Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 465702, 9 pages
http://dx.doi.org/10.1155/2014/465702

Research Article

A Synthesized Heuristic Task Scheduling Algorithm

Yanyan Dai and Xiangli Zhang

Institute of Information and Communication, Guilin University of Electronic Technology, Guilin 541004, China

Correspondence should be addressed to Yanyan Dai; 799582091@qq.com

Received 3 June 2014; Revised 24 July 2014; Accepted 7 August 2014; Published 1 September 2014

Academic Editor: Dehua Xu

Aiming at the static task scheduling problems in heterogeneous environment, a heuristic task scheduling algorithm named HCPPEFT is proposed. In task prioritizing phase, there are three levels of priority in the algorithm to choose task. First, the critical tasks have the highest priority, secondly the tasks with longer path to exit task will be selected, and then algorithm will choose tasks with less predecessors to schedule. In resource selection phase, the algorithm is selected task duplication to reduce the interresource communication cost, besides forecasting the impact of an assignment for all children of the current task permits better decisions to be made in selecting resources. The algorithm proposed is compared with STDH, PEFT, and HEFT algorithms through randomly generated graphs and sets of task graphs. The experimental results show that the new algorithm can achieve better scheduling performance.

1. Introduction

A heterogeneous computing system (HCS) is a computing platform with diverse sets of interconnected resources via high speed network to execute parallel and distributed applications. Due to diverse computational resources, the efficiency of an application on the available resources is one of the key factors for achieving high performance computing. The general form of task scheduling problem in HCS can be represented as directed acyclic graph (DAG). The common objective of scheduling is to assign tasks onto suitable resources and order their execution so that task precedence requirements are satisfied with a minimum schedule length [1]. The scheduling problem is shown to be NP-complete [2–4], so it is expected to be solved by heuristic algorithm.

A typical task scheduling model is based on a directed acyclic graph, including static and dynamic scheduling. Heuristic algorithm is static scheduling algorithm, which is composed of duplication-based algorithms, clustering algorithms, and list scheduling algorithms [3]. List scheduling algorithms are widely used for their high scheduling efficiency and simple design idea. The basic idea of list scheduling is to construct a schedule list by assigning priority for each task, and tasks are selected to a processor which minimizes the execution time. Classical examples of list scheduling algorithms were proposed in [5–10]. The heterogeneous earliest finish time (HEFT) [5] uses a recursive procedure to compute the rank of a task by traversing the graph upwards from the exit task and vice-versa for critical path on a processor (CPOP) [5]. Improvement heterogeneous earliest finish time (IHEFT) [6] acquires a better task list by changing the task's upward weight calculation method. Stand deviation-based algorithm for task scheduling (SDBATS) [7] uses the standard deviation of the expected execution time of a given task on the available resources as a key attribute for assigning task priority. In [8], the proposed algorithm determines the critical path of the task graph and selects the next task to be scheduled in a dynamic fashion. Predict earliest finish time (PEFT) [9] algorithm is only based on computation of an optimistic cost table (OCT) that is used to rank tasks and for resource selection. A novel algorithm named the longest dynamic critical path (LDCP) [10] assigns priorities by considering the critical path attribute of the given DAG.

The idea of task duplication is to try to duplicate the parents of the current selected task onto the selected resource or on to the other resources, aiming to reduce or optimize the task finish time. Many duplication-based algorithms are proposed in recent years; for example, in [11], a path priority-based heuristic task scheduling algorithm was studied. Heterogeneous critical task (HCT) [12] scheduling algorithm defines the critical task and the idle time slot. Selected task duplication for heterogeneous (STDH) [13] duplicates

the parent tasks for advancing the earliest starting time of the current candidate task to reduce the inter processor communication cost. Heterogeneous earliest finish time with duplication (HEFD) [14] uses task variance as computation capacity heterogeneity factor for setting weights to tasks and edges. The algorithm in [15] is a three-phase algorithm with a dynamic phase to assign a priority to each task. To search and delete redundant task duplications dynamically in the process of scheduling, a novel resource-aware scheduling algorithm with duplications (RADS) was proposed in [16]. There are some researches such as [17, 18] combining DVS technique with duplication strategy to reduce energy consumption. The algorithms proposed in [19, 20] are designed to solve the problem of resource waste. Other duplication-based algorithms such as selective duplication (SD) [21] and heterogeneous critical parents with fast duplicator (HCPFD) [22] are also worth studied.

Clustering algorithms merge tasks in a DAG to an unlimited number of clusters, and tasks in a cluster are allocated on the same resource. Some classical examples are cluster mapping algorithm (CMA) [23], clustering and scheduling system (CASS) [24], objective flexible clustering algorithm (OFCA) [25], and so on. In [26, 27], two novel algorithms were proposed, but they have limitations in higher heterogeneity systems. To reduce energy consumption without increasing the schedule length, [28] reclaims both static and dynamic slack time and employs different frequency adjusting techniques in different slack time, and [29] studies the slack time for noncritical jobs, extends their execution time, and reduces the energy consumption without increasing the task's execution time as a whole. Additionally, these methods are based on DAG scheduling: heterogeneous critical path first synthesized (HCPFS) [30], heterogeneous select value (HSV) [31], heterogeneous chip multiprocessor global comparatively optimum task scheduling (HGCOTS) [32], and so on.

The HEFT [5] algorithm has two phases: the task prioritizing phase and the processor selection phase. In the first phase, a task list is generated by sorting the tasks in decreasing order of the upward rank. In the processor selection phase, tasks are scheduled to the best processor that minimizes the task's finish time. However, the algorithm does not take into account the critical task and interprocessor communication cost impact on the whole task graph scheduling time. Task scheduling efficiency is not high which still should be optimized further.

The PEFT [9] algorithm is based on the computation of an optimistic cost table (OCT) on which task priority and processor selection are based. The OCT is a matrix in which the rows indicate the number of resources, where each element $\mathrm{OCT}(t_i, r_j)$ indicates the maximum of the shortest paths of t_i children's tasks to the exit node considering that resource r_j is selected for task t_i. The algorithm has the same time complexity as the HEFT and SDBATS algorithms, that is, $O(n^2 \cdot m)$ for n tasks and m resources. But the algorithm does not take into account the critical tasks, which decide the whole DAG scheduling time. So this algorithm delays the scheduling length to some extent.

The STDH [13] algorithm duplicates the parent tasks for advancing the earliest starting time of the current candidate task to reduce the interprocessor communication cost. By this way, the overall run time of application is shortened, but the algorithm scheduled only by one attribute. In the case of that different tasks should have the same priority, it cannot work well. Therefore, in order to evaluate the priority of tasks reasonably, multiple attributes are more suitable for sorting tasks.

In this paper, we propose a synthesized heuristic task scheduling algorithm based on both duplication-based techniques and list-based approach for heterogeneous computing systems. The HCPPEFT algorithm has two phases, the task prioritizing phase and the resource selection phase (see Algorithm 1). In the first phase, we suggest a new approach of critical task, the tasks with longer path to exit task, and the number of predecessors to construct the scheduling queue. In the second phase, the duplication of tasks is optimized for all immediate parent tasks and look ahead policies is to guarantee the tasks ahead will finish earlier. The rest of the paper is organized as follows. In Section 2, we define the task scheduling problem and discuss some basic attributes of DAG scheduling. Section 3 introduces HCPPEFT algorithm and in Section 4 the results of the experiment are discussed. Section 5 concludes the paper.

2. Problem Definition

2.1. Task Model. A scheduling system model is composed of an application in heterogeneous computing environment. An application is represented by $G = (T, E)$, where T is the set of t tasks and E is the set of e edges between the tasks. Each edge $e_{ij} \in E$ represents the precedence constraint such that task t_i should complete its execution before task t_j starts. Communication cost required to be transmitted from task t_i to task t_j is represented by a $C(t \times t)$ matrix, in which each c_{ij} gives the amount of data on e_{ij} edge. The heterogeneous system consists of a set of resources $R = \{r_1, r_2, \ldots, r_m\}$ of m independent resources fully connected by a high speed network. A computation cost matrix is represented as $W(t \times m)$, in which each $w_{i,j}$ gives the estimated time to execute t_i on resource r_j. We further assume that any two connecting tasks scheduled on the same resource have zero communication costs.

2.2. DAG Scheduling Attributes. Before proceeding to the next section, it is necessary to discuss some scheduling attributes such as rank upward, rank downward, earliest start time, earliest finish time, and optimistic cost table, which will be used in the proposed scheduling algorithm.

Definition 1. A task with no predecessors is called an entry task (t_{entry}), and $\mathrm{pred}(t_i)$ denotes the set of immediate predecessors of task t_i in a given task graph. Similarly, a task with no successors is called an exit task (t_{exit}), and $\mathrm{succ}(t_i)$ denotes the set of immediate successors of task t_i in a given task graph. If a task graph has multiple entry or exit nodes,

(1) Compute rank upward, rank downward, predecessors and OCT table for all tasks
(2) Compute $\mathrm{CT}(t_i) = \mathrm{rank}_u(t_i) + \mathrm{rank}_d(t_i)$ for each task t_i in the graph.
If $\mathrm{CT}(t_i) = \mathrm{CT}(t_{\mathrm{entry}})$, where t_{entry} is the entry task, CT = $\{t_{\mathrm{entry}}\}$ is the set of tasks on the critical path.
(3) Sort the tasks in scheduling queue by critical task, decreasing order of rank value and increasing order of predecessors.
(4) **While** there are unscheduled tasks in the queue **do**
(5) Select the first task t_i from the queue for scheduling
(6) **for** each resource r_j in the resource set ($r_j \in r$) **do**
(7) Sort immediate parents t_m of task t_i by decreasing order of data arrival time
(8) **while** the duplication condition is satisfied do
(9) **if** the duplication of t_m can reduce its start execution time **then**
(10) Duplicate t_m and update the earliest start time of t_i and free time slots.
(11) **end**
(12) **end**
(13) Compute the earliest finish time by (5).
(14) $O_{\mathrm{EFT}}(t_i, r_j) = \mathrm{EFT}(t_i, r_j) + \mathrm{OCT}(t_i, r_j)$
(15) **end**
(16) Assign task t_i to the resource r_j with minimize O_{EFT} of task t_i.
(17) **end**

Algorithm 1: The HCPPEFT algorithm.

a dummy entry or exit node with zero communication edges and zero weight can be added to the graph.

Definition 2. The upward rank of a task t_i is calculated using the following equation:

$$\mathrm{rank}_u\left(t_i\right) = \overline{w_i} + \max_{t_j \in \mathrm{succ}\left(t_i\right)} \left\{\overline{c_{ij}} + \mathrm{rank}_u\left(t_j\right)\right\}, \quad (1)$$

where $\overline{w_i}$ is the average computation cost of task t_i, and $\overline{c_{ij}}$ is the average communication cost of the edge from task t_i to task t_j. For the exit task t_{exit}, upward rank value is

$$\mathrm{rank}_u\left(t_{\mathrm{exit}}\right) = \overline{w_{\mathrm{exit}}}. \quad (2)$$

Similarly, the downward rank of task t_i is defined by the following equation:

$$\mathrm{rank}_d\left(t_i\right) = \max_{t_j \in \mathrm{pred}(t_i)} \left(\overline{w_j} + \overline{c_{ji}} + \mathrm{rank}_d\left(t_j\right)\right). \quad (3)$$

For the entry task t_{entry}, the downward rank value is equal to zero.

Definition 3. $\mathrm{EST}(t_i, r_j)$ and $\mathrm{EFT}(t_i, r_j)$ are the earliest start time and earliest finish time of a task t_i on the resource r_j and are defined by

$$\mathrm{EST}\left(t_i, r_j\right) = \max\left\{\mathrm{avail}\left(t_i, r_j\right), \max_{t_m \in \mathrm{pred}\left(t_i\right)} \left\{\mathrm{AFT}\left(t_m\right) + c_{mi}\right\}\right\}, \quad (4)$$

$$\mathrm{EFT}\left(t_i, r_j\right) = w_{i,j} + \mathrm{EST}\left(t_i, r_j\right), \quad (5)$$

where $\mathrm{avail}(t_i, r_j)$ is the earliest time at which resource r_j is ready for task execution, $\mathrm{AFT}(t_m)$ is actual finish time of task t_i, and c_{mi} is communication cost of the edge from task t_m to task t_i. Before computing earliest finish time of a task t_i, all the immediate predecessor tasks of t_i must have been scheduled. For the entry task t_{entry}, $\mathrm{EST}(t_{\mathrm{entry}}, r_j) = 0$.

Definition 4. If the task start execution time depends on the arrival time of parent tasks, there will be a free time slot, that is, the task to wait for the time of arrival of the parent task, can be defined as

$$\mathrm{slot}\left(t_i, r_j\right) = \mathrm{EST}\left(t_i, r_j\right) - \mathrm{avail}\left(t_i, r_j\right). \quad (6)$$

Definition 5. The OCT is a matrix in which the rows indicate the number of tasks and the columns indicate the number of resources, where each element $\mathrm{OCT}(t_i, r_j)$ represents the maximum optimistic processing time of the children of task t_i. The OCT value of task t_i on resource r_j is recursively defined by

$$\mathrm{OCT}\left(t_i, r_j\right) = \max_{t_j \in \mathrm{succ}\left(t_i\right)} \left[\min_{r_k \in r} \left\{\mathrm{OCT}\left(t_j, r_k\right) + w_{j,k} + c_{ij}\right\}\right]. \quad (7)$$

Definition 6. The schedule length (makespan) of the task graph denotes the finish time of the last task and is defined as follows:

$$\mathrm{makespan} = \max\left\{\mathrm{AFT}\left(t_{\mathrm{exit}}\right)\right\}. \quad (8)$$

The objective function of the task scheduling problem is to determine an assignment of tasks of a given task graph to resources, such that its schedule length is minimized, which satisfying all precedence constraints.

3. The Proposed Algorithm HCPPEFT

3.1. Task Prioritizing Phase. The critical path is a path with the longest execution time in directed acyclic graph, which

of task play a decisive role to finish time. In the process of task scheduling, critical task is assigned the highest priority to make it firstly scheduled. However, critical task usually have one or more parent tasks. If parent tasks cannot get a reasonable scheduling, the start execution time will be delayed. Therefore it not only assigns besides the critical tasks the highest priority, but also needs the parent tasks of critical tasks to allocate higher-priority. Immediate parent tasks are sorted in a decreasing order of rank upward values. If the rank upward values are equal, select the tasks with respect to the predecessors increase. In order to construct the task scheduling queue, two empty queues of tasks are constructed firstly, which are CTQ and RTQ. CTQ is used to store critical task and RTQ is used to store the task scheduling ready queue. The process of constructing task scheduling queues is expressed as Figure 1. The detailed explain is presented as follows.

(1) Upward rank (rank_u), downward rank (rank_d), and the summation of upward and downward ranks ($\text{rank}_u + \text{rank}_d$) values for all tasks are computed.

(2) The critical path length is equal to the entry task's priority. The entry task is marked as a critical path task. If $\text{CT}(t_i) = \text{rank}_u(t_i) + \text{rank}_d(t_i)$ for each task t_i in DAG, then $\text{CT}(t_i) = \text{CT}(t_{\text{entry}})$, where t_{entry} is the entry task, $\text{CT} = \{t_{\text{entry}}\}$ is the set of tasks on the critical path. The all critical path task will be added to the CTQ by decreasing order of rank_u.

(3) The CTQ at the beginning contains all critical tasks. In fact a critical task will be omitted from the CTQ and added to the RTQ after all parent tasks are added to the RTQ. The priority of the parent tasks is determined based on their rank_u. The parent task with the highest rank_u gets high priority.

(4) If there are two or more parent tasks of rank_u are equal. The priority of the parent tasks is generated by increasing order of the number of predecessors.

(5) The above steps are executed until the CTQ queue is null. The RTQ queue is task scheduling queue.

To illustrate how the task prioritizing queue is constructed, a random task graph with 10 tasks and 16 edges is shown in Figure 2. The corresponding computation cost with respect to each resource (i.e., $\{r_1, r_2, r_3\}$) is presented in Table 1. Based on the values in Table 2, the critical path in Figure 2 is $\{t_1, t_3, t_7, t_{10}\}$. The all critical tasks will be added to the CTQ by decreasing order of rank upward values; that is, $\text{CTQ} = \{t_1, t_3, t_7, t_{10}\}$. Because the entry task t_1 has no parent task, it will be omitted from the CTQ and added to the RTQ immediately. After that, task t_3 is added in RTQ and deleted from CTQ. Until parents tasks t_2 and t_5 have been selected, the critical task t_7 can be selected. Till now, we can get $\text{CTQ} = \{t_{10}\}$ and $\text{RTQ} = \{t_1, t_3, t_5, t_2, t_7\}$. Before the critical task t_{10} is selected, the parent tasks t_8 and t_9 must have been scheduled. But the rank upward values of parent tasks t_8 and t_9 are equal, selecting tasks with respect to the predecessors increase. Task t_8 is selected and then is task t_9. At this time, the scheduling order of remaining tasks is $\{t_4, t_8, t_6, t_9, t_{10}\}$. Finally, the task queue is constructed, which is $\{t_1, t_3, t_5, t_2, t_7, t_4, t_8, t_6, t_9, t_{10}\}$.

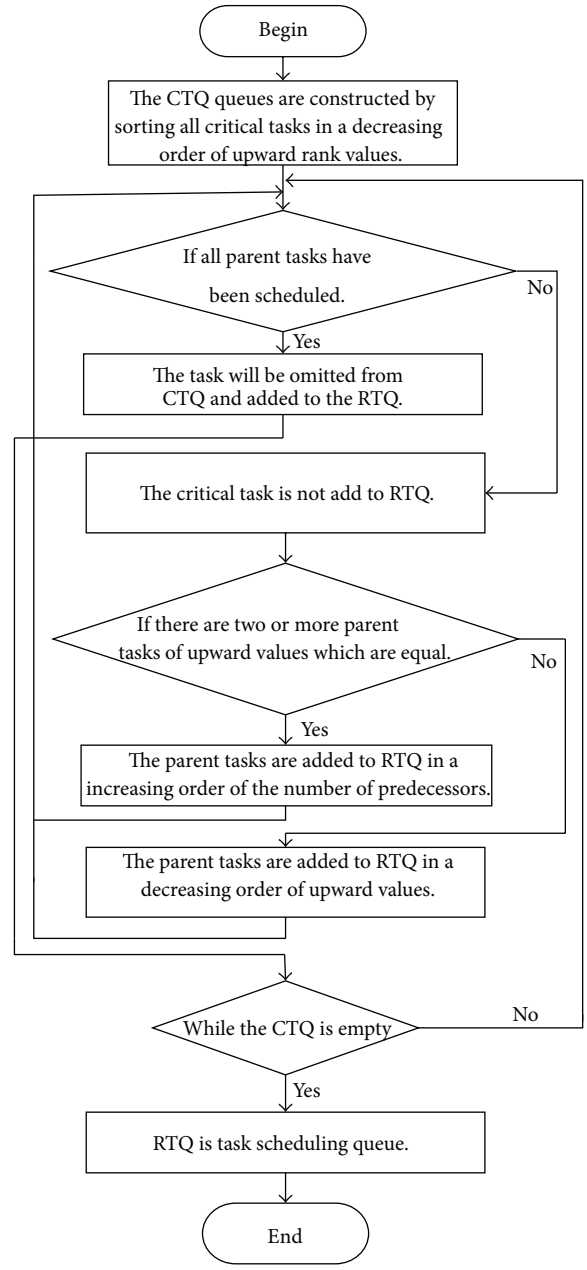

Figure 1: Constructing task scheduling queues.

3.2. Resource Selection Phase. In this phase, tasks are assigned to the resources and duplication is employed to minimize the finish time. To start execution on a resource, the task t_i has to wait for the data arrivals from all of its immediate parents, so as to meet precedence constraints. The parent task t_m of task t_i whose schedule on different resources and whose data arrival time at task t_i is the latest parent task. Duplication condition [13] is defined as

$$\begin{aligned} &\text{slot}\left(t_i, r_j\right) \geq w_{m,j}, \\ &\text{EFT}\left(t_m, r_j\right) < \text{EST}\left(t_i, r_j\right). \end{aligned} \tag{9}$$

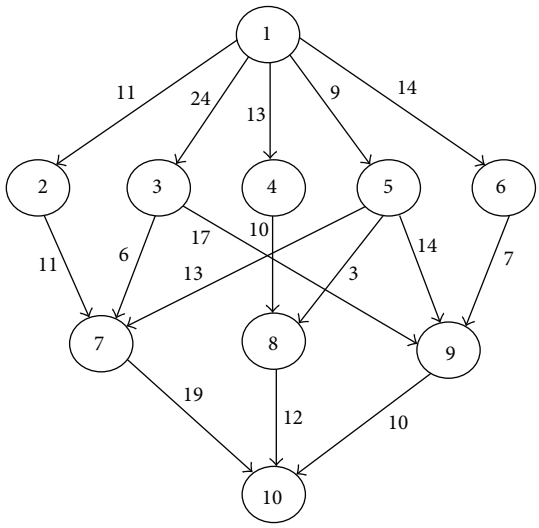

Figure 2: A sample task graph with 10 tasks.

Table 1: Computation costs.

Task	r_1	r_2	r_3
1	11	19	6
2	18	5	12
3	7	20	13
4	12	16	4
5	19	5	16
6	8	23	14
7	14	24	19
8	7	9	20
9	12	14	16
10	17	7	13

If it is satisfied, then the parent task t_m will be duplicated on the resource that t_i assigned to. Simultaneously compute the earliest finish time by (5) and update the free time slots by (6). It is to be noted that the best-suited resource may not be achieved if we followed by (5). Because the best scheduling consider not only the current task's gain in complete time but also the gain in a sequence of tasks. With computation of an OCT by (7) not only cannot increase the time complexity but also guarantee that the tasks ahead will finish earlier. In this way, we compute the optimistic EFT (O_{EFT}), which sums to EFT the computation time of the longest path to the exit task, to select a best-suited resource. O_{EFT} is defined as

$$O_{\text{EFT}}\left(t_i, r_j\right) = \text{EFT}\left(t_i, r_j\right) + \text{OCT}\left(t_i, r_j\right). \quad (10)$$

For a given DAG, the time complexity of scheduling algorithm is usually expressed in terms of number of tasks n and number of resources m. The time complexity of STDH is of constructing schedule queue $O(n^2)$. The resource selection for all tasks can be done in time complexity $O(n^2 \cdot m)$. The total time is $O(n^2 \cdot m)$. PEFT requires the computation of an OCT table is $O(m(n^2+n))$, and to assign the tasks to resources, the time complexity is of the order $O(n^2 \cdot m)$. The total time is $O(n^2 \cdot m)$. The HEFT algorithm has a $O(n^2 \cdot m)$ complexity for n tasks and m resources. The HCPPEFT algorithm requires the construing task scheduling queue is $O(n^2 \cdot m)$, and the time complexity of resource select phase is $O(m(n^2 + n))$. That is, the HCPPEFT algorithm has the same time complexity as the STDH, PEFT, and HEFT algorithms.

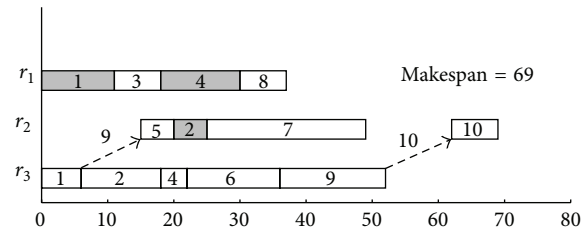

Figure 3: Schedule of task graph with the HCPPEFT algorithm.

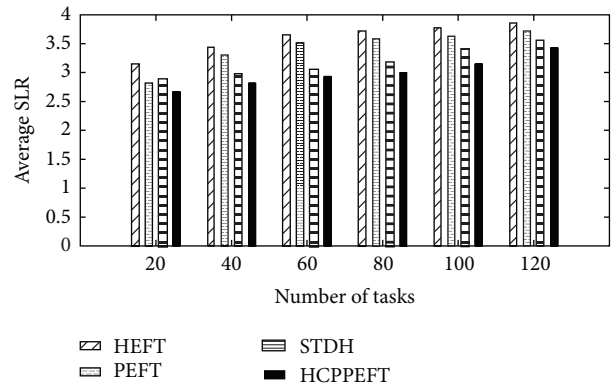

Figure 4: Average SLR with CCR = 2 for random task graphs.

3.3. HCPPEFT Algorithm Implementation. To evaluate the performance of the HCPPEFT algorithm, we implement this algorithm on the PC platform using MATLAB software.

A randomly generated task graph is shown in Figure 2 firstly. Then the corresponding computation costs are presented in Table 1. By calculating (7), the OCT values are shown in Tables 3 and 4 which give an example that demonstrates the HCPPEFT for the DAG of Figure 2 in the end.

As an illustration, Figure 3 presents the schedules obtained by the HCPPEFT algorithm, where the gray blocks with numbers are the duplicated tasks. The schedule length, which is equal to 69, is shorter than that of the related algorithms; specifically, the schedule lengths of STDH, PEFT, and HEFT Algorithm are 73, 78, and 77, respectively.

4. Experimental Results and Discussion

This section presents a performance comparison of the proposed algorithm HCPPEFT with four well-known task scheduling algorithms such as STDH, PEFT, and HEFT by randomly generated task graph and sets of task graphs. Three metrics are used for performance evaluation.

Table 2: Values of attributed used in HCPPEFT algorithm for task graph in Figure 2.

t_i	$\text{rank}_u(t_i)$	$\text{rank}_d(t_i)$	$\text{rank}_u(t_i) + \text{rank}_d(t_i)$	Number of predecessors
t_1	105.667	0.000	105.667	0
t_2	73.000	23.000	96.000	1
t_3	69.667	36.000	105.667	1
t_4	57.000	25.000	82.000	1
t_5	76.667	21.000	97.667	1
t_6	58.333	26.000	84.333	1
t_7	50.333	55.333	105.667	3
t_8	36.333	45.667	82.000	2
t_9	36.333	66.333	102.667	3
t_{10}	12.333	93.333	105.667	3

Table 3: Optimistic cost table for the DAG of Figure 2.

Task	r_1	r_2	r_3
t_1	47	51	45
t_2	31	31	32
t_3	31	31	32
t_4	29	29	29
t_5	31	31	32
t_6	28	21	28
t_7	17	7	13
t_8	17	7	13
t_9	17	7	13
t_{10}	0	0	0

4.1. Comparisons Metrics

4.1.1. Scheduling Length Ratio. Since a large set of task graphs with different properties is used, it is necessary to normalize the schedule length to the lower bound, which is called the schedule length ratio (SLR). SLR is defined as follows:

$$\text{SLR} = \frac{\text{makespan}}{\sum_{t_i \in \text{CP}_{\min}} \min_{r_j \in R} \left\{ w_{\{i,j\}} \right\}}, \tag{11}$$

where $\text{CP}_{\min}$ is the minimum computation cost of the critical path tasks in SLR. There is no makespan less than the denominator of the SLR equation since the denominator is the lower bound. Therefore, the algorithm with the lowest SLR is the best algorithm with respect to performance.

4.1.2. Speedup. The speedup value for a given task graph is defined as the ratio of sequential execution times to the parallel execution times. The sequential execution time is computed by assigning all tasks to a single resource that minimizes the computation costs. Speedup is defined as follows:

$$\text{speedup} = \frac{\min_{r_j \in R} \left\{ \sum_{t_i \in T} w_{i,j} \right\}}{\text{makespan}}. \tag{12}$$

4.1.3. Efficiency. In the general case, efficiency is defined as the ratio of the speedup value to the number of resources used in schedule task graph. Efficiency is defined as follows:

$$\text{Efficiency} = \frac{\text{speedup}}{\text{number of resource}}. \tag{13}$$

4.2. Randomly Generated Task Graph and Performance Comparison.

To evaluate the performance of the HCPPEFT algorithm, we first considered randomly generated task graphs. For this purpose, a random graph generator available at [13] was implemented to generate a variety of weighted graphs with various characteristics. The input parameters of the generator are the communication to computation ratio (CCR), number of tasks, out degree of a node (out_degree), edge weight, node weight, and number of resources. Our simulation framework first generate a large set of random task graphs with different characteristics, which is followed by the execution of the task scheduling algorithms, and finally, it computes the performance metrics.

The performance of the HCPPEFT, STDH, PEFT, and HEFT algorithms is compared with respect to various graph characteristics according to SLR, speedup, and efficiency.

Firstly, we have evaluated the performance of the algorithm with respect to various numbers of tasks and the number of resources was considered as a fixed value of 10. Each value of the experimental results is the average of the results produced from 200 different random task graphs. In these experiments CCR and out_degree were considered as fixed value of 2 and 5, and n was restricted to the following values: $n \in \{20, 40, 60, 80, 100, 120, 150, 200\}$. The edge weight is generated randomly from 1 to 300, as the node weight is 1 to 30. In Figures 4 and 5, the simulation results show that the new algorithm outperforms the other algorithms according to the SLR and speedup. The average SLR value of the HCPPEFT algorithm is better than the HEFT algorithm by 19.99%, the PEFT algorithm by 14.43%, and the STDH algorithm by 5.72%; and the average speedup value of the HCPPEFT algorithm is better than the HEFT algorithm by 16.33%, the PEFT algorithm by 10.79%, and the STDH algorithm by 8.69%.

The next experiment is with respect to CCR increment. Each value of the experimental results is the average of the results produced from 200 different random task graphs. In

Table 4: Schedule produced by the HCPPEFT algorithm in each iteration.

Step	Ready queue	Task selected	EFT			O_{EFT}			Resource selected
			r_1	r_2	r_3	r_1	r_2	r_3	
1	t_1	t_1	11	19	**6**	58	70	**51**	r_3
2	t_3	t_3	**18**	39	19	**49**	70	51	r_1
3	t_5, t_2, t_7	t_5	37	**20**	22	68	**51**	54	r_2
4	t_2, t_7	t_2	36	25	**18**	67	56	**50**	r_3
5	t_7	t_7	**47**	49	52	64	**56**	65	r_2
6	$t_4, t_8, t_6, t_9, t_{10}$	t_4	31	65	**22**	60	94	**51**	r_3
7	t_8, t_6, t_9, t_{10}	t_8	**37**	58	43	**54**	65	56	r_1
8	t_6, t_9, t_{10}	t_6	45	72	**36**	73	93	**64**	r_3
9	t_9, t_{10}	t_9	55	63	**52**	72	70	**65**	r_3
10	t_{10}	t_{10}	79	**69**	81	79	**69**	81	r_2

Table 5: Reference and related parameters for the sets of task graphs.

Serial number	Number of tasks	Number of resource	CCR	Reference
1	10	3	0.6025	[5, 6, 32]
2	5	3	0.4508	[13]
3	12	4	0.3003	[12]
4	10	3	0.3946	[30]
5	10	3	0.4170	[9]

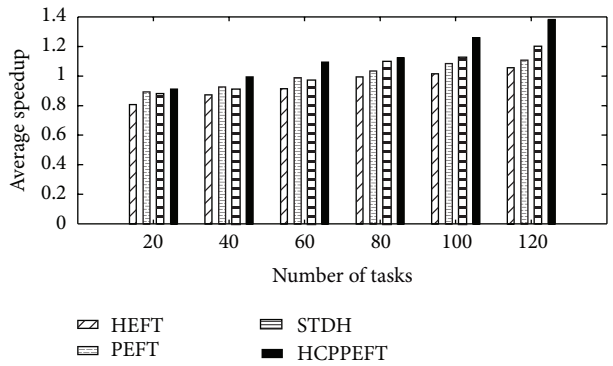

Figure 5: Average speedup with CCR = 2 for random task graphs.

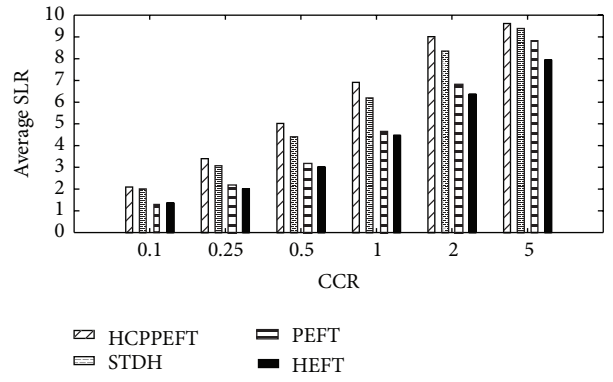

Figure 6: Average SLR with 100 nodes for random task graphs.

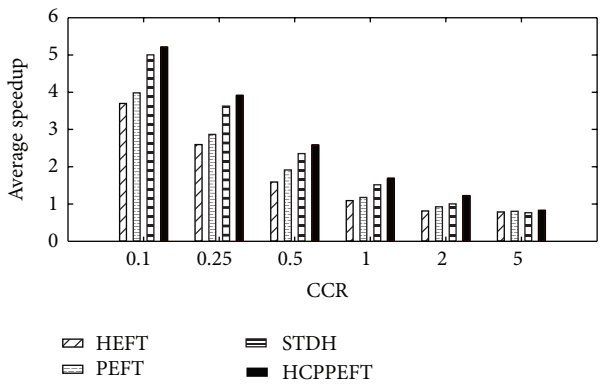

Figure 7: Average speedup with 100 nodes for random task graphs.

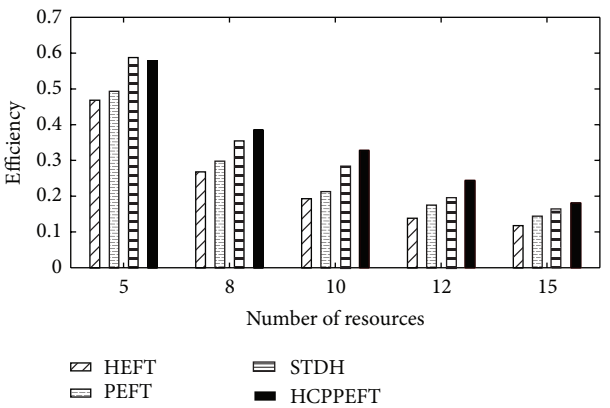

Figure 8: Efficiency for different numbers of resources.

each graph, the CCR is randomly selected from 0.1, 0.25, 0.5, 1, 2, and 5, and the node weight is generated randomly from 1 to 30, as the edge weight is 1 to 300. Also m, n, and out_degree are fixed to 10, 100, and 5, respectively. The results are shown in Figures 6 and 7. In comparison to the HEFT, PEFT, and STDH algorithms on all the generated graphs, the average SLR value obtained by the HCPPEFT algorithm is better by as much as 43.35%, 32.83%, and 7.08%, respectively; and the average speedup value is better by as much as 31.62%, 24.56%, and 7.72%, respectively. This improvement is due to the duplication phase and predicting earliest finish time. The communication cost can reach its lowest ratio by duplication with respect to the increment of CCR; therefore the new algorithm will have great improvement. On the other hand, we can select the best-suited resource that achieves a shorter finish time for the tasks in the next steps by forecasting the impact of an assignment for all children of the current task.

The last set of experiments compare the performance of the algorithm as the value of resource numbers increase. Each value of the experimental results is the average of the results produced from 200 different random task graphs. In these experiments CCR, n, and out_degree were fixed to 0.5, 150, and 5, respectively, and the edge weight is generated randomly from 1 to 300 and the node weight is 1 to 30. The number of resources was restricted to the following values: $m \in \{5, 8, 10, 12, 15\}$. In Figure 8, the simulation results show that the HCPPEFT algorithm outperforms the HEFT, PEFT, and STDH algorithms by 30.98%, 22.95%, and 7.45%, respectively. As was expected, the average scheduling length of HCPPEFT, HEFT, PEFT, and STDH algorithms is reduced as the number of resources increases. This decrement is due to parallelism characteristics.

4.3. Sets of Task Graphs and Performance Results. In addition to randomly generated task graph, we also use the sets of task graphs to evaluate the performance of the new algorithm. The reference and related parameters of the sets of task graphs are shown in Tables 3 and 5. The schedule lengths of HCPPEFT, STDH, PEFT, and HEFT algorithmss are presented in Figure 9. The results show that the performance of the HCPPEFT algorithm outperforms the other algorithms.

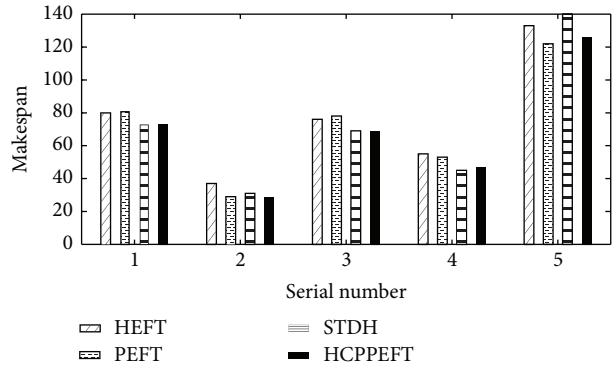

Figure 9: Makeapan with HCPPEFT, STDH, PEFT, and HEFT for the sets of task graphs.

5. Conclusion

In this paper, we have proposed a synthesized task scheduling algorithm for heterogeneous computing systems called HCPPEFT. This new algorithm is a two-phase algorithm that combines mechanisms of list-scheduling-based, duplication-based, and look-ahead-based algorithms. Therefore, the HCPPEFT algorithm provides a more efficient way to schedule general task graphs. In the task prioritizing phase, three levels of priority are proposed to choose task. The method of constructing task scheduling queue not only takes account of critical tasks but also takes account of the importance of parent tasks. In the resource selection phase, the duplication of parent tasks is to reduce communication costs. Forecasting the impact of an assignment for all children of the current task is to select a best resource. The effective performance of the new algorithm is compared to three of the best existing scheduling algorithms: HEFT, PEFT, and STDH algorithms. The comparative study is based on randomly generated task graph and the sets of task graphs. The HCPPEFT algorithm outperforms the other algorithms in terms of average SLR, speedup, and efficiency.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Foundation items (NFS of China: no. 61363031, GuangXiSci. and Tech. Development Foundation: no. GKG1211-80172C, GuilinSci. and Tech. Development Foundation: no. 20120104-13) for supporting our work.

References

[1] R. C. Corrêa, A. Ferreira, and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 8, pp. 825–837, 1999.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, NY, USA, 1979.

[3] E. G. Coffman, *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, USA, 1976.

[4] J. D. Ullman, "*NP*-complete scheduling problems," *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, 1975.

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

[6] X. Wang, H. Huang, and S. Deng, "List scheduling algorithm for static task with precedence constraints for cyber-physical systems," *Acta Automatica Sinica*, vol. 38, no. 11, pp. 1870–1879, 2012.

[7] E. U. Munir, S. Mohsin, A. Hussain, M. W. Nisar, and S. Ali, "SDBATS: a novel algorithm for task scheduling in heterogeneous computing systems," in *Proceedings of the IEEE 27th*

International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW '13), vol. 43, pp. 43–53, Cambridge, Mass, USA, May 2013.

[8] Y. Kwok and I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, 1996.

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

[10] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 399–409, 2008.

[11] J. Y. Zhu and D. Xiao, "Path priority-based heuristic task scheduling algorithm for cloud computing," *Computer Engineering and Design*, vol. 34, no. 10, pp. 3511–3515, 2013.

[12] Z. Lan and S. X. Sun, "Scheduling algorithm based on critical tasks in heterogeneous environments," *Journal of Systems Engineering and Electronics*, vol. 19, no. 2, pp. 398–405, 2008.

[13] X. Meng and W. Liu, "A DAG scheduling algorithm based on selected duplication of precedent tasks," *Journal of Computer-Aided Design and Computer Graphics*, vol. 22, no. 6, pp. 1056–1062, 2010.

[14] X. Y. Tang, K. L. Li, G. P. Liao, and R. F. Li, "List scheduling with duplication for heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 323–329, 2010.

[15] M. Hosseinzadeh and H. S. Shahhoseini, "An effective duplication-based task-scheduling algorithm for heterogeneous systems," *Simulation*, vol. 87, no. 12, pp. 1067–1080, 2011.

[16] J. Mei, K. Li, and K. Li, "A resource-aware scheduling algorithm with reduced task duplication on heterogeneous computing systems," *The Journal of Super Computing*, vol. 68, no. 3, pp. 1347–1377, 2014.

[17] J. Mei and K. Li, "Energy-aware scheduling algorithm with duplication on heterogenous computing systems," in *Proceedings of the 13th ACM/IEEE International Conference on Grid Computing (GRID '12)*, pp. 122–129, September 2012.

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

[19] D. Bozdag, F. Ozguner, and U. V. Catalyurek, "Compaction of schedules and a two-stage approach for duplication-based DAG scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 857–871, 2009.

[20] K. S. Shin, M. J. Cha, M. S. Jang, J. Jung, W. Yoon, and S. Choi, "Task scheduling algorithm using minimized duplications in homogeneous systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1146–1156, 2008.

[21] S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 6, pp. 533–544, 2003.

[22] T. Hagras and J. Janeček, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," *Parallel Computing*, vol. 31, no. 7, pp. 653–670, 2005.

[23] C. Boeres, J. V. Filho, and V. E. F. Rebello, "A cluster-based strategy for scheduling task on heterogeneous processors," in *Proceedings of the SBAC-PAD 16th Symposium on Computer Architecture and High Performance Computing*, vol. 214, pp. 27–29, 2004.

[24] J. C. Liou and M. A. Palis, "An efficient task clustering heuristic for scheduling dags on multiprocessors," in *Workshop on Resource Management, Symposium on Parallel and Distributed Processing*, 1996.

[25] F. Fu, Y. Bai, X. Hu, J. Wang, M. Yu, and J. Zhan, "An objective-flexible clustering algorithm for task mapping and scheduling on cluster-based NoC," in *Proceedings of the Laser Physics and Laser Technologies (RCSLPLT) and 2010 Academic Symposium on Optoelectronics Technology (ASOT), 2010 10th Russian-Chinese Symposium*, pp. 369–373, August 2010.

[26] B. Cirou and E. Jeannot, "Triplet: a clustering scheduling algorithm for heterogeneous systems," in *International Conference on Parallel Processing Workshops*, pp. 231–236, 2001.

[27] C. Boeres, J. V. Filho, and V. E. F. Rebello, "A cluster-based strategy for scheduling task on heterogeneous processors," in *Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (9SBAC-PAD '04)*, pp. 214–221, October 2004.

[28] A. H. Liang, L. M. Xiao, Y. N. Li, Z. Z. Zhang, and L. Ruan, "Energy aware scheduling for precedence constrained parallel tasks in a power-scalable cluster," in *Proceedings of the IEEE International Conference on High Performance Computing and Communications*, pp. 1016–1021, Zhangjiajie, China, 2013.

[29] L. Wang, S. U. Khan, D. Chen et al., "Energy-aware parallel task scheduling in a cluster," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1661–1670, 2013.

[30] H. Zhao, W. Jiang, and X. H. Li, "Synthesized heuristic task scheduling algorithm for heterogeneous system," *Journal of Computer Application*, vol. 30, no. 5, pp. 1316–1320, 2010.

[31] G. Q. Xie, R. F. Li, X. G. Xiao, and Y. K. Chen, "A high-performance DAG task scheduling algorithm for heterogeneous networked embedded systems," in *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications (AINA '14)*, vol. 1011, pp. 13–16, 2014.

[32] J. M. Li, D. W. Sun, and Y. X. Wu, "Global comparatively optimum static task scheduling algorithm," *Application Research of Computers*, vol. 31, no. 4, pp. 1027–1030, 2014.