IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ARTIFICIAL INTELLIGENCE IN PARALLEL AND DISTRIBUTED COMPUTING

Received October 21, 2019, accepted November 19, 2019, date of publication December 2, 2019,
date of current version December 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2956759

DBEFT: A Dependency-Ratio Bundling Earliest Finish Time Algorithm for Heterogeneous Computing

TAO LI, (Member, IEEE), DINGYUAN CAO, YE LU, TEHUI HUANG, CHENGJUN SUN, QIANKUN DONG, AND XIAOLI GONG, (Member, IEEE)
College of Computer Science, Nankai University, Tianjin 300350, China
Tianjin Key Laboratory of Network and Data Security Technology, Tianjin 300350, China

Corresponding authors: Qiankun Dong (qiankund@nankai.edu.cn) and Xiaoli Gong (gongxiaoli@nankai.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003405, Grant 2018YFB2100304, and Grant 2016YFC0400709, in part by the National Natural Science Foundation under Grant 61872200 and Grant 61702286, in part by the Natural Science Foundation of Tianjin under Grant 18JCYBJC15600, Grant 19JCZDJC31600, and Grant 19JCQNJC00600, and in part by the China Civil Aviation Security Capacity Building Fund Project under Grant PESA 2018082.

ABSTRACT Performance effective task scheduling algorithms are essential for taking advantage of the heterogeneous multi-processor in heterogeneous computing environments. In this paper, we present a task scheduling algorithm named as Dependency-ratio Bundling Earliest Finish Time (DBEFT). DBEFT is a list based scheduling algorithm combined with task duplication, which can achieve high performance and low time complexity simultaneously. DBEFT selects the task from the perspective of extending parallelism between tasks instead of giving priorities to tasks on the critical path. Also, DBEFT reduces communication cost by adopting a bundling scheduling strategy. The experiments were conducted on both random graph set and real-world applications, and the results show that DBEFT obtained significant performance improvement, outperforming CEFT by 15%, PEFT by 30% and HEFT 33% in terms of SLR respectively.

INDEX TERMS Heterogeneous system, task scheduling, task duplication, task dependency ratio, bundling scheduling.

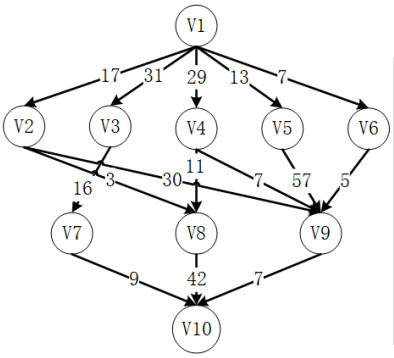
I. INTRODUCTION

In recent years, there has been an increasing number of heterogeneous computing systems, including many systems at leadership computing facilities or warehouse-scale data centers. Normally, there are several phases when constructing and improving the large-scale computing systems or data centers. In each phase, either the initial deployment or the later upgrade, the mainstream processors at that time will be equipped in the system, which makes the entire system or data center equipped with a number of different processors. On the other hand, with the recent emergence of hardware accelerators or co-processor technologies, such as CPU + GPU and CPU + FPGA, chips with different performance exist in the same system, which makes it a heterogeneous platform too. In addition, in the embedded system, in order to purchase a higher performance-energy ratio, the heterogeneous cores are integrated into a single chip, such as the big.LITTLE architecture proposed by ARM [7], which makes heterogeneous computing a widespread technology for embedded devices. All these possible architecture choices make heterogeneous computing systems ubiquitous today.

Especially in the research field of supercomputing, there are many notable examples of large-scale heterogeneous systems in the top-ranking HPC (High-Performance Computing) data centers, including Tianhe, Titan, Trinity, PizDaint, and Hazel Hen [1]. For example, there are 7168 general-purpose nodes in Tianhe-1 system connected with Tianhe autonomous high-speed network and 512 nodes of HPC1 system connected with Infiniband. High-performance heterogeneous computing can be applied to many aspects, such as big data processing [42], machine learning [16], [24], image and video processing [13]–[15], [17].

Efficient scheduling is one of the key factors for high performance on heterogeneous systems. Task scheduling is a traditional problem in heterogeneous computing. In the

The associate editor coordinating the review of this manuscript and approving it for publication was Yong Chen.



Computation time matrix

Task	P1	P2	P3
V1	22	21	36
V2	22	18	18
V3	32	27	43
V4	7	10	4
V5	29	27	35
V6	26	17	24
V7	14	25	30
V8	29	23	36
V9	15	21	8
V10	13	16	33

FIGURE 1. A sample application DAG and computation time matrix of the tasks on each processor.

scheduling problem, applications are divided into parallel tasks and the relationship among tasks are represented as a directed acyclic graph (DAG). In the DAG, each task is represented as a node and the edges between nodes represent the dependency requirement for arbitrary executions such as the example shown in Figure 1. The scheduling algorithms take the DAG as input, and generate an execution order for tasks, and assign them onto different processors. In order to take advantage of the heterogeneous multi-processors, an optimal solution is required so that task dependency requirements are satisfied while minimizing the makespan (total completion time of the application). Many researchers have tried to achieve efficient scheduling [5], [6], [11], [19], [20], [25], [33], [37], [40], [44]. However, scheduling is an NP-hard problem [38] and currently it is impossible to get an optimal solution in an acceptable time.

In order to obtain the approximately optimal heuristic solutions, there are some assumptions and constraints on the general-purpose scheduling problem. Static scheduling [5], [6], [19], [20], [25], [37], [44] and dynamic scheduling [11], [33] are two major categories of the research field. Dynamic scheduling is a runtime scheduling, in which all information about tasks (such as execution cost and communication cost) is unknown and the execution order is determined at runtime. In contrast, static scheduling is a compile-time scheduling that all information is available beforehand, which indicates we can optimize decisions in advance according to the whole workload. In this paper, we focus on the problem of static scheduling algorithms for a general model of deterministic parallel tasks.

Static scheduling consists of heuristic-based algorithms [6], [19], [20], [25], [37] and guided random search-based algorithms [35], [41], [47]. Generally, high quality guided random search-based algorithms have expensive iteration computing cost. Therefore, most of the existing static scheduling algorithms are heuristic-based algorithms, which try to produce a solution good enough rather than optimal, with lower computing complexity.

By extending the parallelism between tasks and reducing communication cost, the quality of the scheduling algorithms can be improved. However, extending the parallelism will bring more communication cost among tasks. Therefore, efficient scheduling algorithms with the consideration of communication cost is critical to explore the potential of multi-processor computing platforms. Many existing algorithms mainly focus on the computational requirements rather than the communication cost. However, with the enormous computing power of heterogeneous systems, the communication cost may become the bottleneck in these systems, particularly when executing applications with huge communication requirements, such as image processing, weather modeling, and distributed database systems.

In this paper, a heuristic algorithm named DBEFT (Dependency-ratio Bundling Earliest Finish Time) is introduced to reduce the communication cost by bundling the tasks with heavy communication cost as one node. We evaluated DBEFT with randomly generated DAGs [2] and DAGs constructed based on well-known real applications such as Gaussian Elimination and Fast Fourier Transform (FFT). Results show that DBEFT obtains better performance than the state-of-art solutions including CEFT [25], PEFT [6] and HEFT [37]. In particular, we make the following contributions.

- We introduce the concept of the task dependency ratio to locate the bottleneck of data communication in task scheduling.
- We propose a heuristic algorithm DBEFT. Based on the analysis for task dependency, the bundling scheduling strategy and duplication mechanism are adopted to mitigate communication cost.
- We introduce two new metrics, redundant computation ratio(RCR) and average idle time(AIT), to evaluate the static heuristic scheduling algorithms.
- Comprehensive experiments were conducted to evaluate DBEFT and compare it with state-of-art solutions.

The rest of this paper is organized as follows. In SectionII, we introduce the DAG scheduling problem. The related work in scheduling DAGs on heterogeneous systems is presented in SectionIII. SectionIV describes the details of DBEFT. SectionV demonstrates the experimental details and analyzes the results. We conclude the paper in SectionVI.

II. THE DAG SCHEDULING PROBLEM

With the widespread use of heterogeneous high-performance computing systems in recent years, scheduling in heterogeneous multi-processor computing environments also becomes a hot topic for researches [23], [26]. In recent years, many models have been proposed by researchers to describe the requirements of practical applications, and many scheduling methods have been studied based on those models. The task scheduling model based on DAGs is one of the popular methods.

In order to explore the potential of applications' parallelism, a general way is to divide the entire application into multiple fine-grained tasks, and then build a DAG based on the dependency between tasks. In the task scheduling model based DAG, each basic operation of the target application

is considered as a node in the DAG. The directed edge in the DAG is the dependency between nodes. A task node is ready only when all its predecessor nodes were finished. If considering the communication cost, a node also has to wait until the data required is transferred from different processors.

A DAG can be described as $G = (V, E, W, C)$ to represent an application, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of n task vertices, E is the set of edges $e_{i,j}$ in DAG, W is the computation-cost matrix of tasks, and $c_{i,j} \in C$ is the weight of $e_{i,j}$ that represents the communication cost of the data transmission between task v_i and v_j. For example, Figure 1 shows a DAG generated from a sample application. The numbers on the edges are the communication cost, which means the time to transmit the computation result from the predecessor node to the successor node. Generally, the data is stored in the local memory. Therefore, the communication cost will be zero if the predecessor and successor tasks are assigned to the same processor. The right side of the Figure 1 is the computation time matrix. There are 3 different processors in this example, and the time consumption of every node on each processor is demonstrated in the matrix.

A schedule of a DAG is an assignment of the tasks in the graph to the processors, such that the predecessor constraints are respected. The objective of the scheduling problem discussed in this paper is to minimize the makespan or schedule length for a given DAG.

In Table 1, we present symbols that will be referred to in the following sections. In this paper, we assume that the topology of the target heterogeneous platform is fully connected, which means the result on one processor can be transmitted to any other processor directly. The execution of any task is considered as non-preemptive. On each processor, the execution of tasks and the communication with other processors can be overlapped.

TABLE 1. Symbol definition.

Symbol	Definition
V	the set of tasks in DAG, task $v_i \in V$
E	the set of edges in DAG, edge $e_{i,j} \in E$
P	the set of all processors in system, processor $p_j \in P$
W	a $\|V\| * \|P\|$ matrix where $w_{i,j} \in W$ is the computation cost of task v_i on processor p_j
C	the communication overload, $c_{i,j} \in$C is the weight of $e_{i,j}$ and represents the communication cost when the data is transmitted from task v_i to v_j if v_i and v_j are executed on different processors. If v_i and v_j are executed on the same processor, then $c_{i,j}$=0.
$\overline{w_i}$	$\overline{w_i}$ denotes the average computation cost of v_i on all processors. $\overline{w_i} = \Sigma_{j \in P} w_{i,j}/\|P\|$
$pred(v_i)$	$pred(v_i)$ denotes the set of immediate predecessors of task v_i in a given DAG. v_i cannot be scheduled until task $v_j \in pred(v_i)$ have finished.
$succ(v_i)$	$succ(v_i)$ denotes the set of immediate successors of task v_i in a given DAG. Before v_i is finished, no $v_j \in succ(v_i)$ can be scheduled.
v_{entry}	The entry node or starting node in a given DAG. There may be multiple entry nodes in a DAG.
v_{exit}	The exit node or stopping node in a given DAG. Of course, there may be multiple exit nodes in a DAG.
$makespan$	or schedule length, it denotes the completion time of the last finished node in DAG and can be computed by $makespan = max\{AFT(v_{exit})\}$
$EST(v_i,p_j)$	the earliest start time of task v_i on processor p_j.For the entry task v_{entry}, the task has no predecessors, $EST(v_{entry},p_j)$=0.
$EFT(v_i,p_j)$	the earliest finish time of task v_i on processor p_j
AFT	the actual finish time, $AFT(v_i,p_j)$ denotes the actual finish time of task v_i on processor p_j, $AFT(v_i)$ is the actual finish time of task v_i
$dp(v_i)$	$dp(v_i)$ denotes an immediate processor of v_i that is duplicated. Duplicating $dp(v_i)$ to the execution processor of v_i can decrease the communication overload.
$T_{Available}(p_j)$	the earliest time at which processor p_j is ready.

III. RELATED WORKS

Since many parallel applications running on heterogeneous systems require intensive data processing and data communication, scheduling strategies to deploy the applications have a significant impact on the overall system performance. Static scheduling algorithms consist of heuristic-based scheduling and random search-based scheduling. Random search-based scheduling algorithms [35], [41], [47] are less efficient and have much higher time complexity than the heuristic-based algorithms. The heuristic-based scheduling algorithms [6], [9], [18], [22], [37], [39], [46] can be classified as list scheduling, cluster scheduling [10], [28], [31], and task duplication-based scheduling [4], [8], [27], [32]. Generally, cluster scheduling has the limitation of time complexity in high heterogeneity systems. On the contrary, list scheduling is more feasible in real-time scheduling because of its simplicity. Duplication based scheduling considers re-computing the result instead of waiting for data communication, which has higher performance when the communication is the bottleneck of the system.

A. LIST-BASED SCHEDULING ALGORITHMS

Substantial research efforts have been devoted to list scheduling because of its high performance with low time complexity. List scheduling algorithms consist of two phases. In the first phase, each task is assigned with a priority and then added to a list of ready tasks in a decreasing order of priority. In the second phase, once any processor becomes available, the task with the highest priority is selected and assigned to the processor. The performance of list scheduling algorithms is heavily dependent on the effectiveness of the heuristics mechanism. For example, on fine-grained task graphs with high communication to computation cost ratio (CCR), the performance of list scheduling algorithms tends to deteriorate dramatically because of communication overhead.

B. TASK DUPLICATION-BASED SCHEDULING ALGORITHMS

As we know, if two dependent tasks are assigned to the same processor, there is no data transmission between them.

On the other hand, if they are assigned to two different processors, the communication overhead is incurred. The idea behind the duplication-based scheduling algorithms is to reduce the transmission time by copying some of the predecessor tasks of task v_i to its execution processor. Dong and Luo proposed Heterogeneous Dynamic Critical Path and multi-task Duplication (HDCPD) algorithm [21], where actual execution time of scheduled tasks instead of average execution time participates in computing the critical path and multi-level and multi-task duplication mechanism is used to achieve high performance. Lotfifar and Shahhoseini proposed Multiple Critical Path Dominator (MCPD) [30], in which task nodes in a given DAG are sorted by their contacts with the static critical path, and the processor with minimum ready time for executing the task is selected. The time complexity of MCPD is $O(v^2)$. Agarwal and Kunar presented Heterogeneous Economical Duplication (HED) [3] to optimize the duplication when a schedule has been generated. In practical, task duplication scheduling usually assumes that the processors are identical and the computing complexity increases dramatically for heterogeneous systems.

C. COMBINATION OF LIST-BASED SCHEDULING AND TASK DUPLICATION-BASED SCHEDULING

The reason behind duplication-based heuristics is to achieve the reduction of the communication overhead by allocating some tasks to multiple processors redundantly [4], [8], [27], [32]. However, the duplication strategy is quite different from the traditional duplication based scheduling algorithms. The most common technique in duplication based algorithms is to recursively duplicate ancestor nodes in a bottom-up manner, as done in the CPFD [4] and SD [8] algorithms. Even though duplication-based algorithms obtain higher performance during the task execution, due to the complexity of backward searches for the ancestor nodes, duplication-based heuristics have higher complexity during the task assignment and scheduling. In order to achieve high performance with low complexity, an interesting approach proposed by combining list scheduling with task duplication [29], [34], [36], [37], [45], which allocates the nodes to more than one processor, in order to reduce the waiting time of the dependent tasks.

D. SELECTED BASELINE SCHEDULING ALGORITHMS

In this subsection, we introduce three state-of-the-art list scheduling heuristics algorithms, HEFT [37], PEFT [6] and CEFT [25], which are also used as baseline algorithms for comparison experiments.

Heterogeneous Earliest Finish Time (HEFT) algorithm [37] defines task priority according to $rank_u$ that represents the length of the critical path from the task to the exit task and is given by Equation(1). $Succ(v_i)$ is a set of immediate successors of task v_i. The task list is ordered by the decreasing priority of tasks. The task v_i on the top of the task list is assigned to the processor that can minimize EFT of v_i, which indicates HEFT aims to achieve global optimal solution (makespan) by obtaining local optimal solution in the scheduling of every task. HEFT is a highly competitive algorithm in terms of scheduling length with low complexity of the order $O(n^2m)$.

$$rank_u(v_i) = \overline{w_i} + \max_{v_i \in succ(v_i)} \left\{ \overline{c_{i,j}} + rank_u(v_j) \right\} \tag{1}$$

Predict Earliest Finish Time (PEFT) algorithm [6] defines task priority rank_{oct} according to the optimistic cost table (OCT) which is given by Equation(2). Each element $\text{OCT}(v_i, p_k)$ represents the maximum of the shortest path of v_i's children tasks to the exit task considering that processor p_j is selected for task v_i. The sum of $EFT(v_i, p_k)$ and $\text{OCT}(v_i, p_k)$ is used for selecting the execution processor for tasks, which indicates the aim of PEFT is to guarantee that the tasks ahead will finish earlier. PEFT is a high-performance algorithm with forward-looking feature while maintaining the time complexity of the order $O(n^2m)$.

$$rank_{oct}(v_i) = \sum_{k=1}^{P} \max_{v_j \in succ_i} \left[\min_{p_w \in P} \{ OCT(v_i, p_w) + w(v_j, p_w) + \overline{c_{i,j}} \} \right] / P \tag{2}$$

Communication-aware earliest finish time (CEFT) algorithm [25] combines list-based scheduling with task duplication. CEFT defines task priority according to rank_u as in HEFT and communication ration(CR). According to the task priority Equation(4), before every scheduling, local normalization is applied to the $rank_u$(as defined in Equation(1)) and CR (as defined in Equation(3)) for all tasks in the current ready-list. Then task priority is assigned according to these two attributes of tasks. The ready task with the maximum $rank_{prio}$ is selected to be scheduled. The existence of CR makes it more possible to schedule dependent tasks with higher communication cost on the same processor. If the communication overhead is not eliminated by scheduling the dependent tasks on the same processor, try duplicating a task's immediate predecessor to its execution processor to further reduce communication. If duplicating $dp(v_i)$ to v_i's execution processor can decrease $EST(v_i)$, task v_i and $dp(v_i)$ will be scheduled to their best processor. The best processor is the one on which minimum EFT of v_i can be obtained.

$$CR(v_i) = max\left\{C_{pred(v_i)}\right\} / max\left\{C_{succ(v_i)}\right\} \tag{3}$$

$$rank_{prio}(v_i) = \alpha * \frac{rand_u(v_i)}{\sum_{v_j \in ready-list} rank_u(v_j)} + (1-\alpha) * \frac{CR_i}{\sum_{v_j \in ready-list} CR_j} \tag{4}$$

IV. DEPENDENCY-RATIO BUNDLING EARLIEST FINISH TIME (DBEFT) ALGORITHM

A. TASK DEPENDENCY RATIO

In order to improve the performance of scheduling algorithms, we try to simultaneously increase the task-level parallelism and reduce communication cost.

Considering the list-based scheduling algorithms, at any point of time, all tasks in the ready-list can be executed concurrently, but the number of tasks in the ready-list is not fixed, as the completion time of each task varies. We define the number of tasks that can be executed concurrently after the completion of task v_i as task parallelism number (TPN(v_i)). Task dependency ratio (TDR) is defined in Equation(5) and can be used to evaluate the effect of the completion of task v_i on TPN(v_i).

$$TDR(v_i) = \sum_{v_i \in succ(v_i)} (1/|pred(v_i)|) \tag{5}$$

The dependencies of tasks in DAG graph can be classified into single-task dependency and multi-task dependency.

- *Single-task dependency.* If task v_i is the only immediate predecessor of task v_j, then task v_j has single-task dependency with task v_i.
- *Multi-task dependency.* Task v_j has more than one immediate processors and task v_i is one of the immediate predecessors of task v_j, then task v_j has the multi-task dependency with task v_i.

B. BUNDLING SCHEDULING STRATEGY

In order to improve performance, DBEFT reduces communication cost by overlapping computing and communication of tasks as much as possible. In our DAG-based scheduling model, the execution of tasks and communication with other processors can be done on each processor simultaneously without contention. It is obviously that hiding the communication cost with computation is better than duplication-based strategy. However, in some cases, the high communication cost is too high to be hidden completely, and in this case the communication cost will make processors stay idle waiting for data transmission, which is the main cause for low resource utilization.

In DBEFT, bundled scheduling strategy is adopted to eliminate high communication cost. The communication weight greater than the 5% observation point value on the right side of the normal distribution is defined as "ultra-long" communication time. 1.65 is the corresponding value of the 0.9505-quantile on standard normal distribution table. Two tasks with high communication cost between them will be bundled into one big task. DBEFT defines high communication costs (HC) in the given DAG by Equation(6), μ is the mean value and σ^2 is the variance of computation costs in the given DAG. In other words, two tasks with high communication cost between them will be frozen until both of them are ready to be scheduled, and once the two tasks are ready, they will be dispatched to the same execution processor.

$$HC \geq 1.65 \times \sigma + \mu \tag{6}$$

Taking the DAG in Figure 1 as an example, the mean communication cost is 22.70 and the standard deviation is 7.55. Based on Equation(6), the 5% observation point value on the right side of the normal distribution is 35.16, with mean 22.70 and variance 7.55^2. The communication cost higher than the criteria in the sample DAG are 57 and 42. Therefore, tasks v_5 and v_9 are considered as a high communication cost pair and they are bundled. The bundled tasks are denoted as [v_5, v_9]. v_5 and v_9 will be frozen before the two immediate predecessor tasks v_2 and v_4 are scheduled. Once v_2 and v_4 are scheduled, tasks v_5 and v_9 will be dispatched together. Similarly, tasks v_8 and v_{10} are bundled and frozen.

C. DETAILED DESCRIPTION OF DBEFT ALGORITHM

In this section, we explain the details of DBEFT algorithm, including task priority assignment, duplication mechanism, execution processor selection, and bundling scheduling strategy.

1) TASK PRIORITY ASSIGNMENT

As shown in Equation(5), $TDR(v_i)$) is introduced to explore the potential of the parallelism for an application. DBEFT set up the priorities for tasks based on $TDR(v_i)$. In addition, the overlap between the calculation and the communication is also fully exploited to help determine the task priority. There are two basic principles when setting up the priority for a task:

- Considering that the completion of a task with high TDR can make more successor tasks into the ready state, it is more conducive to increase the parallelism among tasks. Therefore, a task with a higher TDR value should be given a higher priority.
- The lower communication cost is, the more it is likely to be overlapped by the calculation of the successor tasks. If some tasks have the same TDR, then the task that has minimum communication cost with its immediate predecessors should have a higher priority.

Considering the above two principles, the task priority is calculated as Equation(7):

$$Prio(v_i) = w_1 * TDR(v_i) + w_2 * 1/max_{v_j \in pred(v_i)}(c_{i,j}) \tag{7}$$

where TDR(v_i) is the task dependency ratio of v_i (TDR(v_i) should be updated in real-time). The $max_{v_j \in pred(v_i)}(c_{i,j})$ is the maximum communication cost between task v_i and its immediate predecessors. w_1 and w_2 represent the weights of the TDR and the communication cost respectively when determining the task priority. Since DBEFT determines the priorities according to TDR, when there are two tasks with the same TDR, the communication cost is considered. Therefore, we should ensure that $w_1 > w_2$. In this paper, we assume $w_1 = 10$, $w_2 = 1$. Equation(7) can be expressed as Equation(8):

$$Prio(v_i) = 10 * TDR(v_i) + 1/max_{v_j \in pred(v_i)}(c_{i,j}) \tag{8}$$

The task with highest priority in ready-list will be dispatched, except bundled tasks (described in Section IV-C.4).

2) TASK DUPLICATION MECHANISM

Single immediate predecessor task duplication strategy is adopted in DBEFT, which means only one immediate predecessor of the current scheduled task can be duplicated at

most. Suppose v_k is one of the immediate predecessors of task v_i and has been scheduled. Before dispatching task v_i, we will consider whether to duplicate v_k on task v_i's execution processor p_j. If duplicating v_k on p_j can make v_i start on p_j in advance, then we will duplicate v_k on p_j. We will lable the duplicated v_k as $dp(v_i)$.

3) PROCESSOR SELECTION

In the processor selection phase, we calculate the earliest finish time (EFT) of task v_i according to Equation(9) on all available processors. The $dp(v_i)$ is one of the immediate predecessors of task v_i, which need to be duplicated on the execution processor of v_i. Task v_i and $dp(v_i)$ (if $dp(v_i)$ exists) will be scheduled to the processor with minimum EFT of v_i.

$$\begin{aligned} &EFT(v_i, p_j) = EST(v_i, p_j) + w_{i,j}, \\ &EST = min\{AFT(dp(v_i), p_j), \\ &max\{T_{Avaible(p_j)}, max_{v_m \in pred(v_i)}\{AFT(v_m, p_j) + c_{m,i}\}\}\} \end{aligned} \quad (9)$$

EST(v_i, p_j) is the earliest start time of task v_i on processor p_j, while $T_{Available}(p_j)$ is the available time of the processor p_j. AFT(v_i, p_j) is the actual finish time of task v_i on processor p_j. In this paper, AFT is used instead of EFT in actual scheduling.

4) TREATMENT OF BUNDLED TASKS

In DBEFT, the "big task" (the bundled task of v_i and its successor) with high communication cost is placed in the frozen-list. The tasks in frozen-list are in frozen state and cannot be dispatched until other immediate predecessors of v_i's successor tasks have been dispatched. Once the other predecessor tasks are dispatched, v_i is put into the ready-list and will be dispatched immediately. A bundled task chooses the processor that will minimize the EST of its successor tasks.

5) ALGORITHM DETAILS

The pseudo code of DBEFT algorithm is demonstrated in Algorithm 1. The algorithm starts by computing communication cost (HC) according to Equation(6) and then put the tasks with high communication cost into the frozen-list. After that, an empty ready-list is created and the entry tasks v_{entry} (the task without predecessors) are placed on top of the list. In the **while** loop, the selected tasks are scheduled in every iteration. The processor p_j that can achieve the minimum EFT(v_i, p_j) is selected as the execution processor of task v_i.

Table 2 shows an example that demonstrates how DBEFT works for the DAG in Figure 1. The scheduling procedure is shown in Figure 2. Compared with PEFT, DBEFT shortens EFT by increasing the EST of task v_2 on p_1, and tasks v_5 and v_7 are similar. Compared with CEFT, the DBEFT will bundle v_8 with v_{10} and dispatch the bundled tasks after all the other immediate predecessors of v_{10}. It can effectively reduce the performance loss caused by communication cost between different processors. DBEFT takes full advantage of the idle time on the processor and achieves higher scheduling performance than other algorithms.

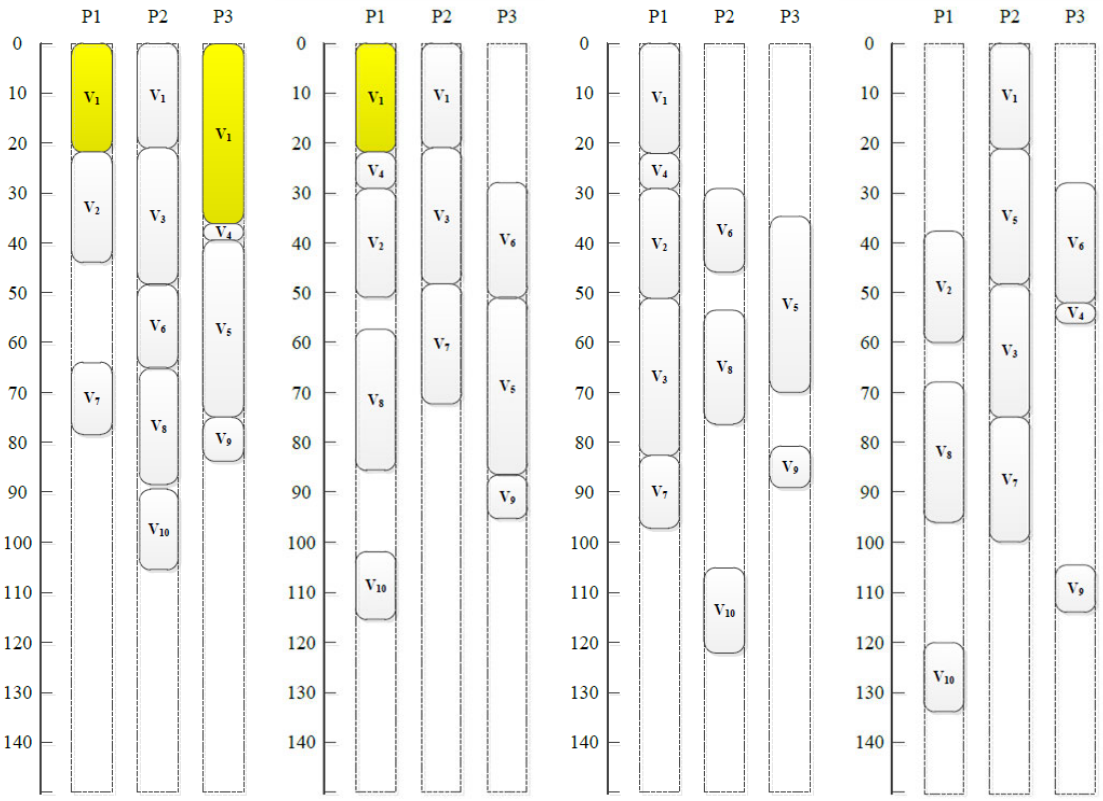
Algorithm 1 DBEFT Algorithm

1: Compute TDR(v_i) for all tasks in DAG. Select tasks with high communication costs and bundle them, then put the bundled tasks into the frozen-list, create empty ready-list and put v_{entry} as initial task into the ready-list. v_{active} is a variable used to preserve the task that need to be scheduled now and is initiated to 0.
2: **while** ready-list is NOT empty **do**
3: **for all** task v_i in ready-list **do**
4: Compute task priority according to equation(8) and sort the priorities
5: **for all** processor p_j in processor set P **do**
6: Compute $T_{Available}(p_j)$
7: **if** exist $dp(v_i)$ in $pred(v_i)$ **then**
8: Compute EST(v_i, p_j),EFT(v_i, p_j) after duplicating $dp(v_i)$ on p_j
9: **else**
10: Compute EST(v_i, p_j),EFT(v_i, p_j)
11: **end if**
12: **end for**
13: **end for**
14: **if** exist bundled tasks from frozen-list in ready-list **then**
15: v_{active} =bundled tasks[v_i, $succ(v_i)$]
16: Assign tasks [v_i, $succ(v_i)$] to processor p_k that minimizes EFT of task $succ(v_i)$
17: **else**
18: $v_{active} \leftarrow$ the task with the highest $Prio(v_i)$ in ready-list
19: Assign task v_{active} and $dp(v_{active})$ (if exists) to the processor p_k that minimizes EFT of task v_{active}
20: **end if**
21: Remove v_{active} from ready-list
22: **if** exist [v_i, $succ(v_i)$] can be removed from frozen-list **then**
23: Remove [v_i, $succ(v_i)$] from frozen-list, update frozen-list. Put [v_i, $succ(v_i)$] into ready-list
24: **end if**
25: Put the ready tasks whose predecessors have all been scheduled into ready-list
26: Update the ready-list
27: Update TDR(v_i) for all tasks those are not scheduled in DAG
28: **end while**

The time complexity of the frozen-list creation is $O(e)$. The **for** loop at line 3 has the complexity $O(n)$ for n tasks. The duplication mechanism at line 19 is $O(p * degree(e))$ ($degree(e)$ denotes the maximum out/in degree in the given DAG). The update of ready-list and TDR in line 26 and line 27 has the complexity $O(n * degree(e))$. Thus the total time comlexity is $O(e + n * m * degree(e) + n * degree(e) + n * degree(e))$. For dense graphs, e is about n^2 and the total complexity is $O(max\{n^2, n * m * degree(e)\})$.

TABLE 2. Schedule produced by the DBEFT algorithm in each iteration.

Step	Ready-list	Frozen-list	Task selected	EST			EFT			Processor selected	other
				p_1	p_2	p_3	p_1	p_2	p_3		
1	v_1	$[v_5,v_9],[v_8,v_{10}]$	v_1	0	0	0	22	21	36	p_2	
2	v_2,v_3,v_4,v_6	$[v_5,v_9],[v_8,v_{10}]$	v_3	22	21	36	54	48	79	p_2	
3	v_2,v_4,v_6,v_7	$[v_5,v_9],[v_8,v_{10}]$	v_2	22	48	36	44	66	54	p_1	$dp(v_2)$=v_1
4	v_4,v_6,v_7	$[v_5,v_9],[v_8,v_{10}]$	v_4	44	48	36	51	58	40	p_3	dp(v_4)=v_1
5	$[v_5,v_9],v_6,v_7$	$[v_8,v_{10}]$	$[v_5,v_9]$	44	48	40	88	96	83	p_3	Bundled task
6	v_6,v_7	$[v_8,v_{10}]$	v_6	44	48	83	70	65	107	p_2	
7	v_7	$[v_8,v_{10}]$	v_7	64	65	83	78	90	113	p_1	
8	$[v_8,v_{10}]$	empty	$[v_8,v_{10}]$	78	65	83	120	106	152	p_2	Bundled task



(a) DBEFT (b) CEFT (c) PEFT (d) HEFT

FIGURE 2. Schedules of the sample DAG in Figure 1 with (a) DBEFT (makespan=106), (b) CEFT(α = 0.25; makespan=115), (c) PEFT (makespan=122), (d) HEFT (makespan=133). The yellow tasks are duplicated tasks.

Consequently, DBEFT has lower time complexity than CEFT, PEFT and HEFT.

V. EVALUATION AND DISCUSSION

We have conducted a series of experiments to evaluate the proposed DBEFT algorithm and compared it with numerous existing state-of-the-art scheduling algorithms. Our evaluation is based on a series of comprehensive evaluation metrics. In this section, we discuss the evaluation metrics first, and then shows the evaluation results in both simulation workloads and real-world workloads.

A. EVALUATION METRICS

Two traditional metrics, scheduling length ratio(SLR) and speedup, are involved to evaluate DBEFT. In addition, two new metrics are introduced in this study, which are Average Idle Time(AIT) and Redundant Computing Ratio(RCR). These metrics are discussed in detail below.

1) SCHEDULING LENGTH RATIO

Makespan is a metric used to evaluate a single DAG schedule, while scheduling length ratio(SLR) [37] is a metric used to evaluate the performance of an algorithm on DAGs with various topologies. For a given DAG, SLR represents the makespan normalized to the minimum computation cost of the critical path (CP_{MIN}), which is the lower bound of makespan, and is given in Equation(10). Therefore, a good algorithm should have a low SLR.

$$SLR = \frac{makespan(solution)}{\sum_{n_i \in CP_{MIN}} min_{p_j \in P} w_{i,j}} \tag{10}$$

2) SPEEDUP

In general, speedup is defined as the ratio of the sequential execution time to the parallel execution time, i.e. makespan, as shown in Equation(11). The sequential execution time is obtained by assigning all tasks to a single processor, which can minimize the total computation cost for a given DAG.

$$Speedup = \frac{min_{p_j \in P}\left[\sum_{v_i \in V} w_{i,j}\right]}{makespan(solution)} \tag{11}$$

3) AVERAGE IDLE TIME

We define average idle time(AIT) in Equation(12) to evaluate the computing resource utilization of an algorithm. $\sum_{v_i \in V} w_{dp(v_i),p_j}$ is the total redundant computation cost caused by all $dp(v_i)$. A low AIT implies less idle time on processors.

$$AIT = (|P| * makespan - \sum_{v_i \in V, p_j \in P} w_{i,j} - \sum_{v_i \in V} w_{dp(v_i),p_j})/|P| \tag{12}$$

4) REDUNDANT COMPUTING RATIO

The redundant computing ratio (RCR) is shown in Equation(13) to evaluate the efficiency of task duplication based scheduling algorithms. The numerator is the total redundant computation cost brought by all $dp(v_i)$. It is obvious that with the same makespan, the algorithm with lower RCR is more efficient.

$$RCR = \sum_{v_i \in V} w_{dp(v_i),p_j}/makespan(solution) \tag{13}$$

B. EXPERIMENTAL RESULTS

In this section, we use both synthetic benchmarks and real-world applications as two test graph sets. Several state-of-the-art heuristic algorithms presented above are used as baseline algorithms to evaluate the performance of DBEFT. The workload tasks are assumed to be scheduled on a virtual heterogeneous platform, which has 32 virtual processors and each processor has different processing frequency.

1) WORKLOAD GRAPH GENERATOR

To evaluate the performance of the heuristic algorithms, we first consider random graphs, which are generated using a DAG generation program available at https://github.com/frs69wq/daggen. The generator requires the following input parameters.

- n: the number of tasks in the DAG
- *width*: the width of the DAG, which is the maximum number of tasks that can be executed concurrently. A small value will lead to a thin DAG (e.g., chain) with low task parallelism.
- *density*: it determines the number of edges between two levels of the DAG, with a low value leading to fewer edges and a large value representing a multitude of edges.
- *regularity*: it determines the uniformity of the number of tasks in each level. A high value indicates that all levels contain similar numbers of tasks.
- *jump*: the maximum number of levels spanned by inter-task communications, which means that an edge can go from level h to level $h+jump$. This allows to generate the DAGs with various execution paths of different lengths.

When generating the DAGs, the computation cost and communication cost are created according to the following two parameters: communication-to-computation (CCR) and β.

- CCR: the ratio of the sum of the edge weights to the sum of the node weights in a DAG.
- β: the heterogeneity factor for processor speeds. A high β value implies higher heterogeneity and bigger variance of computation cost on different processors. The average computation cost $\overline{w_i}$ of a task v_i in a given DAG is selected randomly from a uniform distribution in range $[0, 2w_{DAG}]$, where w_{DAG} is the average computation cost of a random graph. The computation cost of each task v_i on each processor p_j is randomly set in the following range:

$$\overline{w_i} \times (1 - \frac{\beta}{2}) \leq w_{i,j} \leq \overline{w_i} \times (1 + \frac{\beta}{2}) \tag{14}$$

For DAG generation, we considered the following parameters in our experiments.

$n = [10; 20; 30; 40; 50; 60; 70; 80; 90; 100, 200, 300, 400]$;
$fat = [0.1; 0.4; 0.8]$
$density = [0.2; 0.5; 0.8]$
$regularity = [0.2; 0.5; 0.8]$
$jump = [1; 2; 4]$
$processors = [4; 8; 16; 32]$
$\beta = [0.1; 0.2; 0.4; 0.8; 1.0; 2.0]$
$CCR = [0.1; 0{:}5; 1.0; 2.0; 5.0; 10.0]$;

These combinations produce 151,632 different DAGs. For each DAG, 5 different random graphs are generated with the same structure but with different weights of the edges and nodes. Thus, there are 758,160 random DAGs in the experiments.

Figure 3 shows the average SLR on graphs of different sizes for all algorithms. It can be seen that DBEFT is the best algorithm. On average, DBEFT outperforms CEFT by 3% to 15%, PEFT by 15% to 30%, HEFT by 17% to 33%.

Figure 4 shows the average AIT on different graphs for all algorithms. DBEFT has the lowest idle time of processors among all the algorithms. As the number of tasks in DAGs increases to 400, the idle time of DBEFT is less than one-third of the other algorithms. Figure 3 and Figure 4 indicate that DBEFT achieved the highest performance by acquiring the highest processor utilization.

Figure 5 demonstrates the speedup with respect to the number of processors for all the algorithms. It can be seen that DBEFT has the best performance. Also, as the number of processors increases, DBEFT has a higher speedup, which implies that DBEFT has better robustness in terms of the number of processors than CEFT, PEFT, and HEFT.

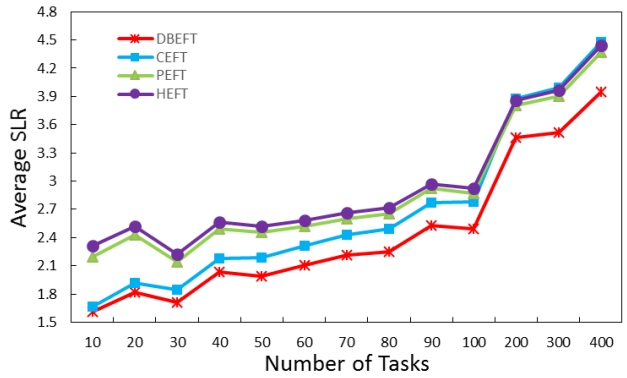


FIGURE 3. Average SLR under graph size. The x-coordinate is the number of tasks, while the y-coordinate is the average of SLR for all experiments. The lower the SLR means better algorithms.

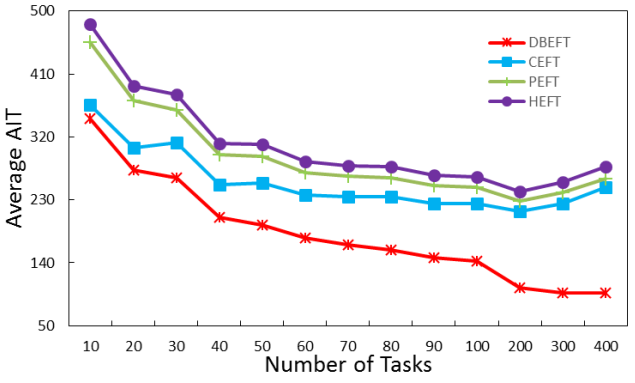


FIGURE 4. Average AIT under graph size. The x-coordinate is the number of tasks, while the y-coordinate is the average of AIT for all experiments. The lower AIT means the higher utilization of processors.

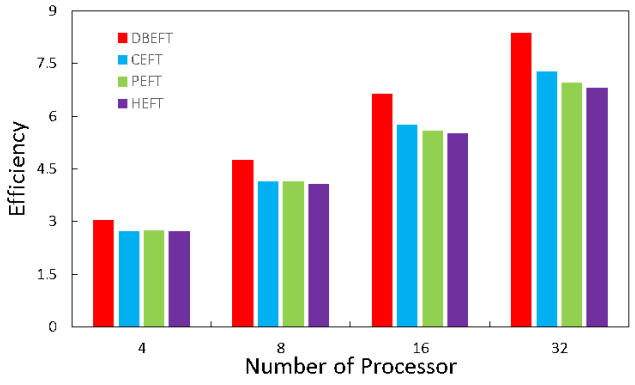


FIGURE 5. Speedup with respect to the number of processors. The x-coordinate is the number of processors, while the y-coordinate is speedup ratio to the makespan of the serialized version.

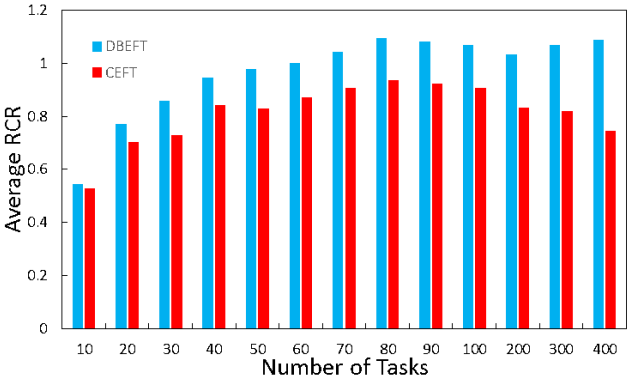


FIGURE 6. Average RCR under graph size. The x-coordinate is the number of tasks, while the y-coordinate is average RCR. The DBEFT has higher RCR comparing to CEFT because DBEFT generates more duplicated tasks.

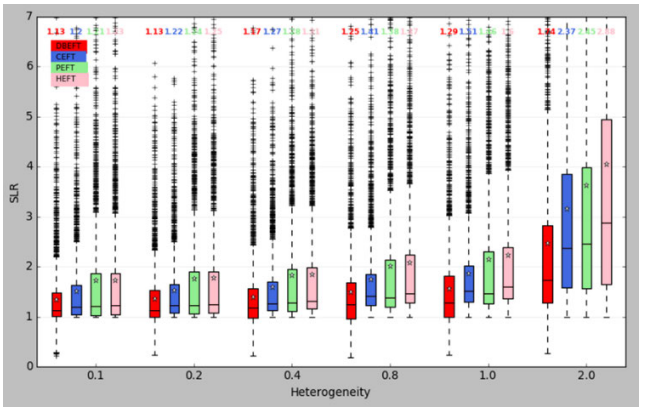


FIGURE 7. Box-plot of SLR under heterogeneity. The y-coordinate is average SLR, while the x-coordinate is the heterogeneity, which means the difference when tasks running on different processors.

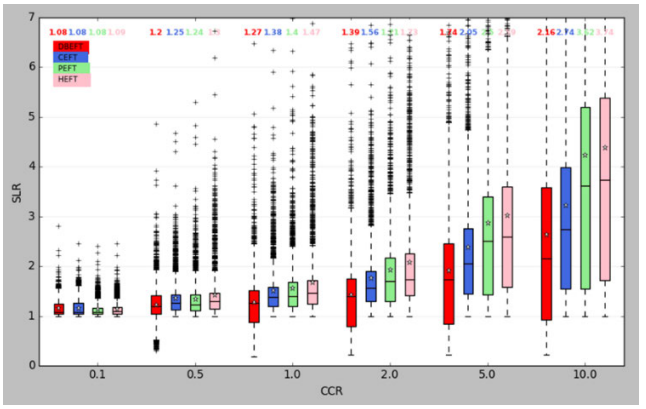


FIGURE 8. SLR under CCR. The y-coordinate is average SLR, while the x-coordinate is the CCR. Higher CCR means more communication cost comparing to the computation.

Average RCR is given in Figure 6. DBEFT has higher an RCR than CEFT because DBEFT tends to duplicate the immediate predecessor of a task, if there is any chance to improve the start time of the task. Also, the duplicated tasks has higher priorities than other ready tasks. This implies DBEFT may need to duplicate more tasks than CEFT.

Figure 7, Figure 8, Figure 9, and Figure 10 show the box-plot of SLR for all the algorithms in respect of heterogeneity, CCR and density. The border of the box shows the third quartile and the first quartile of the SLR distribution, while the horizon line in the box indicates the median of SLRs in each experiment. The smaller box means the algorithm has stabler performance. The DBEFT is the best algorithm with

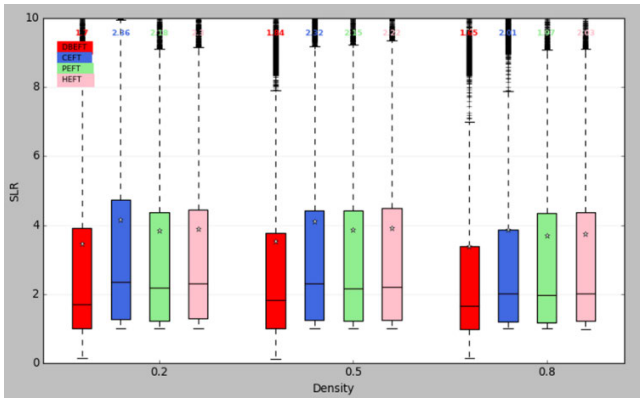


FIGURE 9. SLR under density. The y-coordinate is average SLR, while the x-coordinate is the density. Higher density means more task dependence among tasks.

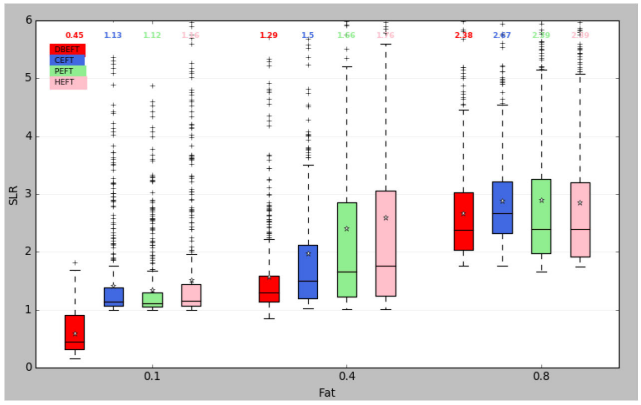


FIGURE 10. SLR under width. The y-coordinate is average SLR, while the x-coordinate is the width. Higher width means more parallel tasks in the DAT.

the lowest dispersion of SLR in the distribution of the results. In terms of heterogeneity, the bigger heterogeneity means the bigger difference of run time on different processors, which means more difficult to schedule the tasks. The DBEFT gets better performance as heterogeneity increase because DBEFT can overlap the communication with computation. In terms of CCR, the bigger CCR means more communication cost. Every algorithm slows down as the communication cost increase. The DBEFT outperforms other algorithms because of the task bundle strategy and task duplication mechanism, which can further hide the communication cost. In terms of density, bigger density means more edges in the DAGs, which means more dependence among tasks. DBEFT has the lowest average SLR and smallest dispersion with the increment of density, which implies DBEFT can achieve good performance in DAGs with more dependencies between tasks. It is because DBEFT stress the communication cost elimination between tasks. In terms of width, the wider the DAG is, the more parallel tasks there are, which lead to better performance, while in this situation, DBEFT still outperforms other algorithms.

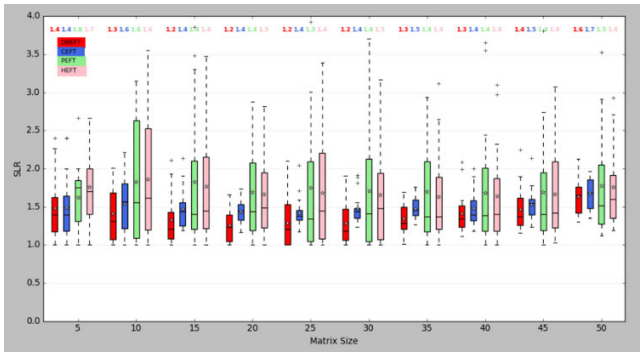


FIGURE 11. SLR under matrix size for gaussian elimination.

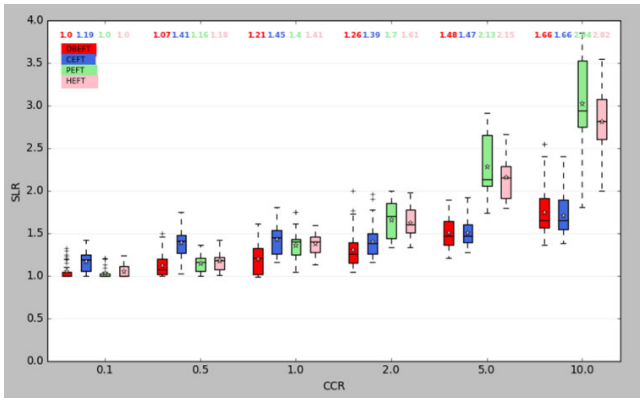


FIGURE 12. SLR under CCR for gaussian elimination.

2) REAL-WORLD APPLICATION GRAPHS

We use two real world applications to evaluate DBEFT, which are Gaussian Elimination and Fast Fourier Transform (FFT). The DAGs are built based on the traditional implication of the two applications.

We use a parameter m to denote matrix size on each dimension for Gaussian Elimination [43]. The total number of tasks in a Gaussian Elimination graph is $(m^2 + m - 2)/2$. The values considered for m are [5;10;15;20; 25;30;35;40;45;50] and CPU number is in the range of {4,8,16,32,64}.

The box plots of SLR respect to the matrix size, CCR and the number of processors are shown in Figure 11, Figure 12 and Figure 13 respectively. The border of the box shows the third quartile and the first quartile of the SLR distribution, while the horizon line in the box indicates the median of SLRs in each experiment. Figure 11 indicates DBEFT produces lower dispersion in the distribution of SLR for all matrix sizes. Figure 12 indicates DBEFT has the lowest SLR for CCR up to 5.0. When CCR is greater than 5.0, DBEFT has similar result as CEFT. Figure 13 demonstrates DBEFT has the best performance in terms of the number of processors, and as the number increases, DBEFT produces lower SLR. Therefore, DBEFT algorithm has the best robustness in Gaussian Elimination.

In the DAG generated for FFT, all the tasks are critical tasks since all paths are critical paths, and the communication

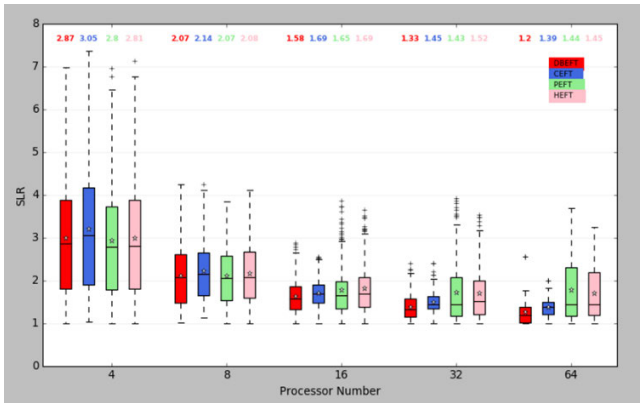


FIGURE 13. SLR under processor number for gaussian elimination.

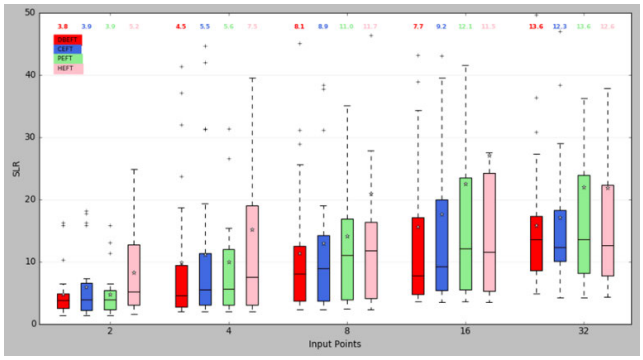


FIGURE 14. SLR under input points for fast fourier transform task.

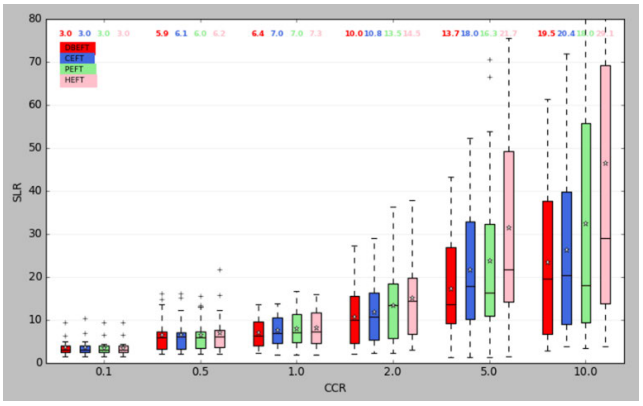


FIGURE 15. SLR under CCR for fast fourier transform task.

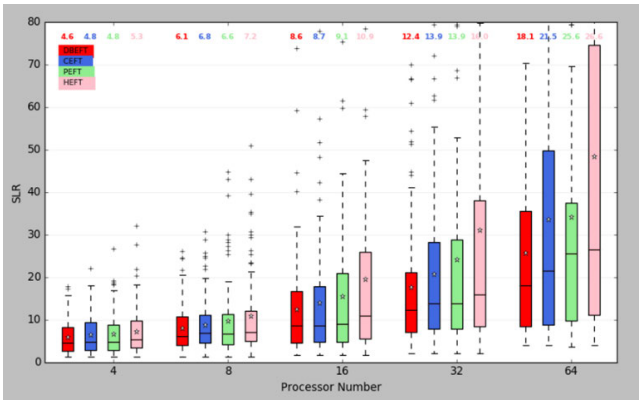


FIGURE 16. SLR under processor number for fast fourier transform task.

cost between two neighboring level tasks are the same. As mentioned in [20], we can separate the FFT algorithm into two parts: recursive calls and the butterfly operations. The number of tasks depends on the number of point n. There are $2 * (n - 1) + 1$ recursive tasks and $nlog_2 n$ butterfly operation tasks. The number of tasks are 5, 15, 39, 95, 223 with n in the range of {2,4,8,16,32}. We use CCR in the range of {0.1,0.5,1,2,5,10} and CPU number in the range of {4,8,16,32,64}.

The box plots of SLR respect to the matrix size, CCR and the number of processors are shown in Figure 14, Figure 15 and Figure 16 respectively. Figure 14 indicates that DBEFT algorithm produces lower dispersion in the distribution of SLR. Different from Gaussian Elimination, Figure 15 indicates that DBEFT algorithm has the lowest SLR among all the algorithms with FFT task graph. Figure 16 demonstrates DBEFT algorithm has the best performance in terms of the number of processors, and as the number increases, DBEFT algorithm produces lower SLR. Therefore, DBEFT algorithm has the better scalability in FFT.

C. DISCUSSION

HEFT needs to calculate the $rank_u$ value for all tasks before task scheduling begins. Since it is impossible to know in advance which processor that each task will be dispatched to, HEFT uses the average execution time of tasks on all the processors instead of the task execution time. However, in heterogeneous systems, tasks have different computation times on processors with different capabilities. In DBEFT, the task dependency ratio was introduced to determine the task's priority, which is continually updated during task execution.

PEFT uses the average value of all processors' OCT as the task's priority, but without knowing the predecessors and successors will eventually be executed on which processor, it uses the average communication cost instead of the actual communication cost when calculating OCT. But in the actual scheduling process, if two tasks with dependency relationship are scheduled to the same processor, the communication cost between the two tasks can be ignored. After each scheduling, DBEFT calculate with the actual communication cost and computing cost of tasks which have been scheduled rather than their estimated average cost. At the same time, DBEFT uses the task replication strategy to reduce the effects of long communication cost.

CEFT has greatly reduced the communication delay between tasks by adjusting the task priority order and task duplication strategy, but the problem of finding the optimal parameters can not be ignored. DBEFT further reduces the delay caused by the long communication cost by using the bundling strategy based on CEFT. The parallelism degree

among tasks is also expanded. Compared with CEFT, DBEFT not only expands the parallelism degree among tasks, but also improves the utilization of computing resources in the system.

Compared with other algorithms, DBEFT can obtain better time performance. As the number of tasks increases, the advantage of DBEFT tends to be more obvious. This is because adjusting task priorities in other algorithms brings some delays in tasks with high $rank_u$ and low CR values, but these delays offset the benefits of changing task priorities. DBEFT can adopt the single-task replication strategy and overlap communication time and task computing time as much as possible, which maximizes the parallelism degree between tasks and reduces the idle latency of available processor resources. In addition, the bundling strategy adopted by DBEFT eliminates singular values of communication cost and allows the algorithm to have good robustness to the number of processors.

VI. CONCLUSION

In this paper, we solve the problem of scheduling task graphs on a heterogeneous system, which consists of a set of heterogeneous machines. We propose a performance effective algorithm with low time complexity, DBEFT, to minimize the makespan by reducing communication overhead and maximizing the resource utilization. DBEFT achieves high resource utilization by minimizing communication overhead and maximizing task parallelism. DBEFT sets up the priority of tasks according to TDR, a notion defined in the paper to expand the task parallelism at runtime, and minimizes communication overhead by overlapping computing and communication as much as possible. Task bundling strategy is also adopted to eliminate communication overhead caused by high communication costs between tasks. We have used both simulation benchmarks and real-world applications to evaluate the algorithm proposed and compared it with the state-of-the-art heuristic algorithms. Extensive experimental results demonstrate that in the best case our new heuristic algorithm DBEFT improves performance by 15% over CEFT, 30% over PEFT, 33% over HEFT in terms of SLR and Speedup respectively.

REFERENCES

[1] *TOP 10 Sites for June 2016*. Accessed: Jun. 2017. [Online]. Available: https://www.top500.org/lists/2016/06/

[2] *Daggen*. Accessed: Jul. 2017. [Online]. Available: https://github.com/frs69wq/daggen

[3] A. Agarwal and P. Kumar, "Economical duplication based task scheduling for heterogeneous and homogeneous computing systems," in *Proc. IEEE Int. Adv. Comput. Conf.*, Mar. 2009, pp. 87–93.

[4] I. Ahmad and Y.-K. Kwok, "On exploiting task duplication in parallel program scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 9, pp. 872–892, Sep. 1998.

[5] S. G. Ahmad, C. S. Liew, E. U. Munir, T. F. Ang, and S. U. Khan, "A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 87, pp. 80–90, Jan. 2016.

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

[7] ARM. *big.LITTLE Technology: The Future of Mobile*. Accessed: Jul. 2017. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[8] S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 6, pp. 533–544, Jun. 2003.

[9] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *Proc. 18th Euromicro Conf. Parallel, Distrib. Netw.-Based Process.*, Feb. 2010, pp. 27–34.

[10] C. Boeres, J. V. Filho, and V. E. Rebello, "A cluster-based strategy for scheduling task on heterogeneous processors," in *Proc. 16th Symp. Comput. Archit. High Perform. Comput.*, Oct. 2004, pp. 214–221.

[11] C.-W. Chang, J.-J. Chen, T.-W. Kuo, and H. Falk, "Real-time task scheduling on island-based multi-core platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 538–550, Feb. 2015.

[12] L. Tao, D. Qiankun, W. Yifeng, G. Xiaoli, and Y. Yulu, "Dual buffer rotation four-stage pipeline for CPU–GPU cooperative computing," *Soft Comput.*, vol. 23, no. 3, pp. 859–869, 2019.

[13] X. Chang, Z. Ma, M. Lin, Y. Yang, and A. G. Hauptmann, "Feature interaction augmented sparse learning for fast Kinect motion detection," *IEEE Trans. Image Process.*, vol. 26, no. 8, pp. 3911–3920, Aug. 2017.

[14] X. Chang, Z. Ma, Y. Yang, Z. Zeng, and A. G. Hauptmann, "Bi-level semantic representation analysis for multimedia event detection," *IEEE Trans. Cybern.*, vol. 47, no. 5, pp. 1180–1197, May 2017.

[15] X. Chang, F. Nie, S. Wang, Y. Yang, X. Zhou, and C. Zhang, "Compound rank-k projections for bilinear analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 7, pp. 1502–1513, Jul. 2016.

[16] X. Chang and Y. Yang, "Semisupervised feature analysis by mining correlations among multiple tasks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2294–2305, Oct. 2017.

[17] X. Chang, Y.-L. Yu, Y. Yang, and E. P. Xing, "Semantic pooling for complex event analysis in untrimmed videos," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 8, pp. 1617–1632, Aug. 2017.

[18] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 399–409, Apr. 2008.

[19] K. Deng, K. Ren, S. Liu, and J. Song, "DAG scheduling for heterogeneous systems using biogeography-based optimization," in *Proc. IEEE 21st Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2015, pp. 708–716.

[20] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *Proc. 19th Annu. ACM Symp. Parallel Algorithms Archit.*, Jun. 2007, pp. 280–288.

[21] D. Fang and L. Junzhou, "A heterogeneous dynamic critical path and duplication based task scheduling algorithm for pervasive computing," in *Proc. 2nd Int. Conf. Pervasive Comput. Appl.*, Jul. 2007, pp. 457–462.

[22] T. Hagras and J. Janecek, "A simple scheduling heuristic for heterogeneous computing environments," in *Proc. 2nd Int. Symp. Parallel Distrib. Comput.*, Oct. 2003, pp. 104–110.

[23] L. A. Hall, "Approximation algorithms for scheduling," in *Approximation Algorithms for NP-Hard Problems*. Boston, MA, USA: PWS Publishing, 1996, pp. 1–45.

[24] K. Hamedani, L. Liu, A. Rachad, J. Wu, and Y. Yi, "Reservoir computing meets smart grids: Attack detection using delayed feedback networks," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 734–743, Feb. 2018.

[25] T. Huang, T. Li, Q. Dong, K. Zhao, W. Ma, and Y. Yang, "Communication-aware task scheduling algorithm for heterogeneous computing," *Int. J. High Perform. Comput. Netw.*, vol. 10, nos. 4–5, pp. 298–309, 2017.

[26] D. Karger, C. Stein, and J. Wein, "Scheduling algorithms," in *Algorithms and Theory of Computation Handbook*. Boca Raton, FL, USA: CRC Press, 2010, p. 20.

[27] C.-S. Lin, C.-S. Lin, Y.-S. Lin, P.-A. Hsiung, and C. Shih, "Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems," *J. Syst. Archit.*, vol. 59, no. 10, pp. 1083–1094, 2013.

[28] J.-C. Liou and M. A. Palis, "An efficient task clustering heuristic for scheduling dags on multiprocessors," in *Proc. Workshop Resource Manage., Symp. Parallel Distrib. Process.*, 1996, pp. 152–156.

[29] C.-H. Liu, C.-F. Li, K.-C. Lai, and C.-C. Wu, "A dynamic critical path duplication task scheduling algorithm for distributed heterogeneous computing systems," in *Proc. 12th Int. Conf. Parallel Distrib. Syst.*, vol. 1, Jul. 2006, p. 8.

[30] F. Lotfifar and H. S. Shahhoseini, "A low-complexity task scheduling algorithm for heterogeneous computing systems," in *Proc. 3rd Asia Int. Conf. Modelling Simulation*, May 2009, pp. 596–601.
[31] P. K. Mishra, A. Mishra, K. S. Mishra, and A. K. Tripathi, "Benchmarking the clustering algorithms for multiprocessor environments using dynamic priority of modules," *Appl. Math. Model.*, vol. 36, no. 12, pp. 6243–6263, 2012.
[32] H.-J. Park and B. K. Kim, "Optimal task scheduling algorithm for cyclic synchronous tasks in general multiprocessor networks," *J. Parallel Distrib. Comput.*, vol. 65, no. 3, pp. 261–274, 2005.
[33] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Parallel real-time scheduling of DAGs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 12, pp. 3242–3252, Dec. 2014.
[34] K. Shin, M. Cha, M.-S. Jang, J. Jung, W. Yoon, and S. Choi, "Task scheduling algorithm using minimized duplications in homogeneous systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 8, pp. 1146–1156, 2008.
[35] S. Song, K. Hwang, and Y.-K. Kwok, "Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 703–719, Jun. 2006.
[36] X. Tang, K. Li, G. Liao, and R. Li, "List scheduling with duplication for heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 70, no. 4, pp. 323–329, Apr. 2010.
[37] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[38] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.
[39] G. Wang, H. Guo, and Y. Wang, "A novel heterogeneous scheduling algorithm with improved task priority," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun., IEEE 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, Aug. 2015, pp. 1826–1831.
[40] Y. Wang, K. Li, and K. Li, "Partition scheduling on heterogeneous multicore processors for multi-dimensional loops applications," *Int. J. Parallel Program.*, vol. 45, no. 4, pp. 827–852, 2017.
[41] Y. W. Wong, R. S. M. Goh, S.-H. Kuo, and M. Y. H. Low, "A tabu search for the heterogeneous dag scheduling problem," in *Proc. 15th Int. Conf. Parallel Distrib. Syst.*, Dec. 2009, pp. 663–670.
[42] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Greening big data," *IEEE Syst. J.*, vol. 10, no. 3, pp. 873–887, Sep. 2016.
[43] M.-Y. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 3, pp. 330–343, Jul. 1990.
[44] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Inf. Sci.*, vol. 270, pp. 255–287, Jun. 2014.
[45] C.-H. Yang, P. Lee, and Y.-C. Chung, "Improving static task scheduling in heterogeneous and homogeneous computing systems," in *Proc. Int. Conf. Parallel Process.*, Sep. 2007, p. 45.
[46] Y. Yang, X. Lu, H. Jin, and X. Liao, "A stochastic task scheduling algorithm based on importance-ratio of makespan to energy for heterogeneous parallel systems," in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun., IEEE 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, Aug. 2015, pp. 390–396.
[47] 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 Trans. Comput.*, vol. 60, no. 3, pp. 360–374, Mar. 2011.

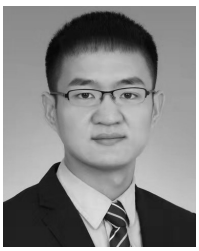
TAO LI received the Ph.D. degree in computer science from Nankai University, China, in 2007. He was with the College of Computer Science, Nankai University, as a Professor. He is a member of the IEEE Computer Society and the ACM, and the distinguished member of the CCF. His main research interests include heterogeneous computing, machine learning, and the Internet of Things.

DINGYUAN CAO received the B.E. degree from the College of Computer and Control Engineering, Nankai University, Tianjin, China, in 2018, where she is currently pursuing the M.S. degree with the College of Computer Science. Her research interests include machine learning and text categorization.

YE LU received the B.S. and Ph.D. degrees from Nankai University, Tianjin, China, in 2010 and 2015, respectively. He is currently an Associate Professor with Nankai University. His main research interests include embedded systems, the Internet of Things, and artificial intelligence.

TEHUI HUANG recevied the B.S. degree from the Department of Information and Computing Science, Anhui University of Science and Technology, in 2012, and the M.S. degree from the Department of Computer Science, Nankai University, in 2016. She is currently a Software Engineer with Blot Info & Tech (Beijing) Company, Ltd.

CHENGJUN SUN received the B.E. and M.S. degrees from the College of Computer and Control Engineering, Nankai University, Tianjin, China. Her research interests include high-performance computing and human–computer interaction.



QIANKUN DONG received the B.S. and M.S. degrees from Nankai University, Tianjin, in 2012 and 2016, respectively. His research interests are computer architecture, heterogeneous computing, and parallel processing.

XIAOLI GONG was born in 1983. He is currently an Associate Professor with Nankai University. His main research interests include system virtualization, and embedded system design and optimization. He is a member of the ACM and China Computer Federation.