Received: 30 September 2015 | Revised: 7 July 2016 | Accepted: 25 July 2016
DOI 10.1002/cpe.3944

WILEY

RESEARCH ARTICLE

A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table

Naqin Zhou | Deyu Qi | Xinyang Wang | Zhishuo Zheng | Weiwei Lin

School of Computer Science &Engineering, South China University of Technology, Guangzhou 510006Guangdong, China

Correspondence
Naqin Zhou, School of Computer Science &Engineering, South China University of Technology, Guangzhou 510006, Guangdong, China.
Email: zhou.naqin@mail.scut.edu.cn

Summary
This paper presents a novel list-based scheduling algorithm called Improved Predict Earliest Finish Time for static task scheduling in a heterogeneous computing environment. The algorithm calculates the task priority with a pessimistic cost table, implements the feature prediction with a critical node cost table, and assigns the best processor for the node that has at least 1 immediate successor as the critical node, thereby effectively reducing the schedule makespan without increasing the algorithm time complexity. Experiments regarding aspects of randomly generated graphs and real-world application graphs are performed, and comparisons are made based on the scheduling length ratio, robustness, and frequency of the best result. The results demonstrate that the Improved Predict Earliest Finish Time algorithm outperforms the Predict Earliest Finish Time and Heterogeneous Earliest Finish Time algorithms in terms of the schedule length ratio, frequency of the best result, and robustness while maintaining the same time complexity.

KEYWORDS
DAG scheduling, heterogeneous systems, list scheduling, static scheduling, task graphs

1 | INTRODUCTION

A heterogeneous computing system is a computation platform composed of different computing resources that are interconnected by a high-speed network. The emergence of heterogeneous computing systems has made it possible to execute complex computationally intensive applications by using idle computing resources located throughout the network. Efficient application scheduling is critical for achieving high performance in heterogeneous computing systems.[1] The primary objectives of a scheduling mechanism are to map tasks onto processors and then to order the execution of the tasks such that the precedence requirements are satisfied and the minimum overall completion time is achieved.[1–3] However, in most situations, the problem of scheduling tasks according to the required precedence relationship has been proven to be NP-complete.[4,5] Thus, previous studies have mainly focused on seeking suboptimal scheduling solutions with lower complexity, with heuristic methods being widely used. Many heuristic methods have been proposed in recent years.[1,6–22] These methods can be classified into task-clustering scheduling algorithms, task-duplication scheduling algorithms, and list scheduling algorithms.

Task-clustering scheduling algorithms are generally divided into 2 phases: (1) map all of the tasks of a given task graph to clusters with unlimited node numbers and assign the high-communication tasks to the same cluster; and (2) implement a final schedule on the mapped clusters. Classic task clustering algorithms[6–8] are mainly applicable to homogeneous systems. Algorithms for heterogeneous systems were proposed in 2 studies[9,10]; however, these algorithms have limitations in systems with a high degree of heterogeneity.[11].

Task-duplication scheduling algorithms minimize the communication cost among tasks by duplicating a task to different processors. Classical examples of task duplication heuristics were presented in 4 studies.[12–15] Although task duplication scheduling algorithms can obtain shorter scheduling lengths, they often require higher time complexity and a large amount of processor resources.

The main purpose of list scheduling algorithms is to construct an ordered list by assigning a priority to each task in a given directed acyclic graph (DAG), assigning the first task node from this ordered list to the processor that minimizes the predefined cost function, and then repeating this process until all of the tasks in the list are scheduled. Classical examples of list scheduling algorithms were presented in several studies.[1,11,16–22] Compared with the other types of algorithms, list scheduling algorithms have a lower time complexity and yield superior scheduling results; thus, they have always become the primary option for task scheduling.

In 1 study,[23] the authors compared 20 types of heuristics and found that the list scheduling algorithm Heterogeneous Earliest Finish

Time (HEFT)[16,18] performs best in terms of robustness and schedule length. The authors of 1 study[11] proposed another list scheduling algorithm, Predict Earliest Finish Time (PEFT), in 2014; this algorithm outperforms the HEFT algorithm in terms of the scheduling length ratio (SLR) and frequency of the best result while maintaining the time complexity.

In this paper, we aim to develop a new list scheduling algorithm for full connection heterogeneous processor architectures that outperforms the HEFT and PEFT algorithms in terms of the schedule length ratio, robustness, and frequency of the best results while maintaining the same time complexity.

The remainder of this paper is organized as follows: the task scheduling problem is introduced in Section 2; a detailed comparison of related works is presented in Section 3; a description of the Improved PEFT (IPEFT) algorithm is provided in Section 4; and the experimental results and conclusions are provided in Sections 5 and 6, respectively.

2 | TASK SCHEDULING PROBLEM

This paper focuses on the static scheduling of a single application in a heterogeneous computing environment. The modeling of heterogeneous computing environments and applications and their simplifications are consistent with those found in several studies.[11,18,21,24].

Assume that the heterogeneous computing environment is a set P, which is composed of p heterogeneous processors that are fully interconnected. In each processor, the executions and communications of tasks can be performed concurrently, and task execution is assumed to be nonpreemptive.

As shown in Figure 1, an application is represented by a DAG G. Let $G = (V, E)$, where V is a set of v tasks and E is a set of directed edges among tasks. (The terms "task" and "node" are used interchangeably throughout the paper). Each edge $e_{i,j} \in E$ represents the task dependence constraint, ie, task v_i must finish its execution and transfer the resulting data to solve the data dependency before task v_j starts. The weight of each edge $e_{i,j} \in E$ represents the communication cost between task v_i and v_j, denoted by $c_{i,j}$. Because $c_{i,j}$ can be computed only after defining where tasks v_i and v_j will be executed, the average communication costs are used to label the edges.[11,18] The average communication cost $\overline{c_{i,j}}$ of an edge $e_{i,j}$ is defined as

$$\overline{c_{i,j}} = \overline{L} + \frac{data_{i,j}}{\overline{B}}, \tag{1}$$

where $\overline{L}$ is the average latency of all processors, $\overline{B}$ is the average bandwidth among processors, and $data_{i,j}$ is the amount of data elements that v_i sends to v_j. If v_i and v_j are assigned to the same processor, $c_{i,j}$ becomes 0 because the intraprocessor communication cost is negligible compared with the interprocessor communication cost. In this study, the latency is assumed to be negligible, and the bandwidth is assumed to be 1.0.[1,21] Hence, the average communication cost and amount of data to be transferred will be identical.

A matrix W is generally applied as a supplement to the DAG. W is a $v \times p$ computation cost matrix, where v represents the task number, p represents the processor number, and $w_{i,j}$ denotes the estimated execution time for task v_i in processor p_j. Therefore, the average execution time $\overline{w_i}$ of task v_i is defined as

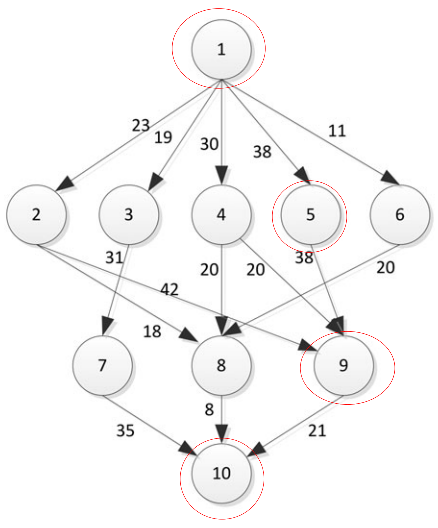
$$\overline{w_i} = \sum_{j=1}^{p} w_{i,j}/p. \tag{2}$$

The following will define some common properties for task scheduling, and these definitions will be referred to in the forthcoming sections.

Definition 1. [11] *Let pred*(v_i) be the set of immediate predecessors of task v_i in a given DAG; if *pred*(v_i) = φ, v_i is called an entry task, denoted as v_{entry}. Let *succ*(n_i) be the set of immediate successors of task v_i in a given DAG; if *succ*(v_i) = φ, v_i is called an exit task, denoted as v_{exit}. For convenience, if there is more than 1 entry task (exit task) in a DAG, a dummy entry task (exit task) with 0 weight and 0 communication can be added to the graph.

Definition 2. [11,18] *Makespan (or schedule length) represents the finish time of the last task in the scheduled DAG, ie,*

$$makespan = \max\{AFT(v_{exit})\}, \tag{3}$$



Computation Costs

Task	P1	P2	P3
1	17	12	27
2	24	19	28
3	18	19	9
4	8	25	16
5	10	44	44
6	35	16	32
7	45	11	6
8	43	5	31
9	22	16	8
10	40	14	34

FIGURE 1 Application directed acyclic graph and computation time matrix of the tasks in each processor for a 3-processor machine

where $AFT(v_{exit})$ is the actual finish time of the exit task. When there is more than 1 exit task, without adding any redundant tasks, the makespan is the maximum actual finish time of all of the exit tasks.

Definition 3. [18] *The critical path (CP)* is the longest path from the entry task to the exit task in the DAG. The lower bound of the schedule length is the minimum critical-path length (CP_{MIN}), which is accumulated by the minimum computational costs of each task in the critical path.

Definition 4. [11,18] $EST(v_i, p_j)$ represents the earliest start time of task v_i on processor p_j, ie,

$$EST(v_i, p_j) = \max\left\{T_{\text{Available}}(p_j), \max_{v_m \in pred(v_i)}\{AFT(v_m) + c_{m,i}\}\right\}, \quad (4)$$

where $T_{\text{Available}}(p_j)$ is the earliest available time for processor p_j and $\max_{v_m \in pred(v_i)}\{AFT(v_m) + c_{m,i}\}$ represents the arrival time of all input data for task v_i on processor p_j. If task v_m is assigned to p_j, $c_{m,i} = 0$. For all entry tasks, $EST(v_{entry}, p_j) = 0$.

Definition 5. [11,18] $EFT(v_i, p_i)$ represents the earliest finish time of task v_i on processor p_j, which equals the sum of the earliest start time and the computation cost for task v_i on processor p_j, ie,

$$EFT(v_i, p_j) = EST(v_i, p_j) + w_{i,j}. \quad (5)$$

Definition 6. [19] $AEST(v_i)$ represents the average earliest start time of task v_i, which can be calculated recursively by traversing the DAG downward starting from the entry task v_{entry}, ie,

$$AEST(v_i) = \max_{v_m \in pred(v_i)}\{AEST(v_m) + \overline{w_m} + \overline{c_{m,i}}\}, \quad (6)$$

where $pred(v_i)$ is the set of immediate predecessors of task v_i and $AEST(v_{entry}) = 0$.

Definition 7. [19] $ALST(v_i)$ represents the average latest start time of task v_i, which can be calculated recursively by traversing the DAG upward starting from the exit task v_{exit}, ie,

$$ALST(v_i) = \min_{v_m \in succ(v_i)}\{ALST(v_m) - \overline{c_{i,m}}\} - \overline{w_i}, \quad (7)$$

where $succ(v_i)$ is the set immediate successors of task v_i, and $ALST(v_{exit}) = AEST(v_{exit})$.

Definition 8. [19] *The critical node (CN) is a node where the AEST and ALST are equal. Let CNs be the set of CNs in a given DAG, ie,*

$$\forall v_i \in CNs, AEST(v_i) = ALST(v_i). \quad (8)$$

3 | RELATED WORK

In the past few years, task scheduling research has mainly focused on finding suboptimal scheduling algorithms with lower complexity, namely, the heuristic methods. Compared with other algorithm categories, list-based scheduling has lower time complexity and better scheduling results and is thus widely accepted and studied.[1].

The HEFT[18] and Critical Path On a Processor (CPOP)[18] are both list-based scheduling methods that were proposed when there were few research results from heterogeneous environments. The HEFT can be divided into 2 phases: a task prioritizing phase and processor selection phase. In the first phase, it defines an upward rank for each task priority, calculates the upward rank of a computation task by a recursive procedure, and then generates a task list in order of descending upward rank value. In the second phase, it selects processors for tasks according to the sequence in the task priority list, assigning tasks to the processor that can minimize the task completion time. The time complexity of HEFT is $O(v^2 \cdot p)$. The HEFT is often selected for comparison with other algorithms owing to its desirable performance in terms of complexity and scheduling. Similar to HEFT, there are also 2 phases in CPOP, but CPOP uses different attributes for setting the task priorities and a different policy for determining the "best" processor for each selected task. In the first phase, CPOP adopts the sum of the upward and downward ranks as each task priority. In the second phase, if the selected task is on the critical path, then it is scheduled on the critical-path processor; otherwise, the task is assigned a processor, as in HEFT. The CPOP has a similar time complexity as HEFT, but the scheduling result is worse.[18].

Fast Load Balancing[17] reduces the time complexity of HEFT to $O(v \cdot (\log p + \log v) + v^2)$. However, the scheduling result from Fast Load Balancing is worse than that of HEFT for irregular task graphs and more heterogeneous processors.[11].

Heterogeneous Critical Parent Trees (HCPT)[19] uses a new mechanism to construct the scheduling list rather than assigning priorities to the tasks. The HCPT achieves superior scheduling results than CPOP while maintaining the same time complexity. However, according to 1 study,[11] the frequency of the best results in HCPT is lower than that in HEFT.

High-Performance Task Scheduling (HPS)[20] and Performance Effective Task Scheduling (PETS)[1] are designed as 3 phases: level sorting, task prioritization, and processor selection. The time complexity for both algorithms is $O(v^2 \cdot (p + \log v))$. The largest distinction between the 2 is the use of different properties for setting the priorities. The Longest Dynamic Critical Path (LDCP)[21] builds and maintains a DAG for each processor, ie, DAGP. Thus, all of the DAGPs must be updated when a task is scheduled to a processor. The complexity of LDCP is $O(v^3 \cdot p)$. The Lookahead algorithm[22] improves upon the processor selection strategy of HEFT. The improved strategy predicts and considers the impact of the schedule decision on the allocation of current task child nodes, not simply the completion time of the current task. The complexity of Lookahead is $O(v^4 \cdot p)$. Although the scheduling results of HPS, PETS, LDCP, and Lookahead are superior to that of HEFT, they have higher time complexities.

Similar to the HEFT algorithm, the PEFT[11] algorithm has 2 phases that are based on an optimistic cost table (OCT). The OCT represents a matrix in which the rows indicate the number of tasks and the columns indicate the number of processors, and when task v_i chooses processor

p_k, each $OCT(v_i, p_k)$ is the maximum value of the shortest paths from the children tasks of v_i to the exit task. The OCT value of task v_i on processor p_k can be recursively defined as follows:

$$OCT(v_i, p_k) = \max_{v_j \in succ(t_i)} \left[\min_{p_m \in P} \left\{ OCT(v_j, p_m) + w_{j,m} + \overline{c_{i,j}} \right\} \right]. \tag{9}$$

Predict Earliest Finish Time outperforms HEFT, HCPT, HPS, and PETS in terms of the SLR and the frequency of the best results while maintaining a similar time complexity. However, the algorithm does not account for critical tasks or the parent critical task that determines the DAG scheduling time.

4 | THE IPEFT ALGORITHM

In this section, a new list-based scheduling algorithm for a bounded number of heterogeneous processors called IPEFT is introduced.

Improved PEFT has 2 main phases: task prioritization and processor selection. In the first phase, each task is assigned a priority based on the length of the longest path from a current task to an exit task. The length of the longest path from a current task to an exit task is based on the current task average execution time and pessimistic cost table (PCT). In the second phase, the selection of a processor for the current task v_i is based on $CNP(v_i)$ and the CN cost table (CNCT). $CNP(v_i)$ indicates whether v_i is the parent of a CN. If v_i is not a CN and has an immediate successor as the CN, then $CNP(v_i) = true$; otherwise, $CNP(v_i) = false$, ie,

$$CNP(v_i) = \begin{cases} \text{true}, & \text{if } \exists v_j \in succ(v_i),\ v_j \text{ is } CN \text{ and } v_i \text{ is not } CN \\ \text{false}, & \text{otherwise} \end{cases}. \tag{10}$$

The algorithm attempts to assign parents of a CN onto the processors that result in the earliest finish time such that it can advance the earliest start time of the CN and reduce the schedule length. When the algorithm assigns the processor for the current task, the algorithm considers information regarding not only the EFT of the current task but also the impact of the chosen processor on the path length from the current task's key child nodes to the exit nodes.

4.1 | Pessimistic cost table and critical node cost table

The PCT is a matrix in which the rows represent the number of tasks and the columns represent the number of processors, where each element $PCT(v_i, p_k)$ is the maximum value of the longest path from the immediate successors of task v_i to the exit task when task v_i is selected to processor p_k. The PCT value is recursively defined by the following equation:

$$PCT(v_i, p_k) = \max_{v_j \in succ(v_i)} \left[\max_{p_m \in P} \left\{ PCT(v_j, p_m) + w_{j,m} + \overline{c_{i,j}} \right\} \right], \overline{c_{i,j}} = 0 \text{ if } p_m = p_k. \tag{11}$$

The average communication cost and execution cost for each processor are as explained above. For the exit task, $PCT(v_{exit}, p_k) = 0$ for all processors $p_k \in P$.

The CNCT is a matrix in which the rows represent the number of tasks and the columns represent the number of processors, where each $CNCT(v_i, p_k)$ is the maximum value of the shortest path from the critical immediate successors of task v_i to the exit task if v_i has at least 1 immediate successor as a CN; otherwise, it is the maximum value of the shortest path from the immediate successors of task v_i to the exit task. The CNCT is recursively calculated by the following equation:

$$CNCT(v_i, p_k) = \begin{cases} \max\limits_{(v_j \in succ(v_i)) \wedge (v_j \in CNs)} \left[\min\limits_{p_m \in P} \left\{ CNCT(v_j, p_m) + w_{j,m} + \overline{c_{i,j}} \right\} \right] & \text{if} \exists v_j \in succ(v_i) \wedge v_j \text{ is a CN} \\ \max\limits_{v_j \in succ(v_i)} \left[\min\limits_{p_m \in P} \left\{ CNCT(v_j, p_m) + w_{j,m} + \overline{c_{i,j}} \right\} \right] & \text{otherwise.} \end{cases} \tag{12}$$

The average communication cost and execution cost for each processor are as stated above.

The values of the CNCT in Figure 1 are shown in Table 1.

TABLE 1 CNCT for the DAG of Figure 1

Task	P1	P2	P3
1	67	74	86
2	57	30	42
3	56	25	40
4	50	30	42
5	57	30	42
6	39	19	39
7	40	14	34
8	22	14	22
9	35	14	34
10	0	0	0

Abbreviations: CNCT, critical node cost table; DAG, directed acyclic graph.

4.2 | Phases of IPEFT

4.2.1 | Task prioritization phase

The task priority processing phase calculates and assigns a priority for each task. To assign a priority for task v_i, the length of the longest path from v_i to the exit task is computed, which can be calculated by the following equation:

$$rank_{PCT}(v_i) = \frac{\sum_{k=1}^{P} PCT(v_i, p_k)}{p} + \overline{w_i}. \tag{13}$$

The task with the highest $rank_{PCT}$ value receives the highest priority, followed by the task with the next highest $rank_{PCT}$ value, and so on. Thus, the $rank_{PCT}$ values based on the PCT that may lead to the worst scheduling results are preferentially scheduled, which can improve the scheduling performance.

A similar strategy with the PEFT algorithm has be used to calculate a priority for each task, which is calculated based on CNCT, namely,

$$rank_{CNCT}(v_i) = \frac{\sum_{k=1}^{P} CNCT(v_i, p_k)}{p}. \tag{14}$$

For the tests with random graphs in Section 5, the average SLR is 2.3% better than that of PEFT algorithm, when using $rank_{CNCT}$. But the average SLR is 3.6% worse than that using $rank_{PCT}$. Therefore, using $rank_{PCT}$ increases the performance of results.

The values of the PCT, AEST, ALST, $rank_{PCT}$, and CNP for the DAG in Figure 1 are shown in Table 2.

4.2.2 | Processor selection phase

To select the processors for a task, the earliest finish time of the task on each processor is first calculated. The insertion-based policy is applied to compute *EFT*, and the possibility of inserting tasks in the earliest idle time slot between 2 scheduled tasks on the identical processor should be considered. The idle time slot should be at least capable of handling the computation cost of the task to be scheduled, and scheduling on this idle time slot should preserve precedence constraints.

Then, the CNCT-based EFT (EFT_{CNCT}) for the current task is calculated. If the current task v_i is the parent of a CN, the value of EFT_{CNCT} is equal to *EFT*; otherwise, the value is the summation of *EFT* and *CNCT*, ie,

$$EFT_{CNCT}(v_i, p_j) = \begin{cases} EFT(v_i, p_j) & , \text{ if } CNP(v_i) = \text{true} \\ EFT(v_i, p_j) + CNCT(v_i, p_j), & \text{otherwise} \end{cases}. \tag{15}$$

Finally, the processor with the minimum EFT_{CNCT} value is selected for the current task.

This processor selection policy shortens the schedule length in 2 ways. First, if the current task has an immediate successor that is a critical task (namely, the current task is the parent of a critical task), the current task is assigned to the processor that achieves the EFT, which may advance the earliest start time of critical tasks to reduce the schedule length. Second, if the current task is not the parent of a critical task, the task is assigned to a processor based on not only the EFT of the current task but also the predicted impact of the chosen processor on the path length from the current task's key child nodes to the exit nodes. Therefore, although the finish time of the chosen processor is not always the earliest, this policy ensures a shorter finish time for critical tasks, which reduces the schedule length.

4.3 | Detailed description of the IPEFT algorithm

The pseudocode of IPEFT is shown in Figure 2.

In the algorithm, AEST, ALST, CNCT, PCT, CNP, and $rank_{PCT}$ values of all of the tasks are first calculated by lines 1 to 3. Second, an empty ready list is created, and the entry task is placed on top of the list by line 4. Third, the algorithm schedules the task with the highest $rank_{PCT}$ in every iterative step of the loop from lines 6 to 16. The EFT_{CNCT} values of this task on all of the processors are calculated by the IPEFT algorithm. The minimum $EFT_{CNCT}(v_i, p_k)$ is obtained by p_k in line 15, and the processor is selected to execute task v_i.

The time complexities of each step in the algorithm are as follows:

1. Calculating the AEST and ALST for all tasks: $O(e + v)$.[19]

TABLE 2 PCT, AEST, ALST, $rank_{PCT}$, and CNP for the DAG of Figure 1

Task	P1	P2	P3	AEST	ALST	$rank_{PCT}$	CNP
1	197	197	197	0.0	0.0	215.7	FALSE
2	119	119	119	41.7	61.7	142.7	TRUE
3	117	145	145	37.7	61.7	151.0	FALSE
4	99	105	105	48.7	91.0	119.3	TRUE
5	115	115	115	56.7	56.7	147.7	FALSE
6	99	105	105	29.7	81.7	130.3	FALSE
7	69	75	75	84.0	108.0	93.7	TRUE
8	42	48	48	85.0	129.3	72.3	TRUE
9	55	61	61	127.3	127.3	74.3	FALSE
10	0	0	0	163.7	163.7	29.3	FALSE

Abbreviations: AEST, average earliest start time; ALST, average latest start time; DAG, directed acyclic graph; PCT, pessimistic cost table.

```
1.  Traverse the graph downward and compute $AEST$ for each node;
2.  Traverse the graph upward and compute $ALST$ for each node;
3.  Compute the CNCT, PCT, CNP and $rank_{pct}$ for all tasks
4.  Create the Empty List ready-list and set $v_{entry}$ as the initial task
5.  While ready-list is not empty do
6.      $v_i \leftarrow$ the task with highest $rank_{PCT}$ from ready-list
7.      For each processor $p_k$ in the processor set $P$ do
8.          Compute the $EFT(v_i, p_k)$ value using the insertion-based scheduling policy
9.          If $CNP(v_i) = true$ then
10.             Compute $EFT_{CNCT}(v_i, p_k) = EFT(v_i, p_k)$
11.         else
12.             Compute $EFT_{CNCT}(v_i, p_k) = EFT(v_i, p_k) + CNCT(v_i, p_k)$
13.         End if
14.     End for
15.     Assign task $v_i$ to the processor $p_k$ that minimizes the $EFT_{CNCT}$ of task $v_i$
16.     Update ready-list
17. End while
```

FIGURE 2 Improved Predict Earliest Finish Time algorithm

2. Calculating the CNCT and PCT, similar to the OCT, for all tasks: $O(p \cdot (v+e))$.
3. Calculating the CNP for all tasks: $O(e+v)$.
4. Assigning processors for all tasks: $O(v^2 \cdot p)$.

Thus, the general time complexity of IPEFT is $O(v^2 \cdot p)$, which is equivalent to that of PEFT and HEFT.

The procedure for selecting the processors for all tasks in Figure 1 is shown in Table 3.

Figure 3 shows the scheduling result of the DAG in Figure 1 using the IPEFT, PEFT, and HEFT algorithms. The scheduling length of IPEFT is 116, which is shorter than that of PEFT and HEFT (126 and 143, respectively).

5 | EXPERIMENTAL RESULTS AND DISCUSSION

In this section, a comparison among IPEFT, HEFT, and PEFT will be conducted from aspects of randomly generated DAGs and real-world application graphs. The primary comparison metrics are presented first.

5.1 | Comparison metrics

The comparisons of the algorithms are based on the following 3 metrics:

1. Scheduling length ratio

The SLR[11,18] is the normalization of the schedule length, ie,

$$SLR = \frac{makespan(solution)}{\sum_{v_i \in CP_{MIN}} \min_{p_j \in P}(w_{i,j})}, \tag{16}$$

where the denominator is the minimum computation cost of the critical-path tasks (CP_{MIN}). A lower SLR indicates a superior algorithm.

2. Number of occurrences of better quality of schedules (NOBQS)

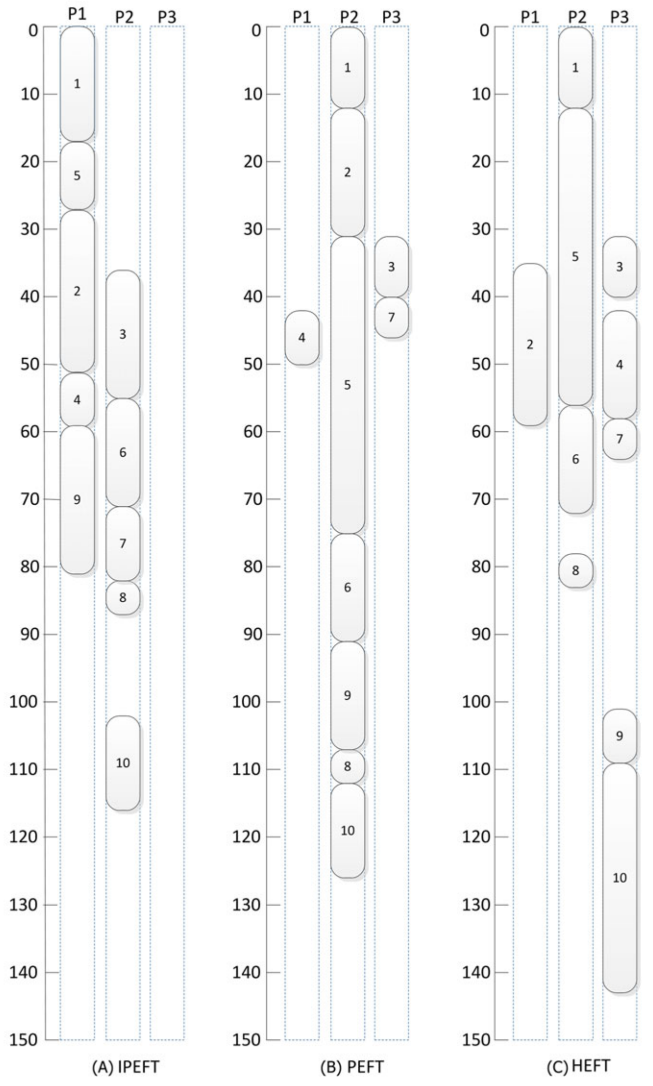


FIGURE 3 Schedules of the sample task graph in Figure 1 with (A) Improved Predict Earliest Finish Time (makespan = 116), (B) Predict Earliest Finish Time (makespan = 126), and (C) Heterogeneous Earliest Finish Time (makespan = 143)

TABLE 3 Schedule produced by the IPEFT algorithm in each iteration

			EFT			EFT_{OCT}			EFT_{CNCT}			
Step	Ready List	Task Selected	P1	P2	P3	P1	P1	P3	P1	P1	P3	CPU Selected
1	1	1	17	**12**	27	89	**86**	113	**84**	86	113	P1
2	2, 3, 4, 5, 6	3	**35**	55	45	91	**80**	85	91	**80**	85	P2
3	2, 4, 5, 6, 7	5	**27**	99	99	**84**	129	141	**84**	129	141	P1
4	2, 4, 6, 7	2	**51**	74	68	108	**104**	110	**51**	74	68	P1
5	4, 6, 7	6	86	71	**60**	125	**90**	99	125	**90**	99	P2
6	4, 7	4	**59**	96	63	109	126	**105**	**59**	96	63	P1
7	7, 8, 9	7	131	**82**	92	171	**96**	126	131	**82**	92	P2
8	8, 9	9	**81**	109	101	**116**	123	135	**116**	123	135	P1
9	8	8	134	**87**	122	156	**101**	144	134	**87**	122	P2
10	10	10	157	**116**	151	157	**116**	151	157	**116**	151	P2

Abbreviations: CPU, central processing unit; EFT, Earliest Finish Time; IPEFT, Improved Predict Earliest Finish Time.

The bold emphasis indicates that the processor is assigned based on EFT, EFT_{oct} and EFT_{CNCT} for the current task in each iteration, respectively.

The NOBQS is the percentages of better, equal, and worse schedule lengths generated by 1 algorithm compared with another algorithm.

3. Slack

Slack[11,25] is a measure of the robustness of the schedules produced by an algorithm compared with the uncertainty in the task processing time and can be expressed as

$$Slack = \left[\sum_{v_i \in V} M - b_{\text{level}}(v_i) - t_{\text{level}}(v_i) \right] / n, \tag{17}$$

where M is the makespan of the DAG, n is the task number, b_{level} is the length of the longest path from task v_i to the exit task, and t_{level} denotes the length of the longest path from the entry task to task v_i (not including v_i).

5.2 | Randomly generated application graphs

In this section, the generation method of the random graphs applied in the experiments is introduced, and the performances of the IPEFT, PEFT, and HEFT algorithms are compared.

5.2.1 | Random graph generator

A composite task graph generator[11,26] is adopted as the random graph generator in this paper. Its primary parameters are as follows:

- n: the number of nodes in the DAG, namely, the number of task applications.
- fat: affects the height and width of the DAG; the width of each level in the DAG is defined by a uniform distribution that equals $fat.\sqrt{n}$; the number of levels (or the height) is calculated until n tasks are defined in the DAG; and the width of the DAG is the maximum number of tasks that can be executed synchronously.
- density: determines the number of edges between the 2 levels of the DAG.
- regularity: determines the uniformity of task numbers in every level; a lower value indicates that there is a dissimilarity of task numbers among levels.
- jump: distance of an edge from level l to level $l + jump$; a jump of 1 indicates a direct connection between 2 consecutive levels.

Different DAG structures are established by setting different parameter values to the generator. The following arguments are used to calculate the computation and communication costs.

- CCR (ratio of communication to computation): the rate between the sum of the edge weights and the sum of the node weights in a DAG;
- β (range of computing cost percentage on the processors): the basic heterogeneous factor of processor speed. A high β value indicates a higher degree of heterogeneity and different computation costs among processors, whereas a low value indicates that the computation costs for a given task are nearly equal among processors[11,18]; the average computing cost $\overline{w_i}$ of task n_i is randomly selected in the range of the uniform distribution $[0, 2\times\overline{w_{\text{DAG}}}]$, where $\overline{w_{\text{DAG}}}$ is the average computing cost of a given DAG that is randomly generated. The computing cost of task n_i for each processor p_j is randomly set in the following range:

$$\overline{w_i}\times\left(1-\frac{\beta}{2}\right)\le w_{i,j}\le\overline{w_i}\times\left(1+\frac{\beta}{2}\right). \tag{18}$$

In the experiments, arguments below are used to generate random graphs.

$$n = [10,\ 20,\ 30,\ 40,\ 50,\ 60,\ 70,\ 80,\ 90,\ 100,\ 150,\ 200,\ 250,\ 300,\ 350,\ 400]$$
$$fat = [0.1,\ 0.4,\ 0.8]$$
$$density = [0.2,\ 0.8]$$
$$regularity = [0.2,\ 0.8]$$
$$jump = [1,\ 2,\ 4]$$
$$CCR = [0.1,\ 0.25,\ 0.5,\ 0.8,\ 1,\ 2,\ 5,\ 8,\ 10,\ 15,\ 20,\ 25,\ 30]$$
$$\beta = [0.1,\ 0.2,\ 0.5,\ 0.75,\ 1,\ 2]$$
$$Processors = [4,\ 8,\ 16,\ 32]$$

Using these argument combinations, 179 712 DAGs are generated, and all of the DAGs are assigned to different edge and node weights to create 20 random graphs. The total number of DAGs is 3 594 240.

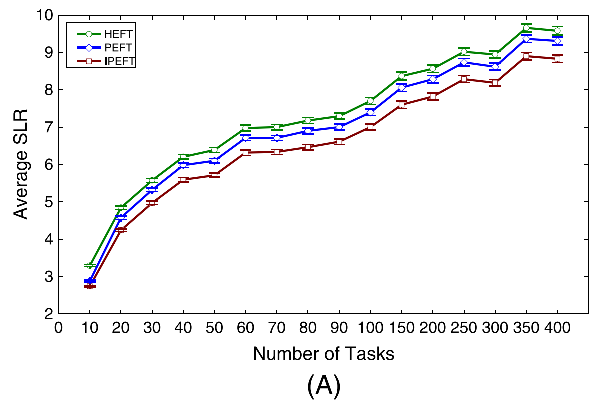
5.2.2 | Performance results

Figure 4 shows the average SLR and average slack for different numbers of tasks. Each data node is the average value from 224 640 experiments. The experimental results illustrate that the average SLR value of the IPEFT algorithm is lower than that of the PEFT algorithm. Compared with the HEFT algorithm, when the task quantity is 10, the average SLR of the IPEFT algorithm is 16.7% lower than that of the HEFT algorithm, and when the number of tasks reaches 100 and 400, the decrement percentages decrease to 9.1% and 7.8%, respectively. This implies that the effect of key child nodes on scheduling decisions for parent nodes decrease for an increasing number of tasks.

The average slack obtained by the IPEFT algorithm is smaller than that obtained by the PEFT and HEFT algorithms. Hence, for any number of nodes in our range, the IPEFT algorithm achieves better results than the IPEFT and HEFT algorithms in terms of the SLR and slack.

Figure 5A presents the average SLR of the algorithms for CCR values of $[0.1, 0.25, 0.5, 0.8, 1, 2, 5, 8, 10, 15, 20, 25, 30]$. When $CCR < 0.8$, the IPEFT algorithm obtains similar average SLRs as the PEFT algorithm. When $CCR \ge 0.8$, the IPEFT algorithm outperforms the PEFT algorithm. Compared with the HEFT algorithm, the IPEFT algorithm has similar average SLRs when $CCR < 0.5$ and has smaller average SLRs when $CCR \ge 0.5$.

Figure 5B presents the average SLR of the algorithms for heterogeneity values of $[0.1, 0.2, 0.5, 0.75, 1, 2]$. The IPEFT algorithm outperforms the other algorithms in terms of the average SLR for random task graphs with various values of heterogeneity.



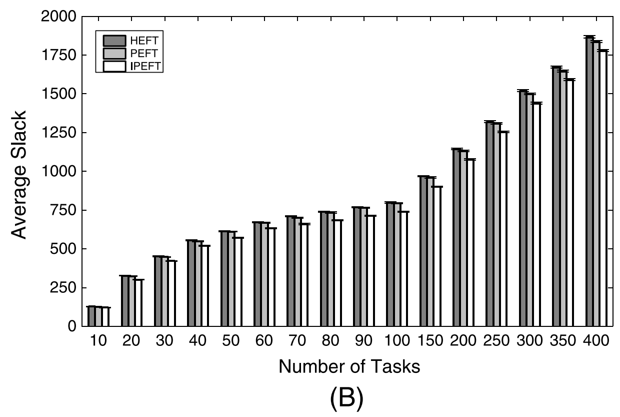


FIGURE 4 For randomly generated directed acyclic graphs: (A) average scheduling length ratio (SLR) for different numbers of tasks. The bars represent the 95% confidence intervals of the mean. B, Average slack for different numbers of tasks. The bars represent the 95% confidence intervals of the mean. HEFT indicates Heterogeneous Earliest Finish Time; IPEFT, Improved Predict Earliest Finish Time; PEFT, Predict Earliest Finish Time

Table 4 lists the percentage of better, equal, and worse scheduling lengths produced by IPEFT compared with the other algorithms. Compared with the HEFT and PEFT algorithms, the IPEFT algorithm achieves better scheduling in 80% and 61% of runs, worse schedules in 14% and 12% of runs, and equivalent schedules in 6% and 27% of runs, respectively.

TABLE 4 Pairwise schedule length comparison of the scheduling algorithms

		IPEFT (%)	PEFT (%)	HEFT (%)
IPEFT	Better	*	61	80
	Worse		12	14
	Equal		27	6
PEFT	Better	12	*	66
	Worse	61		32
	Equal	27		2
HEFT	Better	14	32	*
	Worse	80	66	
	Equal	6	2	

Abbreviations: HEFT, Heterogeneous Earliest Finish Time; IPEFT, Improved Predict Earliest Finish Time; PEFT, Predict Earliest Finish Time.

The asterisk indicates calculating the number of occurrences of better quality of schedules is no need.

5.3 | Real-world application graphs

In addition to randomly generated task graphs, application graphs of 3 real-world problems are also considered: Gaussian elimination,[18] Montage,[27] and molecular dynamics code.[18] In these graphs, the computation and communication cost-related arguments are set as follows:

- *CCR* = [0.1, 0.25, 0.5, 0.8, 1, 2, 5, 8, 10, 15, 20, 25, 30];
- β = [0.1, 0.2, 0.5, 0.75, 1, 2];
- *Processors* = [4, 8, 16, 32].

5.3.1 | Gaussian elimination

In Gaussian elimination, the task quantity is determined by matrix *m*, ie, the quantity $n = \frac{m^2+m+2}{2}$. In this experiment, *m* = [5, 10, 15, 20, 25, 30]. The average SLR for different values of *m* and different CCRs are shown in Figure 6A and B, respectively. The experimental results demonstrate that the IPEFT algorithm obtains smaller average SLRs than the HEFT and IPEFT algorithms for all matrix sizes. The IPEFT algorithm also

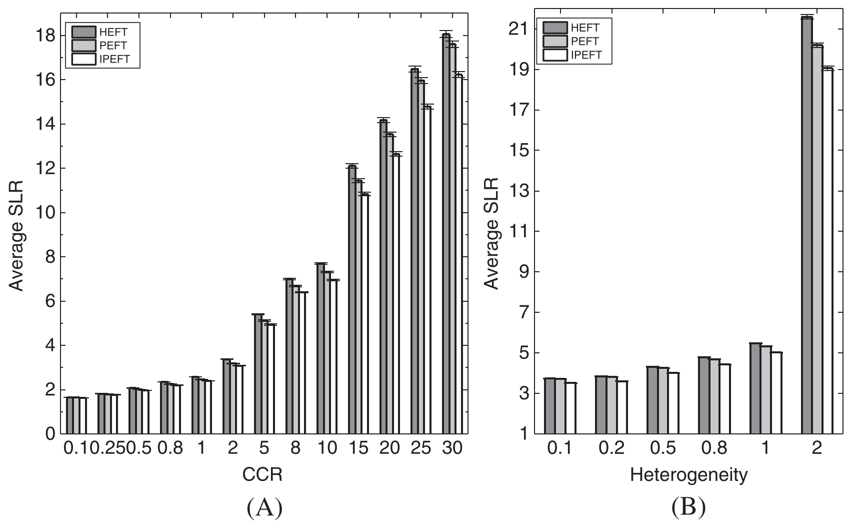


FIGURE 5 For randomly generated directed acyclic graphs: (A) average scheduling length ratio (SLR) for different ratios of communication to computation (CCRs). The bars represent the 95% confidence intervals of the mean. B, Average SLR for different heterogeneous values. The bars represent the 95% confidence intervals of the mean. HEFT indicates Heterogeneous Earliest Finish Time; IPEFT, Improved Predict Earliest Finish Time; PEFT, Predict Earliest Finish Time

achieves smaller average SLRs than the HEFT algorithm for all CCRs. Compared with the PEFT algorithm, the IPEFT algorithm has similar average SLRs when $CCR < 1$ and smaller average SLRs when $CCR \geq 1$.

5.3.2 | Montage

The second real-world application DAG is the Montage task DAG with a task number of 25. The average SLRs for different levels of heterogeneity and different CCRs are illustrated in Figure 7A and B. The experimental analysis indicates that the average SLRs of the IPEFT algorithm are 5.5%, 7.4%, 8.2%, 9.2%, 9.5%, and 16.0% smaller than those of the HEFT algorithm and 4.0%, 5.9%, 4.7%, 3.4%, 4.8%, and 5.7% smaller than those of the PEFT algorithm when the heterogeneity values are 0.1, 0.2, 0.5, 0.75, 1, and 2, respectively. When a low CCR value of 0.1 is given, the average SLR improvement of IPEFT over HEFT and PEFT is 2.5% and 1.2%, respectively. The improvement over

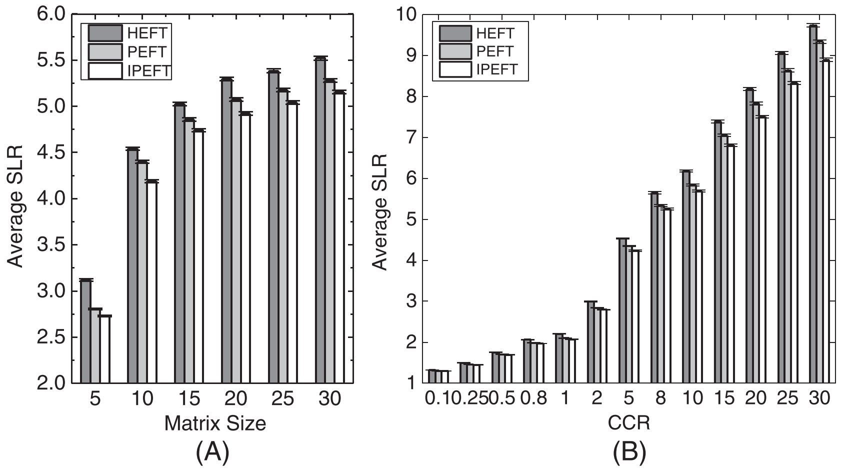


FIGURE 6 For Gaussian elimination directed acyclic graphs: (A) average scheduling length ratio (SLR) for different matrix sizes. The bars represent the 95% confidence intervals of the mean. B, Average SLR for different ratios of communication to computation (CCRs). The bars represent the 95% confidence intervals of the mean. HEFT indicates Heterogeneous Earliest Finish Time; IPEFT, Improved Predict Earliest Finish Time; PEFT, Predict Earliest Finish Time

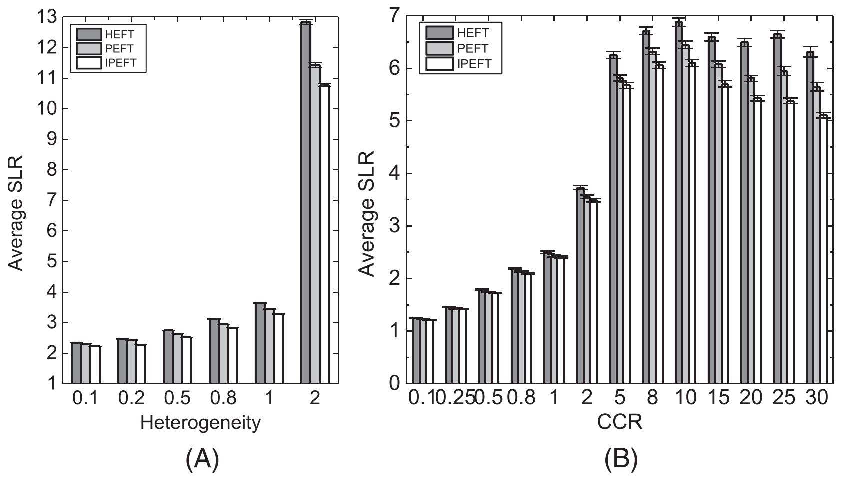


FIGURE 7 For Montage directed acyclic graphs: (A) average scheduling length ratio (SLR) for different heterogeneities. The bars represent the 95% confidence intervals of the mean. B, Average SLR for different ratios of communication to computation (CCRs). The bars represent the 95% confidence intervals of the mean. HEFT indicates Heterogeneous Earliest Finish Time; IPEFT, Improved Predict Earliest Finish Time; PEFT, Predict Earliest Finish Time

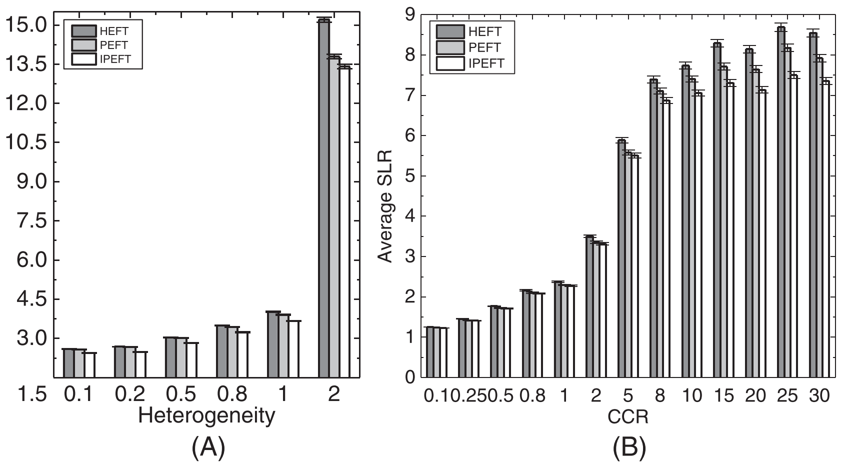


FIGURE 8 For molecular dynamics code directed acyclic graphs: (A) average scheduling length ratios (SLRs) for different heterogeneities. The bars represent the 95% confidence intervals of the mean. B, Average SLR for different ratios of communication to computation (CCRs). The bars represent the 95% confidence intervals of the mean. HEFT indicates Heterogeneous Earliest Finish Time; IPEFT, Improved Predict Earliest Finish Time; PEFT, Predict Earliest Finish Time

HEFT and PEFT increases to 19.1% and 9.6%, respectively, while CCR is equal to 30.

5.3.3 | Molecular dynamics code

The third real-world application is the molecular dynamics code. The average SLRs for different heterogeneities and different CCRs in the molecular dynamics code are shown in Figure 8A and B, respectively. For heterogeneity, the average SLR produced by the IPEFT algorithm is smaller than those of the PEFT and HEFT algorithms. For CCR, the average SLR produced by the IPEFT algorithm is smaller than that of the HEFT algorithm. Compared with the PEFT algorithm, the IPEFT algorithm has similar average SLRs when $CCR < 8$ and smaller average SLRs when $CCR \geq 8$.

6 | CONCLUSIONS

In this paper, we proposed IPEFT, a new 2-phase list-based scheduling algorithm with quadratic complexity for heterogeneous systems. In contrast to existing algorithms, IPEFT calculates the PCT-based priority $rank_{PCT}$ in the task prioritization phase, and the PCT table is computed ahead of scheduling. Every (task, processor) pair in the PCT table indicates the maximum processing time of the longest path from the immediate successors of current task to the exit task, and each task on this path is assigned to the poorest processors. The computation of the priority $rank_{PCT}$ is based on the PCT, which ensures the prior scheduling of tasks that generate the worst scheduling result. In the processor selection phase, the impact of the parent task on the critical task is considered, and each parent is assigned to the processor that minimizes the EFT, which may advance the earliest start time of the critical task and improve scheduling performance. The algorithm considers the (task, processor) pairs in the CNCT, which can predict the impact of current task scheduling on the allocation of critical subtasks. In this manner, the completion time of the selected processor is not always the shortest for current tasks, but it guarantees a shorter finish time for the critical tasks in the next phase. The new algorithm has the same time complexity as the HEFT and PEFT algorithms but performs better than both. Regarding schedule length, the IPEFT algorithm achieved more favorable results than the HEFT and PEFT algorithms in random graphs when the number of tasks ranges from 10 to 400. The robustness of the IPEFT algorithm is also better than that of the PEFT and HEFT algorithms. In addition, the IPEFT algorithm outperformed the HEFT and PEFT algorithms in terms of the frequency of the best results. The IPEFT algorithm also outperformed the HEFT and PEFT algorithms on real-world application graphs.

ACKNOWLEDGMENTS

We would like to show our gratitude to Jorge G. Barbosa and Hamid Arabnejad for providing the source of the PEFT algorithm and their help. We would also like to thank the reviewers for improving the paper by their helpful and perceptive comments. This work was partially supported by the National Natural Science Foundation of China (grant no. 61402183 and 61070015), Guangdong Provincial Science and Technology Projects (grant nos. 2014B090901028, 2014B010117001, 2014A010103022, and 2014A010103008), and Fundamental Research Funds for the Central Universities, SCUT(no. x2jsD2153930).

REFERENCES

1. Ilavarasan E, Thambidurai P. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *J Com Sci*. 2007;3(2):94–103.
2. Feitelson DG, Rudolph L, Schwiegelshohn U, *et al.* Theory and practice in parallel job scheduling. In: *Proceedings of the Job Scheduling Strategies for Parallel Processing, 1997*. London, UK: Springer; 1997:1–34.
3. Dai Y, Zhang X. A synthesized heuristic task scheduling algorithm. *Scientific World Journal*. 2014;2014(2014):1–9. doi: 10.1155/2014/465702.
4. Graham RL, Lawler EL, Lenstra JK, *et al.* Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann Discrete Math*. 1979;5:287–326.
5. Casavant TL, Kuhl JG. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans Softw Eng*. 1988;14(2):141–54. doi: 10.1109/32.4634.
6. Yang T, Gerasoulis A. DSC: scheduling parallel tasks on an unbounded number of processors. *IEEE Trans Parallel Distrib Syst*. 1994;5(9):951–67. doi: 10.1109/71.308533.
7. Liou J-C, Palis MA. An efficient task clustering heuristic for scheduling dags on multiprocessors. Proceedings of the workshop on resource management, symposium of parallel and distributed processing, October1996;152–6.
8. Kwok Y-K, Ahmad I. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Trans Parallel Distrib Syst*. 1996;7(5):506–21. doi: 10.1109/71.503776.
9. Boeres C, Rebello VE. A cluster-based strategy for scheduling task on heterogeneous processors. Proceedings of 16th Symposium on Computer Architecture and High Performance Computing, October 2004. *IEEE*. 2004;214–21.
10. Cirou B, Jeannot E. Triplet: a clustering scheduling algorithm for heterogeneous systems. Proceedings of the International Conference on Parallel Processing Workshops, Valencia, September 2001. *IEEE*. 2001;231–6.
11. Arabnejad H, Barbosa JG. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans Parallel Distrib Syst*. 2014;25(3):682–94. doi: 10.1109/Tpds.2013.57.
12. Ahmad I, Kwok Y-K. On exploiting task duplication in parallel program scheduling. *IEEE Trans Parallel Distrib Syst*. 1998;9(9):872–92. doi: 10.1109/71.722221.
13. Choe T-Y, Park C-I. A task duplication based scheduling algorithm with optimality condition in heterogeneous systems. Proceedings of the International Conference on Parallel Processing Workshops, 2002. *IEEE*. 2002;531–6.
14. Bajaj R, Agrawal DP. Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans Parallel Distrib Syst*. 2004;15(2):107–18. doi: 10.1109/TPDS.2004.1264795.
15. Lan Z, Sun SX. Scheduling algorithm based on critical tasks in heterogeneous environments. *J Syst Eng Electron*. 2008;19(2):398–405. doi: 10.1016/S1004-4132(08)60099-7.
16. Topcuoglu H, Hariri S, M-Y W. Task scheduling algorithms for heterogeneous processors. Proceedings of the Heterogeneous Computing Workshop, 1999(HCW'99) Proceedings Eighth, 1999. *IEEE*. 1999;3–14.
17. Radulescu A, Van Gemund AJ. Fast and effective task scheduling in heterogeneous systems. Proceedings of the Heterogeneous Computing Workshop, 2000(HCW 2000) Proceedings 9th, Cancun, 2000. *IEEE*. 2000;229–38.
18. Topcuoglu H, Hariri S, Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distrib Syst*. 2002;13(3):260–74. doi: 10.1109/71.993206.

19. Hagras T, Janecek J. A simple scheduling heuristic for heterogeneous computing environments. Proceedings of the Parallel and Distributed Computing, International Symposium on2003; IEEE Computer Society104–10.

20. Ilavarasan E, Thambidurai P, Mahilmannan R. High performance task scheduling algorithm for heterogeneous computing system. *Distributed and Parallel Computing Springer*. 2005;193–203.

21. Daoud MI, Kharma N. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *J Parallel Distr Com*. 2008;68(4):399–409. doi: 10.1016/j.jpdc.2007.05.015.

22. Bittencourt LF, Sakellariou R, Madeira ERM. DAG scheduling using a lookahead variant of the Heterogeneous Earliest Finish Time algorithm. Proceedings of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'10)2010;27–34. doi: 10.1109/pdp.2010.56.

23. Canon L-C, Jeannot E, Sakellariou R, *et al*. Comparative evaluation of the robustness of dag scheduling heuristics. Proceedings of Grid Computing, Achievements and Prospects2008; Springer73–84.

24. Hagras T, Janeček J. A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. *Parallel Computing*. 2005;31(7):653–70. doi: 10.1016/j.parco.2005.04.002.

25. Shi Z, Jeannot E, Dongarra JJ. Robust task scheduling in non-deterministic heterogeneous computing systems. Proceedings of IEEE International Conference on Cluster Computing. September*IEEE*. 2006;1–10.

26. Suter F. A synthethic task graph generator. https://github.com/frs69wq/daggen [8 September 2014].

27. Deelman E, Singh G, Su M-H, *et al*. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Sci Prog*. 2005;13(3):219–37. doi: 10.1155/2005/128026.

How to cite this article: Zhou N, Qi D, Wang X, Zheng Z, Lin W. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. *Concurrency Computat.: Pract. Exper.* 2017:29(5);e3944. https://doi.org/10.1002/cpe.3944