Cluster Comput (2014) 17:537–550
DOI 10.1007/s10586-013-0297-0

Energy-aware task scheduling in heterogeneous computing environments

Jing Mei · Kenli Li · Keqin Li

Received: 18 March 2013 / Revised: 12 June 2013 / Accepted: 12 August 2013 / Published online: 6 September 2013

Abstract Efficient application scheduling is critical for achieving high performance in heterogeneous computing (HC) environments. Because of such importance, there are many researches on this problem and various algorithms have been proposed. Duplication-based algorithms are one kind of well known algorithms to solve scheduling problems, which achieve high performance on minimizing the overall completion time (makespan) of applications. However, they pursuit of the shortest makespan overly by duplicating some tasks redundantly, which leads to a large amount of energy consumption and resource waste. With the growing advocacy for green computing systems, energy conservation has been an important issue and gained a particular interest. An existing technique to reduce energy consumption of an application is dynamic voltage/frequency scaling (DVFS), whose efficiency is affected by the overhead of time and energy caused by voltage scaling. In this paper, we propose a new energy-aware scheduling algorithm with reduced task duplication called Energy-Aware Scheduling by Minimizing Duplication (EAMD), which takes the energy consumption as well as the makespan of an application into consideration. It adopts a subtle energy-aware method to search and delete redundant task copies in the schedules generated by duplication-based algorithms, and it is easier to operate than DVFS, and produces no extra time and energy consumption. This algorithm not only consumes less energy but also maintains good performance in terms of makespan compared with duplication-based algorithms. Two kinds of DAGs, i.e., randomly generated graphs and two real-world application graphs, are tested in our experiments. Experimental results show that EAMD can save up to 15.59 % energy consumption for HLD and HCPFD, two classic duplication-based algorithms. Several factors affecting the performance are also analyzed in the paper.

Keywords Directed acyclic graph · Duplication-based algorithm · Energy-aware scheduling · Heterogeneous computing system

J. Mei · K. Li (✉) · K. Li
College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China
e-mail: lkl@hnu.edu.cn

J. Mei
e-mail: jingmei1988@163.com

K. Li
National Supercomputing Center in Changsha, Changsha, Hunan 410082, China

K. Li
Department of Computer Science, State University of New York, New Paltz, NY 12561, USA
e-mail: lik@newpaltz.edu

1 Introduction

In the past decade, more and more attention has focused on the problem of scheduling applications on heterogeneous computing systems. A heterogeneous computing (HC) system is defined as a suite of distributed computing machines with different capabilities which are interconnected by different high speed links and are utilized to executed parallel applications [1, 2]. Many task scheduling algorithms are proposed for HC systems, and their performance is evaluated with the only criterion, i.e., the schedule length (makespan). Among those proposed algorithms, duplication-based algorithms are a kind of efficient algorithms, which assign some tasks to several processors to reduce the communication between tasks, hence minimizing the makespan. However, redundant duplications lead

to large overhead of energy consumption. In recent years, with the growing advocacy for green computing systems, energy conservation has become an important issue and gained a particular interest. To overcome the drawbacks of duplication-based algorithms but retain their advantages, we present a new task scheduling algorithm on HC systems in this paper, which can decrease energy consumption while not degrading the makespan of duplication based algorithms.

There are much research focusing on the energy-aware problem in task scheduling, and various techniques have been developed. Two general and widely used techniques to reduce energy consumption at system level scheduling are dynamic power management (DPM) and dynamic voltage/frequency scaling (DVFS). DPM turns idle components off to reduce power consumption, and DVFS-enabled processors scale down their voltages and clock frequencies during idle periods. These two techniques are efficient in some cases. Several energy-aware algorithms are proposed based on these two techniques, such as [3–10]. However, the high overhead of energy and time is the drawback of the DPM and DVFS techniques [11, 12], which would degrade the performance in terms of makespan to certain extent. Hence, we adopt a different mechanism instead of DPM and DVFS to reduce energy consumption in this paper.

In our paper, we assume that all information of an application including task execution times, the sizes of data communicated between tasks, and task dependencies are known *a priori* for static scheduling. Static task scheduling takes place during compile time before task execution. Once the schedule is determined, the tasks can be executed following the orders and assignments. Task scheduling is to map tasks of an application to processors, so that precedence requirements are satisfied and the minimal makespan is achieved [13]; however, it is in general NP-hard [14, 15]. Therefore, heuristics can be used to obtain sub-optimal schedules. Task scheduling has been extensively studied and various heuristics have been proposed in the literature [16–24]. The general task scheduling algorithms can be classified into a variety of categories, such as list scheduling algorithms, cluster algorithms, and duplication-based algorithms, and so on.

As we mentioned before, duplication-based algorithms have the highest performance in terms of makespan compared with other kinds of algorithms. The idea of duplication-based algorithms is to schedule a task graph by mapping some of its tasks redundantly, which reduces the communication between tasks. However, duplication-based algorithms improve the makespan at the cost of higher resource waste and energy consumption. In a traditional schedule, each task is assigned to a processor and is executed only one time, while a schedule generated by a duplication-based algorithm executes some tasks more than one time, resulting in a large increase of energy consumption. Actually, the finish time of a task is mainly determined by its important immediate parent, so it is unnecessary to finish all of its parents as early as possible via duplication. That is to say, some duplications can be removed from a schedule, which does not affect the overall makespan. Those removable duplications are defined as redundant copies of tasks. In this paper, for a schedule generated by a duplication-based algorithm, we try to explore (1) how to search for redundant copies of tasks; (2) and how to delete them but not affect the performance of the original schedule. The proposed methods can be applied to the output of any duplication-based algorithms.

The main contributions of this paper are summarized as follows.

- We propose a subtle energy-aware method, which not only is easier to operate than both DPM and DVFS, but also produces no overhead of time and energy. In addition, the performance in terms of makespan is not degraded compared with the duplication-based algorithms.
- Experiments are given to verify that the proposed algorithm can reduce a large amount of energy consumption compared with the duplication-based algorithms while not sacrificing the performance of makespan.
- The factors affecting the performance of our algorithm are analyzed.

The remainder of this paper is organized as follows. In Sect. 2, some related work are reviewed, which include different scheduling heuristics on heterogeneous systems, power reduction techniques, and several energy-aware scheduling algorithms. In Sect. 3, we define the problem and present the related models. In Sect. 4, we give a detailed description of the EAMD algorithm and an analysis of its time complexity. An example is also provided in this section to explain our algorithm better. The experimental results are presented and some analysis is given in Sect. 5. Section 6 concludes the work of our paper and provides an overview of future research.

2 Related work

The existing task scheduling algorithms can be classified to a variety of categories, such as list scheduling algorithms, cluster algorithms, duplication-based algorithms, and some other algorithms [25]. The list scheduling algorithms provide good quality of schedules and their performance is comparable with other categories at lower time complexity. Some examples are dynamic critical-path (DCP) [26], heterogeneous earliest finish time (HEFT) [20], critical path on a processor (CPOP) [20], and the longest dynamic critical path (LDCP) [18]. The cluster algorithms merge tasks

in a graph to an unlimited number of clusters, and tasks in a cluster are scheduled on the same processor. Some examples in this category are clustering for heterogeneous processors (CHP) [27], clustering and scheduling system(CASS) [28], and objective-flexible clustering algorithm (OFCA) [29]. The idea of duplication-based algorithms is to schedule a task graph by mapping some tasks redundantly, which reduces the interprocessor communication overhead. There are many duplication-based algorithms, for examples, selective duplication (SD) [24], heterogeneous limited duplication (HLD) [19], heterogeneous critical parents with fast duplicator (HCPFD) [22], and heterogeneous earliest finish with duplication (HEFD) [30]. This kind of algorithms can reduce the makespan effectively, but they do not take energy efficiency into consideration.

Energy efficiency has become an important issue to be considered in the field of high-performance computing. There are some energy-aware scheduling algorithms proposed recently. Two typical techniques usually adopted to reduce energy consumption at system level scheduling are dynamic power management (DPM) and dynamic voltage/frequency scaling (DVFS), respectively. DPM turns idle components off to reduce the power consumption. However, a large amount of energy and time overheads will be produced when the components are rebooted; hence, DPM is working only when the idle times are long enough. The DPM technique is mostly applied to laptops and PDAs. For the scheduling of an application, the idle time between two task execution is little, and DPM is not suitable.

Another technique, DVFS, has been proven to be a very promising technique with its demonstrated capability for energy savings [3–10]. With the growing advocacy for green computing systems, many DVFS-enabled processors are manufactured to cater to the requirements, such as Transmeta's Crusoe [31], Intel Speed Step [32], and AMD K6 [33], etc. DVFS-enabled processors are to scale down their voltages and clock frequencies when the peak performance is unnecessary, further reducing power consumption and heat generation. The power consumption by a CPU is proportional to voltage quadratically, and voltage and frequency can vary considerably. So a decrease of voltage can lead to a significant decrease of power consumption. Therefore, the DVFS technique can reduce energy dissipation efficiently. Whereas, DVFS has its disadvantages as DPM. One particular disadvantage is the energy and time overheads. Generally speaking, when processors are scaled between two different voltages, the energy overhead and time delay are related to the difference of voltages, and the overheads, especially the time delay, could affect the overall performance in terms of makespan, hence cannot be neglected. The calculation equations of the overheads of energy and time can be found in [11].

In this paper, we take both energy efficiency and makespan into consideration. A new method is adopted instead of DVFS and DPM, which can reduce energy consumption efficiently but not degrade the performance in terms of makespan compared with the existing duplication-based algorithms. As we said before, duplication-based algorithms obtain better performance in terms of makespan at the cost of significant increase of energy consumption. Through the analysis of the duplication-based algorithms, we find that some copies of tasks can be deleted without affecting the task precedence constraint, which are called redundant copies. Deleting redundant copies can reduce resource waste and energy consumption. Therefore, the proposed algorithm is to search and delete the redundant copies of tasks in the schedules generated by duplication-based algorithms. The precondition is that the performance of an original schedule is not worsen. The proposed algorithm can be applied to the output of any duplication-based algorithms.

3 Models

A scheduling system consists of a target computing environment, an application, and performance criteria of scheduling. In the following subsections, we introduce the computing system model, the application model, and performance criteria adopted in the paper.

3.1 Computing system model

A computing system considered in this paper is heterogeneous. The heterogeneous systems can be classified into two categories [19], i.e.,

- mixed heterogeneity computing system model (MHM);
- fixed heterogeneity computing system model (FHM).

In the MHM, the target system consists of a mixed suite $P = \{p_k : k = 0, 1, \ldots, n-1\}$ of n processors, each is best suited to process a particular type of program code. Therefore, the execution time of a task v_i on processor p_k will depend on how well the architecture of p_k matches v_i's processing requirement. A task scheduled on its best suited processor will spend less execution time than on a less suited processor. The best processor for one task may be the worst processor for another task. This type of model is described in [34] and used in [16, 20–22].

In the FHM, the target system also consists of n processors $P = \{p_k : k = 0, 1, \ldots, n-1\}$. For one processor, it executes tasks with the same processing rate no matter what types they are. For different processors, however, their processing rates are different from each other. For example, given two tasks v_i and v_j, their execution times on two processors p_k and p'_k are $w_{i,k}, w_{i,k'}$ and $w_{j,k}, w_{j,k'}$ respectively, where $w_{i,k} \neq w_{i,k'}, w_{j,k} \neq w_{j,k'}$, but $w_{i,k}/w_{i,k'} = w_{j,k}/w_{j,k'}$ (where $w_{i,k}$ refers to the execution time of a task v_i on processor p_k). This type of model is used in [30, 35].

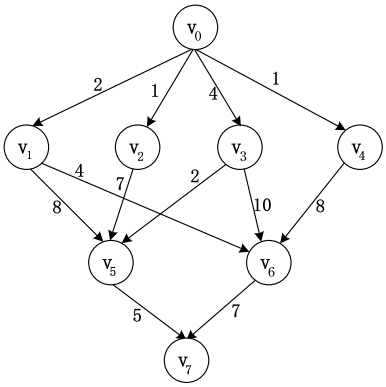


Fig. 1 A simple DAG representing an application graph with precedence constraint

The computing system adopted in this paper is based on the MHM, as it is used more commonly than the FHM, and algorithms are compared based on the MHM. In addition, algorithms designed for the MHM are applicable to the FHM as well, since the FHM is a special case of the MHM.

3.2 Application model

An application is represented as a *directed acyclic graph* (DAG) with both node and edge weights, denoted by $G(V, E, [w_{i,k}], c)$. A set of nodes $V = \{v_i \mid 0 \leq i \leq m-1\}$ represent the tasks in the application, and a set of directed edges E represent dependencies among tasks. $[w_{i,k}]$ is an $m \times n$ matrix of computation times, where $w_{i,k}$ represents the computation time of task v_i when it is assigned to processor p_k, for $0 \leq i \leq m-1$ and $0 \leq k \leq n-1$. The average computation cost of task v_i is defined as

$$\overline{w_i} = \frac{1}{n} \sum_{k=0}^{n-1} w_{i,k}. \tag{1}$$

Let $c_{i,j}$ be the weight associated with edge $e_{i,j}$, which represents the required communication time to send data from v_i to v_j, where v_i is called a parent of v_j, and v_j is called a child of v_i.

Figure 1 gives the DAG of an application, and Table 1 lists the computation time matrix $[w_{i,k}]$.

A task having no parent is called an *entry task*, such as task v_0 in Fig. 1. A task having no child is called an *exit task*, such as v_7. In this paper, we only discuss the scheduling of DAGs with single-entry and single-exit tasks. For those DAGs with multi-entry and multi-exit tasks, we can transform them by adding zero-cost pseudo entry/exit tasks with zero-cost edges, which do not affect the schedule.

Table 1 Computation cost matrix $[w_{i,k}]$

Task node	p_0	p_1	p_2	p_3	$\overline{w_i}$	$rank_u$
v_0	1	1	2	1	1.25	32.00
v_1	3	2	4	2	2.75	19.75
v_2	5	6	3	4	4.50	20.50
v_3	2	4	4	2	3.00	26.75
v_4	4	8	7	8	6.75	28.50
v_5	3	3	1	2	2.25	9.00
v_6	5	5	5	5	5.00	13.75
v_7	1	2	2	2	1.75	1.75

3.3 Performance criteria

3.3.1 Makespan

The objective of task scheduling is to find an assignment of the tasks onto the processors of the target system, which results in the fastest possible execution, while respecting the precedence constraints expressed by the edges. It is natural that the makespan is selected as the main criterion to measure the performance of scheduling algorithms. In a given schedule, $st(v_i, p_k)$ and $ft(v_i, p_k)$ represent the *start time* and *finish time* of task v_i on p_k, respectively. Because preemptive execution is not allowed, $ft(v_i, p_k) = st(v_i, p_k) + w_{i,k}$. Makespan is the overall finish time of the whole application, which is denoted as

$$makespan = ft(v_{exit}). \tag{2}$$

3.3.2 Energy consumption

The second objective of the proposed algorithm in this paper is to reduce energy consumption of a valid schedule generated by a duplication-based algorithm without degrading its makespan. Thus, we introduce the energy consumption as the second performance criterion in this paper. Energy consumption of a computing system equals to the product of the power consumption and the execution time of processors. Power consumption is related to the design technology of a processor and it is different according to the state of the processor. A processor has two states. One is the busy state, in which some tasks are being executed by the processor; the other is the idle state, in which the processor is idle and no task is executed. The power consumption of a processor in the busy state is much different from that in the idle state. Table 2 presents the power consumption parameters of Intel XScale PXA270 processor [36], which is also the power model used in our paper.

PXA270 processor is a kind of processor with the mechanism of dynamic voltage/frequency scaling (DVFS). It has six frequency levels. The frequency of 624 MHz listed in

Table 2 Power consumption of PXA270 [36]

Frequency	Idle power	Busy power
624 MHz	260 mW	925 mW

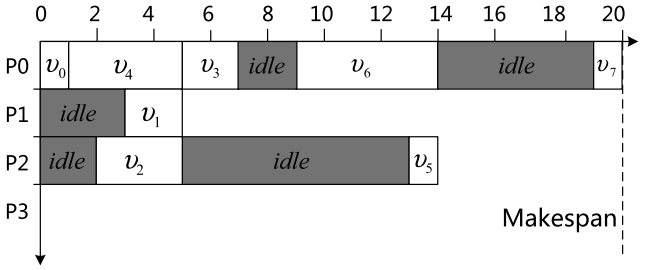


Fig. 2 A schedule of the DAG in Fig. 1

Table 2 is the maximum frequency of PXA270. Because our algorithm does not adopt the mechanism of DVFS, we assume that all processors are running at a fixed frequency of 624 MHz. According to the data sheet of Intel XScale PXA270 processor [36], the power consumption of a processor in the idle state P_{idle} is 260 mW and that in the busy state P_{busy} is 925 mW. The power consumption in the busy state is much greater than that in the idle state. The energy consumption of a processor is calculated by:

$$E_{total} = P_{busy}t_{busy} + P_{idle}t_{idle}, \tag{3}$$

where t_{busy} and t_{idle} are the periods of a processor in the busy and the idle states, respectively. From Eq. (3) we can see that the energy consumption of a given computing system is mainly determined by execution times of processors since the power consumption is fixed.

Figure 2 gives a schedule of the application given in Fig. 1. The makespan is 20 ms. Assume that the processors without tasks are turned off and consume no energy. The power consumption of processors in the busy state and the idle state are 925 mW and 260 mW, respectively. The periods of processors in the busy state and the idle state are 19 ms and 20 ms, respectively. Therefore, the total energy consumed by the schedule is $19 \times 925 + 20 \times 260 = 22.775$ mJ.

4 The proposed algorithm

The proposed algorithm, named *Energy-Aware scheduling algorithm by Minimizing Duplication* (EAMD for short), is described in detail in this section. EAMD is essentially an improved algorithm for an existing duplication-based algorithm. The schedule generated by a duplication-based algorithm is the input of the EAMD algorithm. The objective of EAMD is to reduce the energy consumption of an input schedule without degrading the original performance in terms of makespan.

The input schedule is denoted by S, which consists of a series of mapping information $(v_i, p_k, st_{i,k}, ft_{i,k})$, where $(v_i, p_k, st_{i,k}, ft_{i,k})$ represents that task v_i is assigned onto processor p_k, and $st_{i,k}$ and $ft_{i,k}$ are the start and finish times of v_i on p_k, respectively. In a valid input schedule S, a task, except the exit task, would be assigned onto more than one processor due to duplication.

A duplication-based algorithm is always greedy. It assigns a task to the processor on which the task finishes at the earliest time. The greedy feature leads to a common drawback of duplication-based algorithms. That is, they only consider to optimize the execution of the current task, but neglect its effect on the execution of its children. When considering the children, the best processor for the current task is no longer the best. In order to optimize the execution of the children, a duplication mechanism is adopted and a parent task is duplicated on the objective processors of children. Therefore, the original assignment of the parent might become useless, which is defined as a redundant copy. Those redundant copies can be removed from schedule S. In addition, those deleted copies might rely on the duplication of their own parents further, and their relied duplication copies can be deleted as well.

Before giving the detailed description to our algorithm, it is necessary to introduce some definitions, which can help us to understand the algorithm precisely.

Definition 1 A schedule S of an application is feasible if and only if all tasks of the application are assigned to processors and all precedence constraints between tasks are satisfied.

To delete redundant copies from a feasible schedule but maintain its feasibility, first and foremost, it must be known which kind of task copies are the latent redundancies. According to the first condition of Definition 1, each task must be assigned to one processor. Hence, tasks which are scheduled one time in a schedule cannot be deleted undoubtedly. For a given schedule S, the tasks are divided into three categories according to the number of task copies:

- Single-copy task: A task that is scheduled one time in S;
- Dual-copy task: A task that is scheduled two times in S;
- Multi-copy task: A task that is scheduled more than two times in S.

According to Definition 1, only dual-copy tasks and multi-copy tasks have opportunity to be redundant and deleted. In the following, we discuss how to search redundant copies for these two kinds of tasks, respectively.

For a task assigned to several processors, the assignment of each copy has its particular objective. In general, the copy

of a task that is assigned first is always the one that finishes the earliest among all copies, which is to ensure that the task is executed at the optimal time. The other copies are assigned to reduce the communication with its children. A definition is given in the following to distinguish the two kinds of copies.

Definition 2 In a schedule, the copy with the earliest finish time is called its *original copy*, and the others are called *duplicated copies*.

We list all copies of task v_i in the nondecreasing order of finish times, denoted by $S(v_i) = \{v_i^1, \ldots, v_i^j, \ldots, v_i^l\}$, where v_i^j represents the jth earliest copy of task v_i. According to Definition 2, v_i^1 is the original copy, and the others are duplicated copies. Assume that v_i^j is a copy assigned to processor p_k, and there exists a child v_c which is executed on the same processor p_k and receives data from v_i^j on p_k. In this case, we say that v_i^j is the *local parent* of v_c, denoted by $pare_l(v_c, p_k) = v_i^j$, and v_c is the *local child* of v_i^j, denoted by $child_l(v_i^j, p_k) = v_c$.

To determine if a copy of task v_i can be deleted, the key is to decide if the precedence constraints with all children are guaranteed, that is, if the other copies of v_i can afford data to all its children. The determination methods used for the dual-copy tasks and the multi-copy tasks are different and will be discussed separately in the following.

If v_i is a dual-copy task, it has two copies, i.e., the original copy and a duplicated copy. That means, there is only one of its children executed in advance due to the duplication of v_i, and the aim of the duplicated copy is to provide data for its local child. Hence, the duplicated copy cannot be deleted. Whether the original copy can be deleted depends on whether the duplicated copy can afford the data needed by its other children. The original copy can be deleted if the duplicated copy v_i^2 satisfies the following condition:

$$\begin{aligned} &ft\left(v_i^2, p_k\right) + \overline{c_{i,j}} \le st(v_j, p_l), \\ &\forall v_j \in child(v_i),\ p_l \in P, \end{aligned} \tag{4}$$

where $ft(v_i^2, p_k)$ represents the finish time of the duplicated copy of task v_i assigned onto processor p_k, and $child(v_i)$ is the set of children of v_i. If $p_k = p_l$, $\overline{c_{i,j}} = 0$. When the duplicated copy of task v_i can provide the data of all children and does not violate the precedence constraints, the original copy can be deleted.

If v_i is a multi-copy task, the analysis is more complex than that of a dual-copy task. Assume that v_i has a copy on processor p_k, and it can offer data of all its children if Eq. (4) is satisfied, the other copies of v_i can be deleted. However, only one copy cannot supply the needed data of all children in most cases. It is more possible that there exists a combination of several copies of v_i which can provide all data needed by its children, and the other copies excluding the combination can be deleted. Let $S(v_i) = \{v_i^1, v_i^2, \ldots, v_i^l\}$ be l copies of task v_i in a schedule. There exists an available combination $CS \subseteq S(v_i)$ if it satisfies:

$$\begin{aligned} &\forall v_j \in child(v_i),\ \text{where } v_j \text{ is assigned to } p_l \in P, \\ &\exists v_i^* \in CS,\ \text{where } v_i^* \text{ is assigned to } p_k \in P, \\ &\text{such that } ft\left(v_i^*, p_k\right) + \overline{c_{i,j}} \le st(v_j, p_l). \end{aligned} \tag{5}$$

If there exists this combination, we delete all other copies of v_i except the combination; otherwise, we turn to deal with the next task. Whereas, how to find the available combination is a complex issue to be solved, and it is inadvisable to traverse all combinations of v_i's copies. We give two steps as follows to find an available combination of v_i.

Steps of finding redundancy for multi-copy tasks:

1. If $child_l(v_i^1)$, the set of local children of the original copy v_i^1, is empty, set all copies except v_i^1 as a combination, denoted by $CS = \{v_i^2, \ldots, v_i^l\}$. Find out the task copy which finishes the earliest among CS, saying v_i^r, and determine if it can afford communication requests of all its children which have no local copy of v_i (if Eq. (4) is satisfied). If true, delete v_i^1.
2. If $child_l(v_i^l)$ is not empty, find out the local children for each copy v_i^j $(1 \le j \le l)$, denoted by $child_l(v_i^j)$. For each copy v_i^j, determine if the original copy v_i^1 can afford all data needed by its local children $child_l(v_i^j)$. If true, delete v_i^j.

While deleting the redundant copies of tasks, we traverse tasks in the nondecreasing order of *upward ranks* of tasks to make sure that all children have been processed when dealing with a parent. The upward rank $rank_u$ is computed by traversing a DAG upward starting from the exit task, which is recursively defined by

$$rank_u(v_i) = \overline{w_i} + \max_{v_j \in child(v_i)} \left(c_{i,j} + rank_u(v_j)\right), \tag{6}$$

where $child(v_i)$ is the set of immediate children of task v_i in the DAG. For the exit task v_{exit}, the upward rank is

$$rank_u(v_{exit}) = \overline{w_{exit}}. \tag{7}$$

A complete description of the EAMD algorithm is given in Algorithm 1.

The EAMD algorithm is given as Algorithm 1. Its input is a schedule S generated by any duplication-based algorithm. For each input S, all tasks are traversed in a nondecreasing order of $rank_u$ (line 1). If task v_i has two copies, its redundancy is searched following the steps shown in lines 3–6. If task v_i is assigned to more than two processors, its process is shown in line 9.

Algorithm 1 EAMD algorithm

Require: A schedule S produced by a duplication-based algorithm.
Ensure: Schedule S after deleting the redundant copies.

```
for each task v_i in nondecreasing order of rank_u do
  if v_i is a dual-copy task then
    order all copies of v_i in nondecreasing ft, denoted by L = {v_i^1, v_i^2}
    if the duplicated copy v_i^2 of v_i satisfies Eq. (4) then
      delete the original copy v_i^1 from S, so do its relied duplicated copies on the same processor
    end if
  else
    if v_i is a multi-copy task then
      follow the steps of finding redundancy for multi-copy tasks introduced above, and delete all found redundant copies from S
    end if
  end if
end for
```

Assume that v_i^1 is the original copy of task v_i, and it is removed from a schedule by our algorithm. Due to the duplication mechanism adopted by duplication-based algorithms, when assigning v_i for the first time, which is the original copy, it is possible that its parents are duplicated to bring forward its finish time. If the original copy is removed from a schedule, those duplicated copies of its parents become redundant and can be removed from the schedule as well.

4.1 Time-complexity analysis

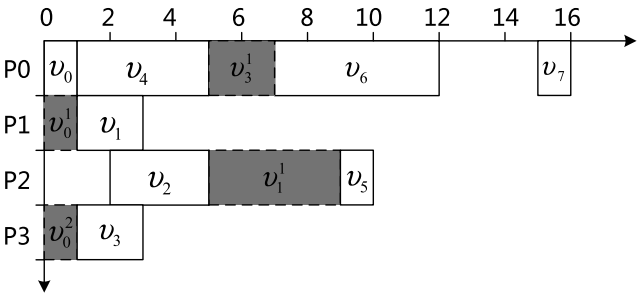
The time complexity of a task scheduling algorithm is usually expressed in terms of the number of tasks $|V|$, the number of edges $|E|$, and the number of processors $|P|$. The time complexity of EAMD is analyzed as follows.

Before optimizing the input schedule, a priority queue is determined first, which can be done in $O(|V|\log|V|)$ time. All tasks are considered to search redundant copies. For a dual-copy task, all its children are calculated to decide if the original copy can be deleted. For a multi-copy task, all its children are calculated to decide if the original copy can afford the data that they need. The time complexity of the decision phase is $O(|E|)$. Since $|E|$ is bounded by $O(|V|^2)$, the overall time complexity of EAMD is $O(|V|^2)$.

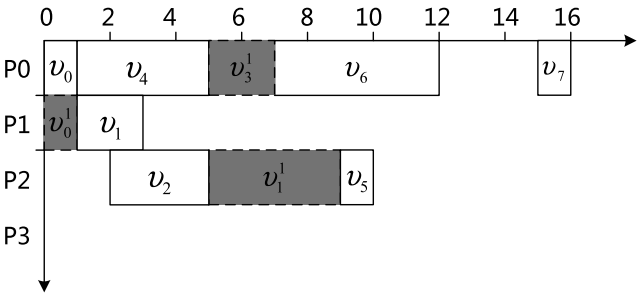
4.2 An illustrative example

Figure 3 gives schedules of the application in Fig. 1 using the HLD algorithm and the EAMD algorithm. Compared with HLD, EAMD deletes the redundant copies of tasks so that the energy consumption is reduced. The detailed process is described as follows.

The schedule generated by HLD is the input of the EAMD algorithm, shown in Fig. 3(a). First, a priority queue L is constructed in the nondecreasing order of upward ranks.



(a) A schedule generated by the HLD algorithm



(b) A schedule generated by the EAMD algorithm

Fig. 3 Comparison between HLD and EAMD algorithms

For the given example, $L = \{v_7, v_5, v_6, v_1, v_2, v_3, v_4, v_0\}$. Second, we select tasks from L one by one, and determine if deletion operation can be done for each task. The first selected task is v_7, and it does not rely on duplication of any task. The second task is v_5. From Fig. 3(a) we can see that it relies on the duplicated copy v_1^1 of v_1. Task v_1 has two children v_5 and v_6. From Fig. 3(a), task v_1^1 is completed at $t = 9$ and v_6 starts at $t = 7$, and the precedence constraint is violated if v_1 is deleted, so we do not delete v_1. The search continues. For task v_6, its relied duplication v_3 has two children, v_5 and v_6. Obviously, the precedence constraint between v_3 and v_6 is satisfied when v_3 is deleted. The finish time of v_3^1

on processor p_0 is 7, and the communication time between v_3 and v_5 is 2. If v_5 receives data from v_3^1, the data ready time of v_5 is $7 + 2 = 9$, which is equal to the start time of v_5 on p_2. Therefore, task v_3 can be deleted. The schedule result is shown as Fig. 3(b).

Comparing the two schedules, the makespan is the same, but their energy consumption is different. In Fig. 3(a), the total busy time of four processors is 27 ms, and the total idle time is 5 ms. In Fig. 3(b), the total busy time and idle time are 24 ms and 5 ms, respectively. According to Table 2, the power consumption of busy period and idle period are 925 mW and 260 mW. Therefore, the energy consumption of the two schedules are 26.275 mJ and 23.5 mJ. The EAMD algorithm reduces 10.56 % of the energy consumption of HLD in this example.

5 Experimental results and analysis

In this section, we apply the EAMD algorithm proposed in this paper to two classic duplication-based algorithms HLD and HCPFD. Both HLD and HCPFD algorithms are well known duplication-based algorithms with good performance in terms of makespan, but they are energy "unconscious". In this paper, we combine the EAMD algorithm with HLD and HCPFD, and label them as HLD+EAMD and HCPFD+EAMD for short. This comparison between HLD and HLD+EAMD, HCPFD and HCPFD+EAMD clearly demonstrates the energy saving capability of our proposed algorithm. The two compared algorithms, i.e., HLD and HCPFD, are briefly described as follows.

- The HLD algorithm is proposed by S. Bansal et al. [19], which is to improve the performance of the list-based heuristics with addition of limited duplication. The algorithm schedules the tasks strictly in the order of their global priority, and restricts duplications to the most crucial immediate parents so as to avoid redundant replications.
- The HCPFD algorithm [22] introduces a simple list-scheduling mechanism instead of the classical prioritization phase and a low complexity duplication-based mechanism for the machine assignment phase. This algorithm assigns higher priority values to the tasks on the critical path. In the machine assignment phase, it considers duplicating not only the first critical parent but also the second one. This mechanism improves the performance in terms of makespan.

The performance of the EAMD algorithm is evaluated with two kinds of applications, i.e., randomly generated applications and real-world applications. The two real-world parallel applications used for our experiments are the Gaussian elimination algorithm [37, 38] and a molecular dynamic code algorithm [39]. Typically, the makespan is usually adopted as the main performance criterion. Whereas, our algorithm is proposed on the basis of duplication-based algorithms and does not change the original makespan, so we do not compare their makespan in our experiments. The main performance metric chosen for comparison is energy consumption. Since the energy consumption of applications vary with the numbers of tasks inside a large variation range, it is necessary to normalize the energy consumption. Here we define a parameter energy-ratio *ER* as the energy metric:

$$ER = \frac{E}{E_{\mathrm{HCPFD}}}, \tag{8}$$

where E is the energy consumption of a compared algorithm, and E_{HCPFD} is the energy consumption of algorithm HCPFD.

5.1 Randomly generated applications

5.1.1 Application graphs random generation

The randomly generated graphs are commonly used to compare the performance of scheduling algorithms, and the generating method is described in [18, 20, 23, 24, 30]. Three fundamental characteristics in this paper are considered:

- DAG size n: The number of tasks in a DAG.
- Communication to computation cost ratio CCR: The average communication cost divided by the average computation cost of an application DAG.
- Parallelism factor λ: The number of levels in a DAG is generated randomly using a uniform distribution with a mean value of $\frac{\sqrt{n}}{\lambda}$ and rounded up to the nearest integer. The width of a DAG is generated randomly using a uniform distribution with a mean value of $\lambda\sqrt{n}$ and rounded up to the nearest integer. A small λ leads to a DAG with low parallelism degree.

In our experiments, graphs are generated based on the parameters introduced above. The number of nodes in a DAG ranges from 20 to 640. To generate a DAG with a given size, the number of levels is determined by the parallelism factor λ (0.2, 0.5, 1.0, 2.0, 5.0) firstly, and then the number of tasks on each level is determined. Edges are only generated between the nodes in adjacent levels, obeying a 0-1 distribution. To obtain the desired CCR for a graph, computation costs are taken randomly from a uniform distribution. The communication costs are also randomly selected from a uniform distribution, whose mean depends on the product of CCR (0.1, 0.5, 1.0, 2.0, 5.0) and the average computation cost. 500 random graphs are generated for each set of above parameters in order to avoid scattering effects. The experimental results are the average of the data obtained for these graphs.

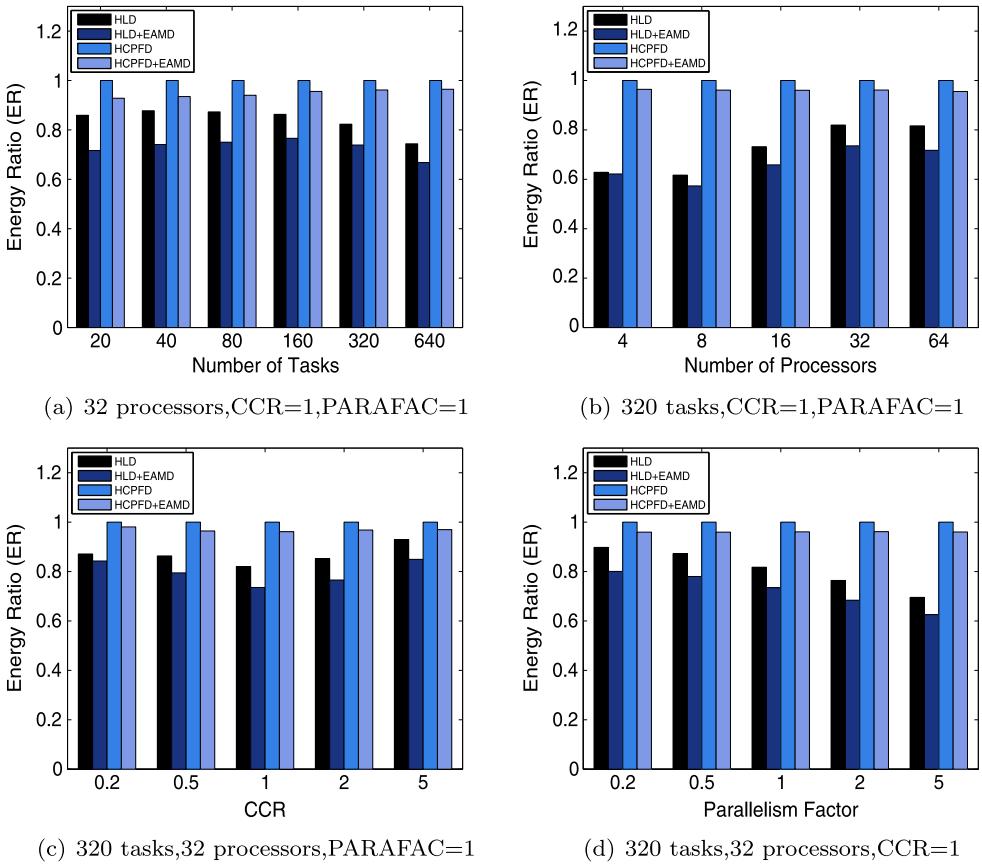


Fig. 4 Average energy saving of random DAGs

(a) 32 processors,CCR=1,PARAFAC=1

(b) 320 tasks,CCR=1,PARAFAC=1

(c) 320 tasks,32 processors,PARAFAC=1

(d) 320 tasks,32 processors,CCR=1

5.1.2 Random applications performance analysis

The energy consumption of algorithms is compared with respect to various graph characteristics. The overall experimental results are presented in Fig. 4 and Table 3. Figure 4 gives an intuitive presentation that both HLD and HCPFD obtain obvious improvement on energy consumption when combining with EAMD. Table 3 clearly shows the extent of improvement of our algorithm on HLD and HCPFD algorithms in terms of energy efficiency.

The first set of experiments compare the energy consumption of the algorithms with respect to various graph sizes (see Fig. 4(a)). The performance on energy saving of the two improved algorithms, HLD+EAMD and HCPFD+EAMD, outperforms HLD and HCPFD. According to Table 3, the average energy consumption of HLD+EAMD is less than HLD algorithm by 15.59 %, 14.04 %, 11.23 %, 10.24 %, and 10.20 %, for number of tasks of 20, 40, 80, 160, and 320, respectively. For HCPFD+EAMD, the corresponding average energy savings are 6.49 %, 5.98 %, 4.43 %, 3.84 %, and 3.56 %. Overall, EAMD performs better on HLD than on HCPFD. The energy saving decreases with the increasing number of tasks. The reason is as follows. When the number of tasks is small, the load on each processor is light and there are enough idle time to duplicate tasks. So more duplicated tasks can be deleted using the EAMD algorithm, hence leading to more energy saving.

The second set of experiments compare the energy consumption of the algorithms with respect to different numbers of processors. The average energy savings of HLD+EAMD and HCPFD+EAMD compared to HLD and HCPFD are (9.55 %, 3.57 %), (9.78 %, 3.91 %), (10.03 %, 3.96 %), (10.28 %, 3.88 %), and (12.15 %, 4.47 %), for number of processors of 4, 8, 16, 32, and 64, respectively. Therefore, our algorithm outperforms both HLD and HCPFD algorithms and along with the increasing number of processors, the performance gets better. The reason is the same as that of the first set of experiments.

In the third set of experiments, when combined with the EAMD algorithm, the HLD and HCPFD algorithms consume much less energy than the original algorithms for all tested CCR values. Table 3 shows that the peak performance is achieved at CCR = 1, i.e., 10.28 % for HLD and 3.88 % for HCPFD. The possible reasons resulting in this phenomenon are explained as follows. When the CCR value is less than 1, the applications generated in our experiments are compute-intensive ones. Duplication is unnecessary if duplicating a parent spends much more time than communication. Therefore, the latent redundancy becomes less, which leads to lower improvement of EAMD on both HLD and

Table 3 Energy conservation with respect to various characteristics

Number of tasks	20	40	80	160	320
HLD+EAMD vs. HLD	15.59 %	14.04 %	11.23 %	10.24 %	10.20 %
HCPFD+EAMD vs. HCPFD	6.49 %	5.98 %	4.43 %	3.84 %	3.56 %
Number of processors	4	8	16	32	64
HLD+EAMD vs. HLD	9.55 %	9.78 %	10.03 %	10.28 %	12.15 %
HCPFD+EAMD vs. HCPFD	3.57 %	3.91 %	3.96 %	3.88 %	4.47 %
CCR	0.2	0.5	1	2	5
HLD+EAMD vs. HLD	3.27 %	7.92 %	10.28 %	10.15 %	8.62 %
HCPFD+EAMD vs. HCPFD	1.94 %	3.58 %	3.88 %	3.18 %	3.04 %
PARAFAC	0.2	0.5	1	2	5
HLD+EAMD vs. HLD	10.85 %	10.61 %	10.15 %	10.47 %	10.07 %
HCPFD+EAMD vs. HCPFD	4.04 %	4.05 %	3.92 %	3.88 %	3.98 %

Table 4 Energy conservation of 320 tasks and 8 processors with respect to parallelism degree

PARAFAC	0.2	0.5	1	2	5
HLD+EAMD vs. HLD	7.88 %	9.00 %	7.02 %	2.89 %	0.82 %
HCPFD+EAMD vs. HCPFD	2.98 %	3.66 %	3.94 %	4.16 %	4.30 %

HCPFD. When the CCR value is greater than 1, the applications are prone to be communicate-intensive. When determining whether the original copy can be deleted, the greater communication cost leads to greater probability that it cannot be deleted. Therefore, the energy saving gets smaller. In summary, only when the communication and computation costs are equivalent, the best performance is achieved.

The last set of experiments are with respect to the graph structure. From Table 3 we notice that the parallelism degree has little impact on energy saving. After analyzing, we find that the setting of parameters is improper, which shields the varying tendency of energy saving. Because 32 processors are enough to execute 320 tasks in parallel no matter how large the parallelism degree is. In order to make the varying tendency of performance with the increasing parallelism degree more apparent, we set 320 tasks and 8 processors and do another group of experiments. The results are shown in Table 4. When λ is equal to 0.2, the generated graphs have greater depths with low degrees of parallelism, and it is shown that the energy consumption of HLD+EAMD is less than HLD by 7.88 %, and 2.98 % for HCPFD+EAMD compared with HCPFD. With the increasing of the parallelism factor λ, the performance of EAMD degrades on HLD but upgrades on HCPFD algorithm. That is because the priority queuing of HLD is based on upward rank which is breadth-first while that of HCPFD is based on the critical path which is depth-first. Therefore, the varying tendency of performance is different for two algorithms.

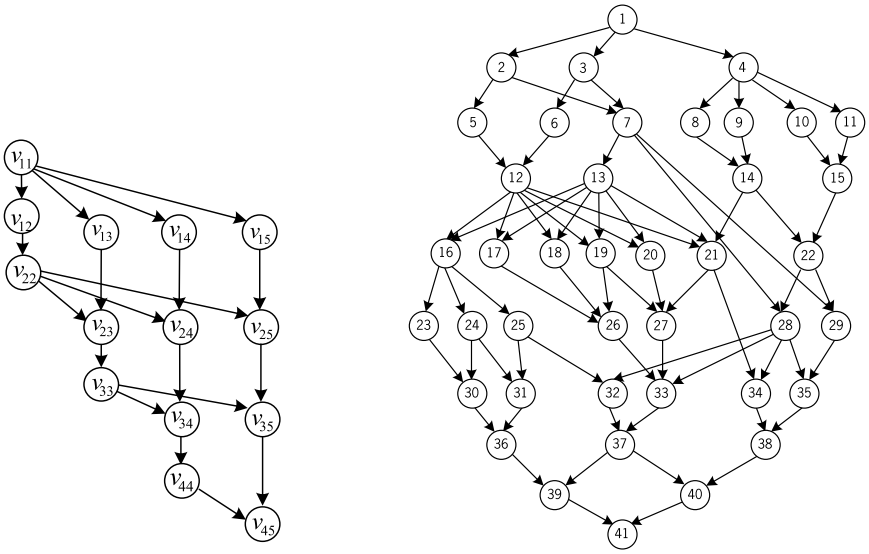
5.2 Application graphs of real-world problems

In addition to randomly generated task graphs, we also consider two application graphs of real-world problems, i.e., the Gaussian elimination algorithm and a molecular dynamic code algorithm. Because the number of tasks and graph structure are fixed, we only consider the number of processors and CCR as the varying parameters.

5.2.1 Gaussian elimination

Gaussian elimination is used to determine the solution of linear equations [40]. In this section, we consider the schedule of Gaussian elimination solving a 5×5 matrix. The DAG is shown in Fig. 5(a).

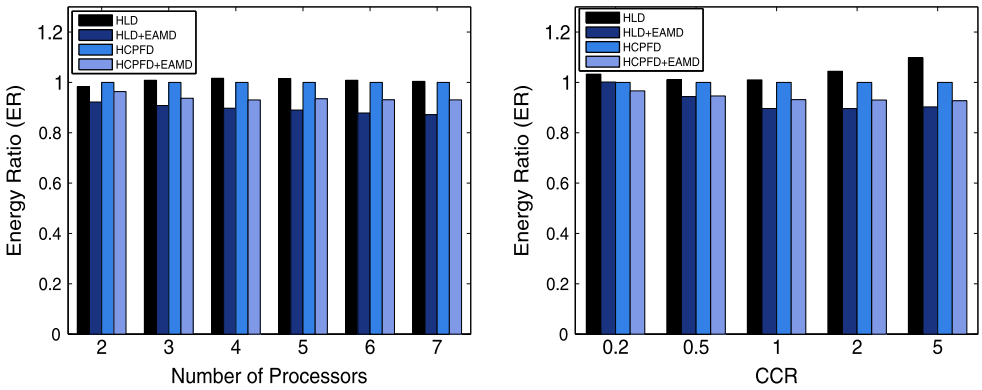
For the experiments of Gaussian elimination, the same CCR values (0.2, 0.5, 1, 2, 5) are used. The number of processors in our experiments varies from 2 to 7. Figure 6 shows the comparison results. When combined with EAMD, the HLD and HCPFD algorithms consume less energy. With the increasing number of processors, EAMD can obtain greater percentage of energy saving compared with HLD and HCPFD. For example, when the tasks are scheduled on two processors, the energy saving of EAMD compared with HLD is 6.20 %, and up to 13.17 % when the number of processors is 7. Along with the increasing of CCR values, EAMD can reduce more energy consumption compared with the HLD algorithm, which is up to 17.79 % at CCR = 5.



(a) Gaussian elimination for a matrix of size 5

(b) A molecular dynamics code

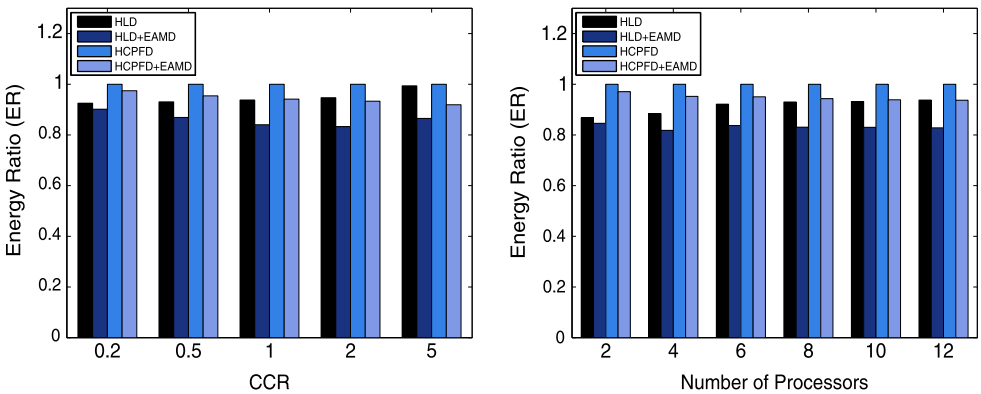
Fig. 5 Directed acyclic graphs for two real-world applications



(a) CCR=1

(b) 4 processors

Fig. 6 Average energy saving for Gaussian elimination



(a) 8 processors

(b) CCR=1

Fig. 7 Average energy saving for molecular dynamic code

5.2.2 Molecular Dynamic Code

Figure 5(b) gives the task graph of a molecular dynamic code introduced in [39]. Since the number of tasks is fixed in the graph and the structure is known, only CCR and number of processors are considered. The number of processors in our experiments varies from 2 to 12 in step of 2, and the same CCR values (0.2, 0.5, 1, 2, 5) are used. Figure 7 shows

Table 5 A global comparison of energy consumption for Gaussian elimination

Number of processors	HLD+EAMD vs. HLD			HCPFD+EAMD vs. HCPFD		
	Better	Equal	Worse	Better	Equal	Worse
2	71.0 %	29.0 %	0 %	45.5 %	54.6 %	0 %
3	83.2 %	16.8 %	0 %	58.5 %	41.2 %	0 %
4	88.4 %	11.6 %	0 %	63.8 %	36.2 %	0 %
5	88.7 %	11.3 %	0 %	60.8 %	39.2 %	0 %
6	89.0 %	11.0 %	0 %	62.4 %	37.6 %	0 %
7	91.0 %	9.0 %	0 %	62.6 %	37.8 %	0 %

Table 6 A global comparison of energy consumption for molecular dynamic code

CCR	HLD+EAMD vs. HLD			HCPFD+EAMD vs. HCPFD		
	Better	Equal	Worse	Better	Equal	Worse
2	80.4 %	19.6 %	0 %	78.4 %	21.6 %	0 %
4	99.8 %	0.2 %	0 %	93.6 %	6.4 %	0 %
6	99.8 %	0.2 %	0 %	94.8 %	5.2 %	0 %
8	100.0 %	0.0 %	0 %	97.0 %	3.0 %	0 %
10	100.0 %	0.0 %	0 %	97.0 %	3.0 %	0 %
12	100.0 %	0.0 %	0 %	97.6 %	2.4 %	0 %

the experimental results. Figure 7(a) is with respect to five different CCR values when the number of processors is set as 8. From the figure we can see that EAMD algorithm always outperforms HLD and HCPFD. According to the experimental results, HLD+EAMD consumes 8 % less energy than HLD on average, and HCPFD+EAMD reduces 5 % energy consumption on average compared with HCPFD. Figure 7(b) presents the experimental results with respect to six different numbers of processors when CCR is fixed to 1. When the number of processors is 12, the energy savings of EAMD compared with HLD and HCPFD are 11.67 % and 6.30 %, respectively.

In Tables 5 and 6, we present the probabilities among 500 times that the EAMD algorithm performs better than, worse than, or equal to the original algorithms on energy consumption for various numbers of processors and CCR values. As shown in the two tables, the EAMD algorithm can consume less energy than HLD and HCPFD with great probability, and EAMD performs better on the molecular dynamic code than on Gaussian elimination, because the former has more tasks than the latter. When scheduling a DAG with more number of tasks, the probability that duplicated copies can be deleted is more than scheduling one with less number of tasks. In Table 5 we can also find that EAMD outperforms the other two algorithms with increasing probability among 500 times with the increasing number of processors. That is because there is more chance for both HLD and HCPFD to duplicate tasks and EAMD can delete those copies with larger probability.

6 Conclusions

In this paper, we propose a new scheduling algorithm called EAMD scheduling algorithm. The aim of the algorithm is to reduce the energy consumption of duplicated-based algorithms, which is caused by redundant mapping of some tasks. Two kinds of graphs are adopted in our experiments to evaluate the performance of the proposed algorithm. Through the experimental results, EAMD can reduce energy consumption by up to 15.59 % compared with HLD and HCPFD algorithms. Similarly, EAMD can be combined with all other duplication-based algorithms, which can obtain good performance on energy saving as well.

The amount of energy saving is affected by many factors, such as the number of processors, CCR, and parallelism degree. For fixed number of tasks, the amount of energy saving increases with the increasing number of processors due to the increasing idle time and duplications. When the average communication cost is almost equal to the average computation cost, it also means when CCR is around 1, the duplicated-based algorithms can map tasks in the most processors, which leads to the greatest amount of energy saving compared to other CCR values. For the parallelism of DAGs, the percentage of energy saving decreases with the increase of width. Overall, our algorithm EAMD can achieve good performance compared to the duplication-based algorithms. Future work can involve combining EAMD with the DVFS technique to reduce more energy consumption.

Acknowledgements Thanks are due to three anonymous reviewers for their comments and suggestions on improving the manuscript. This

research was partially funded by the Key Program of National Natural Science Foundation of China (Grant No. 61133005), and the National Natural Science Foundation of China (Grant Nos. 90715029, 61070057, 61370095, 61202109), the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (Grant No. 708066), the Ph.D. Programs Foundation of Ministry of Education of China (20100161110019) and Project supported by the National Science Foundation for Distinguished Young Scholars of Hunan (12JJ1011).

References

1. Freund, R.F., Siegel, H.J.: Heterogeneous processing. Computer **26**(6), 13–17 (1993)
2. Maheswaran, M., Braun, T.D., Siegel, H.J.: Heterogeneous distributed computing. In: Encyclopedia of Electrical and Electronics Engineering, vol. 8, pp. 679–690. Wiley, New York (1999)
3. Zhang, Y., Hu, X., Chen, D.Z.: Task scheduling and voltage selection for energy minimization. In: Proceedings of 39th Design Automation Conference, pp. 183–188 (2002)
4. Zhu, D., Melhem, R., Childers, B.R.: Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. IEEE Trans. Parallel Distrib. Syst. **14**(7), 686–700 (2003)
5. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory Comput. Syst. **43**, 67–80 (2008)
6. Bunde, D.P.: Power-aware scheduling for makespan and flow. J. Sched. **12**(5), 489–500 (2009)
7. Baskiyar, S., Abdel-Kader, R.: Energy aware dag scheduling on heterogeneous systems. Clust. Comput. **13**, 373–383 (2010)
8. Lee, Y.C., Zomaya, A.Y.: Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst. **22**(8), 1374–1381 (2011)
9. Diaz, C.O., Guzek, M., Pecero, J.E., Danoy, G., Bouvry, P., Khan, S.U.: Energy-aware fast scheduling heuristics in heterogeneous computing systems. In: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 478–484. IEEE Press, New York (2011)
10. Lee, Y.C., Zomaya, A.Y.: Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst. **22**, 1374–1381 (2011)
11. Burd, T.D., Brodersen, R.W.: Design issues for dynamic voltage scaling. In: Proceedings of the International Symposium on Low Power Electronics and Design, 2000. ISLPED'00, pp. 9–14. IEEE Press, New York (2000)
12. de Langen, P., Juurlink, B.: Leakage-aware multiprocessor scheduling. J. Signal Process. Syst. **57**(1), 73–88 (2009)
13. Kwok, Y.-K., Ahmad, I.: Benchmarking the task graph scheduling algorithms. In: Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998, Parallel Processing Symposium 1998 (IPPS/SPDP 1998), Mar–Apr 1998, pp. 531–537 (1998)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1990)
15. Ullman, J.D.: Np-complete scheduling problems. J. Comput. Syst. Sci. **10**, 384–393 (1975)
16. Radulescu, A., van Gemund, A.J.C.: Fast and effective task scheduling in heterogeneous systems. In: Proceedings of 9th Heterogeneous Computing Workshop (HCW 2000), pp. 229–238 (2000)
17. Lotfifar, F., Shahhoseini, H.S.: A low-complexity task scheduling algorithm for heterogeneous computing systems. In: Third Asia International Conference on Modelling Simulation (AMS'09), May 2009, pp. 596–601 (2009)
18. Daoud, M.I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **68**(4), 399–409 (2008)
19. Bansal, S., Kumar, P., Singh, K.: Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. J. Parallel Distrib. Comput. **65**(4), 479–491 (2005)
20. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
21. Ranaweera, S., Agrawal, D.P.: A scalable task duplication based scheduling algorithm for heterogeneous systems. In: Proceedings of 2000 International Conference on Parallel Processing, pp. 383–390 (2000)
22. Hagras, T., Brevecek, J.J.: A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. Parallel Comput. **31**(7), 653–670 (2005)
23. Lai, K.-C., Yang, C.-T.: A dominant predecessor duplication scheduling algorithm for heterogeneous systems. J. Supercomput. **44**, 126–145 (2008)
24. Bansal, S., Kumar, P., Singh, K.: An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. **14**(6), 533–544 (2003)
25. Luo, P., Lü, K., Shi, Z.: A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput. **67**(6), 695–714 (2007)
26. Kwok, Y.-K., Ahmad, I.: Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Trans. Parallel Distrib. Syst. **7**(5), 506–521 (1996)
27. Boeres, C., Filho, J.V., Rebello, V.E.F.: A cluster-based strategy for scheduling task on heterogeneous processors. In: 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2004), Oct. 2004, pp. 214–221 (2004)
28. Liou, J.C., Palis, M.A.: An efficient task clustering heuristic for scheduling dags on multiprocessors. In: Proceedings of Parallel and Distributed Processing Symposium (1996)
29. Fu, F., Bai, Y., Hu, X., Wang, J., Yu, M., Zhan, J.: An objective-flexible clustering algorithm for task mapping and scheduling on cluster-based noc. In: 2010 10th Russian–Chinese Symposium on Laser Physics and Laser Technologies (RCSLPLT) and 2010 Academic Symposium on Optoelectronics Technology (ASOT), 28 July–1 August 2010, pp. 369–373 (2010)
30. Tang, X., Li, K., Liao, G., Li, R.: List scheduling with duplication for heterogeneous computing systems. J. Parallel Distrib. Comput. **70**(4), 323–329 (2010)
31. Transmeta's design guides and datasheets
32. http://en.wikipedia.org/wiki/speedstep
33. Mobile AMD-k6 processor power supply design. Application note
34. Khokhar, A.A., Prasanna, V.K., Shaaban, M.E., Wang, C.-L.: Heterogeneous computing: challenges and opportunities. Computer **26**(6), 18–27 (1993)
35. Menasce, D.A., Saha, D., Porto, S.C.D., Almeida, V.A.F., Tripathi, S.K.: Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. J. Parallel Distrib. Comput. **28**(1), 1–18 (1995)
36. Intel: Pxa270 processor, electrical, mechanical, and thermal specification (2004)
37. Wu, M.-Y., Gajski, D.D.: Hypertool: a programming aid for message-passing systems. IEEE Trans. Parallel Distrib. Syst. **1**(3), 330–343 (1990)
38. Cosnard, M., Marrakchi, M., Robert, Y., Trystram, D.: Parallel Gaussian elimination on an mimd computer. Parallel Comput. **6**(3), 275–296 (1988)

39. Kim, S.J., Browne, J.C.: A general approach to mapping of parallel computation upon multiprocessor architectures. In: Proceedings of the International Conference on Parallel Processing, pp. 1–8 (1988)
40. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT, Cambridge (2001)

Jing Mei Jing Mei is currently working towards the Ph.D. degree at Hunan University of China. Her research interests include modeling and scheduling for embedded systems, distributed computing systems, and parallel algorithms.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003, and the B.S. degree in mathematics from Central South University, China, in 2000. He has been a visiting scholar at University of Illinois at Champaign and Urbana from 2004 to 2005. He is now a professor of Computer science and Technology at Hunan University and the deputy director of National Supercomputing Center in Changsha. He is a senior member of CCF and has published more than 80 peer reviewed papers. His major research contains parallel computing, Grid and Cloud computing, and DNA computer.

Keqin Li is a SUNY Distinguished Professor of computer science in the State University of New York at New Paltz. He is also an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He has contributed extensively to processor allocation and resource management; design and analysis of sequential/parallel, deterministic/probabilistic, and approximation algorithms; parallel and distributed computing systems performance analysis, prediction, and evaluation; job scheduling, task dispatching, and load balancing in heterogeneous distributed systems; dynamic tree embedding and randomized load distribution in static networks; parallel computing using optical interconnections; dynamic location management in wireless communication networks; routing and wavelength assignment in optical networks; energy-efficient computing and communication. His current research interests include lifetime maximization in sensor networks, file sharing in peer-to peer systems, power management and performance optimization, and cloud computing. Dr. Li has published over 240 journal articles, book chapters, and research papers in refereed international conference proceedings. He is currently on the editorial board of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, Journal of Parallel and Distributed Computing, International Journal of Parallel, Emergent and Distributed Systems, International Journal of High Performance Computing and Networking, and Optimization Letters.