STATIC SCHEDULER
FOR HARD REAL-TIME TASKS
ON MULTIPROCESSOR SYSTEMS

by

Tzu-Chiang Chang

September 1992

Thesis Advisor: Man-Tak Shing

Approved for public release; distribution is unlimited.
STATIC SCHEDULER FOR HARD REAL-TIME TASKS ON MULTIPROCESSOR SYSTEMS

Tzu-Chiang Chang

Task scheduling is one of the most important issues in a hard real-time system, because it is the schedule that ensures the tasks meet their deadlines and precedence constraints. Given a set of hard real-time tasks, to determine whether a feasible schedule exists such that the timing constraints and precedence constraints of the tasks are satisfied, and to produce such a schedule if one exists are the purposes of a static scheduler. The previous work done for the static scheduler in the computer aided prototyping system (CAPS) was mainly for the single processor environment.

The major work of this thesis is to develop several algorithms for scheduling hard real-time tasks on multiprocessor systems so that the associated timing and precedence constraints, as well as the communication requirements are met under the worst-case situation.
STATIC SCHEDULER
FOR HARD REAL-TIME TASKS
ON MULTIPROCESSOR SYSTEMS

by
Tzu-Chiang Chang
Capt. R.O.C. (Taiwan) Army
B.S. of Applied Math. in Computer Science
Chung Cheng Institute of Technology
the Republic Of China, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SYSTEM ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1992

Author: Tzu-Chiang Chang

Approved By: Man-Tak Shing, Thesis Advisor

Prof. Jeffrey B. Knorr, Chairman,
Electronic Warfare Academic Group
ABSTRACT

Task scheduling is one of the most important issues in a hard real-time system, because it is the schedule that ensures the tasks meet their deadlines and precedence constraints. Given a set of hard real-time tasks, to determine whether a feasible schedule exists such that the timing constraints and precedence constraints of the tasks are satisfied, and to produce such a schedule if one exists are the purposes of a static scheduler. The previous work done for the static scheduler in the computer aided prototyping system (CAPS) was mainly for the single processor environment.

The major work of this thesis is to develop several algorithms for scheduling hard real-time tasks on multiprocessor systems so that the associated timing and precedence constraints, as well as the communication requirements are met under the worst-case situation.
# TABLE OF CONTENTS

I. INTRODUCTION ..................................................................................................................... 1  
A. BACKGROUND ....................................................................................................................... 1  
   1. Hard Real-Time System .................................................................................................... 1  
   2. Computer Aided Prototyping System (CAPS) .............................................................. 3  
   3. Prototype System Description Language (PSDL) .......................................................... 5  
B. SCHEDULING PROBLEM ..................................................................................................... 6  
   1. Nature of the Task (Operator) ......................................................................................... 6  
   2. Constraints and Requirements ...................................................................................... 8  
   3. The Usage of the Schedule ............................................................................................ 11  
C. OBJECTIVES ....................................................................................................................... 13  
D. ORGANIZATION ................................................................................................................ 14  

II. SURVEY OF PREVIOUS WORK .......................................................................................... 15  
A. DEFINITION OF TERMS ................................................................................................. 15  
   1. HBL .............................................................................................................................. 15  
   2. Instance ....................................................................................................................... 15  
   3. Tardiness and Cost ........................................................................................................ 16  
   4. Latency ....................................................................................................................... 16  
   5. MET ............................................................................................................................. 16  
   6. Finish_Within ............................................................................................................... 17  
   7. Pipeline ....................................................................................................................... 17  
   8. Legal Solution and Feasible Solution ........................................................................... 17  
   9. Optimal Solution and Approximate Solution .............................................................. 18  
B. PREVIOUS RESEARCH ...................................................................................................... 18  
   1. Static Scheduling for Uniprocessor ............................................................................... 18  
   2. Static Scheduling for multiprocessor ........................................................................... 20
I. INTRODUCTION

A. BACKGROUND

This thesis addresses the development of hard real-time systems using the computer aided prototyping system (CAPS) and the prototype system description language (PSDL). The following sections introduce them in detail.

1. Hard Real-Time System

In (soft) real-time systems, tasks (activities, operations, jobs) are performed by the system as fast as possible, but they are not constrained to finish by a specific time. In a particularly restricted form of real-time systems, tasks have to be performed not only correctly, but also in a timely fashion; otherwise, there might be severe consequences. In other words, this kind of the real-time system is characterized by the fact that severe consequences will result if logical as well as timing correctness properties of the system are not satisfied. Such a real-time system is often referred to as a hard real-time system, as opposed to a soft real-time system. Therefore, a hard real-time system can be defined as a system in which the correctness of the system depends not only on the logical results of the computation, but also on the time at which the results are produced. If the results are not produced in a timely manner, disastrous consequences may occur. Most of the hard real-time systems are special-purpose and complex, requiring a high degree of fault tolerance, and are typically embedded in a larger system [LEV91].

Typically, a hard real-time system consists of a controlling system and a controlled system. For example, in an automated factory, the controlled system is the factory floor with its robots, assembling stations, and the assembled parts, while the controlling system includes the computer and human interfaces that manage and
coordinate the tasks on the factory floor. Thus, the controlled system can be viewed as the environment with which the computer interacts.

The controlling system interacts with its environment based on the information available about the environment, say, from various sensors attached to it. It is imperative that the state of the environment, as perceived by the controlling system, is consistent with the actual state of the environment. Otherwise, the effects of the controlling system's tasks may be disastrous. Hence, periodic monitoring of the environment as well as timely processing of the sensed information is necessary [STR88].

Timing correctness requirements arise in a hard real-time system because of the physical impact of the controlling systems' tasks upon its environment. For example, if the computer controlling a robot does not stop or turn it on time, the robot might collide with another object on the factory floor. Needless to say, such a mishap can result in a major catastrophe.

A number of new and sophisticated hard real-time applications are currently being contemplated by governments and industries around the world. In addition to automated factories, application can be found in avionics, undersea exploration, process control, flight control, robot and vision systems, as well as military applications such as C3I systems, strategic defense initiative (SDI) systems, and so forth.

In summary, the difference between the hard real-time system and the traditional system, where there is a separation between the correctness and performance, is that the correctness and performance are very tightly interrelated in the hard real-time system. Thus, hard real-time systems solve the problem of missing deadlines in ways specific to the requirements of the target application. However, it
should be said that the sooner a system determines that a deadline is going to be missed, the more flexibility it will have in dealing with the exception [STR88].

Depending on where the application is used, hard real-time systems can be categorized as either centralized or distributed systems.

- **In centralized systems**, the processors are located at a single point and the inter-processor communication cost is negligible compared to the processor execution cost. A multiprocessor system with shared memory or a system using a single processor is an example of such systems.

- **In distributed systems**, the processors are distributed at different points and the inter-processor communication cost is not negligible compared to the processor execution cost. A local area computer network (LAN) is an example of such systems. The inter-processor communication cost is an important factor which must be explicitly taken into account in scheduling problems of distributed systems.

2. **Computer Aided Prototyping System (CAPS)**

The **computer aided prototyping system** (CAPS), currently being developed at the Naval Postgraduate School, is designed to improve software technology which helps the software engineers design hard real-time software systems, automatically construct a real-time schedule, and automatically generate an executable Ada prototype of the proposed system from the PSDL specification. The Ada prototype is a combination of CAPS-generated Ada programs and reusable atomic Ada components.

CAPS also supports system management and helps control a system's evolution [LUQ89]. This support helps designers give timely responses to modification requests and helps protect the system's integrity as it evolves.
extending its life. The CAPS consists of three primary subsystems: the user interface, the execution support, and the software database system (see Figure 1).

Figure 1: The Main Components and Their Associated Tools of CAPS

The user interface includes a graphic editor, a syntax-directed editor, and a tool interface. The graphic editor lets the designer edit a graphical representation of the prototype and automatically produces a PSDL representation that other CAPS tools can use. The designer can specify parts of a prototype using graphical objects to represent PSDL computational structures like tasks and data streams. The designer enters text annotations with the syntax-directed editor. The tool interface hides the details of the interfaces among CAPS tools from the designer.
The software database system, which includes a design database and software base, holds reusable components and manages the configuration. It uses existing object-oriented databases and formal models for prototyping design database and software base [BOE89].

The execution support system includes a translator, static scheduler, dynamic scheduler, and debugger. The translator generates codes that bind the reusable components extracted from the software database. Its main functions are to implement data streams, control constraints, and timers. The static scheduler uses several algorithms to allocate time slots for tasks with real-time constraints on single processor before execution begins [STR88]. If this allocation succeeds, all the tasks are guaranteed to meet their deadlines even in the worst case. The dynamic scheduler allocates time slots for tasks that are not time-critical. The debugger monitors timing constraints and various aspects of design integrity as the prototype runs, reports failures, and lets the designer adjust deadlines.

The CAPS is being developed as an ongoing research effort, and some of the functions just listed were not ready when the experiments were begun. Detailed information about CAPS is contained in [WHI89], [LUK88], [JAN88], and [MAR88].

3. Prototype System Description Language (PSDL)

The *prototype system description language*, which integrates the tools in CAPS, provides the designer with a uniform conceptual framework and a high level system description and determines the properties of proposed designs via prototype execution and static analysis. It can describe prototypes of large software systems with hard real-time constraints on different levels of abstraction.
PSDL simplifies the design of a system with hard real-time constraints by presenting a high-level description in terms of networks of independent tasks to the designer, and automatically introducing any required interleaving of the codes via the execution support system.

PSDL has been designed to help the requirements analysts determine how to adjust the proposed functions of the system, the target architecture, or both to ensure that the requirements are feasible and provide the best service possible to the users of the proposed software. PSDL also provides a description of a proposed design that can be smoothly transformed into a final implementation after the requirements have been validated and the design has been verified.

B. SCHEDULING PROBLEM

Task scheduling is one of the most important issues in hard real-time systems because it is the schedule that ensures all tasks meet their deadlines. A scheduling problem in a hard real-time system is characterized by the nature of the tasks to be scheduled, the constraints associated with the system, and the usage of the schedule. Each of these is described in the proceeding sections.

1. Nature of the Task (Operator)

A task is a software module that can be invoked to perform a particular function and it is the entity of the scheduling problem in a system. In PSDL, a task corresponds to an operator and is represented by a vertex (circle) in the implementation graph. So, we will use “operator” as a synonym of “task” from now on. The following paragraphs introduce some characteristics of the task.
a. **Time-Critical and Non-Time-Critical**

A task is said to be *time-critical*, sometimes it is called a hard real-time task, if there is at least one timing constraint associated with it, otherwise it is *non-time-critical*. Ideally, the computer should execute time-critical tasks such that each task will meet its timing constraints, whereas it should execute the non-time-critical tasks such that the average response time of these tasks is minimized. The need to meet the timing requirements for all time-critical tasks is one issue that makes the scheduling problem a difficult one.

b. **Periodic and Non-periodic**

In hard real-time systems, tasks can be either periodic or non-periodic. A *periodic* task is defined as one which is activated exactly once per period P. In other words, once a periodic task is invoked at time \( t_0 \), then it will be activated exactly at time \( t_0 + P \), \( t_0 + 2P \), \( t_0 + 3P \),...etc.

*Non-periodic* tasks are those whose activation are not periodic in nature. Such tasks can be subdivided into two categories [BUR91]: aperiodic and sporadic. The difference between these two categories lies in the nature of their activated frequencies. *Aperiodic* tasks are those whose activated frequency is unbounded. In the extreme, this could lead to an arbitrarily large number of simultaneously active tasks. *Sporadic* tasks are those who have a maximum frequency such that only one instance of a particular task can be active at a time.

c. **Preemptive and Non-Preemptive**

In hard real-time systems, tasks are also distinguished as preemptive and non-preemptive. A task is *preemptive* if its execution can be interrupted by other tasks at any time and its resumption can be at the same time on a different processor
or at a later time on any processor afterwards. A task is *non-preemptive* if it must finish without interruption once it starts. So far, non-preemptive periodic tasks have received little attention. To schedule a periodic task non-preemptively, it is usually assumed that the task is executed with a fixed time between successive executions of the same task on the same processor.

d. **Dependent and Independent**

A task is considered *independent* if there are no relationships between its execution and other tasks’ executions. In other words, an independent task does not have to wait for execution until some other tasks finish their execution. Otherwise it is considered *dependent*. Usually, a complex task, for example, one requiring access to many resources, is better handled by breaking it up into multiple sub-tasks and each requiring a subset of the resources. In PSDL, the sub-tasks are designated as *atomic operators*. From now on, all the tasks to be scheduled are considered as atomic operators.

2. **Constraints and Requirements**

In a hard real-time system, a task is subject to the timing constraints, precedence constraints, communication requirements, as well as resource requirements. In this thesis, the resource requirements, except for the processor resources, are always met.

a. **Precedence Constraints**

The *precedence constraints* among a set of tasks specify the relations between the tasks. A precedence constraint ensures that a task (parent) which produces data for another task (child) will be scheduled to complete before data is required. These constraints can be specified between tasks scheduled on the same
processor as well as between tasks scheduled on different processors. Each precedence constraint is represented by an ordered pair of tasks. For example, a task $T_i$ is said to precede task $T_j$, denoted as $T_i < T_j$, if $T_i$ must finish before $T_j$ begins.

The whole precedence constraints between tasks being scheduled form a \textit{precedence graph}. The precedence graph of a set of tasks is an acyclic directed graph. It may be a chain, a tree, a series-parallel graph, or an arbitrary one. In a static system, the precedence graph are known in advance. Figure 2 illustrates on example of the precedence graph in a system.

![Figure 2: A Example of Precedence Graph](image)

\textbf{b. Timing Constraints}

The timing constraints of a task are specified by giving bounds on one or more of the following basic timing parameters:

- The \textit{period} ($P$): the time interval between any two consecutive temporal triggering events (instance) for a periodic task.

- The \textit{starting time} (START): the actual time (instant) when a task starts to execute. It is also called "the firing time".
• The **stop time** (STOP): the time (instant) when a task finishes its execution. In PSDL, the time interval from the starting time to the stop time of a task forms a **scheduling interval** for the task.

• The **arrival time** (A): the time (instant) when a task is activated in the system. The arrival time for an instance of a periodic task with period P specifies the time at which the instance of the periodic task is activated. It is specified as follows:

\[
A(i+1) = A(i) + P
\]

for \( i \geq 1 \)  \text{ (Eq 1.1)}

where \( A(i) \) is the arrival time of the \( i \)-th instance of the periodic task.

• The **ready time** (R): The time (instant) when a task is ready to begin execution. It is the earliest possible starting time for a task. The ready time of a task is equal to or greater than its arrival time, i.e., \( A \leq R \).

• The **deadline** (D): The time (instant) by which a task must finish. The deadline of instances of a periodic task with period P must satisfy the following inequality:

\[
D(i) \leq A(i) + P
\]

for \( i \geq 1 \) \text{ (Eq 1.2)}

where \( D(i) \) is the deadline of the \( i \)-th instance of the periodic task.

Being a time-critical task, at least one of the following timing constraints associates with it:

1. \( R(i) \leq \text{START}(i) \)
2. \( \text{STOP}(i) \leq D(i) \) \text{ for } i \geq 1

where \( R(i) \), \( \text{START}(i) \) and \( \text{STOP}(i) \) are the ready time, the starting time and the stop time of the \( i \)-th instance of a periodic task respectively.
Figure 3 shows a graphic view of the timing parameters for a periodic task.

Figure 3: Timing Constraints for A Periodic Task

c. Communication Requirements

The communication requirement which is the time delay for the data transfer between tasks is constrained to those which have precedence relationship. In PSDL, it is specified by the “LINK” statement. The meaning of this constraint is that data can not be read from a stream until this time delay has elapsed.

3. The Usage of the Schedule

Depending on how and where the schedule is used, the scheduling problem can be categorized as follows.
a. Static & Dynamic Scheduling

A static approach schedules tasks off-line before the system begins to operate. Therefore, it requires the complete prior knowledge of the constraints and requirements of the system. Although this approach has low run-time cost, they are inflexible and hardly adapted to a changing environment or to an environment whose behavior is not completely predictable. Whenever there are new tasks which are added to such system, the schedule for the entire system must be recalculated and the cost in terms of time and money is very expensive.

In contrast, a dynamic approach progressively determines the schedule for tasks on-line and allows tasks to be dynamically activated. This approach, because of the way it is designed, involves higher run-time cost, but it is flexible and can be easily adapted to changes in the environment. Hence, it emerges as a challenging new problem especially for distributed hard real-time systems.

For a system using static scheduling approach, called a static system, the set of tasks along with their nature in the system is pre-specified, so the number of tasks to be scheduled and the associated constraints in the system is known beforehand. However, in a dynamic system, new tasks are allowed to arrive (to be activated) at unpredictable times so the number of tasks that must be scheduled changes at run time.

b. Uniprocessor & Multiprocessor Scheduling

A schedule can either assign tasks to single processor (uniprocessor) or multiprocessor system. In uniprocessor scheduling, where tasks are scheduled only on a single time frame, each task can execute one by one based on the order of the schedule. There is no overlapping between any two execution of tasks, because the CPU can execute only one job at one time. On the other hand, multiprocessor
**scheduling** assigns tasks to more than one processors, then different tasks can execute at the same time based upon the schedule on these processors. The scheduling problem on multiprocessor system is much more complicated than on single processor system. One important aspect of the multiprocessor is its application in real-time systems. Computer archieture had made rapid progress in the manufacture of chips. This makes processors cheaper than before. Progress is now limited by software problems. Scheduling problem is one of the problems that must be solved so that the utilization of the processors can achieve maximum.

**C. OBJECTIVES**

The static schedule for multiprocessor systems is produced for the execution of a prototype, which is a fixed number of sequences tasks being developed from the prototype system description language input specification for the prototype that obeys some predefined properties, such as timing constraints and precedence relationships. The number of sequences depends upon the number of processors which are available for scheduling. The static schedule gives the precise execution order and timing of tasks with hard real-time constraints in such a manner that all timing and precedence constraints are guaranteed to be met [OHE88].

This thesis builds upon work previously done in the development of the CAPS. The major work is to improve upon the existing version of the static scheduler for the hard real-time applications in single processor systems and to develop new algorithms for scheduling tasks on multiprocessor systems to satisfy the associated constraints of the problems.

The function of the scheduling algorithm is to determine, for a given fixed number of processors and a given set of tasks, whether a schedule (the sequence and
time periods) for executing the tasks such that the timing, precedence, and resource constraints of the tasks are met, and to produce such a schedule if one exists.

D. ORGANIZATION

The rest of this thesis is organized as follows:

Chapter II defines the basic terms necessary for the implementation of the static scheduler and surveys previous research about the scheduling problems. Chapter III describes the scope of the scheduling problem being concerned by CAPS and three scheduling algorithms being developed for the hard real-time systems. Chapter IV describes the system flow of the implementation for the static scheduler and explains the modification of the existing packages as well as the new packages being developed. Chapter V summarizes the whole work in this thesis and presents recommendations for future work. Appendix A gives some examples of the input test data for the new static scheduler. Appendix B lists the scheduled tables obtained from each scheduling algorithm for each case of the input test data. Appendix C and D are the modified ADA codes of the existing packages and the new ADA packages being developed for the static scheduler applied to multiprocessor systems.
II. SURVEY OF PREVIOUS WORK

Much research has been done in hard real-time scheduling problems. In this chapter, the previous research related to the static scheduling problem as well as the definition of terms used in the CAPS static scheduler will be discussed.

A. DEFINITION OF TERMS

The following paragraphs define the terms necessary to understand the static scheduling algorithms.

1. HBL

The Harmonic Block Length (HBL) is the least common multiple (LCM) of all the periods of tasks being scheduled. The reason to create the HBL is that once a schedule has been developed for a HBL, it can be repeated over and over again.

Scheduling periodic tasks naturally leads to periodic schedules. A schedule is called periodic with a period HBL if the following holds:

\[ \text{if task is executed at time } t \text{ then it is also executed at time } (t + HBL). \]

The algorithm to calculate the HBL is introduced in the package of “HARMONIC_BLOCK_BUILDER” [KIM89].

2. Instance

The instance of a periodic task is the repetition of this task in one Harmonic Block Length. The number of instances (N) for each task in a Harmonic Block Length is calculated as follows:

\[ N := \frac{HBL}{P} \quad (\text{Eq 2.1}) \]
3. Tardiness and Cost

The amount of time by which the stop time of an instance missed its deadline is called the **tardiness** of the instance. It is an essential quantity to calculate the cost of a schedule. If the stop time is less than or equal to the deadline, then the tardiness is zero. The tardiness of instance \(i\), \(T(i)\), of a task is calculated as follows:

\[
T(i) := \max \{ 0, \text{STOP}(i) - D(i) \}, \quad \text{for } i \geq 1
\]

(Eq 2.2)

The **cost** \((C)\) of a schedule can be defined as either the largest tardiness or the sum of all tardiness experienced by tasks when executed according to the schedule, i.e.,

\[
C := \max \{ T(i) \mid \text{for all instances } i \text{ of all tasks in the schedules} \} \quad \text{(Eq 2.3)}
\]

or

\[
C := \sum T(i), \quad \text{for all instances } i \text{ of all tasks in the schedules} \quad \text{(Eq 2.4)}
\]

4. Latency

To express the behavior of distributed systems, PSDL has been extended to define optional latency attribute. The **latency** between two tasks is an upper bound on the duration of the time interval between the time instant a data value is written out to the data streams and the time instant that data value becomes available for reading by the next task. In the absence of explicit specifications, the latency of a stream has the default value zero (no delay). The purpose of these constraints is for the specification of communication constraints due to hardware limitations imposed by external constraints on how the software functions must be allocated to different physical nodes of a distributed system.

5. MET

The **maximum execution time** (MET) is the maximum time interval of execution for a task. It is the CPU time required to execute a task under worst-case
conditions. A correct implementation of a time-critical task must provide a guarantee of service with respect to bounded computational resources, i.e., the static scheduler must ensure that at least this much CPU time is allocated to a task between each activation time and its deadline.

6. **Finish Within**

The *finish within* is the hard deadline which is the largest permissible time interval between the temporal triggering event and the completion of the execution for a periodic task. The finish within is constrained by the following relation:

\[ P \geq \text{finish within} \geq MET, \quad \text{and} \]

\[ \text{finish within} := P, \quad \text{by default.} \]

Hence, the deadline of the i-th instance of a time-critical periodic task equals:

\[ D(i) := A(i) + \text{finish within} \]

for \( i \geq 1 \) \hspace{1cm} (Eq 2.5)

7. **Pipeline**

If a task can be pipelined, then different repetitions (instances) of the task can be scheduled on different processors with overlapping, i.e., more than one instance of this task can be firing at the same time. A time-critical task whose period is less than its maximum execution time, i.e., \( P < MET \), can be realized only if it can be pipelined. If a periodic task can be pipelined then successive executions of the task can overlap in the schedule as long as its next instance of this task has been invoked.

8. **Legal Solution and Feasible Solution**

If a schedule satisfies all the precedence constraints, then it is said to be a *legal* solution.
A schedule is called a **feasible** solution, if it is legal and every task when executes according to the schedule meets its deadline, i.e., the schedule meets not only the precedence constrains but also the timing constraints.

9. **Optimal Solution and Approximate Solution**

A schedule is said to be **optimal** if, for any set of tasks, it always produces a schedule which satisfies all the constraints of the tasks.

A feasible solution with value (cost) close to the value (cost) of an optimal solution is called an **approximate solution**. An approximate solution may not lead to the optimal result. However, there are many problems that have no exact solutions, and can only be solved by using approximation methods. We need an efficient heuristic algorithm to produce a close approximation to the optimal solution.

B. **PREVIOUS RESEARCH**

Many researchers have attempted to solve the general case of the scheduling problems with strong bounding conditions. The following paragraphs summarizes previous research relating to the static scheduling problems applied to both uniprocessor and multiprocessor environments for hard real-time systems.

1. **Static Scheduling for Uniprocessor**

Horn [HOR74] developed an optimal algorithm for scheduling preemptive independent tasks with arbitrary ready time and deadlines. His approach is based on the **earliest deadline first** policy. Liu and Layland [LIL73] developed a **rate-monotonic priority** scheme to determine the schedulability of a set of preemptive periodic tasks. This scheme assigns higher priorities to tasks with shorter periods. They showed that this scheme is optimal among fixed-priority schemes. Teixeira [TEI78] presented a fixed-priority assignment scheme for a slightly different
problem. He assumed that the relative deadline of a periodic task can be different from the period of the task. Sha and Lehoczky [LES86] described a technique to modify the periods of tasks in such a way that while tasks' timing constraints continue to be met, better processor utilization is achieved. This modification consists of breaking up one period task into two, each with half the computation time and half the period as the original task. The above approaches are different from the conventional priority-driven scheduling approaches, because they assign priorities to tasks based on a simple function of the timing constraints, instead of one that combines timing constraints and criticalness, of the periodic tasks.

Scheduling non-preemptive tasks is more difficult whether on uniprocessor or multiprocessor systems. Moore [MOO68] showed that the earliest deadline algorithm is optimal for scheduling a set of tasks with the same ready time. Kise [KIS78] developed an algorithm for the case in which a task has an earlier ready time if and only if it has an earlier deadline. Bratley, Florian, and Robillard [BFR71] developed an implicit enumeration algorithm to determine schedule for non-preemptive tasks with arbitrary ready time and deadlines [BFR75]. Baker and Su [BAS74] used a similar approach to minimize the maximum tardiness of tasks. Erschler et al [EFM83] developed a necessary condition for scheduling tasks with arbitrary ready time and deadlines. Their theories can be used to reduce the search space of an enumeration algorithm.

The original design of the static scheduler of CAPS was described in [LUQ86]. This design was further developed into a conceptual design for the pioneer prototype of the static scheduler [OHE88], and then was implemented by the Ada programming language in 1988 [JAN88],[MAR88]. The current static scheduler consists of four algorithms for scheduling hard real-time tasks on single processor.
They are *earliest start first, earliest deadline first* [CER89], [KIM89], *exhaustive enumeration with branch and bound* [FAN90], and *simulated annealing* [LEV91].

2. **Static Scheduling for multiprocessor**

Horn [HOR74] described an algorithm to schedule preemptive independent tasks with arbitrary ready times and deadlines. His approach is based on the network flow method and considers processors with identical processing speed. Many researchers adopted a partition approach to solve the problems of scheduling preemptive periodic tasks on multiprocessor systems. The main idea of the approach is to partition a set of periodic tasks among a minimum number of processors such that each partition of the periodic tasks can be scheduled on one processor according to the earliest deadline scheme or the rate-monotonic priority scheme. Davari and Dhall [DAD86] showed that, if the earliest deadline scheme is used, a *bin-packing* algorithm can be used to determine a suboptimal partition pattern of periodic tasks among multiple processors preemptively. Bannister and Trivedi [BAT83] proposed a simple *best-fit partition* scheme. This scheme can be used in conjunction with both the earliest deadline scheme and the rate-monotonic priority scheme. For rate-monotonic priority scheme, Dhall and Liu [DHL78] improved these schemes and developed a more efficient *next-fit partition* scheme.

As described above, many of the scheduling algorithms designed for the preemptive periodic tasks are based on a fixed-priority assignment scheme. The advantage of a fixed-priority assignment scheme is that they have a very small scheduling overhead, because they are designed for prioritized-interrupt handling systems and the priority mechanism is often supported by hardware. However, in general, these schemes are very inflexible, because it is expensive to change the priority assignment once it is fixed on a system.
To schedule the non-preemptive tasks, a polynomial optimal algorithm is available only for the tasks with unit computation time [SIM80], [SIM83], [SIS84], [LAF76]. Otherwise, there is no polynomial optimal algorithm available so far for scheduling non-preemptive tasks on multiprocessor systems. Bratley, Florian, and Robillard [BFR75] developed a multi-stage enumeration algorithm to schedule tasks with arbitrary ready time and deadlines. Because the worst case exponential time complexity, the approach is designed to run off-line. Blazewicz, Drabowski, and Weglarz [BDW86] investigated an interesting scheduling problem in which tasks need multiple processors at the same time for processing. They showed that polynomial-time algorithms exist if the number of processors and the processing time required by tasks are constant.

If precedence constraints are subject to the tasks in multiprocessor system, the scheduling problems will be much more difficult. For example, scheduling tasks with arbitrary precedence constraints and unit computation time is \textit{NP-hard} both for the preemptive and the nonpreemptive cases [ULL75], [ULL76].

Kasahara and Narita [KAN84] have developed an implicit heuristic search algorithm to determine the minimum schedule length for a set of nonpreemptive tasks with arbitrary precedence constraints. They showed that their enumeration algorithm can provide optimal or suboptimal solutions to large-scale problems within a time limit. However, the worst case execution time grows exponentially. Elsayed [ELS82] presented a number of heuristic algorithms for finding suboptimal solution to a similar scheduling problem. These heuristic algorithms do not enumerate over multiple paths in a search space. They are designed based on a straightforward topological search scheme and the \textit{critical path} method combined with a heuristic rule. Therefore, such heuristic algorithms are much more efficient than the implicit enumeration algorithm described above. Hsu [HUS90] introduced some basic
concepts needed to schedule tasks in the multiprocessor system in CAPS. But there is no implemented codes available.
III. ALGORITHM DESIGN

This chapter describes the assumptions of the scheduling problem which is going to be dealt with, and then briefly describes three scheduling algorithms being developed for the hard real-time tasks in multiprocessor environment.

A. ASSUMPTIONS

Because this research is pioneers work in developing the static scheduler for CAPS in multiprocessor environments, some assumptions about the scheduling problem must be clarified before designing the algorithm.

(1) Since the purpose of the effort is to develop a static scheduler, all the requirements are static in nature (off-line schedule).

(2) Let \( T := \{ o_1, \ldots, o_n \} \) be a set of \( n \) periodic tasks. For each \( o \in T \), a maximum execution time \( \text{MET}(o) \) and a period \( \text{P}(o) \) are given. If the task is a sporadic task, it has to be converted into a periodic task. (periodic tasks)

(3) Once an execution of a task is started, it will be completed without interruption from the same processor. (non-preemptive tasks)

(4) The processors are supposed to be identical, that is to say, each task can be executed on any processor and the time to execute each task does not depend on the processor. Furthermore, a processor can only execute one task at a time. (identical processors)

(5) The ready time of the first instance for the operator who has no parent is assumed to be 0, but for the operator who has parents is the actual starting time of its first instance.

(6) In the scheduling process, two attributes are defined. The "LOWER" is the lower bound of the starting time for an instance of an operator, that is the ready time
for that instance, which ensures that at least one period is passed from the ready time of previous instance.

The "UPPER" is the upper bound of the starting time for an instance of an operator, which ensures that the instance is scheduled early enough so that it can finish execution prior to the deadline.

(7) If $o_1$ and $o_2$ are periodic operators, then instance $i_1$ of $o_1$ precedes instance $i_2$ of $o_2$, written as $(o_1, i_1) < (o_2, i_2)$, if and only if $o_1$ is a parent of $o_2$ and $(i_1 - 1) \times P(o_1) = (i_2 - 1) \times P(o_2)$. The purpose of the second condition is to define corresponding synchronization points for the operators, as explained below.

If $(o_1, i_1) < (o_2, i_2)$ then instance $i_1$ of operator $o_1$ must complete execution before instance $i_2$ of operator $o_2$ can start execution, and instance $i_2$ of operator $o_2$ must read its inputs before instance $i_1 + 1$ of operator $o_1$. The purpose of this constraint is to allow the instance $(o_2, i_2)$ to operate on the data produced by the instance $(o_1, i_1)$. The first constraint is necessary to ensure the output of $(o_1, i_1)$ has been produced before it is used by $(o_2, i_2)$, and the second constraint is needed to ensure that the output of $(o_1, i_1)$ is not over-written by the output of $(o_1, i_1 + 1)$ before it can be read by $(o_2, i_2)$. When such a relation is guaranteed between two instances of periodic operators, the two instances are synchronized. The PSDL scheduling policy guarantees that the first instances of any two periodic operators connected by the precedence graph must be synchronized, and that subsequent instances are synchronized at intervals corresponding to the least common multiple of the periods of the two operators. In particular, if both operators have the same period, then they are synchronized for every pair of corresponding instances. This is illustrated in Figure 4 on next page.
<table>
<thead>
<tr>
<th>period (a)</th>
<th>period (b)</th>
<th>synchronization points</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>(a, 1) &lt; (b, 1), (a, 2) &lt; (b, 2), (a, 3) &lt; (b, 3)....</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>(a, 1) &lt; (b, 1), (a, 4) &lt; (b, 3), (a, 7) &lt; (b, 5)....</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>(a, 1) &lt; (b, 1), (a, 3) &lt; (b, 2), (a, 5) &lt; (b, 3)....</td>
</tr>
</tbody>
</table>

Figure 4: Synchronization Points for Connected Periodic Operators

B. EARLIEST STARTING TIME FIRST ALGORITHM

The earliest starting time first algorithm assigns the highest priority to the task with the earliest starting time, i.e. the task with the least lower bound of the starting time among all activated tasks will be scheduled first. This algorithm schedules the task with the earliest starting time to execute on the earliest available processor to ensure all its descendants tasks can execute as soon as possible, so as not to violate the deadline constraints. The reason to choose the earliest available processor is to minimize the processor idle time.

In this algorithm, a priority queue is used for storing the information about the tasks which are going to be scheduled. The order of the priority queue depends on the lower bound of the starting time of each task (instance). The task, located at the top of the queue, with the smallest lower bound of the starting time will be extracted from the queue and be scheduled. Whenever a task is extracted from the priority queue, its lower bound has to be re-calculated. A recursive procedure is used to perform this function. Once there is a synchronized point between the task and its parent, the lower bound of the task has to be adjusted until all its parents with
synchronization has been completed. If any one of its parents with synchronization has not been scheduled, the task is pushed into a waiting list. The next task is then extracted from the top of the priority queue. The process keeps going until a task's lower bound has been completely calculated according to the stop time of all its parents having synchronization. The tasks in the waiting list are popped out and re-inserted into the priority queue. The earliest starting time first algorithm is described as follows.

BEGIN -- earliest starting time first algorithm

while not empty (WORK_LIST) loop
  schedule the task;
  if INSTANCE < (HBL / P) then
    create an additional node for next instance of the same task;
    insert it into the priority queue;
  end if;

  insert all its children whose parents have all been scheduled into the priority queue;

  next (WORK_LIST);
end loop;

while not empty (priority queue) loop
  extract the task from the top of the priority queue;
  if INSTANCE = 1 then
    schedule the task;
    insert all its children whose parents have all been scheduled;
  else
    determine the new lower bound;
  end if;
end loop;
schedule the task;
end if;

if START > UPPER or STOP > HBL then
    VALID_SCHEDULE := false;
end if;

if INSTANCE < (HBL / P) then
    create an additional node for next instance of the same task;
    insert it into the priority queue;
end if;
end loop;
reverse the schedule;
ed EARLIEST_START;

-- The WORK_LIST links the tasks which have no tasks precede them.

-- The steps of scheduling a task include: assign a processor to the task, calculate the upper bound, the instance number, the starting time and the stop time for the given task.

-- according to the method for adding new tasks into the linked list, the schedule was developed in the reverse order, so it need to be reversed.

C. EARLIEST DEADLINE FIRST ALGORITHM

The earliest deadline first algorithm, similar to the earliest starting time first algorithm, assigns the highest priority to the task with the earliest deadline. In other words, this algorithm schedules the task with the smallest upper bound of starting
time to execute, before other tasks, on the earliest available processor to ensure that the task meets its deadline constraint. The reason to choose the earliest available processor is the same as mentioned in the earliest starting time first algorithm. The task with the smallest upper bound of the starting time is the most urgent task should execute before any other one because if the task was postponed for a moment, the deadline might be missed.

In this algorithm, a priority queue is also used for storing the information about the tasks which are going to be scheduled. The task located at the top of the queue has the smallest upper bound of starting time and will be extracted first. The algorithm, similar to the earliest starting time first algorithm, is described as follows.

```
BEGIN -- earliest deadline first algorithm
   while not empty (WORK_LIST) loop
      calculate the upper bound for the task;
      insert it into the priority queue;
      next (WORK_LIST);
   end loop;
   while not empty (priority queue) loop
      extract the task from the top of the priority queue;
      if INSTANCE = 1 then
         schedule the task;
         insert all its children whose parents have all been scheduled
            into the priority queue;
      else
         determine the new lower bound;
      end if;
   end loop;
END
```
schedule the task;
end if;

if START > UPPER or STOP > HBL then
   VALID_SCHEDULE := false;
end if;

if INSTANCE < (HBL / P) then
   create an additional node for next instance of the same task;
   insert it into the priority queue;
end if;
end loop;

reverse the schedule;
end EARLIEST_DEADLINE;

D. SIMULATED ANNEALING ALGORITHM

1. Generic Description

The use of simulated annealing to solve combinatorial optimization problems is an area that has received much attention lately. Combinatorial optimization problems are those whose configuration of elements are finite or countably infinite. An example of a combinatorial optimization problem is the assignment problem where there are numbers of personnel available to do an equal number of jobs. The cost for each person to do each job is known. The goal is to assign each person to a job so that the total cost is as small as possible [OTV89]. There is a wide range of combinatorial optimization problems that the simulated
annealing approach can be utilized. These include graph partitioning, graph coloring, number partitioning, VLSI design, and travelling salesman type problems.

Simulated annealing is based on the behavior of physical systems and the laws of thermodynamics. The way that liquids freeze and crystallize or metals cool and anneal are the principles upon which simulated annealing is based. At high temperature, liquid molecules move freely with respect to one another. As the liquid cools, this mobility is lost. Atoms line up and form a pure crystal that is at a minimum energy level. As the system cools slowly nature finds the minimum energy state [FLO84]. Examining simulated annealing in non-physical terms, a comparison is made to the concept of local optimization or iterative improvement. Local optimization repeatedly improves an initial solution until no further improvement of the solution is possible. This is known as iterative improvement or "hill climbing". Simulated annealing differs from local optimization in that random uphill movements (acceptance of a worse solution) are permitted. This prevents the algorithm from being trapped in a poor local optimal solution as demonstrated in Figure 5 on next page. Because simulated annealing avoids poor local optima, significantly better results can be found utilizing it as opposed to local optimization [JOH89].

The key to the use of the simulated annealing approach to solving combinatorial optimization problems is the random acceptance of worse iterative solutions. When the system is in a high energy state (high temperature), the probability is greater that a worse iterative solution is accepted. As the system cools this probability decreases, but even at the lower energy states the probability for making an uphill move still exists.
As indicated in Figure 5 above, uphill moves allow the algorithm to leave a poor local solution (point A or point B) and reach a better solution in the vicinity of point C. This general scheme of always taking a downhill step while occasionally taking an uphill step is known as the Metropolis algorithm, named after Metropolis, the scientist, who with his coworkers first investigated simulated annealing in 1953 [FL084].

The choice of a probability function to determine if an uphill movement is allowed is an important consideration. At each step of the simulated annealing algorithm a new state is constructed based on the current state. This new state is
constructed by randomly displacing or adjusting a randomly selected element. If this
new state has a lower cost than the current state, the new state is accepted as the
current state. If the new state has a higher cost than the current state, the new state is
accepted with probability:
\[ \exp(-\Delta e/kT) \]

This probability function is known as the Boltzman probability distribution
where:
\[ \Delta e = \text{difference in cost between new state and current state} \]
\[ k = \text{Boltzman's constant of nature relating temperature to energy} \]
\[ T = \text{Current Temperature} \]

A characteristic of this probability function is that at very high temperatures
every new state has almost the same chance of being accepted as the current state. At
low temperatures, the states with a lower cost have a higher probability of being
accepted than the current state.

Simulated annealing is simple to implement and can be applied to a variety
of combinatorial optimization problems. To apply annealing, a description of the
problem state space, a procedure or routine to adjust the state, and some function to
determine the cost of the solution in the state space are all that is required [OTV89].
The next section will address these requirements as well as examine the algorithm
for simulated annealing.

2. Algorithm Description

The concept of simulated annealing is closely related to local optimization.
According to Johnson [JOH89], simulated annealing, which is a combinatorial
optimization algorithm, can be specified by identifying a set of solutions together
with a cost function that applies a value to each solution. There exists an optimum
solution which has the minimum cost possible. (NOTE: There may be more than one optimum solution). Starting with an arbitrary initial solution, the algorithm attempts to improve on the initial solution by performing incremental changes on that solution. A cost function is used to evaluate each iterative solution that is developed.

To utilize the simulated annealing algorithm, the following four elements must be provided:

(1) A description of possible system configurations.
(2) A generator of random changes in this configuration, these changes are "options" presented to the system.
(3) An objective function E (analog of energy) whose minimization is the goal of the procedure.
(4) A control parameter, T (analog of temperature) and an annealing schedule which tells how T is lowered and cooled.

The annealing schedule sets the number of random changes sampled for each temperature T and rate at which T decreases. The range of the annealing temperature and the value of the annealing schedule are normally established from trial and error experimentation [FLO84]. A pseudo code representation of the simulated annealing algorithm based on the algorithm proposed in [JOH89] follows:

**Input:**

(1) The solution space of the optimization problem.
(2) The control parameters for the annealing process, which include
   
   (a) $T_0$ - the initial value of the control temperature T,
   
   (b) Freeze - the final value of T,
   
   (c) $R$ - the reduction factor for T (typically $0.70 \leq R \leq 0.99$),
(d) $L$ - the maximum number of attempted moves at any each value of $T$,

(e) $L_a$ - the maximum number of accepted moves at any each value of $T$.

Output:

An optimal or near-optimal solution.

Algorithm

Begin

Current_Solution := some solution from the given solution space;

$T := T_0$;

Best_Solution := Current_Solution;

While ($T > \text{Freeze}$) do

begin

$N := 0$; /* tracks the number of moves attempted at the current value of $T$/

$N_s := 0$; /* tracks the number of moves accepted at the current value of $T$/

while (($N < L)$ and ($N_s < L_a$)) do

begin

/* perturb current solution randomly to obtain a new legal solution */

New_Solution := move (Current_Solution);

$N := N + 1$;

$\Delta C := \text{cost} (\text{New_Solution}) - \text{cost} (\text{Current_Solution})$;

if ($\Delta C \leq 0$) or (random () $\leq e^{-\Delta C / T}$)

then begin

Current_Solution := New_Solution;

$N_s := N_s + 1$

if (cost (Current_Solution) $<$ cost (Best_Solution))

then Best_Solution := Current_Solution;

end

end

end
end;

end;

$T := T \times R$

end;

Output: (Best\_Solution);

End.

The choice of values for $T_0$, $R$, and $L$ have a significant impact on the annealing schedule. The higher the initial temperature, the higher the cooling factor, and the larger the number of trials at each temperature result in more solutions being examined in order to find an optimum solution. The goal in choosing these parameters is to ensure that a sufficient, but not excessive, number of solutions are examined. These values are normally chosen arbitrarily and adjusted through experimentation.
IV. IMPLEMENTATION

Previous works on the static scheduler in CAPS developed by Janson [JAN88], Cervantes [HUS90], and Levine [LEV91] all applied to uniprocessor environment. The three algorithms developed in this thesis are used for scheduling hard real-time periodic tasks in multiprocessor environment. The ADA programming language is used to develop the new static scheduler packages as well as to modify the existing packages. This chapter describes the system flow of the static scheduler, also explains the modification for the existing packages and new packages being developed.

A. SYSTEM FLOW

The system flow diagram is given by Figure 6 as follows.

![System Flow Diagram of the Static Scheduler](image)

Figure 6: System Flow Diagram of the Static Scheduler
The first module "FRONT_END" reads the text input file "ATOMIC.INFO", which contains the operators' identifiers, timing information, and "LINK" statements which describe the PSDL implementation graphs, and separates the information of the time-critical operators and stores them in a linked list "OP_LIST". It also produces the number, "OP_COUNT", of the time-critical operators which are going to be scheduled.

The second module "PROCESSOR" calculates the periods for the sporadic operators and tests the time-critical operators' information from the linked list "OP_LIST". At this stage, all sporadic operators are converted to their periodic equivalents.

The third module "HARMONIC_BLOCK_BUILDER" uses to the period information of the periodic operators from the "OP_LIST" to determine the harmonic block length (HBL) of the static schedule as mentioned earlier.

The forth module "NEW_DAT_STRUCTURES" is a generic package which is instantiated in the declarative part of the package "FRONT_END". It produces a record type of data structure named "GRAPH" which includes two entries, "OP_ARRAY" and "OP_MATRIX" (see Figure 7).

![Figure 7: Graph Structure](image-url)
The first entry "OP_ARRAY" of "GRAPH" is a one-dimension array type which contains all relevant information about the operators to be scheduled from the "OP_LIST". Once the information is stored in the array, the operators can be referred to by their index number (position) in the array. This allows for immediate access of all relevant operator information instead of having to traverse a linked list to find the desired operator's information. Identifying operators by their index numbers as opposed to their names reduces the storage space required for operator's identifications throughout the static scheduler.

The second entry "OP_MATRIX" of "GRAPH" is a two-dimension \((n \times n)\) array type, where \(n\) is the number of operators to be scheduled, which contains the information about the parent-child relationships between operators as well as the pipeline information about each operator.

The fifth module "TOP_SORT" performs a topological sort on all the operators according to the information produced by the package of "NEW_DATA_STRUCTURE". It creates a true topological ordering which is not dependent on a specific ordering of the operators in the PSDL input file. The result, called "PRECEDENCE_LIST", is a linked list data structure containing each operator's index number according to the position of the operator in the "OP_ARRAY". In the head of the list, there is a dummy operator (index number 0) created to lead all other operators.

The last module "STATIC_SCHEDULER" is the heart of the whole system which combines the output of "TOP_SORT" (PRECEDENCE_LIST), "FRONT_END" (OP_COUNT), "NEW_DATA_Structures" (OP ARRAY, OP_MATRIX), and "HARMONIC_BLOCK_BUILDER" (HBL) to produce a static schedule which can be applied to multiprocessor systems. This module includes three static scheduling algorithms which have been introduced in last chapter.
B. MODIFICATIONS ON EXISTING PACKAGES

This section describes the modifications of the existing packages in the proposed implementation.

1. DATA

The original "DATA" package contains the definitions of all types, instantiation of generic packages, and global variables used by the static scheduler for uniprocessor environment. Most of the packages remain the same. Some additional global variables and data types which are necessary for multiprocessor scheduling are added in this package.

* NOP : NATURAL := 4;

It is a global variable used for storing the number, which equals four in the experimental case, of the available processors which can be applied for the tasks' execution.

* type PROCESSORARRAY is array (L..NOP) of NATURAL;

This data type is used for the variables, for example "PROCESSOR_STOP", which track the available time (instant) of each processor at each stage during the scheduling process.

* type SCHEDULEARRAY is array (1..NOP) of SCHEDULEINPUTS_LIST.LIST;

It is used for the variables, for example "AGENDA", which store the pointers pointing to the linked lists of the schedule for each processor.

The other additional global variables are listed in "APPENDIX C".
2. NEW DATA STRUCTURES

Besides the original procedures and functions in the old version of this package, two functions are created. Also, one procedure and one data type are modified.

- type MATRIX_OP_INFO is
  record
    PARENT : INTEGER;
    CHILD : INTEGER;
    DELAY_PIPELINE : INTEGER;
  end record;

This data type resides in the two dimension array (matrix) "OP_MATRIX" and the additional entry "DELAY_PIPELINE" is used to store the information about the operators whether they can be pipelined or not and the information about the latency between each pair of operators. If the "DELAY_PIPELINE" in the diagonal cell \([i, i]\) of the matrix equals, then the operator \(i\) can be pipeline. The "DELAY_PIPELINE" in other cells \([i, j]\), for \(i \neq j\), of the matrix determines the latency between operator \(i\) and operator \(j\) (where \(i, j\) are the index number of the operators).

- procedure "PRODUCE_OP_MATRIX":

In the old version of this procedure, the "LINK" statements were skipped in the PSDL input file. Because the new version of the static scheduler considers the communication delay between operators, this procedure is modified in order to store the information of the latency between operators into the proper place.
• function "LATENCY":
  This function is used for the new static scheduler to return the value of the latency between two operators. The first parameter is the parent’s index number and the second parameter is the child’s index number.

• function "PIPELINE":
  This function returns the boolean constant "TRUE" if the operator can be pipelined, otherwise it returns "FALSE". Its parameter is the index number of the operator.

3. DIAGNOSTICS

This package includes four procedures: "OUTPUT_OP_ID", "OUTPUT_SCHEDULE", "OUTPUT_HARMONIC_BLOCK_LENGTH", and "OUTPUT_PRECEDENCE_LIST". These procedures output the information to the terminal for the purpose of diagnostics. The procedure "OUTPUT_SCHEDULE" is modified in order to output the schedule which will be applied to the multiprocessor other than the uniprocessor to the terminal.

C. NEW PACKAGES

There are two packages, "UTILITY_PKG" and "NEW_SCHEDULER_PKG", created in the new version of the static scheduler. They are the major effort of the implementation of the static scheduler. Each will be described in the following paragraphs.

1. UTILITY_PKG

This package consists of procedures and functions which help the scheduling algorithms to solve specific problems.
a. **RANDOM_INITIALIZE**

This procedure which was separate in the original version of the static scheduler initializes the random number generator. The input parameter should be an odd integer.

b. **RANDOM_NEXT**

This function creates a random number which has a type of “FLOAT” ranging from 0 to 1 and was originally separate in the old version.

c. **DETERMINE_THE_UPPER**

It is a procedure that calculates the upper bound of the starting time for each instance of the tasks. If the starting time of an instance exceeds its upper bound, the instance may miss its deadline. In other words, this task violates its timing constraints and the resulting schedule would not be feasible.

d. **DETERMINE_START_STOP**

This procedure calculates the actual starting time and stop time (scheduling interval) for an instance of a task. Even though an instance has a lower bound of the starting time, it still cannot execute unless there is at least one processor available after the lower bound. If there are more than one processor available after this lower bound, the processor with the earliest release time will be chosen.

e. **CREATE_ADDL_NODE**

After an instance of an operator has been scheduled, the operator will be checked if it has more instances in the HBL to be scheduled. If so, an additional node of the scheduling information about the next instance of this operator will be created with a record data type of “SCHEDULE_INPUTS” declared in the “NEW_DATA_STRUCTURES” generic package. The entry “THE_START” of this
additional node is used to store the information about the synchronization instead of the starting time of next instance, because it is not known at this time when the node is created. This process includes in the “CREATE_ADDL_NODE” procedure.

\[ f. \text{ TEST\_SCHEDULE} \]

This procedure calculates the cost of a given schedule. In other words, it finds out the maximum tardiness of all instances who miss their deadlines and whose stop time exceeds the HBL. If the cost equals 0, a feasible schedule is found. Otherwise, the given schedule is not a feasible solution.

\[ g. \text{ ANNEAL\_FUNCTION} \]

This function used in the “simulated annealing” algorithm returns a “FLOAT” type number between 0 and 1. After comparing this number with the random number, the annealing process then decides whether to accept the new solution or not. The first parameter of this function is the cost of the new solution, and the second parameter is the cost of the original one. The last parameter is the current temperature.

\[ h. \text{ ADJUST\_PRECEDENCE} \]

This procedure used in the “simulated annealing” algorithm adjusts the “PRECEDENCE\_LIST” to get a new ordering of the operator.

2. NEW\_SCHEDULER\_PKG

In this package, three algorithms were developed:

- “EARLIEST\_START”
- “EARLIEST\_DEADLINE”
- “SIMULATED\_ANNEALING”
Each of these has been introduced in chapter three. The following paragraphs describe the procedures in “SIMULATED_ANNEALING”.

a. **SCHEDULE_1st_INSTANCES**

   This procedure schedules the first instance for each operator. Its algorithm is described as follows.

   BEGIN -- schedule the first instance for each operator
   
   *duplicate* P_LIST to WORK_LIST;
   
   *while not empty* (CHILD_LIST of the dummy node) *loop*
   
   calculate the upper bound, start time, stop time for the task;
   
   *schedule the task*; -- (lower bound := 0)
   
   *remove this task from the WORK_LIST*;
   
   *if INSTANCE = (HBL / P) then*
   
   *remove this task from the P_LIST*;
   
   -- no need to schedule the next instance for the task
   
   *else*
   
   *create an additional node of the next instance*;
   
   *add it into an additional list (A_LIST)*; -- to be scheduled for the future
   
   *end if*;
   
   *next (CHILD_LIST)*;
   
   *end loop*;
   
   *while not empty (WORK_LIST) loop*
   
   calculate the lower bound, upper bound, start time and stop time;
   
   *schedule the task*;
   
   *if INSTANCE = (HBL / P) then*
   
   *remove this task from the P_LIST*;
   
   *end if*;

44
-- no need to schedule the next instance for the task

else

create an additional node of the next instance;

add it into an additional list (A_LIST); -- to be scheduled for the future

end if;

next (WORK_LIST);

end loop;

end SCHEDULE_1st_INSTANCES;

-- the P_LIST is the precedence list of the operators from the TOP_SORT package output.

b. SCHEDULE_REST_OF_BLOCK

It is a continuous work for the "SCHEDULE_1st_INSTANCE" procedure. This procedure schedules the rest of the instances other than the first instance of the tasks in one harmonic block length. After this procedure, an initial solution is obtained for the annealing process to anneal if needed. The algorithm is described as follows.

BEGIN -- schedule the rest instances other than the first instance in a HBL

while not empty (P_LIST) loop

duplicate P_LIST to WORK_LIST;

while not empty (WORK_LIST) loop

extract the same task of WORK_LIST from the A_LIST;

DETERMINE_THE_LOWER; -- recalculate the lower bound

if the lower bound has been recalculated then

45
schedule the task;
if INSTANCE = (HBL / P) then
    remove this task from the P_LIST;
    -- no need to schedule the next instance for the task
else
    create an additional node of the next instance;
    add it into an additional list (A_LIST);
    -- to be scheduled for the future
end if;
remove this task from the A_LIST;
end if;
next (WORK_LIST);
end loop;
end loop;

end SCHEDULEREST_OF_BLOCK;

-- The "P_LIST" is part of the PRECEDENCE_LIST coming from the output of
the "SCHEDULE_1st_INSTANCES" procedure.

-- The sub-procedure "DETERMINE_THE_LOWER" checks all the parents of
the task to be scheduled. Whenever there is any parent having a synchronized
point with this task, the lower bound of the task has to be recalculated. If the
parent has not been scheduled for the synchronized instance, the task can not
be scheduled at this time. Then the sub-procedure returns a waiting message.
c. \textit{ANNEAL\_PROCESS}

Actually this procedure is derived from the generic simulated annealing algorithm introduced in chapter three. It occurs when the initial solution is not feasible. It uses a priority queue, named "QUE", as in the "earliest starting time first" algorithm. The order of the tasks in "QUE" depends on the lower bound of each task. The task with the smallest lower bound will be put on the top of the priority queue and will be extracted before any other tasks. The modification for the generic simulated annealing algorithm to solve the static scheduling problem is described as follows.

\textbf{INPUT:}

\begin{itemize}
  \item HBL;
  \item AGENDA; \textit{-- schedule on multiple processors.}
  \item PENALTY\_COST; \textit{-- cost for a given schedule.}
  \item PRECEDENCE\_LIST;
\end{itemize}

\textbf{OUTPUT:}

\begin{itemize}
  \item AGENDA;
  \item PENALTY\_COST;
  \item FEASIBLE\_SOLUTION: \textbf{boolean};
\end{itemize}

\textbf{BEGIN} -- annealing process

\begin{itemize}
  \item duplicate AGENDA to BEST\_AGENA and TEMP\_AGENDA;
  \item initialize the temperature $T$;
  \item BEST\_COST := PENALTY\_COST;
  \item \textbf{while} not FEASIBLE\_SOLUTION and $T > \text{Freeze}$ loop
  \begin{itemize}
    \item clear ACCEPT\_COUNT and TRIAL\_COUNT;
    \item \textbf{while} not SOLUTION\_FOUND and
  \end{itemize}
\end{itemize}
ACCEPT_COUNT < ACCEPT_NO and
TRIAL_COUNT < TRIAL_NO

loop

REARRANGE_P := false;

find the first task whose START > UPPER on each processor;

insert them into (QUE);

while not empty (QUE) loop

extract the task from (QUE);

recalculate the lower bound;

if LOWER > UPPER then

REARRANGE_P := true;

exit the loop;

end if;

if LOWER < START then

promote the task into a suitable time slot;

if can not find a suitable time slot then

REARRANGE_P := true;

exit the loop;

else

remove the task from TEMP_AGENDA and QUE;

next TEMP_AGENDA of the same processor;

insert it into QUE;

end if;

else

remove the task from QUE;

next TEMP_AGENDA of the same processor;

insert it into QUE;
end if;
end loop;

if not REARRANGE_P then

find the first task whose STOP > HBL on each processor;
inset them into (QUE);

while not empty (QUE) loop

extract the task from (QUE);
OLD_LOWER := LOWER;
recalculate the lower bound;
if LOWER = OLD_LOWER and LOWER = START then

REARRANGE_P := true;
exit the loop;
else

promote the task into a suitable time slot;
if can not find a suitable time slot then

REARRANGE_P := true;
exit the loop;
else

remove the task from TEMP_AGENDA and QUE;
next TEMP_AGENDA of the same processor;
insert it into QUE;
end if;
end if;
end loop;
end if;

if REARRANGE_P then
ADJUST_PRECEDENCE;
create another solution TEMP_AGENDA for the new precedence list;
end if;
TEST_SCHEDULE; -- output "TEMP_COST"
if TEMP_COST < BEST_COST then
    BEST_COST := TEMP_COST;
duplicate TEMP_AGENDA to BEST_AGENDA;
end if;
if TEMP_COST = 0 then
    FEASIBLE_SOLUTION := true;
elsif REARRANGE_P or else
    TEMP_COST <= PENALTY_COST or else
        random number < ANNEAL_FUNCTION then
    ACCEPT_COUNT := ACCEPT_COUNT + 1;
PENALTY_COST := TEMP_COST;
duplicate TEMP_AGENDA to AGENDA;
else
    duplicate AGENDA to TEMP_AGENDA;
end if;
TRIAL_COUNT := TRIAL_COUNT + 1;
end loop;
T := T * COOL_FACTOR;
end loop;
AGENDA := BEST_AGENDA;
PENALTY_COST := BEST_COST;
end ANNEAL_PROCESS;

-- The TEMP_AGENDA is a temporary schedule for the use of the annealing process. Its cost is called TEMP_COST.

-- REARRANGE_P is a boolean constant. Its value is “true” when the schedule cannot be annealed any more and the precedence list needs to be adjusted.

-- A time slot is a time interval where the processor is free from any tasks. If a task’s MET is less than or equal to this interval, the task can be scheduled during this interval.
V. CONCLUSIONS

A. RESULTS FROM THE STATIC SCHEDULER

Four examples of the input PSDL text files (atomic.info) are presented in appendix A in forms of tables and graphs. Each table lists the name (operator name), index number (No.), maximum execution time (MET), finish_within (Within), period (P) and the number of instances (N) in the HBL for each operator to be scheduled. The index number (No.) and the number of instances (N) are derived entries from the static scheduler. Each diagram in appendix A is a precedence graph for each case. Operators are represented by circles and their index numbers are shown in the circles. The latencies between operators are also shown in the diagrams by the numbers above the edges between circles.

The results from the static scheduler for each case of test data are shown in appendix B. The schedules on each processor are listed in separate tables. Each table lists the index number (OP), the instance number (IN), the starting time (START TIME), the stop time (STOP TIME), the lower bound (LOWER) and the upper bound (UPPER) of the starting time for each operator.

For case 1 and case 2 test data, the feasible schedules were found by all three algorithms. But in cases 3 and 4, feasible schedules can not be found when the earliest deadline first algorithm is used. The tasks which violate the timing constraints are highlighted in the table.

When using the simulated annealing algorithm, the initial solutions for case 1 and 2 are feasible. In other words, there is no need to use the annealing process to adjust the schedules. But in case 3 and 4, the initial solutions are not feasible. After the annealing process, the feasible schedules are found. Furthermore, during the
annealing process, case 3 needs to adjust the PRECEDENCE_LIST several times to obtain the feasible schedule. The proposed heuristic algorithm, based on the simulated annealing approach, appears to be the best compromise between simple-minded and exponential time algorithms implemented in CAPS.

B. SUMMARY

The thesis presents the research conducted to develop a static scheduler for hard real-time tasks in multiprocessor systems. Three scheduling algorithms were developed in the new static scheduler for CAPS. The *earliest starting time first* algorithm produces the schedule according to the earliest possible starting time of each instance of operators. The instance with the smallest earliest possible starting time (the smallest LOWER bound) will be scheduled before any others. In a similar way, the *earliest deadline first* algorithm produces the schedule according to the deadline of each instance of operators. The instance with the most urgent deadline, i.e. the smallest UPPER bound, will be scheduled before any others.

The *simulated annealing algorithm* first produces an initial solution based on the topological ordering (PRECEDENCE_LIST) of the operators. If the initial solution is feasible, there is no need to anneal the schedule, otherwise, the annealing process is used. The annealing process starts to find out the first instance, whose starting time is greater than its upper bound (i.e., missing its deadline) of the schedule on each processor. From these instances, the annealing process then starts to adjust each instance by choosing the instance with the smallest lower bound of the starting time (LOWER). During the annealing process, the ordering (precedence) list of the operators may be required to adjust in order to create another new solution.
Any feasible schedule produced by these scheduling algorithms guarantees that both timing and precedence constraints are met, and the objective of the static scheduler is achieved.

C. FUTURE WORK

This is the first work to implement the static scheduling algorithms for PSDL tasks on multiprocessor systems. Future research is required for identification of possible weaknesses. The continued work is recommended in the following areas:

- **Modifying Proposed Algorithms Using Better Heuristics**
  Most heuristic methods suffer from several shortcomings such as the difficulty in assuring the accuracies of solutions [KAN84]. If one algorithm can be proven to be better than the other, which had been proven to be optimal, then this algorithm can also be called optimal. The rule of thumb can be applied to all scientific inventions, including scheduling problems. The better the understanding of the problem, the more opportunities to invent a heuristic solution.

- **Modifying the “Top-Sort” Procedure to Obtain a Better Ordering List**
  Since in most hard real-time systems, there exists more than one topological ordering of operators where there are cases in which one ordering may produce a feasible schedule while another will not. It will speed up the existing simulated annealing algorithm if the “top_sort” procedure is good enough to produce a topological ordering of operators such that there is no need to adjust this ordering during the annealing process as frequently as in the present implementation. The improvement of this procedure is recommended by using a similar method of critical path, introduced in [HUS90], with the consideration of giving a weight to each operator.
• Changing the Assumptions of Scheduling Problem

Different assumptions can lead to different results. For instance, tasks are assumed to be non-preemptive in this research, but they could be preemptive. There are still many open problems to be considered, such as periodic or non-periodic, any constraints or not, whether the precedence graph is a tree or network, scheduling tasks on centralized or on distributed system, and so forth.

Real-time systems have substantial amounts of knowledge concerning the characteristics of the application and the environment built into the system. A majority of today’s systems assume that much of this knowledge is available a priori and, hence, are based on static nature of many of these systems contributes to their high cost and inflexibility. The next generation hard real-time systems must be designed to be dynamic and flexible.

Where as a large proportion of currently implemented hard real-time systems are static in nature, by necessity, next-generation systems will have to adopt solutions that are more dynamic and flexible. This is because we believe that such systems will be large and complex and that they will function in environments that are dynamic while being physically distributed. More important, they will have to be maintainable and extensible due to their evolving nature and projected long lifetimes.
APPENDIX A. EXAMPLES OF TEST DATA

1. Case 1

<table>
<thead>
<tr>
<th>operator name</th>
<th>No.</th>
<th>MET</th>
<th>Within</th>
<th>P</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP_1</td>
<td>1</td>
<td>2000</td>
<td>9000</td>
<td>10000</td>
<td>3</td>
</tr>
<tr>
<td>OP_2</td>
<td>2</td>
<td>2000</td>
<td>10000</td>
<td>15000</td>
<td>2</td>
</tr>
<tr>
<td>OP_3</td>
<td>3</td>
<td>2000</td>
<td>12000</td>
<td>15000</td>
<td>2</td>
</tr>
<tr>
<td>OP_4</td>
<td>4</td>
<td>2000</td>
<td>20000</td>
<td>30000</td>
<td>1</td>
</tr>
<tr>
<td>OP_5</td>
<td>5</td>
<td>1000</td>
<td>8000</td>
<td>10000</td>
<td>3</td>
</tr>
<tr>
<td>OP_6</td>
<td>6</td>
<td>1000</td>
<td>12000</td>
<td>15000</td>
<td>2</td>
</tr>
<tr>
<td>OP_7</td>
<td>7</td>
<td>3000</td>
<td>18000</td>
<td>30000</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram](image)
2. Case 2

<table>
<thead>
<tr>
<th>operator name</th>
<th>No.</th>
<th>MET</th>
<th>Within</th>
<th>P</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP_1</td>
<td>1</td>
<td>2000</td>
<td>9000</td>
<td>10000</td>
<td>3</td>
</tr>
<tr>
<td>OP_2</td>
<td>2</td>
<td>1000</td>
<td>10000</td>
<td>15000</td>
<td>2</td>
</tr>
<tr>
<td>OP_3</td>
<td>3</td>
<td>5000</td>
<td>15000</td>
<td>30000</td>
<td>1</td>
</tr>
<tr>
<td>OP_4</td>
<td>4</td>
<td>1000</td>
<td>15000</td>
<td>30000</td>
<td>1</td>
</tr>
<tr>
<td>OP_5</td>
<td>5</td>
<td>3000</td>
<td>11000</td>
<td>15000</td>
<td>2</td>
</tr>
<tr>
<td>OP_6</td>
<td>6</td>
<td>1000</td>
<td>12000</td>
<td>15000</td>
<td>2</td>
</tr>
<tr>
<td>OP_7</td>
<td>7</td>
<td>1000</td>
<td>18000</td>
<td>30000</td>
<td>1</td>
</tr>
<tr>
<td>OP_8</td>
<td>8</td>
<td>1000</td>
<td>10000</td>
<td>15000</td>
<td>2</td>
</tr>
</tbody>
</table>
## 3. Case 3

<table>
<thead>
<tr>
<th>operator name</th>
<th>No.</th>
<th>MET</th>
<th>Within</th>
<th>P</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMS_LINKS</td>
<td>1</td>
<td>100</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>PARSE_INPUT_FILE</td>
<td>2</td>
<td>250</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>DECIDE_FOR_ARCHIVING</td>
<td>3</td>
<td>100</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>EXTRACT_TRACKS</td>
<td>4</td>
<td>150</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>FILTER_COMMS_TRACKS</td>
<td>5</td>
<td>500</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>ADD_COMMS_TRACK</td>
<td>6</td>
<td>100</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>CREATE_SENSOR_DATA</td>
<td>7</td>
<td>100</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>ANALYZE_SENSOR_DATA</td>
<td>8</td>
<td>250</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>PREPARE_SENSOR_TRACK</td>
<td>9</td>
<td>250</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>FILTER_SENSOR_TRACKS</td>
<td>10</td>
<td>500</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>ADD_SENSOR_TRACK</td>
<td>11</td>
<td>500</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>CREATE_POSITION_DATA</td>
<td>12</td>
<td>500</td>
<td>-</td>
<td>3000</td>
<td>7</td>
</tr>
<tr>
<td>MONITOR_OWNSHIP_POSIT</td>
<td>13</td>
<td>500</td>
<td>-</td>
<td>3000</td>
<td>7</td>
</tr>
<tr>
<td>WEAPONS_SYSTEMS</td>
<td>14</td>
<td>100</td>
<td>-</td>
<td>3000</td>
<td>7</td>
</tr>
<tr>
<td>WEAPONS_INTERFACE</td>
<td>15</td>
<td>100</td>
<td>-</td>
<td>3000</td>
<td>7</td>
</tr>
<tr>
<td>PREPARE_PERIODIC_REPORT</td>
<td>16</td>
<td>500</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>MAKE_ROUTING</td>
<td>17</td>
<td>300</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>FORWARD_FOR_TRANSMISSION</td>
<td>18</td>
<td>100</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
<tr>
<td>CONVERT_TO_TEXT_FILE</td>
<td>19</td>
<td>100</td>
<td>-</td>
<td>7000</td>
<td>3</td>
</tr>
</tbody>
</table>
## 4. Case 4

<table>
<thead>
<tr>
<th>operator name</th>
<th>No.</th>
<th>MET</th>
<th>Within</th>
<th>P</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMS_LINKS</td>
<td>1</td>
<td>1200</td>
<td>-</td>
<td>10000</td>
<td>6</td>
</tr>
<tr>
<td>PARSE_INPUT_FILE</td>
<td>2</td>
<td>500</td>
<td>-</td>
<td>10000</td>
<td>6</td>
</tr>
<tr>
<td>DECIDE_FOR_ARCHIVING</td>
<td>3</td>
<td>500</td>
<td>-</td>
<td>10000</td>
<td>6</td>
</tr>
<tr>
<td>EXTRACT_TRACKS</td>
<td>4</td>
<td>500</td>
<td>-</td>
<td>10000</td>
<td>6</td>
</tr>
<tr>
<td>FILTER_COMMS_TRACKS</td>
<td>5</td>
<td>500</td>
<td>-</td>
<td>10000</td>
<td>6</td>
</tr>
<tr>
<td>ADD_COMMS_TRACK</td>
<td>6</td>
<td>500</td>
<td>-</td>
<td>10000</td>
<td>6</td>
</tr>
<tr>
<td>CREATE_SENSOR_DATA</td>
<td>7</td>
<td>800</td>
<td>-</td>
<td>20000</td>
<td>3</td>
</tr>
<tr>
<td>ANALYZE_SENSOR_DATA</td>
<td>8</td>
<td>500</td>
<td>-</td>
<td>20000</td>
<td>3</td>
</tr>
<tr>
<td>PREPARE_SENSOR_TRACK</td>
<td>9</td>
<td>500</td>
<td>-</td>
<td>20000</td>
<td>3</td>
</tr>
<tr>
<td>FILTER_SENSOR_TRACKS</td>
<td>10</td>
<td>500</td>
<td>-</td>
<td>20000</td>
<td>3</td>
</tr>
<tr>
<td>ADD_SENSOR_TRACK</td>
<td>11</td>
<td>500</td>
<td>-</td>
<td>20000</td>
<td>3</td>
</tr>
<tr>
<td>CREATE_POSITION_DATA</td>
<td>12</td>
<td>800</td>
<td>-</td>
<td>20000</td>
<td>3</td>
</tr>
<tr>
<td>MONITOR_OWNSHIP_POSITION</td>
<td>13</td>
<td>500</td>
<td>-</td>
<td>20000</td>
<td>3</td>
</tr>
<tr>
<td>WEAPONS_INTERFACE</td>
<td>14</td>
<td>500</td>
<td>-</td>
<td>5000</td>
<td>12</td>
</tr>
<tr>
<td>WEAPONS_SYSTEMS</td>
<td>15</td>
<td>500</td>
<td>-</td>
<td>5000</td>
<td>12</td>
</tr>
<tr>
<td>MAKE_ROUTING</td>
<td>16</td>
<td>500</td>
<td>-</td>
<td>15000</td>
<td>4</td>
</tr>
<tr>
<td>FORWARD_FOR_TRANSMISSION</td>
<td>17</td>
<td>500</td>
<td>-</td>
<td>15000</td>
<td>4</td>
</tr>
<tr>
<td>CONVERT_TO_TEXT_FILE</td>
<td>18</td>
<td>800</td>
<td>-</td>
<td>15000</td>
<td>4</td>
</tr>
<tr>
<td>CONVERT_TO_TEXT_FILE</td>
<td>19</td>
<td>800</td>
<td>-</td>
<td>15000</td>
<td>4</td>
</tr>
</tbody>
</table>
APPENDIX B. OUTPUT SCHEDULES FOR TEST DATA

1. Case 1

The HBL := 30000

**********************************************************************************
** the Earliest Starting Time First Scheduling Algorithm **
**********************************************************************************

A feasible solution found; Elapsed time = 0.02 sec.

PROCESSOR # 1

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>18000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>12000</td>
<td>10000</td>
<td>17000</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>17000</td>
<td>18000</td>
<td>17000</td>
<td>28000</td>
</tr>
</tbody>
</table>

PROCESSOR # 2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3000</td>
<td>6000</td>
<td>3000</td>
<td>18000</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>15000</td>
<td>17000</td>
<td>15000</td>
<td>25000</td>
</tr>
</tbody>
</table>

PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>8000</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>13000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>15000</td>
<td>17000</td>
<td>15000</td>
<td>23000</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>22000</td>
<td>23000</td>
<td>22000</td>
<td>29000</td>
</tr>
</tbody>
</table>

PROCESSOR # 4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>7000</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>9000</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>12000</td>
<td>13000</td>
<td>12000</td>
<td>19000</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>22000</td>
<td>20000</td>
<td>27000</td>
</tr>
</tbody>
</table>
**the Earliest Deadline First Scheduling Algorithm**

A feasible schedule found; Elapsed time = 0.03 sec.

**PROCESSOR # 1**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>9000</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>12000</td>
<td>13000</td>
<td>12000</td>
<td>19000</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>22000</td>
<td>23000</td>
<td>22000</td>
<td>29000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 2**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>4000</td>
<td>5000</td>
<td>4000</td>
<td>15000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>15000</td>
<td>17000</td>
<td>15000</td>
<td>23000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 3**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>8000</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2000</td>
<td>4000</td>
<td>0</td>
<td>18000</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5000</td>
<td>8000</td>
<td>5000</td>
<td>20000</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>15000</td>
<td>17000</td>
<td>15000</td>
<td>25000</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>19000</td>
<td>20000</td>
<td>19000</td>
<td>30000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 4**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>7000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>12000</td>
<td>10000</td>
<td>17000</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>22000</td>
<td>20000</td>
<td>27000</td>
</tr>
</tbody>
</table>
**the Simulated Annealing Scheduling Algorithm**

A feasible schedule found; Elapsed time = 0.02 sec.

### PROCESSOR # 1

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>18000</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>17000</td>
<td>18000</td>
<td>17000</td>
<td>28000</td>
</tr>
</tbody>
</table>

### PROCESSOR # 2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>9000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>15000</td>
<td>17000</td>
<td>15000</td>
<td>23000</td>
</tr>
</tbody>
</table>

### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>8000</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3000</td>
<td>6000</td>
<td>3000</td>
<td>18000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>12000</td>
<td>10000</td>
<td>17000</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>12000</td>
<td>13000</td>
<td>12000</td>
<td>19000</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>22000</td>
<td>20000</td>
<td>27000</td>
</tr>
</tbody>
</table>

### PROCESSOR # 4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>7000</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>13000</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>15000</td>
<td>17000</td>
<td>15000</td>
<td>25000</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>22000</td>
<td>23000</td>
<td>22000</td>
<td>29000</td>
</tr>
</tbody>
</table>
2. Case 2

The HBL := 30000

The Earliest Starting Time First Scheduling Algorithm

A feasible schedule found; Elapsed time = 0.01 sec.

**PROCESSOR # 1**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>16000</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7000</td>
<td>8000</td>
<td>7000</td>
<td>24000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>17000</td>
<td>18000</td>
<td>17000</td>
<td>26000</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>25000</td>
<td>26000</td>
<td>25000</td>
<td>34000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 2**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>2000</td>
<td>7000</td>
<td>2000</td>
<td>12000</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>10000</td>
<td>11000</td>
<td>10000</td>
<td>19000</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>22000</td>
<td>20000</td>
<td>27000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 3**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>11000</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7000</td>
<td>10000</td>
<td>7000</td>
<td>15000</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>18000</td>
<td>19000</td>
<td>18000</td>
<td>29000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 4**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>7000</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3000</td>
<td>4000</td>
<td>3000</td>
<td>14000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>12000</td>
<td>10000</td>
<td>17000</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>22000</td>
<td>25000</td>
<td>22000</td>
<td>30000</td>
</tr>
</tbody>
</table>
A feasible schedule found; Elapsed time = 0.01 sec.

** the Earliest Deadline First Scheduling Algorithm  **

<table>
<thead>
<tr>
<th>PROCESSOR # 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROCESSOR # 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROCESSOR # 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROCESSOR # 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
**the Simulated Annealing Scheduling Algorithm**

A feasible schedule found; Elapsed time = 0.02 sec.

**PROCESSOR # 1**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>7000</td>
<td>8000</td>
<td>7000</td>
<td>24000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>17000</td>
<td>18000</td>
<td>17000</td>
<td>26000</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>22000</td>
<td>20000</td>
<td>27000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 2**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>2000</td>
<td>7000</td>
<td>2000</td>
<td>12000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>12000</td>
<td>10000</td>
<td>17000</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>25000</td>
<td>26000</td>
<td>25000</td>
<td>34000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 3**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>16000</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3000</td>
<td>4000</td>
<td>3000</td>
<td>14000</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>10000</td>
<td>11000</td>
<td>10000</td>
<td>19000</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>18000</td>
<td>19000</td>
<td>18000</td>
<td>29000</td>
</tr>
</tbody>
</table>

**PROCESSOR # 4**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>0</td>
<td>7000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2000</td>
<td>3000</td>
<td>2000</td>
<td>11000</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7000</td>
<td>10000</td>
<td>7000</td>
<td>15000</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>22000</td>
<td>25000</td>
<td>22000</td>
<td>30000</td>
</tr>
</tbody>
</table>
3. **Case 3**

The HBL := 21000

**the Earliest Starting Time First Scheduling Algorithm**

A feasible schedule found; Elapsed time = 0.02 sec.

**PROCESSOR # 1**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>2900</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>900</td>
<td>1150</td>
<td>900</td>
<td>7650</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1650</td>
<td>1900</td>
<td>1650</td>
<td>8400</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2650</td>
<td>2800</td>
<td>2650</td>
<td>9500</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>3000</td>
<td>3100</td>
<td>3000</td>
<td>5900</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>3600</td>
<td>3700</td>
<td>3600</td>
<td>6500</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>4300</td>
<td>4400</td>
<td>4300</td>
<td>11200</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>6600</td>
<td>6700</td>
<td>6600</td>
<td>9500</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>7300</td>
<td>7800</td>
<td>7300</td>
<td>9800</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>8650</td>
<td>8900</td>
<td>8650</td>
<td>15400</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>9200</td>
<td>9300</td>
<td>9200</td>
<td>16100</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>9650</td>
<td>9800</td>
<td>9650</td>
<td>16500</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>10300</td>
<td>10800</td>
<td>10300</td>
<td>12800</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>12000</td>
<td>12100</td>
<td>12000</td>
<td>14900</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>14000</td>
<td>14100</td>
<td>14000</td>
<td>20900</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>15000</td>
<td>15100</td>
<td>15000</td>
<td>17900</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>15400</td>
<td>15700</td>
<td>15400</td>
<td>22100</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>16300</td>
<td>16800</td>
<td>16300</td>
<td>18800</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>17400</td>
<td>17900</td>
<td>17400</td>
<td>23900</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>18600</td>
<td>18700</td>
<td>18600</td>
<td>21500</td>
</tr>
</tbody>
</table>

**PROCESSOR # 2**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>500</td>
<td>0</td>
<td>2500</td>
</tr>
</tbody>
</table>
## PROCESSOR # 2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1300</td>
<td>1550</td>
<td>1300</td>
<td>8050</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2050</td>
<td>2150</td>
<td>2050</td>
<td>8950</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>2800</td>
<td>2900</td>
<td>2800</td>
<td>9700</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>3400</td>
<td>3900</td>
<td>3400</td>
<td>9900</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>6000</td>
<td>6100</td>
<td>6000</td>
<td>8900</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7000</td>
<td>7100</td>
<td>7000</td>
<td>13900</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>8300</td>
<td>8550</td>
<td>8300</td>
<td>15050</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>9000</td>
<td>9100</td>
<td>9000</td>
<td>11900</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>9400</td>
<td>9900</td>
<td>9400</td>
<td>15900</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>11300</td>
<td>11400</td>
<td>11300</td>
<td>18200</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>13300</td>
<td>13800</td>
<td>13300</td>
<td>15800</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>14900</td>
<td>15150</td>
<td>14900</td>
<td>21650</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>15600</td>
<td>15700</td>
<td>15600</td>
<td>18500</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>16200</td>
<td>16300</td>
<td>16200</td>
<td>23100</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>16800</td>
<td>16900</td>
<td>16800</td>
<td>23700</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>18000</td>
<td>18100</td>
<td>18000</td>
<td>20900</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>193000</td>
<td>19800</td>
<td>19300</td>
<td>21800</td>
</tr>
</tbody>
</table>

## PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>6900</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>600</td>
<td>700</td>
<td>600</td>
<td>3500</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1400</td>
<td>1700</td>
<td>1400</td>
<td>8100</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>2200</td>
<td>2300</td>
<td>2200</td>
<td>9100</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>3000</td>
<td>3500</td>
<td>3000</td>
<td>5500</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>4300</td>
<td>4800</td>
<td>4300</td>
<td>6800</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7000</td>
<td>7100</td>
<td>7000</td>
<td>13900</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>7900</td>
<td>8150</td>
<td>7900</td>
<td>14650</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>9000</td>
<td>9500</td>
<td>9000</td>
<td>11500</td>
</tr>
</tbody>
</table>

69
## PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>2</td>
<td>9800</td>
<td>9900</td>
<td>9800</td>
<td>16700</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>10400</td>
<td>10900</td>
<td>10400</td>
<td>16900</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>12600</td>
<td>12700</td>
<td>12600</td>
<td>15500</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>14100</td>
<td>14600</td>
<td>14100</td>
<td>20600</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>15300</td>
<td>15550</td>
<td>15300</td>
<td>22050</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>16050</td>
<td>16150</td>
<td>16050</td>
<td>22950</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>16650</td>
<td>16800</td>
<td>16650</td>
<td>23500</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>17300</td>
<td>17800</td>
<td>17300</td>
<td>23800</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>18300</td>
<td>18400</td>
<td>18300</td>
<td>25200</td>
</tr>
</tbody>
</table>

## PROCESSOR # 4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>6900</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>100</td>
<td>600</td>
<td>0</td>
<td>6500</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1300</td>
<td>1800</td>
<td>1300</td>
<td>3800</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2400</td>
<td>2900</td>
<td>2400</td>
<td>8900</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3300</td>
<td>3800</td>
<td>3300</td>
<td>9800</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>6000</td>
<td>6500</td>
<td>6000</td>
<td>8500</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>7100</td>
<td>7600</td>
<td>7100</td>
<td>13600</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>8400</td>
<td>8700</td>
<td>8400</td>
<td>15100</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9050</td>
<td>9150</td>
<td>9050</td>
<td>15950</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>9600</td>
<td>9700</td>
<td>9600</td>
<td>12500</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>10300</td>
<td>10800</td>
<td>10300</td>
<td>16800</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>12000</td>
<td>12500</td>
<td>12000</td>
<td>14500</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>14000</td>
<td>14100</td>
<td>14000</td>
<td>20900</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>15000</td>
<td>15500</td>
<td>15000</td>
<td>17500</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>15650</td>
<td>15900</td>
<td>15650</td>
<td>22400</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>16400</td>
<td>16900</td>
<td>16400</td>
<td>22900</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>18000</td>
<td>18500</td>
<td>18000</td>
<td>20500</td>
</tr>
</tbody>
</table>
** the Earliest Deadline First Scheduling Algorithm **

feasible schedule not found; Elapsed time = 0.02 sec.

### PROCESSOR # 1

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>1</td>
<td>1300</td>
<td>1800</td>
<td>1300</td>
<td>3800</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>4300</td>
<td>4800</td>
<td>4300</td>
<td>6800</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>7300</td>
<td>7800</td>
<td>7300</td>
<td>9800</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>10300</td>
<td>10800</td>
<td>10300</td>
<td>12800</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>10800</td>
<td>11160</td>
<td>10800</td>
<td>15700</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>13300</td>
<td>13800</td>
<td>13300</td>
<td>15800</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>13800</td>
<td>13900</td>
<td>11300</td>
<td>18200</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>15600</td>
<td>15700</td>
<td>15600</td>
<td>18500</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>15700</td>
<td>15950</td>
<td>13850</td>
<td>20600</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>18000</td>
<td>18100</td>
<td>18000</td>
<td>20900</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>18100</td>
<td>18400</td>
<td>17500</td>
<td>22700</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>18400</td>
<td>18900</td>
<td>16850</td>
<td>23350</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>18900</td>
<td>19000</td>
<td>18900</td>
<td>25200</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>20100</td>
<td>20600</td>
<td>20100</td>
<td>26600</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>21100</td>
<td>21200</td>
<td>21100</td>
<td>28000</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>27100</td>
<td>27600</td>
<td>27100</td>
<td>33600</td>
</tr>
</tbody>
</table>

### PROCESSOR # 2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>600</td>
<td>700</td>
<td>600</td>
<td>3500</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>700</td>
<td>1200</td>
<td>0</td>
<td>6500</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>3600</td>
<td>3700</td>
<td>3600</td>
<td>6500</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>6000</td>
<td>6100</td>
<td>6000</td>
<td>8900</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>6100</td>
<td>6350</td>
<td>4100</td>
<td>10850</td>
</tr>
</tbody>
</table>
### PROCESSOR # 2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>6350</td>
<td>6600</td>
<td>4400</td>
<td>11150</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>9000</td>
<td>9100</td>
<td>9000</td>
<td>11900</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>9100</td>
<td>9350</td>
<td>6800</td>
<td>13600</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9350</td>
<td>9450</td>
<td>7100</td>
<td>14000</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>9450</td>
<td>9550</td>
<td>7100</td>
<td>14000</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>12000</td>
<td>12500</td>
<td>12000</td>
<td>14500</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>12500</td>
<td>12650</td>
<td>9950</td>
<td>16800</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>12650</td>
<td>12750</td>
<td>10200</td>
<td>17100</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>15000</td>
<td>15100</td>
<td>15000</td>
<td>17900</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>15100</td>
<td>15600</td>
<td>13100</td>
<td>19600</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>18000</td>
<td>18500</td>
<td>18000</td>
<td>20500</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>18500</td>
<td>18600</td>
<td>16500</td>
<td>23400</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>18600</td>
<td>18750</td>
<td>16950</td>
<td>23800</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>18750</td>
<td>18850</td>
<td>17200</td>
<td>24100</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>20000</td>
<td>20250</td>
<td>20000</td>
<td>25150</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>21100</td>
<td>21200</td>
<td>21100</td>
<td>28000</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>23950</td>
<td>24100</td>
<td>23950</td>
<td>30800</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>30500</td>
<td>30600</td>
<td>30500</td>
<td>37400</td>
</tr>
</tbody>
</table>

### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>2900</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>3000</td>
<td>3500</td>
<td>3000</td>
<td>5500</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>3500</td>
<td>3800</td>
<td>2000</td>
<td>8700</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>6600</td>
<td>6700</td>
<td>6600</td>
<td>9500</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>9600</td>
<td>9700</td>
<td>9600</td>
<td>12500</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>12000</td>
<td>12100</td>
<td>12000</td>
<td>14900</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>12100</td>
<td>12600</td>
<td>9850</td>
<td>16350</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>12600</td>
<td>12700</td>
<td>10100</td>
<td>17000</td>
</tr>
</tbody>
</table>
### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>13550</td>
<td>13800</td>
<td>13550</td>
<td>17850</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>13900</td>
<td>14150</td>
<td>13900</td>
<td>18150</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>16300</td>
<td>16800</td>
<td>16300</td>
<td>18800</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>19300</td>
<td>19800</td>
<td>19300</td>
<td>21800</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>20150</td>
<td>20650</td>
<td>20150</td>
<td>26650</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>23850</td>
<td>24350</td>
<td>23850</td>
<td>30350</td>
</tr>
</tbody>
</table>

### PROCESSOR # 4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>500</td>
<td>0</td>
<td>2500</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>3000</td>
<td>3100</td>
<td>3000</td>
<td>5900</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3100</td>
<td>3200</td>
<td>0</td>
<td>6900</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3200</td>
<td>3300</td>
<td>0</td>
<td>6900</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>6000</td>
<td>6500</td>
<td>6000</td>
<td>8500</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>6500</td>
<td>6600</td>
<td>4300</td>
<td>11200</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>9000</td>
<td>9500</td>
<td>9000</td>
<td>11500</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>9500</td>
<td>10000</td>
<td>7700</td>
<td>14200</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>12600</td>
<td>12700</td>
<td>12600</td>
<td>15500</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>15000</td>
<td>15500</td>
<td>15000</td>
<td>17500</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>15500</td>
<td>16000</td>
<td>13150</td>
<td>19650</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>16000</td>
<td>16100</td>
<td>14650</td>
<td>21000</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>16100</td>
<td>16200</td>
<td>14100</td>
<td>21000</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>16200</td>
<td>16700</td>
<td>14700</td>
<td>21200</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>18600</td>
<td>18700</td>
<td>18600</td>
<td>21500</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>18700</td>
<td>18800</td>
<td>17100</td>
<td>24000</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>19650</td>
<td>19900</td>
<td>19650</td>
<td>24850</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>20850</td>
<td>21100</td>
<td>20850</td>
<td>27600</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>23500</td>
<td>23600</td>
<td>23500</td>
<td>30400</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>27150</td>
<td>27650</td>
<td>27150</td>
<td>33650</td>
</tr>
</tbody>
</table>
the Simulated Annealing Scheduling Algorithm

A feasible schedule found; Elapsed time = 0.25 sec.

**PROCESSOR # 1**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>2900</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1400</td>
<td>1700</td>
<td>1400</td>
<td>8100</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1700</td>
<td>1800</td>
<td>600</td>
<td>3500</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2550</td>
<td>2800</td>
<td>2550</td>
<td>9300</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2800</td>
<td>3050</td>
<td>1300</td>
<td>8050</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4150</td>
<td>4300</td>
<td>4150</td>
<td>11000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7000</td>
<td>7100</td>
<td>7000</td>
<td>13900</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>7900</td>
<td>8150</td>
<td>7900</td>
<td>14650</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>9000</td>
<td>9100</td>
<td>9000</td>
<td>11900</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>9800</td>
<td>9900</td>
<td>9800</td>
<td>16700</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>10550</td>
<td>10650</td>
<td>10550</td>
<td>17450</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>11800</td>
<td>12300</td>
<td>11800</td>
<td>18300</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>12600</td>
<td>12700</td>
<td>12600</td>
<td>15500</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>14100</td>
<td>14600</td>
<td>14100</td>
<td>20600</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>15000</td>
<td>15500</td>
<td>15000</td>
<td>17500</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>15600</td>
<td>15700</td>
<td>15600</td>
<td>18500</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>16300</td>
<td>16800</td>
<td>16300</td>
<td>18800</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>18300</td>
<td>18800</td>
<td>18300</td>
<td>24800</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>19300</td>
<td>19800</td>
<td>19300</td>
<td>21800</td>
</tr>
</tbody>
</table>

**PROCESSOR # 2**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>500</td>
<td>0</td>
<td>2500</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>2200</td>
<td>2300</td>
<td>2200</td>
<td>9100</td>
</tr>
</tbody>
</table>
### PROCESSOR # 2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>2</td>
<td>3000</td>
<td>3100</td>
<td>3000</td>
<td>5900</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>3600</td>
<td>3700</td>
<td>3600</td>
<td>6500</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>4300</td>
<td>4800</td>
<td>4300</td>
<td>10800</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>4800</td>
<td>5300</td>
<td>3000</td>
<td>5500</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>6000</td>
<td>6500</td>
<td>6000</td>
<td>8500</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>7100</td>
<td>7600</td>
<td>7100</td>
<td>13600</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>9000</td>
<td>9500</td>
<td>9000</td>
<td>11500</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>9600</td>
<td>9700</td>
<td>9600</td>
<td>12500</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>10300</td>
<td>10800</td>
<td>10300</td>
<td>12800</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>11300</td>
<td>11800</td>
<td>11300</td>
<td>17800</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>12000</td>
<td>12500</td>
<td>12000</td>
<td>14500</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>14000</td>
<td>14100</td>
<td>14000</td>
<td>20900</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>15000</td>
<td>15100</td>
<td>15000</td>
<td>17900</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>16200</td>
<td>16300</td>
<td>16200</td>
<td>23100</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>17300</td>
<td>17800</td>
<td>17300</td>
<td>23800</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>18000</td>
<td>18500</td>
<td>18000</td>
<td>20500</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>18600</td>
<td>18700</td>
<td>18600</td>
<td>21500</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>19800</td>
<td>19900</td>
<td>19800</td>
<td>26700</td>
</tr>
</tbody>
</table>

### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>6900</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1300</td>
<td>1800</td>
<td>1300</td>
<td>3800</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1800</td>
<td>2050</td>
<td>900</td>
<td>7650</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>3300</td>
<td>3800</td>
<td>3300</td>
<td>9800</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>5800</td>
<td>5900</td>
<td>5800</td>
<td>12700</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>6000</td>
<td>6100</td>
<td>6000</td>
<td>8900</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>6600</td>
<td>6700</td>
<td>6600</td>
<td>9500</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>8400</td>
<td>8700</td>
<td>8400</td>
<td>15100</td>
</tr>
</tbody>
</table>

75
### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>9550</td>
<td>9800</td>
<td>9550</td>
<td>16300</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9800</td>
<td>10050</td>
<td>8300</td>
<td>15050</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>11150</td>
<td>11300</td>
<td>11150</td>
<td>18000</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>12000</td>
<td>12100</td>
<td>12000</td>
<td>14900</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>14000</td>
<td>14100</td>
<td>14000</td>
<td>20900</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>14900</td>
<td>15150</td>
<td>14900</td>
<td>21650</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>16800</td>
<td>16900</td>
<td>16800</td>
<td>23700</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>17550</td>
<td>17650</td>
<td>17550</td>
<td>24450</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>18000</td>
<td>18100</td>
<td>18000</td>
<td>20900</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>18800</td>
<td>19300</td>
<td>18800</td>
<td>25300</td>
</tr>
</tbody>
</table>

### PROCESSOR # 4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>6900</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>100</td>
<td>600</td>
<td>0</td>
<td>6500</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>2800</td>
<td>2900</td>
<td>2800</td>
<td>9700</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3550</td>
<td>3650</td>
<td>3550</td>
<td>10450</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4800</td>
<td>5300</td>
<td>4800</td>
<td>11300</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>6100</td>
<td>6600</td>
<td>6100</td>
<td>6800</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7000</td>
<td>7100</td>
<td>7000</td>
<td>13900</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>7300</td>
<td>7800</td>
<td>7300</td>
<td>9800</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>9200</td>
<td>9300</td>
<td>9200</td>
<td>16100</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>10300</td>
<td>10800</td>
<td>10300</td>
<td>16800</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>12800</td>
<td>12900</td>
<td>12800</td>
<td>19700</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>13300</td>
<td>13800</td>
<td>13300</td>
<td>15800</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>15400</td>
<td>15700</td>
<td>15400</td>
<td>22100</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>16550</td>
<td>16800</td>
<td>16550</td>
<td>23300</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>16800</td>
<td>17050</td>
<td>15300</td>
<td>22050</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>18150</td>
<td>18300</td>
<td>18150</td>
<td>25000</td>
</tr>
</tbody>
</table>

76
4. Case 4

The HBL := 60000

******************************************************************************
** theEarliest Starting Time First Scheduling Algorithm **
******************************************************************************

A feasible solution found; Elapsed time = 0.03 sec.

PROCESSOR # 1

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>0</td>
<td>500</td>
<td>0</td>
<td>4500</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>500</td>
<td>1300</td>
<td>0</td>
<td>14200</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>2100</td>
<td>2600</td>
<td>2100</td>
<td>16600</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3400</td>
<td>3900</td>
<td>3400</td>
<td>12900</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>4600</td>
<td>5100</td>
<td>4600</td>
<td>24100</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6400</td>
<td>6900</td>
<td>6400</td>
<td>15900</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12400</td>
<td>12900</td>
<td>12400</td>
<td>21900</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>15400</td>
<td>15900</td>
<td>15400</td>
<td>24900</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>17100</td>
<td>17600</td>
<td>17100</td>
<td>31600</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>20000</td>
<td>20800</td>
<td>20000</td>
<td>39200</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>21600</td>
<td>22100</td>
<td>21600</td>
<td>41100</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>23600</td>
<td>24100</td>
<td>23600</td>
<td>43100</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>25400</td>
<td>25900</td>
<td>25400</td>
<td>34900</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>30000</td>
<td>30500</td>
<td>30000</td>
<td>34500</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>32100</td>
<td>32600</td>
<td>32100</td>
<td>46600</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>34100</td>
<td>34900</td>
<td>34100</td>
<td>48300</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>36400</td>
<td>36900</td>
<td>36400</td>
<td>45900</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>40000</td>
<td>40500</td>
<td>40000</td>
<td>44500</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>41000</td>
<td>41500</td>
<td>41000</td>
<td>45500</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>42600</td>
<td>43100</td>
<td>42600</td>
<td>62100</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>44600</td>
<td>45100</td>
<td>44600</td>
<td>64100</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>46000</td>
<td>46500</td>
<td>46000</td>
<td>50500</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>49100</td>
<td>49900</td>
<td>49100</td>
<td>63300</td>
</tr>
</tbody>
</table>
### Processor #1

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
<td>52400</td>
<td>52900</td>
<td>52400</td>
<td>61900</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>55400</td>
<td>55900</td>
<td>55400</td>
<td>64900</td>
</tr>
</tbody>
</table>

### Processor #2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>800</td>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1600</td>
<td>2100</td>
<td>1600</td>
<td>21100</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>3100</td>
<td>3600</td>
<td>3100</td>
<td>17600</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4400</td>
<td>4900</td>
<td>4400</td>
<td>13900</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>6000</td>
<td>6500</td>
<td>6000</td>
<td>10500</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>11000</td>
<td>11500</td>
<td>11000</td>
<td>15500</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>15000</td>
<td>15500</td>
<td>15000</td>
<td>19500</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>16400</td>
<td>16900</td>
<td>16400</td>
<td>25900</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>20000</td>
<td>20800</td>
<td>20000</td>
<td>39200</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>21600</td>
<td>22100</td>
<td>21600</td>
<td>41100</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>23400</td>
<td>23900</td>
<td>23400</td>
<td>32900</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>25000</td>
<td>25500</td>
<td>25000</td>
<td>29500</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>30000</td>
<td>31200</td>
<td>30000</td>
<td>38800</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>32400</td>
<td>32900</td>
<td>32400</td>
<td>41900</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>34400</td>
<td>34900</td>
<td>34400</td>
<td>43900</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>36000</td>
<td>36500</td>
<td>36000</td>
<td>40500</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>40000</td>
<td>41200</td>
<td>40000</td>
<td>48800</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>42400</td>
<td>42900</td>
<td>42400</td>
<td>51900</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>44400</td>
<td>44900</td>
<td>44400</td>
<td>53900</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>45500</td>
<td>46300</td>
<td>45500</td>
<td>59700</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>48100</td>
<td>48600</td>
<td>48100</td>
<td>62600</td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>51000</td>
<td>51500</td>
<td>51000</td>
<td>55500</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>55000</td>
<td>55500</td>
<td>55000</td>
<td>59500</td>
</tr>
</tbody>
</table>
### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>800</td>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1000</td>
<td>1500</td>
<td>1000</td>
<td>5500</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2400</td>
<td>2900</td>
<td>2400</td>
<td>11900</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>3600</td>
<td>4100</td>
<td>3600</td>
<td>23100</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>5000</td>
<td>5500</td>
<td>5000</td>
<td>9500</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>11200</td>
<td>10000</td>
<td>18800</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>14400</td>
<td>14900</td>
<td>14400</td>
<td>23900</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>16000</td>
<td>16500</td>
<td>16000</td>
<td>20500</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>19100</td>
<td>19900</td>
<td>19100</td>
<td>33300</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>20000</td>
<td>20500</td>
<td>20000</td>
<td>24500</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>21000</td>
<td>21500</td>
<td>21000</td>
<td>25500</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>22600</td>
<td>23100</td>
<td>22600</td>
<td>42100</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>24600</td>
<td>25100</td>
<td>24600</td>
<td>44100</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>26400</td>
<td>26900</td>
<td>26400</td>
<td>35900</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>31000</td>
<td>31500</td>
<td>31000</td>
<td>35500</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>33400</td>
<td>33900</td>
<td>33400</td>
<td>42900</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>35400</td>
<td>35900</td>
<td>35400</td>
<td>44900</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>40000</td>
<td>40800</td>
<td>40000</td>
<td>59200</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>41600</td>
<td>42100</td>
<td>41600</td>
<td>61100</td>
</tr>
<tr>
<td>10</td>
<td>3*</td>
<td>43600</td>
<td>44100</td>
<td>43600</td>
<td>63100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>45400</td>
<td>45900</td>
<td>45400</td>
<td>54900</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>47100</td>
<td>47600</td>
<td>47100</td>
<td>61600</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>50000</td>
<td>50500</td>
<td>50000</td>
<td>54500</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>53400</td>
<td>53900</td>
<td>53400</td>
<td>62900</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>56000</td>
<td>56500</td>
<td>56000</td>
<td>60500</td>
</tr>
</tbody>
</table>

### PROCESSOR # 4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1200</td>
<td>0</td>
<td>8800</td>
</tr>
</tbody>
</table>

79
<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>1</td>
<td>1600</td>
<td>2100</td>
<td>1600</td>
<td>21100</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2600</td>
<td>3100</td>
<td>2600</td>
<td>22100</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>4100</td>
<td>4900</td>
<td>4100</td>
<td>18300</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5400</td>
<td>5900</td>
<td>5400</td>
<td>14900</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>10000</td>
<td>10500</td>
<td>10000</td>
<td>14500</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>13400</td>
<td>13900</td>
<td>13400</td>
<td>22900</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>15500</td>
<td>16300</td>
<td>15500</td>
<td>29700</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>18100</td>
<td>18600</td>
<td>18100</td>
<td>32600</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>21200</td>
<td>20000</td>
<td>28800</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>22400</td>
<td>22900</td>
<td>22400</td>
<td>31900</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>24400</td>
<td>24900</td>
<td>24400</td>
<td>33900</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>26000</td>
<td>26500</td>
<td>26000</td>
<td>30500</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>30500</td>
<td>31300</td>
<td>30500</td>
<td>44700</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>33100</td>
<td>33600</td>
<td>33100</td>
<td>47600</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>35000</td>
<td>35500</td>
<td>35000</td>
<td>39500</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>40000</td>
<td>40800</td>
<td>40000</td>
<td>59200</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>41600</td>
<td>42100</td>
<td>41600</td>
<td>61100</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>43400</td>
<td>43900</td>
<td>43400</td>
<td>52900</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>45000</td>
<td>45500</td>
<td>45000</td>
<td>49500</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>46400</td>
<td>46900</td>
<td>46400</td>
<td>55900</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>50000</td>
<td>51200</td>
<td>50000</td>
<td>58800</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>54400</td>
<td>54900</td>
<td>54400</td>
<td>63900</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>56400</td>
<td>56900</td>
<td>56400</td>
<td>65900</td>
</tr>
</tbody>
</table>
**the Earliest Deadline First Scheduling Algorithm**

feasible schedule not found; Elapsed time = 0.03 sec.

**PRECESSOR # 1**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2</td>
<td>5000</td>
<td>5500</td>
<td>5000</td>
<td>9500</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>11000</td>
<td>11500</td>
<td>11000</td>
<td>15500</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>13400</td>
<td>13900</td>
<td>13400</td>
<td>22900</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>20000</td>
<td>20500</td>
<td>20000</td>
<td>24500</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>26000</td>
<td>26500</td>
<td>26000</td>
<td>30500</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>26500</td>
<td>27000</td>
<td>17900</td>
<td>37400</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>35000</td>
<td>35500</td>
<td>35000</td>
<td>39500</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>41000</td>
<td>41500</td>
<td>41000</td>
<td>45500</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>43400</td>
<td>43900</td>
<td>43400</td>
<td>52900</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>50000</td>
<td>50500</td>
<td>50000</td>
<td>54500</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>56000</td>
<td>56500</td>
<td>56000</td>
<td>60500</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>57100</td>
<td>57600</td>
<td>57100</td>
<td>68000</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>58700</td>
<td>59200</td>
<td>58700</td>
<td>78200</td>
</tr>
</tbody>
</table>

**PROCESSOR # 2**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1200</td>
<td>0</td>
<td>8800</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2400</td>
<td>2900</td>
<td>2400</td>
<td>11900</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4400</td>
<td>4900</td>
<td>4400</td>
<td>13900</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5400</td>
<td>5900</td>
<td>5400</td>
<td>14900</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6400</td>
<td>6900</td>
<td>6400</td>
<td>15900</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>6900</td>
<td>7700</td>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>7700</td>
<td>8500</td>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>15000</td>
<td>15500</td>
<td>15000</td>
<td>19500</td>
</tr>
</tbody>
</table>
### Processor #2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>Start Time</th>
<th>Stop Time</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>5</td>
<td>21000</td>
<td>21500</td>
<td>21000</td>
<td>25500</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>22400</td>
<td>22900</td>
<td>22400</td>
<td>31900</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>22900</td>
<td>23700</td>
<td>18900</td>
<td>33100</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>24400</td>
<td>24900</td>
<td>24400</td>
<td>33900</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>25400</td>
<td>25900</td>
<td>25400</td>
<td>34900</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>31000</td>
<td>31500</td>
<td>31000</td>
<td>35500</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>32400</td>
<td>32900</td>
<td>32400</td>
<td>41900</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>34400</td>
<td>34900</td>
<td>34400</td>
<td>43900</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>35400</td>
<td>35900</td>
<td>35400</td>
<td>44900</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>36400</td>
<td>36900</td>
<td>36400</td>
<td>45900</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>36900</td>
<td>37700</td>
<td>32500</td>
<td>46700</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>37700</td>
<td>38200</td>
<td>27500</td>
<td>47000</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>38200</td>
<td>39000</td>
<td>33900</td>
<td>48100</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>40000</td>
<td>41200</td>
<td>40000</td>
<td>48800</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>42400</td>
<td>42900</td>
<td>42400</td>
<td>51900</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>44400</td>
<td>44900</td>
<td>44400</td>
<td>53900</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>45400</td>
<td>45900</td>
<td>45400</td>
<td>54900</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>46400</td>
<td>46900</td>
<td>46400</td>
<td>55900</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>46900</td>
<td>47400</td>
<td>39100</td>
<td>57400</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>50000</td>
<td>51200</td>
<td>50000</td>
<td>58800</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>51200</td>
<td>52000</td>
<td>47500</td>
<td>61700</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>53400</td>
<td>53900</td>
<td>53400</td>
<td>62900</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>55400</td>
<td>55900</td>
<td>55400</td>
<td>64900</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>55900</td>
<td>56700</td>
<td>47700</td>
<td>66900</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>57500</td>
<td>58000</td>
<td>57500</td>
<td>68800</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>67500</td>
<td>68000</td>
<td>67500</td>
<td>87000</td>
</tr>
</tbody>
</table>
### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1</td>
<td>1000</td>
<td>1500</td>
<td>1000</td>
<td>5500</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3400</td>
<td>3900</td>
<td>3400</td>
<td>12900</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>3900</td>
<td>4700</td>
<td>0</td>
<td>14200</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>10000</td>
<td>10500</td>
<td>10000</td>
<td>14500</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>10500</td>
<td>11000</td>
<td>5500</td>
<td>20000</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>16000</td>
<td>16500</td>
<td>16000</td>
<td>20500</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>16500</td>
<td>17000</td>
<td>11500</td>
<td>26000</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>21200</td>
<td>20000</td>
<td>28800</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>21200</td>
<td>22000</td>
<td>17500</td>
<td>31700</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>23400</td>
<td>23900</td>
<td>23400</td>
<td>32900</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>30000</td>
<td>30500</td>
<td>30000</td>
<td>34500</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>36000</td>
<td>36500</td>
<td>36000</td>
<td>40500</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>36500</td>
<td>37300</td>
<td>26900</td>
<td>46100</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>37300</td>
<td>38100</td>
<td>27700</td>
<td>46900</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>38100</td>
<td>38600</td>
<td>38100</td>
<td>48000</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>38900</td>
<td>39400</td>
<td>38900</td>
<td>48800</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>45000</td>
<td>45500</td>
<td>45000</td>
<td>49500</td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>51000</td>
<td>51500</td>
<td>51000</td>
<td>55500</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>52400</td>
<td>52900</td>
<td>52400</td>
<td>61900</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>52900</td>
<td>53700</td>
<td>48900</td>
<td>63100</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>54400</td>
<td>54900</td>
<td>54400</td>
<td>63900</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>54900</td>
<td>55400</td>
<td>54500</td>
<td>65000</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>56400</td>
<td>56900</td>
<td>56400</td>
<td>65900</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>62500</td>
<td>63300</td>
<td>62500</td>
<td>76700</td>
</tr>
</tbody>
</table>

### PROCESSOR # 4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>0</td>
<td>500</td>
<td>0</td>
<td>4500</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>6000</td>
<td>6500</td>
<td>6000</td>
<td>10500</td>
</tr>
<tr>
<td>OP</td>
<td>IN</td>
<td>START TIME</td>
<td>STOP TIME</td>
<td>LOWER</td>
<td>UPPER</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>------------</td>
<td>-----------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>11200</td>
<td>10000</td>
<td>18800</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12400</td>
<td>12900</td>
<td>12400</td>
<td>21900</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>14400</td>
<td>14900</td>
<td>14400</td>
<td>23900</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>15400</td>
<td>15900</td>
<td>15400</td>
<td>24900</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>16400</td>
<td>16900</td>
<td>16400</td>
<td>25900</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>16900</td>
<td>17400</td>
<td>8500</td>
<td>28000</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>17400</td>
<td>17900</td>
<td>9300</td>
<td>28800</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>25000</td>
<td>25500</td>
<td>25000</td>
<td>29500</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>25500</td>
<td>26000</td>
<td>24500</td>
<td>35000</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>26400</td>
<td>26900</td>
<td>26400</td>
<td>35900</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>30000</td>
<td>31200</td>
<td>30000</td>
<td>38800</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>31200</td>
<td>31700</td>
<td>26500</td>
<td>41000</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>40000</td>
<td>40500</td>
<td>40000</td>
<td>44500</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>40500</td>
<td>41000</td>
<td>39800</td>
<td>50000</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>46000</td>
<td>46500</td>
<td>46000</td>
<td>50500</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>46500</td>
<td>47000</td>
<td>41500</td>
<td>56000</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>47000</td>
<td>47500</td>
<td>38700</td>
<td>58200</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>55000</td>
<td>55500</td>
<td>55000</td>
<td>59500</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>55500</td>
<td>56300</td>
<td>46900</td>
<td>66100</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>56300</td>
<td>56800</td>
<td>47500</td>
<td>67000</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>56800</td>
<td>57300</td>
<td>56500</td>
<td>71000</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>57900</td>
<td>58400</td>
<td>57900</td>
<td>77400</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>78700</td>
<td>79200</td>
<td>78700</td>
<td>98200</td>
</tr>
</tbody>
</table>
A feasible schedule found; Elapsed time = 0.12 sec.

**the Simulated Annealing Scheduling Algorithm**

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>1</td>
<td>0</td>
<td>500</td>
<td>0</td>
<td>4500</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>500</td>
<td>1300</td>
<td>0</td>
<td>14200</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1300</td>
<td>1800</td>
<td>1000</td>
<td>5500</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1800</td>
<td>2300</td>
<td>1600</td>
<td>21100</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>2300</td>
<td>2800</td>
<td>1600</td>
<td>21100</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>4300</td>
<td>4800</td>
<td>4300</td>
<td>23800</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5800</td>
<td>6300</td>
<td>5800</td>
<td>15300</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>17100</td>
<td>17600</td>
<td>17100</td>
<td>31600</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>20000</td>
<td>20800</td>
<td>20000</td>
<td>39200</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>21000</td>
<td>21500</td>
<td>21000</td>
<td>25500</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>24300</td>
<td>24800</td>
<td>24300</td>
<td>43800</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>25000</td>
<td>25500</td>
<td>25000</td>
<td>29500</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>26000</td>
<td>26500</td>
<td>26000</td>
<td>30500</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>32100</td>
<td>32600</td>
<td>32100</td>
<td>46600</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>35000</td>
<td>35500</td>
<td>35000</td>
<td>39500</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>36000</td>
<td>36500</td>
<td>36000</td>
<td>40500</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>40000</td>
<td>40800</td>
<td>40000</td>
<td>59200</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>41000</td>
<td>41500</td>
<td>41000</td>
<td>45500</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>44300</td>
<td>44800</td>
<td>44300</td>
<td>63800</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>44800</td>
<td>45300</td>
<td>44800</td>
<td>54300</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>46000</td>
<td>46500</td>
<td>46000</td>
<td>50500</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>48100</td>
<td>48600</td>
<td>48100</td>
<td>62600</td>
</tr>
</tbody>
</table>
### PROCESSOR # 2

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>800</td>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>3100</td>
<td>3600</td>
<td>3100</td>
<td>17600</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>5300</td>
<td>5800</td>
<td>5300</td>
<td>24800</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>6000</td>
<td>6500</td>
<td>6000</td>
<td>10500</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>15500</td>
<td>16300</td>
<td>15500</td>
<td>29700</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>16800</td>
<td>17300</td>
<td>16800</td>
<td>26300</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>17800</td>
<td>18300</td>
<td>17800</td>
<td>27300</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>20000</td>
<td>21200</td>
<td>20000</td>
<td>28800</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>21600</td>
<td>22100</td>
<td>21600</td>
<td>41100</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>22400</td>
<td>22900</td>
<td>22400</td>
<td>31900</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>24800</td>
<td>25300</td>
<td>24800</td>
<td>34300</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>27800</td>
<td>28300</td>
<td>27800</td>
<td>37300</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>30000</td>
<td>30500</td>
<td>30000</td>
<td>34500</td>
</tr>
<tr>
<td>14</td>
<td>7</td>
<td>31000</td>
<td>31500</td>
<td>31000</td>
<td>35500</td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>33100</td>
<td>33600</td>
<td>33100</td>
<td>47600</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>40000</td>
<td>40800</td>
<td>40000</td>
<td>59200</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>43300</td>
<td>43800</td>
<td>43300</td>
<td>62800</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>45000</td>
<td>45500</td>
<td>45000</td>
<td>49500</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>47100</td>
<td>47600</td>
<td>47100</td>
<td>61600</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>49100</td>
<td>49900</td>
<td>49100</td>
<td>63300</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>54800</td>
<td>55300</td>
<td>54800</td>
<td>64300</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>55800</td>
<td>56300</td>
<td>55800</td>
<td>65300</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>56800</td>
<td>57300</td>
<td>56800</td>
<td>66300</td>
</tr>
</tbody>
</table>

### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>800</td>
<td>0</td>
<td>19200</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>2100</td>
<td>2600</td>
<td>2100</td>
<td>16600</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>3300</td>
<td>3800</td>
<td>3300</td>
<td>22800</td>
</tr>
</tbody>
</table>
### PROCESSOR # 3

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>START TIME</th>
<th>STOP TIME</th>
<th>LOWER</th>
<th>UPPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>3800</td>
<td>4300</td>
<td>2400</td>
<td>11900</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4800</td>
<td>5300</td>
<td>4800</td>
<td>14300</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>7800</td>
<td>8300</td>
<td>7800</td>
<td>17300</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10000</td>
<td>11200</td>
<td>10000</td>
<td>18800</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12400</td>
<td>12900</td>
<td>12400</td>
<td>21900</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>14800</td>
<td>15300</td>
<td>14800</td>
<td>24300</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>15800</td>
<td>16300</td>
<td>15800</td>
<td>15300</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>19100</td>
<td>19900</td>
<td>19100</td>
<td>33300</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>20000</td>
<td>20500</td>
<td>20000</td>
<td>24500</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>21600</td>
<td>22100</td>
<td>21600</td>
<td>41100</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>23000</td>
<td>23500</td>
<td>23000</td>
<td>44800</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>25800</td>
<td>26300</td>
<td>25800</td>
<td>35300</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>26800</td>
<td>27300</td>
<td>26800</td>
<td>36300</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>30000</td>
<td>31200</td>
<td>30000</td>
<td>38800</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>34100</td>
<td>34900</td>
<td>34100</td>
<td>48300</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>35800</td>
<td>36300</td>
<td>35800</td>
<td>45300</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>40000</td>
<td>41200</td>
<td>40000</td>
<td>48800</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>41600</td>
<td>42100</td>
<td>41600</td>
<td>61100</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>42400</td>
<td>42900</td>
<td>42400</td>
<td>51900</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>45500</td>
<td>46300</td>
<td>45500</td>
<td>59700</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>50000</td>
<td>51200</td>
<td>50000</td>
<td>58800</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>52400</td>
<td>52900</td>
<td>52400</td>
<td>61900</td>
</tr>
<tr>
<td>14</td>
<td>12</td>
<td>56000</td>
<td>56500</td>
<td>56000</td>
<td>60500</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>57800</td>
<td>58300</td>
<td>57800</td>
<td>67300</td>
</tr>
</tbody>
</table>
### Processor #4

<table>
<thead>
<tr>
<th>OP</th>
<th>IN</th>
<th>Start Time</th>
<th>Stop Time</th>
<th>Lower</th>
<th>Upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1200</td>
<td>0</td>
<td>8800</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>4100</td>
<td>4900</td>
<td>4100</td>
<td>18300</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>5000</td>
<td>5500</td>
<td>5000</td>
<td>9500</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>6800</td>
<td>7300</td>
<td>6800</td>
<td>16300</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>10000</td>
<td>10500</td>
<td>10000</td>
<td>14500</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>11000</td>
<td>11500</td>
<td>11000</td>
<td>15500</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>15000</td>
<td>15500</td>
<td>15000</td>
<td>19500</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>16000</td>
<td>16500</td>
<td>16000</td>
<td>20500</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>18100</td>
<td>18600</td>
<td>18100</td>
<td>32600</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>20000</td>
<td>20800</td>
<td>20000</td>
<td>39200</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>23300</td>
<td>23800</td>
<td>23300</td>
<td>42800</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>30500</td>
<td>31300</td>
<td>30500</td>
<td>44700</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>32400</td>
<td>32900</td>
<td>32400</td>
<td>41900</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>34800</td>
<td>35300</td>
<td>34800</td>
<td>44300</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>36800</td>
<td>37300</td>
<td>36800</td>
<td>46300</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>37800</td>
<td>38300</td>
<td>37800</td>
<td>47300</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>40000</td>
<td>40500</td>
<td>40000</td>
<td>44500</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>41600</td>
<td>42100</td>
<td>41600</td>
<td>61100</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>45300</td>
<td>45800</td>
<td>45300</td>
<td>64800</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>45800</td>
<td>46300</td>
<td>45800</td>
<td>55300</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>46800</td>
<td>47300</td>
<td>46800</td>
<td>56300</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>47800</td>
<td>48300</td>
<td>47800</td>
<td>57300</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>50000</td>
<td>50500</td>
<td>50000</td>
<td>54500</td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>51000</td>
<td>51500</td>
<td>51000</td>
<td>55500</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>55000</td>
<td>55500</td>
<td>55000</td>
<td>59500</td>
</tr>
</tbody>
</table>
APPENDIX C. ADA CODES OF THE MODIFIED PACKAGES

1. DATA

with VSTRINGS;
with SEQUENCES;
with TEXT_IO;

-- This package contains all of the global declarations and definitions
-- of data structures that are necessary for the Static Scheduler

package DATA is
package VARSTRING is new VSTRINGS(80);
use VARSTRING;
subtype OPERATOR_ID is VSTRING;
subtype VALUE is NATURAL;
subtype MET is VALUE;
subtype MRT is VALUE;
subtype MCP is VALUE;
subtype PERIOD is VALUE;
subtype WITHIN is VALUE;
subtype STARTS is VALUE;
subtype STOPS is VALUE;
subtype LOWERS is VALUE;
subtype UPPERS is VALUE;
Exception-Operator: OPERATOR_ID;
TEST_VERIFIED BOOLEAN true;
NOP NATURAL 4; -- number of available processors (6/25/92)
type PROCESSOR_ARRAY is array (1..NOP) of VALUE; --(8/12/92)
type OPERATOR is
   record
      THE_OPERATOR_ID : OPERATOR_ID;
      THE_MET : MET := 0;
      THE_MRT : MRT := 0;
      THE_MCP : MCP := 0;
      THE_PERIOD : PERIOD := 0;
      THE_WITHIN : WITHIN := 0;
   end record;

package V_LISTS is new SEQUENCES(OPERATOR); use V_LISTS;
type SCHEDULE_INPUTS is
  record
    THE_OPERATOR : INTEGER;
    THE_START : STARTS := 0;
    THE_STOP : STOPS := 0;
    THE_LOWER : LOWERS := 0;
    THE_UPPER : UPPERS := 0;
    THE_INSTANCE : INTEGER := 1;
  end record;
end;

package SCHEDULE_INPUTS_LIST is new SEQUENCES(SCHEDULE_INPUTS);

type SCHEDULE_ARRAY is array (1..NOP) of SCHEDULE_INPUTS_LIST;

package NODE_LIST is new SEQUENCES(INTEGER);

NONCRITS TEXTIO.FILE_TYPE;
AGOUTFILE TEXTIO.FILE_TYPE;
INPUT TEXTIO.FILE_MODE := TEXT_IO.IN_FILE;
OUTPUT TEXTIO.FILE_MODE := TEXT_IO.OUT_FILE;
CurrentValue VALUE;
NewWord VARSTRING.VSTRING;
Cur_Opt OPERATOR;
OP_COUNT INTEGER;
OP_LIST V_LISTS.LIST;
-- the following global variables are new for multiprocessor scheduling (7/30/92)
OP_NUM INTEGER;
PARENT_COUNT INTEGER;
PARENT_NUM INTEGER;
CHILD_COUNT INTEGER;
PROCESSOR_NUM INTEGER;
PARENT_OP OPERATOR;
PARENT_LIST NODE_LIST.LIST;
CHILD_LIST NODE_LIST.LIST;
LIST_HEAD NODE_LIST.LIST;
TEMP OPERATOR;
NEW_INPUT SCHEDULE_INPUTS;
ADDLNODE SCHEDULE_INPUTS;
FIRST SCHEDULE_INPUTS;
BEST SCHEDULE_INPUTS;
end DATA;
2. NEW_DATA_STRUCTURES

with TEXT_IO;
with DATA; use DATA;

-- This package contains the specifications for a graph data structure that can represent an
-- acyclic graph. Functions and procedures exist to access the information that is stored in the graph
-- as well as to find out the relationships between vertices in the graph.

generic

package NEW_DATA_STRUCTURES is

  type GRAPH (SIZE: INTEGER) is limited private;
  type GRAPH_LINK is access GRAPH;
  THE_GRAPH       : GRAPH_LINK;

  procedure PRODUCE_OP_ARRAY ( INFO_LIST : in out V_LISTS.LIST;
                                COUNT    : in INTEGER);
    -- Transfer operator info from linked list to array

  function OP_POSITION ( OP_NAME : VARSTRING.V STRING;
                        COUNT    : INTEGER) return INTEGER;
    -- Given an operator name return the operator's position in the array

  procedure PRODUCE_OP_MATRIX (COUNT : in INTEGER);
    -- Create a matrix to represent the acyclic graph of operator relationship

  function OP_RETURN (OP_POSITION: INTEGER) return OPERATOR;
    -- Given an operator's position in the array, return the operator

  function IS_PARENT (OP_1, OP_2: INTEGER) return BOOLEAN;
    -- Return true if OP_1 is a parent of OP_2 or if OP_1 is OP_2

  function IS_CHILD (OP_1, OP_2: INTEGER) return BOOLEAN;
    -- Return true if OP_1 is a child of OP_2 or if OP_1 is OP_2

  procedure RETURN_PARENT_LIST ( PARENT_LIST : in out NODE_LIST.LIST;
                                 OP       : in INTEGER;
                                 COUNT    : in out INTEGER);
    -- Return a list of all the parents of an operator
procedure RETURN_CHILD_LIST ( CHILD_LIST : in out NODE_LIST.LIST;
    OP       : in INTEGER;
    COUNT    : in out INTEGER);
-- Return a list of all the children of an operator

procedure FREE_GRAPH (A_GRAPH : in out GRAPH_LINK);
-- Free the memory space used by the graph

function LATENCY(OP1,OP2: INTEGER) return INTEGER: -- (7/10/92)

function PIPELINE(OP: INTEGER) return BOOLEAN: -- (7/10/92)

private

type INFO_ARRAY is array (INTEGER range <>) of OPERATOR;

type MATRIX_OP_INFO is
    record
        PARENT : INTEGER := -1;
        CHILD  : INTEGER := -1;
        DELAY_PIPELINE : INTEGER := 0; -- (7/10/92)
    end record;

type MATRIX is array (INTEGER range <>,INTEGER range <>) of MATRIX_OP_INFO;

type GRAPH (SIZE: INTEGER) is
    record
        OP_ARRAY : INFO_ARRAY(0..SIZE);
        OP_MATRIX : MATRIX(0..SIZE, 0..SIZE);
    end record;
end NEW_DATA_STRUCTURES;
package body NEW_DATA_STRUCTURES is

pragma LINK_WITH ("heaplib.sparc.ar");

procedure FREE is new UNCHECKED_DEALLOCATION(GRAPH, GRAPH_LINK);

package int_io is new TEXT_IO.INTEGER_IO(INTEGER); use int_io;

procedure PRODUCE_OP_ARRAY (INFO_LIST : in out VLISTS.LIST; COUNT : in INTEGER) is

HEAD : VLISTS.LIST := INFO_LIST;

function MAKE_START_NODE return OPERATOR is

START_OP : OPERATOR;

begin
START_OP.THE_OPERATOR_ID := VARSTRING.VSTR("DUMMY START NODE");
START_OP.THE_MET := 0;
START_OP.THE_MRT := 0;
START_OP.THE_MCP := 0;
START_OP.THE_WITHIN := 0;
return START_OP;
end MAKE_START_NODE;

begin
for INDEX in reverse 1..COUNT loop
  THE_GRAPH.OP_ARRAY(INDEX) := V_LISTS.VALUE(INFO_LIST);
  V_LISTS.NEXT(INFO_LIST);
end loop;
THE_GRAPH.OP_ARRAY(0) := MAKE_START_NODE;
V_LISTS.FREE_LIST(HEAD); --* THIS LIST IS NO LONGER NEEDED.
end PRODUCE_OP_ARRAY;
function OP_POSITION ( OP_NAME : VARSTRING.VSTRING; COUNT : INTEGER ) return INTEGER is
begin
  for INDEX in 1..COUNT loop
    if VARSTRING.EQUAL(OP_NAME, THE_GRAPH.OP_ARRAY(INDEX).THE_OPERATOR_ID) then return INDEX;
  end if;
end loop;
return -1; -- Operator is external since it is not in the array.
end OP_POSITION;

procedure PRODUCE_OP_MATRIX (COUNT:in INTEGER) is

  COLUMN,
  ROW,
  PARENT_OP,
  CHILD_OP : INTEGER;
  LINK : constant VARSTRING.VSTRING := VARSTRING.VSTR("LINK");

procedure INITIALIZE ( COUNT : in INTEGER;
OP_MATRIX : in out MATRIX) is
begin
  for ROW in 0..COUNT loop
    THE_GRAPH.OP_MATRIX(ROW,ROW).PARENT := ROW;
    THE_GRAPH.OP_MATRIX(ROW,ROW).CHILD := ROW;
  end loop;
end INITIALIZE; --(6/5/92, S)

procedure INITIALIZE_START_NODE ( COUNT : in INTEGER;
OP_MATRIX : in out MATRIX) is
begin
  for INDEX in 0..COUNT loop
    if THE_GRAPH.OP_MATRIX(INDEX, INDEX).PARENT = INDEX then
      THE_GRAPH.OP_MATRIX(INDEX,INDEX).PARENT := 0;
      THE_GRAPH.OP_MATRIX(0,INDEX).CHILD := THE_GRAPH.OP_MATRIX(0,0).CHILD;
      THE_GRAPH.OP_MATRIX(0,INDEX).CHILD := INDEX;
      THE_GRAPH.OP_MATRIX(0,INDEX).PARENT := INDEX;
    end if;
  end loop;
end INITIALIZE_START_NODE;
begin -- PRODUCE_OP_MATRIX
  TEXT_IO.OPEN (AG_OUTFILE, INPUT, "atomic.info");
  INITIALIZE (COUNT, THE_GRAPH.OP_MATRIX);
  VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
  while not TEXT_IO.END_OF_FILE (AG_OUTFILE) loop
    if VARSTRING.EQUAL (New_Word, LINK) then
      -- keyword "LINK"
      TEXT_IO.SKIP_LINE (AG_OUTFILE);
      -- skip LINK name
      VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
      PARENT_OP := OP_POSITION (NEW_WORD, DATA.OP_COUNT);
      int_io.GET (AG_OUTFILE, Current_Value); -- (7/28/92)
      TEXT_IO.SKIP_LINE (AG_OUTFILE);
      VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
      CHILD_OP := OP_POSITION (New_Word, DATA.OP_COUNT);
      if (PARENT_OP /= -1 and CHILD_OP /= -1) then
        if THE_GRAPH.OP_MATRIX (PARENT_OP, CHILD_OP).DELAY_PIPELINE < Current_Value then
          THE_GRAPH.OP_MATRIX (PARENT_OP, CHILD_OP).DELAY_PIPELINE := Current_Value;
        end if;
        if (THE_GRAPH.OP_MATRIX (PARENT_OP, CHILD_OP).CHILD = -1) then
          THE_GRAPH.OP_MATRIX (PARENT_OP, PARENT_OP).CHILD := CHILD_OP;
        end if;
      end if;
    end if;
  end loop;
  TEXT_IO.CLOSE (AG_OUTFILE);
  INITIALIZE_STARTNODE (COUNT, THE_GRAPH.OP_MATRIX);
end PRODUCE_OP_MATRIX;

function OP_RETURN (OP_POSITION: INTEGER) return OPERATOR is
  OP: OPERATOR;
begin
  OP := THE_GRAPH.OP_ARRAY (OP_POSITION);
  return OP;
end OP_RETURN;
function IS_PARENT (OP_1, OP_2: INTEGER) return BOOLEAN is
    -- Return true if OP_1 is a parent of OP_2 or if OP_1 is OP_2

    PARENT: BOOLEAN := false;

begin
    if OP_1 = OP_2 then
        PARENT := true;
    elsif THE_GRAPH.OPMATRIX(OP_1, OP_2).PARENT /= -1 then
        PARENT := true;
    end if;

    return PARENT;
end IS_PARENT;

function IS_CHILD (OP_1, OP_2: INTEGER) return BOOLEAN is
    -- Return true if OP_1 is a child of OP_2 or if OP_1 is OP_2

    CHILD: BOOLEAN := false;

begin
    if OP_1 = OP_2 then
        CHILD := true;
    elsif THE_GRAPH.OPMATRIX(OP_2, OP_1).CHILD /= -1 then
        CHILD := true;
    end if;

    return CHILD;
end IS_CHILD;

procedure RETURN_PARENT_LIST (PARENT_LIST : in out NODE_LIST.LIST;
                               OP : in INTEGER;
                               COUNT : in out INTEGER) is

    ROW: INTEGER := OP;

begin
    COUNT := 0;
    while THE_GRAPH.OPMATRIX(ROW, OP).PARENT /= OP loop
        NODE_LIST.ADD(THE_GRAPH.OPMATRIX(ROW, OP).PARENT, PARENT_LIST);
        COUNT := COUNT + 1;
        ROW := THE_GRAPH.OPMATRIX(ROW, OP).PARENT;
    end loop;
end RETURN_PARENT_LIST;
procedure RETURN_CHILD_LIST ( CHILD_LIST : in out NODE_LIST.LIST;
OP : in INTEGER;
COUNT : in out INTEGER) is

COLUMN: INTEGER := OP;
begin
COUNT := 0;
while THEGRAPH.OP_MATRIX(OP, COLUMN).CHILD /= OP loop
    NODE_LIST.ADD(THEGRAPH.OP_MATRIX(OP, COLUMN).CHILD, CHILDLIST);
    COUNT := COUNT + 1;
    COLUMN := THEGRAPH.OP_MATRIX(OP, COLUMN).CHILD;
end loop;
end RETURN_CHILD_LIST;

procedure FREE_GRAPH (A_GRAPH : in out GRAPH_LINK) is
begin
    FREE(A_GRAPH);
end FREE_GRAPH;

function LATENCY (OP1, OP2: INTEGER) return INTEGER is -- (7/10/92)
beg
return THEGRAPH.OP_MATRIX(OP1,OP2).DELAY_PIPELINE;
end LATENCY;

function PIPELINE(OP: INTEGER) return BOOLEAN is --(7/10/92)
beg
if THEGRAPH.OP_MATRIX(OP,OP).DELAY_PIPELINE = 1 then
    return true;
else
    return false;
end if;
end PIPELINE;
end NEW_DATA_STRUCTURES;
3. DIAGNOSTICS

with DATA; use DATA;

package DIAGNOSTICS is

    procedure OUTPUT_SCHEDULE (AGENDA: in SCHEDULE_ARRAY); --(7/21/92)

    procedure OUTPUT_OP_ID (OP_EL: in V_LISTS.LIST);

    procedure OUTPUT_HARMONIC_BLOCK_LENGTH (H_B_LENGTH: in INTEGER);

    procedure OUTPUTPRECEDENCE_LIST (PREC_LIST: in NODE_LIST.LIST);

diagnositics;

with DATA; use DATA;
with TEXT_10; use TEXT_10; --(7/21/92)
with FRONTEND; use FRONT_END; --(7/21/92)

package body DIAGNOSTICS is

package int_io is new integer_io(integer); use int_io;

procedure OUTPUT_SCHEDULE (AGENDA: in SCHEDULE_ARRAY) is --(7/21/92)

    AGENDA_1: SCHEDULE_ARRAY := AGENDA; --(7/21/92)

begin
    for PROCESSOR_NUM in 1..NOP loop
        SETCOL(1);
        PUT("THE PROCESSOR NO. : ");
        int_io.PUT(PROCESSOR_NUM, WIDTH=>2);
        NEW_LINE;
        SETCOL(3);
        PUT("OP# ");
        SETCOL(8);
        PUT("INSTANCE # ");
        SETCOL(20);
        PUT("START TIME ");
        SET_COL(32);
        --

        for OP_EL in AGENDA_1 loop
            for INSTANCE_NUM in 1..MAX_INST loop
                SETCOL(1);
                PUT("INST: ");
                int_io.PUT(INSTANCE_NUM, WIDTH=>2);
                NEW_LINE;
                SETCOL(3);
                PUT("OP# ");
                SETCOL(8);
                PUT("START TIME ");
                SET_COL(32);
                --

            end loop;

        end loop;

    end loop;

    for OP_EL in AGENDA_1 loop
        for INSTANCE_NUM in 1..MAX_INST loop
            SETCOL(1);
            PUT("INST: ");
            int_io.PUT(INSTANCE_NUM, WIDTH=>2);
            NEW_LINE;
            SETCOL(3);
            PUT("OP# ");
            SETCOL(8);
            PUT("START TIME ");
            SET_COL(32);
            --

        end loop;

    end loop;

end DIAGNOSTICS;
PUT("STOP TIME");
SET_COL(46);
PUT("LOWER");
SET_COL(56);
PUT("UPPER ");
NEW_LINE;

while SCHEDULE_INPUTS_LIST.NON_EMPTY(AGENDA_1(PROCESSOR_NUM)) loop
SET_COL(3);
int_io.PUT(SCHEDULE_INPUTS_LIST.VALUE(AGENDA_1(PROCESSOR_NUM)).
THE_OPERATOR, width => 3);
SET_COL(6);
PUT("-");
SET_COL(7);
int_io.PUT(SCHEDULE_INPUTS_LIST.VALUE(AGENDA_1(PROCESSOR_NUM)).
THE_INSTANCE, width => 10);
SET_COL(20);
int_io.PUT(SCHEDULE_INPUTS_LIST.VALUE(AGENDA_1(PROCESSOR_NUM)).
THE_START, width => 10);
SET_COL(31);
int_io.PUT(SCHEDULE_INPUTS_LIST.VALUE(AGENDA_1(PROCESSOR_NUM)).
THE_STOP, width => 10);
SET_COL(41);
int_io.PUT(SCHEDULE_INPUTS_LIST.VALUE(AGENDA_1(PROCESSOR_NUM)).
THE_LOWER, width => 10);
SET_COL(51);
int_io.PUT(SCHEDULE_INPUTS_LIST.VALUE(AGENDA_1(PROCESSOR_NUM)).
THE_UPPER, width => 10);
NEW_LINE;
SCHEDULE_INPUTS_LIST.NEXT(AGENDA_1(PROCESSOR_NUM));
end loop;
end loop;
end OUTPUT_SCHEDULE;

procedure OUTPUT_HARMONIC_BLOCK_LENGTH (H_B_LENGTH: in INTEGER) is

begin
PUT("The Harmonic Block Length is:");
int_io.PUT(H_B_LENGTH);
NEW_LINE;
end OUTPUT_HARMONIC_BLOCK_LENGTH;

99
procedure OUTPUT_OP_ID (OP_EL: in V_LISTS.LIST) is

   TRAVERSE: V_LISTS.LIST := OP_EL;

begin
   VARSTRING.PUT(V_LISTS.VALUE(TRAVERSE).THE_OPERATOR_ID);
   end OUTPUT_OP_ID:

procedure OUTPUT_PRECEDENCE_LIST (PREC_LIST: in NODE_LIST.LIST) is

   TRAVERSE: NODE_LIST.LIST := PREC_LIST:

begin
   while NODE_LIST.NON_EMPTY(TRAVERSE) loop
      VARSTRING.PUT(NEWGRAPH.OP_RETUR(NODE_LIST.VALUE(TRAVERSE)).
                     THE_OPERATOR_ID);
      NEW_LINE:
      NODE_LIST.NEXT(TRAVERSE);
   end loop:
   end OUTPUT_PRECEDENCE_LIST;

end DIAGNOSTICS;
APPENDIX D. ADA CODES OF THE NEW PACKAGES

1. UTILITY_PKG

with DATA: use DATA;

package UTILITY_PKG is

   M : CONSTANT := 2**13;
   subtype NUMBER is FLOAT range 0.0..1.0;
   subtype SEED is INTEGER range 1..M-1;

   procedure RANDOM_INITIALIZE(START_VALUE: in SEED); -- initialize random number generator.

   function RANDOM_NEXT return NUMBER; -- gives a random number between 0 and 1.

   procedure DETERMINE_THE_UPPER(TEMP : in OPERATOR;
                                  NEW_NODE : in out SCHEDULE_INPUTS);

   procedure DETERMINE_START_STOP( TEMP : in OPERATOR;
                                    PROC_NUM : in out INTEGER;
                                    NEW_NODE : in out SCHEDULE_INPUTS;
                                    PROC_STOP : in out PROCESSOR_ARRAY);

   procedure CREATE_ADDL_NODE(NEW_NODE : in SCHEDULE_INPUTS;
                               TEMP : in OPERATOR;
                               ADDL_NODE : in out SCHEDULE_INPUTS);

   procedure TEST_SCHEDULE( HBL : in INTEGER
                           AGENDA : in SCHEDULE_ARRAY;
                           COST : in out INTEGER);

   function ANNEAL_FUNCTION( COST_1 : INTEGER;\n                            COST_2 : INTEGER;
                            CURRENT_TEMP : FLOAT ) return FLOAT;

   procedure ADJUST_PRECEDFNCE( COUNT : in INTEGER;
                                P_LIST : in out NODE_LIST.LIST);

end UTILITY_PKG;
with DATA; use DATA;
with FRONT_END; use FRONT_END;
with PRIORITY_QUEUES;
with MATH; use MATH;
  with TEXT_IO; use TEXT_IO;
  with DIAGNOSTICS;

package body UTILITY_PKG is

  U : NATURAL;
  K : CONSTANT := 5**5;

package int_io is new TEXT_IO.INTEGER_IO(INTEGER);

procedure RANDOM_INITIALIZE(START_VALUE : in SEED) is

begin
  U := START_VALUE;
end RANDOM_INITIALIZE;

function RANDOM_NEXT return NUMBER is

begin
  U := U * K mod M;
  return FLOAT(U)/FLOAT(M);
end RANDOM_NEXT;

procedure DETERMINE_THE_UPPER(TEMP : in OPERATOR;
                               NEW_NODE : in out SCHEDULE_INPUTS) is

begin

  if TEMP.THE_WITHIN /= 0 then
    NEW_NODE.THE_UPPER :=
      NEW_NODE.THE_LOWER + TEMP.THE_WITHIN - TEMP.THE_MET;
  else
    NEW_NODE.THE_UPPER :=
      NEW_NODE.THE_LOWER + TEMP.THE_PERIOD - TEMP.THE_MET;
  end if;

end DETERMINE_THE_UPPER;
procedure DETERMINE_START_STOP( TEMP : in OPERATOR;
  PROC_NUM : in out INTEGER;
  NEW_NODE : in out SCHEDULE_INPUTS;
  PROC_STOP : in out PROCESSOR_ARRAY) is

  PROC_FREE: VALUE := NEW_NODE.THE_LOWER;

begin

  NEW_NODE.THE_START := NEW_NODE.THE_LOWER;
  for I in 1..NOP loop
    if PROC_STOP(I) <= NEW_NODE.THE_LOWER then
      if PROC_STOP(I) <= PROC_FREE then
        NEW_NODE.THE_START := NEW_NODE.THE_LOWER;
        PROC_FREE := PROC_STOP(I);
        PROC_NUM := I;
      end if;
    else
      if I = 1 then
        NEW_NODE.THE_START := PROC_STOP(1);
        PROC_NUM := 1;
      elsif PROC_STOP(I) <= NEW_NODE.THE_START then
        NEW_NODE.THE_START := PROC_STOP(I);
        PROC_NUM := I;
      end if;
    end if;
  end loop;
  NEW_NODE.THE_STOP := NEW_NODE.THE_START + TEMP.THE_MET;
  PROC_STOP(PROC_NUM) := NEW_NODE.THE_STOP;
end DETERMINE_START_STOP;

procedure CREATE_ADDL_NODE(NEW_NODE : in SCHEDULE_INPUTS;
  TEMP in OPERATOR;
  ADDL_NODE : in out SCHEDULE_INPUTS) is

begin

  ADDL_NODE.THE_OPERATOR := NEW_NODE.THE_OPERATOR;
  ADDL_NODE.THE_START := TEMP.THE_PERIOD * NEW_NODE.THE_INSTANCE;
  -- store synchronous information
  ADDL_NODE.THE_INSTANCE := NEW_NODE.THE_INSTANCE + 1;
  if not NEW_GRAPH.PIPELINE(NEW_NODE.THE_OPERATOR)
    and then ADDL_NODE.THE_LOWER < NEW_NODE.THE_STOP then
    ADDL_NODE.THE_LOWER := NEW_NODE.THE_STOP;
  end if;
end CREATE_ADDL_NODE;
procedure TEST_SCHEDULE( HBL : in INTEGER;
  AGENDA: in SCHEDULE_ARRAY;
  COST : in out INTEGER) is

V : SCHEDULE_ARRAY := AGENDA;
PREVIOUS : SCHEDULE_INPUTSLIST.LIST := null;

begin
  COST := 0;
  for I in 1..NOP loop
    while SCHEDULE_INPUTSLIST.NON_EMPTY(V(I)) loop
      if COST < (SCHEDULE_INPUTSLIST.VALUE(V(I)).THE_START
        - SCHEDULE_INPUTSLIST.VALUE(V(I)).THE_UPPER) then
        COST := SCHEDULE_INPUTSLIST.VALUE(V(I)).THE_START
        - SCHEDULE_INPUTSLIST.VALUE(V(I)).THE_UPPER:
      end if;
      PREVIOUS := V(I);
      SCHEDULE_INPUTSLIST.NEXT(V(I));
    end loop;
    if SCHEDULE_INPUTSLIST.VALUE(PREVIOUS).THE_STOP > HBL and then
      COST < (SCHEDULE_INPUTSLIST.VALUE(PREVIOUS).THE_STOP - HBL) then
      COST := SCHEDULE_INPUTSLIST.VALUE(PREVIOUS).THE_STOP - HBL;
    end if;
  end loop;
end TEST_SCHEDULE:

function ANNEAL_FUNCTION( COST_1 : INTEGER;
  COST_2 : INTEGER;
  CURRENT_TEMP : FLOAT) return FLOAT is

DELTA_C : FLOAT;

begin
  DELTA_C := (FLOAT(COST_1 - COST_2)/CURRENT_TEMP);
  if DELTA_C <= 15.0 then
    return EXP(-DELTA_C);
  else
    return 0.0;
  end if;
end ANNEAL_FUNCTION;
procedure ADJUST_PRECEDENCE( COUNT : in INTEGER;
                          P_LIST : in out NODE_LIST.LIST)
is
    MOVE_COUNT : INTEGER := 0;
    OP_NO : INTEGER;
    PRE_NO : INTEGER;
    PRECEDENCE_NEW : BOOLEAN := false;
    WORK_LIST : NODE_LIST.LIST;
    ADJUST_OP : NODE_LIST.LIST;
    AHEAD : NODE_LIST.LIST;

    begin
        while not PRECEDENCE_NEW loop
            NODE_LIST.FREE_LIST(WORK_LIST);
            WORK_LIST := P_LIST;
            while NODE_LIST.NON_EMPTY(WORK_LIST) loop  --Move to tail of list
                ADJUST_OP:= WORK_LIST;
                NODE_LIST.NEXT(WORK_LIST);
            end loop;
            MOVE_COUNT := INTEGER(RANDOM_NEXT * FLOAT(COUNT));
            while MOVE_COUNT > 1 loop
                NODE_LIST.PREVIOUS(ADJUST_OP);
                MOVE_COUNT := MOVE_COUNT - 1;
            end loop;
            WORK_LIST := ADJUST_OP;
            OP_NO := NODE_LIST.VALUE(ADJUST_OP);
            NODE_LIST.PREVIOUS(WORK_LIST);
            AHEAD := WORK_LIST;
            while NODE_LIST.NON_EMPTY(AHEAD) and not PRECEDENCE_NEW loop
                PRE_NO := NODE_LIST.VALUE(WORK_LIST);
                if NEW_GRAPH.IS_PARENT(PRE_NO,OP_NO) then
                    NODE_LIST.PREVIOUS(ADJUST_OP);
                    OP_NO := NODE_LIST.VALUE(ADJUST_OP);
                    WORK_LIST := ADJUST_OP;
                    NODE_LIST.PREVIOUS(WORK_LIST);
                    AHEAD := WORK_LIST;
                else
                    op
                end loop;
            end loop;
        end loop;
    end;

PRE_NO := NODE_LIST.VALUE(WORK_LIST);
end if;
end loop;
NODE_LIST.REMOVE(OP_NO,P_LIST);
NODE_LIST.INSERT_NEXT(OP_NO,WORK_LIST);
PRECEDENCE_NEW := true;
end if;
end loop;
end loop;
end ADJUSTPRECEDENCE;

end UTILITY_PKG;
2. **NEW_SCHEDULER_PKG**

with DATA; use DATA;

package NEW_SCHEDULER_PKG is

    procedure EARLIEST_START(COUNT : in INTEGER;
                                HBL : in INTEGER;
                                VALID_SCHEDULE : in out BOOLEAN;
                                AGENDA : in out SCHEDULE_ARRAY);

    procedure EARLIEST_DEADLINE(COUNT : in INTEGER;
                                 HBL : in INTEGER;
                                 VALID_SCHEDULE : in out BOOLEAN;
                                 AGENDA : in out SCHEDULE_ARRAY);

    procedure SIMULATED_ANNEAL(COUNT : in INTEGER;
                               HBL : in INTEGER;
                               PRECEDENCE_LIST : in out NODE_LIST_LIST;
                               AGENDA : in out SCHEDULE_ARRAY;
                               VALID_SCHEDULE : in out BOOLEAN);

end NEW_SCHEDULER_PKG;
with TEXT_IO; use TEXT_IO;
with DATA; use DATA;
with FRONT_END; use FRONT_END;
with PRIORITY_QUEUES;
with UTILITY_PKG; use UTILITY_PKG;
with MATH; use MATH;
with DIAGNOSTICS;

package body NEWSCHEDULER_PKG is

package int_io is new TEXT_IO.INTEGER_IO(INTEGER); use int_io;

procedure EARLIEST_START(COUNT : in INTEGER;  
HBL : in INTEGER;  
VALID_SCHEDULE : in out BOOLEAN;  
AGENDA : in out SCHEDULE_ARRAY) is

package EST_QUEUES is new PRIORITY_QUEUES(SCHEDULE_INPUTS_LOWERS."<");

type IN_ARRAY is array (0..COUNT) of VALUE;

PROCESSOR_STOP : PROCESSOR_ARRAY := (others => 0);  
REV_AGENDA : SCHEDULE_ARRAY := (others => null);  
WORK_LIST : NODE_LIST.LIST := null;  
PARENT_HEAD : NODE_LIST.LIST := null;  
CHILD_HEAD : NODE_LIST.LIST := null;  
QUE : EST_QUEUES.LINK := null;  
WAIT_LIST : SCHEDULE_INPUTS_LIST.LIST := null;  
PARENT_OP : OPERATOR;  
CHILD_OP : OPERATOR;  
CHILD_NUM : INTEGER;  
TEMP_NODE : SCHEDULE_INPUTS;  
INSERT : BOOLEAN;  
FINISH : IN_ARRAY := (others => 0);

-- store the finishing time of 1st instance of each op
INSTANCE_NO : IN_ARRAY := (others => 0);

-- store the # of instances of each op that have been scheduled
READY : IN_ARRAY := (others => 0);

-- store the lower bound of the 1st instance of each op

108
procedure DETERMINE_BEST_LOWER(REV_AGENDA: in SCHEDULE_ARRAY;  
INST_NO : in out IN_ARRAY;  
NEW_NODE : in out SCHEDULE_INPUTS;  
WAIT_LIST : in out SCHEDULE_INPUTS_LIST_LIST;  
PRI_Q : in out EST_QUEUES.LINK) is

NEXT_PROCESSOR : BOOLEAN;  
INSTANCE_FOUND : BOOLEAN;  
OLD : SCHEDULE_INPUTS;  
TEMP_SCHEDULE : SCHEDULE_ARRAY;  
OP_NO : INTEGER;  
L : NATURAL;

begin

OP_NO := NEW_NODE.THE_OPERATOR;
if not NEW_GRAPH.IS_CHILD(OP_NO,0) then
  NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST,OP_NO,PARENT_COUNT);
LIST_HEAD := PARENT_LIST;
while NODE_LIST.NON_EMPTY(PARENT_LIST) loop
  PARENT_NUM := NODE_LIST.VALUE(PARENT_LIST);
  PARENT_OP := NEW_GRAPH.OP_RETURN(PARENT_NUM);
  if (NEW_NODE.THE_START mod PARENT_OP.THE_PERIOD) = 0 then
    I := (NEW_NODE.THE_START / PARENT_OP.THE_PERIOD) + 1;
    if I > INST_NO(PARENT_NUM) then
      SCHEDULE_INPUTS_LIST.LIST.ADD(NEW_NODE.WAIT_LIST);
      NEW_NODE := EST_QUEUES.READ_BEST_FROM_PRIORITY_QUEUE(PRI_Q);
      EST_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(PRI_Q);
      DETERMINE_BEST_LOWER(REV_AGENDA,INST_NO,NEW_NODE.WAIT_LIST,PRI_Q);
    else
      TEMP_SCHEDULE := REV_AGENDA;
      L := 1;
      INSTANCE_FOUND := false;
      while L <= NOP and not INSTANCE_FOUND loop
        NEXT_PROCESSOR := false;
        while not (INSTANCE_FOUND or NEXT_PROCESSOR) loop
          if SCHEDULE_INPUTS_LIST.NON_EMPTY(TEMP_SCHEDULE(L)) then
            OLD := SCHEDULE_INPUTS_LIST.VALUE(TEMP_SCHEDULE(L));
            if OLD.THE_STOP > NEW_NODE.THE_LOWER then
              if OLD.THE_OPERATOR = PARENT_NUM and then
                OLD.THE_INSTANCE = I then
                INSTANCE_FOUND := true;
              end if;
            end if;
          end if;
          L := L + 1;
        end loop;
      end loop;
    end if;
  end if;
end if;
end loop;
end loop;
end DETERMINE_BEST_LOWER;
if NEW_NODE.THE_LOWER < OLD.THE_STOP +
    NEW_GRAPH.LATENCY(PARENT_NUM,OP_NO) then
    NEW_NODE.THE_LOWER := OLD.THE_STOP +
    NEW_GRAPH.LATENCY(PARENT_NUM,OP_NO);
endif;
else
    SCHEDULE_INPUTS_LIST.NEXT(T TEMP_SCHEDULE(L));
endif;
else
    NEXT_PROCESSOR := true;
endif;
else
    NEXT_PROCESSOR := true;
endif;
end loop;
L := L + 1;
end loop;
endif; -- "! > INST_NO"
endif;
NODE_LIST.NEXT(PARENT_LIST);
end loop;
NODE_LIST.FREE_LIST(LIST_HEAD);
end if;
dermine_BEST_LOWER;

begin -- procedure EARLIEST_START
    NODE_LIST.FREE_LIST(PARENT_LIST);
    NODE_LIST.FREE_LIST(CHILD_LIST);
    NEW_GRAPH.RETURN_CHILD_LIST(WORK_LIST,0,CHILD_COUNT);
    LIST_HEAD := WORK_LIST;
    while NODE_LIST.NON_EMPTY(WORK_LIST) loop
        OP_NUM := NODE_LIST.VALUE(WORK_LIST);
        TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
        NEW_INPUT.THE_OPERATOR := OP_NUM;
        NEW_INPUT.THE_LOWER := 0;
        NEW_INPUT.THE_INSTANCE := 1;
        DETERMINE_UPPER(TEMP,NEW_INPUT);
        DETERMINE_START_STOP(TEMP,PROCESSOR_NUM,NEW_INPUT,PROCESSOR_STOP)
        READY(OP_NUM) := NEW_INPUT.THE_START;
        FINISH(OP_NUM) := NEW_INPUT.THE_STOP;
        INSTANCE_NO(OP_NUM) := 1;
    end loop;
end if;
end PROCEDURE EARLIEST_START;
SCHEDULE_INPUTSLIST.ADD(NEW_INPUT,REV_AGENDA(PROCESSOR_NUM));
if NEW_INPUT.THE_INSTANCE < (HBL / TEMP.THE_PERIOD) then
  ADDL_NODE.THE_LOWER := READY(OP_NUM) + TEMP.THE_PERIOD;
  DETERMINE_THE_UPPER(TEMP,ADDL_NODE);
  CREATE_ADDL_NODE(NEW_INPUT,TEMP,ADDL_NODE);
  EST_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE,ADDL_NODE.THE_LOWER,QUE);
end if;
NEW_GRAPH.RETURN_CHILD_LIST(CHILD_LIST,OP_NUM,CHILD_COUNT);
CHILD_HEAD := CHILD_LIST;
while NODE_LIST.NON_EMPTY(CHILD_LIST) loop
  CHILD_NUM := NODE_LIST.VALUE(CHILD_LIST);
  INSERT := true;
  NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST,CHILD_NUM,PARENT_COUNT);
  PARENT_HEAD := PARENT_LIST;
  FIRST.THE_LOWER := 0;
  while NODE_LIST.NON_EMPTY(PARENT_LIST) loop
    PARENT_NUM := NODE_LIST.VALUE(PARENT_LIST);
    if INSTANCE_NO(PARENT_NUM) = 0 then
      INSERT := false;
      PARENT_LIST := null;
    else
      if FINISH(PARENT_NUM) + NEW_GRAPH.
        LATENCY(PARENT_NUM,CHILD_NUM) > FIRST.THE_LOWER then
        FIRST.THE_LOWER := FINISH(PARENT_NUM) + NEW_GRAPH.
        LATENCY(PARENT_NUM,CHILD_NUM);
      end if;
    end if;
  end loop;
  NODE_LIST.NEXT(PARENT_LIST);
end if;
end loop;
NODE_LIST.FREE_LIST(PARENT_HEAD);
if INSERT then
  FIRST.THE_OPERATOR := CHILD_NUM;
  CHILD_OP := NEW_GRAPH.OP_RETURN(CHILD_NUM);
  DETERMINE_THE_UPPER(CHILD_OP,FIRST);
  EST_QUEUES.INSERT_IN_PRIORITY_QUEUE(FIRST,FIRST.THE_LOWER,QUE);
end if;
NODE_LIST.NEXT(CHILD_LIST);
end loop;
NODE_LIST.FREE_LIST(CHILD_HEAD);
NODE_LIST.NEXT(WORK_LIST);
end loop;
while EST_QUEUES.NON_EMPTY(QUE) loop
  BEST := EST_QUEUES.READ_BEST_FROM_PRIORITY_QUEUE(QUE);
  EST_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(QUE);
  if BEST.THE_INSTANCE = 1 then
    OP_NUM := BEST.THE_OPERATOR;
    NEW_INPUT := BEST;
    TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
    DETERMINE_START_STOP(TEMP,PROCESSOR_NUM,NEW_INPUT,PROCESSOR_STOP);
    READY(OP_NUM) := NEW_INPUT.THE_LOWER;
    FINISH(OP_NUM) := NEW_INPUT.THE_STOP;
    INSTANCE_NO(OP_NUM) := 1;
    SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT,REV_AGENDA(PROCESSOR_NUM));
    NEW_GRAPH.RETURN_CHILD_LIST(CHILD_LIST,OP_NUM,CHILD_COUNT);
  end if;
  CHILD_HEAD := CHILD_LIST;
  while NODELIST.NON_EMPTY(CHILD_LIST) loop
    CHILD_NUM := NODELIST.VALUE(CHILD_LIST);
    INSERT := true;
    NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST,CHILD_NUM,CHILD_COUNT);
    PARENT_HEAD := PARENT_LIST;
    FIRST.THE_LOWER := 0;
    while NODELIST.NON_EMPTY(PARENT_LIST) loop
      PARENT_NUM := NODELIST.VALUE(PARENT_LIST);
      if INSTANCE_NO(PARENT_NUM) = 0 then
        INSERT := false;
        PARENT_LIST := null;
      else
        if FINISH(PARENT_NUM) + NEW_GRAPH.LATENCY
          (PARENT_NUM,CHILD_NUM) > FIRST.THE_LOWER then
          FIRST.THE_LOWER := FINISH(PARENT_NUM) + NEW_GRAPH.LATENCY
          (PARENT_NUM,CHILD_NUM);
        end if;
      end if;
      NODELIST.NEXT(PARENT_LIST);
    end loop;
  end loop;
  NODELIST.FREE_LIST(PARENT_HEAD);
  if INSERT then
    FIRST.THE_OPERATOR := CHILD_NUM;
    CHILD_OP := NEW_GRAPH.OP_RETURN(CHILD_NUM);
    DETERMINE_THE_UPPER(CHILD_OP,FIRST);
EST_QUEUES.INSERT_IN_PRIORITY_QUEUE(FIRST, FIRST, THE_LOWER, QUE);
end if;

NODE_LIST.NEXT(CHILD_LIST);
end loop;

else

if NEW_INPUT.THE_INSTANCE < (HBL / TEMP.THE_PERIOD) then

ADDL_NODE.THE_LOWER := READY(NEW_INPUT.THE_OPERATOR) + TEMP.THE_PERIOD * NEW_INPUT.THE_INSTANCE;

DETERMINE_THE_UPPER(TEMP, ADDL_NODE);
CREATE_ADDL_NODE(NEW_INPUT, TEMP, ADDL_NODE);
EST_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE, ADDL_NODE.THE_LOWER, QUE);

end if;
end loop;

for I in 1..NOP loop

SCHEDULE_INPUTS_LIST.LIST_REVERSE(AGENDA(I), REV_AGENDA(I));
SCHEDULE_INPUTS_LIST.FREE_LIST(REV_AGENDA(I));
end loop;

end EARLIEST_START;
procedure EARLIEST_DEADLINE( COUNT : in INTEGER;
HBL : in INTEGER;
VALID_SCHEDULE : in out BOOLEAN;
AGENDA : in out SCHEDULE_ARRAY) is

package EDL_QUEUES is new PRIORITY_QUEUES(SCHEDULE_INPUTS.UPPERS,"<");

type IN_ARRAY is array (0..COUNT) of VALUE:

PROCESSOR_STOP : PROCESSOR_ARRAY := (others => 0);
REV_AGENDA : SCHEDULE_ARRAY := (others => null);
WORK_LIST : NODE_LIST.LIST := null;
PARENT_HEAD : NODE_LIST.LIST := null;
CHILD_HEAD : NODE_LIST.LIST := null;
WAIT_LIST : SCHEDULE_INPUTS_LIST.LIST := null;
PARENT_OP : OPERATOR;
CHILD_OP : OPERATOR;
CHILD_NUM : INTEGER;
QUE : EDL_QUEUES.LINK := null;
INSERT : BOOLEAN;
TEMP_NODE : SCHEDULE_INPUTS;
FINISH : IN_ARRAY := (others => 0);
INSTANCE_NO : IN_ARRAY := (others => 0);
READY : IN_ARRAY := (others => 0);

procedure DETERMINE_BEST_LOWER(REV_AGENDA : in SCHEDULE_ARRAY;
INST_NO : in out IN_ARRAY;
NEW_NODE : in out SCHEDULE_INPUTS;
WAIT_LIST : in out SCHEDULE_INPUTS_LT.LIST;
PRIQ : in out EDL_QUEUES.LINK) is

begin
OP_NO := NEW_NODE.THE_OPERATOR;
if not NEW_GRAPH.IS_CHILD(OP_NO,0) then
NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST.OP_NO,PARENT_COUNT):
LIST_HEAD := PARENT_LIST:
while NODE_LIST.NON_EMPTY(PARENT_LIST) loop
    PARENT_NUM := NODE_LIST.VALUE(PARENT_LIST);
    PARENT_OP := NEW_GRAPH.OP_RETURN(PARENT_NUM);
    if (NEW_NODE.THE_START mod PARENT_OP.THE_PERIOD) = 0 then
        I := (NEW_NODE.THE_START / PARENT_OP.THE_PERIOD) + 1;
        if I > INST_NO(PARENT_NUM) then
            SCHEDULE_INPUTS_LIST.ADD(NEW_NODE.WAIT_LIST);
            NEW_NODE := EDL_QUEUES.READ_BEST_FROM_PRIORITY_QUEUE(PRI_Q);
            EDL_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(PRI_Q);
            DETERMINE_BEST_LOWER
                (REV_AGENDA,INST_NO,NEW_NODE.WAIT_LIST,PRI_Q);
        else
            TEMP_SCHEDULE := REV_AGENDA;
            L := 1;
            INSTANCE_FOUND := false;
            while L <= NOP and not INSTANCE_FOUND loop
                NEXT_PROCESSOR := false;
                while not (INSTANCE_FOUND or NEXT_PROCESSOR) loop
                    if SCHEDULE_INPUTS_LIST.NON_EMPTY(TEMP_SCHEDULE(L)) then
                        OLD := SCHEDULE_INPUTS_LIST.VALUE(TEMP_SCHEDULE(L));
                        if OLD.THE_STOP > NEW_NODE.THE_LOWER then
                            if OLD.THE_OPERATOR = PARENT_NUM and then
                                OLD.THE_INSTANCE = I then
                                INSTANCE_FOUND := true;
                            if NEW_NODE.THE_LOWER < OLD.THE_STOP
                                + NEW_GRAPH.LATENCY(PARENT_NUM,OP_NO) then
                                NEW_NODE.THE_LOWER := OLD.THE_STOP
                                + NEW_GRAPH.LATENCY(PARENT_NUM,OP_NO);
                            end if;
                        else
                            SCHEDULE_INPUTS_LIST.NEXT(TEMP_SCHEDULE(L));
                        end if;
                    else
                        NEXT_PROCESSOR := true;
                    end if;
                end loop;
            end while;
        end if;
    end if;
end loop;
L := L + 1;
end loop;
end if; -- "I > INST_NO"
end if;
NODE_LIST.NEXT(PARENT_LIST);
end loop;
NODE_LIST.FREE_LIST(LIST_HEAD);
end if;

end DETERMINE_BEST_LOWER;

begin -- procedure EARLIEST_DEADLINE
NODE_LIST.FREE_LIST(PARENT_LIST);
NODE_LIST.FREE_LIST(CHILD_LIST);

NEW_GRAPH.RETURN_CHILD_LIST(WORK_LIST,0,CHILD_COUNT);
LIST_HEAD := WORK_LIST;
while NODE_LIST.NON_EMPTY(WORK_LIST) loop
  OP_NUM := NODE_LIST.VALUE(WORK_LIST);
  TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
  FIRST.THE_OPERATOR := OP_NUM;
  FIRST.THE_LOWER := 0;
  DETERMINE_THE_UPPER(TEMP,FIRST);
  EDL_QUEUES.INSERT_IN_PRIORITY_QUEUE(FIRST,FIRST.THE_UPPER,QUE);
  NODE_LIST.NEXT(WORK_LIST);
end loop:
NODE_LIST.FREE_LIST(LIST_HEAD);

while EDL_QUEUES.NON_EMPTY(QUE) loop
  BEST := EDL_QUEUES.READ_BEST_FROM_PRIORITY_QUEUE(QUE);
  EDL_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(QUE);
  if BEST.THE_INSTANCE = 1 then
    OP_NUM := BEST.THE_OPERATOR;
    NEW_INPUT := BEST;
    TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
    DETERMINE_START_STOP
      (TEMP,PROCESSOR_NUM,NEW_INPUT,PROCESSOR_STOP);
    if NEW_GRAPH.IS_CHILD(OP_NUM,0) then
      READY(OP_NUM) := NEW_INPUT.THE_START;
    else
      ...
  end if;
end loop;
READY(OP_NUM) := NEW_INPUT.THE_LOWER;
end if;
FINISH(OP_NUM) := NEW_INPUT.THE_STOP;
INSTANCE_NO(OP_NUM) := 1;
SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT, REV_AGENDA(PROCESSOR_NUM));

NEW_GRAPH.RETURN_CHILD_LIST(CHILD_LIST, OP_NUM, CHILD_COUNT);
CHILD_HEAD := CHILD_LIST;
while NODE_LIST.NON_EMPTY(CHILD_LIST) loop
  CHILD_NUM := NODE_LIST.VALUE(CHILD_LIST);
  INSERT := true;
  NEW_GRAPH.RETURN_PARENT_LIST
    (PARENT_LIST, CHILD_NUM, PARENT_COUNT);
  PARENT_HEAD := PARENT_LIST;
  FIRST.THE_LOWER := 0;
  while NODE_LIST.NON_EMPTY(PARENT_LIST) loop
    PARENT_NUM := NODE_LIST.VALUE(PARENT_LIST);
    if INSTANCE_NO(PARENT_NUM) = 0 then
      INSERT := false;
      PARENT_LIST := null;
    else
      if FIRST.THE_LOWER < FINISH(PARENT_NUM)
        + NEW_GRAPH.LATENCY(PARENT_NUM, CHILD_NUM) then
        FIRST.THE_LOWER := FINISH(PARENT_NUM)
        + NEW_GRAPH.LATENCY(PARENT_NUM, CHILD_NUM);
      end if;
    end if;
    NODE_LIST.NEXT(PARENT_LIST);
  end if;
end loop;
NODE_LIST.FREE_LIST(PARENT_HEAD);

if INSERT then
  FIRST.THE_OPERATOR := CHILD_NUM;
  CHILD_OP := NEW_GRAPH.OP_RETURN(CHILD_NUM);
  DETERMINE_THE_UPPER(CHILD_OP, FIRST);
  EDL_QUEUES.INSERT_IN_PRIORITY_QUEUE(FIRST, FIRST.THE_UPPER, QUE);
end if;
NODE_LIST.NEXT(CHILD_LIST);
end loop;
NODE_LIST.FREE_LIST(CHILD_HEAD);
else

117
DETERMINE_BEST_LOWER(REV_AGENDA.INSTANCE_NO,BEST,WAIT_LIST.QUE);
OP_NUM := BEST.THE_OPERATOR;
NEW_INPUT := BEST;
TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
DETERMINE_START_STOP(TEMP.PROCESSOR_NUM,NEW_INPUT.PROCESSOR_STOP);
SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT,REV_AGENDA(PROCESSOR_NUM));
INSTANCE_NO(NEW_INPUT.THE_OPERATOR) := NEW_INPUT.THE_INSTANCE;
while SCHEDULE_INPUTS_LIST.NON_EMPTY(WAIT_LIST) loop
  TEMP_NODE := SCHEDULE_INPUTS_LIST.VALUE(WAIT_LIST);
  EDL_QUEUES.INSERT_IN_PRIORITY_QUEUE
    (TEMP_NODE,TEMP_NODE.THE_UPPER.QUE);
  SCHEDULE_INPUTS_LIST.REMOVE(TEMP_NODE,WAIT_LIST);
end loop;
end if;

if NEW_INPUT.THE_START > NEW_INPUT.THE_UPPER
  or NEW_INPUT.THE_STOP > HBL then
  VALID_SCHEDULE := false;
end if;

if NEW_INPUT.THE_INSTANCE < (HBL / TEMP.THE_PERIOD) then
  ADDL_NODE.THE_LOWER := READY(NEW_INPUT.THE_OPERATOR) +
    TEMP.THE_PERIOD * NEW_INPUT.THE_INSTANCE;
  DETERMINE_THE_UPPER(TEMP,ADDL_NODE);
  CREATE_ADDL_NODE(NEW_INPUT,TEMP,ADDL_NODE);
  EDL_QUEUES.INSERT_IN_PRIORITY_QUEUE
    (ADDL_NODE,ADDL_NODE.THE_UPPER.QUE);
end if;
end loop;

for I in 1..NOP loop
  SCHEDULE_INPUTS_LIST.LIST_REVERSE(REV_AGENDA(I), AGENDA(I));
  SCHEDULE_INPUTS_LIST.FREE_LIST(REV_AGENDA(I));
end loop;
end EARLIEST_DEADLINE;
procedure SIMULATED_ANNEAL( COUNT in INTEGER; HBL in INTEGER; PRECEDENCE_LIST in out NODE_LIST.LIST; AGENDA in out SCHEDULE_ARRAY; VALID_SCHEDULE in out BOOLEAN) is

package float_io is new TEXT_IO.FLOAT_IO(FLOAT); use float_io;
package PRIORITY_Q is new PRIORITY_QUEUES(SCHEDULE_INPUTS,LOWERS,"<"); use PRIORITY_Q;

type IN_ARRAY is array (0..COUNT) of VALUE;
PROCESSOR_STOP : PROCESSOR_ARRAY;
REV_AGENDA : SCHEDULE_ARRAY;
INSTANCE_NO : IN_ARRAY;
READY : IN_ARRAY;
P_LIST : NODE_LIST.LIST := null;
ADDL_LIST : SCHEDULE_INPUTS_LIST.LIST;
A_LIST : SCHEDULE_INPUTS_LIST.LIST := null;
HEAD_A : SCHEDULE_INPUTS_LIST.LIST := null;
COST : INTEGER;
QUE : PRIORITY_Q.LINK := null;
procedure SCHEDULE_1st_INSTANCES(HBL in INTEGER;
READY in out IN_ARRAY;
P_LIST in out NODE_LIST.LIST;
PROC_STOP in out PROCESSOR_ARRAY;
REV_AGGENDA in out SCHEDULE_ARRAY;
A_LIST in out SCHEDULE_INPUTS_LIST.LIST) is

FINISH : IN_ARRAY := (others => 0);
WORK_LIST : NODE_LIST.LIST;
PARENT_HEAD : NODE_LIST.LIST;

begin

NODE_LIST.FREE_LIST(PARENT_LIST);
NODE_LIST.FREE_LIST(CHILD_LIST);
SCHEDULE_INPUTS_LIST.FREE_LIST(A_LIST);
REV_AGGENDA := (others=> null);
PROC_STOP := (others=> 0);
READY := (others => 0);
NODE_LIST.REMOVE(0,P_LIST);
NODE_LIST.DUPLICATE(P_LIST,WORK_LIST);
NEW_GRAPH.RETURN_CHILD_LIST(CHILD_LIST,0,CHILD_COUNT);
LIST_HEAD := CHILD_LIST;

while NODE_LIST.NON_EMPTY(CHILD_LIST) loop

OP_NUM := NODE_LIST.VALUE(CHILD_LIST);
TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
NEW_INPUT.THE_OPERATOR := OP_NUM;
NEW_INPUT.THE_LOWER := 0;
NEW_INPUT.THE_INSTANCE := 1;
DETERMINE_THE_UPPER(TEMP,NEW_INPUT);
DETERMINE_START_STOP(TEMP,PROCESSOR_NUM,NEW_INPUT,PROC_STOP);
READY(OP_NUM) := NEW_INPUT.THE_START;
FINISH(OP_NUM) := NEW_INPUT.THE_STOP;
SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT,REV_AGGENDA(PROCESSOR_NUM));
NODE_LIST.REMOVE(OP_NUM,WORK_LIST);
if NEW_INPUT.THE_INSTANCE = (HBL / TEMP.THE_PERIOD) then

NODE_LIST.REMOVE(OP_NUM,P_LIST);
else

ADDL_NODE.THE_LOWER := READY(OP_NUM) + TEMP.THE_PERIOD;
DETERMINE_THE_UPPER(TEMP,ADDL_NODE);
CREATE_ADDL_NODE(NEW_INPUT,TEMP,ADDL_NODE);
SCHEDULE_INPUTS_LIST.ADD(ADDL_NODE,A_LIST);

end loop;

end SCHEDULE_1st_INSTANCES;
end if;
NODE_LIST.NEXT(CHILD_LIST);
end loop;
NODE_LIST.FREE_LIST(LIST_HEAD);
LIST_HEAD:= WORK_LIST;
while NODE_LIST.NON_EMPTY(WORK_LIST) loop
  OP_NUM := NODE_LIST.VALUE(WORK_LIST);
  TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
  NEW_INPUT.THE_OPERATOR := OP_NUM;
  NEW_INPUT.THE_LOWER := 0;
  NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST,OP_NUM,PARENT_COUNT);
  PARENT_HEAD := PARENT_LIST;
  while NODE_LIST.NON_EMPTY(PARENT_LIST) loop
    PARENT_NUM := NODE_LIST.VALUE(PARENT_LIST);
    PARENT_OP := NEW_GRAPH.OP_RETURN(PARENT_NUM);
    if NEW_INPUT.THE_LOWER < FINISH(PARENT_NUM) + NEW_GRAPH.LATENCY(PARENT_NUM,OP_NUM) then
      NEW_INPUT.THE_LOWER := FINISH(PARENT_NUM) + NEW_GRAPH.LATENCY(PARENT_NUM,OP_NUM);
    end if;
    NODE_LIST.NEXT(PARENT_LIST);
  end loop;
  NODE_LIST.NEXT(PARENT_LIST);
end loop;
NODE_LIST.FREE_LIST(PARENT_HEAD);
DETERMINE_THE_UPPER(TEMP,NEW_INPUT);
DETERMINE_START_STOP(TEMP,PROCESSOR_NUM,NEW_INPUT.PROC_STOP);
READY(OP_NUM) := NEW_INPUT.THE_LOWER;
FINISH(OP_NUM) := NEW_INPUT.THE_STOP;
SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT,REV_AGGENDA(PROCESSOR_NUM));
if NEW_INPUT.THE_INSTANCE = (HBL / TEMP.THE_PERIOD) then
  NODE_LIST.REMOVE(OP_NUM,P_LIST);
else
  ADDL_NODE.THE_LOWER := READY(OP_NUM) + TEMP.THE_PERIOD;
  DETERMINE_THE_UPPER(TEMP,ADDL_NODE);
  CREATE_ADDL_NODE(NEW_INPUT,TEMP,ADDL_NODE);
  SCHEDULE_INPUTS_LIST.ADD(ADDL_NODE,A_LIST);
end if;
NODE_LIST.NEXT(WORK_LIST);
end loop;
NODE_LIST.FREE_LIST(LIST_HEAD);
end SCHEDULE_1ST_INSTANCES;
procedure SCHEDULE_REST_OF_BLOCK( HBL : in INTEGER; 
P_LIST : in out NODE_LIST.LIST; 
A_LIST : in out SCHEDULE_INPUTS_LIST.LIST; 
REV_AGENDA: in out SCHEDULE_ARRAY; 
PROC_STOP : in out PROCESSOR_ARRAY; 
INST_NO : in out IN_ARRAY; 
READY : in IN_ARRAY) is

WORK_LIST : NODE_LIST.LIST; 
HEAD_A : SCHEDULE_INPUTS_LIST.LIST; 
OP_FOUND : BOOLEAN; 
AWAIT : BOOLEAN;

procedure DETERMINE_THE_LOWER( REV_AGENDA : in SCHEDULE_ARRAY; 
AWAIT : out BOOLEAN; 
INST_NO : in out IN_ARRAY; 
NEW_NODE : in out SCHEDULE_INPUTS) is

PARENT_FOUND : BOOLEAN; 
OLD : SCHEDULE_INPUTS; 
TEMP_SCHEDULE : SCHEDULE_ARRAY; 
OP_NO : INTEGER; 
J,M : INTEGER; 
begin 
OP_NO := NEW_NODE.THE_OPERATOR; 
NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST,OP_NO,PARENT_COUNT); 
if NODE_LIST.VALUE(PARENT_LIST) = 0 then 
NODE_LIST.REMOVE(0,PARENT_LIST); 
end if; 
LIST_HEAD := PARENT_LIST; 
while NODE_LIST.NON_EMPTY(PARENT_LIST) loop 
PARENT_NUM := NODE_LIST.VALUE(PARENT_LIST); 
PARENT_OP := NEW_GRAPH.OP_RETURN(PARENT_NUM); 
if (NEW_NODE.THE_START mod PARENT_OP.THE_PERIOD) = 0 then 
J := (NEW_NODE.THE_START / PARENT_OP.THE_PERIOD) + 1: 
AWAIT := false; 
if J > INST_NO(PARENT_NUM) then 
AWAIT := true; 
NODE_LIST.FREE_LIST(LIST_HEAD);

122
exit;
else
  TEMP_SCHEDULE := REV_AGENDA;
PARENT_FOUND := false;
for M in 1..NOP loop
  while SCHEDULE_INPUTS_LIST.NON_EMPTY(TEMP_SCHEDULE(M))
    and not PARENT_FOUND loop
    OLD := SCHEDULE_INPUTS_LIST.VALUE(TEMP_SCHEDULE(M));
    if OLD.THE_STOP > NEW_NODE.THE_LOWER then
      if OLD.THE_OPERATOR = PARENT_NUM
        and then OLD.THE_INSTANCE = J then
        PARENT_FOUND := true;
        TEMP_SCHEDULE(M) := null;
      if NEW_NODE.THE_LOWER < OLD.THE_STOP
        NEW_GRAPH.LATENCY(PARENT_NUM, OP_NO) then
        NEW_NODE.THE_LOWER := OLD.THE_STOP
        + NEW_GRAPH.LATENCY(PARENT_NUM, OP_NO);
        end if;
      else
        SCHEDULE_INPUTS_LIST.NEXT(TEMP_SCHEDULE(M));
      end if;
    else
      TEMP_SCHEDULE(M) := null;
    end if;
  end loop;
  if PARENT_FOUND then
    exit;
  end if;
end loop;
end if;  -- "if J > "
end if;
NODE_LIST.NEXT(PARENT_LIST);
end loop;
NODE_LIST.FREE_LIST(LIST_HEAD);
end DETERMINE_THE_LOWER;
begin -- schedule the rest of the block

INST_NO := (others=> 1);
while NODE_LIST.NON_EMPTY(P_LIST) loop
  NODE_LIST.DUPLICATE(P_LIST,WORK_LIST);
  LIST_HEAD := WORK_LIST;
  while NODE_LIST.NON_EMPTY(WORK_LIST) loop
    OP_NUM := NODE_LIST.VALUE(WORK_LIST);
    TEMP:= NEW_GRAPH.OP_RETURN(OP_NUM);
    OP_FOUND:= false;
    HEAD_A := A_LIST;
    while SCHEDULE_INPUTS_LIST.VALUE(A_LIST).THE_OPERATOR /= OP_NUM loop
      SCHEDULE_INPUTS_LIST.NEXT(A_LIST);
    end loop;
    BEST := SCHEDULE_INPUTS_LIST.VALUE(A_LIST);
    NEW_INPUT := BEST;
    DETERMINE_THE_LOWER(REV_AGENDA,AWAIT,INST_NO,NEW_INPUT);
    if not AWAIT then
      DETERMINE_START_STOP(TMP,PROCESSOR_NUM,NEW_INPUT,PROC_STOP);
      SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT,REV_AGENDA(PROCESSOR_NUM));
      INST_NO(OP_NUM) := NEW_INPUT.THE_INSTANCE;
      if NEW_INPUT.THE_INSTANCE = (HBL/TEMP.THE_PERIOD) then
        NODE_LIST.REMOVE(OP_NUM,P_LIST);
      else
        ADDL_NODE.THE_LOWER := READY(OP_NUM)
          + TEMP.THE_PERIOD * NEW_INPUT.THE_INSTANCE;
        DETERMINE_THE_UPPER(TEMP,ADDL_NODE);
        CREATE_ADDL_NODE(NEW_INPUT,TEMP,ADDL_NODE);
        SCHEDULE_INPUTS_LIST.insert_next(ADDL_NODE,A_LIST);
      end if;
      A_LIST := HEAD_A;
      SCHEDULE_INPUTS_LIST.REMOVE(BEST,A_LIST);
    else
      A_LIST := HEAD_A;
    end if;
  NODE_LIST.NEXT(WORK_LIST);
end loop;
NODE_LIST.FREE_LIST(LIST_HEAD);
end loop;

end SCHEDULE_REST_OF_BLOCK;
procedure ANNEAL_PROCESS( HBL : in INTEGER;
AGENDA : in out SCHEDULE_ARRAY;
SOLUTION : in out BOOLEAN;
PENALTY_COST : in out INTEGER;
INST_NO : in out IN_ARRAY;
PREC_LIST : in out NODE_LIST.LIST)

BEST_AGENDA : SCHEDULE_ARRAY;
TEMP_AGENDA : SCHEDULE_ARRAY;
TEMPERATURE : FLOAT;
BEST_COST : INTEGER;
TEMP_COST : INTEGER;
TRIAL_NUM : INTEGER := 100;
ACCEPT_NUM : INTEGER := 35;
FREEZE : FLOAT := 1.0;
COOLING_FACTOR : FLOAT := 0.95;
TRIAL_COUNT : INTEGER;
ACCEPT_COUNT : INTEGER;
P_LIST_NEW : BOOLEAN := false;
P_LIST : NODE_LIST.LIST;
APFOUND : BOOLEAN;
REARRANGE_P : BOOLEAN;
TEMP_LOWER : INTEGER;
OLD_LOWER : INTEGER;
V : SCHEDULE_ARRAY;

procedure DETERMINE_NEW_LOWER( TEMP : in OPERATOR;
TEMP_AGENDA : in SCHEDULE_ARRAY;
NEW_NODE : in out SCHEDULE_INPUTS)

NEXT_PROC : BOOLEAN;
INS_FOUND : BOOLEAN;
INST, I : INTEGER;
OP_NO : INTEGER;
W : SCHEDULE_ARRAY := TEMP_AGENDA;

begin
OP_NO := NEW_NODE. THE_OPERATOR;
if TEMP.THE_WITHIN /= 0 then
NEW_NODE.THE_LOWER := NEW_NODE.THE_UPPER + TEMP.THE_MET - TEMP.THE_WITHIN;
else
NEW_NODE.THE_LOWER := NEW_NODE.THE_UPPER + TEMP.THE_MET - TEMP.THE_PERIOD
end if;
if not NEW_GRAPH.PIPELINE(OP_NO) then
    INS_FOUND := false;
for I in 1..NOP loop
    NEXT_PROC := false;
    while SCHEDULE_INPUTS_LIST.NON_EMPTY(W(I)) and not NEXT_PROC loop
        if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_OPERATOR = OP_NO then
            if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_INSTANCE 
                >= NEW_NODE.THE_INSTANCE - 1 then
                NEXT_PROC := true;
            else if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_INSTANCE = 
                NEW_NODE.THE_INSTANCE - 1 then
                INS_FOUND := true;
            else
                SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_STOP := 
                NEW_NODE.THE_LOWER then
                NEW_NODE.THE_LOWER := 
                SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_STOP;
            end if;
        end if;
    end loop;
    if INS_FOUND then
        exit;
    end if;
end loop;
end if; -- " not PIPELINE"
NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST,OP_NO,PARENT_COUNT):
LIST_HEAD := PARENT_LIST;
if NODE_LIST.VALUE(PARENT_LIST) = 0 then
    NODE_LIST.NEXT(PARENT_LIST);
end if;
while NODE_LIST.NON_EMPTY(PARENT_LIST) loop
    PARENT_NUM := NODE_LIST.VALUE(PARENT_LIST);
PARENT_OP := NEW_GRAPH.OP_RETURN(PARENT_NUM);
if ((NEW_NODE.THE_INSTANCE - 1) * (TEMP.THE_PERIOD))
    mod PARENT_OP.THE_PERIOD = 0 then
    INST := (((NEW_NODE.THE_INSTANCE - 1) * (TEMP.THE_PERIOD))
        / PARENT_OP.THE_PERIOD) + 1;
    INS_FOUND := false;
    W := TEMP_AGENDA;
    for I in 1..NOP loop
        NEXT_PROC := false;
        while SCHEDULE_INPUTS_LIST.NON_EMPTY(W(I)) and not NEXT_PROC loop
            if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_OPERATOR = PARENT_NUM then
                if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_INSTANCE >= INST then
                    NEXT_PROC := true;
                end if;
                if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_INSTANCE = INST then
                    INS_FOUND := true;
                    if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_STOP+NEW_GRAPH.LATENCY(PARENT_NUM,OP_NO)
                        > NEW_NODE.THE_LOWER then
                        NEW_NODE.THE_LOWER := SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_STOP+
                            NEW_GRAPH.LATENCY(PARENT_NUM,OP_NO);
                        end if;
                end if;
            else
                SCHEDULE_INPUTS_LIST.NEXT(W(I));
            end if;
        end loop;
        if INS_FOUND then
            exit;
        end if;
    end loop;
    NODE_LIST.NEXT(PARENT_LIST);
end loop;
NODE_LIST.FREE_LIST(LIST_HEAD);
end DETERMINE_NEW_LOWER;
procedure ADJUST_JIT(TEMP in OPERATOR;
REARRANGE in out BOOLEAN;
NEW_NODE in out SCHEDULE_INPUTS;
TEMP_AGENDA in out SCHEDULE_ARRAY) is

DIFF, I ANTEGER;
NEW_PROCESSOR : INTEGER;
STOP_TIME : INTEGER;
W : SCHEDULE_ARRAY := TEMP_AGENDA;

begin
DIFF := -1;
for T in 1..NOP loop
while SCHEDULE_INPUTS_LIST.NON_EMPTY(W(I)) loop
if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_STOP > NEW_NODE.THE_LOWER then
if (SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_START - NEW_NODE.THE_STOP) > DIFF then
DIFF := (SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_START - NEW_NODE.THE_STOP);
NEW_PROCESSOR := 1;
end if;
exit;
else
SCHEDULE_INPUTS_LIST.NEXT(W(I));
end if;
end loop;
end loop
if DIFF < 0 then
NEW_NODE.THE_START := NEW_NODE.THE_UPPER + 10;
for I in 1..NOP loop
if SCHEDULE_INPUTS_LIST.NON_EMPTY(W(I)) then
STOP_TIME := SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_STOP;
SCHEDULE_INPUTS_LIST.NEXT(W(I));
while SCHEDULE_INPUTS_LIST.NON_EMPTY(W(I)) loop
if SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_START - STOP_TIME < TEMP.THE_MET then
STOP_TIME := SCHEDULE_INPUTS_LIST.VALUE(W(I)).THE_STOP;
SCHEDULE_INPUTS_LIST.NEXT(W(I));
else
exit;
end if;
end loop;
if STOP_TIME < NEW_NODE.THE_START then
NEW_NODE.THE_START := STOP_TIME;
NEW_PROCESSOR := 1;
end if;
end loop;
end if;
end if;
end if;
end loop;

if NEW_NODE.THE_START > NEW_NODE.THE_UPPER then
    REARRANGE := true;
else
    NEW_NODE.THE_STOP := NEW_NODE.THE_START + TEMP.THE_MET;
end if;

begin-- ANNEAL PROCESS!

for I in 1..NOP loop
    SCHEDULE_INPUTS_LIST.DUPLICATE_AGENDA_AGENDA(I);
    SCHEDULE_INPUTS_LIST.DUPLICATE_AGENDA_TEMP_AGENDA(I);
end loop;

TEMPERATURE := 0.9 * FLOAT(PENALTY_COST);
BEST_COST := PENALTY_COST;
while not SOLUTION and TEMPERATURE > FREEZE loop
    ACCEPT_COUNT := 0;
    TRIAL_COUNT := 0;
    while not SOLUTION and ACCEPT_COUNT < ACCEPT_NUM and TRIAL_COUNT < TRIAL_NUM loop
        REARRANGE := false;
        QUE := INITIALIZE_PRIORITY_QUEUE;
        V := TEMP_AGENDA;
        for I in 1..NOP loop
            AP_FOUND := false;
            while SCHEDULE_INPUTS_LIST.NON_EMPTY(V(I)) and not AP_FOUND loop
                if SCHEDULE_INPUTS_LIST.VALUE(V(I)).THE_START
                > SCHEDULE_INPUTS_LIST.VALUE(V(I)).THE_UPPER then
                    TEMP_LOWER := SCHEDULE_INPUTS_LIST.VALUE(V(I)).THE_LOWER;
                    AP_FOUND := true;
                    INSERT_IN_PRIORITY_QUEUE(
                        SCHEDULE_INPUTS_LIST.VALUE(V(I)), TEMP_LOWER, QUE);
                else
                    SCHEDULE_INPUTS_LIST.NEXT(V(I));
                end if;
            end loop;
        end for
    end loop;
end while...
while PRIORITY_Q.NON_EMPTY(QUE) loop

BEST := READ_BEST_FROM_PRIORITY_QUEUE(QUE);
OP_NUM := BEST.THE_OPERATOR;
for I in 1..NOP loop
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(I)) and then
        SCHEDULE_INPUTS_LIST.VALUE(V(I)) = BEST then
        PROCESSOR_NUM := I;
        exit;
    end if;
end loop;

NEW_INPUT := BEST;
TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
DETERMINE_NEW_LOWER(TEMP, TEMP_AGENDA, NEW_INPUT);
if NEW_INPUT.THE_LOWER > NEW_INPUT.THE_UPPER then
    REARRANGE_P := true;
    exit;
end if;
if NEW_INPUT.THE_LOWER < NEW_INPUT.THE_START then
    NEW_INPUT.THE_START := NEW_INPUT.THE_LOWER;
    NEW_INPUT.THE_STOP := NEW_INPUT.THE_START + TEMP.THE_MET;
    ADJUST_IT(TEMP, REARRANGE_P, NEW_INPUT, TEMP_AGENDA);
    if REARRANGE_P then
        exit;
    else
        SCHEDULE_INPUTS_LIST.NEXT(V(PROCESSOR_NUM));
        SCHEDULE_INPUTS_LIST.REMOVE(BEST, TEMP_AGENDA(PROCESSOR_NUM));
        REMOVE_BEST_FROM_PRIORITY_QUEUE(QUE);
        if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(PROCESSOR_NUM)) then
            TEMP_LOWER := SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR)).THE_LOWER;
            INSERT_IN_PRIORITY_QUEUE
                (SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)), TEMP_LOWER, QUE);
        end if;
    end if;
else
    REMOVE_BEST_FROM_PRIORITY_QUEUE(QUE);
    SCHEDULE_INPUTS_LIST.NEXT(V(PROCESSOR_NUM));
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(PROCESSOR_NUM)) then
        TEMP_LOWER := SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)).THE_LOWER;
    end if;
end if;
else
    REMOVE_BEST_FROM_PRIORITY_QUEUE(QUE);
    SCHEDULE_INPUTS_LIST.NEXT(V(PROCESSOR_NUM));
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(PROCESSOR_NUM)) then
        TEMP_LOWER := SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)).THE_LOWER;
    end if;
    SCHEDULE_INPUTS_LIST.NEXT(V(PROCESSOR_NUM));
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(PROCESSOR_NUM)) then
        TEMP_LOWER := SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)).THE_LOWER;
    end if;
    SCHEDULE_INPUTS_LIST.NEXT(V(PROCESSOR_NUM));
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(PROCESSOR_NUM)) then
        TEMP_LOWER := SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)).THE_LOWER;
    end if;
end if;
end loop;
INSERT_IN_PRIORITY_QUEUE(SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)).THE_LOWER.QUE);
end if;
end if;
end loop;

if not REARRANGE_P then
QU:= INITIALIZE_PRIORITY_QUEUE;
V:= TEMP_AGENDA;
for I in 1..NOP loop
AP_FOUND := false;
while SCHEDULE_INPUTS_LIST.NON_EMPTY(V(I)) and not AP_FOUND loop
if SCHEDULE_INPUTS_LIST.VALUE(V(I)).THE_STOP > HBL then
TEMP_LOWER := SCHEDULE_INPUTS_LIST.VALUE(V(I)).THE_LOWER;
AP_FOUND := true;
INSERT_IN_PRIORITY_QUEUE(SCHEDULE_INPUTS_LIST.VALUE(V(I)),TEMP_LOWER.QUE);
else
SCHEDULE_INPUTS_LIST.NEXT(V(I));
end if;
end loop;
end loop;

while PRIORITYQ.NON_EMPTY(QUE) loop
BEST := READ_BEST_FROM_PRIORITY_QUEUE(QUE);
OP_NUM := BEST.THE_OPERATOR;
for I in 1..NOP loop
if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(I)) and then
SCHEDULE_INPUTS_LIST.VALUE(V(I)) = BEST then
PROCESSOR_NUM := I;
exit;
end if;
end loop;
end loop;

NEW_INPUT := BEST;
TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
OLD_LOWER := NEW_INPUT.THE_LOWER;
DETERMINE_NEW_LOWER(TEMP,TEMP_AGENDA,NEW_INPUT);
if NEW_INPUT.THE_LOWER = OLD_LOWER and then
NEW_INPUT.THE_START = NEW_INPUT.THE_LOWER then
REARRANGE_P := true;

131
exit;
else
    NEW_INPUT.THE_START := NEW_INPUT.THE_LOWER;
    NEW_INPUT.THE_STOP := NEW_INPUT.THE_START + TEMP.THE_MET;
    ADJUST_IT(TEMP, REARRANGE_P, NEW_INPUT, TEMP_AGENDA);
    if REARRANGE_P then
        exit;
    end if;
else
    SCHEDULE_INPUTS_LIST.NEXT(V(PROCESSOR_NUM));
    SCHEDULE_INPUTS_LIST.REMOVE(BEST, TEMP_AGENDA(PROCESSOR_NUM));
    REMOVE_BEST_FROM_PRIORITY_QUEUE(QUE);
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(V(PROCESSOR_NUM)) then
        TEMP_LOWER :=
        SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)).THE_LOWER
        INSERT_IN_PRIORITY_QUEUE
            (SCHEDULE_INPUTS_LIST.VALUE(V(PROCESSOR_NUM)), TEMP_LOWER.QUE);
    end if;
end if;
end loop;
end if;

if REARRANGE_P then
    ADJUST_PRECEDENCE(COUNT, PREC_LIST);
    NODE_LIST.DUPLICATE(PREC_LIST, P_LIST);
    SCHEDULE_1st_INSTANCES(HBL, READY, P_LIST, PROCESSOR_STOP, REV_AGENDA, ADDL_LIST);
    SCHEDULE_INPUTS_LIST.LIST_REVERSE(ADDL_LIST, A_LIST);
    SCHEDULE_INPUTS_LIST.FREE_LIST(ADDL_LIST);
    SCHEDULE_REST_OF_BLOCK(HBL, P_LIST, A_LIST, REV_AGENDA, PROCESSOR_STOP, INST_NO, READY);
    for i in 1..NOP loop
        SCHEDULE_INPUTS_LIST.FREE_LIST(TEMP_AGENDA(I));
        SCHEDULE_INPUTS_LIST.LIST_REVERSE(REV_AGENDA(I), TEMP_AGENDA(I));
    end loop;
end if;
TEST_SCHEDULE(HBL, TEMP_AGENDA, TEMP_COST);
if TEMP_COST < BEST_COST then
    BEST_COST := TEMP_COST;
    for I in 1..NOP loop
        SCHEDULE_INPUTS_LIST.DUPLICATE(TEMP_AGENDA(I), BEST_AGENDA(I));
    end loop;
end if;

132
if TEMPCOST <= 0 then
  SOLUTION := true;
elsif REARRANGE_P or else TEMPCOST <= PENALTY_COST or else
  RANDOM_NEXT < ANNEAL_FUNCTION(TEMPCOST,PENALTY_COST,TEMPERATURE) then
    ACCEPT_COUNT := ACCEPT_COUNT + 1;
    PENALTY_COST := TEMPCOST;
  for I in 1..NOP loop
    SCHEDULE_INPUTS_LIST.DUPLICATE(TEMP_AGENDA(I),AGENDA(I));
  end loop;
else
  for I in 1..NOP loop
    SCHEDULE_INPUTS_LIST.DUPLICATE(AGENDA(I), TEMP_AGENDA(I));
  end loop;
end if:
  TRIAL_COUNT := TRIAL_COUNT + 1;
end loop;
TEMPERATURE := TEMPERATURE * COOLING_FACTOR;
end loop;
AGENDA := BEST_AGENDA;
PENALTY_COST := BEST_COST;

exception

when MATH.RANGE_ERROR =>
  TEXT_IO.PUT_LINE("THE FOLLOWING VALUES CAUSED A RANGE ERROR");
  TEXT_IO.PUT("PENALTY COST: ");
  TEXT_IO.SET_COL(15);
  int_io.put(PENALTY_COST, width=>5);
  TEXT_IO.NEW_LINE;
  TEXT_IO.PUT("TEMP COST: ");
  TEXT_IO.SET_COL(15);
  int_io.put(TEMPCOST, width=>5);
  TEXT_IO.NEW_LINE;
  TEXT_IO.PUT("TEMPERATURE: ");
  TEXT_IO.SET_COL(15);
  float_io.PUT(TEMPERATURE, fore=>5, aft=>5, exp=>0);
  TEXT_IO.NEW_LINE;
  AGENDA := (others=> null);
  end ANNEAL_PROCESS;
begin -- main SIMULATED ANNEAL

NODE_LIST.DUPLICATE(PRECEDENCE_LIST,P_LIST);
SCHEDULE_1st_INSTANCES(HBL,READY,P_LIST,PROCESSOR_STOP,REV_AGENDA,ADDL_LIST);
SCHEDULE_INPUTS_LIST.LIST_REVERSE(ADDL_LIST,A_LIST);
SCHEDULE_INPUTS_LIST.FREE_LIST(ADDL_LIST);
SCHEDULE_REST_OF_BLOCK
    (HBL,P_LIST,A_LIST,REV_AGENDA,PROCESSOR_STOP,INSTANCE_NO,READY);
for I in 1..NOP loop
    SCHEDULE_INPUTS_LIST.LIST_REVERSE(REV_AGENDA(I),AGENDA(I));
    SCHEDULE_INPUTS_LIST.FREE_LIST(REV_AGENDA(I));
end loop:
TEST_SCHEDULE(HBL,AGENDA,COST);

if COST > 0 then
    VALID_SCHEDULE := false;
    RANDOM_INITIALIZE(2*COUNT + 1);
    --* Initialize Random Number Generator with an odd number.
    ANNEAL_PROCESS
        (HBL,AGENDA,VALID_SCHEDULE,COST,INSTANCE_NO,PRECEDENCE_LIST);
else
    VALID_SCHEDULE := true;
end if;

end SIMULATED_ANNEAL;

end NEW_SCHEDULER_PKG:
LIST OF REFERENCES


INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943

3. Computer Science Department
   Code CS
   Naval Postgraduate School
   Monterey, CA 93943

4. Office of the Assistant Secretary of the Navy
   Research Development and Acquisition
   Department of the Navy
   Attn: Mr. Gerald A. Cann
   Washington, DC 20380-1000

5. Office of the Chief of Naval Operations
   OP-094
   Department of the Navy
   Attn: VADM J. O. Tuttle, USN
   Washington, DC 20301-3040

6. Director of Defense Information
   Office of the Assistant Secretary of Defense
   (Command, Control, Communications, & Intelligence)
   Attn: Mr. Paul Strassmann
   Washington, DC 20301-0208

7. Center for Naval Analysis
   4401 Ford Avenue
   Alexandria, VA 22302-0268

8. Prof. Man-Tak Shing, Code CS/Sh
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943
9. Chairman, Code CS  
   Computer Science Department  
   Naval Postgraduate School  
   Monterey, CA 93943-5100

10. Prof. Luqi, Code CS/Lq  
    Computer Science Department  
    Naval Postgraduate School  
    Monterey, CA 93943

11. Chief of Naval Research  
    Attn: ADM. Miller  
    800 N. Quincy Street  
    Arlington, VA 22217

12. Director, Ada Joint Program Office  
    OUSDRE (R&AT)  
    Room 3E114, The Pentagon  
    Attn: Dr. John P. Solomond  
    Washington, DC 20301-0208

13. Carnegie Mellon University  
    Software Engineering Institute  
    Attn: Dr. Dan Berry  
    Pittsburgh, PA 15260

14. Office of Naval Technology (ONT)  
    Code 227  
    Attn: Dr. Elizabeth Wald  
    800 N. Quincy St.  
    Arlington, VA 22217-5000

15. Defense Advanced Research Projects Agency (DARPA)  
    Integrated Strategic Technology Office (ISTO)  
    Attn: Dr. B. Boehm  
    1400 Wilson Boulevard  
    Arlington, VA 22209-2308

16. Defense Advanced Research Projects Agency (DARPA)  
    ISTO  
    1400 Wilson Boulevard  
    Attn: LCol Eric Mattala  
    Arlington, VA 2209-2308
<table>
<thead>
<tr>
<th></th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.</td>
<td>Defense Advanced Research Projects Agency (DARPA)</td>
</tr>
<tr>
<td></td>
<td>Director, Tactical Technology Office</td>
</tr>
<tr>
<td></td>
<td>1400 Wilson Boulevard</td>
</tr>
<tr>
<td></td>
<td>Arlington, VA 2209-2308</td>
</tr>
<tr>
<td>18.</td>
<td>National Science Foundation</td>
</tr>
<tr>
<td></td>
<td>Division of Computer and Computation Research</td>
</tr>
<tr>
<td></td>
<td>Attn: K. C. Tai</td>
</tr>
<tr>
<td></td>
<td>Washington, DC 20550</td>
</tr>
<tr>
<td>19.</td>
<td>Commander Space and Naval Warfare Systems Command</td>
</tr>
<tr>
<td></td>
<td>SPAWAR 3212</td>
</tr>
<tr>
<td></td>
<td>Department of the Navy</td>
</tr>
<tr>
<td></td>
<td>Attn: Cdr M. Romeo</td>
</tr>
<tr>
<td></td>
<td>Washington, DC 20363-5100</td>
</tr>
<tr>
<td>20.</td>
<td>Office of Naval Research</td>
</tr>
<tr>
<td></td>
<td>Computer Science Division, Code 1133</td>
</tr>
<tr>
<td></td>
<td>Attn: Dr. Gary Koob</td>
</tr>
<tr>
<td></td>
<td>800 N. Quincy Street</td>
</tr>
<tr>
<td></td>
<td>Arlington, VA 22217-5000</td>
</tr>
<tr>
<td>21.</td>
<td>Office of Naval Research</td>
</tr>
<tr>
<td></td>
<td>Computer Science Division, Code 1133</td>
</tr>
<tr>
<td></td>
<td>Attn: Dr. A. M. Van Tilborg</td>
</tr>
<tr>
<td></td>
<td>800 N. Quincy Street</td>
</tr>
<tr>
<td></td>
<td>Arlington, VA 22217-5000</td>
</tr>
<tr>
<td>22.</td>
<td>Office of Naval Research</td>
</tr>
<tr>
<td></td>
<td>Computer Science Division, Code 1133</td>
</tr>
<tr>
<td></td>
<td>Attn: Dr. R. Wachter</td>
</tr>
<tr>
<td></td>
<td>800 N. Quincy Street</td>
</tr>
<tr>
<td></td>
<td>Arlington, VA 22217-5000</td>
</tr>
<tr>
<td>23.</td>
<td>University of CA at Berkeley</td>
</tr>
<tr>
<td></td>
<td>Department of Electrical Engineering and Computer Science</td>
</tr>
<tr>
<td></td>
<td>Computer Science Division</td>
</tr>
<tr>
<td></td>
<td>Computer Science Division</td>
</tr>
<tr>
<td></td>
<td>Attn: Dr. C.V. Ramamoorthy</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA 90024</td>
</tr>
</tbody>
</table>
24. University of MD
   College of Business Management
   Tydings Hall, Room 0137
   Attn: Dr. Alan Hevner
   College Park, MD 20742

25. University of MD
   Computer Science Department
   Attn: Dr. N. Roussapoulos
   College Park, MD 20742

26. University of Massachusetts
   Department of Computer and Information Science
   Attn: Dr. John A. Stankovic
   Amherst, MA 01003

27. University of Pittsburgh
   Department of Computer Science
   Attn: Dr. Alfs Berztiss
   Pittsburgh, PA 15260

28. Commander, Naval Surface Warfare Center,
    Code U-33
    Attn: Dr. Philip Hwang
    10901 New Hampshire Avenue
    Silver Spring, MD 20903-5000

29. Attn: Joel Trimble
    1211 South Fern Street, C107
    Arlington, VA 22202

30. United States Laboratory Command
    Army Research Office
    Attn: Dr. David Hislop
    P. O. Box 12211
    Research Triangle Park, NC 27709-2211

31. Persistent Data Systems
    75 W. Chapel Ridge Road
    Attn: Dr. John Nester
    Pittsburgh, PA 15238
32. Prof. Amr Zaky, Code CS/Za
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

33. Library of
   Chung Shan Institute of Science and Technology
   Lung-Tan, Tao-Yuan
   Taiwan, R.O.C.

34. Library of
   Chung Cheng Institute of Technology
   Ta-Shi, Tao-Yuan
   Taiwan, R.O.C.

35. Department of Electrical Engineering
   Chung Cheng Institute of Technology
   Ta-Shi, Tao-Yuan
   Taiwan, R.O.C.

36. Department of Computer Science
   Chung Cheng Institute of Technology
   Ta-Shi, Tao-Yuan
   Taiwan, R.O.C.

37. Computer Center
   Chung Cheng Institute of Technology
   Ta-Shi, Tao-Yuan
   Taiwan, R.O.C.

38. Tzu-Chiang Chang
    4 F, No.12, Alley 21, Lane 136,
    Ming Chih Rd, Sec.2,
    Tai-Shan, 243, Taipei,
    Taiwan, R.O.C.