Energy-Aware Processor Merging Algorithms for Deadline Constrained Parallel Applications in Heterogeneous Cloud Computing

Guoqi Xie, *Member, IEEE*, Gang Zeng, *Member, IEEE*, Renfa Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract—Energy efficiency has become a key issue for cloud computing platforms and data centers. Minimizing the total energy consumption of an application is one of the most important concerns of cloud providers, and satisfying the deadline constraint of an application is one of the most important quality of service requirements. Previous methods tried to turn off as many processors as possible by integrating tasks on fewer processors to minimize the energy consumption of a deadline constrained parallel application in a heterogeneous cloud computing system. However, our analysis revealed that turning off as many processors as possible does not necessarily lead to the minimization of total energy consumption. In this study, we propose an energy-aware processor merging (EPM) algorithm to select the most effective processor to turn off from the energy saving perspective, and a quick EPM (QEPM) algorithm to reduce the computation complexity of EPM. Experimental results on real and randomly generated parallel applications validate that the proposed EPM and QEPM algorithms can reduce more energy than existing methods at different scales, parallelism, and heterogeneity degrees.

Index Terms—Dynamic voltage and frequency scaling (DVFS), deadline constraint, energy-aware, heterogeneous cloud computing systems, processor merging

1 Introduction

1.1 Background

Heterogeneous cloud computing and data centers consisting of diverse sets of processors or virtual machines offer computing and data storage services and solutions at a large scale. However, these solutions entail high costs and environmental effects because of high energy consumptions at various levels of computational and data storage processes. For example, the Tianhe-2 system in the National Supercomputer Center in Guangzhou (China), the world's fastest supercomputing system in November 2014 and 2015, contains 16,000 computer nodes and consumes 17,808 KW of power [1]. Energy consumption is a major issue that affects the development and use of computing systems as well as the human environment. In heterogeneous cloud computing systems, a parallel application with precedence-constrained tasks is represented by a directed acyclic graph (DAG), in which the nodes represent the tasks and the edges represent the communication messages between tasks [2], [3], [4], [5].

Resource providers and users are the two types of roles with conflicting requirements in cloud computing systems. For providers, minimizing the total energy consumption of an application is one of the most important concerns. For users, the deadline constraint (i.e., timing constraint, response-time constraint, real-time constraint) of an application is one of the most important quality of service (QoS) requirements [6]. The problem of minimizing the energy consumption of a deadline constrained application with precedence constrained tasks on heterogeneous cloud computing systems has been studied recently in a number of studies by using dynamic voltage and frequency scaling (DVFS) energy control technology to simultaneously scales down the processor's supply voltage and frequency while tasks are running [4], [5]. In [4], the problem is solved by reclaiming the slack time based on the latest finish time (LFT); however, this strategy merely minimizes the dynamic energy consumption and ignores the static energy consumption. A processor still consumes some amount of static energy even when it is idle. Furthermore, different from embedded systems where static energy consumption is only a small fraction of the total energy consumption, in large-scale cloud computing systems, each processor still needs to execute several system softwares and middlewares when the application is not executed on it. Turning off processors is an available way in some platforms [7]. In [5], the authors presented the DVFS-enabled energy-efficient workflow task scheduling (DEWTS) algorithm by processor

- *G. Xie and R. Li are with the College of Computer Science and Electronic Engineering, Hunan University, Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan 410082, China. E-mail: {xgqman, lirenfa}@hnu.edu.cn.*
- *G. Zeng is with the Graduate School of Engineering, Nagoya University, Aichi 4648603, Japan. E-mail: sogo@ertl.jp.*
- *K. Li is with the College of Computer Science and Electronic Engineering, Hunan University, Key Laboratory for Embedded and Network Computing of Hunan Province, Hunan 410082, China, and the Department of Computer Science, State University of New York, New Paltz, New York 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 14 July 2016; revised 5 Apr. 2017; accepted 14 May 2017. Date of publication 17 May 2017; date of current version 13 July 2017. (Corresponding author: Renfa Li.)
Recommended for acceptance by C. Dobre, G. Mastorakis, C.X. Mavromoustakis, and F. Xhafa.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSUSC.2017.2705183

merging to reduce static and dynamic energy consumption. The DEWTS algorithm is implemented by turning off processors with a small number of tasks (if two processors have the same task number, then the processor with low dynamic energy utilization defined in Definition 1 is turned off) to implement processor merging while satisfying the deadline. However, the DEWTS algorithm has the following limitations.

(1) Turning off the processors with a small number of tasks or processors with low dynamic energy utilization is not always effective with respect to energy consumption comparing with turning off any other processor. Such a policy is not energy-aware; thus, the achieved energy reduction is limited.

(2) Turning off as many processors as possible does not necessarily lead to the minimization of total energy consumption because the energy efficiency of each processor could be different. Therefore, energy consumption should be considered explicitly when turning off processors.

1.2 Our Contributions

This study aims to implement energy-aware processor merging to minimize the total energy consumption of deadline constrained parallel applications in heterogeneous cloud computing systems. The main contributions of this study are as follows.

(1) Considering the fact that each processor may have different energy efficiency, we propose an energy-aware processor merging (EPM) algorithm to minimize both dynamic and static energy consumption while satisfying the deadline constraint of the application. The algorithm is energy-aware because it always chooses the most effective processor to turn off in terms of saving energy.

(2) In order to overcome the high computation complexity of the EPM algorithm, we propose a quick EPM (QEPM) algorithm to deal with large-scale parallel applications. The algorithm can achieve a good balance between significant energy saving and reasonable computation time.

(3) Experiments on randomly generated and real parallel applications including fast Fourier transform, Diamond graph, and Gaussian elimination are conducted extensively. Experimental results validate that the proposed EPM and QEPM algorithms can reduce more energy than the state-of-the-art algorithm under different deadline constraints and scale conditions.

The rest of this paper is organized as follows. Section 2 reviews related studies. Section 3 develops related models and preliminaries. Section 4 presents the EPM and QEPM algorithms. Section 5 evaluates the performance of the EPM and QEPM algorithms. Section 6 concludes this study.

2 Related Work

Energy-aware design and algorithms have been well studied in computing systems, such as embedded computing, cloud computing, and cluster computing systems [8], [9], [10], [11], [12], and networking systems, such as cognitive radio networks, wireless networks, and vehicular delay-tolerant networks [13], [14], [15], [16], [17], [18]. In this study, we focus on energy optimization of computing systems, particularly on cloud computing systems.

Many studies have been conducted recently to minimize energy consumption while satisfying deadline constraint [19], [20], [21], [22]. However, these studies are restricted to independent tasks. As heterogeneous systems continue to be scaled up, DAG-based parallel applications with precedence constrained tasks, such as fast Fourier transform and Gaussian elimination applications, increase in number [2], [3], [4].

The problem of scheduling tasks on multiple processors is NP-hard [23]. Numerous meta-heuristic algorithms, such as tabu search [24], genetic algorithm (GA) [25], artificial immunity [26], simulated annealing [27], ant colony algorithm [28], and chemical reaction optimization (CRO) [29], are widely used in DAG-based parallel application scheduling. These algorithms usually generate better schedule quality than heuristic algorithms, but the computation time of these algorithms is higher than heuristic algorithms due to the poor search efficiency and frequent solution evaluation [30]. Scheduling tasks on heterogeneous processors with the objective of minimizing the schedule length of a DAG-based parallel application is a well-known NP-hard optimization problem, and heuristic list scheduling algorithms [2], [3], [11], [12], [31] have been proposed to generate near-optimal solutions of the multi-processors scheduling. A review of recent related research on energy consumption optimization for DAG-based parallel applications with precedence constrained tasks is provided in the following.

Zong et al. [32] considered energy-aware duplication scheduling algorithms for a parallel application in a homogeneous system. Lee and Zomaya [33] presented energy-conscious scheduling to minimize the energy consumption and schedule length of a parallel application simultaneously in a heterogeneous cloud system. However, these studies do not consider the deadline constraint of the application. The problem of minimizing the energy consumption of a deadline constrained application with precedence constrained sequential tasks [20] and precedence constrained parallel tasks (i.e., a parallel application) [1] has been investigated in different studies. However, these studies merely focused on homogeneous multiprocessors with shared memory. In [11], [12], the problem of minimizing the schedule length of a energy consumption constrained parallel application were studied by presenting a heuristic solution that minimum energy values are preassigned to unassigned tasks. Huang et al. [4] studied the problem of minimizing the energy consumption of a deadline constrained parallel application in a heterogeneous cloud system by using the enhanced energy-efficient scheduling (EES) algorithm, which reclaims the slack time by extending the finish time of each task to its LFT; however, this strategy merely minimizes the dynamic energy consumption and does not aim to minimize the static energy consumption.

Static energy consumption is an important part of total energy consumption in cloud computing systems. Turning off partial processors can reduce static energy consumption significantly and can therefore minimize the total energy consumption. In [34], the authors considered static power consumption and presented leakage-aware scheduling heuristics that determine the best trade-off among three techniques: DFVS, turning off processor, and finding the optimal number of processors. However, the scheduling heuristics is limited to a homogeneous multi-processor system with

TABLE 1
Important Notations Used in This Study

Notation	Definition
$c_{i,j}$	Communication time between the tasks n_i and n_j
$w_{i,k}$	Execution time of the task n_i running on the processor u_k with the maximum frequency
$D(G)$	Given deadline of the application G
$LB(G)$	Lower bound of the application G
$SL(G)$	Schedule length of the application G
$E_{\rm s}(G)$	Static energy consumption of the application G
$E_{\rm d}(G)$	Dynamic energy consumption of the application G
$E_{\rm total}(G)$	Total energy consumption of the application G
$EST(n_i, u_k, f_{k,h})$	Earliest start time of the task n_i on the processor u_k with the frequency $f_{k,h}$
$EFT(n_i, u_k, f_{k,h})$	Earliest finish time of the task n_i on the processor u_k with the frequency $f_{k,h}$
$E_{\rm d}(n_i, u_k, f_{k,h})$	Dynamic energy consumption of the task n_i on the processor u_k with the frequency $f_{k,h}$
$AFT(n_i)$	Actual finish time of the task n_i
$LFT(n_i, u_k)$	Latest finish time of the task n_i on the processor u_k
$TN(u_k)$	Assigned task number on the processor u_k
$DEU(u_k)$	Dynamic energy utilization on the processor u_k
$SL_k(G)$	Schedule length of the application G when the processor u_k is turned-off
$E_{k,\rm s}(G)$	Static energy utilization of the application G when the processor u_k is turned-off
$E_{k,\rm d}(G)$	Dynamic energy utilization of the application G when the processor u_k is turned-off
$E_{k,\rm total}(G)$	Total energy utilization of the application G when the processor u_k is turned-off

shared memory. In [35], the authors presented energy-aware task scheduling algorithms to sustain the schedule length and energy consumption of a parallel application simultaneously by identifying inefficient processors and turning them off to reduce energy consumption. However, these algorithms do not aim at deadline constrained parallel applications and do not use DVFS technique. In [5], the authors adopted the existing EES algorithm and presented the DEWTS algorithm to reduce static and dynamic energy consumption by turning off as many processors as possible. As pointed out in the Section 1.2, the DEWTS is not energy-aware and is inefficient to reduce energy.

3 Models and Preliminaries

Table 1 lists important notations and their definitions that are used in this study.

3.1 Application Model

Let $U = \{u_1, u_2, \ldots, u_{|U|}\}$ denote a set of heterogeneous processors, where $|U|$ represents the size of set U. For any set X, $|X|$ is used to denote its size. Similar to [5], we also assume that communication can be overlapped with computation, which means data can be transmitted from one processor to another while a task is being executed on the recipient processor. A parallel application running on processors is represented by a DAG $G = (N, M, C, W)$ [2], [3], [4], [5], [11], [12].

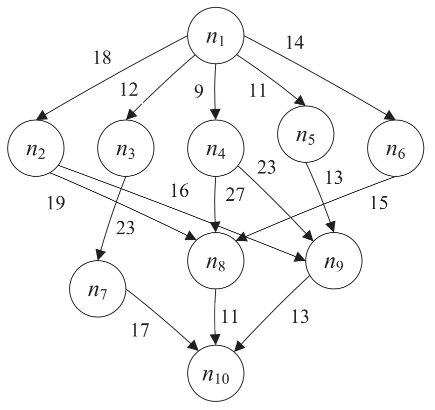


Fig. 1. Motivating example of a DAG-based parallel application with 10 tasks [2], [11], [12], [31].

(1) N represents a set of nodes in G, and each node $n_i \in N$ represents a task with different execution time values on different processors. In addition, task executions of a given application are assumed to be non-preemptive which is possible in many systems [5]. $pred(n_i)$ represents the set of the immediate predecessor tasks of n_i. $succ(n_i)$ represents the set of the immediate successor tasks of n_i. The task that has no predecessor task is denoted as $n_{\rm entry}$; the task that has no successor task is denoted as $n_{\rm exit}$. If an application has multiple entry or multiple exit tasks, then a dummy entry or exit task with zero-weight dependencies is added to the graph.

(2) W is a $|N| \times |U|$ matrix, where $w_{i,k}$ denotes the execution time of n_i running on u_k with the maximum frequency [8].

(3) M is a set of communication edges, and each edge $m_{i,j} \in M$ represents the communication message from n_i to n_j. Accordingly, $c_{i,j} \in C$ represents the communication time of $m_{i,j}$ if n_i and n_j are not assigned to the same processor. When tasks n_i and task n_j are allocated to the same processor, $c_{i,j}$ becomes zero because we assume that the intra-processor communication cost can be ignored [5].

Let $LB(G)$ represent the lower bound of the application, and is the minimum schedule length of application G when all tasks are executed on the processors with the maximum frequencies by using a DAG-based scheduling algorithm (e.g., HEFT [2], [4], [5]). $D(G)$ represents the deadline of application G and should be larger than or equal to $LB(G)$. $SL(G)$ represents the generated schedule length of G.

Fig. 1 shows a motivating example of a DAG-based parallel application. Table 2 shows a matrix of execution time values in Fig. 1. The example shows 10 tasks executed on 3 processors $\{u_1, u_2, u_3\}$. The weight 14 of n_1 and u_1 in Table 2 represents the execution time with the maximum frequency denoted by $w_{1,1} = 14$. The same task has different execution time values on different processors because of the heterogeneity of the processors. The weight 18 of the edge between n_1 and n_2 represents the communication time denoted as $c_{1,2}$ if n_1 and n_2 are not assigned to the same processor.

3.2 Power and Energy Model

Given that an almost linear relationship exists between voltage and frequency, DVFS scales down voltage and

TABLE 2
Execution Time Values of Tasks on Different Processors with the Maximum Frequencies of the Application in Fig. 1 [2], [11], [12], [31]

Task	u_1	u_2	u_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

frequency simultaneously to save energy. Similar to [8], [9], [10], we use the term "frequency change" to refer to changing voltage and frequency simultaneously. Considering a DVFS-capable system, we also adopt the system-level power model used in [8], [9], [10], [11], [12], where the power consumption at frequency f is provided by

$$P(f) = P_{\mathrm{s}} + h(P_{\mathrm{ind}} + P_{\mathrm{d}}) = P_{\mathrm{s}} + h(P_{\mathrm{ind}} + C_{\mathrm{ef}} f^m).$$

P_s represents the static power and can be removed only by powering off the processor. P_{ind} represents frequency-independent dynamic power and can be removed by putting the system into the sleep state. P_{d} represents frequency-dependent dynamic power and depends on frequencies. h represents system states and indicates whether dynamic powers are currently consumed in the system. When the system is active, h = 1; otherwise, when system is in the sleep mode, h = 0. C_{ef} represents effective switching capacitance. m represents the dynamic power exponent and is not less than 2. Both C_{ef} and m are processor-dependent constants.

Excessive overhead is associated with turning on/off a system. P_{s} is always consumed and not manageable unless the processor is turned off [8], [9], [10], [11], [12]. Given that static energy consumption accounts for an important part of the total energy consumption in cloud computing systems, this study simultaneously considers static and dynamic powers [4], [5]. Note that we do not take the communication energy consumption into consideration for simplicity because we mainly focus on the task energy reduction in this study.

Because of the P_{ind}, less P_{d} does not result less energy consumption. That is, a minimum energy-efficient frequency f_{ee} exists [8], [9], [10], [11], [12] and it is denoted by

$$f_{\mathrm{ee}} = \sqrt[m]{\frac{P_{\mathrm{ind}}}{(m-1)C_{\mathrm{ef}}}}. \tag{1}$$

Assuming that the frequency of a processor varies from minimum available frequency $f_{\min}$ to maximum frequency $f_{\max}$, the lowest frequency to execute a task should be

$$f_{\mathrm{low}} = \max(f_{\min}, f_{\mathrm{ee}}). \tag{2}$$

Hence, any actual effective frequency f_h should belong to the scope of $f_{\mathrm{low}} \leqslant f_{\mathrm{h}} \leqslant f_{\max}$.

Given that the number of processors is $|U|$ in the system and these processors are completely heterogeneous, each processor should have individual power parameters [11], [12]. Here, we define the static power set as

$$\{P_{1,\mathrm{s}}, P_{2,\mathrm{s}}, \ldots, P_{|U|,\mathrm{s}}\},$$

frequency-independent dynamic power set

$$\{P_{1,\mathrm{ind}}, P_{2,\mathrm{ind}}, \ldots, P_{|U|,\mathrm{ind}}\},$$

frequency-dependent dynamic power set

$$\{P_{1,\mathrm{d}}, P_{2,\mathrm{d}}, \ldots, P_{|U|,\mathrm{d}}\},$$

effective switching capacitance set

$$\{C_{1,\mathrm{ef}}, C_{2,\mathrm{ef}}, \ldots, C_{|U|,\mathrm{ef}}\},$$

dynamic power exponent set

$$\{m_1, m_2, \ldots, m_{|U|}\},$$

minimum energy-efficient frequency set

$$\{f_{1,\mathrm{ee}}, f_{2,\mathrm{ee}}, \ldots, f_{|U|,\mathrm{ee}}\},$$

and actual effective frequency set

$$\left\{\begin{array}{l} \{f_{1,\mathrm{low}}, f_{1,\alpha}, f_{1,\beta}, \ldots, f_{1,\max}\}, \\ \{f_{2,\mathrm{low}}, f_{2,\alpha}, f_{2,\beta}, \ldots, f_{2,\max}\}, \\ \ldots, \\ \{f_{|U|,\mathrm{low}}, f_{|U|,\alpha}, f_{|U|,\beta}, \ldots, f_{|U|,\max}\} \end{array}\right\}.$$

Let $E_{\mathrm{s}}(G)$ represent the processor-generated static energy consumption of application G. Considering that turned-off processors do not consume energy, $E_{\mathrm{s}}(G)$ must be the sum of the static energy consumptions of all turned-on processors. Therefore, $E_{\mathrm{s}}(G)$ is calculated by

$$E_{\mathrm{s}}(G) = \sum_{k=1, u_k \textit{ is on}}^{|U|} \big(P_{k,s} \times SL(G)\big). \tag{3}$$

Let $P_{\mathrm{d}}(n_i, u_k, f_{k,h})$ represent the dynamic power consumption of task n_i on processor u_k with frequency $f_{k,h}$. Considering that $P_{\mathrm{d}}(n_i, u_k, f_{k,h})$ contains frequency-independent dynamic power P_{ind} and frequency-dependent dynamic power $P_{k,\mathrm{d}}$, it is calculated by

$$P_{\mathrm{d}}\big(n_i, u_k, f_{k,h}\big) = \big(P_{k,\mathrm{ind}} + C_{k,\mathrm{ef}} \times {f_{k,h}}^{m_k}\big). \tag{4}$$

Let $E_{\mathrm{d}}(n_i, u_k, f_{k,h})$ represent the dynamic energy consumption of task n_i on processor u_k with frequency $f_{k,h}$, which is calculated by

$$E_{\mathrm{d}}\big(n_i, u_k, f_{k,h}\big) = \big(P_{k,\mathrm{ind}} + C_{k,\mathrm{ef}} \times {f_{k,h}}^{m_k}\big) \times \frac{f_{k,\max}}{f_{k,h}} \times w_{i,k}, \tag{5}$$

by substituting Eqs. (4) into (5).

Considering that the application's dynamic energy consumption $E_{\mathrm{d}}(G)$ is sum of the dynamic energy consumptions of all tasks, $E_{\mathrm{d}}(G)$ is calculated by

$$E_{\mathrm{d}}(G) = \sum_{i=1}^{|N|} E_{\mathrm{d}}\big(n_i, u_{pr(i)}, f_{pr(i),hz(i)}\big), \tag{6}$$

where $u_{pr(i)}$ and $f_{pr(i),hz(i)}$ represent the assigned processor and frequency of n_i, respectively.

TABLE 3
Upward Rank Values for Tasks of the Motivating Parallel Application

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
$rank_{\rm u}(n_i)$	108	77	80	80	69	63.3	42.7	35.7	44.3	14.7

Considering that the application's total energy consumption $E_{\rm total}(G)$ is the sum of its static energy consumption $E_{\rm s}(G)$ and dynamic energy consumption $E_{\rm d}(G)$, $E_{\rm total}(G)$ is calculated by

$$E_{\rm total}(G) = E_{\rm s}(G) + E_{\rm d}(G). \tag{7}$$

In this study, we ignore the overheads of the frequency transitions due to the negligible amount of time (e.g., 10 μs-150 μs [5], [33]).

3.3 Lower Bound Certification

Similar to state-of-the art studies [4], [5], this study also employs the heterogeneous earliest finish time (HEFT) algorithm to certify the lower bound of a parallel application. The lower bound is calculated by

$$LB(G) = \min_{u_k \in U}\{EFT(n_{\rm exit}, u_k, f_{k,\max})\}. \tag{8}$$

A relative deadline $D(G)$, which is larger than or equal to lower bound $LB(G)$, is then provided for the application. List scheduling includes two phases, namely, task prioritization and task allocation. The HEFT algorithm is one of the most popular DAG-based scheduling algorithms for reducing the schedule length to a minimum while achieving low complexity and high performance in heterogeneous systems and has been employed in energy-efficient scheduling [4], [5]. The two-phase HEFT algorithm has two important functions.

1) Task Prioritization. HEFT utilizes the upward rank value ($rank_{\rm u}$) of a task (Eq. (9)) as the task priority standard. In this case, the tasks are ordered according to the descending order of $rank_{\rm u}$. Table 3 shows the $rank_{\rm u}$ values of all tasks of the motivating parallel application (Fig. 1) obtained by using

$$rank_{\rm u}(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)}\{c_{i,j} + rank_{\rm u}(n_j)\}, \tag{9}$$

where $\overline{w_i}$ represents the average execution time of task n_i and is calculated by

$$\overline{w_i} = \left(\sum_{k=1}^{|U|} w_{i,k}\right) / |U|.$$

2) Task Assignment. $EST(n_i, u_k, f_{k,\max})$ and $EFT(n_i, u_k, f_{k,\max})$ represent the earliest start time (EST) and earliest finish time (EFT), respectively, of task n_i on processor u_k with maximum frequency $f_{k,\max}$. $EFT(n_i, u_k, f_{k,\max})$ is considered as the task assignment criterion in HEFT because it can satisfy the local optimal of each task. The aforementioned attributes are calculated by

TABLE 4
Power Parameters of Processors (u_1, u_2, and u_3)

u_k	$P_{k,{\rm s}}$	$P_{k,{\rm ind}}$	$C_{k,{\rm ef}}$	m_k	$f_{k,{\rm ee}}(f_{k,{\rm low}})$	$f_{k,\max}$
u_1	0.3	0.06	0.8	2.9	0.33	1.0
u_2	0.2	0.07	1.2	2.7	0.29	1.0
u_3	0.1	0.07	1.0	2.4	0.29	1.0

$$\begin{cases} EST(n_{\rm entry}, u_k, f_{k,\max}) = 0 \\ EST(n_i, u_k, f_{k,\max}) = \max\left\{avail[k], \max\limits_{n_x \in pred(n_i)}\{AFT(n_x) + c'_{x,i}\}\right\}, \end{cases} \tag{10}$$

and

$$EFT(n_i, u_k, f_{k,\max}) = EST(n_i, u_k, f_{k,\max}) + w_{i,k}. \tag{11}$$

$avail[k]$ is the earliest available time that processor u_k is ready for task execution, and $AFT(n_x)$ represents the actual finish time (AFT) of task n_x. $c'_{x,i}$ represents the actual communication time between n_x and n_i. If n_x and n_i are assigned to the same processor, then $c'_{x,i} = 0$; otherwise, $c'_{x,i} = c_{x,i}$. n_i is assigned to the processor with the minimum EFT by using the insertion-based scheduling strategy. n_i is inserted into the slack with the minimum EFT. The difference between EFT and AFT is that EFT is the value before task assignment, whereas AFT is the value after task assignment.

We provide an example to show the results of the HEFT algorithm. To evaluate energy consumption, the power parameters for all the processors are shown in Table 4. The minimum energy-efficient frequency $f_{k,{\rm ee}}$ (considered $f_{k,{\rm low}}$ in this example) for each processor is obtained according to Eq. (1). Meanwhile, similar to [8], [9], the maximum frequency $f_{k,\max}$ for each processor is assumed to be 1.0.

Example 1. Table 5 shows the task assignment of the motivating parallel application (Fig. 1) using the HEFT algorithm. Each row represents a task assignment and its corresponding values. The final static and dynamic energy consumptions are $E_{\rm s}(G) = 48$ and $E_{\rm d}(G) = 122.52$, respectively; hence, the total energy consumption is $E_{\rm total}(G) = 170.52$.

Fig. 2 shows the Gantt chart of parallel application G (Fig. 1) using the HEFT algorithm. The lower bound is obtained as $LB(G) = 80$, and the deadline is set to $D(G) = 100$. The arrows in Fig. 2 represent the generated communication time values between tasks.

3.4 Problem Statement

We assume that we have a parallel application G and a heterogeneous multi-processor set U with different frequency levels. The problem to be addressed is to assign an available processor with a proper frequency for each task while saving energy consumption and ensuring that the obtained makespan of the application does not exceed its deadline. The objective is to determine the processor and frequency assignments of all tasks to minimize the total energy consumption

$$E_{\rm total}(G) = E_{\rm s}(G) + E_{\rm d}(G),$$

subject to the deadline constraint

$$SL(G) \leqslant D(G),$$

TABLE 5
Task Assignment of the Motivating Parallel Application (Fig. 1) Using the HEFT Algorithm

n_i	u_k	$f_{k,\max}$	$AST(n_i)$	$AFT(n_i)$	$E_{\mathrm{d}}(n_i, u_k, f_{k,\max})$
n_1	u_3	1.0	0	9	9.63
n_3	u_3	1.0	9	28	20.33
n_4	u_2	1.0	18	26	10.16
n_2	u_1	1.0	27	46	11.18
n_5	u_3	1.0	28	38	10.70
n_6	u_2	1.0	26	42	20.32
n_9	u_2	1.0	56	68	15.24
n_7	u_3	1.0	38	49	11.77
n_8	u_1	1.0	57	62	4.3
n_{10}	u_2	1.0	73	80	8.89
$LB(G) = 80 < D(G) = 100$, $E_{\mathrm{total}}(G) = 48 + 122.52 = 170.52$					

and the frequency selection constraint

$$f_{pr(i),\mathrm{low}} \leqslant f_{pr(i),hz(i)} \leqslant f_{pr(i),\max},$$

for $\forall i : 1 \leqslant i \leqslant |N|, u_{pr(i)} \in U$.

4 Energy-Aware Processor Merging

4.1 Processor Merging

We list some important basic concepts as follows before explaining the details in this section.

Let $TN(u_k)$ represent the assigned task number on the processor u_k using a scheduling algorithm. Then the processor with the minimum assigned task number is calculated by

$$TN(u_{\min}) = \min_{u_k \in U}\{TN(u_k)\}. \tag{12}$$

Definition 1 (Dynamic Energy Utilization). *The dynamic energy utilization is defined as the ratio of the dynamic energy consumption on the processor to the total energy consumption on the processor*

$$DEU(u_k) = \frac{\sum_{i=1,u_{pr(i)}=u_k}^{|N|} E_{\mathrm{d}}(n_i, u_k, f_{k,hz(i)})}{\sum_{i=1,u_{pr(i)}=u_k}^{|N|} E_{\mathrm{d}}(n_i, u_k, f_{k,hz(i)}) + P_{k,s} \times SL(G)}. \tag{13}$$

Then the processor with the minimum dynamic energy utilization is calculated by

$$DEU(u_{\min}) = \min_{u_k \in U}\{DEU(u_k)\}. \tag{14}$$

Definition 2 (Latest Finish Time (LFT)). *The LFT of a task is considered to be the task's deadline and is calculated by*

$$\begin{cases} LFT(n_{\mathrm{exit}}) = D(G) \\ LFT(n_i) = \min\Big\{ \min\limits_{n_i \in succ(n_i)} \{AST(n_i) + c'_{i,j}\}, AST(n_{dn(i)}) \Big\} \end{cases}, \tag{15}$$

where $n_{dn(i)}$ represents the downward neighbor (DN) task of n_i on the same processor.

Considering that each processor entails static energy consumption and it can be removed only by turning off the processor, the state-of-the-art DEWTES algorithm is proposed by turning off several processors to further minimize the energy consumption while still satisfying the deadline

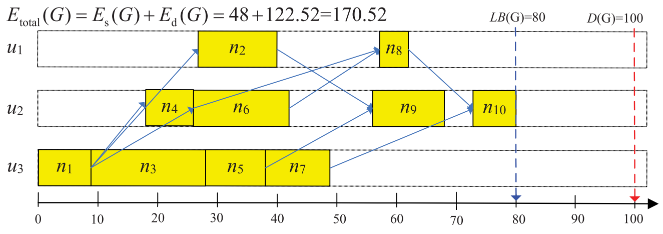


Fig. 2. Scheduling Gantt chart of the motivating parallel application using the HEFT algorithm.

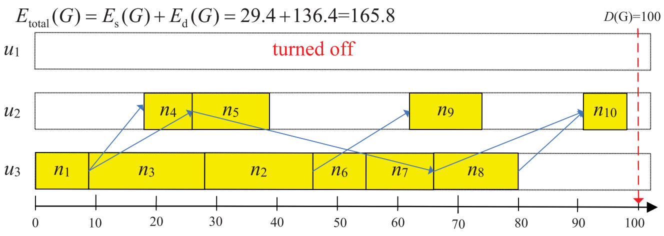


Fig. 3. HEFT-generated scheduling of the motivating parallel application on u_2 and u_3 when u_1 is turned off.

constraint of the application. The core objective of DEWTS is to gradually turn off partial processors with low utilization and reassign the tasks on these turned-off processors to the turned-on processors until the schedule length exceeds the deadline. We illustrate the motivating parallel application to explain the entire process of the DEWTS algorithm.

(1) DEWTS first invokes the HEFT algorithm for all the turned-on processors (u_1, u_2, and u_3), and the result is similar to that in Fig. 2. Obviously, we have $SL(G) = 80$, and the deadline of the parallel application is satisfied.

(2) Given that processor u_1 has the minimum task number calculated by Eq. (12) in Fig. 2, it should be turned off in advance (if two processors have the same task number, then the processor with low dynamic energy utilization calculated by Eq. (13) is turned off as pointed out earlier). Hence, DEWTS continues to invoke the HEFT algorithm for all the turned-on processors (u_2 and u_3), and we have $SL(G) = 98$, as shown in Fig. 3. In this case, as $SL(G) = 98$, the deadline of the parallel application is still satisfied.

(3) DEWTS invokes the EES algorithm [4] for u_2 and u_3 to save energy while satisfying the deadline of the parallel application, as shown in Fig. 4. The main idea of the EES algorithm is that the $AFT(n_i)$ obtained by the HEFT algorithm can be extended to $LFT(n_i)$ defined in Definition 2 by using the EES algorithm because slacks exist between two adjacent tasks in the same processor. For example, as shown in Fig. 2, we have $n_{dn(2)} = n_8$ and $n_{dn(9)} = n_{10}$. In other words, the LFT extension of n_i is restricted by its downward neighbor. For example, as shown in Fig. 2, the AFT of n_{10} can be extended to 100, and the AFT of n_9 can be extended to the AST of n_{10} while satisfying the deadline constraint.

4) Given that processor u_2 has a small task number, it should be turned off. DEWTS continues to invoke the HEFT algorithm for all the turned-on processors (u_3); however, the generated schedule length is $SL(G) = 143$ and cannot satisfy the deadline of the

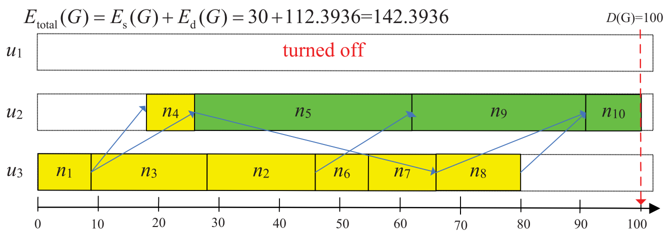


Fig. 4. EES-generated scheduling of the motivating parallel application on u_2 and u_3 when u_1 is turned off.

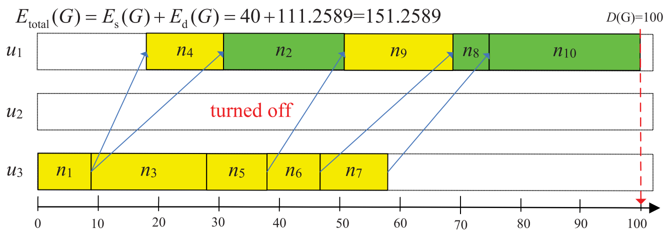


Fig. 5. EES-generated scheduling of the motivating parallel application on u_1 and u_3 when u_2 is turned off.

application. Then, the final result should be that in Fig. 4. The final static and dynamic energy consumptions are $E_s(G) = 30$ and $E_d(G) = 112.3936$, respectively; hence, the total energy consumption is $E_{total}(G) = 142.3936$, which is less than the 160.6569 of the EES algorithm.

4.2 Energy-Aware Processor Merging

We list some important basic concepts as follows before explaining the details in this section.

Let $E_{k,s}(G)$ (calculated by Eq. (3)) and $E_{k,d}(G)$ (calculated by Eq. (6)) represent the static and dynamic energy consumption of the application G, respectively, when the processor u_k is turned-off. Thus, the total energy consumption $E_{k,total}(G)$ can be calculated by

$$E_{k,total}(G) = E_{k,s}(G) + E_{k,d}(G). \quad (16)$$

Let $SL_k(G)$ represent the schedule length of the application G by the HEFT algorithm when the processor u_k is turned-off. Then the processor u_{min} that can lead to the minimum total energy consumption among the remaining active processors while guaranteeing that the schedule length is less than or equal to the deadline can be found by

$$E_{min,total}(G) = \min_{u_k \in U, u_k \text{ } is \text{ } off, \text{ } SL_k(G) \leqslant D(G)} \{E_{k,total}(G)\}, \quad (17)$$

As mentioned early, although the DEWTS algorithm can satisfy the deadline constraint of the application, it merely reduces the energy consumption through simple processor merging, and its energy efficiency is limited. Consider the same example in Fig. 4, if u_2 or u_3 is turned off instead of the u_1, then the results of Figs. 5 and 6 can be obtained.

Compared with Figs. 4, 5, and 6, turning off u_3 can lead to the minimum total energy consumption. The static energy consumption of processor may be different because each processor has its own static power $P_{k,s}$. Such results indicate that turning off the processor with a small number of tasks does not necessarily lead to the maximum reduction of energy in heterogeneous systems, and more sophistical selection is needed to further decrease the energy consumption.

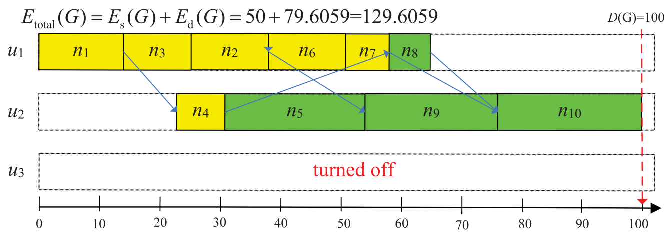


Fig. 6. EES-generated scheduling of the motivating parallel application on u_1 and u_2 when u_3 is turned off.

Inspired by the aforementioned analysis, we present the energy-aware processor merging algorithm called EPM described in Algorithm 1 to further minimize the energy consumption while satisfying the deadline constraint of the parallel application.

Algorithm 1. The EPM Algorithm

```
Input: $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $G$, and $D(G)$
Output: $E_{total}(G)$
1:  Invoke the HEFT algorithm on all the processors
    to obtain the initial total energy consumption $E_{total}(G)$;
2:  Put all the processors in to the turned-on processor
    set $turned\_on\_processor\_set$;
3:  while ($turned\_on\_processor\_set$ is not null) do
4:    for (each turned-on processor
      $u_k \in turned\_on\_processor\_set$) do
5:      Let $u_k$ be a turned off state;
6:      Invoke the HEFT algorithm on all turned-on processors to obtain the schedule length $SL_k(G)$ if $u_k$ is
        turned off;
7:      if ($SL_k(G)$ <= $D(G)$) then
8:        Invoke the EES algorithm on all turned-on processors to obtain the dynamic energy consumption
          $E_{k,d}(G)$;
9:        Calculate the static energy consumption $E_{k,s}(G)$ on
          all turned-on processors using Eq. (3);
10:       Calculate the total energy consumption $E_{k,total}(G)$
          Eq. (16);
11:     end if
12:   end for
13:   if (no result satisfies $SL_k(G)$ <= $D(G)$) then
14:     break;
15:   else
16:     Select the processor $u_{min}$ with the
        minimum $E_{min,total}(G)$, namely, could generate
        the minimum energy consumption in the
        remaining processors;
17:     Turn off $u_{min}$ and remove it from
        $turned\_on\_processor\_set$ ;
18:     Update the total energy consumption
        $E_{total}(G)$ as $E_{min,total}(G)$
19:   end if
20: end while
```

The main idea of the EPM algorithm is that it executes energy-aware processor merging by turning off the processor without which can save the maximum energy consumption while satisfying the deadline constraint of the application. The details of EPM are explained as follows (Lines 3-20).

(1) In Lines 4-12, the algorithm tries to turn off each processor. The energy consumption of the application

on the remaining active processors is calculated by using the HEFT and EES algorithm. Let $E_{k,\text{total}}(G)$ and $SL_k(G)$ represent the total energy consumption and schedule length of the application when the processor u_k is turned-off, respectively.

(2) In Lines 15-19, the processor that can lead to the minimum total energy consumption (Eq. (17)) among the remaining active processors is actually turned off while guaranteeing that the schedule length is less than or equal to the deadline.

(3) In Lines 13-15, if turning off any processor cannot satisfy the deadline constraint of the application, then the algorithm is terminated.

Consider the motivating example that each processor is attempted to be turned off. The scheduling Gantt charts are shown in Figs. 4 (turning off u_1), 5 (turning off u_2), and 6 (turning off u_3). Turning off any processor can satisfy the deadline constraint of the application. Subsequently, u_3 is actually turned off because the minimum energy consumption is generated. Hence, the final energy consumption using the EPM algorithm is 129.6059 (Fig. 6), which is less than that using the DEWTS algorithm (Fig. 4).

The time complexity of the EPM algorithm is analyzed. The maximum number of turned-off processors is O($|U|$) time (Line 3). Traversing all processors for selection can be done in O($|U|$) time (Line 4). Invoking the HEFT algorithm should be done in O($|N|^2 \times |U|$) time (Line 6). Therefore, the complexity of the EPM is algorithm O($|N|^2 \times |U|^3$). However, the state-of-the DEWTS algorithm entails O($|N|^2 \times |U|^2$); hence, EPM is one exponent higher than the DEWTS algorithm.

4.3 Quick Energy-Aware Processor Merging

Although the EPM algorithm can minimize energy consumption through energy-aware processor merging, it requires large computation effort. To reduce the energy consumption of a large-scale parallel application with an acceptable amount of computation time, a quick energy-aware processor merging (QEPM) algorithm is presented. Similar to EPM, the QEPM can also reduce energy consumption while satisfying the deadline constraint of the application, but it has low computation complexity. The detailed algorithm is described in Algorithm 2.

The main difference between QEPM and EPM is that the operation of traversing all processors for selection in Lines 4-12 of Algorithm 1 is removed from the while loop, as shown in Lines 2-10 of Algorithm 2. In other words, QEPM no longer attempts to turn off each processor in each loop but just directly selects the processor $u_{\min}$ with the minimum $E_{\min,\text{total}}(G)$ from the turned-on processor set based on previous ordering in Line 11. The main advantage of QEPM is that its time complexity is reduced to O($|N|^2 \times |U|^2$), but such a simplification may result in larger energy consumption than the EPM algorithm.

Algorithm 2. The QEPM Algorithm

```
Input: $U = \{u_1, u_2, \ldots, u_{|U|}\}$, $G$, and $D(G)$
Output: $E_{\text{total}}(G)$
1: Invoke the HEFT algorithm on all all the processors to
   obtain the initial total energy consumption $E_{\text{total}}(G)$;
2: for (each turned-on processor
   $u_k \in turned\_on\_processor\_set$) do
3:   Let $u_k$ be a turned off state;
4:   Invoke the HEFT algorithm on all turned-on processors
     to obtain the schedule length $SL_k(G)$ if $u_k$ is turned off;
5:   if ($SL_k(G) <= D(G)$) then
6:     Invoke the EES algorithm on all turned-on processors
       to obtain the dynamic energy consumption $E_{k,\text{d}}(G)$;
7:     Calculate the static energy consumption $E_{k,\text{s}}(G)$ on all
       turned-on processors using Eq. (3);
8:     Calculate the total energy consumption $E_{k,\text{total}}(G)$
       Eq. (16);
9:   end if
10: end for
11: Put all the processors into the active processor set
    $turned\_on\_processor\_set$ according to an ascending
    of $E_{k,\text{total}}(G)$;
12: while ($turned\_on\_processor\_set$ is not null) do
13:   Select the processor $u_{\min}$ with the minimum $E_{\min,\text{total}}(G)$
      from $turned\_on\_processor\_set$;
14:   Assume that $u_{\min}$ is turned off;
15:   Invoke the HEFT algorithm on all turned-on processors
      to obtain the schedule length $SL_{\min}(G)$ if $u_{\min}$ is turned
      off.
16:   if ($SL_{\min}(G) <= D(G)$) then
17:     Invoke the EES algorithm on all turned-on processors
        to obtain the dynamic energy consumption $E_{\text{d}}(G)$;
18:     Calculate the static energy consumption $E_{\text{s}}(G)$ on all
        turned-on processors using Eq. (3);
19:     Calculate the total energy consumption $E_{\text{total}}(G)$
        Eq. (7);
20:     Turn off $u_{\min}$ and remove it from
        $turned\_on\_processor\_set$;
21:   end if
22: end while
```

5 Performance Evaluation, Experimental Results, and Discussion

5.1 Performance Evaluation Metrics

The performance metrics selected for comparison are as listed follows for a given application:

- Total energy consumption
- Computation time of task assignment
- Number of turned-on processors

We select the state-of-the-art algorithm i.e., the DEWTS [5] for comparison in the experiments. We also present the results of using the standard HEFT algorithm [2]. Considering that the obtained schedule lengths using the DEWTS, EPM, and QEPM algorithms are equal to the given deadlines, we no longer provide the schedule lengths.

The values of the processor and application parameters are as follows [36]: 10 h $\leqslant w_{i,k} \leqslant$ 100 h, 10 h $\leqslant c_{i,j} \leqslant$ 100 h, 0.1 $\leqslant P_{k,\text{s}} \leqslant$ 0.5, 0.03 $\leqslant P_{k,\text{ind}} \leqslant$ 0.07, 0.8 $\leqslant C_{k,\text{ef}} \leqslant$ 1.2, 2.5 $\leqslant m_k \leqslant$ 3.0, and $f_{k,\max} = 1$ GHz. All frequencies are discrete, and the precision is 0.01 GHz. All frequencies are discrete, with a precision of 0.01 GHz. All parallel applications are executed on a simulated multiprocessor system 64 heterogeneous processors by creating 64 processor objects based on known parameter values using Java on a standard desktop computer with 2.6 GHz Intel CPU and 4 GB memory.

Real parallel applications with precedence constrained tasks, such as fast Fourier transform, Diamond graph, and Gaussian elimination applications, are widely used in high-performance cloud computing systems [2], [4], [20]. To validate the effectiveness and validity of the proposed

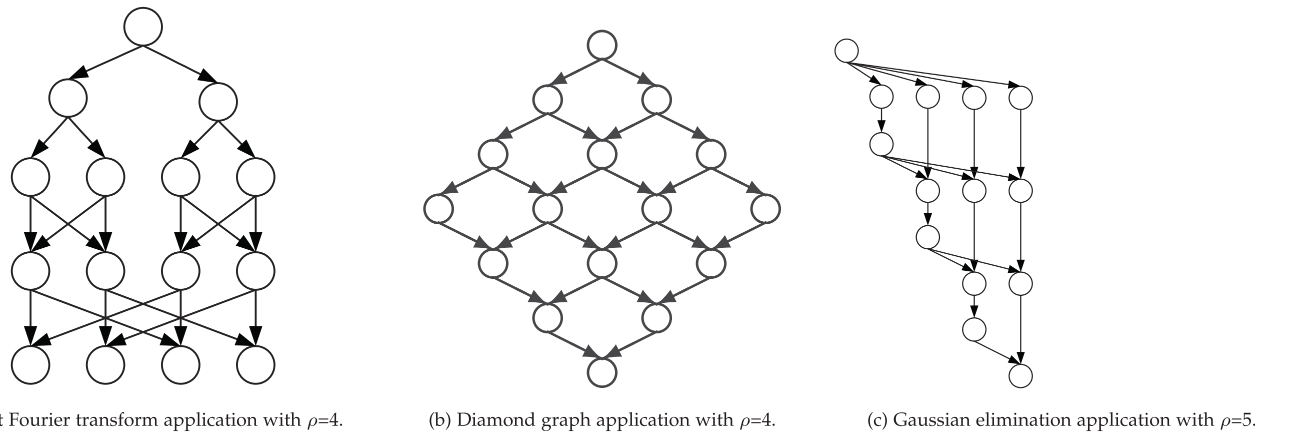

(a) Fast Fourier transform application with ρ=4. (b) Diamond graph application with ρ=4. (c) Gaussian elimination application with ρ=5.

Fig. 7. Example of real parallel applications.

algorithm, we use these real parallel applications to compare the results of the four algorithms.

The structures of these applications are described as follows:

(1) Fast Fourier transform application. A new parameter ρ is used as the size of the fast Fourier transform parallel application, and the total number of tasks should be $|N| = (2 \times \rho - 1) + \rho \times \log_2{}^{\rho}$ [2], [3], where we assume that $\rho = 2^y$ for some integer y. Notably, ρ exit tasks exist in the fast Fourier transform parallel application with the size of ρ.

(2) Diamond graph application. The new parameter ρ is used as the size of the Diamond graph application, and the total number of tasks is $|N| = \rho^2$ [20]. Fig. 7c shows an example of the Diamond graph application with $\rho = 4$.

(3) Gaussian elimination application. A new parameter ρ is used as the matrix size of the Gaussian elimination application and the total number of tasks should be [2], [3] $|N| = \frac{\rho^2+\rho-2}{2}$. Fig. 7b shows an example of the Gaussian elimination application with $\rho = 5$.

Fourier transform is a high-parallelism application, whereas Diamond graph and Gaussian elimination are low-parallelism applications because Fast Fourier transform can generate much shorter schedule length than Diamond graph and Gaussian elimination in an approximately equal scale, as shown in Table 6. The readers can refer to [37] with regard to the parallelism degree of a DAG-based parallel application.

All the results in experiments are obtained by executing one run of the algorithms for one application. Many applications with the same parameter values and scales are tested and the relatively stable results in the same experiment are shown. In other words, all experiments are repeatable and do not affect the consistency of the results.

TABLE 6
Average Schedule Lengths (Unit: h) of Applications Using HEFT

Application	Task number	Average schedule lengths
Fast Fourier transform	2,559	1,198
Diamond graph	2,555	7,150
Gaussian elimination	2,601	7,458

5.2 Fast Fourier Transform Application

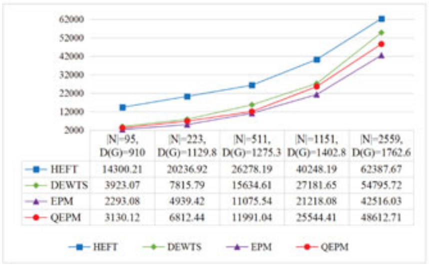
Experiment 1. This experiment is conducted to compare the energy consumption values, computation time, and turned-on processor number of fast Fourier transform applications for varying numbers of tasks. We limit $D(G)$ to $D(G) = LB(G) \times 1.4$. ρ is changed from 16 to 256, that is, the numbers of tasks vary from 95 (small scale) to 2,559 (large scale).

Fig. 8a shows the energy consumption values of fast Fourier transform applications for varying numbers of tasks by using all the algorithms. HEFT is the standard algorithm in all the experiments. Among the three processor merging algorithms, the EPM algorithm generates the minimum energy consumption values, followed by QEPM and DEWTS at all scales. The average energy consumption reduction per task for EPM and QEPM compared with DEWTS in the small scale ($|N| = 96$) is 16.98 and 8.26 GWh, respectively, whereas that in the large scale ($|N| = 2{,}559$) is 8.26 and 2.42 GWh, respectively. These results indicate that the proposed energy-aware processor merging algorithms can save more energy consumption than the sate-of-the DEWTS algorithm at different task scales.

Fig. 8b shows the computation time of fast Fourier transform applications for varying numbers of tasks using the DEWTS, EPM, and QEPM algorithms. The values show that the computation time using EPM is 30-70 times of that using DEWTS. The computation time using QEPM is only 1-2.6 times that of using DEWTS. Specifically, the computation time is large for EPM, whereas QEPM can reduce the computation time to an acceptable and low value.

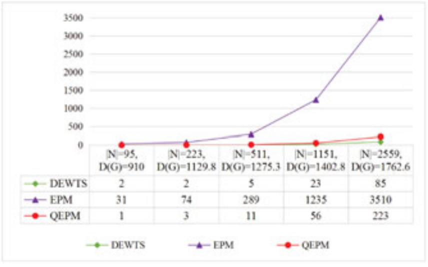
The results of Figs. 8a and 8b show that EPM and QEPM algorithms can save more energy consumption than the sate-of-the-art DEWTS algorithm. Specifically, the EPM algorithm is more energy efficient than the QEPM algorithm, but it is time consuming for large-scale parallel applications.

Fig. 8c shows the number of turned-on processors of fast Fourier transform applications for varying numbers of tasks using the HEFT, DEWTS, EPM, and QEPM algorithms. The standard HEFT algorithm has a total processor number of 64. Although the EPM algorithm generates the minimum energy consumption values compared with the DEWTS and QEPM algorithms (Fig. 8a), the turned-on processor number is not the smallest. On the contrary, EPM has more turned-on processors than DEWTS and QEPM. These results indicate that (1) minimizing the turned-on processor number



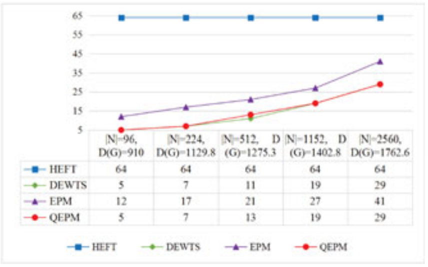
	$\lvert N\rvert$=95, D(G)=910	$\lvert N\rvert$=223, D(G)=1129.8	$\lvert N\rvert$=511, D(G)=1275.3	$\lvert N\rvert$=1151, D(G)=1402.8	$\lvert N\rvert$=2559, D(G)=1762.6
HEFT	14300.21	20236.92	26278.19	40248.19	62387.67
DEWTS	3923.07	7815.79	15634.61	27181.65	54795.72
EPM	2293.08	4939.42	11075.54	21218.08	42516.03
QEPM	3130.12	6812.44	11991.04	25544.41	48612.71

(a) Energy consumptions (unit: GWh).



	$\lvert N\rvert$=95, D(G)=910	$\lvert N\rvert$=223, D(G)=1129.8	$\lvert N\rvert$=511, D(G)=1275.3	$\lvert N\rvert$=1151, D(G)=1402.8	$\lvert N\rvert$=2559, D(G)=1762.6
DEWTS	2	2	5	23	85
EPM	31	74	289	1235	3510
QEPM	1	3	11	56	223

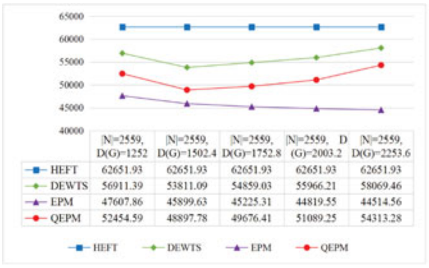
(b) Computation time values (unit: s).



	$\lvert N\rvert$=96, D(G)=910	$\lvert N\rvert$=224, D(G)=1129.8	$\lvert N\rvert$=512, D (G)=1275.3	$\lvert N\rvert$=1152, D (G)=1402.8	$\lvert N\rvert$=2560, D(G)=1762.6
HEFT	64	64	64	64	64
DEWTS	5	7	11	19	29
EPM	12	17	21	27	41
QEPM	5	7	13	19	29

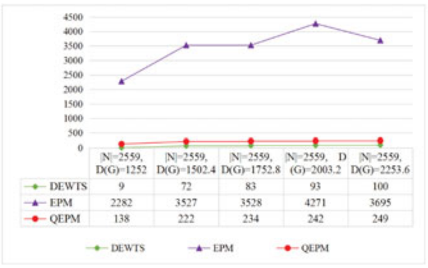
(c) Turned-on processor numbers.

Fig. 8. Results of fast Fourier transform application with the deadline constraint $D(G) = LB(G) \times 1.4$ for varying numbers of tasks (Experiment 1).



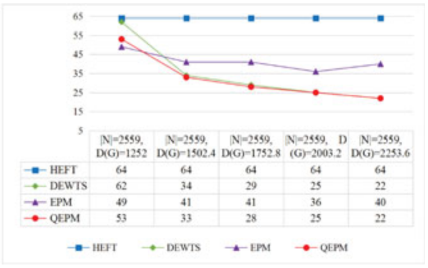
	$\lvert N\rvert$=2559, D(G)=1252	$\lvert N\rvert$=2559, D(G)=1502.4	$\lvert N\rvert$=2559, D(G)=1752.8	$\lvert N\rvert$=2559, D (G)=2003.2	$\lvert N\rvert$=2559, D(G)=2253.6
HEFT	62651.93	62651.93	62651.93	62651.93	62651.93
DEWTS	56911.39	53811.09	54859.03	55966.21	58069.46
EPM	47607.86	45899.63	45225.31	44819.55	44514.56
QEPM	52454.59	48897.78	49676.41	51089.25	54313.28

(a) Energy consumptions (unit: GWh).



	$\lvert N\rvert$=2559, D(G)=1252	$\lvert N\rvert$=2559, D(G)=1502.4	$\lvert N\rvert$=2559, D(G)=1752.8	$\lvert N\rvert$=2559, D (G)=2003.2	$\lvert N\rvert$=2559, D(G)=2253.6
DEWTS	9	72	83	93	100
EPM	2282	3527	3528	4271	3695
QEPM	138	222	234	242	249

(b) Computation time values (unit: s).



	$\lvert N\rvert$=2559, D(G)=1252	$\lvert N\rvert$=2559, D(G)=1502.4	$\lvert N\rvert$=2559, D(G)=1752.8	$\lvert N\rvert$=2559, D (G)=2003.2	$\lvert N\rvert$=2559, D(G)=2253.6
HEFT	64	64	64	64	64
DEWTS	62	34	29	25	22
EPM	49	41	41	36	40
QEPM	53	33	28	25	22

(c) Turned-on processor numbers.

Fig. 9. Results of fast Fourier transform application with $\rho = 256$ (i.e., $|N| = 2{,}559$) for varying deadlines (time unit: h) (Experiment 2).

does not mean minimizing the total energy consumption of the application and (2) turning off processors while satisfying the deadline constraint of the application is not an effective method; the most effective processors to be turned off need to be determined so that a reasonable tradeoff between DVFS and turned-on processor number can be achieved.

Experiment 2. To observe the performance on different deadlines, an experiment is conducted to compare the final energy consumption values of the fast Fourier transform application for varying deadline constraints. We limit the size of the application to $\rho = 256$ (i.e., $|N| = 2{,}559$) and change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.8$ with $LB(G) \times 0.2$ increments, where $LB(G) = 1{,}252$ is calculated using the HEFT algorithm.

Fig. 9a shows the energy consumption of the fast Fourier transform application with $\rho = 256$ (i.e., $|N| = 2{,}559$) for varying deadlines by using all the algorithms. Among the three processor merging algorithms, the EPM algorithm generates the minimum energy consumption values, followed by QEPM and DEWTS in all scales. Another important result is that with the increase in deadline (except for $D(G) = LB(G)$), the energy consumption values increase gradually using the DEWTS and QEPM algorithms, whereas these values decrease using the EPM algorithm. When $D(G) = LB(G) \times 1.8$, the average energy consumption reduction per task for EPM and QEPM compared with DEWTS is 5.295 and 1.467 GWh, respectively. That is, the EPM and QEPM algorithms (especially EPM) are more energy efficient for the fast Fourier transform application than the DEWTS algorithm. These results indicate that turning off the processors with a small task number or low energy utilization does not mean that energy consumption could be reduced; both EPM and QEPM directly consider the energy-aware purpose, so they are useful for energy-efficient designs.

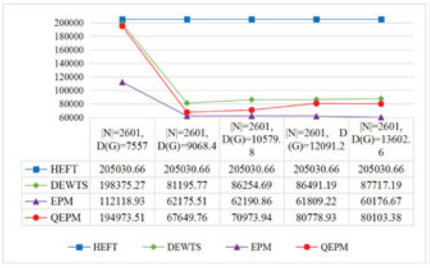
Fig. 9b shows the computation time of the fast Fourier transform application with $\rho = 256$ (i.e., $|N| = 2{,}559$) for varying deadlines by using the DEWTS, EPM, and QEPM algorithms. An obvious result is that EPM is much more time consuming than DEWTS and QEPM. With the increase in deadline, the computation time for EPM also increases gradually from 2,282 s to 4,271 s in general, whereas that for DEWTS (about 8-100 s) and QEPM (about 128-250 s) is relatively stable.

Fig. 9c shows the turned-on processor number of the fast Fourier transform application with $\rho = 256$ (i.e., $|N| = 2{,}559$) for varying deadlines by using all the algorithms. Except for $D(G) = LB(G)$, the EPM algorithm turns on more processors (7-18) than both DEWTS and QEPM. These results imply that turning off more processors may not benefit more energy reduction. Thus, how to select processors for turning off should be considered carefully.

5.3 Diamond Graph Application

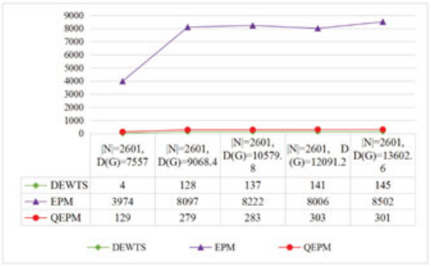
Experiment 3. This experiment is conducted to compare the final energy consumption values of the Diamond graph application for varying deadline constraints. We limit the size of the application to $\rho = 51$ (i.e., $|N| = 2{,}601$, which is approximately equal to the number of tasks of the fast Fourier transform application in *Experiment* 2) and change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.8$ with $LB(G) \times 0.2$ increments, where $LB(G) = 7{,}557$ is calculated using the HEFT algorithm.

Fig. 10a shows the energy consumption of the Diamond graph application with $\rho = 51$ (i.e., $|N| = 2{,}601$) for varying deadlines by using all the algorithms. Among the three processor merging algorithms, the EPM algorithm generates the minimum energy consumption values, followed by QEPM and DEWTS at all scales. The main difference between Diamond graph (Fig. 10a) and fast Fourier transform (Fig. 9a) applications is that the reduced energy consumption for the fast Fourier transform application in all cases is relatively smooth, whereas that for the Diamond graph application in $D(G) = LB(G)$ is small but large in other cases.



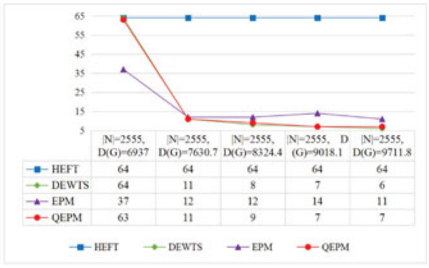
	|N|=2601, D(G)=7557	|N|=2601, D(G)=9068.4	|N|=2601, D(G)=10579.8	|N|=2601, D(G)=12091.2	|N|=2601, D(G)=13602.6
HEFT	205030.66	205030.66	205030.66	205030.66	205030.66
DEWTS	198375.27	81195.77	86254.69	86491.19	87717.19
EPM	112118.93	62175.51	62190.86	61809.22	60176.67
QEPM	194973.51	67649.76	70973.94	80778.93	80103.38

(a) Energy consumptions (unit: GWh).



	|N|=2601, D(G)=7557	|N|=2601, D(G)=9068.4	|N|=2601, D(G)=10579.8	|N|=2601, D(G)=12091.2	|N|=2601, D(G)=13602.6
DEWTS	4	128	137	141	145
EPM	3974	8097	8222	8006	8502
QEPM	129	279	283	303	301

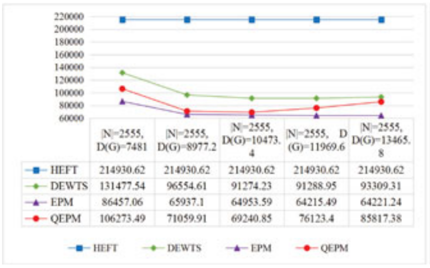
(b) Computation time values (unit: s).



	|N|=2555, D(G)=6937	|N|=2555, D(G)=7630.7	|N|=2555, D(G)=8324.4	|N|=2555, D(G)=9018.1	|N|=2555, D(G)=9711.8
HEFT	64	64	64	64	64
DEWTS	64	11	8	7	6
EPM	37	12	12	14	11
QEPM	63	11	9	7	7

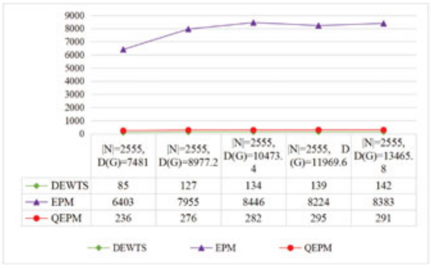
(c) Turned-on processor numbers.

Fig. 10. Results of Diamond graph application with $\rho = 51$ (i.e., $|N| = 2{,}601$) for varying deadlines (time unit: h) (Experiment 3).



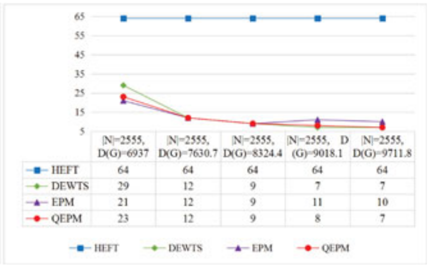
	|N|=2555, D(G)=7481	|N|=2555, D(G)=8977.2	|N|=2555, D(G)=10473.4	|N|=2555, D(G)=11969.6	|N|=2555, D(G)=13465.8
HEFT	214930.62	214930.62	214930.62	214930.62	214930.62
DEWTS	131477.54	96554.61	91274.23	91288.95	93309.31
EPM	86457.06	65937.1	64953.59	64215.49	64221.24
QEPM	106273.49	71059.91	69240.85	76123.4	85817.38

(a) Energy consumptions (unit: GWh).



	|N|=2555, D(G)=7481	|N|=2555, D(G)=8977.2	|N|=2555, D(G)=10473.4	|N|=2555, D(G)=11969.6	|N|=2555, D(G)=13465.8
DEWTS	85	127	134	139	142
EPM	6403	7955	8446	8224	8383
QEPM	236	276	282	295	291

(b) Computation time values (unit: s).



	|N|=2555, D(G)=6937	|N|=2555, D(G)=7630.7	|N|=2555, D(G)=8324.4	|N|=2555, D(G)=9018.1	|N|=2555, D(G)=9711.8
HEFT	64	64	64	64	64
DEWTS	29	12	9	7	7
EPM	21	12	9	11	10
QEPM	23	12	9	8	7

(c) Turned-on processor numbers.

Fig. 11. Results of Gaussian elimination application with $\rho = 71$ (i.e., $|N| = 2{,}555$) for varying deadlines (time unit: h) (Experiment 4).

Fig. 10b shows the computation time of the Diamond graph application with $\rho = 51$ (i.e., $|N| = 2{,}601$) for varying deadlines by using the DEWTS, EPM, and QEPM algorithms. As expected, the regular pattern of Fig. 10b is similar to that of Fig. 9b. However, the Diamond graph application needs more computation time (about 8,000 s) than the fast Fourier transform application (nearly 4,000 s) to calculate the energy consumption values, although they are on the same scale in terms of number of tasks.

Fig. 10c shows the turned-on processor number of Diamond graph application with $\rho = 51$ (i.e., $|N| = 2{,}601$) for varying deadlines by using all the algorithms. The regular pattern of Fig. 10c is similar to that of Fig. 9c, but the average turned-on processor number for the Diamond graph application is about 12, whereas that for the fast Fourier transform is about 40 when using the EPM algorithm. The reason is that the Diamond graph application has lower parallelism than the fast Fourier transform application, such that many processors are not sufficiently employed. That is, the DEWTS algorithm may be suitable for such an application, but it is still less energy efficient than EPM and QEPM algorithms (Fig. 10a).

5.4 Gaussian Elimination Application

Experiment 4. This experiment is conducted to compare the final energy consumption values of the Gaussian elimination application for varying deadline constraints. We limit the size of the application to $\rho = 71$ (i.e., $|N| = 2{,}555$) and change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.8$ with $LB(G) \times 0.2$ increments, where $LB(G) = 7{,}481$ is calculated using the HEFT algorithm.

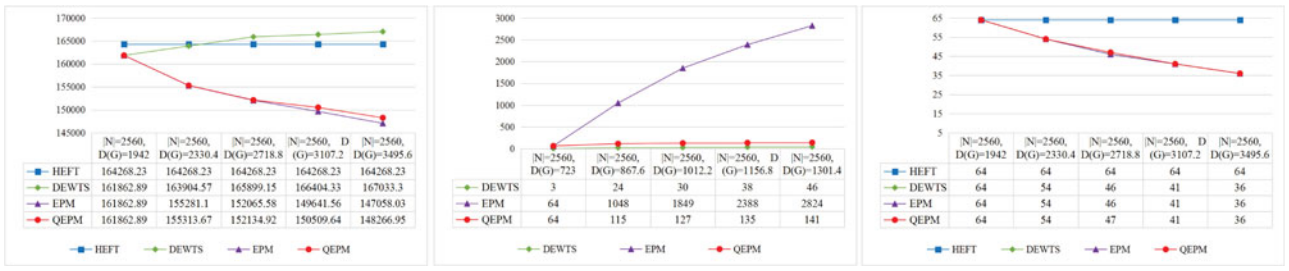
Fig. 11a shows the energy consumption of the Gaussian elimination application with $\rho = 71$ (i.e., $|N| = 2{,}555$) for varying deadlines by using all the algorithms. Among the three processor merging algorithms, similar to the results in Figs. 9a and 10a, the EPM algorithm generates the minimum energy consumption values, followed by QEPM and DEWTS at all scales. The main difference between Gaussian elimination (Fig. 11a) and Diamond graph (Fig. 10a) applications is that when $D(G) = LB(G)$, the reduced energy consumption for the Gaussian elimination is large, whereas that for the Diamond graph application is small when the DEWTS and QEPM algorithms are used.

Fig. 11b shows the computation time of the Gaussian elimination application with $\rho = 71$ (i.e., $|N| = 2{,}555$) for varying deadlines by using the DEWTS, EPM, and QEPM algorithms. The regular pattern of Fig. 11b is similar to that of Figs. 9b and 10b. The Gaussian elimination and Diamond graph applications need the same computation time under the same conditions.

Fig. 11c shows the turned-on processor number of the Gaussian elimination application with $\rho = 256$ (i.e., $|N| = 2{,}560$) for varying deadlines by using all the algorithms. The regular pattern of Fig. 11c is similar to that of Figs. 9c and 10c. Compared with Figs. 9c, 10c, and 11c, maximum processor numbers are turned-off for the Gaussian elimination application (Fig. 11c), followed by Diamond graph (Fig. 10c) and fast Fourier transform (Fig. 9c) applications. These results indicate that the Gaussian elimination application has the lowest parallelism, followed by Diamond graph and fast Fourier transform; many processors are not sufficiently employed.

5.5 Randomly Generated Parallel Application

To extensively demonstrate the performance benefits of the proposed algorithms, we consider randomly generated parallel applications using the task graph generator in [38]. As the objective cloud computing platform consists of heterogeneous processors, heterogeneity degrees may also affect the energy consumption of application. Heterogeneity degree is easy to be implemented for the task graph generator by adjusting the heterogeneity factor values [39]. In this study, randomly generated parallel applications are generated depending on the following parameters: Average



	|N|=2560, D(G)=1942	|N|=2560, D(G)=2330.4	|N|=2560, D(G)=2718.8	|N|=2560, D(G)=3107.2	|N|=2560, D(G)=3495.6
HEFT	164268.23	164268.23	164268.23	164268.23	164268.23
DEWTS	161862.89	163904.57	165899.15	166404.33	167033.3
EPM	161862.89	155281.1	152065.58	149641.56	147058.03
QEPM	161862.89	155313.67	152134.92	150509.64	148266.95

(a) Energy consumptions (unit: GWh).

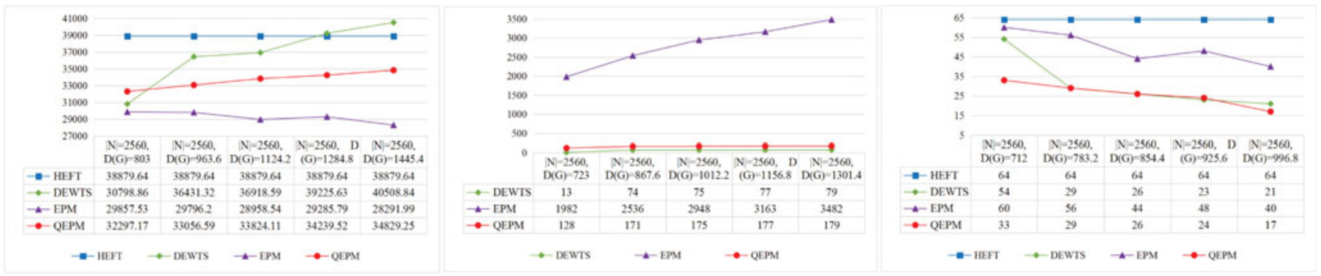
	|N|=2560, D(G)=723	|N|=2560, D(G)=867.6	|N|=2560, D(G)=1012.2	|N|=2560, D(G)=1156.8	|N|=2560, D(G)=1301.4
DEWTS	3	24	30	38	46
EPM	64	1048	1849	2388	2824
QEPM	64	115	127	135	141

(b) Computation time values (unit: s).

	|N|=2560, D(G)=1942	|N|=2560, D(G)=2330.4	|N|=2560, D(G)=2718.8	|N|=2560, D(G)=3107.2	|N|=2560, D(G)=3495.6
HEFT	64	64	64	64	64
DEWTS	64	54	46	41	36
EPM	64	54	46	41	36
QEPM	64	54	47	41	36

(c) Turned-on processor numbers.

Fig. 12. Results of large-scale low-heterogeneity randomly generated parallel application with $|N| = 2{,}560$ for varying deadlines (time unit: h) (Experiment 5).



	|N|=2560, D(G)=803	|N|=2560, D(G)=963.6	|N|=2560, D(G)=1124.2	|N|=2560, D(G)=1284.8	|N|=2560, D(G)=1445.4
HEFT	38879.64	38879.64	38879.64	38879.64	38879.64
DEWTS	30798.86	36431.32	36918.59	39225.63	40508.84
EPM	29857.53	29796.2	28958.54	29285.79	28291.99
QEPM	32297.17	33056.59	33824.11	34239.52	34829.25

(a) Energy consumptions (unit: GWh).

	|N|=2560, D(G)=723	|N|=2560, D(G)=867.6	|N|=2560, D(G)=1012.2	|N|=2560, D(G)=1156.8	|N|=2560, D(G)=1301.4
DEWTS	13	74	75	77	79
EPM	1982	2536	2948	3163	3482
QEPM	128	171	175	177	179

(b) Computation time values (unit: s).

	|N|=2560, D(G)=712	|N|=2560, D(G)=783.2	|N|=2560, D(G)=854.4	|N|=2560, D(G)=925.6	|N|=2560, D(G)=996.8
HEFT	64	64	64	64	64
DEWTS	54	29	26	23	21
EPM	60	56	44	48	40
QEPM	33	29	26	24	17

(c) Turned-on processor numbers.

Fig. 13. Results of large-scale high-heterogeneity randomly generated parallel application with $|N| = 2{,}560$ for varying deadlines (time unit: h) (Experiment 6).

computation time is 50 h, communication to computation ratio (CCR) is 1, and shape parameter is 1. The heterogeneity degree (factor) values belong to the scope of (0,1] in the task graph generator, where 0 and 1 represent the lowest and highest heterogeneity factors, respectively [39]. Without loss of generality, we use large-scale randomly generated parallel application with 2,560 tasks, which are approximately equal to those of fast Fourier transform, Diamond graph, and Gaussian elimination applications in Experiments 2-4.

Experiment 5. This experiment compares the final energy consumption values of a large-scale low-heterogeneity (with the heterogeneity factor 0.1) randomly generated parallel application for varying deadline constraints. We limit the size of the application to $|N| = 2{,}560$ and change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.8$ with $LB(G) \times 0.2$ increments, where $LB(G) = 1{,}942$ is calculated using the HEFT algorithm. The results are shown in Fig. 12.

Experiment 6. This experiment compares the final energy consumption values of a large-scale high-heterogeneity (with the heterogeneity factor 0.9) randomly generated parallel application for varying deadline constraints. We still limit the size of the application to $|N| = 2{,}560$ and change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.8$ with $LB(G) \times 0.2$ increments, where $LB(G) = 803$ is calculated using the HEFT algorithm. The results are shown in Fig. 13.

The results of both low-heterogeneity application in Experiment 5 and high-heterogeneity application in Experiment 6 still show that EPM generates the minimum energy consumptions, followed by QEPM and DEWTS among them, whereas EPM is the most time-consuming among them. The main differences between the results of low-heterogeneity and high-heterogeneity applications are follows:

(1) Low-heterogeneity application has more than 10 times higher energy consumption than high-heterogeneity applications when using the proposed algorithms. In other words, the application with high heterogeneity may have higher potential to save energy.

(2) The energy consumptions decrease with the increment of deadline for low-heterogeneity application (Figs. 12a), whereas the opposite law occurs for high-heterogeneity application using EPM and QEPM algorithms (Figs. 13a).

(3) The energy consumptions increase with the increment of deadline for both low-heterogeneity and high-heterogeneity applications using DETWS (Figs. 12a and 13a). In addition, the energy consumptions generated by DETWS exceeds HEFT when $D(G)$ exceeds $LB(G) \times 1.2$ for low-heterogeneity application in Fig. 12a and $LB(G) \times 1.8$ for high-heterogeneity application in Fig. 13a. That is, DETWS is not energy-aware for both low-heterogeneity and high-heterogeneity applications.

(4) QEPM and EPM are both energy-efficient as shown in Figs. 12a and 13a. In addition, EPM has the most turned-on processors in this case shown in Fig. 13c. The results further indicate that turning off as many processors as possible does not necessarily lead to the minimum energy consumption.

5.6 Summary of Experiments

The following observations and conclusions for deadline constrained parallel applications can be obtained based on the experimental results.

(1) Turning off the processors with a small number of tasks or processors with low energy utilization does not necessarily lead to the minimization of energy consumption. Both EPM and QEPM consider the energy consumption explicitly when choosing processors to turn off, which results in higher energy efficiency.

(2) Turning off as many processors as possible does not necessarily lead to the minimization of total energy consumption of the application. The most effective processors to be turned off need to be determined by the tradeoff between DVFS and the number of active processors.

(3) According to the analysis of the number of active processors, The low-parallelism Gaussian elimination and Diamond graph applications have lower turned-on processors than the high-parallelism fast Fourier Transform application. The lower the parallelism of the application is, the more the turned-off processors could be.

(4) The proposed energy-aware processor merging algorithms, EPM and QEPM, can save more energy than the state-of-the-art DEWTS algorithm at all scales, parallelism, and heterogeneity degrees.

(5) If the computation time is not the concern, the EPM can be utilized to minimize the energy consumption; otherwise, the QEPM is an alternative to reduce energy within reasonable computation time.

(6) For application with high heterogeneity, the EPM is preferred to QEPM in terms of saving energy consumption.

6 Conclusion

In this study, two energy-aware processor merging algorithms EPM and QEPM were proposed for real-time parallel applications in a heterogeneous cloud computing system. The EPM algorithm tried to minimize the energy consumption while satisfying the deadline constraint of the application by turning off the most effective processor from the energy saving perspective. To decrease the computation complexity of EPM, a QEPM algorithm was also introduced to deal with large-scale parallel applications in an acceptable amount of time. Our intensive experiments on several real parallel applications and randomly generated parallel applications validated that both EPM and QEPM can save more energy than the state-of-the-art DEWTS algorithm, and the QEPM algorithm can complete computation within reasonable time. Moreover, an energy-aware processor merging guide for deadline constrained parallel applications was presented based on the experimental results. We believe that the proposed algorithms can effectively facilitate an energy-aware design for deadline constrained parallel applications in heterogeneous cloud computing systems.

Supplement Material

The web page http://esnl.hnu.edu.cn/index.php/tsusc/ publishes the experimental codes of the paper.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers whose constructive comments have helped to improve the manuscript. This work was partially supported by the National Key Research and Development Plan of China under Grant No. 2016YFB0200405 and 2012AA01A301-01, the National Natural Science Foundation of China with Grant Nos. 61672217, 61432005, 61379115, 61402170, 61370097, 61502162 and 61502405, the CERNET Innovation Project under Grant No. NGII20161003, and the China Postdoctoral Science Foundation under Grant No. 2016M592422.

References

[1] K. Li, "Power and performance management for parallel computations in clouds and data centers," *J. Comput. Syst. Sci.*, vol. 82, no. 2, pp. 174–190, Mar. 2016.

[2] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[3] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.

[4] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang, and X. Huang, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *Proc. 12th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 781–786.

[5] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment," *J Grid Comput.*, vol. 14, no. 1, pp. 55–74, Mar. 2016.

[6] G. Xie, L. Liu, L. Yang, and R. Li, "Scheduling trade-off of dynamic multiple parallel workflows on heterogeneous distributed computing systems," *Concurrency Comput.-Parctice Exp.*, vol. 29, no. 8, pp. 1–18, Jan. 2017.

[7] J. M. Batalla, M. Kantor, C. X. Mavromoustakis, G. Skourletopoulos, and G. Mastorakis, "A novel methodology for efficient throughput evaluation in virtualized routers," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 6899–6905.

[8] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Transact. Des. Autom. Electron. Syst.*, vol. 18, no. 2, pp. 99–109, Mar. 2013.

[9] D. Zhu and H. Aydin, "Reliability-aware energy management for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 58, no. 10, pp. 1382–1397, Oct. 2009.

[10] B. Zhao, H. Aydin, and D. Zhu, "On maximizing reliability of real-time embedded applications under hard energy constraint," *IEEE Trans. Ind. Inf.*, vol. 6, no. 3, pp. 316–328, Aug. 2010.

[11] X. Xiao, G. Xie, R. Li, and K. Li, "minimizing schedule length of energy consumption constrained parallel applications on heterogeneous distributed systems," in *Proc. 14th IEEE Int. Symp. Parallel Distrib. Process. Appl.*, 2016, pp. 1471–1476.

[12] G. Xie, X. Xiao, R. Li, and K. Li, "Schedule length minimization of parallel applications with energy consumption constraints using heuristics on heterogeneous distributed systems," in *Proc. Concurrency Comput.-Parctice Exp.*, Nov. 2016, pp. 1–10.

[13] A. Bourdena, C. X. Mavromoustakis, G. Kormentzas, E. Pallis, G. Mastorakis, and M. B. Yassein, "A resource intensive traffic-aware scheme using energy-aware routing in cognitive radio networks," *Future Gen. Comput. Syst.*, vol. 39, pp. 16–28, Oct. 2014.

[14] C. D. Dimitriou, C. X. Mavromoustakis, G. Mastorakis, and E. Pallis, "On the performance response of delay-bounded energy-aware bandwidth allocation scheme in wireless networks," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2013, pp. 631–636.

[15] K. Papanikolaou, C. X. Mavromoustakis, G. Mastorakis, A. Bourdena, and C. Dobre, "Energy consumption optimization using social interaction in the mobile cloud," in *Proc. Int. Conf. Mobile Netw. Manage.*, 2014, pp. 431–445.

[16] C. X. Mavromoustakis, A. Bourdena, G. Mastorakis, E. Pallis, and G. Kormentzas, "An energy-aware scheme for efficient spectrum utilization in a 5G mobile cognitive radio network architecture," *Telecommun. Syst.*, vol. 59, no. 1, pp. 63–75, May 2015.

[17] C. X. Mavromoustakis, G. Mastorakis, A. Bourdena, and E. Pallis, "Energy efficient resource sharing using a trafficoriented routing scheme for cognitive radio networks," *IET Netw.*, vol. 3, no. 1, pp. 54–63, Mar. 2014.

[18] J. A. Dias, J. J. Rodrigues, F. Xia, and C. X. Mavromoustakis, "A cooperative watchdog system to detect misbehavior nodes in vehicular delay-tolerant networks," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, pp. 7929–7937, Dec. 2015.

[19] G. Zeng, Y. Matsubara, H. Tomiyama, and H. Takada, "Energy-aware task migration for multiprocessor real-time systems," *Future Gen. Comput. Syst.*, vol. 56, pp. 220–228, Mar. 2016.

[20] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1668–1681, Dec. 2012.

[21] J. Barnett, "Dynamic task-level voltage scheduling optimizations," *IEEE Trans. Comput.*, vol. 54, no. 5, pp. 508–520, May 2005.

[22] D. P. Bunde, "Power-aware scheduling for makespan and flow," in *Proc. 18th Annu. ACM Symp. Parallelism Algorithms Archit.*, 2006, pp. 190–196.

[23] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, Jun. 1975.

[24] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, p. 50, May 2015.

[25] A. S. Wu, H. Yu, S. Jin, K.-C. Lin, and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, pp. 824–834, Sep. 2004.

[26] A. Swiecicka, F. Seredynski, and A. Y. Zomaya, "Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 3, pp. 253–262, Mar. 2006.

[27] M. H. Kashani and M. Jahanshahi, "Using simulated annealing for task scheduling in distributed systems," in *Proc. Int. Conf. Comput. Intell. Modelling Simul.*, 2009, pp. 265–269.

[28] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 6, pp. 911–924, Jun. 2010.

[29] Y. Xu, K. Li, L. He, L. Zhang, and K. Li, "A hybrid chemical reaction optimization scheme for task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3208–3222, Dec. 2015.

[30] S. Chen, Z. Li, B. Yang, and G. Rudolph, "Quantum-inspired hyper-heuristics for energy-aware scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 6, pp. 1796–1810, Jun. 2016.

[31] G. Xie, R. Li, and K. Li, "Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems," *J. Parallel Distrib. Comput.*, vol. 83, pp. 1–12, Sep. 2015.

[32] Z. Zong, A. Manzanares, X. Ruan, and X. Qin, "EAD and PEBD: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," *IEEE Trans. Comput.*, vol. 60, no. 3, pp. 360–374, Mar. 2011.

[33] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011.

[34] P. D. Langen and B. Juurlink, "Leakage-aware multiprocessor scheduling," *J. Signal Process. Syst.*, vol. 57, no. 1, pp. 73–88, Oct. 2009.

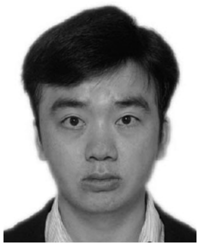
[35] T. Thanavanich and P. Uthayopas, "Efficient energy aware task scheduling for parallel workflow tasks on hybrids cloud environment," in *Proc. Int. Comput. Sci. Eng. Conf.*, 2013, pp. 37–42.

[36] M. W. Convolbo and J. Chou, "Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources," *J. Supercomput.*, vol. 72, no. 3, pp. 985–1012, Mar. 2016.

[37] S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 6, pp. 533–544, Jun. 2003.

[38] (2015. Aug.). [Online]. Available: https://sourceforge.net/projects/taskgraphgen/

[39] G. Xie, et al., "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Trans. Serv. Comput.*, to be published. doi: 10.1109/TSC.2017.2665552.



Guoqi Xie received the PhD degree in computer science and engineering from Hunan University, China, in 2014. He was a postdoctoral researcher at Nagoya University, Japan, from 2014 to 2015. Since 2015, he has been working as a postdoctoral researcher at Hunan University. He has received the best paper award from ISPA 2016. His major interests include embedded and real-time systems, parallel and distributed systems, and software engineering and methodology. He is a member of the IEEE, ACM, and CCF.

Gang Zeng received the PhD degree in information science from Chiba University in 2006. He is an associate professor in the Graduate School of Engineering, Nagoya University. From 2006 to 2010, he was a researcher, and then assistant professor in the Center for Embedded Computing Systems (NCES), the Graduate School of Information Science, Nagoya University. His research interests mainly include power-aware computing and real-time embedded system design. He is a member of the IEEE and IPSJ.

Renfa Li is a professor of computer science and electronic engineering, and the dean of the College of Computer Science and Electronic Engineering, Hunan University, China. He is the director of the Key Laboratory for Embedded and Network Computing of Hunan Province, China. He is also an expert committee member of the National Supercomputing Center in Changsha, China. His major interests include computer architectures, embedded computing systems, cyber-physical systems, and Internet of things. He is a member of the council of CCF, a senior member of the IEEE, and a senior member of the ACM.

Keqin Li is a SUNY Distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things, and cyber-physical systems. He has published more than 475 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.