electronics

MDPI

Article

Task-Level Aware Scheduling of Energy-Constrained Applications on Heterogeneous Multi-Core System

Kai Huang [1], Ming Jing [1], Xiaowen Jiang [1,*], Siheng Chen [1], Xiaobo Li [2,3], Wei Tao [2], Dongliang Xiong [1] and Zhili Liu [4]

1 Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China; huangk@vlsi.zju.edu.cn (K.H.); 21831043@zju.edu.cn (M.J.); 21860277@zju.edu.cn (S.C.); xiongdl@zju.edu.cn (D.X.)
2 Digital Grid Research Institute, Artificial Intelligence and Chip Application Research Department, CSG, Guangzhou 510623, China; lixb1@csg.cn (X.L.); taowei@csg.cn (W.T.)
3 Electric Power Research Institute, China Southern Power Grid (CSG), Guangzhou 510623, China
4 Hangzhou Sec-Chip Technology Co., Ltd., Hangzhou 310027, China; liuzhili@sec-chip.com
* Correspondence: xiaowen_jiang@zju.edu.cn

Received: 10 November 2020; Accepted: 3 December 2020; Published: 5 December 2020

Abstract: Minimizing the schedule length of parallel applications, which run on a heterogeneous multi-core system and are subject to energy consumption constraints, has recently attracted much attention. The key point of this problem is the strategy to pre-allocate the energy consumption of unscheduled tasks. Previous articles used the minimum value, average value or a power consumption weight value as the pre-allocation energy consumption of tasks. However, they all ignored the different levels of tasks. The tasks in different task levels have different impact on the overall schedule length when they are allocated the same energy consumption. Considering the task levels, we designed a novel task energy consumption pre-allocation strategy that is conducive to minimizing the scheduling time and developed a novel task schedule algorithm based on it. After getting the preliminary scheduling results, we also proposed a task execution frequency re-adjustment mechanism that can re-adjust the execution frequency of tasks, to further reduce the overall schedule length. We carried out a considerable number of experiments with practical parallel application models. The results of the experiments show that our method can reach better performance compared with the existing algorithms.

Keywords: heterogeneous multi-core system; energy consumption; task schedule length; task level; frequency re-adjustment

1. Introduction

Computer systems nowadays must perform much better than ever before, ensuring the simultaneous running of many applications. By using heterogeneous multi-core processors and increasing the number of processor cores, it is possible to improve the performance while keeping energy consumption at the bay [1–5]. From small, embedded devices to large data centers, heterogeneous multi-core systems have been widely used. It is expected that in the near future, the number of heterogeneous processors and cores in these systems will increase dramatically [6–8]. On the other hand, although the performance of such systems has been greatly improved, the power consumption is also increasing. Huge energy consumption has caused various problems, such as economy, environment, technology and so on [9–13]. Therefore, energy consumption is one of the main design constraints for such heterogeneous multi-core systems. A well-known typical mechanism for reducing power consumption of computing systems is dynamic voltage and frequency scaling (DVFS), which is realized to achieve the balance between energy consumption and performance by reducing the power supply

voltage and frequency of the processor at the same time, when the task is running [1,14–19]. Therefore, energy consumption constrained task scheduling on processors with adjustable voltage and frequency has attracted extensive research [20–23].

In the existing research, works related to the energy consumption of heterogeneous multi-core processors, some of the objectives are to minimize the energy consumption under certain performance requirements. Others are to minimize or maximize other indicators, such as performance and reliability, under certain energy consumption constraints [24–31]. Our studies fall into the second category. To be more precise, we study the following problem: minimizing the schedule length of energy consumption constrained parallel applications on heterogeneous multi-core systems. There have been some studies in recent years that are consistent with our goals. In order to determine the energy consumption constraint of each task, the common method in previous studies is to pre-allocate the energy consumption of the unscheduled tasks [20–23]. Therefore, these studies focus on how to allocate the overall energy consumption constraints to each task reasonably. Xiao et al. [20] came up with an algorithm called MSLECC (minimizing schedule length of energy consumption constrained). It takes the minimum energy consumption of a task as its pre-allocation energy consumption. The drawback of this method is that it is unfair to low priority tasks. High priority tasks take up too much energy consumption, so that low priority tasks can only be allocated to low-power processor cores and close to the lowest execution frequency due to the energy consumption constraint, which will lead to an increase in the overall schedule length. After that, some studies improved the energy pre-allocation method, so that the energy pre-allocation is fair to each task. One study pre-allocated the energy consumption according to the task execution time proportion [21], another study used the average energy consumption as the pre-allocation energy consumption of each task [22], and another study used a defined task energy consumption weight value as the pre-allocation energy consumption of tasks, whose algorithm is called ISAECC [23]. However, it is not the best practice to treat each task fairly and the task level is an important issue when pre-allocating energy consumption to the tasks. We pre-allocate more energy consumption to the tasks in the task levels with fewer tasks, because when their execution time becomes shorter, the overall scheduling length is more likely to decrease. In addition, previous articles ignored the negative impact of local optimal scheduling algorithm. Therefore, we develop a task execution frequency re-adjustment mechanism that also uses DVFS to further reduce the schedule length. To summarize, the main contributions of this article are as follows.

- We design a novel energy pre-allocation strategy considering task level and prove its feasibility.
- We develop a novel scheduling algorithm to minimize the schedule length under energy consumption constraints based on the new energy pre-allocation strategy.
- We introduce a frequency re-adjustment mechanism after task scheduling to reduce the negative impact of local optimization.
- We evaluate our algorithm based on real parallel applications. The experimental results consistently prove the superiority and competitiveness of our algorithm.

The structure of this article is as follows. Section 2 reviews some existing studies that are relevant to us. Section 3 gives some preliminaries related to the problem of minimizing the schedule length for energy consumption constrained parallel applications. In Section 4, we present our approach for this problem. In Section 5, we discuss and analyze the experimental results. Finally, we conclude the paper in Section 6.

2. Related Work

Energy saving design technology based on DVFS was first proposed by [1]. Nowadays, DVFS has been widely used in multi-core task scheduling problems related to energy consumption. Reference [2] studied the problem of minimizing the schedule length of independent sequential applications with energy consumption constraints. In Reference [29], the task scheduling problem with energy

consumption constraints is considered as a combinatorial optimization problem. In Reference [31], the authors consider three constraints (energy consumption, deadline and reward). These studies are mainly focused on homogeneous systems, so they are different from our study.

In addition to the above research, many researchers have studied the task scheduling problem on heterogeneous multi-core systems. For example, reference [32] proposed an energy-saving workflow task scheduling algorithm based on DVFS. Huang et al. [33] proposed an enhanced energy-saving scheduling algorithm to minimize energy consumption under the condition of satisfying a certain performance level. Rusu et al. [31] added constraints and proposed an efficient algorithm for minimizing energy consumption under multiple constraints. The goal of these studies is generally contrary to ours. We study the minimization of the schedule length under energy consumption constraints, while those studies focus on minimizing energy consumption under other constraints.

There are also a lot of excellent studies that are closely related to our study. For example, a representative paper proposed the classical Heterogeneous Earliest Finish Time (HEFT) algorithm, which was developed to minimize the schedule length in heterogeneous multicore systems. The application model and energy consumption model they use are consistent with ours, but they do not consider the constraints of energy consumption. At present, the studies close to us should be the four articles mentioned in Section 1 [20–23]. They pre-allocate energy consumption to each task according to the minimum energy consumption of the task [20], the execution time ratio of the task [21], the overall average energy consumption [22], or the defined task energy consumption weight [23]. What is not considered in the above studies is that different task hierarchies have different impacts on the overall schedule length. Moreover, they ignored the negative impact of the local optimal characteristics of scheduling algorithm on the schedule length. Based on this, we propose a novel energy pre-allocation strategy considering task level, and, in order to reduce the negative impact of local optimal characteristics, we develop a scheduling task execution frequency re-adjustment mechanism. Finally, our method achieves better performance than previous studies.

3. Models and Preliminaries

In this section, we first introduce the application model (Section 3.1), then the energy consumption model (Section 3.2), and next we describe in detail the issues that need to be addressed (Section 3.3). Finally, we briefly introduce the current situation and reveal its limitations (Section 3.4). Table 1 shows the main notations we use.

Table 1. The main notations we use.

Notation	Description
$w_{i,k}$	Execution time of task n_i on the processor core u_k with the maximum frequency
$c_{i,j}$	Communication time from n_i to n_j
$pred(n_i)$	The set of direct predecessor tasks of task n_i
$succ(n_i)$	The set of direct successor tasks of task n_i
n_{entry}	Entry task of an application
n_{exit}	Exit task of an application
$E(n_i, u_k, f_{k,h})$	The energy consumption of the task n_i on the processor core u_k with the frequency $f_{k,h}$
$EST(n_i, u_k)$	The earliest start time of task n_i running on processor core u_k
$EFT(n_i, u_k, f_{k,h})$	The earliest finish time of task n_i running on processor core u_k with frequency $f_{k,h}$
$AST(n_i)$	The actual start time of task n_i
$AET(n_i)$	The actual execution time of task n_i
$AFT(n_i)$	The actual finish time of task n_i
$LFT(n_i)$	The latest finish time of task n_i
$L(n_i)$	The level of task n_i
$u_{pr(i)}$	The processor core allocated to task n_i
$f_{pr(i),hz(i)}$	The execution frequency allocated to task n_i on processor core $u_{pr(i)}$
$E_{given}(n_i)$	The calculated energy consumption constraint of task n_i
$E_{pre}(n_i)$	The pre-allocated energy consumption of task n_i
$E_{given}(G)$	The given energy consumption constraint of application G
$E(G)$	The energy consumption of application G
$SL(G)$	The schedule length of application G

3.1. Application Model

As in previous studies [20–24,34], we also use directed acyclic graph (DAG) to represent parallel application models. As for the processor cores, we define $U = \left\{u_1, u_2, \ldots, u_k, \ldots, u_{|U|}\right\}$ to represent a collection of processor cores, where $|U|$ is defined as the number of processor cores. Note that for any set X, we use $|X|$ to denote its size. We define the DAG application model as $G = \{N, W, C\}$. $N = \left\{n_1, n_2, \ldots, n_i, \ldots, n_{|N|}\right\}$ represents the set of nodes in the graph, that is, the set of tasks in the application. Due to the heterogeneous nature of the processor, the execution time of $n_i \in N$ on different processor cores is different. W refers to a matrix with size $|N| \times |U|$, where $w_{i,k}$ denotes the execution time for n_i to run on u_k with the maximum frequency. C denotes the weight of edges between connected nodes in DAG, that is, the communication time between tasks. $c_{i,j} \in C$ represents the communication time from n_i to n_j. If $c_{i,j} = 0$, it means there is no communication from n_i to n_j. We define $pred(n_i)$ and $succ(n_i)$ as the set of direct predecessor tasks and the set of direct successor tasks of task n_i. For example, $pred(n_2) = \{n_1\}$ and $succ(n_2) = \{n_8, n_9\}$. We define n_{entry} and n_{exit} as the entry task and the exit task of an application. In Figure 1, n_{entry} and n_{exit} are n_1 and n_{10}.

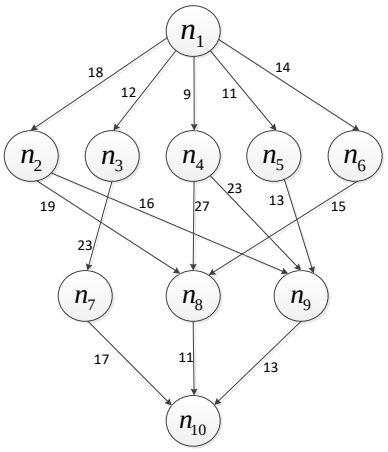

Figure 1. An example of a directed acyclic graph (DAG)-based parallel application with ten tasks.

Figure 1 shows an example of a parallel application based on DAG with ten tasks. Each node in Figure 1 represents a task, and the values on the edges between connecting nodes represent the communication time between the two nodes if they are not assigned to the same processor core. For example, the value 18 on the edge between n_1 and n_2 indicates that the communication time between n_1 and n_2 is 18.

Assuming that there are three heterogeneous processor cores $\{u_1, u_2, u_3\}$ in the system, Table 2 shows the execution time of the tasks in Figure 1 running on each processor core with the maximum frequency. For example, the first number 14 in Table 2 indicates that the execution time of n_1 running on u_1 with the maximum frequency is 14.

Table 2. Execution time of tasks on different processors with the maximum frequency of the application in Figure 1.

Core \ Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
u_1	14	13	11	13	12	13	7	5	18	21
u_2	16	19	13	8	13	16	15	11	12	7
u_3	9	18	19	17	10	9	11	14	20	16

3.2. Energy Model

In DVFS technology, the relationship between supply voltage and operating frequency is almost linear. Therefore, DVFS will also adjust the supply voltage when adjusting the clock frequency. Similar to [20–23], we use frequency regulation to indicate simultaneous regulation of supply voltage and frequency. In this article, we use the same energy model as the references [20–23]. Therefore, the calculation formula of system power consumption with respect to frequency is as follows:

$$P(f) = P_s + h(P_{ind} + P_d) = P_s + h\left(P_{ind} + C_{ef} f^m\right). \tag{1}$$

In the above equation, P_s denotes static power and can only be removed when the system is completely powered down. P_{ind} is a constant that represents the frequency-independent dynamic power, that is, it corresponds to power independent of CPU processing speed. P_d denotes frequency-dependent power, including the power primarily consumed by the CPU and any power that depends on the system processing frequency f. h denotes the system state, specifically, $h = 1$ means the system is active and the application is executing; $h = 0$ means the system is in the sleep mode or powered down. C_{ef} denotes the effective capacitance and m denotes the dynamic power exponent and is no smaller than 2. C_{ef} and m are constants related to the processor system.

Our study is in the active state of the system ($h = 1$), so dynamic power consumption is the main part of the whole energy consumption. Considering the unmanageability of static power consumption, this article, like references [20–23], does not consider static power consumption. Therefore, the calculation formula of system power consumption in this article becomes the following equation:

$$P(f) = P_{ind} + C_{ef} f^m. \tag{2}$$

Due to the heterogeneity of processors, each processor should have its own parameters. Assuming that the frequency range of the processor u_k is from the lowest frequency f_{min} to maximum frequency f_{max} temporarily, we define the following sets of parameters:

- The set of P_{ind}: $\left\{P_{1,ind}, P_{2,ind}, \ldots, P_{|U|,ind}\right\}$;
- The set of P_d: $\left\{P_{1,d}, P_{2,d}, \ldots, P_{|U|,d}\right\}$;
- The set of C_{ef}: $\left\{C_{1,ef}, C_{2,ef}, \ldots, C_{|U|,ef}\right\}$;
- The set of m: $\left\{m_1, m_2, \ldots, m_{|U|}\right\}$;

The set of execution frequency:

$$\left\{ \begin{array}{c} \left\{f_{1,min}, f_{1,a}, \ldots, f_{1,max}\right\}, \\ \left\{f_{2,min}, f_{2,a}, \ldots, f_{2,max}\right\}, \\ \ldots, \\ \left\{f_{|U|,min}, f_{|U|,a}, \ldots, f_{|U|,max}\right\} \end{array} \right\}.$$

The execution time of the task n_i on the processor core u_k with the frequency $f_{k,h}$ can be obtained by the following equation:

$$w_{i,k,h} = w_{i,k} \times \frac{f_{k,max}}{f_{k,h}}. \tag{3}$$

Then the energy consumption $E\left(n_i,\ u_k,\ f_{k,h}\right)$ of the task n_i on the processor core u_k with the frequency $f_{k,h}$ can be obtained by the following equation:

$$E\left(n_i,\ u_k,\ f_{k,h}\right) = \left(P_{k,ind} + C_{k,ef} \times \left(f_{k,h}\right)^{m_k}\right) \times w_{i,k} \times \frac{f_{k,max}}{f_{k,h}}. \tag{4}$$

Therefore, the energy consumption of application G will be

$$E(G) = \sum_{i=1}^{|N|} E\left(n_i, u_{pr(i)}, f_{pr(i),hz(i)}\right). \tag{5}$$

As a result of the P_{ind}, E is not monotonic with f and less f does not always result less energy. Therefore, we can get the minimum value of energy-effective frequency by finding the minimum value of Equation (4). Similar to [20–23], we define the minimum value of energy-effective frequency as f_{ee}. After calculation, we can get that

$$f_{ee} = \sqrt[m]{\frac{P_{ind}}{(m-1)C_{ef}}}. \tag{6}$$

When the execution frequency is less than f_{ee}, it is meaningless to continue to reduce the frequency, because this will increase energy consumption. Therefore, the range of execution frequency variation is $[f_{low}, f_{max}]$, where $f_{low} = \max(f_{min}, f_{ee})$. The new set of execution frequency becomes as follows:

$$\left\{ \begin{array}{c} \left\{f_{1,low}, f_{1,a}, \ldots, f_{1,max}\right\}, \\ \left\{f_{2,low}, f_{2,a}, \ldots, f_{2,max}\right\}, \\ \ldots, \\ \left\{f_{|U|,mi}, f_{|U|,a}, \ldots, f_{|U|,max}\right\} \end{array} \right\}.$$

3.3. Preliminaries

3.3.1. Problem Description

The problem to be solved in this study is to assign a suitable frequency and processor core to each task, and minimize the schedule length of the application under the condition that the energy consumption of the application does not exceed the energy consumption constraint [35–38].

First, we define the earliest start time (EST) and the earliest finish time (EFT) of tasks. Given a task n_i executed on processor u_k, its earliest start time (EST) is denoted as $EST(n_i, u_k)$, which is computed as

$$EST(n_i, u_k) = max\left\{avail[k], \max_{n_j \in pred(n_i)} \left\{AFT\left(n_j\right) + c'_{i,j}\right\}\right\}. \tag{7}$$

where $avail[k]$ is the earliest available time of processor core u_k, that is, all tasks executed on processor core u_k have been completed, and processor core u_k is ready to execute new tasks. $AFT\left(n_j\right)$ is the actual finish time of task n_j. $c'_{i,j}$ is the actual communication time between task n_i and n_j. If n_i and n_j are assigned to the same processor core, $c'_{i,j} = 0$; otherwise, $c'_{i,j} = c_{i,j}$.

The earliest finish time (EFT) of task n_i executed on processor u_k with frequency $f_{k,h}$ is the earliest start time plus the execution time of task n_i, which is computed as

$$EFT(n_i, u_k, f_{k,h}) = EST(n_i, u_k, f_{k,h}) + w_{i,k} \times \frac{f_{k,max}}{f_{k,h}}. \tag{8}$$

We define $SL(G)$ as the schedule length of application, where

$$SL(G) = AFT(n_{exit}).$$

We define $E_{given}(G)$ as the given energy consumption constraint of application G. Therefore, the problem to solve can be expressed as minimizing $SL(G)$ while

$$E(G) = \sum_{i=1}^{|N|} E\left(n_i, u_{pr(i)}, f_{pr(i),hz(i)}\right) \leq E_{given}(G). \tag{9}$$

where $u_{pr(i)}$ denotes the processor core assigned to the task n_i, and $f_{pr(i),hz(i)}$ denotes the execution frequency assigned to the task n_i.

3.3.2. Effective Range of Energy Consumption Constraint

Since the execution time of each task on each processor core is known, we can obtain the minimum and maximum energy consumption of n_i represented by $E_{min}(n_i)$ and $E_{max}(n_i)$ respectively by traversing all processors. $E_{min}(n_i)$ and $E_{max}(n_i)$ perform task n_i at minimum and maximum frequencies, respectively. The equations are as follows:

$$E_{min}(n_i) = \min_{u_k \in U} E\left(n_i, u_k, f_{k,min}\right), \tag{10}$$

$$E_{max}(n_i) = \max_{u_k \in U} E\left(n_i, u_k, f_{k,max}\right). \tag{11}$$

Therefore, the minimum and maximum energy consumption of application G can be computed as follows:

$$E_{min}(G) = \sum_{i=1}^{|N|} E_{min}(n_i), \tag{12}$$

$$E_{max}(G) = \sum_{i=1}^{|N|} E_{max}(n_i). \tag{13}$$

It should be noted that the given energy consumption constraint has a reasonable range. If $E_{given}(G) < E_{min}(G)$, the energy consumption constraint can never be satisfied; if $E_{given}(G) > E_{max}(G)$, the energy constraint can always be met. Both of the above situations are unreasonable, so the reasonable range of the given energy consumption constraint is $E_{min}(G) \leq E_{given}(G) \leq E_{max}(G)$.

3.3.3. Task Priority Determination

Before scheduling, we need to determine the priority of tasks. Similar to [20–23], we use the upward rank value ($rank_u$) as the criterion to determine the priority of tasks. $rank_u$ is defined as follows:

$$rank_u(n_i) = \frac{\sum_{k=1}^{|U|} w_{i,k}}{|U|} + \max_{n_i \in succ(n_i)} \{c_{i,j} + rank_u(n_j)\}. \tag{14}$$

The priority of tasks is sorted in descending order of $rank_u$, that is, the higher $rank_u$ of a task, the higher the priority of it. Table 3 shows the upward rank values of all the tasks in Figure 1. Therefore, the task priority list of the application in Figure 1 will be $\{n_1, n_3, n_4, n_2, n_5, n_6, n_9, n_7, n_8, n_{10}\}$.

Table 3. Upward rank values for tasks of the application in Figure 1.

Task	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
$rank_u$	108	77	80	80	69	63.33	42.67	35.67	44.33	14.67

3.4. The ISAECC Method

In this subsection, we review the existing method that is closest to us and reveal its limitations.

3.4.1. Method Description

The ISAECC method is proposed in [23], and it consists of several major steps:

1. It prioritizes tasks by using the upward rank value which is defined in Section 3.3.3;
2. It uses a self-defined energy consumption weight value to give each unscheduled task a pre-allocated energy consumption;

3. It calculates the energy consumption constraint of each task according to the given energy consumption constraint of the whole application and the pre-allocated energy consumption value of each task;
4. According to the order of the task priority list, it assigns each task the processor core and frequency that can minimize its EST time by traversing each processor core and optional execution frequency.

For the sake of generality, we use $\left\{n_{o(1)}, n_{o(2)}, \ldots, n_{o(|N|)}\right\}$ to represent the task priority order. Assuming that the currently scheduled task is $n_{o(j)}$, then the scheduled task set is $\left\{n_{o(1)}, n_{o(2)}, \ldots, n_{o(j-1)}\right\}$ and the unscheduled task set is $\left\{n_{o(j+1)}, n_{o(j+2)}, \ldots, n_{o(|N|)}\right\}$. Therefore, when scheduling the task $n_{o(j)}$, the overall energy consumption of application G can be expressed as

$$E_{o(j)}(G) = \sum_{x=1}^{j-1} E\left(n_{o(x)}, u_{pr(o(x))}, f_{pr(o(x)),hz(o(x))}\right) + E\left(n_{o(j)}\right) + \sum_{y=j+1}^{|N|} E_{pre}\left(n_{o(y)}\right), \tag{15}$$

where $E_{pre}\left(n_{o(y)}\right)$ denotes the pre-allocation energy consumption of task $n_{o(y)}$.

According to the energy consumption constraints shown in Equation (8), we can get

$$E_{o(j)}(G) \leq E_{given}(G). \tag{16}$$

Therefore, we can get

$$E\left(n_{o(j)}\right) \leq E_{given}(G) - \sum_{x=1}^{j-1} E\left(n_{o(x)}, u_{pr(o(x))}, f_{pr(o(x)),hz(o(x))}\right) - \sum_{y=j+1}^{|N|} E_{pre}\left(n_{o(y)}\right). \tag{17}$$

Let the energy consumption constraint of task $n_{o(j)}$ be

$$E_{given}\left(n_{o(j)}\right) = E_{given}(G) \ - \sum_{x=1}^{j-1} E\left(n_{o(x)}, u_{pr(o(x))}, f_{pr(o(x)),hz(o(x))}\right) - \sum_{y=j+1}^{|N|} E_{pre}\left(n_{o(y)}\right). \tag{18}$$

With Equation (17), we only need to consider the energy consumption constraint of each task which is shown as follows:

$$E\left(n_{o(j)}\right) \leq min\left\{E_{given}\left(n_{o(j)}\right), E_{max}\left(n_{o(j)}\right)\right\}. \tag{19}$$

Therefore, the key problem is how to determine the pre-allocation energy consumption ($E_{pre}\left(n_{o(j)}\right)$) of each task. The central idea of the method ISAECC used is to pre-allocate the energy consumption for unscheduled tasks by a weight mechanism. First, they define the improvable energy of application G called $E_{ie}(G)$, which is computed as

$$E_{ie}(G) = E_{given}(G) - E_{min}(G). \tag{20}$$

Then, they define $E_{ave}(n_i)$ and $E_{ave}(G)$ as the energy consumption level of task n_i and the energy consumption level of application G. $E_{ave}(n_i)$ and $E_{ave}(G)$ are computed as

$$E_{ave}(n_i) = \frac{E_{min}(n_i) + E_{max}(n_i)}{2}, \tag{21}$$

$$E_{ave}(G) = \frac{E_{min}(G) + E_{max}(G)}{2}. \tag{22}$$

Next, they define $el(n_i)$ as the weight of energy consumption level of task n_i, which is computed as

$$el(n_i) = \frac{E_{ave}(n_i)}{E_{ave}(G)}.$$

After that, they calculated the pre-allocated energy consumption for task n_i as follows:

$$E_{pre}(n_i) = min\{E_{wa}(n_i), E_{max}(n_i)\}$$

where $E_{wa}(n_i)$ is computed as

$$E_{wa}(n_i) = E_{ie}(G) \times el(n_i) + E_{min}(n_i).$$

After determining the pre-allocated energy consumption of each task, the task scheduling can be completed according to steps 3 and 4, described at the beginning of this section (Section 3.4.1).

3.4.2. Limitations of ISAECC

ISAECC solves the problem of increasing schedule length caused by unfair energy constraint allocation of low priority tasks by MSLECC in [20]. However, we find that it is not the best practice to treat each task fairly, because tasks in different levels have different impacts on the whole application. For example, in Figure 1, the task n_1 and task n_{10} should be assigned more energy than other tasks, because their execution time can affect the overall schedule length of the application G directly. It is obvious that if the execution time of task n_1 or task n_{10} is shortened or lengthened under other same conditions, the overall schedule length will shorten or lengthen the same amount accordingly. Other tasks like n_2 cannot be compared to n_1 and n_{10}. If the execution time of task n_2 is shortened or lengthened under other same conditions, the schedule length of application G may not change obviously, or even not change at all. Therefore, we should consider the levels of the tasks and the number of tasks in each level when pre-allocating the energy consumption of tasks. In addition, previous studies ignored the negative impact of the local optimal characteristics of scheduling algorithm on the schedule length. In our design, these problems have been greatly improved.

4. Our Solution

In this section, we introduce our new strategy in detail. First, we introduce a new task energy pre-allocation method considering task level (Section 4.1). Then, we give a new task scheduling algorithm to minimize the schedule length under the constraint of energy consumption (Section 4.2). Finally, we describe the task execution frequency re-adjustment mechanism we added after getting the preliminary scheduling results (Section 4.3).

4.1. The New Task Energy Pre-Allocation Method

The central idea of our pre-allocation method is to consider the levels of tasks. The impact of tasks in different hierarchies on the overall schedule length of an application is different. For example, in Figure 1, if we shorten the execution time of task n_1 by increasing its execution frequency, the schedule length of application G will certainly shorten the corresponding time; but if we shorten the execution time of task n_2, the schedule length of application G is unlikely to change much, because there are still many tasks in a similar position to it. Therefore, a more reasonable approach is to appropriately pre-allocate more energy consumption to task n_1 in the case of Figure 1. Based on this idea, we put forward a new energy pre-allocation method, based on the weight value of energy consumption and the levels of tasks.

4.1.1. Method Description

We define the level of tasks as follows:

$$\begin{cases} L\left(n_{entry}\right) = 0 \\ L(n_i) = \max\limits_{n_x \in pred(n_i)} \{L(n_x) + 1\} \end{cases}.$$

We define $N_l = \{n_i, n_j, \ldots\}$ as the set of tasks contained in level l, where $L(n_i) = L(n_j) = L(\ldots) = l$. The number of tasks in level l is $|N_l|$. We can have that the maximum of the level of tasks is $L(n_{exit})$.

We define the improvable energy of application G ($E_{ie}(G)$) and the energy consumption level of task n_i ($E_{ave}(n_i)$) the same as ISAECC. We define that the energy consumption level of the task level l is the sum of energy consumption level in level l, which is computed as

$$E_{ave}(N_l) = \sum_{n_i \in N_l} E_{ave}(n_i). \tag{23}$$

We define $el(n_{i,l})$ as the energy consumption weight of task n_i in its level l, which is computed as

$$el(n_{i,l}) = \frac{E_{ave}(n_i)}{E_{ave}(N_l)}. \tag{24}$$

We define $E_{var}(N_l)$ as the variation energy consumption of N_l, which is computed as

$$E_{var}(N_l) = \frac{E_{ave}(N_l)}{K},\ K = \begin{cases} |N_l|,\ |N_l| < |U| \\ |U|,\ |N_l| \geq |U| \end{cases}. \tag{25}$$

Correspondingly, the variation energy consumption of application G is computed as

$$E_{var}(G) = \sum_{l} E_{var}(N_l). \tag{26}$$

The energy consumption weight of N_l in application G can be defined as

$$el(N_l) = \frac{E_{var}(N_l)}{E_{var}(G)}. \tag{27}$$

We define $E_{ie}(N_l)$ as the improvable energy of N_l, which is computed as

$$E_{ie}(N_l) = E_{ie}(G) \times el(N_l). \tag{28}$$

Therefore, we can get the new energy pre-allocation formula of n_i as follows:

$$E_{pre}(n_i) = min\{E_{wa}(n_i), E_{max}(n_i)\}, \tag{29}$$

where

$$E_{wa}(n_i) = E_{ie}(N_l) \times el(n_{i,l}) + E_{min}(n_i). \tag{30}$$

4.1.2. Feasibility of the Task Energy Pre-Allocation Mechanism

In order to prove the feasibility of our method, we need to prove the following theorem: Given an application G, if the unscheduled tasks are pre-allocated energy consumption according to our method, then each task n_j can satisfy Equation (15).

We use mathematical induction to prove the above theorem. First, we need to prove task $n_{o(1)}$ can satisfy Equation (15), and the other tasks are all unscheduled. By Equations (14), (22)–(29), we can have

$$\begin{aligned} E_{o(1)}(G) &= E(n_{o(1)}) + \textstyle\sum_{y=2}^{|N|} E_{pre}(n_{o(y)}) \\ &\leq E(n_{o(1)}) + \textstyle\sum_{y=2}^{|N|} E_{wa}(n_{o(y)}) \\ &= E(n_{o(1)}) + \textstyle\sum_{y=1}^{|N|} E_{wa}(n_{o(y)}) - E_{wa}(n_{o(1)}) \\ &= E(n_{o(1)}) + E_{given}(G) - E_{wa}(n_{o(1)}). \end{aligned}$$

We know that

$$E_{wa}\left(n_{o(1)}\right) = E_{ie}(N_l) \times el\left(n_{i,l}\right) + E_{min}(n_i) \geq E_{min}\left(n_{o(1)}\right). \tag{31}$$

We can at least find a situation in which

$$E\left(n_{o(1)}\right) = E_{min}\left(n_{o(1)}\right).$$

In other words, at least when $E\left(n_{o(1)}\right) = E_{min}\left(n_{o(1)}\right)$, we can have

$$E_{o(1)}(G) \leq E\left(n_{o(1)}\right) + E_{given}(G) - E_{wa}\left(n_{o(1)}\right) \leq E_{given}(G).$$

From the above derivation, we prove that task $n_{o(1)}$ can satisfy Equation (15).
Then, we assume that task $n_{o(j)}$ can satisfy Equation (15). That is

$$\begin{aligned} E_{o(j)}(G) &= \sum_{x=1}^{j-1} E\left(n_{o(x)}, u_{pr(o(x))}, f_{pr(o(x)),hz(o(x))}\right) + E\left(n_{o(j)}\right) + \sum_{y=j+1}^{|N|} E_{pre}\left(n_{o(y)}\right) \\ &= \sum_{x=1}^{j} E\left(n_{o(x)}, u_{pr(o(x))}, f_{pr(o(x)),hz(o(x))}\right) + \sum_{y=j+1}^{|N|} E_{pre}\left(n_{o(y)}\right) \\ &\leq E_{given}(G). \end{aligned}$$

The above formulation can be written as

$$\sum_{x=1}^{j} E\left(n_{o(x)}, u_{pr(o(x))}, f_{pr(o(x)),hz(o(x))}\right) \leq E_{given}(G) - \sum_{y=j+1}^{|N|} E_{pre}\left(n_{o(y)}\right). \tag{32}$$

Next, we prove task $n_{o(j+1)}$ can satisfy Equation (15). By Equation (30), we can have

$$\begin{aligned} E_{o(j+1)}(G) &= \sum_{x=1}^{j} E\left(n_{o(x)}, u_{pr(o(x))}, f_{pr(o(x)),hz(o(x))}\right) + E\left(n_{o(j+1)}\right) + \sum_{y=j+2}^{|N|} E_{pre}\left(n_{o(y)}\right) \\ &\leq E_{given}(G) - \sum_{y=j+1}^{|N|} E_{pre}\left(n_{o(y)}\right) + E\left(n_{o(j+1)}\right) + \sum_{y=j+2}^{|N|} E_{pre}\left(n_{o(y)}\right) \\ &= E_{given}(G) + E\left(n_{o(j+1)}\right) - E_{pre}\left(n_{o(j+1)}\right). \end{aligned}$$

From Equations (28) and (30), we can have

$$E_{pre}\left(n_{o(j+1)}\right) = min\left\{E_{wa}\left(n_{o(j+1)}\right), E_{max}\left(n_{o(j+1)}\right)\right\} \geq E_{min}\left(n_{o(j+1)}\right).$$

Therefore, at least when $E\left(n_{o(j+1)}\right) = E_{min}\left(n_{o(j+1)}\right)$, we can have

$$E_{o(j+1)}(G) \leq E_{given}(G) + E\left(n_{o(j+1)}\right) - E_{pre}\left(n_{o(j+1)}\right) \leq E_{given}(G).$$

In summary, given an application G, if the unscheduled tasks are pre-allocated energy consumption according to our method, then each task n_j can satisfy Equation (15). The feasibility of our method has been proved.

4.2. The Proposed Algorithm for Minimizing Schedule Length

In this section, we show our new task scheduling algorithm in Algorithm 1. In the algorithm, Line 1 is to prioritize tasks in the input application; Lines 2–10 calculate some required values for each task, each level and the application G; Lines 11 and 12 calculate the pre-allocation energy consumption of each task; Lines 13–26 are to select processor and frequency for each task; Lines 27 and 28 are to calculate the actual energy consumption E(G) and the final schedule length SL(G).

Algorithm 1. A new scheduling algorithm for minimizing schedule length.

Input: G = {N, W, C}, U, $E_{given}(G)$
Output: SL(G), E(G)
Sort tasks in a list *tl* by descending order of $rank_u$;
for $(\forall i,\ n_i \in N)$ **do**
  Compute $E_{min}(n_i)$ and $E_{max}(n_i)$; //By (10) (11)
  Compute $E_{ave}(n_i)$; //By (21)
for $(\forall l,\ 1 \le l \le L(n_{exit}))$ **do**
  Compute $E_{ave}(N_l)$; //By (23)
  Compute $E_{var}(N_l)$; //By (25)
Compute $E_{min}(G)$ and $E_{max}(G)$; //By (12) (13)
Compute $E_{ave}(G)$; //By (22)
Compute $E_{var}(G)$; //By (26)
for $(\forall i,\ n_i \in N)$ **do**
  Compute $E_{pre}(n_i)$; //By (29)
while $(tl \neq \varnothing)$ **do**
  $n_i = tl$.out();
  $AFT(n_i) = \infty$;
  Compute $E_{given}(n_i)$; //By (18)
  for $(\forall k,\ u_k \in U)$ **do**
    for $(\forall f_{k,h},\ f_{k,h} \in [f_{k,low}, f_{k,max}])$ **do**
      Compute $E(n_i,\ u_k,\ f_{k,h})$; //By (4)
      if $(E(n_i,\ u_k,\ f_{k,h}) > E_{given}(n_i))$ **then**
        continue;
      Compute $EFT(n_i,\ u_k,\ f_{k,h})$; //By (8)
      if $(EFT(n_i,\ u_k,\ f_{k,h}) < AFT(n_i))$ **then**
        Let $u_{pr(i)} = u_k$ and $f_{pr(i),hz(i)} = f_{k,h}$;
        $E(n_i,\ u_{pr(i)},\ f_{pr(i),hz(i)}) = E(n_i,\ u_k,\ f_{k,h})$;
        $AFT(n_i) = EFT(n_i,\ u_{pr(i)},\ f_{pr(i),hz(i)})$;
Compute actual energy consumption $E(G)$;
Compute the schedule length $SL(G)$;
return SL(G), E(G).

For each task, selecting the processor with the minimum EFT has complexity $O(|N| \times |U| \times |F|)$, where $|F|$ represents the maximum number of discrete frequencies from $f_{k,low}$ to $f_{k,max}$. Therefore, the complexity of Algorithm 1 is $O(|N|^2 \times |U| \times |F|)$ the same as ISAECC in [23].

4.3. The Task Execution Frequency Re-Adjustment Mechanism

Through the new task energy pre-allocation method in Section 4.1 and the new task scheduling algorithm in Section 4.2, we can get the preliminary scheduling results. Other methods generally end here such as those in [20–23], but they do not realize that the scheduling results can be optimized to further shorten the schedule length. The same as the scheduling algorithms in [20–23], the algorithm in Section 4.2 makes tasks finish as soon as possible when scheduling them, which is not entirely reasonable. Premature completion of some tasks cannot shorten the overall schedule length, but will take up more energy consumption. Therefore, we introduce the concept of the latest finish time of tasks [28,39,40], which is defined as follows:

$$\begin{cases} LFT(n_{exit}) = AFT(n_{exit}) \\ LFT(n_i) = min\left\{ \min\limits_{n_j \in succ(n_i)} \left\{ AST(n_j) - c'_{i,j} \right\}, AST(n_{dn(i)}) \right\} \end{cases}, \quad (33)$$

where $n_{dn(i)}$ represents the downward neighbor task of n_i, that is, $n_{dn(i)}$ is on the same processer core as n_i and it is the first case after n_i.

Through Equation (32), we can replace AFT of tasks with LFT to delay the finish time of some tasks without increasing the schedule length. As the execution time of tasks is prolonged, their running frequency will be reduced accordingly, which can save some energy consumption. Therefore, the new execution time of task n_i is changed from $AFT(n_i) - AST(n_i)$ to $LFT(n_i) - AST(n_i)$, and the new frequency of task n_i can be changed as follows:

$$f_{pr(i),nhz(i)} = f_{pr(i),hz(i)} \times \frac{AFT(n_i) - AST(n_i)}{LFT(n_i) - AST(n_i)}.$$

The frequency range of task n_i is $\left[f_{pr(i),low}, f_{pr(i),max}\right]$, so $f_{pr(i),nhz(i)}$ should be

$$f_{pr(i),nhz(i)} = max\left\{f_{pr(i),hz(i)} \times \frac{AFT(n_i) - AST(n_i)}{LFT(n_i) - AST(n_i)}, f_{pr(i),low}\right\}. \tag{34}$$

Therefore, the actual execution time (AET) of task n_i will be

$$AET(n_i) = w_{i,pr(i)} \times \frac{f_{pr(i),max}}{f_{pr(i),nhz(i)}}. \tag{35}$$

Finally, the new $AST(n_i)$ should be updated as

$$ST(n_i) = LFT(n_i) - AET(n_i) = LFT(n_i) - w_{i,pr(i)} \times \frac{f_{pr(i),max}}{f_{pr(i),nhz(i)}}. \tag{36}$$

On the basis of above equations, the algorithm to save energy after preliminary scheduling is shown in Algorithm 2. In Line 2, we reorder the tasks in descending order of the actual finish time (AFT) of tasks according to the scheduling results of Algorithm 1. In Lines 4–10, $AFT(n_i)$, $AST(n_i)$ and $f_{pr(i),hz(i)}$ are updated. In Line 11, we compute the new $E(n_i)$ using the above updated values. In Lines 12 and 13, we compute the saved energy $E_{save}(G)$.

Algorithm 2. The method to save energy.

Input: G = {N, W, C}, U, $E_{given}(G)$
Output: $E_{save}(G)$
1: Call Algorithm 1 to get the preliminary scheduling results;
2: Sort the tasks in the list *al* according to the descending $AFT(n_i)$ order of values;
3: **while** $(al \neq \varnothing)$ **do**
4: $n_i = tl$.out();
5: Compute $LFT(n_i)$; //By (33)
6: Compute the new frequency $f_{pr(i),nhz(i)}$; //By (34)
7: Compute the new $AET(n_i)$; //By (35)
8: Update $AFT(n_i) \leftarrow LFT(n_i)$;
9: Update $AST(n_i) \leftarrow LFT(n_i) - AET(n_i)$; //By (36)
10: Update $f_{pr(i),hz(i)} \leftarrow f_{pr(i),nhz(i)}$;
11: Compute the new $E(n_i)$; //By (4)
12: Compute the new energy consumption of G $E_{new}(G)$; //By (5)
13: Compute the saved energy $E_{save}(G) = E(G) - E_{new}(G)$;
14: **return** $E_{save}(G)$.

Generally speaking, after the task scheduling is completed, there will be remaining energy consumption that is not used up as follows:

$$E_{rem}(G) = E_{given}(G) - E(G). \tag{37}$$

Therefore, the energy consumption that can be reused is

$$E_{reu}(G) = E_{rem}(G) + E_{save}(G). \tag{38}$$

After calculating the energy consumption that can be reused, we need to re-allocate the energy consumption to the tasks that can directly affect the overall schedule length. Directly affecting the overall schedule length means that according to how much the task running time changes, the overall schedule length will also change. For example, task n_1 in Figure 1, the total schedule length will be shortened or extended as much as its execution time is shortened or extended. However, due to the diversity of task models, it is very difficult to find out which tasks can directly affect the overall schedule length. Therefore, we adopt a more direct way: If the application model level is strict (the tasks in level l only communicate with the tasks in their adjacent levels which are level l+1 and level l−1), we re-allocate the reused energy consumption ($E_{reu}(G)$) to the tasks that have not reached the highest execution frequency in the levels whose $|N| = 1$; otherwise, we only re-allocate the reused energy consumption to n_{entry} and n_{exit}. For ease of description, we define N_{das} as the set of tasks that can directly affect the overall schedule length.

After the assignment object of $E_{reu}(G)$ is determined, we need to determine the allocation proportion. We define the maximum energy consumption of $n_i \in N_{das}$ on processer core $u_{pr(i)}$ is

$$E_{max}\left(n_i, u_{pr(i)}\right) = \left(P_{pr(i),ind} + C_{pr(i),ef} \times \left(f_{pr(i),max}\right)^{m_{pr(i)}}\right) \times w_{i,pr(i)}. \tag{39}$$

Therefore, the growable energy consumption of $n_i \in N_{das}$ on processer core $u_{pr(i)}$ will be

$$E_{gro}(n_i) = E_{max}(n_i, pr(i)) - E(n_i). \tag{40}$$

We take the values of $E_{gro}(n_i)$ as the allocation proportion of $E_{reu}(G)$. Therefore, the reused energy consumption of $n_i \in N_{das}$ will be

$$E_{reu}(n_i) = E_{reu}(G) \times \frac{E_{gro}(n_i)}{\sum_{i=1}^{|N|} E_{gro}(n_i)}. \tag{41}$$

Adding $E_{reu}(n_i)$ and $E(n_i)$ which is computed in Algorithm 2, we can get the energy consumption that $n_i \in N_{das}$ can use will be

$$E_{use}(n_i) = E_{reu}(n_i) + E(n_i). \tag{42}$$

According to $E_{use}(n_i)$, we can find the new frequency $f_{pr(i),nh}$ of $n_i \in N_{das}$ by traversing the execution frequency on processor core $u_{pr(i)}$. Then we can compute the shortened actual execution time of $n_i \in N_{das}$ as follows:

$$AET_{shor}(n_i) = w_{i,pr(i)} \times f_{pr(i),max} \times \left(\frac{1}{f_{pr(i),h}} - \frac{1}{f_{pr(i),nh}}\right). \tag{43}$$

Therefore, the new length of the application G will be

$$SL_{new}(G) = SL(G) - \sum_{n_i \in N_{das}} AET_{shor}(n_i). \tag{44}$$

Combined with the Algorithms 1 and 2, the task execution frequency re-adjustment mechanism is shown in Algorithm 3. Lines 1 and 2 call Algorithms 1 and 2 to get the preliminary scheduling results and $E_{save}(G)$. Line 3 compute the reused energy consumption $E_{reu}(G)$. Lines 6 and 7 calculate some required values for each task belonging to N_{das}. Lines 8–14 are to select the new frequency for each task belonging to N_{das}. Finally, we compute the new schedule length of the application G $SL_{new}(G)$

after the task execution frequency re-adjustment mechanism in Line 15. The value of $SL_{new}(G)$ is the final minimum schedule length we get.

In general, our method includes Algorithms 1–3. Algorithm 1 is a task energy pre-allocation strategy, and Algorithms 2 and 3 constitute the task execution frequency re-adjustment mechanism. For a given application, we operate in the order of Algorithms 1, Algorithm 2 and Algorithm 3, then we can get the scheduling result of minimizing the schedule length of the application.

Algorithm 3. The task execution frequency re-adjustment mechanism.

Input: G = {N, W, C}, U, $E_{given}(G)$
Output: $SL(G)$
1: Call Algorithm 1 to get the preliminary scheduling results;
2: Call Algorithm 2 to get $E_{save}(G)$;
3: Compute $E_{reu}(G)$; //By (38)
4: **for** ($\forall n_i, n_i \in N_{das}$) **do**
5: $E(n_i) = 0$;
6: Compute $E_{gro}(n_i)$; //By (40)
7: Compute $E_{use}(n_i)$; //By (42)
8: **for** ($\forall f_{pr(i),h}$, $f_{pr(i),h} \in \left[f_{pr(i),low}, f_{pr(i),max}\right]$) **do**
9: Compute $E\left(n_i, u_{pr(i)}, f_{pr(i),h}\right)$; //By (4)
10: **if** ($E\left(n_i, u_{pr(i)}, f_{pr(i),h}\right) > E_{use}(n_i)$) **then**
11: continue;
12: **if** ($E\left(n_i, u_{pr(i)}, f_{pr(i),h}\right) > E(n_i)$) **then**
13: Let $E(n_i) = E\left(n_i, u_{pr(i)}, f_{pr(i),h}\right)$;
14: Let $f_{pr(i),nh} = f_{pr(i),h}$;
15: Compute $SL_{new}(G)$; //By (44)
16: Update $SL(G) \leftarrow SL_{new}(G)$;
17: **return** $SL(G)$.

5. Experiments

In this section, we use four algorithms, MSLECC [20], WALECC [21], EECC [22] and ISAECC [23], which are the same as the goal of this article to compare with our proposed method. The configuration of the experimental platform is AMD Ryzen 5 2500U CPU @ 2.00 GHz, 8 GB RAM (Santa Clara, CA, USA), 64-bit Windows 10 Home Edition. The whole set of codes is mainly implemented by C and scripts. The final schedule length SL(G) is the only evaluation standard of these algorithms.

The parameters of the processors and applications as follows: 10 ms $\leq w_{i,k} \leq$ 100 ms, 10 ms $\leq c_{i,j} \leq$ 100 ms, 0.03 $\leq P_{k,ind} \leq$ 0.07, 0.8 $\leq C_{k,ef} \leq$ 1.2, 2.5 $\leq m_k \leq$ 3.0, and $f_{k,max} = 1$ GHz. The execution frequency is discrete, and the precision is 0.01 GHz. The simulated heterogeneous platform for testing the problem of minimizing the schedule length uses four processor cores.

We chose two DAG models to evaluate our algorithm, which are two real-world applications (Fast Fourier transform and Gaussian elimination).

5.1. Fast Fourier Transform Application

We first consider the fast Fourier transform (FFT), Figure 2a shows an example of the FFT parallel application with $\rho = 4$. The parameter ρ can represent the size of application models. For FFT application, the number of tasks is $|N| = (2 \times \rho - 1) + \rho \times \log_2 \rho$, and $\rho = 2^x$ where x is an integer. We can see that there are four exit tasks (task 12, 13, 14, 15) in the FFT graph in Figure 2a, and there will be ρ exit tasks in the FFT graph with parameter ρ. In order to match the application model in Section 3.1, we add a dummy exit task whose execution time is 0. We also connect the dummy exit task to the last ρ exit tasks, and we set their communication time is 0. For example, Figure 2b shows the changed FFT parallel application with $\rho = 4$ which is added a dummy exit task.

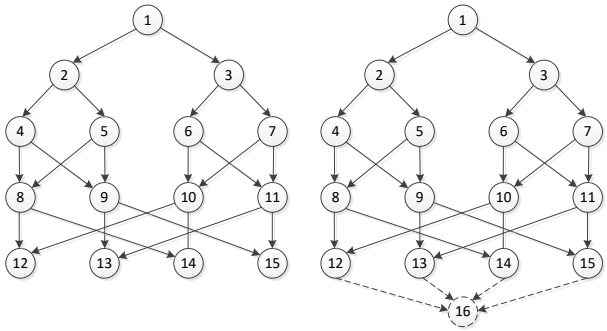

Figure 2. Fast Fourier transform (FFT) application model with $\rho = 4$: (**a**) original FFT application model; (**b**) FFT application model with a dummy exit task.

Experiment 1. In order to observe the performance on different energy consumption constraints, this experiment is carried out to compare the final schedule length values of the FFT application for varying energy consumption constraints. We use the FFT application with $\rho = 32$, that is, the number of tasks is 233 ($|N| = 233$). We set the energy consumption constraints $E_{given}(G) = E_{min}(G) + \frac{M}{233}(E_{max}(G) - E_{min}(G))$, where $1 \leq M \leq 232$ and M is an integer.

Table 4 shows the details of the final schedule lengths of FFT application with $\rho = 32$ for varying $E_{given}(G)$ by using all the algorithms, and a more intuitive feeling can be performed through Figure 3. It can be seen that our algorithm has the obvious advantage on the schedule length $SL(G)$ compared to other algorithms. From the experimental results, we can get that our method outperforms MSLECC by about 28.13%~36.35%, and it outperforms the newest method ISAECC by about 3.65%~10.48%. The results of WALECC and EECC are similar to that of ISAECC.

Table 4. Results of FFT applications with $\rho = 32$ for varying $E_{given}(G)$.

$E_{given}(G)$	$SL(G)$				
	MSLECC	**WALECC**	**EECC**	**ISAECC**	**Our method**
2086	10,989	8023	7955	7814	6846
2714	5915	4491	4564	4417	3887
3342	4644	3302	3487	3390	3108
4048	4052	2918	2937	2966	2779
5226	3381	2546	2609	2484	2388
6168	2947	2253	2315	2273	2190

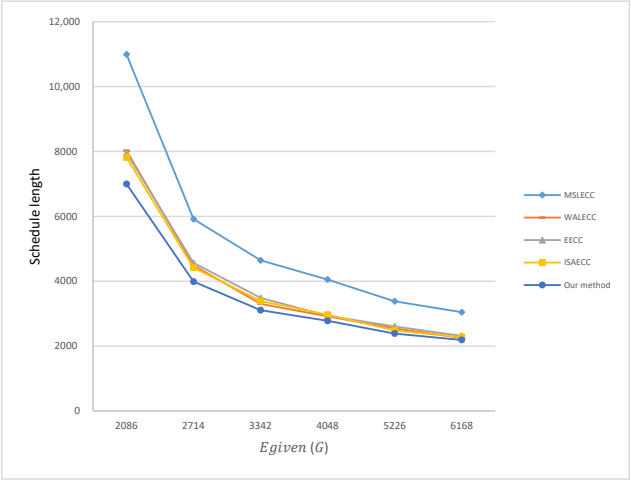

Figure 3. Final schedule length of FFT application varying $E_{given}(G)$.

Further, we can find that the gaps between our method and other algorithms increase when $E_{given}(G)$ decreases. This is because all tasks can be assigned to a relatively large energy consumption constraint when the energy consumption constraint is large, so that the impact of task level is smaller. Moreover, at this time, most tasks belonging to N_{das} have reached the maximum frequency, and cannot continue to increase the frequency to shorten the schedule length. In addition, as we expected, the larger $E_{given}(G)$ is, the shorter schedule length we can obtain.

Experiment 2. In order to observe the algorithm performance under different number of tasks, an experiment is carried out to compare the final schedule length values of the FFT application for varying number of tasks. The parameter ρ is changed from 8 to 256. In order to get relatively obvious results, we set the energy consumption constraints to a relatively small value: $E_{given}(G) = E_{min}(G) + \frac{E_{max}(G)-E_{min}(G)}{8}$.

Table 5 shows the results of FFT applications for different number of tasks by using all the algorithms, and a more intuitive feeling can be performed through Figure 4. The results show that our method has better performance than other algorithms. Our method outperforms MSLECC by about 21.24%–31.10%, and outperforms ISAECC by about 4.80%–7.32%. From the results, we can find that the more tasks we have, the better our method performs. This is because that there will be more different hierarchies in FFT graphs as the number of tasks increases, so that our new energy pre-allocation strategy will become more advantageous.

Table 5. Results of FFT applications for varying number of tasks.

ρ	*SL*(*G*)				
	MSLECC	**WALECC**	**EECC**	**ISAECC**	**Our Method**
8	805	654	679	666	634
16	1619	1331	1358	1294	1227
32	3664	2960	2777	2814	2641
64	8433	6340	6314	6326	5922
128	18,954	13,892	13,975	14,131	13,168
256	43,524	31,982	32,814	32,354	29,987

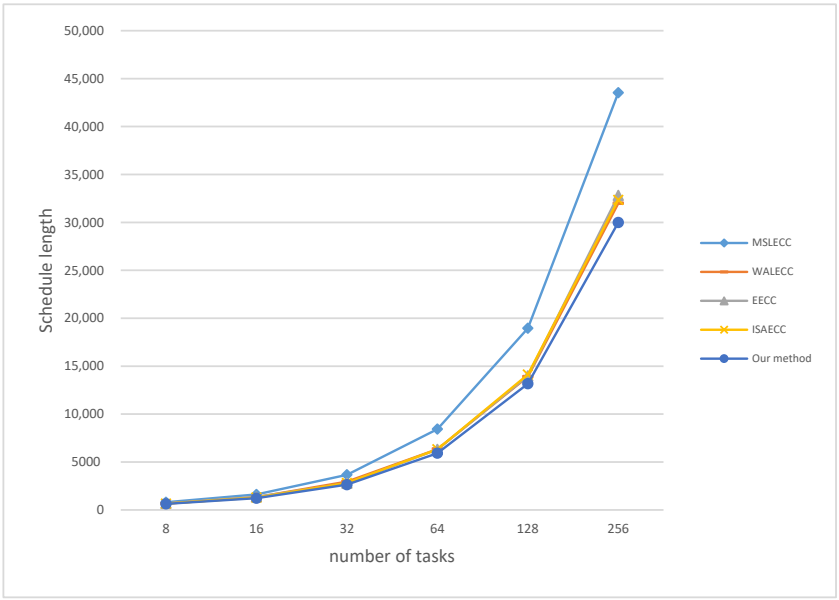

Figure 4. Final schedule length of FFT application varying number of tasks.

5.2. Gaussian Elimination Application

Similarly, in Gaussian elimination (GE) application, we define ρ as the size of the application, and the number of tasks can be calculated by $|N| = \frac{\rho^2+\rho-2}{2}$. Figure 5 shows the GE application model with $\rho = 5$.

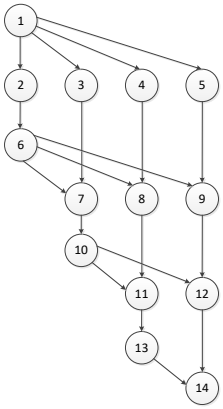

Figure 5. Gaussian elimination (GE) application model with $\rho = 5$.

Experiment 3. This experiment compares the final schedule length values of GE application for varying energy consumption constraints. We use the GE application with $\rho = 21$, that is, the number of tasks is 230 ($|N| = 230$). We set the energy consumption constraints $E_{given}(G) = E_{min}(G) + \frac{M}{230}(E_{max}(G) - E_{min}(G))$, where $1 \leq M \leq 229$ and M is an integer.

Table 6 shows the results of the final schedule lengths of GE application with $\rho = 21$ for varying $E_{given}(G)$ by using all the algorithms, and a more intuitive feeling can be performed through Figure 6. We can see that our method still performs better than other algorithms, specifically, it outperforms MSLECC by about 14.69%–34.55%, and outperforms ISAECC by about 3.94%–9.60%, but the improvement is not obvious compared with FFT models. This is because that the level of GE application is not as strict as the FFT application, so that the effect of our method has diminished.

Table 6. Results of GE applications with $\rho = 21$ for varying $E_{given}(G)$.

$E_{given}(G)$	$SL(G)$				
	MSLECC	**WALECC**	**EECC**	**ISAECC**	**Our Method**
2436	6826	6144	5912	5973	5692
2763	6155	5389	5603	5398	5128
3665	5519	4457	4521	4324	4119
6535	4721	3776	3502	3418	3090
7191	3864	3093	3214	3186	3043
8749	3397	3012	3129	3017	2898

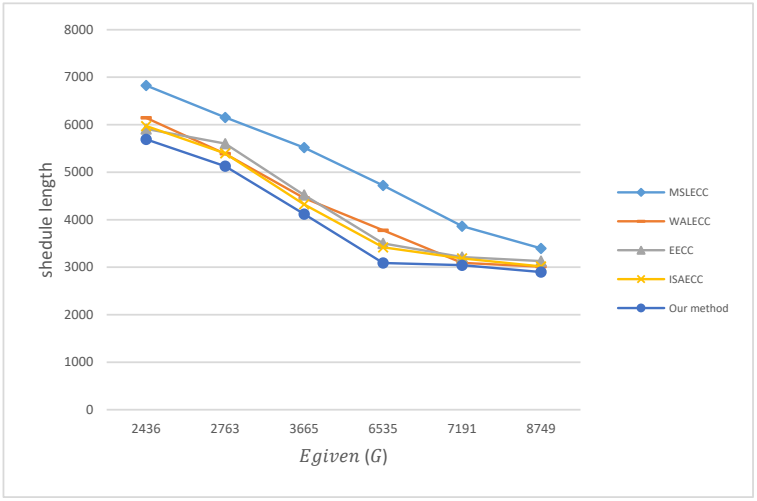

Figure 6. Final schedule length of GE application varying $E_{given}(G)$.

Experiment 4. This experiment compares the final schedule length values of GE application for varying number of tasks. We set that $E_{given}(G) = E_{min}(G) + \frac{E_{max}(G)-E_{min}(G)}{4}$ and change the number of tasks.

Table 7 shows the results of the final schedule lengths of GE application for varying number of tasks by using all the algorithms, and a more intuitive feeling can be performed through Figure 7. Our method still has a better effect on the schedule length than other algorithms which performs better 16.00%–28.85% than MSLECC and 3.36%–10.98% than ISAECC.

Table 7. Results of GE applications for varying number of tasks.

ρ	*SL(G)*				
	MSLECC	**WALECC**	**EECC**	**ISAECC**	**Our Method**
8	1056	897	911	893	847
13	2185	1794	1805	1737	1599
21	3698	3269	3082	3214	3106
41	18,845	14,504	14,729	14,563	13,534
83	39,584	31,375	31,551	31,422	30,032
100	59,951	46,210	48,238	47,916	42,657

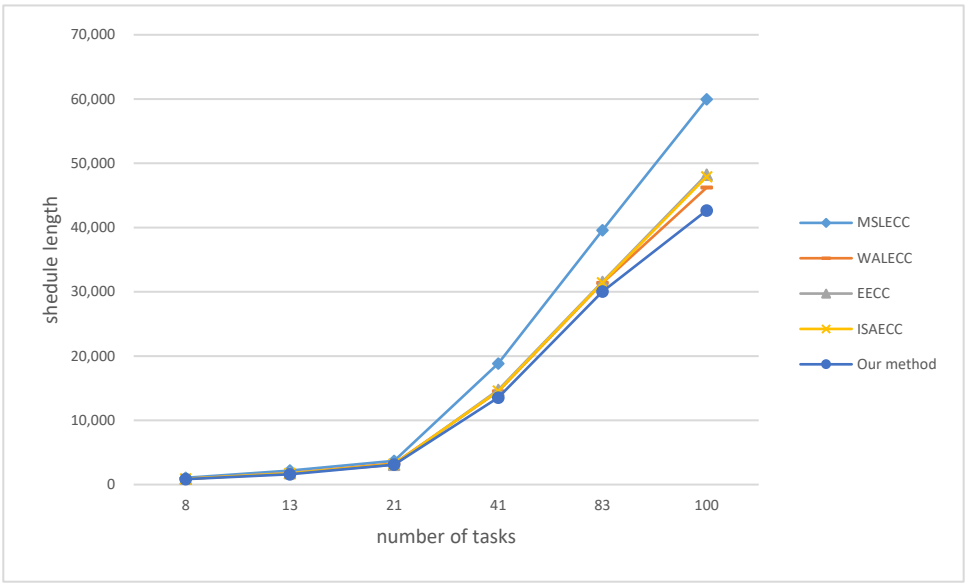

Figure 7. Final schedule length of GE application varying number of tasks.

5.3. Analysis and Summary of Experimental Results

We can obtain that our method has better performance compared with the other algorithms from the experimental results. However, when $E_{given}(G)$ becomes larger or the number of tasks is relatively small, the advantage of our method is not particularly obvious. The former is because that all tasks can be allocated to a relatively large energy consumption constraint when $E_{given}(G)$ is large, so that the impact of task level will be smaller. The latter is because that the experimental results are more accidental and cannot reflect the general law when the number of tasks is small. We can also find that our method performs better on FFT models than on GE models with similar $E_{given}(G)$ and the number of tasks. This is because that the task level of GE application is not as strict as the FFT application, so that the effect of our method has diminished. In summary, our method is more applicable when the energy constraints are stringent, the number of tasks is large, or the task level of the application is strict.

6. Conclusions

In this article, we propose a novel method to minimize the scheduling length of energy-constrained applications which run on a heterogeneous multi-core system. Our method mainly includes two parts: a novel task energy pre-allocation strategy and the schedule algorithm based on it; a re-adjustment mechanism of task execution frequency after preliminary scheduling. The core idea of our method is that the tasks in different hierarchies have different impacts on the whole application and the negative impact of local optimal scheduling should be reduced. Our method can be integrated into actual multi-core embedded systems, and it is particularly suitable for wearable devices, mobile robots and other products with high requirements for energy saving and performance. We carry out a considerable number of experiments with two practical parallel application models (FFT and GE). The results of experiments show that our method is generally superior to other existing algorithms. However, the experimental results also demonstrate the limitations of our method, which are that our method does not offer much of an advantage when the energy constraints are not stringent, the number of tasks is small or the task level of the application is not strict.

In the future, we will improve and extend our method, and some further studies will be done. The points that can be further studied are as follows:

- Consider other factors that affect the length of application scheduling, such as the way to determine the priority of tasks.
- Explore ways to improve the limitations of our method and make it more universal.
- Integrate our method into an actual embedded multi-core system and test its performance.
- Extend our method to study other indicators in multi-core task scheduling, such as reliability.

Author Contributions: Conceptualization, M.J. and X.J.; methodology, K.H., M.J. and X.J.; software, M.J.; validation, K.H., M.J. and S.C.; formal analysis, M.J. and X.J.; investigation, M.J., S.C., X.L. and W.T.; resources, K.H., X.J. and Z.L.; data curation, M.J., X.L. and W.T.; writing—original draft preparation, M.J.; writing, review and editing, K.H., M.J., X.J. and D.X.; visualization, K.H.; supervision, K.H. and X.J.; project administration and funding acquisition, K.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key R&D Program of China (2020YFB0906000, 2020YFB0906001).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Weiser, M.; Welch, B.; Demers, A.; Shenker, S. *Scheduling for Reduced CPU Energy*; Springer: Boston, MA, USA, 1994.
2. Li, Keqin. Scheduling Precedence Constrained Tasks with Reduced Processor Energy on Multiprocessor Computers. *IEEE Trans. Comput.* **2012**, *61*, 1668–1681. [CrossRef]
3. Xie, G.; Zeng, G.; Li, R.; Li, K. Energy-Aware Processor Merging Algorithms for Deadline Constrained Parallel Applications in Heterogeneous Cloud Computing. *IEEE Trans. Sustain. Comput.* **2017**, *2*, 62–75. [CrossRef]
4. Kang, L.; Wang, Z.J.; Quan, Z.; Wu, W.; Guo, S.; Li, K.; Li, K. An Efficient Method for Optimizing PETSc on the Sunway TaihuLight System. In Proceedings of the 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Guangzhou, China, 8–12 October 2018.
5. Islam, M.A.; Ren, S.; Mahmud, A.H.; Quan, G. Online Energy Budgeting for Cost Minimization in Virtualized Data Center. *IEEE Trans. Serv. Comput.* **2016**, *9*, 421–432. [CrossRef]
6. Zhang, J.; Wang, Z.J.; Quan, Z.; Yin, J.; Chen, Y.; Guo, M. Optimizing power consumption of mobile devices for video streaming over 4G LTE networks. *Peer Peer Netw. Appl.* **2017**, *11*, 1101–1114. [CrossRef]
7. Ranjan, R.; Wang, L.; Zomaya, A.Y.; Georgakopoulos, D.; Sun, X.H.; Wang, G. Recent advances in autonomic provisioning of big data applications on clouds. *IEEE Trans. Cloud Comput.* **2015**, *3*, 101–104. [CrossRef]

8. Xu, C.; Wang, K.; Li, P.; Guo, S.; Luo, J.; Ye, B.; Guo, M. Making Big Data Open in Edges: A Resource-Efficient Blockchain-Based Approach. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 870–882. [CrossRef]
9. Jin, P.; Hao, X.; Wang, X.; Yue, L. Energy-Efficient Task Scheduling for CPU-Intensive Streaming Jobs on Hadoop. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *30*, 1298–1311. [CrossRef]
10. Choi, Y.; Rhu, M. PREMA: A Predictive Multi-Task Scheduling Algorithm for Preemptible Neural Processing Units. In Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), San Diego, CA, USA, 22–26 February 2020.
11. Xu, C.; Wang, K.; Guo, M. Intelligent Resource Management in Blockchain-Based Cloud Datacenters. *IEEE Cloud Comput.* **2018**, *4*, 50–59. [CrossRef]
12. Fu, Z.; Tang, Z.; Yang, L.; Liu, C. An Optimal Locality-Aware Task Scheduling Algorithm Based on Bipartite Graph Modelling for Spark Applications. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 2406–2420. [CrossRef]
13. Xie, G.; Zeng, G.; Liu, Y.; Zhou, J.; Li, R.; Li, K. Fast Functional Safety Verification for Distributed Automotive Applications during Early Design Phase. *IEEE Trans. Ind. Electron.* **2017**, *65*, 4378–4391. [CrossRef]
14. Cho, H.; Kim, C.; Sun, J.; Easwaran, A.; Park, J.D.; Choi, B.C. Scheduling Parallel Real-Time Tasks on the Minimum Number of Processors. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *31*, 171–186. [CrossRef]
15. Gharehbaghi, K.; Koçer, F.; Külah, H. Optimization of Power Conversion Efficiency in Threshold Self-Compensated UHF Rectifiers with Charge Conservation Principle. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2017**, *64*, 2380–2387. [CrossRef]
16. Sandoval, F.; Poitau, G.; Gagnon, F. Hybrid Peak-to-Average Power Ratio Reduction Techniques: Review and Performance Comparison. *IEEE Access* **2017**, *5*, 27145–27161. [CrossRef]
17. Chen, C.Y. An Improved Approximation for Scheduling Malleable Tasks with Precedence Constraints via Iterative Method. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 1937–1946. [CrossRef]
18. Huang, K.; Zhang, X.; Zheng, D.; Yu, M.; Jiang, X.; Yan, X.; de Brisolara, L.B.; Jerraya, A.A. A Scalable and Adaptable ILP-Based Approach for Task Mapping on MPSoC Considering Load Balance and Communication Optimization. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2018**, *38*, 1744–1757. [CrossRef]
19. Zahaf, H.E.; Lipari, G.; Bertogna, M.; Boulet, P. The Parallel Multi-Mode Digraph Task Model for Energy-Aware Real-Time Heterogeneous Multi-Core Systems. *IEEE Trans. Comput.* **2019**, *68*, 1511–1524. [CrossRef]
20. Xiao, X.; Xie, G.; Li, R.; Li, K. Minimizing Schedule Length of Energy Consumption Constrained Parallel Applications on Heterogeneous Distributed Systems. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/I SPA, Tianjin, China, 23–26 August 2016.
21. Hu, F.; Quan, X.; Lu, C. A Schedule Method for Parallel Applications on Heterogeneous Distributed Systems with Energy Consumption Constraint. In Proceedings of the 3rd International Conference on Multimedia Systems and Signal Processing, Shenzhen, China, 28 April 2018; pp. 134–141.
22. Li, J.; Xie, G.; Li, K.; Tang, Z. Enhanced Parallel Application Scheduling Algorithm with Energy Consumption Constraint in Heterogeneous Distributed Systems. *J. Circuits Syst. Comput.* **2019**, *28*, 1950190. [CrossRef]
23. Quan, Z.; Wang, Z.J.; Ye, T.; Guo, S. Task Scheduling for Energy Consumption Constrained Parallel Applications on Heterogeneous Computing Systems. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *31*, 1165–1182. [CrossRef]
24. Ren, J.; Su, X.; Xie, G.; Yu, C.; Tan, G.; Wu, G. Workload-Aware Harmonic Partitioned Scheduling of Periodic Real-Time Tasks with Constrained Deadlines. In Proceedings of the 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, 2–6 June 2019.
25. Rapp, M.; Sagi, M.; Pathania, A.; Herkersdorf, A.; Henkel, J. Power- and Cache-Aware Task Mapping with Dynamic Power Budgeting for Many-Cores. *IEEE Trans. Comput.* **2019**, *69*, 1–13. [CrossRef]
26. Li, K. Power and performance management for parallel computations in clouds and data centers. *J. Comput. Syst. Ences* **2016**, *82*, 174–190. [CrossRef]
27. Ye, T.; Wang, Z.J.; Quan, Z.; Guo, S.; Li, K.; Li, K. ISAECC: An Improved Scheduling Approach for Energy Consumption Constrained Parallel Applications on Heterogeneous Distributed Systems. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018.
28. Xie, G.; Jiang, J.; Liu, Y.; Li, R.; Li, K. Minimizing Energy Consumption of Real-Time Parallel Applications using Downward and Upward Approaches on Heterogeneous Systems. *IEEE Trans. Ind. Inform.* **2017**, *13*, 1068–1078. [CrossRef]

29. Li, K. Performance Analysis of Power-Aware Task Scheduling Algorithms on Multiprocessor Computers with Dynamic Voltage and Speed. *IEEE Trans. Parallel Distrib. Syst.* **2008**, *19*, 1484–1497.
30. Li, K.; Tang, X.; Li, K. Energy-Efficient Stochastic Task Scheduling on Heterogeneous Computing Systems. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 2867–2876. [CrossRef]
31. Rusu, C. Maximizing rewards for real-time applications with energy constraints. *Acm Trans. Embed. Comput. Syst.* **2003**, *2*, 537–559. [CrossRef]
32. Tang, Z.; Qi, L.; Cheng, Z.; Li, K.; Khan, S.U.; Li, K. An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. *J. Grid Comput.* **2016**, *14*, 55–74. [CrossRef]
33. Huang, Q.; Su, S.; Li, J.; Xu, P.; Shuang, K.; Huang, X. Enhanced energy-efficient scheduling for parallel applications in cloud. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, Canada, 13–16 May 2012; pp. 781–786.
34. Meng, J.; Tan, H.; Li, X.Y.; Han, Z.; Li, B. Online Deadline-Aware Task Dispatching and Scheduling in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 1270–1286. [CrossRef]
35. He, Q.; Guan, N.; Guo, Z. Intra-Task Priority Assignment in Real-Time Scheduling of DAG Tasks on Multi-Cores. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2283–2295. [CrossRef]
36. Lin, X.; Wang, Y.; Xie, Q.; Pedram, M. Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment. *IEEE Trans. Serv. Comput.* **2015**, *8*, 175–186. [CrossRef]
37. Xie, G.; Chen, Y.; Liu, Y.; Wei, Y.; Li, R.; Li, K. Resource Consumption Cost Minimization of Reliable Parallel Applications on Heterogeneous Embedded Systems. *IEEE Trans. Ind. Inform.* **2017**, *13*, 1629–1640. [CrossRef]
38. Anselmi, J.; Doncel, J. Asymptotically Optimal Size-Interval Task Assignments. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2422–2433. [CrossRef]
39. Cheng, D.; Zhou, X.; Lama, P.; Ji, M.; Jiang, C. Energy Efficiency Aware Task Assignment with DVFS in Heterogeneous Hadoop Clusters. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *29*, 70–82. [CrossRef]
40. Canon, L.C.; Marchal, L.; Simon, B.; Vivien, F. Online Scheduling of Task Graphs on Heterogeneous Platforms. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *31*, 721–732. [CrossRef]