An Effective Task Scheduling Method with Link Contention Constraints for Heterogeneous System

Shuo-Zhan Ho, Yi-Hsuan Lee, Shun-Min Hsu, and Cheng Chen
Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
E-mail: {sjho, yslee, smhsu, cchen}@csie.nctu.edu.tw

Abstract

Sufficient and various computing resources make heterogeneous system suitable for parallel and distributed applications. A task scheduling problem is to arrange tasks of application on computing resources and aim to achieve minimum schedule length. Many effective scheduling algorithms have been proposed, but most of them assume that the network is fully-connected and contention free. In order to make this problem more practical, we take the link contention constraints into consideration. In this paper, we propose an effective and efficient *Communication Look-ahead Scheduling* (*CLS*) algorithm extended from list-scheduling algorithm. *CLS* contains two scheduling phases. In the first phase, we prioritize tasks with a new value which integrates both information from the concept of critical path and the communication behavior of each task. In the second phase, we propose a processor selecting mechanism with communication look-ahead manner to allocate tasks. According to our performance evaluations, *CLS* is superior to other algorithms both in effectiveness and efficiency.

Keywords: *Task scheduling, Heterogeneous system, Link contention*

1 Introduction

Heterogeneous system is a computing platform which consists of different kinds of resources interconnected with a high-speed network. Because of various and sufficient computing resources, a heterogeneous system requires compile-time and runtime support for executing parallel programs. Thus, an efficient task scheduling mechanism becomes one of the key factors for achieving high performance [2-4, 6, 8-15].

The task scheduling program includes two problems of assigning tasks to appropriate processors and ordering task executions on each processor. In general, an application is represented by a *Directed Acyclic Graph* (*DAG*) in which nodes and edges represent tasks and data dependencies, respectively. The scheduling goal is to find a schedule with minimum schedule length that satisfies precedence constraints [13].

Many related scheduling algorithms on heterogeneous system have been proposed [3-4, 6, 8-15]. However, most of them assume that the network is fully-connected (clique) and contention free. This assumption is unrealistic due to the expensive network cost. Therefore, in this paper, we take the link contention constraints into consideration to make the scheduling problem more practical [1].

Our *Communication Look-ahead Scheduling* (*CLS*) algorithm contains two phases doing task prioritizing and processor selecting respectively. In the first phase, we define a *weight* value for every task, which can use to prioritize tasks more suitably when link contentions are considered. In the second phase, we propose a processor selecting mechanism with communication look-ahead manner to allocate tasks. It considers not only the completion time of the task itself, but also message transferring costs that the task may occur. Based on our performance valuations, *CLS* can achieve effective results compared with an existed algorithm *Bubble Scheduling and Allocation* (*BSA*) in most cases. Moreover, *CLS* is much efficient than *BSA*, especially for larger input DAGs.

The paper is organized as follows. In section 2, we describe the problem and system architecture. Our proposed algorithm and theoretical analysis are introduced in section 3. Section 4 gives some simulation results. Finally, some conclusions and future work are listed in section 5.

2 Fundamental Background and Related Work

2.1 Fundamental Background on Task Scheduling [2-4, 6, 8-15]

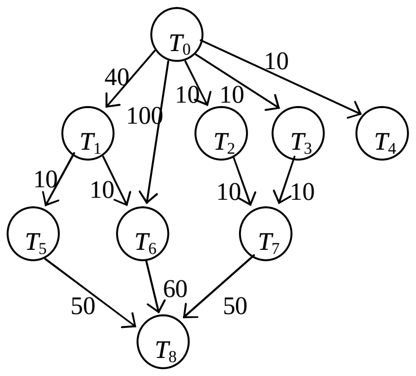
A *parallel program* is represented by a *Directed Acyclic Graph* (*DAG*) $G = (T, E, C)$, where T is the set of tasks, E is the set of dependencies, and C is a function from E to integer representing the communication cost. The *heterogeneous system* consists of m heterogeneous processors $P_1 \ldots P_m$, which connected in different network topologies and link contentions may happen due to the scarcity of communication resource [6, 12]. In order to represent different network topologies, an *interconnection network matrix* N is defined. If there exists a direct physical link between processors P_i and P_j, components n_{ij} and n_{ji} in N are both equal to one. Since a task on different processor has different computation cost in heterogeneous system, a *computation cost matrix* W is defined to record computation costs of all task-processor pairs. In this matrix, component w_{ij} indicates the computation cost of task T_i on processor P_j. Examples of DAG, network topology, and matrices N and W are shown in Figure 1.

We also assume that two tasks allocated to different processors communicate by message passing. That is, T_i sends message M_{ij} to T_j on network link L_{xy}, which is the communication resource of the routing path between processor P_x and P_y. Notice that the communication cost on the same processor is zero, and computation and communication can be overlapped.

In the following we define some terminologies. w_i is the average computation cost of task T_i. $EST(T_i, P_j)$ is the *Earliest Start Time* of task T_i on processor P_j defined as follows:

$$\begin{cases} EST(T_i, P_j) = 0 & \text{if task } T_i \text{ is the } \textit{entry} \text{ task} \\ EST(T_i, P_j) = \mathbf{max}(avail(j), \mathbf{max}_{T_m \in pred(T_i)}(AFT(T_m) + c_{mi})) & \text{otherwise} \end{cases}$$

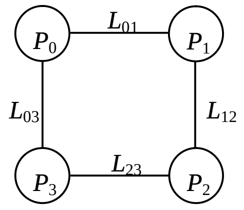
where $pred(T_i)$ is the set of immediate predecessors of T_i, $avail(j)$ is the available time of P_j, and $AFT(T_i)$ will be defined later.



(a)

$$W = \begin{bmatrix} 39 & 7 & 2 & 6 \\ 21 & 50 & 57 & 56 \\ 15 & 28 & 39 & 6 \\ 54 & 14 & 16 & 55 \\ 45 & 42 & 97 & 12 \\ 15 & 20 & 57 & 78 \\ 33 & 43 & 51 & 60 \\ 51 & 18 & 47 & 74 \\ 8 & 16 & 15 & 20 \end{bmatrix}$$

(b)



(c)

$$N = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

(d)

Figure 1. (a) DAG, (b) computation cost matrix, (c) network topology, (d) interconnection network matrix.

Next, *EFT*(T_i, P_j) is the *Earliest Finish Time* of task T_i on processor P_j, which is the summation of *EST*(T_i, P_j) and w_{ij}. After T_i is actually scheduled to P_j, its *Actual Start Time* and *Actual Finish Time* is represented by *AST*(T_i) and *AFT*(T_i), respectively. Finally, the schedule length, also called *makespan*, is equal to **max** (*AFT*(T_m)), where T_m is an *exit* task.

2.2 Related Work [6, 12]

Dynamic Level Scheduling (*DLS*) is a list scheduling algorithm which prioritizes tasks with *Dynamic Level DL*(T_i, P_j) [12]. *DL*(T_i, P_j) is defined as *SL*(T_i) – *ST*(T_i, P_j), where *Static Level SL*(T_i) is the maximum computation costs along a path from T_i to exit task, and *ST*(T_i, P_j) is the *Start Time* of T_i on P_j. After each scheduling step, it computes *DL* for every ready task, and the task with largest *DL* is scheduled. Unfortunately, the task with largest *DL* may not on the critical path, so *DLS* cannot always select the most important task in each scheduling step.

Bubble Scheduling and Allocation (*BSA*) is another list scheduling algorithm that is more effective than *DLS* [6]. It main mechanism is *task migration*, which moves tasks to neighbor processors to improve their finish time. *BSA* contains two phases. In the first phase, it *serializes* tasks according to the value of *bottom level* and topological order, where the *bottom level* of T_i is the maximum computation cost from T_i to exit task. In the second phase, it selects the *pivot processor*, which gives the minimum schedule length if all tasks are allocated to it. Then, every task is tried to migrate to neighbor processors in series if the *makespan* can be improved. After considering all tasks, another processor is chosen as new pivot processor and repeats task migration steps until all processors are considered.

task	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
weight	942	292	143	155	49	106	120	111	14

Figure 2. The *weight* values of tasks in Figure 1.

Although the *BSA* algorithm usually performs more effective than *DLS* algorithm, its time complexity is higher. Hence, in this paper, we propose a *Communication Look-ahead Scheduling* (*CLS*) algorithm to solve the task scheduling problem efficiently in heterogeneous system with link contention constraints.

3 Communication Look-ahead Scheduling (CLS) Algorithm

Based on our observation, *Heterogeneous Earliest-Finish-Time* (*HEFT*) is an effective and efficient task scheduling algorithm used in heterogeneous system [13]. Thus, in our *CLS*, we will extend its concepts to consider link contentions. Similar to many list scheduling algorithms, *CLS* contains two phases doing task prioritizing and processor selecting. Detailed scheduling steps will be described as follows.

3.1 Task Prioritizing Phase

In *HEFT* and many other task scheduling algorithms, *bottom level* is often used to prioritize tasks. Although bottom level also can consider communication overhead, they may not proper when we consider link contention in various network topologies. In *CLS*, we prioritize tasks by using the value *weight* which is defined below:

$$\begin{cases} weight\,(T_i) = w_i & \text{if task } T_i \text{ is the exit task} \\ weight\,(T_i) = w_i + \sum\limits_{T_j \in succ\,(T_i)} (c_{ij} + weight\,(T_j)) & \text{otherwise} \end{cases}$$

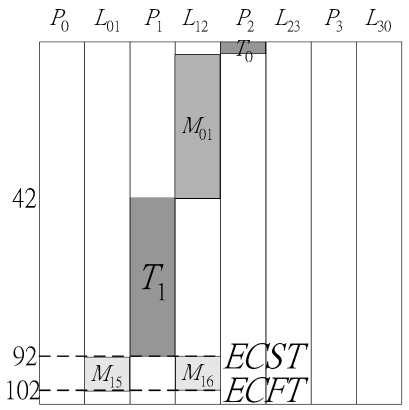
where $succ(T_i)$ is the set of immediate successors of task T_i.

For example, Figure 2 lists *weight* values of tasks in Figure 1. Different from *bottom level*, we sum all communication costs of a task into its *weight* value, instead of selecting only the maximum one. From above definition, a task with larger *weight* value indicates it may contain more successors, higher communication cost, or larger average computation cost. A task with any one of above feature should be scheduled earlier, especially when link contention constraints are considered. In Section 3.3 we will analyze the difference between *bottom level* and *weight* more detailed. Finally, tasks will be sorted in non-increasing order of *weight* values.

3.2 Processor Selecting Phase

Many task scheduling algorithm used on heterogeneous system suggest to allocate task to the processor which can complete it earliest, and their results have been proven quite effective. But once link contentions are considered, this selecting mechanism seems not precise enough, because the chosen processor may not contain sufficient communication resources to transfer necessary messages. Hence, in *CLS*, we propose a new processor selecting mechanism to avoid above situation.

In the beginning, we will introduce some terminologies at first. $DL_i = \{L_{ij} \mid i \neq j \text{ and } n_{ij} = 1\}$ is the set of direct links connected to processor P_i. $ECST(T_i, DL_j) = \mathbf{max}\,(avail(DL_j), EFT(T_i, P_j))$ is the *Estimative Communication Start Time* of task T_i on link set DL_j, where $avail(DL_j)$ returns



(a)

	P_0	P_1	P_2	P_3
EST	42	42	2	42
EFT	63	92	59	98
ECST	63	92	59	98
ECFT	73	102	69	108

(b)

Figure 3. (a) Partial schedule of DAG in Figure 1, (b) related values.

the earliest available time provided by any direct link in DL_j. Next, $ECFT(T_i, DL_j)$ is the *Estimative Communication Finish Time* of task T_i on link set DL_j, which equals to $ECST(T_i, DL_j) + d$, where d represents the time needed by task T_i to transfer all possible messages to its immediate successors.

If task T_i is allocated to processor P_j, all possible messages from T_i to its immediate successors should be arranged on DL_j. As for the message arrangement, in *CLS* we use the simplest *ASAP* (*As Soon As Possible*) mechanism, which will transfer messages as soon as any appropriate network link is available. We use the partial schedule of DAG in Figure 1 as an example. If T_0 and T_1 are allocated to P_2 and P_1 respectively, messages M_{01}, M_{15}, and M_{16} should be arranged as shown in Figure 3(a). Figure 3(b) lists related values if T_1 is allocated to other processors.

After calculating *ECST* and *ECFT* values of a task on all processors, this task is allocated to the processor with minimum *ECFT*. In above example, T_1 will be allocated to P_2. Notice that when we calculate *ECFT* values of task T_i, all possible messages from T_i are assumed to be transferred. Actually this assumption is unduly considered, because some transformations are unnecessary. However, this assumption can lead us to select the appropriate processor for T_i.

In summary, *CLS* contains both the concept of *EFT*, which is widely used in many task scheduling methods in heterogeneous system, and the extra consideration of limited communication resource. The entire *CLS* algorithm is listed in Figure 4, and in the next subsection we will analyze it theoretically.

3.3 Preliminary Analysis

At first, we discuss the difference between using *bottom level* and *weight* values to prioritize tasks when link contention is considered. Suppose there is a task T_i in the lower level of a DAG with small computation cost and large number of immediate successors. In *BSA* T_i will be scheduled very late, because its *bottom level* is small. Unfortunately the *makespan* may be lengthened, since all its successors must wait for its completion and transferring necessary messages. This situation is much serious when link contention is

```
Input: DAG and matrices $N$, $C$, and $W$
Output: Schedule result and makespan

1. Calculate weight values for all tasks in DAG
2. Sort tasks in non-increasing order of weight values
3. for (task $T_i$ with largest weight value)
    (a) Calculate $ECFT(T_i, DL_j)$ for every processor $P_j$
    (b) Allocate $T_i$ to $P_k$ with minimum $ECFT(T_i, DL_k)$
    (c) Delete $T_i$ from the task queue
4. Repeat step 3 until all tasks are scheduled
5. Output the schedule result and makespan
```

Figure 4. *Communication Look-ahead Scheduling* (*CLS*) algorithm.

considered. On the other hand, in *CLS* T_i can be scheduled earlier to avoid above situation, due to its larger *weight* value caused by lot of successors.

As for the processor selection, we use a look-ahead mechanism by prearranging all possible messages on direct links. This feature can help us select the beneficial processor, which will not postpone unscheduled tasks. Moreover, since that we consider the number of links connected to a processor when calculating *ECFT*, this processor selecting mechanism also can tolerate various network topologies.

The time complexity is derived as follows. Suppose that the given DAG contains n tasks and e edges and there are m processors in the system. In the task prioritizing phase, *weight* values can be computed by traversing the DAG, so its time complexity is $O(n+e)$. In the processor selecting phase, every task takes $O(me)$ to compute *ECST* and *ECFT*, so its time complexity is $O(men)$. Therefore, the entire time complexity of *CLS* is $O(mem)$. Obviously, our *CLS* algorithm is more efficient than that of *BSA* which complexity is $O(m^2en)$ [6].

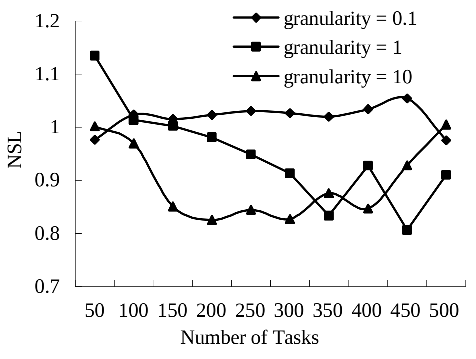
4 Performance Studies

4.1 Simulation Environment

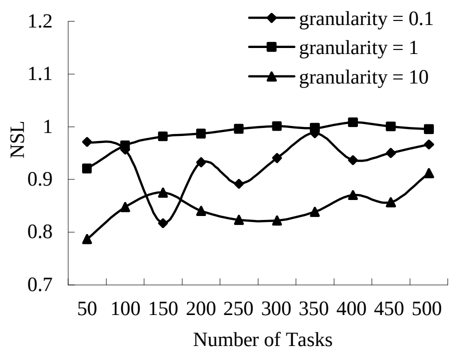
In this section, we construct a simulation and evaluation environment to evaluate *BSA* and *CLS*. Our environment contains three parts. The *Random Graph Generator* part is used to generate DAGs randomly with different number of tasks and *granularity*, which is defined as the average computation cost divided by the average communication cost in the DAG [6-7]. In our simulation we let the number of tasks between 50 and 500, and the *granularity* may be 0.1, 1, or 10. The *Network Topology* part is used to construct four network topologies containing 16 processors, including *Ring*, *Hypercube*, *Clique*, and *Mesh* [5]. Finally in the *Algorithm* part we implement algorithms *BSA* and *CLS*.

4.2 Experimental Results

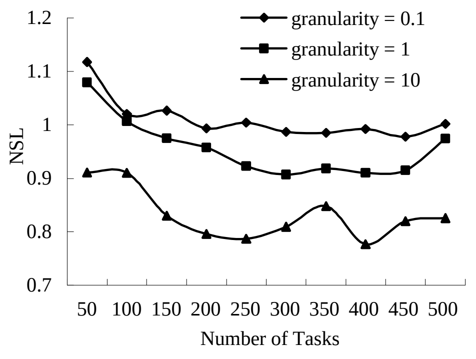
First at all, we define *Normalized Schedule Length* (*NSL*) as the schedule length of *CLS* divided by the schedule length of *BSA*. In other words, if *NSL* is less than one, the schedule obtained by *CLS* is shorter. Next, another value *speed* is defined as the scheduling



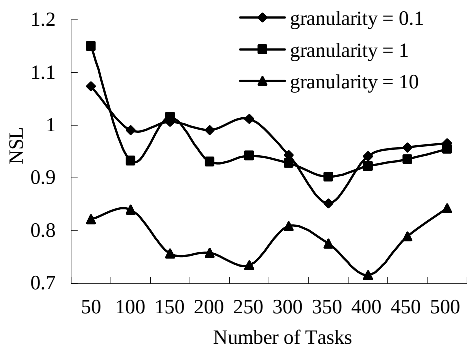
(a)



(b)



(c)



(d)

Figure 5. Simulation results. (a) Clique, (b) Ring, (c) Hypercube, (d) Mesh.

time of *CLS* divided by the scheduling time of *BSA*. That is to say, if *speed* is larger, *BSA* spends much more time doing scheduling.

In the following we describe experimental results in different network topologies. Figure 5(a) is simulation results on *Clique* network. This network topology offers sufficient communication resources, and their *NSL* are smaller than or nearly equal to one whether *granularities* are. Results on *Ring* network are illustrated in Figure 5(b). On the contrary, this network topology has limited communication resources. In the figure, obviously all *NSL* are also smaller than or equal to one. From above observations, it shows that *CLS* has effective performance on two extreme network topologies with respect to the communication resource. Figures 5(c) and 5(d) are simulation results on *Hypercube* and *Mesh* networks, respectively. We can see that *CLS* still outperforms than that of *BSA* in most case.

Then, we analyze simulation results in different *granularities*. From Figure 5, it is clear that *CLS* performs much effectively than that of *BSA* for coarse-grain DAGs (*granularity* = 10). For these computation-dominated DAGs, the performance of task serialization becomes more important. In previous section we have mentioned that prioritizes tasks according to their *weight* values is much suitable, so our *CLS* can outperform *BSA*. On the other hand, performances of *CLS* and *BSA* are similar for

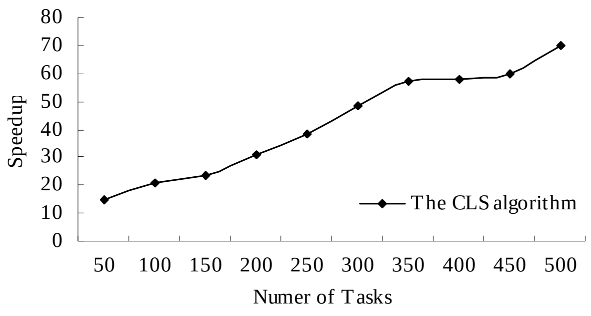


Figure 6. Simulation results.

fine-grain DAGs (*granularity* = 0.1). For these DAGs with heavier communication costs, the arranging of messages becomes critical. Intuitively *CLS* should also achieve better results than that of *BSA* since it contains communication look-ahead mechanism, but actually their performances are similar. We think the reason is that our communication look-ahead mechanism is not accurate enough, because we simply assume that all possible messages are need to be transferred. Hence, the processor selected for every task is not the most appropriate. We remain this problem to be solved in our future work.

Finally, we discuss the efficiency of *CLS* algorithm. Based on previous complexity analysis, *CLS* runs at least *m* times faster than *BSA*, where *m* is the number of processors. In Figure 6, we find that speedup increases with the number of tasks increasing, which means that *CLS* is much efficient in larger DAGs.

5 Conclusions and Future Work

In this paper, we propose a *Communication Look-ahead Scheduling* (*CLS*) algorithm to schedule tasks on heterogeneous system considering link contention constraints. It contains a new *weight* value to effectively prioritize tasks, and allocates tasks by a mechanism with communication look-ahead manner. According to our theoretical analysis and simulation results, *CLS* is not only effective but also efficient than *BSA* in most cases.

There is still a promising issue for future research. As mentioned in previous section, *CLS* cannot achieve expected results for DAGs with smaller *granularity*. This is because we simply assume all possible messages from a task will be transferred, but actually some on them are unnecessary. This unduly assumption may mislead the processor selecting result. In the future, we can design a mechanism to predict messages that are really need to be transferred. After integrating this mechanism into the processor selecting phase, we believe performances of *CLS* can be further improved.

References

[1] Donglai Dai and Dhabaleswar K. Panda, “How Much Does Network Contention Affect Distributed Shared Memory Performance”, *Proc. of International Conference on Parallel Processing*, pp. 454-461, 1997.

[2] I. Ekmecic, I. Tartalja, and V. Milutinovic, “A Survey of Heterogeneous Computing:

Concepts and Systems", *Proc. of IEEE*, Vol. 84, pp. 1127-1144, Aug. 1996.

[3] T.S. Hsu, Joseph C. Lee, Dian Rae Lpoez, and William A. Royce, "Task Allocation on a Network of Processors", *IEEE Transactions on Computers*, Vol. 49, No. 12, pp.1339-1353, Dec. 2000.

[4] C.C. Hui and Samuel T. Chanson, "Allocation Task Interaction Graphs to Processors in Heterogeneous Networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol.8, No. 9, pp. 908-925, Sep. 1997.

[5] Kai Hwang and Zhiwei Xu, **Scalable Parallel Computing**, Mc Graw-Hill Book Company, 1998.

[6] Yu-Kwong Kwok and Ishfaq Ahmad, **"**Link Contention-constrained Scheduling and Mapping of Tasks and Messages to a Network of Heterogeneous Processors", *Proc. of International Conference on Parallel Processing*, pp. 551-558, 1999.

[7] Yu-Kwong Kwok and Ishfaq Ahmad, "Benchmarking the Task Graph Scheduling Algorithm," *Proc. of Parallel and Distributed Processing Symposium*, pp. 531-537, 1998.

[8] Yu-Kwong Kwok and Ishfaq Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 506-521, May 1996.

[9] Yu-Kwong Kwok, Ishfaq Ahmad, and J. Gu, "FAST: A Low-Complexity Algorithm for Efficient Scheduling of DAGs on Parallel Processors", *Proc. of International Conference on Parallel Processing*, Vol. 2, pp. 150-157, 1996.

[10]J. Liou and M.A Palis, "An Efficient Clustering Heuristic for Scheduling DAGs on Multiprocessors", *Proc. of Parallel and Distributed Processing Symposium*, 1996.

[11]P. Shroff, D.W. Watson, N.S, Flann, and R.Freund, "Genetic Simulated Annealing for Scheduling Data-Dependent Tasks in Heterogeneous Environments", *Proc. of Heterogeneous Computing Workshop*, pp. 98-104, 1996.

[12]G. C. Sih and E. A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No.2, pp. 175-186, Feb. 1993.

[13]H. Topcuoglu, S. Hariri, and Min-You 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, March 2002.

[14]M. Wu, W. Shu, and J. Gu, "Efficient Local Search for DAG Scheduling", *IEEE Transactions on Parallel and Distributed System*, Vol. 12, No. 6, pp. 617-627, June 2001.

[15]T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number Processors", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 9, pp.951-967, Sep. 1994.