Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 3676149, 11 pages
http://dx.doi.org/10.1155/2016/3676149

Research Article

HSIP: A Novel Task Scheduling Algorithm for Heterogeneous Computing

Guan Wang,[1,2] Yuxin Wang,[3] Hui Liu,[1] and He Guo[1]

[1]*School of Software Technology, Dalian University of Technology, Dalian 116620, China*
[2]*Liaoning Police College, Dalian 116036, China*
[3]*School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China*

Correspondence should be addressed to Yuxin Wang; wyx@dlut.edu.cn

Received 22 September 2015; Revised 9 January 2016; Accepted 16 February 2016

Academic Editor: Bronis R. de Supinski

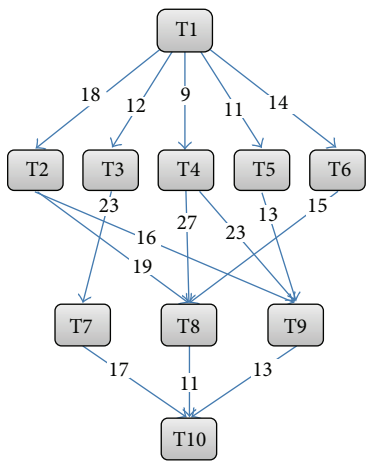
High-performance heterogeneous computing systems are achieved by the use of efficient application scheduling algorithms. However, most of the current algorithms have low efficiency in scheduling. Aiming at solving this problem, we propose a novel task scheduling algorithm for heterogeneous computing named HSIP (heterogeneous scheduling algorithm with improved task priority) whose functionality relies on three pillars: (1) an improved task priority strategy based on standard deviation with improved magnitude as computation weight and communication cost weight to make scheduling priority more reasonable; (2) an entry task duplication selection policy to make the makespan shorter; and (3) an improved idle time slots (ITS) insertion-based optimizing policy to make the task scheduling more efficient. We evaluate our proposed algorithm on randomly generated DAGs, using some real application DAGs by comparison with some classical scheduling algorithms. According to the experimental results, our proposed algorithm appears to perform better than other algorithms in terms of schedule length ratio, efficiency, and frequency of best results.

1. Introduction

In the era of big data, data intensive computing cannot rely on a single processor to be completed. It often relies on heterogeneous computing system (HCS). A heterogeneous computing system defined as high-speed network interconnection of multiple processors computing platform can carry out parallel and distributed intensive computing [1, 2]. The effectiveness of performing similar applications on heterogeneous computing systems relies on task scheduling methods [3, 4]. Typically, an effective task scheduling method can improve the efficiency of a heterogeneous computing system. Task scheduling methods aim to minimize the overall time of completion (makespan) [5]. In detail, a task scheduling algorithm needs to record the operations of the processors and command their completion under the requirement of task precedence.

Typical task scheduling algorithms include Heterogeneous Earliest Finish Time [6] and Critical Path On a Processor [6], Standard Deviation-Based Algorithm for Task Scheduling [7], and Predict Earliest Finish Time [8]. Although they have been widely used in heterogeneous computing systems, they still have three drawbacks. First, most of them ignore heterogeneity of different computing resources and different communication between computing resources. Second, current methods adopting entry task duplication to all the processors lead to the overload of CPU. Finally, they do not have an effective inserting-based policy.

Aiming at solving the three problems, this paper proposes the Heterogeneous Scheduling with Improved Task Priority (HSIP). It works in two steps: task prioritizing stage followed by processors selection stage. In the first step, the algorithm combines the standard deviation with the communication cost weight to determine the priorities of the tasks. In the second stage, we proposed an entry task duplication strategy to determine whether there is a need for entry task duplicate to other processors. At the same time, the improved insertion-based optimizing policy makes the makespan shorter. The experimental results show that our projected algorithm performs better than other algorithms in



Task	P_1	P_2	P_3
T1	14	16	9
T2	13	19	18
T3	11	13	19
T4	13	8	7
T5	12	13	10
T6	13	16	9
T7	7	15	11
T8	5	11	14
T9	18	12	20
T10	21	7	16

Figure 1: Example of DAG task model and computation cost matrix.

terms of schedule length ratio, efficiency, and frequency of best results.

The rest of this paper is constructed as follows: Section 2 presents the task scheduling problem and Section 3 outlines the related work on the task scheduling in heterogeneous computing systems. The projected algorithm is clarified in Section 4 and the experimental outcomes are presented in Section 5. Section 6 summarizes and concludes the findings of this paper.

2. Heterogeneous Task-Scheduling Problem

A task scheduling model is composed of an application, a target computing environment, and performance benchmarks. An application can be characterized by a directed acyclic graph (DAG), $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of v nodes and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. A DAG model example in Figure 1 is similar to literature [6]. Each node $v_i \in V$ denotes an application task. Each $e(i, j) \in E$ denotes that the communication cost between two jobs under the task dependence constraint such as task v_i should fulfill its execution prior to task v_j. In a given application DAG, a task without any parent is known as an entry task while a task without any child is regarded as an exit task. If a DAG has more than one entry (exit) node, a mock entry (exit) node with zero both in weight and in communication edges is added to the graph. The DAG is supplemented by a matrix W that is equal to $v \times p$ computation cost matrix, where v represents the number of tasks and p symbolizes the number of processors in the system. $W_{i,j}$ provides the estimated time for task v_i completion on machine p_j. The mean time for task completion of task v_i is calculated as the following equation:

$$\overline{w_i} = \frac{\left(\sum_{j \in P} w_{i,j}\right)}{P}. \tag{1}$$

$c_{i,j}$ denotes the communication cost between task v_i and task v_j. When both v_i and v_j are processed on the same processor, $c_{i,j}$ becomes zero due to neglecting the interprocessor communication costs. The mean communication costs are commonly calculated to label the edges [6]. The average communication cost $\overline{c_{i,j}}$ of an edge $e(i, j)$ is calculated as the following equation:

$$\overline{c_{i,j}} = \overline{L} + \frac{\text{data}_{i,j}}{\overline{B}}, \tag{2}$$

where $\overline{L}$ indicates the average latency time of all processors and $\overline{B}$ is the mean bandwidth of all links that connects the group of P processors. $\text{data}_{i,j}$ is the quantity of data elements that task v_i needs to send to task v_j.

Furthermore, in our model, the processors are considered in a fully linked topology. The task execution and communication with other processors can be attained for each processor simultaneously and without conflict. And now we will present some of the common characteristics used in task scheduling, which we will discuss in the following sections.

Definition 1. Makespan, or schedule length, represents the finish time of the last task in the scheduled DAG, defined as (3), where $\text{AFT}(v_{\text{exit}})$ denotes the Actual Finish Time of the exit node:

$$\text{makespan} = \max\left\{\text{AFT}\left(v_{\text{exit}}\right)\right\}. \tag{3}$$

Definition 2. $\text{EST}(v_i, p_j)$ indicates the Earliest Start Time (EST) of a node v_i on a processor p_j and is defined as

$$\begin{aligned}&\text{EST}\left(v_i, p_j\right)\\ &\quad = \max\left\{T_{\text{Avl}}\left(p_j\right), \max_{v_m \in \text{pred}(v_i)}\left\{\text{AFT}\left(v_m\right) + c_{m,j}\right\}\right\},\end{aligned} \tag{4}$$

where $T_{\text{Avl}}(p_j)$ is the earliest time when processor p_j is ready while $\text{pred}(v_i)$ is the group of immediate predecessor tasks of task v_i. The inner max bracket in the EST equation represents the time all data were requested by v_i arriving at the processor p_j. The communication cost $c_{m,i}$ is zero when the predecessor node v_m is given to processor p_j. For the entry task, $\text{EST}(v_{\text{entry}}, p_j) = 0$.

Definition 3. $\text{EFT}(v_i, p_j)$ represents the Earliest Finish Time (EFT) of a node v_i on a processor p_j and is defined as

$$\text{EFT}\left(v_i, p_j\right) = \text{EST}\left(v_i, p_j\right) + w_{i,j}, \tag{5}$$

which represents the Earliest Start Time of a node v_i on a processor p_j plus the computational cost of v_i on a processor p_j. For the entry task, $\text{EFT}(v_{\text{entry}}, p_j) = w_{v_{\text{entry}},j}$.

Definition 4. Out-degree communication cost weight (OCCW) of task v_i means the possible max sum of communication costs generated by v_i with its immediately successors and is defined as follows:

$$\begin{aligned}\text{occw}\left(v_i\right) &= \sum_{v_j \in \text{succ}(v_i)} c_{ij},\\ \text{occw}\left(v_{\text{exit}}\right) &= 0.\end{aligned} \tag{6}$$

Out-degree communication cost weight also affects the task priorities ordering. If a task with larger out-degree communication cost weight was not executed, all its successors would not be ready.

The aim of the scheduling issue is to determine an assignment of the tasks in a given DAG to processors so that the schedule length is reduced to a minimum. When all nodes in the DAG are scheduled, the schedule length will now become AFT, the Actual Finish Time of the exit task, as expressed by (3).

3. Related Works

Recently, a number of task scheduling algorithms in heterogeneous computing systems have been projected. They can approximately be categorized into two groups, dynamic scheduling and static scheduling. In the dynamic category, the execution, communication costs, and the relationship of the tasks are unknown. Decisions are made at runtime. While in the static category, such information is known ahead of time. Dynamic scheduling is runtime scheduling, whereas static scheduling is compile-time scheduling.

Dynamic scheduling means when new task comes, only the task is about to be executed and the freshly arrived task will be reflected in the rescheduling process. Dynamic scheduling is adequate for conditions in which the system and task parameters are unknown at the compiled time. Therefore decisions are made at runtime with further observations. Some typical dynamic scheduling algorithms have been presented in the literatures, such as Batch Mode Mapping Heuristics [9], Dynamic Mapping Heuristics [10], Dynamic Scheduling Cycle Strategy [11], and dynamic scheduling method [12].

Static scheduling algorithms are categorized into two major groups, that is, guided random search-based algorithms and heuristic-based algorithms. Typical guided random search-based algorithms include GA Multiprocessor Task Scheduling [13], Knowledge-Augmented Genetic Approach [14], and Problem-Space Genetic Algorithm (PSGA) [15]. They give approximate solutions through more iterations, which increase the costs as opposed to the heuristic-based approach. The heuristic-based group comprises three subcategories: list, clustering, and duplication scheduling. Heterogeneous Earliest Finish Time [6], Critical Path On a Processor [6], Standard Deviation-Based Algorithm for Task Scheduling [7], Predict Earliest Finish Time [8], Longest Dynamic Critical Path (LDCP) [16], Heterogeneous Critical Parent Trees (HCPT) [17], High-Performance Task Scheduling (HPS) [18], low complexity Performance Effective Task Scheduling (PETS) [19], Heterogeneous Earliest Finish with Duplicator (HEFD) [20], and Selective Duplication Algorithm [21] are typical heuristic-based algorithms. Clustering heuristics are primarily proposed for homogeneous systems, but they have limitations in higher level heterogeneity systems. The duplication heuristics generate the shortest makespan but cause a higher time complexity. The execution of task duplication consumes more processor power. This not only causes more power consumption, but also more importantly in the sharing resource occupies processors that are used for other tasks. List scheduling heuristics ensure comparatively more efficient schedule with a complexity that generally can be quadratic with respect to the number of tasks. Because it produces relative shorter scheduling length with low algorithm complexity as $O(v^2, p)$, HEFT algorithm [6] becomes the most popular and widely used algorithm.

HEFT uses the mean value of the computation cost and the mean value of communication cost as the rank value to determine the scheduling sequence. But it is considered less reasonable for the heterogeneous environment. If the computation costs of the same task on different processors are too large, and if the communication cost weights of the task are too large, the HEFT algorithm will not give a justified scheduling. CPOP algorithm [6] also has a complexity of $O(v^2, p)$. In this approach all critical path tasks are assigned to the same processor, which causes load unbalance of processors and increases the schedule length.

SDBATS algorithm is based on the HEFT algorithm and makes the performance significantly improved [7]. But SDBATS uses the standard deviation of the computation cost to calculate the rank value for priority instead of the mean value of the computation cost. This will cause the unfairness of task scheduling when the communication cost is too large. It is not necessary to use standard deviation of the communication cost when the communication cost of each node to the lower level node is 0 or another certain value. SDBATS algorithm also runs entry task on all the processors at the beginning of the scheduling. This policy will increase the scheduling length if there is a remarkable difference in the computation cost among the processors.

The latest excellent DAG scheduling algorithm is the PEFT algorithm [8]. This algorithm puts forward the priority weights Optimistic Cost Table (OCT) by introducing a look-ahead feature, which chooses the minimum sum of the computational cost and communication cost in all child nodes for task scheduling. It also uses this strategy in the processor allocation. But when the same node in different processor calculation cost difference is large, the algorithm does not give a reasonable allocation strategy. When the subnode communication cost weight difference is large (high parallelism in the DAG and big communication data), PEFT loses its advantage.

4. The Proposed HSIP Algorithm

In this section, we introduce a new scheduling algorithm for a confined number of heterogeneous processors, known as Heterogeneous Scheduling with Improved Task Priority (HSIP). The algorithm contains two key stages: a task prioritizing stage for calculating task priorities and a processor selection stage for choosing the best processor to execute the current task.

4.1. Detailed Description of HSIP Algorithm. In task prioritizing stage, we improved the task priority strategy. In the processor selection stage, according to the priority of

```
Input: DAG, set of tasks V, set of Processors P
Output: Schedule result, makespan
(1)  Starting from the exit node, compute rank_u for all tasks by using "Improved Task Priority Strategy".
(2)  Sort the tasks in scheduling list by decreasing order of rank_u value.
(3)  While there are unscheduled tasks in the list do
(4)      Select the first task v_i from the list for scheduling
(5)      If the task is the entry task
(6)          Use "Entry Task Duplication Selection Policy"
(7)      Else (task v_i is not the entry task)
(8)          if satisfy the condition of ITS insertion-based optimizing policy
(9)              Use "ITS Insertion-based Optimizing Policy"
(10)         else
(11)         for each processor p_j in the processor set (p_j ∈ P) do
(12)             Compute the earliest finish time (EFT) by (5)
(13)         end
(14)         Assign task v_i to the processor p_j that minimize EFT of task v_i
(15)         End if
(16)     End if
(17)     Update list
(18) End while
```

Algorithm 1

task scheduling order, tasks are assigned to the minimum EFT processor to be executed [6]. On the basis of the above strategy, we proposed two innovative policies, entry task duplication selection policy and idle time slots (ITS) insertion-based optimizing policy. They improve the efficiency of scheduling algorithm.

The detailed description of HSIP algorithm is as shown in Algorithm 1.

4.1.1. An Improved Task Priority Strategy. In descending order of rank_u value as a scheduling priority, the upward rank, rank_u, of each task has been calculated using the following equation:

$$\begin{aligned} &\text{rank}_u(v_i) \\ &\quad = \max_{v_j \in \text{succ}(v_i)} \left\{ \overline{w_i} \times \sigma_i + \text{occw}(v_i) + \text{rank}_u(v_j) \right\}, \\ &\text{rank}_u(v_{\text{exit}}) = \overline{w_{\text{exit}}} \times \sigma_{\text{exit}}, \end{aligned} \tag{7}$$

where σ_i is the standard deviation of computation cost of any given task v_i on the available pool of processors.

Standard deviation works better than the mean value with the response to the differences of the computation cost. When computation costs of the same task in different processors differ largely, the standard deviation value will be big. Or else, it will be small. Therefore, using the standard deviation can prioritize the node with larger computation cost differences and improve the overall scheduling results.

However, the standard deviation value of the calculation cost is far below the communication cost weight of task in terms of magnitude. Our algorithm multiplies standard deviation by the average cost as the calculation cost weight. Thus, the task with larger difference of computing cost can get higher priority, as well as the task with large transmission time to child nodes. In fact, our algorithm rank_u equation can produce better scheduling policy comparing to the other state-of-the-art algorithms, and the results are shown in Figure 2. Descending order of the upward rank $\text{rank}_u(v_i)$ is as the task priority in our approach. The upward rank of each task for instance provided by Figure 1 is shown in Table 1.

4.1.2. Entry Task Duplication Selection Policy. Traditional task duplication algorithm has shorter length of scheduling. But it is limited by the overload of the processor utilization, mentioned in the literature [6, 8, 20, 21]. However, for the entry task (the first scheduled task), when it is running on one processor in the beginning, the other processors are idle at this time. So there is no need to take processor overload problem into consideration. And other tasks do not have to wait when the processors run a copy. At the same time, if only the entry task was duplicated, the overloading problem could be avoided as much as possible. Our algorithm uses entry task duplication selection policy to avoid processor overloading and to improve the overall efficiency of the scheduling. In order to make the child nodes get faster data transmission time, this strategy evaluates the necessity of entry task scheduling in each processor. This policy is good for the entry task with little computation cost difference and with large communication cost. It also does not affect the other tasks scheduling results because the entry task duplication will not run if it cannot improve the scheduling result according to the judgment mechanism as follows.

Only the entry task needs the following duplication selection policy:

(i) Choose the processor p_j, which produce the minimum EFT for entry task.

(ii) Determine whether the entry task in another processor p_i ($p_i \in P$) needs to be duplicated. If (8) is true,

Table 1: Priority weights of tasks.

Function	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
$\text{rank}_u(v_i)$	335.6	233.4	209.6	229.1	182.2	184.7	137.9	133.4	154.6	85.0

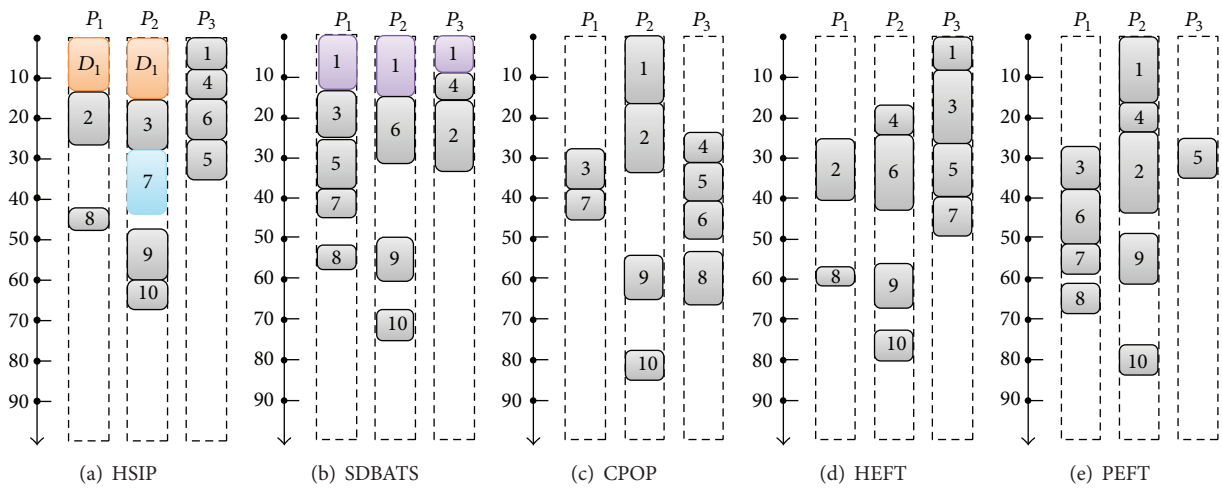


Figure 2: Schedules of the sample task graph in Figure 1 with HSIP, SDBATS, CPOP, HEFT, and PEFT.

then do entry task duplication in p_i; otherwise, do nothing. Consider

$$w_{v_{\text{entry}},i} < w_{v_{\text{entry}},j} + c_{v_{\text{entry}},v_n}, \tag{8}$$

where v_n is the immediate successor node of the entry task. The end condition of the above loop is the first satisfying one from the following two judgments:

(1) All processors have been assigned tasks; namely, each processor's entry task duplication judgment has been completed.

(2) All immediate successor nodes of the entry task (v_n) are scheduled; namely, they do not need entry task to transmit data.

4.1.3. ITS Insertion-Based Optimizing Policy. Insertion-Based strategy is proposed by the HEFT algorithm and adopted by many other scheduling algorithms. But there is no precise mathematical description for this mechanism. When multiple idle time slots (ITS) meet the insert conditions, HEFT algorithm just selects the first ITS rather than the fastest one to be completed. This strategy will cause the unreasonable scheduling problem. We refine HEFT algorithm's inserting-based constraints and provide a choosing policy when multiple slots satisfy the conditions. The detailed description is as follows:

(i) After completion of task allocation, update each processors' ITS queue.

(ii) When allocating v_i, look up ITS of all processors to find a slot with $w_{i,j} \leq$ ITS.

(iii) To all of the ITS meet the condition in step (2) and determine when v_i is assigned to the ITS execution and whether the EFT is less than or equal to the lower limit time of ITS.

(iv) When there are multiple time slots satisfying steps (2) and (3), choose the ITS with the smallest EFT.

HSIP algorithm has the same time complexity with the HEFT algorithm. Computing $\text{rank}_u(v_i)$ must traverse all tasks and compare processors, which can be done within $O(v, p)$ in initialization stage. Scheduling all tasks must traverse all tasks, which can be done in $O(v)$. Computing the EFT of all tasks can be done in $O(v, p)$. Thus, the complexity of the algorithm HSIP is $O(v^2, p)$.

4.2. A Case Study. Figure 2 and Table 2 show the results of scheduling for the sample DAG in Figure 1 with the algorithms HSIP, SDBATS, CPOP, HEFT, and PEFT. The corresponding result is shown in Table 2, and we can see that HSIP algorithm has shorter makespan and lesser communication costs. It is worth mentioning that, in Figure 2(a), D_1 is determined by entry task duplication selection policy of v_{entry} duplication; v_7 is determined by ITS insertion-based optimizing policy allocation on the processor P_2; both are the reason our algorithm can achieve significant scheduling results.

5. Experimental Results and Discussion

This section provides comparisons between the performance of the HSIP algorithm and the algorithms presented above. For this purpose, two sets of workload graphs are taken into consideration: randomly produced application graphs [22] along with graphs that represent some real-world applications. We start off with presenting the comparison metrics applied to assess the performance.

Table 2: Results of scheduling DAG.

	HSIP	SDBATS	CPOP	HEFT	PEFT
Task prioritizing	$v_1, v_2, v_4, v_3, v_6, v_5, v_9, v_7, v_8, v_{10}$	$v_1, v_3, v_4, v_2, v_6, v_5, v_7, v_9, v_8, v_{10}$	$v_1, v_2, v_3, v_7, v_4, v_5, v_9, v_6, v_8, v_{10}$	$v_1, v_3, v_2, v_4, v_5, v_6, v_9, v_7, v_8, v_{10}$	$v_1, v_4, v_2, v_5, v_3, v_6, v_7, v_9, v_8, v_{10}$
Makespan	67	76	86	80	85

5.1. Comparison Metrics. The metric that is most commonly adopted to appraise a schedule for a DAG, is the makespan, as defined by (3). Because of the implementation of a sizable set of task graphs with various characteristics, it is necessary to standardize the schedule length to the lower bound, known as the schedule length ratio (SLR), defined as follows:

$$\mathrm{SLR} = \frac{\text{makespan}}{\sum_{v_i \in \mathrm{CP_{MIN}}} \min_{p_j \in P} \left(w_{i,j}\right)}. \tag{9}$$

The denominator in the equation is the minimum computation cost of the critical path tasks ($\mathrm{CP_{MIN}}$). Makespan is always greater than the denominator in the SLR equation. Thus, the best algorithm is the algorithm that has the lowest SLR.

Efficiency is defined as the calculation of the speedup divided by the number of processors applied in each run, and the speedup value is calculated as dividing the time of sequential execution by the time of parallel execution (i.e., the makespan). The sequential execution time is calculated by assigning the entire tasks to a single one processor that minimizes the overall computation cost of the task graph, as shown in the following equation:

$$\mathrm{Speedup} = \frac{\min_{p_j \in P} \left\{\sum_{v_i \in V} w_{i,j}\right\}}{\text{makespan}}. \tag{10}$$

5.2. Random Graph Generator. In order to achieve a broad range of test DAGs, we have designed a task graph generator that can randomly generate DAGs with various features depending on input parameters the same as literature [8]. The parameters include number of tasks (v), shape parameter (fat), number of edge factors (density), symmetry parameter (regularity), the degree of leaping (jump), Communication to Computation Ratio (CCR), and range percentage of computation cost (η). By changing fat value we are able to generate different shapes of the task graphs. The height of the graph is related to $\sqrt{v}/\mathrm{fat}$, and the width for each level is equivalent to $\sqrt{v} * \mathrm{fat}$. A dense graph (shorter graph with high parallelism) is created by selecting fat $\gg 1.0$, while fat $\ll 1.0$ determines a longer graph (low parallelism).

The density defines the number of edges between two node levels, the lower value generates fewer edges, and the higher value generates more edges. That affects the connectivity between nodes of each level.

The regularity defines the uniformity of each level. The small value will cause the numbers of nodes in each level to differ largely, namely, an unsymmetrical DAG. On the contrary, the number of nodes in each level will be similar.

The jump is the degree of leaping, which decides the steps that show how the node jumps down. The jump value denotes how many leaping steps from the current node level to the down level, and jump = 1 denotes that the node of current layer connects next layer's nodes properly.

The range percentage of computation costs on processors (η) basically is the heterogeneity aspect for processors speeds. Parameter w_i is the average computation cost for each individual task v_i. The w_i is selected randomly from a uniform distribution with a range of $[0, 2 * W_{\mathrm{DAG}}]$. Where W_{DAG} represents the average computation cost in the given graph. The computation cost of each individual task v_i in the system on each processor p_j is decided randomly from the following range:

$$w_i \times \left(1 - \frac{\eta}{2}\right) \le w_{i,j} \le w_i \times \left(1 + \frac{\eta}{2}\right). \tag{11}$$

For the purpose of the experiments, we chose the range of values for the parameters as follows: $v = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500\}$; CCR $= \{0.1, 0.5, 0.8, 1, 2, 5, 10\}$; $\eta = \{0.1, 0.2, 0.5, 1, 2\}$; jump $= \{1, 2, 4\}$; regularity $= \{0.2, 0.8\}$; fat $= \{0.1, 0.4, 0.8\}$; density $= \{0.2, 0.8\}$; Processors $= \{4, 8, 16, 32\}$. These parameters produce 70560 different DAG models. Every DAG model generates 10 random DAGs with different edges and node weights. So there are 705600 random DAGs used in our research. This is also the same for the data in literature [8].

Average SLR is the key factor that evaluates the performance of the algorithm in terms of the graph structure. Figure 3 demonstrates the average SLR for the changing numbers of tasks. Figure 4 demonstrates the average SLR for different CCR. Figure 5 demonstrates the average SLR by heterogeneity (η) values with 0.1, 0.2, 0.5, 1.0, or 2.0. The efficiency values attained from each of the algorithms with different numbers of processors are shown in Figure 6. The standard deviations of the experimental errors are calculated and are typically between 3–6%.

In Figure 3 as the number of tasks increases, the SLR of our algorithm is smaller than other algorithms. Comparing the PEFT algorithm, in 10 tasks, the SLR of our algorithm is close to 10% higher than the PEFT. In 500 tasks, our algorithm is 5% higher. Figures 4 and 5 show that, with increasing CCR and heterogeneous parameters, SLR of our algorithm are better than other algorithms. Figure 6 shows that, with different numbers of processors, our algorithm also has higher efficiency than the others. The results further emphasize that HSIP algorithm outperforms the reported algorithms with respect to average SLR and efficiency for random task graphs with different shapes.

Table 3 shows the pairwise schedule length comparison of the scheduling algorithms. We can see that the HSIP is 68% better than, 31% worse than, and 1% equal to PEFT. Our

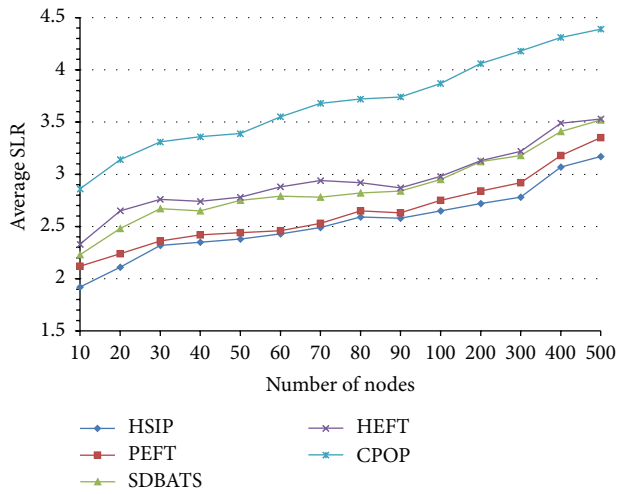


Figure 3: Average SLR for different number of tasks.

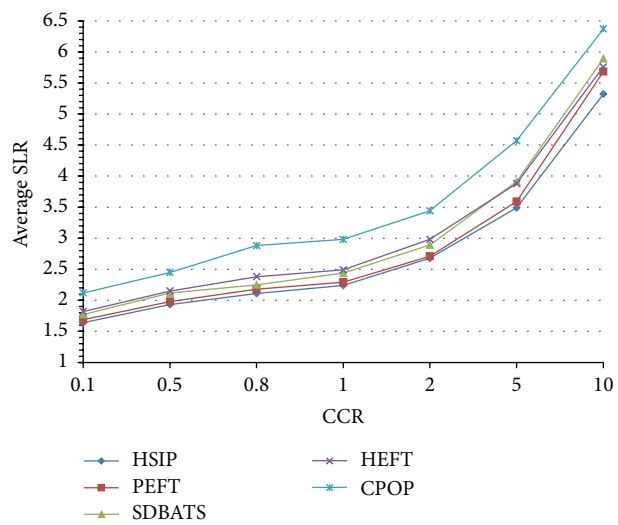


Figure 4: Average SLR for different CCR.

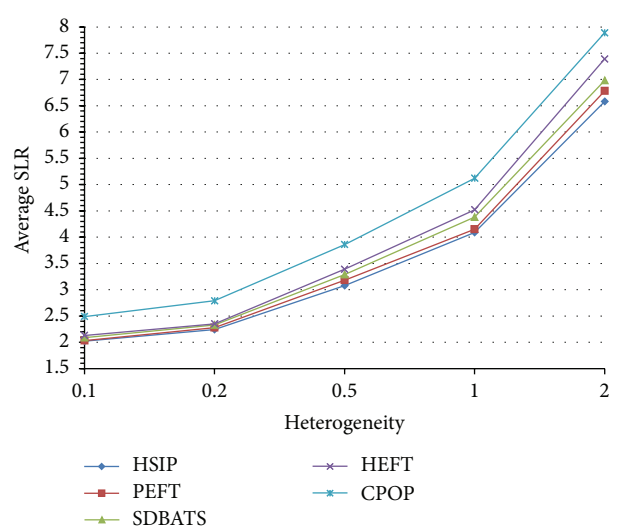


Figure 5: Average SLR for different heterogeneity.

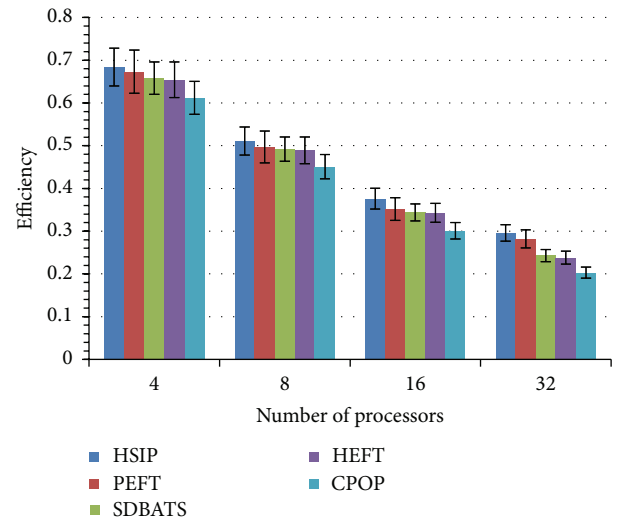


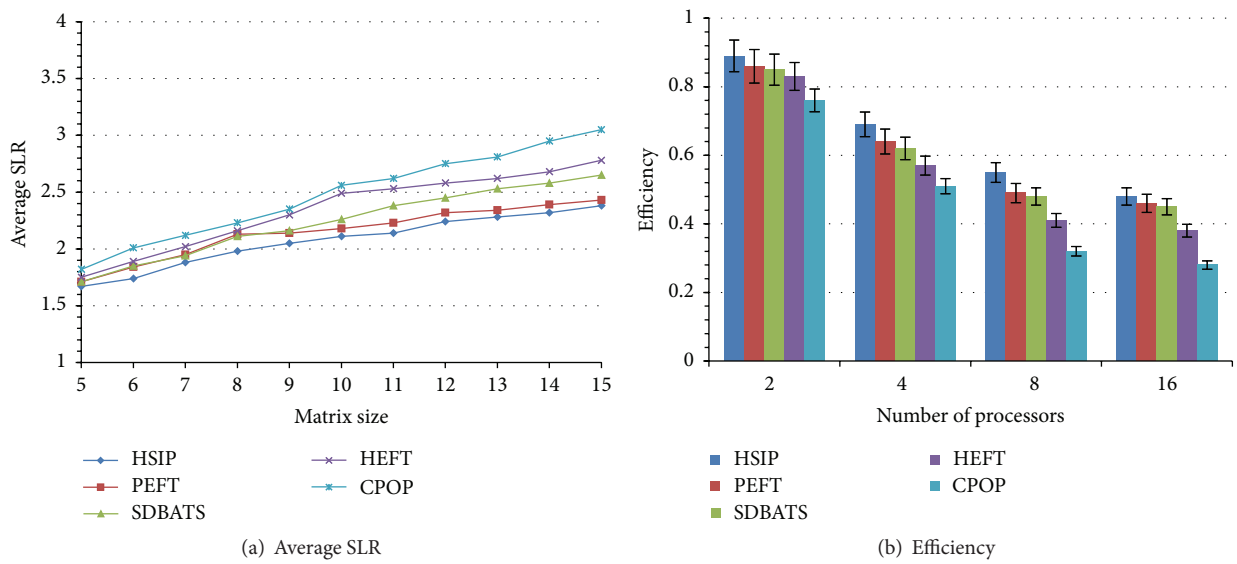
Figure 6: Efficiency for different number of processors.

Table 3: Pairwise schedule length comparison of the scheduling algorithms.

		HSIP	PEFT	SDBATS	HEFT	CPOP
HSIP	Better		68%	75%	81%	97%
	Worse	*	31%	17%	14%	2%
	Equal		<1%	8%	5%	<1%
PEFT	Better	31%		77%	70%	95%
	Worse	68%	*	32%	26%	4%
	Equal	<1%		<1%	4%	<1%
SDBATS	Better	17%	32%		61%	92%
	Worse	75%	77%	*	33%	7%
	Equal	8%	<1%		6%	<1%
HEFT	Better	14%	26%	33%		85%
	Worse	81%	70%	61%	*	14%
	Equal	5%	4%	6%		<1%
CPOP	Better	2%	4%	7%	14%	
	Worse	97%	95%	92%	85%	*
	Equal	<1%	<1%	<1%	<1%	

algorithm also holds the superiority, when it is compared with other algorithms.

Due to introducing a look-ahead feature, PEFT algorithm firstly considers the child nodes for the scheduling priority. So it has some advantages when the parallelism is low. But when the parallelism becomes high, this advantage is not obvious, especially when the DAG has a big heterogeneous difference. When the computation cost differences and communication cost weight of the child nodes are large, the PEFT algorithm often does not enjoy this advantage. Sometimes it is even not better than HEFT. The SDBATS algorithm is better than HEFT in some cases. But SDBATS ignores the insertion-based strategy and focuses on the computation cost differences too much. It does not have the advantage in the case of large communication cost. HEFT is the most classical scheduling algorithm. The effect is relatively stable, which can



(a) Average SLR (b) Efficiency

Figure 7: Experimental result for Gaussian elimination.

be seen from its maximum equivalence rate of scheduling results comparing with other algorithms. CPOP has the worst scheduling results, because it pays too much emphasis on the tasks of the critical path on the same processor. But it occasionally has good performance when the tasks on the critical path meet the scheduling optimal conditions.

The experiment results show that HSIP is better than other comparative algorithms in random DAG experiments of various parameters. Especially in the case of big heterogeneous difference, the advantage of our algorithm is more obvious. It is because HSIP pays more attention to the balance between the computational cost difference and communication cost weight as presented in Section 4.

5.3. Real-World Application Graphs. In this section, we take the application graphs of some real-world problems into account, namely, Gaussian elimination [23], Fast Fourier Transformation (FFT) [24], Montage [25], and Epigenomics [26], which are used in [6, 8].

In the Gaussian elimination applications experiment, heterogeneous computing systems using five processors, CCR and η, provided in Section 5.2, are adopted. As the structure of the application is already established, the parameters such as the number of tasks, jump, regularity, fat, and density are therefore not required. A new parameter matrix size (m) is used as opposed to number of tasks (v). The total number of tasks is equal to $(m^2 + m - 2)/2$, in a Gaussian elimination graph. We appraise the performance of the algorithms at a range of matrix sizes that spans from 5 to 15. The size of the graphs in this experiment increases from a minimum of 14 tasks to the largest 119 tasks. The results of the simulation are presented in Figure 7, which shows that HSIP algorithm performs better than other reported algorithms in terms of average SLR and efficiency in various matrix sizes.

In FFT related experiments, because the application structure is established, only the CCR and range percentage parameters (η) are applied. In our experiments, the number of data points in FFT provides another parameter, which increases from 2 to 32 with incremental powers of 2. Figure 8(a) shows the average SLR values of the FFT graphs at a variety of sizes of input points. Figure 8(b) represents the efficiency values attained from each of the algorithms with regard to different numbers of processors with 32 data points graphs. The HSIP algorithm performs better than other reported algorithms.

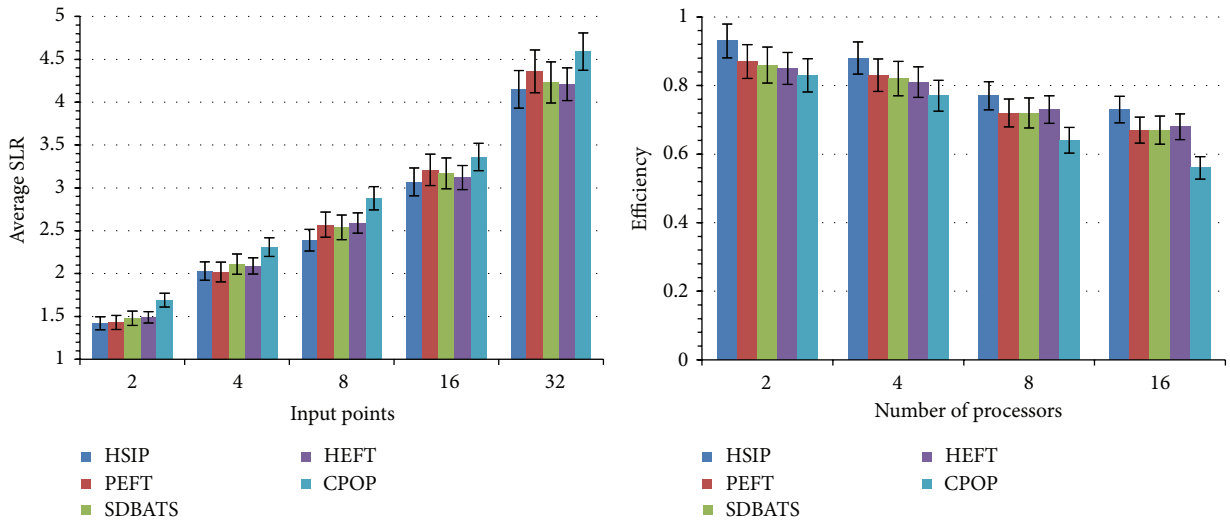
The Montage is used to construct an application of astronomical image mosaic in the sky. We use 25 and 50 task nodes to make experiments. Like other real applications, the application structure has been established, only using the CCR, CPU number, and range rate parameters (η). Figure 9 shows the experimental results under the different parameters of SLR. Our algorithm is still better than other algorithms.

Epigenomic is used to compare the genetic performance of genetic state of human cells in whole genome range. Like other real applications, the application structure has been established, only using the CCR, CPU number, and range rate parameters (η). In this experiment, we selected 24 and 46 task nodes. Figure 10 shows the experimental results under different parameters of the SLR. HSIP is still dominant position by comparison.

The standard deviations of all the above real-world problems' experimental errors are calculated, and they are in the 4–7% range.

6. Conclusions

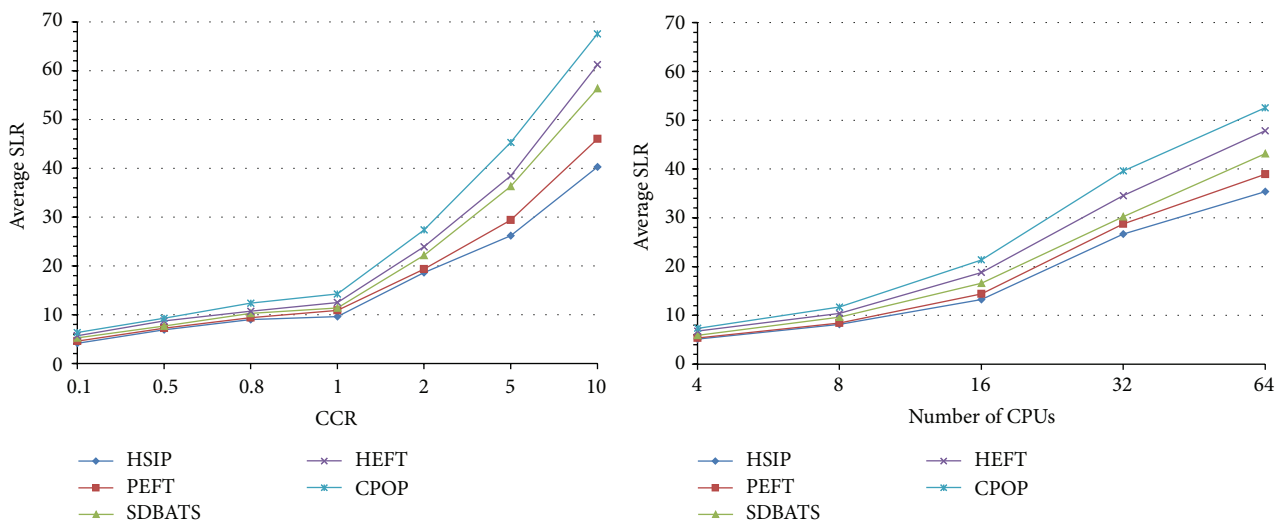
In this paper, we proposed a new list scheduling algorithm for heterogeneous systems named HSIP. The task scheduling algorithm proposed in this paper has demonstrated that the scheduling DAG structured applications performs better in heterogeneous computing system in respect of performance matrices (average schedule length ratio, speedup, efficiency, and frequency of best results). The performance of the HSIP



(a) Average SLR

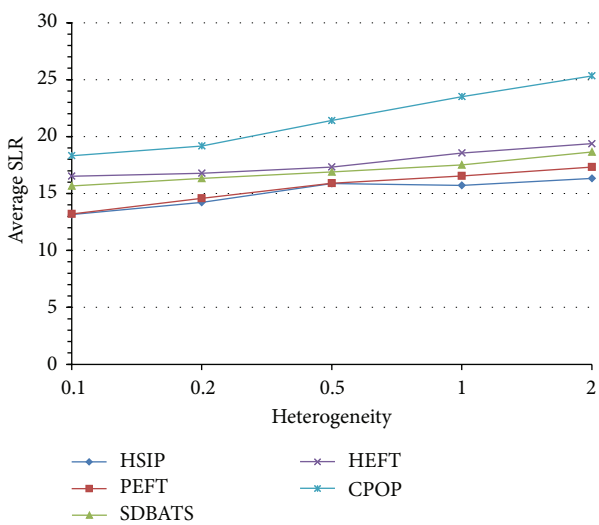
(b) Efficiency

Figure 8: Experimental result for FFT.



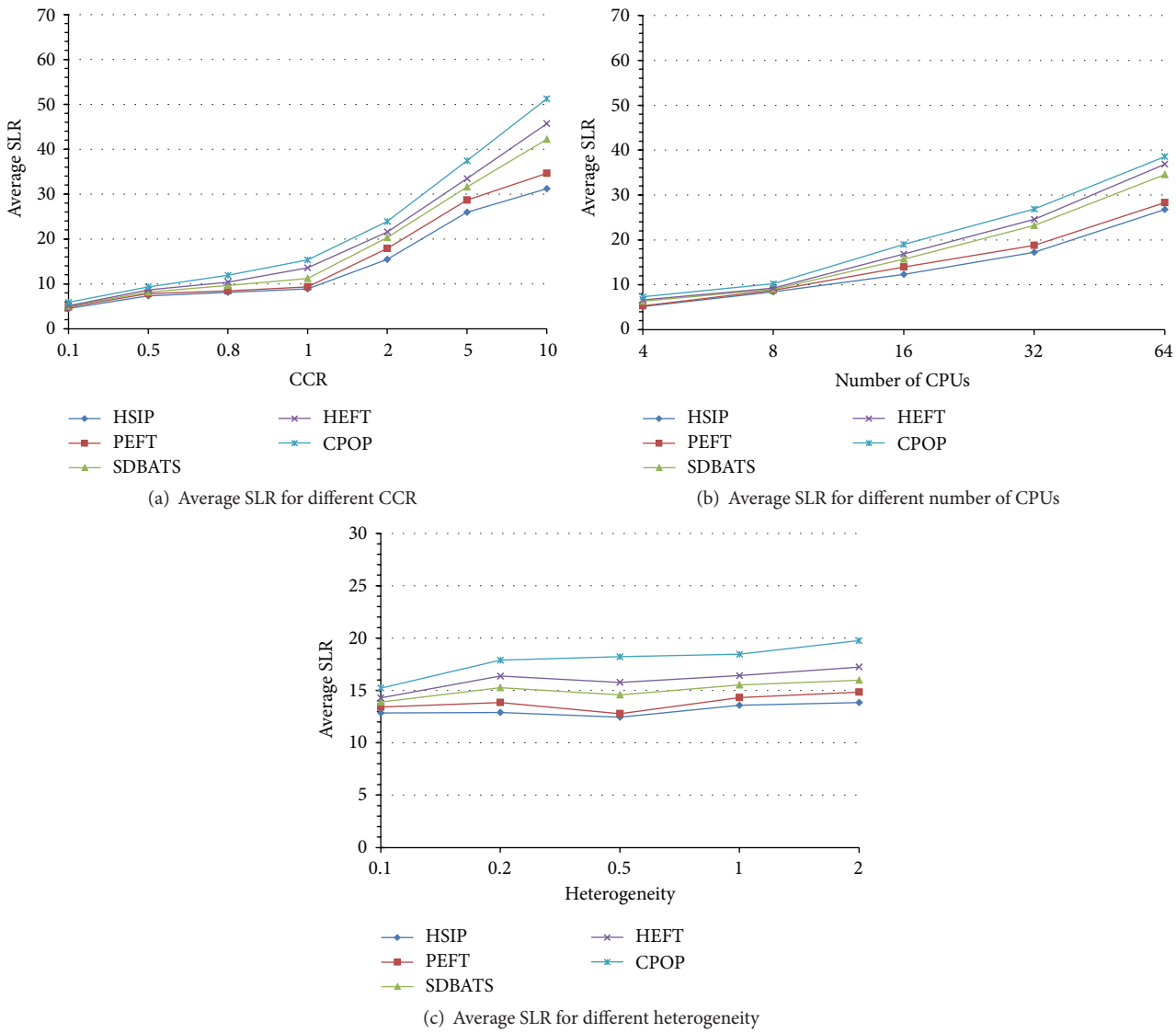
(a) Average SLR for different CCR

(b) Average SLR for different number of CPUs



(c) Average SLR for different heterogeneity

Figure 9: Experimental result for Montage.



(a) Average SLR for different CCR

(b) Average SLR for different number of CPUs

(c) Average SLR for different heterogeneity

Figure 10: Experimental result for Epigenomic.

algorithm has been experimentally observed by applying a large set of task graphs created randomly with various characteristics and application graphs of multiple real-world issues, such as Gaussian elimination, Fast Fourier Transformation, Montage, and Epigenomics. The simulation results backup the fact that HSIP algorithm is better than the existing algorithms, PEFT, SDBATS, CPOP, and HEFT, for instance. The complexity of HSIP algorithm is $O(v^2, p)$, which is the same time complexity in comparison with other scheduling algorithms illustrated in this paper.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This paper is partially supported by The National Natural Science Foundation of China under Grant no. 11372067 and The General Project of Science and Technology Research from Education Department of Liaoning Province (Grant no.: L2014508).

References

[1] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," in *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, Ed., vol. 8, pp. 679–690, John Wiley & Sons, New York, NY, USA, 1999.

[2] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing*, vol. 1291 of *Lecture Notes in Computer Science*, pp. 1–34, Springer, Berlin, Germany, 1997.

[3] Y. K. Kwok and I. Ahmad, "Benchmarking the task graph scheduling algorithms," in *Proceedings of the 1st Merged International and Symposium on Parallel and Distributed Processing, and IEEE Parallel Processing Symposium (IPPS/SPDP '98)*, pp. 531–537, March-April 1998.

[4] J. C. Liou and M. Palis, "A comparison of general approaches to multiprocessor scheduling," in *Proceedings of the 11th International Parallel Processing Symposium*, pp. 152–156, IEEE, Geneva, Switzerland, April 1997.

[5] T. Hagras and J. Janeček, "A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems," *Parallel Computing*, vol. 31, no. 7, pp. 653–670, 2005.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[7] E. U. Munir, S. Mohsin, A. Hussain, M. W. Nisar, and S. Ali, "SDBATS: a novel algorithm for task scheduling in heterogeneous computing systems," in *Proceedings of the IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW '13)*, pp. 43–53, IEEE, Cambridge, Mass, USA, May 2013.

[8] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014.

[9] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel & Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999.

[10] J.-K. Kim, S. Shivle, H. J. Siegel et al., "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment," *Journal of Parallel and Distributed Computing*, vol. 67, no. 2, pp. 154–169, 2007.

[11] W. Sun, Y. Zhang, and Y. Inoguchi, "Dynamic task flow scheduling for heterogeneous distributed computing: algorithm and strategy," *IEICE Transactions on Information and Systems*, vol. E90-D, no. 4, pp. 736–744, 2007.

[12] J. G. Barbosa and B. Moreira, "Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters," *Parallel Computing*, vol. 37, no. 8, pp. 428–438, 2011.

[13] E. S. H. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113–120, 1994.

[14] R. C. Correa, A. Ferreira, and P. Rebreyend, "Integrating list heuristics into genetic algorithms for multiprocessor scheduling," in *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, pp. 462–469, New Orleans, La, USA, October 1996.

[15] M. K. Dhodhi, I. Ahmad, A. Yatama, and I. Ahmad, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1338–1361, 2002.

[16] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 399–409, 2008.

[17] T. Hagras and J. Janecek, "A simple scheduling heuristic for heterogeneous computing environments," in *Proceedings of the Second International Conference on Parallel and Distributed Computing (ISPDC '03)*, pp. 104–110, IEEE, 2003.

[18] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, "High performance task scheduling algorithm for heterogeneous computing system," in *Distributed and Parallel Computing*, pp. 193–203, Springer, Berlin, Germany, 2005.

[19] E. Ilavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *Journal of Computer Science*, vol. 3, no. 2, pp. 94–103, 2007.

[20] X. Tang, K. Li, G. Liao, and R. Li, "List scheduling with duplication for heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 323–329, 2010.

[21] S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 6, pp. 533–544, 2003.

[22] P. F. Dutot, T. N'Takpé, F. Suter, and H. Casanova, "Scheduling parallel task graphs on (almost) homogeneous multicluster platforms," *IEEE Transactions on Parallel & Distributed Systems*, vol. 20, no. 7, pp. 940–952, 2009.

[23] A. K. Amoura, E. Bampis, and J.-C. König, "Scheduling algorithms for parallel Gaussian elimination with communication costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 7, pp. 679–686, 1998.

[24] Y. C. Chung and S. Ranka, "Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors," in *Proceedings of the ACM/IEEE Conference on Supercomputing (Supercomputing '92)*, pp. 512–521, IEEE, Minneapolis, Minn, USA, November 1992.

[25] Montage: An Astronomical Image Mosaic Engine, 2013, http://montage.ipac.caltech.edu/.

[26] USC Epigenome Center, 2013, http://epigenome.usc.edu/.