This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2730876, IEEE Transactions on Parallel and Distributed Systems

Energy-efficient Scheduling Algorithms for Real-time Parallel Applications on Heterogeneous Distributed Embedded Systems

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

Abstract—Energy consumption minimization is one of the primary design requirements for heterogeneous distributed systems. State-of-the-art algorithms are used to study the problem of minimizing the energy consumption of a real-time parallel application with precedence constrained tasks on a heterogeneous distributed system by introducing the concept of latest finish time (LFT) to reclaim the slack time based on the dynamic voltage and frequency scaling (DVFS) energy-efficient design optimization technique. However, the use of DVFS technique alone is insufficient, and the energy consumption reduction is limited because scaling down the frequency is restricted in practice. Furthermore, these studies merely minimize energy consumption through a local energy-efficient scheduling algorithm, such as reducing the energy consumption for each task on the fixed processor, rather than a global energy-efficient scheduling algorithm, such as reducing the energy consumption for each task on different processors. This study solves the problem of minimizing the energy consumption of a real-time parallel application on heterogeneous distributed systems by using the combined non-DVFS and global DVFS-enabled energy-efficient scheduling algorithms. The non-DVFS energy-efficient scheduling (NDES) algorithm is solved by introducing the concept of deadline slacks to reduce the energy consumption while satisfying the deadline constraint. The global DVFS-enabled energy-efficient scheduling (GDES) algorithm is presented by moving the tasks to the processor slacks that generate minimum dynamic energy consumptions. Results of the experiments show that the combined NDES&GDES algorithm can save up to 36.25-55.65% of energy compared with state-of-the-art counterparts under different scales, parallelism, and heterogeneity degrees of parallel applications.

Index Terms—directed acyclic graph (DAG), dynamic voltage and frequency scaling (DVFS), energy-efficient scheduling, heterogeneous distributed embedded systems, real-time

1 Introduction

1.1 Motivation

Heterogeneous distributed systems are increasingly implemented from small-scale embedded platforms, such as laptops, and smartphones [1], to large-scale datacenters, such as grid, cluster, cloud, and service-oriented systems [2]. Energy consumption management is crucial in modern parallel and distributed systems because of the significance of thermal issues and the need for various adaptive management techniques to maximize energy efficiency. Dynamic voltage and frequency scaling (DVFS) achieves energy-efficient scheduling by simultaneously scaling down the supply voltage and frequency of a processor [3]. Deadline is an important design constraint for real-time applications in embedded systems; missing the hard deadlines of these applications is not functional correctly and will cause catastrophic consequences [1], [4]. Therefore, the deadline must be satisfied for a hard real-time application. Parallel applications with precedence constrained tasks, such as fast Fourier transform and Gaussian elimination applications, are increasing in number in heterogeneous distributed systems [2], [3]. There are some typical models describing a parallel application with precedence constrained tasks, such as directed acyclic graph (DAG), hybrid DAG (HDAG), and task interaction graph (TIG), et al. [5]. In this study, a parallel application is represented by a DAG, in which the nodes represent the tasks and the edges represent the communication messages between tasks [2], [3], [6].

The problem of minimizing the energy consumption of a real-time application with precedence constrained tasks has been solved currently [7]. However, these studies mostly focus on homogeneous systems, whereas heterogeneous multiprocessors or multicores continue to scale up and increasingly become the key components of distributed platforms [3]. The same problem for heterogeneous distributed systems has been studied by proposing enhanced energy-efficient scheduling (EES) algorithm to reclaim the slack time for each task on the same processor based on the latest finish time (LFT) [8], [9]; however, this strategy can be further improved for the following reasons:

First, the LFT-based strategy minimizes the energy consumption through the DVFS-enabled energy-efficient scheduling; using DVFS technique alone is insufficient, and the energy consumption reduction is limited because lower frequency than a given threshold may result in more energy consumption [10], [11], [12]. In addition, heterogeneous DVFS-enabled processors may not be supported in some computing systems [13], so it is necessary to propose low energy consumption task scheduling method for non-DVFS environments. Non-DVFS approach can also minimize the energy consumption if used with an effective task scheduling algorithm.

Second, the LFT-based strategy merely minimizes energy consumption through a local energy-efficient scheduling algorithm,

- *Guoqi Xie, Xiongren Xiao, Renfa Li, and Keqin Li are with the College of Computer Science and Electronic Engineering, Hunan University, Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha, Hunan 410082, China. E-mail: xgqman@hnu.edu.cn; xxr@hnu.edu.cn; lirenfa@hnu.edu.cn; lik@newpaltz.edu.*
- *Gang Zeng is with the Graduate School of Engineering, Nagoya University, Aichi 4648603, Japan. E-mail: sogo@ertl.jp.*
- *Keqin Li is also with the Department of Computer Science, State University of New York, New Paltz, New York 12561, USA.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2730876, IEEE Transactions on Parallel and Distributed Systems

which reclaims slack time values on the same processor for each task, rather than a global energy-efficient scheduling algorithm, which reclaims slack time values on different processors for each task. Applying a global energy-efficient scheduling algorithm to minimize energy consumption of distributed parallel application will be more efficient.

Third, considering that the non-DVFS approach can also reduce the energy consumption, it can be used until the energy consumption cannot be dropped down, then the DVFS-enable approach can be employed to further reduce the energy consumption ultimately. In this way, the combined non-DVFS and DVFS-enable approach will be more effective intuitively.

1.2 Our Contributions

In this study, we aim to solve the problem of minimizing the energy consumption of a real-time parallel application with precedence constrained tasks on heterogeneous distributed systems. The contributions of this study are summarized as follows.

(1) We propose the deadline slack algorithm (Algorithm 1) by introducing the concept of deadline slack to effectively assign the task to the processor with the minimum dynamic energy consumption without using DVFS while satisfying its deadline constraint as far as possible.

(2) We propose the non-DVFS energy-efficient scheduling (NDES) algorithm (Algorithm 2) by introducing the concept of variable deadline slack to implement the energy consumption reduction by iteratively calling the deadline slack algorithm (Algorithm 1) to guarantee that the deadline of the application is always satisfied.

(3) We propose the global DVFS-enabled energy-efficient scheduling (GDES) algorithm (Algorithm 3) by moving the tasks to the processor slacks that generate minimum dynamic energy consumptions without violating the precedence constraints among tasks and the deadline constraint of the application.

The rest of this paper is organized as follows. Section 2 reviews related research. Section 3 builds related models and preliminaries. Section 4 proposes the NDES algorithm. Section 5 proposes the GDES algorithm. Section 6 verifies the performance of the presented NDES, GDES, NDES&EES, and NDES&GDES algorithms. Section 7 verifies the NDES&GDES algorithm in real platform. Section 8 concludes this study.

2 Related Work

In [14], the authors presented a survey of energy-aware scheduling algorithms based on DVFS, dynamic power management (DPM), or both, proposed for real-time systems. In [15], the authors presented a survey of energy-cognizant scheduling techniques. In the following, we review the related work of DAG-based energy-efficient scheduling algorithms that are relevant to this study.

DVFS-enabled energy-efficient design techniques and algorithms usually aim at maximizing performance with energy constraint [12] or minimizing energy consumption while satisfying deadline constraint [7] for a DAG-based parallel application. The authors in [3] presented energy-conscious scheduling to implement joint minimization of energy consumption and schedule length of a parallel application on heterogeneous distributed systems; however, they did fails to consider the deadline constraint of the application. Although Ref. [7] studied the problem of energy consumption optimization of a real-time DAG-based application, it focused on homogeneous distributed systems. The authors in [4] presented energy-efficient task assignment for a DAG-based application with guaranteed probability while satisfying deadline constraint for heterogeneous embedded systems; however, they considered a shared memory for processors (i.e., without communication between any two tasks).

The authors in [8] studied the problem of minimizing energy consumption of a real-time parallel application on heterogeneous distributed systems by presenting the EES algorithm, which reclaims slack time for each task on its fixed assigned processor. The authors in [9] solved the same problem by presenting the DVFS-enabled energy-efficient workflow task scheduling (DEWTS) algorithm, which introduces the feature of turning off the relatively inefficient processors to reduce the static energy consumption and realize EES-based slack time reclamation [8]. However, turning off processors is practically unrealistic in most embedded systems, such as laptops, smartphones, automobiles, and avionics. In addition, static energy consumption accounts for only a small part of the total energy consumption in these systems. These state-of-the-art algorithms [8], [9] are limited in that they only minimize energy consumption through DVFS-enabled and local energy-efficient scheduling algorithms but fail to apply the non-DVFS and global energy-efficient scheduling algorithms, thereby limiting the energy efficiency. In this study, we aim to implement the same objective of parallel application by combining the non-DVFS and global DVFS-enabled energy-efficient scheduling algorithms.

3 Models and Preliminaries

Table 1 gives the important notations and their definitions used in this study.

TABLE 1: Important notations in this study.

Notation	Definition
$w_{i,k}$	WCET of the task n_i on the processor $u_k$with the maximum frequency
$c_{i,j}$	WCRT between the tasks n_i and n_j
$D(G)$	Given deadline of the application G
$LB(G)$	Lower bound of the application G
$SL(G)$	Schedule length of the application G
$DS(G)$	Deadline slack of the application G
$VDS(G)$	Variable deadline slack 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 $f_{k,h}$
$E_{\mathrm{d}}(n_i, u_k, f_{k,h})$	Dynamic energy consumption of the task n_i on the processor u_k with the frequency $f_{k,h}$
$E_{\mathrm{s}}(G)$	Static energy consumption of the application G
$E_{\mathrm{d}}(G)$	Dynamic energy consumption of the application G
$E_{\mathrm{total}}(G)$	Total energy consumption of the application G
$AST(n_i)$	Actual start time of the task n_i
$AFT(n_i)$	Actual finish time of the task n_i
$AET(n_i)$	Actual execution time of the task n_i
$LFT(n_i, u_k)$	Latest finish time of the task n_i on the processor u_k

3.1 System Architecture and Application Model

In [16], the system topology is well discussed. This study considers a common distributed embedded architecture where several processors are mounted on the same controller area network (CAN) bus [17], as shown in Fig. 1. Each processor contains a central processing unit (CPU), random-access memory (RAM) and non-volatile memory, and a network interface card. A task executed completely in one processor sends messages to all its successor tasks, which may be located in different processors. For example, task n_1 is executed on processor u_1. It then sends a message $m_{1,2}$ to its successor task n_2 located in u_6 (see Fig. 1). Let $U = \{u_1, u_2, ..., u_{|U|}\}$ represent a set of heterogeneous multiprocessors, where $|U|$ represents the size of set U. For any set X, this study uses $|X|$ to denote its size.

A parallel application running on processors is represented by a DAG G=(N, W, M, C) [2], [3], [8], [9], [12], [18].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2730876, IEEE Transactions on Parallel and Distributed Systems

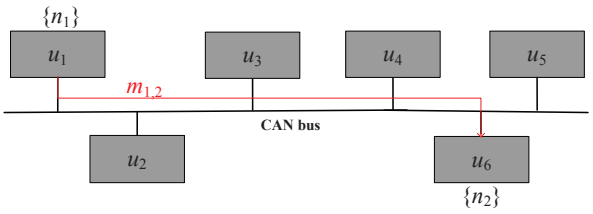


Fig. 1: Heterogeneous distributed embedded system architecture.

(1) N represents a set of nodes in G, and each node $n_i \in N$ represents a task. $pred(n_i)$ 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_{entry}; and the task that has no successor task is denoted as n_{exit}. If a function has multiple n_{entry} or multiple n_{exit} tasks, then a dummy entry or exit task with zero-weight dependencies is added to the graph.

(2) W is an $|N|\times|U|$ matrix, where $w_{i,k}$ denotes the worst case execution time (WCET) of n_i running on u_k with the maximum frequency. Each task $n_i \in N$ has different WCET values on different processors owing to the heterogeneity of processors. The WCET of a task is the maximum execution time among all possible real execution time values when the task is executed on a specific processor with the maximum frequency. All the WCETs of tasks are known and determined through the analysis methods performed (i.e., WCET analysis [19]) during the analysis phase.

(3) The communication between tasks mapped to different processors is performed through message passing over the bus. 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 worst case response time (WCRT) of $m_{i,j}$ if n_i and n_j are not assigned to the same processor. The WCRT of a message is the maximum response time among all possible real response time values when the message is transmitted on a specific hardware platform. If n_i and n_j are assigned to the same processor, then the communication time is 0. All the WCRTs of the messages are also known and determined through analysis methods performed (i.e., WCRT analysis [20]) during the analysis phase.

(4) Let $D(G)$ represent the deadline of the application G, which should be larger than or equal to the lower bound on schedule length $LB(G)$ [21]. The lower bound refers to the minimum schedule length (i.e., response time) of an application when all tasks are executed on the processors with the maximum frequencies using a well-studied DAG-based scheduling algorithm; for example, heterogeneous earliest finish time (HEFT) [18] and predict EFT (PEFT) [2] are typically representative algorithms. Let $SL(G)$ represent the final schedule length of G and is the actual finish time (AFT) of the exit task obtained by certain scheduling algorithm. The scheduling can be static or dynamic, and preemptive or non-preemptive [22]. The real-time application can be soft or hard [22]. We consider the non-preemptive static scheduling for a hard real-time application in this study.

Fig. 2 shows a motivating parallel application [12], [18]. Table 2 shows the WCET matrix of the application in Fig. 2. The example shows 10 tasks executed on 3 processors $\{u_1, u_2, u_3\}$. The weight value of 14 of n_1 and u_1 in Table 2 represents the WCET denoted by $w_{1,1}$=14. The same task has different WCETs on different processors because of the heterogeneity of processors. The weight value of 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. For simplicity, all units of all parameters are ignored in the example.

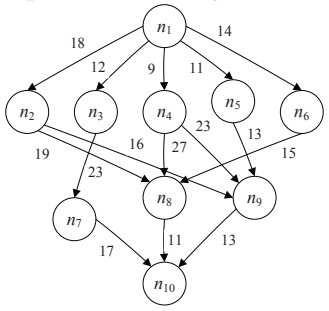


Fig. 2: Motivating example of a DAG-based parallel application [12], [18].

TABLE 2: WCETs of tasks on different processors with the maximum frequencies of the motivating parallel application [12], [18].

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

3.2 Power and Energy Models

Similar to [10], [11], [12], we use the term frequency change to stand for changing the voltage and frequency simultaneously. Considering that a DVFS-enabled system is used, we adopt the system-level power model widely used in [10], [11], [12]. In this model, the power consumption at frequency f is given by

$$P(f) = P_{\text{s}} + h(P_{\text{ind}} + P_{\text{d}}) = P_{\text{s}} + h(P_{\text{ind}} + C_{\text{ef}} f^m). \quad (1)$$

P_s represents the static power and can be removed only by turning off the power of the entire system. P_{ind} represents frequency-independent dynamic power and can be removed by switching the system into the sleep mode. P_{d} represents frequency-dependent dynamic power and depends on frequencies. h represents the system state and indicates whether dynamic powers are currently consumed in the system. When the system is active, h = 1; otherwise, h = 0. C_{ef} represents effective switching capacitance, and m represents the dynamic power exponent with a value no smaller than 2. C_{ef} and m are processor-dependent constants.

When an excessive overhead associated with turning on/off a system exists, P_{s} is usually consumed and unmanageable [10], [11], [12]. Similar to the above works, this study concentrates on managing the dynamic power (i.e., P_{ind} and P_{d}), but we still include the static energy consumption into the total energy consumption. Note that we do not include the communication energy consumption into consideration because we mainly focus on the DVFS energy-efficient design technique of processors. Given P_{ind}, lower frequency than a given threshold may result in more dynamic energy consumption. That is, a minimum dynamic energy-efficient frequency f_{ee} exists [10], [11], [12] and is denoted by

$$f_{\text{ee}} = \sqrt[m]{\frac{P_{\text{ind}}}{(m-1)C_{\text{ef}}}}. \quad (2)$$

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

$$f_{\text{low}} = \max(f_{\text{min}}, f_{\text{ee}}). \quad (3)$$

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

The number of processors is $|U|$ in the system, and thus, these processors are completely heterogeneous and each processor should have individual power parameters [12]; we define the frequency-independent dynamic power set as

$$\{P_{1,\text{ind}}, P_{2,\text{ind}}, ..., P_{|U|,\text{ind}}\},$$

the frequency-dependent dynamic power set as

$$\{P_{1,\text{d}}, P_{2,\text{d}}, ..., P_{|U|,\text{d}}\},$$

the effective switching capacitance set as

$$\{C_{1,\text{ef}}, C_{2,\text{ef}}, ..., C_{|U|,\text{ef}}\},$$

the dynamic power exponent set as

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

the lowest energy-efficient frequency set as

$$\{f_{1,\text{low}}, f_{2,\text{low}}, ..., f_{|U|,\text{low}}\},$$

and the actual effective frequency set as

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

Then, let $E_\text{s}(G)$ represent the processor-generated static energy consumption of application G; it is calculated by

$$E_\text{s}(G) = \sum_{k=1}^{|U|} (P_{k,s} \times SL(G)). \quad (4)$$

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

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

where $\frac{f_{k,\text{max}}}{f_{k,h}} \times w_{i,k}$ represents the actual execution time of n_i on u_k with the frequency $f_{k,h}$.

The dynamic energy consumption of the application is calculated by

$$E_\text{d}(G) = \sum_{i=1}^{|N|} E_\text{d}\left(n_i, u_{pr(i)}, f_{pr(i),hz(i)}\right), \quad (6)$$

where $u_{pr(i)}$ and $f_{pr(i),hz(i)}$ represent the assigned processor and frequency of n_i, respectively. The total energy consumption of the application is the sum of $E_\text{s}(G)$ and $E_\text{d}(G)$, namely,

$$E_\text{total}(G) = E_\text{s}(G) + E_\text{d}(G). \quad (7)$$

In this study, we will also ignore the overheads of the frequency transitions because they take negligible amount of time (e.g., 10 μs-150 μs [3], [9]).

Table 3 shows the power values of the three processors, where the frequency precision is 0.01 in the example. The lowest energy-efficient frequency $f_{k,\text{low}}$ in this example for each processor is obtained according to Eq. (2). Meanwhile, similar to that in [10], [11], the maximum frequency $f_{k,\text{max}}$ for each processor is assumed to be 1.0.

TABLE 3: Power parameters of processors (u_1, u_2, and u_3).

u_k	$P_{k,\text{s}}$	$P_{k,\text{ind}}$	$C_{k,\text{ef}}$	m_k	$f_{k,\text{low}}$	$f_{k,\text{max}}$
u_1	0.01	0.02	1.3	2.9	0.19	1.0
u_2	0.01	0.05	0.5	2.1	0.32	1.0
u_3	0.01	0.04	0.2	3.0	0.46	1.0

3.3 Problem Description

The problem description of this study is to determine the processor and frequency assignments of all tasks and thus minimize the total energy consumption of the application:

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

subject to the deadline constraint:

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

and the frequency selection constraint:

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

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

4 Non-DVFS Energy-efficient Scheduling

This section proposes the deadline slack algorithm (Algorithm 1) and the NDES algorithm (Algorithm 2) together to implement non-DVFS energy-efficient scheduling.

4.1 Lower Bound Certification

Scheduling tasks with quality of service (QoS) requirement for optimality on multiprocessors is known to be an NP-hard optimization problem and many heuristic list scheduling algorithms, such as PEFT [2] and HEFT [18], have been proposed to generate near-optimal solutions for DAG-based scheduling on heterogeneous systems. List scheduling includes two phases: task prioritizing and task allocation. The HEFT algorithm proposed in [18] is a well-studied and commonly used DAG-based scheduling algorithm for reducing the schedule length while achieving low complexity and high performance in heterogeneous distributed systems; this algorithm has also been used in energy-efficient scheduling [8], [9]. Similar to [8], [9], this study also uses the HEFT algorithm to obtain the lower bound of a parallel application. The lower bound is calculated by

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

A deadline $D(G)$, which is larger than or equal to the lower bound $LB(G)$, is then provided for the application. The two-phase HEFT algorithm has two important functions.

(1) Task prioritizing. HEFT uses the upward rank value ($rank_\text{u}$) of a task (Eq. (9)) as the task priority standard. In this case, tasks are ordered according to the descending order of $rank_\text{u}$. Table 4 shows the $rank_\text{u}$ values of all tasks of the motivating parallel application (Fig. 2) obtained by Eq. (9):

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

where $\overline{w_i}$ represents the average WCET of task n_i and is calculated by $\overline{w_i} = \left(\sum_{k=1}^{|U|} w_{i,k} \right) / |U|$.

TABLE 4: Upward rank values, lower bounds, and deadlines of tasks of the motivating parallel application.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
$rank_\text{u}(n_i)$	108	77	80	80	69	63.3	42.7	35.7	44.3	14.7
$LB(n_i)$	9	40	28	26	38	42	49	62	68	80
$D(n_i)$	29	60	48	46	58	62	69	82	88	100

(2) Task assignment. $EST(n_i, u_k, f_{k,\text{max}})$ and $EFT(n_i, u_k, f_{k,\text{max}})$ represent the earliest start time and earliest finish time (EFT) of the task n_i on the processor u_k with the maximum frequency $f_{k,\text{max}}$. $EFT(n_i, u_k, f_{k,\text{max}})$ 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

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

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 when 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 using the insertion-based scheduling strategy, where n_i can be inserted into the slack with the minimum EFT. EFT and AFT differ in that EFT is the value before task assignment, whereas AFT is the value after task assignment.

Fig. 3 shows the Gantt chart of the motivating parallel application using HEFT. The lower bound is obtained as $LB(G) = 80$, and the deadline is set as $D(G) = 100$. The arrows in Fig. 3 represent the generated communication between tasks. The total energy consumption is $E_{\text{total}}(G) = E_{\text{s}}(G) + E_{\text{d}}(G) = 2.4 + 59.17 = 61.57$.

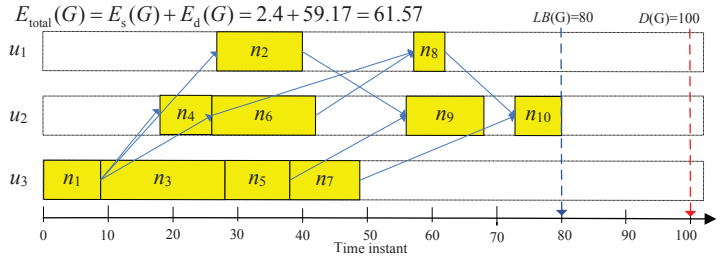


Fig. 3: Scheduling Gantt chart of the motivating parallel application using the HEFT algorithm.

4.2 Deadline Slack

We provide the following definitions to set the deadline of each task:

Definition 1. (**Deadline slack**) The deadline slack of an application represents the slack between the deadline and the lower bound of the application, that is,

$$DS(G) = D(G) - LB(G). \tag{12}$$

Considering the obtained $LB(G)$ using HEFT, $DS(G)$ can be determined by $D(G)$. On the basis of the example in Fig. 3, we obtain $DS(G) = D(G) - LB(G) = 100 - 80 = 20$. A deadline slack exists between the lower bound and the deadline; hence, if a scheduling algorithm can reassign the tasks to the processors with the maximum frequency, while reducing the energy consumption of the application and ensuring that the AFT of the application does not exceed its deadline, then this scheduling algorithm can be regarded as a non-DVFS approach.

In the following discussion, we let the deadline slacks of all tasks be equal to that of the parallel application, that is, $DS(n_i) = DS(G)$; then, the deadline of task n_i is calculated by

$$D(n_i) = LB(n_i) + DS(G), \tag{13}$$

where $LB(n_i)$ represents the lower bound of n_i, which is equal to $AFT(n_i)$ using HEFT. That is, all the tasks have individual lower bounds, as shown in Table 4.

HEFT assigns each task to the processor with the maximum frequency and minimum EFT as mentioned earlier. A deadline is defined for each task (Eq. (13)); hence, we aim to find a proper processor $u_{pr(i)}$ with the maximum frequency $f_{pr(i),\max}$ for the task n_i that minimizes

$$E\left(n_i,u_k,f_{k,\max}\right) = \left(P_{k,\text{ind}} + C_{k,\text{ef}} \times \left(f_{k,\max}\right)^{m_k}\right) \times w_{i,k},$$

subject to:

$$AFT(n_i) \leqslant D(n_i).$$

Accordingly, energy can be reduced without using DVFS. Inspired by the above formal description, we propose the deadline slack algorithm shown in Algorithm 1.

Algorithm 1 The Deadline Slack Algorithm

Input: $U = \{u_1, u_2, ..., u_{|U|}\}$, G, $D(G)$
Output: $SL(G)$, $E_{\text{total}}(G)$
1: Call the HEFT algorithm [18] to obtain the assigned processor and AFT of each task and the lower bound $LB(G)$ of the application G;
2: Sort the tasks in a list $downward\_task\_list$ by descending order of $rank_{\text{u}}$ values.
3: Calculate $D(n_i)$ for each task in $downward\_task\_list$ using Eq. (13);
4: **while** (there are tasks in $downward\_task\_list$) **do**
5: $n_i \leftarrow downward\_task\_list.out()$; // take out one task from the $downward\_task\_list$;
6: **for** (each processor $u_k \in U$) **do**
7: Calculate $EFT(n_i,u_k,f_{k,\max})$ value using Eq. (11) based on the insertion-based scheduling policy;
8: **if** ($EFT(n_i,u_k,f_{k,\max}) \leqslant D(n_i)$) **then**
9: Calculate $E_{\text{d}}(n_i,u_k,f_{k,\max})$ using Eq. (5);
10: **end if**
11: **if** ($EFT(n_i,u_k,f_{k,\max}) \leqslant D(n_i)$ exist for any processor) **then**
12: Assign task n_i to the processor $pr(i)$ with the minimum dynamic energy consumption $E_{\text{d}}(n_i,u_{\text{pr(i)}},f_{k,\max})$ under satisfying its deadline $D(n_i)$;
13: **else**
14: Assign task n_i to the processor $pr(i)$ with the minimum EFT;
15: **end if**
16: **end for**
17: **end while**
18: $SL(G) \leftarrow AFT(n_{\text{exit}})$;
19: Calculate $E_{\text{s}}(G)$ using Eq. (4);
20: Calculate $E_{\text{d}}(G)$ using Eq. (6);
21: Calculate $E_{\text{total}}(G)$ using Eq. (7);

The main idea of the deadline slack algorithm is that the deadline of the application is transferred to that of each task. Each task only selects the processor with the maximum frequency and minimum dynamic energy consumption while satisfying its deadline. The deadline slack algorithm does not use the DVFS technique. The core details of the deadline slack algorithm are explained as follows:

(1) In the task prioritizing phase, similar to HEFT, the deadline slack algorithm ranks tasks according to the descending order of $rank_{\text{u}}$ (Line 2).

(2) In the task assignment phase, the deadline slack algorithm assigns the task to the processor with the minimum dynamic energy consumption while satisfying its deadline (Line 11).

(3) If all the processors cannot satisfy the deadline of the task, then the task is assigned to the processor with the minimum EFT (Lines 12-14).

(4) If n_{exit} has $AFT(n_{\text{exit}}) > D(n_{\text{exit}})$, then $SL(G) > D(G)$; that is, the deadline slack algorithm cannot always obtain a safe schedule length (i.e., not exceeding its deadline), but it will be used in the next subsection.

(5) The deadline slack algorithm has a low time complexity of $O(|N|^2 \times |U|)$, which is similar to that of HEFT.

Example 1. We still let $D(G) = 100$. Table 5 and Fig. 4 show the task assignment and scheduling Gantt chart of the motivating parallel application using the deadline slack algorithm.

TABLE 5: Task assignment of the motivating parallel application using the deadline slack algorithm.

n_i	u_k	$f_{k,h}$	$AST(n_i)$	$AFT(n_i)$	$D(n_i)$	$E_d(n_i)$
n_1	u_3	1.0	0	9	29	2.16
n_3	u_3	1.0	9	28	48	4.56
n_4	u_3	1.0	28	45	46	4.08
n_2	u_2	1.0	27	46	60	10.45
n_5	u_3	1.0	45	55	58	2.4
n_6	u_2	1.0	46	62	62	8.8
n_9	u_3	1.0	62	82	88	4.8
n_7	u_1	1.0	51	58	69	9.24
n_8	u_1	1.0	77	82	82	6.6
n_{10}	u_2	1.0	**95**	**102**	**100**	**3.85**
$SL(G) = 102,$ $E_{\text{total}}(G) = E_s(G) + E_d(G) = 3.06 + 56.94 = 60$						

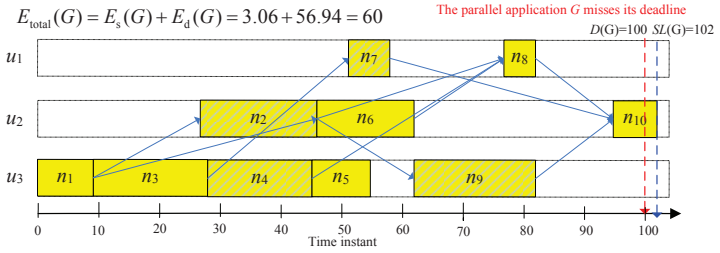


Fig. 4: Scheduling Gantt chart of the motivating parallel application using the deadline slack algorithm.

As shown in Fig. 4, the tasks n_4, n_2, n_9 and n_7 have changed their assigned processors when compared with those in Fig. 3. The total consumed energy using the deadline slack algorithm is $E_{\text{total}}(G) = 60$, which is less than the result using the HEFT algorithm (61.57 in Fig. 3); however, the obtained schedule length using the deadline slack algorithm is larger than the deadline and is unsafe (i.e., $SL(G) = 102 > D(G) = 100$). This example verifies that using the deadline slack algorithm cannot ensure the safety of the obtained schedule length. Moreover, the total energy consumption using the deadline slack algorithm is insignificantly decreased compared with that using the HEFT algorithm. Thus, further optimization is required.

4.3 Variable Deadline Slack

The unsafe schedule length obtained by the deadline slack algorithm cannot ensure that all tasks are finished before individual deadlines on each assigned processor. The reason is that processors are heterogeneous and there is communication time between any two precedence constrained tasks if they are not assigned to the same processor, such that the finish time of all the tasks do not change linearly. Considering the example in Table 5, although all the preceding assignments are safe, n_{10} misses its deadline (marked in bold text). Intuitively, different deadline slacks should be set for different tasks. Unfortunately, finding an optimal deadline slack for each task is very time consuming because all possible values should be exhausted. For example, the deadline slack of each task should be in the scope of [0, 20] due to $DS(G) = 20$ for the motivating parallel application; the exhausting number of deadline slack combinations for the application should be 20^{10} (10 tasks with 20 deadlines each), if the deadline slack is an integer. To obtain a safe schedule length without exhausting all deadline slack combinations, we perform the following optimization.

(1) A variable deadline slack $VDS(G)$ for the application is defined and should be in the scope of [0,$DS(G)$]. Unlike the method of exhausting all deadline slack combinations of the above solution, we let all tasks have the same $VDS(n_i)$, which is equal to $VDS(G)$. That is,

$$\begin{cases} D(n_{\text{exit}}) = D(G) \\ D(n_i) = LB(n_i) + VDS(G). \end{cases} \quad (14)$$

Accordingly, the number of deadline slack combinations is reduced to $DS(G)$.

(2) Considering the possibly large $DS(G)$ (e.g., 1000 ms) of the real parallel application, a fixed step value $S(G)$ (e.g., 10 ms) should be provided for the application to reduce the number of variable deadline slacks (e.g., from 900 ms to 0 ms, with 10 ms decrement). However, it is a complicated problem to decide the size of $S(G)$ according to the size of $DS(G)$. If $S(G)$ is too large, it will reduce the accuracy of the computation result; if $S(G)$ is too small, it will increase the computation time of the algorithm. Therefore, it is necessary to achieve a reasonable tradeoff between precision and computing time. For this reason, we let $S(G)$ be

$$S(G) = \frac{DS(G)}{100},$$

in this study. If $S(G)$ is less than 1, we set it as 1.

4.4 The NDES Algorithm

On the basis of the aforementioned analysis, we present the NDES algorithm described in Algorithm 2.

Algorithm 2 The NDES Algorithm

Input: $U = \{u_1, u_2, ..., u_{|U|}\}$, G, $D(G)$,$S(G)$
Output: $SL(G)$, $E_{\text{total}}(G)$
1: Call the deadline slack algorithm (Algorithm 1) to obtain initial $E_{\text{total}}(G)$ and schedule length $SL(G)$ based on $DS(n_i)$ calculated by Eq. (13);
2: **if** $(SL(G) > D(G))$ **then**
3: $VDS(G) \leftarrow DS(G)$-$S(G)$;
4: **while** $(VDS(G) \geqslant 0)$ **do**
5: Calculate new $D(n_i)$ for each task using $VDS(G)$ using Eq. (14);
6: Call the deadline slack algorithm (Algorithm 1) to obtain new $SL_{\text{new}}(G)$ and $E_{\text{new}}(G)$ based on new $D(n_i)$;
7: **if** $(SL_{\text{new}}(G) \leqslant D(G))$ **then**
8: **if** $(E_{\text{new}}(G) < E_{\text{total}}(G))$ **then**
9: $E_{\text{total}}(G) \leftarrow E_{\text{new}}(G)$;
10: $SL(G) \leftarrow SL_{\text{new}}(G)$;
11: **end if**
12: **end if**
13: Update $VDS(G) \leftarrow VDS(G)$-$S(G)$;
14: **end while**
15: **else**
16: $VDS(G) \leftarrow DS(G)$+$S(G)$;
17: $MVDS(G) \leftarrow \min_{i \in [1,|N|-1]} \{D(G) - LB(n_i), \}$;
18: **while** $(VDS(G) \leqslant MVDS(G))$ **do**
19: Calculate new $D(n_i)$ for each task using $VDS(G)$ using Eq. (14);
20: Call the deadline slack algorithm (Algorithm 1) to obtain new $SL_{\text{new}}(G)$ and $E_{\text{new}}(G)$ based on new $D(n_i)$;
21: **if** $(SL_{\text{new}}(G) \leqslant D(G))$ **then**
22: **if** $(E_{\text{new}}(G) < E_{\text{total}}(G))$ **then**
23: $E_{\text{total}}(G) \leftarrow E_{\text{new}}(G)$;
24: $SL(G) \leftarrow SL_{\text{new}}(G)$;
25: **end if**
26: **end if**
27: Update $VDS(G) \leftarrow VDS(G)$+$S(G)$;
28: **end while**
29: **end if**

The time complexity of the NDES algorithm is $O(|N|^2 \times |U| \times V)$, where V represents the number of variable deadline slacks. The main idea of the NDES algorithm is that it iteratively calls the deadline slack algorithm (Algorithm 1) and thus obtains a safe schedule length with the minimum dynamic energy consumption. The core steps of the NDES algorithm are explained as follows:

(1) Call the deadline slack algorithm to obtain the initial $E_{\text{total}}(G)$ and schedule length $SL(G)$ based on $D(n_i)$ of Eq. (13) in Lines 1-2.

(2) If the deadline slack algorithm cannot obtain a safe schedule length in Lines 1-2, all variable deadline slacks (from

$DS(G)-S(G)$ to 0) should be traversed to obtain a safe schedule length with the minimum dynamic energy consumption based on the preceding analysis in Lines 3-15.

(3) In case that the deadline slack algorithm can obtain a safe schedule length in Lines 1-2, the energy consumption can be further optimized by iterating all variable deadline slacks in the scope of $DS(G)$ and $MVDS(G)$ in Lines 16-29. The maximum variable deadline slack is calculated by

$$MVDS(G) = \min_{i\in[1,|N|-1]} \{D(G) - LB(n_i),\}$$

because all the finish time values of tasks can not exceed the deadline of the application.

Theorem 1. NDES can always obtain a safe schedule length with as less dynamic energy consumption as possible.

Proof. On the one hand, considering that HEFT generates a safe lower bound $LB(G)$ of the application according to lower bound certification in Section 4.1, the deadline of each task using HEFT is as follows:

$$\begin{cases} D(n_{\text{exit}}) = LB(n_{\text{exit}}) \\ D(n_i) = LB(n_i). \end{cases} \quad (15)$$

In this case, all the tasks are safe.

On the other hand, the deadline of each task using NDES is as follows:

$$\begin{cases} D(n_{\text{exit}}) = D(G) \\ D(n_i) = LB(n_i). \end{cases} \quad (16)$$

when $VDS(G) = 0$ according to Eq. (14).

The sole difference between Eq. (15) and Eq. (16) is that Eq. (16) has longer deadline than Eq. (15) for the exit task. Therefore, Eq. (16) can also obtain the same schedule length as Eq. (15) as long as the exit task's AFT is $LB(G)$. In addition, as the exit task's deadline is equal to the application's deadline using Eq. (16), it is always safe. Considering all tasks are safe using Eq. (16), NDES can always obtain a safe schedule length.

Considering that the exit task's AFT belongs to scope of [LB(G), D(G)] using Eq. (16), NDES can always consume no more dynamic energy than HEFT. In addition, NDES traverses all variable deadline slacks and selects one with the minimum dynamic energy consumption while satisfying the deadline constraint. Therefore, NDES can always obtain a safe schedule length with as less dynamic energy consumption as possible. ■

Example 2. Table 6 lists the results of the motivating parallel application using the NDES algorithm. This example shows the following facts: 1) when $VDS(G) \leqslant 18$, the obtained schedule length is safe; 2) specifically, when $VDS(G) = 17$, the consumed energy is the minimum. Another important discovery is that adjacent variable deadline slacks have the same results, which indicate that using a step value $S(G)$ is necessary for reducing the number of variable deadline slacks, as shown in Table 6.

Table 7 and Fig. 5 show the task assignment and scheduling Gantt chart of the motivating parallel application using the NDES algorithm, respectively.

As shown in Fig. 5, the tasks n_2, n_4, n_5, n_6, and n_7 have changed their assigned processors when compared with those in Fig. 4. That is, the task assignment and scheduling Gantt chart of **Example 2** (Fig. 5 and Table 7) using the NDES algorithm is completely different from those of **Example 1** (Fig. 4 and Table 5) using the deadline slack algorithm. The total energy consumption using the NDES algorithm is $E_{\text{total}}(G) = 51.1$, which is less than the result using the deadline slack algorithm (60 in Fig. 4); moreover, the obtained schedule length using the NDES algorithm is less than the deadline and is safe (i.e., $SL(G) = 95 < D(G) = 100$). The example above shows that the NDES algorithm not only can satisfy the deadline constraint of the application, but also can further reduce the energy consumption without using the DVFS technology.

TABLE 6: Results of the motivating parallel application using the NDES algorithm with variable deadline slacks.

$VDS(G)$	$SL(G)$	$D(G)$	$E_{\text{total}}(G)$	Satisfying deadline constraint?	Minimum energy consumption?
20	102	100	60	No	No
19	102	100	67.37	No	No
18	96	100	52.47	Yes	No
17	**95**	**100**	**51.1**	**Yes**	**Yes**
16	95	100	52.09	Yes	No
15	95	100	52.09	Yes	No
14	95	100	52.09	Yes	No
13	95	100	52.09	Yes	No
12	97	100	67.14	Yes	No
11	97	100	67.14	Yes	No
10	97	100	67.14	Yes	No
9	97	100	67.14	Yes	No
8	97	100	67.14	Yes	No
7	97	100	67.14	Yes	No
6	97	100	67.14	Yes	No
5	97	100	55.43	Yes	No
4	97	100	62.07	Yes	No
3	97	100	62.07	Yes	No
2	97	100	62.07	Yes	No
1	97	100	62.07	Yes	No
0	97	100	62.07	Yes	No

TABLE 7: Task assignment of the motivating parallel application using NDES.

n_i	u_k	$f_{k,h}$	$AST(n_i)$	$AFT(n_i)$	$D(n_i)$	$E_{\text{d}}(n_i)$
n_1	u_3	1.0	0	9	29	2.16
n_3	u_3	1.0	9	28	48	4.56
n_4	u_2	1.0	18	26	46	4.4
n_2	u_3	1.0	28	46	60	4.32
n_5	u_2	1.0	26	39	58	7.15
n_6	u_3	1.0	46	55	62	2.16
n_9	u_3	1.0	55	75	88	4.8
n_7	u_2	1.0	51	66	69	8.25
n_8	u_1	1.0	70	75	82	6.6
n_{10}	u_2	**1.0**	**88**	**95**	**100**	**3.85**
$SL(G) = 95,$ $E_{\text{total}}(G) = E_{\text{s}}(G) + E_{\text{d}}(G) = 2.85 + 48.25 = 51.1$						

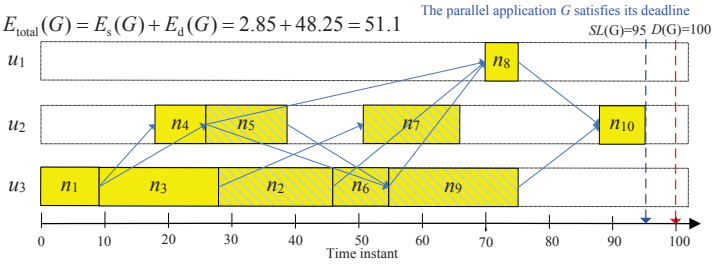


Fig. 5: Scheduling Gantt chart of the motivating parallel application using NDES.

5 Global DVFS-enabled Energy-efficient Scheduling

This section mainly proposes the GDES (Algorithm 3) to implement global DVFS-enabled energy-efficient scheduling.

5.1 Existing EES Algorithm

The obtained schedule length using the NDES algorithm is safe. However, if $LB(G)$ is equal to $D(G)$, then the algorithm cannot work because the deadline slack is 0 (i.e., $DS(G) = 0$). Furthermore, the NDES algorithm fails to use the DVFS technology, and slacks exist between the two adjacent tasks in the same processor. These slacks can be eliminated or decreased using DVFS to further reduce energy consumption. An example is shown in Fig.

5. In this example, 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} using the DVFS technology while still satisfying the deadline constraint. We combine the NDES and EES algorithms to form the NDES&EES algorithm to further reduce the energy consumption of the application in this study.

Fig. 6 shows the scheduling of the motivating parallel application using the NDES&EES algorithm. A major optimization of the NDES&EES algorithm is that the AFTs of partial tasks (i.e., n_5, n_7, n_8, and n_{10}, marked in underline) are extended to individual LFTs. This example shows that the NDES&EES algorithm can further minimize the energy consumption of the application without violating the precedence constraints among tasks and the deadline constraint of the application. As expected, the total consumed energy using the NDES&EES algorithm is reduced to 42.0558, which is less than 51.1 using the NDES algorithm alone.

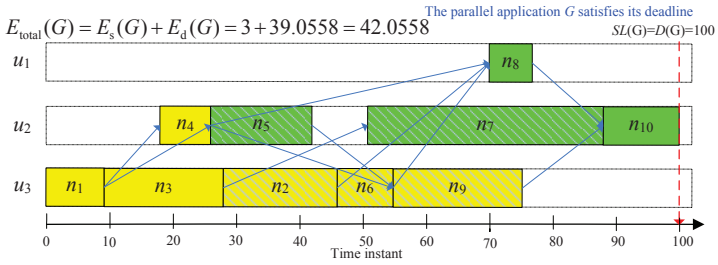


Fig. 6: Scheduling Gantt chart of the motivating parallel application using the NDES&EES algorithm.

The major limitation of using the NDES&EES algorithm is that only a small part of tasks that are near to the exit task can be optimized, and these tasks can only be optimized on the fixed processors (n_5, n_7, n_8, and n_{10} on u_2 and n_8 on u_1 in Fig. 6) through a local energy-efficient scheduling algorithm. Notably, a large part of tasks can be optimized by moving them to other processors through a global energy-efficient scheduling algorithm. In the following, the aforementioned problems will be solved.

5.2 Revision of Computation Time

To implement global DVFS-enabled energy optimization, we make the following revisions of computation time:

(1) Earliest start time. The EST defined in Eq. (10) considers the earliest available time $avail[k]$ that processor u_k is ready for task execution. However, as we consider global energy consumption optimization in this study, the task n_i should not be restricted by $avail[k]$, which may be changed, as any task can be migrated from the current processor u_k to another; thus, we update EST as follows:

$$\begin{cases} EST(n_{\text{entry}}, u_k) = 0 \\ EST(n_i, u_k) = \max\limits_{n_x \in pred(n_i)} \{AFT(n_x) + c'_{x,i}\} \end{cases}. \quad (17)$$

(2) Latest finish time. The task can be migrated to another processor if it generates less dynamic energy consumption; thus, each task should have individual LFTs on different processors, as follows:

$$\begin{cases} LFT(n_{\text{exit}}, u_k) = D(G) \\ LFT(n_i, u_k) = \min\limits_{n_j \in succ(n_i)} \{AST(n_j) - c'_{i,j}\} \end{cases}. \quad (18)$$

(3) Available slacks. Some slacks remain in the processors after using the HEFT algorithm, and the slack set for processor u_k is defined as follows:

$$S_k = \{S_{k,1}, S_{k,2}, S_{k,|S_k|}\},$$

where $S_{k,1}$ represents the first slack on u_k. Each slack has a start time (ST) and end time (ET), and we define yth slack $S_{k,y}$ as follows:

$$S_{k,y} = [t_s(S_{k,y}), t_e(S_{k,y})],$$

where $t_s(S_{k,y})$ and $t_e(S_{k,y})$ represent ST and ET, respectively. As the application has a given deadline, the end time of the last slack should be $t_e(S_{k,|S_k|}) = D(G)$.

For example, when assigning the task n_{10} in Fig. 3, the slacks on u_1, u_2, and u_3 are $S_1 = \{[0, 27], [40, 57], [62, 100]\}$, $S_2 = \{[0, 18], [42, 56], [68, 100]\}$, and $S_3 = \{[59, 100]\}$. Notably, n_{10} itself should be removed, such that optimal slacks can be selected for n_{10}.

5.3 Energy-efficient Dynamic Energy Consumption

(1) Maximum execution time. To avoid violating the precedence constraints among tasks, task n_i should be assigned to the slacks that satisfy the following constraints:

$$EST'(n_i, u_k) = \max\{EST(n_i, u_k), t_s(S_{k,t})\}, \quad (19)$$

and

$$LFT'(n_i, u_k) = \min\{LFT(n_i, u_k), t_e(S_{k,t})\}. \quad (20)$$

Then, maximum execution time (MET) for n_i on u_k should be derived as follows:

$$MET(n_i, u_k) = LFT'(n_i, u_k) - EST'(n_i, u_k). \quad (21)$$

For example, when assigning task n_{10} in Fig. 3, we obtain $EST'(n_{10}, u_1) = 81$, $EST'(n_{10}, u_2) = 73$, $EST'(n_{10}, u_3) = 81$, and $LFT'(n_{10}, u_1) = LFT'(n_{10}, u_2) = LFT'(n_{10}, u_3) = 100$. Then, the METs for n_{10} should be $MET(n_{10}, u_1) = 19$, $MET(n_{10}, u_2) = 27$, $MET(n_{10}, u_3) = 19$.

Meanwhile, the constraint in Eq. (22) must be satisfied, else n_i cannot be inserted into the slack, even if maximum frequency is used.

$$MET(n_i, u_k) \geqslant w_{i,k}. \quad (22)$$

For example, n_{10} cannot be inserted into u_1 because $MET(n_{10}, u_1) = 19$ is less than $w_{10,1} = 21$, which is the execution time of n_{10} on u_1 with maximum frequency.

(2) Energy-efficient MET. As each processor has the lowest frequency ($f_{k,\text{low}}$), the upper bound execution time (UBET) of n_i on u_k can be calculated as follows:

$$UBET(n_i, u_k) = \frac{f_{k,\max}}{f_{k,low}} \times w_{i,k}. \quad (23)$$

Finally, the lowest energy-efficient MET of n_i on u_k should be derived as follows:

$$MET_{\text{low}}(n_i, u_k) = \min\{MET(n_i, u_k), UBET(n_i, u_k)\}. \quad (24)$$

Correspondingly, the lowest energy-efficient frequency of n_i on u_k should be expressed as follows:

$$f_{\text{low}}(n_i, u_k) = \frac{w_{i,k}}{MET_{\text{low}}(n_i, u_k)} \times f_{k,\max}. \quad (25)$$

(3) Lowest energy-efficient dynamic energy consumption. As $f_{\text{low}}(n_i, u_k)$ is obtained, the minimum dynamic energy consumption can be generated for n_i on u_k, as follows:

$$\begin{aligned} E_{\text{low}}(n_i, u_k) &= E_d(n_i, u_k, f_{\text{low}}(n_i, u_k)) \\ &= (P_{k,\text{ind}} + C_{k,\text{ef}} \times f_{\text{low}}(n_i, u_k)^{m_k}) \times \frac{f_{k,\max}}{f_{\text{low}}(n_i, u_k)} \times w_{i,k} \\ &= (P_{k,\text{ind}} + C_{k,\text{ef}} \times f_{\text{low}}(n_i, u_k)^{m_k}) \times MET_{\text{low}}(n_i, u_k) \\ &= \left(P_{k,\text{ind}} + C_{k,\text{ef}} \times \left(\frac{w_{i,k} \times f_{k,\max}}{MET_{\text{low}}(n_i, u_k)}\right)^{m_k}\right) \times MET_{\text{low}}(n_i, u_k). \end{aligned} \quad (26)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2730876, IEEE Transactions on Parallel and Distributed Systems

Then, we assign n_i to $u_{pr(i)}$ with $E_{\text{low}}\left(n_i, u_{pr(i)}\right)$, and the AFT and AST of n_i should be updated as follows:

$$AFT(n_i) = LFT^{'}(n_i, u_{pr(i)}), \tag{27}$$

and

$$AST(n_i) = LFT^{'}(n_i, u_{pr(i)}) - MET_{\text{low}}{}^{'}(n_i, u_{pr(i)}). \tag{28}$$

5.4 The GDES Algorithm

Considering the aforementioned definitions and equations, this subsection presents the GDES algorithm to minimize energy consumption without violating precedence constraints among tasks and application's deadline. The effectiveness of the EES algorithm is limited, as it merely minimizes energy consumption through a local energy-efficient scheduling algorithm on a fixed processor. By contrast, the proposed GDES algorithm described in Algorithm 3 applies a global energy-efficient scheduling algorithm that reduces more energy consumption than EES.

Algorithm 3 The GDES Algorithm

Input: $U = \{u_1, u_2, ..., u_{|U|}\}$, G, and $D(G)$
Output: $E_{\text{total}}(G)$
1: Sort the tasks in a list $upward\_task\_list$ by descending order of $AFT(n_i)$ values.
2: **while** (there are tasks in $upward\_task\_list$) **do**
3: $n_i \leftarrow upward\_task\_list.out()$;
4: **for** (each $u_k \in U \&\& u_k$ is turned-on) **do**
5: Calculate $EST(n_i, u_k)$ using Eq. (17);
6: Calculate $LFT(n_i, u_k)$ using Eq. (18);
7: Calculate $EST^{'}(n_i, u_k)$ using Eq. (19);
8: Calculate $LFT^{'}(n_i, u_k)$ using Eq. (20);
9: Calculate $MET(n_i, u_k)$ using Eq. (21);
10: **if** $(MET(n_i, u_k) \leqslant w_{i,k})$ **then**
11: **continue**; //If Eq. (22) $(MET(n_i, u_k) \leqslant w_{i,k})$ cannot be satisfied, then ignore the current processor;
12: **end if**
13: Calculate $UBET(n_i, u_k)$ using Eq. (23);
14: Calculate $MET_{\text{low}}(n_i, u_k)$ using Eq. (24);
15: Calculate $f_{\text{low}}(n_i, u_k)$ using Eq. (25);
16: Calculate $E_{\text{low}}(n_i, u_k)$ using Eq. (26);
17: **end for**
18: Assign n_i to the $u_{pr(i)}$ with the $E_{\text{low}}\left(n_i, u_{pr(i)}\right)$;
19: Update $AFT(n_i)$ using Eq. (27);
20: Update $AST(n_i)$ using Eq. (28);
21: **end while**
22: Calculate $E_{\text{s}}(G)$ using Eq. (4);
23: Calculate $E_{\text{d}}(G)$ using Eq. (6);
24: Calculate $E_{\text{total}}(G)$ using Eq. (7);

The main idea of the GDES algorithm is to reassign tasks to processor slacks with minimum dynamic energy consumptions. The core details of GDES are explained as follows:

(1) In Line 1, the tasks in the list $upward\_task\_list$ are sorted according to the descending order of $AFT(n_i)$ values.

(2) In Lines 2-21, global energy-efficient scheduling is implemented, based on the careful analysis in Sections 5.1 and 5.2.

(3) In Lines 22-24, the new $E_{\text{total}}(G)$ is calculated.

(4) GDES has low time complexity of $O(|N|^2 \times |U|)$, which is similar to that of the HEFT algorithm. Thus, GDES implements energy-efficient scheduling with low time complexity.

Similar to the combined NDES&EES algorithm, we also combine the NDES and GDES algorithms to form the NDES&GDES algorithm to further reduce the energy consumption of the application.

Example 3. Table 8 lists the task assignment of motivating parallel application using the NDES&GDES algorithm. We determine that the obtained schedule length of the application is equal to its deadline (i.e., $SL(G) = D(G)$=100). As expected, the total energy consumption using NDES&GDES is 33.4165, which is less than 42.0558 when using NDES&EES.

Compared with the results obtained by the NDES&EES algorithm, NDES&GDES further scales down the frequency of tasks on the fixed processor (for instance, frequency is scaled from 1.0 to 0.32 for n_4) and migrates the tasks n_9, n_7 and n_5 (highlighted in bold text) to other processors to reduce dynamic energy consumption.

TABLE 8: Task assignment of the motivating parallel application using the NDES&GDES algorithm.

n_i	u_k	$f_{k,h}$	$AST(n_i)$	$AFT(n_i)$	$E_{\text{d}}(n_i)$
n_{10}	0.00,0.59,0.00u_2	0.00,0.59,0.000.58	88	100.0	2.5114
n_8	u_1	0.71	70	77	3.5105
n_9	u_2	**0.55**	**66**	**88**	**4.2344**
n_7	u_3	**0.69**	**51**	**71**	**1.6912**
n_6	u_3	1.0	46	55	2.16
n_2	u_3	1.0	28	46	4.32
n_5	u_1	**0.36**	**20**	**53**	**2.8768**
n_3	u_3	1.0	9	28	4.56
n_4	u_2	0.32	18	43	2.3922
n_1	u_3	1.0	0	9	2.16
$SL(G)$=$D(G)$=100, $E_{\text{total}}(G) = E_{\text{s}}(G) + E_{\text{d}}(G) = 3 + 30.4165 = 33.4165$					

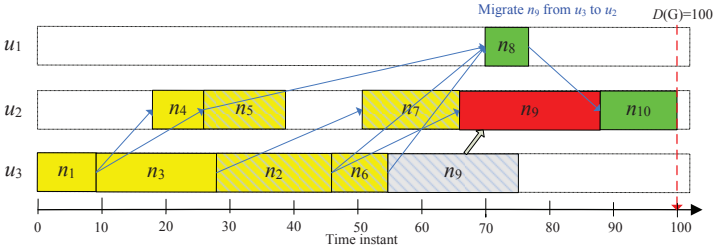


Fig. 7: Scheduling tasks n_{10}, n_8, and n_9 of the motivating parallel application using the NDES&GDES algorithm.

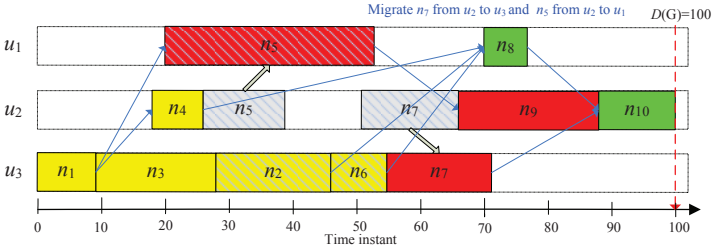


Fig. 8: Scheduling tasks n_7, n_6, n_2, and n_5 of the motivating parallel application using the NDES&GDES algorithm.

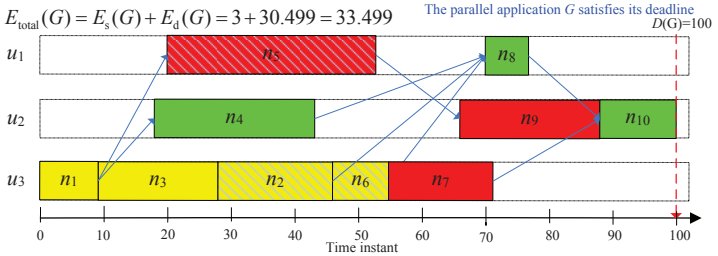


Fig. 9: Scheduling n_3, n_4, and n_1 of the motivating parallel application using the NDES&GDES algorithm.

Figs. 7-9 show the Gantt charts of the motivating parallel application using the NDES&GDES algorithm.

(1) The tasks n_{10} and n_8 extend individual AFTs to LFTs on fixed processors (highlighted in underline), as shown in Fig. 7; even n_{10} and n_8 ar not moved them to other processors, they consume minimum dynamic energy consumption. When reassigning n_9, it is migrated from u_3 to u_2 because $E_{\text{low}}\left(n_9, u_2\right) = 4.2344$, which is less than $E_{\text{low}}\left(n_9, u_3\right) = 4.8$. Furthermore, n_9 cannot be assigned to u_1 because Eq. (22) cannot be satisfied.

(2) As n_9 has been reassigned to u_2 from u_3, a larger slack exists in u_3 (Fig. 7). The following n_7 can be migrated to u_3 with minimum dynamic energy consumption, as shown in Fig. 8. Both n_6 and n_2 cannot extend their AFTs or be reassigned to other processors; however, n_5 can be migrated from u_2 to u_1 with lowest dynamic energy consumption with $E_{\text{low}}\ (n_5, u_1) = 2.8768$.

(3) The remaining tasks are n_3, n_4, and n_1 (Fig. 9). Neither n_3 nor n_1 can extend their AFTs nor be reassigned to other processors. Meanwhile, the AFT of n_4 can be extended to 43, because n_5 has been migrated to n_1, creating a larger slack in u_2.

A significant improvement of the GDES algorithm is shown in **Example 3**: implementing global task reassignment can make predecessor tasks consume less dynamic energy consumption and produce a butterfly effect. For example, moving n_9 is useful for n_7, and moving n_5 is useful for n_4. By contrast, the EES algorithm cannot generate this effect.

6 Experiments

6.1 Experimental Metrics

To conduct a comprehensive comparison, the following four algorithms are worthy of analysis: NDES, NDES&EES, GDES, and NDES&GDES. The reason is that not only the NDES and GDES algorithms can be used independently, but also can be combined with EES or each other to form NDES&EES and NDES&GDES algorithms. The algorithms compared with the above algorithms are the lower bound certification algorithm HEFT [18] and the state-of-the-art EES [8] and DEWTS [9] algorithms. EES and DEWTS study the same problem of minimizing energy consumption of a real-time parallel application on heterogeneous distributed systems. The performance metrics selected for comparison are the total energy consumption $E_{\text{total}}(G)$ (Eq. (7)) of the application and the reduced energy ratio with respect to the HEFT algorithm. The ratio is calculated by

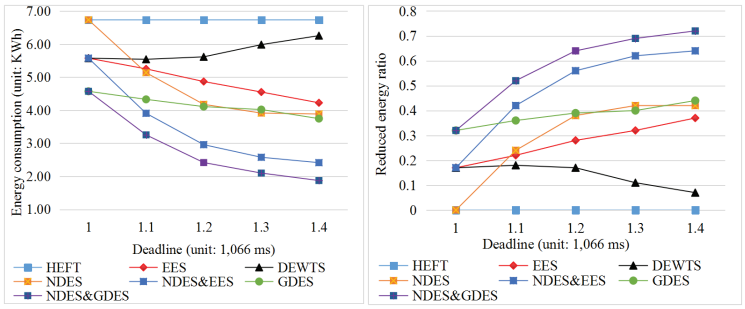
$$R^{\text{X}}(G) = \frac{E^{\text{HEFT}}_{\text{total}}(G) - E^{\text{X}}_{\text{total}}(G)}{E^{\text{HEFT}}_{\text{total}}(G)},$$

where $E^{\text{X}}_{\text{total}}(G)$ and $E^{\text{HEFT}}_{\text{total}}(G)$ represent the total energy consumption generated by The X and HEFT algorithms, respectively.

The values of the processor and application parameters taken from [11], [12] are as follows: 10 ms $\leqslant w_{i,k} \leqslant$ 100 ms, 10 ms $\leqslant c_{i,j} \leqslant$ 100 ms, $P_{k,\text{s}} = 0.01$, $0.03 \leqslant P_{k,\text{ind}} \leqslant 0.07$, $0.8 \leqslant C_{k,\text{ef}} \leqslant 1.2$, $2.5 \leqslant m_k \leqslant 3.0$, and $f_{k,\text{max}}$=1 GHz. All frequencies are discrete, and the precision is 0.1 GHz. These parameter values basically reflect the characters of some high-performance embedded processors, such as Intel Mobile Pentium III and ARM Cortex-A9. All parallel applications are executed on a simulated multiprocessor distributed system with 64 heterogeneous processors using Java on a desktop computer. Real parallel applications with precedence constrained tasks, such as fast Fourier transform and Gaussian elimination applications, are widely used in distributed systems [2], [18]. The Fourier transform and Gaussian elimination application are two typical parallel applications that have high and low parallelism, respectively. A new parameter ρ is used as the size of the fast Fourier transform application. The total number of tasks is $|N| = (2 \times \rho - 1) + \rho \times \log_2 \rho$, where $\rho = 2^y$ for some integer y [18]. The parameter ρ is used as the matrix size of the Gaussian elimination application, and the total number of tasks is $|N| = \frac{\rho^2+\rho-2}{2}$ [18]. To verify the effectiveness and validity of the proposed algorithms, we use the two types of real parallel applications to compare the results of all the algorithms. Many applications with the same parameter values and scales are tested and show the same regular pattern and relatively stable results in the same experiment. In other words, all experiments are repeatable and do not affect the consistency of the results. Therefore, we list all the results in experiments by executing one run of the algorithms for one application in this study. Experiments are completed using Java on a standard desktop computer with 2.6 GHz Intel CPU and 4 GB memory.

6.2 Fast Fourier Transform Application

Experiment 1. This experiment is conducted to compare the total energy consumption values of the fast Fourier transform application for varying deadline constraints. We limit the number of the application to $|N| = 1151$ and change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.4$ with $LB(G) \times 0.1$ increment, where $LB(G) = 1066$ ms is calculated by the HEFT algorithm.



(a) Total energy consumptions (unit: KWh). (b) Reduced energy ratios.

Fig. 10: Results of the fast Fourier transform application with $|N| = 1151$ for varying deadlines.

The results are shown in Figs. 10(a) and 10(b), and we can draw the following observations.

(1) The maximum energy consumption using HEFT is 6.7355 KWh, and the six compared algorithms can reduce the energy consumption to a certain extent.

(2) The schedule lengths obtained by the EES, DEWTS, NDES&EES, GDES, and NDES&GDES algorithms are equal to the given deadlines by using the concept of LFT; therefore, this experiment does not list the scheduling lengths of the algorithms.

(3) When $D(G) = LB(G)$, NDES&GDES and GDES have the same energy consumption values. The reason is that NDES cannot work as the deadline slack is 0; hence, NDES has the same energy consumption value with HEFT in this case and the same energy consumption values are obtained for NDES&EES and EES (Fig. 10(a)). However, when $D(G) \geqslant 1.1 \times LB(G)$, NDES generates less energy consumptions than EES and DEWTS.

(4) NDES&GDES-generated energy consumption values are less than or equal to those using all the algorithms. Even when $D(G) = LB(G)$, NDES&GDES is superior to HEFT, EES, DEWTS, NDES, NDES&EES, respectively (Fig. 10(b)). As the deadlines increase, the NDES&GDES algorithm consumes less energy than these algorithms. Specifically, when $D(G) = LB(G) \times 1.4$, the reduced energy ratio using NDES&GDES reaches 0.72 (Fig. 10(b)).

(5) When $D(G) = LB(G) \times 1.0$, DEWTS allows for a slightly lower energy consumption than those of EES, NDES, and NDES&EES. This result is due to that DEWTS turns off the partial processors with low-energy utilizations and migrates the tasks assigned in these processors to other processors with high-energy utilizations; however, DEWTS increases energy consumptions and is no longer energy efficient when $D(G) > LB(G) \times 1.1$. Such results indicate that DEWTS is suitable mainly for minimizing static energy consumption by turning off processors but is not energy efficient for dynamic energy consumption.

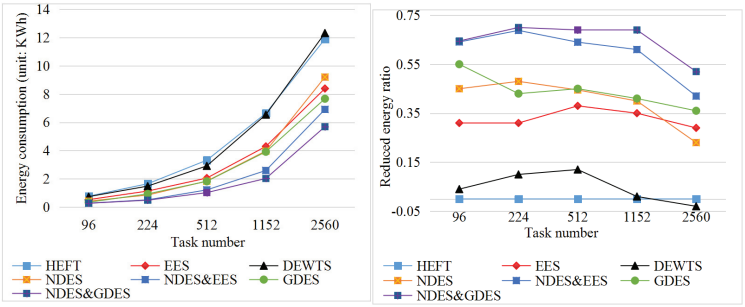
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2730876, IEEE Transactions on Parallel and Distributed Systems

(6) Both NDES&GDES and NDES&EES generate less (equal) energy consumptions than (to) GDES and EES, respectively, in all cases. Furthermore, NDES itself can generate less energy consumption values than the EES algorithm in most cases. Such results indicate that a well-designed non-DVFS algorithm can even achieve more energy reduction than the existing DVFS algorithm.

(7) Both GDES and NDES&GDES generate less energy consumptions than EES and NDES&EES, respectively, in all cases. This finding indicates that reducing energy consumption is limited by the local EES and DEWTS energy-efficient scheduling algorithms and the global energy-efficient scheduling algorithm is superior to the local energy-efficient scheduling algorithm even when the deadline is equal to the lower bound.

On the aforementioned results and analysis, we can derive that synthetically applying combined non-DVFS and global DVFS-enabled energy-efficient scheduling can implement the objective of minimizing energy consumption, and the superiority is more significant in all cases. The results of Fig. 10(a) show that NDES&GDES can save up to 55.65% and 70.05% of energy compared with state-of-the-art EES and DEWTS. If these algorithms are executed in non-DVFS environments, only NDES and NDES&GDES can work. The results of Fig. 10(b) show the reduced energy ratios with respect to HEFT for NDES are from 0.32 to 0.44. Therefore, the proposed NDES&GDES algorithm is very effective in both non-DVFS and DVFS environments for parallel applications.

Experiment 2. We aim to observe the performance on different scales of applications. Hence, an experiment is conducted to compare the total energy consumption values of fast Fourier transform applications for varying numbers of tasks. We limit $D(G)$ to $D(G) = LB(G) \times 1.4$; furthermore, ρ is changed from 16 to 256, that is, the number of tasks varies from 95 (small scale) to 2559 (large scale).



(a) Total energy consumptions. (b) Reduced energy ratios.

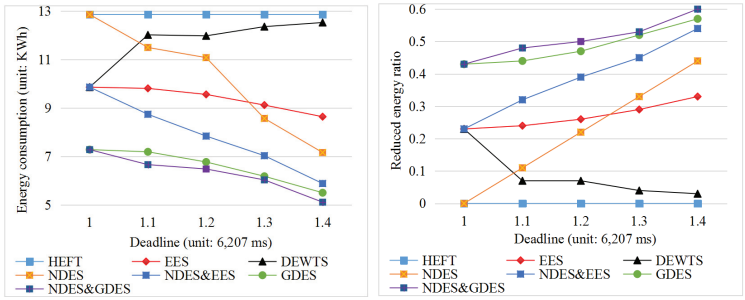
Fig. 11: Results of fast Fourier transform applications with the deadline constraint $D(G) = LB(G) \times 1.4$ for varying numbers of tasks.

As shown in Fig. 12(a), the proposed NDES&GDES algorithm generates the minimum energy consumption values. The superiority of NDES&GDES is always obvious than all other algorithms in different task numbers. Specifically, NDES&GDES reduces energy consumption by 0.69, compared with that of HEFT, when the task number is $|N| = 1151$ (Fig. 11(b)). These results further indicate that synthetically using non-DVFS and global DVFS-enabled energy-efficient scheduling is more energy efficient than using only local DVFS-enabled energy-efficient scheduling algorithm alone.

Combining the results of Figs. 10 and 12, we also calculate out that the average contribution percentage with energy consumption reduction (with respect to the HEFT-generated energy consumption) for non-DVFS (i.e., NDES) and global DVFS-enable (i.e., GDES) algorithm is 60% and 40%, respectively, using NDES&GDES. That is, the NDES algorithm is very effective even in DVFS-unable environments.

6.3 Gaussian Elimination Application

Experiment 3. This experiment is conducted to compare the total energy consumption values of the Gaussian elimination application for varying deadline constraints. We limit the size of the application to $\rho = 48$ (i.e., $|N| = 1175$, which is approximately equal to the number of tasks of the fast Fourier transform application in **Experiment 1**). We also change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.4$ with $LB(G) \times 0.1$ increment, where $LB(G) = 6207$ ms is calculated using the HEFT algorithm. The results are shown in Fig. 12.



(a) Total energy consumptions. (b) Reduced energy ratios.

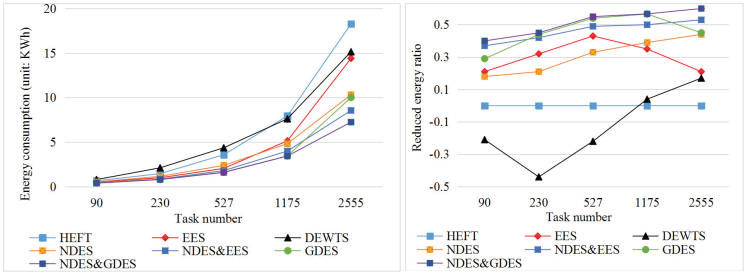
Fig. 12: Results of the Gaussian elimination application with $\rho = 48$ (i.e., $|N| = 1175$) for varying deadlines.

The differences between the fast Fourier transform application in **Experiment 1** and the Gaussian elimination application in **Experiment 3** are as follows: the NDES&EES algorithm can usually generate less energy consumption values than the GDES algorithm for fast Fourier transform application; the reverse law is found for Gaussian elimination application. However, synthetically applying NDES and GDES can still generate the minimum energy consumption values.

The results of Fig. 12(a) show that NDES&GDES can save up to 36.25% and 56.06% of energy compared with state-of-the-art EES and DEWTS. If these algorithms are executed in non-DVFS environments, only NDES and NDES&GDES can work. The results of Fig. 12(b) show the reduced energy ratios with respect to HEFT for NDES are from 0.43 to 0.57. Therefore, the considerable performance further indicates that NDES&GDES is very effective in both non-DVFS and DVFS environments for both high-parallelism and low-parallelism applications.

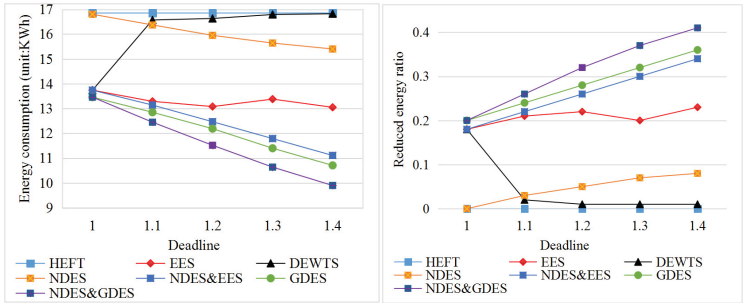
Experiment 4. Similar to **Experiment 2**, this experiment is conducted to compare the total energy consumption values of Gaussian elimination applications for varying numbers of tasks. We limit $D(G)$ to $D(G) = LB(G) \times 1.4$. We change ρ from 13 to 71, that is, the numbers of tasks vary from 90 (small scale) to 2555 (large scale), which are similar to those of fast Fourier transform applications. The results are shown in Fig. 14.

In general, **Experiment 4** illustrates the similar pattern with **Experiment 2** for all the algorithms. A major difference is that the NDES&EES algorithm generates less energy consumption values than the GDES algorithm for fast Fourier transform application (Fig. 12), whereas the results are alternate for Gaussian elimination applications (Fig. 14) in general. In addition, the average contribution percentage with energy consumption reduction (with respect to the HEFT-generated energy consumption) for non-DVFS (i.e., NDES) and DVFS-enable (GDES) approaches is 73% and 27%, respectively, when the task number is 2555 using



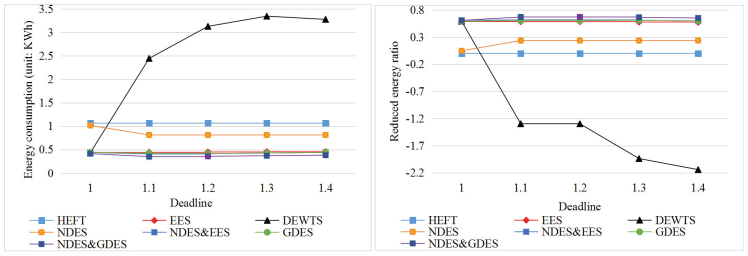
(a) Total energy consumptions. (b) Reduced energy ratios.

Fig. 13: Results of Gaussian elimination applications with the deadline constraint $D(G) = LB(G) \times 1.4$ for varying numbers of tasks.



(a) Total energy consumptions. (b) Reduced energy ratios.

Fig. 14: Results of the low-heterogeneity randomly generated parallel applications with $|N| = 1175$ for varying deadlines.



(a) Total energy consumptions. (b) Reduced energy ratios.

Fig. 15: Results of the high-heterogeneity randomly generated parallel applications with $|N| = 1175$ for varying deadlines.

NDES&GDES. Such results indicate that NDES is necessary, especially for large-scale parallel applications.

6.4 Randomly Generated Parallel Applications

Without loss of generality, we consider parallel application samples that are randomly generated by the task graph generator [23]. Given that the objective platform consists of heterogeneous distributed systems, heterogeneity may also affect the energy of the application. Heterogeneity can be easily implemented for randomly generated parallel applications provided that the heterogeneity factor values are adjusted. We also use the randomly generated parallel applications with 1175 tasks, which are approximately equal to those of fast Fourier transform and Gaussian elimination in **Experiment 1** and **Experiment 3**. Randomly generated parallel applications are generated according to the following parameters:

(1) The average computation time is 50 ms.

(2)The communication-to-computation ratio (CCR) is selected from $\{0.1, 0.5, 1.0, 2.0, 5.0\}$; the smaller the CCR, the higher the communication time, and vice versa.

(3) The shape parameter means the density of the graph and belongs to the range of [0.35, $\sqrt{|N|/3}$].

(4) The heterogeneity factor values belong to the scope of 0 to 1 in the task graph generator, where 0.1 and 1 represent the lowest and highest heterogeneity factors respectively.

Experiment 5. This experiment is conducted to compare the total energy consumption values of the low-heterogeneity randomly generated parallel applications for varying deadline constraints. We limit the size of the applications with $|N| = 1175$ and the heterogeneity factor of 0.1. We change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.4$ with $LB(G) \times 0.1$ increment. The shape parameter's upper bound is $\sqrt{1175/3}$=19.79, and we set that the shape parameters are changed from 1 to 19 with 1 increment. Therefore, there are 19 different applications will be generated and we use the average values as the results shown in Fig. 14.

Experiment 6. This experiment is conducted to compare the total energy consumption values of the high-heterogeneity randomly generated parallel application for varying deadline constraints. The number of tasks is still $|N| = 1175$ and the heterogeneity factor of 1. We also change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.4$ with $LB(G) \times 0.1$ increment. The shape parameters are also changed from 1 to 19 with 1 increment, and the average values are shown in Fig. 15.

Both **Experiment 5** and **Experiment 6** still show that NDES&GDES saves much energy consumption than all other algorithms for randomly generated parallel applications.

The main differences between low-heterogeneity and high-heterogeneity applications are follows:

(1) Low-heterogeneity application generates longer schedule length than high-heterogeneity application under the same scales.

(2) Low-heterogeneity application generates more 10 times energy consumptions than high-heterogeneity application using NDES&GDES. These results indicate that increasing the heterogeneity will decrease the energy consumption.

(3) High-heterogeneity reduces more energy ratios than low-heterogeneity application using NDES&GDES. That is, NDES&GDES is more effective in energy saving for high-heterogeneity than for low-heterogeneity parallel applications.

(4) A special case is that DEWTS even consumes more energy consumptions than HEFT for high-heterogeneity application shown in Fig. 15. The results validate that turning off processors is unrealistic if static energy consumption accounts for only a small part of the total energy consumption.

In summary, results of all the experiments show that the combined NDES&GDES algorithm can save up to 36.25-55.65% of energy than state-of-the-art counterparts under different scales, parallelism, and heterogeneity degrees of parallel applications.

7 Real Platform Verification

This subsection verifies the difference between theoretical results and practical results on a real embedded platform by a small experiment. Based on the Cortex-A20 we have, a cluster consisting of a six homogeneous dual-core processers is built with the 1GB memory, the operating system (OS) Debian 4.7.2-5, and the DVFS Tool CPUFreq. The discrete frequency set is 1.010, 0.960, 0.912, 0.864, 0.816, 0.768, 0.744, 0.720, 0.696, 0.672, 0.648, 0.600, 0.528, 0.480, 0.408, 0.384, 0.360, and 0.336 GHz.

The verification process contains 3 steps: test of the processor core's power parameters, theoretical experiments based on tested power parameters, and practical experiments on the real platform.

TABLE 9: Core's dynamic power values (unit: W) at different frequencies (unit: GHz).

Frequency	1.01	0.960	0.912	0.864	0.816	0.768	0.744	0.720	0.696	0.672	0.648	0.600	0.528	0.480	0.408
Practical P_{ind}	0.15	0.15	0.125	0.10	0.10	0.10	0.075	0.075	0.075	0.05	0.05	0.05	0.035	0.035	0.035
Theoretical P_{ind}	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
Practical P_{d}	1.05	0.90	0.80	0.70	0.60	0.50	0.45	0.40	0.35	0.35	0.30	0.25	0.15	0.15	0.10
Theoretical P_{d}	1.030	0.885	0.759	0.645	0.543	0.453	0.412	0.373	0.337	0.303	0.272	0.216	0.147	0.111	0.068

Step (1) **Test of the processor core's power parameters**. Given the use of OS and other components, the tested power includes the core's frequency-dependent dynamic power P_{d}, the core's frequency-independent dynamic power P_{ind}, the core's static power P_{s}, and other components' power:

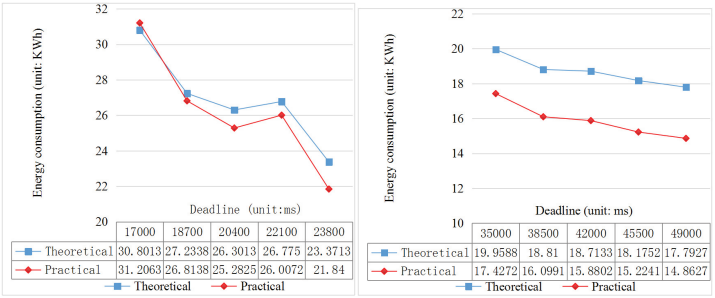
1) The core's frequency-dependent dynamic power is $P_{\text{d}} = C_{\text{ef}} f^m$, where $C_{\text{ef}} = 1$ and $m = 3$ in the platform by data fitting.

2) We tested that the core's static power and other components' power are difficult to separate in this platform, so we treat them as a whole denoted with P_{s} according to the model of Eq. (1). P_{s} is tested as 1.35 W when the processor is idle and the frequency is the minimum available frequency of f_{min} =0.336 GHz.

3) We tested that the total idle power is P_{idle} =1.5 W when no tasks are running in the maximum frequency of f_{max} =1.01 GHz. Therefore, the core's frequency-independent dynamic power P_{ind} is P_{idle}-P_{s} = 1.5-1.35=0.15 W.

4) The minimum dynamic energy-efficient frequency f_{ee} calculated by Eq. (2) is $f_{\text{ee}} = \sqrt[3]{\frac{0.15}{(3-1)\times 1}} = 0.422$ GHz. Considering that 0.422 GHz does not exist in the discrete frequency set, we take the nearest and largest 0.480 GHz from the 0.422 GHz as f_{ee}. That is, we have $f_{\text{ee}} = 0.480$ GHz.

Step (2) **Theoretical experiments based on tested power parameters**. We first do the simulation experiments based on tested power parameters. We take the small-scale fast Fourier transform ($|N|$=96) and Gaussian elimination ($|N|$=90) applications to observe the results using NDES&GDES, respectively. Considering that the static energy consumption is too large due to the use of OS and other components, we only record the dynamic energy consumption. We set the execution time with the maximum frequency as 1000 ms. We change $D(G)$ from $LB(G) \times 1.0$ to $LB(G) \times 1.4$ with $LB(G) \times 0.1$ increment, where $LB(G) = 17000$ ms for fast Fourier transform and $LB(G) = 35000$ ms for Gaussian elimination, respectively. The results denoted with "Theoretical" are shown in Fig. 16.



(a) Fast Fourier transform. (b) Gaussian elimination.

Fig. 16: Theoretical and practical results.

Step (3) **Practical experiments on the real platform**. We create applications by invoking assembly instructions using C language. We first test the instructions per second (IPS) in the platform, where the IPS is 0.67 Giga by recording the actual number of assembly instructions. We then create fast Fourier transform and Gaussian elimination applications according to individual structures by setting 0.67 Giga instructions for each task. In other words, the execution time with the maximum frequency is 1000 ms for each task. We then execute the applications according to the assigned processor, start time, and end time of each task generated by the simulation experiments in Step 2. In other words, theoretical and practical experiments have the same task assignments. The results denoted with "Practical" are shown in Fig. 16.

Fig. 16 shows that the trend of the theoretical results and practical results are consistent and the results are similar. A surprising phenomenon is that the actual energy consumptions are generally lower than the corresponding theoretical values. To find out the reason, we record the core's dynamic power values at different frequencies in the real platform, as shown in Table 9, where corresponding theoretical values are also shown.

An important finding is that the P_{ind} decreases with the decrease of frequency. One reason is that P_{ind} still contains the power of other components, which is frequency dependent and is hard to separate. Besides P_{ind}, practical P_{d} values also have some differences with the theoretical P_{d} values, which are calculated by f^3. We can see that the practical P_{d} values are slightly higher than the theoretical P_{d} values. On the basis of the above values, we can find why the actual energy consumptions are generally lower than the corresponding theoretical values. The reason is that P_{ind} is decreased rather than fixed with the decrease of frequency. In any case, we conclude that the actual results are basically the same as the theoretical results by real platform verification.

8 Conclusions

We have presented an effective energy consumption minimization approach called NDES&GDES for real-time parallel applications with precedence constrained tasks on heterogeneous distributed embedded systems. First, NDES&GDES is implemented by combining the presented non-DVFS and global DVFS-enable energy-efficient scheduling algorithms. Second, NDES&GDES demonstrates more effective energy consumption minimization than the state-of-the-art algorithms. Third, NDES&GDES is highly efficient for different types of real-time parallel applications under different scales, parallelism, and heterogeneity degrees of parallel applications.

Acknowledgments

The authors are thankful to three anonymous reviewers for their criticism and comments on the manuscript. This work was partially supported by the National Key Research and Development Plan of China under Grant No. 2016YFB0200405, 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] J. Liu, Q. Zhuge, S. Gu, J. Hu, G. Zhu, and E. H.-M. Sha, "Minimizing system cost with efficient task assignment on heterogeneous multicore processors considering time constraint," *IEEE Trans. Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2101–2113, 2014.

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

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

[4] J. Niu, C. Liu, Y. Gao, and M. Qiu, "Energy efficient task assignment with guaranteed probability satisfying timing constraints for embedded systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2043–2052, 2014.

[5] M. Naghibzadeh, "Modeling and scheduling hybrid workflows of tasks and task interaction graphs on the cloud," *Future Generation Computer Systems*, vol. 65, pp. 33–45, Dec. 2016.

[6] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, 2012.

[7] K. Li, "Energy and time constrained task scheduling on multiprocessor computers with discrete speed levels," *J. Parallel and Distributed Computing*, vol. 95, pp. 15–28, 2016.

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

[9] 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 Computing*, vol. 14, no. 1, pp. 55–74, 2016.

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

[11] B. Zhao, H. Aydin, and D. Zhu, "Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints," *ACM Trans. Design Automation of Electronic Systems*, vol. 18, no. 2, pp. 99–109, 2013.

[12] X. Xiao, G. Xie, R. Li, and K. Li, "minimizing schedule length of energy consumption constrained parallel applications on heterogeneous distributed systems," in *Proc. 2016 14th IEEE Int. Symp. on Parallel and Distributed Processing with Applications*. IEEE Computer Society, 2016, pp. 1471–1476.

[13] C. F. Kuo and Y. F. Lu, "Task assignment with energy efficiency considerations for non-dvs heterogeneous multiprocessor systems," *ACM Sigapp Applied Computing Review*, vol. 14, no. 4, pp. 8–18, 2015.

[14] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embedded Computing Systems*, vol. 15, no. 1, pp. 303–307, 2016.

[15] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Trans. Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1447–1464, 2013.

[16] J. Singh, S. Betha, B. Mangipudi, and N. Auluck, "Contention aware energy efficient scheduling on heterogeneous multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1251–1264, 2015.

[17] 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.

[18] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[19] G. Bernat, A. Colin, and S. M. Petters, "Wcet analysis of probabilistic hard real-time systems," in *Proc. 23rd IEEE Real-Time Systems Symp.* IEEE, 2002, pp. 279–288.

[20] X. Yong, Z. Gang, C. Yang, R. Kurachi, H. Takada, and L. Renfa, "Worst case response time analysis for messages in controller area network with gateway," *IEICE TRANSACTIONS on Information and Systems*, vol. 96, no. 7, pp. 1467–1477, Jul. 2013.

[21] K. M. Tarplee, R. Friese, A. A. Maciejewski, H. J. Siegel, and E. K. Chong, "Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques," *IEEE Trans. Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1633–1646, 2016.

[22] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[23] [Online]. Available: https://sourceforge.net/projects/taskgraphgen/

Guoqi Xie received his Ph.D. degree in computer science and engineering from Hunan University, China, in 2014. He is currently an Assistant Professor and a Yuelu Scholar at Hunan University. He was a Postdoctoral Researcher at Nagoya University, Japan, from 2014 to 2015, and at Hunan University, from 2015 to 2017. He has received the best paper award from ISPA 2016. His major interests include embedded and cyber-physical systems, parallel and distributed systems, software engineering and methodology. He is a member of IEEE, ACM, and CCF.

Gang Zeng is an Associate Professor at the Graduate School of Engineering, Nagoya University. He received his Ph.D. degree in Information Science from Chiba University in 2006. From 2006 to 2010, he was a Researcher, and then Assistant Professor at 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 IEEE and IPSJ.

Xiongren Xiao is Ph.D. candidate and Assistant Professor in College of Computer Science and Electronic Engineering, Hunan University, China. He has received the best paper award from ISPA 2016. His main research interests include power-aware computing, energy-efficient computing, parallel and reliable systems, and embedded systems. He is member of ACM and CCF.

Renfa Li is a Professor of computer science and electronic engineering, and the Dean of 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 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 IEEE, and a senior member of 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 over 485 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 *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Sustainable Computing*. He is an IEEE Fellow.