# Techniques for Task Scheduling in Practical Parallel and Distributed System

by

*Chan Wai Yip, BEng (Hons)*

A dissertation submitted in partial fulfillment of the requirements for the

Degree of Master of Philosophy

The Hong Kong Polytechnic University

Supervisor: Dr. Li Chi Kwong

Department of Electronic and Information Engineering

The Hong Kong Polytechnic University,

Hong Kong

This work was supported in part by

The Hong Kong Polytechnic University projects 340/978

# Table of contents

# List of Figures

## List of Tables

## List of Algorithms

# Acknowledgements

Firstly, I would like to express my deep gratitude towards my MPhil. Supervisor, Dr. Chi Kwong Li, for his endless support, invaluable advice and excellent guidance. His kindness and patience have spiritually helped in one way or another to solve the problems. Besides, I am grateful to Prof. Wan Chi Siu and Dr. Peter Tam for their assistance and patience in applying for various research funds to support my MPhil. Study.

Then, I would like to thank Mr. Wai Kin Lam who has enlightened me on the subject of the parallel and distributed systems and the contractual computing paradigm with his frequent and detailed discussions. In addition, without the continuous encouragement and technical support of Dr. Kwok Fai Wan and Hing Cheung Hung, it is impossible for me to complete the work on time. I am also greatly indebted to Dr. Wai Kit Lam, Dr. Hui Wei Guan, Dr. Ying Shan, Wai Kwong Cheuk and Eric Tze for their friendly advice and help. Thanks are also due to Chi Yuen Fung, Chi Cheong Law and Kam Tai Yam who contributed in studying part of concept and development of the programs.

The last but not the least, I would like to thank my friends and my family, for whom I could not make as much time as I would have liked to.

# Prefect

In the last two decades, massively connected parallel computer systems and networked distributed systems provide the processing power, which is demanded for solving problems of extensive computation in nature. Aiming to achieve better performance by using these systems, scheduling algorithms are used. The objectives of scheduling algorithms are to allocate tasks into processors and to order their execution so that data dependencies are satisfied and the length of the produced schedule (parallel time) are minimized. However, it was proved that such scheduling problem is NP-complete [Chret89] [Papadimitriou90] [Sarkar89]. It is very difficult to obtain an optimal scheduling solution in polynomial time complexity.

In order to obtain a scheduling solution in lower time complexity, heuristic approaches of task scheduling algorithms were proposed to approximate the optimal solution, which can be completed in polynomial time complexity [ACDi74] [Papadimitriou90]. Hence, heuristic approach of task scheduling algorithm for homogeneous processors environment is becoming an active research topic until now and a number of heuristic scheduling algorithms were proposed [ACDi74] [Elre90] [Krua87] [Yang94] [Kruatrachue]. However, most of these algorithms were bounded by a number of restrictions while implementing in practice, which seriously affected the performance of such those proposed task schedulers in the real-life situation.

To begin with, a study of evolution of the parallel and distributed systems and classical task scheduling techniques is performed. It has been identified that the list scheduling heuristic is a widely accepted approach for task scheduling. Algorithms like the Dominant Sequence Clustering (DSC) [Yang91] [Yang94], the Mobility Direct (MD) [Wu90] and the Relative Mobility Scheduling (RMS) [Chan97a] algorithms, which are all based on heuristic, will produce reasonably good scheduling solutions in a homogeneous and dedicated processor environment. Nevertheless, in the realization of these scheduling solutions in practical environment, they are restricted and solutions cannot be promptly achieved. In this dissertation, two environmental restrictions in classical scheduling techniques were identified. They are:

1. the heterogeneous requirement of processors configuration and
2. the non-deterministic nature of available resources.

To overcome the first limitation, a heuristic technique called "Heterogeneous List Scheduling" is proposed [Chan97b] in Chapter 2. This generalizes heuristics in the List Scheduling Heuristic so that the design of task scheduling algorithms can also be applicable in a heterogeneous environment. In order to demonstrate the usefulness and the characteristics of the technique, three different scheduling heuristics are proposed and used for the demonstration. They are the Heterogeneous Dominant Sequence Cluster (HDSC) [Chan97c], the Heterogeneous

Mobility Direct (HMD) and the Heterogeneous Relative Mobility Scheduling (HRMS) [Chan97d]. Through experiments, these algorithms can schedule parallel tasks into heterogeneous environment in which data dependencies are satisfied in a short time.

The other challenging problem for task scheduling in practical environment is the non-deterministic processor's workload (non-deterministic resources). This non-determination comes from various factors including uncertainties in resources used by operating systems, multiple-users and multiple-applications [Lewis98] [Lam95a][Lam95b]. Research in the last decade had made an effort on the study of workload estimation techniques for dynamic scheduling during the run-time. Although these procedures can adapt to a non-deterministic environment for an improved scheduling solution, the non-determination problem still exists. Besides, the huge amount of overhead required in the workload estimation and the dynamic scheduling and rescheduling in the run-time will restrict the use of parallel and distributed computation in real-life applications.

Focusing on the limitation, a contractual resource management scheme for assisting scheduling problem is outlined in Chapter 3. The Contractual Computing paradigm (CC), a revolution computation and networking resource management scheme, is proposed by Lam and Li [Lam96][Lam97]. The concept is to modularize the management of computation as well as the networking resources into two different layers. Applications demanded for resources within the two layers are engaged themselves by making a resource contract between the layers. The contracting mechanism is similar to likes an advanced resource booking system. Once the resources are booked, the provider should guarantee the contracted resources to be available in run-time. Consequently, a deterministic execution becomes possible.

With the study of the paradigm, three approaches for designing the appropriate task scheduling algorithms have been determined in Chapter 5, 6 and 7 respectively. They are the "Scheduler Oriented" (SO) approach, the "Resources Oriented" (RO) approach and the "Contractor Oriented" (CO) approach. The aim of the first two approaches is to merge the classical scheduling algorithm into the one feasible contractual computing paradigm. The third approach uses the advanced scheduling information for making task-scheduling decisions. The implementations based on the HDSC, the HMD and the HRMS algorithm have been developed for these three approaches. Based on the implementation, the properties including the booking probability, the task scheduling performance, the efficiency in resources usage and their inherence problems have been studied. Consequently, the properties of the proposed scheduling algorithms were illustrated by extensive simulation studies.

# Chapter 1: An Introduction to Parallel and Distributed Computing

In the last three decades, startling advances in computing technology that were stimulated by the availability of faster, more reliable and cheaper electronic components have been witnessed. These resolute developments cultivated a wide range of solution for computational intensive problems [Shurkin84] [Wilkes85]. However, better devices albeit essential are not the sole factors contributing to the high performance. It shows that the performance of uni-processor computers is limited, because device technology places an upper bound on the speed of a single processor.

The evolutionary transition from sequential to parallel and distributed computing (PDC) offers the promise of quantum leap in the processing power that can be brought into solving on many important problems. Although this technology has risen to prominence during the last 20 years, there remain many unresolved issues. The field is in a state of rapid flux where advances are still being made on several fronts. PDC systems are recognized today as an important vehicle for the solution of many problems, especially those known as the "Grant Challenges" [NSF92] such as climate modeling, superconductor modeling and pollution dispersion modeling. These and many other applications generally require heavy processing power.

Theoretically, a parallel algorithm can be executed efficiently on a MIMD model [Albert96]. Thus, this model can be used to build parallel computers for a variety of applications. Such computers are said to have a general-purpose architecture. In practice, it is quite reasonable in many applications to assemble several processors in a configuration specifically designed for a particular problem. The examples of these systems are pipelining architectures in super-scalar processors, array processors, vector processors and neural networks processors. Thus, a number of specifically designed scheduling algorithms were proposed to schedule tasks executing in such specific architectures [Chan96] [Guan95]. Consequently, a computer that solves a particular problem efficiently may not be suitable for other purposes. The scalability and the portability of these systems are limited.

In order to improve the scalability, the portability and the provision of a cost effectively environment for using computer resources, "Distributed Computing" (or Clustering computing, a networked workstation cluster) is widely used in the recent PDC system. It is another popular classification of PDC systems by the degree of coupling among the different processing components within the system [Lawson92]. This paradigm enables multiple computers to cooperate simultaneously to solve a computational intensive problem. The processing elements can be wired directly together and are communicated via buses or cables, or they can be remotely located and are communicated by radio or microwave signals. Hence, the type and the mechanism for coupling can often be distinguished between a distributed processing and a parallel processing.

From the viewpoint of hardware, a distributed computing system is simply a collection of independent systems, typical workstations (called "processor" in latter text) and the connection via a commodity network. Figure 1.1 shows the configuration of the distributed environment setup for this study. Workstations communicate via connections oriented or connectionless transport protocols. General-purpose computers are installed with a multi-user and multi-tasking Operation System. All the processing units have their local memories, local I/O support and local schedules of the CPU time so that they can accept and execute tasks from different users simultaneously.

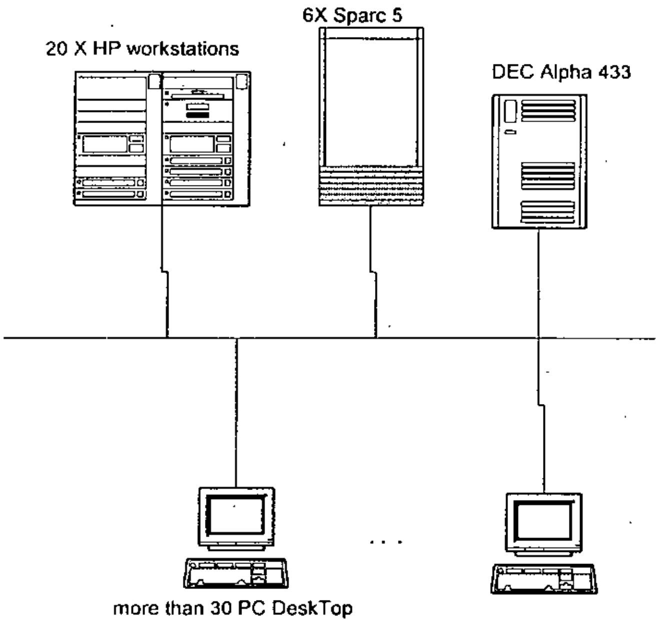


**Figure 1.1, an example of parallel and distributed system**

In order to run parallel applications efficiently in such kind of systems, software platforms like PVM [Beguelin91], P4 [Ralph92] [Ewing87], CHIMP [EPCC91] [EPCC92], or Express [Parasoft88] were developed. Among most of the software tools, they create a layer of abstraction between programmers and the suit of machines. Some providing explicit constructions needed to express synchronization and communications among the tasks within the application. Lastly, they reduce the development cycle of parallel applications by using parallel and distributed system (PDS).

Nevertheless, difficulties in designing and implementing applications in PDS are encountered. Firstly, the availability of efficient and reliable network is essential for achieving task-executing performance. Some algorithms can make excellent use of a certain network topology, however they might not fit well into another. Secondly, the distributed computing system by its nature will normally consist of heterogeneous systems. The number of processing units supporting the system varies greatly from a few to thousands. The capability of these processing units can range from a very simple and capable of performing a single floating-point operation to a very powerful unit, which can be considered as a computer in its own part. Lastly, computational resources are non-deterministic where computers composed of the distributed computing system are general-purposed computer workstations installed with a generic multi-users and multi-tasking operating system (like UNIX). The computational resources (e.g. CPU time) are competed among users and tasks. This result in the failure of providing resources necessary for the running applications. The non-deterministic resources are greatly restricted to the application of PDS in solving wide ranges of problems.

The prime attraction of the distributed computing is the ability to utilize idle resources distributed across an enterprise network. Several studies at national laboratories in the United States have indicated that the idle time of most workstations is more than 75 percent [Albert96]. The usage of these "free" computing resources is a primary issue of the so-called "enterprise clusters". The essential requirement to configure an enterprise cluster is the management software, which includes the implementation of *availability policies* [Albert96]. The Contractual Computing Paradigm (CC) [Lam95a][Lam96][Lam97], which is based on the availability policies in enterprise clusters, was proposed by Lam and Li to manage resources in a contractual discipline. In the proposal, all the resources (e.g. CPU) in the PDS are managed by a contractual mechanism. Once a contract of resources is made, the resources are guaranteed in any occasion of system loading. Therefore, information regarding to resources under management can be given explicitly to applications. Applications as well as task schedulers can know exactly what and when the resources can be available. Hence, a reliable and high performance parallel application becomes feasible to be scheduled and to be executed in a dynamic, parallel and distributed environment without performance degradation.

# Chapter 2: The Task Scheduling Problem

## 2.1 The Task Scheduling Problem and its Goals

The task Scheduling problem consists of a set of resources (e.g. CPU time) and a set of consumers (e.g. parallel applications) served by the resources according to certain policies [EL-Rewini94a]. Based on the "consumers and resources" model, the problem becomes how to find an efficient policy for managing such resources access. The use of the resources by various consumers in order to optimize some desired performance measures such as parallel time has to be considered. Accordingly, a scheduling system can be considered as a set of consumers, a set of resources and a scheduling policy as shown in Figure 2.1. In order to make a scheduling decision; the consumers have to submit the constraints of an application such as data dependencies between sub-tasks as well as their timing constraints. Besides, the resource provider (such as network O.S.) has to acknowledge the schedulers explicitly concerning the amount of available resources, such as the CPU resources. Using this information, the schedulers can exercise its policy to arrange tasks to execute into the target system to meet the desired scheduling goals.

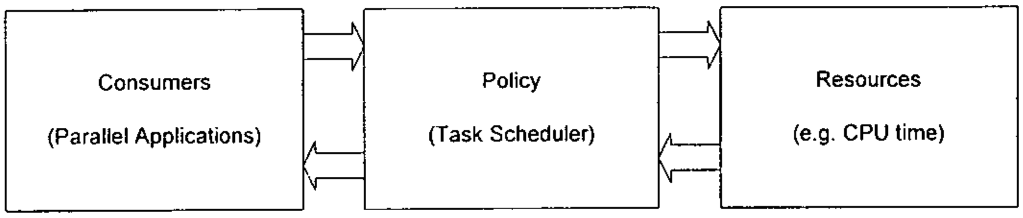


**Figure 2.1: A general scheduling system**

The scheduling problem was classified as one of the great challenging problems in parallel and distributed computing. It can be generalized by a MAX-MIN optimization problem [Kruatrachue88] that optimizes the maximization of parallelism and the minimization of communication overhead. There are three preliminary goals to measure task-scheduling performance [Yang94][CVRamam]; i.e.

**G1:** The complexity should be low;

**G2:** The parallel time should be minimized;

**G3:** The efficiency should be maximized.

Conflict may arise in order to meet all the above goals, G1, G2 and G3. For example, when there is a conflict in the maximization of efficiency with the minimization of parallel time, G1 is given priority over G2 and G3 and G2 over G3. It is obvious that finding an optimum solution of scheduling problem is NP-complete in its general form. In recent years, researchers are interested in algorithms that attain the minimum parallel time and the maximum efficiency in a low complexity.

## 2.2 Classical Task Scheduling Techniques

In the classical school of task scheduling techniques, scheduling problems are classified into static scheduling and dynamic scheduling and they are summarized as below.

### *Static Scheduling*

In static scheduling, the assignment of tasks to the processors is done before the execution of the program. Information regarding the task's execution time and the processing resource is assumed known at the time of compilation. A task is always executed on the processor to which it is assigned; that is, static scheduling methods are processors non-preemptive. The scheduling goals of the static scheduling methods [L088] [Sarkar86] [Shirazi90] [stonE57] are:

1. To predict the program execution behavior during compilation (that is to estimate the task, execution times and the communication delays);
2. To partition tasks into coarse-grain processes, in an attempt to reduce communication costs; and
3. To allocate processes to the processors.

The major advantage of static scheduling methods is that all overheads of the scheduling processes are incurred at the compilation time, resulting in a more efficient execution time environment compared to dynamic scheduling methods. However, static scheduling suffers from a wide range of problems. The most notable of which are as follows:

1. They are inefficient and inaccurate in estimation of task-execution time and communication delays, which can cause unpredictable performance degradations [Lee91] [Wang91]. The compile-time estimation of the task-execution time is difficult due to unknown prior conditionals and loop construction.
2. They are also unable to accurately estimate network traffic and computational resources workload, which again contribute to performance degradations. The compile-time estimation of the computer workload is often difficult, e.g. PDS, in which multiple-users are served with multiple-tasks. Estimating communication latency and processors workload at compile time is not feasible due to the unknown traffic conditions in the network and unknown processor workloads during the run-time.

However, static scheduling schemes can be augmented with a tool providing a performance profile of the predicted execution of the schedule on a given architecture. In this situation, the user cannot utilize this tool to improve the schedule nor to adapt to a different architecture.

### *Dynamic Scheduling*

Dynamic scheduling is an attempt to place tasks into the processors and to decide the task/sub-task's start time during run-time, such that the parallel program can finish as earliest as possible [El-Rewini94b]. The difference between dynamic load balancing and static scheduling is that dynamic scheduling needs to collect system parameters and perform rescheduling at run-time. The flexibility inherent in dynamic load balancing allows adaptation to the unforeseen variation in system loading before run-time. An optimization problem exists that is affected by the following factors:

1. The execution time and the communication delay for parallel program itself.
2. The uncertain of the availability of system resources at run-time: Although the dynamic scheduler collects the current and the past information of the system loading. It is uncertain that this information is valid at the run-time. Using the current and the past information to estimate the system loading during scheduling, the non-deterministic nature of resources can be reduced but they cannot be totally eliminated.
3. Overhead in rescheduling: The overhead on collecting information on the loading of the processors and the network traffic, decision-making of rescheduling tasks to the processors also causes communication delay.

***Optimum, Sub-optimal and Heuristic Scheduling***

One of the most critical shortcomings of scheduling problems is that generating an optimal scheduling solution is a NP-complete problem [Chret89] [Papadimitriou90] [Shakar89]. Alternatively, the research and development in this area have been focusing on sub-optimal scheduling techniques, which are based on approximation and heuristic approaches. In the approximation approach, the solution space is searched in either a depth-first or a breadth-first strategy. However, instead of searching the entire solution space for an optimal solution, the algorithm will stop when a "good" (or acceptable/predefined) solution is reached.

The heuristic approach, as the name indicates, relies on the rules-of-thumb to guide the scheduling processes in the right direction to reach a "near" optimal solution. For example, the DSC algorithm [Yang94] uses the length of a critical path in a task graph as a scheduling heuristic. It should be noted that there is no universally accepted Figure or standard for defining a "good" solution or a degree of "nearness' to an optimal solution. Researchers often use a loose lower bound on the execution time of a concurrent program (for example, the length of a critical path as a benchmark). They indicate that their methods can always achieve schedules with the execution time within a factor of this lower bound.

## 2.3 Task Scheduling in Practical Parallel and Distributed System

A number of obstacles associated with parallel applications and execution environments challenge researchers in designing, implementing and analyzing task-scheduling techniques in practical PDS. They include non-determinism in the application program. The execution environments are configured heterogeneously and they confirm non-determinism in execution environments.

***A) Non-determinism in the application program.***

The non-determinism comes from uneven grain size of tasks, lack of a priori knowledge of the number of iterations in loops and in variable program branching. The uncertainties that rule out a simple scheduling can be led to the notion of sophisticated scheduling algorithms, which greatly increase the time complexity and the amount of computational resource to complete the scheduling.

***B) The execution environments are configured heterogeneously***

As defined in previous sections, distributed systems are loosely coupled general-purpose computer systems running with multiple-users and operating system. In practice, the processing units are in different bands and different performances. Their processing powers are heterogeneous. Their communications are through a multiple-access and time-variant interconnected network such as Ethernet. The speed of communications among hosts is also heterogeneous. These will increase the complexity of the task scheduling algorithms and complicate their design.

***C) Non-determinism in execution environments***

Since computational and communicational resources are consumed by multiple users and multiple applications, the exact loading of the computer and the network is changing dynamically in a practical environment. The task scheduler has to collect this information for proper scheduling. However, this information is not provided in most networks available O.S. and a huge amount of effort is required to collect in dynamic scheduler. Even with this available information, it does not guarantee that the required resources are available for the scheduling application. The non-determinism in the computational and communicational environments still cannot be overcome.

## 2.4 Problem Definition

To start the List Scheduling Heuristic, the representative task scheduling technique in classical scheduling theory, was studied. It was then implemented in a practical execution environment and was found that the heterogeneous resources and non-deterministic resources are the most obvious restrictions to the practical implementation[1]. Designs and implementations of

[1] Although the non-determinism in program execution is a factor that leads to performance degradation, there were several good techniques proposed by researchers to minimize the non-determinism. For examples, high precision prediction algorithms for task execution time and data communication delay, branch prediction and dynamic scheduling in loops and branches techniques. By using these techniques, the non-determinism in program execution can be improved significantly. In order to simplify the analysis and isolate the negative effect of non-determinism in program execution introduced to task scheduling performance, the non-determinism in program execution is ignored. In other words, the parallel programs to be scheduled by the scheduler under analysis are assumed deterministic in the task execution time and the data communication delay.

task scheduling techniques a heterogeneous in resource configuration with execution environment was studied and reported in the next Chapter. Simulation studies were carried out to verify and compare their properties and functionality.

A novel computation paradigm, the "Contractual Computing" paradigm, was studied. With it, a stable and deterministic execution environment is provided. Three different approaches of implementing task-scheduling algorithms for the "Contractual Computing" paradigm were proposed in the latter Chapter. Simulation studies were also carried out to evaluate the properties of the proposed approaches and to validate the corresponding claims. A remarkable approach for task scheduling in real-life parallel and distributed systems was concluded.

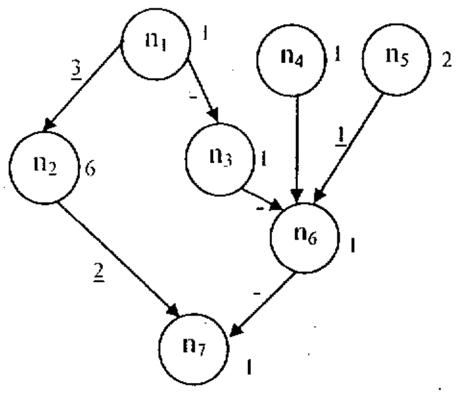
## 2.5 Assumptions

It is assumed that applications to be scheduled can be represented in directed acyclic weighted task graphs (DAG's). An example of DAG is shown in Figure 2.2. [Yang94] The execution time is presented in the right side of the bullets and the corresponding edge weight is written on the edges. A DAG is defined by a *tuple G = (N, E, C, W)*, where $N$ is the set of task nodes and $n = |N|$ is the number of nodes, $E$ is the set of communication edges, $e = |E|$ is the number of edges, $C$ is the set of edge communication cost and $W$ is the set of node computation costs. It is assumed that the task and edge weights are deterministic. The value $c_{ij} \in C$ is the communication cost incurred along the edge $e_{ij} = (n_i, n_j) \in E$, which is zero if both nodes are mapped onto the same processor. The value $w_i \in W$ is the execution time of node $n_i \in N$. An $PRED(n_x)$ is the degree of precedent level of the task. The scheduling problem consists of two parts: the task to processor assignment, called *clustering* and the task execution ordering each processor. A scheduled DAG is a DAG with a given clustering and task execution ordering. A clustered graph is a DAG with a given clustering, but without the task execution ordering. The critical path (CP) of a clustered graph is the longest path in that graph, including non-zero communication edge cost and task weights in that path. The parallel time in executing cluster DAG is determined by the critical path of scheduled DAG, is not by the critical path of the clustered DAG. The critical path of scheduled DAG is called the dominant sequence (DS) as compared with the critical path of the clustered DAG. For example, Figure 2.2a shows a typical DSC and Figure 2.2b shows a schedule of the task graph. For Figure 2.2c, the critical path of the clustered graph is $\langle n_1, n_2, n_7 \rangle$ and the DS is still that path. If the weight of $n_5$ is changed from 2 to

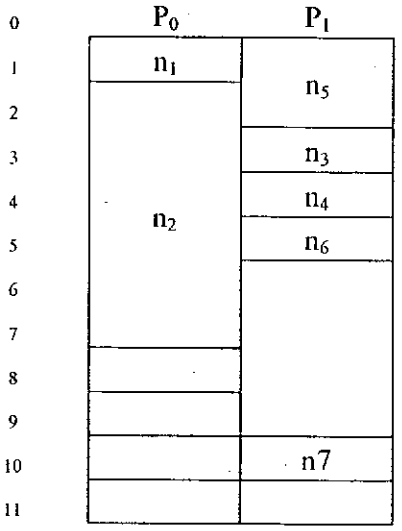
6, then the critical path of the clustered graph remains the same, $\langle n_1, n_2, n_7 \rangle$, but the DS changes to $\langle n_5, n_3, n_4, n_6, n_7 \rangle$. Let tlevel($n_x$) be the length of the longest path from an entry (top) node to $n_x$, excluding the weight of $n_x$ in a DAG. Symmetrically, let blevel($n_x$) be the length of the longest path from $n_x$ to an exit (bottom) node. For example, in Figure 2.2c, tlevel($n_1$)=0, blevel($n_1$)=12, tlevel($n_3$)=2, blevel($n_3$)=5.

The task execution model is a macro-dataflow model. A task receives all the input before starting execution in parallel executes to the completion without any interruption and it immediately sends the output to all the successors tasks in parallel [Wu90][Sarkar89]. Moreover, task duplication in task scheduling is not allowed. Unless specified, the target environment of the scheduling problem is a network of limited/unlimited number of heterogeneous general purpose computers installed with a platform under the specification of contractual computing paradigm. Besides, it is assumed without lost of generality that these computers are fully connected by using a delicate network with unified communication rates among computers.

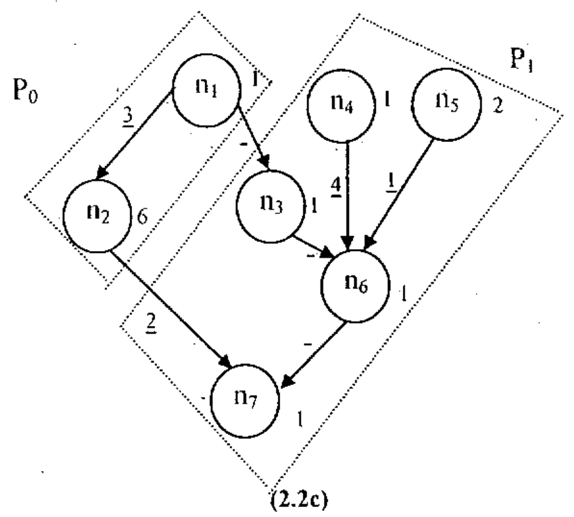
The contractual computing is a novel concept in the parallel and distributed computing technology. A platform designed upon the specification of the paradigm called "IECC" is under developed by fellow researchers in related projects in our group. During the study, a development platform called "Local Manager" is served as a blueprint for the design. The Local Manager is addressed to the processing power management by modifying the processes scheduling inside the LINUX kernel so that the processing power can be booked in advance. The other resources such as network traffic and memory usage will be implemented in the second phase.



(2.2a)



(2.2b)



(2.2c)

**Figure 2.2a-c. An example of a DAG,**

**(2.2a) A weighted DAG. (2.2b) A Gantt chart for a schedule, (2.2c) A scheduled DAG**

It is assumed that the target machines have enough memories and storage spaces and the communication links are of uniform and deterministic. In other words, it is free from external interference.

In summary, it can be concluded that the processing power is the only resource that can be scheduled and booked, the resources other than processing powers are assumed deterministic and dedicated and the term "Resources" is restricted to the processing power only. The task-scheduling problem was described and the obstacles ahead were studied. It was found that the scheduling performance of existing scheduling techniques was restricted by two major factors: heterogeneous nature of system configurations and non-deterministic in the availability of resources. Therefore, the main objective of this research is to overcome these two problems.

# Chapter 3: Task Scheduling Techniques for Heterogeneous Environment

## 3.1 Chapter Summary

This Chapter studied the techniques for scheduling tasks in the heterogeneous computing environments, beginning with the list scheduling heuristic. The Dominant Sequence Clustering (DSC) and the Mobility Direct (MD) algorithms were investigated to improve the task scheduling solution (shorter parallel time). Then, an improved algorithm called Relative Mobility Scheduling (RMS) algorithm was proposed which is a MD algorithm with a modified task assignment strategy. Such improvement was demonstrated by extensive studies (to be presented in Chapter 8, Experiment 1) that scheduling solution with shorter parallel time than that of the MD algorithm was observed. The proposed algorithm can also improve resources utilization rate with an unbounded number of processors.

The above algorithms were implemented in a practical environment using PVM. However, implementation difficulties were encountered in a heterogeneous environment. Using heuristics, the Heterogeneous List-Scheduling techniques (HLS) was proposed for the design of task scheduling algorithms to be implemented in a heterogeneous environment. Applying the same philosophy, any task scheduling heuristics used in the list scheduling can be modified for the heterogeneous environment.

To study the functionality and the properties of the proposed technique, the DSC, MD and RMS algorithms were implemented in heterogeneous forms as HDSC, HMD and HRMS algorithms. The functionality and the properties were studied through extensive simulation studies, which will be shown in Chapter 8.

## 3.2 The List Scheduling Heuristic (LSH)

List Scheduling is a class of scheduling heuristics in which tasks are assigned in order of priorities and placed in a list of descending order. Whenever tasks contend for processors, the one with the highest priority is being selected for scheduling. In case of more than one task of the same priority, ties are broken randomly. A generic procedure of list scheduling Algorithm 3.1 is given as below

```
Step 1. Each node in the task graph is assigned a priority. A priority queue is
        initialized for the ready tasks. The other tasks that have no immediate
        predecessors are inserted. The tasks are sorted in a the descending order
        of priorities,
Step 2. As long as there is a priority queue, do the following:
        2.1. Obtain a task from the front of the queue.
        2.2  Select an idle processor to run the task.
        2.3. When all the immediate predecessors of a particular task are
             executed, that task is called free task (or ready task) so that it can be
             inserted into the priority queue.
Step 3. Repeat Steps 1 and 2 until all tasks are scheduled.
```

**Algorithm 3.1. The List Scheduling heuristic.**

With no considering any communication overhead, this type of heuristic is appropriate for a shared-memory parallel processor environment with messages passing at memory-cycle speeds. A number of investigation using different heuristics were reported and they are the HLEFT (Highest Levels First with Estimated Times), the HLFNET (Highest Levels First with No Estimated Times), the RANDOM, the SCFET (Smallest Co-levels First with Estimated Times) and the SCFNET (Smallest Co-levels Fist with No Estimated times heuristic) [El-Rewini94a]. Adam Chandy and Dickon [ACDIi74] conducted an extensive empirical performance study of the above heuristics, which perform schedules without consideration any communication overhead. The results of the study showed that among all the priority schedulers, level priority is the best at getting closer to the optimal solution. Highest Level First (HLF), which is also known as Critical Path (CP), is superior than the others for it provided schedulers within 5 percent of the optimum in a 90 percent of random cases (The random case means random generated task graph).

However, in MIMD distributed-memory parallel computers such as the network and workstations, the communication overhead cannot be ignored when compared with shared memory systems. There are two problems associated with the inherent of communication overhead. They are the (A) Parallelism versus communication delay and (B) level alteration.

## *A) Parallelism versus communication delay*

When there is no communication delay among the tasks, all the ready tasks can be allocated to all the available processors so that the overall execution time of the task graph can be deduced easily. In a possible situation, a shared memory parallel processor system with messages passing at a memory-cycle speed. No communication overhead is being considered. However, in real-life situation, communication delay must be taken into account before each processor is ready for execution. The possibility of the ready tasks with long communication delay being assigned to the same processor as their immediate predecessors may occur.

This situation was described by El-Rewini and Lewis [El-Rewini94a] and shown in Figure 3.1a-c. Figure 3.1a shows a task graph being scheduled. Figure 3.1b indicates that the start time of task 3 on $P_2$ is later than its start time on $P_1$ due to the communication delay, $D_x$ of task 3, which is greater than the execution time of task 2. So task 3 should be assigned to $P_1$, that has its immediate predecessor. Conversely, as shown in Figure 3.1c, if the communication delay $D_x$ is less than the execution time of task 2, task 3 should be assigned to $P_2$ instead.

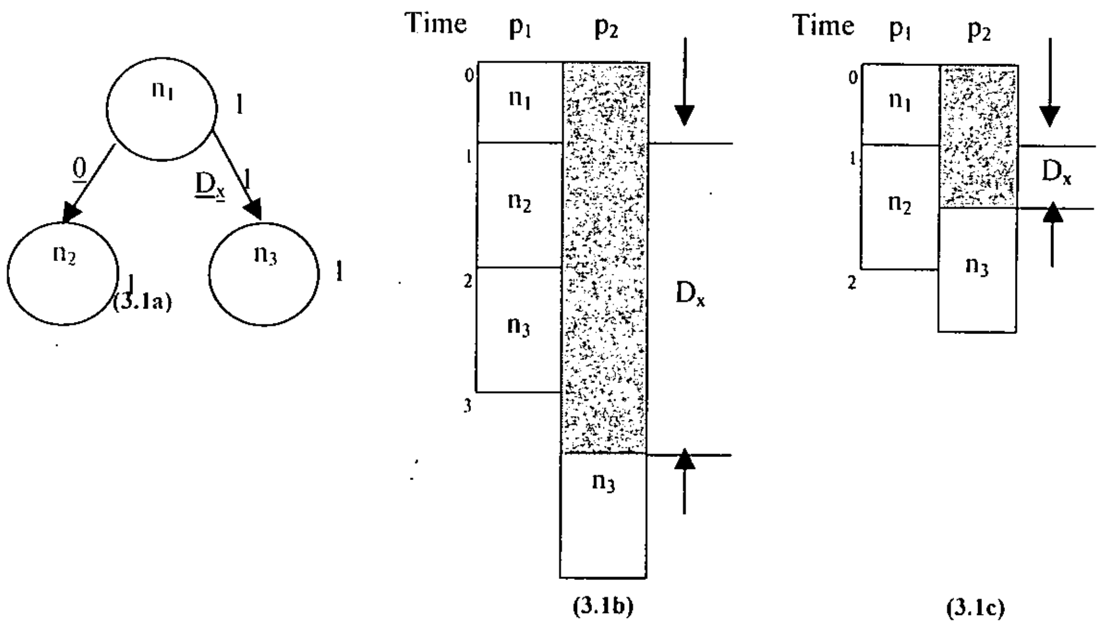


**Figure 3.1a-c. Scheduling consideration due to communication delay.**
**(3.1a) A simple Task Graph, (3.1b) Gantt chart with larger $D_x$ > execution time of $n_2$,**
**(3.1c) Gantt Chart with smaller $D_x$ < execution time of $n_2$.**

It is found that with addition communication delay constraint, the difficulty of arriving an optimal schedule in which the scheduler must examine the starting time of each node on each available processor in order to obtain the best one, will increase. It would be a mistake to start each task as soon as possible in order to improve the parallelism. Distributing parallel tasks to as many processors as possible tends to increase the communication delay, but it improves the overall execution time. In short, there is a trade off between the maximizing parallelism and minimizing communication delay. This problem is called the MAX-MN problem [Kruatrachue88] in parallel processing.

The dramatic effect of the MAX-MIN problem is further demonstrated in Figure 3.2a-d. If the communication delay $D_3$ between task 1 and task 3 is less than the execution time of task 2, task 3 is then assigned to $P_2$ to start its execution sooner. Because task 2 and task 3 are the immediate predecessors of task 4 and they are assigned to different processors, task 4 cannot avoid the communication delay from one of its immediate predecessors. Thus, the execution time of this task graph is equal to the summation of the execution time of tasks 1, 2, 4, plus communication delay $D_x$ or $D_y$ depending on the assignment of the task 4.

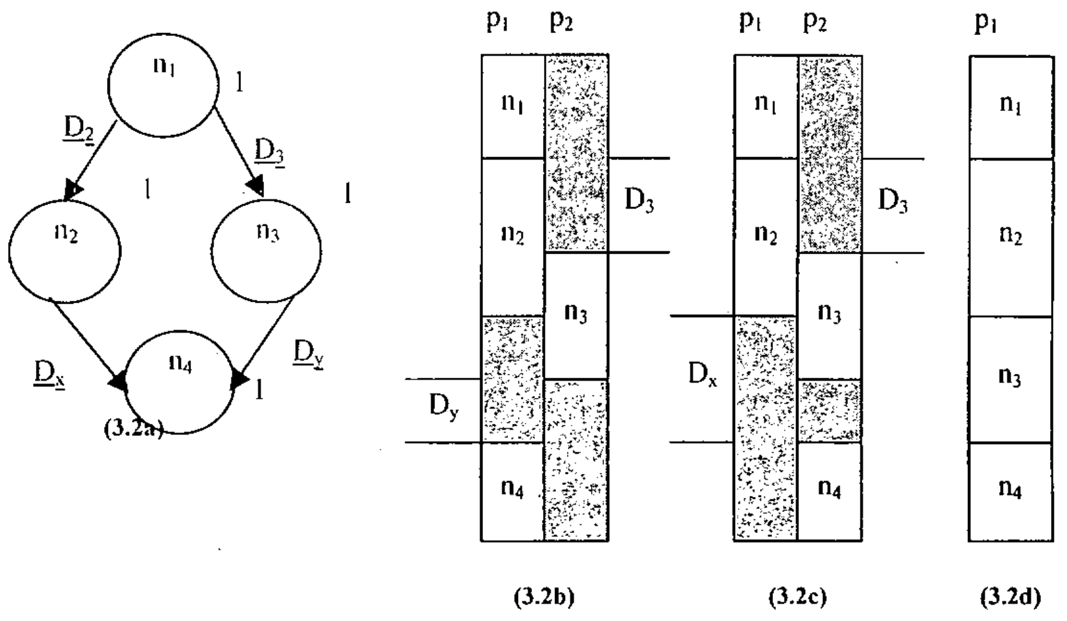


**Figure 3.2a-d. Trade-off between parallelism and communication delay.**
**(3.2a) A simple task graph. (3.2b) A schedule A. (3.2c) A schedule b. (3.2d) A schedule c.**

However, if the communication delay of task 4 is longer than the execution time of task 3, task 3 is assigned to $P_1$ resulting in a shorter task-graph execution time. This happens even if task 3 finishes its execution later than the previous assignment as shown in Gantt chart shown in Figure 3.2c. Current communication delays scheduling heuristics try to take the advantage of parallelism and reduce the communication delay. One solution is to start the tasks earlier. A method for solving the MAX-MIN problem is by duplicating tasks [Kruatrachue87] where necessary to reduce the overall communication delay and maximize parallelism at the same time. To avoid over complexity of the analyze of the scheduling algorithms, the duplication of tasks will not be considered.

### *B) Level Alteration*

Another important problem due to non-zero communication delays is the alteration in task levels and their impact on critical path calculation. Any heuristic that uses the level numbers or the critical path length faces this same problem. The level of a node is defined as the length of the longest path from that node to the exit node. This length includes all the node execution times and all the communication-delay times along the path. Unfortunately, the level numbers do not remain constant when the communication delays are considered because the level of each node changes as the length (measured in the unit of normalized time) of the path leading to the exit node changes. The length varies depending on the communication delay while the changes of the communication delay depend on the task allocation. The communication delay is zero if tasks are allocated to the same processor and is nonzero if tasks are allocated to different processors. In addition, the number of hops between the processors makes in computing the communication delay in portion of the level, the level number problem arises.

## 3.3 Implementation Consideration of the List Scheduling Heuristic

The characteristic of the list scheduling heuristic is flexible in heuristic design and simple in implementation, so that it was widely adopted in modern PDS. In this research, it has been used as a foundation for all the studies. There are many research studies on different heuristics in priority values assignment and heuristic in task to processor assignment for designing scheduling algorithms under the list scheduling heuristic. [McCreary94], [Yang94] and [El-Rewini94a-c] conducted various research studies on different heuristics used in priority values generation. It was included the largest edges cost first [Sark89], Critical Path [Yang94], Modified Critical Path [WuGa88] and Relative Mobility [WuGa88]. They conclude that different heuristics perform differently depending on the granularity and the structure of scheduling task graph. The heuristic of Critical Path provides better scheduling solution among the others while scheduling task in Direct Acyclic Graph (DAG). The solution comes from Relative Mobility, Modified Critical Path and the largest edges cost first heuristics degrade gradually. The reasons of the Critical Path are always used as the reference for accessing scheduling algorithms.

Different strategies for task to processor assignment were conducted by Kruatrache [Krua87]. A comparison of NISH [1], ISH0[2], ISH1 and ISH2 and DSH[3]. The simulation results showed that on average, the descending order of speed up is DSH, ISH2, ISH1, NISH and ISH0. However, the DSH should pay extra cost and resources for duplicating a task execution and high effort in determining the task to be duplicated. Owing to this reason, the ISH2 has the best overall performance in terms of complexity and resources utilization.

The performance of task scheduling solution of List Scheduling heuristic is highly dependent on the heuristic used in priority values generation and task to processor assignment. The combination of heuristic used in the two portions of the list scheduling heuristic would result in different designs of task scheduling algorithm in which they have different performances. In the

[1] Non-Insertion Scheduling Heuristic

[2] Insertion Scheduling Heuristic generation 0 to 3; ISH0 (a task was assigned to the first contact processor that executed among the idle time slots and it executed in other processors easily), ISH1 (A task was assigned within the idle slot of a processors that a message ready time was no sooner than the idle-time start time so that no other processor could process earlier), ISH2 (the same criterion ISH1 expected the absolute idle time slot)

[3] Duplication-Scheduling Heuristic.

following study, the Dominant sequent Clustering algorithm (DSC) [Yang94], the Mobility Direct algorithm (MD) [WuGa88] and the Relative Mobility Scheduling (RMS) [Chan97a][Chan97b] algorithm are studied in details. These algorithms use two distinct sets of heuristic priority value generation and task to processor assignment that are summarized in Table 3.1. Owing to the differences, the effect of different heuristic used for scheduling task in practical environment can be observed. A brief description of the algorithms is presented in Sections 3.3.1 to 3.3.3.

| | DSC | RMS | MD |
|---|---|---|---|
| Heuristic of Priority | Critical Path | Relative Mobility | Relative Mobility |
| Heuristic of task to processors assignment | Earliest finish time through minimization procedure (Consequent of ISA2) | Processor satisfies FACT 1 with earliest finish time (Consequent of ISA2) | First processor that satisfies FACT 1 (Consequent of ISA1) |
| DS task first | Yes | Yes | Yes |
| Join/Fork | Optimal | Optimal | Optimal |

**Table 3.1. A survey on task scheduling heuristics**

### 3.3.1 The Dominant sequence Clustering algorithm (DSC)

The DSC algorithm was originally proposed by Tao Yang [Yang94]. The main idea is to perform a sequence of edge zeroing steps with the goal of reducing the length of the dominant sequence (DS) in each step. An algorithmic description of the initial design of DSC was given as

```
1. Let $EG = NULL, UEG = V$ where $EG$ = Examined Graph, $UEG$ = Un-
   Examined Graph and $V$ is the set of tasks in the scheduling task graph.
2. Compute $blevel(n_x)$ for each node $n_x$ and set $tlevel(n_x) = 0$ for each entry
   node.
3. Every task is marked unexamined and assumed to constitute one unit
   cluster.
4. While $UEG \neq NULL$
    4.1 Find a free node $n_f$ with the highest priority (i.e.
        $\max\{blevel(n_f) + tlevel(n_f)\}$) from $UEG$.
    4.2 Merge $n_f$ with the cluster of one of its predecessors so that $tlevel(n_f)$
        decreases in a maximum degree. If all zeroing increase $tlevel(n_f)$,
        $n_f$ remains in a unit cluster.
    4.3 Update the priority values of $n_f$'s successors.
    4.4 $UEG = UEG - \{n_f\}; EG = EG + \{n_f\}$
   End While
```

**Algorithm 3.2. The initial design of DSC algorithm.**

Although the algorithm adopts clustering idea in task scheduling, it can be classified as subset of list scheduling heuristic. It is observed that the algorithm uses the sum of top and bottom level to generate the priority list, where top level of a task $n_x$ (tlevel($n_x$)) is the length of the longest path from an entry (top) node to $n_x$ excluding the weight of $n_x$ in a DAG and the bottom level of a task $n_x$ ($blevel(n_x)$) is the length of the longest path from $n_x$ to an exit (bottom) node. The list is formed with the highest priority free task is selected to a processor with $tlevel(n_f)$ decreasing in a maximum degree. In other words, this is to find a processor that can execute the task in the earliest time.

The DSC algorithm was proved to be optimal for scheduling the fork and joining DAG graph [Yang94]. It was further proved that the algorithm can produces a scheduling solution with a minimum parallel time after consecutive scheduling iterations and can produce near optimum scheduling solution. However, it was pointed out that the algorithm suffers from zeroing non-DS edges because of the topological ordering of the traversal. That is, when a DS node $n_y$ is partially free, DSC suspends the schedule of $n_y$ and examines the current non-DS free nodes according to a priority-base topological order. Hence, scheduling task in non-DS result does not guarantee to produce parallel time that is a minimum. To overcome the difficulty, an improved form of DSC algorithm is proposed. In the proposal, a partial free list (PFL) is maintained to support the priority list and to impose Dominant Sequence Length Reduction Warranty (DSRW) constrain to eliminate the situation. The algorithmic description of the improved algorithm is shown in Figure 7 and 8 of [Yang94]. However, such change complicated the algorithm in comparing with other heuristic scheduling design. We do not take into consideration in this discussion.

### 3.3.2 The Mobility Direct algorithm (MD)

The MD algorithm [WuGa88] uses relative mobility as heuristic of priority list generation. The "relative mobility" ($M_r$) of a node is defined as $M_r(n_i) = [T_L(n_i) - T_S(n_i)]/w(n_i)$, where $w(n_i)$ is the task weight, $T_S(n_i)$ is the as-soon-as-possible (ASAP) starting time of a task $n_i$, $T_L(n_i)$ is the as-late-as-possible (ALAP) time of a task $n_i$ and $T_F(n_i)$ is the latest finishing time. The relative mobility reflects the relative moving range of a scheduling task in the priority list. Hence, it is to identify the importance of a task inside the list. For instance, a task with zero $M_r$ implies that the task should start its execution without suffering any form of delay. Hence, it is referred as critical tasks in critical path of the scheduling task graph. On the other hand, tasks with $M_r$ greater than zero is the non-critical tasks. They suffer delay in starting execution. In which a priority list generated by sorting the $M_r$ in descending order for prioritizing those tasks to be scheduled in each scheduling step is considered.

The DSC algorithm is similar to that of the MD algorithm in the sense that the smallest $M_r$ identifies a task that have maximum sum of top and bottom level, but, the way to identify DS task is different. The DSC uses a priority function with $O(\log v)$ computing scheme, whereas MD uses the relative mobility function with a computing cost of $O(v+e)$. Another difference is that when DSC picks a task $n_i$ to be scheduled, it reduces the top level of this task and thus decreases the length of DS going through this task. On the other hand, the MD scans the processors from left to right searching the first processor satisfying Fact 1 [WuGa88].

The strategy of assigning the selected task to processors was studied and published in [Chan97b] that the argument of finding the first processor from the given processors set in which satisfies the Fact 1 in MD algorithm indeed can be able to suggest the minimum number of processors to execute the parallel application with non-increase of current critical path guarantee. But it does not imply shortening of the path length. Unnecessary delays of parallel time may be accumulated in each scheduling step. This lowers the efficiency of processors' utilization so that parallelism of the task graph cannot be maximized. Hence, the Relative Mobility Scheduling algorithm is proposed to improve the MD algorithm [Chan 97a][Chan97b] and is described in the following section.

### 3.3.3 The Relative Mobility Scheduling algorithm (RMS)

The proposed Relative Mobility Scheduling algorithm (RMS) [Chan97d] is a more aggressive version of the MD algorithm. In which the heuristic used in the MD algorithm is modified where the task is scheduled not only satisfy Fact 1, but also can complete in the earliest time. In this way, the path length in each scheduling step can be further minimized. Thus, the capacity of the target processors can be fully utilized and the parallel time of the scheduling task graph is maximized realistically.

An algorithmic description of the RMS algorithm is described in algorithms 3.4a and 3.4b.

```
1. Calculate the relative mobility of all tasks.
2. Let $L$ include all tasks initially. Let $L'$ be the group of tasks in $L$ with minimum
   relative mobility. Let $n_i$ be a task in $L'$ that does not have any predecessors in $L'$.
   Find a processors $P_m$ in the processors set $P$ in which $n_i$ would satisfy Condition 1
   and complete in minimum time (or the earliest finish time is reached). When $n_i$ is
   scheduled on $P_m$, all the edges connecting $n_i$ and other tasks already scheduled
   to $P_m$ are changed to zero. If $n_i$ is scheduled before task $n_j$ on $P_m$, add an edge
```

with zero cost from $n_j$ to $n_i$ in the graph. If $n_i$ is scheduled after node $n_j$ on $P_m$, add an edge with zero cost from $n_j$ to $n_i$ in the graph.

3. Recalculate the relative mobility of the modified graph. Delete $n_i$ from $L$ and Repeat Step 2 until $L$ is empty.

**Algorithm 3.4a. The Relative Mobility Scheduling (RMS) algorithm**

Assuming a task $n_i$ is considering to be scheduled to $P_m$, in which $l$ tasks, $n_{m1}, n_{m2}, \ldots n_{ml}$, have been scheduled into $P_m$. If the moving intervals of these tasks do not intersect with the moving interval of $n_p$, then $n_p$ can be scheduled to $P_m$ Otherwise, assume the moving intervals of tasks $n_{mi}, n_{mi+1}, \ldots n_{mj}$ $(1 \le i \le j \le l)$ intersect the moving interval of $n_i$. $n_p$ can be scheduled to $P_m$, if there $k (i \le k \le j+1)$ exists,

$$W(n_i) \le \min\{T_F(n_i), T_L(n_{mk})\} - \max\{T_S(n_i), T_S(n_{mk-1}) + W(n_{mk-1})\}$$

where W($n_i$) is the cost of $n_i$; and the $T_F(n_i)$ and $T_S(n_i)$ are the latest finishing time and the latest starting time of a task $n_i$ respectively; if $n_{mi-1}$ does not exist, $T_S(n_{mi-1}) = 0$, $W(n_{mi-1}) = 0$; if $n_{mj+1}$ does not exist, $T_L(n_{mj+1}) = \infty$.

Otherwise, the task cannot be scheduled to $P_m$.

**Algorithm 3.4b. Condition 1 of the RMS algorithm**

The most significant improvement of the RMS algorithm over the MD algorithm is the task to processor assignment strategy. The RMS algorithm schedules a task into a processor that can be finished earliest. That is similar to heuristic suggested in ISH2. Experiment 1 in Chapter 8 shows that the scheduling performance between the MD and the RMS algorithm. It is found that the scheduling performance of the RMS algorithm is better than the MD algorithm in most circumstances. It is found that the RMS algorithm produces schedule with 48% shorter in parallel time than the MD algorithm in different granularity of task. Not less than 90% ($p_r \ge 0.9$) cases of schedule produced by RMS algorithm is shorter than that of the MD algorithm.

As compare with the MD algorithm, the RMS tries to explore the given set of bounded (or unbounded) processors for executing a scheduling task in the earliest time. In this occasion, the number of processor required by the schedule of the RMS algorithm may not less than the minimum number of processors required by the MD algorithm. The scheduling performance as measured by the sum of total processing power may not as good as the MD algorithm. However, if the same number of processors is used to scheduling task by RMS and MD algorithms, the parallel time of schedule produced by the RMS algorithm could better than that of the MD algorithm (Experiment 1) in most situation (over 90%). It is concluded that that the change in RMS algorithm can produce scheduling solution better than the MD algorithm does.

### 3.3.4 Implementation experiences

The List Scheduling Heuristic, the studied algorithms (the DSC and the MD algorithm) and the proposed algorithm (the RMS algorithm) were implemented in an ideal execution environment through simulation and a practical environment under PVM [Beguelin91]. It is found that the list scheduling heuristic and those designed based on the list scheduling heuristic blueprint unable to work in practical distributed systems. The two major obstacles in obtaining the prompted scheduling performance of the algorithms are (i) heterogeneous configurations in execution environment and (ii) non-deterministic resources. The proposed algorithms cannot be applied in a heterogeneous environment because the assumptions made in the design are based on homogeneous situations and cannot be directly plotted into its heterogeneous counterpart. The different is highlighted as the following,

#### *A) To implement in the homogeneous environment*

The design of the list scheduling heuristic is based on the homogeneous environment, the computation speeds among the processors and the communication rate are assumed uniform. In this, the priority values such as level priority, co-level priority, critical path and relative mobility are directly obtained from data dependence constrains. As uniform computation speed and communication rate, the priority values measure their determination is obtained by a common value of reference. Consequently, the priority values can be calculated in the same cost model ("cost model" is defined as the method for describing and determining the task cost or edge cost in a scheduling task graph). Therefore, they can be computed directly for priority list generation and task selection. The priority list is meaningful and objective enough to identify an important task to be scheduled. Moreover, homogeneous environments also simplify the processes of determining the best processors to run the selected task in Step 2.2 of Algorithm 3.1. This is because the selected task executed in any processors or the data communicated in any communication channel are in same speed and rate, data dependence constrains are preserved in any assignment. The data dependence constrains can be determined and the task assignment decisions can be made directly from the scheduling task graph.

#### *B) To implement in the heterogeneous environment*

There encounters difficulties in implementation those scheduling solutions or scheduling algorithms in the heterogeneous environment. In general, a heterogeneous environment is a system containing processors configured in different computation speeds and communication links configured in different communication rates. In real-life, the configuration of a heterogeneous environment could be of bound number of processors. However, most existing scheduling algorithms like the DSC and MD are only applicable with unbound number of processors. These algorithms cannot be directly applied to heterogeneous environment. Even for a designed, with which the number of clusters less than that of the configured processors, it is still different perform to cluster processor are assigned. This is because in a heterogeneous environment there contains different grades of processors with resent to processing speed and communication rate. Dependence constrains maintained by the homogeneous environment is no longer valid in the heterogeneous counterpart. Furthermore, the set of parameter for obtaining the optimum MAX-MIN and level alteration cannot be maintained. This is because different grades of processors will validate the assumption made in the MAX-MIN assumptions. Owing to the heterogeneity, the scheduling solution produced by existing algorithms cannot be directly used in the heterogeneous counterpart. The prompted scheduling performance cannot be maintained in the practical execution.

It is also not possible to apply to the List Scheduling Heuristic directly to task scheduling in the heterogeneous environment. This is because data dependence among the tasks not only rely on the entry task graph, but also depend upon the task allocation to be scheduled. The dependence constrains cannot be directly determined from the task graph and it is difficult to decide priority values to Steps 1, 2.1 and 2.3 in Algorithm 3.1. The heterogeneous environment makes the decision on finding the best processor assignment in Step 2.2 of Algorithm 3.1 very complicated. The reason is that the dependence constrain would directly be affected by task assignment. The dependence constrain cannot be directly extracted from the task graph. Large amount of computation cost is required to re-calculation the dependence constraint in Step 2.2. Hence, the list scheduling heuristic is not suitable to be used directly in the heterogeneous environment. The Heterogeneous List Scheduling technique (HLS) is proposed to solve the problem and is prepared in the next section.

## 3.4 The List Scheduling Heuristic in Heterogeneous Environment

### 3.4.1 The Heterogeneous List Scheduling Heuristic

The List Scheduling heuristic is modified to be applicable in the heterogeneous environment. This novel technique "Heterogeneous List Scheduling (HLS)", is developed and presented in this section. An algorithmic description of the technique is given as Algorithm 3.5, which can be classified as a subset of a generalized List Scheduling Heuristic design procedure using two different phases, which is summarized as:

```
I.   Set up a priority list without candidates initially
II.  As long as the tasks in the scheduling task graph are not scheduled, do the
     following:
     Phase 1:  Task selection for scheduling (Priority list generation)
```

```
        1.1 Each node in the task graph is assigned a priority. The priority list
            is updated for unscheduled free tasks by inserting every task that
            has no immediate predecessors. They are sorted in the descending
            order of priorities. (To determine priority for those tasks, which
            have been scheduled, they are determined by the actual scheduled
            processor speed and communication rate. On the other hand, for
            those unscheduled tasks, they are determined by assuming the
            target processors speed and target communication links rate are in
            normalized uniform speed and rate as a reference of measurement
            respectively)
        1.2 A task is selected from the front of the queue.
Phase 2: Task assignment to processors (To make decision on task to
         processors assignment).
        2.1 Select "the best" idle processor to run the task. (The decision is
            made by considering the actual speed of assigning processor and
            the rate of communication link)
        2.2 When all the immediate predecessors of a particular task are
            executed, that successor is called "free task" and can be inserted
            into the priority queue.
```

**Algorithm 3.5. The Heterogeneous List Scheduling technique (HLS).**

### 3.4.2 Phase 1: Task selection for scheduling (Priority List generation)

This phase subjects to Steps 1, 2.1 and 2.3 of the Algorithm 3.1. A priority list of free tasks is generated by using heuristic of priority values based on primitives, such as relative mobility [WuGa88] or critical path [Yang94]. Then, the highest priority task is selected for assigning to a processor. As mentioned in sections 3.3.4, these steps are naturally be implemented in homogeneous environments. However, it encounters with the problem of non-deterministic speed in processors and communication rate in determining task priority in a heterogeneous environment. Therefore, a priority list cannot be generated. According to the HLS shown in Algorithm 3.5, this problem is overcome by using heuristic approaches.

The reason behind the difficulty is that the essential information on determining priorities is the method for describing the cost of task ("Cost model") in a scheduling task graph. In the homogeneous environment, a "cost model" is compiled with the properties that the processing speed and the communication rate are uniform. Therefore, the priority value can be directly obtained from the organization of the scheduling task graph and it is not affected by the execution environment before and after the tasks is scheduled. This can subjectively compare with the urgency of selecting those ready tasks to be scheduled. Unfortunately, the above properties are not valid in a heterogeneous counterpart. This is because the "cost model" is not only dependent on the organization of the scheduling task graph, but also dependent on the actual placement of tasks in the execution environment. The cost for those unscheduled tasks cannot be found without knowing their actual schedule.

In the HLS, a referenced uniform value of processing speed and communication rate is used to measure the cost of those unscheduled tasks. Since the references are scalable, any values of processing speed and communication rate from the set of target processors and communication link can be selected. It is found that using the slowest processing speed and the slowest communication link rate as the reference values can facilitate the determination of dependence constrains. Thus, the slowest processing speed and the slowest communication rate are always chosen as the reference. On the other hand, since the processing speed and the communication rate of those scheduled tasks are known, their costs are determined by the actual scheduled processing speed and communication rate.

In other words, it suggests to determine the priority values and the construct priority list by using the mixture of "pretend cost" and 'actual cost" as the cost of tasks. The scheduling task graph is separated into two portions: Examined part of task Graph (EG) and Un-Examined part of task Graph (UEG). In determining the cost model, the actual schedule of tasks and edges in the EG are known and their cost can be determined directly from their actual schedule. However, for those tasks in the UEG, their costs are determined by the assumption that they are executed in a reference speed of a processor (the obtained cost is called pretended cost). Similar argument is also applied to the communication cost. Therefore, the cost model of the scheduling iterations (steps) would be the mixture of the actual cost and the pretended cost.

In the fact that rescheduling is not considered, those scheduled tasks should not be rescheduled again. From the viewpoint of the unscheduled tasks, the cost for those scheduled tasks becomes static constant. The cost model formed in each scheduling instance with the pre-assumed reference values of processing speed and communication rate are reducible to a task graph like the homogeneous environment. Besides, the tasks in the EG will not be constituted in the task priority list again. The dependence constrains and the determination of heuristic in the EG and UEG of a scheduling task graph consistent in each scheduling iteration. Therefore, the generated priority value and the priority list can be used to identify which is a more important task to be scheduled.

Using this technique, the priority values can be determined and compared objectively in a common base of references. Consequently, a priority list can be generated for the selection of tasks to be scheduled in the heterogeneous environment. This will be further shown by some proposed implementations and their simulation studies that the heuristic can produce a better scheduling solution than selecting a task at random.

### 3.4.3 Phase 2: Task assignment to processors (To make decision on task to processor assignment)

This phase subjects to Step 2.2 and Step 2.3 of the List Scheduling heuristic. This is used to determine a suitable processor and a suitable location for executing the selected task in phase 1. There are many native heuristics such as the Insertion Scheduling Heuristic (ISH0-2) or the Non-insertion Scheduling Heuristic (NISH) [chapter92] were widely used in the homogeneous version of List Scheduling in designing and implementing to heterogeneous environment, the environmental characteristic should be further considered.

Generally, the task to processors assignment raises a heuristic search problem which is to find a suitable processor and a suitable timing location from a set of bounded number processors to execute the scheduled task. In order to maintain the realistic data-dependence constrain of the scheduled tasks, the searching should consider the dependence constrains and the cost function in the actual speed of assigned processor and the actual rate of assigning communication link. Therefore, a scheduling decision comes out which is based on the actual execution parameters.

## 3.5 Proposed Implementation of Heterogeneous Task Scheduling Algorithms

### 3.5.1 Implementation Issues

To design the scheduling algorithms under the HLS technique, designers have to identify the set of heuristic used in two phases. Then choose a reference values of processing speed and communication link rate through an analysis of the target environment. After that, apply phase 1 of the technique to determine the priority value of all unscheduled free tasks. Based on the priority, a priority list of free task is generated for the selection of free task to be scheduled. Finally, the selected task is scheduled into a processor according to the phase 2 of the HLS.

It is obvious that the scheduling performances of those designed algorithms would be directly affected by different sets of heuristics used in two phases. To study this, the application of the HLS technique in different scheduling heuristic will be analyzed in the coming sections. Three scheduling algorithms called Heterogeneous Dominant Sequent Cluster algorithm (HDSC), Heterogeneous Mobility Direct algorithm (HMD) and Heterogeneous Relative Mobility Scheduling algorithm (HRMS) are proposed to demonstrate the applicability of the technique in different sets of heuristics in two phases.

### 3.5.2 Assumptions

Before going on to the proposals, several assumptions are made for the proposed algorithms in following sub-sections:

1. The term "heterogeneous environment" is applicable to heterogeneous in processor's power (or called processing speed). Resources other than processing powers are keeping homogeneous.
2. Processors and other resources are dedicated and deterministic for the scheduling application.
3. The task cost and the communication cost of the application are deterministic.
4. Rescheduling of scheduled tasks and task duplications is not allowed.
5. The term "bounded number of processors" describes the case that total number of processors $|P|$ is less than total number of tasks to be scheduled $|N|(|P| < |N|)$.
6. The term "bounded number of processors" describes the case that total number of processors $|P|$ is greater than or equal to the total number of tasks to be scheduled $|N|(|P| \geq |N|)$.

### 3.5.3 The Heterogeneous Dominant Sequence Cluster algorithm (HDSC)

#### *A) The HDSC algorithm*

An algorithmic description of HDSC algorithm is shown in Algorithm 3.6. Firstly, the set of heterogeneous processors is sorted in the descending order of speed. Then, a priority list is generated by sorting the sum of top level and bottom level (tlevel($n_x$)+blevel($n_x$)) of free tasks in decreasing order (Step 3-5.1). It is difficult to determine the tlevel($n_x$) and blevel($n_x$) for those tasks, which have not been scheduled into the processors in heterogeneous environments. Therefore, the solution proposed by the HLS technique is used so that all the unscheduled tasks are assumed constituted in a unit cluster with the reference speed ε (where ε is defined in Definition 3.1 and Criterion 3.1). However, this criterion validates the definition of those paths (CP and DS) obtained by the sum of the top level and the bottom level. These paths can no longer satisfy the definition of critical paths and dominant sequences that are defined originally in the DSC algorithm [Yang94]. Therefore, the concept of these paths is extended to heterogeneous environments in which will be described in Section 3.5.3B. Based on the assumption, the priority value of tasks can be determined even they are unscheduled. Therefore, a priority list of free task can be generated by sorting those unscheduled free tasks in descending order. The highest priority free task $n_x$ can be selected from the list of scheduling to a processor in the second phase.

In the second phase, the selected task $n_x$ is scheduled into a processor under two conditions that the task can be completed in the earliest time and the value of the tlevel($n_x$) does not increase. If there is no processors that can satisfy these two conditions, the task will be scheduled to a processor that can satisfy the first condition. As specified in the HLS technique, the decision made in the searching procedure and the scheduling procedure should base on the actual computation speed and communication rate. Therefore, data dependence among the tasks can be maintained.

```
1. Sort the available processors set $P$ in the descending order of speed.
2. Let $EG = NULL, UEG = V$ where $EG$ = Examined Graph, $UEG$ = Un-Examined Graph and $V$ is the set of tasks in the scheduling task graph.
3. Every task is marked unexamined and assumed to virtually constitute to one unit of processors with the reference speed $\varepsilon$.
4. Set the top level is equal to zero for each free task and compute the top level as well as bottom level of each task.
5. While $UEG \neq NULL$ does
    5.1. Using the sum of the top level and the bottom-level of a task as priority value, generate a priority list of free tasks $L$ in which the tasks are sorted in the descending order of priority values. Select the first priority task $n_x$ from $L$.
    5.2. Find a processors $p_i$ to execute the task $n_x$ so that it can be finished in the earliest time and tlevel($n_x$) does not increase. If there is no processors satisfy the condition, the task will be scheduled to a processor that the task can be completed in the earliest time.
    5.3. Add pseudo edges between two independent tasks, which are scheduled into the same cluster.
    5.4 Mark the task examined. Remove $n_x$ from $UEG$ and put $n_x$ onto $EG$.
    5.5 Re-compute the priorities of the $n_x$'s successors tasks.
End While
```

**Algorithm 3.6. The Heterogeneous Dominant Sequence Cluster algorithm (HDSC).**

*Definition 3.1:* *ε is defined as the least speed of computation among processors inside a given processor set P.*

$\varepsilon = \min(S_i)$, where $S_i$ = speed of processor $i$, $i = p_0 \ldots p_n$ and $p_i \in P$.

*Criterion 3.1:* *If a task in a task graph has not been scheduled into any processors, it is assumed to be virtually scheduled to a processor with computation speed ε. Otherwise, the task is running on the scheduled processor.*

### *B) The Extend Critical Path (ECP) and Extend Dominant Sequence (EDS)*

As mentioned, the DSC algorithm uses critical path (CP) and dominant sequent (DS) (the maximum sum of top level and bottom level of tasks) as the task scheduling heuristic. In order to determine the important task to be scheduled, the DSC algorithm initially schedules all tasks in the scheduling task graph onto a unit cluster. After that, the top and the bottom level of all tasks are computed to find the critical path and dominated sequence. This method is feasible in homogeneous environment because processors are at the same speed. The critical path and the dominant sequence are not affected by scheduling tasks to any processors in each scheduling step. They are clearly identified in each scheduling step. However, the same technique cannot be used to determine critical path and dominated sequence in heterogeneous environment. This is because computation speeds are different among different processors. The critical path as well as dominant sequence will be directly affected by where the tasks are actually scheduled. So that the top level and bottom level of those unscheduled tasks cannot be calculated. In other words, the DSC algorithm cannot be plotted into heterogeneous environment directly.

To simplify the problem, the HLS technique is applied. The HLS technique suggests to determine the cost for those unscheduled task by if they are scheduled into a dedicated processor with a common reference speed. That is subjected to the Definition 3.1 and Criterion 3.1. Based on the criterion, the top level and bottom level of those tasks in the scheduling task graph can be determined. Since the task graph is constituted by mixture of tasks that some are scheduled to their target processors (actual cost) and the rest are scheduled to processors by Criterion 3.1 (pretend cost). The found CP and DS are not the CP and DS defined originally. Therefore, two heuristics called "Extended Critical Path" (ECP) and "Extended Dominant Sequence"(EDS) are defined to describe these found paths:

*Definition 3.2:* *The Extended Clustered Graph (ECG) is defined as a DAG with a task to processor assignment applying Criterion 3.1, but without task execution ordering.*

*Definition 3.3:* *The Extended Scheduled Graph (ESG) is defined as a DAG with a task to processor assignment applying Criterion 3.1, with task execution ordering.*

*Definition 3.4:* *The Extended Critical Path (ECP) is defined as the longest path in an extended clustered graph, including both non-zero communication edges cost and task weights in that path.*

*Definition 3.5:* *The Extended Dominant Sequence (EDS) is defined as the longest path in an extended scheduled graph, including both non-zero communication edges cost and task weights in that path.*

*Lemma 3.1:* *In a homogeneous environment, the Extended Clustered Graph (ECG) is the same as the Clustered Graph (CG).*

According to Definition 3.1 and the Criterion 3.1, it is observed that the ECG is reduced to CG in a homogeneous environment.

*Corollary 3.1:* *In a homogeneous environment, the Extended Scheduled Graph (ESG) is the same as the Scheduled Graph(SG)*

It is after Lemma 3.1.

*Lemma 3.2:* *In a homogeneous environment, the Extended Critical Path (ECP) is the same as the Critical Path(CP).*

According to Definition 3.1, Criterion 3.1 and Lemma 3.1, it is observed that the ECP is reduced to CP in homogeneous environment.

*Corollary 3.2:* *In a homogeneous environment, the Extended Dominant Sequence (EDS) is the same as the Dominant Sequence(DS).*

It is after Lemma 3.2.

## C) Experimental study

The proposed HDSC algorithm was implemented and an experimental study was conducted in Experiment 2 as shown in Chapter 8 to evaluate the functionality and properties of the proposed HDSC algorithm. The conclusion and properties are summarized in section D.

## D) Properties

*Property 3.1:* *In homogeneous environment, the Criterion 3.1 is reduced to scheduling tasks to any processors with the same speed.*

In a homogeneous environment, all processors are in the same speed. Those tasks are assigned to processors with the same speed. That is reduced to the case in a heterogeneous environment.

*Property 3.2:* *The task assignment strategy of HDSC algorithm is to schedule task to a processor that can finish it as soon as possible*

This is supported by Conclusion 3 of the Experiment 2 as shown in Chapter 8.

*Property 3.3:* *The HDSC algorithm produces a shorter parallel time schedule as the processor set is configured more deviated from homogeneous. In other words, the HDSC algorithm favor in scheduling jobs onto a single high-performance processor.*

This is supported by Conclusion 4 of the Experiment 2 as shown in Chapter 8.

*Property 3.4:* *In a homogeneous environment with bounded number of processors, the HDSC algorithm cannot reduce to DSC algorithm.*

This is because the HDSC algorithm is designed for scheduling task to bounded number of heterogeneous processors environment. Owing to the processors bound, the reduction of top level of a scheduling task is not guarantee. Therefore, the HDSC cannot reducible to DSC algorithm even in homogeneous environment.

*Property 3.5:* *In homogeneous environment with unbounded number of processors, the HDSC algorithm is reducible to DSC algorithm.*

According to Lemma 3.1, Corollary 3.1 and Lemma 3.2, the task scheduling heuristic of ECP and EDS is reducible to CP and DS using in the DSC algorithm in a homogeneous environment. In case of unbounded number of homogeneous processors (where total number of processors is much more than total number of scheduling tasks), there should exist at least one processors which can satisfy the constrain of top level reduction. Therefore, the HDSC algorithm is reducible to the DSC algorithm.

### 3.5.4 The Heterogeneous Mobility Direct algorithm (HMD)

The HMD algorithm is modified from the scheduling heuristic from the MD algorithm proposed by Wu and Gajski [WuGa88].

#### *A) The algorithm*

An algorithmic description of HMD algorithm is shown in Algorithm 3.7a. The Fact 1 of MD algorithm [WuGa88] is modified to Condition H as shown in Algorithm 3.7b for making schedule decision in a heterogeneous processors environment. Firstly, the set of heterogeneous processor $P$ is sorted in the descending order of speed in Step 1. Then, the Extended Relative Mobility $EM_r$ of each unscheduled task is determined by using Criterion 3.1 (This will be defined in sections 4.5.4b). And generate a priority list $L'$ of free tasks with minimum $EM_r$. In the list, all tasks have the same priority, which are the minimum among other tasks excluded from the list. In this case, a task is randomly selected from the list $L'$ and using the Condition 1 to schedule that a free task to a processor in the earliest finish time. After that, the scheduled task is removed from the list and the Steps 2 to 4 are repeated until all tasks in the task graph are scheduled.

```
Let L compose of all tasks in the scheduling task graph initially,
1.  Sort the speed of target processors in the descending order of speed and
    ordered from left to right. Hence, obtain $\varepsilon$ from the slowest speed processor.
2.  Calculate Extended relative mobility for all tasks ($EM_r$) for all tasks.
        Let $L$ include all tasks initially. Let $L'$ be the group of tasks in $L$ with
    minimum relative mobility. Let $n_i$ be a task in $L'$ that does not have any
    predecessors in $L'$. Using Condition 1, schedule $n_i$ on the first processor. If $n_i$
    cannot be scheduled on the first processor, schedule it on the second processor
    and so on. If no processor can satisfy the condition, the task will be scheduled
    to the first processor that the task can be completed in the earliest time.
3.  When $n_i$ is scheduled on $P_m$, all the edges connecting $n_i$ and other tasks
    already scheduled to $P_m$ are changed to zero. If $n_i$ is scheduled before task $n_j$
    on $P_m$, add an edge with zero cost from $n_i$ to $n_j$ in the graph. If $n_i$ is
    scheduled after node $n_j$ on $P_m$, add an edge with zero cost from $n_j$ to $n_i$ in
```

the graph.

4. Recalculate $EM_r$ of the modified graph. Delete $n_i$ from $L$ and Repeat Step 2 until $L$ is empty.

**Algorithm 3.7a. The HMD algorithm**

Assuming a task $n_i$ is considering to be scheduled to $P_m$, in which $l$ tasks, $n_{m1}, n_{m2}, \ldots, n_{ml}$, have been scheduled into $P_m$. If the moving intervals of these tasks do not intersect with the moving interval of $n_p$, then $n_p$ can be scheduled to $P_m$ Otherwise, assume the moving intervals of tasks $n_{mi}, n_{mi+1}, \ldots, n_{mj}$ $(1 \le i \le j \le l)$ intersect the moving interval of $n_i$. $n_p$ can be scheduled to $P_m$, if there $k(i \le k \le j+1)$ exists,

$$W(n_p)_{p_m} \le \min\{T_F(n_p), T_L(n_{mk})\} - \max\{T_S(n_p), T_S(n_{mk-1}) + W(n_{mk-1})_{p_m}\}$$

, Where $W(n_i)_m$ is the cost function of $n_i$ running in $P_m$ and the computation of $T_L(n_p), T_F(n_p), T_S(n_p)$ should assume that task $n_p$ is scheduled to processor $P_m$.

if $n_{mi-1}$ does not exist, $T_S(n_{mi-1}) = 0, W(n_{mi-1}) = 0$; if $n_{mj+1}$ does not exist, $T_L(n_{mj+1}) = \infty$.

Otherwise, the task cannot be scheduled to $P_m$.

**Algorithm 3.7b. Condition H of the HMD algorithm**

## *B) Heuristics used*

There are two types of heuristics used in the HMD algorithm: the Extended Relative Mobility for priority list generation and schedule task to the first processor that can satisfy the condition H (similar to the ISH1). To generate a priority list of free task, Step 2 of the HMD algorithm subjecting to the Step 1 of the List scheduling heuristic wherein the Extend Relative Mobility ($EM_r$) is used. The Extend Relative Mobility is an extended concept of Relative Mobility using in the HMD algorithm for determining tasks priority. Details explanation of $EM_r$ will be given in part C. Based on the generated priority list, the free task with the highest priority (the $EM_r$ is the lowest) can be selected to be scheduled.

The task to processors assignment strategy is perform in Step 2-3 (Algorithm 3.7b) and the condition H (Algorithm 3.7b). Firstly, the highest priority task is selected from the priority list. Then Step 2-3 try to find the first processor from the high speed processor to low speed processor in the sorted processors queue that can satisfy execute the task with Condition H satisfaction. It is found that the assignment strategy is similar to MD algorithm in homogeneous environment. However, the major difference of HMD is that the calculation of moving intervals and the finishing time of a task are normalized to the speed of the running processor in heterogeneous processors environment. The HMD is designed for scheduling tasks in the bounded number of heterogeneous processors environment. Tasks that are unable to satisfy the Condition H will be scheduled to the first processor that the task can be finished in the earliest time.

### C) *EMr: an extended concept on Relative Mobility*

Since the Criterion 3.1 is used for calculating the Relative Mobility ($M_r$). The concept of the Relative Mobility cannot be retained in the HRMS algorithm. Recalled that relative mobility ($M_r$) used in homogeneous version of the MD and the RMS algorithm as scheduling heuristic for identifying an important free task to be scheduled. The $M_r$ is defined in homogeneous processors systems as $M_r$ ($n_i$)=$M(n_i)/W(n_i)$, where $M(n_i)$ is the mobility of a task $n_i$ and $W(n_i)$ is the computation time of a task $n_i$. Since the computation speed of processors in homogeneous processors systems are equal, the $M(n_i)$, the $W(n_i)$ and the $M_r$ ($n_i$) can be clearly found. However, the processor speed in heterogeneous processors systems is different. The cost (in term of time unit) of task $n_i$ running in different processors are not the same. The $M(n_i)$, the $W(n_i)$ and the $M_r$ ($n_i$) cannot be identified before a task is not scheduled.

Owing to the above reasons, the HRMS algorithm applies the Criterion 3.1 proposed in HLS technique to determine the scheduling heuristic. A normalized factor ε, which was defined in Definition 3.1 and Criterion 3.1, is used as a reference speed of priority values calculation. In fact this is subjected to Step 1 of the HLS technique that for those tasks, which have not been scheduled, into any processors. This assumes the unscheduled tasks are running on a processor speed ε. On the other hand, the cost for scheduled tasks are calculated by the actual processor speed. Since, the definition of task cost in the scheduling task graph is deviated from the original definition. Thus, the definition of relative mobility using in HRMS is no longer satisfies. Therefore, Extended Relative Mobility $EM_r$ is introduced in the HRMS algorithm to describe the heuristic used. According to Definition 3.5 to 3.7, the $EM_r$ heuristic extends the concept of $M_r$ using as scheduling heuristic in heterogeneous processors systems. The Extended Mobility $EM$ ($n_i$), the cost of a task running in processor $p_m$ ($W(n_i)_{p_m}$) as well as the Extended Relative Mobility $EM_r$ ($n_i$) is defined as follows.

*Definition 3.5:* *The cost of a task running in a processor $p_m$ is defined as:*

$$W(n_i)_{p_m} = i(n_i)/s(p_m)$$

*where $i(n_i)$ =total number of instruction of task $n_i$, $s(p_m)$ is the speed of processor $p_m$.*

*Definition 3.6:* *The Extended Mobility (EM) is defined as:*

$$EM(n_i) = T_L(n_i) - T_S(n_i)$$

*where $T_S(n_i)$ = earliest start time of $n_i$ and $T_L(n_i)$ =latest start time of $n_i$ that they are calculated under Criterion 3.1 and Definition 3.2.*

*Definition 3.7:* *The Extended Relative Mobility EMr is defined as:*

$EMr(n_i) = EM(n_i)/W(n_i)_{p_{c1}}$, where $p_{c1}$ is a processor under Criterion 3.1.

## C) Experimental study

The proposed HMD algorithm was implemented and an experimental study was conducted in Experiment 2 as shown in Chapter 8 to evaluate the functionality and properties of the proposed HMD algorithm. They are summarized in the following section.

## D) Properties

*Property 3.6:* *In homogeneous processors systems, the Extended Relative Mobility (EMr) is the Relative Mobility.*

In homogeneous processors systems, all processors are the same in computation speed

$s_{p_0} = s_{p_1} = s_{p_2} = \ldots = s_{p_m} = \varepsilon \ (m = p-1)$. Therefore, $EM_r\ (n_i) \equiv M_r\ (n_i)$.

*Property 3.7:* *The task assignment strategy of HMD algorithm is to schedule tasks onto the first available processor ranked from fast processors to slow processors with the goal of satisfying the condition H.*

It is supported by Conclusion 3 of the Experiment 2 as shown in Chapter 8.

*Property 3.8:* *The HMD algorithm produces a shorter parallel time schedule as the processor set is configured more deviated from homogeneous. In other words, the HDD algorithm favor in scheduling jobs onto single high-performance processor.*

It is supported by Conclusion 4 of the Experiment 2 as shown in Chapter 8.

*Property 3.9:* *In homogeneous environment with bounded number of processors, the HMD algorithm cannot reduce to MD algorithm.*

This is because the HMD algorithm is designed for scheduling task to bounded number of heterogeneous processors environment. Owing to the processors bound, there would schedule some task that cannot satisfy condition H. Therefore, the HMD cannot be reducible to MD algorithm even in a homogeneous environment.

*Property 3.10:* *In homogeneous environment with unbounded number of processors, the HMD algorithm is reducible to MD algorithm.*

According to Property 3.5, the task scheduling heuristic of $EM_r$ is reducible to "$M_r$" using in the MD algorithm in homogeneous environment. Beside, in case of unbounded number of homogeneous processors (where total number of processors is much more than total number of scheduling tasks), there should exist at least one processor that can satisfy the condition. Therefore, the HMD algorithm is reducible to the MD algorithm.

### 3.5.5 The Heterogeneous Relative Mobility algorithm (HRMS)

This section proposes the third algorithm called Heterogeneous Relative Mobility Scheduling (HRMS) algorithm.

#### *A) The algorithm*

An algorithmic description of the algorithm is shown in Algorithm 3.8a-b. The first step of the algorithm shown in 3.8a is to sort the set of given processors in decreasing order of speed. Therefor the processor with the slowest speed can be identified for obtaining the normalized factor ε. Then, the Extended Relative Mobility $EM_r$ (for which was defined in the Section 3.5.4b) of each unscheduled task in the task graph is determined in Step 2. After that, a priority list of free task $L'$ is generated in ascending order of $EM_r$. Hence, the highest priority free task can be selected from the list for scheduling. To determine a task into the suitable processor, the condition H shown in Algorithm 3.8b is used. It finds a processor that can satisfy the condition H and finish in the earliest time. If there is no processors satisfy the condition, the task will be scheduled to the first processor that the task can be completed in the earliest time. The Steps 2 to 4 of the Algorithm 3.8a are repeated until all tasks in the task graph are scheduled.

```
Let L compose all tasks in the scheduling task graph initially.
1.  Sort the speed of target processors in the descending order of speed and order
    from left to right. Hence, obtain ε from the slowest speed processor.
2.  Calculate the Extended relative mobility of all tasks ($EM_r$).
    Let L include all tasks initially. Let L' be the group of tasks in L with the
    minimum relative mobility. Let $n_i$ be a task in L' that does not have any
    predecessors in L'. Find a processor $p_m$ in the processors set P in which $n_i$
    would satisfy Condition H and is completed in the earliest time. If no processor
    can satisfy the condition, the task will be scheduled to the first processor that
    the task can be completed in the earliest time.
3.  When $n_i$ is scheduled on $p_m$, all the edges connecting $n_i$ and other tasks
    scheduled to $p_m$ are changed to zero. If $n_i$ is scheduled before task $n_j$ on $p_m$,
    add an edge with zero cost from $n_i$ to $n_j$ in the graph. If $n_i$ is scheduled after
```

```
   node $n_j$ on $p_m$, add an edge with zero cost from $n_j$ to $n_i$ in the graph.
4. Recalculate $EM_r$ of the modified graph. Delete $n_i$ from $L$ and Repeat Step 2
   until $L$ is empty.
```

**Algorithm 3.8a: The HRMS algorithm**

Assuming a task $n_i$ is scheduled to $p_m$, in which $l$ tasks, $n_{m1}, n_{m2}, \ldots, n_{ml}$, have been scheduled into $p_m$. If the moving intervals of these tasks do not intersect with the moving interval of $n_p$, then $n_p$ can be scheduled to $p_m$ Otherwise, assume the moving intervals of tasks $n_{m_i}, n_{m_{i+1}}, \ldots, n_{m_j}$ $(1 \le i \le j \le l)$ intersect the moving interval of $n_i$. $n_p$ can be scheduled to $p_m$, if $k(i \le k \le j+1)$ exists,

$$w(n_p)_{p_m} \le \min\{T_F(n_p), T_L(n_{m_k})\} - \max\{T_S(n_p), T_S(n_{m_{k-1}}) + w(n_{m_{k-1}})_{p_m}\}$$

Where $w(n_i)_{p_m}$ is the cost function of $n_i$ running in $p_m$ and the computation of $T_L(n_p)$, $T_F(n_p)$ and $T_S(n_p)$ should assume that task $n_p$ is scheduled to processor $p_m$ .if $n_{m_{i-1}}$ does not exist, $T_S(n_{m_{i-1}}) = 0, w(n_{m_{i-1}}) = 0$; if $n_{m_{j+1}}$ does not exist, $T_L(n_{m_{j+1}}) = \infty$.

Otherwise, the task cannot be scheduled to $p_m$.

**Algorithm 3.8b: Condition H of the HRMS algorithm**

**Algorithm 3.8a-b. The Heterogeneous Relative Mobility Scheduling algorithm (HRMS)**

## B) *Heuristics used*

There are two types of heuristics used in the HRMS algorithm:

a) The Extended Relative Mobility for priority list generation and the schedule of a selected task to a processor that it can be completed in the earliest time

b) The satisfaction of condition H (similar to the insertion schedule with the earliest time) for task to processors assignment.

To generate a priority list of free tasks, Step 2 of the HRMS algorithm subjecting to Step 1 of the List scheduling heuristic wherein the Extend Relative Mobility ($EM_r$) is used. Since $EM_r$ is the same as the one used in the HMD algorithm, it will not be described again in this section.

For the heuristic of task to processor assignment, the idea of insertion schedule with the earliest finishing time is used. This is formulated in Step 2-3 of Algorithm 3.8a and condition H in Algorithm 3.8b. If more than one processor satisfy the condition, the task will be scheduled to a processor that the tasks can be completed as soon as possible. However, the design of the HRMS algorithm is for scheduling tasks in a bounded number of heterogeneous processor environments. Some tasks would not be scheduled to any processors under the Condition H. In this case, the task will be scheduled to the first processor that the task can be completed in the earliest time. It is found that the assignment mechanism is similar to that of the RMS algorithm. However, the HRMS algorithm applies the HLS technique in accessing the moving interval determination. Thus, the scheduling decision is made based on the actual speed of assignment processors and the rate of communication links.

### C) Experimental study

The proposed HRMS algorithm is implemented and an experimental study has been conducted in Experiment 2 as shown in Chapter 8 to study the functionality and properties of the proposed HRMS algorithm. A summary of the experimental results and properties is presented in the following section.

### D) Properties

*Property 3.11:* *The task assignment strategy of HRMS algorithm is to schedule tasks to the first available processor ranked from fast processors to slow processors with the goal of satisfying condition H.*

*Property 3.12:* *The HRMS algorithm produces a shorter parallel time schedule as the processor set is configured more deviated from homogeneous. In other words, the HRMS algorithm favors in scheduling tasks to a single high-performance processor.*

It is supported by conclusion 4 of the Experiment 2 as shown in Chapter 8.

*Property 3.13:* *In homogeneous environments with a bounded number of processors, the HRMS algorithm cannot be reduced to the RMS algorithm.*

The HRMS algorithm is designed for scheduling tasks to a bounded number of heterogeneous processor environments. Owing to the processors bound, the tasks that cannot satisfy condition H would not be scheduled. Therefore, the HRMS algorithm cannot be reducible to the RMS algorithm even in homogeneous environments.

*Property 3.14:* *In homogeneous environments with a unbounded number of processors, the HRMS algorithm is reducible to the RMS algorithm.*

According to the Property 3.5, the task scheduling heuristic of $EM_r$ is reducible to $M_r$ in homogeneous environments. Therefore, the HRMS algorithm is reducible to the RMS algorithm.

# Chapter 4: Task Scheduling Techniques for Resources Non-Deterministic Environment

## 4.1 Chapter Summary

The scheduling algorithms for scheduling tasks into classical computation model face the problem of non-determination in resources utilization in either homogeneous or heterogeneous execution environment. This makes their scheduling performances unpredictable and unreliable and the produced schedule meaningless in practical environment. The use of Contractual Computing Paradigm for task scheduling is proposed to overcome the situation.

A brief introduction of the contractual computing paradigm is presented. The paradigm proposed the use of booking mechanism (contracting) in resources management and the advanced scheduling information for scheduling decisions making. The problem of resources non-deterministic can be optimized. Having the characteristics, the paradigm is selected for the study of task-scheduling techniques in environment with non-deterministic resources.

Three task scheduling approaches namely the Scheduler, Resources Oriented and Contractor Oriented approaches were proposed for resources management in scheduling tasks to a practical environment based on the contractual computing paradigm. The detail design and the implementation of these approaches will be formally presented and described in Chapter 5, 6 and 7 respectively.

## 4.2 The Contractual Computing Paradigm

The Contractor paradigm is a generalization of the existing computing model from stand-alone machines to a network computer environment [Lam97]. The relationship among various components in the paradigm can be described as in Figure 4.1. This manages resources in an upper layer and a lower layer. The O.S. layer (lower layer) manages the local computation resources within individual computer system. The contractor layer (upper layer) manages the parallel computation resources interconnected by a common communication network. Since the contractor locates in between the applications (resources consumer) and the operating systems, (managing resources provider such as CPU time) to cooperate resources usage among applications. Applications do not need to coordinate and competition resource usage themselves. Thus, the target environment (the networked, multi-user system with parallel accessible computation resources) are transparence from applications and task schedulers.

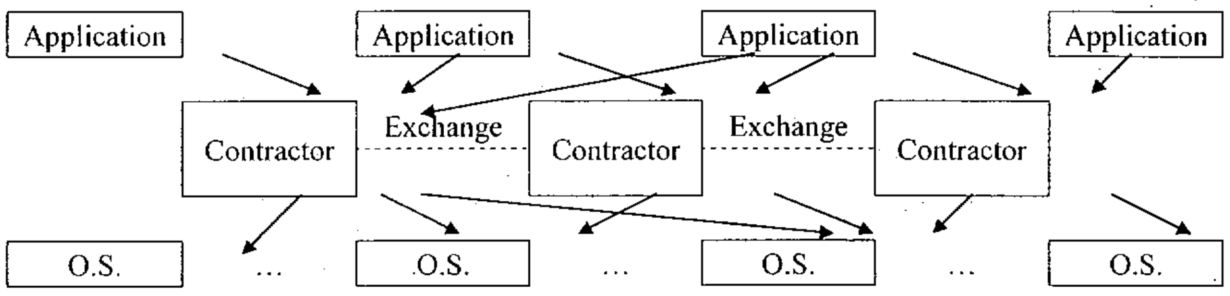


**Figure 4.1. The contractual computing paradigm.**

As compared to the traditional computing model, applications running on the paradigm cannot directly obtain resources from the O.S. Resource is only granted by contractor through an a-c contract (this will be described in Section 4.2.2). In the a-c contract, applications should explicitly specify the logical resources requirements as well as their constrains. The contractor assesses the requirements, whether the application can be executed under current and scheduled resources situation. In the O.S. layer, it is responsible for explicating current and scheduled local resources, grant of resources to contractors and maintaining resources guaranteed for those granted resources. Through the interrelation of these consecutive layers, granted resources in lower layer are also implied guaranteed in higher layer. Consequently, the uppermost layer applications enjoy a stable execution environment. In order to have a better view of the paradigm, studies of the contractor, the contracting, the specification of advanced scheduling information and the strengths and weaknesses of the paradigm are presented in the following sections.

### 4.2.1 What is a contractor?

In the networked computer hierarchy, the significant of the paradigm is the contractor layer, which is responsible for coordination of resources granting local resources management. It has different roles in the applications and the operating systems and the other point of view of the contractor.

The different properties are summarized as:

#### *A) From the viewpoint of an application:*

The contractor is just a "virtual network operating system" which manages the networked computation resources efficiently. Each contractor can be viewed as a "virtual NUMA (non-uniform memory access) machine" with adaptive architectures. It accepts the jobs submitted by an application, which contains parameters of executing the jobs. The contractor makes the most effective use of the computation resources under its control.

***B) From the viewpoint of an operating system:***

The contractor is a pre-registered application that has a very high level of privilege. The contractor can know explicitly the local resource and the scheduled resources available from individual operating systems. Whenever the contractor comes with a schedule (this schedule may be ahead of time), the O.S. grants its local resource by means of advanced booking/ticketing. The operating system should support those contractors' requests whenever possible.

***C) From the viewpoint of other contractors:***

The contractor is a business middleman. Contractors can exchange resources that are already in their respective accounts. There is no priority among contractors. Resources are freely exchanged for the benefits of any participating parties. It should be emphasized that each party can have its own definition of benefits. This exchanged of resources must be recognized by the correspondent operating system.

### 4.2.2 The contracts

A contractor has to coordinate computation resources for different applications and negotiate with the operating system and/or other contractors for granting computation resources. The procedures of such resources request and resources grant are generalized as a contract making.

*A) Type of contracts*

Contracts defining the relationship and the content of information flow between layers in the hierarchy. Lam's [Lam97], proposed three kinds of contracts and they are the application-contractor contract (a-c contract), the contractor-O.S. contract (o-c contract) and the contractor-contractor contract (c-c contract).

The function of the "a-c contract" is an explicit procedure of submitting jobs by an application to a contractor for execution. An application has to submit computation resources requests to the contractor and state the logical resources requirements and the physical constrains. The contractor then grants the opposite resources being requested.

The "o-c contract" is an explicit procedure of computational resources acquisition by contractor from an operating system. The contractor translates requests of the application into physical computational resources requirements and relays to the operating system. The operating system grants, immediately, or in advance, computational resources to the requesting contractor.

The "c-c contract" is an explicit procedure for contractors to exchange computational resources with those individually acquired through "o-c contract". It has to negotiate with other contractors for the exchange of computational resource that aims at greater benefit defined by the contractor itself. Consequently, those layers can be co-operated themselves in an independent and systematic manner.

### *B) The resources contracting procedure*

The contracting procedure can be treated as resource booking. As shown in Figure 4.2, the procedure involves two parties: resources requester and resources provider. When the resources requester tries to book resources, it initiates a resources request to its lower layer. The lower layer receives the information and then determines whether the requested resources are granted or not. In determining the resources grant, the precondition resources is guaranteed. Finally, the status of contract is acknowledged by a request and the acknowledgement is returned to the requester.

Resources request information Request acknowledgement

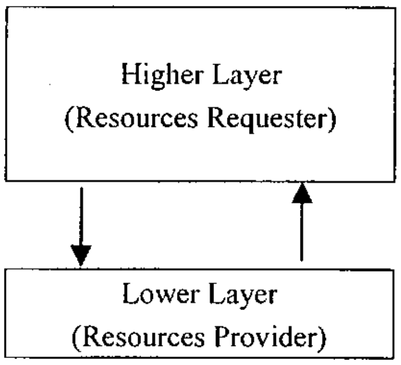


**Figure 4.2.The contracting procedure.**

**Example 4.1:**

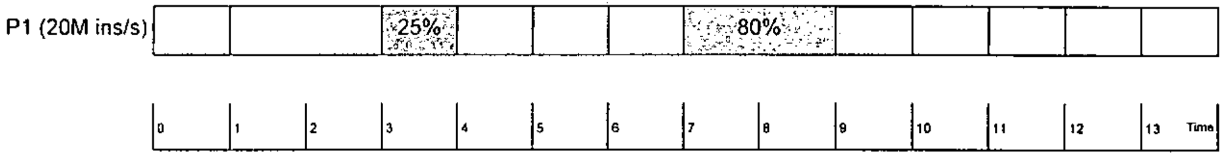
It is to illustrate the idea of the resources booking mechanism. Consider task 1 of an application 1 requests for CPU time resources with start time in 1 time unit, end time in 3 time unit and constitutes 50 percentage of CPU speed from processors $P_1$. In $P_1$, some time slots are occupied by other tasks as shown in Figure 4.3c. The resource request of this request is represented in the right hand side of Figure 4.3a. Since another job uses 25% of the resources, there is 75% of CPU time left for other application. Therefore, the resources request is accepted. A contract is made and an acknowledgement is returned to the requester as shown in the right hand side of Figure 4.3b. Finally, the workload situation of P1 after the booking is shown in Figure 4.3d.

| Description of resources request | Example of requesting CPU time |
|---|---|
| Applicant I.D. | Task 1 of Application 1 |
| Start time of using the resource | 1 |
| End time of using the resources | 3 |
| Percentage of using the required resources | 50% of the CPU speed |

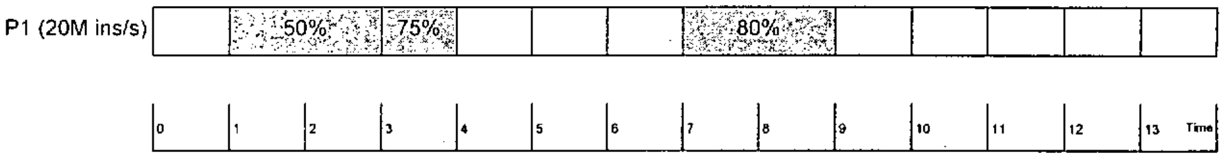
**4.3a: Resource request information**

| Description of request acknowledgement | Example of requesting CPU time |
|---|---|
| A booking I.D. (token) of the granted resources | Task 1 of Application 1 |
| Granted start time using the resources | 1 |
| Granted end time using the resources | 3 |

**4.3b: Description of request acknowledgement**



**4.3c: Advanced-scheduling information of P1 before contracting**



**4.3d: Advanced scheduling information of P1 after the contract is accepted**

**Figure 4.3a-d. An example of information flow for the contracting procedure.**

### 4.2.3 The Advanced Scheduling Information

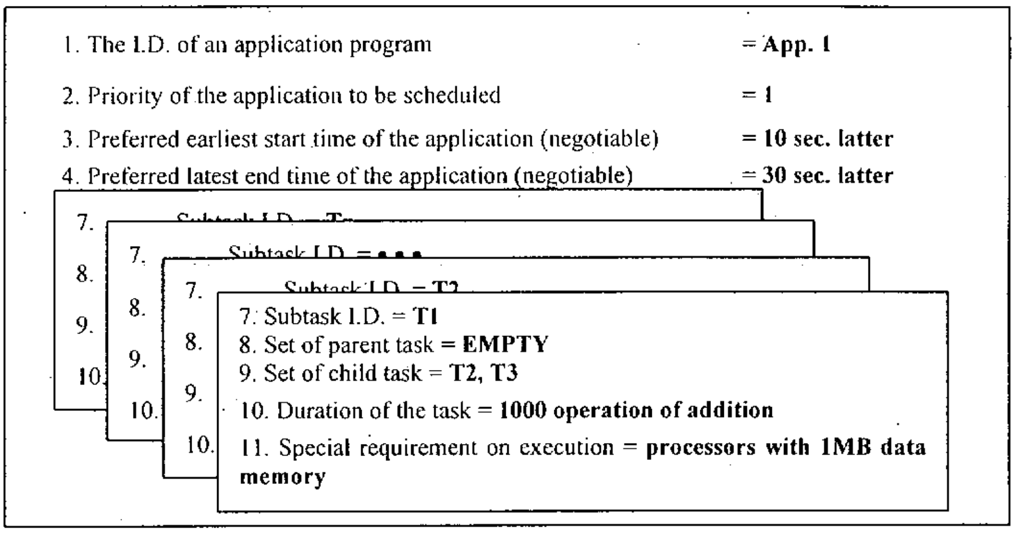
The Contractual Computing Paradigm encourages the use of resources with explicit planning. An advanced booking or ticketing schemes is suggested. A contractor obtains the status of current and scheduled resources from operating systems. Then, it matches with the resources requirements of the scheduling application. If the requirements are satisfied, the resources will be booked in advanced. Regarding the current and the scheduled resources, the scheduling instructions from users, the specifications of an application and the information relating to scheduling and resources allocation are classified as "Advanced Scheduling Information". There are two main sources of information including the application specifications and the resources specifications.

#### *A) Application specifications*

This specifies the structural characteristics, the execution constraints and the resources requirements of an application program. Since the job natures of each application such as interactive jobs, the computation intensive applications and the complex multiple tasks applications are different, the specifications are different from each other. For simplicity and consistency with the task-scheduling technique we have studied so far, the specification focused on parallel program applications represented in DAG is being studied. It is demonstrated as an example as shown in Figure 4.4.

#### ***B) Resources specifications***

It specifies current and scheduled resources of the entire system. Since the concept of contract applies in resources management. Lower layer guarantees those resources granted to higher layer. Unconsciously, actual and realistic information describing the current and scheduled resources has already existed. This section studies the idea from the viewpoint of processor workload. However, ideas can also be extended to other resources in a networked-computing system. Readers are recommended to study the discussion in [Lam96] for detailed discussions.


1. The I.D. of an application program = **App. 1**
2. Priority of the application to be scheduled = **1**
3. Preferred earliest start time of the application (negotiable) = **10 sec. latter**
4. Preferred latest end time of the application (negotiable) = **30 sec. latter**

7. Subtask I.D. = **T1**
8. Set of parent task = **EMPTY**
9. Set of child task = **T2, T3**
10. Duration of the task = **1000 operation of addition**
11. Special requirement on execution = **processors with 1MB data memory**

**Figure 4.4. Application specification.**

Figure 4.5 shows the resources specification of four processors collected from O.S. layer. $P_1$ executes in20 Mips, $P_2$ executes in Mips. This is observed that the target execution environments may compose of computers with different computation performances. It can then be treated as heterogeneous environments.

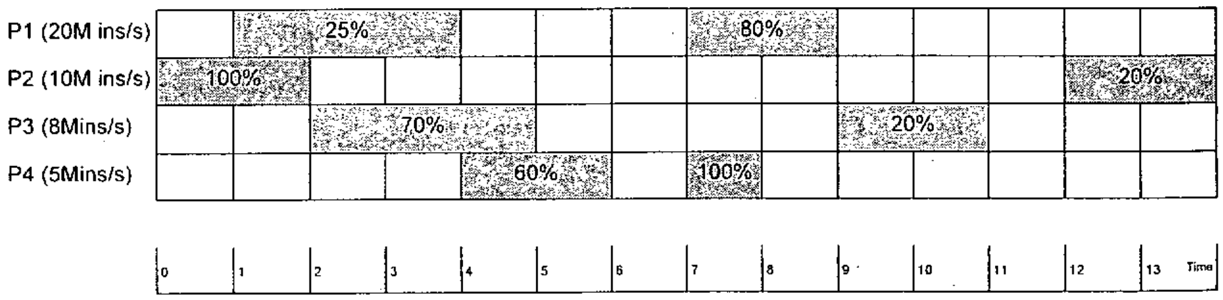


**Figure 4.5. Advanced scheduling information for processors P1, P2, P3 and P4.**

The information also shows the details of the current and scheduled workload of each processor. A graphical representation is shown as a grant chart in Figure 4.5. For example, in processor $P_1$, the interval between 1 and 4 has already consumed by other consumer with 25% of the origional CPU power. Based on this information, the scheduler or other resources requester should not request the resource more than 75% of CPU power within that interval. Otherwise, the processors will overload. Since the information is automatically captured by contractor and /or O.S., application programs and/or the task scehdulers do not need to maintain and manage the networked resources. That greatly reduce the cost of workload estimation in resources non-deterministic environment.

### 4.2.4 The claimed strengths and weaknesses of the model

The proposed model is claimed to have certain strengths and weaknesses [Lam97].

*A) Strengths:*

- A contractor is a transparent and easy-to-use interface for users in the network and parallel-computing environment.
- A set of rules for the construction of contractors is implemented simply and easily.
- A resource booking/ticketing system encourages better planning in the use of computational resources.
- The clear divisions of labor (the two-layered resources management architecture) facilitate the development of future network computing environments.
- The special treatment of uncertainty lets the applications using the resources in a proper order.
- The properties of decentralizing control of network computational resources allow each contractor to have full privilege to manage the control of its own computational resources from any centralized mechanism. Therefore, the efficiency of resource management is highly increased.

*B) Weaknesses:*

- The inaccurate estimation in resources request is the great challenge to the model. It will result in improper allocation of resources to these applications. A penalty is suggested to punish an application with inaccurate estimation. However, this mechanism is very difficult to make in the implementation of a system under contractual paradigm.
- Apart from the applications submitted by users, there are also a number of system functions such as interrupt handling which the local operating system. These functions would also be competed for processing time unless they are also contracted. However, this type of contractor is very difficult to implement.

## 4.3 Task Scheduling in Resources Non-Deterministic Environment

It is found that the paradigm is proceeded with advantage on solving task-scheduling problem of practically resources non-deterministic environment:

- It encourages the user to plan their applications before submitting application to be scheduled for execution. This specifies the preferable start time of execution, the preferable stop time, resource requirement, etc. This information greatly improves the efficiency of resource allocation and the scheduling performance.

- Using the advanced booking/ticketing mechanism together with the contract binding non-deterministic of computational resources, it is significant desensitized. The response time for both parallel and sequential applications can be easily predictable.
- Based on the above feature, the contractual computing environment supports the scheduling of multi- applications in a networked computing resources space. There can be more than one application to be scheduled at any time into a parallel-distributed computing environment without causing resources competition. This can cut down the non-deterministic nature of resources caused by the multiple users and multiple application environments in the practical parallel and distributed systems.
- The static load balancing strategies can be implemented with ease. Due to the amount of time that a computation node spends on a certain task is affected by instantaneous loading. Load balancing strategies will only treat the system to be virtually dedicated computer systems.
- The construction of automatic load balance becomes much easier because with the available of the advanced information from the applications and the operating system, schedulers with contractor binding highly increase the performance of pre-allocation of task to processors.
- Load balancing can be separated from the applications during design and operation/execution.
- Task scheduling strategies used in the dynamic environment can reduce the need for dynamic load balancing. The dynamic load balancing strategies can improve the scheduling performance of task running in dynamic environment, but it is costly. The contractor model on the other hand provides a stable computational resources environment. It is possible to use static scheduling strategies instead.

All these features provide a stable and reliable execution environment for practical parallel and distributed systems. Thus, this motivates the development of static and automatic task scheduling techniques. They take the advantages much further from dynamic task scheduling. That is the reason why the contractual computing paradigm is tried as the target platform in studying task-scheduling techniques for practical parallel and distributed systems.

## 4.4 Issues on Developing Task Schedulers

### 4.4.1 The goals of task scheduling

The function of task scheduling algorithm is to order the task execution so that data dependencies are satisfied and the goals of task scheduling are achieved. In Chapter 2, three preliminary goals for classical scheduling problem are that: **G1**-The complexity should be low, **G2**-The parallel time should be minimized and **G3**-The efficiency should be minimized. However, the paradigm suggests running applications with advanced resources booking. Applications may not be executed so that it is simply caused by failing to commit resources contract. In this occasion, the goals G1, G2 and G3 for which have mentioned before become meaningless. For this reason, a task scheduler for the paradigm should not only satisfy the goals of G1, G2 and G3, but also should require the resources grant. The four preliminary goals of the task-scheduling problem can be generalized as;

**Goal 1**: Committing resources contract request (This is the essential goal)

**Goal 2**: The parallel time should be minimized

**Goal 3**: The efficiency of using resources should be minimized

**Goal 4**: The complexity of the scheduling algorithm should be low

The first goal is the essential and the necessary resources to run the application must be guaranteed for execution. Resource that granted in the contractor layer can be enforced. It proceeds with a suitable schedule in the O.S. layer and allows multiple applications to use the networked computing resources efficiently. This encourages one application running with others applications with co-existence with advanced planning. Having satisfied this primary goal, the evaluation of scheduling performance follows the other goals.

### 4.4.2 The roles of task scheduling

In traditional scheduling concepts, scheduler is constituted in a part of compiler (for static scheduling) or in a part of network operating system (for dynamic scheduling). They have to understand and to manage the entire networked computer resources (such as CPU time) for load balancing. This is no doubt a high cost on task schedulers and restriction of resources sharing to different applications on multiple users and multiple applications systems are imposed.

It is clear that the contractual computing paradigm provide divisions of labors in managing resources. This can be realized as a two-layer hierarchy for resources management. Each layer has to maintain and schedule their resources for supporting its adjacent layers. Thus, scheduling constitutes in different portions of the hierarchy, which include local resources scheduling in O.S. layer, task scheduling in contractor layer and task scheduling in an application layer.

#### *A) Local resources scheduling in O.S. layer*

The operating system should support the contractor's request whenever possible. It should even commit to the future allocation of resources if the contractor can submit schedule ahead of time. Here, the local schedulers of the operating system may have a certain booking/ticketing system for scheduling it own resources. There can be more than one contractor registered with an operating system. Some priorities and quota systems have to be implemented to settle the resources contention. A higher priority contractor may be granted more quota, or may be committed to more relaxed conditions. However, committed resources should not be re-claimed without the consent of the contractor owner. The local scheduler supports contractor in a different way compared with that in the conventional operating system, in terms of its commitment of resources allocation to jobs. Ideally, there should be no queuing, waiting, or other forms of uncertainty involved in the resources scheduling.

The implementation of O.S. kernel supporting the above specifications is a challenging problem in the distributed computing technology. Researches are being carried out to design and implement a kernel satisfying the specification [Lam96b][Siu96][Cheuk97]. According to the specification, a modified LINUX kernel called "The Local Manager" is being developed [Cheuk97] for addressing to processing power management. This modifies the method of scheduling processing inside LINUX kernel so that the processing power can be booked in advanced. In the rest of this research, the design, the implementation and the analysis of task scheduling technique for the paradigm will be studied under the kernel.

### *B) Task schedulers in contractor layer*

Contractor is responsible for managing network resource, which maintains the advanced scheduling information and carry out resources booking directly to the lower layer. Hence, it naturally to implement task scheduler function in the contractor layer. As one of the primitive operations of a contractor is to coordinate and manage the available resources (both current and scheduled resources) for the usage of various consumers (applications or tasks). The contractor should have all the current and scheduled resources information of individual machine. The task scheduler in contractor uses this information to organize and to allocate those scheduled sub-tasks into target machines in a cost-effective manner. The advanced booking/ticketing resources mechanism for resources allocation can be developed to avoid contention of the scheduled tasks with other tasks in execution. Any multiple-users multiple-application can be executed in a PDS with a virtual deterministic resources environment.

### *C) Task schedulers in application layer*

For an application, it not only has to state the resources requirement and physical constrains to contractor explicitly, but also constitutes in a part of task scheduling. In fact, the contractual model does not restrict to execute applications, which possess with its own task scheduling mechanism and it provides a stable and reliable environment for executing these applications. Thus, the existing sequential and parallel applications can transit to and take the advantage of the new paradigm.

## 4.5 Proposals of Task Schedule Techniques in Contractual Computing Paradigm

The contractual computing paradigm provides a stable, informative and reliable execution environment for parallel and distributed applications. This is a suitable paradigm for developing task-scheduling techniques, which are free from non-deterministic resources. Stated in previous section, scheduling is contributed in three roles of the paradigm. This dissertation will focus on how the existing task scheduling techniques and algorithms transit to the new paradigm for overcome the problem of resources non-deterministic. The major goals of task scheduler in contractor layer and application layer are to commit resources granted. The granted resources imply the resources guaranteed so that virtually dedicated resources are provided for solving non-deterministic resources. This idea motivates the development of "Scheduler Oriented" (SO) approach.

### 4.5.1 Scheduler Oriented (SO) approach

The proposal and the analysis of the SO approach will be detailed in Chapter 5. This glued the existing scheduling algorithms and the contractual computing paradigm together. The underlining principle is the use of classical scheduling algorithms as the core of task to processor scheduling. With the produced schedule, the required resources of each scheduled task are booked in advance. This is a conservative method of transiting task scheduling solution, which is produced by classical scheduling algorithms, to a contract based resources management paradigm in order to overcome the problem of resources non-deterministic. Through the advanced resources booking, the execution of scheduled tasks indeed can get the performance stated by the scheduling algorithm, but this was found that the tided resources requirement specification from scheduled tasks restricts the rate of successful resources booking. This leads to the development of the "Resources Oriented" (RO) approach as presented in Chapter 6.

### 4.5.2 Resources Oriented (RO) approach

The Resources Oriented approach attempts to overcome the difficulties encountered in the SO approach. In the "Resources Oriented" approach, the concept of the booking part of processing power from each processor to form a virtually dedicated processor (VDP) (this action is called resource partitioning) is performing before task scheduling. Having the resources booking in advance, task schedulers can schedule tasks to the VDP as if in the dedicated resource environment. Hence, many of the classical task scheduling techniques, such as the List

Scheduling Heuristic (LSH) and the Heterogeneous List Scheduling Heuristic (HLST) can be directly adapted. With the approach, resources pre-allocation can improve the chance of resources booking over the SO approach, but there will be wastage of "overbooking" resources while the booked resources are not fully utilized by the schedule of the application. The non-realized resources in scheduling with the RO approach will be analyzed in Chapter 6. Scheduling using either the SO or RO approach suffer from one kind or the other shortfall.

### 4.5.3 Contractor Oriented (CO) approach

An intermediate approach "Contractor Oriented" (CO) approach is proposed. The principle behind this approach is that the workload situation of the execution environment from contractor is extracted and analysis for task scheduling decision. The extracted information can be used to identify the available resources during scheduling. Task scheduler can even make scheduling decision in time of resources contention and the successful rate of resources booking can then be improved. It is found from the experimental studies that this approach can be adapted to the real time workload situation and with a higher probability of successive resources booking.

# Chapter 5: The Scheduler Oriented (SO) Approach

## 5.1. Chapter Summary

This Chapter introduces the Scheduler Oriented approach for scheduling tasks into practical parallel and distributed systems under the contractual computing paradigm. The proposed SO approach is to study the characteristics of classical scheduling algorithms and their scheduling solutions when resources booking mechanism is applied directly in contractual computing paradigm. This approach encourages application possessing theirs own task schedule or using existing scheduling algorithm for scheduling tasks with regardless of the workload situation as the core of task to processor assignment. Then, resources booking overcome the problem of non-determinism. All tasks in the application can obtain their resources granted. The application can be completed promptly in a dedicated resources environment. Otherwise, the schedule is not suitable to be executed in the current workload situation.

Based on this approach, three suggested implementations are presented. Their properties, the design approach and the significant of improvements were studied. It was found that from Experiment 3 (Chapter 8) the probability of resources booking rate is decreasing gradually as the workload of the processors is increasing. Besides, it also decreases as the number of sub-tasks in an application increases and the sizes of those sub-tasks increase. This concluded that the SO approach is only applicable in narrow range of the specifications. Therefore, intensive investigation on the task scheduling technique for the contractual computing paradigm will be conducted and presented in the subsequent chapters.

## 5.2 The Scheduler Oriented Approach

There are a large number of scheduling algorithms and theories proposed by numerous researchers [McCreary94] [Yang94] [El-Rewini94a-c] [Sark89] [Yang94] [WuGa88] [WuGa88] and most of them can produce an optimum or near optimum scheduling solutions. With these solutions, most of the scheduling cannot achieve good performance while being implemented in a practical environment. This is because most scheduling theories and scheduling algorithms assume that resources for scheduling are deterministic and dedicated for the application under consideration. In practice, resources in parallel and distributed systems neither are dedicate nor are deterministic to all applications. Many of the assumptions made will no longer be valid. Therefore, sophisticated task scheduling methods for massively parallel computer system or dynamic scheduling technique were proposed to minimize the requirement of resources dedication and workload estimation [MVDeva89]. Although the use of these solutions can improve the situation, the desired scheduling performance still cannot be achieved in a practical execution environment [Kunz91].

As the contractual computing paradigm is introduced to solve the problem, the obvious advantage of the Paradigm over classical computing model is the support of advanced resources booking. The proposal of advanced scheduling information and resources guaranteed results in forming a deterministic and dedicated environment. Under the paradigm, the assumption being made in classical scheduling techniques becomes valid and execution of a produced schedule becomes feasible in practical PDS.

Applying the proposed SO approach to those classical scheduling algorithms scheduling solutions can be generated directly with the including of the resources booking. This approach encourages application either possessing their own task schedule or using existing scheduling algorithm to schedule tasks as the core of "task to processor" assignment. Then, resources booking can overcome the problem of non-determinism and all tasks in the application can obtain their resources granted and the application can be completed promptly in a dedicated resources space. Otherwise, the schedule is rejected, which implies that it is not possible to execute the schedule in the current workload situation.

### 5.2.1 An algorithmic description of the SO approach

For an application consists of $n$ task $N = \{n_1, n_2, \ldots, n_n\}$ with their relationship represented in certain format such as direct acyclic task graph (DAG). Assume $P = \{p_1, p_2, \ldots, p_k\}$ be a set of $k$ processors that satisfies the physical resources requirement to execute the application. Let TPS and TPE be the pre-specified, start time and end time for executing the application respectively.

Step 1:

1.1 Using classical scheduling algorithms, schedule the tasks into $P$ without considering the workload situation of $P$.

1.2 If the parallel time (PT) of the produced schedule is completed within the period of preferred start time ($T_{PS}$) and preferred end time ($T_{PE}$) ($PT \le T_{PE} - T_{PS}$), then proceed to step two. Otherwise, report the schedule is rejected and quit.

Step 2: Let $R$[*] $= \{r(n_1), r(n_2), \ldots, r(n_{|N|})\}$, where $r(n_i) = \{T_S(n_i), T_E(n_i), p_n, B(p_n)\}$ be a set of resources booking specifications for the scheduled tasks in the task set $N$.

DO

[*] The $R$ and $r(n_i)$ is defined in definition 5.1

```
2.1 Select a request $r(n_x)$ from $R$ and book the required resources for the
    scheduled task $n_x$ through the resources booking mechanism. Then remove
    $n_x$ from $R$.
2.2 Collect resources booking acknowledgement information from lower layer
    contractor. If the booking is rejected, release the booked resources. Report
    the schedule is rejected and quit.
WHILE $R$ is not empty AND the schedule is not rejected.
```

**Algorithm 5.1, The Scheduler Oriented (SO) approach.**

A two-steps-algorithmic description of the Scheduler Oriented approach is generalized in Algorithm 5.1. The first step is to use the existing or classical scheduling algorithms to produce a scheduling solution, in which resources are assumed deterministic and dedicated. The second step (Step 2.1) is to book resources from resources providers. In Step 2.2, whether all tasks in the application can acquire the required resources for execution is examined. If so, the schedule is accepted and the application will be executed according to the generated schedule. Otherwise, the schedule is rejected.

*Definition 5.1:* *An information of booking request is defined as:*

$$r(n_x) = \{T_S(n_x), T_E(n_x), p_u, B(p_u)\}$$

*where $T_S(n_x), T_E(n_x)$ is the start time and end time of the scheduled task $n_x, p_u$ is the scheduled processor and $B(p_u)$ is the percentage of processing power of processors $p_u$ required for the booking.*

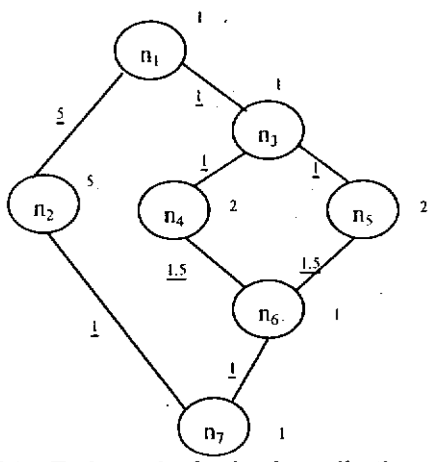
### 5.2.2 To obtain a schedule of an application program

The SO approach encourages scheduling task by using the existed schedule of the application or existing/classical scheduling algorithms. Classical scheduling techniques such as the List Scheduling Heuristic and the Heterogeneous List scheduling technique can be used as the core in Step 1.1 of Algorithm 5.1. Since the design of these scheduling algorithms assumes resources dedicated and deterministic, the execution environment should be treated as dedicated and deterministic. That is, the inherence workloads occupied by other applications are not considered. Then perform classical task scheduling algorithms without algorithmic modification. An example is shown in example 5.1, in which the best scheduling performance can be achieved if all the scheduled tasks can get their required resources in Step 2.

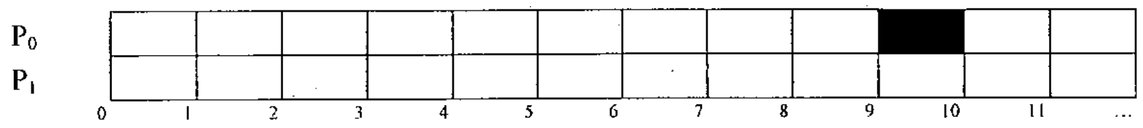
**Example 5.1**

Consider a task graph shown in Figure 5.1a to be scheduled into two processors that are specified in Figure 5.1b. According to the SO approach, the processors set $P$ should assumed dedicated and deterministic (as described in Figure 5.1c, there is no pre-booked task inside the processors) during task scheduling. After a schedule is produced (for example a grant chart shown in Figure 5.1d), a summary of the required resources for all tasks in the scheduling task graph can be given in Figure 5.1e. This information allows the resources booking routine in Step 2 of the SO approach to book the required resources.

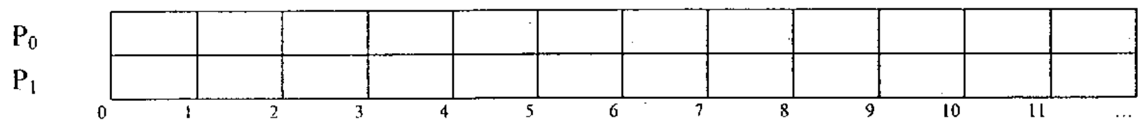
After performing task scheduling in Step 1.1, timing constrain of the produced schedule will be examined in Step 1.2. The objective of the examination is to assess whether the produced schedule is able to complete within the user's specified start time and end time. Accordingly, eliminating unnecessary resources booking in step two can reduce the complexity of the scheduling algorithm. In which, the start time and end time are parts of advanced scheduling information specified by users prior launching of any application (refer to Chapter 2 for details description). Decision is made in accordance with the Lemma 5.1. Satisfying the Lemma implies that the produced schedule will be completed within the pre-specified period. The scheduled tasks in the application program will either successfully book the required resources in Step 2. Or else, the schedule is rejected due to that the user's specification can be satisfied. In latter steps, it is impossible to execute the application within the given specifications and such booking is meaningless.



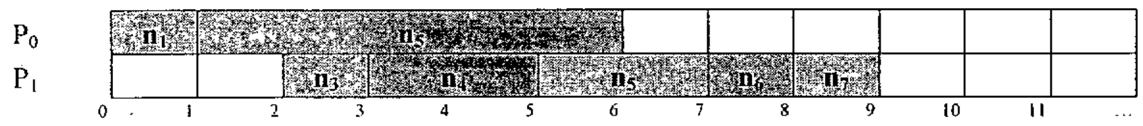
5.1a: Task graph of a simple application



5.1b: The loaded processors configuration



5.1c: The processor configuration before scheduling



5.1d: The produced schedule

| R | $R(n_x)$ | | | |
|---|---|---|---|---|
| | $T_S$ | $T_E$ | $P_x$ | $B(P_n)$ |
| $n_1$ | 0 | 1 | $P_0$ | 100% |
| $n_2$ | 2 | 3 | $P_1$ | 100% |
| $n_3$ | 3 | 5 | $P_1$ | 100% |
| $n_4$ | 5 | 7 | $P_1$ | 100% |
| $n_5$ | 1 | 6 | $P_0$ | 100% |
| $n_6$ | 7 | 8 | $P_1$ | 100% |
| $n_7$ | 8 | 9 | $P_1$ | 100% |

**5.1e: A table of scheduled task specifications, where B(Pn) is the amount of processing power being booked in a given period.**

**Figure 5.1a-e. An example of scheduling task in the SO approach.**

*Lemma 5.1:* *The necessary condition to execute an application program within pre-specified start time and pre-specified end time is:*

*Let the pre-specified start time and the pre-specified end time of an application program be $T_{PS}$ and $T_{PE}$ respectively and the parallel time of the application program to be scheduled is $PT$. To execute the application program within $T_{PE}, T_{PS}$ it should satisfy the following inequality:*

$$PT \leq T_{PE} - T_{PS}$$

*Otherwise, the application program cannot be completed within the pre-specified period.*

### 5.2.3 Resources booking

The objective of resources booking in Step 2 is to book the required resources from the lower-level resource manager in order to produce the required schedule. The concept of resource booking is similar to processor assignment in classical scheduling theory. However, layer's approach proposed by the contractual computing paradigm separates the resources management from the application program/users to the lower-level resource management. Any application program/users requesting for resources should specify their requirements explicitly and book the resources through a resource booking mechanism. The lower-lever resource manager in accordance with its current and scheduled situation can then grant the resources. From the viewpoint of task scheduling, the role of resources management can be reduced. The design of task scheduling algorithm can become simple and cost effective. The proposed SO approach takes this advantage so that it needs not to re-design the core scheduling algorithms being used. Theories and algorithms of classical scheduling can be directly migrated to the new paradigm.

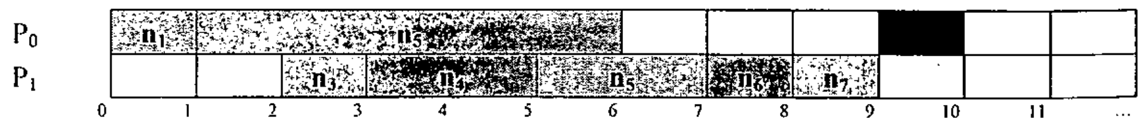
The Step 2 of Algorithm 5.1 described a generic method of resources booking. The method is to book resources for each scheduled task, according to the specification of each scheduled task

$R(n_x)$ one by one. The resources manager (say local manager) receives the requests, decision of resources granted would be examined and decided. An example illustrate the method is shown in example 5.2. Since the tasks had already been scheduled, dependence constrain among the tasks is resolved by the task schedulers. In which the order of the booking sequence is not important, however, any resource requested by all tasks in the application program should be granted by the resources provider.

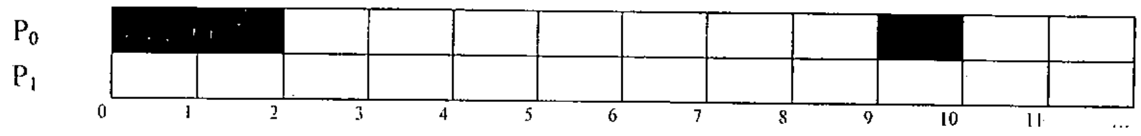
The necessary condition for a program being executed properly is that all scheduled tasks should be granted due to the nature of inter-dependent among the tasks within a scheduled application. An application would come to a dead lock if a task in the produced schedule cannot be started or finished properly. In practice, some requested resources may be booked by other applications or user and this would result in resources conflict. Therefore, successful resource booking is not usually fulfilled. Unfortunately, most scheduling algorithms do not cater for the workload situation in scheduling time. It is possible for a generated scheduling to work in a dedicated environment, but such a good schedule produced not necessary applied under the contractual environment. Therefore, in Step 2.2 in order to reduce the complexity of the algorithm, the resources booking procedure for any task in the application, which cannot be booked will be terminated. The probability of resources booking will be discussed in Experiment 3 in Chapter 8. It is found that the probability of successful resources booking gradually decreases as the workload and the booking size increase. The SO approach is restricted to applications with lower number of tasks, small size of task and high resource availability.

**Example 5.2**

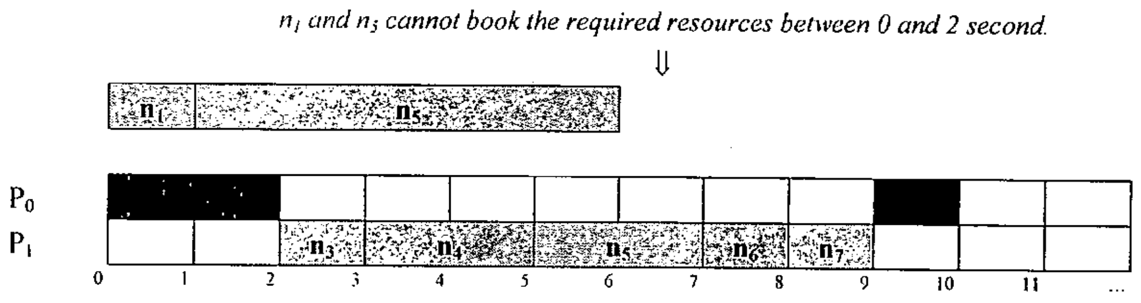
Figure 5.2a show the results of resources booking after Step 2 is applied in Experiment 5. With the workload of the target processors set changes from Figure 5.1b to Figure 5.2b, the application program cannot be started and executed, when the same schedule (shown in Figure 5.1c) is employed this cases. It is because there is another scheduled task occupying the processor $p_0$ in between 0 and 2 second. Resources between task $n_1$ and $n_5$ (Figure 5.2c). That is, the resources requested by task $n_1$ and $n_5$ cannot be granted simultaneously. Since $n_1$ cannot start on time, this results in the failure of starting the remaining tasks in the application. The Figure 5.2d shows an alternate schedule that can fit the set of processors, but the core scheduling algorithms used in the SO approach has not provided for the advanced scheduling information.



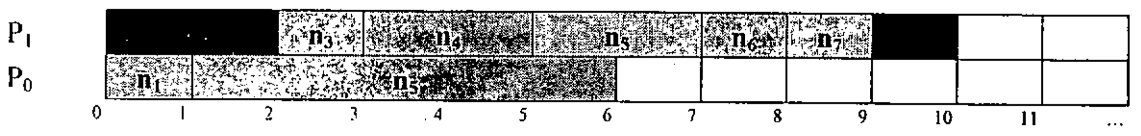
**5.2a: Results of a successful resources booking in example**



5.2b: Another configuration of processors


$n_1$ and $n_5$ cannot book the required resources between 0 and 2 second.
⇓
n1
n5
P0
P1
n3
n4
n5
n6
n7
0 1 2 3 4 5 6 7 8 9 10 11 ...

5.2c: Result of unsuccessful booking)



5.2d: An alternative scheduling solution to suit the workload situation

Figure 5.2a-d. An example of resources booking.

## 5.3 Implementations of Scheduling Using SO Approach

In this section, the implementations of SO concept are applied to three different task-scheduling algorithms. They are the Scheduler Oriented Heterogeneous Dominant Sequent Cluster algorithms (SO-HDSC), the Scheduler Oriented Heterogeneous Mobility Direct algorithm (SO-HMD) and the Scheduler Oriented Heterogeneous Relative Mobility algorithm (SO-HRMS). SO approach is being integrated. The core of the scheduling algorithms remains unchanged, but the concept of detailed algorithmic descriptions of the HDSC, HMD and HRMS are not given here (refer to Chapter 3 for their details).

### 5.3.1 The Scheduler Oriented HDSC Algorithm (SO-HDSC)

For an application consists of $n$ task $N=\{n_1,n_2,\ldots,n_n\}$ with their relationship represented in certain format such as direct acyclic task graph (DAG). Assume $P=\{p_1,p_2,\ldots,p_k\}$ be a set of $k$ processors that satisfies the physical resources requirement to execute the application. Let $T_{PS}$ and $T_{PE}$ be the pre-specified start time and the pre-specified end time for executing the application respectively.

Step 1:

1.1 Using HDSC algorithm, Using classical scheduling algorithms, schedule the tasks into $P$ without considering the workload situation of $P$.

1.2 If the parallel time (PT) of the produced schedule is completed within the period of preferred start time ($T_{PS}$) and preferred end time ($T_{PE}$) ($PT \le T_{PE} - T_{PS}$), then proceed to step two. Otherwise, report the schedule

```
        is rejected and quit.
Step 2: Let $R = \{r(n_1), r(n_2), \ldots, r(n_{|N|})\}$, where $r(n_i) = \{T_S(n_i), T_E(n_i), p_n, B(p_n)\}$
        be a set of resources booking specifications for the scheduled tasks in the
        task set N.
   DO
   2.1 Select a request $r(n_x)$ from $R$ and book the required resources for the
       scheduled task $n_x$ through the resources booking mechanism. Then remove
       $n_x$ from $R$.
   2.2 Collect resources booking acknowledgement information from lower layer
       contractor. If the booking is rejected, release the booked resources. Report
       the schedule is rejected and quit.
   WHILE $R$ is not empty AND the schedule is not rejected.
```

**Algorithm 5.2. The Scheduler Oriented HDSC algorithm (SO-HDSC).**

### 5.3.2 The Scheduler Oriented HMD Algorithm (SO-HMD)

For an application consists of $n$ task $N = \{n_1, n_2, \ldots, n_n\}$ with their relationship represented in certain format such as direct acyclic task graph (DAG). Assume $P = \{p_1, p_2, \ldots, p_k\}$ be a set of $k$ processors that satisfies the physical resources requirement to execute the application. Let $T_{PS}$ and $T_{PE}$ be the pre-specified start time and the pre-specified end time for executing the application respectively.

```
   Step 1:
   1.1 Using HMD algorithm, schedule the tasks into $P$ without considering the
       workload situation of $P$.
   1.2 If the parallel time (PT) of the produced schedule is completed within the
       period of preferred start time ($T_{PS}$) and preferred end time ($T_{PE}$)
       ($PT \le T_{PE} - T_{PS}$), then proceed to step two. Otherwise, report the schedule
       is rejected and quit.
Step 2: Let $R^* = \{r(n_1), r(n_2), \ldots, r(n_{|N|})\}$, where $r(n_i) = \{T_S(n_i), T_E(n_i), p_n, B(p_n)\}$
        be a set of resources booking specifications for the scheduled tasks in the
        task set N.
   DO
   2.1 Select a request $r(n_x)$ from $R$ and book the required resources for the
       scheduled task $n_x$ through the resources booking mechanism. Then remove
       $n_x$ from $R$.
   2.2 Collect resources booking acknowledgement information from lower layer
       contractor. If the booking is rejected, release the booked resources. Report
       the schedule is rejected and quit.
```

[*] The $R$ and $r(n_i)$ is defined in definition 5.1

```
WHILE $R$ is not empty AND the schedule is not rejected.
```

**Algorithm 5.3. The Scheduler Oriented HMD algorithm (SO-HMD).**

### 5.3.3 The Scheduler Oriented HRMS algorithm (SO-HRMS)

For an application consists of $n$ task $N = \{n_1, n_2, \ldots, n_n\}$ with their relationship represented in certain format such as direct acyclic task graph (DAG). Assume $P = \{p_1, p_2, \ldots, p_k\}$ be a set of $k$ processors that satisfies the physical resources requirement to execute the application. Let $T_{PS}$ and $T_{PE}$ be the pre-specified start time and the pre-specified end time for executing the application respectively.

```
    Step 1:
  1.1  Using HRMS algorithm, schedule the tasks into $P$ without considering the
       workload situation of $P$.
  1.2  If the parallel time (PT) of the produced schedule is completed within the
       period of preferred start time ($T_{PS}$) and preferred end time ($T_{PE}$)
       ($PT \le T_{PE} - T_{PS}$), then proceed to step two. Otherwise, report the schedule
       is rejected and quit.
Step 2:  Let $R = \{r(n_1), r(n_2), \ldots, r(n_{|N|})\}$, where $r(n_i) = \{T_S(n_i), T_E(n_i), p_n, B(p_n)\}$
         be a set of resources booking specifications for the scheduled tasks in the
         task set $N$.
    DO
  2.1  Select a request $r(n_x)$ from $R$ and book the required resources for the
       scheduled task $n_x$ through the resources booking mechanism. Then remove
       $n_x$ from $R$.
  2.2  Collect resources booking acknowledgement information from lower layer
       contractor. If the booking is rejected, release the booked resources. Report
       the schedule is rejected and quit.
    WHILE $R$ is not empty AND the schedule is not rejected.
```

**Algorithm 5.4. The Scheduler Oriented HRMS algorithm (SO-HRMS).**

## 5.4 Properties of SO Based Algorithms

### 5.4.1 Resources acquisition

This is shown in Experiment 3 of Chapter 8 that the rate of successful booking resources decreases as the number of task increases and the workload situation of the booking resources is busy. Besides, A schedule produced by task scheduler does not guarantee their required resources that can be acquired for the execution. This is because task scheduler does not take into account of the workload situation during scheduling. Since the booking of the required resources cannot be guaranteed. Even with the best obtainable in Step 1, does not imply applicable can be executed according to the prescribed schedule. This execution performance can only be guaranteed while all tasks in the application can obtained their required resources.

### 5.4.2 Efficiency in resources utilization

The Scheduler Oriented approach achieves zero wastage in booked resources. Resources are booked in accordance with a produced schedule, the booked resources should be consumed by the application. The un-booked areas can be freely assigned to other application programs/users. Therefore, there is no wastage of resources created by task schedulers or applications under scheduling.

### 5.4.3 Property of the processors table after scheduling

The booked resources are in small pieces and they are distributed among the set of processors being booked. The size of each booking is proportional to the cost of task in the scheduling task graph. Their sizes are small as compared with the application program. They are distributed among the booking processors set resulted from maintaining dependence constrains among the tasks.

Besides, each booking area should consume 100% of processing power of the booked processors. It is because classical scheduling algorithm is used for task scheduling, they assume tasks are executed in 100% of the processor speed (scheduled to a set of dedicated processors). The processing power consumed by each booking portion is 100%. Furthermore, Concurrent execution of tasks is suppressed in the SO approach. According to Property 5.5, each of the booking area should consume 100% of processing power of the booked processors. Therefore, the scheduled task cannot be executed concurrently with other tasks in the booking region.

### 5.4.4 Performance in task scheduling

For resources can be granted to all the scheduled tasks, the performance of the produced schedule can be attended to the claimed performance of the scheduling algorithm using. It is because the SO approach is to schedule the task graph before reserving resources. The scheduling procedure makes the assumption of using the resources with maximum availability. Hence, the performance of the produced schedule should be the maximum achievable performance of the scheduling algorithm in the given resources. Besides, owing to the properties of resources guaranteed while they are granted, the application runs on a multiple-users multiple-tasks environment in a dedication environment. Therefore, the scheduling performance is guaranteed.

Besides, the resources booking probability is independent of the core task scheduling using. This is because the task-scheduling algorithm using in the SO approach does not cater for the workload situation. The probability of successful resources booking for schedules that are produced by difference algorithms is the same. Base on the arguments, there is no classical task-scheduling algorithm (under our considerations) can take advantage from this approach.

### 5.4.5 Inherence problems

Applications may not be executed due to unsatisfactory scheduling result or insufficient booked resources. Since the major difference in executing application programs between contractual computing paradigm and traditional computing paradigm is that resources should be booked in advance. An application cannot be executed while one of the tasks cannot book its required resources. As shown in Property 5.1 and Experiment 3 (Chapter 8), the resources booking rate for an application is very low. The application program cannot be started in most situations. In order to keep the program executable, application users should release the constraint on the execution such as a delay of the preferred start time or an extend of the preferred end time.

In a traditional computation paradigm, if an application is being executed, another application can still be executed concurrently. That would result in suffering delay. The application is guaranteed to be executable. However, using the SO approach an application cannot be executed concurrently with other applications when the booked region of the application is collided with other applications. That is the shortfall of using the SO approach on the paradigm.

### 5.4.6 Applicability

The proposed SO approach is suitable for scheduling applications with the following characteristics:

1. An application proceeds with its own schedule (the schedule is based on the assumption of dedicated resources) and looks for a stable and reliable execution environment for execution.
2. An application can be effectively scheduled by classical/traditional task scheduling algorithms (the schedule is based on the assumption of dedicated resources) and looks for a stable and reliable execution environment for execution.
3. Small number of sub-tasks can be decomposed by the scheduling application (refer to Experiment 3 and Experiment 4 in Chapter 8).
4. The size of the scheduling sub-tasks in an application is small (refer to Experiment 3 in Chapter 8).
5. The workload situation of the target processor set is low. Ideally, the best performance of this approach is to schedule task in an off-loaded processor set.

# Chapter 6: The Resources Oriented (RO) Approach

## 6.1 Chapter Summary

This Chapter studies the Resources Oriented approach for scheduling tasks in practical parallel and distributed systems running with the contractual computing paradigm. The idea of scheduling tasks into dedicated processors in classical scheduling algorithm is highlighted and it is possible to work in practical parallel and distributed systems. Conceptually speaking, the concept of cooperative usage of shared resources by partitioning is widely accepted in classical computing paradigm. For examples, hard disk spaces partitioning database partitioning in distributed database management and module concept in multiple-access memory system. However, owing to the belief that processing power is non-deterministic and uncontrollable, the concept is seldom extended to processing power partitioning. On the development of contractual computing paradigm, processing power becomes deterministic and controllable. The idea of resources partitioning is extended to processing power. The Resources Oriented approach is proposed for scheduling tasks using a dedicated resource partition.

Based on the concept of resources partitioning, a virtually dedicated processor set (VDP) is obtained. Classical task scheduling and application with its own schedule can follow the approach to schedule and execute tasks in a practical non-deterministic resource environment. Since the booked resources in a VDP set are dedicated to an application, the application can be executed in a dedicated environment. This approach is very suitable for those applications with its own schedule or the classical scheduling algorithm is retained for scheduling tasks in the contractual computing paradigm.

## 6.2 The Resources Oriented Approach

The "Resources Oriented" approach is based or using resources booking for schedule design. The booking of processing resource in advance is to form a “Virtual Dedicated Processor” set before performing task scheduling. Having secured the resources in advance by this booking mechanism task can be scheduled to the VDP, which can be regarded as a dedicated environment. Therefore, classical task scheduling algorithms such as List Scheduling Heuristic (LSH) or Heterogeneous List Scheduling Heuristic (HLST) can be directly implemented. Scheduling solutions can then execute in a practical parallel and distributed system.

### 6.2.1 An algorithmic description of the RO approach

The RO approach is generalized in two-step algorithm shown in Algorithm 6.1. Step 1 is a partitioning procedure which partitions part of processing resources from each processor of a given processor set *P* to form a virtually dedicated processor (VDP) set *C*. A generic method for resources partition is described in Section 6.2.2. With the VDP, classical scheduling algorithms such as List Scheduling Heuristic (LSH) or Heterogeneous List Scheduling Heuristic (HLST) can be directly applied for task scheduling in Step 2.1. Finally, the produced schedule is undergoing an inspection procedure in Step 2.2 to ensure that the produced schedule can meet the pre-defined scheduling specification.

For an application consists of $n$ task $N=\{n_1,n_2,\ldots,n_n\}$ with their relationship represented in certain format such as a direct acyclic task graph (DAG). And $P=\{p_1,p_2,\ldots,p_k\}$ be a set of $k$ processors that satisfies the physical resources requirement to execute the application.

```
Step 1:  Resources partitioning
         Using procedure of resources partitioning such as Algorithm 6.2,
         partition the given processor set $P$ to form a set of
         homogeneous/heterogeneous VDP set $C=\{c_{p1},c_{p2},\ldots,c_{pk}\}$, where
         $c_{p1} \in p_1, c_{p2} \in p_2 \cdots$ etc.
Step 2   For a VDP, set $C$ cannot be obtained in Step 1. Report the schedule is
         rejected and quit. Otherwise,
    2.1  Perform classical homogeneous/heterogeneous task-scheduling
         algorithms for scheduling tasks into the VDP set $C$.
    2.2  The produced schedule can start beyond the pre-specified start
         time ($T_{PS}$). In addition, parallel time ($PT$) of the produced
         schedule is less than or equal to the period pre-specified start
         ($T_{PS}$) and pre-specified end time ($T_{PE}$) ($PT \le T_{PE} - T_{PS}$), then
         report the schedule is accepted. Otherwise, report the schedule
         is rejected. Release the booked resources and quit.
```

**Algorithm 6.1. The Resources Oriented (RO) approach.**

### 6.2.2 Resources partitioning

Resource partitioning is the first and the most important step in design based on the RO approach. The partition part of processing power from the given processor set *P* forms a virtual dedicated processor (VDP) set $C$. In multiple-users and multiple-tasks scenario, any tasks/applications can put forward their request simultaneously in practical parallel and distributed systems. However, with such a varying situation, the problem is how a dedicated processor environment can be partitioned. This problem is addressed by advance resources booking and resources in the proposed contractual computing paradigm. This provides an advanced scheduling information described in Section 4.2.3. According to the information and the advanced resource booking mechanism, advanced resources booking can form a VDP set. In this, part of processing power in a processor set *P* is partitioned to form VDP set $C$ (where

$C = \{c_{p1}, c_{p2}, \ldots, c_{p|P|}\}$) in a given period of time. Before going into details of resources partitioning method, the characteristics of the VDP set are explored in the following sections.

## *A) Characteristics of a VDP set*

The concept of a VDP set is to generate a vital dedicated resource space, which looks as if a resource deterministic environment for tasks scheduling. Therefore, the characteristics of a VDP set should be the design of classical scheduling algorithms being proceeded. The feature of the List Scheduling Heuristic and Heterogeneous List Scheduling Heuristic must abeit to the VDP set $C$ possesses with the following characteristics:

- Start time and end time of each partition ($c_{pi}$) in the partition set $C$ should be the same.
- The amount of resource (such as processing power) within a partition $c_{pi}$ should be constant.
- The amount of resource (such as processing power) between partitions not necessary be identical; in such case, a heterogeneous VDP set is formed. Otherwise, a homogeneous VDP set is formed.
- Total capacity (processing power) of the VDP set $C$ should be greater than or equal to that required by the application to be executed.

**Example 6.1**

Consider an application, which costs 110 instructions, should start at t=1s and stop at t=7s. A possible VDP set $C$ is given in Figure 6.1. The start time and the end time of each partition should be the same so that the VDP are transparent to any scheduling algorithms. In real-life, most designs of classical scheduling algorithms assume that each processor has constant processing power. Therefore, the processing power within each partition of the VDP set should keep constant. However, the amount of processing power between different partitions may not necessary be the same. It is possible to form a homogeneous VDP by a VDP set that each cluster has the same processing power. Homogeneous task scheduling techniques such as List Scheduling Heuristic can be directly applied. However, a heterogeneous VDP set can be formed by a VDP set that each cluster has difference processing power. Advanced task-scheduling techniques such as Heterogeneous List Scheduling Heuristic proposed in previous Chapter can be employed. Again total processing power of the obtained VDP should be equal to the processing power required by the application; otherwise, the application program cannot be completed due to lack of resources.

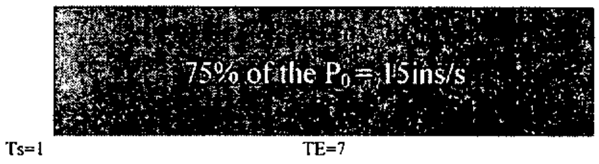
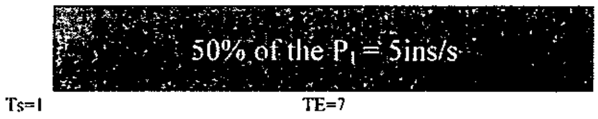
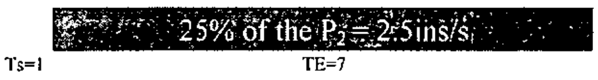
| Processor set $P$ | Partition specifications | Obtained VDP set C | Format of the obtained partitions |
|---|---|---|---|
| $P_0$=20ins/s | TS=1, TE=7, 75% of the $P_0$ | $C_{p0}$ = 15 ins/s |  |
| $P_1$=10ins/s | TS=1, TE=7, 50% of the $P_1$ | $C_{p1}$ = 5 ins/s |  |
| $P_2$=10ins/s | TS=1, TE=7, 25% of the $P_2$ | $C_{p2}$=2.5 ins/s |  |

**Figure 6.1. Characteristic of VDP set *C***

## B) *A resources partitioning method: Partition by Assignment*

Algorithmic description of a generic resources method "Partition by assignment" is presented as Algorithm 6.2. This is to partition the available processing power of a given processor set in accordance with a set of specification(s) called "Partition Specifications" (PS) which is given by the application users. The PS pre-specifies the start time of the application, the end time of executing the program and the amount of processing power required in each partition. Column 2 of Figure 6.1 shows an example of partitioning a VDP set $C$. Based on the specification, resources bookings are submitted to resources providers, such as Local Manager (LM) for resources granted and once all the required resources are granted, a VDP set $C$ can be obtained.

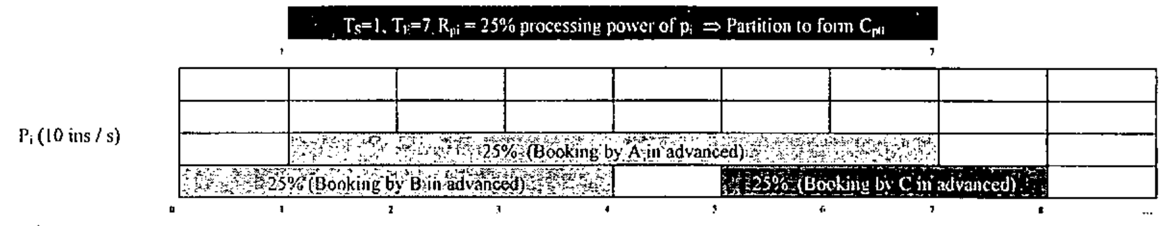
```
Let $T_S$ and $T_E$ denote the pre-specified start and end time of an application
program respectively, $I$ be the processing power required by the application
program. For processors set $P = \{p_0, p_1, \ldots, p_{|P|}\}$ (where $|P|$ be the total number of
processors in $P$) can be partitioned and used by the scheduler. Let
$C = \{c_{p0}, c_{p1}, \ldots, c_{p|P|}\}$, where $p_i \in P$ be the set of partition specification for each
processor in $P$. And let $C$ initially be an empty set,
    Step 1:
        1.1 Determine partition specification, $T_S$, $T_E$ and the set $R$, from application
            users.
            FOR i = 0 to $|P|$
            1.2 Submit a resource request with the partition specification
                $\{T_S, T_E, R_{pi}\}$ to the lower layer contractors for resources grant.
            1.3 The resources granter assesses the specification with its current and
                scheduled resources. The resources are granted while the
                specification is achievable. A positive acknowledgement returns with
        the exact amount of granted resources. Otherwise, return with a
        negative acknowledgement.
    1.4 Upon receiving the acknowledgement, a partition of processors $c_{pi}$ is
        created. If the required resources can be granted, the amount of
        processing power of $c_{pi}$ is equal to the amount of granted resources
        returned from the resources granter. Otherwise, the processing
        power of $c_{pi}$ is equal to zero.
    NEXT i
Step 2:
    If total processing power of the obtained VDP set $C$ is greater than that of
    $I$, return set $C$. Otherwise, release the booked resources and report failure
    in resources partitioning.
```

**Algorithm 6.2. Resources partitioning by assignment.**

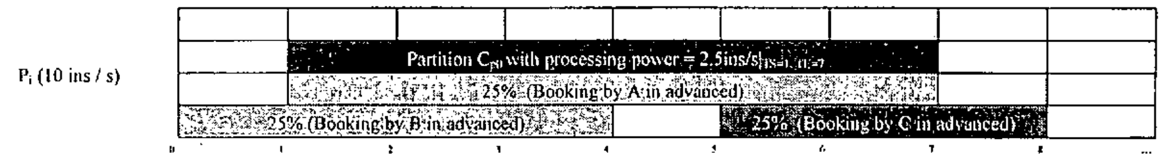
The Step 1.1 of the algorithm is to obtain Partition Specification (PS) from user's application. The partition specification is a triple composite of the pre-specified start time of the partition ($T_S$), the pre-specified end time of the partition ($T_E$) and the amount of processing power required (R) to describe the shape of VDP set. When the application starts, $T_S$ is being specified. The value of $T_S$ can be easily obtained from users, but it is difficult to determine $T_E$ and VDP (R) automatically due to their dependence on the application configuration. This information can only be obtained from a generated schedule. This may be contradicted with the design concept of the Resources Oriented approach. Therefore, the exact amount of resources required in each partition cannot be determined directly and users should try their best to estimate pre-specified end time and pre-specified partition specification $C = \{c_{p0}, c_{p1}, \ldots, c_{p|P|}\}$ of the desired VDP set in the partition.

In Step 1.2 to Step 1.4, resource booking is requested to the lower layer resources provider for the VDP set. Figure 6.2a-c illustrates the procedure for granting of requested resources granted. Consider a partition specification occupied 25% processing power of $p_i$ between the first and the seventh second (R=25%, $T_S$=1, $T_E$=7). The instantaneous workload of processor $p_i$ is summarized in Figure 6.2a. It is found that 50% of processing power had been booked by application A, B and $C$ during that period and the other 50% re3main free. Then, the lower level contractor in $p_i$ is possible to book 25% of the remaining processing power from the prescribed duration (i.e. t=1 to 7). Partition $c_{pi}$ with processing power of 2.5 instructions in one seccond (ins/s) as it is obtained as shown in Figure 6.2b. Say, there is another application D (in Figure 6.2c), which had booked the remaining 50% of processing power fromt=3 to 6. Then the request for the 25% processing power requested from the first to the seventh second cannot be satisfied.

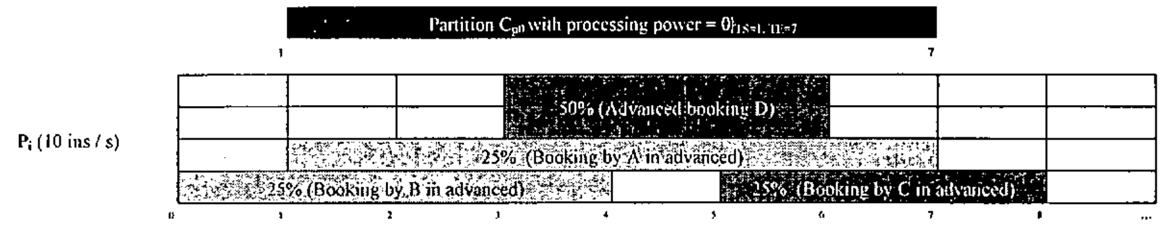
Therefore, the processing power of partition $c_{pi}$ becomes 0. Hence, no cluster is available for the schedule.


$T_S=1, T_E=7, R_{pi}$ = 25% processing power of $p_i \Rightarrow$ Partition to form $C_{pi}$
1
7
$P_i$ (10 ins / s)
25% (Booking by A in advanced)
25% (Booking by B in advanced)
25% (Booking by C in advanced)
0
1
2
3
4
5
6
7
8
...

(6.2a)


$P_i$ (10 ins / s)
Partition $C_{pi}$ with processing power = 2.5ins/s$|_{TS=1, TE=7}$
25% (Booking by A in advanced)
25% (Booking by B in advanced)
25% (Booking by C in advanced)
0
1
2
3
4
5
6
7
8
...

(6.2b)


Partition $C_{pi}$ with processing power = 0$|_{TS=1, TE=7}$
1
7
$P_i$ (10 ins / s)
50% (Advanced booking D)
25% (Booking by A in advanced)
25% (Booking by B in advanced)
25% (Booking by C in advanced)
0
1
2
3
4
5
6
7
8
...

(6.2c)

**Figure 6.2a-c. an example of resources partition by assignment.**

After obtaining a VDP set the partitioning algorithm (Step 2) is used to probe into the amount of resources for VDP to complete an application. In real-life, the workload situation can be varied. It is possible to guarantee the desired partition through any partitioning procedure. For the total processing powers of VDP set $C$ less than that of the required by an application, the application cannot be executed in the VDP set due to the insufficient of resources. Then the booked resources should be released and the scheduling procedure terminated. On the other hand, the obtained VDP set possesses the least resources required to execute the application program, which may be executed in domain $C$, through proper task ordering. With classical scheduling algorithm applying to schedule tasks into the VDP set, $C$, a desired solution can be obtained in Step 2 of the RO approach.

### 6.2.3 Invoking scheduling algorithms

After a VDP set is obtained, the second step of the RO approach is to schedule the application. Since the obtained VDP satisfies the necessary assumptions, with dedicated resources in applying classical scheduling algorithm. The procedure of LS and HLS algorithm can be applied directly in homogeneous or heterogeneous VDP set respectively in a real processor environment. The only difference between the VDP and the real dedicated processor is that the

VDP is only consistent within the pre-specified interval ($T_S$ to $T_E$). The booked resources can only be guaranteed for a task scheduled within $T_S$ and $T_E$. In Step 2.2 this situation is examined so that the produced schedule should be executed within that interval.

## 6.3 Implementations of Scheduling Using RO Approach

Three proposed implementations of task scheduling algorithms applying the Resources Oriented attempt is presented in this section. They are the Resources Oriented Heterogeneous Dominant Sequent Cluster algorithm (RO-HDSC), the Resources Oriented Heterogeneous Mobility Direct algorithm (RO-HMD) and the Resources Oriented Heterogeneous Relative Mobility algorithm (RO-HRMS). Through the designs and implementations of these algorithms, the properties of the RO approach can be observed. The design procedure is similar to that of Chapter 5.3.

### 6.3.1 The Resources Oriented HDSC Algorithm (RO-HDSC)

For an application consists of $n$ task $N=\{n_1,n_2,\ldots,n_n\}$ with their relationship represented in certain format such as direct acyclic task graph (DAG). Moreover $P=\{p_1,p_2,\ldots,p_k\}$ be a set of $k$ processors that satisfies the physical resources requirement to execute the application. Let $T_{PS}$ and $T_{PE}$ denote the pre-specified start and end time of an application program respectively, $I$ be the processing power required by the application program. Processors set $P=\{p_1,p_2,\ldots,p_k\}$ can be partitioned and used by the scheduler. Let $C=\{c_{p0},c_{p1},\ldots,c_{p|P|}\}$ where $p_i \in P$ be the set of partition specifications for each processor in $P$. And, let $C$ initially be an empty set,

```
Step 1:
  1.1 Determine the partition specification $T_S$, $T_E$ and the set $R$ from
      application users.
FOR i = 0 to $|P|$
  1.2 Submit a resource request with the partition specification $\{T_s,T_E,R_{pi}\}$ to
      the lower layer contractors for resources granted.
  1.3 The resources granter assesses the specification with its current and
      scheduled resources. The resources are granted while the specification is
      achievable. A positive acknowledgement returns with the exact amount of
      granted resources. Otherwise, return with a negative acknowledgement.
  1.4 Upon receiving the acknowledgement, a partition of processors $c_{pi}$ is
      created. If the required resources can be granted, the amount of
      processing power of $c_{pi}$ is equal to the amount of granted resources
      returned from the resources granter. Otherwise, the processing power of
      $c_{pi}$ is equal to zero.
```

```
NEXT i
 Step 2:
    If total processing power of the obtained VDP set $C$ is greater than that of
    $I$, return set $C$. Otherwise, release the booked resources and report failure in
    resources partitioning.
Step 3  For a VDP set $C$ cannot be obtained in Step 1 report the schedule is
        rejected and quit. Otherwise,
   3.1  Perform HDSC for scheduling tasks into the VDP set $C$.
   3.2  The produced schedule can start beyond the pre-specified start time ($T_{PS}$).
        Moreover, parallel time (PT) of the produced schedule is less than or equal
        to the period pre-specified start ($T_{PS}$) and pre-specified end time ($T_{PE}$)
        ($PT \le T_{PE} - T_{PS}$), then report the schedule is accepted. Otherwise, report
        the schedule is rejected. Release the booked resources and quit.
```

**Algorithm 6.3. The HDSC algorithm in Resources Oriented Implementation (RO-HDSC).**

### 6.3.2 The Resources Oriented HMD algorithm (RO-HMD)

```
For an application consists of $n$ task $N = \{n_1, n_2, \ldots, n_n\}$ with their relationship
represented in certain format such as direct acyclic task graph (DAG).
Moreover $P = \{p_1, p_2, \ldots, p_k\}$ be a set of $k$ processors that satisfies the physical
resources requirement to execute the application. Let $T_{PS}$ and $T_{PE}$ denote the pre-
specified start and end time of an application program respectively, $I$ be the
processing power required by the application program. Processors set
$P = \{p_1, p_2, \ldots, p_k\}$ can be partitioned and used by the scheduler. Let
$C = \{c_{p0}, c_{p1}, \ldots, c_{p|P|}\}$ where $p_i \in P$ be the set of partition specifications for each
processor in $P$. And let $C$ initially be an empty set,
Step 1:
   1.1  Determine the partition specification $T_S$, $T_E$ and the set $R$ from
        application users.
  FOR i = 0 to $|P|$
   1.2  Submit a resource request with the partition specification $\{T_s, T_E, R_{pi}\}$ to
        the lower layer contractors for resources granted.
   1.3  The resources granter assesses the specification with its current and
        scheduled resources. The resources are granted while the specification is
        achievable. A positive acknowledgement returns with the exact amount of
        granted resources. Otherwise, return with a negative acknowledgement.
   1.4  Upon receiving the acknowledgement, a partition of processors $c_{pi}$ is
        created. If the required resources can be granted, the amount of
        processing power of $c_{pi}$ is equal to the amount of granted resources
        returned from the resources granter. Otherwise, the processing power of
        $c_{pi}$ is equal to zero.
```

```
NEXT i
Step 2:
    If total processing power of the obtained VDP set $C$ is greater than that of
    $I$, return set $C$. Otherwise, release the booked resources and report failure in
    resources partitioning.
Step 3  For a VDP set $C$ cannot be obtained in Step 1 report the schedule is
        rejected and quit. Otherwise,
    3.1 Perform HMD for scheduling tasks into the VDP set $C$.
    3.2 The produced schedule can start beyond the pre-specified start time $(T_{PS})$.
        Moreover, parallel time (PT) of the produced schedule is less than or equal
        to the period pre-specified start $(T_{PS})$ and pre-specified end time $(T_{PE})$
        $(PT \le T_{PE} - T_{PS})$, then report the schedule is accepted. Otherwise, report
        the schedule is rejected. Release the booked resources and quit.
```

**Algorithm 6.4. The Resources Oriented HMD algorithm (RO-HMD).**

### 6.3.3 The Resources Oriented HRMS algorithm (RO-HRMS)

```
For an application consists of $n$ task $N = \{n_1, n_2, \ldots, n_n\}$ with their relationship
represented in certain format such as direct acyclic task graph (DAG).
Moreover $P = \{p_1, p_2, \ldots, p_k\}$ be a set of $k$ processors that satisfies the physical
resources requirement to execute the application. Let $T_{PS}$ and $T_{PE}$ denote the pre-
specified start and end time of an application program respectively, $I$ be the
processing power required by the application program. Processors set
$P = \{p_1, p_2, \ldots, p_k\}$ can be partitioned and used by the scheduler. Let
$C = \{c_{p0}, c_{p1}, \ldots, c_{p|P|}\}$ where $p_i \in P$ be the set of partition specifications for each
processor in $P$. And let $C$ initially be an empty set,
Step 1:
    1.1 Determine the partition specification $T_S$, $T_E$ and the set $R$ from
        application users.
  FOR i = 0 to $|P|$
    1.2 Submit a resource request with the partition specification $\{T_S, T_E, R_{pi}\}$ to
        the lower layer contractors for resources granted.
    1.3 The resources granter assesses the specification with its current and
        scheduled resources. The resources are granted while the specification is
        achievable. A positive acknowledgement returns with the exact amount of
        granted resources. Otherwise, return with a negative acknowledgement.
    1.4 Upon receiving the acknowledgement, a partition of processors $c_{pi}$ is
        created. If the required resources can be granted, the amount of
        processing power of $c_{pi}$ is equal to the amount of granted resources
        returned from the resources granter. Otherwise, the processing power of
        $c_{pi}$ is equal to zero.
    NEXT i
Step 2:
    If total processing power of the obtained VDP set $C$ is greater than that of $I$,
    return set $C$. Otherwise, release the booked resources and report failure in
    resources partitioning.
Step 3  For a VDP set $C$ cannot be obtained in Step 1 report the schedule is
        rejected and quit. Otherwise,
    3.1 Perform HRMS for scheduling tasks into the VDP set $C$.
    3.2 The produced schedule can start beyond the pre-specified start time ($T_{PS}$).
        Moreover, parallel time (PT) of the produced schedule is less than or equal
        to the period pre-specified start ($T_{PS}$) and pre-specified end time ($T_{PE}$)
        ($PT \le T_{PE} - T_{PS}$), then report the schedule is accepted. Otherwise, report
        the schedule is rejected. Release the booked resources and quit.
```

**Algorithm 6.5. The Resources Oriented HRMS algorithm (RO-HRMS).**

## 6.4 Properties of RO Based algorithms

### 6.4.1 Resources acquisition

According to Experiment 3 (Chapter 8), the successful rate of obtaining a partition depends upon the workload of the partitioning processor. It has found that the successful rate decreases as the workload situation of the partitioning processors set increases. It has also found that in the same experiment that the amount of processing power required by the partition increases as the successful rate decreases. These findings can be explained by the fact that resources are completed among different applications. While an application consuming resources, less resource becomes available to other applications. Therefore, workload of the partitioning processor increases and the successful rate of obtaining resources partition by other applications would decrease. The successful rate of obtaining a partition depends upon the workload situation of the partitioning processor, the partition period and the amount of processing power required

Since it is not always possible to obtain a successful partition, a VDP $c_{pi}$ does not guarantee obtainable from a partitioning processor $p_i$. In such a way, the application program cannot be scheduled and executed. Although the total processing power of obtained VDP set satisfies the basic requirements, the application cannot be guaranteed schedulable and executable in the VDP set. This is because the dependent constrains and parallelism of parallel program would dominate the usage of resources in different partitions. The application program may be scheduled out of the boundary region of resources guaranteed in the VDP set. Consequently, the application may not be executable simply due to insufficient resources. To sum up, an application program can not guarantee schedulable and executable in a obtained VDP set.

On the other hand, if VDP set can be form, those reserved resources are guaranteed available and usable within the booked intervals (or booked partitions) in run time when resources are granted. Therefore, the task scheduling problem is free from non-deterministic resources. There is no need to use others advanced task-scheduling techniques such as dynamic scheduling.

### 6.4.2 Efficiency in resources utilization

There would be wastage of resources in the Resources Oriented approach. This is because a block of resources is booked before task scheduling, the schedule may not occupy all the resources in the block due to parallelism and data dependence. There exist resources wastage among scheduled tasks inside the block. To calculate the wastage, let L be the amount of processing power being wasted, $S(c_x)$ be the speed of partition $c_x$, $I(A)$ be the total number of instructions of an application "A" and $T_S$ and $T_E$ be the start time and end time of the partition respectively.

Equation 6.1: $$L = \left[ \sum_{i=0}^{i=|P|} S(c_{pi}) \right] \times (T_E - T_S) - I(A)$$

According to Equation 6.1, reducing the interval of the partitions can minimize the waste of resources. An optimum partition is exactly equal to the parallel time of the scheduled application. Therefore, the resources waste beyond completing the application can be minimized.

### 6.4.3 Property of the processor tables after scheduling

The processor tables of partitioning processors are partitioned into partitions of wide interval with constant processing power. The Resources Oriented approach proposes to partition part of resources to form a virtually dedicated processor set (VDP) for task scheduling. According to the characteristics of VDP defined in Section 6.2.2A, the processors are booked to form an interval of constant processing power. Besides, the length of the interval should be at least the parallel time of the produced schedule. The width should always be much wider than a sub-task in the application.

### 6.4.4 Performance in task scheduling

The scheduling performance of the RO approach depends upon the configuration VDP set. The suggested implementation of RO approach is based on the Heterogeneous List Scheduling Heuristic. As mentioned in Chapter 4, Experiment 2 (Chapter 8), even the same scheduling algorithm is used for scheduling the same task graph. Parallel time of the produced schedule would be affected by the configuration of the target processors. The finding of the experiments shows that algorithms based on the HLST tend to schedule tasks into small number of high-speed processors. Therefore, parallel time of the produced schedule can be improved by scheduling task into VDP which partition as much processing power from the partitioning processors as possible.

Moreover, the scheduling performance of the RO approach depends upon the scheduling algorithm used. In Chapter 4, Experiment 2 (Chapter 8) it is found that different scheduling algorithms would obtain different scheduling outputs in the same set of VDP. For example, according to the experimental result of the suggested implementations, the RO-HDSC can produce a shorter parallel time scheduling solution than that of RO-HMD and RO-HRMS while the same set of VDP, which the application is schedulable in the VDP, is used as the target platform. This gives an insight that resources are limited by the VDP set. The use of different scheduling techniques can improve the task scheduling performance. This catalyzes the development of both homogeneous and heterogeneous scheduling algorithm for the application in practical parallel and distributed systems.

### 6.4.5 Inherence problems

To obtain an optimum solution of resources partition is the great challenge of the RO approach. An optimal VDP solution should optimum for resources utilization and application performance. This is because partitions with large amounts of resources may not be fully used by application requirements. On the other hand, the application program may not be executed due to lack of resources. The optimum amount of resources for each partition should be the exact amount of resources required by the application. However, resource partitioning is done ahead of task scheduling. The knowledge of where the application program be scheduled is lacking. Therefore, the exact amount of resources required for each partition is not known in the resources partitioning procedure. An optimum solution is hard to obtain.

Besides, it does not guarantee that a schedule, which can be successfully executed within the pre-specified period, is obtainable. Firstly, the VDP set is obtained by booking parts of processing power from the given processor set. The amount of booked processing power depends upon the workload situation of the processor set within the booking period. It is found from Experiment 3 (Chapter 8) while the processor set is in high workload situation, the amount of processing power in each cluster becomes low. The application program may not be executed because of insufficient resources. Secondly, the goal of most classical scheduling algorithms is focused on parallel time reduction but not on the pre-specified start and end time. Therefore, the use of classical scheduling algorithms in RO approach for scheduling tasks does not guarantee the application can be completed within the required period.

### 6.4.6 Applicability

The proposed RO approach is suitable for scheduling applications with the following characteristics:

1. The application can be effectively scheduled by existing task scheduling algorithms (the schedule is based on the assumption of dedicated resources) but it looks for a stable and reliable execution environment for execution.
2. To minimize non-utilized resources, the RO approach is more favorable for applications with coarse grain (the cost of computation is much larger than the cost of communication) and applications of low degree of parallelism (The application is highly loaded on few processors).
3. The information of "partition specifications" is known.
4. The existence a resource partitioning strategy for maintaining resources partitioning among different users and applications.
5. Resources wastage is acceptable.

# Chapter 7: The Contractor Oriented (CO) Approach

## 7.1 Chapter Summary

The third approach, the Contractor Oriented approach for designing and implementing task scheduling algorithms in the contractual computing paradigm is presented. In this approach, scheduling maintained from lower layer contractors is based to make task-scheduling decision. Tasks are scheduled free from the additional workload introduced by other applications. With the resources booking mechanism, processor can be scheduled to serve different applications in different time intervals without contention. Hence, resources can be better utilized by all the applications. Concurrent task assignment and non-concurrent task assignment methods were introduced. Three different scheduling algorithms are presented and the performance and property of the CO approach is studied. These results were supported in Experiment 4 and 5 (Chapter 8). The performance of the proposed scheduling algorithm under the CO approach is found to be superior than that of the SO and the RO approaches in real-life environment.

## 7.2 The Contractor Oriented Approach

In the Contractor Oriented (CO) approach uses the advanced scheduling information is used for making scheduling decisions. Once a task is scheduled, resources are booked immediately. As in Section 4.2.3, the advanced scheduling information tells explicitly the current and scheduled resources, which states precisely the workload and the available resources in any time interval. Based on the information, task schedulers can foresee the workload situation to make an appropriate scheduling decision. After a scheduling decision is made, those resources required by the procedure schedules are booked immediately and the advanced scheduling information is automatically updated. Therefore, tasks created in other applications can be scheduled in accordance with the updated information. Resources can be shared among different applications in a mutually cooperative manner.

```
For an application consists of $n$ task $N = \{n_1, n_2, \ldots, n_n\}$ schedule to a processors
set $P = \{p_1, p_2, \ldots, p_k\}$ that satisfies the physical resources requirement to execute
the application. Let $T_{PS}$ and $T_{PE}$ denote the pre-specified start and end time of an
application program respectively
Step 1.  Until all tasks in the scheduling task graph are scheduled do the
    following:
    1.1 Select a task $n_x$ from the scheduling task-graph.
    1.2 Capture the advanced scheduling information from lower-layer resource
        managers.
    1.3 Based on the information, examine the workload situation of each
        processor to find a suitable place to schedule the task $n_x$.
    1.4 Schedule $n_x$ to the suitable processor and the suitable time slot by
        booking the required resources.
Step 2. If the scheduled application cannot be executed within the users pre-
        specified period, report the scheduling fail and release the booked
        resources then quit. Otherwise, report the schedule is complete.
```

**Algorithm 7.1. The Contractor Oriented approach.**

An algorithmic description of the design of the Contractor Oriented approach based on the LS Heuristic is shown in Algorithm 7.1. In Step 1.1, a task is selected from the scheduling task graph in accordance with certain task scheduling heuristics such as critical path or relative mobility. After that, the workload situations of all candidate processors are collected from the lower layer contractors in term of advanced scheduling information. Based on the information, in Step 1.3 a suitable location to schedule the selected task is obtained. Once the location is found, it would be used for the advanced resources booking mechanism to book the required resources (as in Step 1.4). Since the schedule is a step ahead of the current and scheduled workload situation, the booking could be achieved with ease. Finally, Steps 1.1 to 1.4 are repeated until all tasks in the scheduling task graph are scheduled. In the algorithm, two major parameters can be changed to obtain different scheduling results. They are the ways to identify the scheduling task $n_x$ from the application and the ways to use the advanced scheduling information for making a task scheduling decision. They are presented in the following sections

### 7.2.1 To select a task for scheduling

The method to identify an important task from an application to schedule is the key to a successful scheduling algorithm for the class of LS Heuristic. According to the list scheduling heuristic, all unscheduled tasks are assigned a priority to generate a priority list. Based on the priority list, the task with the highest priority is selected to be scheduled first. As mentioned, a number of scheduling heuristics is proposed to identify an importance of tasks. In the study, the critical path and the relative mobility are used to evaluate the suggested implementations. The study shows in Experiment 5 (Chapter 8) that different heuristics contribute differently on reaching different scheduling solutions.

### 7.2.2 Using advanced scheduling information in making scheduling decision

The concept of using advanced scheduling information for making a scheduling decision had never been used in classical scheduling algorithms. It is because such information does not traditionally exist. The advanced scheduling information provides an explicit and an updated current and scheduled information of the target processors. It includes a precise interval and position of processing power occupied by current and scheduled tasks. Since the lower layer resources managers (or contractors) maintain the information, task schedulers do not require to gather this information in order to maintain their consistence.

Using the information in scheduling algorithm design, task schedulers should obtain such information from lower layer resources managers (or contractors) first. Then, the information is used to find a suitable location that can schedule the selected task. To reduce the complexity of this procedure, heuristic can be employed. For example, the MD algorithm using heuristic of scheduling tasks is assigned to the first processor. The RMS algorithm has to find a processor that can satisfy Condition 1 for the task to be finished as soon as possible (as discussed in Chapter 3). However, the above heuristics cannot directly be applied in the CO approach. It is because alias tasks, which are not belonged to the scheduling application, exist and they are belonged to other applications in a practical execution environment. The concept of such heuristics is not valid. Satisfaction of Fact 1 is considered as not only the moving range of scheduled tasks in the scheduling application, but also the moving range of those alias tasks. Therefore, the design and the use of scheduling heuristic in CO approach should take extra precaution. Study of allowing and disallowing of concurrently task execution are also studied and presented in following sub-sections.

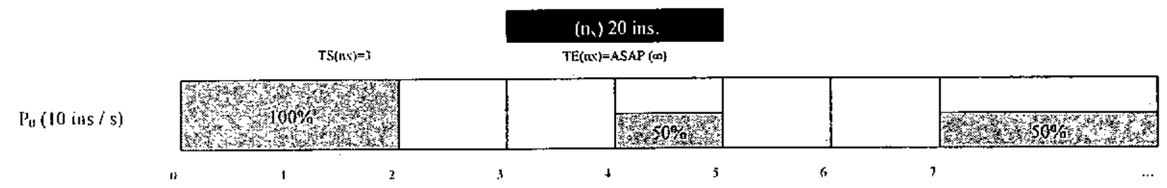
## *A) Tasks scheduling without inherit of concurrent execute*

The idea of this method comes from the insertion scheduling heuristic (ISH) that a task is only able to schedule in the idle period of two scheduled tasks. In other words, concurrent execution of task is prohibited. Its difference from the ISH is that the scheduled tasks in ISH must be those tasks that belonged to the same application. In practical parallel and distributed environment, the scheduled tasks may or may not belong to the same applications (or they can be alias tasks). It is obvious that while all the scheduled tasks belong to the same scheduling application (task graph), task scheduler can manage their organization. Advanced scheduling heuristic and rescheduling can be applied in order to obtain a better task allocation.

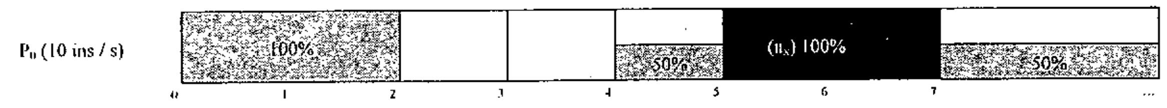
Resources booked for those alias tasks (those tasks are not belong to the same application program) cannot be overridden by task schedulers in any circumstances. Even task scheduler has scheduled a task to a suitable location where it is not possible to arrogate those resources to satisfy the booking request. So the task should be scheduled to another location, the scheduling performance may not be as good as desired. Nevertheless, this approach takes the advantage of providing the contractual computing paradigm. The booked resources are guaranteed and resources can be shared among different applications. This makes resources more efficiently utilized among different applications. However, there may be trade off between the scheduling performance and the efficiency of resources utilization in this approach.

***Example 7.1***

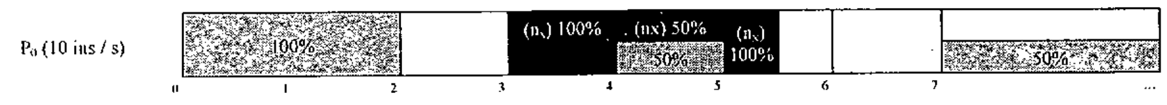
In Figure 7.1a-c, a task $n_x$ cost 20 instruction units (20 ins) is scheduled into processor $P_0$ with start time equal to 3 unit ($T_S(n_x) = 3$) and minimum finish time (i.e. $T_F(n_x)$ = As Soon As Possible (ASAP)). Figure 7.1a shows the scheduler with current scheduled information be the constraint specified as the booked request. At t = 0 to 2, $P_0$ are fully occupied by another tasks that consumed 100% of the processing power and $n_x$ should start at the third second and $n_x$ would not be scheduled on that portion. After that, there is an idle period at t = 2 to 4. According to the timing constrains of $n_x$ which must start at t = 3. With the cost of $n_x$ = 20 instructions, it cannot be completed between the third and the fourth second without overlapping with other scheduled tasks. The task $n_x$ will not be scheduled in that interval. Afterwards, another task with 50% processing power occupies the period between the fourth and the fifth second. From the principle of scheduling task that disallows concurrent tasks execution, $n_x$ will not be scheduled there. Finally, the task would be scheduled between fifth and seventh second shown in Figure 7.1b. The earliest finishing time of $n_x$ is at seven second.



**7.1a: Scheduling constraint**



**7.1b: Scheduling task disallows concurrent execution**



**7.1c: Scheduling task allows concurrent execution**

**Figure 7.1a-c. An example of tasks scheduling.**

From the above example, the CO approach can only schedule a task into a completely empty time slot. Even if a task is occupied in a small portion within the required period (say a task in the fourth and the fifth second inside the idle period between the third and the seventh second period), the task cannot be scheduled. This situation becomes worse when the scheduling of processors with high workload. The task scheduling performance would highly depend on the workload situation of the scheduling processors.

Scheduling that allows concurrent task execution is demonstrated in Figure 7.1c. It is found that scheduling task which allows concurrent execution of tasks cannot produce a better scheduling solution than scheduling tasks that disallows concurrent execution of tasks, but it uses resources in a more cost effective manner. Owing to this reason, another approach can schedule tasks in an overlapping manner.

### *B) Task scheduling with tasks concurrent execution*

With tasks being scheduled concurrently into a processor seems natural in classical computing paradigm because different tasks are already executed concurrently in a multiple-user and multiple-tasking environment. Processor scheduler within the OS kernel manages their concurrence. In this occasion, the execution behavior of the scheduled task is totally controlled by processor scheduler within the O.S. The schedule produced by task scheduler is meaningless due to non-determinism being introduced by the processor scheduler in the OS.

The problem is to be solved by using the Contractual Computing paradigm with advanced resources booking and the guaranteed resources. Owing to the guarantee, non-deterministic resources would no long. It greatly improves the reliability in this concurrent execution. A precise and reliable information on the current and scheduled resources is provided. Whenever a schedule overlaps with scheduled tasks, the amount of booked resources can be modified in the overlapped region. The scheduled task can then be executed concurrently with other tasks in the system. This dedicated execution environment is always maintained in the overlapped region and the idea is illustrated as an example 7.2.

### *Example 7.2*

In the previous example, the $n_x$ starts its execution at t=3, but it cannot be completed before the fourth seconds. There remain 10 instruction units to be executed. However, there are only 50% of processing power consumed by other applications between the fourth and fifth second. There remains 50% of processing power. It is found that if the remainder of $n_x$ can be executed concurrently with the idle region and $n_x$ can be completed in 5-time unit less. So the $n_x$ is scheduled with an overlapped scheduled task within the fourth and fifth seconds (it is shown in Figure 7.1c). Finally, the remaining five instructions of $n_x$ will be scheduled between the fifth and a half second. It is found that the task $n_x$ can be completed in the fifth and a half seconds instead of the seventh seconds. The parallel time of the produced schedule would shorter than that of using the previous method.

## 7.3 Implementations of Scheduling Using CO Approach

This section suggests three implementations of task scheduling algorithms applying the Contractor Oriented approach. Contractor Oriented HDSC algorithm (CO-HDSC), Heterogeneous Mobility Direct algorithm in Contractor Oriented implementation (CO- HMD) and Heterogeneous Relative Mobility algorithm in Contractor Oriented implementation (CO-HRMS) are shown in Section 7.3.1, Section 7.3.2 and Section 7.3.3 respectively. Their designs based on the HDSC, HMD and HRMS algorithm have been proposed in Chapter 4 respectively. The CO-HDSC algorithm follows the design of the HDSC that tries to schedule task to processor that the task can be finished as soon as possible. Other than this, the idea of concurrent task execution is applied to make a task scheduling decision. On the other hand, the CO-HMD and the CO- HRMS follows the design of the HMD and the HRMS algorithm respectively. The condition H using in the HMD and HRMS algorithm indeed applies the concept of scheduling tasks that disallows tasks executing concurrently. These algorithms are implemented in Experiment 5 (Chapter 8) to study the properties of the CO approach and the relationship between the scheduling performance and the inherence workload situation.

### 7.3.1 The Contractor Oriented HDSC Algorithm (CO-HDSC)

```
For an application consists of $N$ tasks schedule to a processors set
$P=\{p_1,p_2,\ldots,p_k\}$ that satisfies the physical resources requirement to execute the
application. Let $T_{PS}$ and $T_{PE}$ denote the pre-specified start and end time of an
application program respectively
1.  Sort the available processors set $P$ in the descending order of speed.
2.  EG=NULL, UEV=$N$, where EG = Examined Graph, UEG = Un-Examined
    Graph and $N$ is the set of tasks in the scheduling task graph.
3.  Every task is marked unexamined and assumed to virtually constitute to one
    unit processor cluster with reference speed $\varepsilon$.
4.  Set the top level is equal to zero for each free task and compute the top level as
    well as bottom level of each task.
5.  While $UEG \neq NULL$ do
    5.1 Using the sum of the top level and the bottom-level of a task as priority
        value, generate a priority list of free tasks $L$ in which the tasks are sorted
        in the descending order of priority values. Select the first priority task $n_x$
        from $L$.
    5.2 Capture the advanced information of the P processors from lower-layer
        resource managers.
    5.3 Base on the information, examine the workload situation of each processor
        to find a processor $p_i$ to execute the task $n_x$ so that it can finish in the
        earliest time and tlevel($n_x$) does not increase. If there is no processors
        satisfy the condition, the task will be scheduled to a processor that the
        task can be completed in the earliest time.
    5.4 Schedule $n_x$ to the suitable processor and the suitable time slot by
        booking the required resources.
    5.5 Add pseudo edges between two independent tasks scheduled into the same
        cluster.
    5.6 Mark the task examined. Remove $n_x$ from UEG and put $n_x$ onto EG.
    5.7 Re-compute the priorities of the $n_x$'s successors tasks.
    End While
6.  If the scheduled application cannot be executed within the users pre-specified
    period, report the scheduling failure and release the booked resources then
    quit. Otherwise, report the schedule is complete.
```

**Algorithm 7.2. The CO-HDSC algorithm[*].**

### 7.3.2 The Contractor Oriented HMD Algorithm (CO-HMD)

For an application consists of $N$ tasks schedule to a processors set $P = \{p_1, p_2, \ldots, p_k\}$ that satisfies the physical resources requirement for execution. Let $T_{PS}$ and $T_{PE}$ denote the pre-specified start and end time of the application program and let $L$ compose of all tasks in the initial scheduling task graph,

1. Sort the speed of target processors in the descending order of speed and ordered from left to right. Hence, obtain ε from the slowest speed processor.
2. Calculate Extended relative mobility for all tasks (EMr) for all tasks.
3. Let $L$ include all tasks initially. Let $L'$ be the group of tasks in $L$ with minimum relative mobility. Let $n_i$ be a task in $L'$ that does not have any predecessors in $L'$. Capture the advanced information from lower-layer resource managers. Base on the information, examine the workload situation of each processor to use Fact 1, schedule $n_i$ on the first processor. If $n_i$ cannot be scheduled on the first processor, schedule it on the second processor and so on. If no processor can satisfy the condition, the task will be scheduled to the first processor that the task can be completed in the earliest finished time. Schedule $n_x$ to the suitable processor and the suitable time slot by booking the required resources.
4. When $n_i$ is scheduled on $p_m$, all the edges connecting $n_i$ and other tasks already scheduled to $p_m$ are changed to zero. If $n_i$ is scheduled before task $n_j$ on $p_m$, add an edge with zero cost from $n_j$ to $n_i$ in the graph. If $n_i$ is scheduled after node $n_j$ on $p_m$, add an edge with zero cost from $n_j$ to $n_i$ in the graph.
5. Recalculate EMr of the modified graph. Delete $n_i$ from $L$ and Repeat Step 2 until $L$ is empty.

[*] The bolded text identify the change due to the application of the Contractor Oriented approach

6. If the scheduled application cannot be executed within the users pre-specified period, report the scheduling failure and release the booked resources then quit. Otherwise, report the schedule is complete.

**Algorithm 7.3, THE CO-HMD algorithm.**

### 7.3.3 The Contractor Oriented HRMS Algorithm (CO-HRMS)

For an application consists of $N$ tasks schedule to a processors set $P=\{p_1,p_2,\ldots,p_k\}$ that satisfies the physical resources requirement to execute the application. Let $T_{PS}$ and $T_{PE}$ denote the pre-specified start and end time of an application program respectively, Let $L$ compose of all tasks in the scheduling task graph initially,

1. Sort the speed of target processors in the descending order of speed and ordered from left to right. Hence, obtain ε from the slowest speed processor.
2. Calculate Extended relative mobility for all tasks (EMr) for all tasks.
3. Let $L$ include all tasks initially. Let $L'$ be the group of tasks in $L$ with minimum relative mobility. Let $n_i$ be a task in $L'$ that does not have any predecessors in $L'$. Capture the advanced information from lower-layer resource managers. Based on the information, examine the workload situation of each processor to find a processor $p_m$ in the processors set $P$ in which $n_i$ would satisfy Condition H and the task can be completed in the earliest time (or the earliest finishing time is reached). If no processor can satisfy the condition, the task will be scheduled to the first processor that the task can be completed in the earliest finished time. Schedule $n_x$ to the suitable processor and the suitable time slot by booking is the required resources.
4. When $n_i$ is scheduled on $p_m$, all the edges connecting $n_i$ and other tasks already scheduled to $p_m$ are changed to zero. If $n_i$ is scheduled before task $n_j$ on $p_m$, add an edge with zero cost from $n_j$ to $n_i$ in the graph. If $n_i$ is scheduled after node $n_j$ on $p_m$, add an edge with zero cost from $n_j$ to $n_i$ in the graph.
5. Recalculate EMr of the modified graph. Delete $n_i$ from $L$ and Repeat Step 2 until $L$ is empty.
6. If the scheduled application cannot be executed within the users pre-specified period, report the scheduling fail and release the booked resources then quit. Otherwise, report the schedule is complete.

**Algorithm 7.4, The CO-HRMS algorithm.**

## 7.4 Properties of CO Based Algorithms

### 7.4.1 Resources acquisition

The scheduling decision made by the CO approach guarantee resources booking is successful. The task scheduling decision made by the CO approach is based on the current and scheduled workload situation of a processor. Therefore, the scheduler can schedule tasks to where it can achieve scheduling goals without any resource conflict. Therefore, resource booking is guaranteed.

Moreover, as compare with the SO and the RO approaches, the resource booking probability in the CO approach is much higher. It is because the CO approach considers the resource situation in any task scheduling time. The required resources for the produced schedule can ensure to be booked. Hence, the probability of resources booking is the best as compared with the SO and RO approaches.

### 7.4.2 Efficiency in resources utilization

There is no resources wastage introduced by the CO approach. This is because task scheduler can determine the exact amount of resources required by the schedule. Resources booked for the schedule should be fully utilized by the scheduled application. Besides, for those resources that are not booked by the applications can free to other applications. There is no resource booked be idle.

### 7.4.3 Performance in task scheduling

CO approach can lead task-scheduling algorithms adapting to dynamic change in workload situation. Since it schedules task in accordance with the advanced scheduling information, this reflects the dynamic changing workload situation of a system. Therefore, the CO approach is another form of dynamic scheduling approach. The major difference is that the classical dynamic scheduling approach is based on the workload estimation by task scheduler itself but the CO approach is based on the actual workload information captured by contractors in contractor layer.

The strengthen of the CO approach over dynamic scheduling can be summarized as follows:

- In the CO approach, workload situation is not required to maintain by task schedulers.
- In the CO approach, workload estimation is not required in task schedulers.
- In the CO approach, exhaustive rescheduling of tasks is not required in run time.

If the target processors set is loaded by other applications, the scheduling result of the CO-HDSC algorithm, the CO-HMD algorithm, the CO-HRMS algorithm would not same as the

HDSC algorithm, the HMD algorithm and the HRMS algorithm respectively. Otherwise, their properties are the same. As consequence to the study in Section 7.2.2, loaded processors would contain alias tasks. The task to processor assignment heuristics using in the CO-HDSC algorithm, the CO-HMD algorithm, the CO-HRMS algorithm are not valid. On the other hand, if the target processors set is offloaded, no alias task does exist. Therefore, the heuristics using in the CO-HDSC algorithm, the CO-HMD algorithm and the CO-HRMS algorithm are valid. The properties of the CO-HDSC algorithm, the CO-HMD algorithm and the CO-HRMS algorithm can be reduced to the HDSC algorithm, the HMD algorithm and the HRMS algorithm respectively. On the other hands, if the target processors set is loaded, the CO-HDSC algorithm, the CO-HMD algorithm and the CO-HRMS algorithm are not reducible to the HDSC algorithm, the HMD algorithm and the HRMS algorithm respectively. Otherwise, they are reducible.

### 7.4.4 Property on the processor table after scheduled

The schedules produced by the CO approach can book the scheduled resources spanned over the set of booking processors in which they have different percentages of processing power consumption. The CO approach encourages using advanced scheduling information together with task scheduling heuristic for making a task scheduling and resources booking decision. This is done by analyzing the current and scheduled resources as well as the amount of resources required by the scheduling task to find a suitable place to schedule the task on. Therefore, the amount required by the schedule depends upon the workload situation and the scheduling algorithm used. The percentage of processing power consumed by different tasks can be different. Moreover, the data dependence and the scheduled workload situation of the targeted processors are different. Thus, tasks in an application may be distributed among the target processor sets.

### 7.4.5 Inherence problems

Overhead would include on analysis the advanced scheduling information when a scheduling decision is made in scheduling time. It is because, the CO approach should maintain the up-to-date advanced scheduling information to make a precise scheduling decision. Therefore, there are overheads on analyzing the workload situation before making scheduling decisions. It demands high complexity searching. Overhead is imposed on making such decision.

Furthermore, The CO approach discourages the design of scheduling algorithms with backtracking and rescheduling. The proposed CO approach books the resources as long as a task is scheduled. The cost is high on "un-book" task in which they have already been booked. This prohibited the development of scheduling algorithm with backtracking or rescheduling. A possible solution for this problem is to book the resources in a batch. The cost on un-booking can be eliminated by booking resources in a batch after a freezing schedule (freeze schedule is occurred when all tasks in an application are scheduled) is produced.

### 7.4.6 Applicability

The proposed CO approach is suitable for scheduling applications with the following characteristics:

1. The workload situation of the execution environment is dynamically changing
2. Non-real time applications that can suffer delay when loaded of the execution environment are changing.

# Chapter 8: Experimental Studies

## Experiment 1 Performance between MD and RMS Algorithms

### E1.1 Objective

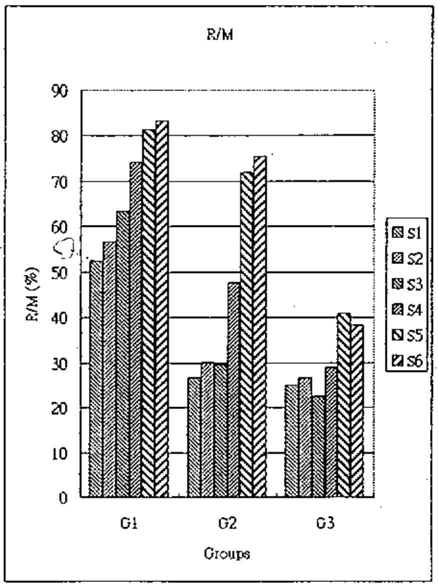
It is to study task-scheduling performance between the MD algorithm and the RMS algorithm in unbounded, bounded and minimum number of homogeneous processors.

### E1.2 Procedures

Please refer to Appendix A1 for detail description of the experiment procedure.

### E1.3 Results

#### *A) To schedule tasks into Set 1(|P|≥|N|)*



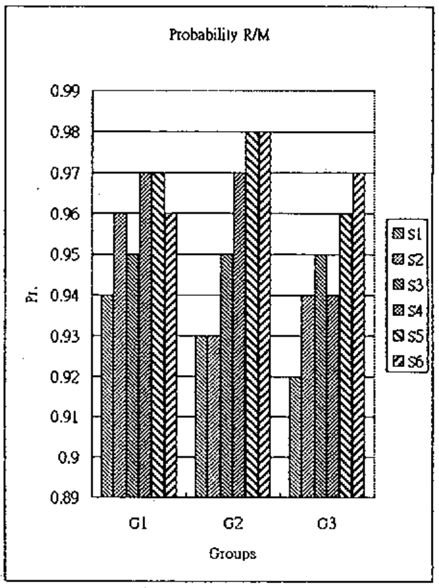
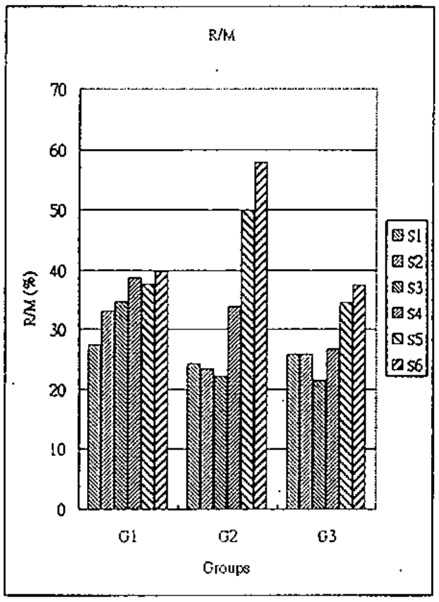


**Figure E1.1. Performance of task schedule into processor set 1(|P|≥|N|)**

## *B) To schedule tasks into Set 2 (|P| = |P$_{min}$|).*



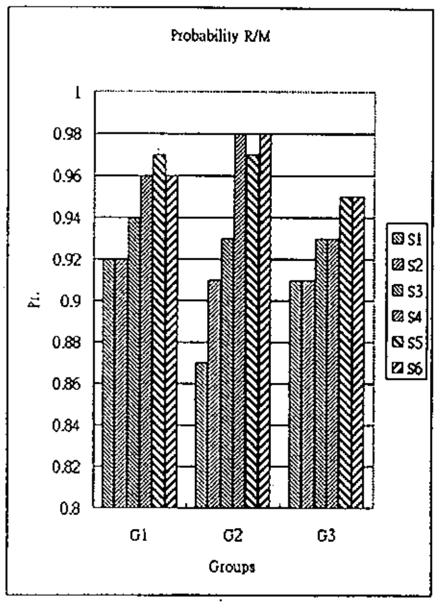
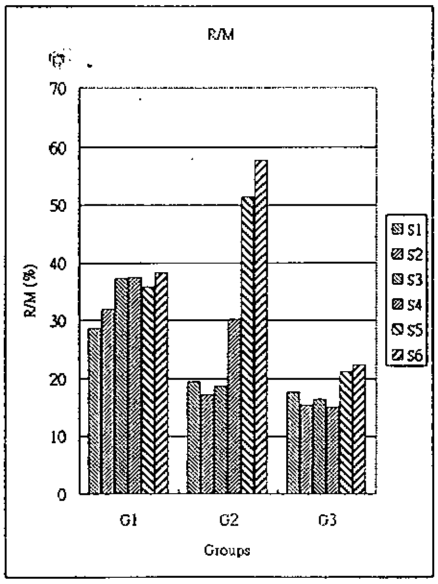


**Figure E1.2. To Performance of task schedule into processor set 2 (|P| = |P$_{min}$|)**

## *C) To schedule tasks into Set 3 (|N|>|P|>P$_{min}$)*



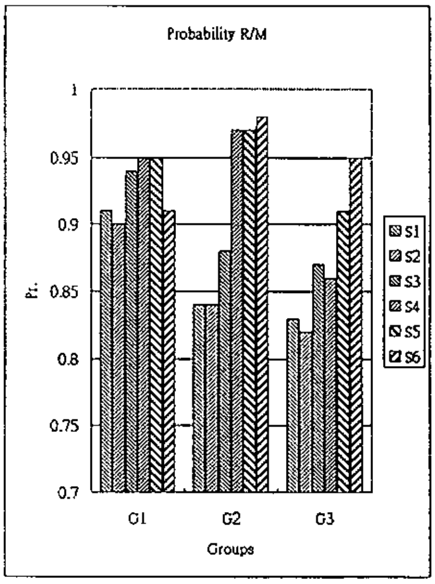


**Figure E1.3. Performance of task schedule into processor set 3 (|N|>|P|>P$_{min}$)**

### E1.4 Observations:

According to the experimental results shown in Figures E1.1 to E1.3, it is found that:

1. The proposed RMS algorithm can produce shorter parallel time than that of the MD algorithm in most cases (the probability is over 0.8): fine grain, medium grain and coarse grain task graphs.
2. Even the parallel time produced by the MD algorithm is shorter than that of the RMS algorithm, parallel time produced by the RMS is not longer than 20% of the MD algorithm
3. The R/M increases as the number of tasks and edge increases in the scheduling task graphs.
4. The R/M of the fine grain group task graphs is higher than median-grain group tasks graph and coarse grain group task graphs (refer to G1 to G3 of the Figures E1.1 to E1.3). In other words, the RMS algorithm has a more significant improvement than the MD algorithm in fine-grain task graphs. However, it becomes less significant than that in coarse-grain task graphs.
5. When the task graphs are scheduled into set 2 and set 3, the R/M is smaller than that scheduled into set 1.

### E1.5 Conclusions

From these results, more than 80 % of the task graphs scheduled by the RMS algorithm in the three sets of processors configuration are found to produce a positive R/M. Figure 3 shows that RMS is used to schedule task in the processor set using the MD algorithm ($|P|= P_{min}$). Over 80% of the produced scheduling solution will produce parallel time shorter than the MD algorithm in most situations. It is because the MD algorithm stops exploring suitable processor for executing once the first processor satisfied "Fact 1" in the MD algorithm. On the other hand, shortening the parallel time makes the RMS algorithm more aggressive than the other schedule. This can maximize the utilization of available processors and can increase the parallelism of the overall schedule. Consequently, scheduler designed using the R/M of RMS algorithm can perform better than that of the MD algorithm.

It is obviously that the R/M value in fine-grain groups is higher than the median grain group and the coarse grain group of task graphs. In the fine-grain task graph, over 90% of the schedule produced by the RMS algorithm have 28% shorter parallel time than the MD algorithm. At the granularity approaches to 10 (coarse grain), there are over 90% of the schedule produced by the RMS algorithm have over 18% parallel time shorter than that of the MD algorithm. This implies that the parallel time produced by the RMS algorithm is shorter than that of the MD algorithm in most fine-grain task graphs. Increasing of the granularity, the parallel time produced by the RMS algorithm becomes closer to but probably shorter than that of the MD algorithm produced.

Both MD and RMS algorithms make use of heuristic scheduling decision and they do not guarantee to produce an optimum solution. Therefore, parallel time produced by RMS is always longer than that of MD algorithm (shown in the graphs of probability again task graph groups of Figures E1.1 to E1.3). In the experiment a probability value (Pr) is used to count the number of cases, in which parallel time of the schedule produced by RMS is better than that of MD algorithm. According to the Observation 1, there are over 80% cases of parallel time produced by RMS being shorter than that of MD algorithm in three different processor configurations. This implies that the RMS algorithm is better than MD algorithm in most kinds of scheduling task graphs. The weakness of the RMS algorithm is that it cannot work out a minimum number of processors required for executing the task graph, while the MD algorithm can. However, the task scheduling performance of the RMS algorithm is always better than the MD algorithm. The RMS algorithm is more suitable than the MD algorithm when a shorter parallel time schedule needs to be produced.

## Experiment 2 Properties of HDSC Algorithm

### E2.1 Objective

It is to study the properties of the HDSC algorithm in scheduling task to the practical heterogeneous processor environments.

### E2.2 Procedures

Evaluating heterogeneous scheduling algorithms environment is a difficult problem because scheduling solutions are not only sensitive to the scheduling task graph, but also sensitive to the configuration of target processors. Hence, fixing one of these parameters and studying them case by case can do the evaluation. In this experiment, the performances and the properties of using the HDSC algorithm in practical applications and practical heterogeneous environments are going to be studied. An environment for solving the problems, which are described in Appendix A2 has to be established in the laboratory. Four sets of target processor (platform) are proposed in Appendix A3. This is found that the use of the HDSC algorithm in scheduling these applications to different platforms would result in schedules with different parallel time. Based on these results, the properties of the HDSC algorithm is going to be analyzed.

The scheduling result is measured by:

1. The produced Parallel Time (PT).
2. Parallel Time Improved ratio as compare with homogeneous processors configuration (PTI). This is calculated by:

$$PTI = \frac{(PT_{\text{homogeneous}} - PT_{\text{heterogeneous}})}{PT_{\text{homogeneous}}}$$

3. The distribution of the scheduled tasks inside the processors set.

### E2.3 Results

This experiment verifies the design and the implementation of the HDSC algorithm that can be used to schedule task in heterogeneous processors environment so that data dependencies are satisfied. The detailed descriptions of the produced schedules (K1 and K30) are attached in Appendix A2. The important information including parallel time of the produced schedule and the distribution of the scheduled tasks are captured and summarized in Table E2.2 to Table E2.5.

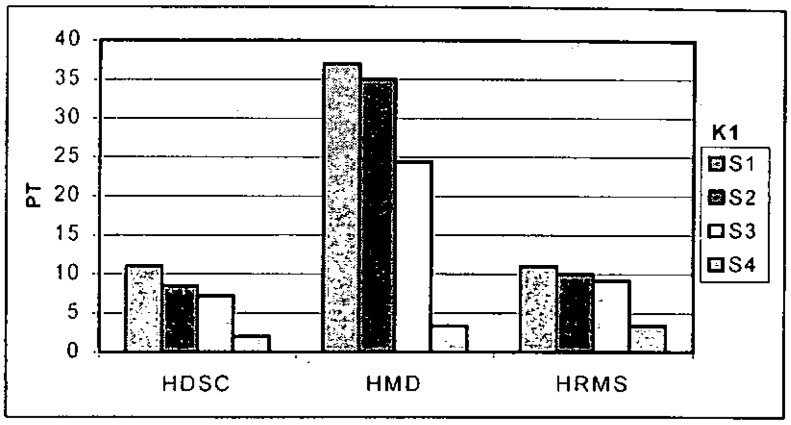


**Figure E2.1. Parallel time of executing K1 task graph using different scheduling algorithms**

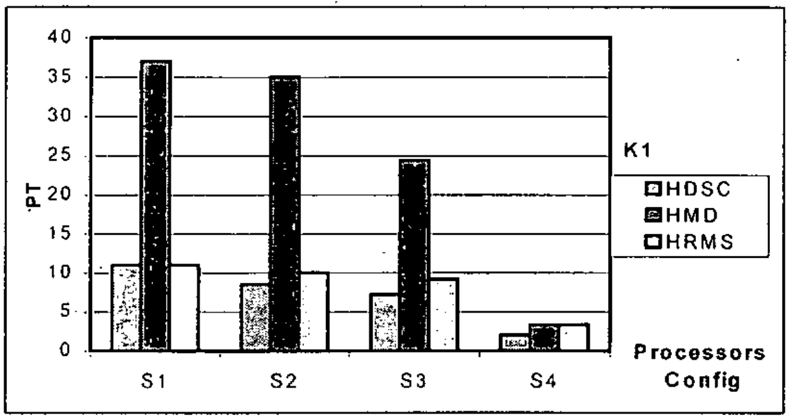


**Figure E2.2. Parallel time of executing K1 task graph using different processors configurations**

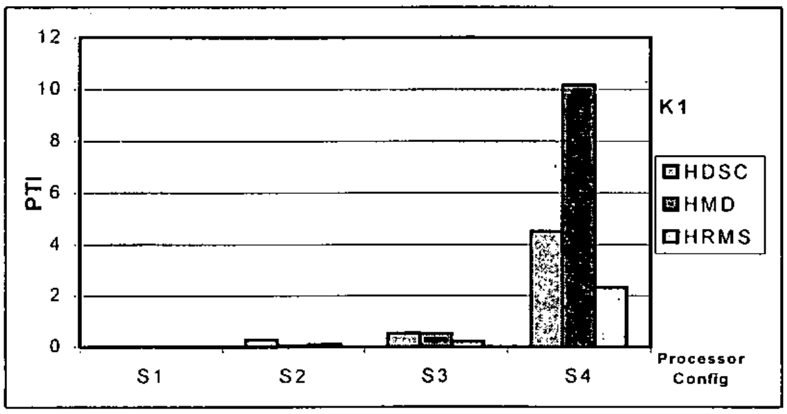


**Figure E2.3. Parallel time improvement ratio of executing K1 task graph using different processors configurations**

| | HDSC | HMD | HRMS |
|---|---|---|---|
| S1 | $p_0$-$p_{14}$ | $p_0$-$p_{14}$ | $p_0$-$p_{14}$ |
| S2 | $p_0$-$p_{11}$ | $p_0$-$p_{12}$ | $p_0$-$p_{12}$ |
| S3 | $p_0$-$p_8$ | $p_0$-$p_{12}$ | $p_0$-$p_{12}$ |
| S4 | $p_0$ | $p_0$-$p_1$ | $p_0$-$p_1$ |

**Table E2.1. Parallelism of schedule produced by different scheduling algorithm in K1 task graph**

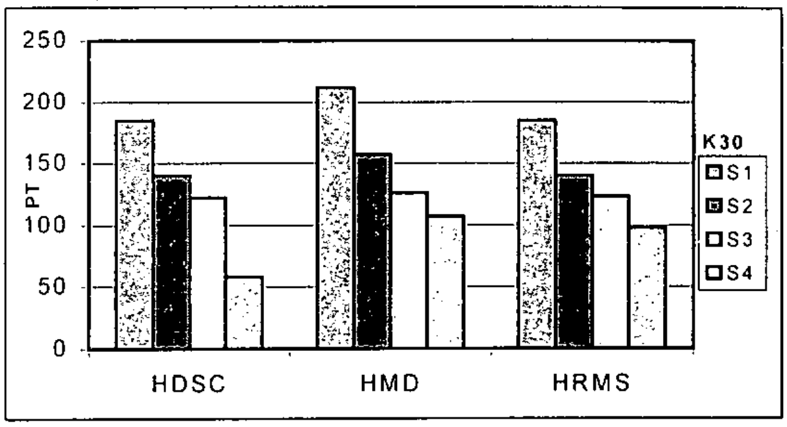


**Figure E2.4. Parallel time of executing K30 task graph using different scheduling algorithms**

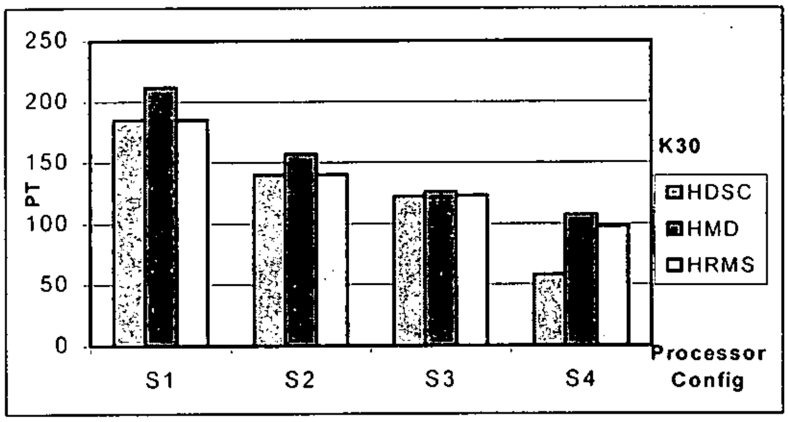


**Figure E2.5. Parallel time of executing K30 task graph using different processors configurations**

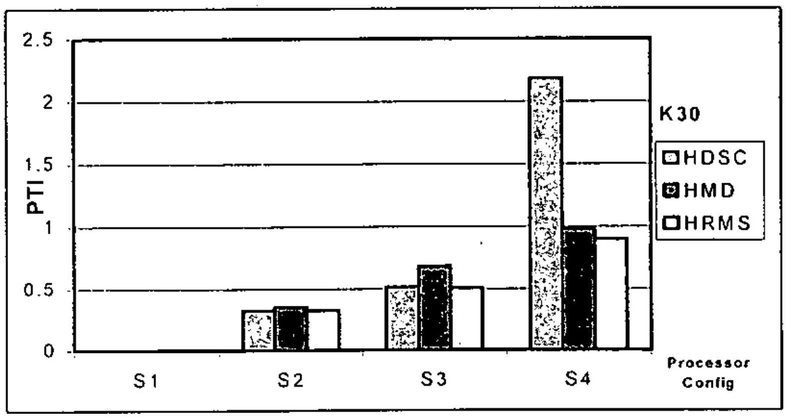


**Figure E2.6. Parallel time improvement ratio of executing K30 task graph using different processors configurations**

|  | HDSC | HMD | HRMS |
|---|---|---|---|
| S1 | $p_0$-$p_{15}$ | $p_0$-$p_{15}$ | $p_0$-$p_{15}$ |
| S2 | $p_0$-$p_{13}$ | $p_0$-$p_{14}$ | $p_0$-$p_{14}$ |
| S3 | $p_0$-$p_{11}$ | $p_0$-$p_{11}$ | $p_0$-$p_{11}$ |
| S4 | $p_0$-$p_1$ | $p_0$-$p_4$ | $p_0$-$p_4$ |

**Table E2.2. Parallelism of schedule produced by different scheduling algorithm in K30 task graph**

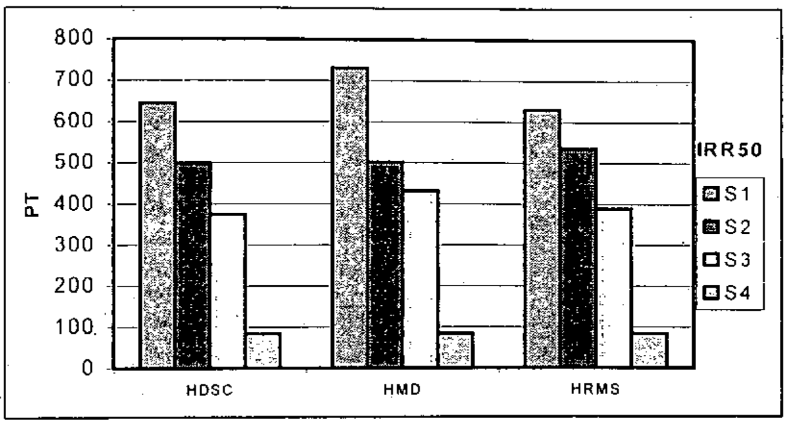


**Figure E2.7. Parallel time of executing IRR50 task graph using different scheduling algorithms**

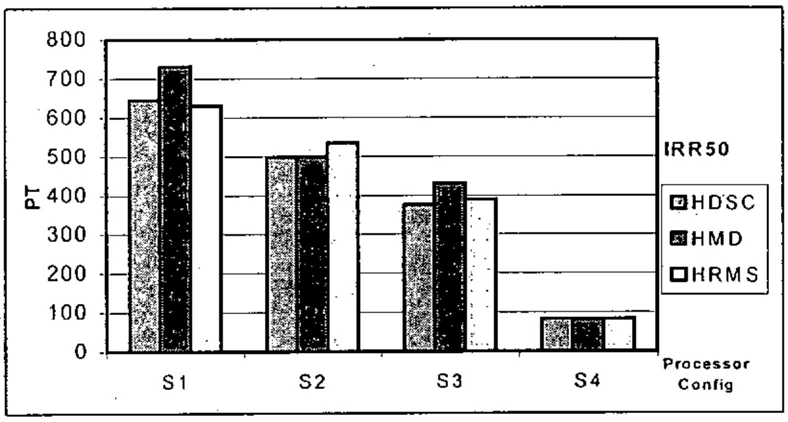


**Figure E2.8. Parallel time of executing IRR50 task graph using different processors configurations**

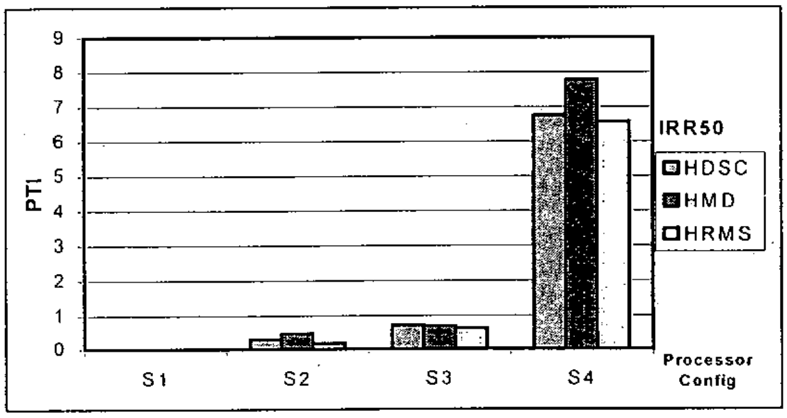


**Figure E2.9. Parallel time improvement ratio of executing IRR50 task graph using different processors configurations**

| | HDSC | HMD | HRMS |
|---|---|---|---|
| S1 | $P_0$-$P_4$ | $P_0$-$P_4$ | $P_0$-$P_4$ |
| S2 | $P_0$-$P_5$ | $P_0$-$P_3$ | $P_0$-$P_3$ |
| S3 | $P_0$-$P_2$ | $P_0$-$P_2$ | $P_0$-$P_2$ |
| S4 | $P_0$ | $P_0$ | $P_0$ |

**Table E2.3. Parallelism of schedule produced by different scheduling algorithm in IRR50 task graph**

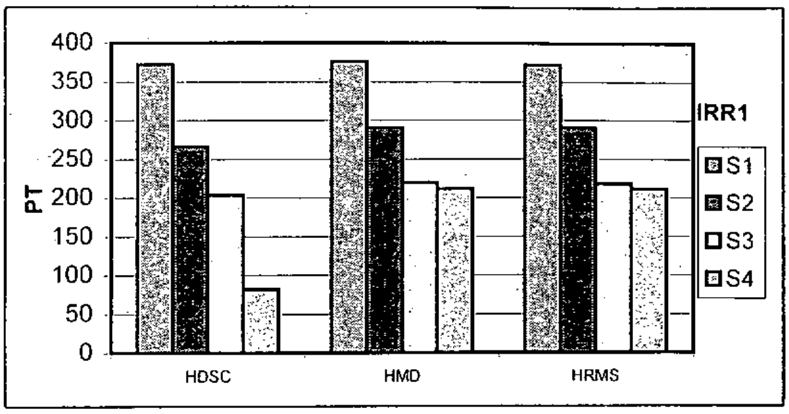


**Figure E2.10. Parallel time of executing IRR1 task graph using different scheduling algorithms**

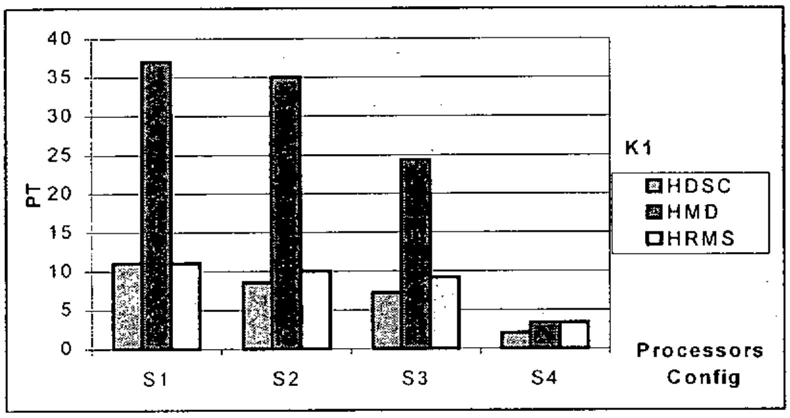


**Figure E2.11. Parallel time of executing IRR1 task graph using different processors configurations**

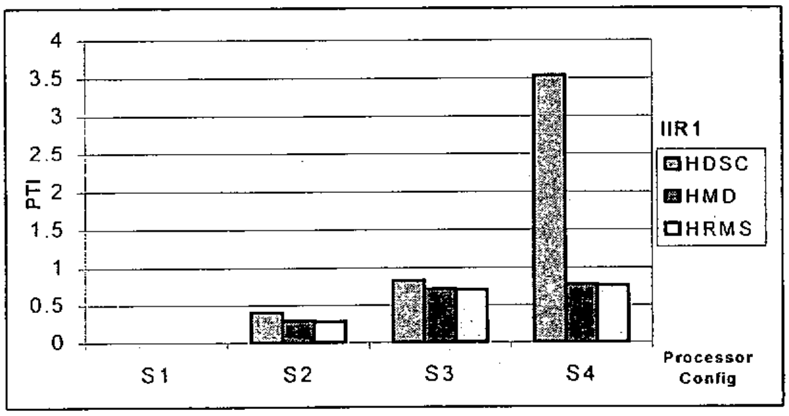


**Figure E2.12. Parallel time improvement ratio of executing IRR1 task graph using different processors configurations**

| | HDSC | HMD | HRMS |
|---|---|---|---|
| S1 | $p_0$-$p_6$ | $p_0$-$p_7$ | $p_0$-$p_7$ |
| S2 | $p_0$-$p_5$ | $p_0$-$p_7$ | $p_0$-$p_7$ |
| S3 | $p_0$-$p_6$ | $p_0$-$p_8$ | $p_0$-$p_7$ |
| S4 | $p_0$-$p_4$ | $p_0$-$p_4$ | $p_0$-$p_4$ |

**Table E2.4. Parallelism of schedule produced by different scheduling algorithm in IRR1 task graph**

## E2.4 Observations

According to the experimental results, it is observed that:

1. According to Table E2.1 to E2.4, the algorithms tend to schedule tasks into high performance processors for target processors set being configured more deviated from homogeneity (say S2, S3, S4). As a result, parallelism is decreased. For example using the HDSC algorithm, tasks in K1 will be scheduled dispersely from $p_0$ to $p_{14}$ when they are scheduled to a processor set S1 (refer to Table E2.1 and Appendix B). However, if processor $p_0$ to $p_1$ are replaced with those of twice the performance (the processor set S2), most of the tasks will be scheduled to $p_0$ and $p_1$. If a super-computer is being used (16 time the original performance as shown in processor configuration S4), all tasks will be scheduled to those highest speed processors.
2. According to Figure E2.2, E2.5, E2.8, E2.11, E2.3, E2.6, E2.9 and E2.12 Parallel time decreases as PTI increases with the target processors set being configured more deviated from homogeneity. For example, while using the HDSC algorithm to schedule K1, PTI of S4 becomes higher than that of S3, PTI of S3 is higher than that of S2 respectively.
3. If application K1 and IRR50 (fine grain applications) are scheduled to processor set S4, the application is effectively executed in a single computer. The algorithm can order tasks execution so that data dependence among the tasks can be maintained.
4. If the K30 application and the IRR1 application (coarse grain applications) are scheduled to processor set S4, it does not schedule every task in the applications to the high performance processor. Some of them will be scheduled to other processors to achieve a shorter parallel time. Consider an application K30 to be scheduled by the HDSC algorithm. It executes serially in $p_0$ of S4 and completed in 60-time unit. However, the parallel time becomes 58.12 when tasks in the application are scheduled between $p_0$ and $p_1$ (Figure E2.1 and Appendix B). Similarly, if IRR1 were executed serially in $p_0$ of S4, it becomes 83.25-time unit. However, the parallel time would be 82 when tasks in the application are scheduled from $p_0$ to $p_4$.
5. Referring to Table E2.1 to Table E2.4 and schedules shown in Appendix B, the fork and joint type of task graphs (K1 and K30) would be scheduled by HDSC, HMD and HRMS algorithms schedule in a higher degree of parallelism than irregular type of task graphs (IRR1 and IRR30).
6. According to Figure E2.1, E2.4, E2.7 and E2.10 the HDSC algorithm can produce shorter parallel time schedule than the HMD algorithm and the HRMS algorithm in most target

processors configurations and application task graphs. Besides, the HRMS algorithm can also produce shorter parallel time schedule than the HMD algorithm.

## E2.5 Conclusions

1. This experiment verifies the performance of the HDSC, HMD and HRMS algorithm that can be used to schedule task in homogeneous and heterogeneous processors environment so that data dependencies are satisfied and parallel time can also be reduced.
2. The proposed algorithms can be used to schedule fine grain as well as coarse grain applications of regular or irregular structures.
3. According to the running trace and the schedules produced by the algorithms shown in Appendix B, the algorithm prefers to schedule task into a processor that can be completed as soon as possible.
4. Observation 1 to 3 implies that the proposed algorithm favor in scheduling task to a processor set that is more deviated from homogeneous configuration.
5. Observation 4 and 5 indicates that the property of the proposed algorithms more favorable for coarse-grain applications. This is because the bottleneck for scheduling a coarse grain task graph is the task cost. According to conclusion 3 and 4, the designed specification of the HLS technique, the proposed algorithms tend to schedule tasks to use higher-speed processors first. This effectively reduces the task costs. Parallel time of an application can be directly reduced by the cost reduction. Besides, the communication cost of a coarse grain task graph is low (this should be much lower than task cost), they will not be increased by scheduling tasks to the high-speed processors. The overhead on running in parallel tasks is much lower than that in a fine grain task graphs. Therefore, the coarse grain task graphs can take maximum advantage out of the HDSC algorithm.
6. The use of the HDSC algorithm follows the argument of the MAX-MIN problem [Kruatrachue88]. Since the scheduling strategy of the HDSC algorithm is to trace a processor that can complete the scheduling task as soon as possible. A processor with higher processing power implies the task, which can be completed much earlier. Therefore, most of the tasks will be scheduled to it (this is supported by Observation 1 to conclusion 3). As following the argument of the MAX-MIN problem [Kruatrachue88], the decrease in parallelism would result in communication overhead decreasing. In this case, the communication cost becomes the dominant factor for decreasing parallelism and increasing parallel time. Therefore, parallel time will become shorter when high performance processors are installed. This encourages scheduling more tasks to the high performance processors. Consequently, the parallelism is decreased.

7. Referring a laboratory in setting up 16 processors to solve the applications K1, K30, IRR50, IRR1, a possible configuration for achieving parallel time reduction (as compared with serial execution) is by configuring a set of processor set homogeneously. However, the proposed HDSC, HMD and HRMS algorithm can enhance the parallel time by upgrading one or few processors to higher performance processors. It is a rule of thumb that program execution time can be greatly improved by using a outstanding processors such as super-computers to solve the problems i.e. the configuration of processors set S4.

## Experiment 3 Resources Booking Characteristic for Task of Different Sizes

### E3.1 Objective

It is to study the probability of the booking resources with different task sizes in a contracted processor with different workload situations.

### E3.2 Procedures

To evaluate the behavior of the resources booking in a contracted computer environment, the experiment is to study the probability of the successful resources booking in randomly generated workload situation of the contracted computer. Firstly, the workload situations of contracted computers is randomly generated and classified in ten groups ranging from 0% to 100% (with step size in 10%) as shown in Figure E3.1a to E3.1k. Then, 1000 samples tasks are generated randomly with their size ranging from 0 to 254 unit (with step size of 1 unit) are generated and the processing power ranging from 0% to 100% (with step size of 10%) is consumed. The randomly generate task will be presented for resources book in the contracted computer. It is not possible to avoid any booking request being collision with the inherence bookings, the booking request may either be succeeded or failed. The booking procedure has to repeat until the probability of the successful booking becomes saturated (three significant Figures) in each sample.

### E3.3 Results

Figures E 3.1a to 3.1 k are the results of the experiment with the probability of a successful booking against task size.

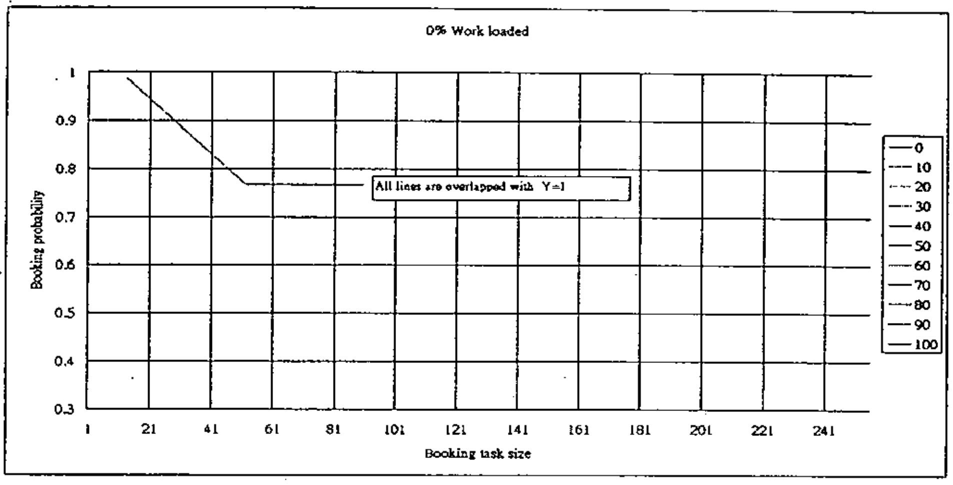


**Figure E3.1a. 0% work loaded**

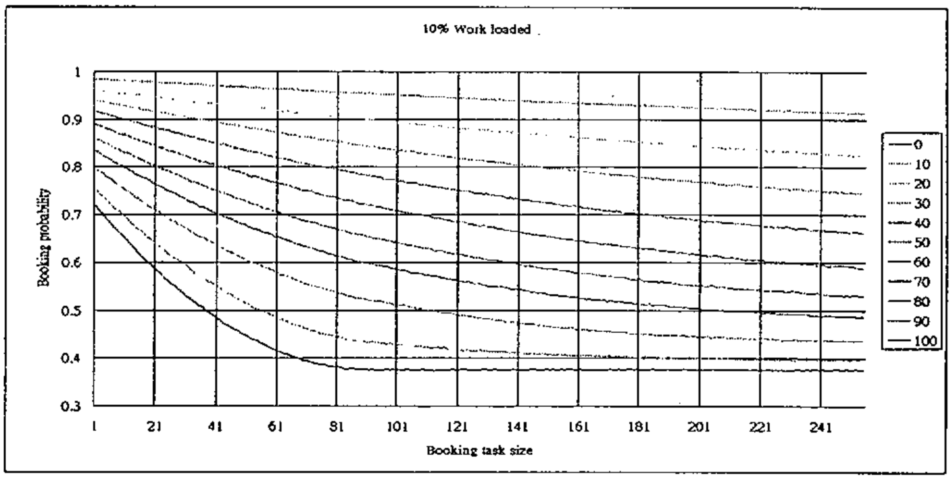


**Figure E3.1b. 10% work loaded**

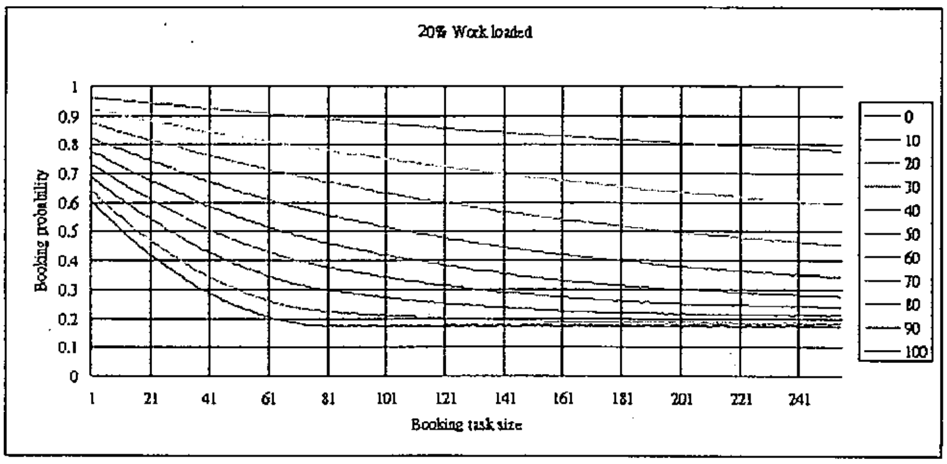


**Figure E3.1c. 20% work loaded**

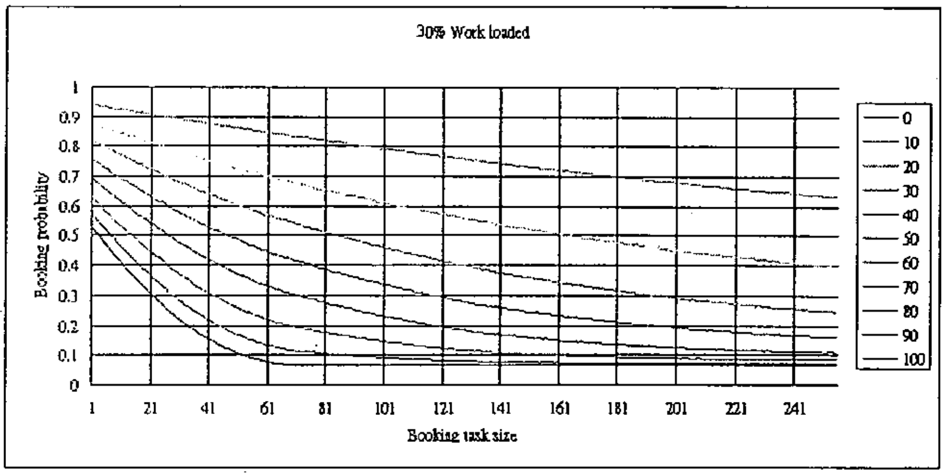


Figure E3.1d. 30% work loaded

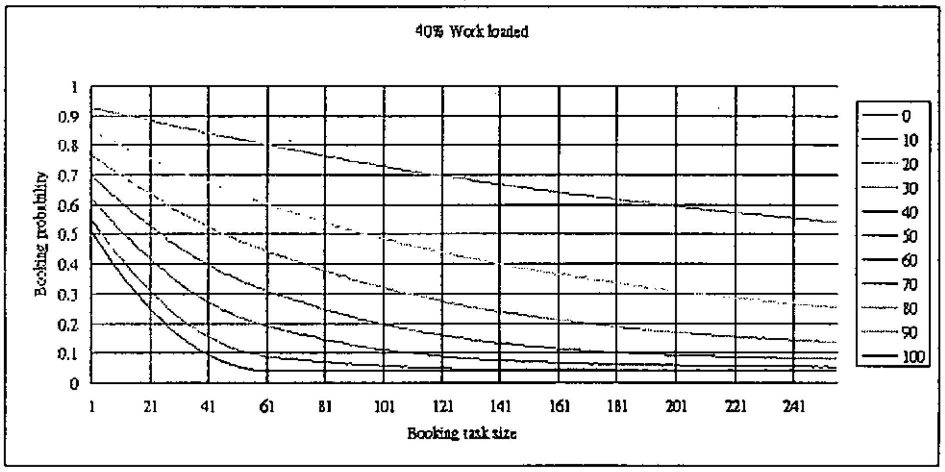


Figure E3.1e. 40% work loaded

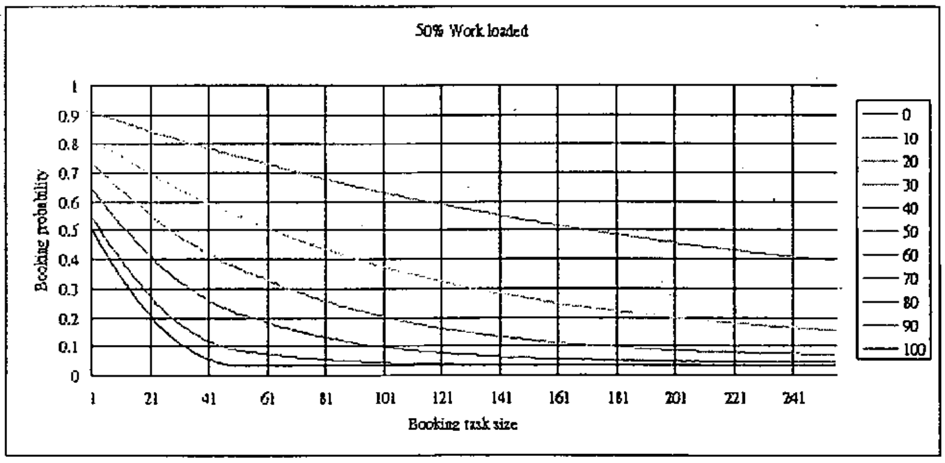


Figure E3.1f. 50% work loaded

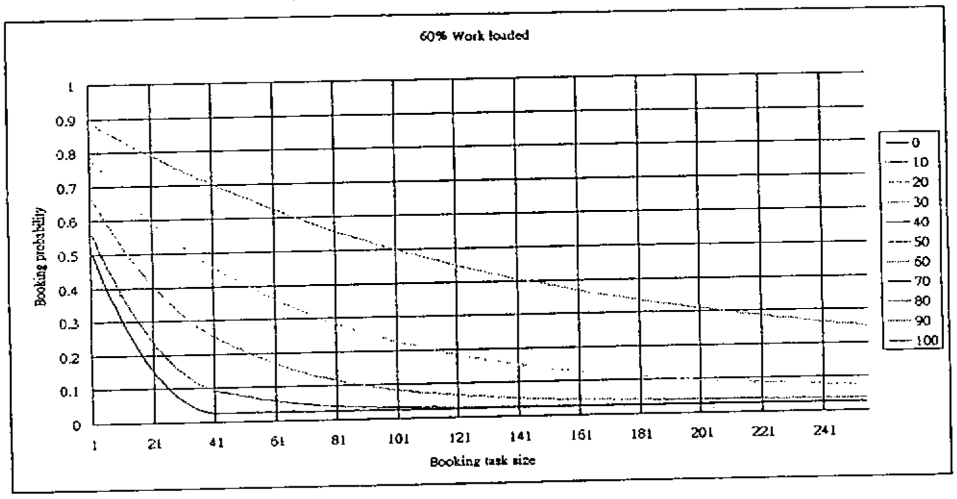


Figure E3.1g. 60% work loaded

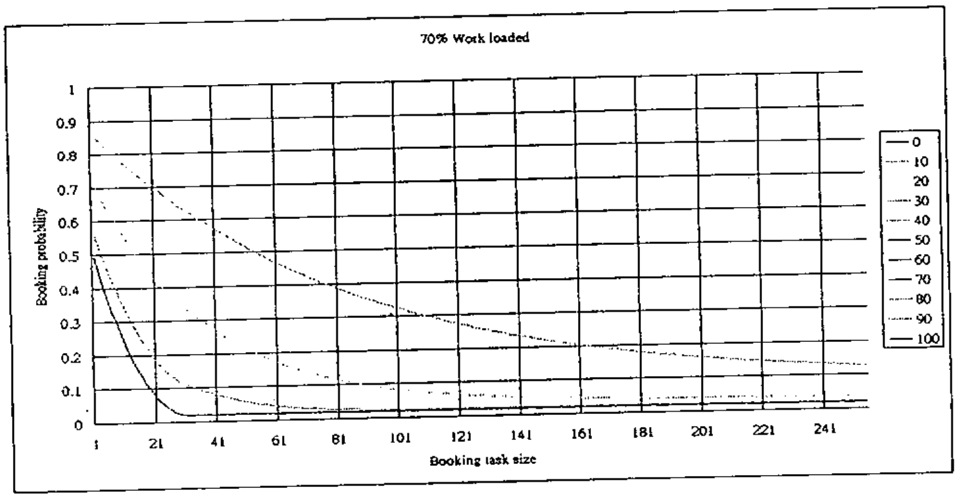


Figure E3.1h. 70% work loaded

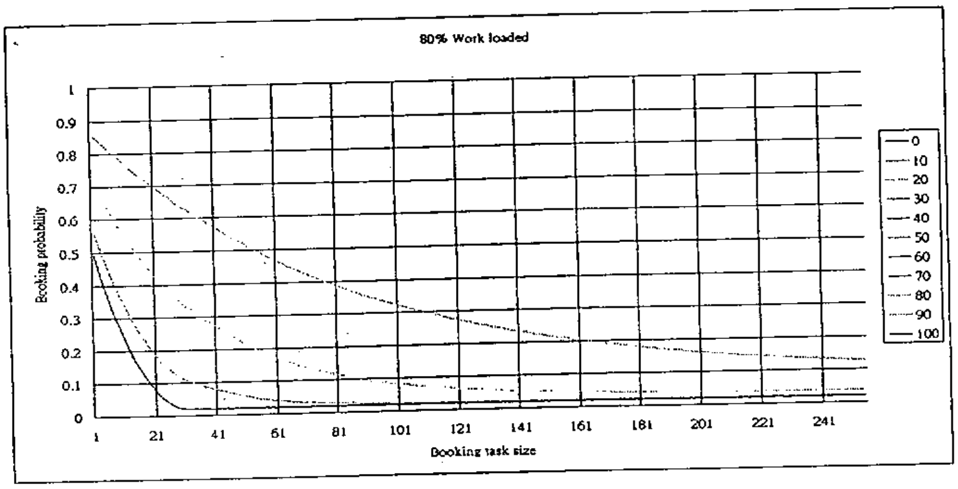


Figure E3.1i. 80% work loaded

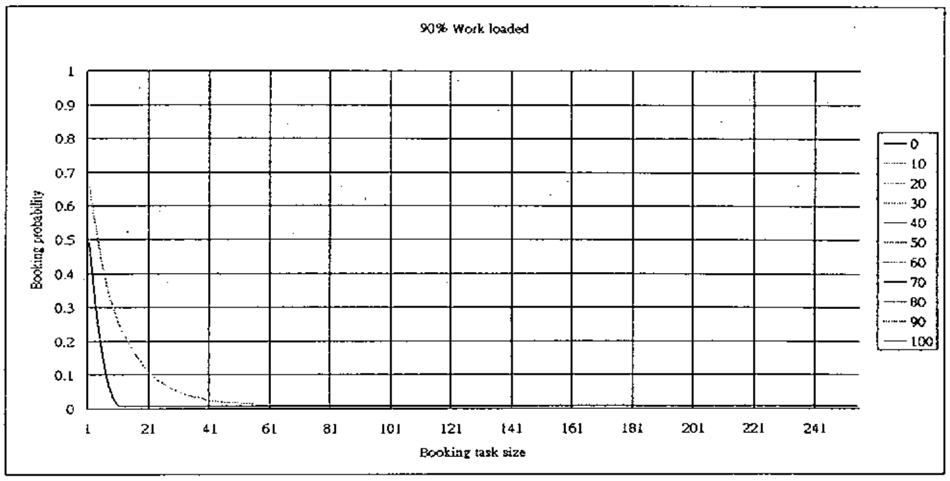


Figure E3.1j. 90% work loaded

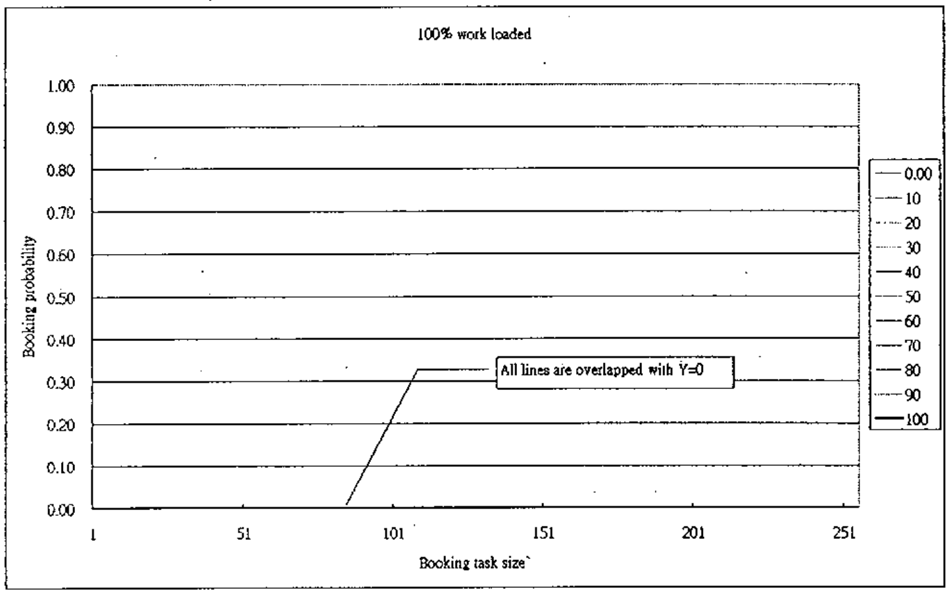


Figure E3.1k. 100% work loaded

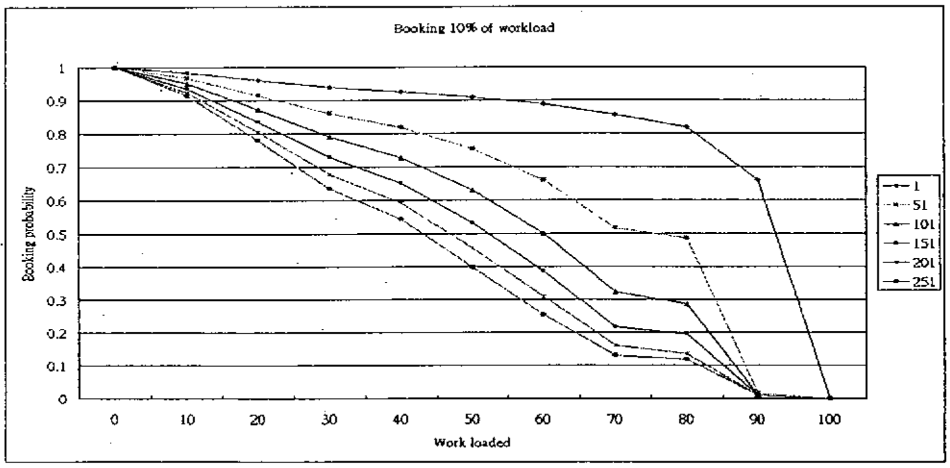


Figure E3.2. A graph of booking probability for requesting 10 % from pre-booked workload

## E3.4 Observations

1. When a computer is offloaded, the probability of the successful booking for any task size is 1.
2. When a computer is fully loaded, the probability of the successful booking for any task size is 0.
3. The probability of the successful booking exponentially decreases as the size of booking increases.
4. The probability of the successful booking decreases as the percentage of processing power required for a booking increases. (This is further illustrated in Figure E3.1k)

## E3.5 Conclusions

The experiment concludes that the probability of the booking resources depends on three factors: (1) The current work load situation; (2) The size of the booking request and; (3) The percentage of resources consumed within the booking period.

The characteristic of the booking resources probability is decreasing in the conditions that either the current work load situation of the processors becomes heavy duty, or the booking tasks size becomes large, or the percentages of resources being booked in the booking region increases. In other words, it is possible to book the resources in higher probability by reducing the task size or the resources being booked. With this exception, a task can also be booked when the resources are in a lightly loaded machine with higher booking probability.

Based on the result, it is found that a task with smaller size can obtain higher resources booking probability when the workload situation and the percentage of processing power consumed by the booking are constant. For example, consider that a task $T_A$ costs 1 unit and requires 90% of the processing power during the interval. The probability of a successful booking in 10% workload situation is 0.75 (shown in Figure E3.1b). However, if another task $T_B$ costs 24 units and requires 90% of the processing power, the probability will decrease to 0.64. The 23 units increased in task cost will decrease the probability by 0.1.

Consider that the task $T_B$ can further be equally decomposed to 24 sub-tasks ($T_{B1}$, $T_{B2}$, ... $T_{B24}$) for which the booking resource will be done individually, since the multiple bookings of the same machine are mutually exclusive events. For simplicity, assume those sub-tasks will book their resources from an individual machine. Thus, the probability of successfully booking of all the sub-tasks is $0.75^{24}=0.001$. The probability will be much lower when these tasks of the booking resources are from the same machine. This concludes that the decomposition of a large task into small one decreases the probability of booking those required resources successfully. Based on this argument, the SO approaches suffers the declination in the probability of successful resources booking when the number of the tasks increases and the amount of resources being booked keeps constant. This supports the probability 4.1 given in Chapter 5 and it motivates the development of the RO approach in Chapter 6 for the task scheduling under the contractual computing paradigm.

## Experiment 4 Properties of SO Approach

### E4.1 Objective

It is to investigate the use of Scheduler Oriented (SO) approach for tasks scheduling in the sets of processors with different workload situations.

### E4.2 Procedures

The proposed Scheduling-First approach makes use of the existing task scheduling algorithm as its core. The SO-HDSC, the SO-HMD and the SO-HRMS algorithm are the modified version of HDSC, HMD and HRMS respectively. Firstly, this basic core scheduling mechanism to a set of pre-assumed dedicated processors (a set of offloaded processor) in a homogeneous configuration schedules an application. Then the resources required are booked in accordance with the produced schedule. In which the booked resources are guaranteed for that application, the task scheduling performance of the scheduled application will be the same as those claimed by the corresponding non-SO scheduling algorithm being used.

The major factor affecting performance in the SO approach is the rate of successfully booking of the required resources for the produced schedules. In this experiment the way the booking rate is affecting the complexity of the application and the workload situation of the processors being booked. Two typical applications (represented as task graphs) with different organizations and guaranties are selected and presented in Appendix A2. These applications are then scheduled to a set of sixteen homogeneous processors in which workload is randomly generated (this is described in Appendix A4). The chance of successful booking of resources is recorded. In this control experiment being conducted to study the percentage of the parallel time as the application is executed concurrently with the existing workload, the corresponding results are presented in E4.1b, E4.2b, E4.3b. Based on the results, the properties of the Schedulers Oriented approach are extracted.

The experimental results are measured by:

1. According to the characteristic of the SO approach, the resources required for the scheduled application can be booked. The application can be executed promptly (refer to Section 5.4.4 of Chapter 5 and the contractual computing paradigm stated in Chapter 3). The scheduling performance is exactly the performance of the core scheduling algorithm used (refer to

Section 5.4.4 in Chapter 5). The primitive factor for the Scheduler Oriented approach is to book the resources for a produced schedule. Therefore, the total numbers of the successful booking resources required for the scheduled application are counted. This is formulated as:

$C$ = Total number of the successful booking resources for an application.

2. In the control experiment, the percentage increased in Parallel Time for running the applications concurrently in 100 set of loaded processor set is measured by taking their average.

$$PTI = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\overline{PT_i} - PT_i}{PT_i}\right), \text{where } N = 100$$

Where $\overline{PT_i}$ and $PT_i$ is the parallel time of running application concurrently with loaded processor and parallel time for executing the application in a offload processors set of the i$^{th}$ sample of a workload group (WL) respectively.

## E4.3 Results

Figure E4.2a, E4.3a & E4.4a show the bar charts of the successful booking count ($C$) against the workload of the target processor sets whereas the SO-HDSC, the SO-HMD and the SO-HRMS algorithm are used to schedule the application K1, K30, IRR1 and IRR50. Figure E4.1b, E6.2b & E4.3b shows the bar charts of PTI against the workload of the target processor sets whereas the SO-HDSC, the SO-HMD and the SO-HRMS algorithm are used to schedule the application K1, K30, IRR1 and IRR50.

### *A) The SO-HDSC algorithm*

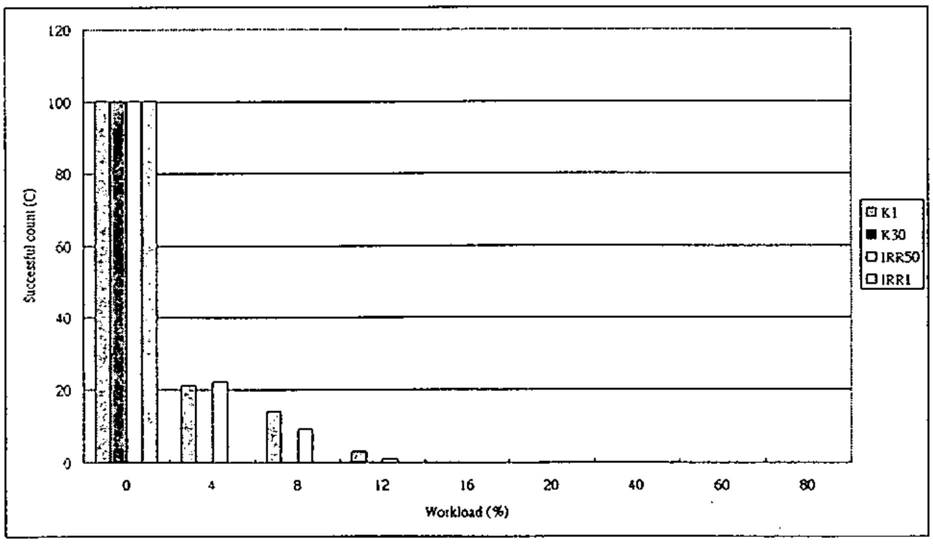


**Figure E4.1a. Bar charts of successful resources booking count against workload.**

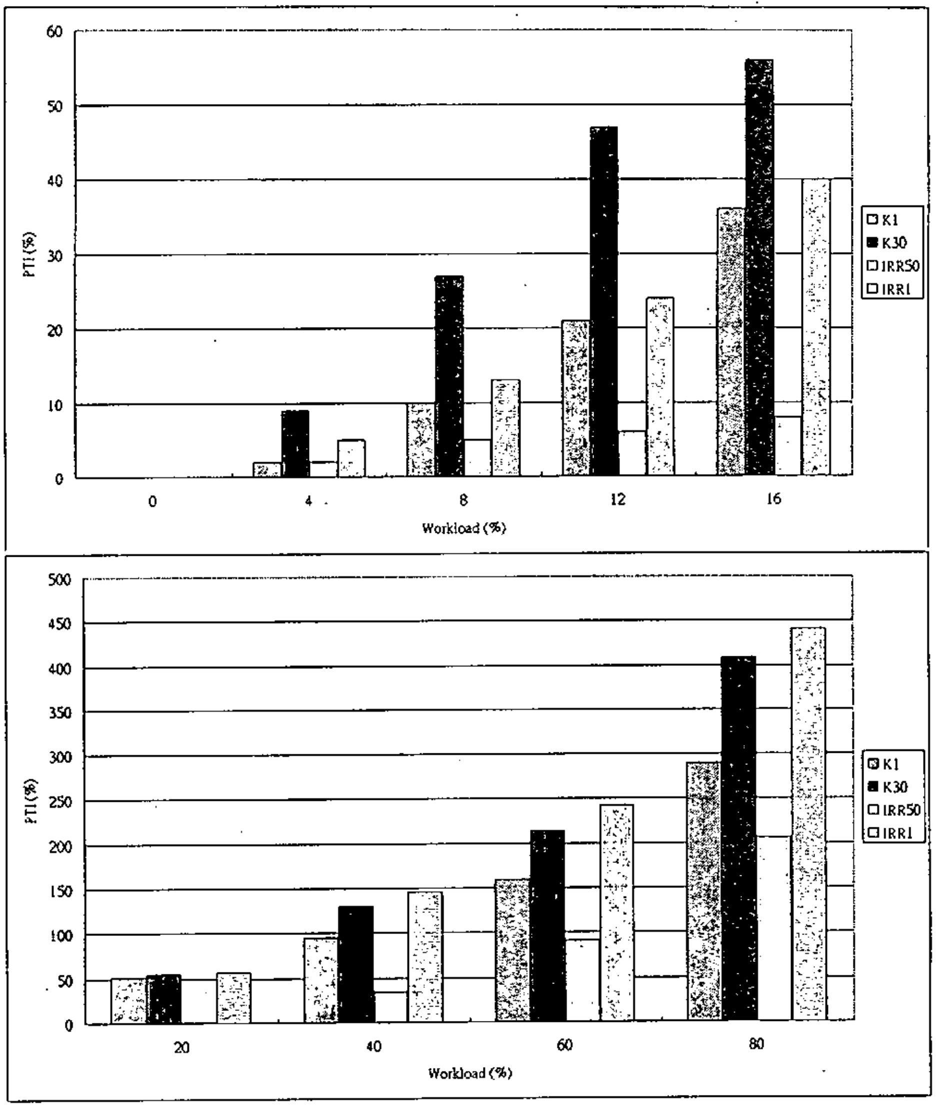


**Figure E4.1b. Bar chart of average parallel time increases concurrent execution with existing workload.**

## *B) The SO-HMD algorithm*

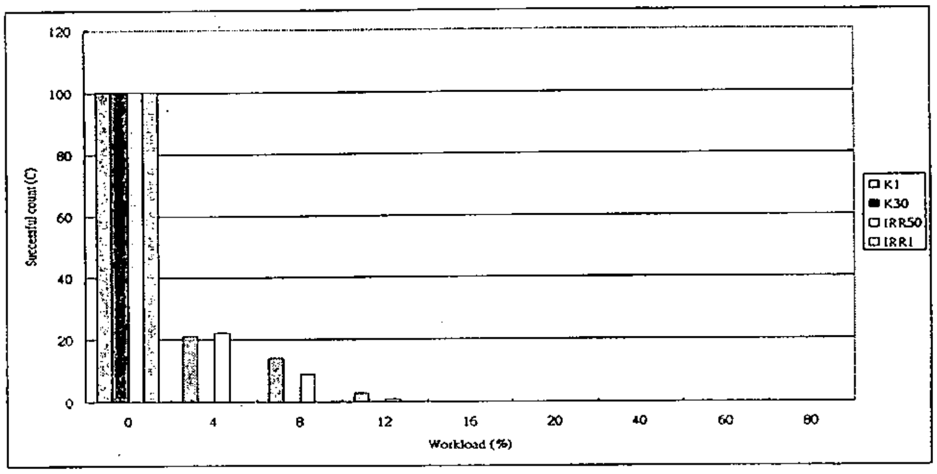


**Figure E4.2a. Bar charts of successful resources booking count against workload.**

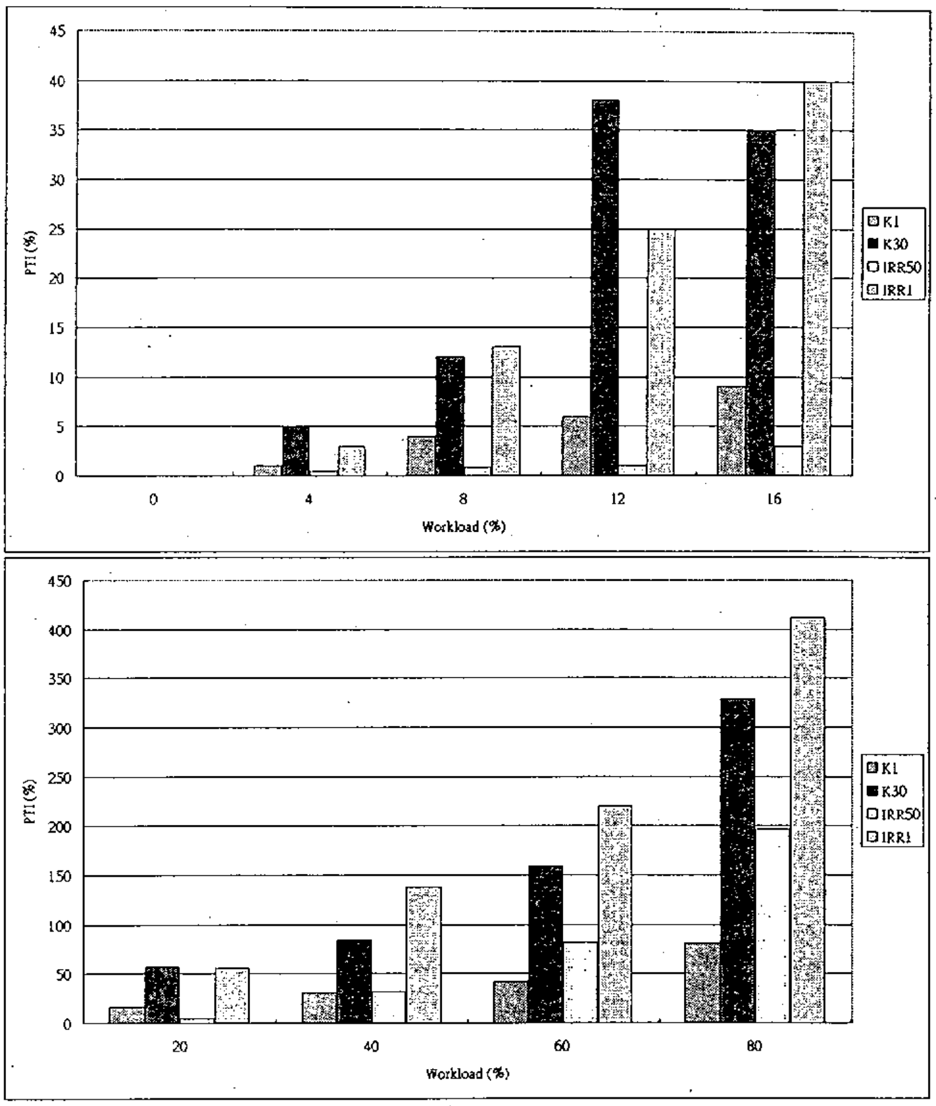


**Figure E4.2b. Bar chart of average parallel time increases as concurrent execution with existing workload.**

## *C) The SO-HRMS algorithm*

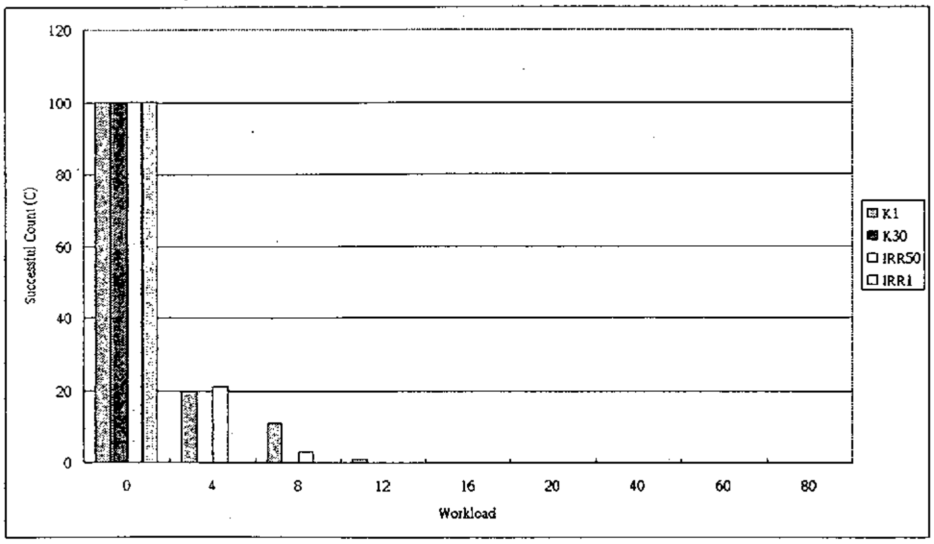


**Figure E4.3a. Bar charts of successful resources booking count against workload.**

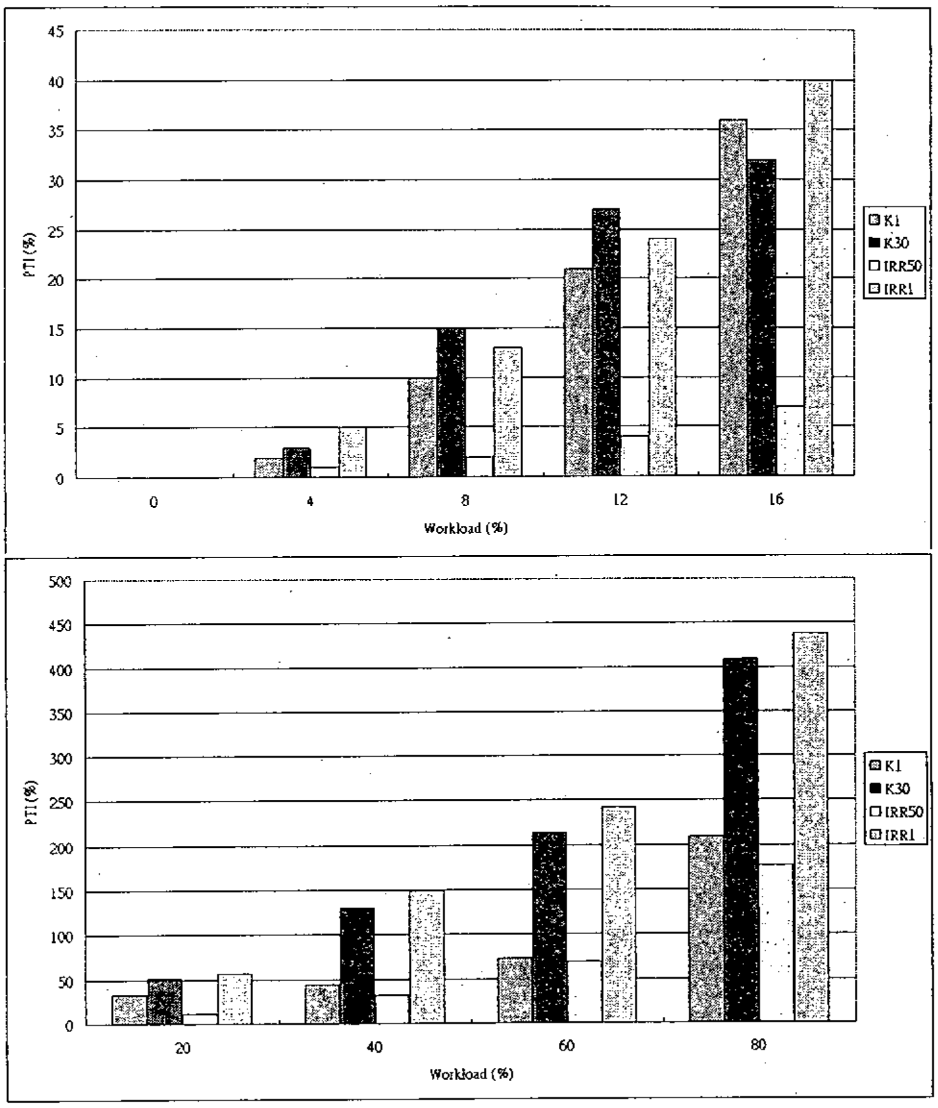


**Figure E4.3b. Bar chart of average parallel time increases as concurrent execution with existing workload**

## E4.4 Observation

### Observations from Figure E4.1a, Figure E4.2a and Figure E4.3a

A1. This is observed that booking resources from an offload processor set is always successful.

A2. The workload of the processors set increases, the chance of successful booking decreases. For example, when the workload is 4%, the chance of successful booking resources for application K1 is less than 30 counts (refer to Figure 6.1a), the required resources for both application K30 and application IRR1 cannot be booked.

A3. The rate of the successful booking in application K1 is higher than that in the application IRR1 under 4%, 8% and 12% workload situations.

A4. The rate of the successful booking for all the application is zero, when the workload situation is higher than 20%.

**Observation from Figure E4.1b, Figure E4.2b and Figure E4.3b:**

B1. The parallel time of running applications (K1, K30, IRR1 and IRR50) in an offloaded processor set is not changed.

B2. Executing applications (K1, K30, IRR1 and IRR50) would increase the parallel time concurrently in a set of loaded processors.

B3. The average percentage increased in parallel time (PTI) would increase as running applications (K1, K30, IRR1 and IRR50) concurrently in a set of higher workload processors.

B4. The average percentage increased in parallel time (PTI) for application K30 is higher than that of application K1 in using the three tasks scheduling algorithms.

B5. The average percentage increased in parallel time (PTI) for application IRR1 is higher than that of application IRR50.

## E4.4 Conclusions

The observations A1 to A4 can be explained by the fact that the booking resource for a scheduled application is a random process. As illustrated in Experiment 3, the successful count depends upon the required processing power for the booking region, the total number of the tasks in an application and the workload situations of the booking processors. According to the specifications of the scheduling applications, the application K1 and the application IRR50 are classified as the fine-grain application in which their task costs are much smaller than the communication cost. Therefore, the successful rate of the booking resources for the application

K1 in a set of processor with lower workload situation (such as 4% and 8% workload) is higher than the coarse grain version of the application K30. This argument is also applicable to book the resources for applications that have irregular structures. The reason is that the successful booking count of the application K1 is much higher than that of the IRR50. Besides, for the processor set with the higher workload (workload > 16%), the successful count is closed to zero. This can be explained by the results shown in Experiment 3 that the chance for the booking resources for the applications in relative high load situation tends to zero. This concludes that the SO approach is only feasible for the application in which are being executed in the sets of processors that is in low workload situations.

The Observation B1 is obvious that running an application in an offloaded processor set is equivalent to running an application in a dedicated environment. However, when parts of resources are consumed by other applications (the processors set is loaded), the scheduled application has to run the scheduled application concurrently with the existing workload. In Figure 4.1b, Figure 4.2b and Figure 4.3b show the percentage increment of parallel time when an application is scheduled by the proposed scheduling algorithms. It comes out with the Observation B2 supporting the fact that while tasks are being executed concurrently, the scheduled application would suffer execution delay.

According to these Figures, Observation A3 further points out that the PTI of the scheduled application increases as the workload increases. This is because the concurrency of running an application would be increased by increasing the workload of processors set. Since the Scheduler Oriented approach schedules the task with regardless of the workload situation, the concurrently would exist with regardless of the types of task scheduling algorithm being used.

Finally, the Observation B5 point out that PTI for application K30 is higher than that of application K1 and the application IRR1 is higher than that of application IRR50. According to the structural characteristic of the application K1 and the application IRR50, they are generalized as fine-grain task graph. Besides, the application K30 and the application IRR1 are classified as coarse-grain task graph. Since the task cost in fine-grain task graphs would much less than the communication cost. On the other hand, the task cost in coarse-grain task graphs would much higher than the communication cost. Therefore, delay due to concurrent execution of the tasks in the coarse-grain task type of task graph would higher than that of the fine-grain type of task graphs. This argument explains the Observation B5. Based on this result, it is found that a coarse-grain application may suffer from the higher percentage of execution delay. It is due to concurrently executing the applications with the existing workload higher than that in the fine-grain applications. In other words, it is desirable to avoid executing the coarse-grain applications in a loaded set of processors. Thus, the development of the Contractor Oriented (CO) approach gives a solution to this problem.

To sum up, this experiment concludes that:

1. The rate of the successful booking resources for an application depends on the booking task size, the total number of tasks in an application, the organization of an application and the workload situations.
2. The Scheduler Oriented approach gives an "invisible schedule"; a schedule that cannot guarantee executable in a non-deterministic resources environment. A successfully executable schedule highly depends on the current and the scheduled workload situations.
3. Once the resources required for the scheduled application can be booked, the application can promptly be executed in a non-deterministic resources environment (This follows the resources guaranteed in contractual computing paradigm).
4. The Scheduler Oriented approach is suitable for those applications with the following characteristics:
   - An application that processes its own schedule or uses the classical task-scheduling algorithm can be used for scheduling the application.
   - The cost of the task size in an application is low.
   - There are small numbers of sub-tasks in a scheduling application.
   - The workload situation of the target processors set is low.
5. Accordingly, the Scheduling Oriented approach is applicable to a narrow range of applications

## Experiment 5 Properties of CO Approach

### E5.1 Objective

It is to study the properties of the CO-HDSC algorithm, CO-HMD and CO-HRMS algorithms for scheduling tasks in processors set with different workload situations.

### E5.2 Procedure

The proposed CO-HDSC, CO-HMD and CO-HRMS algorithms evaluated by scheduling tasks into two typical types of applications. They are defined in Appendix A2, which includes the fork and joint task graph and the irregular task graph. These graphs are configured in both fine grain structure and coarse grain structure. The tasks are then scheduled to a set of sixteen processors homogeneous configuration with different workload conditions as specified in Appendix A2. The parallel time and the average percentage of parallel time extension with reference to off-load situation are measured (specify in Appendix A4). The properties of the proposed algorithm under different workload situations are evaluated.

The performances and properties of the proposed algorithms are measured by:

The average parallel time (PT)

$$PT = \frac{\sum_{i=1}^{1000} PT_i}{1000}$$

Where $PT_i$ is the parallel time of an application scheduled to the i<sup>th</sup> sample of a processors set in a specific workload (WL).

The average percentage of Parallel Time produced by loaded processor set longer than offloaded processor set (PTI).

$$PTI = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\overline{PT_i} - PT_i}{PT_i}\right), \text{where } N = 1000$$

Where $\overline{PT_i}$ and $PT_i$ is the parallel time of an application scheduling to a loaded and an offload processors set of the ith sample of a workload group (WL) respectively.

## E5.3 Results

Following the execution trace and the produced schedule, it has found that all the algorithms can be used to schedule loaded processors so that data dependencies are satisfied. Figure E5.1a, E5.2a, E5.3a and E5.4a, demonstrated that the change of PT under different workload conditions while these algorithms were being scheduled for a particular application, K1, K30, IRR1 and IRR50 respectively. Figure E5.1b-c, E5.2b-c, E5.3b-c and E5.4b-c show the comparison among the different proposed algorithms together with their corresponding schedules. The applications under consideration are scheduled all the three algorithms that is HDSC, HMD and HRMS to a set of pre-assumed dedicated processors, concurrently under the same workload situation.

### *A) The application K1*

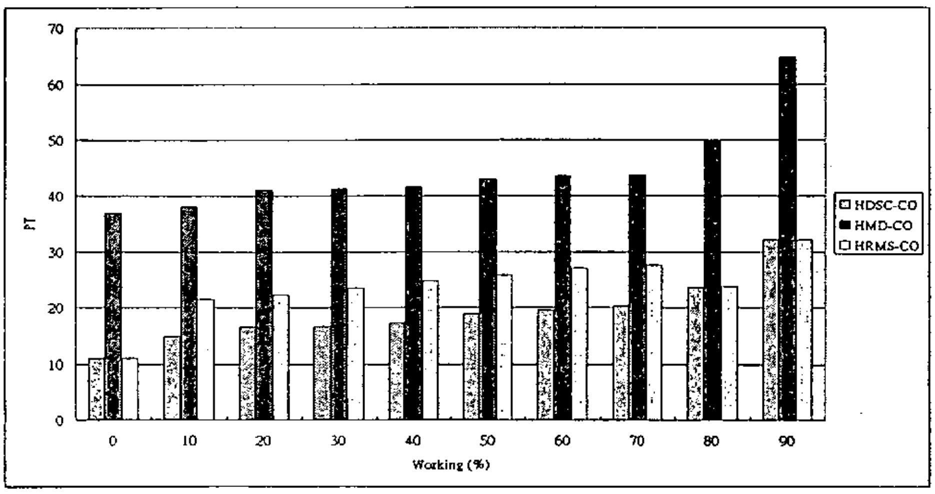


**Figure E5.1a. A bar chart of average parallel time against workload.**

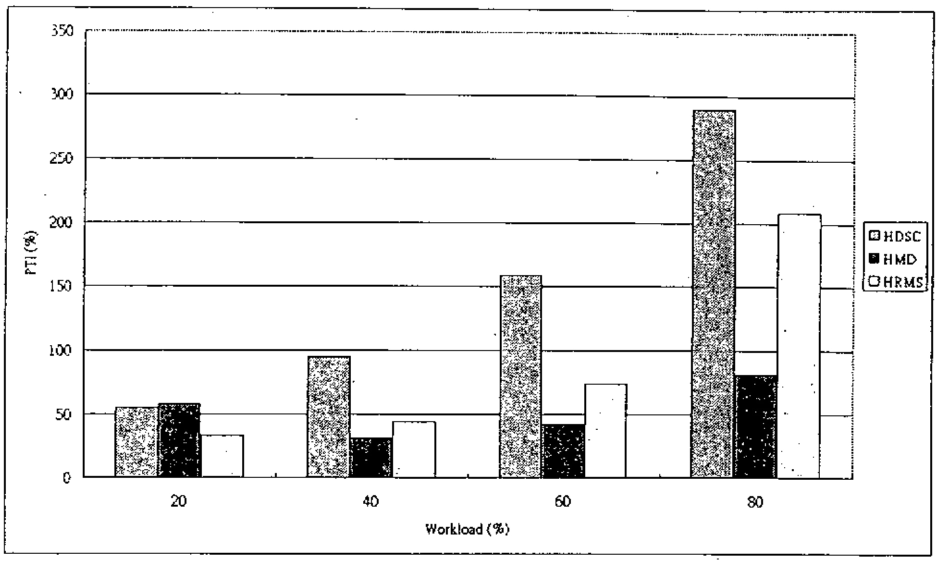


**Figure E5.1b. A bar chart of average percentage of parallel time increase against workload**

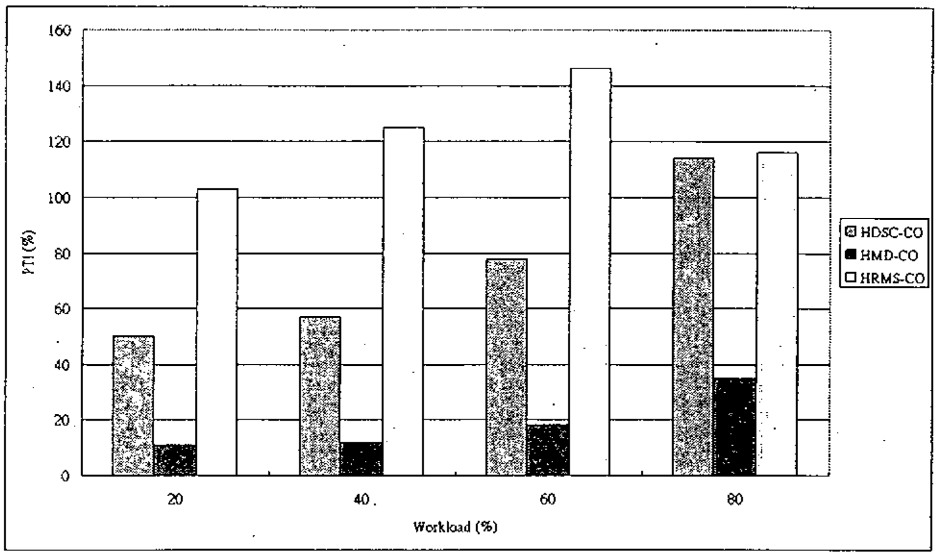


**Figure E5.1c. A bar chart of average % of parallel time increases as executing the applications concurrently against workload.**

## *B) The application K30*

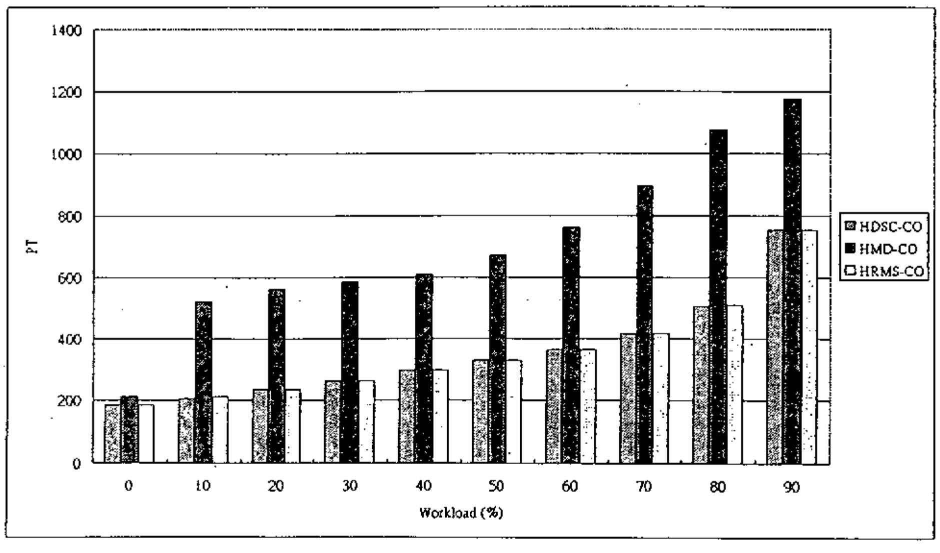


**Figure E5.2a. A bar chart of average parallel time against workload.**

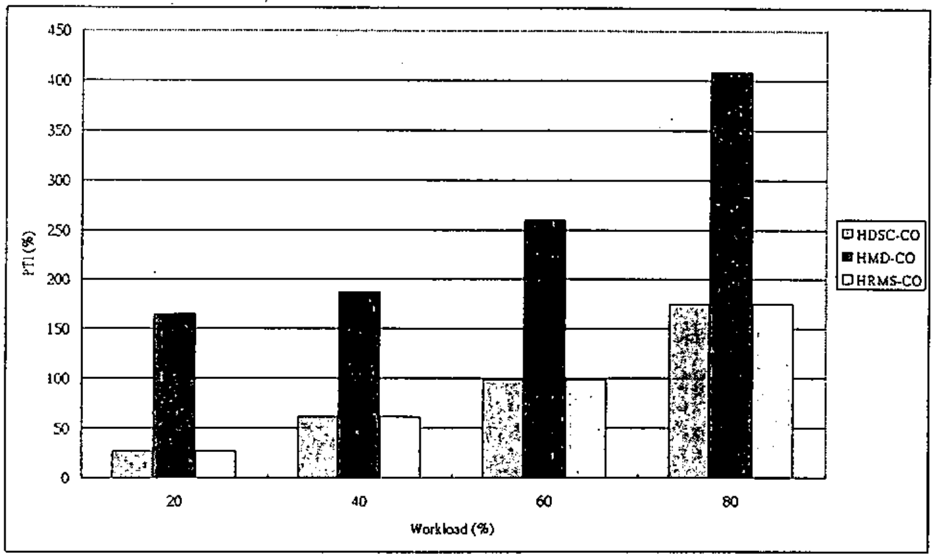


**Figure E5.2b. A bar chart of average percentage of parallel time increase against workload.**

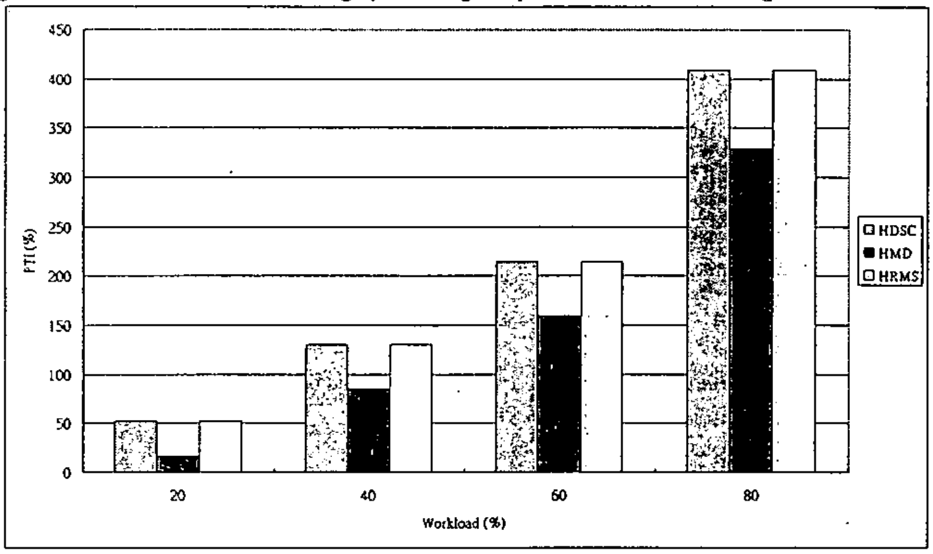


**Figure E5.2c. A bar chart of average percentage of parallel time increase as executing the applications concurrently against workload.**

## C) *The application IRR50*

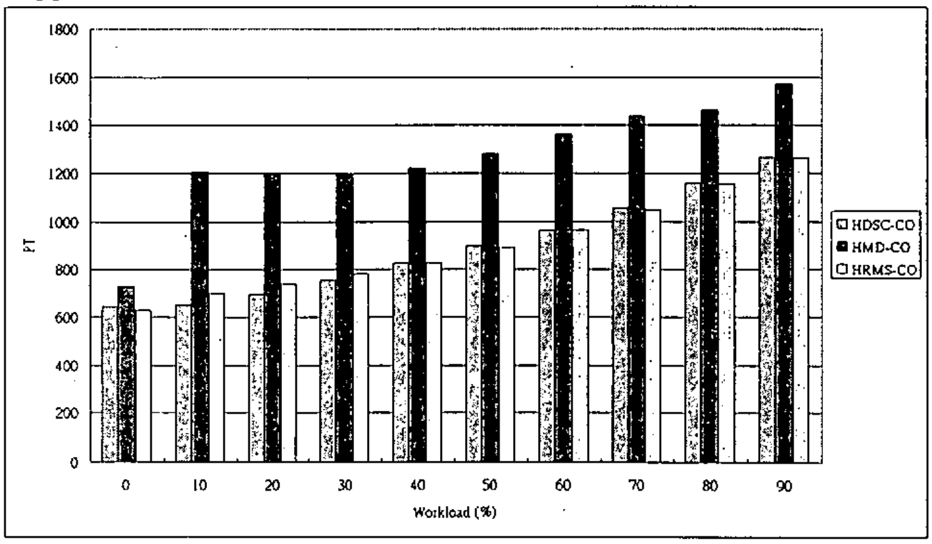


**Figure E5.3a. A bar chart of average parallel time against workload.**

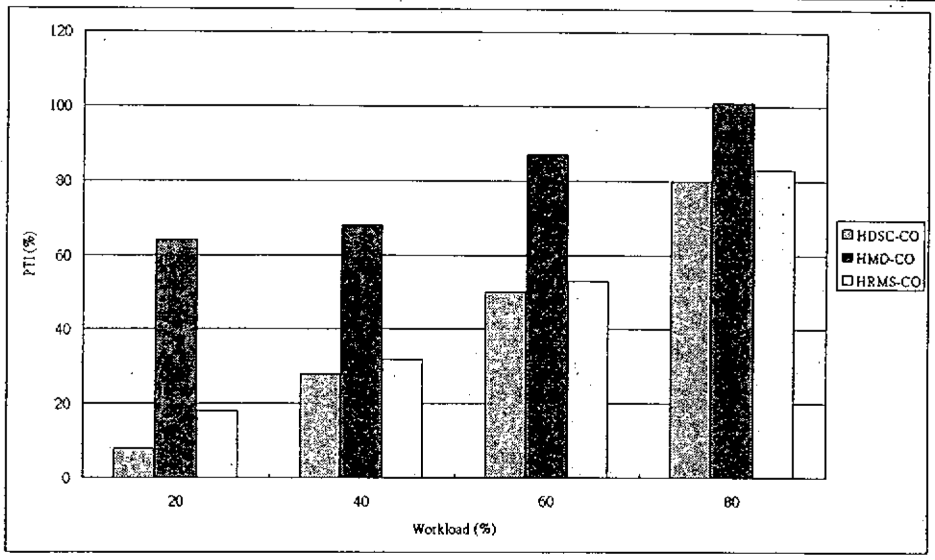


**Figure E5.3b. A bar chart of average percentage of parallel time increase against workload.**

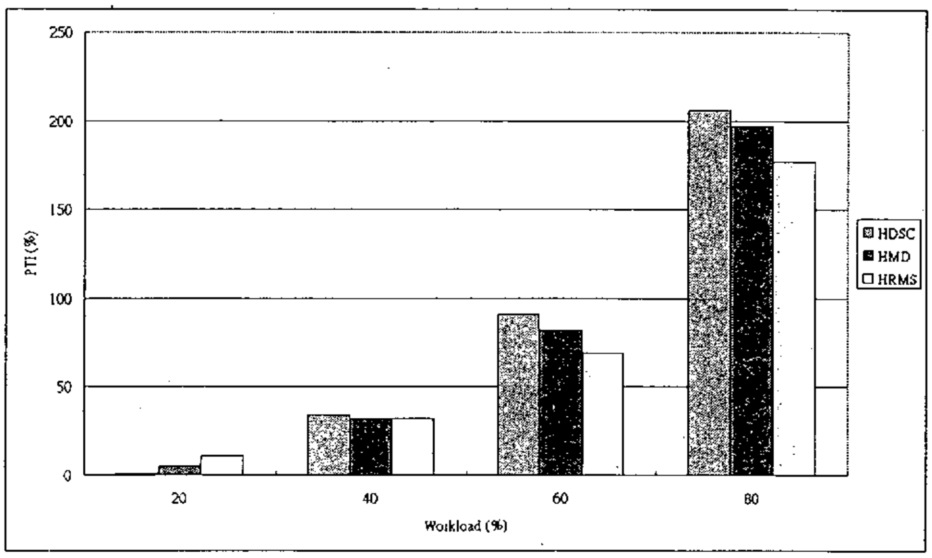


**Figure E5.3c. A bar chart of average percentage of parallel time increase as executing the applications concurrently against workload.**

## *D) The application IRR1*

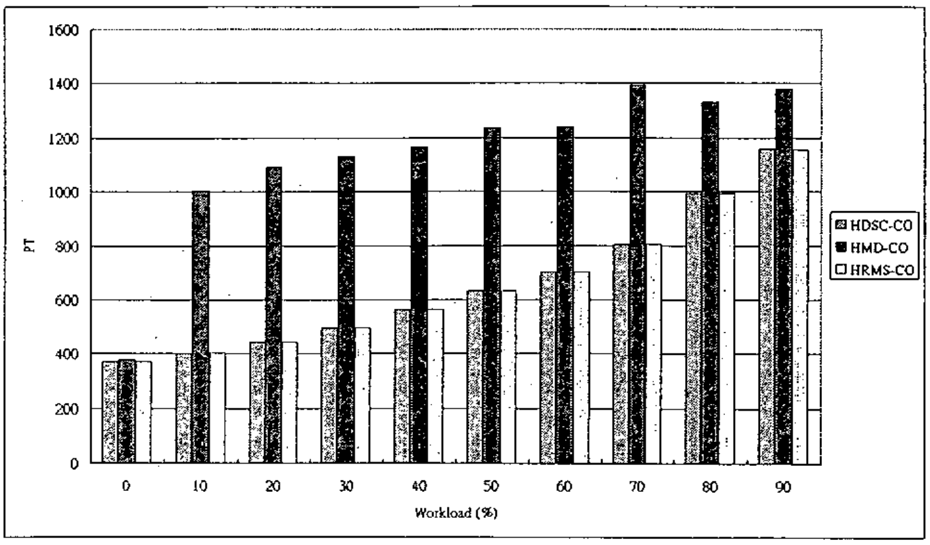


**Figure E5.4a. A bar chart of average parallel time against workload.**

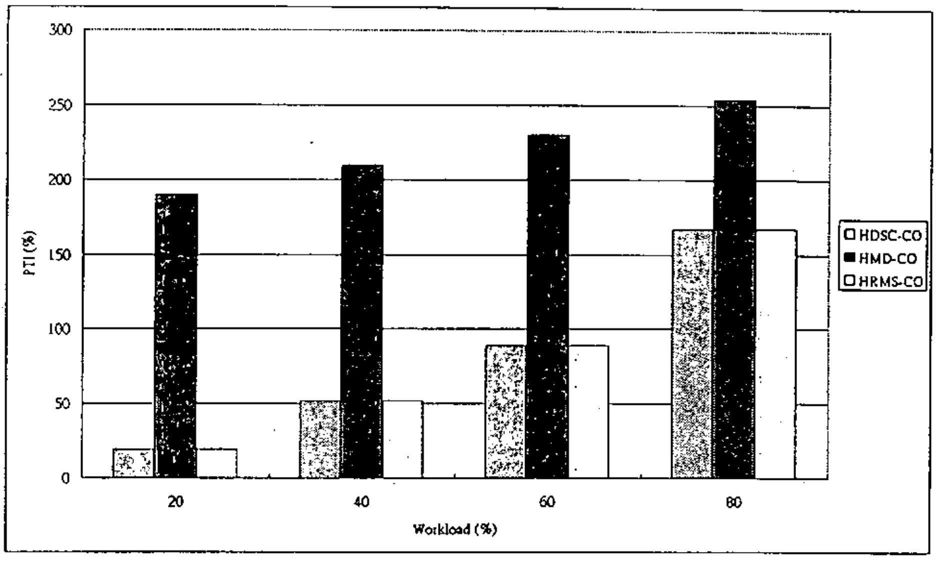


**Figure E5.4b. A bar chart of average percentage of parallel time increase against workload.**

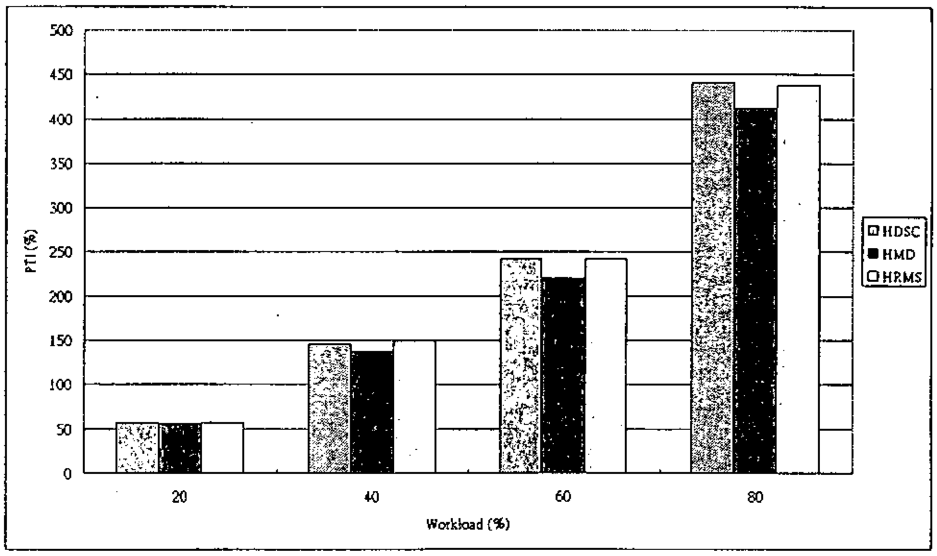


**Figure E5.4c. A bar chart of average percentage of parallel time increase as executing the applications concurrently against workload**

## E5.4 Observations

From Figure E5.1a, E5.2a, E5.3a and E5.4a, the following observations are being made:

1. The average parallel time (PT) of schedules produced by scheduling the applications (K1, K30, IRR50, IRR1) the CO-HDSC, the CO-HMD and the CO-HRMS algorithm are different. The average parallel time of a set of loaded processors would be longer than that of a set of offloaded processors.
2. For the applications K1, K30, IRR50 and IRR5, the average parallel time (PT) of schedules produced by the CO-HDSC, the CO-HMD and the CO-HRMS algorithm would increase as the workloads increase.

3. For the applications K1, K30, IRR50 and IRR5, the average parallel time (PT) of schedules produced by the CO-HMD algorithm is longer than that produced by the CO-HDSC and CO-HRMS algorithms.
4. For the applications K1, K30, IRR50 and IRR5, the average parallel time (PT) produced by the CO-HRMS algorithm closes to that produced by the CO-HDSC algorithm.
5. The CO-HDSC and CO-HRMS algorithm is less sensitive to the workload change than the CO-HMD algorithm. It is because the slope of the CO-HDSC and CO-HRMS algorithm is smaller than that of the CO-HMD algorithm.

With Figure E5.1b-c, E5.2b-c, E5.3b-c and E5.4b-c, further observations are that made:

6. The schedules produced by the CO-HDSC algorithm and the CO-HRMS algorithm can obtain smaller PTI as compared with the case of concurrently executing the scheduled applications.
7. The schedules produced by the CO-HMD obtain higher PTI as compared with the case of concurrently executing the scheduled applications.

## E5.5 Conclusions

According to the execution trace and the produced schedule, it is found that the proposed CO-HDSC, the CO-HMD and the CO-HRMS algorithms can be scheduled to loaded processor sets such that data dependencies can be satisfied. Observations 1 shows that the algorithms adapted to a suitable location for scheduling tasks in loaded processor set. Thus, it is possible that the parallel time of the produced schedules may be larger than that in an offload condition. Observation 2 states the fact that the parallel time will increase as the workload of a processor set increases.

Observation 3 and 4 state that the performance of the CO-HDSC algorithm is being closed to the CO-HRMS algorithm is better than of the CO-HMD algorithm when they are being used to schedule the four applications. From the viewpoint of parallel time reduction, the CO-HMD algorithm would not perform as good as either the CO-HDSC or the CO-HRMS algorithm. Observation 5 summarizes the implication that the CO-HDSC algorithm is less sensitive to workload changes, but the CO-HMD algorithm is very sensitive to workload changes.

Observation 6 shows that the CO-HDSC and the CO-HRMS algorithm always perform better than executing the application concurrently in loaded processors set. This concludes that the use of Contractor Oriented (CO) approach under the contractual computing paradigm not only can provide a stable and reliable execution environment for running applications, but also allow running applications to be executed with better parallel performance. Thus, the overall parallel time of execution of applications can always be improved. Observation 7 shows a counter example that task scheduling heuristic using in the Contractor Oriented approach can sometimes with the scheduling performance being affected. In this case, the parallel time produced by the CO-HMD algorithm is much longer than running the application concurrently. All in all, it is concluded that the CO-HDSC and the CO-HRMS algorithm are better than the CO-HMD algorithm and the Contractor Oriented approach as they can make use of the advanced scheduling information for making a better scheduling decision.

# Chapter 9: Conclusions

An investigation of task scheduling techniques for practical parallel and distributed systems has been conducted. It began with the technology evolution of the practical parallel and distributed systems and the roles of task scheduling algorithms. Then, the popular task scheduling technique "List Scheduling Heuristic" was studied. The investigation of the two specific task scheduling algorithms, which were designed based on the heuristic called the Dominant Sequence Cluster (DSC) and the Mobility Direct (MD) are conducted. It is found that with slightly modification of the MD algorithm, better performance scheduling solution can be obtained. Based on the experimental result shown in Experiment 1 of Chapter 8 and Table 9.1, the Relative Mobility Scheduling algorithm (RMS), was developed and studied and implemented in a typical, practical, parallel and distributed systems. However, prompted scheduling performance is found to be hard to achieve in practice due to that the homogeneity in processors configuration and the non-deterministic of resources cause such performance degradation. A comprehensive study for task scheduling in practical parallel and distributed systems was focused on these two factors.

Firstly, the technique of using the List Scheduling Heuristic in the heterogeneous environment (a environment that contains dedicated set of heterogeneous processors) was proposed. A novel technique the "Heterogeneous List Scheduling (HLS)" was proposed, which is based on the List Scheduling Heuristic that had been generalized in order to take advantage of the scheduling heuristic in a two step approach. Consequently, scheduling algorithms can be easily designed by plotting heuristics used in the List Scheduling Heuristic in heterogeneous environment. Three heterogeneous scheduling algorithms, the Heterogeneous Dominant Sequent Cluster (HDSC), the Heterogeneous Mobility Direct (HMD) and the Heterogeneous Relative Mobility Scheduling algorithm (HRMS) were proposed and investigated.

It was demonstrated in Experiment 2 that these algorithms are good as the HLS technique, being used to schedule tasks in a heterogeneous environment. The performance of these algorithms depends on (i) the type of scheduling heuristic being used in the algorithms, (ii) the applications to be scheduled and (iii) the configuration of the heterogeneous processors set. Therefore, each of these algorithms would perform differently in various configurations and it is not desirable to compare their performance individually. In general, the proposed scheduling algorithms keep on scheduling task to a narrow set of high performance processors with the goal of minimizing parallel time. A shorter parallel time schedule can be produced as the set of processors is configured more deviated from homogeneous (or even a supper computer can be included). This demonstrated that the proposed algorithms are feasible in applying practical parallel and distributed systems as it always configured in upgrading individual processor from a homogeneous environment (a typical example is shown in Appendix A2). The proposed algorithms can also be adapted to environmental changes and can still produce better scheduling solutions for any upgrade.

Secondly, when dealing with the problem of non-deterministic resources. The concept of using dynamic scheduling technique to improve the shortfall of non-deterministic resources is possible and the unreliable estimation on the workload and the cost in rescheduling can reduced. Based on this observation, a new "Contractual Computing paradigm" for resources management was developed. In the paradigm, a two-layer scheme for resources management is considered. Through a contractual mechanism, the resources management, in which used by different users and applications is self maintained with each layer. Therefore, information describing the instantaneous workload is readily available and is not necessary to be estimated during task scheduling. The contractual paradigm also provides a stable and reliable environment for program execution. This highly increases the feasibility of implementing task scheduling algorithms (or their solutions) in a non-deterministic resources environment, which is the reason steer the study investigation to the Contractual Computing paradigm.

Based on the paradigm, three task-scheduling techniques are proposed and investigated. They are the Scheduler Oriented (SO) approach, the Resources Oriented (RO) approach and the Contractor Oriented (CO) approach. The SO approach aims at performing scheduling regardless of the workload condition with the resources and the required resources are booked according to the produced schedule. It was found by Experiment 3 to 5 that better scheduling solutions are achieved. However, there are difficulties in resources booking because of the tight requirement of the produced schedule. The situation becomes worse while the workload of the system is increasing. According to Experiment 4, this approach is only applicable to applications with lower number of tasks, low cost of task and its scheduling. The application should be executed in target processors set with low workload situation and the applicability of this approach is restricted.

The second approach called the Resources Oriented approach is consider and it reserves a block of resources ahead for task scheduling and it partitions and books parts of resources (processing powers) to form a virtually dedicated processor (VDP). Based on the VDP, those basic scheduling algorithms can be directly applied for task scheduling. The advantage of the RO approach over the SO approach is that the produced schedule should obtain their required resources for execution, but there would be wastage of resources in the booked VDP set as the resources are not fully utilized in the schedule by a scheduling application. The partitioning process under the C C paradigm is a mutual cooperative process among users and applications obtaining an optimum partitioning solution for a specific application is a difficult problem. Hence, the applicability of this approach is still restricted.

This leads to the development of the Contractor Oriented approach, which scheduling tasks in the paradigm produce the best result under the active research and investigation. With this approach, the workload conditions about the target processors are collected to form advanced scheduling information for making task-scheduling decision. At the same time, the resources required by the schedules are to be booked immediately. Both analytical (Section 7.6) and experimental studies (Experiment 5) support the result. The tasks scheduling with foreseeing the workload situation could adapt to dynamic changing workload situations. Experiment 5 shows that a schedule is always obtainable in different workload situations and will not lead to resources wastage. The CO is better than the SO and the RO approach, but the parallel time of the produced schedule still depends on the workload condition. Hence, parallel time produced by the CO approach may not be as good as that produced by the SO and the RO approach. The parallel time of an application is extended by increasing the workload, the parallel time produced by the proposed task scheduling algorithms (CO-HDSC and CO-HRMS) is much shorter than running the application being executed in concurrent and without the scheduling. This showed that the Contractor Oriented approach and its related scheduling algorithms are feasible and adaptive for task scheduling in practical parallel and distributed systems.

After all the previous studies, the SO, RO and CO approach can perform differently in different scheduling applications and different workload situations. Applying of these approaches will depend on the characteristics of the applications, the workload situation and the performance requirement. Chapters 4-5 and Experiments 5-6, the SO and RO approach is suitable for those applications possessing with theirs own schedule or applications that can be effectively scheduled by classical task scheduling algorithms and demanded for a stable execution environment. They are also suitable for applications demanding for high performance scheduling solution in term of parallel time. The SO approach is restricted to those applications with (i) low number of tasks, (ii) low task cost and (iii) scheduled to target processor set with low workload situation. An example of application, which is applicable for the SO approach is asK1 in Experiment 4 in which a low workload processors set (workload less than 10%) is perceived. On the other hand, the RO approach is restricted to those applications that can tolerate resources wastage and possess with pre-specified resources partitioning strategies. The most useful application, which is applicable for the RO approach is those real time computation problem.

Finally, the CO approach overcomes the problem of unsuccessful resources booking arose in the SO approach and the problem of resources wastage in the RO approach. The CO approach can produce a schedule that is adaptive to dynamic change of workload situation. The CO approach is found to be the most suitable for most practical application that can tolerate longer parallel time as the workload situation increases. For example, solving non-real time applications and intensive computation scientific of problem.

In conclusion, the techniques for task scheduling in practical parallel and distributed systems from the viewpoints of heterogeneous resources configuration and non-deterministic resources were conducted. The proposed RMS algorithm improved the MD algorithm for scheduling task in a homogeneous environment with creditable performance. The Heterogeneous List Scheduling

Heuristic (HLS) proposed is a feasible method for designing task scheduling algorithms in heterogeneous environment. It followed with three proposed heterogeneous scheduling algorithms to support the arguments on the HLS techniques. According to Experiment advantages of the HLS technique as well as the proposed algorithms in scheduling task to a set of processors that are configured are more deviated to homogeneous environment. The properties of the proposed algorithms are summarized in Table 9.2.

The problem of resources non-deterministic was overcome by using the developed task-scheduling techniques with contractual computing paradigm. According to experimental studies and analytical studies, the three approaches possess different characteristics in different applications and different behaviors under different workload situations. They can produce feasible scheduling solutions in a non-deterministic resources environment, but the two challenging problems of task-scheduling problem in practical parallel and distributed systems in heterogeneous resources configuration and the non-deterministic nature of resources are studied. A summary of their difference is summarized in Table 9.2, the findings are also given details in previous chapters accordingly.

| | DSC | MD | RMS |
|---|---|---|---|
| **Heuristic of Priority** | Critical Path<br>CP = tlevel($n_x$) + blevel($n_x$) | Relative Mobility<br>$Mr(n_x) = (T_L(n_x) - T_S(n_x))/W(n_x)$ | Relative Mobility<br>$Mr(n_x) = (T_L(n_x) - T_S(n_x))/W(n_x)$ |
| **Heuristic of task to processors assignment** | Earliest finish time through minimization procedure<br>(Consequent of ISH2) | First processor that satisfies FACT 1<br>(Consequent of ISH0) | Processor satisfies FACT 1 with earliest finish time<br>(Consequent of ISH2) |
| **Performance – in term of parallel time (experiment 1)** | Best<br>(optimum for fork and joint)<br>[Yang94] | Fair<br>Sub-optimum | good<br>Sub-optimum and better than MD |
| **Complexity** | O(e+n) | $O(n^2 \log n)$ | $O(n^2 \log n) + O(np)$ |

**Table 9.1, Summary of the DSC, MD and RMS algorithms**

| | Heuristic | Task Assignment Strategy | Algorithm Reducibility | Scheduling Behavior | Tend of Scheduling |
|---|---|---|---|---|---|
| **HDSC** | Critical Path, compile with the heuristic determine constrains in first part of HLS | Schedule task to a processor that can finish it as soon as possible | Reduced to DSC in unbounded number of homogeneous processors | Produces a shorter parallel time schedule as the processors set is configured more deviated from homogeneous | Tend to schedule task to a single high performance processors. |
| **HMD** | Relative Mobility, compile with the heuristic determine constrains in first part of HLS | It will schedule to the first available processor ranking from fast to slow that satisfy the Condition 1 | Reduced to MD in unbounded number of homogeneous processors | | |
| **HRMS** | Relative Mobility, compile with the heuristic determine constrains in first part of HLS | It will schedule to the processor that not only satisfy the Condition 1, but also complete in the earliest time | Reduced to RMS in unbounded number of homogeneous processors | | |

**Table 9.2, Summary of HDSC, HMD and HRMS algorithms**

| Approach | Philosophy | Scheduling algorithms | Adaptive | Task scheduling performance | Resources acquisition | Resources booking behavior | Efficiency |
|---|---|---|---|---|---|---|---|
| **SO** | 1. Perform task scheduling<br>2. Book resources in accordance with the produced schedules | Existing | No | 1. Attend to desired performance of the schedule (if requred resources can be booked)<br>2. Task scheduling algorithm use | Depends on:<br>1. Timing constrain of the scheduled tasks<br>2. Current and scheduled workload | Pieces in maximum consumption | No resources wastage |
| **RO** | 1. Book a VDP first<br>2. Scheduling according to VDP | Existing | According to VDP specificat-ions | 1. Depends on the VDP<br>2. Task scheduling algorithm used | Depends on<br>1. VDP configuration<br>2. resources allocation strategies | Block | Resources wastage<br><br>(de-booking is needed) |
| **CO** | Perform task scheduling by accessing the current and scheduled resources constrains | Dedicated | Yes | Depends on<br>1. current and scheduled workload<br>2. Scheduling constrains<br>3. Task scheduling algorithm use | Guaranteed for relax booking constrins | Pieces in variable consumption | No resources wastage |

**Table 9.3, Summary of SO, RO and CO approaches**

## References

[ACDi74] T.L. Adam, K.M. Chandy, J.R. Dickson, "A comparison of list schedulers for parallel processing systems,"ACM Journal of Communication, vol. 17, pp. 685-690, 1974.

[Beguelin91] A. Beguelin, J.J. Dongarra, G.A. Geist, R. Manchek and V.S. Sunderam, "A user's guide to PVM parallel virtual machine," Technical Report TM-11826, Oak Ridge National Laboratory, July 1001.

[Chan96] Wai-Yip Chan, Chi-Kwong Li, Huiwei Guan, "An Illiac Implementation of BP Neural Network on a Multiple Processor System", Proceedings of the Int'l Conference on Neural Information Processing, vol. 2, pp. 1007-1011, 1996.

[Chan97a] Wai-Yip Chan and Chi-Kwong Li, "An aggressive Parallel Tasks Scheduling Algorithm: Relative Mobility Scheduling Algorithm (RMS)," Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM'97), vol. 2, pp.946-949, August 1997.

[Chan97b] Wai-Yip Chan and Chi-Kwong Li, "The Relative Mobility Scheduling Algorithm (RMS)," in Proceedings of ISCA International Conference on Computer Applications in Industry and Engineering (CAINE-97), 1997.

[Chan97c] Wai-Yip Chan and Chi-Kwong Li, "Heterogeneous Dominant Sequence Cluster (HDSC): A Low Complexity Heterogeneous Scheduling Algorithm," Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM'97), vol 2, pp.956-959, August 1997.

[Chan97d] Wai-Yip Chan and Chi-Kwong Li, "A New Technique of List Scheduling Algorithm for Heterogeneous Processors Systems," in Proceedings ISCA International Conference on Computer Applications in Industry and Engineering (CAINE-97), 1997.

[Chan97e] Wai-Yip Chan and Chi-Kwong Li, "Scheduling Tasks in DAG to Heterogeneous Processors System," Proceedings of the Sixth Euro-micro workshop Parallel and Distributed Processing, pp. 27-31, January 1998.

[Cheuk97] W. K. Cheuk, "Kernel Support for Contractual Computing," Technical Report, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, 1998.

[Chret89] P. Chretienne, "Task Scheduling Over Distributed Memory Machines," Proceedings International Workshop Parallel and Distributed Algorithms, North Holland Publishers, Amsterdam, 1989.

[MVDeva89] M.V. Devarakonda an R.K. Iyer, "Predictability of process resource usage: A measurement-based study on UNIX," IEEE Transactions on software engineering, pp. 1579-1586, December 1989.

[Elre90] H. El-Rewini, "Task partitioning and scheduling on arbitrary parallel processing systems," Ph.D. thesis, Department of Computer Science, Oregon State university, 1990.

[El-Rewini94a] Hesham El-Rewini, Theodore G. Lewis and Hesham H. Ali, "Task Scheduling in Parallel and Distributed Systems," Chapter 5, pp.60-61, Prentice Hall, 1994.

[El-Rewini94b] Hesham El-Rewini, Theodore G. Lewis and Hesham H. Ali, "Task Scheduling in Parallel and Distributed Systems," Chapter 8, pp. 131-149, Prentice Hall press.

[El-Rewini94c] Hesham El-Rewini, Theodore G. Lewis and Hesham H. Ali, "Task Scheduling in Parallel and Distributed Systems," Chapter 5, pp. 51-87, Prentice Hall press.

[EPCC91] Edinburgh Parallel Computing Centre, University of Edinburgh, "CHIMP concepts," June 1991.

[EPCC91] Edinburgh Parallel Computing Centre, University of Edinburgh, "CHIMP Version 1.0 interface," May 1992.

[Ewing87] Ewing Lust, Ross Overbeek, et al., "Portable programs for parallel processors," Holt, Rinehart and Winston, Inc., 1987.

[Guan95] Huiwei Guan, Chi-Kwong Li, Wai-Yip Chan, "A parallel Implementation of BP Neural Network on a Multiple Processor System" Proc. 1995 International Symposium on Artificial Neural Networks, pp. E2-19-24.

[Hu61] T. C. Hu, "Parallel sequencing and assembly line problems," Oper. Res., vol. 9, no. 6, pp.841-848.

[Krua87] B. Kruatrachue, "Static task scheduling and grain packing in parallel processing systems," PhD. Diss., Department of Electrical and Computing Engineering, Oregon state University, Corvallis, 1987.

[Kim88] S. J. Kim and J. C. Browne, "A general approach to mapping of parallel computation upon multi-processor architectures," Proceedings of International Conference on Parallel Processing, vol. III, pp. 1-8, 1988.

[Kruat87] B. Kruatrachue, "Static Task Scheduling and Grain Packing in Parallel Processing

Systems," Ph.D. dissertation, Electrical and Computer Eng. Dept., Oregon State Univ., Corvallis, 1987.

[Kruat88] B. Kruatrachue and T. Lewis, "Grain size determination for parallel processing," IEEE transactions on software, Jan. 1988, pp. 23-32.

[Kunz91] T. Kunz, "The influence of different workload descriptions on a heuristic load balancing scheme," IEEE Transaction on software engineering, pp. 725-730, July 1991.

[Lam95a] W. K. Lam, "The Contractor Model of Networked Parallel Computing," TROCC No. I – Overview, Computer Research Institute, Tsinghua University, PRC, December 1995.

[Lam95b] W. K. Lam, "An Analysis of the Contractor Model of Networked Parallel Computing," TROCC No. II – Relationship with Existing Paradigms, Computer Research Institute, Beijing, PRC, January 1996.

[Lam96a] W. K. Lam, "A Preliminary Investigation on Contractor Technology," TROCC No. III – Implementation Issues, Computer Research Institute, Tsinghua University, Beijing, PRC, January 1996.

[Lam97] Wai-kin Lam, San-li Li, "A Brief Review on the Development of Contractual Computing," Proceedings Int'l workshop on Computational Science and Engineering, IWCSE'97, pp. 171-181, 1997.

[Lo88] V. M. Lo, "Heuristic Algorithms for Task Assignment in Distributed Systems," IEEE Trans. Computers, vol. C-37, no. 11, Nov. 1988, pp. 1384-1397

[McCreary94] C. L. McCreary, A. A. Khan, J.J. Thompson, M. E. McArdle, "A Comparison of Heuristics for Scheduling DAGS on Multiprocessors," Journal of Parallel and Distributed Computing, vol. 16, pp. 276, Dec. 1993.

[Papadiou90] C. Papadimitriou and M. Yannakakis, "Towards an Architecture-Independent Analysis of Parallel Algorithms," SIAM J. Computer, vol.19, pp.322-328, 1990.

[Parasoft88] Parasoft Corporation, "Express Version 1.0: A communication environment for parallel computers," 1988.

[Ralpha92] R. Butler and E. Lusk, "User's guide to the p4 parallel programming system," Technical Report ANL-92/17, Argonne National Laboratory, October 1992.

[CVRamam] C. V. Ramamoorthy, K. M. Chandy and M. J. Gonzalez Jr., "Optimal scheduling strategies in a multi-processor system," IEEE Trans. Comp. C-21, 2 (Feb. 1972), pp. 137-146.

[Sark89] V. Sarkar, "Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors," MIT Press 1989.

[Sarkar86] Sarkar, V. and J. Hennessy, "Compile-Time Partitioning and Scheduling of Parallel Programs," Symposium Compiler Construction, ACM Press, New York, N.Y., 1986, pp.17-26.

[Sarkar89] Sarkar, V., "Partitioning and Scheduling Parallel Programs for Multiprocessors," MIT Press, Cambridge, Mass., 1989.

[Shirazi90] Shirazi, B., M. Wang and G. Pathak, "Analysis and Evaluation of Heuristic Methods for Static Task Scheduling," J. Parallel and distributed Computing, vol. 10, pp.222-232, 1990.

[Siu96] Siu, Y.M., "O.S. Support for Cluster Computer," Technical Report, Department of Electronic and Information Engineering, The Honk Kong Polytechnic University, Hong Kong, December 1996.

[StonE57] Stone, H.S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Trans. Software Eng., vol. SE-3, no. 1, pp. 85-93, Jan. 1997.

[WuGa88] M.Y. Wu and D. Gajski, "A programming aid for Hybercube architecture," Journal of Supercomputing, vol2, pp. 349-372, 1988.

[Yang94] Tao Yang, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," IEEE. Transactions on Parallel and Distributed System, vol.5, no.19, September 1994.

# Appendix A: Experiment Setup

## A1 Procedure of Experiment 1

Since heuristic idea is used in both MD and RMS algorithm, the scheduling problem that is not sure in leading to an optimal scheduling solution is NP-complete. Thus, it is indispensable to compare the average performance of these algorithms by randomly generated graphs. In order to compare their task-scheduling performance in different granularities and structures of the scheduling task graphs, 60,000 task graphs are generated randomly in this experiment. They are classified into three groups in different granularities (ratio of task cost over communication cost: R/C):

| | |
|---|---|
| **Group 1 (G1):** | Fine-grain, R/C range = 0.1-0.3. |
| **Group 2 (G2):** | Median-grain, R/C range = 0.8-1.2; The average cost of computation and communication are close. |
| **Group 3 (G3):** | Coarse-grain, R/C range = 3-10. |

Each of these groups is further divided into six sub-groups summarized in Table A1.1. So 3,400 samples of task graphs in each sub-group are being scheduled by the MD and the RMS algorithm for comparing with their task scheduling performance.

| Sub-Group | Number of layers | Number of tasks | Number of edges |
|---|---|---|---|
| S1 | 9-11 | 50 | 170 |
| S2 | 9-11 | 60 | 200 |
| S3 | 18-21 | 100 | 340 |
| S4 | 18-20 | 180 | 620 |
| S5 | 18-20 | 350 | 1200 |
| S6 | 36-40 | 540 | 1800 |

**Table A1.1. Sub-grouping of random generated task graphs**

After generating the task graphs, each of these will be scheduled by both MD and RMS algorithm. Both can be used to schedule tasks in either bounded or unbounded number of processors. The MD algorithm provides a bound on the minimum number of processors for completing the task graph. Thus, a comprehensive comparison on scheduling performance in between the RMS and the MD algorithm will focus on the following three sets of processors configurations:

**Set 1:** Unbounded number of processors ($|P| \geq |N|$)*;

**Set 2:** Minimum number of processors which is determined by the MD algorithm required to execute the task graph ($|P| = |P_{min}|$)*; and

**Set 3:** Bounded number of processors ($|N| > |P| > P_{min}$)*.

There are two measurement Figures, which are used to measure and compare their task scheduling performance. They are:

1. The probability of the case in which the parallel time produced by the RMS algorithm is shorter than the MD algorithm (Pr);
2. The percentage of parallel time that the RMS improves over the MD algorithm (R/M). The R/M is calculated by:

   $R/M = (1 - PT_{RMS}/PT_{MD}) * 100$.

## A2 Benchmark Applications Applied in Experiment 2 to 5

To perform an adequate study of scheduling algorithm, it is desirable to select a variety of scheduling task graphs (applications) to be scheduled. This is to expect that the scheduling algorithm will work better on one type of task graph than others. That reflects the characteristic of the algorithm. In this experiment, two typical task graphs (applications) with distinct characteristics are selected as the scheduling task graph. They are the fork and joint task graph (K1) [Hu61] [McCreary94] and irregular task graph (IRR50) [Kim88]. A graphical description of the fork and joint task graph (serial time of the application is 32 unit) is given in K1 (Figure A2.1). This is an adaptation of a graph described by the author of the "Linear Clustering" scheduling algorithm proposed by T.C. Hu in 1989. The edges within the hart of the graph alternate between values of 1 and 10. Because all tasks have computation value equal to 1, communication is the dominating factor when deciding how to schedule this graph. Therefore, it is classified to fine-grain task graph. It is suggested to study the property of the HDSC algorithms for scheduling task to coarse-grain task graph (The task cost would be much higher than edge cost). The cost of each task is changed to 30 (Serial time of the application is 960 unit). This task graph is called K30.

Another task graphs to be studied are the adaptation of a program dependent graph (IRR50)

* $|P|$ is the number of processor in a processor set P; $|N|$ is the total number of tasks in the scheduling task graph and $P_{min}$ is the minimum number of processors which is determined by the MD algorithm and is required to execute the task graph.

of a physics algorithm [Kim88] shown in IRR50 (Figure A2.2). The reason for choosing this graph related to the highly irregular layout and fine grain (serial time of the application is 1332 unit). Similarly, a coarse-grain version of the task graph is set up in which the cost of all the edges are reduced to 1. The modified graph is called IRR1.

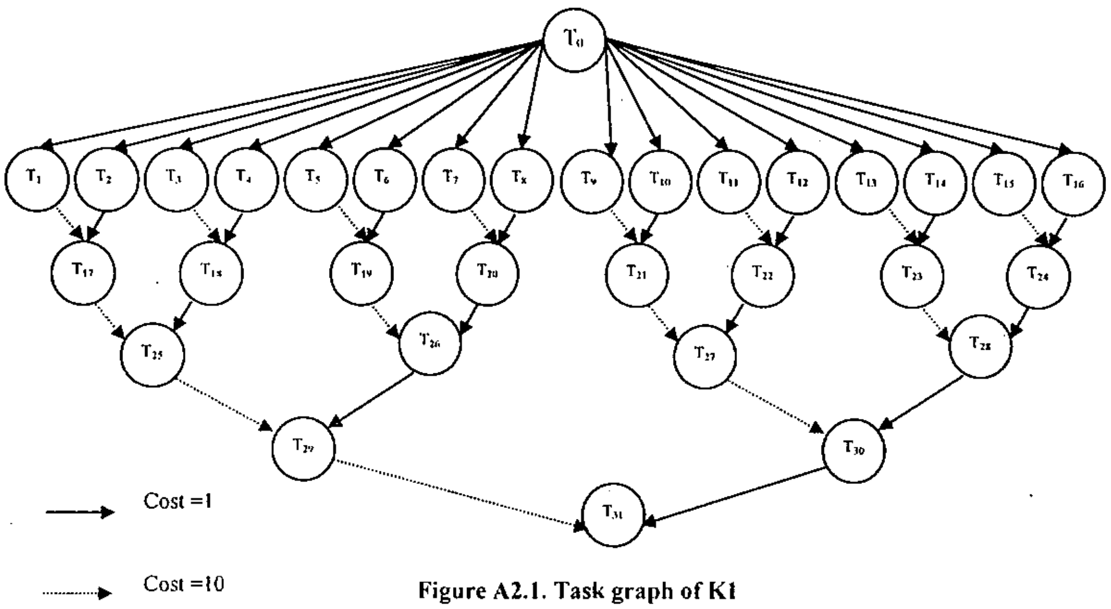


**Figure A2.1. Task graph of K1**

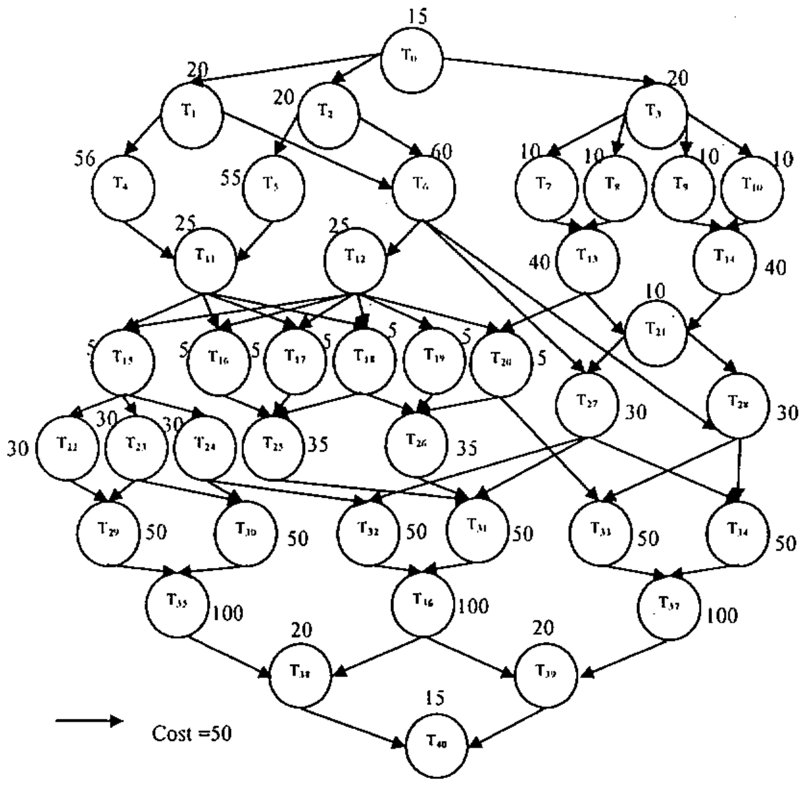


**Figure A2.2. Task graph of IRR50**

## A3 Processor Configuration Used in Experiment 2 to 5

Consider that a laboratory has to set up with 16 computers (processors) for solving the above problems. The configurations of the processors set (*P*) are classified into four sets as shown in Table A3.1. The obvious configuration of these processors is configured homogeneously. That is, all the processors are in the same speed. The set S1 shows a configuration of 16 processors for each of these has 1 unit of processing power. However, according to technology growth and budgeting, it is possible to replace some of these processors by processors with higher processing power. For example, the processors $P_0$ and $P_1$ are replaced by two processors having double processing power. This configuration is called S2. The processors $P_0$, $P_1$ and $P_2$ are replaced by one processor having triple processing and two processors having double processing power respectively. The configuration is called S3. Extremely, it is possible to install a super computer to solve the computation problem. Therefore, the set S4 configures a processor with 16th times processing power than the original processor. This configuration shows the preference of the HDSC algorithm in scheduling tasks to a super-processor (super computer) or a set of distributed computers.

| Set: | $p_0$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ | $p_9$ | $p_{10}$ | $p_{11}$ | $p_{12}$ | $p_{13}$ | $p_{14}$ | $p_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S1: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| S2: | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| S3: | 3 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| S4: | 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table A3.1. Configuration of target processor sets.**

## A4 Processors Configuration Used in Experiments 4 and Experiment 5

The proposed algorithms are designed for scheduling task to the homogeneous and heterogeneous processor environments. For simplicity, the set of sixteen processors configured homogeneously is selected, as the target processors are set. Based on the configuration, the workloads of these processors are randomly generated. In fact, these workloads are generated by booking the processing power from different parts of processor so that they are consumed and cannot be booked by other application. For example, the workload of a processor $p_0$ show in Figure A4.1 is 30%. For instance, a processor set is called "fully-loaded" when the processing power of all the processors in the set is completely consumed. A processor set is called "offloaded" when processing power of all processor in the set are not consumed. A "loaded" processor set is describing the processing power of processor set that its processing power are neither fully loaded nor offloaded. In this experiment, the workload of the processor set is randomly generated and classified into 9 groups summarized in Table A4.2. 100 samples of the workload are generated in each group and the application is scheduled to these samples of processor set so that the parallel time of schedule produced in each group is measured by taking their average.

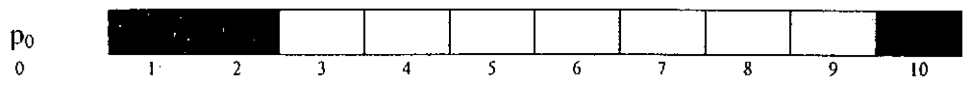


**Figure A4.1. Workload of $p_0$ between 0 to 10 second is 30%.**

| Group | G1 | G2 | G3 | G4 | G5 | G6 | G7 | G8 | G9 |
|---|---|---|---|---|---|---|---|---|---|
| Workload (%) | 0 | 4 | 8 | 12 | 16 | 20 | 40 | 60 | 80 |

**Table A4.2. Grouping of workload**

# Appendix B: Schedule Details

## B1 Schedules of Application K1

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/1.00/1.00 | -> | T1/1.00/2.00/1.00 | -> | T9/2.00/3.00/1.00 | -> |
| | T17/4.00/5.00/1.00 | -> | T21/5.00/6.00/1.00 | -> | T25/6.00/7.00/1.00 | -> |
| | T27/7.00/8.00/1.00 | -> | T29/8.00/9.00/1.00 | -> | T30/9.00/10.00/1.00 | -> |
| | T31/10.00/11.00/1.00 | | | | | |
| P1 | T5/2.00/3.00/1.00 | -> | T19/4.00/5.00/1.00 | -> | T26/6.00/7.00/1.00 | |
| P2 | T3/2.00/3.00/1.00 | -> | T18/4.00/5.00/1.00 | | | |
| P3 | T2/2.00/3.00/1.00 | | | | | |
| P4 | T4/2.00/3.00/1.00 | | | | | |
| P5 | T13/2.00/3.00/1.00 | -> | T23/4.00/5.00/1.00 | -> | T28/6.00/7.00/1.00 | |
| P6 | T11/2.00/3.00/1.00 | -> | T22/4.00/5.00/1.00 | | | |
| P7 | T10/2.00/3.00/1.00 | | | | | |
| P8 | T7/2.00/3.00/1.00 | -> | T20/4.00/5.00/1.00 | | | |
| P9 | T6/2.00/3.00/1.00 | | | | | |
| P10 | T15/2.00/3.00/1.00 | -> | T24/4.00/5.00/1.00 | | | |
| P11 | T12/2.00/3.00/1.00 | | | | | |
| P12 | T8/2.00/3.00/1.00 | | | | | |
| P13 | T14/2.00/3.00/1.00 | | | | | |
| P14 | T16/2.00/3.00/1.00 | | | | | |
| P15 | | | | | | |

**Figure B1. A schedule of K1 to S1 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.50/1.00 | -> | T1/0.50/1.00/1.00 | -> | T9/1.00/1.50/1.00 | -> |
| | T5/1.50/2.00/1.00 | -> | T2/2.00/2.50/1.00 | -> | T17/2.50/3.00/1.00 | -> |
| | T21/3.50/4.00/1.00 | -> | T25/4.00/4.50/1.00 | -> | T19/4.50/5.00/1.00 | -> |
| | T27/5.50/6.00/1.00 | -> | T26/6.00/6.50/1.00 | -> | T29/6.50/7.00/1.00 | -> |
| | T30/7.50/8.00/1.00 | -> | T31/8.00/8.50/1.00 | | | |
| P1 | T3/1.50/2.00/1.00 | -> | T4/2.00/2.50/1.00 | -> | T18/2.50/3.00/1.00 | |
| P2 | T13/1.50/2.50/1.00 | -> | T23/3.50/4.50/1.00 | -> | T28/5.50/6.50/1.00 | |
| P3 | T11/1.50/2.50/1.00 | -> | T22/3.50/4.50/1.00 | | | |
| P4 | T10/1.50/2.50/1.00 | | | | | |
| P5 | T7/1.50/2.50/1.00 | -> | T20/3.50/4.50/1.00 | | | |
| P6 | T6/1.50/2.50/1.00 | | | | | |
| P7 | T15/1.50/2.50/1.00 | -> | T24/3.50/4.50/1.00 | | | |
| P8 | T12/1.50/2.50/1.00 | | | | | |
| P9 | T8/1.50/2.50/1.00 | | | | | |
| P10 | T14/1.50/2.50/1.00 | | | | | |
| P11 | T16/1.50/2.50/1.00 | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B2. A schedule of K1 to S2 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.33/1.00 | -> | T1/0.33/0.67/1.00 | -> | T9/0.67/1.00/1.00 | -> |
| | T5/1.00/1.33/1.00 | -> | T3/1.33/1.67/1.00 | -> | T13/1.67/2.00/1.00 | -> |
| | T11/2.00/2.33/1.00 | -> | T17/2.83/3.17/1.00 | -> | T18/3.17/3.50/1.00 | -> |
| | T25/3.50/3.83/1.00 | -> | T21/3.83/4.17/1.00 | -> | T19/4.17/4.50/1.00 | -> |
| | T22/4.50/4.83/1.00 | -> | T27/4.83/5.17/1.00 | -> | T26/5.17/5.50/1.00 | -> |
| | T29/5.50/5.83/1.00 | -> | T23/5.83/6.17/1.00 | -> | T28/6.17/6.50/1.00 | -> |
| | T30/6.50/6.83/1.00 | -> | T31/6.83/7.17/1.00 | | | |
| P1 | T2/1.33/1.83/1.00 | -> | T10/1.83/2.33/1.00 | | | |
| P2 | T4/1.33/1.83/1.00 | -> | T7/1.83/2.33/1.00 | -> | T20/3.33/3.83/1.00 | |
| P3 | T6/1.33/2.33/1.00 | | | | | |
| P4 | T15/1.33/2.33/1.00 | -> | T24/3.33/4.33/1.00 | | | |
| P5 | T12/1.33/2.33/1.00 | | | | | |
| P6 | T8/1.33/2.33/1.00 | | | | | |
| P7 | T14/1.33/2.33/1.00 | | | | | |
| P8 | T16/1.33/2.33/1.00 | | | | | |
| P9 | | | | | | |
| P10 | | | | | | |
| P11 | | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B3. A schedule of K1 to S3 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.06/1.00 | -> | T1/0.06/0.13/1.00 | -> | T9/0.13/0.19/1.00 | -> |
| | T5/0.19/0.25/1.00 | -> | T3/0.25/0.31/1.00 | -> | T2/0.31/0.38/1.00 | -> |
| | T17/0.38/0.44/1.00 | -> | T4/0.44/0.50/1.00 | -> | T18/0.50/0.56/1.00 | -> |
| | T25/0.56/0.63/1.00 | -> | T13/0.63/0.69/1.00 | -> | T11/0.69/0.75/1.00 | -> |
| | T10/0.75/0.81/1.00 | -> | T21/0.81/0.88/1.00 | -> | T7/0.88/0.94/1.00 | -> |
| | T6/0.94/1.00/1.00 | -> | T19/1.00/1.06/1.00 | -> | T15/1.06/1.13/1.00 | -> |
| | T12/1.13/1.19/1.00 | -> | T22/1.19/1.25/1.00 | -> | T27/1.25/1.31/1.00 | -> |
| | T8/1.31/1.38/1.00 | -> | T20/1.38/1.44/1.00 | -> | T26/1.44/1.50/1.00 | -> |
| | T29/1.50/1.56/1.00 | -> | T14/1.56/1.63/1.00 | -> | T23/1.63/1.69/1.00 | -> |
| | T16/1.69/1.75/1.00 | -> | T24/1.75/1.81/1.00 | -> | T28/1.81/1.88/1.00 | -> |
| | T30/1.88/1.94/1.00 | -> | T31/1.94/2.00/1.00 | | | |
| P1 | | | | | | |
| P2 | | | | | | |
| P3 | | | | | | |
| P4 | | | | | | |
| P5 | | | | | | |
| P6 | | | | | | |
| P7 | | | | | | |
| P8 | | | | | | |
| P9 | | | | | | |
| P10 | | | | | | |
| P11 | | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B4. A schedule of K1 to S4 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/1.00/1.00 | -> | T1/1.00/2.00/1.00 | -> | T2/2.00/3.00/1.00 | -> |
| | T17/3.00/4.00/1.00 | -> | T18/13.00/14.00/1.00 | -> | T25/14.00/15.00/1.00 | -> |
| | T26/24.00/25.00/1.00 | -> | T29/25.00/26.00/1.00 | -> | T30/35.00/36.00/1.00 | -> |
| | T31/36.00/37.00/1.00 | | | | | |
| P1 | T3/2.00/3.00/1.00 | -> | T19/13.00/14.00/1.00 | -> | T27/24.00/25.00/1.00 | |
| P2 | T5/2.00/3.00/1.00 | -> | T21/13.00/14.00/1.00 | -> | T28/24.00/25.00/1.00 | |
| P3 | T9/2.00/3.00/1.00 | -> | T20/13.00/14.00/1.00 | | | |
| P4 | T11/2.00/3.00/1.00 | -> | T22/4.00/5.00/1.00 | -> | T23/13.00/14.00/1.00 | |
| P5 | T4/2.00/3.00/1.00 | -> | T24/13.00/14.00/1.00 | | | |
| P6 | T6/2.00/3.00/1.00 | | | | | |
| P7 | T7/2.00/3.00/1.00 | | | | | |
| P8 | T10/2.00/3.00/1.00 | | | | | |
| P9 | T13/2.00/3.00/1.00 | | | | | |
| P10 | T8/2.00/3.00/1.00 | | | | | |
| P11 | T12/2.00/3.00/1.00 | | | | | |
| P12 | T14/2.00/3.00/1.00 | | | | | |
| P13 | T15/2.00/3.00/1.00 | | | | | |
| P14 | T16/2.00/3.00/1.00 | | | | | |
| P15 | | | | | | |

**Figure B5. A schedule of K1 to S1 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.50/1.00 | -> | T1/0.50/1.00/1.00 | -> | T2/1.00/1.50/1.00 | -> |
| | T17/1.50/2.00/1.00 | -> | T6/2.00/2.50/1.00 | -> | T18/12.00/12.50/1.00 | -> |
| | T25/12.50/13.00/1.00 | -> | T22/13.00/13.50/1.00 | -> | T26/23.00/23.50/1.00 | -> |
| | T29/23.50/24.00/1.00 | -> | T30/34.00/34.50/1.00 | -> | T31/34.50/35.00/1.00 | |
| P1 | T3/1.50/2.00/1.00 | -> | T7/2.00/2.50/1.00 | -> | T20/3.50/4.00/1.00 | -> |
| | T19/12,50/13.00/1.00 | -> | T23/13.00/13.50/1.00 | -> | T28/14.50/15.00/1.00 | -> |
| | T27/23.50/24.00/1.00 | | | | | |
| P2 | T5/1.50/2.50/1.00 | -> | T21/12.50/13.50/1.00 | | | |
| P3 | T9/1.50/2.50/1.00 | -> | T24/12.50/13.50/1.00 | | | |
| P4 | T11/1.50/2.50/1.00 | | | | | |
| P5 | T4/1.50/2.50/1.00 | | | | | |
| P6 | T10/1.50/2.50/1.00 | | | | | |
| P7 | T13/1.50/2.50/1.00 | | | | | |
| P8 | T8/1.50/2.50/1.00 | | | | | |
| P9 | T12/1.50/2.50/1.00 | | | | | |
| P10 | T14/1.50/2.50/1.00 | | | | | |
| P11 | T15/1.50/2.50/1.00 | | | | | |
| P12 | T16/1.50/2.50/1.00 | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B6. A schedule of K1 to S2 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.33/1.00 | -> | T1/0.33/0.67/1.00 | -> | T2/0.67/1.00/1.00 | -> |
| | T17/1.00/1.33/1.00 | -> | T3/1.33/1.67/1.00 | -> | T6/1.67/2.00/1.00 | -> |
| | T18/3.33/3.67/1.00 | -> | T25/3.67/4.00/1.00 | -> | T19/11.83/12.17/1.00 | -> |
| | T20/12.33/12.67/1.00 | -> | T26/12.67/13.00/1.00 | -> | T29/13.00/13.33/1.00 | -> |
| | T27/22.33/22.67/1.00 | -> | T28/23.33/23.67/1.00 | -> | T30/23.67/24.00/1.00 | -> |
| | T31/24.00/24.33/1.00 | | | | | |

| | | | | |
|---|---|---|---|---|
| P1 | T5/1.33/1.83/1.00 | -> | T21/11.83/12.33/1.00 | |
| P2 | T9/1.33/1.83/1.00 | -> | T22/12.33/12.83/1.00 | |
| P3 | T11/1.33/2.33/1.00 | -> | T23/12.33/13.33/1.00 | |
| P4 | T4/1.33/2.33/1.00 | -> | T24/12.33/13.33/1.00 | |
| P5 | T7/1.33/2.33/1.00 | | | |
| P6 | T10/1.33/2.33/1.00 | | | |
| P7 | T13/1.33/2.33/1.00 | | | |
| P8 | T8/1.33/2.33/1.00 | | | |
| P9 | T12/1.33/2.33/1.00 | | | |
| P10 | T14/1.33/2.33/1.00 | | | |
| P11 | T15/1.33/2.33/1.00 | | | |
| P12 | T16/1.33/2.33/1.00 | | | |
| P13 | | | | |
| P14 | | | | |
| P15 | | | | |

**Figure B7. A schedule of K1 to S3 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.06/1.00 | -> | T1/0.06/0.13/1.00 | -> | T2/0.13/0.19/1.00 | -> |
| | T17/0.19/0.25/1.00 | -> | T3/0.25/0.31/1.00 | -> | T5/0.31/0.38/1.00 | -> |
| | T9/0.38/0.44/1.00 | -> | T11/0.44/0.50/1.00 | -> | T4/0.50/0.56/1.00 | -> |
| | T18/0.56/0.63/1.00 | -> | T25/0.63/0.69/1.00 | -> | T6/0.69/0.75/1.00 | -> |
| | T19/0.75/0.81/1.00 | -> | T7/0.81/0.88/1.00 | -> | T10/0.88/0.94/1.00 | -> |
| | T21/0.94/1.00/1.00 | -> | T13/1.00/1.06/1.00 | -> | T8/1.06/1.13/1.00 | -> |
| | T20/1.13/1.19/1.00 | -> | T26/1.19/1.25/1.00 | -> | T29/1.25/1.31/1.00 | -> |
| | T14/1.31/1.38/1.00 | -> | T23/1.38/1.44/1.00 | -> | T15/1.44/1.50/1.00 | -> |
| | T16/1.50/1.56/1.00 | -> | T24/1.56/1.63/1.00 | -> | T28/1.63/1.69/1.00 | -> |
| | T22/3.06/3.13/1.00 | -> | T27/3.13/3.19/1.00 | -> | T30/3.19/3.25/1.00 | -> |
| | T31/3.25/3.31/1.00 | | | | | |
| P1 | T12/1.06/2.06/1.00 | | | | | |
| P2 | | | | | | |
| P3 | | | | | | |
| P4 | | | | | | |
| P5 | | | | | | |
| P6 | | | | | | |
| P7 | | | | | | |
| P8 | | | | | | |
| P9 | | | | | | |
| P10 | | | | | | |
| P11 | | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B8. A schedule of K1 to S4 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/1.00/1.00 | -> | T1/1.00/2.00/1.00 | -> | T2/2.00/3.00/1.00 | -> |
| | T17/3.00/4.00/1.00 | -> | T25/6.00/7.00/1.00 | -> | T29/8.00/9.00/1.00 | -> |
| | T31/10.00/11.00/1.00 | | | | | |
| P1 | T3/2.00/3.00/1.00 | -> | T18/4.00/5.00/1.00 | | | |
| P2 | T5/2.00/3.00/1.00 | -> | T19/4.00/5.00/1.00 | -> | T26/6.00/7.00/1.00 | |
| P3 | T9/2.00/3.00/1.00 | -> | T21/4.00/5.00/1.00 | -> | T27/6.00/7.00/1.00 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | T30/8.00/9.00/1.00 | | | | | |
| P4 | T11/2.00/3.00/1.00 | -> | T22/4.00/5.00/1.00 | | | |
| P5 | T4/2.00/3.00/1.00 | | | | | |
| P6 | T6/2.00/3.00/1.00 | | | | | |
| P7 | T7/2.00/3.00/1.00 | -> | T20/4.00/5.00/1.00 | | | |
| P8 | T10/2.00/3.00/1.00 | | | | | |
| P9 | T13/2.00/3.00/1.00 | -> | T23/4.00/5.00/1.00 | -> | T28/6.00/7.00/1.00 | |
| P10 | T8/2.00/3.00/1.00 | | | | | |
| P11 | T12/2.00/3.00/1.00 | | | | | |
| P12 | T14/2.00/3.00/1.00 | | | | | |
| P13 | T15/2.00/3.00/1.00 | -> | T24/4.00/5.00/1.00 | | | |
| P14 | T16/2.00/3.00/1.00 | | | | | |
| P15 | | | | | | |

**Figure B9. A schedule of K1 to S1 using the HRMS algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.50/1.00 | -> | T1/0.50/1.00/1.00 | -> | T2/1.00/1.50/1.00 | -> |
| | T17/1.50/2.00/1.00 | -> | T6/2.00/2.50/1.00 | -> | T25/5.00/5.50/1.00 | -> |
| | T29/7.50/8.00/1.00 | -> | T31/9.50/10.00/1.00 | | | |
| P1 | T3/1.50/2.00/1.00 | -> | T7/2.00/2.50/1.00 | -> | T18/3.50/4.00/1.00 | -> |
| | T20/4.00/4.50/1.00 | | | | | |
| P2 | T5/1.50/2.50/1.00 | -> | T19/3.50/4.50/1.00 | -> | T26/5.50/6.50/1.00 | |
| P3 | T9/1.50/2.50/1.00 | -> | T21/3.50/4.50/1.00 | -> | T27/5.50/6.50/1.00 | |
| | T30/7.50/8.50/1.00 | | | | | |
| P4 | T11/1.50/2.50/1.00 | -> | T22/3.50/4.50/1.00 | | | |
| P5 | T4/1.50/2.50/1.00 | | | | | |
| P6 | T10/1.50/2.50/1.00 | | | | | |
| P7 | T13/1.50/2.50/1.00 | -> | T23/3.50/4.50/1.00 | -> | T28/5.50/6.50/1.00 | |
| P8 | T8/1.50/2.50/1.00 | | | | | |
| P9 | T12/1.50/2.50/1.00 | | | | | |
| P10 | T14/1.50/2.50/1.00 | | | | | |
| P11 | T15/1.50/2.50/1.00 | -> | T24/3.50/4.50/1.00 | | | |
| P12 | T16/1.50/2.50/1.00 | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B10. A schedule of K1 to S2 using the HRMS algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.33/1.00 | -> | T1/0.33/0.67/1.00 | -> | T2/0.67/1.00/1.00 | -> |
| | T17/1.00/1.33/1.00 | -> | T3/1.33/1.67/1.00 | -> | T6/1.67/2.00/1.00 | -> |
| | T18/3.33/3.67/1.00 | -> | T25/3.67/4.00/1.00 | -> | T29/6.83/7.17/1.00 | -> |
| | T31/8.83/9.17/1.00 | | | | | |
| P1 | T5/1.33/1.83/1.00 | -> | T19/3.00/3.50/1.00 | -> | T26/5.33/5.83/1.00 | |
| P2 | T9/1.33/1.83/1.00 | -> | T21/3.33/3.83/1.00 | -> | T27/5.33/5.83/1.00 | -> |
| | T30/7.33/7.83/1.00 | | | | | |
| P3 | T11/1.33/2.33/1.00 | -> | T22/3.33/4.33/1.00 | | | |
| P4 | T4/1.33/2.33/1.00 | | | | | |
| P5 | T7/1.33/2.33/1.00 | -> | T20/3.33/4.33/1.00 | | | |
| P6 | T10/1.33/2.33/1.00 | | | | | |
| P7 | T13/1.33/2.33/1.00 | -> | T23/3.33/4.33/1.00 | -> | T28/5.33/6.33/1.00 | |
| P8 | T8/1.33/2.33/1.00 | | | | | |

| | | | | |
|---|---|---|---|---|
| P9 | T12/1.33/2.33/1.00 | | | |
| P10 | T14/1.33/2.33/1.00 | | | |
| P11 | T15/1.33/2.33/1.00 | -> | T24/3.33/4.33/1.00 | |
| P12 | T16/1.33/2.33/1.00 | | | |
| P13 | | | | |
| P14 | | | | |
| P15 | | | | |

**Figure B11. A schedule of K1 to S3 using the HRMS algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/0.06/1.00 | -> | T1/0.06/0.13/1.00 | -> | T2/0.13/0.19/1.00 | -> |
| | T17/0.19/0.25/1.00 | -> | T3/0.25/0.31/1.00 | -> | T5/0.31/0.38/1.00 | -> |
| | T9/0.38/0.44/1.00 | -> | T11/0.44/0.50/1.00 | -> | T4/0.50/0.56/1.00 | -> |
| | T18/0.56/0.63/1.00 | -> | T25/0.63/0.69/1.00 | -> | T6/0.69/0.75/1.00 | -> |
| | T19/0.75/0.81/1.00 | -> | T7/0.81/0.88/1.00 | -> | T10/0.88/0.94/1.00 | -> |
| | T21/0.94/1.00/1.00 | -> | T13/1.00/1.06/1.00 | -> | T8/1.06/1.13/1.00 | -> |
| | T20/1.13/1.19/1.00 | -> | T26/1.19/1.25/1.00 | -> | T29/1.25/1.31/1.00 | -> |
| | T14/1.31/1.38/1.00 | -> | T23/1.38/1.44/1.00 | -> | T15/1.44/1.50/1.00 | -> |
| | T16/1.50/1.56/1.00 | -> | T24/1.56/1.63/1.00 | -> | T28/1.63/1.69/1.00 | -> |
| | T22/3.06/3.13/1.00 | -> | T27/3.13/3.19/1.00 | -> | T30/3.19/3.25/1.00 | -> |
| | T31/3.25/3.31/1.00 | | | | | |
| P1 | T12/1.06/2.06/1.00 | | | | | |
| P2 | | | | | | |
| P3 | | | | | | |
| P4 | | | | | | |
| P5 | | | | | | |
| P6 | | | | | | |
| P7 | | | | | | |
| P8 | | | | | | |
| P9 | | | | | | |
| P10 | | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B12. A schedule of K1 to S4 using the HRMS algorithm.**

## B2 Schedules of Application K30

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/30.00/1.00 | -> | T1/30.00/60.00/1.00 | -> | T17/62.00/92.00/1.00 | -> |
| | T25/93.00/123.00/1.00 | -> | T29/124.00/154.00/1.00 | -> | T31/155.00/185.00/1.00 | |
| P1 | T9/31.00/61.00/1.00 | -> | T21/62.00/92.00/1.00 | -> | T27/93.00/123.00/1.00 | -> |
| | T30/124.00/154.00/1.00 | | | | | |
| P2 | T5/31.00/61.00/1.00 | -> | T19/62.00/92.00/1.00 | -> | T26/93.00/123.00/1.00 | |
| P3 | T3/31.00/61.00/1.00 | -> | T18/62.00/92.00/1.00 | | | |
| P4 | T2/31.00/61.00/1.00 | | | | | |
| P5 | T4/31.00/61.00/1.00 | | | | | |
| P6 | T13/31.00/61.00/1.00 | -> | T23/62.00/92.00/1.00 | -> | T28/93.00/123.00/1.00 | |
| P7 | T11/31.00/61.00/1.00 | -> | T22/62.00/92.00/1.00 | | | |
| P8 | T10/31.00/61.00/1.00 | | | | | |
| P9 | T7/31.00/61.00/1.00 | -> | T20/62.00/92.00/1.00 | | | |
| P10 | T6/31.00/61.00/1.00 | | | | | |
| P11 | T15/31.00/61.00/1.00 | -> | T24/62.00/92.00/1.00 | | | |
| P12 | T12/31.00/61.00/1.00 | | | | | |
| P13 | T8/31.00/61.00/1.00 | | | | | |
| P14 | T14/31.00/61.00/1.00 | | | | | |
| P15 | T16/31.00/61.00/1.00 | | | | | |

**Figure B13. A schedule of K30 to S1 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/15.00/1.00 | -> | T1/15.00/30.00/1.00 | -> | T5/30.00/45.00/1.00 | -> |
| | T17/47.00/62.00/1.00 | -> | T19/62.00/77.00/1.00 | -> | T26/78.00/93.00/1.00 | -> |
| | T28/93.00/108.00/1.00 | -> | T29/108.00/123.00/1.00 | -> | T31/125.00/140.00/1.00 | |
| P1 | T9/16.00/31.00/1.00 | -> | T3/31.00/46.00/1.00 | -> | T18/47.00/62.00/1.00 | -> |
| | T25/72.00/87.00/1.00 | -> | T27/87.00/102.00/1.00 | -> | T30/109.00/124.00/1.00 | |
| P2 | T2/16.00/46.00/1.00 | | | | | |
| P3 | T4/16.00/46.00/1.00 | | | | | |
| P4 | T13/16.00/46.00/1.00 | -> | T23/47.00/77.00/1.00 | | | |
| P5 | T11/16.00/46.00/1.00 | -> | T22/47.00/77.00/1.00 | | | |
| P6 | T10/16.00/46.00/1.00 | -> | T21/46.00/76.00/1.00 | | | |
| P7 | T7/16.00/46.00/1.00 | -> | T20/47.00/77.00/1.00 | | | |
| P8 | T6/16.00/46.00/1.00 | | | | | |
| P9 | T15/16.00/46.00/1.00 | -> | T24/47.00/77.00/1.00 | | | |
| P10 | T8/16.00/46.00/1.00 | | | | | |
| P11 | T12/16.00/46.00/1.00 | | | | | |
| P12 | T14/16.00/46.00/1.00 | | | | | |
| P13 | T16/16.00/46.00/1.00 | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B14. A schedule of K30 to S2 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/10.00/1.00 | -> | T1/10.00/20.00/1.00 | -> | T3/20.00/30.00/1.00 | -> |
| | T2/30.00/40.00/1.00 | -> | T17/40.00/50.00/1.00 | -> | T19/50.00/60.00/1.00 | -> |
| | T22/60.00/70.00/1.00 | -> | T26/72.00/82.00/1.00 | -> | T28/82.00/92.00/1.00 | -> |

| | | | | | | |
|---|---|---|---|---|---|---|
| | T30/96.00/106.00/1.00 | -> | T31/112.00/122.00/1.00 | | | |
| P1 | T9/11.00/26.00/1.00 | -> | T4/26.00/41.00/1.00 | -> | T18/41.00/56.00/1.00 | -> |
| | T20/56.00/71.00/1.00 | -> | T27/71.00/86.00/1.00 | | | |
| P2 | T5/11.00/26.00/1.00 | -> | T13/26.00/41.00/1.00 | -> | T21/42.00/57.00/1.00 | -> |
| | T23/57.00/72.00/1.00 | -> | T25/72.00/87.00/1.00 | -> | T29/87.00/102.00/1.00 | |
| P3 | T11/11.00/41.00/1.00 | | | | | |
| P4 | T10/11.00/41.00/1.00 | | | | | |
| P5 | T7/11.00/41.00/1.00 | | | | | |
| P6 | T6/11.00/41.00/1.00 | | | | | |
| P7 | T15/11.00/41.00/1.00 | -> | T24/42.00/72.00/1.00 | | | |
| P8 | T12/11.00/41.00/1.00 | | | | | |
| P9 | T8/11.00/41.00/1.00 | | | | | |
| P10 | T14/11.00/41.00/1.00 | | | | | |
| P11 | T16/11.00/41.00/1.00 | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B15. A schedule of K30 to S3 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/1.88/1.00 | -> | T1/1.88/3.75/1.00 | -> | T9/3.75/5.63/1.00 | -> |
| | T5/5.63/7.50/1.00 | -> | T3/7.50/9.38/1.00 | -> | T2/9.38/11.25/1.00 | -> |
| | T13/11.25/13.12/1.00 | -> | T11/13.12/15.00/1.00 | -> | T10/15.00/16.88/1.00 | -> |
| | T7/16.88/18.75/1.00 | -> | T6/18.75/20.62/1.00 | -> | T4/20.62/22.50/1.00 | -> |
| | T15/22.50/24.38/1.00 | -> | T8/24.38/26.25/1.00 | -> | T14/26.25/28.12/1.00 | -> |
| | T12/28.12/30.00/1.00 | -> | T18/30.00/31.88/1.00 | -> | T17/31.88/33.75/1.00 | -> |
| | T22/33.75/35.62/1.00 | -> | T19/35.62/37.50/1.00 | -> | T23/37.50/39.38/1.00 | -> |
| | T20/39.38/41.25/1.00 | -> | T21/41.25/43.12/1.00 | -> | T24/43.12/45.00/1.00 | -> |
| | T27/45.00/46.88/1.00 | -> | T25/46.88/48.75/1.00 | -> | T26/48.75/50.62/1.00 | -> |
| | T28/50.62/52.50/1.00 | -> | T29/52.50/54.38/1.00 | -> | T30/54.38/56.25/1.00 | -> |
| | T31/56.25/58.12/1.00 | | | | | |
| P1 | T16/2.88/32.88/1.00 | | | | | |
| P2 | | | | | | |
| P3 | | | | | | |
| P4 | | | | | | |
| P5 | | | | | | |
| P6 | | | | | | |
| P7 | | | | | | |
| P8 | | | | | | |
| P9 | | | | | | |
| P10 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B16. A schedule of K30 to S4 using the HDSC algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/30.00/1.00 | -> | T1/30.00/60.00/1.00 | -> | T17/62.00/92.00/1.00 | -> |
| | T25/102.00/132.00/1.00 | -> | T29/142.00/172.00/1.00 | -> | T31/182.00/212.00/1.00 | |
| P1 | T2/31.00/61.00/1.00 | -> | T18/71.00/101.00/1.00 | -> | T26/111.00/141.00/1.00 | -> |
| | T30/151.00/181.00/1.00 | | | | | |
| P2 | T3/31.00/61.00/1.00 | -> | T19/71.00/101.00/1.00 | -> | T27/111.00/141.00/1.00 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| P3 | T5/31.00/61.00/1.00 | -> | T21/71.00/101.00/1.00 | -> | T28/111.00/141.00/1.00 | |
| P4 | T9/31.00/61.00/1.00 | -> | T20/71.00/101.00/1.00 | | | |
| P5 | T11/31.00/61.00/1.00 | -> | T22/62.00/92.00/1.00 | | | |
| P6 | T4/31.00/61.00/1.00 | -> | T23/71.00/101.00/1.00 | | | |
| P7 | T6/31.00/61.00/1.00 | -> | T24/71.00/101.00/1.00 | | | |
| P8 | T7/31.00/61.00/1.00 | | | | | |
| P9 | T10/31.00/61.00/1.00 | | | | | |
| P10 | T13/31.00/61.00/1.00 | | | | | |
| P11 | T8/31.00/61.00/1.00 | | | | | |
| P12 | T12/31.00/61.00/1.00 | | | | | |
| P13 | T14/31.00/61.00/1.00 | | | | | |
| P14 | T15/31.00/61.00/1.00 | | | | | |
| P15 | T16/31.00/61.00/1.00 | | | | | |

**Figure B17. A schedule of K30 to S1 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/15.00/1.00 | -> | T1/15.00/30.00/1.00 | -> | T17/32.00/47.00/1.00 | -> |
| | T18/56.00/71.00/1.00 | -> | T25/71.00/86.00/1.00 | -> | T26/87.00/102.00/1.00 | -> |
| | T29/102.00/117.00/1.00 | -> | T30/127.00/142.00/1.00 | -> | T31/142.00/157.00/1.00 | |
| P1 | T2/16.00/31.00/1.00 | -> | T7/31.00/46.00/1.00 | -> | T19/56.00/71.00/1.00 | -> |
| | T23/71.00/86.00/1.00 | -> | T27/96.00/111.00/1.00 | | | |
| P2 | T3/16.00/46.00/1.00 | -> | T21/56.00/86.00/1.00 | -> | T28/96.00/126.00/1.00 | |
| P3 | T5/16.00/46.00/1.00 | -> | T20/56.00/86.00/1.00 | | | |
| P4 | T9/16.00/46.00/1.00 | -> | T22/56.00/86.00/1.00 | | | |
| P5 | T11/16.00/46.00/1.00 | -> | T24/56.00/86.00/1.00 | | | |
| P6 | T4/16.00/46.00/1.00 | | | | | |
| P7 | T6/16.00/46.00/1.00 | | | | | |
| P8 | T10/16.00/46.00/1.00 | | | | | |
| P9 | T13/16.00/46.00/1.00 | | | | | |
| P10 | T8/16.00/46.00/1.00 | | | | | |
| P11 | T12/16.00/46.00/1.00 | | | | | |
| P12 | T14/16.00/46.00/1.00 | | | | | |
| P13 | T15/16.00/46.00/1.00 | | | | | |
| P14 | T16/16.00/46.00/1.00 | | | | | |
| P15 | | | | | | |

**Figure B18. A schedule of K30 to S2 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/10.00/1.00 | -> | T1/10.00/20.00/1.00 | -> | T7/20.00/30.00/1.00 | -> |
| | T13/30.00/40.00/1.00 | -> | T17/40.00/50.00/1.00 | -> | T19/51.00/61.00/1.00 | -> |
| | T23/61.00/71.00/1.00 | -> | T25/71.00/81.00/1.00 | -> | T27/82.00/92.00/1.00 | -> |
| | T30/98.00/108.00/1.00 | -> | T31/116.00/126.00/1.00 | | | |
| P1 | T2/11.00/26.00/1.00 | -> | T11/26.00/41.00/1.00 | -> | T21/51.00/66.00/1.00 | -> |
| | T26/73.00/88.00/1.00 | -> | T29/91.00/106.00/1.00 | | | |
| P2 | T3/11.00/26.00/1.00 | -> | T8/26.00/41.00/1.00 | -> | T18/42.00/57.00/1.00 | -> |
| | T28/82.00/97.00/1.00 | | | | | |
| P3 | T5/11.00/41.00/1.00 | -> | T22/51.00/81.00/1.00 | | | |
| P4 | T9/11.00/41.00/1.00 | -> | T20/42.00/72.00/1.00 | | | |
| P5 | T6/11.00/41.00/1.00 | -> | T24/51.00/81.00/1.00 | | | |
| P6 | T4/11.00/41.00/1.00 | | | | | |
| P7 | T10/11.00/41.00/1.00 | | | | | |

| | | |
|---|---|---|
| P8 | T12/11.00/41.00/1.00 | |
| P9 | T14/11.00/41.00/1.00 | |
| P10 | T15/11.00/41.00/1.00 | |
| P11 | T16/11.00/41.00/1.00 | |
| P12 | | |
| P13 | | |
| P14 | | |
| P15 | | |

**Figure B19. A schedule of K30 to S3 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/1.88/1.00 | -> | T1/1.88/3.75/1.00 | -> | T11/3.75/5.63/1.00 | -> |
| | T4/5.63/7.50/1.00 | -> | T6/7.50/9.38/1.00 | -> | T7/9.38/11.25/1.00 | -> |
| | T10/11.25/13.12/1.00 | -> | T13/13.12/15.00/1.00 | -> | T14/15.00/16.88/1.00 | -> |
| | T8/16.88/18.75/1.00 | -> | T12/18.75/20.62/1.00 | -> | T15/20.62/22.50/1.00 | -> |
| | T23/22.50/24.38/1.00 | -> | T16/24.38/26.25/1.00 | -> | T24/26.25/28.12/1.00 | -> |
| | T22/28.12/30.00/1.00 | -> | T20/30.00/31.88/1.00 | -> | T28/31.88/33.75/1.00 | -> |
| | T17/33.88/35.75/1.00 | -> | T18/42.88/44.75/1.00 | -> | T25/44.75/46.62/1.00 | -> |
| | T27/82.88/84.75/1.00 | -> | T30/84.75/86.62/1.00 | -> | T29/103.88/105.75/1.00 | -> |
| | T31/105.75/107.62/1.00 | | | | | |
| P1 | T2/2.88/32.88/1.00 | -> | T19/42.88/72.88/1.00 | -> | T26/72.88/102.88/1.00 | |
| P2 | T3/2.88/32.88/1.00 | -> | T21/42.88/72.88/1.00 | | | |
| P3 | T5/2.88/32.88/1.00 | | | | | |
| P4 | T9/2.88/32.88/1.00 | | | | | |
| P5 | | | | | | |
| P6 | | | | | | |
| P7 | | | | | | |
| P8 | | | | | | |
| P9 | | | | | | |
| P10 | | | | | | |
| P11 | | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B20. A schedule of K30 to S4 using the HMD algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/30.00/1.00 | -> | T1/30.00/60.00/1.00 | -> | T17/62.00/92.00/1.00 | -> |
| | T25/93.00/123.00/1.00 | -> | T29/124.00/154.00/1.00 | -> | T31/155.00/185.00/1.00 | |
| P1 | T2/31.00/61.00/1.00 | | | | | |
| P2 | T3/31.00/61.00/1.00 | -> | T18/62.00/92.00/1.00 | | | |
| P3 | T5/31.00/61.00/1.00 | -> | T19/62.00/92.00/1.00 | -> | T26/93.00/123.00/1.00 | |
| P4 | T9/31.00/61.00/1.00 | -> | T21/62.00/92.00/1.00 | -> | T27/93.00/123.00/1.00 | -> |
| | T30/124.00/154.00/1.00 | | | | | |
| P5 | T11/31.00/61.00/1.00 | -> | T22/62.00/92.00/1.00 | | | |
| P6 | T4/31.00/61.00/1.00 | | | | | |
| P7 | T6/31.00/61.00/1.00 | | | | | |
| P8 | T7/31.00/61.00/1.00 | -> | T20/62.00/92.00/1.00 | | | |
| P9 | T10/31.00/61.00/1.00 | | | | | |
| P10 | T13/31.00/61.00/1.00 | -> | T23/62.00/92.00/1.00 | -> | T28/93.00/123.00/1.00 | |
| P11 | T8/31.00/61.00/1.00 | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| P12 | T12/31.00/61.00/1.00 | | | | | |
| P13 | T14/31.00/61.00/1.00 | | | | | |
| P14 | T15/31.00/61.00/1.00 | -> | T24/62.00/92.00/1.00 | | | |
| P15 | T16/31.00/61.00/1.00 | | | | | |

**Figure B21. A schedule of K30 to S1 using the HRMS algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/15.00/1.00 | -> | T1/15.00/30.00/1.00 | -> | T17/32.00/47.00/1.00 | -> |
| | T18/56.00/71.00/1.00 | -> | T25/71.00/86.00/1.00 | -> | T26/87.00/102.00/1.00 | -> |
| | T29/102.00/117.00/1.00 | -> | T31/125.00/140.00/1.00 | | | |
| P1 | T2/16.00/31.00/1.00 | -> | T7/31.00/46.00/1.00 | -> | T19/56.00/71.00/1.00 | -> |
| | T27/87.00/102.00/1.00 | -> | T30/109.00/124.00/1.00 | | | |
| P2 | T3/16.00/46.00/1.00 | -> | T20/56.00/86.00/1.00 | | | |
| P3 | T5/16.00/46.00/1.00 | | | | | |
| P4 | T9/16.00/46.00/1.00 | -> | T21/47.00/77.00/1.00 | | | |
| P5 | T11/16.00/46.00/1.00 | -> | T22/47.00/77.00/1.00 | | | |
| P6 | T4/16.00/46.00/1.00 | | | | | |
| P7 | T6/16.00/46.00/1.00 | | | | | |
| P8 | T10/16.00/46.00/1.00 | | | | | |
| P9 | T13/16.00/46.00/1.00 | -> | T23/47.00/77.00/1.00 | -> | T28/78.00/108.00/1.00 | |
| P10 | T8/16.00/46.00/1.00 | | | | | |
| P11 | T12/16.00/46.00/1.00 | | | | | |
| P12 | T14/16.00/46.00/1.00 | | | | | |
| P13 | T15/16.00/46.00/1.00 | -> | T24/47.00/77.00/1.00 | | | |
| P14 | T16/16.00/46.00/1.00 | | | | | |
| P15 | | | | | | |

**Figure B22. A schedule of K30 to S2 using the HRMS algorithm**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/10.00/1.00 | -> | T1/10.00/20.00/1.00 | -> | T7/20.00/30.00/1.00 | -> |
| | T13/30.00/40.00/1.00 | -> | T17/40.00/50.00/1.00 | -> | T19/51.00/61.00/1.00 | -> |
| | T23/61.00/71.00/1.00 | -> | T27/82.00/92.00/1.00 | -> | T30/97.00/107.00/1.00 | -> |
| | T31/113.00/123.00/1.00 | | | | | |
| P1 | T2/11.00/26.00/1.00 | -> | T11/26.00/41.00/1.00 | -> | T21/51.00/66.00/1.00 | -> |
| | T26/73.00/88.00/1.00 | -> | T29/88.00/103.00/1.00 | | | |
| P2 | T3/11.00/26.00/1.00 | -> | T8/26.00/41.00/1.00 | -> | T18/42.00/57.00/1.00 | -> |
| | T25/60.00/75.00/1.00 | -> | T28/81.00/96.00/1.00 | | | |
| P3 | T5/11.00/41.00/1.00 | -> | T22/51.00/81.00/1.00 | | | |
| P4 | T9/11.00/41.00/1.00 | -> | T20/42.00/72.00/1.00 | | | |
| P5 | T6/11.00/41.00/1.00 | | | | | |
| P6 | T4/11.00/41.00/1.00 | | | | | |
| P7 | T10/11.00/41.00/1.00 | | | | | |
| P8 | T12/11.00/41.00/1.00 | | | | | |
| P9 | T14/11.00/41.00/1.00 | | | | | |
| P10 | T15/11.00/41.00/1.00 | -> | T24/42.00/72.00/1.00 | | | |
| P11 | T16/11.00/41.00/1.00 | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B23. A schedule of K30 to S3 using the HRMS algorithm.**

| | | | | | | |
|---|---|---|---|---|---|---|
| P0 | T0/0.00/1.88/1.00 | -> | T1/1.88/3.75/1.00 | -> | T11/3.75/5.63/1.00 | -> |
| | T4/5.63/7.50/1.00 | -> | T6/7.50/9.38/1.00 | -> | T7/9.38/11.25/1.00 | -> |
| | T10/11.25/13.12/1.00 | -> | T13/13.12/15.00/1.00 | -> | T14/15.00/16.88/1.00 | -> |
| | T8/16.88/18.75/1.00 | -> | T12/18.75/20.62/1.00 | -> | T15/20.62/22.50/1.00 | -> |
| | T23/22.50/24.38/1.00 | -> | T16/24.38/26.25/1.00 | -> | T24/26.25/28.12/1.00 | -> |
| | T22/28.12/30.00/1.00 | -> | T20/30.00/31.88/1.00 | -> | T28/31.88/33.75/1.00 | -> |
| | T17/33.88/35.75/1.00 | -> | T18/42.88/44.75/1.00 | -> | T25/44.75/46.62/1.00 | -> |
| | T27/72.88/74.75/1.00 | -> | T30/74.75/76.62/1.00 | -> | T29/93.88/95.75/1.00 | -> |
| | T31/95.75/97.62/1.00 | | | | | |
| P1 | T2/2.88/32.88/1.00 | | | | | |
| P2 | T3/2.88/32.88/1.00 | | | | | |
| P3 | T5/2.88/32.88/1.00 | -> | T19/32.88/62.88/1.00 | -> | T26/62.88/92.88/1.00 | |
| P4 | T9/2.88/32.88/1.00 | -> | T21/32.88/62.88/1.00 | | | |
| P5 | | | | | | |
| P6 | | | | | | |
| P7 | | | | | | |
| P8 | | | | | | |
| P9 | | | | | | |
| P10 | | | | | | |
| P11 | | | | | | |
| P12 | | | | | | |
| P13 | | | | | | |
| P14 | | | | | | |
| P15 | | | | | | |

**Figure B24. A schedule of K30 to S4 using the HRMS algorithm**

# Appendix C: Publication List

1] Huiwei Guan, Chi-Kwong Li, Wai-Yip Chan, "A parallel Implementation of BP Neural Network on a Multiple Processor System", Proceedings 1995 International Symposium on Artificial Neural Networks (ISANN'95), pp. E2-19-24. 1995, Tai Wan.

[2] Wai-Yip Chan, Chi-Kwong Li, Huiwei Guan, "An Illiac Implementation of BP Neural Network on a Multiple Processor System," Proceedings International Conference on Neural Information Processing (ICONIP'96), 1996, vol. 2, pp. 1007-1011. 1996, Hong Kong

[3] Wai-Yip Chan and Chi-Kwong Li, "An aggressive Parallel Tasks Scheduling Algorithm: Relative Mobility Scheduling Algorithm (RMS)," IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM'97), vol. 2, pp. 946-949, August 1997, Canada.

[4] Wai-Yip Chan and Chi-Kwong Li, "Heterogeneous Dominant Sequence Cluster (HDSC): A Low Complexity Heterogeneous Scheduling Algorithm," IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM'97), vol. 2, pp. 956-959, August 1997, Canada.

[5] Wai-Yip Chan and Chi-Kwong Li, "A New Technique of List Scheduling Algorithm for Heterogeneous Processors Systems," Proceedings ISCA International Conference on Computer Applications in Industry and Engineering (CAINE'97), 1997, USA.

[6] Wai-Yip Chan and Chi-Kwong Li, "The Relative Mobility Scheduling Algorithm (RMS)," Proceedings ISCA (CAINE'97), 1997, USA.

[7] Wai-Yip Chan and Chi-Kwong Li, "Scheduling Tasks in DAG to Heterogeneous Processors System," Proceedings 6th Euromicro Workshop on Parallel and Distributed Processing (EuroMicro'98), pp. 27-31, 1998, Spain.

[8] Chi-Kwong Li, Wai-Yip Chan, Kam-Tai Yam and Wai-Kwong Cheuk, "An Introduction to VCC: An Integrated Visual Programming Environment for Contractual Computing," International Conference on Computer Systems and Applications (IASTED'98), April 1998, Jordan.

[9] Wai-Kin Lam, Wai-Yip Chan, Wai-Kong Cheuk and Chi-Wai Yuen, "An Integrated Environment for Contractual Computing -
Version 1.0L", CD-ROM media, December, 1997

[10] Wai-Kin Lam, Wai-yip Chan, Wai-Kong Cheuk and Chi-Wai Yuen, "An Integrated Environment for Contractual Computing 98' distribution - Version 2.0L", CD-ROM media, June, 1998

[11] Chi-Kwong Li, Wai-Yip Chan and Wai-Kong Cheuk, "Solving Inter-application Dependency with Contractor Oriented Task Scheduling Algorithms", International Conference on Information, Communications and Signal Processing (ICICS'99), 1999.

[12] Wai-Kong Cheuk, Chi-Kwong Li, and Wai-Yip Chan, "Contractual Operating System - Design and Implementation", IEEE International Workshop on Cluster Computing 1999 (IWCC'99), 1999.