CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **28**:4443–4459
Published online 12 February 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3772

Data-aware task scheduling on heterogeneous hybrid memory multiprocessor systems

Junjie Chen[1], Kenli Li[1,*,†], Zhuo Tang[1], Chubo Liu[1], Yan Wang[1] and Keqin Li[1,2]

[1]*School of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha, 410082, China*
[2]*Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

SUMMARY

In this paper, we propose a method about task scheduling and data assignment on heterogeneous hybrid memory multiprocessor systems for real-time applications. In a heterogeneous hybrid memory multiprocessor system, an important problem is how to schedule real-time application tasks to processors and assign data to hybrid memories. The hybrid memory consists of dynamic random access memory and solid state drives when considering the performance of solid state drives into the scheduling policy. To solve this problem, we propose two heuristic algorithms called improvement greedy algorithm and the data assignment according to the task scheduling algorithm, which generate a near-optimal solution for real-time applications in polynomial time. We evaluate the performance of our algorithms by comparing them with a greedy algorithm, which is commonly used to solve heterogeneous task scheduling problem. Based on our extensive simulation study, we observe that our algorithms exhibit excellent performance and demonstrate that considering data allocation in task scheduling is significant for saving energy. We conduct experiments on two heterogeneous multiprocessor systems. Copyright © 2016 John Wiley & Sons, Ltd.

Received 31 August 2015; Revised 17 December 2015; Accepted 19 December 2015

KEY WORDS: data assignment; energy consumption; heterogenous system; hybrid memory; multiprocessor system; task scheduling; time constraint

1. INTRODUCTION

1.1. Motivation

Heterogeneous multiprocessor systems normally consist of heterogeneous processors, heterogeneous memories, and heterogeneous communication interconnections. With the advent of the era of big data, heterogeneous multiprocessor systems, especially heterogeneous memories, are widely used in industry. This is because a huge number of memory access operations cause a performance bottleneck. When resolving task scheduling and data assignment problem, traditionally, the main memory is dynamic random access memory (DRAM), and the secondary memory is hard disk drive (HDD). The completion time always depends on the memory access latency because of the performance bottleneck of *DRAM* and *HDD*. The structural change in the storage and access pattern is becoming a tendency in heterogeneous multiprocessors systems. To bridge the hardware performance bottlenecks and to satisfy the ever-growing performance demands of high-performance computing, a variety of hybrid memory architectures have been put forward such as *PCM* with *NAND* Flash [1], uCache [2], and main memory with NOR Flash [3]. In this paper, we adopt hybrid main memory, which consists of a *DRAM* and a solid state drive (SSD) in heterogeneous multiprocessor systems.

*Correspondence to: Kenli Li, School of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha, 410082, China.
†E-mail: lkl@hnu.edu.cn

Solid state drive is widely adopted as a second level memory to assist a main memory like *DRAM*. There are two major advantages. First, compared with *DRAM*, *SSD* has much lower cost-per-byte and larger capacity. Second, *SSD* outperforms rotation-based *HDD* in both read and write latency with two orders of magnitude improvement and in energy consumption. Hence, *SSD* is able to serve as a memory to complement the capacity deficiency of *DRAM*. On heterogeneous multiprocessor systems, *SSD* can be accessed by both local processor and remote processors. All real-time applications executed on heterogeneous multiprocessor systems must satisfy certain deadlines. Together with the increasing demand on high-performance computing, the energy consumption problem on heterogeneous multiprocessor systems has also become more and more important and received extensive attention as green computing becomes a new trend. Thus, we consider to minimize the energy consumption with certain constraints on heterogeneous hybrid memory multiprocessor systems for the task scheduling and data assignment problem.

As we all know, scheduling on multiprocessor systems is NP-hard [4, 5] in general. In order to achieve a near-optimal result of this problem, we propose a polynomial-time near-optimal heuristic algorithm, which consists of four phases. In the first phase, we adopt the Heterogeneous Earliest Finish Time (HEFT) [6] algorithm to schedule tasks. In the *HEFT* algorithm, each task is scheduled to a processor with the shortest execution time. In the second phase, the share data are distinguished from the single data, and the share data matrix and the single data matrix are generated correspondingly. And then, we assign each data according to the share data matrix and the single data matrix to minimize the total memory access cost. Finally, we optimize the task and data allocation based on the time constraint while decreasing the energy consumption. To the best of our knowledge, this is the first study in task scheduling and data assignment, which considers time constraint and energy consumption on heterogeneous hybrid memory multiprocessor systems.

1.2. Our contributions

The main contributions of this paper include the following aspects.

- 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 to solve the task scheduling and data assignment problem to minimize the total energy consumption.
- We design an improvement greedy algorithm for the problem of task scheduling and data assignment on heterogeneous hybrid $DRAM + SSD$ memory multiprocessor system.
- We propose a polynomial-time algorithm called DAA_TS algorithm. After determining the task scheduling scheme, to minimize the total memory access cost, we first consider share data assignment, and then consider single data assignment.

Experimental result shows that our algorithms have better performance compared with *the greedy algorithm* [7], which does not consider data assignment, which proves that data assignment is very important in the high-performance problem. On an average, compared with *the greedy algorithm*, the energy consumption in the *IG* and DAA_TS algorithms is reduced by 23.36% and 33.26% on the first system and 24.89% and 33.26% on the second system. DAA_TS algorithm can always generate a near-optimal solution, which consumes less energy than *IG* algorithm for all the benchmarks.

The rest of the paper is organized as follows. Related works are discussed in Section 2. In Section 3, we present our heterogeneous multiprocessor systems model and the task model. In Section 4, we use an example to illustrate the motivation and method of this paper. In Section 5, we propose two heuristic algorithms to solve the Heterogeneous Data Allocation and Task Scheduling (HDATS) [8] problem. In Section 6, we evaluate and analyze our techniques to compare with *the greedy algorithm*. In Section 7, we conclude this paper.

2. RELATED WORK

Recently, there are abundant researches about memory architectures. Besides benefiting data storage, they also improve the effectiveness of handling the tasks. Chen *et al*. [9] utilized the hybrid memory to propose data split for big data in hybrid memory with a *DRAM* and an *SSD*. Jiang *et al*.

DOI: 10.1002/cpe

[2] proposed a utility-aware multilevel *SSD* cache policy uCache. uCache is a combination of *DRAM* and *SSD*, which needs a suitable policy to promote effectiveness in data storage. Liu *et al*. [1] adopted a write-activity aware NAND flash memory management scheme to effectively manage NAND flash memory and enhance the endurance of PCM-based embedded systems. Our model depends on [1], [2] and [9]. Hence, the model adopted in this paper is reasonable and correct.

There have so many efforts on task scheduling and data assignment under heterogeneous multiprocessor systems, which are to improve the performance of time constraint and energy consumption [8, 10, 11]. Wang [8] proposed an effective algorithm, which aims to reduce the energy consumption for share memory on heterogeneous multiprocessor systems under the time constraint. Alvarez [10] considered to manage the runtime in many-core architectures with hybrid memories composed of scratchpads and caches. Odagiri, J. [11] presents an Synchronous Dynamic Random Access Memory (SDRAM) data assignment technique for dynamic multi-threaded multimedia applications.

On heterogeneous multiprocessor systems, there are a large number of embedded processor with different execution speed and energy consumption characteristics. So many mechanisms have been studied in [12–14]. Scholars have studied a kind of mechanisms, which incorporate execution speed, communication overhead, and energy consumption. Casas *et al*. [12] researched an approach towards a Runtime-Aware Architecture for a parallel architecture. Wu *et al*. [13] proposed an efficient model, which can offer a unifying and extendable method to realize energy-efficient real-time scheduling with low scheduling overhead. Marchal *et al*. [14] presented a unifying energy-efficient real-time scheduling theory for multiprocessor systems.

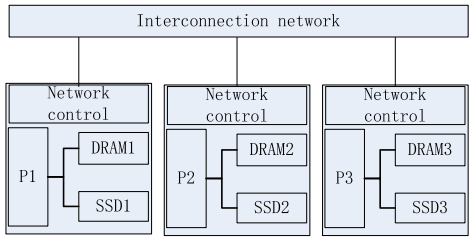
We know that task scheduling and data assignment have already achieved a lot of achievements, but they all did not consider heterogeneous hybrid memory with *DRAM* and *SSD* multiprocessor systems. This paper proposes a new scheduling algorithm based on the systems to solve the problem of task scheduling incorporated with data assignment and energy consumption.

3. THE MODELS

In this section, we first introduce our target hardware architecture model. Then, we describe the directed acyclic graph (DAG) model for our algorithms. Finally, the formal definition of the problem is presented.

3.1. Hardware architecture model

In this paper, the hardware architecture model, which is heterogeneous hybrid memory multiprocessor systems is shown in Figure 1. The architecture model is a set of heterogeneous processors denoted by $P = \{P_1, P_2, \ldots, P_n\}$, where n is the number of heterogeneous processors. The set of heterogeneous processors are linked together through network. Each main memory is equipped with a *DRAM* and an *SSD*. Hence, we refer this memory structure as a hybrid memory. Each processor can access its local hybrid memory and all the other hybrid memory, which belongs to other processors. We call a processor who accesses to its own a *DRAM* or an *SSD* as a local



(a)

MEMORY		DRAM1	DRAM2	DRAM3	SSD1	SSD2	SSD3
P1	AT1	1	2	3	11	12	16
	AE1	8	7	4	18	13	8
P2	AT2	7	1	5	17	12	15
	AE2	2	6	7	11	13	15
P3	AT3	4	2	1	18	13	8
	AE3	1	5	6	11	13	21

(b)

Figure 1. An architecture model. (a) An architecture with three heterogeneous processors, each is embedded with dynamic random access memory (DRAM) and solid state drives (SSD). (b) Access times and energy consumption for transmitting one unit of data between processors and hybrid memories.

DOI: 10.1002/cpe

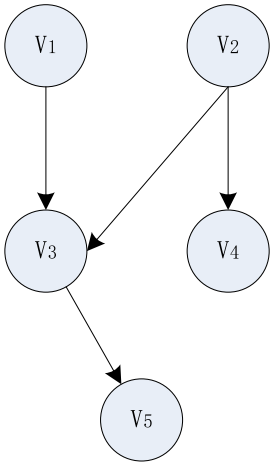
DRAM access or a local *SSD* access, respectively. And a processor access to a *DRAM* or an *SSD*, which belongs to the other processor, we call it as a remote *DRAM* access or a remote *SSD* access. However, the local *DRAM* memory or the local *SSD* memory has shorter time latency than the same type of remote memory. And the remote *DRAM* memory has shorter latency than the local *SSD* memory. For example, consider the processor P_1, whose local hybrid memory consists of $DRAM_1 + SSD_1$, taking $DRAM_2 + SSD_2$ and $DRAM_3 + SSD_3$ as the accessible remote hybrid memories. And the processor P_2, $DRAM_2 + SSD_2$ is the local memory, while $DRAM_1 + SSD_1$ and $DRAM_3 + SSD_3$ are the remote memories. As we all know, every processor has the ability to access the full memory address space in the distribute memory system. Because of the heterogeneous memories and a scalable interconnection network, the structure of memory access in our architecture model is non-uniform. We assume that local access can overlap with remote access.

Totally, different processors accessing to a data in the same memory show different access times and energy consumption. In our model, the local memories are heterogeneous and different from each other in terms of capacity, access concurrency, access time, energy consumption, and other characteristics. We define the access time (AT) as follow: the operation of a processor accessing a memory denotes as $P \rightarrow M$, M contains *DRAM* and *SSD*, where $AT(P_i, M_j)$ is the time for processor P_i to access a unit data from memory M_j. We describe access energy as a function AE: a processor accessing a memory denoted as $P \rightarrow M$, M contains *DRAM* and *SSD*, where $AE(P_i, M_j)$ is the energy consumption of processor P_i to access a unit data from memory M_j.

3.2. Directed acyclic graph model

In this section, we define *DAG* based on [6, 8], and [15]. We describe the memory-access data flow graph ($MDFG$) model, which is used to model an application executed on heterogeneous hybrid $DRAM + SSD$ memory multiprocessor systems. Before we formally describe the $MDFG$ model for the heterogeneous task scheduling and data assignment problem, we introduce a *DAG* model as shown in Figure 2.

Definition 2.1
A *DAG* is denoted by $G = \{ V, E, D, input, output, ET, EE \}$, which is a node-weighted directed graph, where $V = \{ v_1, v_2, ..., v_n \}$ represents a set of tasks, and $E = V \rightarrow V$ is a set of edge. Each edge $e(v_i, v_j) \in E$ represents precedence relation between nodes v_i and v_j. D is made up



(a)

Task	Input data	Output data
V1	A(7KB), B(8KB)	D(8KB)
V2	B(8KB), C(3KB)	E(7KB), F(5KB)
V3	D(8KB), E(7KB)	G(13KB)
V4	F(5KB)	H(12KB)
V5	C(3KB), G(13KB)	I(16KB)

(b)

Figure 2. An input directed acyclic graph (DAG).(a) precedence constraints among tasks. (b) the input data and output data of each task.

DOI: 10.1002/cpe

Table I. The execution time and energy consumption of tasks in the input graph.

	P_1		P_2		P_3	
Task	ET_1	EE_1	ET_2	EE_2	ET_3	EE_3
v_1	4	8	5	6	3	10
v_2	8	19	6	20	10	18
v_3	6	12	4	14	8	13
v_4	10	22	8	20	12	24
v_5	7	21	10	18	13	16

of two parts data, respectively, *input* (v_i) is a set of input data of task v_i, and *output* (v_i) is a set of output data of task v_i. $ET(v_i)$ is used to describe the execution times of task $v_i \in V$ on different processors, namely, $ET(v_i) = \{ et_1(i), et_2(i), ..., et_n(i) \}$, 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., $EE(v_i) = \{ ee_1(i), ee_2(i), ..., ee_n(i) \}$, where $ee_j(i)$ is the energy consumption of v_i on processor P_j. Figure 2 shows a *DAG* application. In this example, there are five tasks, that is, $V = \{ v_1, v_2, v_3, v_4, v_5 \}$.

Figure 2(a) shows the precedence constraints among the tasks. In the *DAG*, the set of $D = \{ A, B, C, D, E, F, G, H, I \}$, and Figure 2(b) depicts the input and output information of tasks. For example, task v_1 reads input data A and B before it starts and writes output data D after it finished.

Table I shows the execution time and energy consumption of each task in the DAG of Figure 2. For example, the value in the cell of column ET_1 and row v_2 indicates that the execution time of task v_2 is eight time units when it is executed on processor P_1, and the value in the cell of column EE_1 and row v_2 indicates that the energy consumption of task v_2 is 18 energy units when it is executed on processor P_1.

Considering each task accessing data from memory, we can treat a memory access operation as a node. Hence, a *DAG* is redefined to obtain a $MDFG$.

Definition 2.2

A *DAG* adds a series of task access memory operation to form a $MDFG$, which is represented by $G' = \{V, U, E, D, var, P, M, AT, AE, ET, EE\}$, where $V = \{v_1, v_2, \ldots, v_{N_1}\}$ is a set of N_1 task nodes, and $U = \{u_1, u_2, \ldots, u_{N_2}\}$ is a set of N_2 memory access operation nodes. $E \in W \times W (W = V \times U)$ is a set of edges. Assume that a task access operation μ has to be executed before a task access operation ω, the dependency between node μ and node ω is represented by an *edge* $(\mu, \omega) \in E$. D is a set of data. And $var : V \times U \times D \rightarrow \{$ true, false $\}$ is a binary function, where $var(v_i, u_l, h)$ denotes whether the memory access operation $u_l \in U$ is transmitting datum $h \in D$ for task $v_i \in V$. $P = \{P_1, P_2, \ldots, P_n\}$ is a set of processors, and $M = \{DRAM_1, SSD_1, DRAM_2, SSD_2, \ldots, DRAM_n, SSD_n\}$ is a set of local memories. AT and AE are access time and access energy functions. $ET(v_i, P_j) = et_j(i)$ denotes the execution time of the task v_i on processor P_j, and $EE(v_i, P_j) = ee_j(i)$ is the energy consumption of task v_i when it is executed on processor P_j.

A $MDFG$ of the *DAG* in Figure 2 is shown in Figure 3, where we have $N_1 = 5$ task nodes and $N_2 = 15$ memory access operation nodes.

3.3. Completion time and energy consumption

In this section, according to the policy of task scheduling and data assignment, we formulate two main formulas about the completion time and the energy consumption. Before describing the completion time formula and the total energy consumption formula, we provide some notations in Table II.

DOI: 10.1002/cpe

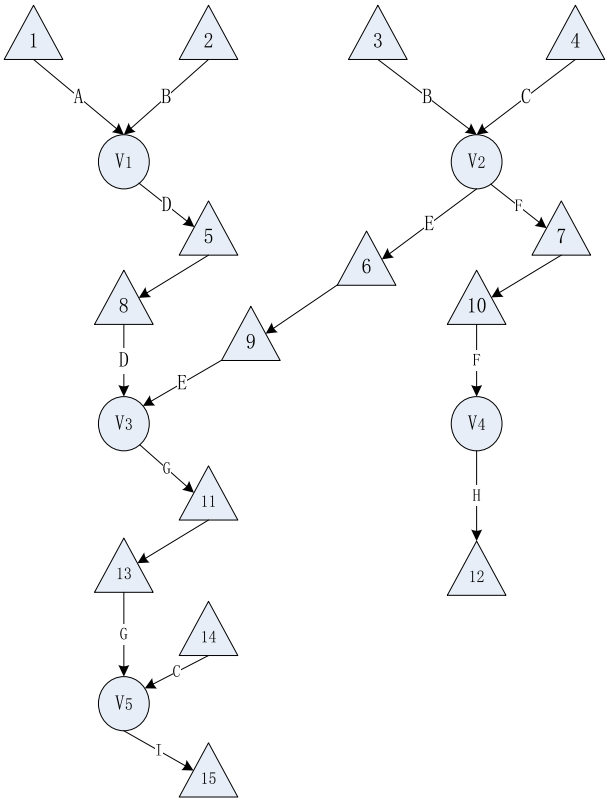


Figure 3. A $MDFG$ obtained from the example input graph.

Table II. Notations used in the formulation.

Notation	Definition
i	Task number
j	Processor number
N_t	Number of task nodes
N_a	Number of memory access operation nodes
n	Number of processors
l	Memory access operation
m	Memory number

The completion time T can be denoted as:

$$T = \sum_{i=1}^{N_t} \sum_{j=1}^{n} E\_T(i,j) + \sum_{l=1}^{N_a} \sum_{m=1}^{2n} A\_T(l,m). \tag{1}$$

The total energy consumption EC is denoted as:

$$EC = \sum_{i=1}^{N_t} \sum_{j=1}^{n} E\_E(i,j) + \sum_{l=1}^{N_2} \sum_{m=1}^{2n} A\_E(l,m). \tag{2}$$

3.4. Problem statement

Given a heterogeneous hybrid memory multiprocessor system that consists of n heterogeneous processors $P_1, P_2, \ldots, P_n$, and each processor P_i contains a hybrid memory, which is equipped with $DRAM_i$ and SSD_i. The access time and access energy of each processor in accessing a unit data

DOI: 10.1002/cpe

from a local hybrid memory are known in advance. The heterogeneous task scheduling and data assignment problem, is defined as follows. Given a *DAG* $G = \{V, E, D, input, output, ET, EE\}$, time constraint S and a *MDFG* $G' = \{V, U, E, D, var, P, M, AT, AE, ET, EE\}$ as input data. The goal of this paper is to find an appropriate assignment policy for both tasks and data on a set of processors and hybrid memories, and satisfied the time constrained, finally, decrease the energy consumed. To achieve the solution, our proposed methods need to solve the following problems:

- a task scheduling $SC : V \rightarrow P$, where $SC(v_i)$ is the processor to execute task $v_i \in V$;
- a data assignment $Mem : D \rightarrow M$, where $Mem(h) \in M$ is the memory to store $h \in D$;
- a schedule, that is, the starting time of each task in V and each memory access operation in U, such that the completion time of the MDFG G' satisfies the constraint $T < S$ and achieves the minimal total energy consumption EC in this time constraint.

4. A MOTIVATIONAL EXAMPLE

In this section, we use an example to illustrate the effectiveness of our algorithm. The example in Figure 2 is executed on the heterogeneous hybrid $DRAM + SSD$ memory multi-processor system shows in Figure 1. The results are shown in Figure 4(a) and (b).

The example of the $MDFG$ of input data is shown in Figure 3. And the result of a greedy algorithm is shown in Figure 4(a). This schedule approach aims at getting the shortest time to complete each task without considering the energy consumption of tasks and data allocation. Hence, in this schedule, task v_1 and task v_5 are scheduled on P_3 and P_1, respectively. Tasks v_2, v_3, and v_4 are scheduled on P_2. The data B, C, and D are allocated to $DRAM_1$. The data A, E, F, and H are allocated to $DRAM_2$, and data D is allocated to $DRAM_3$. The size of *DRAM* [1–3] is not enough to save data G and I. Hence, considering the access time, data I and G are allocated to SSD_1 and SSD_3, respectively. The completion time of this schedule is 608. In Figure 4(a), each point-in-time only shows the logic relationship. The total energy consumption of this schedule is 1679.

However, this approach may not produce a good result in terms of energy consumption, because the approach only considers completion time. Therefore, we propose a new algorithm, which considers energy consumption to obtain a better schedule strategy.

Figure 4(b) shows an improved scheduling algorithm, which considers energy consumption together with a time constraint S = 610. In this schedule, task v_2 and v_4 are scheduled on P_1,

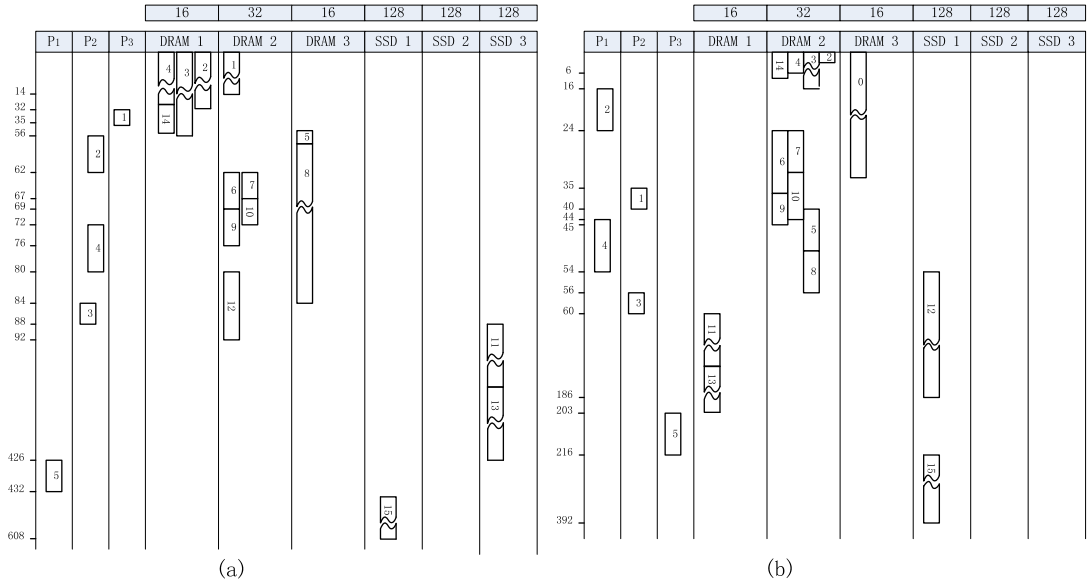


Figure 4. The example of data assignment and task scheduling.(a) a greedy scheduling algorithm. (b) An improve scheduling algorithm.

DOI: 10.1002/cpe

tasks v_1 and v_3 are scheduled on P_2, and tasks v_5 is scheduled on P_3. The data G is allocated on $DRAM_1$, data B, C, D, E, and F are allocated on $DRAM_2$, data A is allocated on $DRAM_3$. Because the size of *DRAM* [1–3] is not enough to save data H and I, considering access time and energy consumption, the data H and I are allocated to SSD_1. The completion time of this improved schedule is 392.

In Figure 4(b), each point-in-time only shows the logic relationship and can not exactly express the true length of time. As shown in Figure 4, the total energy consumption of the schedule strategy is 1159. The schedule has less completion time, and energy consumption is also lower than the greedy schedule. The energy consumption of the schedule is reduced by (1679−1159)/1679 = 30.97% compared with the greedy schedule, while the time constraint $S = 610$ is satisfied.

From the aforementioned example, we can conclude that a heterogeneous architecture has more choices and challenges than a homogeneous architecture in selecting a appropriate processor for a task without violating the time constraint. Therefore, it is important to study the task scheduling and data assignment problem on heterogeneous multiprocessor systems.

5. HEURISTIC ALGORITHMS

In this section, we propose two heuristic algorithms called *IG* algorithm and DAA_TS algorithm. Both of the algorithms adopt *the greedy algorithm*, which allocates a task to the machine with minimum completion time, in the initial scheduling phase. *The greedy algorithm* is HEFT [7] algorithm, which is used to solve the heterogeneous task scheduling problem for real-time bounded number processor applications. It has two main steps, which are as follows.

5.1. Improvement greedy algorithm

The *IG* algorithm is shown in *Algorithm* 1, which mainly considers the task scheduling and data assignment. *IG* algorithm is a straightforward heuristic algorithm and consists of two parts. In the first phase (Step 4), adopting *HEFT* to schedule tasks, and according to the greedy principle, we assign data d to the memory, which can be accessed by shorter time. In the second phase, we only consider one target, that is, the finish time T. Because of the time constraint S, if the finish time T in the first phase is less than the time constraint S, we obtain the result. However, if T greater than S, we should adjust task and data allocation until T less than S.

We need to state some symbols: M consists of with $DARM$ and *SSD*; D is a set of data; a task v has a data d denotes $v.d$; $v.p$ means a task v executed on the processor P; the operation of data d is assigned to memory m, which can be denoted $d.m$.

In the second phase (Steps 5–15), we should find an allocation for each data according to the greedy principle, which is a suitable access operation for tasks accessing the input data. So according to the priority order of tasks, we are sorting data and pushing them into a queue R. Because the data can be needed by different tasks, more than one memory access operation may be associated with the data. Following a priority principle, each data stores in a fixed memory space. Different d has different M, because the allocation of d depends merely on the first task that needs it.

After solving the task scheduling and data assignment problem, we need to detect whether the total completion time T meets the time constraint S. If the total completion time T satisfies the time constraint S, we will obtain a solution; otherwise, we should reallocate some tasks and data (Steps 16–34). To reduce the access time, we reallocate some tasks and data.

However, the *IG* algorithm may not be able to obtain a suitable solution. Because a data allocation is only considered the precedence relationship with tasks and not considered the other complex relationship, like some data accessed more than once, with tasks. Data are divided into share data and single data. Because share data assignment needs to take the data related to tasks into account, it will be accessed for many times. On the other hand, the single data assignment only has relationship with the speedup of the processor to access the memory. Each time the single data is allocated in the memory, which has shortest time accessed by the processor. To solve this problem, we present the DAA_TS algorithm, which considers the types of data, as mentioned before.

DOI: 10.1002/cpe

Input: (1) G (G'); (2) S.
Output: (1) EC.

```
    Build a priority queue Q of tasks;
 2: Build a priority queue R of data;
    Build a priority queue M of memory;
 4: Adopt HEFT scheduling algorithm to the task scheduling;
    while (v_i ← Q) do
 6:   for (v_i.d ← a set of input data D) do
        Build v_i.p a priority access memory queue M;
 8:     while (m ← M) do
          if (v_i.d == -1 and m has space) then
10:         Assign v_i.d into m;
          end if
12:     end while
      end for
14: end while
    Calculate EC and T;
16: if (T > S) then
      for (v_i ← Q) do
18:     for (each P_j) do
          Schedule the v_i to the P_j, recalculate T;
20:       if (T > S) then
            for (each d ← R) do
22:           for (m ← M & T > S) do
                Assign d to m, recalculate T;
24:           end for
            end for
26:       else
            Get the result of energy consumption EC;
28:       end if
        end for
30:   end for
    else
32:   Get the result of energy consumption EC;
    end if
```

Alg. 1: $\mathcal{I}$mprovement $\mathcal{G}$reedy ($\mathcal{IG}$)

To find a better data allocation, it takes at least $O(2N_tN_d\mathrm{N_m})$ time and at most $O(N_tN_d(\mathrm{N_p}+\mathrm{N_m}))$ time to obtain satisfying results T and EC, where $\mathrm{N_m}$ is the number of local memories. Thus, the second phase takes at most $O(N_tN_d(\mathrm{N_p}+\mathrm{N_m}))$ to obtain a better data allocation. If $\mathrm{N_p}$ and $\mathrm{N_m}$ are treated as a constant, the time complexity of the *IG* algorithm is $O(N_t(N_d+N_t))$.

5.2. *Data assignment according to the task scheduling algorithm*

The *DAA_TS* algorithm is shown in *Algorithm* 2, which consists of four phases: task scheduling phase, data-aware phase, data scheduling phase, and optimization phase. The algorithm is a straightforward heuristic algorithm. The first phase aims to find a better processor, which takes minimal time to execute task for each task node (Step 2). The second phase based on the relationship between tasks and data, to find share data and single data (Step 3). The third phase aims to find a better memory for each data, which can be accessed with minimal time (Steps 4–7). With the purpose of the four phase is reallocated tasks and data to obtain the much less energy consumption and spend the time less than the constraint S (Steps 8–18).

DOI: 10.1002/cpe

Input: (1) G (G'); (2) S; (3) $\mathcal{DA}(share\_d, single\_d)$.
Output: (1) EC.

```
Build a queue M of memory;
Adopt HEFT scheduling algorithm to the task scheduling;
Call the Data − Aware Algorithm;
for (each d ∈ D) do
    Call the Data Assignment Algorithm;
end for
Calculate T and EC;
if (T > S) then
    Call the Greater Algorithm;
    Calculate T and EC;
    if (T < S) then
        Call the Less Algorithm;
    else
        Call the Greater Algorithm;
    end if
else
    Get the result of the energy consumption EC;
end if
```

Alg. 2: $\mathcal{D}$ata $\mathcal{A}$ssignment $\mathcal{A}$ccording to the $\mathcal{T}$asks $\mathcal{S}$cheduling ($\mathcal{DAA_TS}$)

Input: (1) G (G').
Output: (1) $share\_d$; (2) $single\_d$.

```
Read benchmark to make a matrix task_task, each edge has a value it denotes weight;
for (i=0; i<data; i++) do
    for (j=0; j<task; j++) do
        Utilize a matrix task_task generate a matrix datain[i][j] (the in-degree data of task);
        Utilize a matrix task_task generate a matrix dataout[j][i] (the out-degree data of task);
    end for
end for
Utilize task_task we can get each task inputdata and outputdata;
for (i=0; i<data; i++) do
    for (j=0; j<task; j++) do
        share = share+datain[i][j];
    end for
    data[i].share = share;
    if (data[i].share>1) then
        Define data[i] as share_d;
    else
        Define data[i] as single_d;
    end if
end for
```

Alg. 3: $\mathcal{D}ata$-$\mathcal{A}ware\ \mathcal{A}lgorithm$ ($\mathcal{D} - \mathcal{A}$)

In DAA_TS algorithm, we first use the *HEFT* algorithm to initialize the schedule. And then, we propose *Data-Aware Algorithm*, which is denoted in *Algorithm* 3 to find the share data and the single data. The share data and the single data are denoted as $share\_d$ and $single\_d$, respectively. We can save much more cost to first assign $share\_d$, because the memory store $share\_d$ will be accessed more than once.

After identifying the type of data, we propose *Data Assignment Algorithm* described in *Algorithm* 4. In *Algorithm* 4, we build two queues, one is a memory queue denoted as M and other one is a

DOI: 10.1002/cpe

Input: (1) G (G'); (2)$\mathcal{D} - \mathcal{A}(share\_d, single\_d)$.
Output: (1)A near-optimal data allocation; (2) T; (3)EC.

```
for (each v) do
    Build a queue M;
    Build the data queue T_D of the task;
    if (d ← T_D is share data) then
        Use address matching to find tasks which own the share data;
        for (m ← M) do
            for (the tasks which own the share data) do
                Calculate sum_{l=1}^{N_2} at(l, d, m);
            end for
            Find minimal of sum_{l=1}^{N_2} at(l, d, m);
        end for
    end if
    if (d ∈ single data) then
        while (m ← dequeue M)&&(m has space) do
            Put d into m;
        end while
    end if
end for
```

Alg. 4: $Data\ Assignment\ Algorithm\ (\mathcal{DAA})$

data queue of each task denoted as T_D. The memory queue M is sorted by the access memory speed of $v.p$. The queue T_D presents the importance of processing sequence of the input data of a task. The share data is needed by different tasks, more than one memory access operation may be associated with the data. However, the single data is needed by a task, only if one memory access operation is associated with the data.

So we assign the share data to a memory with the minimum total access time $\sum_{l=1}^{N_2} at(l,d,m)$ by the processors. Therefore, the access memory m and the operation l, which is related with data d can be denoted as :

$$at(l,d,m) = \sum_{i=1}^{N_t} \sum_{k=1}^{S} y_{l,m,k} var(i,l,d) AT(P(i), M(j)) d(d).$$

For each single data, the data is always placed onto the processor to access the fastest and spatial memory. The data assignment rule is reflected on *Algorithm* 4.

According to the aforementioned description, we can obtain a strategy of tasks scheduling and data assignment. However, this result is not the most ideal. In order to satisfy the constraints of the time and energy consumption, the DAA_TS algorithm is also contained in the following optimization algorithm. The optimization algorithm consists of $Less\ Algorithm$ and $Greater\ Algorithm$. $Less\ Algorithm$ is *Algorithm* 5 and $Greater\ Algorithm$ is *Algorithm* 6 aiming at reducing the energy consumption EC and the completion time T without violating the time constraint S.

In the optimization phase, according to the time of a task execute on each processor, we build a queue PQ for each task. PQ is according to the sorting of the speed of the processors to perform tasks. When a task needs to change processor, processor P dequeues from PQ, and reallocates data. Before changing allocation of each data, we should ensure each task scheduling is completed. And then, we build a queue of memory access speed denoted as M, which is sorted by the processor of task access speed.

DOI: 10.1002/cpe

```
Input:  (1) S; (2) Current tasks and data allocation.
Output: (1) T; (2) EC.
   for (each d) do
 2:  for (each m) do
       Change the d allocation to the m;
 4:    Recalculate T and ec;
       if (m has space & T < S & ec < EC) then
 6:      Assign d to m and EC equals ec;
       end if
 8:  end for
   end for
```

Alg. 5: $\mathcal{L}ess\ \mathcal{A}lgorithm$ ($\mathcal{LA}$)

```
Input:  (1) S; (2) Current tasks and data allocation.
Output: (1) T; (2) EC.
   for (each task v ← Task) do
 2:  for (P ← PQ) do
       Change the position of the task v to the P;
 4:    Recalculate T;
       if (T > S) then
 6:      for (each data d) do
           for (each memory m) do
 8:          Change the data d allocation to the memory m;
             Recalculate T;
10:          if (T > S) then
               Continue;
12:          else
               Call Less algorithm();
14:          end if
           end for
16:     end for
       else
18:     Call Less algorithm();
       end if
20:  end for
   end for
```

Alg. 6: $\mathcal{G}reat\ \mathcal{A}lgorithm$ ($\mathcal{GA}$)

Then, each single data is always placed in the processor to access the fastest and spatial memory. As a result, we draw the tasks and data schedule scheme. However, this result was not the most ideal. In order to satisfy the time constraint and the minimal energy consumption with the resource constraint, the DAA_TS algorithm is also contained in the following optimization algorithm to reduce the energy consumption EC and the completion time T.

At first, we change tasks and data position to achieve the energy consumption of minimal min_E. And then, the task scheduling and data assignment problem is solved, if the total completion time T satisfies the time constraint S. Otherwise, for reaching the time constraint S requirement, tasks and data should be reallocated. After $T < S$, we will adopt tasks migration. While reallocating a set of data D, until the energy consumption cannot be reduced any more. Otherwise, we should reallocate some tasks and some data to guarantee the finish time $T <$ the constraint time S, and then, to reduce the energy consumption.

The DAA_TS 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.

DOI: 10.1002/cpe

The time complexity of DAA_TS algorithm is $O(N_t^2 N_p + N_d(N_m N_d N_t + N_m + N_t))$. If N_p and N_m are treated as constants, the DAA_TS algorithm takes $O(N_t(N_d^2 + N_d + N_t))$ time.

6. PERFORMANCE EVALUATION

This section includes two parts. Firstly, we describe the experimental apparatus. And then, we show the results of evaluating the effectiveness of the proposed algorithm on different systems.

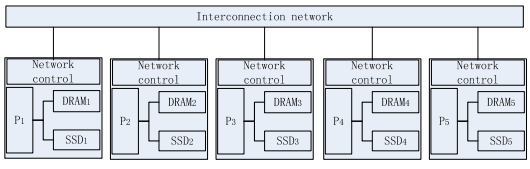
6.1. Experimental apparatus

In our experiments, there are four benchmarks including 2IIR, 2-rls-lat, 2-volt, and 3-8lattic-iir to be used. These benchmarks are from $DSPstone$ [16], and frequently used on multi-core systems [8]. These benchmarks can be compiled with gcc to extract the task graphs and the read/write data sets from gcc. There are three steps to generate the benchmark. First, we need to compile the source codes with profileing(-fprofile-generate). And then, we execute the compile binary with a data set corresponding to the use case. Finally, we compile the source code again with both profile-guided optimization and ABSINTH enabled (-fprofile-use-fabsinth). The number of tasks, data, and the average of data are shown in Table III.

We develop a custom simulator to conduct the experiments. The simulator is similar to the simplescalar [17], which is developed to simulate the process of tasks, and to obtain the data distribution, task execution time, and energy costs. All the experiments for DAGs are executed on two different architecture simulate models, which are proposed in Section 2.1. The model 1 is composed of three heterogeneous processors shown in Figure 1. Each DRAM is equipped with an SSD. Figure 5 shows the model 2, which consists of five heterogeneous processors and each DRAM is equipped with an SSD. The parameters of the two are collected from ARM7 and MSP430 by using the CACTI tools [18] provided by HP. The set of parameters are displayed in Table IV. The row '$Time\ Latency$' and '$Energy\ Consumption$' show the wake-up time and wake-up energy of each processor, respectively. The access time per unit data is $Time\ Latency + \varepsilon_{FS} \times d$. 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 of randomly.

Table III. Sizes of DSPstone benchmarks.

Benchmark	**Tasks**	**Data**	$\overline{D_{DFG}}$
$2IIR$	18	19	25.26
$2-rls-lat$	40	46	22.10
$2-volt$	54	65	20.01
$3-ilattir$	128	141	24.13



(a)

MEMORY		DRAM1	DRAM2	DRAM3	DRAM4	DRAM5	SSD1	SSD2	SSD3	SSD4	SSD5
P1	AT1	1	4	3	4	2	11	13	17	21	18
	AE1	2	3	3	6	2	15	12	18	22	9
P2	AT2	3	1	4	6	3	15	11	14	19	23
	AE2	4	2	5	7	3	21	10	7	17	24
P3	AT3	4	4	1	7	4	22	22	13	25	15
	AE3	6	7	1	10	9	14	16	16	19	19
P4	AT4	5	5	6	2	5	11	15	15	11	13
	AE4	8	9	8	2	13	18	18	16	19	21
P5	AT5	2	3	2	4	1	26	27	24	18	14
	AE5	3	3	2	5	1	13	18	17	14	12

(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 memories.

DOI: 10.1002/cpe

Table IV. System specification for Models 1 and 2.

	Model 1			Model 2				
Parameter	Core 1	Core 2	Core 3	Core 1	Core 2	Core 3	Core 4	Core 5
Frequency	30MHz	64MHz	64Mhz	6MHz	30MHz	7.5MHz	64MHz	64MHz
DRAM size	64KB	128KB	128KB	32KB	64KB	32KB	128KB	64KB
SSD size	512KB	512KB	1024KB	256KB	512KB	256KB	1024KB	512KB
Time latency	1.400ms	1.400ms	1.225ms	1.225ms	2.787ms	2.781ms	3.781ms	0.876ms
Energy consumption	1.47mJ	0.1mJ	0.1mJ	0.593mJ	1.359mJ	1.849mJ	2.187mJ	0.252mJ

DRAM, dynamic random access memory; SSD, solid state drives.

Table V. The results of the four algorithms on the first system with three heterogeneous processors.

		Greedy	IG		DAA_TS		
Benchmark	Time Constraint	energy	energy	%(greedy)	energy	%(greedy)	%(IG)
	2000	14533.0	9915.0	31.78	9575.0	34.16	3.43
	2100	13769.0	9966.0	27.62	8911.0	35.28	10.59
2IIR	2200	12909.0	9725.0	24.66	8843.0	31.50	9.07
(18)	2300	10076.0	8870.0	9.34	8530.0	15.34	3.83
	2400	9901.0	8392.0	15.24	7875.0	20.46	6.16
	8000	43230.0	36018.5	16.68	—	—	—
	8500	37534.9	24218.8	35.48	22786.1	39.29	5.90
2-rls-lat	9000	36209.0	23871.6	34.07	21706.9	40.05	9.07
(40)	9500	34420.9	24530.0	28.74	20797.8	39.58	15.21
	10000	33060.1	21682.9	34.41	19414.5	41.28	10.46
	6300	41360.6	38370.4	7.23	31478.1	23.89	17.96
	6600	40046.4	35798.4	10.61	31156.4	22.20	12.97
2-volt	6900	39311.4	34422.1	12.44	30524.6	22.35	11.32
(54)	7200	37816.3	32033.2	15.29	30087.7	20.43	6.07
	7500	37043.7	31380.9	15.29	29912.8	19.25	4.68
	8200	119829.3	92137.2	23.11	77451.9	35.36	15.94
	8300	112137.9	87782.01	21.72	75068.8	33.06	14.48
3-8lattic-iir	8400	98240.2	81823.8	16.71	74100.4	24.57	9.44
(128)	8500	87081.9	77240.2	1130	73498.8	15.60	4.84
	8600	83433.8	74038.2	11.26	72137.9	13.54	2.57
Average				20.15		27.74	9.16

IG, improvement greedy; DAA_TS, data assignment according to the task scheduling.

6.2. Experimental results and analysis

We employ two different architecture models shown in Figures 1 and 5 to demonstrate the effectiveness of the *IG* algorithm and the DAA_TS algorithm. We measure the schedule effectiveness by the completion time and the energy consumption. In the two sets of experiments, our algorithms are compared with *the greedy algorithm* [7]. *The greedy algorithm* through iterative way to improve the effectiveness. However, *the greedy algorithm* only considers task scheduling, which a processor has the fastest completion time to execute the task rather than considering the near-optimal of the data allocation.

DOI: 10.1002/cpe

This is a traditional task scheduling algorithm. In Table V, the experimental results on three processors architecture model are shown. The 'TC' column represents the time constraint. The '%(*greedy*)' column shows the schedule energy consumption improvement of *IG* and DAA_TS over *the greedy algorithm*. Each '−' represents that the algorithm cannot obtain a solution under the time constraint. Table V shows that our algorithms *IG* and DAA_TS can achieve better performance than $greedy$, and DAA_TS has better performance than *IG*. The reduction rates of our heuristic algorithm compared with *the greedy algorithm* are $(E_g - E)/E$, where E_g denotes the energy consumption of *the greedy algorithm* and E represents the energy consumption of our heuristic algorithm. From the row '*Average*', the *IG* and DAA_TS algorithms reduce the total energy consumption by 20.15% and 27.74% on the average compared with *the greedy algorithm*, respectively. The reduction rates of *IG* compared with DAA_TS are $(E_{IG}\text{-}E_{DAA})/E_{IG}$, where E_{IG} is the energy consumption of the *IG* algorithm and E_{DAA} indicates the energy consumption of the DAA_TS algorithm. From the '$Average$', the average reduction rates of the total energy consumption of the DAA_TS is 9.16%.

Figure 6 shows the second group of the experimental results, which are obtained by running a set of simulation on all benchmarks based on the architectural model shown on Figure 5. From the Figure 6, we can see, with the extension of the time constraint, the energy consumption of all the three algorithms decrease. We know that the energy consumption of the *IG* and DAA_TS algorithms are less than that of *the greedy algorithm*. The *IG* and DAA_TS algorithms reduce the total energy consumption by 9.8% and 18.53% on the average compared with *the greedy algorithm*, respectively. Therefore, the *IG* and DAA_TS algorithms are superior to *the greedy algorithm*. Furthermore, form the figure, we are able to compare *IG* algorithm and DAA_TA algorithm. We can see that the energy consumption of DAA_TS algorithm is less than that of the *IG* algorithm.

From the two groups of experimental results, we know that the DAA_TS algorithm is superior to the *IG* algorithm in the time constraint. However, if the time is enough, the gap of energy

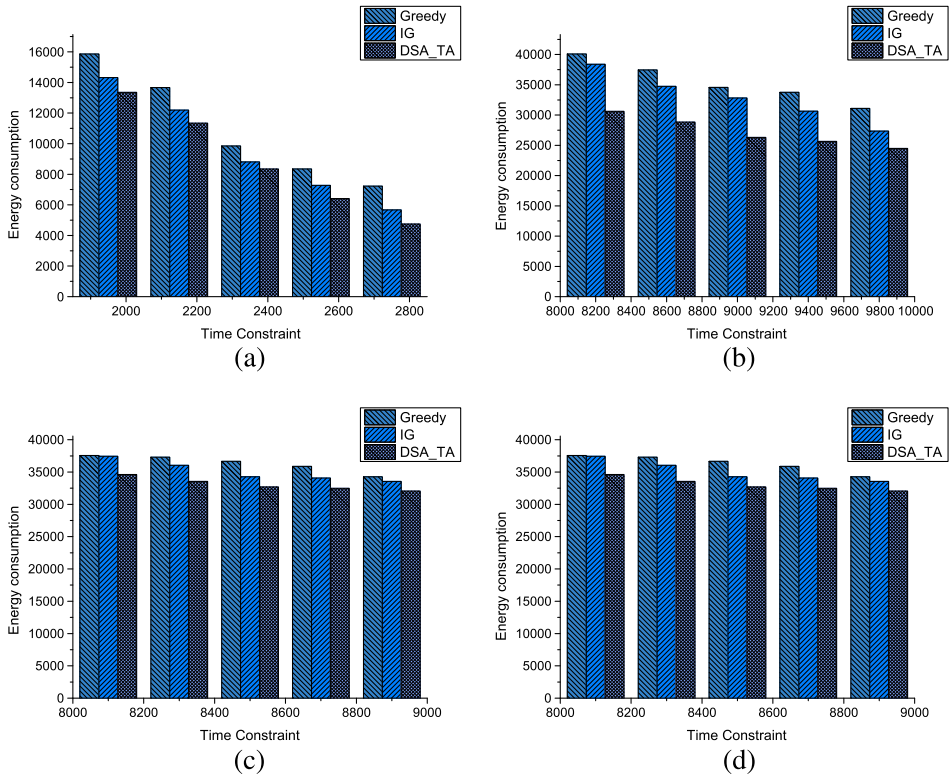


Figure 6. The results of the three algorithms on the second system with five heterogeneous processors. (a)2-IIR. (b)2-rls-lat. (c)2-volt. (d)3–8lattir.

DOI: 10.1002/cpe

consumption will be so small. For example, looking at the energy consumption of Figure 6(a) and Figure 6(b), the energy consumption all have the trend to be minimized.

7. CONCLUSION

In this paper, we study the problem of task scheduling and data assignment on heterogeneous multiprocessor systems with hybrid memory *DRAM* and *SSD*. We propose two efficient heuristic algorithms, which are Improvements Greedy algorithm and Data Assignment According to the Task Scheduling to generate near-optimal results in polynomial time. In the experiments conducted on two systems, the results show that the heuristic algorithms are more effective than *the greedy algorithm* without considering the data allocation and demonstrate that considering the data allocation in the process of scheduling can greatly reduce the energy consumption. Moreover, the DAA_TS algorithm is superior to the *IG* algorithm. Further research can be directed towards finding more effective and efficient algorithms with reduced time complexity and improved energy efficiency.

ACKNOWLEDGEMENTS

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Natural Science Foundation of China (Grant Nos. 61-370095, 61472124), International Science & Technology Cooperation Program of China(2015DFA11240), and the National High-tech R&D Program of China (Grant Nos. 2014AA01A302, 2015AA015303).

REFERENCES

1. Liu D, Wang T, Wang Y, Qin Z, Shao Z. Pcm-ftl: A write-activity-aware nand flash memory management scheme for pcm-based embedded systems. *2011 IEEE 32nd Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011; 357–366.
2. Jiang D, Che Y, Xiong J, Ma X. uCache: A utility-aware multilevel SSD cache management policy. *IEEE 10th International Conference on High Performance Computing and Communications & Embedded and Ubiquitous Computing (HPCC_EUC)*, Zhangjiajie, China, 2013; 391–398.
3. Chang C-W, Yang C-Y, Chang Y-H, Kuo T-W. Booting time minimization for real-time embedded systems with non-volatile memory. *IEEE Transactions on Computers* 2014; **63**(4):847–859.
4. Hartmanis J. Computers and intractability: a guide to the theory of NP-completeness (michael r. garey and david s. johnson). *Siam Review* 1982; **24**:90–91.
5. Garey MR, Johnson DS, Stockmeyer L. Some simplified np-complete problems. *The Sixth Annual ACM Symposium on Theory of Computing*, ACM, New York, USA, 1974; 47–63.
6. Topcuouglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel Distributed Systems* 2002; **13**(3):260–274.
7. Mei J, Li K, Li K. Energy-aware task scheduling in heterogeneous computing environments. *Cluster Computing* 2014; **17**(2):537–550.
8. Wang Y, Li K, Chen H, He L, Li K. Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints. *IEEE Transactions on Emerging Topics in Computing* 2014; **2**(2): 134–148.
9. Chen Z, Lu Y, Xiao N, Liu F. A hybrid memory built by ssd and dram to support in-memory big data analytics. *Knowledge and Information Systems* 2014; **41**(2):335–354.
10. Alvarez L, Moreto M, Casas M, Castillo E, Martorell X, Labarta J, Ayguade E, Valero M. Runtime-guided management of scratchpad memories in multicore architectures. *IEEE International Conference on Parallel Architecture and Compilation*, San Francisco, 2015; 379–391.
11. Odagiri J, Itani N, Nakano Y, Culler DE. Data compression technology dedicated to distribution and embedded systems. *2010 Data Compression Conference*, Snowbird, Utah, USA, 2010; 549.
12. Casas M, Moreto M, Alvarez L, Castillo E, Chasapis D, Hayes T, Jaulmes L, Palomar O, Unsal O, Cristal A. *Runtime-Aware Architectures*. Springer Berlin Heidelberg: Berlin, Germany, 2015.
13. Wu F, Jin S, Wang Y. A simple model for the energy-efficient optimal real-time multiprocessor scheduling. *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, Zhangjiajie, China, 2012; 18–21.
14. Marchal P, Catthoor F, Gomez JI, Pinuel L, Bruni D, Benini L. Integrated task scheduling and data assignment for sdrams in dynamic applications. *IEEE Design Test of Computers* 2004; **21**(5):378–387.
15. Duran A, Ayguade E, Badia RM, Labarta J, Martinell L, Martorell X, Planas J. Ompss: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters* 2011; **21**(2):173–193.

DOI: 10.1002/cpe

16. Zivojnovic V, Velarde JM, Schlager C, Meyr H. Dspstone: a dsp-oriented benchmarking methodology. *Proceedings of the International Conference on Signal Processing Applications and Technology*, Dallas, Texas, USA, 1994; 715–720.
17. Burger DC, Austin TM. The simplescalar tool set, version 3.0. *Computer Architecture News* 1997; **25**(3):366–375.
18. Kreku J, Tiensyrjä K, Vanmeerbeeck G. Automatic workload generation for system-level exploration based on modified gcc compiler. *Proceedings of the Conference on Design, Automation and Test in Europe*: European Design and Automation Association, Dresden, Germany, 2010; 369–374.

DOI: 10.1002/cpe