IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received June 8, 2013; revised December 18, 2013; accepted January 10, 2014. Date of publication 15 January 2014;
date of current version 30 July 2014.

Digital Object Identifier 10.1109/TETC.2014.2300632

Energy-Aware Data Allocation and Task Scheduling on Heterogeneous Multiprocessor Systems With Time Constraints

YAN WANG[1], KENLI LI[1], HAO CHEN[1], LIGANG HE[2], AND KEQIN LI[3]

[1]College of Information Science and Engineering, Hunan University, Changsha 410082, China
[2]College of Information Science and Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, University of Warwick, Coventry CV4 7AL, U.K.
[3]College of Information Science and Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

CORRESPONDING AUTHOR: K. LI (lkl@hnu.edu.cn)

This work was supported by the Key Program of National Natural Science Foundation of China under Grant 61133005, the National Science Foundation of China under Grants 61070057, 90715029, and 61370095, the National Science Foundation for Distinguished Young Scholars of Hunan under Grant 12JJ1011, and the Innovation Fund Designated for Graduate Students of Hunan Province under Grant CX2013B142.

ABSTRACT In this paper, we address the problem of energy-aware heterogeneous data allocation and task scheduling on heterogeneous multiprocessor systems for real-time applications. In a heterogeneous distributed shared-memory multiprocessor system, an important problem is how to assign processors to real-time application tasks, allocate data to local memories, and generate an efficient schedule in such a way that a time constraint can be met and the total system energy consumption can be minimized. We propose an optimal approach, i.e., an integer linear programming method, to solve this problem. As the problem has been conclusively shown to be computationally very complicated, we also present two heuristic algorithms, i.e., task assignment considering data allocation (TAC-DA) and task ratio greedy scheduling (TRGS), to generate near-optimal solutions for real-time applications in polynomial time. We evaluate the performance of our algorithms by comparing them with a greedy algorithm that is commonly used to solve heterogeneous task scheduling problems. Based on our extensive simulation study, we observe that our algorithms exhibit excellent performance. We conducted experimental performance evaluation on two heterogeneous multiprocessor systems. The average reduction rates of the total energy consumption of the TAC-DA and TRGS algorithms to that of the greedy algorithm are 13.72% and 15.76%, respectively, on the first system, and 19.76% and 24.67%, respectively, on the second system. To the best of our knowledge, this is the first study to solve the problem of task scheduling incorporated with data allocation and energy consumption on heterogeneous distributed shared-memory multiprocessor systems.

INDEX TERMS Data allocation, energy consumption, heterogeneous system, task scheduling, time constraint.

I. INTRODUCTION

A. MOTIVATION

A modern high-performance computing system normally consists of heterogeneous computing and communication resources, including heterogeneous processors, heterogeneous memories, and heterogeneous communication interconnections. For instances, both Tianhe-2 [2] and Titan [3] are heterogeneous multiprocessor systems. All real-time applications executed on heterogeneous multiprocessor systems must satisfy certain deadlines. Together with increased demand on high-performance computing, the energy consumption problem in heterogeneous multiprocessor systems has also become more and more important and received extensive attention as green computing becomes a new trend. Heterogeneous multiprocessor systems extend the energy-time tradeoff flexibility, provide more opportunity to

2168-6750 © 2014 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

finer tuning of resource utilization for particular applications, and raise new challenges to the research community.

While a real-time application with data dependencies is executed on a heterogeneous multiprocessor system, the following requirements must be satisfied. First, to guarantee the performance of the system, the application must be completed within a time constraint. Second, to improve energy efficiency, both processor and memory energy consumption should be reduced as much as possible under the time constraint. Third, to enhance parallelism among processors and memories, memory latency should be hidden as much as possible, which implies that memory access operations should be carefully scheduled together with task execution operations.

As more and more heterogeneous processors become available, the same type of operations can be processed by different processors with different execution time and different energy consumption. Furthermore, different components of a distributed shared-memory show significant heterogeneity in data access time and energy consumption. Therefore, there are several important problems arising, i.e., how to assign a proper processor to each computational task; how to assign a proper memory to each datum; and how to generate a schedule for both task execution and data access in such way that certain performance requirement can be met and the energy consumption can be minimized. We call the problem as *heterogeneous data allocation and task scheduling* (HDATS) problem.

However, finding an effective HDATS solution to successfully satisfy the above three requirements is very difficult, because different processors have different task execution time and energy consumption for the same task, and different memories have different data access times and energy consumption for the same processor. Therefore, different HDATS approaches show different capabilities in dealing with the tradeoff of performance constraint and energy consumption. Consequently, finding an efficient HDATS approach to minimizing total energy consumption with a time constraint is significant for real-time applications on heterogeneous multiprocessor systems. The problem of finding an optimal data assignment and an optimal task schedule that have the minimal system energy consumption (i.e., the combined task execution and data access cost) for a given time constraint becomes a critical problem for optimizing the energy-delay tradeoff of a heterogeneous multiprocessor system.

B. RELATED RESEARCH

Generally speaking, the objectives of heterogeneous task scheduling are to map tasks onto processors and to order their executions, so that the task precedence constraints are satisfied and other performance and resource requirements can be met. It is well known that the heterogeneous task scheduling problem with resource constraints is NP-hard [34]. The task scheduling problems on heterogeneous systems have been studied in [8], [14], [17], [23], [24], [26], and [30]. These work mainly addressed the problem of minimizing an application's completion time [24], [26], [29], and provided high-quality schedules, with their performance comparable with other algorithms in the literature at shorter scheduling times. There are also algorithms in the literature [10], [11], [16], [25] incorporating the reliability into heterogeneous multiprocessor systems. In these work, scheduling is performed based on a fixed architecture to maximize the reliability of a system.

Energy savings when solving the heterogeneous data assignment and task scheduling problem can be significant. In heterogeneous multiprocessor systems, there are large families of embedded processors with different execution speed and energy consumption characteristics. Incorporating execution speed, communication overhead, and energy consumption into heterogeneous systems, energy consumption driven assignment and scheduling problems have been studied in [4], [6], [13], [15], and [34]. For example, Shao *et al.* proposed a two-phase approach to solving heterogeneous assignment problems, so that all requirements can be met and the total cost can be minimized [34]. Real-time applications can be modeled as acyclic dataflow graphs that capture task dependencies, where multiple tasks have data dependency constraints. In such an application, each task has some input data and some output data. The process of data allocation determines the location of each datum in software-controlled memories. However, the above mentioned studies attempted to solve the task scheduling problem without considering data allocation in detail.

Due to the influence of data allocation on the performance of systems, task scheduling problems have been extensively studied to incorporate data allocation into consideration, and various heuristic algorithms were proposed in the literature [5], [7], [12], [19], [21], [22], [28], [30], [31]. They mainly focus on optimizing the performance of a system, where the algorithms provide good quality solutions and their performance are compared so that the schedule length or some predefined cost functions can be minimized. Due to the gap in performance between memories and processors, by incorporating memory latency into multiprocessor systems, the memory latency driven assignment and scheduling problems have been studied in [5], [12], [22], [28], and [30]. However, when considering memory constraints, the complexity of the data allocation and task scheduling problem increases significantly, and developing efficient algorithms on heterogeneous multiprocessor systems encounters some challenges. Incorporating memory constraints into multiprocessor systems, the memory constraint driven data assignment and task scheduling problem has been studied in [9], [18], [32], and [35]. In these work, allocation of data in different levels of memory units is based on data access frequencies in order to satisfy performance requirement with both cost and energy efficiency consideration. For example, the authors in [18] studied the problem of minimizing the total cost of computation and communication in a heterogeneous

computing system with a resource constraint. The problem was shown to be NP-hard. A simple and effective iterative greedy algorithm was proposed for the scheduled tasks. The main idea in this algorithm is to improve the quality of an assignment in an iterative manner using results from previous iterations. The algorithm first uses a constructive heuristic to second an initial assignment and iteratively improves it in a greedy way.

However, there exist several problems when these techniques are applied to heterogeneous multiprocessor systems to solve the HDATS problem. This is because distributed heterogeneous memories and scalable interconnection networks in heterogeneous multiprocessor systems lead to non-uniform structures of memory access. The objective of the present paper is to develop efficient algorithms to solve the HDATS problem, which generate a schedule that has minimal energy consumption within certain time constraint.

C. OUR CONTRIBUTIONS

In this paper, we present an optimal algorithm and two heuristic algorithms to solve the HDATS problem. The three algorithms aim to obtain an efficient task schedule incorporated with data allocation, so that certain timing requirement can be met and total system (i.e., both processors and memories) energy consumption can be minimized. To obtain an optimal solution, we present an *integer linear programming* (ILP) formulation to solve the HDATS problem. Since it takes a long time for the ILP method to get results even for medium-sized DAGs with no more than 100 nodes, we propose two heuristic algorithms, i.e., the TAC-DA (*task assignment considering data allocation*) and the TRGS (*task ratio greedy scheduling*) algorithms. The TAC-DA algorithm includes two phases. The first phase uses the *DFG_Assign_CP* algorithm [34] to find a better mapping for each task node. The second phase chooses an assignment for all data whose total energy consumption is minimized within a time constraint according to the result from the first phase. Since it is possible that the time constraint is too tight for TAC-DA to obtain a solution, we propose the TRGS algorithm in which data assignment is considered in conjunction with task scheduling.

Experimental results show that our algorithms have better performance compared with the greedy algorithm [18]. On the average, reduction rates of the total energy consumption of the TAC-DA and TRGS algorithms to that of the greedy algorithm are 13.72% and 15.76% respectively on one system, and 19.76% and 24.67% respectively on another system. The computation time of the ILP method is unacceptable for large-sized DAGs. On the contrary, the TAC-DA algorithm can obtain near-optimal solutions for loose time constraints and the TRGS algorithm can always generate near-optimal solutions efficiently for all the benchmarks, if there exists a solution.

The major contributions of this paper are summarized as follows.

- We consider heterogeneous processors, heterogeneous memories, precedence constrained tasks, input/output data of each task, processor execution times, data access times, time constraints, and energy consumption, in solving the data allocation and task scheduling problem to minimize the total energy consumption.
- We formulate an integer linear programming (ILP) model to solve the HDATS problem and to obtain an optimal solution in which the total energy consumption is the minimum within a time constraint.
- We propose two efficient heuristic algorithms, i.e., the TAC-DA algorithm and the TRGS algorithm, to solve the HDATS problem. The algorithms have improved performance compared with an existing algorithm.

To the best of our knowledge, this is the first study to solve the problem of task scheduling incorporated with data allocation and energy consumption on heterogeneous distributed shared-memory multiprocessor systems.

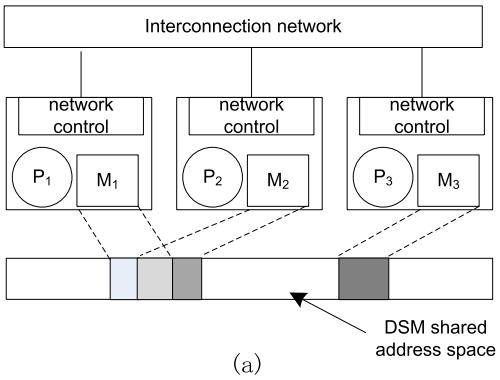
The remainder of this paper is organized as follows. In Section II, we present our heterogeneous system model and task model. In Section III, we use an example to illustrate the motivation and method of this paper. In Section IV, we design an ILP model to obtain an optimal solution. In Section V, we propose two heuristic algorithms to solve the heterogeneous data allocation and task scheduling (HDATS) problem. In Section VI, we evaluate and analyze our techniques compared with the greedy algorithm. In Section VII, we conclude this paper and discuss future work.

II. THE MODELS

In this section, we first describe our heterogeneous multiprocessor system model. Then, we introduce the task scheduling system model for our algorithms. Finally, we define our heterogeneous data allocation and task scheduling (HDATS) problem.

A. ARCHITECTURE MODEL

In this paper, the architecture model is a heterogeneous distributed shared-memory multiprocessor system shown in Fig. 1. The architecture model consists of a set of connected heterogeneous processors denoted by $P = \{P_1, P_2, \ldots, P_n\}$, where n is the number of heterogeneous processors. Each processor P_i is tightly coupled with its own local memory M_i, and all local memories of individual processors form a distributed shared-memory. For example, for processor P_1, M_1 is the local memory, while M_2 and M_3 are remote but accessible memories. For processor P_2, M_2 is the local memory, while M_1 and M_3 are remote memories. Integrating the distributed memories into a global address space, every processor has full access to the memories, and a memory access operation represents the processor's reading from or writing to a memory. Due to distributed heterogeneous memories and a scalable interconnection network, the structure of memory access in our architecture model is non-uniform. Therefore, different processors' accesses to a datum in the same memory show different access times and energy consumption.



memory	P_1		P_2		P_3	
	AT_1	AE_1	AT_2	AE_2	AT_3	AE_3
M_1	1	3	4	6	6	5
M_2	5	8	1	2	7	6
M_3	6	6	5	7	1	1

(b)

FIGURE 1. An architecture model. (a) An architecture with three heterogeneous processors, each is embedded with a local memory. (b) Access times and energy consumption for transmitting one unit of data between processors and local memories.

In our model, local memories are heterogeneous and different from each other in terms of capacity, access concurrency, access time, energy consumption, and other characteristics. We describe the *access time* as a function $AT : P \times M \rightarrow R$, where $AT(P_i, M_j)$ is the time for processor P_i to access a unit data from local memory M_j. For example, in Fig. 1(b), the value in the cell of column "AT_1" and row "M_2" indicates that the access time is 5 time units when processor P_1 accesses a unit data from memory M_2.

Our architecture uses the radio energy model as defined in [27], in which the energy consumption for transmitting a k-bit datum is as follows:

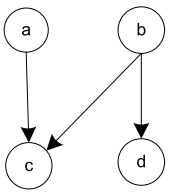
$$k(E_{ele} + \varepsilon_{FS} \times d^4), \tag{1}$$

where E_{ele} represents electronic energy, ε_{FS} denotes a transmit amplifier parameter, and d is the transmission distance. We describe the *access energy* as a function $AE : P \times M \rightarrow R$, where $AE(P_i, M_j)$ is the amount of energy consumed by processor P_i to access a unit data from local memory M_j. For example, in Fig. 1(b), the value in the cell of column "AE_1" and row "M_2" indicates that the access energy is 8 energy units when processor P_1 accesses a unit data from memory M_2.

B. COMPUTATION MODEL

In this subsection, we describe the *memory-access data flow graph* (MDFG) model, which is used to model an application to be executed on a heterogeneous distributed shared-memory multiprocessor system. Before we formally describe the MDFG model for the heterogeneous data allocation and task scheduling (HDATS) problem, let us first introduce a *directed acyclic graph* (DAG) model as shown in Fig. 2. In this paper, we use a DAG as a description of a given input graph.

Definition 2.1: A DAG is a node-weighted directed graph represented by $G = (V, E, D, in, out, ET, EE)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is a set of task nodes, and $E \subseteq V \times V$ is a set of edges that describe the precedence constraints among nodes in V. D is a set of data. $in(v_i) \subseteq D$ is a set of input data of task v_i, and $out(v_i) \subseteq D$ is a set of output data of task v_i. $ET(v_i)$ is used to represent the execution times of task $v_i \in V$ on different processors, i.e., $ET(v_i) = (et_1(i), et_2(i), \ldots, et_n(i))$, where $et_j(i)$ denotes the execution time of v_i on processor P_j. $EE(v_i)$ is used to represent the energy consumption of task $v_i \in V$ on different processors, i.e., $EE(v_i) = (ee_1(i), ee_2(i), \ldots, ee_n(i))$, where $ee_j(i)$ denotes the energy consumption of v_i on processor P_j.



Task	Input data	Output data
a	A(2KB),B(1KB)	A(2KB)
b	B(1KB),C(1KB)	C(1KB)
c	A(2KB),D(1KB)	
d	E(1.5KB),F(1KB)	F(1KB)

(b)

FIGURE 2. An input DAG. (a) Precedence constraints among tasks. (b) The input data and output data of each task.

An example of DAG is shown in Fig. 2. In the example, there are $N = 4$ tasks, i.e., a, b, c, d. Fig. 2(a) shows the precedence constraints among the tasks. The data set is $D = \{A, B, C, D, E, F\}$, and Fig. 2(b) shows the input data and output data of each task. For example, task a reads input data A and B before it is started, and writes output data A after it is finished.

Table 1 shows the execution time and energy consumption of each task in the DAG of Fig. 2. For example, the value in the cell of column "ET_1" and row "b" indicates that the

TABLE 1. The Execution Time and Energy Consumption of the Tasks in the Input Graph.

Task	P_1		P_2		P_3	
	ET_1	EE_1	ET_2	EE_2	ET_3	EE_3
a	4	10	5	8	7	6
b	8	18	6	20	10	18
c	6	15	4	15	8	12
d	10	20	8	24	12	20

execution time of task b is 8 time units when it is executed on processor P_1; and the value in the cell of column "EE_1" and row "b" indicates that the energy consumption of task b is 18 energy units when it is executed on processor P_1.

If we treat a memory access operation as a node, we can redefine a DAG to obtain a memory-access data flow graph (MDFG).

Definition 2.2: An MDFG derived from a DAG is a node-weighted directed graph represented by $G' = (V_1, V_2, E, D, var, P, M, AT, AE, ET, EE)$, where $V_1 = \{v_1, v_2, \ldots, v_{N_1}\}$ is a set of N_1 task nodes, and $V_2 = \{u_1, u_2, \ldots, u_{N_2}\}$ is a set of N_2 memory access operation nodes. $E \subseteq V \times V$ ($V = V_1 \bigcup V_2$) is a set of edges. An edge $(\mu, \nu) \in E$ represents the dependency between node μ and node ν, indicating that task or operation μ has to be executed before task or operation ν. D is a set of data. $var : V_1 \times V_2 \times D \rightarrow \{\texttt{true}, \texttt{false}\}$ is a binary function, where $var(v_i, u_l, h)$ denotes whether the memory access operation $u_l \in V_2$ is transmitting datum $h \in D$ for task $v_i \in V_1$. $P = \{P_1, P_2, \ldots, P_n\}$ is a set of processors, and $M = \{M_1, M_2, \ldots, M_n\}$ is a set of local memories. AT and AE are access time and access energy functions. $ET(v_i, P_j) = et_j(i)$ is the execution time of task v_i when it is executed on processor P_j, and $EE(v_i, P_j) = ee_j(i)$ is the energy consumed by task v_i when it is executed on processor P_j.

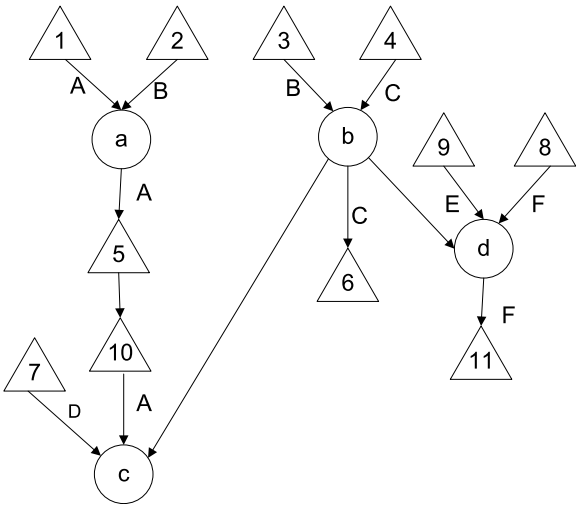


FIGURE 3. An MDFG obtained from the example input graph.

An MDFG of the DAG in Fig. 2 is shown in Fig. 3, where we have $N_1 = 4$ task nodes and $N_2 = 11$ memory access operation nodes.

C. PROBLEM DEFINITION

Assume that we are given a heterogeneous distributed shared-memory multiprocessor system, which consists of n heterogeneous processors $P_1, P_2, \ldots, P_n$, where each processor P_i is tightly coupled with a local memory M_i. The access time and access energy of each processor in accessing a unit data from a local memory are known in advance. The *heterogeneous data allocation and task scheduling* (HDATS) problem is formally defined as follows. Given a DAG $G = (V, E, D, in, out, ET, EE)$, and a time constraint S, we treat a memory access operation as a node and reconstruct the DAG to obtain an MDFG $G' = (V_1, V_2, E, D, var, P, M, AT, AE, ET, EE)$. The HDATS problem is to find (1) a data allocation $Mem : D \rightarrow M$, where $Mem(h) \in M$ is the memory to store $h \in D$; (2) a task assignment $A : V_1 \rightarrow P$, where $A(v_i)$ is the processor to execute task $v_i \in V_1$; (3) and a schedule, i.e., the starting time of each task in V_1 and each memory access operation in V_2, such that the completion time of the MDFG G' satisfies the constraint $T(G') \leq S$, and the total energy consumption $E(G')$ is minimized.

The general HDATS problem is NP-hard. The heterogeneous assignment problem has been proved to be NP-complete [34]. The NP-hardness of the HDATS problem can be easily proved by a reduction from the heterogeneous assignment problem defined by Shao *et al.* [34].

III. A MOTIVATIONAL EXAMPLE

In this section, we use an example to illustrate the effectiveness of our algorithms. The example application in Fig. 2 is executed on a heterogeneous distributed shared-memory multiprocessor systems shown in Fig. 1.

Based on the dependency constraints in the MDFG shown in Fig. 3, a schedule is generated by a greedy scheduling algorithm shown in Fig. 4(a). Conventionally, the approach to attacking the scheduling problem would be to minimize the completion time by scheduling the tasks using the *shortest processing time* policy. Hence, in this schedule, tasks a and c are scheduled on processor P_1, and tasks b and d are scheduled on P_2. The data A and B are allocated to M_1, data C and D are allocated to M_2, and data E and F are allocated to M_3. The completion time of this schedule is 23 time units, and the total energy consumption for completing this schedule is 127.5 energy units. However, this approach may not produce a good result in terms of energy consumption, since this approach only considers completion time. Therefore, we should explore a new technique to obtain a better schedule which considers energy consumption.

Fig. 4(b) shows an improved schedule, which considers energy consumption together with a time constraint, i.e., $S = 30$. In this schedule, task d is scheduled on P_1, task b is scheduled on P_2, and tasks a and c are scheduled on P_3. The data E and F are allocated to M_1, data B and C are allocated to M_2, and A and D are allocated to M_3. The completion time of the improved schedule is 24 time units, and the total energy consumption is 81.5 energy units. Although the schedule has slightly longer completion time, it has considerably lower energy consumption than the greedy schedule. The energy consumption of the schedule is reduced by $(127.5 - 81.5)/127.5 = 36.5\%$ compared with the greedy schedule, while the time constraint $S = 30$ is satisfied.

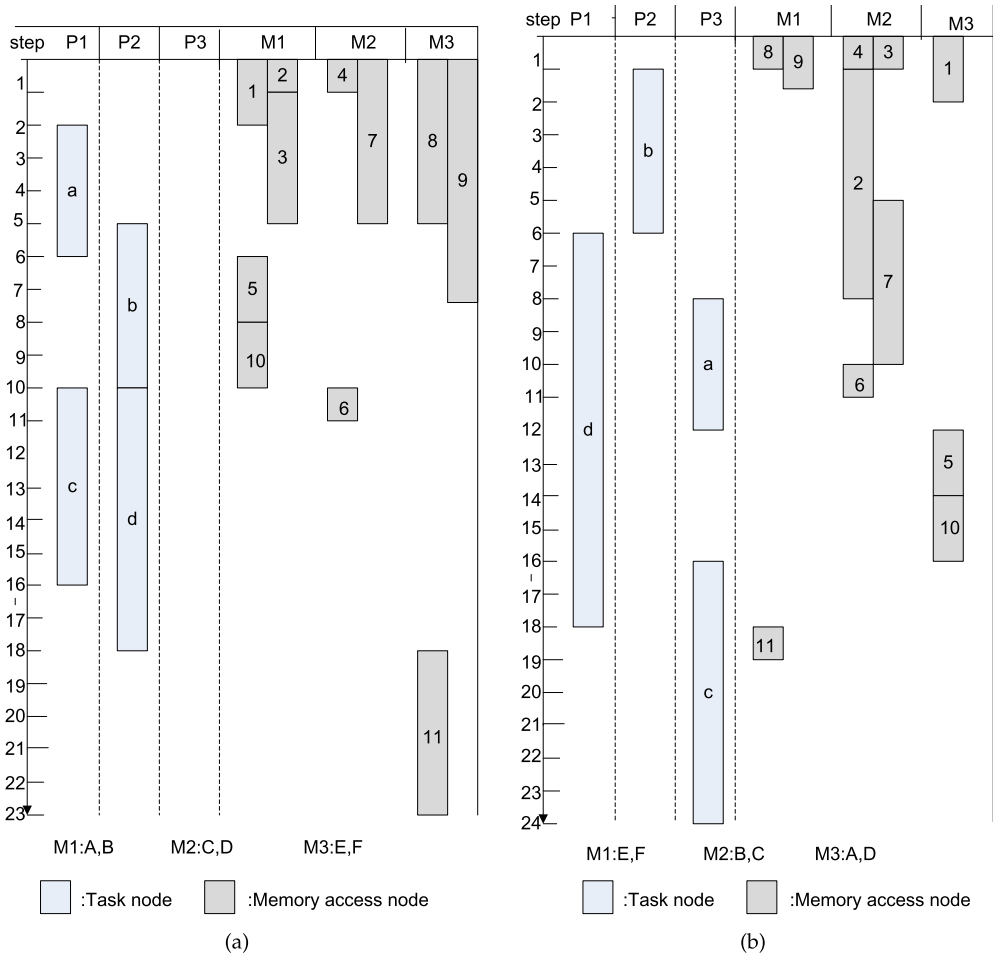


FIGURE 4. Data allocation and task scheduling (the data access times are scaled by a factor of 0.5). (a) A greedy schedule with time 23 and energy 127.5. (b) An improved schedule with time 24 and energy 81.5.

From the above example, we can see that the energy consumption can be reduced by exploring data allocation and task scheduling on a heterogeneous multiprocessor system while a time constraint is satisfied. A heterogeneous architecture has more choices and challenges than a homogeneous architecture in selecting a right processor for a task and in selecting a right memory for a datum in order to achieve our objectives of energy saving and satisfaction of a time constraint. Therefore, it is important to study the data allocation and task scheduling problem on heterogeneous multiprocessor systems.

IV. AN ILP FORMULATION

In this section, we develop our ILP formulation. We aim to find an allocation of all data and an assignment of all tasks in a given input DAG, such that the total energy consumption is minimized under a time deadline requirement and various resource constraints. We build up our ILP formulation step by step, including a task assignment with processor constraint, a data allocation with memory size and concurrency constraints, precedence constraints, a time constraint, and an objective function.

TABLE 2. Notations Used in the ILP Formulation.

Notation	Definition
N_1	Number of task nodes
N_2	Number of memory access operation nodes
n	Number of processors
S	Time constraint
$Size_i$	Size of memory M_i
MA	Concurrent access number
N_d	Number of data
$d(h)$	Size of data h

Before describing data allocation and task scheduling, we provide notations used in Table 2.

The data allocation and task scheduling model consists of two major parts, i.e., a processor part and a memory part. The processor part aims to find a task assignment for all tasks of a given input DAG, and the memory part aims to find a data allocation for all data needed by task executions. In our algorithms, all data have been allocated in different local memories before tasks are executed.

A. TASK ASSIGNMENT AND PROCESSOR CONSTRAINT

In the processor part, a task assignment is modeled with two binary variables $x_{i,j,k}$ and $x'_{i,j,m}$, where $x_{i,j,k}$ denotes whether task v_i in an MDFG G' starts to execute in step k on processor P_j, and $x'_{i,j,m}$ denotes whether task v_i is scheduled in step m on processor P_j. Each task node can start execution in one and only one step and on one and only one processor:

$$\sum_{j=1}^{n}\sum_{k=1}^{S} x_{i,j,k} = 1, \quad \forall i \in [1, N_1]. \tag{2}$$

To make sure that there is at most one task scheduled in any step on any processor, a second constraint is added:

$$\sum_{i=1}^{N_1} x'_{i,j,m} \leq 1, \ \forall j \in [1, n], \quad \forall m \in [1, S]. \tag{3}$$

To satisfy processor constraint, in each step, the number of tasks executed should be bounded by the number of processors:

$$\sum_{i=1}^{N_1}\sum_{j=1}^{n} x'_{i,j,m} \leq n, \quad \forall m \in [1, S]. \tag{4}$$

The processor $P(i)$ on which task v_i is executed can be defined as

$$P(i) = \sum_{j=1}^{n}\sum_{k=1}^{S} j \times x_{i,j,k}, \quad \forall i \in [1, N_1]. \tag{5}$$

B. DATA ALLOCATION AND MEMORY CONSTRAINT

In the memory part, a data allocation is modeled with a binary variable $d_{h,j}$, which denotes whether data h is allocated to local memory M_j. For each data block, it can be allocated to one and only one local memory:

$$\sum_{j=1}^{n} d_{h,j} = 1, \ \forall h \in [1, N_d]. \tag{6}$$

Let $Size_j$ be the capacity of local memory M_j. For each local memory M_j, the size of all data in M_j must be smaller or equal to $Size_j$:

$$\sum_{h=1}^{N_d} d(h) \times d_{h,j} \leq Size_j, \quad \forall j \in [1, n]. \tag{7}$$

To ensure data allocation, recall that data h is allocated to memory $Mem(h)$. The local memory $Mem(h)$ to which data h is allocated can be stated in terms of $d_{h,j}$ as

$$Mem(h) = \sum_{j=1}^{n} j \times d_{h,j}, \quad \forall h \in [1, N_d]. \tag{8}$$

Let binary variable $y_{l,j,k}$ denote whether memory access operation node u_l starts to execute in step k on local memory M_j. Each memory access operation node can start execution in one and only one step and on one and only one local memory:

$$\sum_{j=1}^{n}\sum_{k=1}^{S} y_{l,j,k} = 1, \quad \forall l \in [1, N_2]. \tag{9}$$

Let binary variable $y'_{l,j,m}$ denote whether memory access operation node u_l is scheduled in step m on local memory M_j. In each step, the number of memory access operation nodes should be bounded by the concurrent access number of a local memory:

$$\sum_{l=1}^{N_2} y'_{l,j,m} \leq MA, \quad \forall j \in [1, n], \forall m \in [1, S]. \tag{10}$$

We define a variable $M(l)$ to show the relationship between data allocation and memory access operations. According to the architectural model, a memory access operation must be scheduled on the memory to which the corresponding data has been allocated. The memory module $M(l)$ for the memory access operation u_l which accesses data $h = D(l)$ is defined as

$$Mem(D(l)) = M(l) = \sum_{j=1}^{n}\sum_{k=1}^{S} j \times y_{l,j,k}, \ \forall l \in [1, N_2]. \tag{11}$$

C. PRECEDENCE CONSTRAINTS

In addition, edge $e(u, v) \in E$ represents a precedence relationship. We use four constraints to ensure that each task and each memory access operation correctly follow the precedence constraints. The dependency constraints can be formulated by Eqs. (12)–(15), as shown at the top of the next page. Eq. (12) characterizes the precedence constraints among tasks. Eqs. (13) and (15) show the precedence constraints between tasks and memory access operations. Eq. (14) represents the precedence constraints among memory access operations. Basically, each equation means that u must be completed before v can be started. Notice that $RT(u, j)$ in Eqs. (12)–(13), i.e., the execution time of task u, and $RA\_t(u, j)$ in Eqs. (14)–(15), i.e., the access time of memory access operation u, will be given in Section D.

D. EXECUTION AND ACCESS TIME

Let $RT(i, j)$ represent the real execution time of task v_i on processor P_j:

$$RT(i, j) = \sum_{k=1}^{S} x_{i,j,k} \times ET(v_i, P_j), \tag{16}$$

where $ET(v_i, P_j)$ is given in Definition 2.1. For each task, the following relationship between $x'_{i,j,m}$ and $RT(i, j)$ must be satisfied:

$$\sum_{m=1}^{S} x'_{i,j,m} \leq RT(i, j), \ \forall i \in [1, N_1], \quad \forall j \in [1, n], \tag{17}$$

which is to bound the total number of steps to execute task v_i on P_j. If $x_{i,j,k} = 1$, then $x'_{i,j,m}$ must satisfy the following

$$\sum_{j=1}^{n}\sum_{k=1}^{S}(k+RT(u,j))\times x_{u,j,k} \le \sum_{j=1}^{n}\sum_{k=1}^{S} k \times x_{v,j,k},\ \forall e(u,v)\in G',\forall u\in[1,N_1],\forall v\in[1,N_1]. \tag{12}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{S}(k+RT(u,j))\times x_{u,j,k} \le \sum_{j=1}^{n}\sum_{k=1}^{S} k \times y_{v,j,k},\ \forall e(u,v)\in G',\forall u\in[1,N_1],\forall v\in[1,N_2]. \tag{13}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{S}(k+RA\_t(u,j))\times y_{u,j,k} \le \sum_{j=1}^{n}\sum_{k=1}^{S} k \times y_{v,j,k},\ \forall e(u,v)\in G',\forall u\in[1,N_2],\forall v\in[1,N_2]. \tag{14}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{S}(k+RA\_t(u,j))\times y_{u,j,k} \le \sum_{j=1}^{n}\sum_{k=1}^{S} k \times x_{v,j,k},\ \forall e(u,v)\in G',\forall u\in[1,N_2],\forall v\in[1,N_1]. \tag{15}$$

equation:

$$\sum_{m=k}^{k+RT(i,j)-1} x'_{i,j,m} = RT(i,j),\quad \forall i\in[1,N_1],\forall j\in[1,n], \tag{18}$$

which means that the steps to execute a task must be consecutive.

Let RA_t represent the real memory access time of a memory access operation u_l on local memory M_j:

$$RA\_t(l,j)=\sum_{i=1}^{N_1}\sum_{h=1}^{N_d}\sum_{k=1}^{S} y_{l,j,k}var(v_i,u_l,h)AT(P(i),M_j)d(h), \tag{19}$$

where we notice that u_l accesses data h of size $d(h)$ for task v_i, which is executed on processor $P(i)$ with access time $AT(P(i),M_j)$ for a unit data. For each memory access operation, the following relationship between $y'_{l,j,m}$ and $RA\_t(l,j)$ must be satisfied:

$$\sum_{m=1}^{S} y'_{l,j,m} \le RA\_t(l,j),\ \forall l\in[1,N_2],\forall j\in[1,n], \tag{20}$$

which is similar to Eq. (17). If $y_{i,j,k}=1$, then $y'_{i,j,m}$ must satisfy the following equation:

$$\sum_{m=k}^{k+RA\_t(l,j)-1} y'_{l,j,m} = RA\_t(l,j),\ \forall l\in[1,N_2],\forall j\in[1,n], \tag{21}$$

which is similar to Eq. (18).

E. ENERGY CONSUMPTION

The real energy consumption $RE(i,j)$ of a task v_i on processor P_j can be defined as

$$RE(i,j)=\sum_{k=1}^{S} x_{i,j,k}\times EE(v_i,P_j), \tag{22}$$

where $EE(v_i,P_j)$ is given in Definition 2.1. Also, the real access energy consumption $RA\_e(l,j)$ of a memory access operation u_l on local memory M_j can be defined as

$$RA\_e(l,j)=\sum_{i=1}^{N_1}\sum_{h=1}^{N_d}\sum_{k=1}^{S} y_{l,j,k}var(v_i,u_l,h)AE(P(i),M_j)d(h), \tag{23}$$

which is similar to Eq. (19).

F. OBJECTIVE FUNCTION

Therefore, the objective function can be defined as:

$$E=\sum_{i=1}^{N_1}\sum_{j=1}^{n} RE(i,j)+\sum_{l=1}^{N_2}\sum_{j=1}^{n} RA\_e(l,j), \tag{24}$$

which is to be minimized.

V. HEURISTIC ALGORITHMS

In this section, we propose two polynomial time heuristic algorithms, i.e., the *task assignment considering data allocation* (TAC-DA) algorithm and the *task ratio greedy scheduling* (TRGS) algorithm, to solve the HDATS problem. They aim to reduce total energy consumption, while satisfying time constraints. In the two algorithms, $A(v_i)$ indicates the assignment of node v_i; $T(G')$ represents the completion time of an MDFG; and $E(G')$ represents the total energy consumption of an MDFG.

Before presenting the details of the TAC-DA algorithm and the TRGS algorithm, we define two cost-to-time ratio computing functions as shown in Equations (25) and (26). The two functions decide the reassignments of tasks and data, respectively. First, we define

$$Ratio(v_i,P_j)=\frac{DiffCost(v_i,P_j)}{DiffTime(v_i,P_j)}, \tag{25}$$

where $DiffCost(v_i,P_j)$ is the increased energy consumption when task v_i is moved from the currently assigned processor to a new processor P_j, and $DiffTime(v_i,P_j)$ is the increased execution time when task v_i is moved from the currently assigned processor to a new processor P_j. Next, we define

$$Ratio(u_l,M_j)=\frac{DiffCost(u_l,M_j)}{DiffTime(u_l,M_j)}, \tag{26}$$

where $DiffCost(u_l, M_j)$ is the increased energy consumption when the data accessed by memory access operation u_l is moved from the currently allocated memory to a new memory M_j, and $DiffTime(u_l, M_j)$ is the increased access time when the data accessed by memory access operation u_l is moved from the currently allocated memory to a new memory M_j.

Let $op(h)$ denote the set of memory access operations u_l that access data h. Then, we define

$$DiffTime(h, M_j) = \sum_{u_l \in op(h)} DiffTime(u_l, M_j), \tag{27}$$

and

$$Ratio(h, M_j) = \sum_{u_l \in op(h)} Ratio(u_l, M_j). \tag{28}$$

A. TAC-DA AIGORITHM

The TAC-DA algorithm is shown in Algorithm 1, which consists of two phases and is a straightforward heuristic algorithm. The first phase aims to find a better mapping for each task node, and the second phase aims to find the best assignment for each memory access operation according to the first step.

In the first phase, we use the *DFG_Assign_CP* algorithm [34] to find a better mapping for each task. The *DFG_Assign_CP* algorithm in [34] is used to solve the *heterogeneous assignment problem* for real-time DSP applications, where all requirements should be met and the total cost should be minimized. To obtain a better solution, before using the *DFG_Assign_CP* algorithm to solve the task mapping problem, we set a new deadline $L = \rho S$ for task mapping, where ρ is a correlation coefficient between the memory access operation nodes and the task nodes. For simplicity, we set $\rho = n_1/(n_1 + n_2)$, where n_1 and n_2 are the number of task nodes and the number of memory access operation nodes on a critical path in an MDFG, respectively. The critical path indicates a path with the maximum number of nodes including task nodes and memory access operation nodes among all paths in an MDFG G'. Intuitively, ρ is the proportion of task execution time and $1 - \rho$ is the proportion of memory access time.

In the second phase, we should find an allocation for each data according to the first phase. Since a data may be needed by different tasks, more than one memory access operation may be associated with the data. For each data, we calculate the total energy consumption of each available assignment. Then, we assign the data to a local memory with the minimum total energy consumption of all memory access operations associated with the data. After solving the task mapping and data allocation problem, we need to detect whether the total completion time $T(G')$ meets the time constraint or not. If the total completion time $T(G')$ satisfies the time constraint, we obtain a solution; otherwise, we should re-allocate some data. In re-allocating data to satisfy the time constraint, we select a data with the lowest cost-to-time ratio $Ratio(h, M_j)$ and $DiffTime(h, M_j) < 0$ to be moved to local memory M_j. After adjustment of data allocation is done, the algorithm tries to find a new data allocation and attempts to reduce its completion time until the time constraint is satisfied, or the completion time of G' cannot be reduced any more.

In the TAC-DA algorithm, it takes $O(|V_1||E_1| + |V_1|^2)$ time to find a better mapping for each task node, where V_1 represents the number of task nodes and E_1 represents the number of edges between task nodes. To find a better data allocation, it tasks at most $O(|V_2|M)$ time to calculate the ratios, select a data from the MDFG, and change its allocation, where V_2 indicates the number of memory access operation nodes and M indicates the number of local memory. The second phase iterates at most $O(|V_2|M)$ times, since each local memory of a data is only assigned one time. Thus, the second phase takes $O((|V_2|M)^2)$ to obtain a better data allocation. If M is treated as a constant, the time complex of the TAC-DA algorithm is $O(|V|^2 + |V||E|)$, where $|V|$ and $|E|$ are the number of all nodes and the number of all edges, respectively.

Algorithm 1 TAC-DA Algorithm

Require: (1) A DAG G (MDFG G'); (2) A deadline S.
Ensure: (1) A near-optimal data allocation; (2) A near-optimal task assignment.

```
/* task assignment */
set a deadline L = ρS for task mapping
call DFG_Assign_CP algorithm [34] to find an effective mapping for each task node
/* data allocation */
for each data h ∈ D do
    for each M_j ∈ M do
        if M_j is a new local memory and has space to store the data h then
            compute the energy consumption of all u_l ∈ op(h)
        end if
    end for
    re-allocate the data h to M_k which yields the minimum energy consumption
end for
if T(G') > S then
    repeat
        for each data h ∈ D do
            for each M_j ∈ M do
                if M_j has space to store the data h and DiffTime(h, M_j) < 0 then
                    compute Ratio(h, M_j)
                end if
            end for
        end for
        re-allocate data h to M_k which yields the minimum Ratio(h, M_k)
    until T(G') ≤ S or T(G') cannot be reduced
end if
```

However, the TAC-DA algorithm may not be able to obtain a solution, because data allocation is not considered in conjunction with task scheduling. In an actual case, to solve the HDATS problem, data allocation should be considered together with task scheduling at the same time, so that both data allocation and task scheduling can be improved. Therefore, we present the TRGS algorithm, which considers task scheduling and data allocation simultaneously.

B. TRGS ALGORITHM

The TRGS algorithm is shown in Algorithm 2, which aims to complete all tasks with minimum energy consumption and to meet a given time constraint. In the TRGS algorithm, a *critical path* (CP) is defined as a path $p : u \rightsquigarrow v$ with the maximum completion time among all paths in G'. Thus, the completion time of the critical path p is equal to the completion time of an MDFG G'.

In algorithm TRGS, we first use the list scheduling algorithm to assign each task and each data. In the list scheduling, we assign a data to a local memory according to a task assignment. Each data is located in the same processor of a task which needs the data to execute the earliest. We then find a critical path that has the maximum execution time among all possible paths based on the current assignment. Next, if the execution time of the critical path is greater than the given time constraint S, we reduce it by changing the processor of a task or the local memory of a memory access operation that is selected from the critical path. The task or operation is selected from the critical path, since only the completion time of the critical path is equal to the completion time of the MDFG G'. In order to obtain the minimal energy consumption and satisfy the time constraint with the resource constraint, the TRGS algorithm always selects a node with the lowest cost-to-time ratio to be moved to a processor P_j or a local memory M_j, and the new match must satisfy $DiffTime < 0$ compared with the original match. For the same amount of reduction on completion time, a smaller cost-to-time ratio indicates a smaller increase of energy. After adjusting the assignment of a critical path is done, the algorithm tries to find a new critical path in G' and attempts to reduce its completion time until the time constraint is satisfied, or the completion time of G' cannot be reduced any more.

On the other hand, if the completion time of G' under the initial assignment satisfies the time constraint, the algorithm tries to reduce the energy consumption by moving a node with the lowest cost-to-time ratio to a new processor or local memory. Since the goal of re-assignment is to reduce energy consumption, for each node, the energy consumption of the new assignment must be lower than that of the original assignment. In other words, the new assignment must satisfy $DiffCost < 0$. In these cases, the cost-to-time ratio indicates the benefit of reduction with a sacrifice on time performance. After reassigning a node, the algorithm tries to find another node and continues to make such attempt until the energy consumption of G' cannot be reduced any more. The TRGS algorithm iteratively tries each free processor for task assignment and each local memory with enough space for data allocation to find a schedule with the minimum energy consumption, while the time constraint is satisfied.

Algorithm 2 TRGS Algorithm

Require: (1) A DAG G (MDFG G'); (2) A deadline S.
Ensure: (1) A near-optimal data allocation; (2) A near-optimal task assignment.

obtain a list schedule with data allocation $A(u_h) \leftarrow M_k$ and processor assignment $A(v_i) \leftarrow P_k$
if $T(G') > S$ **then**
  repeat
    find a critical path: $u_i \rightsquigarrow u_j$ and $v_i \rightsquigarrow v_j$ in G'
    $V_{cp} \leftarrow$ all nodes in the critical path
    for each $vc_i \in V_{cp}$ **do**
      if $vc_i \in V_1$ **then**
        for each $P_j \in P$ **do**
          if P_j is a new processor for vc_i and $DiffTime(vc_i, P_j) < 0$ **then**
            compute $Ratio(vc_i, p_j)$
          end if
        end for
      else
        for each $M_j \in M$ **do**
          if M_j is a new local memory and has space to store the data of access operation vc_i, and $DiffTime(vc_i, M_j) < 0$ **then**
            compute $Ratio(vc_i, M_j)$
          end if
        end for
      end if
    end for
    $Ratio(vc_i, M_k)$ or $Ratio(vc_i, P_k) \leftarrow$ the minimal ratio in critical path p
    $A(vc_i) \leftarrow M_k$ or P_k
  until $T(G') \leq S$
else
  repeat
    for each node v_i in G' **do**
      if the node $v_i \in V_1$ **then**
        for each $P_j \in P$ **do**
          if P_j is an available processor for task v_i and $DiffCost(v_i, P_j) < 0$ and $T(G') \leq S$ **then**
            compute $Ratio(v_i, p_j)$
          end if
        end for
      else
        for each $M_j \in M$ **do**
          if M_j is a new local memory and has space to store the data of access operation v_i, and $DiffCost(v_i, M_j) < 0$ and $T(G') \leq S$ **then**
            compute $Ratio(v_i, M_j)$
          end if
        end for
      end if
    end for
    $Ratio(v_i, M_k)$ or $Ratio(v_i, P_k) \leftarrow$ the minimal ratio in critical path p
    $A(v_i) \leftarrow M_k$ or P_k
  until $E(G')$ cannot be reduced
end if

The time complexity of the TRGS algorithm is $O(|V|^2(P+M)^3(2|E|+|V|))$, where $|V|$ is the number of all nodes, P is the number of processors, M is the number of local memories, and $|E|$ is the number of edges. If M and P are treated as constants, the TRGS algorithm takes $O(|V|^2(|E|+|V|))$ time.

VI. PERFORMANCE EVALUATION

In this section, we first describe the experimental setup. We then show the results of evaluating the effectiveness of the proposed algorithms on different systems.

A. EXPERIMENT SETUP

We use the following benchmarks in our experiments, i.e., IIR, Allople, Floyd, Elliptic, and 10-4lattic-iir. These benchmarks are from DSPstone [33], and frequently used on multicore systems. We compile the benchmarks with gcc and extract the task graphs and the read/write data sets from gcc. There are three phases. First, the source codes must be compiled with profiling (`-fprofile-generate`). Then, the compiled binary must be executed with a data set corresponding to the use case. Finally, the source code must be compiled again with both profile-guided optimization and ABSINTH enabled (`-fprofile-use-fabsinth`). The `pass_absinth_bbs` traverses all RTL expressions within each basic block. For each expression, `pass_absinth_bbs` analyzes whether it is an instruction or not, and generates one execute primitive per each instruction [20]. Then, the task graphs and access sets are fed into our simulator. The number of tasks, dependency edges, and data of each benchmark are shown in Table 3.

TABLE 3. Sizes of DSPstone Benchmarks.

Benchmark	Tasks	Edges	Data	β	ETR	$\overline{D_{DFG}}$
IIR	8	7	12	0.2	1.1	6KB
Allople	15	17	24	0.45	1.25	2KB
Floyd	16	20	28	0.75	0.95	5KB
Elliptic	34	47	48	0.5	1.75	4KB
10-4lattic-iir	260	221	380	1.0	1.2	3KB

Our graphs extracted require the following parameters to build weighted MDFGs.

Range of task execution time β – It is basically the heterogeneity factor for processor speeds. A higher β value causes more difference among a task's execution time on the processors. The average execution time $\overline{T_i}$ of task v_i in the graph is selected randomly from a uniform distribution with range $[0, 2\overline{T_{DFG}}]$, where $\overline{T_{DFG}}$ is the average computation time of the given graph, which is set randomly in the algorithm. Then, the execution time of each task v_i on each processor P_j in the system is randomly set from the following range:

$$\left(1-\frac{\beta}{2}\right)\overline{T_i} \le ET(v_i,P_j) \le \left(1+\frac{\beta}{2}\right)\overline{T_i}. \qquad (29)$$

The above equation implies that $ET(v_i, P_j)$ is a uniform random variable with mean $\overline{T_i}$ over an interval of length $\beta\overline{T_i}$, where β is the degree of heterogeneity of task execution time.

Energy to time ratio (ETR) – It is the ratio of the average energy consumption to the average execution time of an MDFG. Then, the energy consumption of task v_i on processor P_j is $EE(v_i, P_j) = \overline{E_j} \times ET(v_i, P_j) \times ETR$, where $\overline{E_j}$ scales the energy consumption of processor P_j.

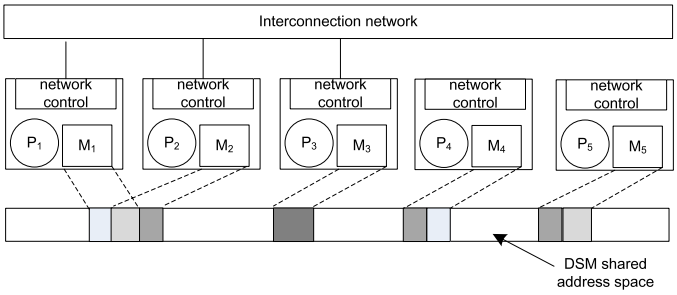
Data parameter α – The number of data needed in the graph is $N_d = \alpha \times \sqrt{V} \times \sqrt{E}$, where V is the number of tasks in the graph and E is the number of edges in the MDFG.

Data size γ – $\overline{D_{DFG}}$ is the average data size of a task graph, which is set randomly in the algorithm. For a data h, the data size parameter γ_h is selected randomly from a uniform distribution in the range [0, 2]. Then, the data size of each data h is $d(h) = \gamma_h \times \overline{D_{DFG}}$.

All the experiments for DAGs are conducted on two different architecture models which are defined in Section 2.1. The first one is composed of three heterogeneous processors shown in Fig. 1. Fig. 5 shows the second one, which consists of five heterogeneous processors. A set of parameters of the two architecture models are collected from ARM7 and MSP430 by using the CACTI tools [1] provided by HP. The set of parameters are shown in Table 4. The row "*Time latency*" and row "*Energy consumption*" show the wake-up time and the wake-up energy of each processor, respectively. The access time per unit data is *time latency* $+ \varepsilon_{FT} \times d$, where ε_{FT} is the transmission parameter for access time and d is the transmission distance between two processors, whose values are set randomly. The access energy per unit data is $E_{ele} + \varepsilon_{FS} \times d^4$, where E_{ele} is the wake-up energy, and ε_{FS} is the transmit amplifier parameter, whose value is set randomly.

For convenience, we have given the execution time and energy consumption of a unit data for the two models shown in Figs. 1 and 5. Values of execution time and energy consumption are given for all task nodes. Data reads and writes are pre-determined. All the experiments are conducted by a simulator on an Intel® Core™ 2 Duo Processor E7500 2.93G with a 2GB main memory running Red Hat Linux 7.3.

We performed two groups of experiments. In the two groups of experiments, we use all benchmarks running on heterogeneous multiprocessor models shown in Figs. 1 and 5 to demonstrate the effectiveness of the TAC-DA algorithm and the TRGS algorithm. Two performance metrics considered in our experiments are completion time and energy consumption. In the two sets of experiments, our algorithms are compared with the greedy algorithm [18]. The greedy algorithm first uses a constructive heuristic to second an initial assignment and iteratively improves it in a greedy way. This is a recently published algorithm to minimize the total cost of computation and communication, which is employed to schedule tasks with time constraints in heterogeneous systems. For example, the greedy algorithm has been applied in application-special DSP processors to optimize scheduling, and it has been shown to be very effective. Therefore, the greedy algorithm is the most related work and an excellent algorithm for comparison. In this paper, the greedy algorithm has been adjusted so that it is suitable to our model to solve the HDATS problem. To make fair compar-



(a)

memory	P_1		P_2		P_3		P_4		P_5	
	AT_1	AE_1	AT_2	AE_2	AT_3	AE_3	AT_4	AE_4	AT_5	AE_5
M_1	1	2	3	4	4	6	5	8	2	3
M_2	4	3	1	2	4	7	5	9	3	3
M_3	3	3	4	5	1	1	6	8	2	2
M_4	4	6	6	7	7	10	2	2	4	5
M_5	2	2	3	3	4	9	5	13	1	1

(b)

FIGURE 5. The second architecture model. (a) An architecture with five heterogeneous processors, each is embedded with a local memory. (b) Access times and energy consumption for transmitting one unit of data between processors and local memories.

TABLE 4. System Specification for Model 1 and Model 2.

Parameter	Model 1			Model 2				
	Core 1	Core 2	Core 3	Core 1	Core 2	Core 3	Core 4	Core 5
Frequency	64MHz	30MHz	7.5Mhz	6MHz	3MHz	1.5MHz	0.75MHz	12MHz
Local memory size	128KB	64KB	32KB	32KB	16KB	8KB	8KB	64KB
Time latency	1.4ms	2.38ms	2.45ms	1.225ms	2.787ms	2.781ms	3.781ms	0.876ms
Energy consumption	0.1mJ	1.47mJ	1.53mJ	0.593mJ	1.359mJ	1.849mJ	2.187mJ	0.252mJ

TABLE 5. The Results of the Four Algorithms on the First System With Three Heterogeneous Processors.

Benchmark	TC	Greedy	ILP method	TAC-DA			TRGS		
		energy	energy	energy	%(greedy)	%(ILP)	energy	%(greedy)	%(ILP)
IIR (8)	20	118	106	115	2.5%	8.4%	109	7.62%	2.83%
	25	118	91	106	10.17%	16.48%	96	18.64%	5.49%
	30	105	84	89	15.23%	5.95%	87	17.14%	3.57%
	35	81	65	67	17.28%	3.07%	66	18.51%	1.53%
	40	72	60	60	16.67%	0.00%	60	16.67%	0.00%
Allople (15)	37	256	232	241	5.86%	3.01%	238	7.03%	2.59%
	42	226	199	204	9.73%	2.51%	203	10.17%	2.01%
	47	196	169	175	10.71%	3.55%	172	12.24%	1.78%
	52	179	139	144	19.55%	3.59%	143	20.11%	2.88%
	57	147	116	119	19.04%	2.59%	119	19.04%	2.59%
Floyd (16)	70	213	176	198	7.04%	12.5%	187	12.21%	6.25%
	75	199	168	174	12.56%	3.45%	172	13.57%	2.38%
	80	156	136	141	9.61%	3.67%	140	10.25%	2.94%
	85	144	121	124	13.89%	2.48%	124	13.89%	2.48%
	90	129	109	110	14.73%	0.92%	110	14.73%	0.92%
Elliptic (34)	114	501	459	–	–	–	476	4.99%	3.70%
	128	439	382	408	7.06%	4.97%	396	9.79%	3.66%
	142	343	280	287	16.33%	2.50%	285	16.91%	1.79%
	156	328	270	275	16.16%	1.85%	275	16.16%	1.85%
	170	328	270	275	16.16%	1.85%	275	16.16%	1.85%
10-4Lat -IIR (260)	120	3689	×	3012	18.35%	×	2983	19.14%	×
	140	2912	×	2246	22.87%	×	2145	26.34%	×
	160	2458	×	1921	21.85%	×	1879	23.56%	×
	180	2143	×	1659	22.58%	×	1654	22.82%	×
	200	2143	×	1578	26.36%	×	1578	26.36%	×
Average					13.72%	2.64%		15.76%	2.06%

isons, we implement all the four algorithms, i.e., greedy, ILP, TAC-DA, and TRGS, within the same scheduling framework. In doing so, we ensure that the performance disadvantage of the greedy algorithm is not due to fundamental limitations of the implementations.

B. RESULTS AND ANALYSIS

The first group of experimental results are shown in Table 5, which reports the statistical performance comparison of all the four algorithms and for all benchmarks based on the architectural model shown in Fig. 1. In the table, column TC shows the given time constraint. Each "×" indicates that the corresponding experiment cannot generate a solution within 24 hours, and each "–" represents that the algorithm cannot obtain a solution under the time constraint. Table 5 shows that our algorithms TAC-DA and TRGS can achieve better performance than the greedy algorithm, and their performance is very close to that of the ILP method. The reduction rates

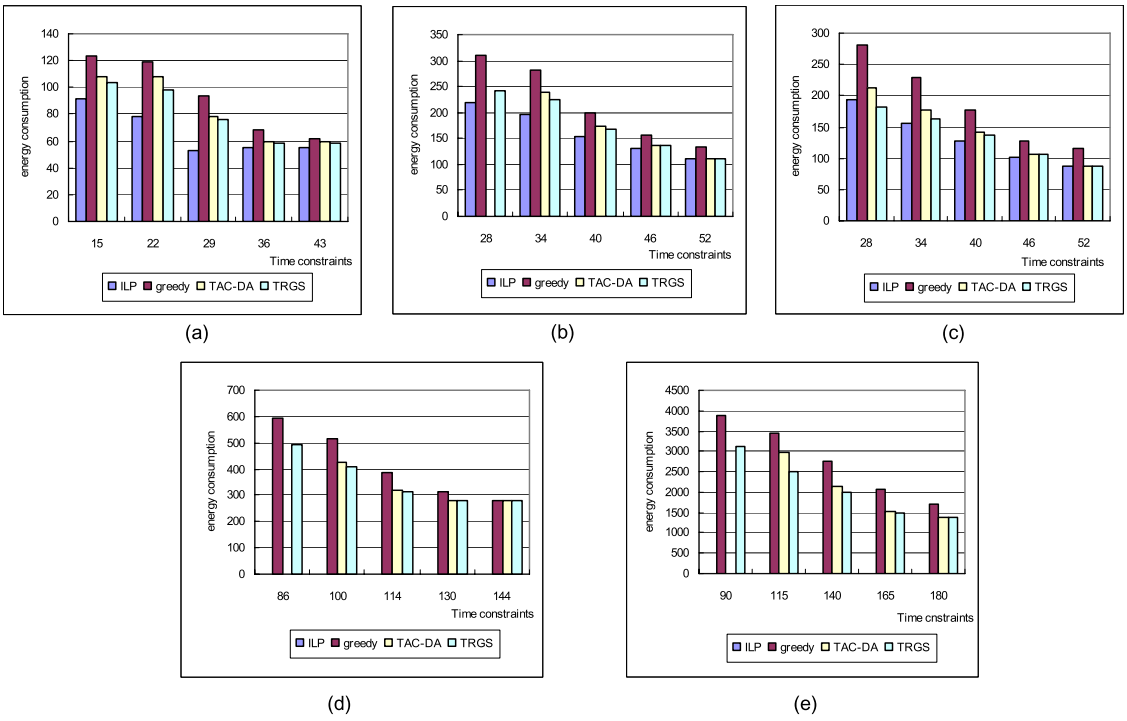


FIGURE 6. **The results of the four algorithms on the second system with five heterogeneous processors. (a) IIR. (b) Allople. (c) Floyd. (d) Elliptic. (e) 10-4lat-IIR.**

of our techniques compared with the greedy algorithm are $(E_g - E)/E_g$, where E_g indicates the energy consumption of the greedy algorithm and E represents the energy consumption of our techniques. From the row "Average," the TAC-DA and TRGS algorithms reduce the total energy consumption by 13.72% and 15.76% on the average compared with the greedy algorithm, respectively. The increment rates of TAC-DA and TRGS compared with ILP are $(E - E_{ILP})/E_{ILP}$, where E_{ILP} is the energy consumption of the ILP method. From the row "Average", the average increment rates of the total energy consumption of the TAC-DA and TRGS algorithms are only 2.64% and 2.06%, respectively.

Fig. 6 shows the second group of experimental results, which are obtained by running a set of simulations on all benchmarks based on the architectural model shown in Fig. 5. As we can see, with the extension of the time constraint, the energy consumption of all the four algorithms decreases and the gap of energy consumption between the four algorithms becomes smaller. From the figure, we know that the energy consumption of the TAC-DA and TRGS algorithms are less than that of the greedy algorithm, and are greater than that of the ILP method. The TAC-DA and TRGS algorithms reduce the total energy consumption by 19.76% and 24.67% on the average compared with the greedy algorithm, respectively. Furthermore, the larger the time constraint, the closer their performance is to that of the ILP method. Therefore, the TAC-DA and TRGS algorithms are superior to the greedy algorithm. The average increment rates of total energy consumption of the TAC-DA and TRGS algorithms are only 1.27% and 0.092%, respectively. Furthermore, from the figure, we are able to compare TAC-DA and TRGS algorithms. We can see that, in general, the energy consumption of the TRGS algorithm is less than that of the TAC-DA algorithm. Although the TAC-DA algorithm might be better than the TRGS algorithm when the time constraint is greater than a special value, it may not obtain a solution under a tight time constraint, because data allocation is not considered in conjunction with task scheduling at the same time. Therefore, the TRGS algorithm is superior to the TAC-DA algorithm.

From the two groups of experimental results, we know that the computation time of the ILP method grows exponentially with increasing size of benchmarks, although the ILP method obtains optimal results on some of the benchmarks. Our experimental results show that the ILP method takes a long time to get results even for a medium-sized DAG with no more than 50 nodes. For example, on the system with three processors shown in Fig. 1, the ILP method takes 287 minutes to calculate the result for a Floyd filter with time constraint $S = 85$, while the TAC-DA and TRGS algorithms take less than 1 minute to produce near-optimal solutions. Furthermore, the ILP method cannot generate a solution for 10-4lattice filter within 24 hours. The computation time of the ILP method is unacceptable for large-sized DAGs. On the contrary, the TAC-DA algorithm can obtain a near-optimal solution with a loose time constraint and the TRGS algorithm can always generate a near-optimal solution efficiently for all the benchmarks.

VII. CONCLUSION

In this paper, we presented an optimal algorithm, i.e., the ILP method, and two heuristic algorithms, i.e., the TAC-DA and TRGS algorithms, to solve the HDATS problem that aims to obtain better task scheduling incorporated with data allocation, such that the total system energy consumption is minimized for a given time constraint. For experimental studies, we employed two heterogeneous multiprocessor systems to execute various applications, where one consists of three heterogeneous processors, and the other consists of five heterogeneous processors, and both systems have different heterogeneity characteristics in terms of access time and access energy. In the experiments conducted on the two systems, both TAC-DA and TRGS algorithms achiev noticeable average reduction rates of the total energy consumption compared with the greedy algorithm. Furthermore, the performance of the two techniques are very close to that of the ILP method, with very low average increment rates of the total energy consumption. Moreover, the TRGS algorithm is superior to the TAC-DA algorithm.

Further research can be directed towards finding more effective and efficient algorithms with reduced time complexity and improved energy efficiency.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the three anonymous reviewers whose constructive comments have helped to improve the manuscript.

REFERENCES

[1] (2008). *Cacti Model* [Online]. Available: http://www.hpl.hp.com/research/cacti

[2] (2013). *tianhe-i* [Online]. Available: http://en.wikipedia.org/wiki/tianhe-i

[3] (2013). *top500* [Online]. Available: http://www.top500.org/system/176958.

[4] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: A simulated annealing approach," *J. Parallel Distrib. Comput.*, vol. 66, no. 10, pp. 1259–1266, 2006.

[5] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems," *IEEE Trans. Comput.*, vol. 55, no. 7, pp. 918–923, Jul. 2006.

[6] T. Chantem, X. Hu, and R. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 10, pp. 1884–1897, Oct. 2011.

[7] T. Chen and J. Sheu, "Communication-free data allocation techniques for parallelizing compilers on multicomputers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 9, pp. 924–938, Sep. 1994.

[8] Y. Chen, H. Liao, and T. Tsai, "On-line real-time task scheduling in heterogeneous multi-core system-on-a-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 118–130, Jan. 2013.

[9] T. Chiang, P. Chang, and Y. Huang, "Multi-processor tasks with resource and timing constraints using particle swarm optimization," *Int. J. Comput. Sci. Netw. Security*, vol. 6, no. 4, pp. 71–77, 2006.

[10] A. Doğan and F. Özgüner, "Scheduling of a meta-task with QoS requirements in heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 66, no. 2, pp. 181–196, 2006.

[11] 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.*, 2007, pp. 280–288.

[12] J. Du, Y. Wang, Q. Zhuge, J. Hu, and E. Sha, "Efficient loop scheduling for chip multiprocessors with non-volatile main memory," *J. Signal Process. Syst.*, vol. 71, no. 3, pp. 261–273, 2012.

[13] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," in *Proc. 45th Annu. Des. Autom. Conf.*, 2008, pp. 191–196.

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

[15] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proc. Des., Autom. Test Eur. Conf. Exhibit.*, vol. 1. 2004, pp. 234–239.

[16] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," in *Proc. Conf. Des., Autom. Test Eur. Conf. Exhibit.*, 2009, pp. 51–56.

[17] E. Ilavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *J. Comput. Sci.*, vol. 3, no. 2, pp. 94–103, 2007.

[18] Q. Kang, H. He, and H. Song, "Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm," *J. Syst. Softw.*, vol. 84, no. 6, pp. 985–992, 2011.

[19] S. Kang and A. Dean, "DARTS: Techniques and tools for predictably fast memory using integrated data allocation and real-time task scheduling," in *Proc. 16th IEEE RTAS*, Apr. 2010, pp. 333–342.

[20] J. Kreku, K. Tiensyrjä, and G. Vanmeerbeeck, "Automatic workload generation for system-level exploration based on modified GCC compiler," in *Proc. Des., Autom. Test Eur. Conf. Exhibit.*, 2010, pp. 369–374.

[21] K. Lakshmanan, D. de Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in *Proc. 30th IEEE RTSS*, Dec. 2009, pp. 469–478.

[22] B. Ouni, R. Ayadi, and A. Mtibaa, "Partitioning and scheduling technique for run time reconfigured systems," *Int. J. Comput. Aided Eng. Technol.*, vol. 3, no. 1, pp. 77–91, 2011.

[23] A. Page, T. Keane, and T. Naughton, "Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system," *J. parallel Distrib. Comput.*, vol. 70, no. 7, pp. 758–766, 2010.

[24] A. Prayati, C. Koulamas, S. Koubias, and G. Papadopoulos, "A methodology for the development of distributed real-time control applications with focus on task allocation in heterogeneous systems," *IEEE Trans. Ind. Electron.*, vol. 51, no. 6, pp. 1194–1207, Dec. 2004.

[25] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters," *J. Parallel Distrib. Comput.*, vol. 65, no. 8, pp. 885–900, 2005.

[26] X. Qin and T. Xie, "An availability-aware task scheduling strategy for heterogeneous systems," *IEEE Trans. Comput.*, vol. 57, no. 2, pp. 188–199, Feb. 2008.

[27] M. Qiu, J. Liu, J. Li, Z. Fei, Z. Ming, and E.-M. Sha, "A novel energy-aware fault tolerance mechanism for wireless sensor networks," in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun.*, Aug. 2011, pp. 56–61.

[28] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *Proc. 1th IEEE Int. Symp. High Perform. Distrib. Comput.*, Jul. 2002, pp. 352–358.

[29] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, Apr. 2004, p. 111.

[30] H. Salamy and J. Ramanujam, "An effective solution to task scheduling and memory partitioning for multiprocessor system-on-chip," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 31, no. 5, pp. 717–725, May 2012.

[31] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated scratchpad memory optimization and task scheduling for MPSoC architectures," in *Proc. Int. Conf. Compil., Archit. Synthesis Embedded Syst.*, 2006, pp. 401–410.

[32] Y. Tian, H. Edwin, C. Chantrapornchai, and P. Kogge, "Optimization for data placement and data scheduling on processor-in-memory arrays," in *Proc. 1st Merged Int. Symp. Parallel Distrib. Process. Parallel Process. Symp.*, Apr. 1998, pp. 57–61.

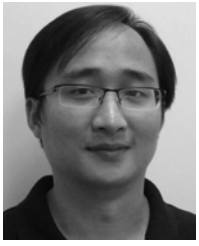
[33] C. S. V. Zivojnovic, J. Martinez, and H. Meyr, "DSPstone: A DSP-oriented benchmarking methodology," in *Proc. ICSPAT*, 1994, pp. 1–6.

[34] C. X. Z. Shao, Q. Zhuge, and E. H. M. Sha, "Efficient assignment and scheduling for heterogeneous DSP systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 6, pp. 516–525, Jun. 2005.

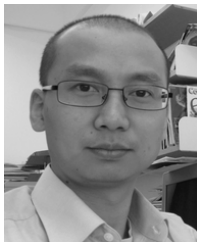
[35] Q. Zhuge, Y. Guo, J. Hu, W. Tseng, S. Xue, and E. Sha, "Minimizing access cost for multiple types of memory units in embedded systems through data allocation and scheduling," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 3253–3263, Jun. 2012.

YAN WANG received the B.S. degree in information management and information technology from Shenyang Aerospace University in 2010. She is currently pursuing the Ph.D. degree with Hunan University, China. Her research interests include modeling and scheduling in parallel and distributed computing systems, and high performance computing.

KENLI LI received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He was a Visiting Scholar with the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently a Full Professor of computer science and technology with Hunan University and an Associate Director of the National Supercomputing Center, Changsha. His current research includes parallel computing, grid and cloud computing, and DNA computing. He has published more than 90 papers in international conferences and journals, such as the IEEE TRANSACTIONS ON COMPUTERS, THE IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, JPDC, ICPP, and CCGrid. He is an Outstanding Member of CCF.



HAO CHEN received the B.S. degree in chemical engineering from Sichuan University, China, in 1998, and the Ph.D. degree in computer science from the Huazhong University of Science and Technology, China, in 2005. He is currently an Associate Professor with the School of Information Science and Engineering, Hunan University, China. His research interests include parallel and distributed computing, operating systems, cloud computing, and systems security. He has published more than 60 papers in journals and conferences, such as the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the International Parallel and Distributed Processing Symposium, and the International Workshop on Quality of Service. He is a member of the ACM.



LIGANG HE received the bachelor's and master's degrees from the Huazhong University of Science and Technology, Wuhan, China, and the Ph.D. degree in computer science from the University of Warwick, U.K. He was a Post-Doctoral Researcher with the University of Cambridge, U.K. In 2006, he joined the Department of Computer Science, University of Warwick, as an Assistant Professor, and is currently an Associate Professor. His areas of interest are parallel and distributed computing, grid computing, and cloud computing. He has published more than 50 papers in international conferences and journals, such as the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (TPDS), the International Parallel and Distributed Processing Symposium, Cluster, CCGrid, and MASCOTS. He served as a member of the program committee for many international conferences and he was the reviewer for a number of international journals, including the IEEE TPDS, IEEE TRANSACTIONS ON COMPUTERS, and the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING.

KEQIN LI is a SUNY Distinguished Professor of computer science. He was an Intellectual Ventures endowed Visiting Chair Professor with Tsinghua University, China, from 2011 to 2013. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He has published over 280 refereed research publications. He has served on the editorial board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the *Journal of Parallel and Distributed Computing*, the *International Journal of Parallel, Emergent and Distributed Systems*, the *International Journal of High Performance Computing and Networking*, the *International Journal of Big Data Intelligence*, and *Optimization Letters*.