

Chapter 1. Introduction

A *Heterogeneous Computing* (HC) system is a geographically distributed machines interconnected by a high-speed network topology. It offers high-speed computation of parallel program (application) with diverse computing needs [1,2,6,17-19,24-29,36,38]. It is envisioned that such a computing system will enable users to execute their applications rapidly on the computing resources. Applications like weather modeling, image processing, distributed database systems show a great deal of parallelism [26,27]. Owing to the technical progressing of *VLSI*, the computation speed of processor increases fast and makes cost down of processor. Therefore, users can use the server consisting of many processors or the computing system constructed by many personal computers (PCs) to execute their applications. It will be used popularly in the future.

In the HC system, one of the most important aims is how to use the processors efficiently to achieve optimal task parallelism. This problem is called task scheduling problem [1,2,6,17-19,24-29,36,38]. Thus, an efficient task scheduling method assigning the tasks of parallel program to the suitable processor is one of the key factors for achieving high performance of a HC system [1,2,6,17-19,24-29,36,38]. The general task scheduling problem includes the problem of assigning the tasks of a parallel program to the suitable processor and the problem of ordering task executions on each processor. When the characteristics of a parallel program which includes computation cost of the tasks, the communication cost between the tasks, and the precedence relation of the tasks are known a priori, it is called *static* model [2].

Resolving the task scheduling problem on the static model is called static task scheduling method [2]. In the general form of a static task scheduling method, an application is represented by a *Directed Acyclic Graph* (DAG) in which nodes

represent the tasks and edges represent the data dependencies between tasks in the parallel program. The objective function of task scheduling method is to assign tasks onto processors and order their executions so that task-precedence requirements are satisfied and a minimum completion time is obtained [2].

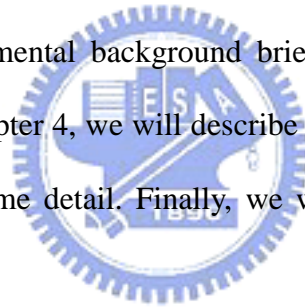
The task scheduling problem is a NP-complete problem [3]. There are many scheduling categories on this problem, including list-based scheduling algorithm [4-9,36], clustering algorithm [4, 10-12], duplication-based algorithm [8, 13-14, 26-35], and guided random search algorithm [15-21]...,etc. The main idea behind duplication-based scheduling algorithms is to schedule a task graph by mapping some of its tasks redundantly, which avoids the inter-processor communication overhead to achieve the goal of reducing completion time of parallel program. The main difference among duplication-based algorithms is the selection strategy of the tasks for duplication. Although scheduling method of this kind usually has higher time complexity than the algorithms in the other categories, it will be a very important factor no more because we can use a fast processor for scheduling tasks.

We find that most task scheduling algorithms on HC system assume the system model connected by the *fully-connected (clique)* network. Moreover, contention for network link is neglected. Very few algorithms model the target system as an arbitrary network of processor and incorporate network link contention. However, Macey and Zomaya showed that the consideration of link contention is significant to produce accurate and efficient schedules [37]. Actually, there are some related works on the task scheduling problem with the link contention constraints [24-25,36,38]. We will construct a convincing and practical system model from previous work.

In this thesis, we propose a *Duplication-based Earliest Finish Time (DEFT)* algorithm to solve the scheduling problem. This algorithm contains two phases. The

first phase is task prioritizing phase for computing the priorities of all tasks by an efficient priority function. In the second phase, we select the processor which can complete the task earliest for the task by a task duplication mechanism. The concept of task duplication mechanism is that we utilize processor idling time for duplicating some predecessors of scheduling task into a processor to avoid communication costs between tasks. We design two similar algorithms. The one is called DEFT₁ that is for target system without link contention constraints, another is called DEFT₂ that is for target system with link contention constraints. Meanwhile, we construct a simulation environment. In our simulation, we find that in most cases of our simulation results, the DEFT algorithm performs more effectively than the related algorithms.

The thesis is organized as follows. In chapter 2, we will survey some related work and some basic fundamental background briefly. Our proposed algorithm is described in chapter 3. In chapter 4, we will describe our simulation environment and evaluate our algorithm in some detail. Finally, we will make conclusion and some future work in chapter 5.

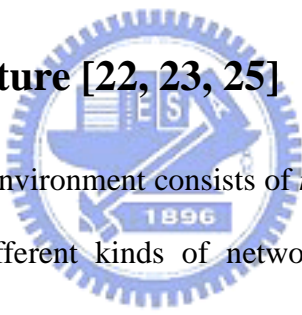


Chapter 2. Fundamental Background and Related Work

In this chapter, we will introduce the system architecture and some basic terminologies in the section 2.1. Next, we will describe some related algorithms that have different system assumptions to solve the same task scheduling problem in the section 2.2. The main different system assumption among related algorithms is that target system exists link contention or target system doesn't exist link contention.

2.1 Fundamental Background

2.1.1 System Architecture [22, 23, 25]



We assume that our HC environment consists of m heterogeneous processors $\{P_1, P_2, \dots, P_m\}$ connected in different kinds of network topologies, such as *clique*, *hypercube*, *mesh*, *ring*, ..., etc. In this HC system, the inter-processor link contention may happen due to the scarcity of network link.

We use the message passing mechanism to transmit the data on the network link. The data transmission needs to be handled in each network topology. Thus, we need the routing table in each network topology. We choose the pre-determined routing table which uses the shortest-routing-path algorithm [39] (such as a hypercube uses the *E-cube* routing method and a mesh uses the *XY-routing* method) for any kind of input network topology in our system. For simplicity, we assume that the distance between two processors doesn't affect the communication cost. Under considering the condition of link contention, the system only allows one direction of data transmission on each link between processors at the same time.

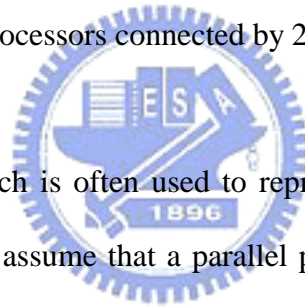
2.1.2 Some Basic Terminologies [2, 36]

In this thesis, our object is to solve the problem of task scheduling with link contention constraints on different kinds of network topologies. Thus, we define the network topology first in Definition 2.1.

Definition 2.1 The topology of the target system is modeled as an undirected graph $G_T=(P, L)$, where

- P is a finite set of $|P|$ vertices and L is a finite set of $|L|$ undirected edges;
- A vertex P_i represents the processor i . And an undirected edge L_{ij} represents a bi-directional communication link between the incident processors P_i and P_j ;

In Figure 2.1, there are four processors connected by 2-D mesh network.



We will define DAG which is often used to represent a parallel program in the task scheduling problem. We assume that a parallel program is composed of n tasks $\{T_1, T_2, \dots, T_n\}$ in which there is a partial order: $T_i < T_j$ implies that T_j cannot start execution until T_i finishes due to the data dependency between them. Formally, we give the following definition.

Definition 2.2 A parallel program can be represented by a *directed acyclic graph* (DAG) $G, G = (T, E, C)$, where

- T is a finite set of $|T|$ vertices and E is a finite set of $|E|$ directed edges;
- A vertex T_i represents the task i . And a directed edge $e_{ij} \in E$ represents a directional data dependency between task T_i and task T_j ;
- C is the function from E to integer in which c_{ij} represents the communication

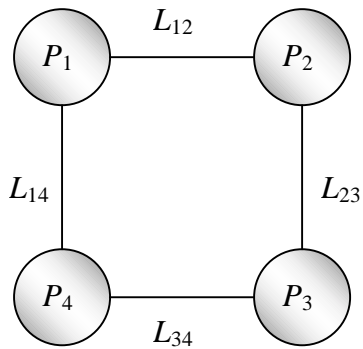


Figure 2.1 2-D mesh network topology

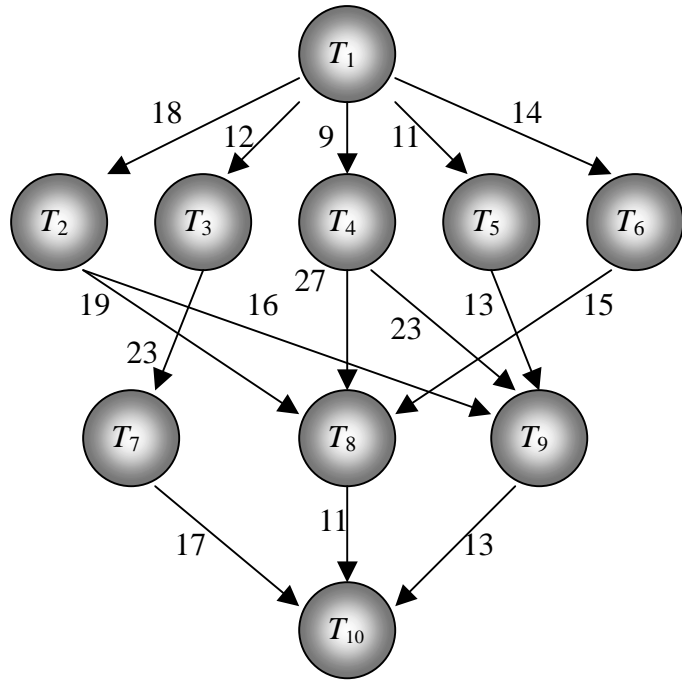


Figure 2.2 A DAG with 10 tasks

$$W = \begin{bmatrix} 14 & 16 & 9 & 2 \\ 13 & 19 & 18 & 3 \\ 11 & 13 & 19 & 21 \\ 13 & 8 & 17 & 24 \\ 12 & 13 & 10 & 40 \\ 13 & 16 & 9 & 3 \\ 7 & 15 & 11 & 28 \\ 5 & 11 & 14 & 8 \\ 18 & 12 & 20 & 15 \\ 21 & 7 & 16 & 8 \end{bmatrix}$$

Figure 2.3 Computation cost matrix

cost from task T_i to task T_j ;

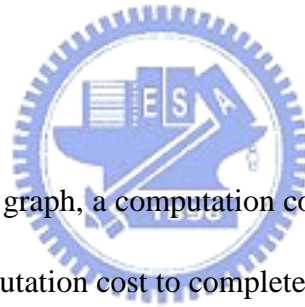
Figure 2.2 shows an example of DAG.

When both T_i and T_j are scheduled on the same processor, c_{ij} becomes zero since

the intra-processor communication cost is negligible when it is compared with the inter-processor communication cost. After introducing about the definition of DAG, we explain some terminology in DAG as follows.

In the DAG, a task without any parent is called an *entry task* and a task without any child is called an *exit task*. If there exists the data dependency from task T_i to task T_j in the DAG, we say that task T_i is the *immediate predecessor* of task T_j and task T_j is the *immediate successor* of task T_i .

In HC system, a task on different processors has different computation costs. We assume that computation can be overlapped with communication. Additionally, computation and communication are both non-preemptive. We need a computation cost matrix W to describe the computation cost. The definition of matrix W is shown as follows.



Definition 2.3 In a given task graph, a computation cost matrix W is a $n \times m$ matrix in which each w_{ij} gives the computation cost to complete task T_i on processor P_j .

For example, Figure 2.3 shows the corresponding computation cost matrix related to the DAG in Figure 2.2 when there are four processors in our system.

Before scheduling, we often label the tasks with the average computation cost. The definition is shown below.

Definition 2.4 In a given task graph, the average computation cost of task T_i is defined as

$$\overline{w}_i = \sum_{j=1}^m w_{ij} / m. \quad (1)$$

,where w_{ij} is the computation cost while task T_i is allocated on processor P_j .

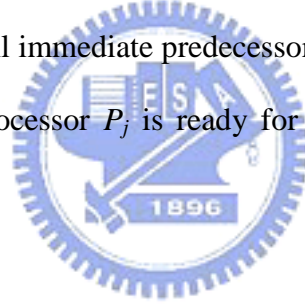
For example, the average computation cost of task T_1 in Figure 2.2 is 10.25.

Next, we will define two attributes of EST (*Earliest Start Time*) and EFT (*Earliest Finish Time*) below.

Definition 2.5 In a given partial schedule, we define the *Earliest Start Time* of task T_i on processor P_j , denoted as $EST(T_i, P_j)$, by the following equation:

$$\left\{ \begin{array}{l} EST(T_i, P_j) = 0, \text{ if task } T_i \text{ is the entry task;} \\ EST(T_i, P_j) = \text{Max}\{\text{avail}[P_j], \text{Max}_{T_m \in \text{pred}(T_i)} (AFT(T_m) + c_{mi})\}, \text{ otherwise.} \end{array} \right. \quad (2)$$

,where $\text{pred}(T_i)$ is the set of all immediate predecessor tasks of task T_i , and $\text{avail}[P_j]$ is the earliest time at which processor P_j is ready for task execution. $AFT(T_m)$ is the actual finish time of a task T_m .



Definition 2.6 In a given partial schedule, we define the *Earliest Finish Time* of task T_i on processor P_j , denoted as $EFT(T_i, P_j)$, by the following equation:

$$EFT(T_i, P_j) = w_{ij} + EST(T_i, P_j), \quad (3)$$

In Eq. (2), the inner *Max* block returns the data ready time, i.e., the time when all data needed by task T_i has arrived at processor. After task T_m is scheduled on processors P_j , the EST and EFT of T_m on processor P_j is equal to the *Actual Start Time*, $AST(T_m)$, and the *Actual Finish Time*, $AFT(T_m)$, of task T_m , respectively.

We need one *object function* to quantify the final schedule after all tasks have been scheduled.

Definition 2.7 In a given final schedule, the schedule length (which is also called *makespan*) is defined as

$$makespan = Max\{AFT(T_{exit})\} \quad (4)$$

,where $AFT(T_{exit})$ is the *actual finish time* of the *exit task*.

Finally, we define a common task priority function. Tasks are ordered in our algorithm by their scheduling priorities that are based on *bottem_level*(*b_level*) value. The *b_level* of a task T_i is defined as follows.

Definition 2.8 In a given task graph, the *b_level* value of a task T_i is recursively defined as

$$\begin{cases} b_level(T_i) = \overline{w}_i, & \text{if task } T_i \text{ is the exit task;} \\ b_level(T_i) = \overline{w}_i + \underset{T_j \in succ(T_i)}{Max}(c_{ij} + b_level(T_j)), & \text{otherwise.} \end{cases} \quad (5)$$

,where $succ(T_i)$ is the set of all immediate successor tasks of task T_i , and \overline{w}_i is the average computation cost of task T_i . c_{ij} is the communication cost from task T_i to task T_j .

The goal of the task-scheduling problem is to determine the assignment of tasks of a given parallel program on processors such that its schedule length is minimized.

2.2 Related Work

We will survey four related algorithms, named HEFT (*Heterogeneous Earliest-Finish-Time*), STDS (*Scalable Task Duplication-based Scheduling*), CLS (*Communication Look-ahead Scheduling*), and BSA (*Bubble Scheduling and*

Allocation) in this section briefly. There are some different system assumptions among them. The HEFT and STDS algorithm have the same system assumptions of target system to be a HC with clique network. Moreover, contention for network link is neglected. We will describe the HEFT, STDS, CLS and BSA algorithm in section 2.2.1~ section 2.2.4, respectively.

2.2.1 HEFT Algorithm [2]

HEFT algorithm is a well-known and effective list-based algorithm. It is a traditional task scheduling method without considering the task duplication. The main concept of this algorithm is to choose the processor which can complete the task earliest. This algorithm has two major phases. The first phase is task prioritizing phase for computing the priorities of all tasks. In this phase, it sets the task priority by $rank_u$ value (which is b_level value as we mentioned before). The $rank_u$ of a task is the length of the longest path from its task to exit task, including the computation cost of the task. The higher $rank_u$ value of a task represents that it needs more computation cost or communication cost from its task to exit task. If such kind of task can select the processor early, it is useful to reduce the schedule length.

The second phase is processor selection phase for selecting the tasks in the order of their priorities and scheduling each selected task on the processor, which minimizes the task's finish time. In this phase, the selected task according to the $rank_u$ value is assigned to the processor which minimizes its earliest finish time.

HEFT algorithm has an $O(n^2m)$ time complexity for n tasks and m processors. After analyzing the HEFT algorithm, we find that there are many idle time-slots in the processors when the parallel program is communication-intensive. It will cause the schedule length increasing.

2.2.2 STDS Algorithm [27]

This algorithm uses task duplication to reduce the length of the schedule. First, it generates initial clusters by traversing the DAG. If the number of processors in the system is more than the number of initial clusters generated, task duplication is carried out. The task duplication process is that it checks clusters to see whether the preceding task of a given task is the task of critical path of its given task. If this is not the case, it duplicates tasks of the critical path of given task and then reassigns those tasks on the origin processor of the cluster in order to improve the finish time of its given task. The remainder tasks in the cluster are assigned to a new processor. This process continues until the task duplication makes the final schedule length increasing or there is no free processor in the system. On the contrary, if the number of initial clusters is larger than the number of processors in the system, it will merge initial clusters together until the number of clusters is equal to the number of processors in the system.

The STDS has an $O(n^2)$ time complexity, where n is the number of tasks. After analyzing the STDS algorithm, we find some shortcomings in the algorithm. First is that each processor executes tasks of a critical path. It means that only one immediate predecessor of given task in the critical path assigns to the same processor of given task. The other immediate predecessor tasks of given task are assigned to different processors. It causes that the communication overhead between immediate predecessor tasks of given task and its given task is very heavy and makes a poor schedule result when the parallel program is communication-intensive. The second is that if the number of initial clusters is larger than the number of processors in the system, the task duplication process will be limited. Thus, it also makes an inferior schedule result.

2.2.3 CLS Algorithm [36]

The CLS algorithm is also a list-based algorithm. This algorithm has to consider the link contention condition that occurs in a practical system. The CLS is an extended algorithm from the HEFT. The main concept of the CLS algorithm is to choose the processor which can complete the task early and has the sufficient network link by communication look-ahead manner because it takes the link contention into account. Thus, this algorithm also contains two phases doing task prioritizing and processor selecting respectively.

In the first phase, different from $rank_u$ in the HEFT, it sums all communication costs of a task into its *weight* value. Thus, a task with larger *weight* value indicates it may contain more immediate successor tasks, higher communication cost between its task and immediate successor task of its task, or larger average computation cost of its task. If such kind of task can select the processor early, it is useful to reduce the schedule length under considering the link contention constraints. In the second phase, it selects the processor which can complete the task early and has the sufficient network link by computing the earliest finish time of assigning the communication cost that forwards to immediate successor on direct link of its processor. The look-ahead mechanism can help that the immediate successor tasks of its task receive the data early.

The CLS algorithm has an $O(nme)$ time complexity for n tasks, m processors and e edges. In this algorithm, there may have some conditions to make inferior performance. First, it can produce inaccurate look-ahead for immediate successors when the data doesn't need transmission by network link to immediate successors. The second is that CLS algorithm still has many idle time-slots in the processors when the parallel program is communication-intensive.

2.2.4 BSA Algorithm [24]

This algorithm also has to consider the link contention constraints. The main concept of the BSA algorithm is *task migration*. The tasks have to be considered for possible migration to the neighbor processors in order to improve their finish time. The BSA algorithm belongs to the list-based algorithm which always contains two steps: task prioritizing and processor selection. Before introducing the main body of the BSA algorithm, we have to explain some terminology. First, the *top level* of task T_i represents the length of the longest path from the entry task to task T_i . Next, the *bottom level* of task T_i represents the length of the longest path from the task T_i to exit task. Finally, the critical path is defined as a path on the given task graph with the largest sum of computation costs and communication costs. After computing *top level* and *bottom level*, the set of tasks with the largest sum of *top level* and *bottom level* is the critical path in a given task graph. The first step is *serialization* which gives the tasks priority according to the value of *bottom level* and the topological order. The second step is to select the *pivot processor* which gives the minimum critical path length and builds the *processor list* with the other processors. In this step, the BSA algorithm chooses the processor by using the concept of task migration mentioned above. After all tasks have been scheduled, it removes the current pivot processor from processor list and reassigns the processor in processor list as the new pivot processor. The algorithm repeats the step of processor selection until the processor list is empty.

The time complexity of task migration is $O(e)$. Since there are $O(n)$ tasks on pivot processor and $O(m)$ neighbor processors, each iteration of processor selection tasks $O(men)$ time. Thus, the BSA algorithm tasks $O(nm^2e)$ time for n tasks, m processors and e edges.

Algorithm	HEFT	STDS	CLS	BSA
Scheduling categories	List-based	Duplication-based	List-based	List-based
Link contention constraints	No	No	Yes	Yes
Network of system	Clique network	Clique network	Arbitrary network	Arbitrary network
Main feature	<i>EFT</i> concept	Duplication of task in a path	Look-ahead mechanism	Task migration
Time complexity	$O(n^2 m)$	$O(n^2)$	$O(nme)$	$O(nm^2 e)$

Table 2.1 Characteristics of related algorithms

Table 2.1 summarizes the characteristics of four related algorithms. After introducing about four related algorithms, we will propose an effective duplication-based algorithm in HC system with link contention constraints in the next chapter.



Chapter 3. Duplication-based Earliest Finish Time Algorithm

As we discussed in section 2.2, the STDS algorithm isn't effective enough. The selection strategy of tasks for duplication is one of the most important factors to affect the performance of a duplication method. We also consider that it is useful to use a better task duplication mechanism to duplicate some predecessor tasks on the processor to avoid the communication overhead between tasks and then reduce the schedule length. Thus, we will propose the *Duplication-based Earliest Finish Time* (DEFT) algorithm in this chapter. Our algorithm contains two phases that are task prioritizing phase and processor selection phase. We will design two similar algorithms. The one is called DEFT₁ that is for target system without link contention constraints, another is called DEFT₂ that is for target system with link contention constraints. We will describe DEFT₁ algorithm and DEFT₂ algorithm in section 3.1 and section 3.2, respectively.

3.1 DEFT₁ Algorithm

In section 3.1.1, we will describe the task prioritizing phase that sets the priority of each task by computing b_level value. Next, we will explain our processor selection mechanism and task duplication method in section 3.1.2.

3.1.1 Task Prioritizing Phase

In some effective two phases list-based algorithms, the b_level priority function is often used to set the priority of each task. Moreover, it is also compared with

T_i	$b_level(T_i)$
T_1	106.5
T_2	71.5
T_3	84.2
T_4	80.7
T_5	74.0
T_6	58.7
T_7	45.2
T_8	33.5
T_9	42.2
T_{10}	13.0

Table 3.1 The b_level value of each task in Figure 2.2.

different priority functions in [38] and shows a better result, whether the target system exists link contention constraints or not. Thus, we also use the b_level priority function in this phase.

The b_level value was defined in definition 2.8, and this value can be computed recursively from the *exit task*. The $b_level(T_i)$ value is the length of the critical path from task T_i to the exit task. Obviously, the higher b_level value of a task represents that it needs more computation cost or communication cost from its task to completion of the parallel program. If such kind of task can be assigned to processor early, it is useful to reduce the schedule length. As an example, Table 3.1 represents the b_level value of each task in Figure 2.2. After computing b_level values of all tasks, we sort tasks in the scheduling list according to the b_level value in nonincreasing order, that is $\{ T_1, T_3, T_4, T_5, T_2, T_6, T_7, T_9, T_8, T_{10} \}$.

3.1.2 Processor Selection Phase

In this phase, the concept of our task duplication process is to utilize processor idling time for duplicating some predecessors of scheduling task into a processor. It can avoid communication costs between tasks. We also use the concept of *EFT* that is broadly used in scheduling problem in this phase. First, we define a terminology *cluster* in our algorithm.

Definition 3.1 For each task T_i in a DAG, a *cluster* $C(T_i)$ represents T_i itself and some predecessors of task T_i that are duplicated to the processor which has minimum *EFT* of T_i .

A simple example is shown in Figure 3.1. There exists the data dependency between task T_j and task T_i , and task T_k and task T_i as shown in DAG. Task T_j and task T_k are allocated on different processors P_1 and P_3 , respectively. We can find that the task T_j and task T_k haven't any predecessor. Thus, cluster $C(T_j)$ contains only task T_j and cluster $C(T_k)$ contains only task T_k . Next, we will try to assign task T_i on each processor in order to select an appropriate processor that has minimum *EFT* of task T_i for task T_i execution. For example, when we try to assign task T_i on the P_2 , the partial schedule is shown in Figure 3.1(a). In the P_2 , we also consider to duplicate the tasks in the $C(T_j)$ and $C(T_k)$ sequentially into this processor in order to reduce the *EFT* of task T_i . We can see Figure 3.1(b) that shows a shorter *EFT* value of task T_i than the result in Figure 3.1(a) when we duplicate task T_j in the cluster $C(T_j)$ and task T_k in the cluster $C(T_k)$ into P_2 . If task T_i finally is scheduled on processor P_2 that has minimum *EFT* of task T_i (it likes the partial schedule in Figure 3.1(b).) comparing with assigning T_i to other processors, the cluster $C(T_i)$ is $\{ T_j, T_k, T_i \}$.

After computing *b_level* value of each task in the first phase, the task scheduling

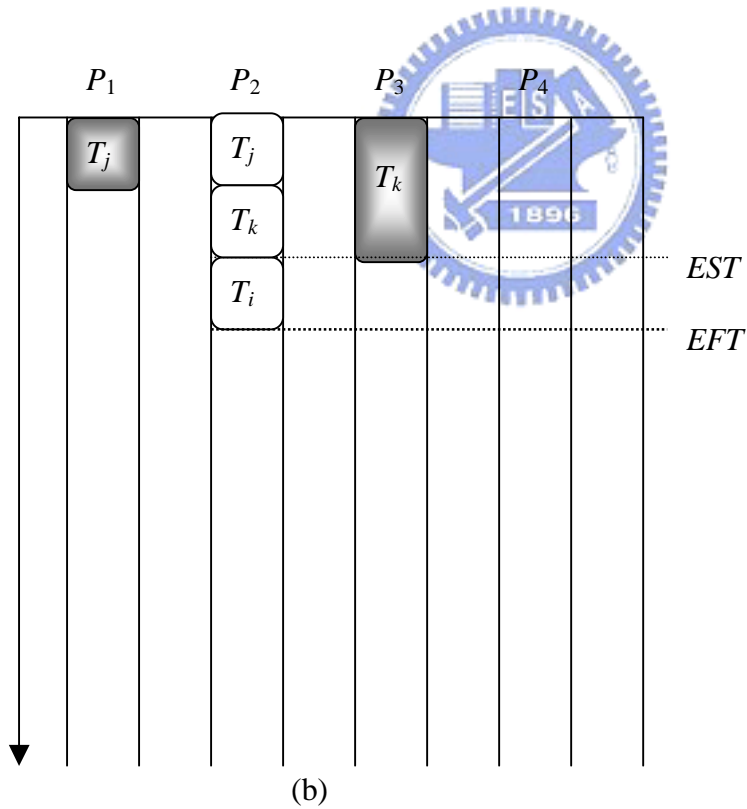
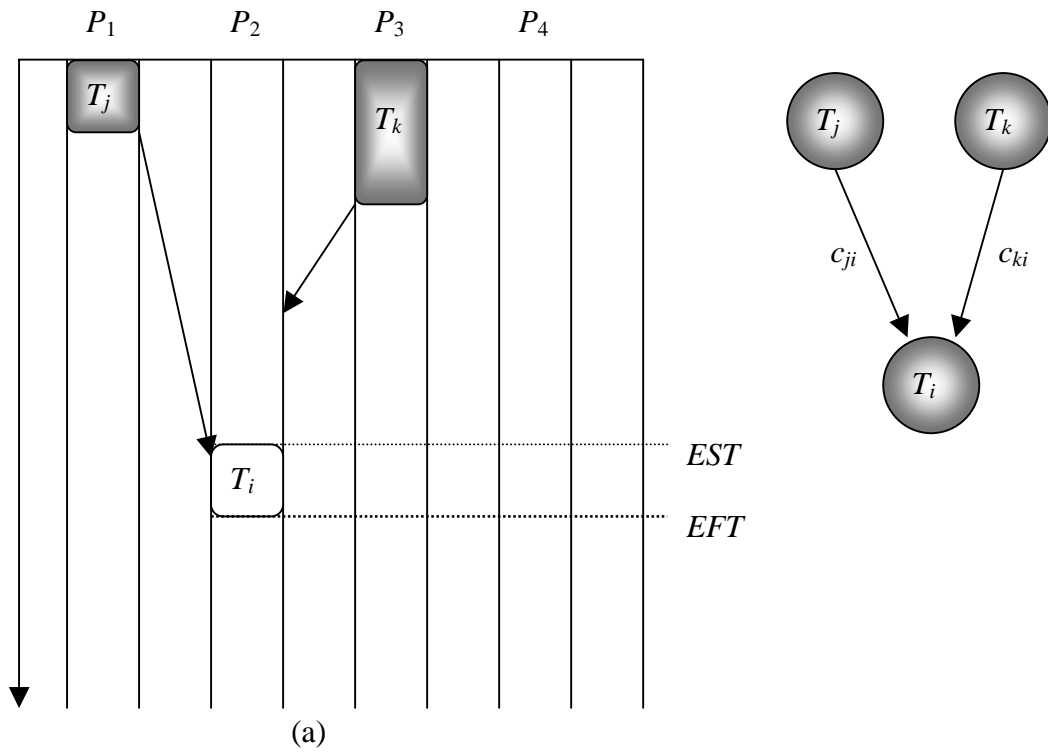


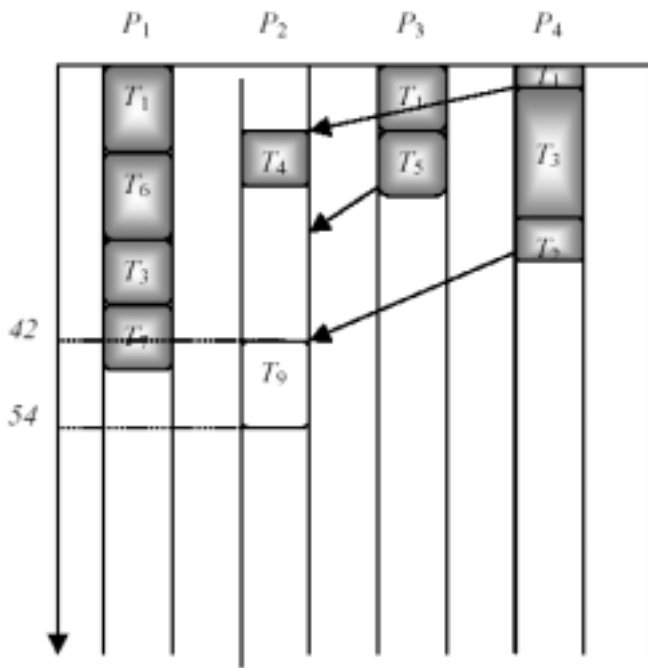
Figure 3.1 The detailed partial schedule of DAG (a) if task T_i is scheduled on P_2 (b) if task T_i is scheduled on P_2 after duplicating tasks in $C(T_j)$ and tasks in $C(T_k)$ into P_2 .

list is produced. According to the order of task in the task scheduling list, we select the task to assign on processor that minimizes earliest finish time of its selected task.

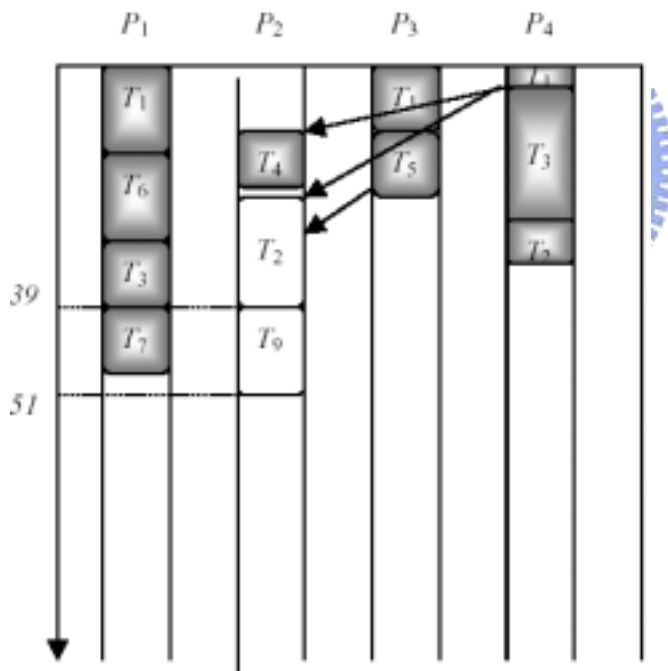
We first utilize the task duplication process repeatedly on each processor in order to find the minimum *EFT* of selected task on each processor. Finally, we assign the selected task to processor that has minimum *EFT* of its selected task. In our duplication process, we try to duplicate the tasks in the clusters of immediate predecessor of selected task into a processor sequentially. As an illustration, we assume the target system to be a HC with clique network. We use the DAG in Figure 2.2 as an example. After scheduling task T_1 , T_3 , T_4 , T_5 , T_2 , T_6 , and T_7 in the task scheduling list, we attempt to schedule task T_9 . We try to assign task T_9 in each processor. The Figure 3.2 (a) shows the detail partial schedule of trying to assign task T_9 on the P_2 . Next, we will execute task duplication process in P_2 . The immediate predecessors of task T_9 are task T_2 , task T_4 and task T_5 , but the task T_4 was assigned on P_2 . Thus, we try to duplicate the tasks in the cluster $C(T_2)$ and $C(T_5)$ into processor P_2 sequentially.

The selection order of these clusters for task duplication is decided by the values of data arrival time of all immediate predecessors of selected task on a processor, because it can reduce the *EFT* of selected task instantly. Let's discuss the example. Before duplicating the tasks in a cluster into processor P_2 , we need to decide the selection order in $C(T_2)$ and $C(T_5)$. We compute the data arrival time of T_2 and T_5 respectively. We find the task T_2 which has largest data arrival time. It also represents to select $C(T_2)$ first. The next selection is $C(T_5)$.

We use a Duplication function to duplicate the tasks in a cluster into a processor, and then return a minimum *EFT* of selected task and corresponding schedule. If the returned *EFT* value is increasing after duplicating the tasks in a cluster into a



(a)



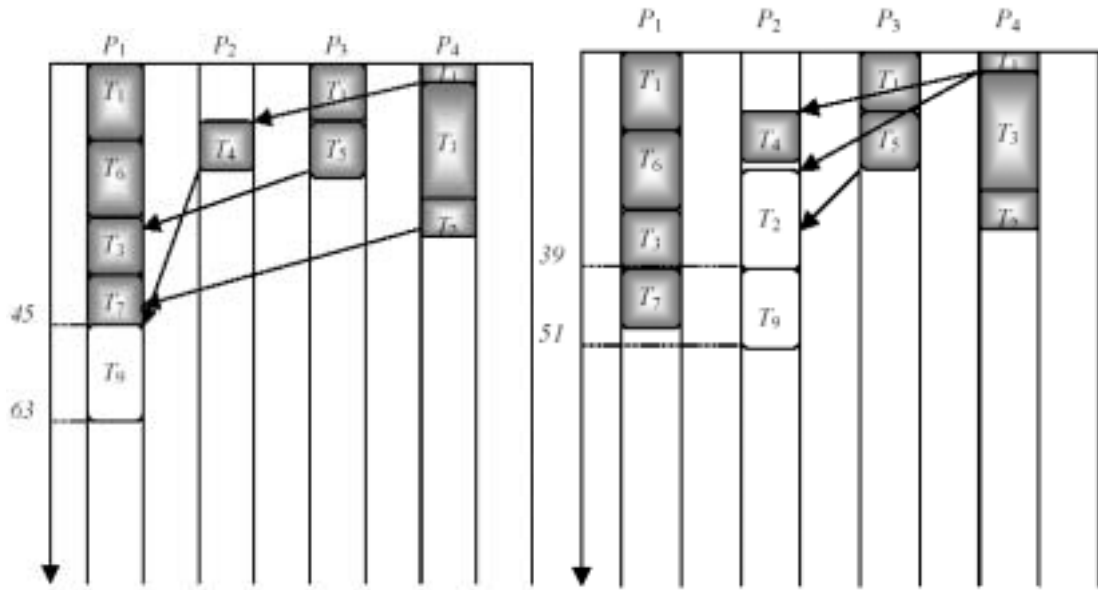
(b)

Figure 3.2 The detailed partial schedule of DAG in Figure 2.2 (a) if task T_9 is scheduled on P_2 and EFT is 54 (b) if task T_9 is scheduled on P_2 after duplicating task T_2 in $C(T_2)$ into P_2 and EFT is 51.

processor, we stop to duplicate the tasks in the next cluster into this processor. It may don't reduce the *EFT* of the selected task anymore if we continue to duplicate the tasks in the next cluster into a processor. Otherwise, there may exist enough big idle time-slot in the processor for assigning tasks in next cluster into this idle time-slot and thus reduce the *EFT* of its selected task again. Let's discuss the example. After allocating task T_2 and T_5 to appropriate processor, we know that the contents of $C(T_2)$ and $C(T_5)$ are $\{T_2\}$ and $\{T_1, T_5\}$, respectively. The idle time-slot between T_4 and T_9 in P_2 is enough to execute the task T_2 . Thus, we first duplicate the task T_2 in the $C(T_2)$ into P_2 and the detailed partial schedule is shown in Figure 3.2(b). We find that the returned minimum *EFT* of task T_9 is decreasing after duplicating task T_2 in the $C(T_2)$ into P_2 . We will continue to duplicate the tasks in the $C(T_5)$ into P_2 .

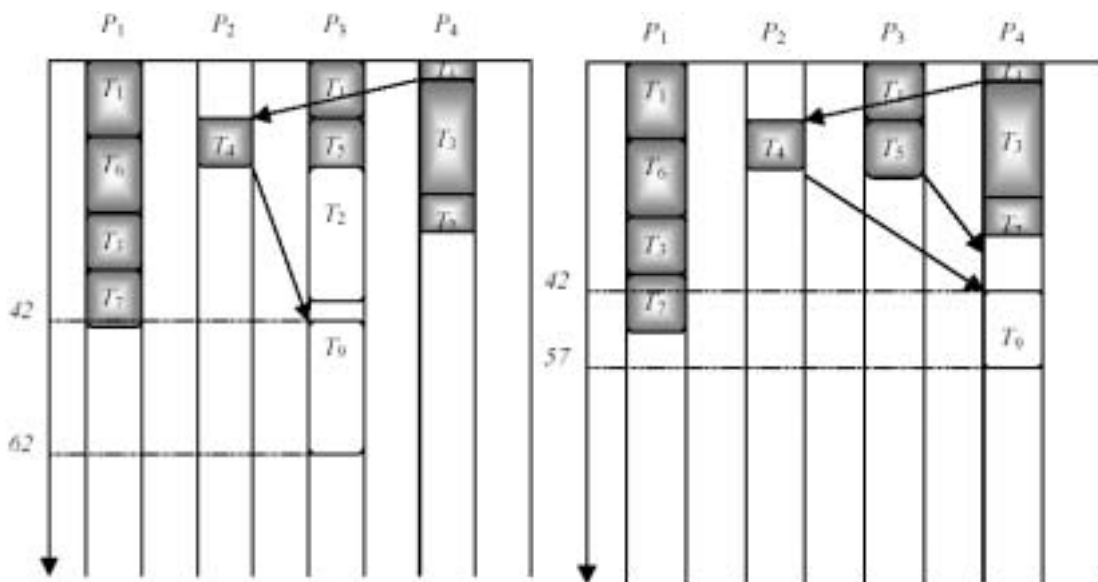
Notice that tasks in the cluster exist an order for duplicating a task into a processor. The order of the tasks is starting from the immediate predecessor of selected task (the last task in the cluster) to other ancestor tasks. We compute and record the *EFT* value of selected task when we duplicate a task in the cluster into a processor. Finally, the returned *EFT* value is minimum value among all recorded *EFT* values. Such duplicate order of tasks can reduce the *EFT* of selected task immediately. Let's discuss the example. Now, we try to duplicate the tasks in the $C(T_5)$ into P_2 . We find that the idle time-slot between T_2 and T_9 in P_2 isn't enough to execute the first task T_5 in the $C(T_5)$. If we continue to duplicate the tasks in the $C(T_5)$, the returned *EFT* value must increase. Thus, we stop the task duplication process on the processor P_2 . Finally, The partial schedule in Figure 3.2(b) represents the schedule status and the minimum *EFT* value of task T_9 on processor P_2 .

After utilizing the task duplication process repeatedly on each processor, the selected task is assigned to the processor with minimum *EFT* of selected task and



(a)

(b)



(c)

(d)

Task T_9 's	P_1	P_2	P_3	P_4
<i>EST</i>	45	39	42	42
<i>EFT</i>	63	51	62	57

(e)

Figure 3.3 The partial schedule of DAG in Figure 2.2 (a) if task T_9 is scheduled on P_1 (b) if task T_9 is scheduled on P_2 (c) if task T_9 is scheduled on P_3 (d) if task T_9 is scheduled on P_4 (e) related variables of partial schedule.

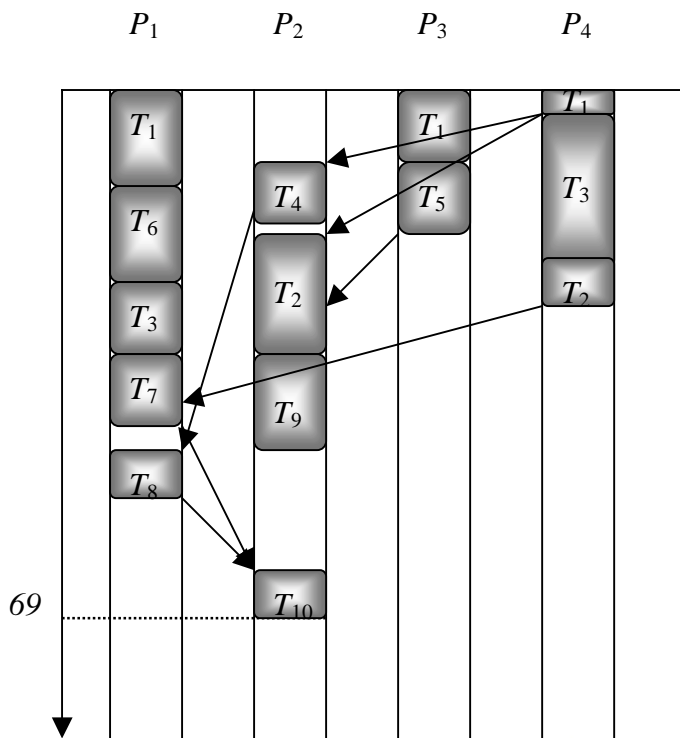


Figure 3.4 The final schedule generated by the DEFT1 algorithm and the schedule length is 69.

cluster of selected task is obtained. Let's discuss the example. In the same way, we assign T_9 to P_1, P_3 and P_4 . The partial schedules are shown in Figure 3.3, that is result of the minimum EFT value of task T_9 on each processor after executing task duplication process on each processor. The Figure 3.3(e) shows the related variables according to the partial schedule. Finally, task T_9 is scheduled to processor P_2 which has the minimum EFT value and the cluster $C(T_9)$ is $\{T_2, T_9\}$.

The tasks in the cluster of selected task are the tasks that are duplicated to processor with minimum EFT of selected task. This cluster includes some predecessors of selected task. It means that those tasks in the cluster are beneficial for reducing the EFT of selected task by duplicating those tasks into a processor.

When all tasks in the scheduling list are scheduled to appropriate processor, the

Input: DAG ,matrix W and network topology G_T

Output: Schedule result and schedule length

1. Sort all tasks in a scheduling list by nonincreasing order of b_level values;
2. **while** (there are unscheduled tasks in the list) **do**
3. Select the first task, T_i , from the list for scheduling;
4. **for** each processor P_k in the processor_set ($P_k \in P$)**do**
5. Compute $EFT(T_i, P_k)$;
6. Ftime= $EFT(T_i, P_k)$;
7. Sort all immediate predecessor tasks T_j that haven't assigned on P_k in a queue by nonincreasing order of $AFT(T_j) + c_{ji}$ values;
8. **while** (there are unvisited tasks in the queue)
9. Select the first task, T_j , from the queue;
10. **Duplication** ($C(T_j)$) ;
11. **if** ($EFT(T_i, P_k) > Ftime$)
12. $EFT(T_i, P_k) = Ftime$;
13. **break**; //Stop the while loop
14. **endif**
15. Ftime= $EFT(T_i, P_k)$;
16. **endwhile**
17. **endfor**
18. Assign T_i and duplicate tasks to the P_l that minimizes EFT of task T_i ;
19. Put the final duplicated tasks on the P_l into $C(T_i)$;
20. $C(T_i) = C(T_i) \cup T_i$;
21. **endwhile**

Figure 3.5 The algorithm of DEFT1.


```

Input:  $C(T_j)$ 

Output:  $EFT(T_i, P_k)$  and corresponding duplicate status

1. Duplication( $C(T_j)$ )
2. while (there are unvisited tasks in the  $C(T_j)$ ) do
3.     Select the last task,  $T_k$ , from the  $C(T_j)$ ;
4.     if (idle time-slot between tasks  $< w_{kk}$ )
5.         if ( $T_k == T_j$ )
6.             return  $EFT(T_i, P_k) = \infty$  ;
7.         endif
8.         break; //Stop the while loop
9.     endif
10.    Duplicate task  $T_k$  into  $P_k$  ;
11.    Compute  $EFT(T_i, P_k)$  and record the duplicate status ;
12. endwhile
13. return minimum  $EFT(T_i, P_k)$  and corresponding duplicate status ;
14.end

```

Figure 3.6 The algorithm of Duplication function.

final schedule result and schedule length are obtained. Let's discuss the example. The final schedule is shown in Figure 3.4. The schedule length, which is equal to 69, is shorter than that of the HEFT and STDS algorithm. The schedule lengths of HEFT and STDS algorithms are 77 and 86, respectively. Figure 3.5 shows the detail DEFT₁ algorithm and the Duplication function is shown in Figure 3.6.

3.2 DEFT₂ Algorithm

In this section, we will focus on the behavior of data transmission with link contention constraints. We also describe the difference of processor selection phase between in DEFT₁ and DEFT₂ algorithm in this section.

3.2.1 Task Prioritizing Phase

This phase is the same as DEFT₁ algorithm. As we mentioned before, the b_level function also is an appropriate priority function in the link contention environment. Thus, in this phase, we also use the b_level value as the priority value of each task.

3.2.2 Processor Selection Phase

Under the condition of link contention occurrence, we need to treat the communication edges in the same way as the tasks of the DAG. It means that the edges are scheduled to the network links in the same way the tasks are scheduled to the processors [38]. Corresponding to the EST and EFT of task, we will define MST (*Message Start Time*) and MFT (*Message Finish Time*) two attributes below. Before we define the attributes, it notices that L_k represents the k^{th} link (path) in a routing path for data transmission.

Definition 3.2 In a given partial schedule, let $R = \{L_1, L_2, \dots, L_n\}$ be a routing path with n links and task T_i transmits the message M_{ij} to task T_j by the routing path. We define the *Message Start Time* of message M_{ij} on link L_k in the routing path, denoted as $MST(M_{ij}, L_k)$, by the following equation:

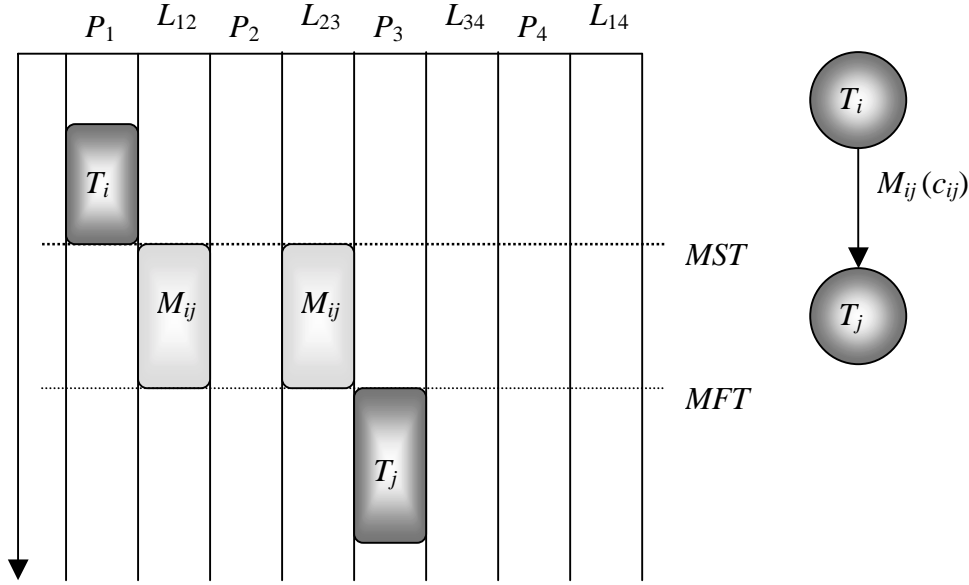


Figure 3.7 The *MST* and *MFT* of a message M_{ij} .

$$\begin{cases} MST(M_{ij}, L_k) = \text{Max}\{\text{avail}[L_k], AFT(T_i)\}, \text{ if } L_k \text{ is the first link } L_1; \\ MST(M_{ij}, L_k) = \text{Max}\{\text{avail}[L_k], MST(M_{ij}, L_{k-1})\}, \text{ otherwise.} \end{cases} \quad (6)$$

,where $\text{avail}[L_k]$ is the earliest time at which link L_k is ready for message transmission.

$AFT(T_i)$ is the actual finish time of a task T_i .

Definition 3.3 In a given partial schedule, let $R = \{L_1, L_2, \dots, L_n\}$ be a routing path with n links and task T_i transmits the message M_{ij} to task T_j by the routing path. We define the *Message Finish Time* of message M_{ij} on link L_k in the routing path, denoted as $MFT(M_{ij}, L_k)$, by the following equation:

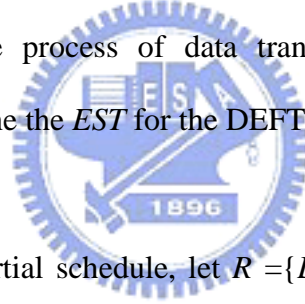
$$MFT(M_{ij}, L_k) = w_{ij} + MST(M_{ij}, L_k) \quad (7)$$

A simple example is shown in Figure 3.7. There exists the data dependency between task T_i and task T_j as shown in DAG. We assume that the communication cost between two tasks is c_{ij} . The network of processor is according to Figure 2.1. Task T_i

and task T_j are allocated on the different processor P_1 and P_3 , respectively. Thus, the message is arranged on the network link L_{12} and L_{23} according to the routing path. The final schedule is on the left side of Figure 3.7.

While the start time of a task is constrained by the data ready time of its incoming communication, the start time of a message is restricted by the finish time of its origin task. The scheduling of a message differs from a task, in that a message might be scheduled on more than one link. A communication between two tasks, which are scheduled on two different but not adjacent processors, utilizes all links of the routing path between the two processors. The message, representing this communication, must be scheduled on each of the involved links.

After understanding the process of data transmission with link contention constraints, we need to redefine the *EST* for the DEFT₂ algorithm in the definition 3.4.



Definition 3.4 In a given partial schedule, let $R = \{L_1, L_2, \dots, L_n\}$ be a routing path with n links. We define the *Earliest Start Time* of task T_i on processor P_j , denoted as $EST(T_i, P_j)$, by the following equation:

$$\begin{cases} EST(T_i, P_j) = 0, \text{ if task } T_i \text{ is the entry task;} \\ EST(T_i, P_j) = \text{Max}\{\text{avail}[P_j], \text{Max}_{T_j \in \text{pred}(T_i)} (\text{MFT}(M_{ji}, L_n))\}, \text{ otherwise.} \end{cases} \quad (8)$$

,where $\text{pred}(T_i)$ is the set of all immediate predecessors of task T_i , and $\text{avail}[P_j]$ is the earliest time at which processor P_j is ready for task execution. L_n is the last link in a routing path.

The definition of *EFT* in the Eq. (3) also is used in the DEFT₂ algorithm. In the

Input: DAG ,matrix W and network topology G_T

Output: Schedule result and schedule length

1. Sort all tasks in a scheduling list by nonincreasing order of b_level values;
2. **while** (there are unscheduled tasks in the list) **do**
3. Select the first task, T_i , from the list for scheduling;
4. **for** each processor P_k in the processor_set ($P_k \in P$)**do**
5. Compute $EFT(T_i, P_k)$;
6. Ftime= $EFT(T_i, P_k)$;
7. Sort all immediate predecessor tasks T_j that haven't assigned on P_k in a queue by nonincreasing order of $MFT(M_{ji}, L_n)$ value;
8. **while** (there are unvisited tasks in the queue)
9. Select the first task, T_j , from the queue;
10. **Duplication** ($C(T_j)$) ;
11. **if** ($EFT(T_i, P_k) > Ftime$)
12. $EFT(T_i, P_k) = Ftime$;
13. **break**; //Stop the while loop
14. **endif**
15. Ftime= $EFT(T_i, P_k)$;
16. **endwhile**
17. **endfor**
18. Assign T_i and duplicate tasks to the P_l that minimizes EFT of task T_i ;
19. Put the final duplicated tasks on the P_l into $C(T_i)$;
20. $C(T_i) = C(T_i) \cup T_i$;
21. **endwhile**

Figure 3.8 The algorithm of DEFT2.

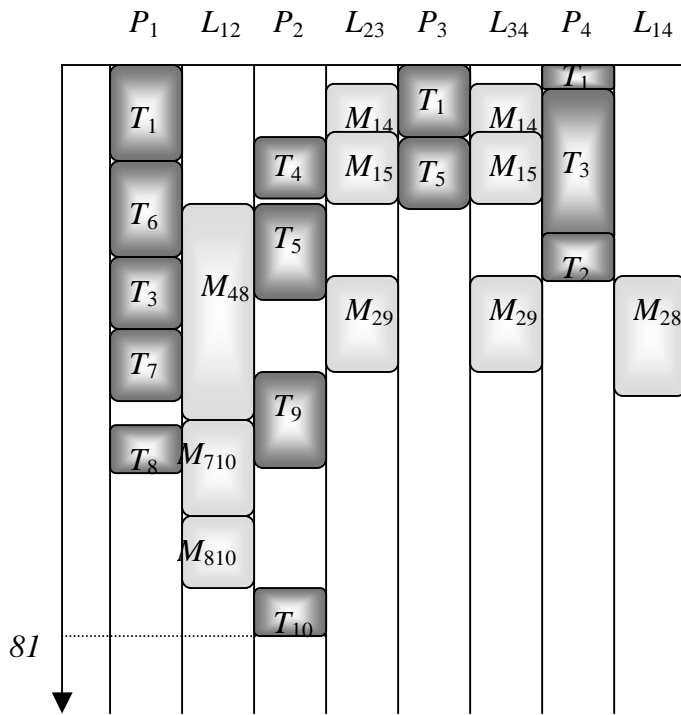


Figure 3.9 The final schedule generated by DEFT2 algorithm and the schedule length is 81.

following, we want to explain the difference of this processor selection phase. Figure 3.8 shows the DEFT₂ algorithm. There is a main difference in the step 7. We sort the $MFT(M_{ji}, L_n)$ value in a nonincreasing order to decide the selection order of clusters for task duplication because of link contention constraints. In this algorithm, the task duplication process or concepts are the same as DEFT₁ algorithm.

As an illustration, Figure 3.9 shows the final schedule of DAG in Figure 2.2 according to the network topology in the Figure 2.1. We can see that each message can only start the transmission on the link after the link is ready. Further, the start time of the message on a link can't be earlier than the start time of the message on previous link in the routing path.

In conclusion, we propose the DEFT (it includes DEFT₁ and DEFT₂) algorithm

which both contains the concept of *EFT*, which is broadly used in many effective task scheduling algorithms of heterogeneous system and the advantage of the task duplication method.

In the following, we will give the analysis on the time complexity of our propose algorithm. The time complexity is derived as follows. The given task graph contains n tasks and e edges, and we have m processors in our system. In the task prioritizing phase, we compute the b_level value of each task by traversing the given task graph. The time complexity of this phase is $O(n+e)$. In the processor selection phase, the time complexity of Duplication function is $O(n^2)$. Thus, the time complexity of whole duplication process is $O(dn^2)$, where d is the maximum number of immediate predecessor of tasks in a DAG. Each task takes the $O(dn^2m)$ time to select a processor. That is, the time complexity of this phase is $O(dn^3m)$. Therefore, the time complexity of the DEFT algorithm would be $O(dn^3m)$. Although it is higher than other related algorithms, it only slight difference in running time comparing with related algorithms by our simulation result.

In order to verify the effectiveness of our algorithm, we construct the simulation environment and implement the related algorithms. In the next chapter, we will explain our simulator and analyze the simulation results.

Chapter 4. Simulation and Performance Evaluations

After describing the Duplication-based Earliest Finish Time (DEFT) algorithm, we will verify the effectiveness of this algorithm by implementation and simulation. At first, we will describe the architecture of the simulator in section 4.1. Next, we will give the performance evaluations in section 4.2.

4.1 Simulation Setup

The flow chart of the simulation is shown in Figure 4.1. We use the C++ language to construct our simulator. There are three parts in our simulator. The first part is *Random Graph Generator*, the second part is *Network Topology Generator* that generates Clique, Hypercube, Mesh and Ring for our target system, and the third part is algorithm. We will give the detailed description about each part in the following.

(a) Random Graph Generator (RGG) [2]

As we defined in definition 2.2, the parallel program with n tasks can be represented as a DAG. A RGG is implemented to generate the DAGs with various characteristics that depend on several input parameters given below.

- Number of tasks in the graph, (n).
- Maximum number of out degree of a task, (out_degree). The out degree value of each task will be randomly generated from a uniform distribution with the interval $[0, out_degree]$.
- Shape parameter of the graph, (α). A dense graph (a shorter graph with high

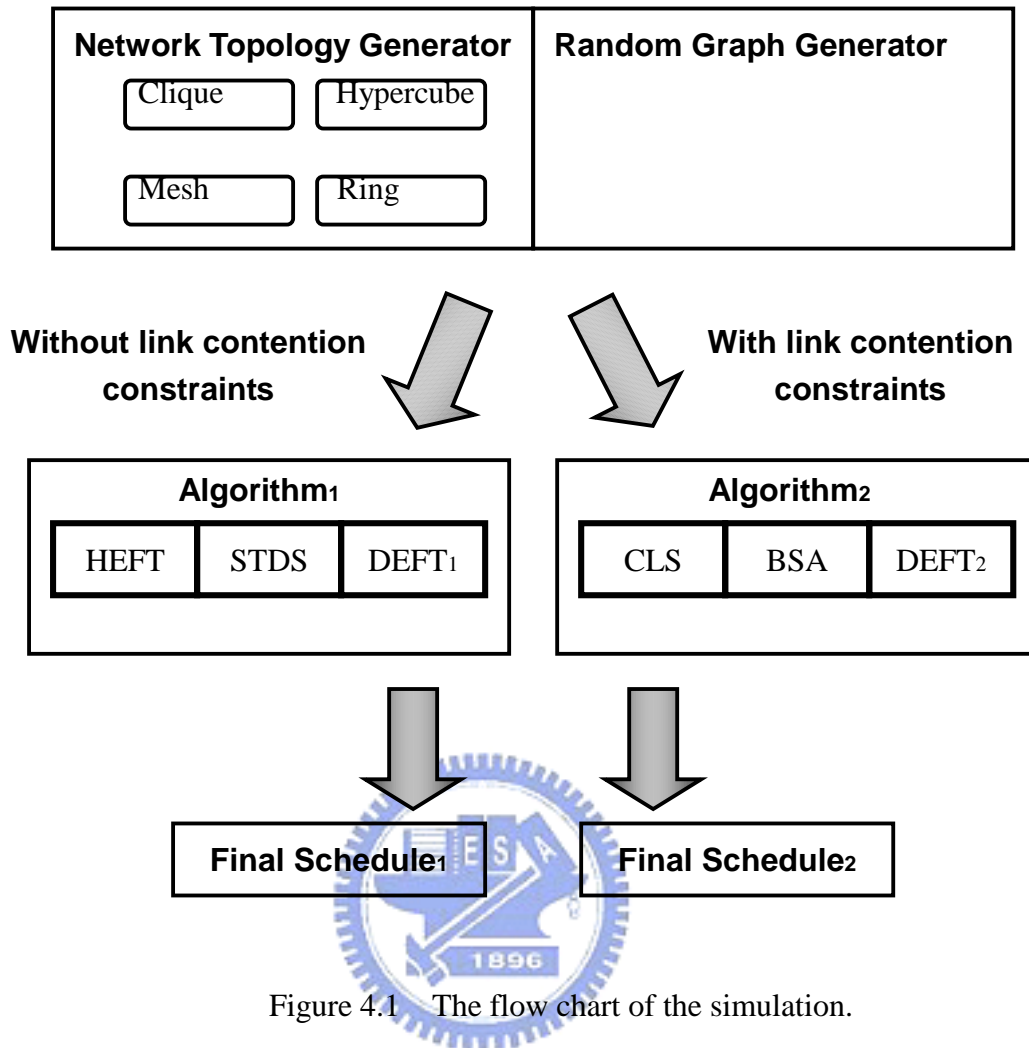


Figure 4.1 The flow chart of the simulation.

parallelism) can be generated by selecting $\alpha > 1.0$. On the contrary, if $\alpha < 1.0$, it will generate a longer graph with low parallelism degree. If $\alpha = 1.0$, then it will be a balanced DAG.

- Communication to computation ratio of a graph, (CCR). It is the ratio of the average communication cost to the average computation cost. If the CCR value of a DAG is very low, it can be considered as a computation-intensive application. On the contrary, it can be considered as a communication-intensive application.
- Maximum range of computation costs of a task on processors, (β). The

maximum multiple of difference among computation costs of a task on processors will be randomly generated from a uniform distribution with the interval $[1, \beta]$. It is basically the heterogeneity factor for processor speeds. A high β value causes a significant difference in computation cost of a task among the processors. A low β value indicates that the expected computation cost of a task is almost equal on each of the given processors in the system.

In each simulation, the values of these parameters are assigned from the corresponding sets given below.

- $SET_n = \{ 50, 100, 200, 300, 400, 500 \}$,
- $SET_{out_degree} = \{ 8, 15, 20 \}$,
- $SET_\alpha = \{ 0.5, 1.0, 2.0 \}$,
- $SET_{CCR} = \{ 0.1, 1.0, 10.0 \}$,
- $SET_\beta = \{ 1.2, 3.0, 7.0 \}$.



These combinations give 486 different DAG types. Since 10 random DAGs are generated for each DAG type, the total number of DAGs used in our simulations is 4860. Assigning several input parameters and selecting each parameter from a set cause the generation of diverse DAGs with various characteristics. Simulations based on diverse DAGs prevent biasing toward a particular scheduling algorithm.

(b) Network Topology Generator [36]

In our simulation, we adopt four kinds of interconnection network topology including clique, hypercube, mesh and ring. Any one of these four kinds of network topology is used widely [39]. We assume that there are 16 processors in our target system. We can observe that the communication resource (network link) varies

according to different kinds of network topology. For example, the ring network has the minimum communication resource and the clique network has the maximum communication resource. We want to show that our proposed algorithm can work well on each kind of network topologies.

(c) Algorithm

The input of the algorithm is a task graph generated from the RGG and one kind of network topology. The output of the algorithm is the final schedule. We implement the DEFT algorithm with some related algorithms in our simulation. As we mentioned in chapter 2, the HEFT algorithm and STDS algorithm assume the target system without the link contention constraints. We plan to simulate those algorithms and DEFT₁ algorithm with the same system assumptions. We also compare the simulation results of three algorithms under different conditions. On the other hand, we simulate the CLS and BSA algorithm that have the link contention constraints and DEFT₂ algorithm, and compare the simulation results among them.

4.2 Performance Evaluations

The comparison metric of a scheduling algorithm on a graph is the schedule length (*makespan*) of its output schedule. We define the *Schedule Length Ratio* (SLR) as the schedule length of the DEFT algorithm divided by the schedule length of the related algorithm. The related algorithm is one of following algorithms, such as HEFT, STDS, CLS and BSA. If the SLR is larger than 1.0 means the related algorithm has the smaller schedule length, that is, the related algorithm has the better scheduling result. On the contrary, if the SLR is smaller than 1.0 means the DEFT algorithm has the better scheduling result.

We evaluate the performances of the DEFT₁ algorithm comparing with the HEFT and STDS algorithm on common clique network topology in section 4.2.1, and the section 4.2.2 is to evaluate the performance of the DEFT₂ algorithm comparing with the CLS and BSA algorithm on four kinds of network topology: clique, hypercube, mesh and ring.

4.2.1 DEFT₁ vs. HEFT and STDS

The simulation result of DEFT₁ and HEFT is illustrated in Figure 4.2. We can find that the average SLR is smaller than 1.0 or nearly equal to 1.0 in all of three cases. It indicates that the DEFT₁ algorithm is more effective than HEFT, especially when CCR equals to 10.0. In such communication-intensive task graph, the average communication cost is ten times of the average computation cost. Thus, we can find many idle time-slots in processors in the schedule result of HEFT algorithm. We utilize these idle time-slots in the processors efficiently by duplicating the tasks in cluster into the processors in our DEFT₁ algorithm. It can get shorter *EFT* values of each task than that of HEFT algorithm. On the contrary, under computation-intensive applications, the returned minimum *EFT* of scheduling task is increasing after duplicating the tasks in first selected cluster on a processor. It stops the duplication process on processor in DEFT₁ algorithm, because it may don't reduce the *EFT* of the scheduling task anymore when we continue to duplicate the tasks in next cluster. Thus, we can observe that the performance of the DEFT₁ algorithm is nearly equal to that of the HEFT in the graph with low *CCR* value.

The simulation result of DEFT₁ and STDS is illustrated in Figure 4.3. We can observe that the average SLR is smaller than 1.0 in all of three cases, and DEFT₁ obviously outperforms the STDS in higher *CCR* value. As we mentioned in chapter 2,

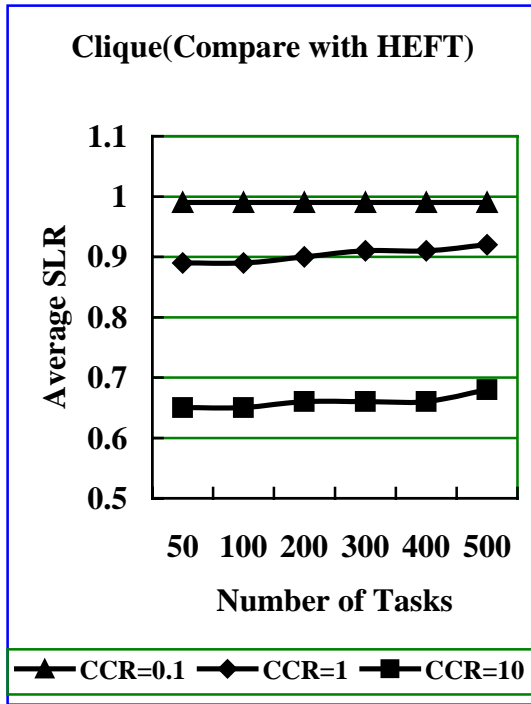


Figure 4.2 The simulation result of DEFT₁ and HEFT.

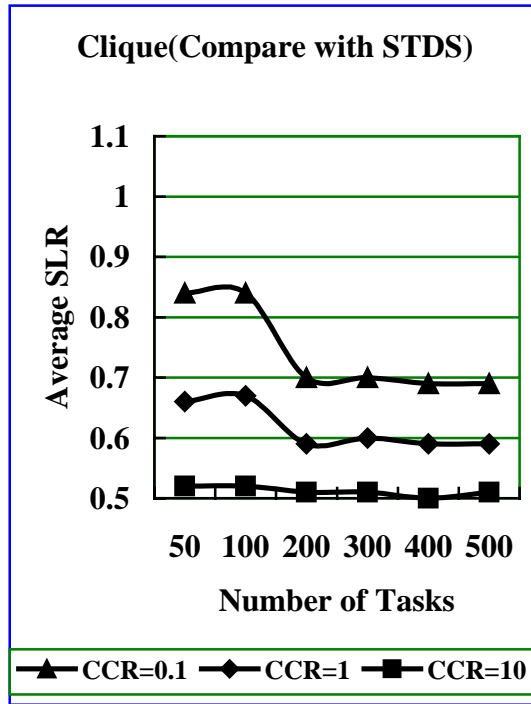


Figure 4.3 The simulation result of DEFT₁ and STDS.

in the STDS, only one immediate predecessor of given task in a critical path assigns to the same processor of given task. The other immediate predecessors of given task are assigned to different processors. It causes that the communication overhead between immediate predecessor tasks of given task in a critical path and its given task is very heavy. Thus, when the parallel program is communication-intensive, it makes a poor schedule result. In our duplication method, each cluster of immediate predecessor of scheduling task includes some predecessors of scheduling task as the selection of task duplication. Thus, we have considering to duplicate other immediate predecessors of a task into a processor. It can avoid the communication cost between immediate predecessor of a task and its task in our algorithm. It notices that the average SLR is slightly diminution when number of task is more than 200 in that the number of initial clusters is larger than the number of processors in the STDS algorithm. Thus, the duplication process of STDS isn't carried out and makes an

inferior schedule result.

The performance ranking of the algorithms will be {DEFT₁, HEFT, STDS}. The average SLR value of DEFT₁ on all generated graphs are 15 and 38 percent smaller than that of HEFT and STDS, respectively.

4.2.2 DEFT₂ vs. CLS and BSA

The simulation results of DEFT₂ and CLS on the clique network, hypercube network, mesh network and ring network are shown in Figure 4.4~Figure 4.7, respectively. The average SLR is also smaller than 1.0 in all three cases on four different networks. Thus, we can say that the DEFT₂ algorithm certainly has more effective performance than that of the CLS algorithm. As we described in chapter 2, the CLS algorithm is extended from the HEFT algorithm. Thus, the CLS algorithm inherits the phenomena in the HEFT algorithm, that is, there still have many idle time-slots in the processors. Similarly, we utilize these idle time-slots for task duplication to achieve the goal of reducing schedule length in the DEFT₂ algorithm. The average SLR value of DEFT₂ on all generated graphs are 18, 16, 16 and 14 percent smaller than that of CLS on the clique network, on hypercube network, on mesh network and on ring network, respectively.

The simulation results of DEFT₂ and BSA on the clique network, hypercube network, mesh network and ring network are shown in Figure 4.8~Figure 4.11, respectively. The average SLR is smaller than 1.0 in all three cases on four different networks. In the Figure 4.8, the clique network offers the sufficient communication resource. Each processor has a direct network link with the other processors. In the BSA algorithm, each task tries to migrate to each processor and to find the minimum finish time of the task. It is similar to the process of HEFT algorithm. Thus, the

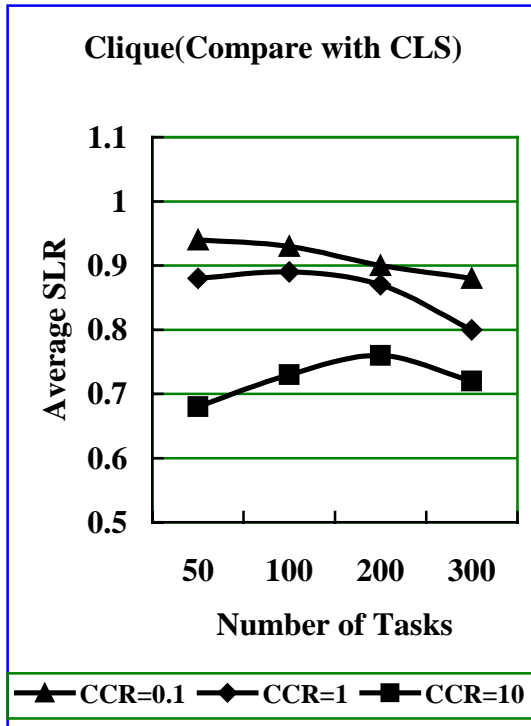


Figure 4.4 The simulation result of DEFT₂ and CLS on the Cliques network.

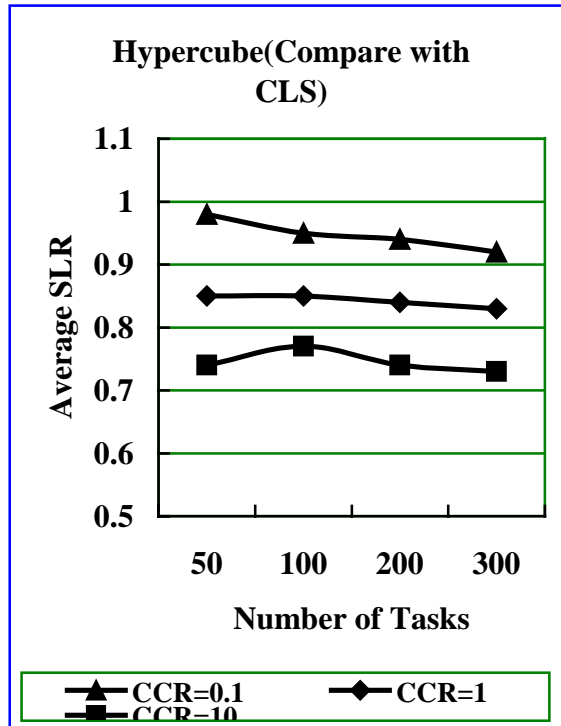


Figure 4.5 The simulation result of DEFT₂ and CLS on the Hypercube network.

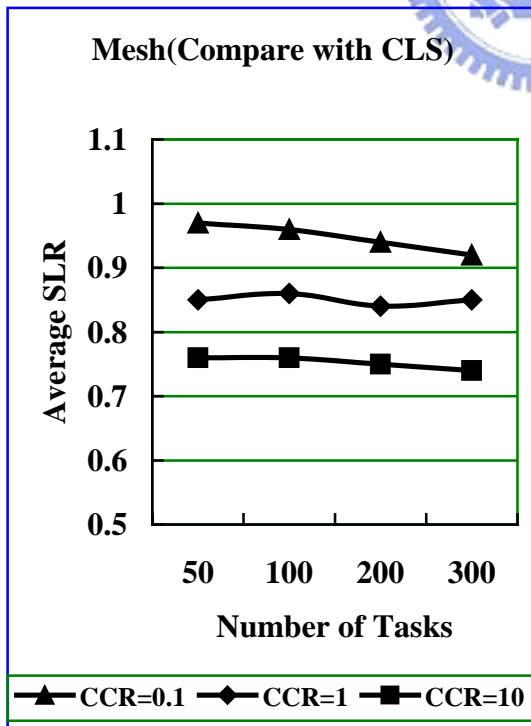


Figure 4.6 The simulation result of DEFT₂ and CLS on the Mesh network.

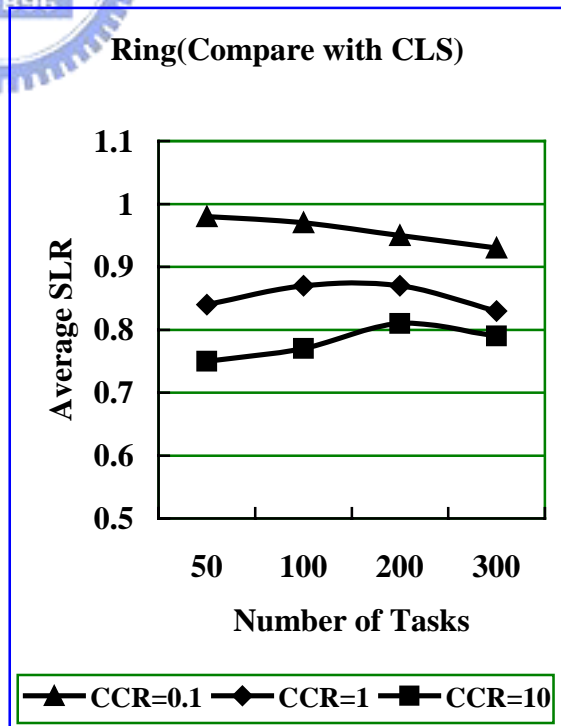


Figure 4.7 The simulation result of DEFT₂ and CLS on the Ring network.

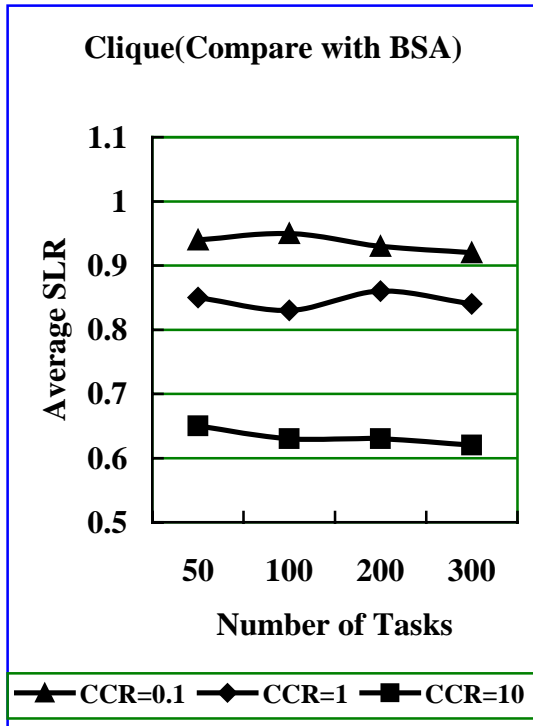


Figure 4.8 The simulation result of DEFT₂ and BSA on the Cliques network.

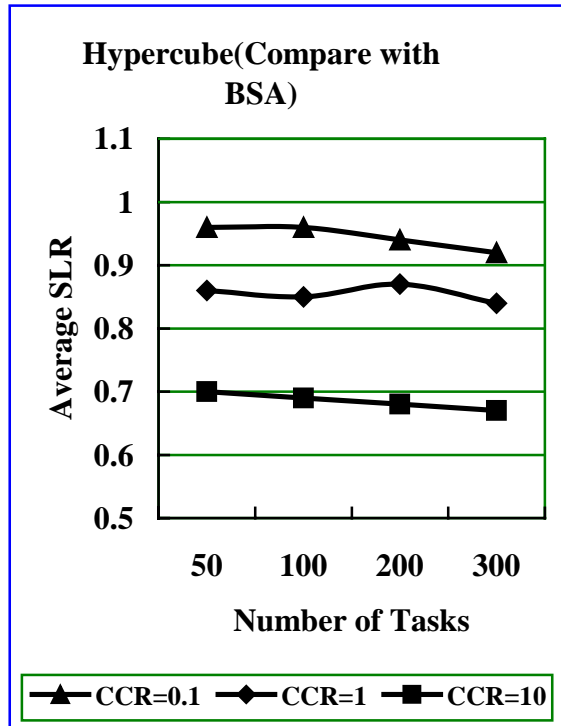


Figure 4.9 The simulation result of DEFT₂ and BSA on the Hypercube network.

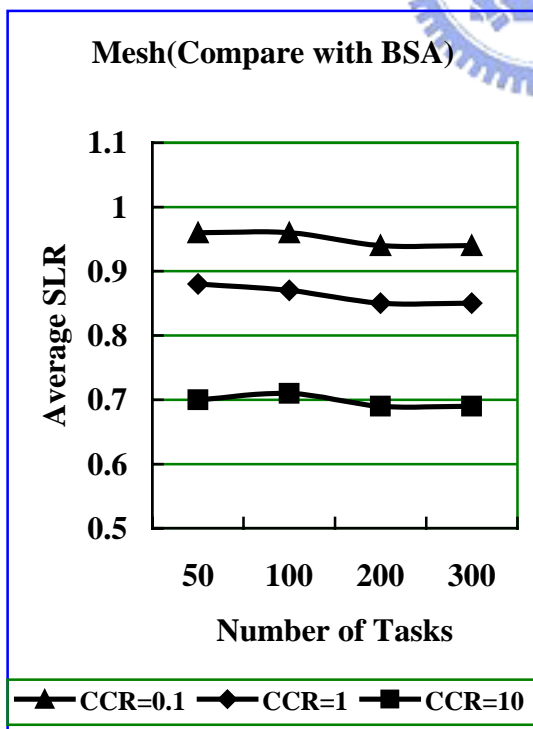


Figure 4.10 The simulation result of DEFT₂ and BSA on the Mesh network.

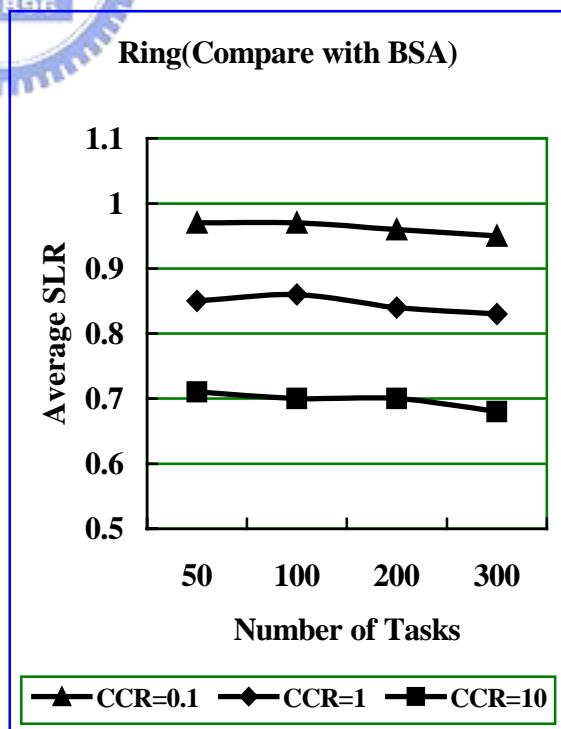


Figure 4.11 The simulation result of DEFT₂ and BSA on the Ring network.

simulation result is also similar to that of HEFT algorithm. The deficient communication resource in the network will limit the task to migrate to each processor. It may affect the task to select a suitable processor in the BSA algorithm and make an inferior schedule result. There still have many idle time-slots in the processors in the schedule results of BSA algorithm on four kinds of network, especially when CCR equals to 10.0. Similarly, we can utilize idle time-slots for task duplication to reduce schedule length in the DEFT₂ algorithm. The average SLR value of DEFT₂ on all generated graphs are 20, 18, 17 and 17 percent smaller than that of BSA on the clique network, on hypercube network, on mesh network and on ring network, respectively.

The performance ranking of the algorithms will be {DEFT₂, CLS, BSA }. The average SLR value of DEFT₂ on all generated graphs are 14~18 and 17~20 percent smaller than that of CLS and BSA on four kinds of network, respectively.


In conclusion, we have verified the effectiveness of the DEFT algorithm through the simulation. The simulation results show that the DEFT algorithms outperform the other algorithms for any graph size in terms of SLR, whether it exists the link contention constraints in the target system or not.

Chapter 5. Conclusions and Future

Work

We have introduced our system architecture and proposed effective algorithm to solve the task scheduling problem in the previous chapters. We design two similar algorithms. DEFT₁ is for target system without link contention constraints, and DEFT₂ is for target system with link contention constraints. Finally, in order to evaluate our algorithm, we construct a simulation environment and compare with the related algorithms. In this chapter, we will make some conclusions and also describe some future work on this research topic.

5.1 Conclusions



We found that the selection strategy of tasks for duplication isn't effective enough in the STDS algorithm. Thus, we have proposed the *Duplication-based Earliest Finish Time* (DEFT) algorithm. This algorithm contains two phases. The first phase is task prioritizing phase for computing the priorities of all tasks by a efficient priority function. In the second phase, we select the processor which can complete the task earliest by a task duplication mechanism. The concept of task duplication mechanism is that we utilize processor idling time for duplicating some predecessors of scheduling task into a processor to avoid communication costs between tasks.

In summary, it has some characteristics compared with other related methods, such as the HEFT, STDS, CLS and BSA algorithm:

- (1) For effectiveness, we design a duplication process to effectively reduce the *EFT* and choose the appropriate processor for each task. We also verify the effectiveness of this method by constructing simulation environment. The simulation result

shows that the DEFT algorithm effectively shortens the schedule length comparing with the related algorithms, especially when CCR equals to 10.0 (a communication-intensive parallel program). In general, the average schedule length of DEFT₁ are 15 and 38 percent smaller than that of HEFT and STDS, respectively. The average schedule length of DEFT₂ are about 14 ~ 18 and 17~20 percent smaller than that of CLS and BSA algorithm on four kinds of network topology, respectively.

(2) For efficiency, the time complexity of the DEFT algorithm is $O(dn^3m)$, where d is the maximum number of immediate predecessor of tasks in a DAG, n is the number of tasks and m is the number of processors. Although it is higher than other related algorithms, it will be a very important factor no more because we can use a fast processor for scheduling tasks.



5.2 Future Work

In addition to the features we discussed before, there are still several promising issues in future researches.

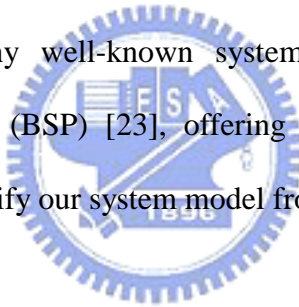
(1) We try to design another priority function when dealing with different task graphs.

Different task graphs may have the different characteristics. In a computation-intensive (i.e., $CCR = 0.1$) parallel program, the average computation cost is about ten times the average task communication cost. We can design an appropriate priority function for such parallel program. It can help us rank the tasks more properly.

(2) By using the concept of look-ahead mechanism like CLS algorithm may improve our method further. The concept of look-ahead mechanism in the CLS algorithm is to select the processor which can complete the scheduling task early and has the

sufficient network link by a look-ahead manner. The manner is to compute the earliest finish time of assigning the communication cost that forwards to immediate successor of scheduling task on direct link of the processor. The look-ahead mechanism can help that the immediate successor tasks of scheduling task receive the data early. However, some inaccurate look-ahead may bring degradation in performance as we mentioned in chapter 2. We will design a look-ahead mechanism for our algorithm by modifying that of CLS algorithm.

(3) We may add some other realistic constraints in our system model and modify the DEFT₂ algorithm more practical. We don't take the consideration of system latency, message size, network bandwidth...,etc. into our system model. As the system model incorporates the realistic constraints, it is more difficult to design a good algorithm. There are many well-known system model, such as *LogP* [22], *Bulk-Synchronous Parallel* (BSP) [23], offering the realistic model of parallel computation. We try to modify our system model from previous work.



Bibliographies

- [1] I. Ekmecic, I. Tartalja, and V. Milutinovic, "A survey of heterogeneous computing: concepts and systems", *Proc. IEEE*, vol. 84, pp. 1127 -1144, Aug. 1996.
- [2] H. Topcuoglu, S. Hariri, and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Trans. on Parallel and Distributed Systems*, vol.13, pp. 260 -274, Mar. 2002.
- [3] M.R. Gary and D.S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W.H. Freeman and Co., 1979.
- [4] M. Wu and D. Gajski, "Hypertool: A Programming Aid for Message Passing Systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, pp. 330-342, July 1990.
- [5] Y. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, May 1996.
- [6] G.C Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no.2, pp. 175-186, Feb. 1993.
- [7] H. El-Rewini and T.G. Lewis, "Scheduling Parallel Program Tasks on Arbitrary Target Machines," *J. Parallel and Distributed Computing*, vol. 9, pp. 138-153, 1990.
- [8] B. Kruatrachue and T.G. Lewis, "Grain Size Determination for Parallel Processing," *IEEE Software*, pp. 23-32, Jan. 1988.
- [9] J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y.Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Costs," *SIAM J.*

- Computing*, vol.18, no. 2, pp. 244257, 1989.
- [10] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number Processors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, no. 9, pp.951-967, Sept. 1994.
- [11] S.J. Kim and J.C. Browne, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures," *Proc. Int'l Conf. Parallel Processing*, vol. 2, pp. 1-8, 1988.
- [12] J. Liou and M.A Palis, "An Efficient Clustering Heuristic for Scheduling DAGs on Multiprocessors," *Proc. Symp. Parallel and Distributed Processing*, 1996.
- [13] I. Ahmad and Y.Kwok, "A New Approach to Scheduling Parallel Programs Using Task Duplication," *Proc. Int'l Conf. Parallel Processing*, vol.2, pp. 47-51, 1994.
- [14] Y. Chung and S. Ranka, "Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors," *Proc. Supercomputing*, pp. 512-521, Nov. 1992.
- [15] E.S.H. Hu, N. Ansari, and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Trans. on Parallel and Distributed Systems*, vol.5, no. 2, pp. 113-120, Feb. 1994.
- [16] H. Singh and A. Youssef, "Mapping and Scheduling Heterogeneous Task Graphs Using Genetic Algorithm," *Proc. Heterogeneous Computing Workshop*, pp. 86-97, 1996.
- [17] L. Wang, H.J. Siegel, and V.P. Roychowdhury, "A Genetic Algorithm-Based Approach for Task Matching and Scheduling in Heterogeneous Computing Environments," *Proc. Heterogeneous Computing Workshop*, 1996.

- [18] P. Shroff, D.W. Watson, N.S. Flann, and R. Freund, "Genetic Simulated Annealing for Scheduling Data-Dependent Tasks in Heterogeneous Environments," *Proc. Heterogeneous Computing Workshop*, pp. 98-104, 1996.
- [19] L. Tao, B. Narahari, and Y.C. Zhan, "Heuristics for Mapping Parallel Computations to Heterogeneous Parallel Architectures," *Proc. Heterogeneous Computing Workshop*, 1993.
- [20] M. Wu, W. Shu, and J. Gu, "Efficient Local Search for DAG Scheduling," *IEEE Trans. on Parallel and Distributed System*, vol. 12, no. 6, pp. 617-627, June 2001.
- [21] Y. Kwok, I. Ahmad, and J. Gu, "FAST: A Low-Complexity Algorithm for Efficient Scheduling of DAGs on Parallel Processors," *Proc. Int'l Conf. Parallel Processing*, vol. 2, pp. 150-157, 1996.
- [22] D. Culler, R. Karp, D. Patterson, A. Shahy, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP-A Practice Model of Parallel Computation," *Comm. ACM*, vol. 39, no. 11, pp. 78-85, 1996.
- [23] L.G. Valiant, "A Bridging Model for Parallel Computation," *Comm. ACM*, pp. 103-111, 1990.
- [24] Yu-Kwong Kwok and Ishfaq Ahmad, "Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors," *Proc. Int'l Conf. Parallel Processing*, pp. 551-558, 1999.
- [25] T.S. Hsu, Joseph C. Lee, Dian Rae Lopez and William A. Royce, "Task Allocation on a Network of Processors", *IEEE Trans. Computers*, vol. 49, no. 12, pp. 1339-1353, Dec. 2000.
- [26] Samantha Ranaweera and Dharma P. Agrawal, "A Task Duplication Based Scheduling for Heterogeneous Systems", *Proc. of 14th International Parallel*

and Distributed Processing Symposium, pp. 445-450, May 2000.

- [27] Samantha Ranaweera and Dharma P. Agrawal, “A Scalable Task Duplication Based Scheduling Algorithm for Heterogeneous Systems”, *Proc. of International Conference on Parallel Processing*, pp. 383-390, Aug. 2000.
- [28] Tae-Young Choe and Chan-Ik Park, “A Task Duplication Based Scheduling Algorithm with Optimality Condition in Heterogeneous Systems”, *Proc. of International Conference on Parallel Processing Workshop*, pp. 531-536, Aug. 2002.
- [29] Atakan Dogan and Fusun Ozguner, “LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems”, *Proc. of International Conference on Parallel Processing*, pp. 352-359, Aug. 2002
- [30] S. Darbha and D.P. Agrawal, “Optimal Scheduling algorithm for distributed memory machines”, *IEEE Trans on parallel and distributed systems*, vol. 9, no. 1, pp. 87-95, January 1998.
- [31] Yu-Kwong Kwok “Parallel program execution on a heterogeneous PC cluster using task duplication” ; *Proceedings. 9th Heterogeneous Computing Workshop, 2000. (HCW 2000)*, pp. 364 –374 1 May 2000
- [32] Chan-Ik Park; Tae-Young Choe ”An optimal scheduling algorithm based on task duplication” *Proceedings. Eighth International Conference on Parallel and Distributed Systems, 2001. ICPADS 2001.*, 26-29 June 2001
- [33] Chan-Ik Park; Tae-Young Choe; “An optimal scheduling algorithm based on task duplication “*IEEE Trans on Computers*, vol 51 Issue: 4 , April 2002
- [34] Bansal, S.; Kumar, P.; Singh, K “An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems”.; *IEEE Transactions on Parallel and Distributed Systems.*, Vol 14 Issue 6 , pp. 533 –544

June 2003

- [35] Li Guodong; Chen Daoxu; Wang Darning; Zhang Defu; “Task clustering and scheduling to multiprocessors with duplication” *Proceedings. International Parallel and Distributed Processing Symposium, 2003.*, pp. 6 –13 April 2003
- [36] Shuo-Zhan Ho and Cheng Chen, ”An Effective Task Scheduling Method with Link Contention Constraints for Heterogeneous System” , *MS thesis, National Chiao Tung University, Taiwan, R.O.C* 2003
- [37] B.S. Macey, A.Y. Zomaya, “A performance evaluation of CP list scheduling heuristics for communication intensive task graphs” ,*Parallel Processing Symposium,1998*, pp. 538-541 1998
- [38] Oliver Sinnen and Leonel Sousa, ”List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures”, *Parallel Computing Symposium,2004*, pp. 81-101 2004
- [39] Behrooz Parhami. **Introduction to Parallel Processing.** Plenum Book Company,1999.