Data-Aware Task Scheduling on Heterogeneous Hybrid Memory Multiprocessor Systems

Junjie Chen¹, Kenli Li¹, Zhuo Tang¹, Chubo Liu¹, Yan Wang¹, Keqin Li¹,²

¹School of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha, 410082, China
²Department 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 (DRAM) and Solid State Drives (SSD) when considering the performance of SSD into the scheduling policy. To solve this problem, we propose two heuristic algorithms called Improvement Greedy (IG) algorithm and the Data Assignment According to the Task Scheduling (DAA,TS) 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 © 2015 John Wiley & Sons, Ltd.

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 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 an *SSD* in heterogeneous multiprocessor systems.

*SSD* is widely adopted as a second level memory to assist a main memory like *DRAM*. There are two major advantages. First, compared to *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 have 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 *HEFT* [6] algorithm to schedule tasks. In the *HEFT* algorithm, the each task is scheduled to a processor with the shortest execution time. In the second phase, the share data is 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 *IG* 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 are 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. [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 is 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 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], [13], [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 (RAA) 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 \( P = \{P_1, P_2, ..., P_n\} \), 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 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. Due to 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.

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

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 $AT(P, M_i)$ is the time for processor $P_i$ to access a unit data from memory $M_i$. 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. DAG 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 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 5 tasks, i.e., $V = \{v_1, v_2, v_3, v_4, v_5\}$.

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.
DATA-AWARE TASK SCHEDULING ON HETEROGENEOUS HYBRID MEMORY MULTIPROCESSOR SYSTEMS

![Figure 2. An input DAG.](image)

(a) precedence constraints among tasks. (b) the input data and output data of each task.

Figure 2. An input DAG.

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 \( ET \) and row \( v_2 \) indicates that the execution time of task \( v_2 \) is 8 time units when it is executed on processor \( P_1 \), and the value in the cell of column \( EE \) and row \( v_2 \) indicates that the energy consumption of task \( v_2 \) is 18 energy units when it is executed on processor \( P_1 \).

<table>
<thead>
<tr>
<th>Task</th>
<th>Input data</th>
<th>Output data</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_1 )</td>
<td>A(7KB), B(8KB)</td>
<td>D(6KB)</td>
</tr>
<tr>
<td>( v_2 )</td>
<td>B(8KB), C(3KB)</td>
<td>E(7KB), F(5KB)</td>
</tr>
<tr>
<td>( v_3 )</td>
<td>D(8KB), E(7KB)</td>
<td>G(13KB)</td>
</tr>
<tr>
<td>( v_4 )</td>
<td>F(6KB)</td>
<td>H(13KB)</td>
</tr>
<tr>
<td>( v_5 )</td>
<td>C(3KB), G(13KB)</td>
<td>I(13KB)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Task</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_1 )</td>
<td>( ET_1 ) ( EE_1 )</td>
<td>( ET_2 ) ( EE_2 )</td>
<td>( ET_3 ) ( EE_3 )</td>
</tr>
<tr>
<td>( v_2 )</td>
<td>8</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>( v_3 )</td>
<td>6</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>( v_4 )</td>
<td>10</td>
<td>22</td>
<td>8</td>
</tr>
<tr>
<td>( v_5 )</td>
<td>7</td>
<td>21</td>
<td>10</td>
</tr>
</tbody>
</table>

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 \( G = \{ V, U, D, var, P, M, AT, AE, ET, EE \} \), where \( V = \{ v_1, v_2, ..., v_{N_1} \} \) is a set of \( N_1 \) task nodes, and \( U = \{ u_1, u_2, ..., u_{N_2} \} \) 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 \( \mu \) has to be executed before a task access operation \( \omega \), the dependency between node \( \mu \) and node \( \omega \) is represented by an edge \( (\mu, \omega) \in E \). \( D \) is a set of data. And \( var : V \times U \times D \rightarrow \{ \text{true, false} \} \) is a binary function, where \( var(v_i, u_i, h) \) denotes whether the memory access operation \( u_i \in U \) is transmitting datum \( h \in D \) for task \( v_i \in V \). \( P = \{ P_1, P_2, ..., P_n \} \) is a set of processors, and \( M = \{ DRAM_1, SSD_1, DRAM_2, SSD_2, ..., DRAM_n, SSD_n \} \) 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.

Table II. Notations used in the formulation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>Task number</td>
</tr>
<tr>
<td>$j$</td>
<td>Processor number</td>
</tr>
<tr>
<td>$N_t$</td>
<td>Number of task nodes</td>
</tr>
<tr>
<td>$N_a$</td>
<td>Number of memory access operation nodes</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of processors</td>
</tr>
<tr>
<td>$l$</td>
<td>Memory access operation</td>
</tr>
<tr>
<td>$m$</td>
<td>Memory number</td>
</tr>
</tbody>
</table>

The completion time $T$ can be denoted as:

$$ T = \sum_{i=1}^{N_t} \sum_{j=1}^{n} E_T(i,j) + \sum_{l=1}^{N_a} \sum_{m=1}^{2n} A_T(l,m). $$

(1)

The total energy consumption $EC$ is denoted as:

$$ EC = \sum_{i=1}^{N_t} \sum_{j=1}^{n} E_E(i,j) + \sum_{l=1}^{N_a} \sum_{m=1}^{2n} A_E(l,m). $$

(2)

3.4. Problem Statement

Given a heterogeneous hybrid memory multiprocessor system, which consists of $n$ heterogeneous processors $P_1, P_2, ..., P_n$, 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 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, i.e., 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 Figure 4(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, since 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 $P_1$, 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 $DRAM_3$. Since 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 $f$ 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 above example, we can conclude that a heterogeneous architecture has more choices and challenges than a homogeneous architecture in selecting an 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 (BNP) applications. It has two main steps, 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, i.e., is the finish time $T$. Due to the time constraint $S$, if the finish time $T$ in the first phase is less than the time constraint $S$, we get 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 $R$. Since a 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 is divided into share data and single data. Since share data assignment needs to take the data related to tasks into account, it will
Input:  (1) \( G \) \((G')\); (2) \( S \).
Output:  (1) \( EC \).

1: Build a priority queue \( Q \) of tasks;
2: Build a priority queue \( R \) of data;
3: Build a priority queue \( M \) of memory;
4: Adopt HEFT scheduling algorithm to the task scheduling;

while \((v_i \leftarrow Q)\) do
6: for \((v_i,d \leftarrow \text{a set of input data } D)\) do
   Build \( v_i,p \) a priority access memory queue \( M \);
8: while \((m \leftarrow M)\) do
   if \((v_i,d == -1 \text{ and } m \text{ has space})\) then
      Assign \( v_i,d \) into \( m \);
   end if
10: end while
12: end for
14: end while
16: if \((T > S)\) then
   for \((v_i \leftarrow Q)\) do
      for (each \( P_j \)) do
         Schedule the \( v_i \) to the \( P_j \), recalculate \( T \);
      end for
5: if \((T > S)\) then
   for (each \( d \leftarrow R \)) do
      for (each \( m \leftarrow M \) & \( T > S \)) do
         Assign \( d \) to \( m \), recalculate \( T \);
      end for
24: end for
26: else
   Get the result of energy consumption \( EC \);
28: end if
30: end for
else
32: Get the result of energy consumption \( EC \);
end if

Alg. 1: Improvement Greedy (IG)

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 \textit{DAA}\textit{TS} algorithm, which considers the types of data, as mentioned before.

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

5.2. \textit{DAA}\textit{TS} Algorithm

The \textit{DAA}\textit{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).

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

1. Build a queue \(M\) of memory;
2. Adopt HEFT scheduling algorithm to the task scheduling;
3. Call the Data – Aware Algorithm;
4. for (each \(d \in D\)) do
   - Call the Data Assignment Algorithm;
5. end for
6. Calculate \(T\) and \(EC\);
7. if \((T > S)\) then
   - Call the Greater Algorithm;
8. else
   - Calculate \(T\) and \(EC\);
9. if \((T < S)\) then
   - Call the Less Algorithm;
10. else
    - Call the Greater Algorithm;
11. end if
12. end if
13. Get the result of the energy consumption \(EC\);
14. end if

Alg. 2: Data Assignment According to the Tasks Scheduling (\(DAA \_TS\))

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 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 \(\sum_{l=1}^{N} at(l, d, m)\) by the processors. Therefore, the access memory \(m\) and the operation \(l\) which is related with data \(d\) can be denoted as:

\[
at(l, d, m) = \sum_{i=1}^{N} \sum_{k=1}^{S} y_{i,m,k \_var}(i, l, d)AT(P(i), M(j))d(d).
\]

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 above description, we can get 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 \(S\).
Input:  
(1) $G$ ($G'$).
Output:  
(1) share, (2) single.

Read benchmark to make a matrix task, each edge has a value it denotes weight;

2: for (i=0; i<data; i++) do
   for (j=0; j<task; j++) do
      Utilize a matrix task generate a matrix datain[i][j] (the in-degree data of task);
      Utilize a matrix task generate a matrix dataout[j][i] (the out-degree data of task);
   end for
end for

Algorithm 3: Data-Aware Algorithm (D - A)

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

for (each v) do
   Build a queue M;
   Build the data queue $D$ of the task;
4: if ($d \leftarrow T, D$ is share data) then
   Use address matching to find tasks which own the share data;
6: for ($m \leftarrow M$) do
      for (the tasks which own the share data) do
         Calculate $\sum_{l=1}^{N_2} at(l,d,m)$;
      end for
   10: Find minimal of $\sum_{l=1}^{N_2} at(l,d,m)$;
end for
12: end if
   if ($d \in$ single data) then
      while ($m \leftarrow$ dequeue M) and (m has space) do
         Put d into m;
      end while
   end if
end for

Algorithm 4: Data Assignment Algorithm (DAA)

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.

#### Alg. 5: Less Algorithm ($\mathcal{L}A$)

```
Input:  (1) $S$; (2) Current tasks and data allocation.
Output: (1) $T$; (2) $EC$.

for (each $d$) do
  for (each $m$) do
    Change the $d$ allocation to the $m$;
  end for
  Recalculate $T$ and $ec$;
  if ($m$ has space & $T < S$ & $ec < EC$) then
    Assign $d$ to $m$ and $EC$ equals $ec$;
  end if
end for
```

#### Alg. 6: Great Algorithm ($\mathcal{G}A$)

```
Input:  (1) $S$; (2) Current tasks and data allocation.
Output: (1) $T$; (2) $EC$.

for (each task $v$← Task) do
  for ($P$← $PQ$) do
    Change the position of the task $v$ to the $P$;
  end for
  Recalculate $T$;
  if ($T > S$) then
    for (each data $d$) do
      for (each memory $m$) do
        Change the data $d$ allocation to the memory $m$;
        Recalculate $T$;
      end for
      if ($T > S$) then
        Continue;
      else
        Call Less algorithm();
      end if
    end for
  end if
  else
    Call Less algorithm();
end if
end for
```

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 $T < S$, 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.

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.

Table III. Sizes of DSPstone benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Tasks</th>
<th>Data</th>
<th>$D_{DFG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2IIR</td>
<td>18</td>
<td>19</td>
<td>25.26</td>
</tr>
<tr>
<td>2-rls-lat</td>
<td>40</td>
<td>46</td>
<td>22.10</td>
</tr>
<tr>
<td>2-volt</td>
<td>54</td>
<td>65</td>
<td>20.01</td>
</tr>
<tr>
<td>3-ilattir</td>
<td>128</td>
<td>141</td>
<td>24.13</td>
</tr>
</tbody>
</table>

6.1. Experimental Apparatus

In our experiments, there are four benchmarks including 2IIR, 2-rls-lat, 2-volt and 3-ilattic-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 profiling (-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.

Table IV. System specification for Model 1 and Model 2

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Core 1</td>
<td>Core 2</td>
</tr>
<tr>
<td>Frequency</td>
<td>30MHz</td>
<td>64MHz</td>
</tr>
<tr>
<td>DRAM size</td>
<td>64KB</td>
<td>128KB</td>
</tr>
<tr>
<td>SSD size</td>
<td>512KB</td>
<td>512KB</td>
</tr>
<tr>
<td>Time latency</td>
<td>1.400ms</td>
<td>1.400ms</td>
</tr>
<tr>
<td>Energy consumption</td>
<td>1.47mJ</td>
<td>0.1mJ</td>
</tr>
</tbody>
</table>

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 a Solid State Drives (SSD). Figure 5 shows the model 2, which consists of five heterogeneous processors and each DRAM is equipped with a Solid State Drives (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 $\text{Time Latency} + \varepsilon_{FS} \times d$. The access energy per unit data is $E_{ele} + \varepsilon_{FS} \times d^4$, where $E_{ele}$ is the wake-up energy, and $\varepsilon_{FS}$ is the transmit amplifier parameter, whose value is set of randomly.
6.2. Experimental Results and Analysis

We employ two different architecture models shown in Figure 1 and Figure 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. This is a traditional task scheduling algorithm. In Table V, the experimental results on 3 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 \((E_{IG} - E_{DAA})/E_{IG}\), 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 algorithm and DAA_TS algorithm are less than that of the greedy algorithm. The IG algorithm and DAA_TS algorithm reduce the total energy consumption by 9.8% and 18.53% on the average compared with the greedy algorithm, respectively. Therefore, the IG algorithm and DAA_TS algorithm 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.

![Figure 6](image)

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.

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 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_T S 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.

ACKNOWLEDGEMENT

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. 61370095, 61472124), and International Science & Technology Cooperation Program of China(2015DFA11240).

REFERENCES