Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Energy-aware scheduling with reconstruction and frequency equalization on heterogeneous systems*

Yong-xing LIU[†1], Ken-li LI[†‡1], Zhuo TANG[1], Ke-qin LI[1,2]

([1]*College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China*)

([2]*Department of Computer Science, State University of New York, New Paltz, New York 12561, USA*)

[†]E-mail: yongxing510@126.com; lkl@hnu.edu.cn

Received Nov. 24, 2014; Revision accepted Apr. 30, 2015; Crosschecked June 5, 2015

Abstract: With the increasing energy consumption of computing systems and the growing advocacy for green computing, energy efficiency has become one of the critical challenges in high-performance heterogeneous computing systems. Energy consumption can be reduced by not only hardware design but also software design. In this paper, we propose an energy-aware scheduling algorithm with equalized frequency, called EASEF, for parallel applications on heterogeneous computing systems. The EASEF approach aims to minimize the finish time and overall energy consumption. First, EASEF extracts the set of paths from an application. Then, it reconstructs the application based on the extracted set of paths to achieve a reasonable schedule. Finally, it adopts a progressive way to equalize the frequency of tasks to reduce the total energy consumption of systems. Randomly generated applications and two real-world applications are examined in our experiments. Experimental results show that the EASEF algorithm outperforms two existing algorithms in terms of makespan and energy consumption.

Key words: Directed acyclic graph, Dynamic voltage scaling, Energy aware, Heterogeneous systems, Task scheduling

doi:10.1631/FITEE.1400399 **Document code:** A **CLC number:** TP314

1 Introduction

In the past decade, with the rapid increase of the high-performance requirements of applications and the rapid development of low-cost computers, heterogeneous computing systems have been increasingly employed to solve complex problems. A suite of distributed computing machines with varied computational capabilities, which are interconnected by high speed links, can be defined as a heterogeneous computing (HC) system (Freund and Siegel, 1993). To satisfy the high-performance requirements of application executions, the main design goal of the processor is to improve the frequency of the processor. However, with the lasting increase of the frequency, energy consumption has been growing exponentially in computers and computing centers, embedded systems, portable devices, etc. (Brown, 2008). The increased energy consumption causes severe economic, ecological, and technical problems, such as huge expense of power supply, excessive carbon dioxide emissions, and massive heat dissipation. So, it is significant to study the strategy of reducing energy consumption in an HC system.

There are two approaches for reducing energy consumption in HC systems. The first approach is hardware design, such as low-power processor architecture and low-power memory hierarchy. The second approach is energy-aware software design. For instance, we can reduce energy dissipation using an

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61133005, 61432005, 61370095, 61472124, and 61402400)

ORCID: Yong-xing LIU, http://orcid.org/0000-0001-8935-9543

energy-aware scheduling algorithm. In particular, the dynamic voltage scaling (DVS) (Mehta and Amrutur, 2012; Mittal, 2014) and dynamic power management (DPM) (Benini *et al.*, 2000; Amador *et al.*, 2012) techniques are employed as the key tool to reduce energy consumption in a scheduling algorithm. For a given HC system, in general we cannot improve the architecture of a processor; energy consumption can be reduced only through software design. It is well known that finding an optimal schedule is an NP hard problem in most cases (Ullman, 1975; Li, 2012). Therefore, one of the challenges in heterogeneous computing is to develop scheduling algorithms that can optimally assign the tasks of an application to processors.

Many task scheduling algorithms have been proposed to minimize the execution time (Kwok and Ahmad, 1999; Topcuoglu *et al.*, 2002; Bajaj and Agrawal, 2004; Hagras and Janeček, 2005; Bozdag *et al.*, 2009; Khan, 2012) of an application running on an HC system. The performance of these algorithms is evaluated based on one criterion in general, i.e., schedule length (or 'makespan'). However, scheduling is inherently a multi-objective problem, especially in the HC scenario, since it usually implies several conflicting objectives in the optimization process. For example, reducing power consumption may lead to slower execution of applications. To reduce energy consumption of computing systems and cater to the trend of green computing, many efforts have been devoted to energy-aware scheduling design.

In this study, we address the problem of scheduling a directed acyclic graph (DAG) in an HC system with a bi-objective of minimizing finish time and energy consumption. In the first phase, we determine the set of paths from a DAG to prepare for the next phase. Then, we reconstruct a DAG based on the determined paths to obtain a reasonable schedule during post-processing. In the last phase, we consider equalizing the frequency of tasks in an application and scaling the frequency of a CPU dynamically, in order to achieve the goal of reducing the overall system energy consumption.

2 Related work

Task scheduling algorithms based on DAG can typically be classified into several sub-categories, such as list-based scheduling algorithms, cluster-based heuristics algorithms, and duplication-based algorithms.

Heterogeneous earliest finish time (HEFT) (Topcuoglu *et al.*, 2002) is the most well-known list-based scheduling algorithm for heterogeneous systems. Several other classical examples of list scheduling algorithms are the critical path on a processor (CPOP) (Topcuoglu *et al.*, 2002), dynamic critical path (DCP) (Kwok and Ahmad, 1996), heterogeneous critical parents with a fast duplicator (HCPFD) (Hagras and Janeček, 2005), etc. Furthermore, Khan (2012) proposed a novel approach called constrained earliest finish time (CEFT), which outperforms HEFT, dynamic level scheduling (DLS) (Sih and Lee, 1993), and levelized min-time (LMT) (Iverson *et al.*, 1995) in a diverse collection of task graphs.

The difference between a list-based algorithm and a cluster-based algorithm is that the former orders all tasks according to calculated priorities prior to assignment, while the latter generates subsets of tasks first and then orders each subset individually. In a cluster algorithm, dominant sequence clustering (DSC) (Yang and Gerasoulis, 1994) is the most well-known approach, but DSC does not support heterogeneous systems. Some other examples in this category include the clustering heuristic scheduling algorithm (CHSA) (Ilyas and Khan, 2001), greedy task clustering and scheduling (GTCS) (Piyatamrong *et al.*, 2000), clustering for heterogeneous processors (CHP) (Boeres and Rebello, 2004), and the objective-flexible clustering algorithm (OFCA) (Fu *et al.*, 2010).

The difference between a duplication-based algorithm and the first two types of algorithms is whether some tasks need to be duplicated in the process of scheduling. Usually, duplication-based algorithms are able to obtain a better performance in terms of makespan compared with list- and cluster-based algorithms. Some classical examples in this category include selective duplication (Bansal *et al.*, 2003), HCPFD (Hagras and Janeček, 2005), heterogeneous limited duplication (HLD) (Bansal *et al.*, 2005), and heterogeneous earliest finish with duplication (HEFD) (Tang *et al.*, 2010). As mentioned before, almost all these algorithms do not consider the energy consumption of systems when scheduling tasks. At the same time, energy dissipation of systems has shown an explosive growth trend in the past

decades.

The explosive energy consumption has led to greater advocacy for green computing. Many efforts have been devoted to energy-aware scheduling to reduce energy consumption in systems. An on-line DVS algorithm called OLDVS can achieve significant energy savings for some applications (Lee and Shin, 2004). An algorithm named energy-aware scheduling by minimizing duplication (EAMD) achieves good energy saving by deleting redundant task copies in the schedules generated by duplication-based algorithms (Mei and Li, 2012). For a set of real-time tasks with precedence constraints executed on a distributed system, a simple static power management scheme (S-SPM) and a dynamic power management (DPM) approach are presented (Mishra *et al.*, 2003). Some other DVS-based examples include energy-conscious scheduling (ECS) (Lee and Zomaya, 2011), energy-efficient scheduling (EES) (Huang *et al.*, 2012), and adaptive energy-efficient scheduling (AEES) (Zhu *et al.*, 2012). Many studies mentioned above have proved that DVS is a very promising technique with its demonstrated capability for energy savings. For this reason, in this study we adopt the DVS technique to reduce energy consumption.

3 Models

In this section, a computing system model and an application model are described in detail.

3.1 Computing system model

We study the task scheduling problem for applications on a set P of m heterogeneous processors that are fully interconnected. The computing system model is denoted by $P=\{p_i|0\leq i\leq m-1\}$, where each processor is DVS enabled and it can operate at different clock frequency levels. Power consumption at each performance state (P-state) (Terzopoulos and Karatza, 2013) for all processors is depicted in Table 1.

The capacity of a processor depends on several factors: processor architecture, task processing requirement, and degree of match between the task and the processor. A processor which is best at performing one task may be bad at performing another task. When a task arrives at a processor, if the processor is idle, it will execute the task at once; otherwise, the task has to wait until the processor is available. A task cannot be suspended when it is running on a processor; that is, the running task is locked against preemption in the system.

Table 1 Performance and power consumption

Frequency	Power (W)		
(GHz)	AMD Opteron	Intel PentiumM	VIA C7-M
2.6	95	–	–
2.4	90	–	–
2.2	76	–	–
2.0	65	–	20
1.8	55	–	18
1.6	–	25	15
1.4	–	17	13
1.2	–	13	–
1.0	32	10	10
0.8	–	8	7
0.6	–	6	6
0.4	–	–	5
Idle	15	5	0.1

3.2 Application model

An application, in general, can be represented by a DAG. A DAG with both node and edge weights is represented as $G=G(V,E,\Omega,\Psi)$, which consists of a set of nodes $V=\{v_i|0\leq i\leq n-1\}$ representing the tasks of the application, and a set of directed edges $E=\{e_{i,j}|0\leq i,j\leq n-1\}$ representing dependencies among tasks. $\Omega=\{\omega_{i,k}|0\leq i\leq n-1,0\leq k\leq m-1\}$ denotes the computation cost of task v_i on processor p_k, and $\Psi=\{\psi(e_{i,j})|0\leq i,j\leq n-1\}$ denotes the communication cost between tasks v_i and v_j. If edge $e_{i,j}$ exists, then v_i is called a parent of v_j and v_j is called a child of v_i. The immediate parent set of task v_i is denoted by pare(v_i) and the immediate child set of task v_i is denoted by child(v_i).

Fig. 1 gives a simple DAG which consists of 12 nodes. Table 2 represents the computation costs of tasks on different processors. 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 task v_{11} in Fig. 1. A DAG may have multiple entry tasks and multiple exit tasks. For a DAG with multiple entry/exit tasks, we can transform it by adding zero-cost pseudo entry/exit tasks with zero-cost edges to a single-entry single-exit DAG, which does not affect the schedule.

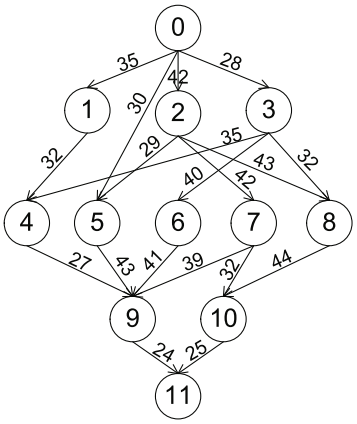


Fig. 1 A simple DAG representing an application graph with precedence constraints

Table 2 Computation costs of tasks in Fig. 1

Task	p_0	p_1	p_2	p_3	$\overline{\omega_i}$
0	29	50	23	29	32.75
1	27	48	41	27	35.75
2	30	43	34	30	34.25
3	18	39	31	18	26.50
4	24	36	28	24	28.00
5	23	47	34	23	31.75
6	21	38	35	21	28.75
7	33	47	44	33	39.25
8	27	54	39	27	36.75
9	25	44	28	25	30.50
10	28	32	30	28	29.50
11	19	46	40	19	31.00

3.3 Performance measurements

The schedule length is undoubtedly one of the most important criteria for performance measurement. So, makespan is adopted as the first criterion in this study. In the scheduling model, let $t_{\mathrm{s}}(v_i, p_k)$ and $t_{\mathrm{f}}(v_i, p_k)$ represent the start time and finish time of task v_i scheduled on processor p_k, respectively. Then, $t_{\mathrm{f}}(v_i, p_k) = t_{\mathrm{s}}(v_i, p_k) + w_{i,k}$. The makespan can be defined as

$$\text{makespan} = \max\{t_{\mathrm{f}}(v_i, p_k)\}. \tag{1}$$

Processor energy consumption is the second performance metric in the system. The total energy consumption of processor E_i includes mainly two parts: active energy consumption $E_{i,\mathrm{active}}$ and idle energy consumption $E_{i,\mathrm{idle}}$. So, the total energy consumption of a system can be calculated by

$$E = \sum_{i=0}^{m-1} (E_{i,\mathrm{active}} + E_{i,\mathrm{idle}}). \tag{2}$$

4 Proposed algorithm

This section presents the details of the EASEF algorithm. In EASEF, all tasks in a DAG are assigned scheduling priorities and the task with the highest priority is scheduled first. The EASEF scheduling process includes mainly three phases: path determining, application reconstructing, and frequency equalizing. To achieve a reasonable assignment, the set of paths in an application is first determined for the reconstruction process, which helps better distribute tasks. Then the EASEF algorithm takes a progressive way to equalize the frequency of two adjacent tasks. The detailed description of EASEF is presented in the following subsections.

4.1 Path determining phase

The main goal of this phase is to determine the set of paths from an application. In general, a critical path (CP) in a DAG is the longest path from the entry node to the exit node. In a heterogeneous computing system, the CP length in a DAG is the sum of the mean computation costs of tasks and the communication costs along the path. The mean computation cost of task v_i is calculated by

$$\overline{\omega}_i = \frac{1}{m} \sum_{k=0}^{m-1} \omega_{i,k}. \tag{3}$$

To determine the CP from a DAG, upward rank and downward rank values are needed in general. The upward rank of task v_i is recursively calculated by

$$\mathrm{rank_b}(v_i) = \begin{cases} \overline{\omega}_{\mathrm{exit}}, & v_i = v_{\mathrm{exit}}, \\ \overline{\omega}_i + \max\limits_{v_\sigma \in \mathrm{child}(v_i)} (\mathrm{rank_b}(v_\sigma) + \psi(e_{i,\sigma})), & \text{otherwise.} \end{cases} \tag{4}$$

Similarly, the downward rank of task v_i is recursively calculated by

$$\mathrm{rank_t}(v_i) = \begin{cases} 0, & v_i = v_{\mathrm{entry}}, \\ \max\limits_{v_j \in \mathrm{pare}(v_i)} (\mathrm{rank_t}(v_j) + \psi(e_{j,i}) + \overline{\omega}_j), & \text{otherwise.} \end{cases} \tag{5}$$

Then, the rank value of task v_i is calculated by

$$\mathrm{rank}(v_i) = \mathrm{rank_b}(v_i) + \mathrm{rank_t}(v_i). \tag{6}$$

The detailed process of determining the set of paths is described in Algorithm 1. First, we can calculate the mean computation cost of tasks by Eq. (3). Then, to guarantee that the calculations in Algorithm 1 are orderly, we need to ensure that the DAG is a single-entry single-exit DAG. Having calculated the upward and downward rank values of tasks, we can determine the rank values of tasks by Eq. (6). Then we can extract a critical path with a maximum rank value from the entry node to the exit node. To continue determining the rest of the paths in the DAG, Algorithm 1 will remove all tasks that belong to a determined critical path from the DAG. Then, repeat the above steps until all paths are determined by Algorithm 1.

Algorithm 1 Determining the set of paths

Require: G. // G: an application graph
Ensure: CP[ ]. // CP[ ]: the set of paths
1: **for** each task v_i in G **do**
2: Calculate $\overline{\omega}_i$ by Eq. (3);
3: **end for**
4: **if** G is not a single-start single-exit graph **then**
5: $G = \text{transform}(G)$; // transform G into a single-start single-exit graph
6: **end if**
7: Let count = 0;
8: **while** G is not null **do**
9: **for** each task v_i in G **do**
10: Calculate $\text{rank}_b(v_i)$ by Eq. (4);
11: Calculate $\text{rank}_t(v_i)$ by Eq. (5);
12: Calculate $\text{rank}(v_i)$ by Eq. (6);
13: **end for**
14: **for** each task v_i in G **do**
15: **if** v_i is a critical task **then**
16: CP[count].add(v_i); // save the critical task based on the rank value of the task
17: **end if**
18: **end for**
19: $G = G.\text{remove}\big(\text{CP[count]}\big)$; // remove the critical tasks from G
20: $G = \text{transform}(G)$;
21: count++;
22: **end while**

The whole processing procedure of this phase for Fig. 1 is represented by Fig. 2. In the first round, the critical path [0-2-8-10-11] with length 318.25 is found. Then these nodes are pruned from the DAG, and the pseudo nodes ($v_{\text{entry}}, v_{\text{exit}}$) with zero-cost edges are added to transform the DAG into a single-entry single-exit DAG. In the second round, the critical path [3-6-9] with length 166.75 is found. Next, the critical paths [1-4], [7], and [5] are determined, for values of 95.75, 39.25, and 31.75, respectively.

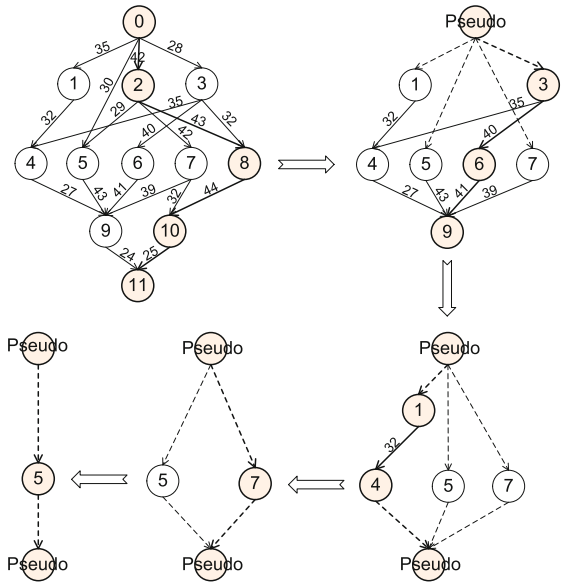


Fig. 2 An example of the processing procedure of Algorithm 1

4.2 Application reconstructing phase

In this phase, we consider the reconstruction of the DAG based on the CPs calculated using Algorithm 1. The main goal of this phase is to deal with the tasks that can be merged according to the CPs and the dependencies of tasks. If a task has n parents, then the indegree of the task is n. Similarly, the outdegree is n if a task has n children. If a task v_i just has one parent, then this task can be expressed by

$$\text{indegree}[v_i] = 1. \tag{7}$$

The CPOP (Topcuoglu *et al.*, 2002) algorithm maps the tasks of the longest path to the critical processor, but it does not consider the individual computation cost of tasks on different processors. The CEFT (Khan, 2012) algorithm assigns the constrained critical paths to a suite of appropriate processors, but it does not consider the relationship between the parent-node and child-node. Based on these findings, we temporarily merge some tasks into one task, so that the child nodes have higher priority than otherwise. Having scheduled a parent-node, the child with a higher priority will be scheduled immediately on the same processor with a large probability. Thus, the tasks on the longer path will be scheduled

in a more reasonable way. The detailed process is described in Algorithm 2.

In the process of reconstruction, the weight of the newly generated node is the sum of all the merged tasks. Let v_e and $[v_j, \cdots, v_k]$ represent the new node and merged tasks, respectively. Then, the weight of node v_e can be calculated by

$$\overline{\omega}_e = \sum_{i=j}^{k} \overline{\omega}_i. \tag{8}$$

To avoid breaking the tie of precedence constraints, Algorithm 3, which is called by Algorithm 2, is used to deal with the dependencies of edges. Typically, there are three types of edges that need to be processed. In the first case, the edges between two merged nodes of a path can be removed directly. In the second case, there is only one edge between the merged nodes of a path and task v_n. Let $[v_a, \cdots, v_i, \cdots, v_m]$ represent the merged nodes of a path in turn, and $e_{i,n}$ the only edge. Then the communication cost between the new node v_e generated by merging and v_n can be determined by

$$\psi(e_{e,n}) = \begin{cases} 0, & \psi(e_{i,n}) < \sum\limits_{j=i+1}^{m} \overline{\omega}_j, \\ \psi(e_{i,n}) - \sum\limits_{j=i+1}^{m} \overline{\omega}_j, & \text{otherwise}, \end{cases} \tag{9}$$

where $e_{i,n}$ will be removed after the calculation of Eq. (9). In the last case, there are several edges between the merged nodes of a path and task v_n. Let $[v_a, \cdots, v_i, \cdots, v_k, \cdots, v_m]$ represent the merged nodes of a path in turn, and $[v_i, \cdots, v_k]$ the nodes that need to send data to v_n. Then the communication cost between the new node v_t generated by merging and v_n can be determined by

$$\psi(e_{t,n}) = \begin{cases} 0, & \sum\limits_{r=i}^{k} \left(\psi(e_{r,n}) - \sum\limits_{j=r+1}^{m} \overline{\omega}_j \right) < 0, \\ \sum\limits_{r=i}^{k} \left(\psi(e_{r,n}) - \sum\limits_{j=r+1}^{m} \overline{\omega}_j \right), & \text{otherwise}. \end{cases} \tag{10}$$

From Algorithm 2, we know that the process has three major phases. First, some tasks are merged, as shown in lines 1–11. Then, three different dependencies of tasks are handled by Algorithm 3. Finally, the priority rule of tasks is mapped from the reconstructed graph to the original graph, as shown in lines 13–21. The result of the reconstruction phase for Fig. 1 is shown in Fig. 3. Having reconstructed the original DAG, we can find that the nodes v_0 and v_2, v_3 and v_6 are merged into the new nodes $v_{0'}$ and $v_{3'}$, respectively. The weight of node $v_{0'}$ in Fig. 3 is the sum of v_0 and v_2 in Fig. 1. Similarly, the weight

Algorithm 2 Reconstructing the DAG

Require: G, CP[ ]. // G: an application graph; // CP[ ]: the set of paths
Ensure: G', RANK. // G': the reconstructed graph; // RANK: the sequence of tasks
1: G'=G.clone();//duplicate the structure of the graph
2: **for** each path CP in CP[ ] **do**
3: **while** CP is not null **do**
4: Task v_i = CP.pop();
5: **if** task v_i satisfies Eq. (7) **then**
6: Merge task v_i with its parent;
7: Calculate the new weight $\overline{\omega}_e$ by Eq. (8);
8: Remove task v_i from G';
9: **end if**
10: **end while**
11: **end for**
12: **call** Algorithm 3 to handle the dependencies of tasks;
13: **for** each task v_i in G' **do**
14: Calculate $\text{rank}_\text{b}(v_i)$ by Eq. (4);
15: **if** v_i is not a new node **then**
16: Let the upward rank value of the same number task v_i of G equal $\text{rank}_\text{b}(v_i)$;
17: **else**
18: Let the upward rank values of all the merged and corresponding tasks $[v_i, \cdots, v_k]$ of G equal $\text{rank}_\text{b}(v_i)$;
19: **end if**
20: **end for**
21: Sort all the tasks in a sequence by non-decreasing order of the upward rank values, and let RANK represent the sequence;

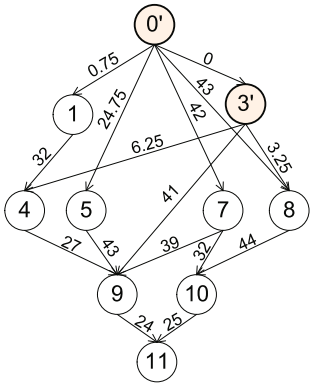


Fig. 3 The DAG reconstructed according to Algorithm 2

Algorithm 3 Handling the dependencies of tasks

Require: G, G'. // G: an application graph
Ensure: G'. // G': the reconstructed graph
1: **for** each task v_n in G **do**
2: **for** each task v_i in pare(v_n) **do**
3: **if** task v_i is not a merged node **then**
4: Do nothing;
5: **else if** v_n and v_i are contained in a node **then**
6: Remove edge $e_{i,n}$ from G';
7: **else if** only one parent of task v_n is contained in a node v_e **then**
8: Calculate $\psi(e_{e,n})$ by Eq. (9);
9: Remove edge $e_{i,n}$ from G';
10: **else**
11: Let $[v_a, \cdots, v_i, \cdots, v_k, \cdots, v_m]$ represent the merged nodes contained in v_t;
12: Let $[v_i, \cdots, v_k]$ represent the nodes that need to send data to v_n;
13: Calculate $\psi(e_{t,n})$ by Eq. (10);
14: Remove the edges $e_{i,n}, \cdots, e_{k,n}$ from G';
15: **end if**
16: **end for**
17: **end for**

of node $v_{3'}$ is the sum of v_3 and v_6 in Fig. 1. Further, we can find that the communication costs between a normal node and the merged nodes are updated by Algorithm 3. Fig. 3 gives the related communication costs of the reconstructed DAG. Having calculated the upward rank values of tasks in the reconstructed DAG, we can map the precedence relation of tasks from the reconstructed DAG to the original application. From Algorithm 2, we can obtain a precedence sequence 0-2-3-6-1-8-7-5-4-9-10-11 based on the example shown in Fig. 1.

4.3 Frequency equalizing phase

In this phase, the idle time slots among tasks on processors can be slacked and redistributed using the DVS technique without violating dependency constraints. In the process of scheduling, the earliest start time ($t_{\rm es}$) of task v_i on processor p_k is defined by

$$t_{\rm es}(v_i,p_k)=\begin{cases}0, & v_i=v_{\rm entry},\\ \max\Big(\min\big(\text{idle}^n_{k.\text{available}}.\text{start}\big), & \\ \quad \max\limits_{v_\varsigma\in\text{pare}(v_i)}(t_{\rm es}\ (v_\varsigma,p_l)+\omega_{i,k}+\psi(e_{\varsigma,i}))\Big), & \text{otherwise},\end{cases} \tag{11}$$

where $\text{idle}^n_{k.\text{available}}.\text{start}$ is the start time of the nth idle slack which satisfies $\text{idle}^n_{k.\text{available}}.\text{end} - \text{idle}^n_{k.\text{available}}.\text{start} \geq \omega_{i,k}$, and if $p_k = p_l$, $\psi(e_{\varsigma,i}) = 0$. The earliest finish time ($t_{\rm ef}$) of task v_i on processor p_k is defined by

$$t_{\rm ef}(v_i,p_k)=t_{\rm es}(v_i,p_k)+\omega_{i,k}. \tag{12}$$

The latest finish time ($t_{\rm lf}$) of task v_i on processor p_k is defined by

$$t_{\rm lf}(v_i,p_k)=\begin{cases}\text{makespan}, & v_i=v_{\rm exit},\\ \min(t_{\rm lf}(v_\tau,p_k)-\omega_{\tau,k}, & \\ \quad \min\limits_{v_\sigma\in\text{child}(v_i)}\big(t_{\rm lf}(v_\sigma,p_l)-\omega_{\sigma,p_l}-\psi(e_{i,\sigma})\big)), & \\ \quad \text{otherwise}, & \end{cases} \tag{13}$$

where v_τ is the task assigned next to v_i on the same processor p_k, v_σ is the task assigned on processor p_l, and if $p_k = p_l$, $\psi(e_{i,\sigma}) = 0$. Then the slack time of task v_i on processor p_k can be defined by

$$\text{slack}(v_i)=t_{\rm lf}(v_i,p_k)-t_{\rm es}(v_i,p_k)-\omega_{i,k}. \tag{14}$$

Usually there is a certain amount of overlap of the slack time between two adjacent tasks (v_i and v_j) scheduled on the same processor p_k. Without loss of generality, we assume $t_{\rm es}(v_i,p_k) < t_{\rm es}(v_j,p_k)$ and $t_{\rm lf}(v_i,p_k) > t_{\rm es}(v_j,p_k)$. Then, different slack assignment methods can lead to different energy consumptions. To avoid a task squeezing the slack time of other tasks and reduce the energy consumption of systems, the proposed approach takes an equalized way to optimize the frequency of tasks progressively. Equalizing the frequency of two adjacent tasks is the main objective in this phase. The detailed process is presented in Algorithm 4.

Let v_i and v_j represent the two adjacent tasks scheduled on processor p_k, and assume that v_j is the task assigned next to v_i. We take the two tasks as a whole to balance the overlap of the slack time between v_i and v_j. Then, the common ideal frequency of the two tasks can be calculated by

$$f_{\rm id}(v_i,v_j)=\frac{\omega_{i,k}+\omega_{j,k}}{t_{\rm lf}(v_j,p_k)-t_{\rm es}(v_i,p_k)}\cdot f_{k,0}, \tag{15}$$

where $f_{k,0}$ represents the highest frequency of processor p_k. Selecting the lowest frequency $f_{\rm id}(v_j)$ of processor p_k from Table 1, which is no less than the $f_{\rm id}(v_i,v_j)$, as the actual running frequency of task

Algorithm 4 EASEF

Require: G, P, RANK. // G: an application graph; // P: set of processors; RANK: sequence of tasks
Ensure: S. // S: a schedule

while RANK is not null **do**
  Task v_i = RANK.pop();
  for each processor p_k in P **do**
    Calculate $t_{\text{es}}(v_i, p_k)$ by Eq. (11);
    Calculate $t_{\text{ef}}(v_i, p_k)$ by Eq. (12);
  end for
  Assign task v_i to processor p_k which minimizes the finish time of task v_i;
end while
Rank all tasks into a sequence by non-decreasing order based on t_{ef} and let FT represent the sequence;
while FT is not null **do**
  Task v_j = FT.pop();
  Let p_k represent the processor assigned to v_j;
  Calculate $t_{\text{lf}}(v_j, p_k)$ by Eq. (13);
  Let v_i represent the nearest task assigned prior to v_j on the same processor p_k;
  if v_i exists **then**
    Calculate $f_{\text{id}}(v_i, v_j)$ by Eq. (15);
    Determine $f_{\text{ac}}(v_j)$ by Eq. (16);
  else
    Calculate $\text{slack}(v_j)$ by Eq. (14);
    Calculate $f_{\text{id}}(v_j)$ by Eq. (19);
    Determine $f_{\text{ac}}(v_j)$ by Eq. (20);
  end if
  Determine $t_{\text{af}}(v_j, p_k)$ by Eq. (17);
  Calculate $t_{\text{as}}(v_j, p_k)$ by Eq. (18);
end while
Rank all tasks into a sequence by non-increasing order based on t_{as} and let ST represent the sequence;
while ST is not null **do**
  Task v_i = ST.pop();
  Let p_k represent the processor assigned to v_i;
  Calculate $t_{\text{es}}(v_i, p_k)$ by Eq. (11);
  Let $t_{\text{ef}}(v_i, p_k) = t_{\text{es}}(v_i, p_k) + t_{\text{af}}(v_i, p_k) - t_{\text{as}}(v_i, p_k)$;
  Update $t_{\text{as}}(v_i, p_k)$ with $t_{\text{es}}(v_i, p_k)$;
  Update $t_{\text{af}}(v_i, p_k)$ with $t_{\text{ef}}(v_i, p_k)$;
end while

v_j, the actual running frequency of task v_j should satisfy

$$f_{\text{ac}}(v_j) = \min_{\text{level}\in\text{PState}} \{f_{k,\text{level}}\} \geq \max\left(f_{\text{id}}(v_i, v_j), \frac{\omega_{j,k} \cdot f_{k,0}}{t_{\text{lf}}(v_j, p_k) - t_{\text{es}}(v_j, p_k)}\right), \tag{16}$$

where PState is the set of frequency levels. Then the actual finish time (t_{af}) of task v_j on processor p_k is updated by

$$t_{\text{af}}(v_j, p_k) = t_{\text{lf}}(v_j, p_k). \tag{17}$$

The actual start time (t_{as}) of task v_j on processor p_k is updated by

$$t_{\text{as}}(v_j, p_k) = t_{\text{af}}(v_j, p_k) - \frac{\omega_{j,k}}{f_{\text{ac}}(v_j)} \cdot f_{k,0}. \tag{18}$$

If v_i is the first task of the assigned processor p_k, then the maximum value of slack time can be calculated using Eq. (14). In this case, the ideal frequency of task v_i is defined by

$$f_{\text{id}}(v_i) = \frac{\omega_{i,k}}{\omega_{i,k} + \text{slack}(v_i)} \cdot f_{k,0}. \tag{19}$$

Then the running frequency of task v_i should satisfy

$$f_{\text{ac}}(v_i) = \min_{\text{level}\in\text{PState}} \{f_{k,\text{level}}\} \geq f_{\text{id}}(v_i). \tag{20}$$

Fig. 4 shows the schedule results of the different algorithms for Fig. 1. The makespan of the EASEF algorithm is 205. Furthermore, the CEFT algorithm produces a schedule of length 231 and the CPOP algorithm produces a schedule of length 251. The makespan of the proposed approach is less than those of CEFT and CPOP by 11.26% and 18.33%, respectively. Meanwhile, the results reveal that the EASEF algorithm is effective and saves energy compared with the other two algorithms. This example demonstrates that we can benefit from the application reconstructing phase and frequency equalizing phase in terms of schedule length and total energy consumption.

5 Performance evaluation

In this section, we evaluate the performance of the proposed EASEF algorithm using randomly generated application graphs and two real-world application graphs.

5.1 Experimental settings

The random graphs are generated with three fundamental characteristics as follows:

n: the number of tasks in a DAG.

CCR: the communication to computation ratio. A low CCR application can be considered as a computation-intensive application and a high CCR application can be considered as a communication-intensive application.

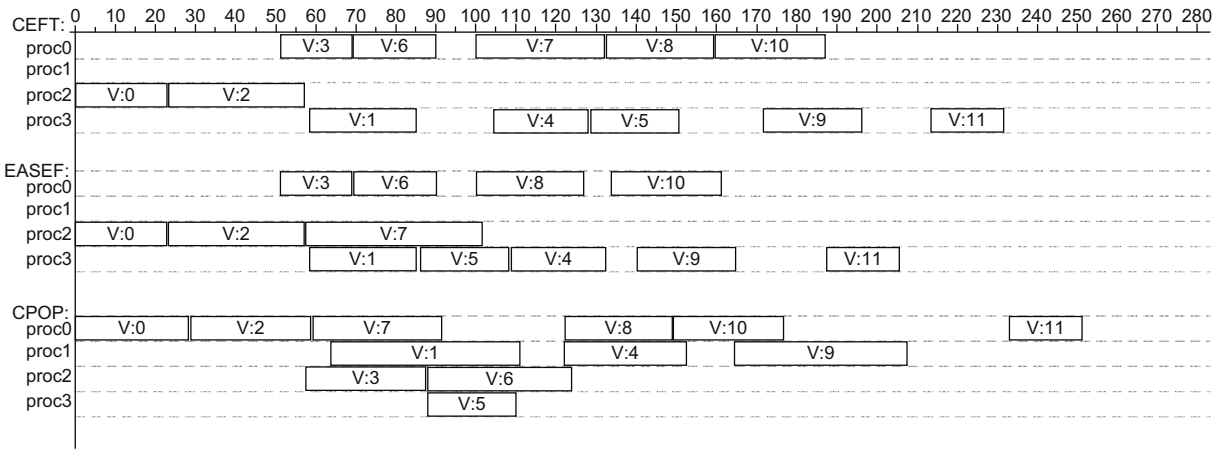


Fig. 4 The schedule results for Fig. 1 using three algorithms

λ: the parallelism factor of DAG. The number of tasks at each level is randomly selected from a uniform distribution with mean value of $\lambda\sqrt{n}$, and the depth of a DAG is randomly generated from a uniform distribution with mean value of $\sqrt{n}/\lambda$. A lower λ leads to a deeper DAG with a low parallelism degree and a higher λ leads to a shorter DAG with a higher parallelism degree.

In our experiments, n is selected from the set $\{16, 32, 64, 100, 200, 400, 500\}$, CCR is determined by the set $\{0.2, 0.5, 1.0, 2.0, 5.0\}$, and λ is chosen from the set $\{0.2, 0.5, 1.0, 2.0, 5.0\}$. To demonstrate the performance improvement of EASEF, two existing algorithms, CPOP and CEFT, with EvenlyDVS (Wang *et al.*, 2013), are used as baseline algorithms to compare the makespan and energy consumption. The schedule length ratio (SLR) and energy consumption ratio (ECR) are defined as follows:

$$\mathrm{SLR} = \frac{\text{makespan}}{\min\limits_{k\in P}\left(\sum\limits_{i\in \mathrm{CP}} \omega_{i,k}\right)}, \tag{21}$$

$$\mathrm{ECR} = \frac{E}{P(f_{s,0}) \cdot \sum\limits_{i\in \mathrm{CP}} \omega_{i,s}}, \tag{22}$$

where s is the processor number determined using Eq. (21), and $P(f_{s,0})$ is the power of processor p_s at the highest frequency.

There are more than 500 random graphs generated for each scenario, and we take the average SLR and average ECR as the final results to avoid scattering effects.

5.2 Random application performance analysis

In this subsection, the effects of three different algorithms on the capability of scheduling random graphs are compared.

Figs. 5 and 6 present the results of the first two sets of experiments with respect to various numbers of tasks. The average SLR and ECR increase with the increase of the number of tasks. In Fig. 5, the average SLR of the EASEF algorithm is less than those of the CPOP and CEFT algorithms by (2.49%, 5.79%), (3.93%, 12.42%), (4.16%, 16.94%), (3.48%, 18.22%), and (2.61%, 21.61%), for 16, 32, 64, 100, and 200 tasks, respectively. The average ECR of the proposed approach is significantly better than those of the two existing algorithms which are not combined with the EvenlyDVS algorithm. With the help of the EvenlyDVS algorithm, both CPOP and CEFT algorithms can obtain good energy saving. However, EASEF still outperforms them in terms of average ECR. Fig. 6 reveals that the average SLR of the CEFT algorithm increases rapidly with the increase of the number of tasks. This is due to the fact that the CEFT algorithm does not fully capture the communication costs of tasks. The proposed approach achieves better performance compared with the two baseline algorithms in terms of schedule length and energy consumption.

The third set of experiments is conducted for comparing the average SLR and ECR of the algorithms with respect to various values of the parallelism factor. Fig. 7 shows that the average SLR and ECR increase with the increase of the parallelism factor. This is because a higher parallelism factor

leads to a shorter DAG; that is, the denominator of Eq. (21) decreases with the increase of the parallelism factor. Further, we can discover that a low parallelism factor leads to a deeper DAG, which is not suitable for distributing tasks on various processors. So, the results of the three algorithms are very close under a low parallelism factor.

Fig. 8 shows the experimental results with respect to different values of CCR. The performance of CEFT is worse than those of CPOP and EASEF, whether CCR is low or high. This is because CEFT always picks a fast processor for the current task without considering communication costs effectively. The average SLR of EASEF is less than those of CPOP and CEFT by (7.33%, 20.90%), (5.99%, 16.80%), (3.23%, 11.40%), (1.78%, 6.77%), and (5.00%, 15.12%), for CCR of 0.2, 0.5, 1.0, 2.0, and 5.0, respectively. The proposed approach achieves better energy saving compared with the two existing algorithms.

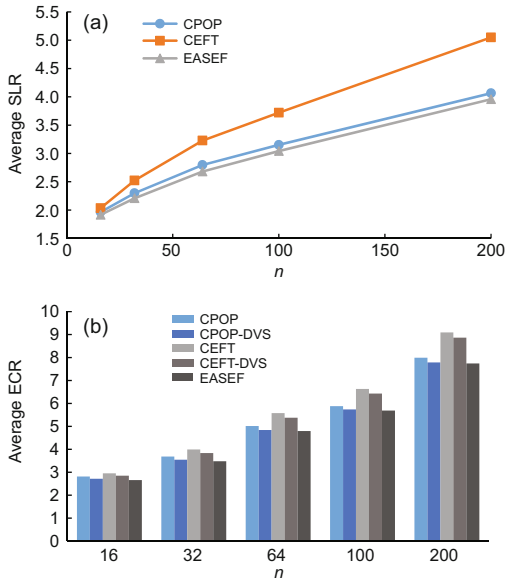


Fig. 5 Average SLR (a) and ECR (b) for various numbers of tasks (CCR=1.0, P=4, λ=1.0)

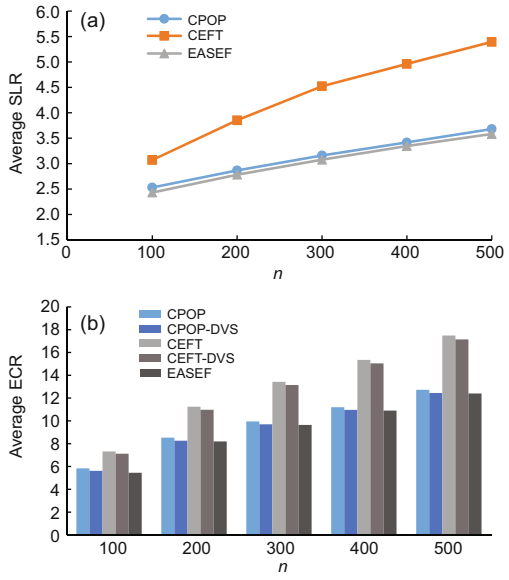


Fig. 6 Average SLR (a) and ECR (b) for various numbers of tasks (CCR=1.0, P=9, λ=1.0)

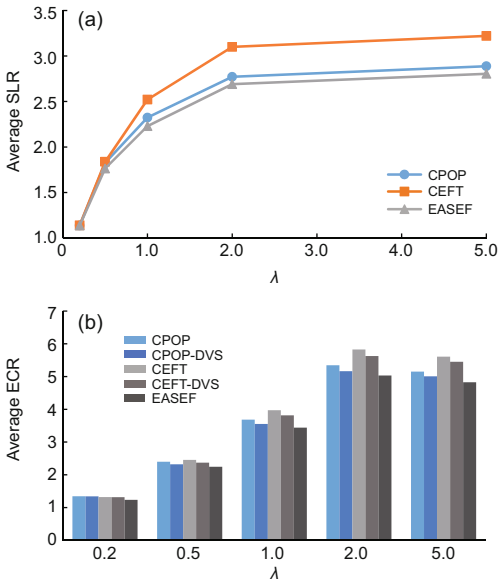


Fig. 7 Average SLR (a) and ECR (b) for various values of the parallelism factor (CCR=1.0, P=4, n=32)

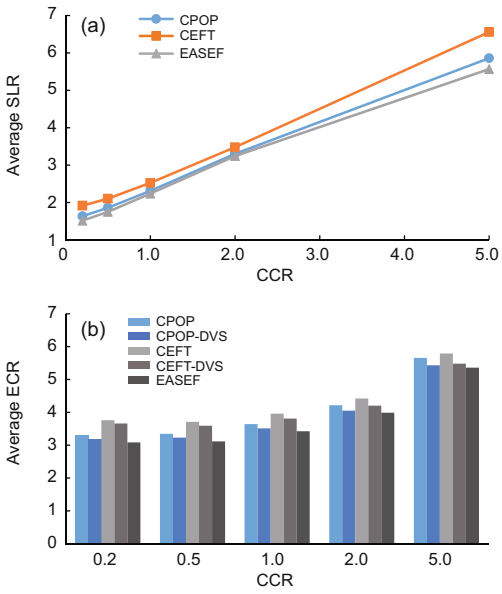


Fig. 8 Average SLR (a) and ECR (b) for various values of CCR (n=32, P=4, λ=1.0)

5.3 Real application performance analysis

In addition to randomly generated applications, we consider two real-world applications. The first is the Gaussian elimination application (Cormen *et al.*, 2009), which is a 5×5 martix. The second is the molecular dynamic code (Kim and Browne, 1988), which consists of 41 tasks. Since the structures of the two real applications are known, it is not necessary to consider the parallelism factor. We just consider the CCR values in our experiments. We assume that the computation cost of a task is randomly generated from a uniform distribution and CCR is selected from the set $\{0.2, 0.5, 1.0, 2.0, 5.0\}$, and the number of processors is set to 4.

Fig. 9 gives the experimental results with respect to different values of CCR for Gaussian elimination applications. The average SLR increases with the increase of CCR, since a higher CCR value means that the system needs more time to transmit data. However, EASEF is still superior to CPOP and CEFT under a high value of CCR. For instance, when CCR equals 5.0, the average SLR of EASEF is less than those of CPOP and CEFT by 26.73% and 29.02%, respectively. The corresponding average energy savings are 15.67% and 16.33%, respectively, when the two counterparts are combined with the EvenlyEVS algorithm. Overall, EASEF outperforms the other two algorithms in terms of average SLR and average ECR.

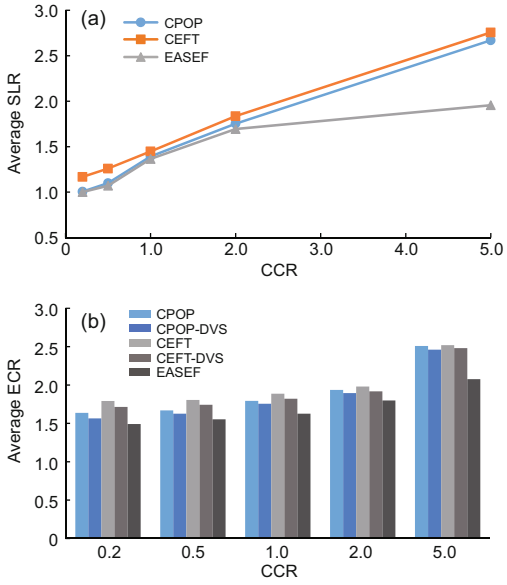


Fig. 9 Average SLR (a) and ECR (b) for Gaussian elimination (n=14, P=4)

Fig. 10 shows the experimental results with respect to various values of CCR for molecular dynamic code applications. Comparing Figs. 9a and 10a, we can find that the performance advantage of EASEF in Fig. 10a is more apparent at a low value of CCR. This is because the parallelism of the molecular dynamic code is higher than that of the Gaussian elimination and the computation cost of an application is the dominant part of schedule length. Moreover, the energy saving effect of EASEF is apparent at a low CCR. For instance, the average energy saving of EASEF compared with CPOP and CEFT is 9.29% and 19.11%, respectively, when the value of CCR is 0.2. Overall, EASEF achieves better performance compared to the two counterparts.

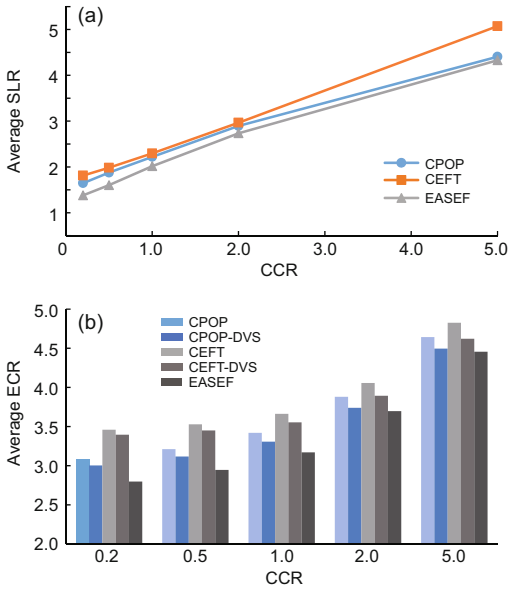


Fig. 10 Average SLR (a) and ECR (b) for molecular dynamic code (n=41, P=4)

6 Conclusions

In this paper, we propose a new scheduling algorithm called EASEF for heterogeneous computing systems. EASEF tries to minimize both finish time and energy dissipation. EASEF comprehensively considers the communication and computation costs of tasks. To better explain the implementation, we divide EASEF into several phases. The first two phases are used to obtain a more reasonable precedence sequence of tasks. In the last phase, EASEF takes a progressive way to equalize the frequency of tasks in order to achieve better energy saving.

We have performed a large number of experiments to demonstrate the effectiveness of EASEF. The results show that in general the proposed algorithm outperforms the other two algorithms in terms of makespan and energy consumption.

In future work, we intend to optimize this algorithm further and adapt some strategies for the cloud environment.

References

Amador, E., Knopp, R., Pacalet, R., *et al.*, 2012. Dynamic power management for the iterative decoding of turbo codes. *IEEE Trans. VLSI Syst.*, **20**(11):2133-2137. [doi:10.1109/TVLSI.2011.2167765]

Bajaj, R., Agrawal, D.P., 2004. Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans. Parall. Distr. Syst.*, **15**(2):107-118. [doi:10.1109/TPDS.2004.1264795]

Bansal, S., Kumar, P., Singh, K., 2003. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. *IEEE Trans. Parall. Distr. Syst.*, **14**(6):533-544. [doi:10.1109/TPDS.2003.1206502]

Bansal, S., Kumar, P., Singh, K., 2005. Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. *J. Parall. Distr. Comput.*, **65**(4):479-491. [doi:10.1016/j.jpdc.2004.11.006]

Benini, L., Bogliolo, A., de Micheli, G., 2000. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst.*, **8**(3):299-316. [doi:10.1109/92.845896]

Boeres, C., Rebello, V.E.F., 2004. A cluster-based strategy for scheduling task on heterogeneous processors. 16th Symp. on Computer Architecture and High Performance Computing, p.214-221. [doi:10.1109/SBAC-PAD.2004.1]

Bozdag, D., Ozguner, F., Catalyurek, U.V., 2009. Compaction of schedules and a two-stage approach for duplication-based DAG scheduling. *IEEE Trans. Parall. Distr. Syst.*, **20**(6):857-871. [doi:10.1109/TPDS.2008.260]

Brown, R., 2008. Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431. Lawrence Berkeley National Laboratory. [doi:10.2172/929723]

Cormen, T.H., Leiserson, C.E., Rivest, R.L., *et al.*, 2009. Introduction to Algorithms. MIT Press, Cambridge.

Freund, R.F., Siegel, H.J., 1993. Guest editor's introduction: heterogeneous processing. *Computer*, **26**(6):13-17.

Fu, F.F., Bai, Y.X., Hu, X.A., *et al.*, 2010. An objective-flexible clustering algorithm for task mapping and scheduling on cluster-based NoC. Academic Symposium on Optoelectronics and Microelectronics Technology and 10th Chinese-Russian Symp. on Laser Physics and Laser Technology Optoelectronics Technology, p.369-373. [doi:10.1109/RCSLPLT.2010.5615317]

Hagras, T., Janeček, J., 2005. A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. *Parall. Comput.*, **31**(7):653-670. [doi:10.1016/j.parco.2005.04.002]

Huang, Q.J., Su, S., Li, J., *et al.*, 2012. Enhanced energy-efficient scheduling for parallel applications in cloud. 12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing, p.781-786. [doi:10.1109/CCGrid.2012.49]

Ilyas, M.U., Khan, S.A., 2001. A clustering heuristic algorithm for scheduling periodic and deterministic tasks on a multiprocessor system. Proc. IEEE Int. Multi Topic Conf., Technology for the 21st Century, p.1-5. [doi:10.1109/INMIC.2001.995305]

Iverson, M.A., Özgüner, F., Follen, G.J., 1995. Parallelizing existing applications in a distributed heterogeneous environment. 4th Heterogeneous Computing Workshop, p.93-100.

Khan, M.A., 2012. Scheduling for heterogeneous systems using constrained critical paths. *Parall. Comput.*, **38**(4-5):175-193. [doi:10.1016/j.parco.2012.01.001]

Kim, S.J., Browne, J.C., 1988. A general approach to mapping of parallel computation upon multiprocessor architectures. Int. Conf. on Parallel Processing, **3**:1-8.

Kwok, Y.K., Ahmad, I., 1996. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Trans. Parall. Distr. Syst.*, **7**(5):506-521. [doi:10.1109/71.503776]

Kwok, Y.K., Ahmad, I., 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, **31**(4):406-471. [doi:10.1145/344588.344618]

Lee, C.H., Shin, K.G., 2004. On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm. 25th IEEE Int. Real-Time Systems Symp., p.319-335. [doi:10.1109/REAL.2004.38]

Lee, Y.C., Zomaya, A.Y., 2011. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Trans. Parall. Distr. Syst.*, **22**(8):1374-1381. [doi:10.1109/TPDS.2010.208]

Li, K.Q., 2012. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. *IEEE Trans. Comput.*, **61**(12):1668-1681. [doi:10.1109/TC.2012.120]

Mehta, N., Amrutur, B., 2012. Dynamic supply and threshold voltage scaling for CMOS digital circuits using in-situ power monitor. *IEEE Trans. VLSI Syst.*, **20**(5):892-901. [doi:10.1109/TVLSI.2011.2132765]

Mei, J., Li, K.L., 2012. Energy-aware scheduling algorithm with duplication on heterogeneous computing systems. ACM/IEEE 13th Int. Conf. on Grid Computing, p.122-129. [doi:10.1109/Grid.2012.32]

Mishra, R., Rastogi, N., Zhu, D.K., *et al.*, 2003. Energy aware scheduling for distributed real-time systems. Proc. Int. Parallel and Distributed Processing Symp., p.1-9. [doi:10.1109/IPDPS.2003.1213099]

Mittal, S., 2014. A survey of techniques for improving energy efficiency in embedded computing systems. *Int. J. Comput. Aided Eng. Technol.*, **6**(4):440-459. [doi:10.1504/IJCAET.2014.065419]

Piyatamrong, B., Ohara, S., Kantakajorn, S., 2000. GTCS: a greedy task clustering and scheduling algorithm for distributed memory processor architecture. Proc. 4th Int. Conf./Exhibition on High Performance Computing in the Asia-Pacific Region, p.310-314. [doi:10.1109/HPC.2000.846567]

Sih, G.C., Lee, E.A., 1993. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parall. Distr. Syst.*, **4**(2):175-187. [doi:10.1109/71.207593]

Tang, X.Y., Li, K.L., Liao, G.P., *et al.*, 2010. List scheduling with duplication for heterogeneous computing systems. *J. Parall. Distr. Comput.*, **70**(4):323-329. [doi:10.1016/j.jpdc.2010.01.003]

Terzopoulos, G., Karatza, H.D., 2013. Dynamic voltage scaling scheduling on power-aware clusters under power constraints. IEEE/ACM 17th Int. Symp. on Distributed Simulation and Real Time Applications, p.72-78. [doi:10.1109/DS-RT.2013.16]

Topcuoglu, H., Hariri, S., Wu, M.Y., 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parall. Distr. Syst.*, **13**(3):260-274. [doi:10.1109/71.993206]

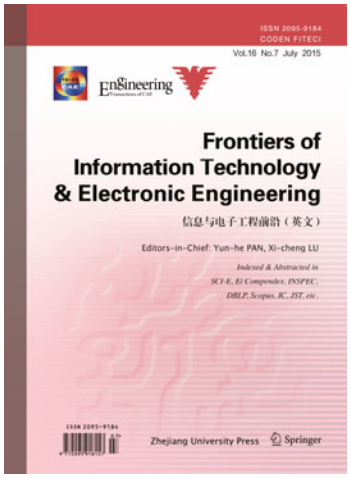
Ullman, J.D., 1975. NP-complete scheduling problems. *J. Comput. Syst. Sci.*, **10**(3):384-393. [doi:10.1016/S0022-0000(75)80008-0]

Wang, L.Z., Khan, S.U., Chen, D., *et al.*, 2013. Energy-aware parallel task scheduling in a cluster. *Fut. Gener. Comput. Syst.*, **29**(7):1661-1670. [doi:10.1016/j.future.2013.02.010]

Yang, T., Gerasoulis, A., 1994. DSC: scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. Parall. Distr. Syst.*, **5**(9):951-967. [doi:10.1109/71.308533]

Zhu, X.M., He, C., Li, K.L., *et al.*, 2012. Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters. *J. Parall. Distr. Comput.*, **72**(6):751-763. [doi:10.1016/j.jpdc.2012.03.005]

FITEE: Call for papers



Editors-in-Chief: Yun-he Pan, Xi-cheng Lu

Frontiers of Information Technology & Electronic Engineering (ISSN 2095-9184, monthly), *FITEE* for short, is an international peer-reviewed journal launched by Chinese Academy of Engineering (CAE) and Zhejiang University, co-published by Springer & Zhejiang University Press. *FITEE* is aimed to publish the latest implementation of applications, principles, and algorithms in the broad area of Electrical and Electronic Engineering, including but not limited to Computer Engineering, Telecommunications, Control Systems, Robotics, Radio Engineering, Signal Processing, Power Engineering, Systems Engineering, Electronics, and Microelectronics.

FITEE is formerly known as *Journal of Zhejiang University-SCIENCE C (Computers & Electronics)* (2010–2014), which has been covered by SCI-E since 2010. Authors of manuscripts submitted or accepted come from 40+ countries and regions, including mainland China, Taiwan, Malaysia, Iran, Korea, Spain, Germany, UK, Greece, USA, Brazil, etc. There are different types of articles for your choice, including **research articles, review articles, science letters, perspective, new technical notes and methods**, etc.

Highlights (metrics & services):

- Key metrics:
  Impact factor: 0.415
  Peer review period: 1–3 months
  From submission to publication (currently): <10 months
  Frequency of publication: monthly
  Editorial board: 16 foreign members, 29 domestic members (including 16 members of CAE)
- Timely and high-quality service for authors and readers
- Rigorous editing and proof-reading
- **Article in press**: Accepted articles will be pushed online immediately after the acceptance
- Innovative techniques adopted:
  CrossMark, to track content changes
  ORCID, to connect research and researchers
- **Peer reviewer comments** (before publication) are selected by editor to be demonstrated and **open peer comments** (after publication) can be provided by readers on the article page
- **English summary** is provided for each paper to give readers a quick view and **Chinese summary** to a wider audience of Chinese readers, both freely accessible
- Abstracted/Indexed in: SCI-E, EI-Compendex, Scopus, INSPEC, Google Scholar, DBLP, etc.
- Full text is available from www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com

Thanks for your attention and welcome your contribution!

Online submission:
http://www.editorialmanager.com/zusc/

Manuscript guidelines:
http://www.zju.edu.cn/jzus/manuscript.php

Contact:
Editorial Office of *J. Zhejiang Univ.-SCIENCE (A/B) & FITEE*
38 Zheda Road, Hangzhou 310027, China
Managing Editors: Helen Zhang & Ziyang Zhai
jzus@zju.edu.cn; jzus_zzy@zju.edu.cn
+86-571-87952276/87952783