# Energy-Aware Scheduling Algorithm with Duplication on Heterogenous Computing Systems

Jing Mei, *Kenli Li
National Supercomputing Center in Changsha
College of Information Science and Enginnering, Hunan University
Changsha, Hunan, China
JingMei1988@163.com
*Corresponding author : lkl510@263.net

***Abstract*—Efficient application scheduling is critical for achieving high performance in heterogeneous computing (HC) environments. Because of its importance, there are many researches on this problem and various algorithms have been proposed. Duplication-based algorithm is a kind of famous algorithm to solve scheduling problem, which achieve high performance on minimizing the overall completion time(makespan) of applications. However, they do not consider energy consumption. With the growing advocacy for green computing system, energy conservation has been an important issue and gained a particular interest. An existing technique to reduce energy consumption of application is dynamic voltage/frequcny scaling(DVFS), but its efficiency is affected by the overhead of time and energy caused by voltage scaling. In this paper, we propose a new energy-aware scheduling algorithm called Energy Aware Scheduling by Minimizing Duplication(EAMD), which considers the energy consumption as well as the makespan of applications. It adopts a subtle energy-aware method to determine and delete the abundant task copies in the schedules generated by duplication-based algorithms, which is easier to operate than DVFS and produces no extra time and energy consumption. This algorithm can reduce large amount of energy consumption while having the same makespan compared with duplication-based algorithms without energy awareness. Randomly generated DAGs are tested in our experiments. Experimental results show that EAMD can save up to 15.59% energy consumption for the existed duplication-based algorithms. Several factors affecting the performance are analyzed in the paper, too.**

***Index Terms*—energy aware; DAG; static scheduling; duplication-based; heterogenous;**

## I. Introduction

In past decade, more and more attention focuses on the scheduling problem of applications on heterogenous computing systems. A heterogenous computing (HC) system is defined as a distributed suite of computing machines with different capabilities interconnected by different high speed links utilized to executed parallel applications [8, 19]. Many task scheduling algorithms are proposed on HC systems, and they evaluate their performance with the only criterion: the schedule length (makespan). Duplication-based algorithm is a kind of efficient algorithm to minimize the makespan by assigning a task to serval processors, which reduces the intercommunication between tasks. However, this method leads to a great overhead of energy consumption. In recent years, with the growing advocacy for green computing system, energy conservation has become an important issue and gained a particular interest. In this paper, we present a new task scheduling algorithm on HC systems which considers two objectives above: reducing energy consumption and minimizing makespan of applications.

In our paper, we assume that all information of applications including execution time of tasks, the data size of communication between tasks, and task dependencies are known a priori, which is represented with a static scheduling. Static task scheduling takes place during compile time before tasks execution. Once the schedule is determined, the tasks can be executed following the orders and assignments. The task scheduling is to map tasks of an application to processors so that precedence requirements are satisfied and minimal makespan is achieved [14], but it is in general NP-complete [9, 25]. Therefore, heuristics can be used to obtain a sub-optimal scheduling. Task scheduling has been extensively studied and various heuristics have been proposed in the literature [1, 2, 6, 10, 15, 18, 21, 22, 24]. The general task scheduling algorithms can be classified into a variety of categories, such as list scheduling algorithms, cluster algorithms, and duplication-based algorithms and so on.

Duplication-based algorithm is a kind of algorithm with highest performance in term of makespan. The idea of duplication-based algorithms is to schedule a task graph by mapping some of its tasks redundantly, which reduces the intercommunication between tasks. Duplication-based algorithms have better performance than list scheduling algorithms in terms of makespan. However, this method makes most tasks executed on more than one processor, which leads to a great increase on energy consumption. Therefore, in this paper we try to explore the method of reducing the abundant copies of tasks so as to reduce energy consumption. This method can be applied to the output of any duplication-based algorithms.

The contributions of this paper list as follows:

- we propose a subtle energy-aware method, which is easier to operate than DVFS and produces no overhead of time and energy. In addition, the performance in terms of makespan is as same as the duplication-based algorithms;

1550-5510/12 $26.00 © 2012 IEEE
DOI 10.1109/Grid.2012.32

- Experiments are given to verify that EAMD can reduce large amount of energy consumption compared with the duplication-based algorithms while not sacrificing the performance on makespan;
- The factors affecting the performance of our algorithm are analyzed.

The remainder of this paper is organized as follows. In Section II, some related work has been described, which includes some different scheduling heuristics on heterogenous systems, power estimation and optimization techniques, and serval energy aware scheduling algorithms. In Section III, we define the problem and present the related models. In Section IV, we give a detailed description of EAMD algorithm, and a brief analysis on time complexity. An example is also given in this section to explain our algorithm better. The experimental results are presented and some analysis is given in Section V. Section VI concludes the work of our paper and an overview of future work.

## II. Related Work

The algorithms can be classified to a variety of categories, such as list scheduling algorithms, cluster algorithms, and duplication-based algorithms. The list scheduling algorithms provide a good quality of schedules and their performance is comparable with the other categories at a lower time complexity. Some examples are: dynamic critical-path(DCP) [13], heterogenous earliest finish time (HEFT) [24], critical path on a processor(CPOP) [24] and the longest dynamic critical path (LDCP) [6]. Cluster algorithms merge tasks in a graph to an unlimited number of clusters and tasks in a cluster are scheduled on the same processor. Some examples in this category are clustering for heterogeneous processors(CHP) [4], clustering and scheduling system(CASS) [17], objective-flexible clustering algorithm(OFCA) [7]. The idea of duplication-based algorithms is to schedule a task graph by mapping some tasks redundantly, which reduces the interprocess communication overhead. There are many duplication-based algorithms, for example, selective duplication(SD) [1], heterogenous limited duplication(HLD) [2], heterogenous critical parents with fast duplicator(HCPFD) [10], heterogenous earliest finish with duplication(HEFD) [23]. This kind of algorithms can reduce the makespan effectively, but it is traded with large amount of energy consumption.

### A. Energy Reduction Techniques

The general two techniques to reduce energy consumption on system level scheduling are Dynamic Power Management (DPM) and Dynamic Voltage/Frequency Scaling (DVFS). DPM turns the idle components off to reduce the power consumption. Because DPM leads to the energy and time overheads, it is working only when the idle time be long enough. DPM technique are mostly applied to laptops and PDA. For the scheduling of an application, the idle time between two tasks execution is little, so DPM is not suitable. DVFS has been proven to be a very promising technique with its demonstrated capability for energy savings [3, 16, 20, 26]. DVFS-enabled processors can scale down their voltages and clock frequencies when idle periods exist. Because energy consumption is proportional to processor voltage quadratically, DVFS technique can reduce energy dissipation while guaranteeing the performance.

Whereas, DVFS has its disadvantages, too. The first disadvantage of DVFS is the overhead of energy and time, which is similar to DPM. General speaking, when processors are scaled between two different voltages, the energy overhead and delay is related with the voltages and cannot be neglected [5]. Second, DVFS reduces the energy consumption by scaling down the voltage and clock frequency, which is at the expense of sacrificing makespan.

In this paper, we propose a new energy aware scheduling algorithm aiming at reducing the energy consumption of duplication-based algorithms without DVFS. As we said before, duplication-based algorithms can achieve a high performance by mapping some tasks redundantly but a significant increase of energy consumption. Through the analysis on the duplication-based algorithms, we find that some copies of tasks can be deleted, which do not affect the task precedence but can reduce energy consumption greatly. The idea of our algorithm is originated from this. The proposed algorithm in the paper aims at reducing energy consumption while reserving the optimal makespan of duplication-based algorithms.

## III. Models

A scheduling system model consists of a target computing environment, an application and performance criteria for scheduling. In this section, we will present our computing system model, application model and performance criteria models.

### A. Computing System Model

The computing system considered in this paper is heterogeneous. Let $P = \{p_k | 0 \leq k \leq n-1\}$ be a set of $n$ processors with different capacities. The capacity of each processor processing tasks depends on how well the architecture of the processor matches tasks' processing requirements. A task scheduled on its best-suited processor will spend shorter execution time than on a less-suited processor. The best processor for one task may be the worst processor for another task. This type of model is described by [12] and used in [10, 21, 22, 24].

### B. Application Model

A *Directed Acyclic Graph* (DAG) $G(V, E, [w_{i,k}])$ consists of a set of nodes $V = \{v_i | 0 \leq i \leq m-1\}$ representing the tasks in an application and a set of directed edges $E$ representing dependencies among tasks. Let $c_{i,j}$ be the weight of edge $e_{i,j}$, which represents the required communication time to send data from $v_i$ to $v_j$. Here $v_i$ is called the parent of $v_j$. $[w_{i,k}]$ is a $m \times n$ matrix of computation time, where $w_{i,k}$ presents the computation

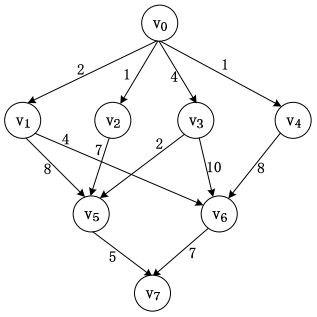



Fig. 1: A simple DAG representing precedence-constraint application graph

TABLE I: Computation cost matrix $[w_{i,k}]$

| Task node | $p_0$ | $p_1$ | $p_2$ | $p_3$ | $\overline{w_i}$ | $rank_u$ |
|---|---|---|---|---|---|---|
| $v_0$ | 1 | 1 | 2 | 1 | 1.25 | 32.00 |
| $v_1$ | 3 | 2 | 4 | 2 | 2.75 | 19.75 |
| $v_2$ | 5 | 6 | 3 | 4 | 4.5 | 20.50 |
| $v_3$ | 2 | 4 | 4 | 2 | 3.0 | 26.75 |
| $v_4$ | 4 | 8 | 7 | 8 | 6.75 | 28.5 |
| $v_5$ | 3 | 3 | 1 | 2 | 2.25 | 9.00 |
| $v_6$ | 5 | 5 | 5 | 5 | 5 | 13.75 |
| $v_7$ | 1 | 2 | 2 | 2 | 1.75 | 1.75 |

time of task $v_i$ executed on processor $p_k$ for $0 \leq i \leq m-1$ and $0 \leq k \leq n-1$. Fig. 1 gives a DAG to represent the application model, and Table II lists the computation time matrix $[w_{i,k}]$.

A task having no parents is called *entry task*, such as task $v_0$ in Fig. 1. A task having no children is called an *exit task*, such as $v_7$. In this paper, we just discuss the scheduling of DAGs with single-entry and single-exit task. For those DAGs with multi-entry and multi-exit tasks, we can transfer them by adding a zero-cost pseudo entry(exit) task with zero-cost edges, which do not affect the schedule.

### C. Performance Criteria

*1) Makespan:* Makespan is a main criterion to measure the performance of scheduling algorithms. For a task $v_i$ scheduled on processor $p_k$, $st(v_i, p_k)$ and $ft(v_i, p_k)$ present the *start time* and *finish time* of task $v_i$ on $p_k$. Because preemptive execution is not allowed, $ft(v_i, p_k) = st(v_i, p_k) + w_{i,k}$. Makespan is denoted as

$$makespan = ft(v_{exit}). \tag{1}$$

The proposed algorithm in this paper is to reduce the energy consumption of a valid schedule generated by duplication-based algorithms without degrading the makespan, so we introduce the energy consumption model used in this paper. Energy consumption of computing system is related to power consumption and the execution time of processors. The power consumption is related to the design technology of processors and it is different according to the state of processors. When tasks of an application are executing on a computing system, processors switch between two states: busy state and idle state. Busy state means there are tasks executed on processors, and idle state means no task is executed on processors. It is apparent that the power consumption of processors in busy state is different from that in idle state. In this paper, we adopt the power consumption values of Intel XScale PXA270 processor [11] shown as Table II.

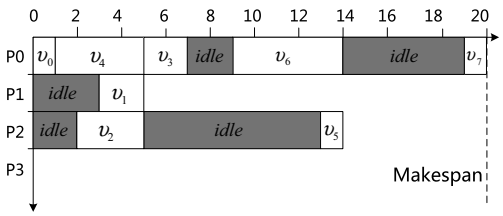



Fig. 2: A schedule of the DAG in Fig. 1

TABLE II: Power consumption of PXA270 [11]

| Frequency | Idle Power | Busy Power |
|---|---|---|
| 624MHZ | 260mW | 925mW |

PXA270 processor is a kind of processor with the mechanism of Dynamic Voltage/Frequency Scaling (DVFS). It has six frequency levels. The frequency of 624MHZ listed in Table II is the maximum frequency of PXA270. Because in our algorithm, the frequency is fixed and the mechanism of DVFS is useless, we assume that all processors are running at the frequency of 624MHZ. As shown in Table II, we assume the power consumption of idle processor is 260mW and that of busy processor is 925mW. The power consumption in busy state is much greater than that in idle state. The energy consumption of computing system is calculated by:

$$E_{total} = P_{busy} \times t_{busy} + P_{idle} \times t_{idle} \tag{2}$$

From Eq. (2) we can see that the for a given computing system energy consumption is mainly determined by execution time of processors as the power consumption is fixed.

Fig. 2 gives a schedule of the application given in Fig. 1. The makespan is 20ms. Assume that the processors without tasks are turned off, and consume no energy. The power consumption of processors in busy state and idle state are 925mW and 260mW, respectively. Therefore, the total energy consumption of the schedule is $20 \times 925 + 20 \times 260 = 23.7mJ$.

## IV. The Proposed Algorithm

In this section, we present the details of Energy Aware Scheduling by Minimizing Duplication algorithm, EAMD for short. The objection of this algorithm is to reduce the energy consumption of an input schedule generated by duplication-based algorithms without degrading the makespan.

In a valid input schedule $S$, a task may be executed on more than one processor except exit tasks. We note that if a copy of task can provide data to all of its children, then the other copies in the schedule can be deleted.

According to the execution times of tasks in a schedule, we divide tasks into three categories:

- Single-Copy Task: Task that scheduled only one time in the schedule,
- Dual-Copy Task: Task that scheduled two times in the schedule,
- Multi-Copy Task: Task that scheduled more than two times in the schedule.

According to our idea, only the copies of dual-copy tasks and multi-copy tasks have chance to be deleted to reduce energy consumption.

**Definition 4.1:** In a schedule, the copy determined earliest of a task is called its **Original Copy**, and the others are called **Duplicated Copy**.

We denote the $j^{th}$ copy of task $v_i$ as $v_i^j$. Assume task $v_i$ has a duplicated copy on processor $p_k$, and its child $v_j$ executes on the same processor $p_k$ and receives data from $v_i$ on $p_k$, we say that $v_i$ is the *Duplicated Parent* of $v_j$, denoted as $DP(v_j, p_k) = v_i$. For task $v_j$ executed on $p_k$, it may have one or no duplicated parent.

If a copy of task $v_j$ can be deleted, it must be a dual-copy task or a multi-copy task. After one copy of task $v_j$ is deleted, its other copies must guarantee the precedence constraint with its parents and children. According to the different situations of dual-copy task and multi-copy task, we give the analysis as following.

When $v_j$ is a dual-copy task, its original copy can be deleted if it satisfies:

$$ft(v_j, p_k) + \overline{c_{j,i}} \leq st(v_i, p_l), \quad \forall v_i \in child(v_j),\ p_l \in P. \tag{3}$$

where $ft(v_j, p_k)$ presents the actual finish time of task $v_j$ duplicated on processor $p_k$. If $p_k = p_l, \overline{c_{j,i}} = 0$. When the duplicated copy of task $v_j$ can provide the data of all its children and does not violate the precedence constraint, the original copy can be deleted.

When $v_j$ is a multi-copy task, the analysis is much more complex than the situation of dual-copy task. Assume that $v_j$ has a copy on processor $p_k$, it can offer data of all its children if Eq. (3) is satisfied, the other copies of $v_j$ can be deleted. However, it is more possible that only one copy cant supply the needed data of all children. In this situation, traverse all combinations of $v_j$'s copies and determine the combination that satisfies all precedence constraint with its children. If there exists the combination, delete all other copies of $v_j$ except the combination.

In order to find the available combination, traverse all combinations of $v_j$'s copies and determine the combination that satisfies all precedence constraint with its children. If there exists the combination, delete all other copies of $v_j$ except the combination; if not, turn to deal with next task.

When deleting the redundant copies of tasks, we traverse tasks in the nondecreasing order of *upward ranks* of tasks make sure that all children have been processed when dealing with parents. The bottom level $rank_u$ is computed by traversing the DAG upward starting exit tasks, which is recursively defined by

$$rank_u(v_i) = \overline{w_i} + \max_{v_j \in child(v_i)} (c_{i,j} + bl(v_j)) \tag{4}$$

where $child(v_i)$ is the set of immediate children of task $v_i$ in DAG. For the exit task $v_{exit}$, the bottom level is

$$rank_u(v_{exit}) = \overline{w_{exit}} \tag{5}$$

**Algorithm 1** EAMD algorithm

**Require:** A schedule $S$ generated by arbitrary duplication-based algorithm;
**Ensure:** The schedule after deleting the redundant copies.

```
for each task v_i in non-increasing order of rank_u do
  for each processor p_k that has a copy of task v_i do
    if v_i on p_k satisfies Eq. (3) then
      delete the other copies of v_i and the duplicated task they rely on
    end if
  end for
  list all pairs of v_i's copies in nondecreasing order of total execution time
  for each pair do
    if the copies of the pair can provide all data needed by its children and do not disobey the constraints then
      delete all other copies except the pair;
      break;
    end if
  end for
end for
```

### A. The Time-Complexity Analysis

Because EAMD algorithm is based on the existed duplication-based algorithms, which have their own time complexity, we only analyze the time complexity of the energy aware phase of deleting redundant copies of tasks. The time complexity of this phase is usually expressed in terms of number of tasks $m$, number of edges $e$, and number of processors $n$. The time complexity of EAMD is analyzed as follows.

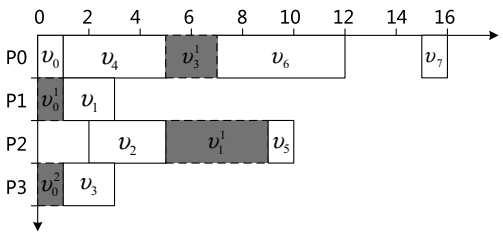
The priority queue needs to be determined before task scheduling, which can be done in time $O(m)$. For each task $v_i$, we need to judge if it has redundant copies to be deleted. Assume that task $v_i$ has $e_i$ children, and then it can be duplicated on as most as $e_i$ processors. Therefore, the time complexity of judging $v_i$ is $O((C_{m_i}^1 + C_{m_i}^2) \times m_i) = O(m_i^3)$. The time complexity of judging all tasks is $O(\sum_{i=1}^{m} e_i^3)$. Because $\sum_{i=1}^{m} e_i = e$, the total time complexity is less than $O(e^3)$.

### B. Example Presentation

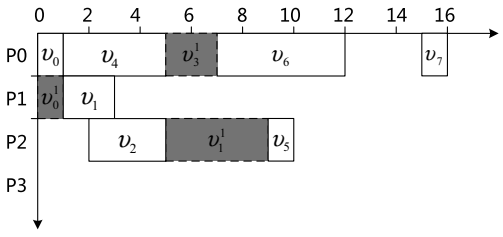
Fig. 3 gives schedules of application in Fig. 1 using HLD algorithm and our EAMD algorithm. Compared with HLD, EAMD deletes the redundant copies of tasks so that a great amount of energy savings is reduced. The process is described as follows.

The schedule is generated by HLD algorithm firstly, shown as Fig. 3(a). A priority queue $L$ is constructed in the non-decreasing order of $bl$. For the given example, $L = \{v_7, v_5, v_6, v_1, v_2, v_3, v_4, v_0\}$. Second, pop tasks from $L$ in sequence, and determine if delete operation can be done for each task. The first popped task is $v_7$, and it does not rely on duplication of any task. The second task is $v_5$. From Fig. 3(a) we can see, it relies on the duplicated task $v_1^1$ of $v_1$, written as $RD(v_5) = v_1^1$. Task $v_1$ has two children $v_5$ and $v_6$. From Fig. 3(a), task $v_1^1$ is complete at $t = 9$ and $v_6$ starts at $t = 7$, the precedence constraint is violated if $v_1$ is deleted, so we do not delete $v_1$ and its relied task $v_0^1$. The search is continue. For task $v_6$, its relied duplication $v_3$ has two children, $v_5$ and $v_6$. Obviously, the precedence constraint between $v_3$ and $v_6$ is satisfied when $v_3^1$ is deleted. The $ft$ of $v_3^1$ on processor $p_0$ is 7, and the communication time between $v_3$ and $v_5$ is 2. If $v_5$ receives the data from $v_3^1$, the $dat$ of $v_5$ is $7 + 2 = 9$, which is equal to the $st$ of $v_5$ on $p_2$. Therefore, task $v_3$ can be deleted. The schedule result is shown as Fig. 3(b).

Comparing the two schedules, the makespan is same, but their energy consumption is different. In Fig. 3(a), the total busy time of four processors is 27ms, and the total idle time is 5ms. In Fig. 3(b), the total busy time and idle time are 24ms and 5ms, respectively. From Table II, the power consumption of busy period and idle period are 925mW and 260mW. Therefore, the energy consumption of the schedule in Fig. 3(a) is $26.275mJ$ and $23.5mJ$ in Fig. 3(b). EAMD algorithm can reduce 10.56% energy consumption than HLD algorithm in this example.




(a) Schedule generated by HLD algorithm




(b) Schedule generated by EAMD algorithm

Fig. 3: Comparison between HLD and EAMD algorithm

## V. Experimental Results and Analysis

In this section, we present the results obtained from algorithm EAMD and two previously proposed algorithms HLD and HCPFD. Both of HLD and HCPFD algorithms are well-known duplication-based algorithms and they are energy "unconscious", but they perform well for the task scheduling problem. In this paper, we combine EAMD algorithm with HLD and HCPFD, and name them HLD+EAMD and HCPFD+EAMD for short. This comparison between HLD and HLD+EAMD, HCPFD and HCPFD+EAMD clearly demonstrates the energy saving capability of our algorithms.

The performance of EAMD algorithm is evaluated with randomly generated application.Typically, the makespan of a task graph generated by a scheduling algorithm is used as the main performance measure; however, as our algorithm is proposed on the basis of duplication based algorithms, and has the same performance on makespan with those algorithms. Therefore, we do not compare makespan in our experiments. The main performance metric chosen for comparison is energy consumption. Since energy consumption of applications varies with the number of tasks and has a large variation range, it's necessary to normalize the energy consumption. Here we define a parameter Energy-Ratio $ER$ as the energy metric:

$$ER = \frac{E}{E_{HCPFD}}. \tag{6}$$

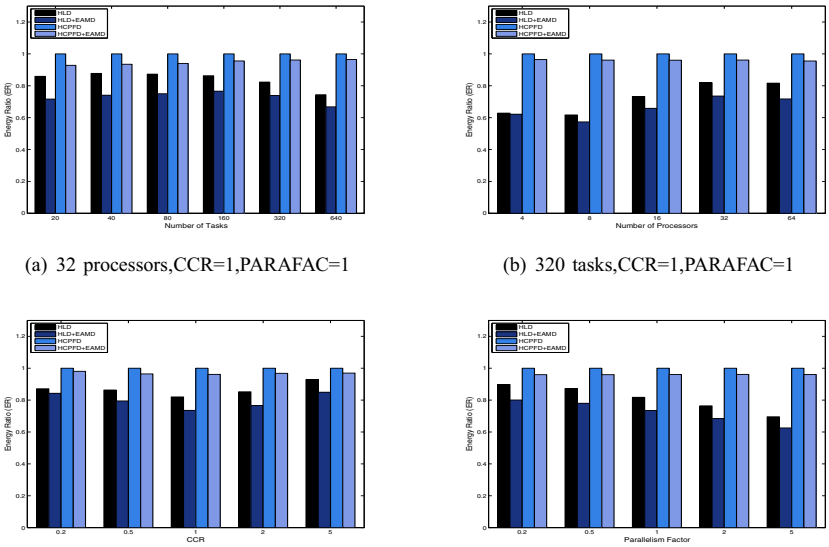
where $E$ is the energy consumption of compared algorithms, and the $E_{HCPFD}$ is the energy consumption of algorithm HCPFD.

### A. Randomly Generated Application

*1) Application Graphs Random Generation:* The random generated graphs are commonly used to compare scheduling algorithms, and the method of generation is described in [1, 6, 15, 23, 24]. Three fundamental characteristics in this paper are considered:

- DAG size, $n$: The size of tasks in the application DAG.
- Communication to computation cost ratio, CCR: The average communication cost divided by the average computation cost of the application DAG.
- Parallelism factor, $\lambda$: The number of levels of the application DAG is generated randomly, using a uniform distribution with a mean value of $\frac{\sqrt{n}}{\lambda}$, and then rounding it up to the nearest integer. The width is generated using a uniform distribution with a mean value of $\lambda\sqrt{n}$, and then rounding it up to the nearest integer. A low $\lambda$ leads to a DAG with a low parallelism degree.

In our experiments, graphs are generated for all combinations of above parameters. The number of nodes in




(a) 32 processors,CCR=1,PARAFAC=1

(b) 320 tasks,CCR=1,PARAFAC=1

(c) 320 tasks,32 processors,PARAFAC=1

(d) 320 tasks,32 processors,CCR=1

Fig. 4: Average Energy Savings of Random DAG

TABLE III: Energy conservation with respect to various characteristics

| Number of Tasks | 20 | 40 | 80 | 160 | 320 |
|---|---|---|---|---|---|
| HLD+EAMD vs. HLD | 15.59% | 14.04% | 11.23% | 10.24% | 10.20% |
| HCPFD+EAMD vs. HCPFD | 6.49% | 5.98% | 4.43% | 3.84% | 3.56% |
| Number of Processors | 4 | 8 | 16 | 32 | 64 |
| HLD+EAMD vs. HLD | 9.55% | 9.78% | 10.03% | 10.28% | 12.15% |
| HCPFD+EAMD vs. HCPFD | 3.57% | 3.91% | 3.96% | 3.88% | 4.47% |
| CCR | 0.2 | 0.5 | 1 | 2 | 5 |
| HLD+EAMD vs. HLD | 3.27% | 7.92% | 10.28% | 10.15% | 8.62% |
| HCPFD+EAMD vs. HCPFD | 1.94% | 3.58% | 3.88% | 3.18% | 3.04% |
| PARAFAC | 0.2 | 0.5 | 1 | 2 | 5 |
| HLD+EAMD vs. HLD | 10.85% | 10.61% | 10.15% | 10.47% | 10.07% |
| HCPFD+EAMD vs. HCPFD | 4.04% | 4.05% | 3.92% | 3.88% | 3.98% |

TABLE IV: Energy conservation of 320 tasks and 8 processors with respect to parallelism degree

| PARAFAC | 0.2 | 0.5 | 1 | 2 | 5 |
|---|---|---|---|---|---|
| HLD+EAMD vs. HLD | 7.88% | 9.00% | 7.02% | 2.89% | 0.82% |
| HCPFD+EAMD vs. HCPFD | 2.98% | 3.66% | 3.94% | 4.16% | 4.30% |

DAG ranges from 20 to 640. To generate a DAG of a given number, the number of levels is determined by the Parallelism factor $\lambda$(0.2, 0.5, 1.0, 2.0, 5.0) firstly, and then the number of tasks at each levels is determined. Edges only generate between the nodes in the adjacent levels, obeying 0-1 distribution. To obtain the desired CCR for a graph, computation costs are taken randomly from a uniform distribution. The communication costs are also randomly selected from a uniform distribution, whose mean depends on CCR(0.1, 0.5, 1.0, 2.0, 5.0) and the average computation cost. 500 random graphs are generated for each set of above parameters in order to avoid scattering effects. The experimental results are the average of the data obtained for these graphs.

*2) Random Application Performance Analysis:* The energy consumption of algorithms is compared with respect to various graphs characteristics. The overall experimental results are presented in Fig. 4 and Table. III. Table. III clearly signifies the superior performance of our algorithm over HLD and HCPFD algorithms.

The first set of experiments compare the energy consumption of algorithms with respect to various graph sizes (see Fig.4(a)). The performance on energy savings of two improved algorithms, HLD+EAMD and HCPFD+EAMD, outperform HLD and HCPFD, respectively. According to the Table III, the average energy consumption of HLD+EAMD is less than HLD algorithm by 15.59%,14.04%,11.23%,10.24% and 10.20%, for number of tasks of 20, 40, 80, 160 and 320, respectively.

For HCPFD+EAMD, the average corresponding energy savings ratios are 6.49%,5.98%,4.43%,3.84% and 3.56%. From Table III we can see that, EAMD performs better on HLD than on HCPFD, and the energy savings ratios are decreasing with the increasing number of tasks. The reason is as follows. When number of tasks is small, the load of each processor is light and there are enough idle time to duplicate task. So more duplicated task can be deleted when using EAMD algorithm, which leads to more energy savings.

The second set of experiments compare the energy consumption of algorithms with respect to different number of processors. The average energy savings of HLD+EAMD and HCPFD+EAMD compared to HLD and HCPFD are (9.55%,3.57%), (9.78%,3.91%), (10.03%, 3.96%), (10.28%,3.88%) and (12.15%,4.47%), for number of processors of 4, 8, 16, 32 and 64, respectively. Therefore, our algorithm outperforms both of HLD and HCPFD algorithms and along with the increasing number of processors, the performance gets better.

In the third group of experiments, when combined with EAMD algorithm, the HLD and HCPFD algorithms can reduce much energy consumption than the original algorithms for all tested CCR values. The table data shows that the peak performance is achieved at CCR=1, 10.28% for HLD and 3.88% for HCPFD. When the CCR value is less than 1, the average communication cost is less than the average computation cost, so duplication is unnecessary if it consumes more time than communication. Therefore, the duplication be deleted is less, which leads to less energy savings. When the CCR value is greater than 1, the average communication cost is greater than the average computation cost. In the duplication deleted phase, when determining whether the origin task can be deleted, the greater communication cost leads to a greater probability that it cannot be deleted. Therefore, the energy savings get smaller. In sum, only when the communication and computation cost is equivalent, the best performance is achieved.

The next experiments is with respect to the graph structure. From Table III we can notice that the parallelism degree has little pact on energy savings. After analyzing, we think that the parameter setting shields the varying tendency of energy savings. That's because 32 processors are enough to execute 320 tasks in parallel no matter how large the parallelism degree is. In order to observe better the varying tendency of performance with the increasing parallelism degree, we set 320 tasks and 8 processors and do another group of experiments. The results are shown in Table IV. When $\lambda$ is equal to 0.2, the generated graphs have greater depths with a low degree of parallelism, it is shown that the energy consumption of HLD+EAMD is less than HLD by 7.88%, and 2.98% for HCPFD+EAMD compared with HCPFD. With the increasing of the parallelism factor $\lambda$, the performance of EAMD degrades on HLD but upgrades on HCPFD algorithm. That's because the priority queuing of HLD is based on upward rank which is breadth first while that of HCPFD is based on the critical path which is depth first. Therefore, the varying tendency of performance is different for two algorithms.

## VI. Conclusions

In this paper, we propose a new scheduling algorithm called EAMD scheduling algorithm. The algorithm aims at reducing the energy consumption of duplicated-based algorithms which is caused by the redundant mapping of some tasks. Through the experimental results, EAMD can reduce energy consumption by up to 15.59% compared with HLD and HCPFD algorithms. These results are based on randomly generated graphs. Similarly, EAMD can be combined with all other duplication-based algorithms and all can obtain good performance on energy savings.

The amount of energy savings is affected by many factors, such as the number of processors, CCR and parallelism degree. For the fixed number of tasks, the amount of energy savings increases with the increasing number of processors due to the increase of idle time and duplications. When the average commination cost is almost equal to the average computation cost, it also means when CCR is around 1, the duplicated-based algorithms can map tasks in the most processors, which leads to the greatest amount of energy savings compared to other CCR values. For the parallelism of DAGs, the ratio of energy savings decreases with the increase of width. Overall, our algorithm EAMD can achieve a good performance compared to the duplication-based algorithms. Future work can involve combining EAMD with DVFS technique to reduce more energy consumption.

## Acknowledgments

This research was partially funded by the Key Program of National Natural Science Foundation of China (61133005) and the National Science Foundation of China (Grant Nos. 61070057, 90715029).

## References

[1] S. Bansal, P. Kumar, and K. Singh. An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(6):533 – 544, june 2003.

[2] S. Bansal, P. Kumar, and K. Singh. Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs. *Journal of Parallel and Distributed Computing*, 65(4):479 – 491, 2005.

[3] S. Baskiyar and R. Abdel-Kader. Energy aware dag scheduling on heterogeneous systems. *Cluster Computing*, 13:373–383, 2010.

[4] C. Boeres, J. Filho, and V. Rebello. A cluster-based strategy for scheduling task on heterogeneous processors. In *16th Symposium on Computer Architecture*

*and High Performance Computing, 2004. (SBAC-PAD 2004)*, pages 214 – 221, oct. 2004.

[5] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design, 2000. (ISLPED '00)*, pages 9 – 14, 2000.

[6] M. I. Daoud and N. Kharma. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 68(4):399 – 409, 2008.

[7] F. Fangfa, B. Yuxin, H. Xinaan, W. Jinxiang, Y. Minyan, and Z. Jia. An objective-flexible clustering algorithm for task mapping and scheduling on cluster-based noc. In *2010 10th Russian-Chinese Symposium on Laser Physics and Laser Technologies (RCSLPLT) and 2010 Academic Symposium on Optoelectronics Technology (ASOT)*, pages 369 –373, 28 2010-aug. 1 2010.

[8] R. Freund and H. Siegel. Heterogeneous processing. *IEEE Computer*, 26(6):13 – 17, 1993.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] T. Hagras and J. J. brevecek. A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. *Parallel Computing*, 31(7):653 – 670, 2005.

[11] Intel. Pxa270 processor, electrical, mechanical, and thermal specification. 2004.

[12] A. Khokhar, V. Prasanna, M. Shaaban, and C.-L. Wang. Heterogeneous computing: challenges and opportunities. *Computer*, 26(6):18 –27, jun 1993.

[13] Y.-K. Kwok and I. Ahmad. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506 –521, may 1996.

[14] Y.-K. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. In *Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998, Parallel Processing Symposium 1998. (IPPS/SPDP 1998)*, pages 531 –537, mar-3 apr 1998.

[15] K.-C. Lai and C.-T. Yang. A dominant predecessor duplication scheduling algorithm for heterogeneous systems. *The Journal of Supercomputing*, 44:126–145, 2008.

[16] Y. C. Lee and A. Y. Zomaya. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems*, 22:1374–1381, 2011.

[17] J. Liou and M. Palis. An efficient task clustering heuristic for scheduling dags on multiprocessors. In *Proceedings of Parallel and Distributed Processing Symposium*, 1996.

[18] F. Lotfifar and H. Shahhoseini. A low-complexity task scheduling algorithm for heterogeneous computing systems. In *Third Asia International Conference on Modelling Simulation, 2009. (AMS '09)*, pages 596 –601, may 2009.

[19] M. Maheswaran, T. D. Braun, and H. J. Siegel. Heterogeneous distributed computing. *Encyclop edia of Electrical and Electronics Engineering*, 8:679–690, 1999.

[20] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theor. Comp. Sys.*, 43:67–80, March 2008.

[21] A. Radulescu and A. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *Proceedings of 9th Heterogeneous Computing Workshop, 2000. (HCW 2000)*, pages 229 –238, 2000.

[22] S. Ranaweera and D. Agrawal. A scalable task duplication based scheduling algorithm for heterogeneous systems. In *Proceedings of 2000 International Conference on Parallel Processing*, pages 383 –390, 2000.

[23] X. Tang, K. Li, G. Liao, and R. Li. List scheduling with duplication for heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 70(4):323 – 329, 2010.

[24] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260 –274, mar 2002.

[25] J. D. Ullman. Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10:384–393, June 1975.

[26] Y. Zhang, X. Hu, and D. Chen. Task scheduling and voltage selection for energy minimization. In *Proceedings of 39th Design Automation Conference*, pages 183 – 188, 2002.