Minimizing Cost of Scheduling Tasks on Heterogeneous Multicore Embedded Systems

JING LIU, Hunan University and Wuhan University of Science and Technology
KENLI LI, Hunan University
DAKAI ZHU, The University of Texas at San Antonio
JIANJUN HAN, Huazhong University of Science and Technology
KEQIN LI, Hunan University and State University of New York

Cost savings are very critical in modern heterogeneous computing systems, especially in embedded systems. Task scheduling plays an important role in cost savings. In this article, we tackle the problem of scheduling tasks on heterogeneous multicore embedded systems with the constraints of time and resources for minimizing the total cost, while considering the communication overhead. This problem is NP-hard and we propose several heuristic techniques—*ISGG*, *RLD*, and *RLDG*—to address the problem. Experimental results show that the proposed algorithms significantly outperform the existing approaches in terms of cost savings.

CCS Concepts: • **Theory of computation → Scheduling algorithms**; • **Computer systems organization → Multicore architectures**; **Heterogeneous (hybrid) systems**; **Embedded systems**

Additional Key Words and Phrases: Graph grouping, heterogeneous multicore systems, task scheduling, time and resource constraints

ACM Reference Format:
Jing Liu, Kenli Li, Dakai Zhu, Jianjun Han, and Keqin Li. 2016. Minimizing cost of scheduling tasks on heterogeneous multicore embedded systems. ACM Trans. Embed. Comput. Syst. 16, 2, Article 36 (December 2016), 25 pages.
DOI: http://dx.doi.org/10.1145/2935749

1. INTRODUCTION

Heterogeneous multicore designs have been widely employed in various types of computing systems such as embedded and mobile devices, servers, and supercomputers. The reason is that heterogeneous designs can provide high performance and flexibility, and at the same time promise low-cost and power-efficient implementations

The research is partially supported by International Science & Technology Cooperation Program of China (2015DFA11240), the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124, 61472150), the Open Fundation of Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System (2016znss26C), and the Natural Science Foundation of Hubei Province (2015CFB335).
Authors' addresses: J. Liu, College of Information Science and Engineering, Hunan University, Changsha 410082, China, and College of Computer Science and Technology, Wuhan University of Science and Technology, and Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, Wuhan 430065, China; email: Idealer@126.com; K. Li (corresponding author), College of Information Science and Engineering, Hunan University, and National Supercomputing Center in Changsha, Changsha 410082, China; email: lkl@hnu.edu.cn; D. Zhu, Department of Computer Science, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, USA; email: dakai.zhu@utsa.edu; J. Han, Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China; email: jasonhan@hust.edu.cn; K. Li, Department of Computer Science, State University of New York, New Paltz, New York 12561, USA; email: lik@newpaltz.edu.

DOI: http://dx.doi.org/10.1145/2935749

[Yi et al. 2009]. Note that the majority of computing devices are embedded, and embedded applications usually have timing constraints. Research has shown that for such application domains, heterogeneous multiprocessor systems can deliver higher performance at a given energy budget than homogeneous multicore solutions [Ter Braak et al. 2010]. Thus, we focus on heterogeneous multicore embedded systems in this study. To meet the increasing demands for higher performance, the number of integrated cores on a single chip continues to increase [Kong et al. 2012; Kim et al. 2014]. Increasing core density in a chip leads to increasing costs, such as energy consumption, hardware cost, and electricity cost [Qiu and Sha 2009]. For example, the cost of electricity is between CNY400,000 and CNY600,000 a day in National Supercomputing Center in Guangzhou, China. The growing power causes more and more energy consumption, which results in increasingly high temperature [Shafique et al. 2015; Khdr et al. 2015; Mohaqeqi et al. 2014] and such increased temperature has a significant impact on multicore systems. It decreases transistor age, slows down signal transition speed, increases the rates of permanent failures and transient faults, and brings high cooling charge. Consequently, cost savings are important issues in multicore systems. This article concentrates on exploring scheduling algorithms to reduce cost in modern heterogeneous multicore embedded systems. Reducing time and saving cost are usually conflicting objectives. Therefore, how to save cost is a big challenge in modern heterogeneous multicore real-time systems.

Recent heterogeneous multicore architectures integrate cores of different types into a chip. Cores of different types differ from each other in computation capability and cost consumption. Generally, a heterogeneous multicore computing system is comprised of a set of clusters connected to communication links. Each cluster consists of a group of cores of the same type. Chip vendors have released several kinds of heterogeneous multicore clusters in 2014, such as Samsung Exynos 5422 and AllWinner A80 [Linder 2014]. Communication links among clusters may have different capacities and costs.

Task scheduling is to assign tasks of an application to processing cores and identify their execution order on the cores [Topcuoglu et al. 2002]. Different task scheduling schemes lead to different makespan and cost. The makespan considered in this article includes the computation time for executing tasks in a Directed Acyclic Graph (DAG) and the time used for exchanging data between tasks. The cost considered in this article is an abstract representation of various costs, such as energy consumption, money, etc. Therefore, task scheduling becomes one of the most important techniques for fully exploiting the potential of heterogeneous multicore systems.

Numerous techniques have been proposed to minimize total cost of computing systems in the past years [Wang and Yao 2011; Zhang et al. 2013; Ge et al. 2014; March et al. 2013; Arras et al. 2015; Huang et al. 2011; Qiu and Sha 2009; Zong et al. 2011]. Studies in Wang and Yao [2011], Zhang et al. [2013], Ge et al. [2014], and March et al. [2013] focus on periodic tasks. The study in Arras et al. [2015] considers memory constraints and aims at minimizing the makespan. The study in Huang et al. [2011] considers the reliability. The study in Qiu and Sha [2009] does not consider communication. The study in Zong et al. [2011] does not consider time constraints.

In this article, we study the problem of scheduling tasks represented by a DAG on heterogeneous multicore embedded systems while considering communication under limited time and resource. Hereafter, this problem is denoted by STCLTR. The goal is to find efficient task scheduling policies for all tasks in a dataflow graph with minimized total cost under limited time and execution resources. This work is an extension of our previous work [Liu et al. 2014], which does not consider scheduling. Since the problem addressed in Liu et al. [2014] is NP-hard and can be reduced to the STCLTR problem, the STCLTR problem is NP-hard as well. The problem in Liu et al. [2014] does not take resource constraint into account, and thus their proposed techniques are not applicable to the STCLTR problem.

To solve the STCLTR problem, we propose three static heuristic algorithms. First, we propose the *ISGG* algorithm to partition tasks of an application modeled by a DAG into several groups, aiming at reducing communication overhead. Second, we propose the *RLD* algorithm to solve the STCLTR problem. *RLD* computes a local deadline for each task and determines the assignment of each task by defined cost-time ratio. Finally, combining algorithms *ISGG* and *RLD*, we propose the *RLDG* algorithm to solve the STCLTR problem. *RLDG* first uses *ISGG* to divide all tasks into a specific number of groups, each group with a limited size. Then it adopts a method similar to *RLD* to schedule each task. All tasks within a group are required to be executed on cores in the same cluster.

Our main contributions are as follows.

—We propose the *ISGG* algorithm to partition all tasks of a DAG into several groups which still form a DAG. The number of tasks in each group is restricted to a constant.
—We propose a novel scheduling algorithm *RLD* to solve the STCLTR problem.
—We propose another scheduling algorithm *RLDG* to solve the STCLTR problem by combining *ISGG* and *RLD*. *RLDG* has higher time complexity than *RLD*, but it can generate better solutions.

We conduct extensive experiments on synthetic benchmarks with various characteristics and real benchmarks to test the effectiveness and efficiency of the proposed algorithms. Experimental results show that the proposed algorithms can greatly reduce cost compared with other existing techniques. For example, for synthetic benchmarks whose ratios of communication to computation (CCR) are 0.5 under the first configuration, *RLDG* can achieve 30.01% and 39.72% reductions in total system cost on average, compared to two well-known scheduling algorithms, *HEFT* [Topcuoglu et al. 2002] and *DBUS* [Bozdag et al. 2006], respectively.

The remainder of this article is organized as follows. Section 2 reviews the related work. Section 3 describes models and defines the STCLTR problem studied in this article. Section 4 gives a motivational example. Section 5 presents an improved safe group graph algorithm. Section 6 proposes two heuristic scheduling algorithms to solve the STCLTR problem. Section 7 evaluates and analyzes the proposed techniques by comparing with other existing approaches. Section 8 concludes this article.

2. RELATED WORK

Researchers have developed plenty of methods to reduce cost in various computing systems in the past decades.

Some focus on embedded systems. Studies in Wang and Yao [2011], Zhang et al. [2013], and March et al. [2013] address scheduling algorithms for period tasks in embedded systems. Ge et al. [2014] present a Reducing Context Switches Scheduling (*RCSS*) algorithm based on preemption thresholds scheduling for real-time embedded systems to decrease system energy consumption. They consider a set of independent periodic or sporadic tasks. A communication energy-aware task mapping heuristic is studied in Singh et al. [2010], however, task computation energy is not considered. Singh et al. [2016] present a novel runtime trace analysis strategy to rapidly identify the maximum throughput mapping to support a use-case while optimizing for throughput and resource usage. However, they do not consider time constraint. Qiu and Sha [2009] propose optimal algorithms for a tree-structural task model and heuristics for a general task model on heterogeneous embedded systems with hard/soft time constraints. They do not consider communication and aim at minimizing cost. Addtionally, Singh et al. [2013b] investigate numerous works on scheduling algorithms in embedded systems. However, the majority of works are based on homogeneous systems. For works based on heterogeneous systems, they differ from our work in one or more of

three aspects: architectures, constraints, and goals, so that the techniques proposed in these works are not suitable to solve the problem studied in our work.

Some are based on other computing systems. Zong et al. [2011] focus on scheduling parallel tasks on homogeneous clusters without considering time constraints. They propose two energy-aware duplication scheduling algorithms, *EAD* and *PEBD*, to balance schedule lengths and energy savings by judiciously replicating predecessors of a task if the duplication can aid in performance without degrading energy efficiency. Xian et al. [2007] present an energy-aware scheduling algorithm based on *EDF* for homogeneous multiprocessor systems that support the Dynamic Voltage and Frequency Scaling (DVFS) techniques with uncertain task execution time. They consider a set of independent, periodic, preemptive, and hard real-time tasks. Lee [2012] studies energy-efficient scheduling of independent and periodic real-time tasks on lightly loaded homogeneous multicore processors that contain more processing cores than running tasks. Han et al. [2015] consider the Voltage/Frequency Island (VFI)-based and DVFS-enabled multicore systems, and study both static and dynamic energy management schemes for real-time tasks. Liu et al. [2011] deal with scheduling parallel applications on heterogeneous clusters, considering no time constraints. They propose an Efficient Energy-based Task Clustering Scheduling (*EETCS*) algorithm that conserves power by judiciously shrinking communication energy consumption. Mishra et al. [2003] propose static and dynamic power management schemes to schedule a set of real-time tasks with precedence constraints executing on distributed systems. They consider preemptive scheduling and aim to save energy. Seo et al. [2008] propose two heuristic algorithms, dynamic repartitioning and dynamic core scaling, to schedule periodic real-time tasks on multicore processors with the Dynamic Voltage Scaling (DVS) technique. Kong et al. [2011] develop algorithms to determine a schedule for independent and real-time tasks on cluster-based multicore systems under time and operating frequency constraints. Singh et al. [2013a] present a DVFS methodology for streaming applications that contain actors having cyclic dependencies. Gerards and Kuper [2013] present a schedule for independent, frame-based, real-time tasks that globally minimizes the energy consumption by applying DVFS and Dynamic Power Management (DPM). Chen et al. [2014] present an energy optimization technique for scheduling periodic real-time tasks on multicore systems with optimal DFVS and DPM combination.

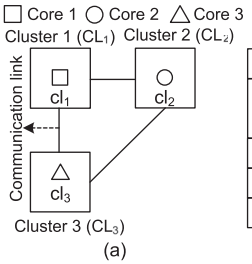
DVFS is a popular technique in reducing energy consumption. Studies [Wang et al. 2010; Xian et al. 2007; Gerards and Kuper 2013; Chen et al. 2014] all use DVFS to save energy. However, Shafique et al. [2013] point out that DVFS scaling potential is diminishing due to the shrinking gap between nominal and threshold voltages and the high overhead of voltage regulators in densely integrated chips with 100s of cores. Zong et al. [2011] also show that communication-intensive applications may decrease the benefits of DVFS. Thus, we do not consider the DVS, DVFS, or DPM techniques in our study. We address scheduling dependent tasks on heterogeneous multicore embedded systems considering communication, time, and resource constraints.

3. MODELS AND PROBLEM DEFINITION

In this section, we introduce the heterogeneous system model, the task model, and the problem to be studied in this article.

3.1. Heterogeneous System Model

The system model of heterogeneous multicore embedded systems adopted in this article is composed of a set of M connected, heterogeneous clusters, denoted by $CL = \{CL_1, CL_2, \ldots, CL_M\}$. These clusters are connected with each other through interconnects. Each cluster includes a finite number of cores, which means that tasks



(a)

Clusters	CL_1		CL_2		CL_3	
Cores	cl_1		cl_2		cl_3	
	UCT_1	UCC_1	UCT_2	UCC_2	UCT_3	UCC_3
cl_1	0	0	2	3	2	2
cl_2	2	3	0	0	3	3
cl_3	2	2	3	3	0	0

(b)

Fig. 1. An example of system model. (a) A system model with three heterogeneous clusters, each of which has finite cores. (b) Values of communication time and cost for transferring unit data volume between cores in (a).

will be executed with resource contention. Cores in the same cluster are identical, while cores in different clusters are heterogeneous. Communication between cores within a cluster passes through interconnects in the cluster. Communication between cores of different clusters passes through interconnects between clusters. Suppose that cluster CL_i contains n_i cores and cores in cluster CL_i are numbered as $cl_{S_{i-1}+1}, cl_{S_{i-1}+2}, \ldots, cl_{S_i}$, where S_i represents the total number of cores of clusters from CL_1 to CL_i, $S_i = S_{i-1} + n_i$, $S_0 = 0$, and $1 \leq i \leq M$. Then the set CL can be rewritten as $CL = \{cl_1, cl_2, \ldots, cl_{S_1}, cl_{S_1+1}, \ldots, cl_{S_2}, \ldots, cl_{S_{M-1}+1}, cl_{S_{M-1}+2}, \ldots, cl_{S_M}\}$. Figure 1(a) shows an example of the heterogeneous system model, which consists of three heterogeneous clusters.

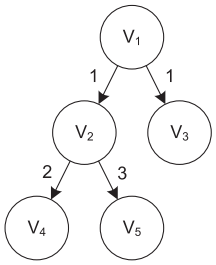
In general, an application consists of multiple tasks. If a task v_i needs the result computed by a task v_j ($j \neq i$) and both v_i and v_j are executed on two cores, communication is required between the two cores.

Different interconnects may present different communication capacities and propagation delays, which also leads to different communication overhead. We define the communication time for transmitting a unit of data through the interconnect from core cl_p to core cl_q as a function $UCT(cl_p, cl_q)$. We define the communication cost for transmitting a unit of data through the interconnect from core cl_p to core cl_q as a function $UCC(cl_p, cl_q)$. In view of communication between cores within a cluster via interconnects of the cluster and communication between cores of different clusters via interconnects among clusters, the communication cost for transmitting one unit of data between clusters is higher than that within a cluster and the communication time for transmitting one unit of data between clusters is longer than that within a cluster. The values of these two functions can be obtained by testing several sets of data sent through communication links. Thus, we assume that the communication time and cost for transmitting a unit of data are known in advance. Like other studies, we assume that that the communication cost and time are negligible between tasks executing on the same core.

Figure 1(b) shows an example of values of communication time and cost for transferring one unit of data between any two cores in Figure 1(a). For example, the value 3 in the cell of column "UCT_3" and row "cl_2" shows that the communication time transferring one unit of data from core cl_2 in cluster CL_2 to core cl_3 in cluster CL_3 is three time units.

3.2. Task Model

The target task model is a Directed Acyclic Dataflow Graph (DADFG). We use DADFG to model tasks of an application to be executed on the target system described in Section 3.1. A DADFG $G = \langle V, E, D \rangle$ is an edge-weighted DAG, where $V = \{v_1, v_2, \ldots, v_N\}$ is a set of nodes, $E \subseteq V \times V$ is a set of edges, and D is a set of edge



(a)

Clusters	CL_1		CL_2		CL_3	
Cores	cl_1		cl_2		cl_3	
Tasks	ET_1	EC_1	ET_2	EC_2	ET_3	EC_3
v_1	1	6	2	5	3	1
v_2	1	7	3	4	4	1
v_3	2	7	3	3	4	1
v_4	1	8	2	3	5	1
v_5	1	9	2	2	3	1

(b)

Fig. 2. An example of DADFG. (a) A tree. (b) Values of execution time and cost for tasks in (a) on the system model in Figure 1.

weights. Each node $v \in V$ represents a task. Each edge $(u, v) \in E$ represents the dependency relationship between node u and node v, indicating that task u should be finished before executing task v. We denote $D(v_i, v_j)$ to be the data volume transferred from task v_i to task v_j. A task without any parent is called an entry task and a task without any child is called an exit task. We assume that a DADFG G has only one entry node and one exit node. If there is more than one exit (entry) task, they can be connected to a zero-cost pseudo exit (entry) task with zero-cost edges. Figure 2(a) shows an example of a DADFG.

Due to the heterogeneity of cores, different types of cores have different computing capacities for a task and consume different cost. Let $ET(v_i, cl_p)$ be the execution time of task v_i when it is assigned to core cl_p. Let $EC(v_i, cl_p)$ be the execution cost of task v_i when it is assigned to core cl_p. Figure 2(b) shows the values of the execution time and cost of tasks in Figure 2(a) when these tasks are executed on the system model in Figure 1. For instance, the value located in column "ET_2" and row "v_3" indicates that the execution time of task v_3 is three time units when it is assigned to core cl_2 in cluster CL_2; the value located in column "EC_2" and row "v_3" indicates that the execution cost of task "v_3" is three units when it is assigned to core cl_2 in cluster CL_2.

Every task in V must be assigned to a core in the system model as described in Section 3.1. We define a task assignment function $A : V \longrightarrow CL$. For $\forall v \in V$, we have $A(v) \in CL$ which represents that task v is assigned to a core $A(v)$ that belongs to the set CL. For example, $A(v_1) = cl_3$ indicates that task v_1 is assigned to core cl_3 of the set CL.

Furthermore, we use functions $CT((u, v), (A(u), A(v)))$ and $CC((u, v), (A(u), A(v)))$ to represent communication time and communication cost on an edge (u, v) under the task assignment function A, respectively. Since we focus on static task scheduling, functions CT and CC can be computed in advance for all interconnects between cores. We assume that communication cost and time between tasks assigned to the same core are negligible. Techniques proposed in this article can be applied to a computing system as long as functions $CT((u, v), (A(u), A(v)))$ and $CC((u, v), (A(u), A(v)))$ are nonnegative and nondecreasing with an increasing number of tasks. In this work, we assume that communication time and cost through an edge are linearly proportional to the volume of data transferred on the edge, which is a reasonable assumption in practice. That is, function CT can be computed as $D(u, v) \times UCT(A(u), A(v))$ and function CC can be computed as $D(u, v) \times UCC(A(u), A(v))$.

After that, we define the completion time (i.e., makespan) and the total cost executing a DADFG $G = \langle V, E, D \rangle$ under the system model addressed in Section 3.1 and a task assignment function A. Since G is completed only if its exit node is finished, the completion time of G is defined to be the actual finish time of its exit node v_{exit} as shown

in Equation (1):

$$makespan = AFT(v_{exit}), \tag{1}$$

where $AFT(v_{exit})$ represents the actual finish time of v_{exit}. The total cost for executing G is the sum of the total execution cost of all nodes and the total communication cost for data communication. Denote the total cost as $C(G)_A$, which can be expressed as Equation (2):

$$C(G)_A = \sum_{v \in V} EC(v, A(v)) + \sum_{(u,v) \in E} CC((u, v), (A(u), A(v))). \tag{2}$$

3.3. Problem Definition

The problem is defined as follows: Given a DADFG $G = \langle V, E, D \rangle$, a system model consisting of a set of M heterogeneous clusters $CL = \{CL_1, CL_2, \ldots, CL_M\}$, each cluster CL_i $(1 \leq i \leq M)$ with n_i cores, and a time constraint L, the STCLTR problem is to find a task assignment function $A(v)$ for each task $v \in V$ and identify the execution order of all tasks on cores in clusters without violating task dependencies so that the total cost $C_A(G)$ is minimized, while the time constraint L is guaranteed, that is, $AFT(v_{exit}) \leq L$.

In this article, since the STCLTR problem is NP-hard, we propose heuristic algorithms to solve it. We propose a grouping algorithm *ISGG* to partition all tasks into a number of groups, and two scheduling algorithms *RLD* and *RLDG* to generate near optimal solutions for the STCLTR problem.

4. A MOTIVATIONAL EXAMPLE

Before presenting the proposed algorithms, we first show the effectiveness of our proposed algorithms *RLD* and *RLDG* through an example. We use the example to illustrate that task scheduling has an important impact on energy saving, assuming that the unit of time is second (s) and the unit of energy is joule (J).

In the example, there are a total of five tasks that form a DADFG as shown in Figure 2(a). These tasks are executed on the system model as shown in Figure 1. The system model consists of three heterogeneous clusters CL_1, CL_2, and CL_3. Figure 1(a) shows the structure of these three clusters and Figure 1(b) shows values of communication time and energy consumption for transferring a unit of data between cores of all clusters. We assume that the performance of cores in clusters CL_1, CL_2, and CL_3 is decreased and their power consumption is decreased in sequence. Figure 2(b) shows the values of execution time and energy consumption of all five tasks on the given system model. The value beside each edge is the data volume transferred on the edge.

Figure 3 shows four different scheduling schemes for five tasks executing on the given system model. In each scheduling scheme, the horizontal, the vertical axis, and a rounded rectangle represent time, core, and a task, respectively. The head of a rounded rectangle corresponds to the starting execution time of the task on its assigned core, and the tail of a rounded rectangle corresponds to the finish execution time of the task on its assigned core. A line with an arrow represents communication incurred by two tasks residing on different cores. For example, in Figure 3(d), the line with an arrow from the rounded rectangle labeled "v_1" to the rounded rectangle labeled "v_3" represents communication between core cl_2 and core cl_3 incurred by data dependency between task v_1 on core cl_2 and task v_3 on core cl_3. The head corresponds to the start time of communication and the tail corresponding to the finish time of communication.

Among four scheduling schemes, the scheme (a) is obtained from *HEFT* [Topcuoglu et al. 2002], where tasks v_1, v_2, v_3, v_4, v_5 are executed in order on core cl_1. The energy consumed by the scheme (a) is $(6 + 7 + 7 + 8 + 9)$J $= 37$J, and the completion time is 6s. The scheme (b) is obtained from *DBUS* [Bozdag et al. 2006], consuming time of 5s

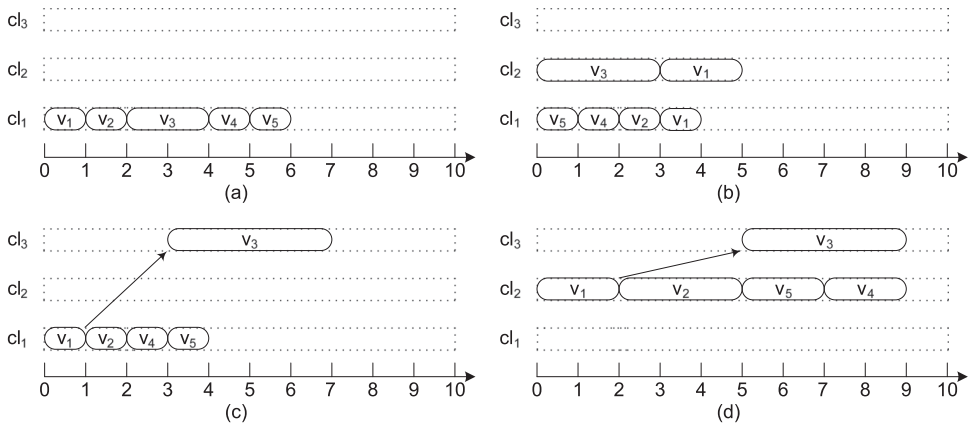


Fig. 3. Four different scheduling schemes. (a) A schedule consuming 37J of energy and 6s of time; (b) a schedule consuming 38J of energy and 5s of time; (c) a schedule consuming 33J of energy and 7s of time; (d) a schedule consuming 18J of energy and 9s of time.

and energy of 38J, respectively. The scheme (c) is obtained from *RLD* to be proposed, consuming time of 7s and energy of 33J. The scheme (d) is obtained from *RLDG* to be proposed, consuming time of 9s and energy of 18J.

This example reveals that different scheduling schemes will generate different results and it is necessary to explore efficient scheduling algorithms for the STCLTR problem.

5. AN IMPROVED SAFE GRAPH GROUPING ALGORITHM

In this section, we present an improved safe graph grouping algorithm, for example, *ISGG*. *ISGG* is a static heuristic and based on the Safe Graph Grouping (*SGG*) algorithm proposed by Sun et al. [2014]. The differences lie in that we consider limited resource of processing cores and restrict the number of tasks in each group, while there are no such restrictions for *SGG*. The basic idea of *ISGG* is to choose a pair of nodes with the largest communication data size from the input DADFG to merge without forming any cycle. This operation is repeated on the newly formed DADFG after every merger until the required DADFG is obtained or no nodes can be merged.

Before presenting *ISGG*, we introduce two binary operations $\vee$ and $\odot$, three matrices bA, bA^x, and $bA^{\geq 2}$, and a theorem.

Definition 5.1 (*Boolean Join* $\vee$). Given two Boolean matrices $A = (a_{ij})_{n\times n}$ and $B = (b_{ij})_{n\times n}$, define $A \vee B = (c_{ij})_{n\times n}$ and $c_{ij} = a_{ij} \vee b_{ij}$. We have

$$c_{ij} = \begin{cases} 1, & \text{if } a_{ij} = 1 \text{ or } b_{ij} = 1, 1 \leq i, j \leq n \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Let $A_1 = (a^1_{ij})_{n\times n}$, $A_2 = (a^2_{ij})_{n\times n}, \ldots, A_n = (a^n_{ij})_{n\times n}$ be n Boolean matrices, and their Boolean join can be defined as $A_1 \vee A_2 \vee \cdots \vee A_n = (c_{ij})_{n\times n}$, where $c_{ij} = a^1_{ij} \vee a^2_{ij} \cdots \vee a^n_{ij}$, $a^k_{ij} \in A_k$, $1 \leq k \leq n$, $1 \leq i, j \leq n$. We have

$$c_{ij} = \begin{cases} 1, & \text{if } \exists k,\ s.t.\ a^k_{ij} = 1, 1 \leq k \leq n, 1 \leq i, j \leq n, \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Definition 5.2 (*Boolean Product* $\odot$). Given two boolean matrices $A = (a_{ij})_{n\times n}$ and $B = (b_{ij})_{n\times n}$, define $A \odot B = (c_{ij})_{n\times n}$ and $c_{ij} = (a_{i1} \odot b_{1j}) \vee (a_{i2} \odot b_{2j}) \vee \cdots \vee (a_{in} \odot b_{nj})$. We

have

$$c_{ij} = \begin{cases} 1, & \text{if } \exists k,\ s.t.\ a_{ik} = 1 \text{ and } b_{kj} = 1, 1 \le k \le n, 1 \le i, j \le n \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Definition 5.3 (*Boolean Adjacency Matrix bA*). Given a DADFG G that contains n nodes, the Boolean adjacency matrix bA of G is an $n \times n$ matrix, where the value of each element $a_{ij} \in bA$ is 0 or 1: $a_{ij} = 1$ if and only if there exists an edge from node v_i to node v_j, $1 \le i, j \le n$.

Definition 5.4 (*Boolean x Matrix* bA^x). Given a DADFG G that contains n nodes, the Boolean x matrix bA^x ($1 < x < n$) of G is an $n \times n$ matrix, where the value of each element $a^x_{ij} \in bA^x$ is 0 or 1: $a^x_{ij} = 1$ if and only if there exists at least one path of at least x edges from node v_i to node v_j, $1 \le i, j \le n$. We have

$$\begin{aligned} bA^x &= bA^{x-1} \odot bA, \\ a^x_{ij} &= \left(a^{x-1}_{i1} \odot a_{1j}\right) \vee \left(a^{x-1}_{i2} \odot a_{2j}\right) \vee \cdots \vee \left(a^{x-1}_{in} \odot a_{nj}\right), \\ & \qquad a^x_{ij} \in bA^x, a^{x-1}_{ik} \in bA^{x-1}, a_{kj} \in bA, 1 \le k \le n, 1 \le i, j \le n. \end{aligned} \tag{6}$$

Note that matrix bA^x defined here is different from that defined in Sun et al. [2014].

Definition 5.5 (*Boolean* ≥ 2 *Matrix* $bA^{\ge 2}$). Given a DADFG G that contains n nodes, the Boolean ≥ 2 matrix $bA^{\ge 2}$ of G is an $n \times n$ matrix, where the value of each element $a^{\ge 2}_{ij} \in bA^{\ge 2}$ is 0 or 1: $a^{\ge 2}_{ij} = 1$ if and only if there exists at least one path from node v_i to node v_j with at least two edges, $1 \le i, j \le n$. We have

$$\begin{aligned} bA^{\ge 2} &= bA^2 \vee bA^3 \vee \cdots \vee bA^{n-1}, \\ a^{\ge 2}_{ij} &= a^2_{ij} \vee a^3_{ij} \vee \cdots \vee a^{n-1}_{ij},\ a^x_{ij} \in bA^x, 1 < x < n, 1 \le i, j \le n. \end{aligned} \tag{7}$$

Theorem 5.6. *Given a DADFG G with n nodes and its Boolean ≥ 2 matrix $bA^{\ge 2}$, merge any pair of nodes v_i and v_j that satisfy $a^{\ge 2}_{ij} = 0$ ($a^{\ge 2}_{ij} \in bA^{\ge 2}$), and then the new graph G' generated by merging remains a DADFG with $n - 1$ nodes.*

According to Corollary 4.1 in Sun et al. [2014], Theorem 5.6 is true and it is used to guarantee that the new graph G' is a DAG. So G' is a DADFG. Note that a node in the original input graph is a task, while a node in the new graph generated after every merge is a group. A group does not represent a task and it maybe includes several tasks.

After that, we will explain how to compute the Boolean adjacency matrix bA' and the Boolean ≥ 2 matrix $bA'^{\ge 2}$ of the new graph G' generated by merging nodes v_i and v_j of graph G. Let v_r be the node generated by merging nodes v_i and v_j, where r is the smaller one of the two indexes i and j, denoted by $r = min\{i, j\}$. Merging can only change connections between node v_r and those nodes that node v_h connects to or is connected by, where h be the larger one of the two indexes i and j. For nodes that v_h connects to or is connected by, after merging, they either connect to v_r or are connected by v_r, which makes the length of path between some nodes change. There are only three cases that can make the longest length of path between nodes change (see Figure 4). So, we can obtain bA' of G' by the following three steps:

Step 1. $a'_{st} = a_{st}, a'_{st} \in bA', a_{st} \in bA, 1 \le s, t \le |V|, s \ne i, j$, and $t \ne i, j$.
Step 2. $a'_{rr} = 0, a'_{rr} \in bA'$.
Step 3. $a'_{rk} = a_{ik} \vee a_{jk}, a'_{kr} = a_{ki} \vee a_{kj}, a_{ik} \in bA, a_{jk} \in bA, a'_{rk} \in bA', a'_{kr} \in bA', 1 \le k \le |V|$ and $k \ne i, j$.

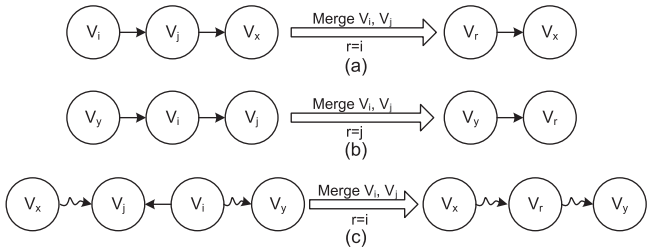


Fig. 4. Three kinds of subgraphs of G that will change values of elements of matrix $bA^{\geq 2}$ after merging nodes v_i and v_j.

ALGORITHM 1: ISGG

Input: A DADFG $G = \langle V, E, D \rangle$, bA and $bA^{\geq 2}$ of G, two positive integers k_1 and k_2.
Output: The target DADFG with total communication data size minimized.

1 regard each node v_i in G as a group, $1 \leq i \leq |V|$;
2 $|v_i| \leftarrow 1$, $1 \leq i \leq |V|$ /* $|v|$ represents the number of tasks in group v*/;
3 **repeat**
4 find a pair of groups v_i and v_j from G that meet $a_{ij}^{\geq 2} = 0$ and have the largest communication data size, $a_{ij}^{\geq 2} \in bA^{\geq 2}$;
5 **if** $|v_i| + |v_j| > k_2$ **then**
6 go to line 4 to find a new pair of groups;
7 **end**
8 $v_r \leftarrow v_i \cup v_j (r = min\{i, j\})$ /*merge v_i and v_j into a new group v_r */;
9 $|V| \leftarrow |V| - 1$; $|v_r| \leftarrow |v_i| + |v_j|$;
10 Apply the previously mentioned methods mentioned above to compute Boolean matrices bA' and $bA'^{\geq 2}$ of the new DADFG G' generated by merging;
11 $G \leftarrow G'$; $bA \leftarrow bA'$; $bA^{\geq 2} \leftarrow bA'^{\geq 2}$;
12 **until** $|V| \leq k_1$ *or no groups in* G *can be merged*;

Also, we can obtain $bA'^{\geq 2}$ of G' by the following five steps:

Step 1. $a_{st}'^{\geq 2} = a_{st}^{\geq 2}$, $a_{st}'^{\geq 2} \in bA'^{\geq 2}$, $a_{st}^{\geq 2} \in bA$, $1 \leq s, t \leq |V|$, and $s, t \neq h$.
Step 2. $a_{rr}'^{\geq 2} = 0, a_{rr}'^{\geq 2} \in bA'^{\geq 2}$.
Step 3. $a_{rx}'^{\geq 2} = 0$, if $a_{rx}'^{\geq 2} = 1$ and the longest path from nodes v_r to v_x with length less than 2, $\forall a_{rx}'^{\geq 2} \in bA'^{\geq 2}$ (see Figure 4(a)).
Step 4. $a_{yr}'^{\geq 2} = 0$, if $a_{yr}'^{\geq 2} = 1$ and the longest path from nodes v_y to v_r with length less than 2, $\forall a_{yr}'^{\geq 2} \in bA'^{\geq 2}$ (see Figure 4(b)).
Step 5. $a_{xy}'^{\geq 2} = 1$, if $a_{xy}'^{\geq 2} = 0$ and node v_x connects to node v_r and v_r connects to node v_y, $\forall a_{xy}'^{\geq 2} \in bA'^{\geq 2}$ (see Figure 4(c)).

Now, we present the *ISGG* algorithm, which is shown in Algorithm 1. First, *ISGG* regards each node v_i in the input graph G as a group and records the number of tasks in each group v_i, $1 \leq i \leq |V|$ (lines 1 and 2). Second, *ISGG* tries to find a pair of groups v_i and v_j that meet $a_{ij}^{\geq 2} = 0$ ($a_{ij}^{\geq 2} \in bA^{\geq 2}$) and have the largest communication data size (line 4). If the total number of tasks in groups v_i and v_j is larger than k_2, then *ISGG* goes to line 4 to find a new pair of groups; otherwise, group v_i and group v_j are merged into a new group denoted by v_r, where r is the smaller one of the two indexes i and j, that is, $r = min\{i, j\}$ (lines 5–8). Third, *ISGG* updates the total number of groups and the number of tasks in group v_r (line 9). Denote the new DADFG generated by merging

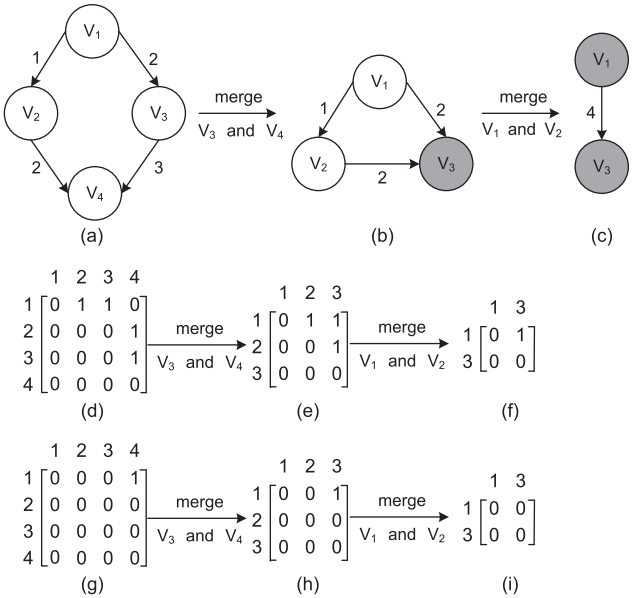


Fig. 5. An example of merging nodes. (a) DAG G. (b) DAG G' generated by merging nodes v_3 and v_4 of G. (c) DAG G'' generated by merging nodes v_1 and v_2 of G'. (d)–(f) are the Boolean adjacency matrices of G, G', and G'', respectively. (g)–(i) are the Boolean ≥ 2 matrices of G, G', and G'', respectively.

groups v_i and v_j to be G'; line 10 computes the Boolean adjacency matrix bA' and the Boolean ≥ 2 matrix $bA'^{\geq 2}$ of G' applying the methods mentioned previously. Line 11 updates graph G and its Boolean matrices bA and $bA^{\geq 2}$. Finally, operations from line 4 to line 11 are repeated until the total number of groups is not larger than k_1 or no groups can be merged.

Figure 5 shows an example to merge nodes by *ISGG*. The input graph $G = \langle V, E, D \rangle$ is shown in Figure 5(a) with four nodes, and let $k_1 = 2$ and $k_2 = 2$. Regard each node in G as a group, Figures 5(a)–5(c) show the new graphs generated after every merge, where the colorless circles represent the groups that do not finish partition, and the gray circles represent the groups that finish partition. Figures 5(d) and 5(g) show the Boolean adjacency matrices and the Boolean ≥ 2 matrices of the original input graph G. For each matrix, a column of numbers on the left and a row of numbers on the top of the matrix are both indexes of all groups, sorted by ascending order of indexes. For example, numbers "1, 2, 3, 4" on the left of the matrix shown in Figure 5(d) are indexes of groups v_1, v_2, v_3, v_4, respectively.

Observing Figures 5(a) and 5(g), we find that the communication data size between groups v_3 and v_4 are the largest among all pairs of groups, $a_{34}^{\geq 2} = 0$, and the total number of tasks in v_3 and v_4 is 2. Therefore, we merge v_3 and v_4, resulting in a new DADFG G' shown in Figure 5(b). Figures 5(e) and 5(h) show the Boolean adjacency matrix and the Boolean ≥ 2 matrix of G', respectively. We go on searching the next pair of groups and find that groups v_1 and v_2 satisfy the merge conditions. After merging v_1 and v_2, a second new DADFG G'' is generated shown in Figure 5(c). Its Boolean adjacency matrix is shown in Figure 5(f) and its Boolean ≥ 2 matrix is shown in Figure 5(i). G'' is the target DADFG, which contains two groups v_1 and v_3. Group v_1 contains two tasks v_1 and v_2; group v_3 contains two tasks v_3 and v_4. Compared with the total communication data size of eight data units before merging, the total communication data size after merging is greatly reduced to four data units.

The time complexity of finding two groups to merge is $O(|V|^2)$, the time complexity of computing bA' and $bA'^{\geq 2}$ is $O(|V|^2)$, and repetition at most occurs $|V|$ times, where $|V|$ is the number of nodes in the input DADFG. Thus, the time complexity of *ISGG* is $O(|V|^3)$, which is lower than the time complexity $O(|V|^4)$ of *SGG*.

6. TWO HEURISTIC SCHEDULING ALGORITHMS

In this section, we propose two heuristic scheduling algorithms for solving the STCLTR problem. One is the ratio and local deadline algorithm, for example, *RLD*. The other is the ratio and local deadline with grouping algorithm, for example, *RLDG*, which is based on *ISGG* and *RLD*.

Before presenting the two scheduling algorithms, we introduce some notations $EST(v_i, cl_p)$, $EFT(v_i, cl_p)$, $rank_u(v_i)$, $ld(v_i)$, and $ratio$ to be used in this section. Given a DADFG $G = \langle V, E, D\rangle$ consisting of n nodes $v_1, v_2, \ldots, v_n$, a time constraint L, and M heterogeneous clusters $CL_1, CL_2, \ldots, CL_M$, suppose that cluster CL_i contains n_i cores, $1 \leq i \leq n$. All cores are numbered by the method described in Section 3.1. We now address these notations one by one.

$EST(v_i, cl_p)$ represents the earliest execution start time of node v_i on core cl_p. It can be computed as follows:

$$EST(v_i, cl_p) = \begin{cases} 0, & \text{if } v_i = v_{entry} \\ \max\{avail[cl_p], \max_{v_m \in pred(v_i)}\{AFT(v_m) + & \\ CT((v_m, v_i), (A(v_m), cl_p))\}\}, & \text{if } v_i \neq v_{entry}, \end{cases} \tag{8}$$

where v_{entry} is the entry node of G and $pred(v_i)$ is the set of immediate predecessor nodes of v_i. $avail[cl_p]$ is the earliest available time for core cl_p to be available for executing a new node. If node v is the last assigned node on core cl_p, then $avail[cl_p]$ is the time that cl_p finishes the execution of v and it is ready to execute another node using a noninsertion-based scheduling policy. $AFT(v_i)$ is the actual finish time of v_i, computed as $AFT(v_i) = \min_{cl_p \in CL}\{EFT(v_i, cl_p)\}$. $EFT(v_i, cl_p)$ represents the earliest execution finish time of node v_i on core cl_p. It can be computed as follows:

$$EFT(v_i, cl_p) = ET(v_i, cl_p) + EST(v_i, cl_p). \tag{9}$$

$rank_u(v_i)$, an upward rank, is used to determine the scheduling order of node v_i, $1 \leq i \leq n$. It is computed recursively by traversing G upward starting from the exit node v_{exit}, and can be computed as follows:

$$rank_u(v_i) = \begin{cases} \overline{ET_i}, & \text{if } v_i = v_{exit} \\ \max_{v_j \in succ(v_i)}(\overline{CT((v_i, v_j), (A(v_i), A(v_j)))} + rank_u(v_j)) & \\ +\overline{ET_i}, & \text{if } v_i \neq v_{exit}, \end{cases} \tag{10}$$

$$\overline{ET_i} = \sum_{p=1}^{S_M} ET(v_i, cl_p)/S_M, \text{ where } S_M = \sum_{p=1}^{M} n_p, \tag{11}$$

$$\overline{CT((v_i, v_j), (A(v_i), A(v_j))} = \overline{UCT} \times D(v_i, v_j), \tag{12}$$

$$\overline{UCT} = \left(\sum_{p=1}^{S_M}\sum_{q=1}^{S_M} UCT(cl_p, cl_q)\right)/S_M^2, \tag{13}$$

where $succ(v_i)$ is the set of immediate successors of node v_i, $\overline{ET_i}$ is the average computation time of node v_i, $\overline{CT((v_i, v_j), (A(v_i), A(v_j)))}$ is the average communication time

between core $A(v_i)$ and core $A(v_j)$, and $\overline{UCT}$ is the average communication time for transferring a unit of data through all communication links. If $rank_u(v_i) > rank_u(v_j)$, v_i has higher priority than v_j and should be scheduled before v_j. We use the preceding method to calculate the priority because similar methods are adopted in other work and have been proved to be good [Topcuoglu et al. 2002].

$ld(v_i)$ $(1 \leq i \leq n)$, a local deadline of node v_i, can be obtained by four steps. First, we compute the upward local deadline denoted by $ld(v_i)_u$ of node v_i as follows:

$$ld(v_i)_u = \begin{cases} L, & \text{if } v_i = v_{exit} \\ \min_{v_j \in succ(v_i)}\{ld(v_j) - \overline{ET_j} - \overline{CT((v_i, v_j), (A(v_i), A(v_j)))}\}, & \text{if } v_i \neq v_{exit}. \end{cases} \quad (14)$$

It is computed recursively by traversing G upward starting from the exit node v_{exit}. One disadvantage of $ld(v_i)_u$ is that $ld(v_i)_u$ of some nodes are negative when L is small, which is invalid. Second, we compute a downward local deadline denoted by $ld(v_i)_d$ for node v_i as follows:

$$ld(v_i)_d = \begin{cases} \overline{ET_i}, & \text{if } v_i = v_{entry} \\ \max_{v_j \in pred(v_i)}\{ld(v_j) + \overline{ET_j}\}, & \text{if } v_i \neq v_{entry}. \end{cases} \quad (15)$$

It is computed recursively by traversing G downward starting from the entry node v_{entry}. Notice that $ld(v_i)_d$ may be larger than the time constraint L, which is also invalid. Third, we recompute a new downward local deadline denoted by $ld'(v_i)_d$ for v_i as follows:

$$ld'(v_i)_d = \frac{ld(v_i)_d \times L}{\max_{1 \leq j \leq n}\{ld(v_j)_d\}}. \quad (16)$$

Also, $ld'(v_i)_d$ has a disadvantage, that is, it may result in a makespan much shorter than the given time constraint L but a large cost. Finally, to avoid adverse factors mentioned in the second and third steps, we have

$$ld(v_i) = \max\{ld(v_i)_u, ld'(v_i)_d\}. \quad (17)$$

ratio, a cost-time ratio, is used to determine the assignment $A(v_i)$ of node v_i, $1 \leq i \leq n$. We compute the total cost and the earliest finish time of executing v_i on every core. Meanwhile, we record the core that makes the earliest finish time of v_i minimum among all cores under its local deadline as $A1(v_i)$, and the core that makes the total cost of v_i minimum among all cores under its local deadline as $A2(v_i)$. Suppose that the total cost and the earliest finish time of v_i executing on core $A1(v_i)$ are $cost(v_i, A1(v_i))$ and $eft(v_i, A1(v_i))$, and the total cost and the earliest finish time of v_i executing on core $A2(v_i)$ are $cost(v_i, A2(v_i))$ and $eft(v_i, A2(v_i))$. We have

$$ratio = -\frac{\Delta cost}{\Delta time} = \frac{cost(v_i, A1(v_i)) - cost(v_i, A2(v_i))}{eft(v_i, A2(v_i)) - eft(v_i, A1(v_i))}. \quad (18)$$

The *ratio* is compared with a nonnegative number $r1$ to determine whether node v_i is assigned to $A1(v_i)$ or $A2(v_i)$. If $ratio > r1$, then $A(v_i) = A2(v_i)$. Otherwise, $A(v_i) = A1(v_i)$.

Other notations used throughout the remainder of this article are listed in Table I.

6.1. The *RLD* Algorithm

In this subsection, we propose the *RLD* algorithm. The main idea of *RLD* is to start with computing a local deadline for each task, schedule the task under its local deadline, and identify the assignment of the task by a cost-time ratio. Then *RLD* changes values of some parameters to obtain better schedules. Finally, *RLD* chooses the schedule with the smallest cost under the given time constraint as the final schedule.

Algorithm 2 is our proposed *RLD* algorithm and shows how to get a near optimal schedule of an input DADFG G. At first, it calculates $rank_u$ values of all tasks of

Table I. Parameters to be Used

Parameter	Meaning
$\|v\|$	the number of tasks in group v
k_1	the number of groups to be partitioned for a given DADFG G
k_2	the number of tasks at most in each group
$r1$	a nonnegative real number that is used to determine the assignment of a task
γ	the relaxation factor of time constraint
$step$	an increment of time constraint before the first successful schedule is found
$step1$	an increment of time constraint after the first successful schedule is found
$A1(v)$	the assignment of task v that makes the earliest finish time of v minimum and satisfies $ld(v)$
$A2(v)$	the assignment of task v that makes the cost of finishing executing v minimum and satisfies $ld(v)$
$A(v)$	the assignment of task v
$EFT(g, CL_p)$	the earliest finish time of group g when executing on cluster CL_p
$cost(g, CL_p)$	the cost of group g when executing on cluster CL_p
$A1(g)$	the assignment of group g that makes the earliest finish time of g minimum
$A2(g)$	the assignment of group g that makes the cost of finishing executing g minimum
$A(g)$	the assignment of group g

ALGORITHM 2: *RLD*

Input: A DADFG $G = \langle V, E, D \rangle$, a time constraint L, and positive integers $step$ and $step1$.
Output: A near optimal schedule of G.

```
1  calculate rank_u(v) for each node v ∈ G by Equation (10);
2  sort all nodes in a scheduling list list by nonincreasing order of rank_u values;
3  L1 ← ∞; L' ← L; cost ← ∞; time ← ∞; S_min ← φ;
4  while L' ≤ L1 do
5      calculate ld(v) for each node v ∈ G by Equation (17);
6      call Function(RLDschedule(list, r1));
7      if (L1 < ∞) then
8          L' ← L' + step1;
9      else
10         L' ← L' + step (step > step1);
11     end
12 end
13 return S_min.
```

G by Equation (10), sorts all tasks in a scheduling list $list$ by nonincreasing order of $rank_u$ values, and initializes some related parameters (lines 1–3). Then, it uses a **while** loop to get as many as possible successful schedules of G by changing the value of the parameter L' (lines 4–12). The **while** loop firstly calculates the local deadline $ld(v)$ for each task v by Equation (17) under the time constraint L', then calls the function $Fuction(RLDschedule(list, r1))$ shown as in Algorithm 3 to try to find successful schedules. If the value of the parameter $L1$ is not ∞, which means that the first successful schedule is obtained, the increment of L' is reduced to a smaller constant $step1$ from a larger constant $step$. Finally, the returned schedule S_{min} is the solution.

Algorithm 3 shows how to obtain as many as possible successful schedules of G by changing values of the parameter $r1$. During finding schedules of G, if the first successful schedule is obtained, the value of the parameter $L1$ is fixed to be $(int)(\gamma \times L')$, where γ is the relaxation factor of the time parameter of L'. In addition, it records the schedule with the minimum cost among all successful schedules that meets the given time constraint L.

ALGORITHM 3: Function(function)

Input: A time constraint L and a positive integer $N1$.
Output: A schedule of DADFG G that satisfies the time constraint L.

```
r1 ← −0.05;
for j ← 0 to N1 do
    r1 ← r1 + α   /* α is a positive increment */;
    if function≠ −1 then
        record the successful schedule as S, and the corresponding cost and time of G as
          cost_S, time_S;
        if cost_S < cost and time_S ≤ L then
            cost ← cost_S; time ← time_S; S_min ← S;
        end
        if S is the first successful schedule then
            L1 ← (int)(γ × L′);
        end
    end
end
```

ALGORITHM 4: RLDschedule($list$, $r1$)

Input: A scheduling list $list$, a set of M heterogeneous clusters $CL = \{cl_1, cl_2, \ldots, cl_{S_M}\}$, a real number $r1$.
Output: A schedule of tasks in $list$.

```
cost_S ← 0;
repeat
    v ← the task with the largest rank_u value in list;
    A(v) ← ScheduleNode(v, 0, S_M, r1, cost_S);
    if A(v) = −1 then
        break;
    end
    cost_S ← cost(v, A(v)); list ← list − {v};
until list ← ∅;
return A(v).
```

Algorithm 4 shows how to obtain a schedule of G with a given value of the parameter $r1$. It picks out the task v with the largest $rank_u$ value from the scheduling list $list$, and calls the function $ScheduleNode(v, 0, S_M, r1, cost_S)$ shown as in Algorithm 5 to schedule task v. After scheduling task v, it takes off v from $list$ and selects another task with the largest $rank_u$ value from $list$ to schedule. This process is repeated until all tasks are tackled.

Algorithm 5 is a function and shows how to get the assignment of a task v. It starts with computing the earliest finished time $EFT(v, cl_q)$ of task v on every given core cl_q by using the insertion-based scheduling policy and the corresponding cost $cost(v, cl_q)$ of finishing executing v. The main idea of the **insertion-based scheduling policy**: Given a node v_i and a core cl_j, find a suitable time slot for v_i on cl_j; the start time of searching is the time when all data from the parent nodes of v_i arrived at cl_j; the searching is repeated until the first time slot that satisfies the execution time of v_i is obtained. Then, it records the assignment that makes the earliest finished time of v is minimal as $A1(v)$, and the assignment that makes the cost of finishing executing v is minimum as $A2(v)$, under the local deadline of v. Finally, it calls the function $Assignment(v, A1(v), A2(v), r1)$ as shown in Algorithm 6. $Assignment(v, A1(v), A2(v), r1)$ shows how to decide the assignment of a task (or group) v.

ALGORITHM 5: ScheduleNode(v, n_{total}, n_p, $r1$, $cost$)

Input: The local deadline $ld(v)$ of node v, the number n_p of cores in CL_p, some real numbers n_{total} , $r1$, $cost$.
Output: The assignment of v.

```
A2(v) ← −1; cost1 ← ∞; time ← ∞;
for q ← n_total + 1 to n_total + n_p do
    calculate EFT(v, cl_q) by Equation (9) using the insertion-based scheduling policy;
    if EFT(v, cl_q) < ld(v) then
        calculate cost(v, cl_q) ← cost + Σ_{u∈pred(v)} CC((u, v), (A(u), cl_q)) + EC(v, cl_q);
        if EFT(v, cl_q) < time then
            A1(v) ← cl_q; time ← EFT(v, cl_q);
        end
        if cost(v, cl_q) < cost1 then
            A2(v) ← cl_q; cost1 ← cost(v, cl_q);
        end
    end
end
call Assignment(v, A1(v), A2(v), r1);
return A2(v).
```

ALGORITHM 6: Assignment(v, $A1(v)$, $A2(v)$, $r1$)

Input: $cost(v, A1(v))$, $cost(v, A2(v))$, $EFT(v, A2(v))$, $EFT(v, A1(v))$ of a task (or group) v.
Output: The assignment of v.

```
if A2(v) = −1 then
    return
end
calculate ratio ← (cost(v,A1(v)) − cost(v,A2(v))) / (EFT(v,A2(v)) − EFT(v,A1(v))) according to Equation (18);
if ratio > r1 then
    A(v) ← A2(v);
else
    A(v) ← A1(v);
end
cost ← cost(v, A(v));
```

The time complexity of Algorithm 4 is $O(|V|^2 \times \sum_{p=1}^{M} n_p)$. Because $step > step1$,

$$\frac{\frac{L1}{1.1} - L}{step} + \frac{L1 - \frac{L1}{1.1}}{step1} < \frac{\frac{L1}{1.1} - L}{step1} + \frac{L1 - \frac{L1}{1.1}}{step1} = \frac{L1 - L}{step1}.$$

The time complexity of the *RLD* algorithm is $O(|N_1| \times \frac{L1-L}{step1} \times |V|^2 \times \sum_{p=1}^{M} n_p)$. Generally, we can set $|N_1| \times \frac{L1-L}{step1} < C$, where C is a constant. Thus, the time complexity of the *RLD* algorithm is $O(|V|^2 \times \sum_{p=1}^{M} n_p)$.

6.2. The RLDG Algorithm

In this subsection, combining the *ISGG* algorithm described in Section 5 and the *RLD* algorithm, we propose the *RLDG* algorithm. The basic idea of *RLDG* is similar to that of *RLD*. It firstly allocates a local deadline for each task, and partitions all tasks into a specified number of groups by *ISGG* to reduce communication overhead. Next, it schedules a group on each cluster by scheduling all the tasks in the group on cores in the cluster under their local deadlines, and determines the assignment of the

ALGORITHM 7: *RLDG*

Input: A DADFG $G = \langle V, E, D \rangle$, $CL = \{CL_1, CL_2, \ldots, CL_M\}$, the Boolean adjacency matrix bA of G, a time constraint L, and some positive integers $N1$, $N2$, $step$, and $step1$.
Output: A near optimal schedule of G.

```
1  calculate rank_u(v) by Equation (10), ∀v ∈ V;
2  compute the Boolean ≥ 2 matrix of G by the method mentioned in Section 5;
3  L1 ← ∞; L' ← L; cost ← ∞; time ← ∞; S_min ← φ;
4  while L' ≤ L1 do
5      calculate ld(v) by Equation (17), ∀v ∈ V;
6      k_1 ← n;
7      while k_1 ≥ 2 do
8          use Algorithm 1 to partition all tasks of graph G into k_1 groups, each group of at
           most k_2 = n/k_1 + 2 tasks, obtaining a new graph G';
9          calculate rank_u values of groups in graph G' by Equation (10), and sort these
           groups in a scheduling group list glist by nonincreasing order of their rank_u values;
10         call Function(ScheduleGroup(glist, r1));
11         k_1 ← k_1 − N2      /* N2 is a decrement */;
12     end
13     if (L1 < ∞) then
14         L' ← L' + step1;
15     else
16         L' ← L' + step (step > step1);
17     end
18 end
19 return S_min.
```

group by a cost-time ratio. Notice that the assignment of a group is a cluster, not a core. Then, it changes values of some parameters to obtain more successful schedules. Finally, it chooses the schedule with the smallest cost from these successful schedules that satisfy the time constraint as the final schedule.

Algorithm 7 is our proposed *RLDG* algorithm and shows how to get a near optimal schedule for an input DADFG G. At first, it calculates $rank_u$ values of all tasks of G by Equation (10), computes the Boolean ≥ 2 matrix of G by the method mentioned in Section 5, and initializes some related parameters (lines 1–3). Then, it uses an outer **while** loop to get as many as possible successful schedules of G by changing the value of the parameter L' (lines 4–18). The outer **while** loop first calculates the local deadline $ld(v)$ for each task v by Equation (17) under the time constraint L', then uses an inner **while** loop to obtain successful schedules by changing the values of the parameter of k_1. The inner **while** loop firstly uses the ISGG algorithm shown as in Algorithm 1 to partition all tasks of the input graph G into k_1 groups, each group of at most $k_2 = n/k_1 + 2$ tasks, obtaining a new graph G'. The inner **while** loop secondly calculates $rank_u$ values of all groups by Equation (10) and sorts these groups in a scheduling group list $glist$ by nonincreasing order of their $rank_u$ values. Thirdly, the inner **while** loop calls the function $Function(ScheduleGroup(glist, r1))$ shown as in Algorithm 3 to try to find successful schedules. If the value of the parameter $L1$ is not ∞, which means that the first successful schedule is obtained, the increment of L' is reduced to a smaller constant $step1$ from a larger constant $step$. Finally, the returned schedule S_{min} is the solution.

The function $Function(ScheduleGroup(glist, r1))$ uses the function $ScheduleGroup(glist, r1)$ shown as in Algorithm 8 to obtain assignments of all groups, where the assignment of a group is a cluster, not a core. Algorithm 8 firstly picks out the group g with the largest $rank_u$ value from the scheduling list $glist$ and initializes some related

ALGORITHM 8: ScheduleGroup($glist$, $r1$)

Input: A scheduling group list $glist$ and a real number $r1$.
Output: A schedule of tasks in $glist$ which minimizes the total cost and satisfies the time constraint L.

```
1  cost(g) ← 0;   /*cost(g) and time(g) represent the final cost and time of group g*/;
2  repeat
3      g ← the group with the largest rank_u value in glist;
4      A(g) ← −1; n_total ← 0; cost ← ∞; time ← ∞ ;
5      for p ← 1 to M do
6          i ← 1; cost(g, p) ← cost(g); EFT(g, p) ← 0;
7          repeat
8              v ← the task with the i^th largest rank_u value in the group g;
9              A(v) ← ScheduleNode(v, n_total, n_p, r1, cost(g, p));
10             if A(v) = −1 then
11                 break;
12             end
13             if EFT(g, p) < EFT(v, A(v)) then
14                 EFT(g, p) ← EFT(v, A(v));
15             end
16             cost(g, p) ← cost(v, A(v)); i ← i + 1;
17         until all tasks in the group g have been performed;
18         n_total ← n_total + n_p;
19         if A(v) = −1 then
20             continue;
21         end
22         if time > EFT(g, p) then
23             time ← EFT(g, p); A1(g) ← CL_p;
24         end
25         if cost > cost(g, p) then
26             cost ← cost(g, p); A2(g) ← CL_p;
27         end
28     end
29     call Assignment(g, A1(g), A2(g), r1);
30     glist ← glist − {g}; cost(g) = cost(g, A(g));
31 until glist ← ∅ or A(g) ← −1;
32 return A(g).
```

parameters. Next, it tries to schedule group g on every cluster CL_p by a **for** loop (lines 5–28). The **for** loop tries to schedule all tasks in g by the order of their $rank_u$ values on cores of cluster CL_p, and records the assignment that makes the earliest finished time of g is minimal as $A1(g)$, and the assignment that makes the cost of finishing executing g is minimum as $A2(g)$. Then, it calls the function $Assignment(g, A1(g), A2(g), r1)$ as shown in Algorithm 6 to determine the assignment of group g. After that, it deletes g from $glist$ and selects another group with the largest $rank_u$ value from $glist$ to schedule. Finally, this process is repeated until all groups are tackled.

The time complexity of Algorithm 7 is

$$O\left(\frac{L1-L}{step1}\times\frac{|V|-2}{N2}\left(|V|^3+N1\times|V|^2\times\sum_{p=1}^{M}n_p\right)\right)$$
$$=O\left(\frac{L1-L}{step1}\times\frac{|V|-2}{N2}\left(|V|^3+N1\times|V|^2\times\sum_{p=1}^{M}n_p\right)\right)$$

Table II. Values of Communication Time and Cost Transferring a Unit of Data Between Different Cores of Clusters, Assuming that the Communication Overhead on the Same Core is Zero

	CL_1		CL_2		CL_3		CL_4	
Clusters	UCT_1	UCC_1	UCT_1	UCC_1	UCT_1	UCC_1	UCT_1	UCC_1
CL_1	1	1	2	3	2	2	5	4
CL_2	2	3	1	1	4	5	3	2
CL_3	2	2	4	5	1	1	2	2
CL_4	5	4	3	2	2	2	1	1

Generally, $\sum_{p=1}^{M} n_p \leq |V|$, $\frac{L1-L}{N2\times step1}$, and $N1$ are bounded by constants. Thus, the time complexity of Algorithm 7 is $O(|V|^4 + |V|^3 \sum_{p=1}^{M} n_p)$.

7. EXPERIMENTAL RESULTS AND DISCUSSION

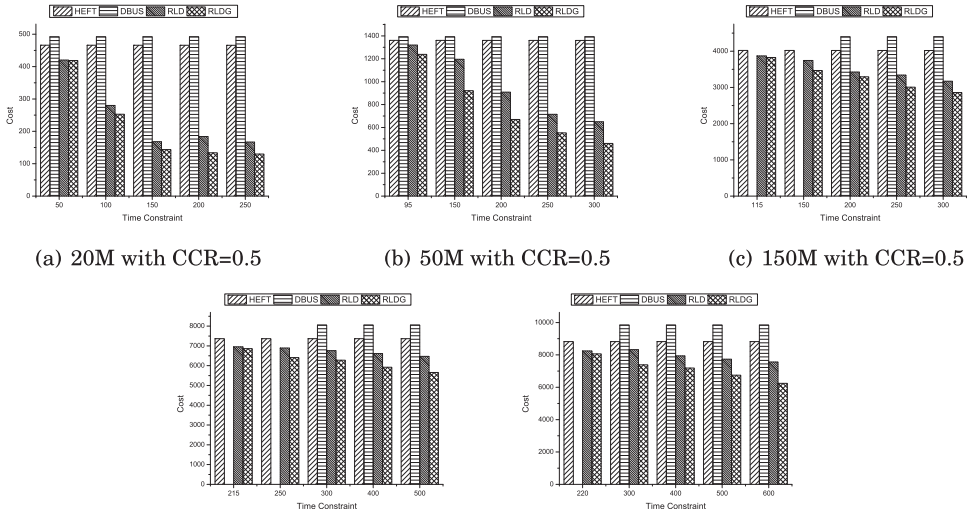
In this section, we evaluate the effectiveness and efficiency of our proposed techniques.

All experiments are conducted on a simulator running on a computer equipped with the 32-bit Windows 7 operating system and two identical processing cores: Intel (R) Core (TM) i5-2400 CPU @ 3.10GHz. The simulator is a C program, and it is used to simulate a system model described in Section 3.1, consisting of four connected heterogeneous clusters CL_1, CL_2, CL_3, and CL_4. We consider two configurations: (1) CL_1, CL_2, CL_3, and CL_4 are composed of six, four, three, and five processing cores, respectively; (2) CL_1, CL_2, CL_3, and CL_4 are composed of six, four, four, and four processing cores, respectively. Values of communication time and cost transferring a unit of data under two configurations are the same and as shown in Table II. We assume that the performance of cores in clusters CL_1, CL_2, CL_3, and CL_4 are decreased and their execution costs are decreased in sequence. That is, cores in cluster CL_1 have the highest computation capacity as well as the highest cost, and cores in clusters CL_2, CL_3, and CL_4 have slower computation capacity with lower cost.

In this study, we consider synthetic benchmarks and real benchmarks. Synthetic benchmarks are some DAGs randomly generated using TGFF [Dick et al. 1998]; real benchmarks come from the DSPstone benchmark suite [Wolf et al. 2008], including IIR, 4-Stage Lattice Filter, Differential Equation Solver, RSL-Languerre Lattice, and 20-4Stage Lattice Filter.

The DAGs are randomly generated with four varying parameters: (1) the number of nodes in a DAG; (2) the number of parents of a node, that is, the indegree of the node; (3) the number of children of a node, that is, the outdegree of the node; and (4) the communication to computation ratio, CCR, which is computed by the average communication time divided by the average computation time on a target system and is selected from set {0.2, 0.5, 1, 5, 10}. According to the indegree and outdegree of nodes in DAGs, we generated three kinds of DAGs: slim DAGs, medium DAGs, and fat DAGs. Each kind of DAG includes five DAGs, which contain 20, 50, 150, 250, and 320 nodes, respectively. The average indegree of nodes in a slim DAG is 1 or 2 and the average outdegree is 1, 2, or 3. Both the average indegree and outdegree of nodes in a fat DAG are $\sqrt{n}$, where n is the number of nodes in the fat DAG. The average indegree and outdegree of nodes in a medium DAG are determined by the following method. Given a slim DAG $G1$, a medium DAG $G2$, and a fat DAG $G3$, all these DAGs contain n nodes. Let the average indegree and outdegree of nodes in $G1$ be $n1$ and $n2$, and then the average indegree and outdegree of nodes in $G2$ are $n1 + \sqrt{n}/2$ and $n2 + \sqrt{n}/2$.

For a randomly generated DAG, each node is assigned two node weights, the execution cost and the execution time, both from interval $(0, 30]$ with a uniform probability distribution. Each edge is assigned an edge weight, the data size transferred on the edge, from interval $(0, 30 \times CCR/\overline{CT}]$ to approximate the desired CCR. For



(a) 20M with CCR=0.5 (b) 50M with CCR=0.5 (c) 150M with CCR=0.5

(d) 250M with CCR=0.5 (e) 320M with CCR=0.5

Fig. 6. Experimental results of five medium benchmarks with CCR = 0.5 under the first configuration. (a)–(e) shows total cost obtained from different algorithms.

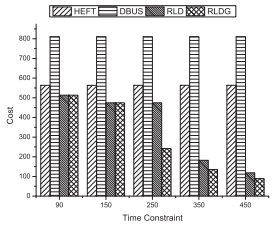
benchmarks from the DSPstone, values of execution time and cost are obtained from Shao et al. [2005]. The data size transmitted on edges is from interval $(0, 5]$ with a uniform probability distribution.

Lee and Zomaya [2011] studied energy conscious scheduling for distributed computing systems using DVS. They used *HEFT* [Topcuoglu et al. 2002] and *DBUS* [Bozdag et al. 2006] as their baselines for the reason "Although the scheduling of these previous algorithms is energy **unconscious**, they were proven to perform well for the task scheduling problem; in addition, none of the existing scheduling algorithms is directly applicable to such a problem." We met the same difficulty as them and their proposed algorithms are not suitable for our problem. So, inspired by them, we also use *HEFT* and *DBUS* as our baselines.

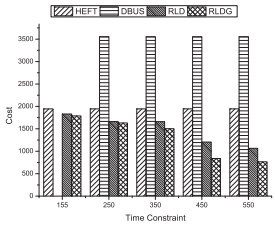
Considering that it is not convenient to list all experimental results and the nature of these results are more or less the same, we only show experimental results for Synthetic benchmarks of medium benchmarks with CCR = 0.5 and CCR = 5, and real benchmarks.

Figures 6 and 7 shows experimental results for five medium benchmarks with CCR = 0.5 and five medium benchmarks with CCR = 5 under the first configuration with varied time constraints. Figures 8 and 9 show experimental results for five medium benchmarks with CCR = 0.5 and five medium benchmarks with CCR = 5 under the second configuration with varied time constraints. For each subfigure, the horizontal axis represents the time constraint, and the vertical axis represents the cost for executing a given benchmark obtained by algorithms *HEFT*, *DBUS*, *RLD* ,and *RLDG* when time constraint varies. The name of a benchmark is composed of an integral value and a letter "M," where the integral value is the number of nodes in the benchmark and "M" represents that the benchmark is a medium DAG. Besides, in some subfigures, some algorithms cannot find solutions for the corresponding benchmarks under some time constraints, so the corresponding costs are not shown.

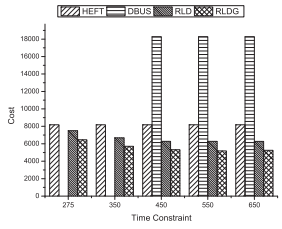
We observe that for each benchmark, the *HEFT* algorithm generates the same solution no matter how the time constraint varies. It is true for the *DBUS* algorithm. Whereas, when the time constraint changes, both *RLD* and *RLDG* can generate varied



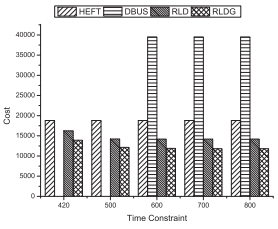
(a) 20M with CCR=5



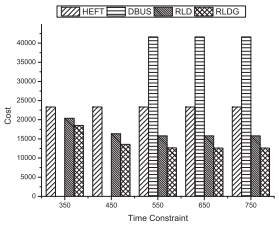
(b) 50M with CCR=5



(c) 150M with CCR=5

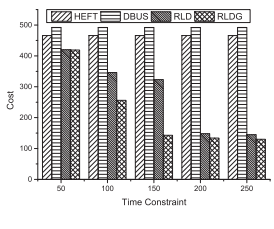


(d) 250M with CCR=5

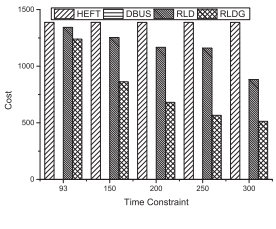


(e) 320M with CCR=5

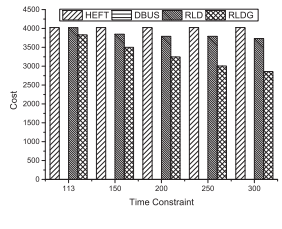
Fig. 7. Experimental results of five medium benchmarks with CCR = 5 under the first configuration. (a)–(e) shows total cost obtained from different algorithms.



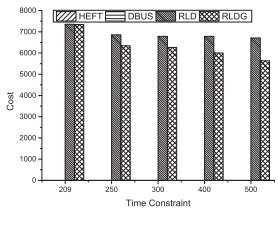
(a) 20M with CCR=0.5



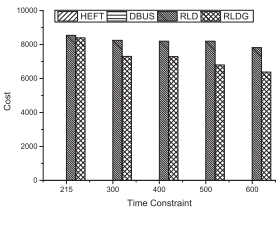
(b) 50M with CCR=0.5



(c) 150M with CCR=0.5



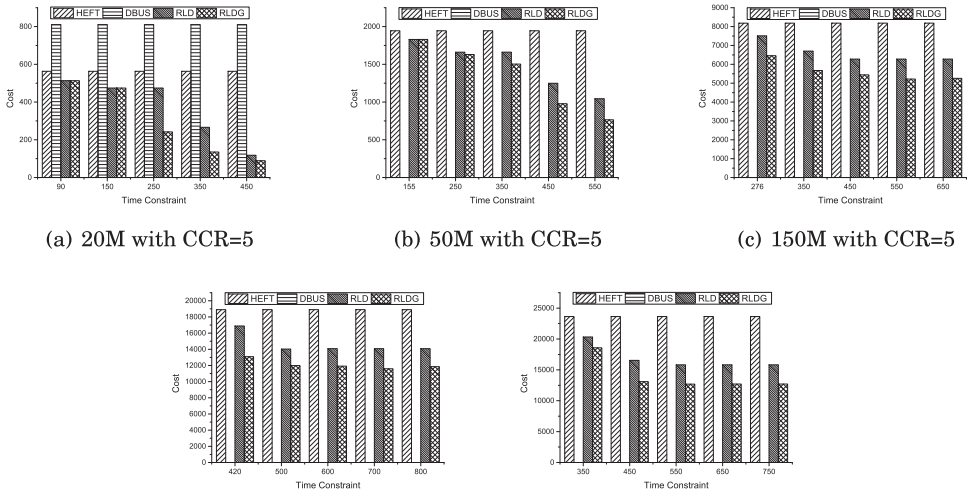
(d) 250M with CCR=0.5



(e) 320M with CCR=0.5

Fig. 8. Experimental results of five medium benchmarks with CCR = 0.5 under the second configuration. (a)–(e) shows total cost obtained from different algorithms.

solutions. Generally, when the time constraint increases, the total cost decreases. Moreover, both *RLD* and *RLDG* can generate better results than *HEFT* and *DBUS*. What is more, total costs produced by *RLDG* are usually less than that produced by *RLD*. For example, when the time constraint is 150, the total cost of benchmark "50M" with CCR = 0.5 in Figure 6 generated by algorithms *HEFT*, *DBUS*, *RLD*, and *RLDG* is 1361, 1392, 1196, and 921, respectively; when the time constraint is 250, the total cost is 1361, 1392, 715, and 553, respectively. Additionally, in some cases, all these four algorithms except for *DBUS* can obtain a solution when the time constraint is



(a) 20M with CCR=5 (b) 50M with CCR=5 (c) 150M with CCR=5

(d) 250M with CCR=5 (e) 320M with CCR=5

Fig. 9. Experimental results of five medium benchmarks with CCR = 5 under the second configuration. (a)–(e) shows total cost obtained from different algorithms.

small. For benchmark "50M" with CCR = 0.5 in Figure 6, *RLDG* can reduce total cost by 44.77% on average compared with *DBUS* under five given time constraints. On the whole, *RLDG* reduces total cost by 30.01% and 39.72% on average compared with *DBUS* and *HEFT*, respectively, for five benchmarks with CCR = 0.5.

In addition, for medium benchmarks with CCR = 5, more identical values appear, which is incurred by the large communication overhead. Once CCR is larger than 1, the completion time is dominated by the communication time. For CCR = 5, the data size between two dependent tasks can be up to 60 data units, and the communication time can be about 150 time units on average and up to 300 time units when these two tasks resided on different cores. If the assignment of one task of them is changed, a large amount of communication time will be produced such that the total time of the new schedule may exceed the given time constraint. If the increment of the time constraint is too small to change the assignments of some tasks in the original scheduling, then the original schedule remains unchanged. If the increment of the time constraint is enough large, then the total cost can be further reduced in general.

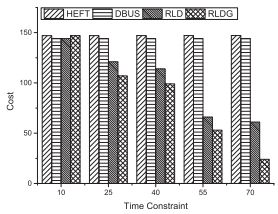
Obviously, the preceding four figures reflects similar information.

Figure 10 shows experimental results for five benchmarks from DSPstone benchmark suite under the first configuration. It reflects similar information as in Figures 6 and 8. Due to limit space, we do not show experimental results for five benchmarks from DSPstone benchmark suite under the second configuration.

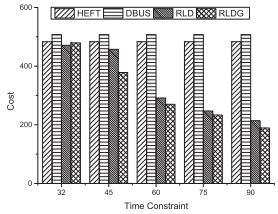
Sometimes *RLD* and *RLDG* can obtain better results than both *HEFT* and *DBUS* in terms of time and cost under these two configurations. Table III shows several examples. For example, *RLD* can find a solution with cost 6,994 and time 308 for benchmark "150S" under the first configuration, and *RLDG* can find a solution with cost 5,376 and time 246. *HEFT* and *DBUS* produce no solutions under time 310.

We also find that the experimental results are invariant when the value of the relaxation factor γ varies, like $\gamma = 1.05, 1.1$, and 2.

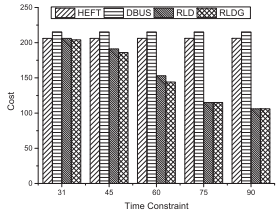
The preceding discussions demonstrate that the proposed algorithms *RLD* and *RLDG* are highly efficient. Both algorithms can take full advantage of the given time to reduce total costs as much as possible.



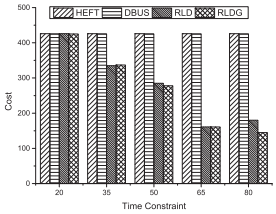
(a) IIR



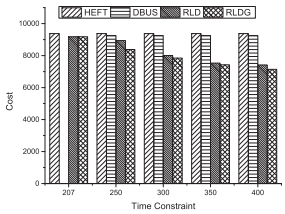
(b) 4-Lat-IIR



(c) Diff



(d) RLS



(e) 20-4Lat-IIR

Fig. 10. Experimental results of five benchmarks from DSPstone benchmark suite under the first configuration. (a)–(e) show the total cost obtained from different algorithms for these benchmarks.

Table III. Some Examples in which Both RLD and RLDG Excel HEFT and DBUS in Terms of Makespan and Total Cost Under Two Configurations

Configuration	CCR	Benchmark	HEFT Time	HEFT Cost	DBUS Time	DBUS Cost	RLD Time	RLD Cost	RLDG Time	RLDG Cost
First	0.5	50S	65	1,219	76	1,483	65	1,188	63	1,147
		250F	164	7,241	202	7,630	164	6,734	163	6,695
	10	150S	310	7,596	326	9,486	308	6,994	246	5,376
		320F	575	29,090	669	56,453	552	28,098	547	28,372
Second	0.5	150M	148	4,526	112	4,023	112	4,036	112	3,830
		250M	297	8,492	210	7,490	207	7,533	208	7,340
	5	50M	152	1,944	159	3,552	152	1,829	130	1,670

8. CONCLUSION

In this article, we have investigated the STCLTR problem. Since the STCLTR problem is NP-hard, we propose heuristic scheduling algorithms to solve it. First, we present the *ISGG* algorithm to partition tasks of the input DADFG into a specified number of groups with the objective of minimizing the total communication data size. Second, we propose the *RLD* algorithm to solve the STCLTR problem. *RLD* allots a local deadline for each task, schedules the task within its local deadline, and determines the assignment of the task by defined cost-time ratio. Third, combining *ISSG* and *RLD*, we present the *RLDG* algorithm to solve the STCLTR problem. Tasks in the same group are assigned to cores in the same cluster. *RLDG* has higher time complexity, but it produces better results than *RLD*. Extensive experiments with various characteristics show that the *RLD* and *RLDG* algorithms significantly outperform the related algorithms. For future work, we will develop techniques to solve task scheduling applying the DVS, DVFS, or DPM techniques on heterogeneous multicore embedded systems. Also, we will study a more complex case where clusters and cores are separately discussed.

ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to the editors and the referees for their valuable time and constructive comments which help improve the quality of the manuscript greatly.

REFERENCES

Paul-Antoine Arras, Didier Fuin, Emmanuel Jeannot, Arthur Stoutchinin, and Samuel Thibault. 2015. List scheduling in embedded systems under memory constraints. *International Journal of Parallel Programming* 43, 6 (2015), 1103–1128.

Doruk Bozdag, Umit Catalyurek, and Fuisun Ozguner. 2006. A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*. 12–23.

Gang Chen, Kai Huang, and Alois Knoll. 2014. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Trans. Embed. Comput. Syst.* 13, 3s (March 2014), 111:1–111:21.

Robert P. Dick, David L. Rhodes, and Wayne Wolf. 1998. TGFF: Task graphs for free. In *Proceedings of the 6th International Workshop on Hardware/Software Codesign*. 97–101.

Yongqi Ge, Yunwei Dong, and Hongbing Zhao. 2014. Energy-efficient task scheduling and task energy consumption analysis for real-time embedded systems. In *Proceedings of the Theoretical Aspects of Software Engineering Conference (TASE'14)*. 135–138.

Marco E. T. Gerards and Jan Kuper. 2013. Optimal DPM and DVFS for frame-based real-time systems. *ACM Trans. Archit. Code Optim (TACO)* 9, 4 (Jan. 2013), 41:1–41:23.

Jian-Jun Han, Man Lin, Dakai Zhu, and Laurence T. Yang. 2015. Contention-aware energy management scheme for NoC-based multicore real-time systems. *IEEE Trans. Parallel Distrib. Syst.* 26, 3 (2015), 691–701.

Jia Huang, Jan Olaf Blech, Andreas Raabe, Christian Buckl, and Alois Knoll. 2011. Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*. 247–256.

Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. 2015. Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 179.

Hyungjun Kim, Boris Grot, Paul V. Gratz, and Daniel A. Jimenez. 2014. Spatial locality speculation to reduce energy in chip-multiprocessor networks-on-chip. *IEEE Trans. Comput.* 63, 3 (March 2014), 543–556.

Fanxin Kong, Wang Yi, and Qingxu Deng. 2011. Energy-efficient scheduling of real-time tasks on cluster-based multicores. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*. 1–6.

Joonho Kong, Sung Woo Chung, and Kevin Skadron. 2012. Recent thermal management techniques for microprocessors. *ACM Comput. Surv.* 44, 3 (June 2012).

Wan Yeon Lee. 2012. Energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors. *IEEE Trans. Parallel Distrib. Syst.* 23, 3 (2012), 530–537.

Young Choon Lee and Albert Y. Zomaya. 2011. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Trans. Parallel Distrib. Syst.* 22, 8 (2011), 1374–1381.

Brad Linder. 2014. Allwinner A80 octa-core chip coming in Q2, 2014. (2014). Retrieved December 6, 2016 from https://liliputing.com/2014/04/allwinner-a80-octa-core-chip-coming-q2-2014.html.

Jing Liu, Qingfeng Zhuge, Shouzhen Gu, Jingtong Hu, Guanyu Zhu, and Edwin H. M. Sha. 2014. Minimizing system cost with efficient task assignment on heterogeneous multicore processors considering time constraint. *IEEE Trans. Parallel Distrib. Syst.* 25, 8 (Aug 2014), 2101–2113.

Wei Liu, Hongfeng Li, Wei Du, and Feiyan Shi. 2011. Energy-aware task clustering scheduling algorithm for heterogeneous clusters. In *Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications (GreenCom)*. 34–37.

Jos Luis March, Julio Sahuquillo, Salvador Petit, Houcine Hassan, and Jos Duato. 2013. Power-aware scheduling with effective task migration for real-time multicore embedded systems. *Concurr. Comput. Prac. Exp.* 25, 14 (2013), 1987–2001.

Ramesh Mishra, Namrata Rastogi, Dakai Zhu, Daniel Mossé, and Rami Melhem. 2003. Energy aware scheduling for distributed real-time systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. 113–121.

Morteza Mohaqeqi, Mehdi Kargahi, and Ali Movaghar. 2014. Analytical leakage-aware thermal modeling of a real-time system. *IEEE Trans. Comput.* 63, 6 (2014), 1378–1392.

Meikang Qiu and Edwin H.-M. Sha. 2009. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. *ACM Trans. Des. Autom. Electron. Syst.* 14, 2 (April 2009).

Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. 2008. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Trans. Parallel Distrib. Syst.* 19, 11 (2008), 1540–1552.

Muhammad Shafique, Dennis Gnad, Siddharth Garg, and Jörg Henkel. 2015. Variability-aware dark silicon management in on-chip many-core systems. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 387–392.

Muhammad Shafique, Benjamin Vogel, and Jorg Henkel. 2013. Self-adaptive hybrid dynamic power management for many-core systems. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 51–56.

Zili Shao, Qingfeng Zhuge, Chun Xue, and E. H.-M. Sha. 2005. Efficient assignment and scheduling for heterogeneous DSP systems. *IEEE Trans. Parallel Distrib. Syst.* 16, 6 (June 2005), 516–525.

Amit Kumar Singh, Anup Das, and Akash Kumar. 2013a. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In *Proceedings of the 50th Annual Design Automation Conference (DAC'13)*. 115:1–115:7.

Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2013b. Mapping on multi/many-core systems: Survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*. ACM, 1.

Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jorg Henkel. 2016. Resource and throughput aware execution trace analysis for efficient run-time mapping on MPSoCs. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 35, 1 (2016), 72–85.

Amit Kumar Singh, Thambipillai Srikanthan, Akash Kumar, and Wu Jigang. 2010. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. *J. Syst. Architect.* 56, 7 (2010), 242–255.

Qunyan Sun, Qingfeng Zhuge, Jingtong Hu, Juan Yi, and E. H.-M. Sha. 2014. Efficient grouping-based mapping and scheduling on heterogeneous cluster architectures. *Comput. Elec. Eng.* 40, 5 (2014), 1604–1620.

Timon D. Ter Braak, Philip K. F. Hölzenspies, Jan Kuper, Johann L. Hurink, and Gerard J. M. Smit. 2010. Run-time spatial resource management for real-time applications on heterogeneous MPSoCs. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 357–362.

Haluk Topcuoglu, Salim Hariri, and Min you Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* 13, 3 (2002), 260–274.

Hwang Cheng Wang and Cheng Wen Yao. 2011. Task migration for energy conservation in real-time multiprocessor embedded systems. In *Proceedings of the 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 393–398.

Lizhe Wang, Gregor Von Laszewski, Jai Dayal, and Fugang Wang. 2010. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS. In *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10)*. 368–377.

Wayne Wolf, Ahmed Amine Jerraya, and Grant Martin. 2008. Multiprocessor system-on-chip (MPSoC) technology. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 27, 10 (2008), 1701–1713.

Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. 2007. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *Proceedings of the 44th ACM/IEEE on Design Automation Conference (DAC'07)*. 664–669.

Ying Yi, Wei Han, Xin Zhao, Ahmet T. Erdogan, and Tughrul Arslan. 2009. An ILP formulation for task mapping and scheduling on multi-core architectures. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'09)*. 33–38.

Yukan Zhang, Yang Ge, and Qinru Qiu. 2013. Improving charging efficiency with workload scheduling in energy harvesting embedded systems. In *Proceedings of the 50th Annual Design Automation Conference*. 1–8.

Ziliang Zong, A. Manzanares, Xiaojun Ruan, and Xiao Qin. 2011. EAD and PEBD: Two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters. *IEEE Trans. Comput.* 60, 3 (March 2011), 360–374.

Received September 2015; revised February 2016; accepted May 2016