqs. (3-8) are variants of Pollaczek-Khinchine formula that can be find in any reference book of queueing theory.

3.4 Problem Statement

***Conditions*:** Given n processors, the available speeds of each processor $\{s_{11}, s_{12}, ..., s_{1M_1}\}, \cdots, \{s_{n1}, s_{n2}, ..., s_{nM_n}\}$, the average arrival rates $\widetilde{\lambda}_1, \widetilde{\lambda}_2, ..., \widetilde{\lambda}_n$ and task sizes $\widetilde{r}_1, \widetilde{r}_2, ..., \widetilde{r}_n$ and performance requirements $\overline{\widetilde{T}_1}, \overline{\widetilde{T}_2}, ..., \overline{\widetilde{T}_n}$ of dedicated tasks, the scheduling strategy for each processor, the average arrival rate $\widehat{\lambda}$ and task size $\widehat{r}$ of general tasks.

***Problem 1*:** *Optimizing performance under given power.* The available system power is assumed to be $\overline{P}$. Under the limited power $\overline{P}$ and above conditions, find the task arrival rates $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n$ and processors speeds $s_1, s_2, ..., s_n$, such that the average response time of general tasks on the system is minimized while satisfying that the average response times of dedicated tasks $\widetilde{T}_1, \widetilde{T}_2, ..., \widetilde{T}_n$ do not exceed $\overline{\widetilde{T}_1}, \overline{\widetilde{T}_2}, ..., \overline{\widetilde{T}_n}$. The mathematical formulation is,

$$minimize\ \widehat{T}(\widehat{\lambda}_1, \ldots, \widehat{\lambda}_n, s_1, \ldots, s_n) = \frac{\widehat{\lambda}_1}{\widehat{\lambda}}\widehat{T}_1 + \frac{\widehat{\lambda}_2}{\widehat{\lambda}}\widehat{T}_2 + \cdots + \frac{\widehat{\lambda}_n}{\widehat{\lambda}}\widehat{T}_n, \tag{9}$$

subject to

$$\begin{cases} \widehat{\lambda}_1 + \widehat{\lambda}_2 + \cdots + \widehat{\lambda}_n = \widehat{\lambda}; \\ \widetilde{T}_i \leq \overline{\widetilde{T}_i}, 1 \leq i \leq n; \\ P_1 + P_2 + \cdots + P_n \leq \overline{P}. \end{cases}$$

***Problem 2*:** *Optimizing power under given performance.* Given the above conditions and an acceptable average response time $\overline{\widehat{T}}$ of general tasks, find the task arrival rates $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n$ and processors speeds $s_1, s_2, ..., s_n$, such that the system power is minimized while satisfying that the average response times of dedicated tasks $\widetilde{T}_1, \widetilde{T}_2, ..., \widetilde{T}_n$ do not exceed $\overline{\widetilde{T}_1}, \overline{\widetilde{T}_2}, ..., \overline{\widetilde{T}_n}$. The mathematical formulation is,

$$minimize\ P(\widehat{\lambda}_1, \ldots, \widehat{\lambda}_n, s_1, \ldots, s_n) = P_1 + P_2 + \cdots + P_n, \tag{10}$$

subject to

$$\begin{cases} \widehat{\lambda}_1 + \widehat{\lambda}_2 + \cdots + \widehat{\lambda}_n = \widehat{\lambda}; \\ \widetilde{T}_i \leq \overline{\widetilde{T}_i}, 1 \leq i \leq n; \\ \frac{1}{\widehat{\lambda}}\left(\widehat{\lambda}_1\widehat{T}_1 + \widehat{\lambda}_2\widehat{T}_2 + \cdots + \widehat{\lambda}_n\widehat{T}_n\right) \leq \overline{\widehat{T}}. \end{cases}$$

Problem 1 focuses on performance optimisation, while Problem 2 pays attention to power optimisation. Both they are of importance, but the emphasis of optimisation should shift between them as the system state changes. To our knowledge, there is no existing solution that can completely solve the Problems 1 and 2, and takes into account the transfer of optimisation objectives.

4 Performance and Power Optimization

We aim to design an approach that can not only solve Problem 1 but also solve Problem 2. Since the two problems are symmetrical, we take Problem 1 to describe the corresponding mathematical derivations and algorithms.

Our approach consists of two parts. In the first part, the performance is optimized by load balancing under the assumption that speeds $s_1, s_2, ..., s_n$ have been given. The details will be elaborated in Section 4.1. In the second part, the power is optimized on the basis of the first part, which will be briefly introduced in Section 4.2.

4.1 Performance Optimization

Due to the exclusion of power optimization, the problem of performance optimization is to minimize:

$$\widehat{T}(\widehat{\lambda}_1, \widehat{\lambda}_2, \ldots, \widehat{\lambda}_n) = \frac{\widehat{\lambda}_1}{\widehat{\lambda}}\widehat{T}_1 + \frac{\widehat{\lambda}_2}{\widehat{\lambda}}\widehat{T}_2 + \cdots + \frac{\widehat{\lambda}_n}{\widehat{\lambda}}\widehat{T}_n, \tag{11}$$

subject to

$$\begin{cases} \widehat{\lambda}_1 + \widehat{\lambda}_2 + \cdots + \widehat{\lambda}_n = \widehat{\lambda}; \\ \widetilde{T}_i \leq \overline{\widetilde{T}_i}, 1 \leq i \leq n. \end{cases}$$

Eq. (11) has output parameters $\widehat{\lambda}_i$, and input parameters $s_i, \overline{\widetilde{T}_i}$, and $\widehat{\lambda}$, ($1 \leq i \leq n$). Minimizing Eq. (11) is a multi-variable and multi-constraint optimization problem, and it could be solved by using KKT method [39]. The constraints $\widetilde{T}_i \leq \overline{\widetilde{T}_i}$ corresponding to Eqs. (3), (5) and (7) are nonlinear equations of $\widehat{\lambda}_i$, which is not convenient to solve the problem. We will transform them to another form.

If the scheduling strategy for processor i is *PS 1*, we have

$$\widetilde{T}_i = \frac{\widetilde{r}_i}{s_i} + \frac{\widehat{\lambda}_i\widehat{r}^2 + \widetilde{\lambda}_i\widetilde{r}^2}{s_i\left(s_i - \widehat{\lambda}_i\widehat{r} - \widetilde{\lambda}_i\widetilde{r}\right)} \leq \overline{\widetilde{T}_i}. \tag{12}$$

Based on Eq. (12), we can obtain

$$\widehat{\lambda}_i \leq \frac{s_i\left(\overline{\widetilde{T}_i}\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)-\widetilde{r}_i\right)}{\widehat{r}\left(\overline{\widetilde{T}_i}s_i-\widetilde{r}_i+\widehat{r}\right)}. \tag{13}$$

Similarly, if the scheduling strategy is *PS 2*, we have

$$\widetilde{T}_i=\frac{\widetilde{r}_i}{s_i}+\frac{\left(\widetilde{\lambda}_i\widetilde{r}_i^2+\widehat{\lambda}_i\widehat{r}^2\right)}{s_i\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)}\leq\overline{\widetilde{T}_i}. \tag{14}$$

Based on Eq. (14), we can obtain

$$\widehat{\lambda}_i \leq \frac{s_i\left(\overline{\widetilde{T}_i}\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)-\widetilde{r}_i\right)}{\widehat{r}^2}. \tag{15}$$

Finally, if the scheduling strategy is *PS 3*, we obtain

$$\widetilde{T}_i=\frac{\widetilde{r}_i}{s_i}+\frac{\widetilde{\lambda}_i\widetilde{r}_i^2}{s_i\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)}\leq\overline{\widetilde{T}_i}. \tag{16}$$

Eqs. (13) and (15) define the ranges of $\widehat{\lambda}_i$, which are transformed from $\widetilde{T}_i \leq \overline{\widetilde{T}_i}$. Eq. (16) indicates that such a range cannot be derived if we use *PS 3*, because $\widetilde{T}_i$ is independent of $\widehat{\lambda}_i$. In this case, we employ the stability of a queue $\rho_i=\frac{\widehat{\lambda}_i\widehat{r}+\widetilde{\lambda}_i\widetilde{r}_i}{s_i}<1$ [38, p. 263] to limit the range of $\widehat{\lambda}_i$, and get

$$\widehat{\lambda}_i<\frac{s_i-\widetilde{\lambda}_i\widetilde{r}_i}{\widehat{r}}. \tag{17}$$

Since the variables we want to solve are $\widehat{\lambda}_1,\widehat{\lambda}_2,\ldots,\widehat{\lambda}_n$, transforming the constraints $\widetilde{T}_i \leq \overline{\widetilde{T}_i}$ $(1\leq i\leq n)$ to a function of $\widehat{\lambda}_i$,

$$f_i\left(\widehat{\lambda}_i\right)=\begin{cases}\widehat{\lambda}_i-\dfrac{s_i\left(\overline{\widetilde{T}_i}\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)-\widetilde{r}_i\right)}{\widehat{r}\left(\overline{\widetilde{T}_i}s_i-\widetilde{r}_i+\widehat{r}\right)}, v_i\in G_1;\\ \widehat{\lambda}_i-\dfrac{s_i\left(\overline{\widetilde{T}_i}\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)-\widetilde{r}_i\right)}{\widehat{r}^2}, v_i\in G_2;\\ \widehat{\lambda}_i-\dfrac{s_i-\widetilde{\lambda}_i\widetilde{r}_i}{\widehat{r}}, v_i\in G_3;\end{cases} \tag{18}$$

is helpful to reduce the difficulty of solving the problem, and dose not affect the solution of the problem. Next, we construct a Lagrange function,

$$L(\widehat{\lambda}_1,...,\widehat{\lambda}_n,\phi,\tau_1,...,\tau_n)=\widehat{T}+\phi\left(\sum_{i=1}^{n}\widehat{\lambda}_i-\widehat{\lambda}\right)+\tau_i\sum_{i=1}^{n}f_i(\widehat{\lambda}_i).$$

Here, ϕ and $\tau_1,\tau_2,...,\tau_n$ are Lagrange multipliers. According to the KKT method, we have

$$\begin{cases}-\dfrac{\partial\widehat{T}}{\partial\widehat{\lambda}_i}=\phi+\tau_i\dfrac{\partial f_i\left(\widehat{\lambda}_i\right)}{\partial\widehat{\lambda}_i}, 1\leq i\leq n;\\ \widehat{\lambda}_1+\widehat{\lambda}_2+\cdots+\widehat{\lambda}_n=\widehat{\lambda};\\ \phi\neq 0;\\ \tau_i\geq 0,\widehat{\lambda}_i\geq 0, 1\leq i\leq n;\\ f_i\left(\widehat{\lambda}_i\right)\leq 0, 1\leq i\leq n;\\ \tau_i f_i\left(\widehat{\lambda}_i\right)=0, 1\leq i\leq n.\end{cases} \tag{19}$$

Since

$$\frac{\partial\widehat{T}}{\partial\widehat{\lambda}_i}=\frac{1}{\widehat{\lambda}}\left(\widehat{T}_i+\widehat{\lambda}_i\frac{\partial\widehat{T}_i}{\partial\widehat{\lambda}_i}\right)$$

and

$$\frac{\partial f_i\left(\widehat{\lambda}_i\right)}{\partial\widehat{\lambda}_i}=1,$$

based on the first equation of Eq. (19), we can obtain

$$-\frac{1}{\widehat{\lambda}}\left(\widehat{T}_i+\widehat{\lambda}_i\frac{\partial\widehat{T}_i}{\partial\widehat{\lambda}_i}\right)=\phi+\tau_i. \tag{20}$$

The $\widehat{T}_i$ in Eq. (20) is different for scheduling strategies *PS 1*, *PS 2*, and *PS 3*. Therefore, it is necessary to separately solve the $\partial\widehat{T}_i\big/\partial\widehat{\lambda}_i$ according to different scheduling strategies, namely,

$$\frac{\partial\widehat{T}_i}{\partial\widehat{\lambda}_i}=\begin{cases}\dfrac{\widehat{r}^2\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)+\widetilde{\lambda}_i\widetilde{r}_i^2\widehat{r}}{s_i\left(s_i-\widehat{\lambda}_i\widehat{r}-\widetilde{\lambda}_i\widetilde{r}_i\right)^2}, v_i\in G_1;\\ \dfrac{\widehat{r}^2\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)+\widetilde{\lambda}_i\widetilde{r}_i^2\widehat{r}}{\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i-\widehat{\lambda}_i\widehat{r}\right)^2}, v_i\in G_2;\\ \dfrac{\widehat{r}^2\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)+\widetilde{\lambda}_i\widetilde{r}_i^2\widehat{r}}{\left(s_i-\widetilde{\lambda}_i\widetilde{r}_i\right)\left(s_i-\widehat{\lambda}_i\widehat{r}-\widetilde{\lambda}_i\widetilde{r}_i\right)^2}, v_i\in G_3.\end{cases} \tag{21}$$

Combining Equations (4), (6), (8), (20), (21) and $\rho_i<1$, we can obtain

$$\widehat{\lambda}_i=\begin{cases}\dfrac{t_i}{\widehat{r}}-\dfrac{1}{\widehat{r}}\sqrt{\dfrac{t_i\left(t_i\widehat{r}+\widetilde{\lambda}_i\widetilde{r}_i^2\right)}{(\phi+\tau_i)\widehat{\lambda}s_i}}, v_i\in G_1;\\ \dfrac{t_i}{\widehat{r}}-\dfrac{1}{\widehat{r}}\sqrt{\dfrac{t_i\left(\widehat{r}t_i+\widetilde{\lambda}_i\widetilde{r}_i^2\right)}{(\phi+\tau_i)\widehat{\lambda}t_i+\dfrac{\widehat{r}}{s_i}\widetilde{\lambda}_i\widetilde{r}_i}}, v_i\in G_2;\\ \dfrac{t_i}{\widehat{r}}-\dfrac{1}{\widehat{r}}\sqrt{\dfrac{t_i\widehat{r}+\widetilde{\lambda}_i\widetilde{r}_i^2}{\phi\widehat{\lambda}}}, v_i\in G_3;\end{cases} \tag{22}$$

where $t_i=s_i-\widetilde{\lambda}_i\widetilde{r}_i$.

Now, we have obtained the closed-form solution of $\widehat{\lambda}_i$, i.e., Eq. (22). The next task is to resolve the values of Lagrange multipliers $\phi,\tau_1,\tau_2,...,\tau_n$.

Eq. (19) has the constraints $\tau_i\geq 0$, $f_i(\widehat{\lambda}_i)\leq 0$ and $\tau_i f_i(\widehat{\lambda}_i)=0$. Here, $\tau_i=0$ and $f(\widehat{\lambda}_i)=0$ can not hold at the same time, else the constraint $f(\widehat{\lambda}_i)=0$ is meaningless. Therefore, we have the following relationship between τ_i and $f_i(\widehat{\lambda}_i)$:

$$\begin{cases}\tau_i=0,\ f_i(\widehat{\lambda}_i)<0;\\ \tau_i>0,\ f_i(\widehat{\lambda}_i)=0.\end{cases} \tag{23}$$

It is easy to observe that $\widehat{\lambda}_i$ can be viewed as an increasing function of ϕ, namely $\widehat{\lambda}_i(\phi)$. Therefore, if we have $\tau_1=\tau_2=\cdots=\tau_n=0$, $\widehat{\lambda}_1,\widehat{\lambda}_2,...,\widehat{\lambda}_n$ can be resolved, because ϕ can be determined with binary search based on $\sum_{i=1}^{n}\widehat{\lambda}_i(\phi)=\widehat{\lambda}$. Of course, the assumption $\tau_1=\tau_2=\cdots=\tau_n=0$ maybe not true, due to the possibility of $f_i(\widehat{\lambda}_i(\phi))\geq 0$. However, we have specified $f_i(\widehat{\lambda}_i)\leq 0$. Therefore, once the case $f_i(\widehat{\lambda}_i(\phi))\geq 0$ exists, we set $f_i(\widehat{\lambda}_i)$ to 0, while in this case $\widehat{\lambda}_i$ can be directly obtained by Eq. (18).

Inspired by the above analyses, our algorithm is designed in terms of the following steps:

(i) We let $\tau_1 = \tau_2 = \cdots = \tau_n = 0$, and employ binary search to solve $\widehat{\lambda}_i$ based on $\sum_{i=1}^{n} \widehat{\lambda}_i(\phi) = \widehat{\lambda}$.

(ii) For those cores with $f(\widehat{\lambda}_i(\phi)) \geq 0$, we directly resolve $\widehat{\lambda}_i$ by $f(\widehat{\lambda}_i) = 0$, and do not consider this core no longer.

(iii) Repeat the steps (i) and (ii), until all $f_i(\widehat{\lambda}_i) \leq 0$ are met and $\widehat{\lambda}_1 + \widehat{\lambda}_2 + \cdots + \widehat{\lambda}_n = \widehat{\lambda}$.

When employing binary search to solve $\widehat{\lambda}_i$ based on $\sum_{i=1}^{n} \widehat{\lambda}_i(\phi) = \widehat{\lambda}$, we can use the constraints $\widehat{\lambda}_i \geq 0$ and $f(\widehat{\lambda}_i(\phi)) \leq 0$ to get the search scope of ϕ, i.e.,
For case $v_i \in G_1$, we have

$$\phi \in \boldsymbol{\phi_i} = \left[\frac{t_i\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i^2}{t_i\widehat{\lambda}s_i}, \frac{t_i\left(\overline{\widetilde{T}_i}s_i - \widetilde{r}_i + \widehat{r}\right)^2}{\widehat{\lambda}s_i\left(\widehat{r}t_i + \widetilde{\lambda}_i\widetilde{r}_i^2\right)} \right].$$

For case $v_i \in G_2$, we have

$$\phi \in \boldsymbol{\phi_i} = \left[\frac{1}{\widehat{\lambda}t_i}\left(\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i\left(\frac{\widetilde{r}_i}{t_i} - \frac{\widehat{r}}{s_i}\right)\right), \frac{t_i\left(t_i\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i^2\right)\widehat{r}^2}{\left(t_i\widehat{r} - s_i\left(\overline{\widetilde{T}_i}t_i - \widetilde{r}_i\right)\right)^2} - \frac{\widehat{r}}{s_i}\widetilde{\lambda}_i\widetilde{r}_i \right].$$

For case $v_i \in G_3$, we have

$$\phi \in \boldsymbol{\phi_i} = \left[\frac{t_i\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i^2}{\widehat{\lambda}}, +\infty \right].$$

The parameter ϕ is a Lagrange multiplier, which is common for all $\widetilde{\lambda}_i$. Thus, the range of ϕ is

$$\phi \in \boldsymbol{\phi_1} \bigcap \boldsymbol{\phi_2} \bigcap \cdots \bigcap \boldsymbol{\phi_n}. \tag{24}$$

The details of resolving the load balancing $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n$ are described in Algorithm 1.

TABLE 2
A motivating example.

		The first iteration			The second iteration		
i	$\overline{\widetilde{T}}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	$\widetilde{T}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	$\widetilde{T}_i$
1	0.700	2.406	0.946	**0.846**	2.225	0.800	0.700
2	0.900	2.310	0.957	0.857	2.348	0.996	0.896
3	0.910	2.214	0.969	0.869	2.252	1.008	0.908
4	0.930	2.119	0.981	0.881	2.155	1.021	0.921
5	0.940	2.023	0.994	0.894	2.058	1.035	0.935
6	0.960	1.928	1.009	0.909	1.962	1.050	0.950
		$\widehat{T} = 0.97434$			$\widehat{T} = 0.98315$		

A Motivating Example. This example is help for understanding Algorithm 1. We consider a system with 6 processors, and the following parameters:

- $\widehat{\lambda}$ is 13; $\widehat{r}$ is 0.25;
- $(\widetilde{r}_i, \alpha_i, P_i, s_i, PS_i)$ is $(0.15, 2.7, 0.1, 1.0, PS1)$ for $1 \leq i \leq 6$;
- $\widetilde{\lambda}_1, ..., \widetilde{\lambda}_6$ are $1.0, 1.2, ..., 2.0$ with step 0.2;
- $\overline{\widetilde{T}}_1, ..., \overline{\widetilde{T}}_6$ are $0.70, 0.90, 0.91, 0.93, 0.94, 0.96$.

We first calculate $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_6$ based on $\widehat{\lambda}_1(\phi) + \cdots + \widehat{\lambda}_6(\phi) = 13$, and get $\widetilde{T}_1 = 0.846$. Due to $\widetilde{T}_1 > \overline{\widetilde{T}_1}$ $(0.846 > 0.7)$, $\widehat{\lambda}_1$ is set to its upper limit 2.225 that is calculated from $\widetilde{T}_1 = \overline{\widetilde{T}_1}$. Let $\widehat{\lambda} = 13 - 2.225 = 10.775$, and proceed to solve the sub-problem based on $\widehat{\lambda}_2(\phi) + \widehat{\lambda}_3(\phi) + \widehat{\lambda}_4(\phi) + \widehat{\lambda}_5(\phi) + \widehat{\lambda}_6(\phi) = 10.775$. After the conditions $\sum_{i=1}^{6} \widehat{\lambda}_i = 13$ and $\widetilde{T}_i \leq \overline{\widetilde{T}_i}$ hold for all $1 \leq i \leq 6$, the problem is solved. The final result is $\widehat{T} = 0.98315$, and the details are reported in Table 2. To verify the quality of solution $\widehat{\lambda}_1 = 2.225$, we can set $\widehat{\lambda}_1$ to 2.224, 2.223, and 2.222, and get $\widehat{T} = 0.98325$, $\widehat{T} = 0.98334$, and $\widehat{T} = 0.98343$ respectively, which are large than $\widehat{T} = 0.98315$ corresponding to $\widehat{\lambda}_1 = 2.225$. Due to the limited space, the information of other parameters is not listed.

Algorithm 1 loadBalance.

Input: $s_i, g_i, \widetilde{\lambda}_i, \widetilde{r}_i, \overline{\widetilde{T}}_i, PS_i$, for all $1 \leq i \leq n$, $\widehat{\lambda}, \widehat{r}$.
Output: $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n$.
Calculate the low bound lb and up bound ub of ϕ based on Eq. (24);
$flag_1 = \cdots = flag_n = 1$;
while $(ub - lb > \varepsilon)$ **do**
  $\phi \leftarrow (ub + lb)/2$;
  for $(i \leftarrow 1; i \leq n; i \leftarrow i + 1)$ **do**
    if $PS_i = 1 \& flag_i$ **then**
      $\widehat{\lambda}_i \leftarrow \frac{t_i}{\widehat{r}} - \frac{1}{\widehat{r}}\sqrt{t_i\left(t_i\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i^2\right)/\phi\widehat{\lambda}s_i}$;
      if $f(\lambda_i) \geq 0$ **then**
        $\widehat{\lambda}_i \leftarrow s_i\left(\overline{\widetilde{T}_i}\left(s_i - \widetilde{\lambda}_i\widetilde{r}_i\right) - \widetilde{r}_i\right)/\widehat{r}\left(\overline{\widetilde{T}_i}s_i - \widetilde{r}_i + \widehat{r}\right)$;
        $flag_i = 0$;
      end if
    end if
    if $PS_i = 2 \& flag_i$ **then**
      $\widehat{\lambda}_i \leftarrow \frac{t_i}{\widehat{r}} - \frac{1}{\widehat{r}}\sqrt{t_i\left(t_i\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i^2\right)\Big/\left(\phi\widehat{\lambda}t_i + \frac{\widehat{r}}{s_i}\widetilde{\lambda}_i\widetilde{r}_i\right)}$;
      if $f(\lambda_i \geq 0)$ **then**
        $\widehat{\lambda}_i \leftarrow s_i\left(\overline{\widetilde{T}_i}\left(s_i - \widetilde{\lambda}_i\widetilde{r}_i\right) - \widetilde{r}_i\right)/\widehat{r}^2$; $flag_i = 0$;
      end if
    end if
    if $PS_i = 3 \& flag_i$ **then**
      $\widehat{\lambda}_i \leftarrow \frac{t_i}{\widehat{r}} - \frac{1}{\widehat{r}}\sqrt{\left(t_i\widehat{r} + \widetilde{\lambda}_i\widetilde{r}_i^2\right)/\phi\widehat{\lambda}}$;
      if $f(\lambda_i \geq 0)$ **then**
        $\lambda_i \leftarrow \left(s_i - \widetilde{\lambda}_i\widetilde{r}_i\right)/\widehat{r}$; $flag_i = 0$;
      end if
    end if
  end for
  if $\widehat{\lambda}_1 + \widehat{\lambda}_2 + ... + \widehat{\lambda}_n < \widehat{\lambda}$ **then**
    $lb \leftarrow \phi$;
  else
    $ub \leftarrow \phi$;
  end if
end while
return $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n$.

4.2 Power Optimization

4.2.1 Algorithm Analysis and Design

Energy efficiency is related to speeds of all processors. One of the troubles in pursuing the energy efficiency of heterogeneous systems is determining the speeds for all processors. Let M_i represent the available speeds of processor i. There are $\prod_{i=1}^{n} M_i$ options to find the speeds $s_1, s_2, ..., s_n$ from sets $\{s_{11}, ..., s_{1M_1}\}, \{s_{21}, ..., s_{2M_2}\}, ..., \{s_{n1}, ..., s_{nM_n}\}$. Therefore, the power optimization problem is a complex combinatorial optimization problem, and it is also difficult to obtain the best solution.

Energy efficiency makes trade off between performance and power. Usually, there are two typical considerations, which are maximization of performance under given power and minimization of power under given performance. These two considerations have a common attribute, namely, the ratio between the performance and power will be optimal

when the best energy efficiency is obtained [29]. Mapping to our model, we have

- $\widehat{T}/P$ is minimum when the problem is to optimize performance with given power.
- $\widehat{T}/P$ is maximal when the problem is to optimize power with given performance.

Before introducing $\widehat{T}/P$, an important conclusion is described as follows.

Theorem 1: If $[\dot{\widehat{\boldsymbol{\lambda}}}\ \dot{\boldsymbol{s}}]$ is the optimal solution of problem $minimizing\ \widehat{T}(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s})$ and satisfies $P(\dot{\widehat{\boldsymbol{\lambda}}}, \dot{\boldsymbol{s}}) = \overline{P}$, it is also the optimal solution of problem $minimizing\ P(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s})$ under the constraint $\widehat{T} \leq \widehat{T}(\dot{\widehat{\boldsymbol{\lambda}}}, \dot{\boldsymbol{s}})$, and has property $\overline{P} \leq P(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s})$. Here, $\widehat{\boldsymbol{\lambda}}$ and $\boldsymbol{s}$ respectively represent any vector $(\widehat{\lambda}_1, \widehat{\lambda}_2, \ldots, \widehat{\lambda}_n)$ and vector $(s_1, s_2, \ldots, s_n)$, where $\sum_{i=1}^{n} \widehat{\lambda}_i = \widehat{\lambda}$.

Proof: Since the solution $[\dot{\widehat{\boldsymbol{\lambda}}}\ \dot{\boldsymbol{s}}]$ is the optimal solution of problem $minimizing\ \widehat{T}(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s})$, and satisfies $P(\dot{\widehat{\boldsymbol{\lambda}}}, \dot{\boldsymbol{s}}) = \overline{P}$, we have

$$\widehat{T}(\dot{\widehat{\boldsymbol{\lambda}}}, \dot{\boldsymbol{s}}) \leq T(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s}). \tag{25}$$

We assume that $[\widehat{\boldsymbol{\lambda}}'\ \boldsymbol{s}']$ ($[\widehat{\boldsymbol{\lambda}}'\ \boldsymbol{s}'] \neq [\dot{\widehat{\boldsymbol{\lambda}}}\ \dot{\boldsymbol{s}}]$) is the optimal solution of problem $minimizing\ P(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s})$ under the constraint $\widehat{T} \leq \widehat{T}(\dot{\widehat{\boldsymbol{\lambda}}}, \dot{\boldsymbol{s}})$, namely,

$$P(\widehat{\boldsymbol{\lambda}}', \boldsymbol{s}') \leq P(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s}), \tag{26}$$

and

$$T(\widehat{\boldsymbol{\lambda}}', \boldsymbol{s}') \leq T(\dot{\widehat{\boldsymbol{\lambda}}}, \dot{\boldsymbol{s}}). \tag{27}$$

From Eq. (25), we get that Eq. (27) dose not hold. Therefore, the optimal solution of $minimizing\ P(\widehat{\boldsymbol{\lambda}}, \boldsymbol{s})$ with constraint $\widehat{T} \leq \widehat{T}(\dot{\widehat{\boldsymbol{\lambda}}}, \dot{\boldsymbol{s}})$ is $[\dot{\widehat{\boldsymbol{\lambda}}}\ \dot{\boldsymbol{s}}]$, and the theorem is proven. ■

Theorem 1 implies that Problems 1 and 2 (defined in Section 3.4) are equivalent to a certain degree. The ratio $\widehat{T}/P$ is unique when a system achieves best energy efficiency, and it is independent on the type of problem. This conclusion indicates that the joint optimization for performance and power could be improved based on $\widehat{T}/P$, because the extreme of $\widehat{T}/P$ corresponds to the optimal solution of problem.

Problem 1 (or 2) includes $2n$ variables that are $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n$ and $s_1, s_2, ..., s_n$. From Section 4.1 we know that $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n$ can be solved immediately if speeds $s_1, s_2, ..., s_n$ are given. Therefore, we can adjust the speeds of the processors to optimize performance and power. Here, an iteration approach is employed. In each iteration, a processor is picked to adjust its speed. How to pick the processor is described as follows:

(1) Processor i offers M_i optional speeds $\{s_{i1}, s_{i2}, ..., s_{iM_i}\}$ ($s_{i1} < s_{i2} < ... < s_{iM_i}$). Let $s_{1g_1}, s_{2g_2}, ..., s_{ng_n}$ represent the current speeds of the processors $1 \sim n$, respectively ($s_{i1} < s_{ig_i} < s_{iM_i}, 1 \leq g_i \leq M_i$). Based on $s_{1g_1}, s_{2g_2}, ..., s_{ng_n}$, we calculate the average response time $\widehat{T}$ and system power P using the load-balancing algorithm (Algorithm 1).

(2) Assume that changing the speed of processor i from s_{ig_i} to s_{ig_i-1} will lead to the average response time $\widehat{T}$ becoming $\widehat{T}'$, as well as the system power P becoming P'. We define a operator ΔR_i as shown below:

$$\Delta R_i = \frac{\Delta T}{\Delta P} = \frac{T - T'}{P' - P}. \tag{28}$$

(3) Traversing all of processors, a set $\{\Delta R_1, \Delta R_2, ..., \Delta R_n\}$ can be obtained. Then, we select a processor whose ΔR_i is maximal/minimal, and adjust its speed into the next-neighboring speed.

The speeds-adjustment-direction can be from high to low ($s_{g_i} \rightarrow s_{g_i-1}$), also be from low to high ($s_{g_i-1} \rightarrow s_{g_i}$). If using the former, we will select the processor with $min\{\Delta R_1, \Delta R_2, ..., \Delta R_n\}$ to change its speed. The philosophy is that more power reduction and less performance degradation is the best choice when decreasing the speeds of processors. If using the later, we will select the processor with $max\{\Delta R_1, \Delta R_2, ..., \Delta R_n\}$, because less power increase and more performance improvement is the best choice when increasing the speeds of processors. No matter which direction we choose, the final results will be the same. Please see the experiments in Section 5.2.

We always pick the processor with min/max ΔR_i to adjust its speed in every iteration, and iterate this process until the power or performance meets the given constraint. The pseudocode is described in Algorithm 2.

Algorithm 2 PerformancePowerOptimization.

Input: $\{s_{i1}, s_{i2}, ..., s_{iM_i}\}, g_i, \widetilde{\lambda}_i, \widetilde{r}_i, \overline{\widehat{T}_i}, PS_i$ for all $1 \leq i \leq n$, $\widehat{\lambda}, \widehat{r}, \overline{P}$.
Output: $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n, s_1, s_2, ..., s_n$.
1: **for** $(i \leftarrow 1; i \leq n; i \leftarrow i + 1)$ **do**
2: $g_i \leftarrow M_i$;
3: **end for**
4: $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n \leftarrow loadBalance$; //Call the Algorithm 1.
5: $\widehat{T} = \frac{\widehat{\lambda}_1}{\widehat{\lambda}}\widehat{T}_1 + \frac{\widehat{\lambda}_2}{\widehat{\lambda}}\widehat{T}_2 + \cdots + \frac{\widehat{\lambda}_n}{\widehat{\lambda}}\widehat{T}_n$, $P \leftarrow P_1 + P_2 + \cdots + P_n$;
6: $optT \leftarrow \widehat{T}$, $optP \leftarrow P$;
7: **while** $(P \geq \overline{P})$ **do**
8: $minRate \leftarrow +\infty, j \leftarrow -1$;
9: **for** $(i \leftarrow 1; i \leq n; i \leftarrow i + 1)$ **do**
10: $g_i \leftarrow g_i - 1$;
11: $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n \leftarrow loadBalance$; //Call the Algorithm 1.
12: $T' \leftarrow \frac{\widehat{\lambda}_1}{\widehat{\lambda}}\widehat{T}_1 + \frac{\widehat{\lambda}_2}{\widehat{\lambda}}\widehat{T}_2 + \cdots + \frac{\widehat{\lambda}_n}{\widehat{\lambda}}\widehat{T}_n$;
13: $P' \leftarrow P_1 + P_2 + \cdots + P_n$;
14: $\Delta R_i = \frac{T - T'}{P' - P}$;
15: **if** $\Delta R_i < minRate$ **then**
16: $j \leftarrow i$; //The index of the candidate.
17: $optT \leftarrow T', optP \leftarrow P'$;
18: **end if**
19: $g_i \leftarrow g_i + 1$;
20: **end for**
21: $\widehat{T} \leftarrow optT, P \leftarrow optP$;
22: $g_j \leftarrow g_j - 1$ //Processor j is selected.
23: **end while**
24: **return** $\widehat{\lambda}_1, \widehat{\lambda}_2, ..., \widehat{\lambda}_n, s_1, s_2, ..., s_n$.

Algorithm 2 can be used to solve Problem 1, and can also be used to solve Problem 2 if the judgment condition $P \geq \overline{P}$ is replaced by $\widehat{T} \leq \overline{\widehat{T}}$ (line 7 of Algorithm 2). The initial speed for each processor is set to the maximum speed that the processor can get (lines 1 and 2 of Algorithm 2). In each iteration, a processor is selected to change its speed (lines 9-22 of Algorithm 2). By iterating the steps (lines 7-23 of Algorithm 2), the optimization problem will be solved once the judgment condition is met (line 7 of Algorithm 2).

4.2.2 Convergence Analysis

In this subsection, we will analyze the convergence of the proposed algorithm. The number of iterations is dictated by the judgment condition. The power constraint $P \leq \overline{P}$ will be taken as the judgment condition when solving Problem 1. The performance constraint $\widehat{T} \leq \overline{\widehat{T}}$, in turn, will be taken as the judgment condition when solving Problem 2. Regardless of the type of problem, the speeds of all processors are always adjusted from high speed to low speed or from low speed to high speed. In short, in the iteration process, the adjustment direction of speed for each processor is consistent, i.e., it either keeps increasing or keeps decreasing.

Obviously, power $P = \sum_{i=1}^{n} P_i$ is a monotonically increasing function of s_i. This means that there must be a set $\{s_1, s_2, ..., s_n\}$ that can make the power $P = \sum_{i=1}^{n} P_i$ less than the constraint $\overline{P}$. Consequently, the algorithm is convergent when power $P \leq \overline{P}$ is employed as the judgment condition.

4.2.3 Time Complexity Analysis

In this section, we will analyze the time complexity. Since our algorithm combines load balancing and speed adjustment, the analysis will be done step by step according to these two parts. The details of the analysis of time complexity are as follows.

(1) The time complexity of load balancing (Algorithm 1). The Algorithm 1 contains one While loop and one For loop, where the For loop is inside the WHILE loop. The number of iterations of the For loop is n, and that of the While loop is $\log\left(\frac{ub-lb}{\varepsilon}\right)$, where ub and lb are respectively the up bound and low bound of ϕ, ε is the accuracy which is the dominant component. Therefore, the time cost of Algorithm 1 is about $n\log\left(\frac{ub-lb}{\varepsilon}\right)$.

(2) The time complexity of a single iteration for speed adjustment. In order to improve load balancing and power allocation, the speeds of processors need to be adjusted iteratively. In an iteration, the set $\{\Delta R_1, \Delta R_2, ..., \Delta R_n\}$ will be calculated. Thus, Algorithm 1 will be executed n times in a single iteration. The time complexity is less than $n^2\log\left(\frac{ub-lb}{\varepsilon}\right)$.

(3) The time complexity of joint optimization for performance and power(Algorithm 2). M_i represents the number of the adjustable speeds of processor i. The total number of adjustment times don't exceed $\sum_{i=1}^{n} M_i$, even if the speeds of all processors are adjusted from the highest speed to the lowest speed. Consequently, the time cost of Algorithm 2 is less than

$$\sum_{i=1}^{n} M_i n^2 \log\left(\frac{ub-lb}{\varepsilon}\right). \tag{29}$$

As n continues to increase, the value of Eq. (29) will get very large. In order to reduce calculation time, we can calculate $\{\Delta R_1, \Delta R_2, ..., \Delta R_n\}$ in parallel when n is very large. For instance, if $\{\Delta R_1, \Delta R_2, ..., \Delta R_n\}$ is calculated simultaneously by n machines, the time cost of Algorithm 2 can reduce to

$$\sum_{i=1}^{n} M_i n \log\left(\frac{ub-lb}{\varepsilon}\right).$$

In a real world application, the whole scheduling also requires other times such as the delay, communication and frequency switching times. These times may depend on real environments and platforms, we don't discuss here.

5 Numerical Experiments

In this section, we illustrate a number of numerical examples. Note that all of the parameters used in our experiments are for illustrative purposes only; they could be changed to any other real values.

Experimental parameters. We consider a system with 9 processors ($n = 9$). Each processor has a set of discrete speeds ranging from 0 to 2.5 with a step 0.01. The average task size and arrival rate of general tasks are $\widehat{r} = 0.25$ and $\widehat{\lambda} = 14$, respectively. To emphasize the heterogeneity between processors, each processor's parameters, such as the preloaded workload $\widetilde{\lambda}_i, \widetilde{r}_i$, task scheduling strategy and power consumption exponent α_i, are designed to be different, as shown in Table 3.

TABLE 3
Experiment parameters

i	1	2	3	4	5	6	7	8	9
$\widetilde{\lambda}_i$	1.0	0.8	0.6	1.0	0.8	0.6	1.0	0.8	0.6
$\widetilde{r}_i$	0.3	0.2	0.1	0.3	0.2	0.1	0.3	0.2	0.1
α_i	2.4	2.5	2.6	2.4	2.5	2.6	2.4	2.5	2.6
$P_i{}^*$	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
PS_i	1	1	1	2	2	2	3	3	3

5.1 Performance Comparison

This section will evaluate the effectiveness of the proposed algorithm. Because the problem of power-constrained performance optimization and the problem of performance-constrained power optimization are symmetrical, we shall conduct the experiment only for the previous problem to avoid repeating similar experiments.

Our problem is a nonlinear problem, which needs to consider both task assignment and power allocation. Some effective ways to deal with this problem are using intelligent search algorithms or distributing tasks and power proportionally. We will introduce several representative ones, which will be used to compare with our algorithm.

Task assignment. Here, the target of task assignment is to distribute the stream of general tasks onto n processors. There are two common approaches for task assignment, namely uniform distribution (UD) and proportional distribution (PD), the details of which are as follows.

- UD means that the stream of general tasks will be split into n sub-streams equally. Thus, the task assignment on processor i is
$$\widehat{\lambda}_i = \frac{1}{n}\widehat{\lambda},\ 1 \leq i \leq n.$$
- The target of PD is to make the workload across over all processors equally. So, we assume
$$\widehat{\lambda}_1 + \widetilde{\lambda}_1 = \widehat{\lambda}_2 + \widetilde{\lambda}_2 = \cdots = \widehat{\lambda}_n + \widetilde{\lambda}_n,$$
where
$$\widehat{\lambda}_1 + \widehat{\lambda}_2 + \cdots + \widehat{\lambda}_n = \widehat{\lambda}.$$

Then, the task assignment is

$$\widehat{\lambda}_i = \frac{\widehat{\lambda} + \widetilde{\lambda}_1 + \widetilde{\lambda}_2 + \cdots + \widetilde{\lambda}_n}{n} - \widetilde{\lambda}_i,\ 1 \le i \le n.$$

Note that $\widehat{\lambda}_i$ must subject to $\widehat{\lambda}_i \ge 0$. Thus, if $\widehat{\lambda}_i < 0$, we will let $\widehat{\lambda}_i = 0$ and $n = n - 1$; this represents that processor i will not be assigned general tasks.

Power allocation. In actual applications, the common and effective approaches for power allocation are equal-speed and equal-power, the details of which are as follows.

- Equal-speed, ES for short, refers to that all processors run at the same speed. Therefore, for frequency-conversion model, we have

$$\sum_{i=1}^{n} \left((\widehat{\lambda}_i \widehat{r}_i + \widetilde{\lambda}_i \widetilde{r}_i) s^{\alpha_i - 1} + P_i^* \right) = \overline{P};$$

for frequency-constant model, we have

$$\sum_{i=1}^{n} (s^{\alpha_i} + P_i^*) = \overline{P}.$$

The definitions of frequency-conversion model and frequency-constant model please see Section 3.2. To solve s, we can treat P as a function of s, and view s as a continuous variable on region $[0, b]$ where b could be any reasonable value such as 4 and 5. Due to $\alpha_i \approx 3$, $P(s)$ is a continuous and strictly increasing function of s. Therefore, s can be solved by binary search under condition $P(s) = \overline{P}$.

- Equal-power, EP for short, refers to that the system power is allocated to each processor equably. For frequency-conversion model, the power of processor i is

$$(\widehat{\lambda}_i \widehat{r}_i + \widetilde{\lambda}_i \widehat{r}_i) s^{\alpha_i - 1} + P_i^* = \frac{\overline{P}}{n},\ i \le 1 \le n.$$

Thus, the speed s_i is

$$s_i = \sqrt[\alpha_i - 1]{\frac{\frac{\overline{P}}{n} - P_i^*}{\widehat{\lambda}_i \widehat{r}_i + \widetilde{\lambda}_i \widehat{r}_i}}.$$

Similarly, for frequency-constant model, the speed s_i is

$$s_i = \sqrt[\alpha_i]{\frac{\overline{P}}{n} - P_i^*}.$$

Based on the above approaches for task assignment and power allocation, we obtain following algorithms, UD&ES, UD&EP, PD&ES and PD&EP.

We also use a genetic algorithm (GA) to solve the considered problem. There are several versions of the GA, and the version we are using is provided by matlab2018. As we know, the result of GA is affected by many parameters, where the initial population and the iteration number are dominant. In our experiments, the initial population is set to

$$\widehat{\lambda}_1 = \widehat{\lambda}_2 = \cdots = \widehat{\lambda}_n = \frac{1}{n}\widehat{\lambda}, s_i = \sqrt[\alpha_i]{\frac{\overline{P}}{n} - P_i^*}, 1 \le i \le n;$$

the maximum generations is set to 50; and the remaining parameters use the default values provided by matlab2018.

In addition, the stat-of-art OPL [12] will also be compared with our approach. OPL algorithm can solve the problem of optimising performance under pow constraints, it can obtain good and even the optimal solutions in the cases of low system heterogeneity. Similar to other algorithms, OPL fails to consider the response time of dedicated tasks. To conduct a fair comparison, we take into account only the response time of general task in our experiments.

We implement these algorithms with C++ on a laptop equipped an Inter i7 CPU, and compare the response time $\widehat{T}$ calculated by them under the given power $\overline{P}$. To obtain a full comparison, the power $\overline{P}$ is gradually adjusted from 13 to 26 with a step size of 1, and the corresponding results are plotted in Figs. 2 and 3 with the tool *OriginPro 9.0*. Fig. 2 corresponds to the results for frequency-constant model, and Fig. 3 to the results for frequency-conversion model. From Figs. 2 and 3, the following observations are drawn.

- Our algorithm can achieve better performance than other algorithms in the frequency-constant model, and the second performance in the frequency-conversion model.
- For task assignment, the PD approach is better than the UP approach. For power allocation, the ES approach is superior to the EP approach.
- GA can provide a good solution under the frequency-constant model; however, its performance is poor under the frequency-conversion model.

The above observations report that our approach outperforms other heuristic algorithms and is similar to the state-of-art OPL, in the aspect of workload and power allocation. The running time of UD&EP, UD&ES, PD&EP, and PD&ES is about 15 ms, that of OPL is 100 ms, and that of our approach is about $20 \sim 80$ ms. We have several reasons to support the value of the proposed algorithm. (1) Our approach gets the best performance in the frequency-constant model, while the frequency-constant model is a very common way of setting frequencies in practical. (2) Our approach can flexibility adjust optimisation objectives according to dynamic requirements. (3) Our approach has the property of easy implementation compared with state-of-art approaches, which is valuable in the field of engineering.

5.2 Convergence Stability

Here, the stability refers to the uniqueness of the result. As mentioned in Section 4.2.1, the speeds of processors can be adjusted from high to low, also from low to high. For many heuristics algorithms, their final results are closely related to initial parameters. This section will show that our results are not only affected by initial speeds, but also not affected by the direction of the speed adjustment. Due to the same conclusion to the frequency-conversion and frequency-constant models, we list only the experiments for frequency-conversion model. The experiment are carried out with the following steps.

Step 1: We consider a heterogeneous system with 9 processors, and assume $\widehat{\lambda} = 25; \widehat{r} = 0.2, \overline{P} = 30$. The initial parameters of processors are the same with Table 3.

Step 2: Set the initial speeds of processors to $(s_1, s_2, \ldots, s_9)$=(1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8).

TABLE 4
Adjusting speeds from high to low
$(s_1, s_2, \ldots, s_9) = (1.81, 1.81, ..., 1.81)$

Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.3	2.25	0.273	1.81	5.154
2	0.227	2.737	0.254	1.81	5.408
3	0.164	3.026	0.247	1.81	5.677
4	0.245	2.016	0.28	1.81	5.154
5	0.176	2.615	0.257	1.81	5.408
6	0.117	2.981	0.25	**1.8**	5.61
7	0.199	1.891	0.298	1.81	5.154
8	0.121	2.554	0.266	1.81	5.408
9	0.057	2.93	0.249	1.81	5.677
$\sum \lambda_i = 23.0006, \widehat{T} = 0.26142, P = 48.6476$					
.....					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.579	2.316	0.543	1.36	3.092
2	0.47	2.757	0.508	1.33	3.04
3	0.37	2.92	0.486	1.29	2.939
4	0.383	2.011	0.585	1.33	2.983
5	0.278	2.587	0.515	1.32	3.002
6	0.195	2.88	0.49	1.29	2.939
7	0.286	2.028	0.618	1.35	3.055
8	0.169	2.628	0.53	1.34	3.079
9	0.081	2.872	0.488	1.3	2.978
$\sum \lambda_i = 23.0000, \widehat{T} = 0.52371, P = 27.1054$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.58	2.319	0.544	1.36	3.092
2	0.463	2.729	0.5	1.33	3.04
3	0.371	2.924	0.487	1.29	2.939
4	0.383	2.015	0.586	1.33	2.983
5	0.278	2.59	0.516	1.32	3.002
6	0.195	2.884	0.491	1.29	2.939
7	0.288	2.032	0.633	**1.34**	3.019
8	0.169	2.632	0.531	1.34	3.079
9	0.081	2.876	0.489	1.3	2.978
$\sum \lambda_i = 23.0001, \widehat{T} = 0.52485, P = 27.0691$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.581	2.323	0.545	1.36	3.092
2	0.464	2.733	0.501	1.33	3.04
3	0.372	2.927	0.488	1.29	2.939
4	0.383	2.019	0.587	1.33	2.983
5	0.278	2.594	0.516	1.32	3.002
6	0.195	2.887	0.492	1.29	2.939
7	0.288	2.004	0.624	1.34	3.019
8	0.169	2.636	0.532	1.34	3.079
9	0.081	2.879	0.5	**1.29**	2.939
$\sum \lambda_i = 22.9999, \widehat{T} = 0.52602, P = 27.0297$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.582	2.326	0.546	1.36	3.092
2	0.464	2.736	0.502	1.33	3.04
3	0.372	2.931	0.489	1.29	2.939
4	0.384	2.022	0.589	1.33	2.983
5	0.278	2.597	0.517	1.32	3.002
6	0.195	2.891	0.493	1.29	2.939
7	0.288	2.007	0.625	1.34	3.019
8	0.171	2.639	0.544	**1.33**	3.04
9	0.081	2.851	0.493	1.29	2.939
$\sum \lambda_i = 23.0001, \widehat{T} = 0.52730, P = 26.9912$					

TABLE 5
Adjusting speeds from high to low
$(s_1, s_2, \ldots, s_9) = (1.57, 1.51, ..., 1.46)$

Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.44	2.537	0.408	1.57	3.952
2	0.351	2.855	0.385	1.51	3.802
3	0.285	2.529	0.4	1.3	2.978
4	0.327	2.042	0.444	1.48	3.562
5	0.231	2.77	0.39	1.52	3.848
6	0.163	3	0.384	**1.46**	3.675
7	0.259	1.91	0.48	1.46	3.48
8	0.159	2.431	0.422	1.42	3.403
9	0.071	2.927	0.381	1.46	3.675
$\sum \lambda_i = 23.0001, \widehat{T} = 0.40618, P = 32.3757$					
.....					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.579	2.316	0.543	1.36	3.092
2	0.47	2.757	0.508	1.33	3.04
3	0.37	2.92	0.486	1.29	2.939
4	0.383	2.011	0.585	1.33	2.983
5	0.278	2.587	0.515	1.32	3.002
6	0.195	2.88	0.49	1.29	2.939
7	0.286	2.028	0.618	1.35	3.055
8	0.169	2.628	0.53	1.34	3.079
9	0.081	2.872	0.488	1.3	2.978
$\sum \lambda_i = 23.0000, \widehat{T} = 0.52371, P = 27.1054$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.58	2.319	0.544	1.36	3.092
2	0.463	2.729	0.5	1.33	3.04
3	0.371	2.924	0.487	1.29	2.939
4	0.383	2.015	0.586	1.33	2.983
5	0.278	2.59	0.516	1.32	3.002
6	0.195	2.884	0.491	1.29	2.939
7	0.288	2.032	0.633	**1.34**	3.019
8	0.169	2.632	0.531	1.34	3.079
9	0.081	2.876	0.489	1.3	2.978
$\sum \lambda_i = 23.0001, \widehat{T} = 0.52485, P = 27.0691$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.581	2.323	0.545	1.36	3.092
2	0.464	2.733	0.501	1.33	3.04
3	0.372	2.927	0.488	1.29	2.939
4	0.383	2.019	0.587	1.33	2.983
5	0.278	2.594	0.516	1.32	3.002
6	0.195	2.887	0.492	1.29	2.939
7	0.288	2.004	0.624	1.34	3.019
8	0.169	2.636	0.532	1.34	3.079
9	0.081	2.879	0.5	**1.29**	2.939
$\sum \lambda_i = 22.9999, \widehat{T} = 0.52602, P = 27.0297$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.582	2.326	0.546	1.36	3.092
2	0.464	2.736	0.502	1.33	3.04
3	0.372	2.931	0.489	1.29	2.939
4	0.384	2.022	0.589	1.33	2.983
5	0.278	2.597	0.517	1.32	3.002
6	0.195	2.891	0.493	1.29	2.939
7	0.288	2.007	0.625	1.34	3.019
8	0.171	2.639	0.544	**1.33**	3.04
9	0.081	2.851	0.493	1.29	2.939
$\sum \lambda_i = 23.0001, \widehat{T} = 0.52730, P = 26.9912$					

TABLE 6
Adjusting speeds from low to high
$(s_1, s_2, \ldots, s_9) = (0.9, 0.9, ..., 0.9)$

Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	3.483	2.089	3.428	0.9	1.777
2	2.879	2.631	2.935	0.9	1.768
3	2.581	3.011	2.748	0.9	1.76
4	0.734	2.02	4.07	0.9	1.777
5	0.514	2.598	3.18	0.9	1.768
6	0.367	2.999	2.826	0.9	1.76
7	0.492	2.019	3.776	**0.91**	1.797
8	0.27	2.598	3.236	0.9	1.768
9	0.119	3.036	3.178	0.9	1.76
$\sum \lambda_i = 23.0000, \widehat{T} = 3.20829, P = 15.9370$					
.....					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.588	2.305	0.551	1.35	3.055
2	0.469	2.715	0.507	1.32	3.002
3	0.367	2.909	0.483	1.29	2.939
4	0.384	2.033	0.592	1.33	2.983
5	0.279	2.608	0.52	1.32	3.002
6	0.197	2.869	0.498	1.28	2.9
7	0.288	2.019	0.629	1.34	3.019
8	0.171	2.618	0.538	1.33	3.04
9	0.081	2.925	0.511	1.29	2.939
$\sum \lambda_i = 23.0001, \widehat{T} = 0.53095, P = 26.8775$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.587	2.301	0.55	1.35	3.055
2	0.458	2.711	0.496	**1.33**	3.04
3	0.374	2.938	0.49	1.29	2.939
4	0.384	2.029	0.591	1.33	2.983
5	0.279	2.604	0.519	1.32	3.002
6	0.197	2.866	0.497	1.28	2.9
7	0.288	2.015	0.628	1.34	3.019
8	0.171	2.614	0.537	1.33	3.04
9	0.081	2.922	0.51	1.29	2.939
$\sum \lambda_i = 23.0001, \widehat{T} = 0.52972, P = 26.9156$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.574	2.298	0.538	**1.36**	3.092
2	0.465	2.739	0.503	1.33	3.04
3	0.373	2.934	0.489	1.29	2.939
4	0.384	2.026	0.59	1.33	2.983
5	0.279	2.601	0.518	1.32	3.002
6	0.197	2.862	0.496	1.28	2.9
7	0.288	2.011	0.626	1.34	3.019
8	0.171	2.611	0.536	1.33	3.04
9	0.081	2.918	0.509	1.29	2.939
$\sum \lambda_i = 23.0000, \widehat{T} = 0.52854, P = 26.9523$					
Core	$\widetilde{\lambda}_i$	$\widehat{\lambda}_i$	$\widehat{T}_i$	s_i	P_i
1	0.582	2.326	0.546	1.36	3.092
2	0.464	2.736	0.502	1.33	3.04
3	0.372	2.931	0.489	1.29	2.939
4	0.384	2.022	0.589	1.33	2.983
5	0.278	2.597	0.517	1.32	3.002
6	0.194	2.859	0.485	**1.29**	2.939
7	0.288	2.007	0.625	1.34	3.019
8	0.171	2.607	0.535	1.33	3.04
9	0.081	2.915	0.508	1.29	2.939
$\sum \lambda_i = 23.0000, \widehat{T} = 0.52730, P = 26.9912$					

Step 3: Based on the above parameters, we can get the result of each iteration by using Algorithm 2. For easy observation, we show the results of the last 4 iterations in Table 4.

Step 4: Set the initial speeds to $(s_1, s_2, \ldots, s_9)$=(1.57, 1.51, 1.3, 1.48, 1.52, 1.47, 1.46, 1.42, 1.46), and run the Algorithm 2 again. The results of the last 4 iterations are shown in Table 5.

Step 5: Set the initial speeds of processors to $(s_1, s_2, \ldots, s_9)$=(0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9), and the other parameters are unchanged. In each iteration, we select a processor with $max\{\Delta R_i | 1 \leq i \leq 9\}$ and increase its speed. The definition of ΔR_i please see Eq. (28). The results of the last 4 iterations are shown in Table 5.

Step 6: From the Tables 4, 5 and 6, we find that the final result are the same, despite of the different initial speeds and even the opposite direction of speeds adjustment.

6 Real Platform Validation

In this section, a real system platform is designed to verify and investigate the difference between theoretical analysis and practical results. The experiment consists of three parts, that is, building the platform, testing performance, and comparing results.

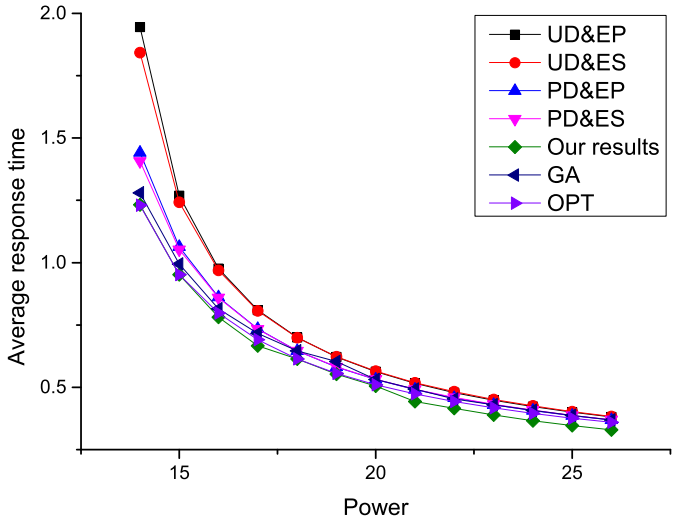

Fig. 2. Comparison of performance under frequency-constant model.

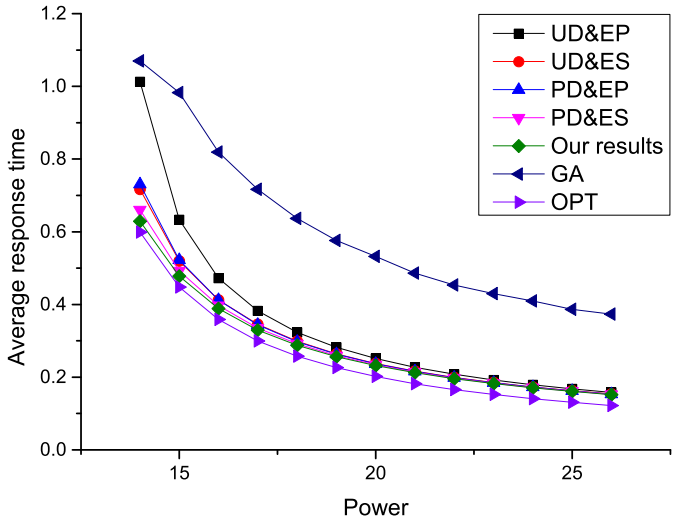

Fig. 3. Comparison of performance under frequency-conversion model.

Part (1) Build a real distributed system. We builded a small-scale distributed system, which includes 6 computing nodes, a switch and a power supply. Each computing node equips with 1GB of physical RAM, a Debian 4.7.2-5 operating system, a Coreter-A20 processor with adjustable frequencies 0.336, 0.360, 0.384, 0.408, 0.528, 0.600, 0.648, 0.672, 0.696, 0.720, 0.744, 0.768, 0.816, 0.864, 0.912, 0.960, 1.01GHz, and a DVFS tool called cpufreq. The switch is used for communication between nodes. The power supply is used to provide the voltage and current of the nodes.

Part (2) Test the system's performance. The speed and power of a processor vary with frequency, therefore we need to test them under different running-frequencies.

(1) A task commonly consists of many assembly instructions, such as JUMP, MOV, CMP, ADD and MUL; its execution time on a processor can be calculated by subtracting its start time from its finish time. By recording the number of instructions and execution time for many tasks, we can calculate the speeds of processors approximately.

(2) The power of a node is the product of voltage and current, and includes the power of both processor and other components. Due to the difficulty for separating the processor's static power and other component's power, we treat them as a whole, denoted as P^*, which can be obtained by placing processor's frequency to 0 GHz. In this experiment, the supply voltage is 5 volts, and the current is 0.27 amperes when $f = 0$ GHz. Thus, $P^* = 5V \times 0.27A = 1.35W$.

Table 7 shows the tested speeds and powers under different frequencies. Based on the obtained speed and power, we can use $\alpha \approx \log_s \left(\frac{P-P^*}{2}\right)$ to calculate parameter α, and show the results in the last row of Table 7. Here, we use $\frac{P-P^*}{2}$ because Cortex A20 is a dual core processor. In fact, the parameter α should be the same even if the processor runs at different frequencies. The reasons for obtaining different α might have several aspects. For instance, speed and power tests are not very accurate; the parameter β may not be 1. But overall, $\alpha \approx 3$ is reasonable and accords with objective facts. Since $\alpha = 3.12$ appears 3 times in Table 7, we employ 3.12 as the value of α.

TABLE 7
Speed and power under different frequency

f	1.01	0.960	0.912	0.864	0.816	0.768	0.744	0.720	0.696	0.336	0	GHz
s	0.660	0.640	0.620	0.597	0.572	0.545	0.515	0.479	0.438	0.225	-	Giga IPS
P	1.9	1.85	1.8	1.75	1.7	1.65	1.6	1.55	1.50	1.40	1.35	watt
α	3.107	3.106	3.120	3.120	3.120	3.126	3.133	3.128	3.137	2.473		

Part (3) Compare theoretical and actual experimental results. Corresponding to the above real platform, we consider a system with 6 processors, and each processor has a set of discrete speeds {0.660, 0.640, 0.620, 0.597, 0.572, 0.545, 0.515, 0.479, 0.438, 0.225} (giga instructions per second). According to the processing capacity of processors, we let $\widehat{r} = 0.1$ giga instructions, $\widehat{\lambda} = 12$ tasks per second, $\overline{P} = 8.8$ Watts, $\alpha_i = 3.12$, $\widetilde{r}_i = 0.12$ giga instructions, $P_i^* = 1.35$ Watts, $PS_i = 1$ for all $1 \leq i \leq 6$, $\widetilde{\lambda}_1, \widetilde{\lambda}_2, ..., \widetilde{\lambda}_6$ are $1.0, 1.2, 1.4, 1.6, 1.8, 2.0$ tasks per second and $\widetilde{\overline{T}}_1, \widetilde{\overline{T}}_2, ..., \widetilde{\overline{T}}_6$ are $0.3, 0.4, 0.8, 1.5, 1.5, 1.5$ seconds. Based on these parameters, we can obtain the theoretical results shown in Table 8(a).

TABLE 8
The results of Section 6

(a) Theoretical results.

i	$\widetilde{T}_i$	$\widehat{T}_i$	$\widehat{\lambda}_i$	s_i	P_i
1	0.300	0.270	1.557	0.660	1.464
2	0.400	0.369	2.126	0.640	1.488
3	0.800	0.763	2.379	0.545	1.462
4	0.880	0.843	2.260	0.545	1.465
5	0.914	0.877	2.059	0.545	1.467
6	1.004	0.965	1.618	0.515	1.448

(b) Real results.

i	$\widetilde{T}_i$	$\nabla\widetilde{T}_i$	$\widehat{T}_i$	$\nabla\widehat{T}_i$	P_i	∇P_i
1	0.301	0.001	0.272	0.002	1.484	0.020
2	0.401	0.001	0.368	0.001	1.504	0.016
3	0.801	0.001	0.756	0.007	1.471	0.009
4	0.878	0.002	0.833	0.010	1.474	0.009
5	0.905	0.009	0.872	0.005	1.474	0.007
6	0.981	0.023	0.945	0.020	1.455	0.007

Next, we will generate lots of tasks according to the above parameters, and assign these tasks onto the real

platform. Our purpose is to observe the gap between the theoretical results and practical results. The whole process includes generating tasks, assigning and executing tasks, analysing results, the details of which are introduced as follows.

Generate tasks. We stochastically generate one batch of general tasks and six batches of dedicated tasks, denoted as $S_G = \{\widehat{t}_1, \widehat{t}_2, ..., \widehat{t}_{|S_G|}\}$, $S_{D1} = \{\widetilde{t}_{1,1}, \widetilde{t}_{1,2}, ..., \widetilde{t}_{1,|S_{D1}|}\}$, $S_{D2} = \{\widetilde{t}_{2,1}, \widetilde{t}_{2,2}, ..., \widetilde{t}_{2,|S_{D2}|}\}, \ldots$, and $S_{D6} = \{\widetilde{t}_{6,1}, \widetilde{t}_{6,2}, ..., \widetilde{t}_{6,|S_{D6}|}\}$ respectively. For convenience, we uniformly use the sign $|S|$ to indicate the size of set S. Each task in the sets $S_G, S_{|D1|}, ..., S_{|D6|}$ consists of many assembly instructions including JUMP, MOV, CMP, ADD and MUL etc. The number of instructions of tasks in the sets $S_G, S_{D1}, S_{D2}, \ldots, S_{D6}$ obeys exponential distributions with mean $\widehat{r}, \widetilde{r}_1, \widetilde{r}_2, ..., \widetilde{r}_6$ giga instructions, respectively.

Assign and execute tasks. First, we assign tasks to the processors. Since the arrival rate of dedicated tasks is $\widetilde{\lambda}_i$ per second, the interval time sending tasks $\{\widetilde{t}_{i,1}, \widetilde{t}_{i,2}, ..., \widetilde{t}_{i,|S_{Di}|}\}$ to processor i follows exponential distribution with mean $1/\widetilde{\lambda}_i$ seconds ($1 \leq i \leq 6$). Similarly, the interval time sending tasks $\{\widehat{t}_1, \widehat{t}_2, ..., \widehat{t}_{|S_G|}\}$ to the system follows exponential distribution with mean $1/\widehat{\lambda}_i$ seconds. When a general task $\widehat{t}_i \in S_G$ arrives at the system, it will be assigned onto processor i with a probability of $\widehat{\lambda}_i/\widehat{\lambda}$. Next, we adjust the frequency of the processors. The theoretical speeds for processors 1-6 are shown in Table 8(b), which are 0.66, 0.64, 0.545, 0.545, 0.545, 0.515, respectively. From Table 7, we can find the corresponding processor frequencies to these theoretical speeds. Therefore, we set the frequencies of processors 1-6 at 1.01GHz, 0.96GHz, 0.778GHz, 0.778GHz, 0.778GHz, 0.744GHz, respectively.

Analyze the results. We start the system at time 0, and record the arrival time at_j, start time st_j and finish time ft_j of task t_j during the test. Therefore, we can get the waiting time wt_j via $st_j - at_j$, the execution time et_j via $ft_j - st_j$, and the response time rt_j via $ft_j - at_j$. Here, we mark S_{G_i} ($S_{G_i} \subseteq S_G$) and S_{D_i} as the set of general and dedicated tasks executed on processor i respectively, and let $|S_{G_i}|$ and $|S_{D_i}|$ represent the number of tasks of set S_{G_i} and set S_{G_i} respectively. Then the average response time of general tasks $\widehat{T}_i$ and that of dedicated tasks $\widetilde{T}_i$ are

$$\widehat{T}_i = \sum_{t_j \in S_{Gi}} \frac{rt_j}{|S_{G_i}|} = \sum_{t_j \in S_{Gi}} \frac{ft_j - at_j}{|S_{G_i}|}, 1 \leq i \leq 6$$

and

$$\widetilde{T}_i = \sum_{t_j \in S_{Di}} \frac{ft_j - at_j}{|S_{D_i}|}, 1 \leq i \leq 6.$$

The utilization of node i is,

$$\rho_i = \frac{\sum_{t_j \in S_{Gi}} et_j + \sum_{t_j \in S_{Di}} et_j}{ft_{exit}}, 1 \leq i \leq 6,$$

where ft_{exit} is the finish time of the last task on processor i. The power is

$$P_i = \rho_i \times \left(\frac{P - P^*}{2} + P^*\right) + (1 - \rho_i) \times P^*, 1 \leq i \leq 6,$$

where P is 1.9, 1.85, 1.65, 1.65, 1.65, 1.5 when i is 1, 2, 3, 4, 5, 6, respectively. The $\widehat{T}_i$, $\widetilde{T}_i$, and P_i tested on real platform are shown in Table 8(b). The term ∇P_i in Table 8 represents the absolute error between theoretical power and real power, similar usages are the terms $\nabla\widetilde{T}_i$ and $\nabla\widehat{T}_i$. From Table 8(b), we observe that $\nabla\widetilde{T}_i$, $\nabla\widetilde{T}_i$, and ∇P_i are less than 0.023 ($0.023/1.004 \approx 2.2\%$), 0.02 ($0.02/0.965 \approx 2\%$), and 0.09 ($0.09/1.465 \approx 6.1\%$), respectively. The errors may be caused by several reasons, such as ignored power, ambient noise, the test accuracy and so on. Overall, the theoretical results are basically consistent with real results.

7 Conclusion

We have highlighted that the focus of system optimization should be flexibly adjusted between performance and power according to the dynamic variation of environments. We have investigated the optimization by assigning a stream of general tasks to a heterogeneous system with multiple processors, wherein each processor has been preloaded with dedicated tasks and employs a different scheduling strategy. Our investigation is based on a multi-queuing model of multi-processors. We have optimized the system performance and system power by load balancing and energy efficiency. We have proposed a load-balancing algorithm, which can achieve the optimal load distribution while satisfying the performance constraints on dedicated tasks, based on a set of given processor frequencies. Based on this algorithm, we have further proposed a performance-satisfied power optimization algorithm that can not only improve performance under given power but also reduce power under given performance. Finally, we have analyzed and verified our approach based on numerical experiments and a real platform. The proposed method can provide reference and guidance for the design of heterogeneous multi-processor systems.

There are still some issues worth of further investigation. First, in our model, it is assumed that the scheduling strategy for each processor has been given in advance. In practice, how to decide the strategy requires analytical research. Second, the communications are not taken into consideration, which may increase the gap between theoretical analysis and practical applications. Finally, the experimental platform is also worth upgrading, it does not represent the actual working system.

Acknowledgments

This work is supported by the Natural Science Foundation of China Grant No. 61902118, and China Postdoctoral Science Foundation No. 2019M662771.

References

[1] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 585–599, 2016.

[2] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 45–58, 2014.

[3] R. Subrata, A. Y. Zomaya, and B. Landfeldt, "Game-theoretic approach for load balancing in computational grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 1, pp. 66–76, 2007.

[4] S. Penmatsa and A. T. Chronopoulos, "Game-theoretic static load balancing for distributed systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 4, pp. 537–555, 2011.

[5] K. Li, "Optimal load distribution for multiple classes of applications on heterogeneous servers with variable speeds," *Software: Practice and Experience*, vol. 48, no. 10, pp. 1805–1819, 2018.

[6] F. Bonomi and A. Kumar, "Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler," *IEEE Transactions on Computers*, vol. 39, no. 10, pp. 1232–1250, 1990.

[7] K. W. Ross and D. D. Yao, "Optimal load balancing and scheduling in a distributed computer system," *Journal of the Acm*, vol. 38, no. 3, pp. 676–689, 1991.

[8] S. Zeltyn, Z. Feldman, and S. Wasserkrug, "Waiting and sojourn times in a multi-server queue with mixed priorities," *Queueing Systems*, vol. 61, no. 4, pp. 305–328, 2009.

[9] K. Li, "Improving multicore server performance and reducing energy consumption by workload dependent dynamic power management," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 122–137, 2016.

[10] T. Atmaca, T. Begin, A. Brandwajn, and H. Castel-Taleb, "Performance evaluation of cloud computing centers with general arrivals and service," *IEEE Transactions on parallel and distributed systems*, vol. 27, no. 8, pp. 2341–2348, 2015.

[11] K. Li, "Optimal load distribution in nondedicated heterogeneous cluster and grid computing environments," *Journal of Systems Architecture*, vol. 54, no. 12, pp. 111–123, 2008.

[12] J. Huang, Y. Liu, R. Li, K. Li, J. An, Y. Bai, F. Yang, and G. Xie, "Optimal power allocation and load balancing for non-dedicated heterogeneous distributed embedded computing systems," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 24–36, 2019.

[13] K. Li, "Computation offloading strategy optimization with multiple heterogeneous servers in mobile edge computing," *IEEE Transactions on Sustainable Computing*, 2019, DOI: 10.1109/TSUSC.2019.2904680.

[14] A. Munir, S. Ranka, and A. Gordon-Ross, "High-performance energy-efficient multicore embedded computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 684–700, 2012.

[15] L. A. Barroso and U. Hlzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[16] M. Al-daloo, A. Yakovlev, and B. Halak, "Energy efficient bootstrapped cmos inverter for ultra-low power applications," in *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2016, pp. 516–519.

[17] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 7–12.

[18] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar, "Reducing peak power with a table-driven adaptive processor core," in *Ieee/acm International Symposium on Microarchitecture*, 2009, pp. 189–200.

[19] G. Kornaros, *Multi-Core Embedded Systems*. CRC Press, Inc., 2010.

[20] G. Xie, G. Zeng, R. Li, and K. Li, "Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 62–75, 2017.

[21] B. Yang, Z. Li, S. Chen, T. Wang, and K. Li, "Stackelberg game approach for energy-aware resource allocation in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3646–3658, 2016.

[22] X. Zhan and S. Reda, "Power budgeting techniques for data centers," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2267–2278, 2015.

[23] M. Shojafar, N. Cordeschi, and E. Baccarelli, "Energy-efficient adaptive resource management for real-time vehicular cloud services," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 196–209, 2016.

[24] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, and K. Li, "Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems," *IEEE Transactions on Sustainable Computing*, vol. PP, no. 99, pp. 167–181, 2017.

[25] A. Das, A. Kumar, and B. Veeravalli, "Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems," *IEEE Transactions on Parallel Distributed Systems*, vol. 27, no. 3, pp. 869–884, 2016.

[26] Y. Xiang and S. Pasricha, "Soft and hard reliability-aware scheduling for multicore embedded systems with energy harvesting," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 4, pp. 220–235, 2015.

[27] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time markov decision processes," in *Design Automation Conference, 1999. Proceedings.*, 1999, pp. 555–561.

[28] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Mobile Computing*. Springer, 1994, pp. 449–471.

[29] S. Yang, R. A. Shafik, G. V. Merrett, and E. Stott, "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning," in *International Workshop on Power and Timing Modeling, Optimization and Simulation*, 2015, pp. 103–110.

[30] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev, "Power-aware performance adaptation of concurrent applications in heterogeneous many-core systems," in *International Symposium on Low Power Electronics and Design*, 2016, pp. 368–373.

[31] J. Huang, R. Li, J. An, D. Ntalasha, F. Yang, and K. Li, "Energy-efficient resource utilization for heterogeneous embedded computing systems," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1518–1531, 2017.

[32] Y. Tian, C. Lin, and K. Li, "Managing performance and power consumption tradeoff for multiple heterogeneous servers in cloud computing," *Cluster Computing*, vol. 17, no. 3, pp. 943–955, 2014.

[33] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, vol. 53, no. 3, pp. 50–58, 2010.

[34] J. Mei, K. Li, and K. Li, "Customer-satisfaction-aware optimal multiserver configuration for profit maximization in cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 1, pp. 17–29, 2017.

[35] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," in *Proceedings. 41st Design Automation Conference, 2004.*, 2004, pp. 868–873.

[36] M. Bambagini, M. Marinoni, H. Aydin, and G. C. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Transactions in Embedded Computing Systems*, vol. 15, no. 1, pp. 1–34, 2016.

[37] J. Zhou, J. Sun, X. Zhou, T. Wei, M. Chen, S. Hu, and X. S. Hu, "Resource management for improving soft-error and lifetime reliability of real-time mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2215–2228, 2019.

[38] A. O. Allen, *Probability, statistics, and queueing theory*. Academic Press, 1990.

[39] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2013.