Journal of Systems Architecture 121 (2021) 102311

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Are task mappings with the highest frequency of servers so good? A case study on Heterogeneous Earliest Finish Time (HEFT) algorithm

Jie Liang [a,b,*], Kenli Li [a,b], Chubo Liu [a,b], Keqin Li [a,c]

[a] School of Information Science and Engineering, Hunan University, Changshang, 410082, China
[b] National Supercomputing Center in Changsha, 410082, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Task scheduling
Dynamic voltage/frequency scaling
Initial mapping frequency
Makespan
Heterogeneous computing

ABSTRACT

In the current heterogeneous computing environment, scheduling strategies play a key role in achieving high performance under current heterogeneous computing environment. Various algorithms have been proposed to generate high-quality schedules. Nonetheless, all the existing methods initialize task mapping with the highest frequencies of servers, creating significant effects on the results. Does mapping with the highest frequencies of servers generate the best schedule? In this paper, an attempt has been made to answer this question by investigating HEFT, which is a commonly used initial schedule method in scheduling the literature under DVFS environments. In this work, the task mapping is initialized with different server frequency level, along with investigating its impacts on the schedule results. Based on this concept, a corresponding Frequency Scaling Algorithm (FSA) has been proposed. By applying FSA combined with three frequency levels mappings, it has been observed that initial task mapping with middle server frequencies can generate much better results. When the initial frequency is the highest, the scheduling result is not optimal. The proposed algorithm requires less execution time and generates stable performance. Also, the proposed frequency scaling method is found to mitigate the impacts of initial mappings to a certain extent.

1. Introduction

1.1. Motivations

With the development of High Performance Computing (HPC) systems, HPC clusters integrate and offer tremendous computing units to deal with certain large-scale applications. A large number of calculation methods and optimization models have been proposed to serve the purpose. For example, a novel high-performance computing method for big data from the perspective of multi-attributes is discussed, followed by its improved version [1]. Also, an important corresponding multi-objective optimization model for CPSS big data has been highlighted [2]. To improve the parallel capabilities of high performance computing architectures, many architectures augmented with accelerators like Graphic Processing Units (GPUs) have also been proposed and have provided more computing resources for many applications. For instance, FlinkCL, an OpenCL-based heterogeneous CPU–GPU cluster memory computing architecture was proposed, enabling Flink to utilize the massive parallel processing capabilities of GPUs [3]. Based on this, GFlink was further proposed, and the original Flink was expanded from a CPU cluster to a heterogeneous CPU–GPU cluster [4], which significantly improved Flink's computing power. Computing architecture is constantly updated and developed, which has brought great computing power and parallel capabilities to HPC.

Under the HPC environment, for matching the computing resources and applications' demands with the objective of minimizing the cost, many frameworks and heuristics involving execution efficiency and energy consumption are proposed [5–8]. Specifically, several algorithms have been introduced for HPC, for example, Predict Earliest Finish Time (PEFT) algorithm, Heterogeneous Earliest Finish Time (HEFT) algorithm [9], Mapping Heuristic (MH) [10], Constrained Earliest Finish Time (CEFT) algorithm [11] , and Critical-Path-On-a-Processor (CPOP) algorithm [9]. Among these classic algorithms, Heterogeneous Earliest Finish Time (HEFT) has proved to be one of the most promising algorithms for makespan optimization.

Meanwhile, energy consumption is another critical issue in HPC besides makespan [12,13]. To achieve considerable performance under the energy limitation, several works leverage the Dynamic Voltage/Frequency Scaling (DVFS) technique, which can adjust the execution speed and corresponding energy consumption [14,15]. In all these works, it has been observed that their task mappings are initialized with the highest frequencies of servers. Nevertheless, this initial setting of

* Corresponding author at: School of Information Science and Engineering, Hunan University, Changshang, 410082, China.
E-mail address: lxj@hnu.edu.cn (J. Liang).

https://doi.org/10.1016/j.sysarc.2021.102311
Received 17 December 2020; Received in revised form 6 September 2021; Accepted 8 October 2021
Available online 23 October 2021

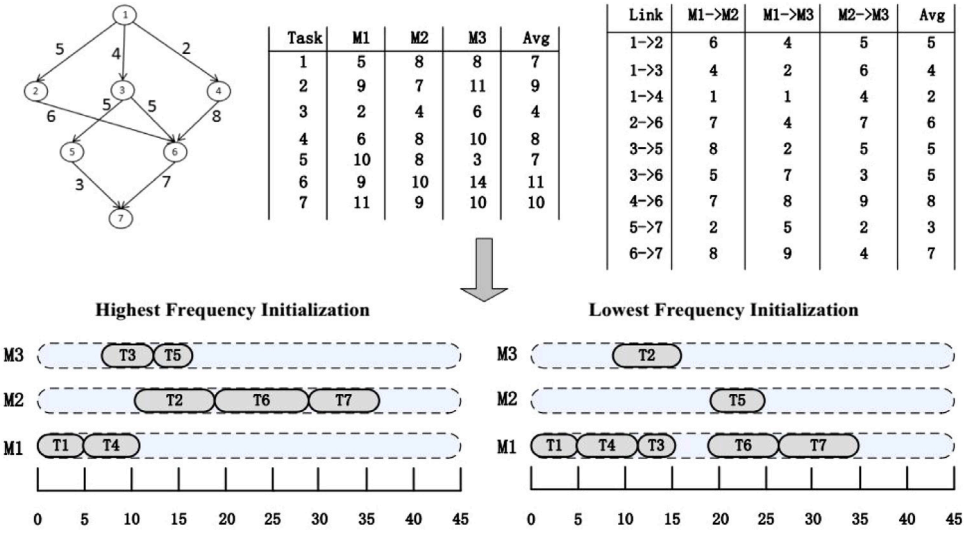


Fig. 1. Initial Scheduling Frequency Illustration.

Table 1
The parameters of processors for the frequency settings with their associated power consumption. The unit of frequency and power are MHz and W respectively.

Level	M1		M2		M3	
	F	P	F	P(W)	F	P
1	1000	7.2	1000	5.0	667	5.3
2	900	5.95	900	3.29	600	4.2
3	800	4.84	500	2.05	533	3.0
4	700	3.85	400	1.64	400	1.9
5	600	3	300	0.97	300	1.3
6	500	2.03	–	–	–	–

the mapping frequency seriously affects the final scheduling result, as it impacts the mapping relationship between the tasks and the hosts [16]. In Table 1, there is a set of the frequencies along with their corresponding energy consumption of processors for three heterogeneous machines. The high frequency and the low frequency correspond to the maximum and minimum frequency of the corresponding processor, respectively. The middle frequency refers to (the maximum+the minimum)/2. Since the processor frequencies are discrete over here, the middle frequencies for M2 and M3 are 500 and 533, respectively. For M1, the frequency of both 700 and 800 is calculated and compared.

Fig. 1 depicts an illustration with the impacts of initial mapping under the DVFS environment for minimizing the makespan. There are seven tasks in the Directed Acyclic Graph (DAG) needing to be deployed on the heterogeneous machines (M1–M3), which are described in Table 1. The execution overhead and data transfer cost are highlighted in Fig. 1, which is with the highest frequency for the DAG application. However, different initialization frequencies result in different mapping results between the task and the processor. Firstly, the task selects a processor based on the initial frequency. With different execution frequencies, each processor has a different execution overhead for each task. It can be noticed that the processors selected by task 3 differ in the two schemes with different initial mappings, and the corresponding final mapping results are different, as illustrated in Fig. 1.

After completing the initial mapping with the initial frequency, the frequency of each processor is adjusted and scaled by DVFS technology to minimize the schedule length. It can be observed that an initialization with low frequency causes higher execution overhead at first, and then the execution time is reduced by stepwise frequency scaling for the processor where each task is located. This process is not the same as adjusting the execution frequency of each task to the highest. Also, the optimal result might not be obtained with the highest initial frequency mapping. The result illustrated in Fig. 1 shows that the lowest initial frequency mapping outperforms the highest initial frequency mapping on the scheduling result.

In addition to the execution overhead, the calculation overhead includes the communication cost. Different initial frequencies lead to different mapping relationships between tasks and processors. The communication overhead changes when the mapping relationship changes with the initial frequency.

This proves that the efficient selection of initial scheduling frequency and the adjustment of frequency is important to generate better task schedules under HPC environments. These two factors are exactly what this article studies.

1.2. Related work

Though DAG scheduling is NP-hard, it has been widely studied and many heuristics are proposed for various considerations. Among them, makespan and energy efficiency are two of the most crucial metrics in various applications [17–21]. Although this type of problem is classic, it is still challenging, even considering the parallelism in the constantly updated high-performance computing architecture.

For makespan minimization, many algorithms have been framed and proposed [22,23], such as list scheduling algorithms, clustering algorithms, and duplication-based algorithms. Among them, the principal method used in list scheduling is determining the task's priority and selecting an appropriate processor for each task. Owing to its simplicity and outstanding performance, HEFT is one of the most frequently used list scheduling algorithms.

For energy saving, DVFS is a promising energy reduction technique that has been used in many approaches [24–27]. DVFS technique can reduce the energy consumption by allocating appropriate execution frequencies to each task, i.e., reducing processor frequency at certain times. The majority of DVFS-enabled heuristics are conducted on both homogeneous [28] and heterogeneous [26,29,30] computing systems. In this paper, only heterogeneous processors with DVFS are studied, regardless of GPUS or FPGAs.

Both makespan and energy saving are simultaneously studied and many approaches are proposed for the same [18,20,31]. These algorithms usually comprise two steps. The first step is to complete the mapping the second step is DVFS scaling. The former is used to mapping the tasks to processors, and the latter is used to optimize some objectives. In the initial task mapping phase, besides HEFT algorithm above [32–34], many methods are used, such as ETF algorithm [35]

Table 2
Comparisons of how existing DAG task scheduling for DVFS-based heuristic cover target and design environment. Also, the specific benefits of FSA for each deployment type have been enlisted.

Target	Energy consumption under time constraints [33–36,39–42] ⊘①				Energy consumption and completion time [26,29,30] ⊘①
Type of resources	Multiple homogeneous processors [35,36,39,40,42] ⊘②		Multiple heterogeneous processors [33,34,41]		Multiple heterogeneous processors
Initial mapping method	ETF algorithm [35,39] ⊘③	HEFT algorithm [33,34]	EDF algorithm [36] ⊘③	Other list scheduling [40–42] ⊘③	Not involved
Initial mapping frequency	Highest frequency [33–35,39] ⊘④		Highest voltage value [36,40,41] ⊘④		Not involved
CPU capacity	Discrete [33–35,39,41,42]		Continuous [36,40] ⊘⑤		Discrete
Main design	Slack time reclamation [33,35,39–42] ⊘⑥		Linear programming [34], ⊘⑥ Integer programming [36] ⊘⑥		Energy conscious scheduling heuristic(ECS)
	Benefits of the Frequency Scaling algorithm with different initial mapping frequency				
Target	**Makespan under energy constraints ①**				
Type of resources	**Multiple heterogeneous processors ②**				
Initial mapping method	**HEFT algorithm ③**				
Initial mapping frequency	**Different mapping frequency, from the lowest to highest ④**				
CPU capacity	**d-DVFS ⑤**				
Main design	**Path-based frequency scaling ⑥**				

and EDF algorithm [36]. DVFS technique is chiefly involved in the second phase [15,33–35,37,38].

Table 2 summarizes the existing related approaches for DAG scheduling in reference to the proposed research work. In this work, a frequency scaling algorithm with different initial mapping frequencies has been adopted. There are six corresponding characteristics or benefits (① ② ③ ④ ⑤ ⑥) of this method displayed on the bottom part of the table. Some are crossed out with slashes (⊘① ⊘② ⊘③ ⊘④ ⊘⑤ ⊘⑥)when the corresponding characteristics or benefits do not exist in the literature. Through such a table, the main work of this paper can be easily understood, making its difference from the other related works. As depicted in Table 2, no matter what (makespan or energy consumption) the work is designed for, if their approaches involve a task mapping phase, all the methods select the highest frequency of the server instead of performing initial mapping based on different mapping frequencies from the lowest to the highest frequencies. However, the initial mapping frequency has significant impacts on the matches between the tasks, the processors, and the scheduling result (see example in Fig. 1), which are ignored by the existing works. Hence, the impacts of different initial mapping frequencies have been investigated in this work. Specifically, different initial mapping frequency levels are adopted to explore the mapping relationship between tasks and hosts. In Table 2, it can be observed that some of the most suitable initial scheduling algorithms are ETF and EDF algorithms. Since HEFT is frequently used in the task mapping phase owing to its simplicity and good performance, it has been chosen here as a representative for the first step. Combined with the characteristics of this method and results obtained based on this phase, a path-based Frequency Scaling Algorithm (FSA) has been proposed for the second phase, i.e., frequency scaling. This is a key innovation that distinguishes this work from the other related works. Many other differences between the proposed method and those of the others are summarized in Table 2.

1.3. Our contributions

In this work, the goal is to study and evaluate the impact of initialization frequency on the DAG mapping results under the DVFS environment. The main contributions are listed below:

- For each execution, three levels of initial mapping frequency are considered instead of the highest value. The mapping relationship between the tasks and the hosts with different initial frequencies is explored. This work may make some supplements in this field.
- Based on the initial mapping, a Frequency Scaling Algorithm (FSA) has been proposed for reducing the makespan.
- Extensive numerical experiments are conducted, and the results show that mappings with middle-frequency levels are better than the other two initial mappings. Besides, it can significantly reduce the schedule length compared to other approaches.
- The parameters that affect the performance are also analyzed. The results illustrate that FSA can mitigate the impacts of initial mappings to a large extent.

1.4. Organization

The article is divided into five sections. Section 2 elaborates the system model and problem definition. Further, the initial mapping algorithm has been introduced with different frequency levels, followed by proposing the frequency scaling algorithm for heterogeneous systems in Section 3. The experimental results and related analysis and discussion are presented in Section 4. The last section of the article summarizes the entire work and gives an outlook for future work.

2. Models and problem formulation

This paper highlights the effect of initial mapping at different frequency levels. It involves two target parameters: makespan and energy consumption. The application, heterogeneous system, makespan, and energy models are described as follows (see Table 5).

2.1. Application model

The computing application includes multiple tasks, and there exist priority constraint relationships among them. Directed Acyclic Graph (DAG) is applied to describe such applications, which is a two-tuple $\langle \mathcal{T}, E \rangle$. $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ represents the task set and E indicates the set of edges among the tasks. An edge $(i, j) \in E$ signifies that t_j is a successor of t_i. There are two special tasks in DAG: exit task and entry task. The entry task has no predecessor, and the existing task has no successor. The calculation overhead of the task is denoted by weighting parameter w_i. The path with the greatest overhead (length) from the entry task to the exit task is known as the Critical Path (CP).

Table 3
Notation.

Notation	Description
$\mathcal{T}$	A set of tasks
edge (i, j)	t_j is a successor of t_i
$C_{i,j}$	Communication overhead between t_i and t_j
p	The number of heterogeneous processors
$\{pe_1, \ldots, pe_k, \ldots, pe_p\}$	A set of heterogeneous processors
$\{v_{k_1}, \ldots, v_{k,m_k}\}$	A set of executable supply voltages for pe_k
$\{f_{k_1}, \ldots, f_{k,m_k}\}$	A set of supply frequencies for pe_k
m_k	Number of voltage/frequency levels for pe_k
ϕ_i	The particular processor which execute task t_i
l	The level of the frequency in the particular processor
$f_{\phi_i,l}$	The frequency of processor ϕ_i to execute t_i
$Ppred(t_i)$	A set of direct adjacent predecessors of t_i
$pred(t_i)$	A set of predecessors executed on different processes of t_i
$W_i(f_{\phi_i,l})$	The execution overhead of t_i executed on processor ϕ_i

The path length is the sum of the weights of the edges along the path. Another weighting parameter $C_{i,j}$ indicates the communication overhead between t_i (executed on one processor) and t_j (executed on another processor). Tasks executed on different processors will produce communication overhead.

2.2. Computing system model

There exist p heterogeneous processors in the computing system, denoted by $PE = \{pe_1, \ldots, pe_k, \ldots, pe_p\}$. These processors are connected with each other through a network. In this heterogeneous model, each pe_k ($pe_k \in PE$) comprises a set of executable supply voltages $v_k = \{v_{k_1}, \ldots, v_{k,m_k}\}$ and a set of supply frequencies $F_k = \{f_{k_1}, \ldots, f_{k,m_k}\}$. m_k represents the number of voltage/frequency levels for pe_k. $L_{k,level} = \{1, \ldots, m_k\}$ indicates the voltage/frequency levels for pe_k. Each processor can perform tasks at different speeds with different frequencies by regulation.

The power is modeled as [43]:

$$P = P_s + \hbar(P_{ind} + P_d), \tag{1}$$

There are three stages of power consumption: static power consumption, frequency-independent active, and frequency-dependent dynamic power, which are represented by P_s, P_{ind} and P_d respectively. $\hbar$ indicates the occurrence of active power consumption ($\hbar = 1$ for active and $\hbar = 0$ for sleep mode). P_d is related to the voltage and frequency of the system. In a CMOS-based processor, dynamic power consumption is the primary part. It can be expressed as:

$$P_d = C_{eff} V_{dd}^2 f, \tag{2}$$

where C_{eff} denotes the switch capacitance, V_{dd} represents the supply voltage, and f is the frequency. The relationship between voltage and operating frequency is defined by the following equation [44]:

$$f = \frac{(V_{dd} - V_{th})^{\alpha}}{K L_d}, \tag{3}$$

or equivalently as:

$$V_{dd} = V_{th} + (K L_d f)^{\frac{1}{\alpha}}, \tag{4}$$

where K and α are the two constant values, V_{th} denotes the threshold voltage, and L_d represents the circuit logic depth. The frequency can be scaled in the interval $[0, f_{max}]$, while the value of voltage varies in the range $[V_{th}, V_{max}]$. During the process of frequency-changing, the execution clock cycles of the task remain unchanged, but the execution time of the task may vary. Let ϕ_i denote the particular processor, which executes the task t_i. Let l represent the level of the frequency in the processor. Thus, $f_{\phi_i,l}$ represents the frequency of processor ϕ_i to execute t_i.

The computation overhead of t_i can be acquired by the following equation:

$$Time_{t_i} = w_i / f_{\phi_i,l}. \tag{5}$$

Thus, the energy consumption can be computed as follows:

$$E_{t_i} = P_d \times Time_{t_i} = C_{eff} V_{dd}^2 f_{\phi_i,l} \times C_i / f_{\phi_i,l} = C_{eff} C_i V_{dd}^2. \tag{6}$$

From the above equation, it can be observed that the energy consumed is proportional to the square of the voltage ($E \propto V_{dd}^2$). By slightly increasing or decreasing the voltage, a great impact on the energy consumption can be observed.

2.3. Performance measurement

$W_{i,k} = \frac{w_i}{f_k^{\max}}$ is denoted as the time overhead for executing on processor pe_k with its highest frequency. The average execution cost is calculated as:

$$\overline{W_i} = \sum_{k=1}^{p} \frac{W_{i,k}}{p}. \tag{7}$$

The average communication overhead can be determined by the following equation:

$$\overline{C_{i,j}} = \frac{data_{i,j}}{\overline{B}}, \tag{8}$$

where $\overline{B}$ indicates the average communication bandwidth.

In this paper, the scheduling is static (executed during compilation). The schedule length (*makespan*) is selected as the main parameter for performance evaluation. The scheduling length of DAG is determined by the exit task, and the completion time of a task is determined by the time when the task starts to execute and its execution time. The Earliest Start Time (EST) is determined by the predecessors' Earliest Finish Times (EFTs). These variables are calculated recursively from the previous entries [9]. Specifically, for the task t_{entry},

$$EST_{t_{entry}} = 0. \tag{9}$$

EST of other tasks in the graph is expressed as:

$$EST(t_i, f_{\phi_i,l}) = \max \left\{ EST_{Ppred(t_i)} + W_{Ppred(t_i)}, \max_{t_j \in pred(t_i)} \left(EST_j + W_j(f_{\phi_j,l}) + \frac{data_{i,j}}{B_{\phi_i,\phi_j}} \right) \right\}, \tag{10}$$

where $Ppred(t_i)$ indicates a set of direct adjacent predecessors of t_i, which are executed on the same process, and $pred(t_i)$ represents a set of predecessors executed on different processes. In this case, communication is required. In the above EFT equation, the first term signifies the earliest time that the processor is ready for a task, while the second term denotes the ready time.

Accordingly, EFT can be expressed as:

$$EFT(t_i, f_{\phi_i,l}) = EST(t_i, f_{\phi_i,l}) + W_i(f_{\phi_i,l}), \tag{11}$$

where $W_i(f_{\phi_i,l})$ represents the execution overhead of t_i executed on the processor ϕ_i. These variables are calculated recursively from the top to the bottom of DAG. When all tasks are mapped to processors, the Actual Finish Time (AFT) of the exit task is defined as *makespan*, which is represented as follows:

$$makespan = \max\{AFT(t_{exit})\}. \tag{12}$$

2.4. Problem formulation

The goal of this paper is to minimize the *makespan* under the energy supply constraints, particularly considering a DAG application and a heterogeneous cluster which supports the DVFS technique. The aim is to distribute the tasks to processors so that its *makespan* can be minimized while ensuring that the power consumption is not greater than a given value E^*.

$$\text{minimize} \quad makespan = \max\{AFT(t_{exit})\}, \tag{13}$$

$$\text{s.t.} \quad E_{total} \le E^*, \tag{14}$$

where E_{total} represents the total power consumption, including all the servers' consumption, whether or not performing the tasks.

3. HEFT mapping with different frequency levels and a frequency scaling algorithm

As mentioned earlier, approaches for DAG scheduling under DVFS environments primarily include two phases: task mapping and later frequency scaling. In this paper, the mapping relationship between the tasks and hosts and the communication time among different processors with different initial frequency mappings are explored. Three kinds of initial mapping frequencies with the HEFT algorithm are considered. Based on the initial mappings, a Frequency Scaling Algorithm (FSA) has been proposed, which is a Critical Path (CP) based algorithm, combining the DVFS technique to further optimize the makespan.

3.1. HEFT mapping

HEFT is one of the most well-known algorithms in scheduling literature. It is frequently adopted to generate the initial schedule for the time-oriented optimization in the first phase in some DVFS-based heuristics. Meanwhile, many proposed algorithms are also developed based on HEFT mapping. However, to the best of the authors′ knowledge, in most previous heuristics that have adopted the HEFT algorithm as an initial mapping method, the highest frequencies of the server are often used for mapping.

In this paper, the frequencies of servers are classified into three levels.

- **Low.** The mapping frequency of each server is set as its lowest frequency level;
- **Middle.** The mapping frequency of each server is set as its middle frequency level ((the lowest+the highest)/2);
- **High.** The mapping frequency of each server is set as its highest frequency level.

Compared with the original HEFT algorithm [9], the selection of the initial mapping frequency and the steps of the algorithm are outlined in Algorithm 1.

Algorithm 1 HEFT Mapping under Different Frequency Levels

```
while (each of the three available frequencies in the heterogeneous system) do
    Select one available frequency f ∈ F_l.
    Assign the weights of the each node and edge in the DAG task graph adopting
    arithmetic average method under the specific frequency respectively.
    Calculate upward rank value of all tasks according to the weight of tasks and edges.
    Generate the task scheduling list.
    while (there exist tasks that need to be scheduled) do
        Pick the first task t_i in the task scheduling list according to decreasing rank value.
        for (pe_k) do
            Calculate EFT(t_i, f_{pe_k,l}).
            Distribute t_i to the best processor φ_i to minimize EFT of t_i.
        end for
    end while
end while
```

Weight Assigning Phase. The weights of each node and edge are assigned on the basis of the predicted execution cost of tasks and communication time between the two adjacent nodes. In heterogeneous environments, different machine resources lead to different reference computing times for each task. Different data links also result in different communication times. In this paper, the weights in the DAG adopt the arithmetic average of different processors calculated by the maximum frequencies. As demonstrated in Fig. 1, the initial computational overhead of each task and the average transfer bandwidth among the processors is calculated using the maximum frequencies. The task execution time is affected by the frequency of the server. In this paper, three initial frequencies are explored with the HEFT method to investigate its effect on different scheduling results.

Task-prioritizing Phase. The main operation of this phase is to prioritize the tasks by calculating tasks' rank value through traversing the entire DAG from the bottom of DAG in reverse.

$$rank_u(t_i) = \overline{\omega_{i,l}} + \max_{t_j \in succ(t_i)}(\overline{c_{i,j}} + rank_u(t_j)), \tag{15}$$

where $\overline{\omega_{i,l}}$ represents the average execution overhead of t_i with frequency $f_{\phi_i,l}$, $succ(t_i)$ represents the direct successors of t_i, and $\overline{c_{i,j}}$ represents the average communication overhead. The upward rank value represents the length of the critical path. The target in this step is to first schedule tasks with the highest rank value. Based on the descending order of rank, a scheduling list is generated and a topological order of the constrained tasks is formed. In this phase, a random selection strategy is adopted in the tie-breaking process to avoid increasing the time complexity, i.e., if two nodes have the same rank value, one of them is randomly selected for scheduling.

Processor Selection Phase. The task is scheduled according to the decreasing upward rank value in the above phase. The principle of scheduling at each step of the above process is to minimize the tasks' EFT. The principle is to minimizes the task's EFT at every step of scheduling.

As an illustration, the execution overhead and data transfer cost are mentioned in Fig. 1. This sample involves three heterogeneous resources M1, M2, and M3. The corresponding rank value is computed according to the calculated weights and transfer overhead, and the result is as follows: 55(T1), 43(T2), 37(T3), 44(T4), 20(T5), 28(T6), 10(T7). Based on the upward rank values, the scheduling order can be obtained by the HEFT algorithm, i.e., $\{T_1, T_4, T_2, T_3, T_6, T_5, T_7\}$. In this order, the tasks are further scheduled onto different processors. Finally, the initialization schedules results with the maximum frequencies for this sample are highlighted in Fig. 1. In this case, the frequency cannot be further scaled up. The initial schedule result is the same as the final schedule results calculated by the proposed scaling method. The schedule length is 37, which is longer than the value (35) that is calculated with the lowest frequency initialization by FSA.

3.2. The frequency scaling algorithm

The Frequency Scaling Algorithm (FSA) has been proposed with different initial frequency levels in this article. The initial phase and the second frequency scaling phase are mutually influential with a mutual connection relationship and not independent of each other in this algorithm. In the second phase, frequency scaling has been invoked, which leverages the DVFS technique on HEFT mappings using Low, Middle, and High frequency levels.

The idea of FSA is somewhat motivated by a natural intuition. In DAG, the shortest time to accomplish the scheduling is the length of the longest path from the entry task to the exit task. As mentioned before, the path with the longest path length is identified as the critical path. It can be inferred from Fig. 2 that, after initial scheduling of the DAG application with the lowest frequency level, *makespan* is determined by the critical path and T_7 is the last task of the critical path. The execution time of the previous tasks in the critical path is exactly the time when

T7 starts execution. Thus, *makespan* is determined by the EST and ET of T_7. Hence, the execution time or start time of T_7 can be scaled to reduce the *makespan*. Notice that T_6 is the predecessor of T_7 in the critical path in this DAG. The same can be obtained that the execution start time of T_7 is determined by T_6. Hence, T_6 can also be scaled to impact the *makespan*. Similarly, the execution start time of T_6 is determined by T_4, where T_4 is dominated by its predecessor T_1, and T_1 is the entry task. From this, we find the path $T_7 \rightarrow T_6 \rightarrow T_4 \rightarrow T_1$, in which the tasks can be scaled to impact the *makespan*. The critical path, which determines *makespan* in the DAG, is depicted, and then the task from the bottom to the top of the path is chosen to adjust the execution frequency. At each iteration, the frequency is adjusted one level up to further reduce the *makespan*.

Algorithm 2 Frequency Scaling Algorithm (FSA)

1: **while** (***true***) **do**
2: Set $len \leftarrow 0$ and $t \leftarrow t_{exit}$
3: **while** (t *is not an empty task*) **do**
4: Put t into the path, i.e., set $Path[len++] \leftarrow t$
5: Find the task $t' \in pred(t)$ which dominates the beginning of t for execution, i.e., $t' \leftarrow \arg\max_{\hat{t} \in pred(t)}(AFT_{\hat{t}} + C_{\hat{t},t})$
6: Set $t \leftarrow t'$;
7: **end while**
8: Record **AFT**, i.e., set $\mathbf{AFT}^{\text{old}} \leftarrow \mathbf{AFT}$
9: **for** (*variable* l *from 0 to* $len-1$) **do**
10: Set $t \leftarrow Path[l]$ and $f_{\phi_t,l_t} \leftarrow f_{\phi_t,l_t+1}$
11: **if** ($l_t + 1 > L_{\phi_t}$) **then**
12: **continue**;
13: **end if**
14: Build a stack S and put task t into S, i.e., $S.push(t)$
15: **while** (S *is not empty*) **do**
16: Set $t' \leftarrow S.pop()$ and calculate $AFT_{t'}$
17: **if** ($AFT_{t'}$ *is not equal to* $AFT^{\text{old}}_{t'}$) **then**
18: **for** (*each succeed task* $\hat{t} \in succd(t')$) **do**
19: Add task $\hat{t}$ into S, i.e., $S.push(\hat{t})$
20: **end for**
21: **end if**
22: **end while**
23: Compute decreased makespan $\Delta makespan_t \leftarrow AFT_{t_{\text{exit}}} - AFT^{\text{old}}_{t_{\text{exit}}}$ and the increased power consumption ΔE_t
24: **end for**
25: Choose the task $t \in Path$ such that $\frac{\Delta makespan_t}{\Delta E_t}$ is minimized
26: **if** (t is an empty task) **then**
27: **break**;
28: **end if**
29: **end while**

For each task in the path, the increased energy ΔE and the corresponding decreased makespan $\Delta makespan$ due to execution frequency adjustment operation can be computed. The task with maximal $\frac{\Delta makespan}{\Delta E}$ to scale frequency has been chosen because the target of this work is to optimize the makespan under energy constraints, namely minimum execution time under the premise of ensuring energy consumption. After accomplishing the scaling operation in a path, the critical path is re-selected, and the above process continues to iterate.

Fig. 2 represents an illustration of the scaling process of FSA. As displayed in Fig. 2, after initial mapping with the lowest frequency, the schedule length is 82.46. At the first iteration, FSA configures the path $T_7 \rightarrow T_6 \rightarrow T_2 \rightarrow T_1$ and determines that scaling T_7 maximizes the value of $\frac{\Delta makespan}{\Delta E}$. Later, it scales the execution frequency of T_7 one level upwards. At the second iteration, the dominated path is still $T_7 \rightarrow T_6 \rightarrow T_2 \rightarrow T_1$ and the chosen task is T_6. Further, it scales its execution frequency one level upwards. At the third iteration, it learns that the dominated path becomes $T_7 \rightarrow T_5 \rightarrow T_3 \rightarrow T_1$ and selects task T_5 to scale one level upwards. The algorithm terminates when there is no task to scale its execution frequency. At the sixth iteration, task T_1 is selected to perform the scaling operation, and the iteration is terminated at this moment with a schedule length of 35.

The entire concept is formulated in Algorithm 2. At each iteration, the path which dominates the *makespan* is identified (Steps 3–8). Specifically, a queue *Path* is used to record tasks in the path. At first, t_{exit} is added to the queue, followed by finding the task among its predecessors, which dominates the start time of t_{exit}. Later, the dominated predecessor is chosen, and similar actions are repeated till the entry task. For each task in the path, its increased power consumption and decreased makespan are calculated (Steps 10–23). Then, the task with maximal $\frac{\Delta makespan}{\Delta E}$ is chosen to scale its frequency level (Step 24). The algorithm continues until there does not exist any such task (Step 25). In the proposed algorithm, the focus is to further minimize the scheduling length with a constraint of energy, based on the initial time-oriented optimization result obtained by HEFT with different frequency levels.

Table 4
Experiment parameters setting.

Experiment parameters	(Fixed)–{Varied range}
Number of tasks	(100)–{50,100,150,200,250,300,400,500}
Number of DAGs	(100)–{50,100,200,300,400}
Number of processors	(16)–{4,8,16,32,64}
CCR values	(1)– {0.1,0.2,0.5,1.0,2.0,5.0,10.0}
Parallelism factor	(1)–{0.5,1.0,2.0,5.0,10.0}
Initialization frequency	{low,middle,high}

4. Experimental evaluation

The evaluation of the performance under three heterogeneous processors is considered in this section. Randomly generated and real-world applications are evaluated. The comparisons of different algorithms are mentioned, and quantitative results and qualitative analyses are presented. The results are based on the two metrics, i.e., schedule length and Schedule Length Ratio (SLR). The former is the key measurement parameter of performance evaluation for the proposed algorithm because the target of this work is to minimize the *makespan*. SLR is used to normalize the schedule length when some task graphs have different characteristics. It is defined by:

$$SLR = \frac{makespan}{\sum_{t_i \in CP} \min_{pe_j \in PE}\{W_{i,j}\}}. \tag{16}$$

For any algorithm, the SLR of a graph will be greater than or equal to one. For the scheduling algorithm, the smaller the SLR, the better the performance. In this paper, since multiple frequency levels are involved, high frequency for each processor is set for the critical path calculation to ensure that the SLR is always greater than or equal to one.

4.1. Processor heterogeneity models

Table 1 represents a set of simultaneous changes of frequency and voltages, along with corresponding energy consumption [25].

In the experiments, 100 random task graphs generated by different parameters are tested. These task sets are carried out in heterogeneous processor environments. For each configuration, each processor can perform tasks with different frequencies. For each processor, available frequency and voltage follow a random yet uniform distribution. Some variables adopted in the experiment are enlisted in Table 4. The same number of tasks on seven sets of processors of 2^n (n ranges from 1 to 7) is evaluated for each graph.

4.2. Randomly generated application graphs

Generating random DAGs involves different parameters such as the ratio of communication to computation time and parallelism factor. These parameters significantly affect the dependence between tasks and communication costs. In the experiments, a large number of randomly generated graphs are used for simulation.

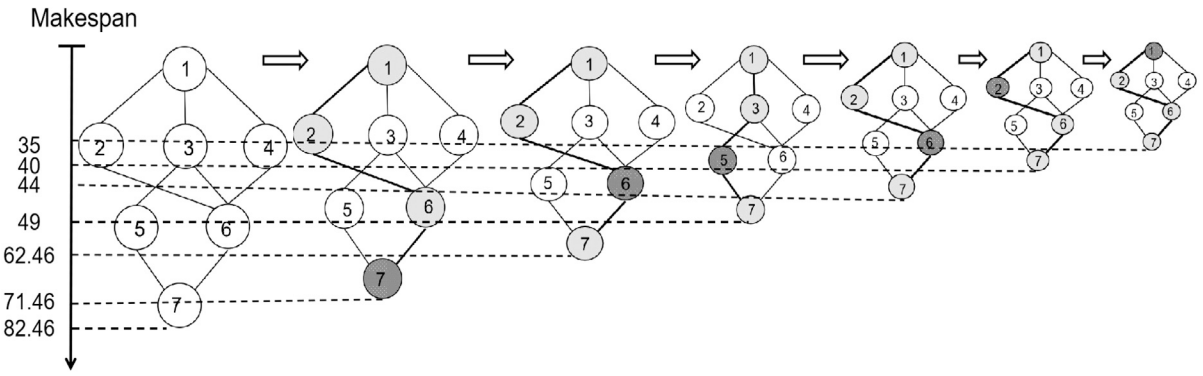


Fig. 2. An intuitive illustration and evolution for FSA of execution time (Makespan): The latest critical path is found in real-time, and the corresponding execution frequency is scaled.

4.2.1. Random application graphs

To build different random graphs, the following different fundamental input parameters are required:

- DAG size n : number of nodes.
- Communication to computation time ratio CCR : average communication overhead divided by the average computation overhead.
- Parallelism factor λ : the height and the width of a graph is generated randomly from a uniform distribution with a mean value of $\sqrt{n/\lambda}$ and $\lambda\sqrt{n}$ respectively, and rounded up to the nearest integer.

In the random application experiments, the combination of the above parameters is recommended for generating the graphs. The value of n ranges from 20 to 500. Based on the value of n, the height of DAG is formed for different λ (0.2, 0.5, 1.0, 2.0, 5.0, 10.0), and then the width of this DAG is computed at each level. Moreover, each task is assigned with the calculation overhead, and each edge is assigned with the communication overhead. The selection of DAG's communication and computational costs follows a uniform distribution. The mean of the communication cost depends on the CCR as well as the calculation overhead. In this paper, 100 random application graphs are tested. These graphs are generated using the above parameters. In each output, the average value of these 100 test results is obtained as the output of the experimental results.

4.2.2. Random application performance results

In this paper, different graph characteristics and input parameters are used to evaluate and compare the performances of the algorithms. The results with three different initialization frequencies and different CCR values are compared with each other (refer to Fig. 3). Initialization frequencies are the lowest, the highest, and the middle frequencies ((the lowest+the highest)/2), respectively. The purpose of these experiments is to evaluate the influence of different initialization frequencies on performance.

The data highlighted in the simulation results of Fig. 3 are from the average of 100 sets of random experiments. In a few of the subfigures of Fig. 3, it can be observed that initialization with different frequencies leads to different final optimization results. As mentioned in Section 3, the initialization frequency determines the processor to which each task will be assigned. Each task is thus scheduled to a different processor at different initialization frequencies. When the number of tasks is constant in HEFT, the makespan results are proportional to the frequency. The higher the frequency, the lesser the execution time and the smaller the makespan are. Therefore, makespan is minimal at the highest frequency for HEFT. However, the optimized makespan results for FSA are not related to the initialization frequency. The minimum makespan does not appear at the highest frequency. As illustrated in the figure, when initializing with the middle frequency, FSA leads to the smallest makespan. On the contrary, its makespan is the largest for the highest frequency. There are communication overheads in addition to the computational overhead. With initial mapping at the highest frequency, the calculation overhead is definitely the smallest, but the corresponding communication overhead is not the case. When initial mapping is performed at the middle frequency or the lowest frequency, the mapping results and the communication overhead are different. Considering the calculation cost and the communication cost comprehensively, the results prove that the experimental results obtained by using middle frequency for initialization are the best.

It can be noticed that with an increase in the number of tasks, FSA outperforms the HEFT significantly at low and middle frequencies in Fig. 3. The presented algorithm is the same as HEFT in cases when the initialization frequency is the highest. This is because, during the frequency scaling in this algorithm, high frequency is selected to reduce the makespan, but at this time, the initial frequency is already the highest; thus, there is no higher frequency to use.

The gaps between the two algorithms lessen marginally with the increase of CCR. Changes in CCR have no particular impact on the experimental results. However, the results of makespan for HEFT differ greatly when initialization mappings are between middle and low frequencies, while the results of makespan for FSA are almost the same. The results distinctly prove that this algorithm can reduce the impact of initialization mappings with different frequencies. For FSA, the difference between the two sets of makespan results is minimal. This difference comes from the initialization mapping topology formed by the initialization mapping with different frequencies.

The three figures in the first row of Fig. 3 reveal the performance of the two algorithms for various number of processors with various CCR. The makespan of the HEFT and FSA algorithms decreases with an increase in the number of processors. The results for a diverse number of processors are consistent with the results of a different numbers of tasks. The makespan is minimized when initializing with the middle frequency, and it is not optimized at the highest frequency.

Different effects of various initializing frequencies are considered on the optimization results in the first set of simulations. From the above experiments, it can be seen that the optimization is the best with middle frequency. In the next sets of experiments, the SLR of the algorithms with the middle initial frequency is compared with various graph characteristics, which are represented in Fig. 3 and Table 4.

In Fig. 4(a), the SLRs of algorithms are compared with diverse graph sizes. The proposed FSA outperforms the HEFT algorithm. From Table 5, it can be noticed that the growth in the number of tasks has a negligible effect on makespan with 32 processors. To make the trend of performance change become more obvious with the increase in the size of graphics, eight processors are used in another set of experiments. The corresponding results are listed in Table 6. The performance of FSA is better than the HEFT algorithm. The performance trends of the two algorithms differ with an increase in the number of tasks.

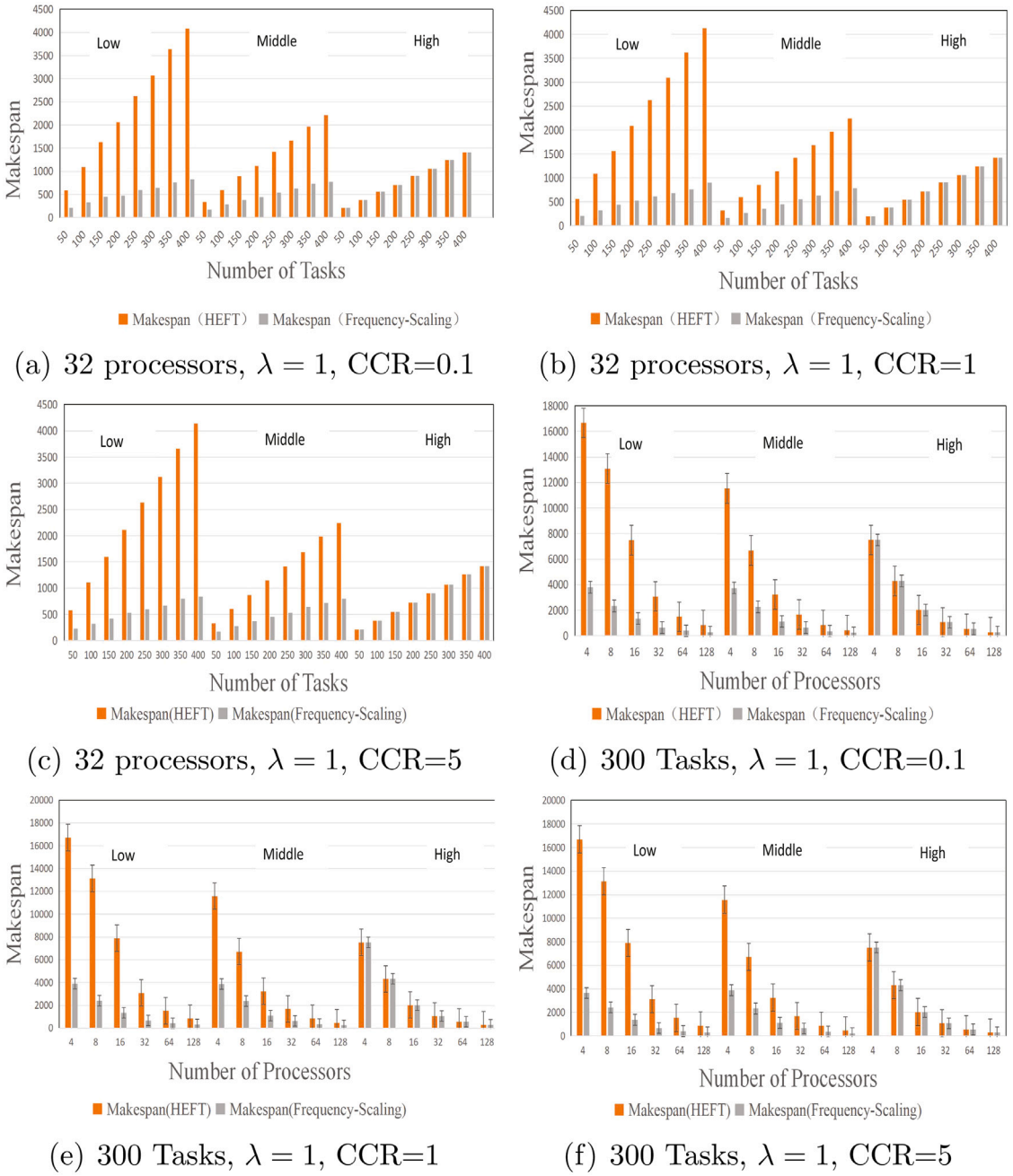


Fig. 3. Execution time (Makespan) of HEFT and Frequency Scaling Algorithms under different initialization frequencies with different CCR.

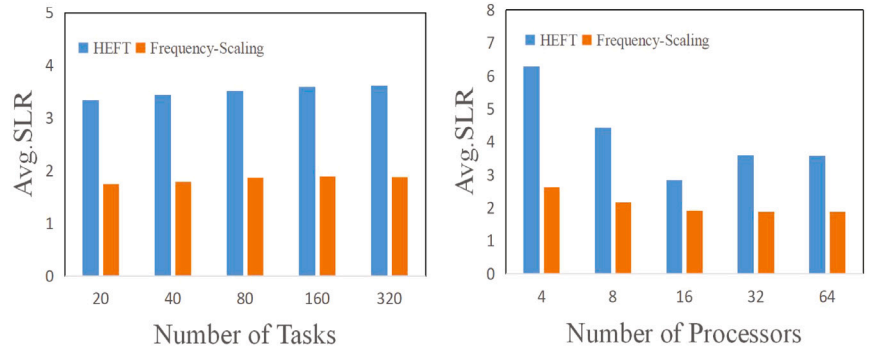
In Fig. 4(b), the SLRs of different algorithms are compared. The average SLR of FSA compared to HEFT is 240%, 204%, 149%, 190%, and 189%, when the number of processors increases by 2^n (n ranges from 2 to 6).

The comparisons for different CCR are illustrated in Fig. 4(c). The gaps between the two algorithms reduce gradually with the increase in CCR. Compared to higher CCR, a lower CCR can bring better experimental results for FSA. From Table 5, it can be seen that the best performance occurs when the value of CCR is 0.2. This phenomenon may be caused by the following reasons: When the CCR is low, the cost of the computation part dominates the whole application. When the CCR is high, the applications are prone to be communication-intensive. Therefore, the proposed algorithm proves to be more suitable for computation-intensive tasks.

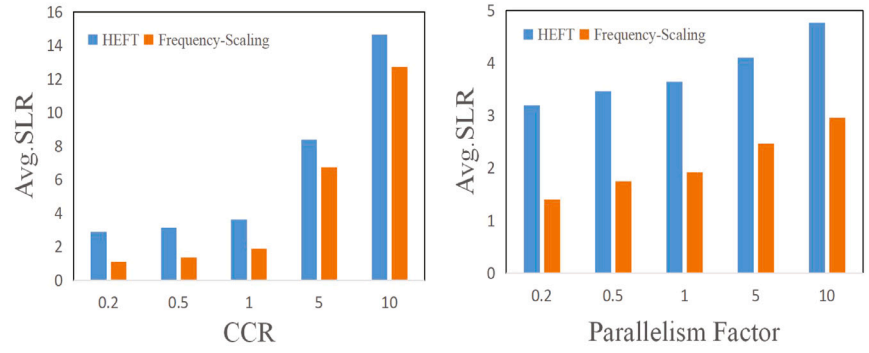
The next experiment is about the effect of the graph structure on the experimental results (see Fig. 4(d)). When the value of λ is 0.2, the performance of FSA is 227 percent better than the HEFT algorithm. When λ is equal to 0.5, 1.0, 5.0, and 10.0, the performance of the FSA exceeds that of HEFT by 198%, 190%, 166%, and 161%, respectively (see Table 7).

4.3. Real-world application graphs

In this paper, real-world application graphs are also added in addition to the random graphs. Two real-world graphs are simulated to test the performance: the Gaussian elimination [45] and the Fast Fourier Transformation (FFT) [46]. In these two applications, the number of processors and CCR are chosen as parameter variables. Since the structure of the real-world application graph is fixed, the parallelism factor is not selected as a variable parameter.



(a) 32 processors, $\lambda = 1$, C-CR=1 (b) 160 Tasks, $\lambda = 1$, CCR=1

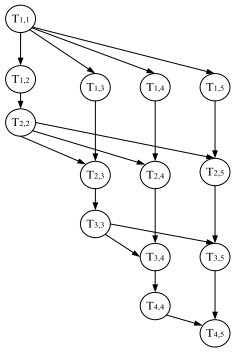


(c) 32 processors, 160 Tasks, $\lambda = 1$ (d) 32 processors, 160 Tasks, C-CR=1

Fig. 4. Average SLR of random DAGs with middle initialization frequency.

$$
\begin{aligned}
&for \quad k=1 \ \ to \ \ m-1 \ \ do \\
&\quad T_{k,k}: \{ \ for \ \ i=k+1 \ \ to \ \ m \ \ do \\
&\qquad\qquad a_{ik} = a_{ik} / a_{kk} \quad \} \\
&\quad for \ \ j=k+1 \ \ to \ \ m \ \ do \\
&\qquad T_{k,j}: \{ \ for \ \ i=k+1 \ \ to \ \ m \ \ do \\
&\qquad\qquad a_{ij} = a_{ij} \cdot a_{ik} \cdot a_{kj} \ \ \}
\end{aligned}
$$

(a)



(b)

Fig. 5. (a) Gaussian elimination algorithm, (b) task graph for a matrix of size 5.

Table 5
Makespan decrease for diverse characteristics.

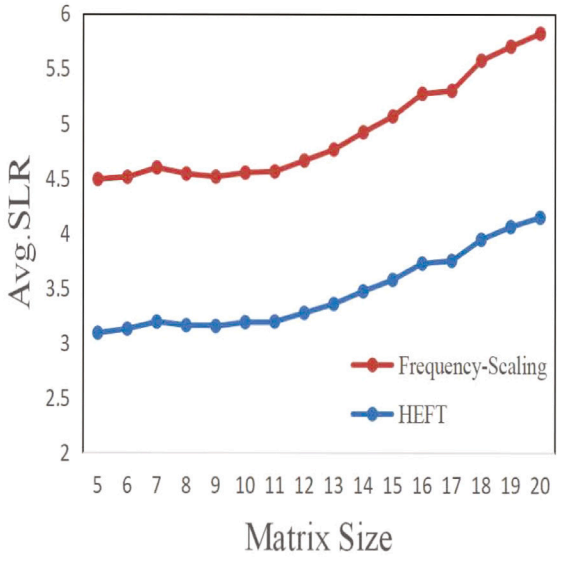
Number of tasks	20	40	80	160	320
HEFT *vs.* Frequency-Scaling	191%	193%	188%	190%	193%
Number of processors	4	8	16	32	64
HEFT *vs.* Frequency-Scaling	240%	204%	149%	190%	189%
CCR	0.2	0.5	1	5	10
HEFT *vs.* Frequency-Scaling	258%	226%	190%	125%	115%
Parallelism factor	0.2	0.5	1	5	10
HEFT *vs.* Frequency-Scaling	227%	198%	190%	166%	161%

Table 6
Makespan decrease for 8 processors with respect to a different number of tasks.

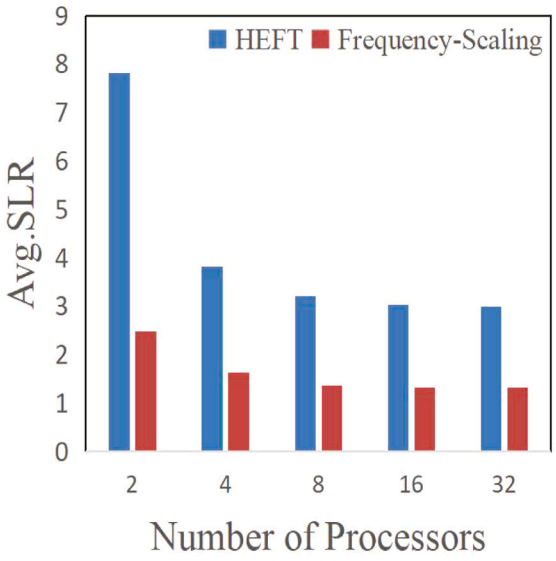
Number of tasks	20	40	80	160	320
HEFT *vs.* Frequency-Scaling	200%	200%	196%	204%	215%

4.3.1. Gaussian elimination

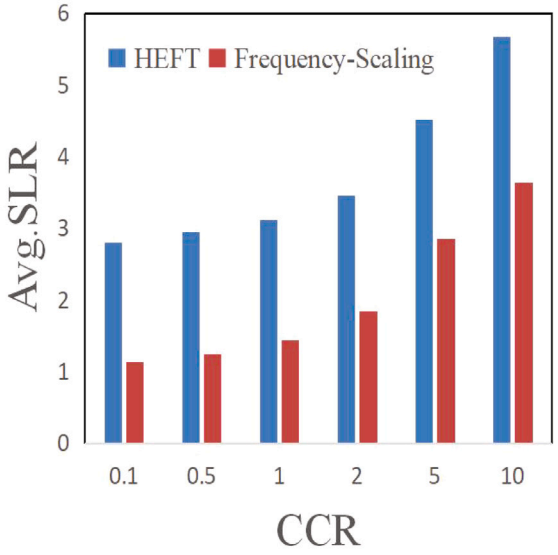
GE is an algorithm to solve the systems of linear equations in mathematics. The sequential program for the GE algorithm is represented in Fig. 5(a) [45]. Fig. 5(b) indicates the data-flow graph for the special case to solve a 5*5 matrix. $T_{k,k}$ and $T_{k,j}$ indicate a pivot column operation and an update operation respectively. A parameter s is applied which affects the DAG size S in experiments. The relationship



(a) Average SLR of the Gaussian elimination task graph with different matrix size.



(b) Average SLR of the Gaussian elimination graph with middle initialization frequency (M=15, CCR=1)



(c) Average SLR of the Gaussian elimination graph with middle initialization frequency (M=5, P=5)

Fig. 6. Average SLR of the Gaussian elimination task graph.

Table 7
Configurations for Gaussian elimination DAG generation.

Parameters	Possible values
CCR	0.1, 0.5, 1, 2, 5, 10
Number of processors	2, 4, 8, 16, 32
Size	5, 6, 7, …, 20

between matrix size and DAG size is $S = \frac{1}{2}(s^2 + s - 2)$. Fig. 6b illustrates a GE graph of matrix size 5. Table 6 enlists some variable parameters applied in the GE graphs. When the matrix size varies from 5 to 20, the number of tasks will increase from 14 to 209.

Fig. 6(a) highlights the average SLR results when matrix sizes changes. Under these circumstances, the number of processors and CCR is 5 and 1, respectively. Increasing the size of the matrix will increase the number of tasks. This results in more tasks outside the

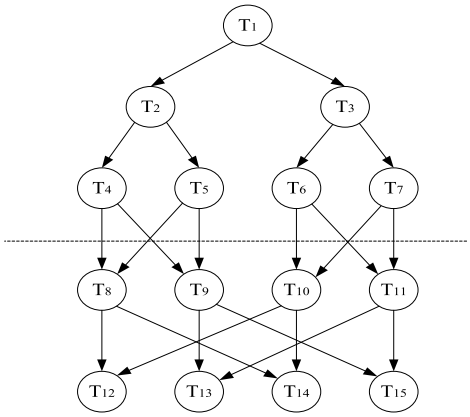


Fig. 7. FFT task graph with 4 input points.

Table 8
Configurations for FFT DAG generation.

Parameters	Possible values
CCR	0.1, 0.5, 1, 2, 5, 10
Number of processors	2, 4, 8, 16, 32

critical path, which increases the scheduling length. Fig. 6(b) demonstrates the experiment results with the same CCR. As the processors increase, FSA can obtain smaller SLR compared to HEFT. The increase in CCR can transform an application from computation-intensive to communication-intensive when the number of processors is the same. In this case, it causes a greater makespan with greater SLR (see Fig. 6(c)).

4.3.2. Fast Fourier transformation

FFT [46] is an algorithm for calculating discrete Fourier transform and its inverse transform. It comprises two operations: the input vector and butterfly operation. The one-dimensional four-point recursive FFT graph in Fig. 7 is considered as an example. The parameters applied in the experiment are illustrated in Table 8. For FFT tasks, the calculation cost and communication cost of each task are equal. Thus, each path is a critical path. Only the CCR is selected as a parameter for performance evaluation. The total number of tasks S and the number of FFT layers n follow this relationship, $S_{2n} = 2^{2n}(n \geq 2)$. With the change of n, the input FFT points change in step 4.

Fig. 8 indicates the average SLR results with a diverse numbers of processors and CCR. The number of processors is varied by the power of 2 from 2 to 64, and six different values are set for CCR. In Fig. 8, it can be observed that the performance of the FSA exceeds the performance of HEFT in all the cases. Fig. 8(a) displays that the increasing number of processors will not improve the performance because four processors are enough to execute the FFT graph of four parallel data points. For FFT, the performance improvement brought by frequency scaling is 151%, 153%, 173%, 202%, and 244% better than the HEFT algorithm when the CCR is 10, 5, 2, 1, 0.5, and 0.1, respectively.

5. Conclusions

In this paper, the influence of various initial frequency levels on the scheduling performance is assessed. Three kinds of initial mapping frequencies are considered with HEFT algorithm in each execution for time-efficient performance. Based on the initial mapping, a Frequency Scaling Algorithm (FSA) has been proposed to further optimize the makespan under the energy constraint while meeting the precedence constraint. This algorithm combines the HEFT algorithm and DVFS technique, and a new method is studied for processor selection and frequency scaling phases. It achieves high performance for the tested applications as well as energy efficiency.

Based on a large number of graphs, the experimental results are somehow unexpected. The mappings with middle frequencies are better than those with the other two initial mappings, including the highest frequencies. Besides, it has also been noticed that the proposed frequency scaling method can mitigate the impacts of initial mappings to a certain extent.

FSA can provide less execution time and stable performance. It is a feasible solution to the DAG scheduling problem in heterogeneous computing systems. Migration methods are introduced to deal with the trade-off between energy conservation and scheduling length in the next work. Furthermore, this algorithm can be extended to edge computing systems and also further critical issues such as system reliability.

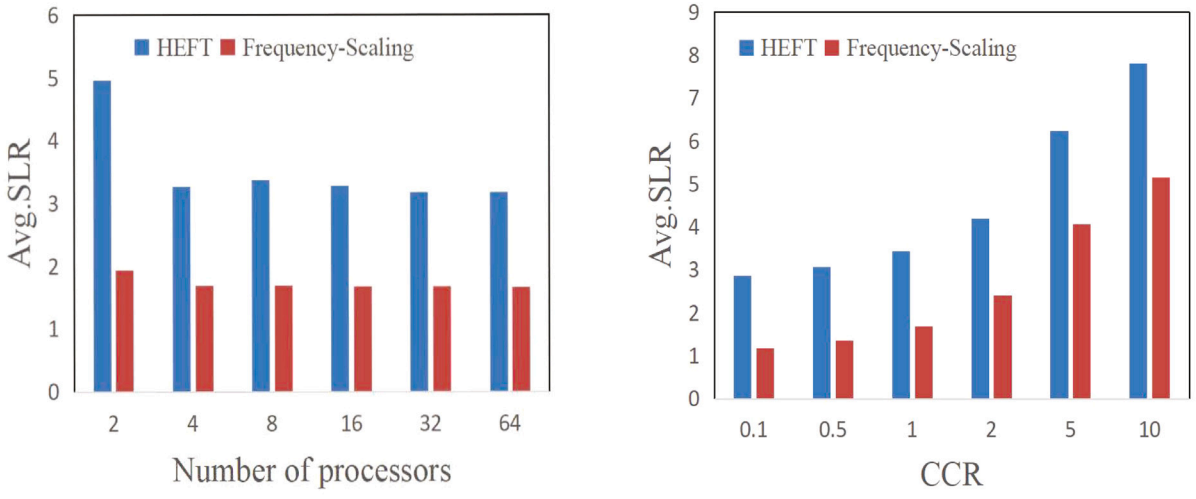


Fig. 8. Average SLR for the FFT graph with middle initialization frequency.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005) and the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202).

References

[1] L.T. Yang, X. Wang, X. Chen, L. Wang, R. Ranjan, X. Chen, M.J. Deen, A multi-order distributed HOSVD with its incremental computing for big services in cyber-physical-social systems, IEEE Trans. Big Data (2018) 1.

[2] X. Wang, L.T. Yang, X. Chen, J.J. Han, J. Feng, A tensor computation and optimization model for cyber-physical-social big data, IEEE Trans. Sustain. Comput. 4 (4) (2019) 326–339.

[3] C. Chen, K. Li, A. Ouyang, Z. Tang, K. Li, GFlink: An in-memory computing architecture on heterogeneous CPU-GPU clusters for big data, in: International Conference on Parallel Processing, 2016.

[4] Cen, Chen, Kenli, Li, Aijia, Ouyang, Keqin, Li, Flinkcl: An opencl-based in-memory computing architecture on heterogeneous CPU-GPU clusters for big data, IEEE Trans. Comput. 67 (12) (2018) 1765–1779.

[5] K. Fox, S. Im, B. Moseley, Energy efficient scheduling of parallelizable jobs, Theoret. Comput. Sci. 726 (2018) 948–957.

[6] Calore, Enrico, Gabbana, Alessandro, Schifano, Sebastiano, Fabio, Tripiccione, Raffaele, Software and DVFS tuning for performance and energy-efficiency on intel KNL processors, J. Low Power Electron. Appl. (2018).

[7] F. Juarez, J. Ejarque, R.M. Badia, Dynamic energy-aware scheduling for parallel task-based application in cloud computing, Future Gener. Comput. Syst. 78 (pt.1) (2018) 257–271.

[8] P. Chaudhuri, J. Elcock, Task scheduling in multiprocessing systems using duplication, J. Syst. Archit. 54 (5) (2008) 519–529.

[9] H. Topcuoglu, S. Hariri, M.Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.

[10] Y.C. Lee, A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Trans. Parallel Distrib. Syst. 22 (8) (2010) 1374–1381.

[11] M.A. Khan, Scheduling for heterogeneous systems using constrained critical paths, Parallel Comput. 38 (4–5) (2012) 175–193.

[12] D. Guo, Q. Xiao, X.U. Jianxin, Cloud computing energy consumption optimization model research based on Weibull distribution, Comput. Eng. Appl. (2017).

[13] S. Vila, F. Guirado, J.L. Lerida, F. Cores, Energy-saving scheduling on laas HPC cloud environments based on a multi-objective genetic algorithm, J. Supercomput. 75 (3) (2019) 1483–1495.

[14] R. Garg, N. Shukla, Energy efficient level by level scheduling for multiple workflows in cloud, Int. J. Softw. Innov. 7 (3) (2019) 102–117.

[15] W. Zheng, S. Huang, An Adaptive Deadline Constrained Energy-Efficient Scheduling Heuristic for Workflows in Clouds, John Wiley and Sons Ltd., 2015, pp. 5590–5605.

[16] W. Zheng, S. Huang, Deadline constrained energy-efficient scheduling for workflows in clouds, in: International Conference on Advanced Cloud & Big Data, 2014, pp. 69–76.

[17] L. Marchal, B. Simon, O. Sinnen, F. Vivien, Malleable task-graph scheduling with a practical speed-up model, IEEE Trans. Parallel Distrib. Syst. (2018) 1357–1370.

[18] B. Young, S. Pasricha, A.A. Maciejewski, H.J. Siegel, J.T. Smith, Heterogeneous makespan and energy-constrained DAG scheduling, ACM (2013).

[19] H.F. Sheikh, I. Ahmad, Simultaneous optimization of performance, energy and temperature for DAG scheduling in multi-core processors, in: Green Computing Conference, 2012.

[20] U.G. Joo, Makespan minimization scheduling problem with energy-efficient turning On/Off mechanism, J. Korean Inst. Ind. Eng. 44 (1) (2018) 1–8.

[21] M. Aitaba, L. Zaourar, A. Munier, Efficient algorithm for scheduling parallel applications on hybrid multicore machines with communications delays and energy constraint, Concurr. Comput. Pract. Exp. (6) (2020) e5573.

[22] V. Kachitvichyanukul, K. Sethanan, P. Golinska-Dawson, Makespan minimization for scheduling unrelated parallel machine with sequence-dependent setup time, 10.1007/978-3-319-19006-8, (Chapter 17) 2015, pp. 253–264.

[23] N. Hashemian, C. Diallo, B. Vizvari, Makespan minimization for parallel machines scheduling with multiple availability constraints, Ann. Oper. Res. 213 (feb.) (2014) 173–186.

[24] Kahng, B. A., Kang, S., Kumar, R., Sartori, J., Enhancing the efficiency of energy-constrained DVFS designs, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 21 (10) (2013) 1769–1782.

[25] N.B. Rizvandi, J. Taheri, A.Y. Zomaya, Some observations on optimal frequency selection in DVFS-based energy consumption minimization, J. Parallel Distrib. Comput. 71 (8) (2011) 1154–1164.

[26] Y.C. Lee, A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Trans. Parallel Distrib. Syst. 22 (8) (2011) 1374–1381.

[27] M. Qiu, Z. Ming, S. Liu, S. Liu, B. Wang, Z. Lu, Three-phase time-aware energy minimization with DVFS and unrolling for chip multiprocessors, J. Syst. Archit. 58 (10) (2012) 439–445.

[28] D. Zhu, R. Melhem, B. Childers, Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems, IEEE Trans. Parallel Distrib. Syst. 14 (7) (2003) 686–700.

[29] M. Mezmaz, N. Melab, Y. Kessaci, Y.C. Lee, E.G. Talbi, A.Y. Zomaya, D. Tuyttens, A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, J. Parallel Distrib. Comput. 71 (11) (2011) 1497–1508.

[30] A. To, An efficient biobjective heuristic for scheduling workflows on heterogeneous DVS-enabled processors, J. Appl. Math.,2014,(2014-7-7) 2014 (12) (2014) 1–15.

[31] Tarplee, Kyle, M, Friese, Ryan, Maciejewski, Anthony, A, Siegel, Howard, Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques, IEEE Trans. Parallel Distrib. Syst. (2016).

[32] L.F. Bittencourt, R. Sakellariou, E. Madeira, DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm, in: 2010 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2010.

[33] Z. Tang, L. Qi, Z. Cheng, K. Li, S.U. Khan, K. Li, An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment, J. Grid Comput. 14 (1) (2016) 55–74.

[34] Y. Zhang, Y. Wang, H. Wang, Energy-efficient task scheduling for DVFS-enabled heterogeneous computing systems using a linear programming approach, in: Performance Computing and Communications Conference, 2017, pp. 1–8.

[35] X. Chen, K. Li, C. Liu, SLA-based energy aware scheduling of precedence-constrained applications on DVFS-enabled clusters, in: IEEE International Conference on Parallel and Distributed Systems, 2014, pp. 336–343.

[36] Y. Zhang, X.S. Hu, D.Z. Chen, Task scheduling and voltage selection for energy minimization, 2002, pp. 183–188.

[37] K. Li, X. Tang, K. Li, Energy-efficient stochastic task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2014) 2867–2876.

[38] Y. Zhang, Y. Wang, C. Hu, CloudFreq: Elastic energy-efficient bag-of-tasks scheduling in DVFS-enabled clouds, in: IEEE International Conference on Parallel and Distributed Systems, 2016, pp. 585–592.

[39] L. Wang, G.V. Laszewski, J. Dayal, F. Wang, Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS, in: Ieee/Acm International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 368–377.

[40] J. Kang, S. Ranka, Slack allocation algorithm for parallel machines, J. Parallel Distrib. Comput. 70 (1) (2010) 23–34.

[41] H. Kimura, M. Sato, Y. Hotta, T. Boku, Emprical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster, in: IEEE International Conference on CLUSTER Computing, 2006, pp. 1–10.

[42] N.B. Rizvandi, J. Taheri, A.Y. Zomaya, Y.C. Lee, Linear combinations of DVFS-enabled processor frequencies to modify the energy-aware scheduling algorithms, in: Ieee/Acm International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 388–397.

[43] D. Zhu, R. Melhem, D. Mosse, The effects of energy management on reliability in real-time embedded systems, in: Ieee/Acm International Conference on Computer-Aided Design, 2004, pp. 35–40.

[44] A.P. Chandrakasan, S. Sheng, R.W. Brodersen, Low-power CMOS digital design, IEEE J. Solid-State Circuits 27 (4) (1995) 473–484.

[45] M.Y. Wu, D.D. Gajski, Hypertool: A programming aid for message-passing systems, Parallel Distrib. Syst. IEEE Trans. 1 (3) (1990) 330–343.

[46] Y.C. Chung, S. Ranka, Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors, in: ACM/IEEE Conference on Supercomputing, 1992, pp. 512–521.

Jie Liang received the B.S. degree in communication engineering from Henan normal university in 2011, and the M.S. degree in information and communication engineering from Hunan university in 2014. She is currently working toward the Ph.D. degree at Hunan University, China. Her research interests are mainly in modeling and scheduling of distributed computing systems, optimization and parallel algorithms, game theory, grid and cloud computing.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar with the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently the dean and a full professor of computer science and technology with Hunan University and deputy director of National Supercomputing Center in Changsha. His major research areas include parallel computing, high-performance computing, grid, and cloud computing. He has published more than 150 research papers in international conferences and journals such as the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Signal Processing*, the *Journal of Parallel and Distributed Computing*, ICPP, and CCGrid. He serves on the editorial board of the *Transactions on Computers*. He is an outstanding member of the CCF. He is a senior member of the IEEE.

Chubo Liu received the B.S. degree and Ph.D. degree in computer science and technology from Hunan University, China, in 2011 and 2016, respectively. He is currently an associate professor of computer science and technology at Hunan University. His research interests are mainly in game theory, approximation and randomized algorithms, cloud and edge computing. He has published over 8 papers in journals and conferences such as the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Cloud Computing*, the *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, the *Theoretical Computer Science*, and ICPADS.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, big data computing, high performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 770 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds nearly 60 patents announced or authorized by the Chinese National Intellectual Property Administration. He has chaired many international conferences. He is currently an associate editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is an IEEE Fellow.