5-13-2006

A Study Of Genetic Representation Schemes For Scheduling Soft Real-Time Systems

Amit Bugde

A STUDY OF GENETIC REPRESENTATION SCHEMES FOR SCHEDULING SOFT REAL-TIME SYSTEMS

By

Amit Bugde

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfllment of the Requirements
for the Degree of Master of Science
in Computer Science and Engineering
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2006

A STUDY OF GENETIC REPRESENTATION SCHEMES FOR

SCHEDULING SOFT REAL-TIME SYSTEMS

By

Amit Bugde

Approved:

Yoginder Dandass Assistant Professor Computer Science and Engineering (Major Professor)	Ioana Banicescu Associate Professor Computer Science and Engineering (Committee Member)
Mahalingam Ramkumar Assistant Professor Computer Science and Engineering (Committee Member)	Edward Allen Associate Professor Computer Science and Engineering (Graduate Coordinator)

Roger King
Associate Dean
for Research and Graduate Studies
of the Bagley College of Engineering

Name: Amit Bugde

Date of Degree: May 13, 2006

Institution: Mississippi State University

Major Field: Computer Science and Engineering

Major Professor: Dr. Yoginder Dandass

Title of Study: A STUDY OF GENETIC REPRESENTATION SCHEMES FOR SCHEDULING SOFT REAL-TIME SYSTEMS

Pages in Study: 71

Candidate for Degree of Master of Science

This research presents a hybrid algorithm that combines List Scheduling (LS) with a Genetic Algorithm (GA) for constructing non-preemptive schedules for soft real-time parallel applications represented as directed acyclic graphs (DAGs). The execution time requirements of the applications' tasks are assumed to be stochastic and are represented as probability distribution functions. The performance in terms of schedule lengths for three different genetic representation schemes are evaluated and compared for a number of different DAGs. The approaches presented in this research produce shorter schedules than HLFET, a popular LS approach for all of the sample problems. Of the three genetic representation schemes investigated, PosCT, the technique that allows the GA to learn which tasks to delay in order to allow other tasks to complete produced the shortest schedules for a majority of the sample DAGs.

DEDICATION

To my parents, Shri Chandrakant Bugde and Smt. Smita Bugde.

ACKNOWLEDGMENTS

I would like to thank Dr. Yoginder Dandass, my major professor, for the tremendous support, encouragement, and guidance that he has constantly provided during my Masters program, and for the research described in this thesis. I sincerely appreciate the effort that he has put in guiding me through the entire process, and providing invaluable advice when needed. I would also like to thank him for fnancial support in the form of a Research Assistantship during my Masters program.

I thank Dr. Ioana Banicescu, and Dr. Mahalingam Ramkumar, my committee members, for their encouragement, and technical and career guidance advice.

I would like to thank the offce staff in the Computer Science and Engineering Department, Mrs. Brenda Collins, Mrs. Jo Coleson, and Ms. Brandi Velcek, for being so kind and helpful when needed to complete formalities.

I would like to specially thank Mr. Sirish Kondi, my friend, who has been an invaluable resource during my entire research activity. He has been very kind to help me solve every kind of technical diffculty that I faced. I really appreciate his patience and intellect when helping me understand and solve problems.

I would like to thank my lab mates for participating and providing insightful comments during my practice presentations.

A special thanks to my friends, Mr. Pramod Jain, and Mr. Matthew Morris for their encouragement, healthy discussions, advice and confdence in me.

I thank my brother, Mr. Abhijit Bugde, and his wife, Mrs. Deepa Bugde, for always being there no matter what circumstances I faced. They have always been there to show me a way out of problems.

Last but not the least, I would like to thank my parents Mr. Chandrakant Bugde and Mrs. Smita Bugde for their love and support throughout the entire duration of my Masters program. This endeavor would not have been possible without their constant support.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS, AND NOMENCLATURE

Nomenclature

Absolute deadline The sum of the release time and the relative deadline of a task.

Accurate Method A method for computing tasks' start and completion time PDFs in a stochastic schedule that uses tasks' execution time requirement PDFs and PDF operators.

Child The chromosome resulting from the application of Genetic Operators on parent chromosomes.

Chromosome A coding of a solution to a problem, used in Genetic Algorithms.

Computation-to-communication ratio The ratio of average vertex weight to average edge weight of a DAG.

Cumulative Distribution Function A function that maps a positive time value to a positive real number representing the sum of probabilities that an event occurs at or before each time value.

Estimate Method A method for computing tasks' start and completion time PDFs by constructing an initial deterministic schedule using estimated fx ed values to represent each task's execution time requirements, and using the deterministic schedule to construct the fnal stochastic schedule.

Fitness Criterion Numerical value of which is used to determine the ftness of a solution to a problem, used in Genetic Algorithms

Genetic Algorithms An optimization algorithm based on the principle of natural selection.

Genetic Operators They are used to manipulate chromosomes in Genetic Algorithms.

Hard real-time A real-time system in which the consequence of missing a deadline are devastating.

Independent Random Variables Random variables are mutually independent if the observation of any particular value of one variable has no infuence on the probability of observing any value of the other variables.

Parents The chromosomes on which Genetic Operators will be applied.

Population A collection of solutions to a problem, used in Genetic Algorithms.

Preemption Interrupting the currently executing task in order to execute another task. The interrupted task may be allowed to resume at a later time.

Probability Distribution Function A function that maps a positive time value to a positive real number. The real number indicates the probability with which an event occurs at each time value.

Processing Elements Units on which tasks are scheduled.

Ready Vertices Vertices which are ready to be scheduled.

Receive The channel used by a Processing Element to receive information from other Proessing Elements.

Release time The time relative to the beginning of the schedule when a task becomes ready to execute.

Reproduction Exchange of genetic material, used in Genetic Algorithms.

Send The channel used by a Processing Element to transmit information to other Processing Elements.

Soft real-time A real-time system in which the consequence of missing a deadline are not devastating.

Slot-ftting Threshold The minimum probability with which a task must ft in a slot in a schedule in order to permit the insertion of the task into the slot.

Superior Individuals with high values of the ftness criterion.

Abbreviations

CDF Cumulative Distribution Function

DAG Directed Acyclic Graphs

EVIS Evolutionary Intracell Scheduler

GA Genetic Algorithm

GLS Genetic List Scheduling

GPS Global Positioning System

LS List Scheduling

OX Ordered crossover

PDF Probability Distribution Functions

PosNT Positional with No Thresholds

PosCT Positional with Customized Thresholds

PriNT Priorities with No Thresholds

QoS Quality of Service

SX Standard Crossover

VX Vector Crossover

WCET Worst Case Execution Time

CHAPTER I

INTRODUCTION

This chapter introduces basic concepts of real-time systems and their classifcation. A brief description of real-time scheduling and various types of scheduling algorithms that are available is provided. Motivation behind choosing the problem, and also the application which would beneft from the results of the study has been mentioned. The chapter provides a detailed defnition of the problem, describes the parallel proessing environment, and assumptions made regarding the communication infrastructure.

1.1 Real-Time Systems

A real-time system is one that has a set of tasks that need to complete execution before their respective deadlines. A real-time system needs to not only perform the required computations accurately, but also complete the computations within a certain interval of time. An operation that occurs too late could be useless or even dangerous depending on the application [4]. However, the timing constraints in real-time systems cannot be arbitrarily extended to ensure that tasks complete within their deadlines [5]. For example, a real-time fight control system generally requires sub-second response time to input from the pilot, whereas a weather forecasting system has several minutes or hours to respond to changes in atmospheric conditions [5]. Also, using faster processors does not guarantee

that tasks will meet timing requirements because interaction of tasks with each other and with the environment have to be taken into consideration in real-time sytems [5]. Some examples of applications that require real-time computing include [4] chemical and nuclear plant control, railway switching systems, automotive applications, fight control systems, telecommunication systems, and industrial auomation.

A real-time application is composed of a set of real-time tasks. Instances of real-time tasks are commonly characterized by the following properties in literature [5]:

- *Release time* is the time relative to the schedule's start time when a task instance becomes available for execution.
- *Start time* is the absolute time when the task instance begins execution.
- *Execution time* is the time required by the task to complete execution.
- *End time* is the absolute time when the task completes execution.
- *Deadline* is the absolute time by which a task instance must complete in order to meet real-time performance requirements.

Here, "absolute time" is the time relative to the start of the schedule. These characteristics are depicted in Figure 1.1.

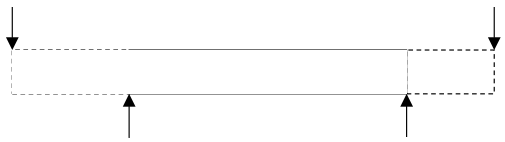


Figure 1.1

Task Execution Characteristics

The consequence of failing to comply with timing constraints is used to classify real-time systems into the following two categories [16]:

- Soft real-time systems: In these types of systems, some degree of fe xibility is acceptable; deadlines may be missed occasionally, but this does not have serious effects. Some examples are *multimedia applications*, *cellular communications*.
- Hard real-time systems: In these types of systems, deadlines are to be met strictly; the cost of missing a deadline is very high. A missed deadline may have disastrous consequences. Some examples are *nuclear plant control*, *fight control systems*.

1.2 Real-Time Scheduling

Scheduling essentially involves allocating system resources to tasks to carry out computations that the tasks are supposed to perform, while keeping in mind the timing constraints that are applicable on task execution. In a system where all tasks are not released simultaneously and tasks cannot be preempted once they have started execution, then the problem of constructing optimal schedules becomes NP-hard in general [5, 17]. Real-time scheduling algorithms can be classifed into the following types based on the properties of the task set under consideration and the objectives of the scheduling algorithm [4, 5]:

Preemptive vs. Non-preemptive. In a preemptive scheduling technique, a task already executing can be interrupted to execute another ready task on the same processor. The execution of the interrupted task is resumed when the interrupting task completes. In a non-preemptive scheduling technique, a task that is executing cannot be interrupted by another task. The other task has to wait till the executing task completes.

Static vs. Dynamic. In a static scheduling algorithm, tasks and task properties are already known, and scheduling decisions are made before the tasks are executed. A dy-

namic scheduling algorithm is one in which scheduling decisions cannot be made before hand since the task set is not known a priori, and can change over the system's lifetime.

Offine vs. Online. In an offine scheduling algorithm, scheduling decisions for the entire task set are made before the system is started. In an online scheduling algorithm, scheduling decisions are made when a currently executing task completes or a new task is ready for execution.

Optimal vs. Heuristic. An optimizing scheduling algorithm minimizes a cost function or maximizes a proft function defned over the system's tasks. A heuristic algorithm strives to achieve optimality, but does not guarantee it.

Scheduling can be performed with a number of goals such as minimizing or improving response time, throughput, completion time, and cost [5]. For hard real-time scheduling, the main goal is to ensure that tasks meet their deadlines; reducing the total length of the schedule is a secondary objective. Traditionally, real-time scheduling typically deals with scheduling hard real-time systems in which the *worst-case execution time* (WCET) of tasks is used to construct deterministic schedules that guarantee the tasks' execution within given time constraints. Although this use of pessimistic execution time assumptions provides real-time guarantees, it comes with the cost of decreased application performance and resource utilization.

Since soft real-time systems can tolerate applications missing occasional deadlines, considerable fe xibility can be afforded in scheduling policies that allow balancing the need for meeting time constraints with the need for improved performance. Such systems

can improve resource utilization and performance based on the fact that, in a given interval of time, the likelihood of all tasks simultaneously requiring their WCET is small. However, as opposed to the pessimistic allocation of the WCET for each task, such a scheme that allocates less than the absolute maximum required time will lead to occasional missed deadlines if tasks require more time than was allocated for them.

There are a number of systems in practice that tolerate occasional deadline misses. For example, in multimedia systems, video frames are decoded and displayed at a fx ed rate. If the system misses a frame-decoding deadline, then either a partial frame is displayed or the frame is skipped entirely. Therefore, viewers will tolerate the slight degradation in video quality resulting from an occasional deadline miss. Some systems may have critical timing requirements, but nevertheless can be considered to be soft real-time systems. An example of this can be a stock price quotation system [16]. It is expected to update the price of each stock as its price changes. In such a system, a late update is highly undesirable. However, in a very dynamic market, occasional late or missed deadlines can be tolerated as a trade-off for other factors, such as the cost of installation and maintenance of the system, and the sheer number of users that the system can serve simultaneously.

This research investigated heuristic techniques for developing static, non-premptive, offine schedules for soft real-time applications in which system resources are utilized more effciently than possible when WCET-based scheduling is performed. The schedul-ing techniques will enable system designers to predictably trade schedule length for *quality-of-service* (QoS). QoS is the probability that the schedule will complete by the assigned

deadline. Essentially, after the schedule is constructed, the system designer will be able to determine the amount of time required by the real-time system in order to guarantee a level of QoS. This research extends and improves on the previous work of Dandass [6] in using genetic list scheduling (GLS) algorithms for scheduling real-time applications that are represented as directed acyclic graphs (DAGs). The schedules produced by the GLS-based techniques in [6] were of higher quality than those produced by non GLS approaches. However, an analysis of those schedules revealed opportunities for further optimization. The techniques in this paper improve on the GLS algorithm and result in schedules with further reductions in schedule lengths.

1.3 Motivation

One class of application to which the scheduling technique developed in this research can be utilized for is a *mobile augmented reality system*. In such systems, computer-generated graphics, termed "augmentations," are rendered so that they overlay relevant parts of the real world in order to enhance information content. Such processes need signifcant real-time computation capacity while dealing with constraints of volume, mass, power consumption, and heat production. The applications for which such systems will be used need to compute and produce results on the input data at a high frame rate. An example application is a fre fghter wearing such a mobile computing system and entering a smoke flled room. Image augmentation such as location of doors, windows, pipes, and electrical wires are overlayed on a transparent display through which the fre fghter is

viewing the real world. Such a system may have video and audio inputs, inputs from a heat sensor, and a Global Positioning System (GPS), all providing information to be displayed to the fre-fghter . The image should correspond to what the fre fghter sees in real time. The system needs to process huge amounts of information quickly; a new image must be rendered based on the orientation of the fre fghter' s head. Tradionally, dedicated processors are allocated to process each input stream; each processor performing one particular type of job. This makes the augmented reality system bulky, consuming large amounts of power, which in turn leads to the problem of increased heat dissipation. This can be avoided by allocating the tasks to a fnite set of homogeneous processors, capable of performing any task assigned to them. The problem then is to obtain effcient scheduling techniques for allocating fnite computing resources if such mobile augmented reality systems are to be built and to be used more commonly.

There are a number of scheduling techniques available, such as *Rate Monotonic Schedul-ing* (RMS) and *Earliest Deadline First* (EDF) [16]. RMS prioritizes tasks according to their period; tasks with shorter periods are given a higher priority. The EDF algorithm ex-ecutes the task with the earliest absolute deadline. However, these algorithms are suitable for scheduling tasks in hard real-time systems. Research in [20] and [1] also dealt with scheduling of real-time tasks. These scheduling algorithms are restricted to uniprocessor systems and hence cannot be utilized in this research. A number of scheduling strategies have also been developed in the feld of Operations Research (OR). However, these can-not be applied directly to real-time scheduling because the models used do not accurately

represent realistic real-time task execution requirements. Also, scheduling techniques developed in OR generally do not deal with tasks that recur, need synchronization or ones that communicate with each other and transfer information, and hence, these techniques seem impractical for scheduling of real-time tasks.

1.4 Problem Defnition

The problem is formally described as follows: *given a directed acyclic graph (DAG), representing a soft real-time application, devise a scheduling algorithm that minimizes schedule lengths while simultaneously enabling the predictable tradeoff of quality-of-service for improved resource allocation.*

The soft real-time applications are composed of two types of tasks:

- *computation tasks* which are a series of computations.
- *communication tasks* which are responsible for transferring information from one processing unit to another.

A DAG $G = (V, E)$ consists of a set $V = (v_1, v_2, ..., v_n)$ of n vertices and a set $E = (e_1, e_2, ..., e_k)$ of k directed edges connecting the vertices. The vertices represent computational tasks and the directed edges represent communication operations. The edge direction specifes the direction of communication. The ordered pair $e_i = (v_{src}, v_{dst})$ indicates that the direction of edge e_i is from vertex v_{src} to v_{dst}. The edges also determine task precedence constraints that have to be satisfed in the application.

Preemption used in most periodic real-time scheduling algorithms can reduce performance because of additional context switching times, and reduced locality (*e.g.*, cache

content and branch prediction table entries setup for the original task are disturbed by the interrupting task) [5]. For this reason, the scheduling technique developed in this research prevents real-time tasks from being preempted at arbitrary instances of time. Instead, tasks can be preempted only at vertex or edge boundaries. Hence, applications which involve rapid task switching require large tasks to be partitioned into strings of vertices or edges. This restriction on preemption enables a more accurate determination of individual tasks' execution times because the disruptive effects as mentioned above are isolated to task boundaries.

In this research, the timing requirements of tasks are assumed to be varying in order to account for the uncertainty in the time to complete these tasks. The execution time requirement of a task is modeled as an *independent random variable*. The assumption of independence of task execution requirements is justifed because the causes of the variance in a task's execution time requirements are restricted to the effects of the processor architecture (*e.g.*, cache, branch prediction); variances in execution times caused by data characteristics (*e.g.*, size and locality) and execution fo ws (*e.g.*, different conditional branches) are excluded [5]. Hence, the class of problems that are addressed here are primarily in the domain of real-time signal and image processing applications where successive "frames" of data gathered by sensors are processed repeatedly by a dedicated system.

The independent random variables representing execution time are expressed in the form of *probability distribution functions* (PDF). A PDF maps a time quantity representing the execution time requirement of the task to the probability that the task will require that

much time to execute. Figure 1.2 shows an example PDF of the execution time of a task. Execution of the task may take anywhere between 3 and 10 ticks, and the probability with which the task executes in a given number of ticks is given by the histogram.

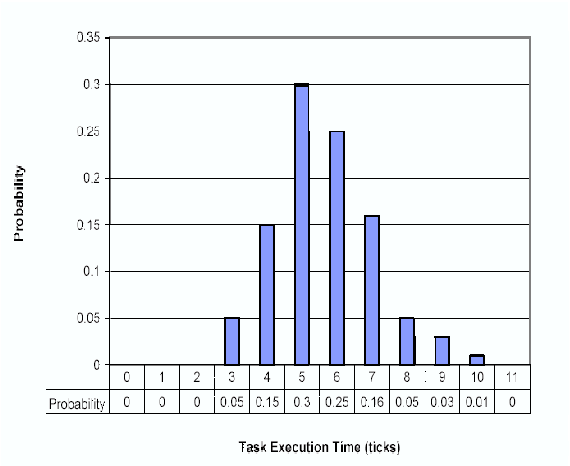


Figure 1.2

A Representative PDF for Task Execution Time Requirements [5]

Figure 1.3 depicts a hypothetical DAG. Edges $e_{\underset{n}{1}}$, $e_{\underset{n}{2}}$, $e_{\underset{n}{3}}$, and $e_{\underset{n}{4}}$ are designated as $(v_{\underset{n}{0}}, v_{\underset{n}{3}})$, $(v_{\underset{n}{1}}, v_{\underset{n}{3}})$, $(v_{\underset{n}{2}}, v_{\underset{n}{3}})$, and $(v_{\underset{n}{3}}, v_{\underset{n}{4}})$, respectively. Task execution time probabilities are depicted at the right. For example, when vertex $v_{\underset{n}{0}}$ is executed, it can take 4, 5, or 6 time units to complete with equal probability.

It is assumed, without loss of generality, that the vertex and edge weights are specifed as integer values, and that the weight probabilities are non-zero over a fnite range of weight values. This is a valid assumption because real-time systems are designed to have

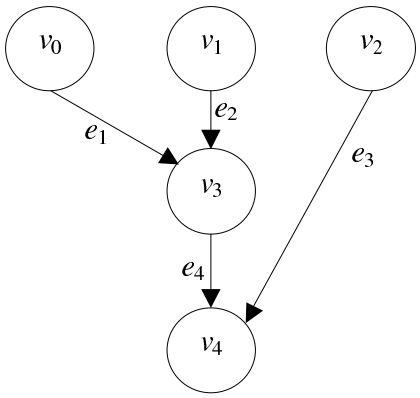
as little variance in execution time as possible. This assumption implies that vertices and edges have weight values only within a well-defned range of integers.

Given the PDF of the start time and execution time of task J_i, the completion time PDF of J_i is computed by the convolution of the start time PDF and the execution time PDF of J_i [5]. Task J_{i+1} is started immediately after J_i completes and its start time PDF is essentially the completion time PDF of J_i that has been *translated* (*i.e.*, shifted) to the right by one time unit. Therefore, in this research, task start time and end time are also represented by PDFs.

The parallel application is assumed to execute on a *homogeneous* multiprocessor machine. All processors in a homogeneous machine are identical to each other in computational capacity. A computational task will thus take the same amount of time to execute on any of the processors. Also, a uniform point-to-point network capacity is assumed over the entire parallel system. This implies that the time needed to complete a particular communication operation is the same over any combination of distinct source and destination processors.

This research presents an offine scheduling technique with two main objectives:

- Create a schedule of tasks in the DAG on a parallel machine so as to minimize the make-span of the schedule while utilizing as few processors as possible. If two schedules have identical make-spans, the schedule requiring fewer number of processors is preferred.
- Compute the completion time PDF of the application. This PDF provides a mean for precisely determining the amount of reduction in the application execution time that can be obtained by making a compromise on the probability of meeting end-to-end deadlines.



Edges e_1, e_2, e_3, and e_4 are designated as (v_0, v_3), (v_1, v_3), (v_2, v_3), and (v_3, v_4), respectively. Task execution time probabilities are depicted at the right. For example, when vertex v_0 is executed, it can take 4, 5, or 6 time units to complete with equal probability.

Task				
v_0	**Weight:**	**4**	**5**	**6**
	Probability:	1/3	1/3	1/3
v_1	**Weight:**	**7**	**8**	**9**
	Probability:	1/3	1/3	1/3
v_2	**Weight:**	**1**	**2**	**3**
	Probability:	1/3	1/3	1/3
v_3	**Weight:**	**1**	**2**	
	Probability:	1/2	1/2	
v_4	**Weight:**	**2**	**3**	**4**
	Probability:	1/3	1/3	1/3
(v_0, v_3)	**Weight:**	**1**	**2**	**3**
	Probability:	1/3	1/3	1/3
(v_1, v_3)	**Weight:**	**7**	**8**	**9**
	Probability:	1/3	1/3	1/3
(v_2, v_4)	**Weight:**	**1**	**2**	**3**
	Probability:	1/2	9/20	1/20
(v_3, v_4)	**Weight:**	**7**	**8**	**9**
	Probability:	1/3	1/3	1/3

Figure 1.3

A hypothetical DAG [5]

Schedules are created for a fnite set of *processing elements* (PEs). PEs are representatives of the actual processor units in the homogeneous multiprocessor system. Each PE is composed of a processor, and a pair of send and receive channels. The processor is responsible for executing computation tasks; the send and receive channels are responsible for executing communication operations. Thus, when a schedule is created, vertices are scheduled on the processor part of the PE, and edges are scheduled on the send and receive channels. Each channel is simplex; the *send* channel is used to transmit information to other PEs, while the *receive* channel is used to receive information from other PEs. The send and receive channesl are a pair of simplex channels that form a full-duplex communication interface to the network. A PE can perform a single communication operation on each simplex link. Simultaneous send and receive operations can occur, however, two send or two receive operations cannot occur simultaneously. The PEs are assumed to be interconnected by point-to-point links, thus forming a fully connected network topology. The network is assumed to be congestion-free and capable of transmitting data without loss or communication errors. Communication and computation tasks can be scheduled simultaneously on a PE provided their start times do not depend upon the completion time of the other, in other words, when the communication and computation tasks are independent of each other.

1.5 Approach

This research employs Genetic Algorithms (GAs) for scheduling of computation and communication tasks. GAs are a broad class of algorithms that are analogous to natural evolution [10] and are based on the principle of "survival of the fttest." GAs try to obtain an optimal solution to a problem by manipulating a coding of the solution rather than the solution itself. A GA maintains a population of individuals, known as *chromosomes*, where each chromosome is an encoding of a solution to the problem. Genetic operators are applied to these chromosomes to produce new individuals, which are added to the population. A ftness criterion is used to decide whether an individual should be included in the development of further generations or not. In this research, the length of the schedule generated by a chromosome is used as a ftness criterion; the shorter the length, the better the schedule. GAs have been shown to be robust in optimization problems [23] because they can effectively and effciently search large search spaces and converge on a global optima [7]. Genetic List Scheduling (GLS) techniques will be applied to obtain schedules for the real-time application.

A schedule is created by allocating ready tasks to processors that can allow the earliest execution of that task. The genetic representation is used for prioritizing the DAG's vertices and equally importantly, its edges. In this research, two new priority-encoding schemes, PosCT and PriNT were implemented and compared to the PosNT scheme previously used by Dandass [5]. These schemes are discussed in detail in Section 3.6. Once a vertex is selected based on the prioritization schemes, an *empty slot* is found on a proces-

sor that begins at or after the start time of the vertex. An empty slot is an interval of time in which no task has been scheduled. A vertex can be scheduled in a slot if the previously scheduled vertex on the same processor has completed and the incident edges on the ready vertex have been scheduled. The start time PDF of a task is calculated by the maximum of the PDFs of the two independent preceeding tasks. This is explained in more detail in Chapter 3. An edge can be scheduled in a common time slot on the send link of the source PE and on the receive link of the destination PE that starts after the source vertex has completed. As mentioned in Section 1.4, the end time PDF for each task is calculated by convoluting the start time and execution time PDFs of the task. This is also explained in more detail in Chapter 3.

PDF manipulations are expensive to perform, hence the time required to construct a schedule becomes prohibitively large. Dandass in [5] has used a fx ed estimate of the execution time requirements of each task instead of the tasks' execution time PDF to construct an initial schedule, from which task-resource allocations and task sequences are used to construct the fnal stochastic schedule using task execution time PDFs. This is discussed in more detail in Section 3.7. The *expected value* of the PDF is used as the fx ed estimate. Two variations of the PosCT scheme are developed in this research: PosCT-Fixed and PosCT-Variable. The PosCT-Fixed uses the fx ed estimate of the execution time PDF, whereas PosCT-Variable uses the detailed start and end time PDFs of the vertices and edges for obtaining better schedules. The difference between PosCT-Fixed and PosCT-Variable are discussed in Section 3.10.

CHAPTER II

LITERATURE SURVEY

This chapter summarizes related work in scheduling of real-time tasks using GAs, deterministic and probabilistic real-time scheduling, and sechduling with multi-processors. It also contains limitations of existing real-time scheduling research.

2.1 Scheduling with Genetic Algorithms

Grajcar in [10] worked on mapping a partially ordered set of tasks communicating over a shared bus to a heterogeneous multiprocessor system, with the goal of minimization of the makespan, taking into consideration the constraints due to data dependencies and resource usage. An approach based on list scheduling (LS) and genetic algorithms (GAs) is presented. The problem is essentially to map a task graph, with nodes as tasks, and edges as communications, onto a target architecture, consisting of a set of processing modules and a set of busses. The result of the algorithm is a schedule that determines the assignment of each task to a resource, and its starting time. The algorithm is capable of handling preemption, however it does not handle migration since heterogeneous processor systems do not support migration. It is essentially a genetic algorithm using list scheduling, in which two parents are chosen to create an offspring, which is evaluated using list scheduling, and an unfeasible individual is selected to be replaced. Evaluation of an individual is done by

creating a schedule using its chromosome and computing the resulting makespan. This algorithm is a heuristic and thus, it cannot be proved that a given solution is optimal.

Montana *et al.* in [7] discussed factors such as large search spaces, dynamically changing problems and variety of constraints that make real-time scheduling diffcult. In their approach, they use an ordered-pair representation for scheduling, each pair consisting of a task, and a resource that is to be used to execute the corresponding task. Genetic operators shuffe task-resource parings as well as the sequence of tasks. The evaluation function measures the goodness of the schedule. To handle dynamically changing problems, the genetic algorithm works on a fx ed problem for a fx ed amount of time, then the best schedule obtained at that point is modifed to take into account the changes that have taken place in this time. Thus, they claim that their approach of using genetic algorithms, reconciliation of changes, and incorporation of hard and soft constraints into genetic operators and an evaluation function addresses the factors that make real-time scheduling diffcult.

Kim *et al.* have talked about a genetic reinforcement algorithm for the machine scheduling problem in [14]. They have developed a genetic reinforcement learning scheduler called *EVIS* (Evolutionary Intracell Scheduler) that is applied to various classes of the machine scheduling problem, and also to the processor scheduling algorithm. It can be looked upon as a search for an optimal priority-list in a pool of priority-lists. It has been shown that the learning-based heuristic is robust and its performance is comparable to other problem-specifc heuristics. EVIS is an implementation of reinforcement leaning with delayed feedback, which determines a maximal reward policy, given a policy gener-

ation method, an evaluation function, an updating function, and a stopping condition. The stopping condition can be either maximum number of generations or minimal improvements in the best chromosome during a specifed number of generations. For application of EVIS to the processor scheduling problem, the cross-over operator is not used. A schedule consists of operation-to-processor assignments as well as operation starting times.

Highest Level First with Estimated Times (HLFET) [2] is a simple and fast LS heuristic in which ready vertices are scheduled according to non-decreasing order of the longest path between the ready vertex and a *terminal* vertex, which is one that does not have any outgoing edges, in the DAG. In this research, HLFET is used as the LS approach to create schedules for each of the DAGs. The schedules created by HLFET are compared with those created by the GLS approaches.

Grajcar in [11] has talked about the strengths and weaknesses of genetic list scheduling for heterogeneous systems. The main lacuna of list scheduling is the lack of information about tasks that are not scheduled yet. Moreover, most list scheduling techniques work with a number of assumptions about the computing environment, such as the processors being alike, no competition for the communication channel, among others, which may not be always true for real-time or multimedia systems. This research treats communication as individual tasks too. Thus, invalidating the assumption of the absence of bus contention. Also, the author mentions that most heuristics ignore precedence constraints, but the heuristic used in this research does not. Information about tasks not scheduled yet can be utilized for determining if a given task may be allotted to a given resource. Looka-

head as described in [3] may be used to some extent to try to solve the stated weakness of list scheduling.

In [6], Dandass has combined list scheduling with genetic algorithms for constructing non-preemptive schedules for soft real-time parallel applications, represented as DAGs, where the task execution time is given in the form of a PDF. The parallel machine used for executing the application is assumed to be a homogeneous machine, where each processor is identical to each other, so that the time taken to execute a computational task is the same on any of the processors. Each processor is assumed to have separate send and receive channels for transmitting and receiving data respectively; these are used to schedule the communication tasks. The execution time of each task was modeled as an independent random variable. The problem was to schedule the tasks in the DAG using the least number of processors and then computing the completion time PDF of the application. List scheduling was used to generate the chromosomes for the population, on which genetic operations were performed. The resulting offspring chromosome was evaluated for its ftness, and the worst chromosome in the population was replaced by the offspring. The vertices to be scheduled are selected from a ready list, and their prioritization is determined by their order in the chromosome being considered. A chosen vertex is then scheduled by searching for idle time slots in which incident edges for the vertex, and the vertex itself can be executed. This is done on all of the available processors and the vertex is fnally scheduled on that processor that allows it to start the earliest. The execution time requirement of each vertex is given by the convolution of the start time PDF and the execution time

PDF of the vertex. The results obtained showed that using the genetic algorithm produced shorter schedules than list scheduling approaches for a sample set of problems.

2.2 Probabilistic and Stochastic Scheduling

A number of scheduling algorithms are available for scheduling real-time tasks. Some commonly used algorithms are *Earliest Deadline First*, *Rate Monotonic*, and *Deadline Monotonic* [16]. However, most of these algorithms rely on preemption of tasks and assume fx ed task execution time requirements.

The method developed in [20], known as *Probabilistic Time Demand Analysis* schedules semi-periodic tasks by treating them as periodic tasks and scheduling them on a fx ed-priority basis. It tries to fnd the probability that any request meets its deadline. This is done by computing the probability from the cumulative probability distribution of the total amount of processor time demanded by higher priority tasks. Abeni and Buttazzo in [1] try to solve the problem of soft real-time scheduling by using a *Bandwidth Reservation Strategy*. According to this strategy, each task is assigned a part of the CPU bandwidth and the scheduling mechanism ensures that the task will not require more than the reserved bandwidth. A task demanding too much time is just delayed, and it does not compromise the QoS guaranteed for other tasks. However, these techniques typically assume that tasks can be preempted, and the preemption cost is negligible, and also these techniques are generally restricted to single processor systems.

Dogan and Ozguner in [9] have tried to solve the problem of stochastic scheduling of a tasks in a heterogeneous distributed computing system. They have performed simulation studies, which showed that using a stochastic scheduling algorithm instead of a deterministic scheduling algorithm improved the performance of scheduling tasks in a heterogeneous system. They have developed a genetic algorithm based scheduling algorithm that makes scheduling decisions either stochastically or deterministically. Task execution times have been treated as random variables. They have assumed the objective of reducing the expected value of the length of the schedule. However, algorithms from [9] cannot be directly used in this research because they deal with independent tasks that have no data dependencies, whereas in this research, tasks having precedence constraints are being considered.

In [3], Beaty has talked about the weaknesses of list scheduling, mainly when dealing with restricted timing. Two methods have been developed, namely *foresight* and *lookahead*, that act to mitigate this weakness. Foresight checks to see whether all those operations that become constrained after scheduling a particular operation can be "easily" scheduled, considering the constraints and conficts caused by resources. If an operation can be scheduled, then it is, and foresight is repeated for the subsequently following operations. The lookahead method was developed with the aim of reducing the scheduling time and increasing the chances of creating valid schedules. It places operations instead of just testing for the possibility of placement; it can remove any or all the nodes in the ready set and make successors of these nodes available for scheduling. After experimentation, it

was found that foresight and lookahead were more important for forming valid schedules than choosing good heuristics, and that lookahead was able to enhance the ability of list scheduling to generate valid schedules.

2.3 Scheduling for Multi-processors

Mingsheng *et al.* in [17] present a list scheduling scheme to schedule tasks of a DAG onto a homogeneous multiprocessor system, with the aim of minimizing not only the schedule length, but also the scheduling time. In the paper, they propose a list scheduling algorithm based on critical paths; all nodes belonging to the critical path are to be scheduled, as soon as they are ready, since they have the greatest infuence on the scheduling length of the task graph. When nodes are to be assigned to processors, then critical path nodes are made to have the earliest start times. This algorithm has a time complexity of $O(pv^2)$, where p is the number of processors and v is the number of nodes in the graph. They compare this algorithm with other list scheduling algorithms and show that the others do not guarantee earliest scheduling of the critical path nodes, and that the time complexity of their algorithm is no more than the others.

In [19], Ramamritham *et al.* describe scheduling algorithms based on heuristic functions for real-time multiprocessor systems. Simulation is used to evaluate two scheduling algorithms; one considers all the tasks not yet scheduled as candidates, where as the other chooses a subset of tasks with the shortest deadlines. They show that the latter is very

effective when the maximum allowable scheduling overhead is fx ed, which makes it appropriate for dynamic scheduling in real-time systems.

In [22] Wang *et al.* have tried to establish bounds on the performance of heuristic algorithms for multiprocessor scheduling of hard real-time tasks by analyzing the performance of list scheduling and *H-scheduling* algorithms. They have taken into consideration two performance aspects to evaluate heuristic algorithms, which are the *ability* of an algorithm to generate a feasible solution, and the *quality* of the solution. The metric used to measure the ability is the ratio of the number of task sets for which the algorithm has found feasible schedules to the total number of task sets at hand, whereas the length of the schedule is used to determine the quality of the schedule. Simulation was used for the analysis and it is shown that tasks with the same computation times as well as those with arbitrary computation times, the complexity is $O(n^{k+1}r)$ for $2 < k <= \frac{m}{n}$, and $O(n^2r)$ for $k = \frac{2}{n}$, where *n* is the number of tasks, *m* is the number of processors, and *r* is the number of resources. However, these research efforts assume tasks with fx ed execution time requirements, which make these algorithms deterministic in nature.

CHAPTER III

APPROACH

3.1 Objective

An objective of this research is to extend the scheduling technique described in [5] for scheduling of soft-real-time parallel applications. They are represented in the form of directed acyclic graphs, with each vertex representing a computation task and each directed edge representing communication between different tasks. The communication operations decide precedence relations between computation tasks.

3.2 Hardware

The parallel application is assumed to execute on a homogenous set of prcossors. Each processor is similar to every other processor in terms of performance, so that a task can be allotted to any of the available processors. The tasks in the application cannot be preempted at any arbitrary instance of time, but can be preempted only at vertex or edge boundaries. This is done to enable a more accurate determination of individual tasks' execution times because the disruptive effects of interrupt handling and task switching are isolated to task boundaries. The application may be a periodic application, in which case, once the optimum schedule is obtained, the entire schedule is repeated to represent the periodic nature of the application. *Processing elements* are the units which are used for

scheduling the computation and communication tasks. Each processing element consists of a processor, and a pair of send and receive channels. The processor is capable of, but restricted to, executing the computation task that is assigned to it. The communication channels are used for transfer of information. Each channel is simplex; the *send* channel is used to send information out of the processing element, while the *receive* channel is used to receive information from another processing element. Thus, we have a pair of simplex channels that form a full-duplex communication interface to the network. Each of the processing elements is assumed to be connected to each other by point-to-point links, thus forming a fully connected network topology. The network is assumed to be congestion-free and capable of transmitting data without loss or communication errors. Communication and computation tasks can be scheduled simultaneously on a processing element provided they are independent of each other.

3.3 Tasks and Task PDFs

Each task has variable start and execution times, which cannot be fx ed in advance. These processing time requirements are modeled as *independent random variables* with bounded minimum and maximum values [5]. Independence of random variables implies that observation of any particular value of a variable is not infuenced by nor does it infu - ence observed values of any other variable. Bounds on the values of end times are valid because real-time tasks are designed to reduce execution time jitter and hence cannot have unbounded end times. These independent variables have values that are given in the form

of a PDF. For experimental purposes, three different types of distributions will be considered, *viz.* exponential, beta, and random. Scheduling algorithms calculate the end time PDF of a task by using its execution time PDF and the end time PDF of the preceding task. In [5], Dandass states that computation costs for such PDF manipulations are high, and thus a fx ed estimate of the execution time requirements of each task is used instead of the tasks' execution time PDF to construct an initial schedule, from which task-resource allocations and task sequences are used to construct the fnal stochastic schedule using task execution time PDFs.

3.4 Genetic Algorithms

Genetic algorithms are a broad class of algorithms that are analogous to natural evolution [10]. Operations taking place in genetic algorithms mimic biological principles such as "survival of the fttest". Genetic algorithms have been shown to be robust in various optimization problems [23]. During their operation, they maintain *populations* of possible solutions to a problem [7]. Each of the solutions is called a *chromosome*. Thus at each step in the algorithm there is a gene pool consisting of *genes* of the chromosomes contained in the population. Genetic algorithms manipulate a coding of the solution, which is the chromosome, rather than the solution itself [18]. After generation of an initial population, applying genetic operators on the chromosomes creates a new population. Individuals in a population are assessed based on a *ftness criterion*. The chromosomes having higher values of the ftness criteria are supposed to be *superior*, and they are selected for *repro-*

duction. Reproduction can take place by the application of one or more *genetic operators*, such as selection, crossover, and mutation. The operators merge the chromosomes of the *parents*, giving rise to a *child*, possibly combining the desirable properties of the parents [10].

The selection operator is used to select the best individual from the entire population or a subset of any size of the population. The score or value of an individual against the ftness criteria is used for comparison. The crossover operator resembles the exchange of genetic material that takes place during reproduction in nature. The child generated as a result of a crossover operation on the parents has qualities of both the parents in it. However, the crossover of two individuals may not always generate an individual better than the previous ones; it just generates a different individual [13]. Each of the results of reproduction also need to be tested against the ftness criteria, and may be eliminated if they are very unft. In some cases, such unft individuals may be retained in the population in order to explore new parts of the search space, which otherwise would not have been accessible. The mutation operator randomly exchanges positions of genes in the selected chromosome to create a completely new individual. The number of mutations taking place in the population is controlled by the mutation rate [13]. Variations of the basic genetic operators may be used in order to create a variety of individuals so that the algorithm does not converge to local maxima. This research uses a *steady state* genetic algorithm in that new chromosomes obtained by genetic operations immediately replace members of the current population.

Genetic algorithms have been successful in various optimization problems since they are capable of effectively and effciently searching large search spaces to fnd nearly global optima [7]. They use an objective function, which is the ftness criterion, to evaluate the quality of solutions to guide their search as opposed to heuristics, which often rely upon problem specifc information for getting results [23]. Genetic algorithms can be easily parallelized; different processors can work with different populations, thus obtaining a very large variety of solutions to the same problem, increasing the chances of getting an optimal solution. Also, information between different processors can be exchanged so that there is variation in the genetic material that each processor works with.

3.5 List Scheduling and Genetic List Scheduling

A number of heuristic algorithms have been proposed for constructing schedules for DAGs to minimize the make-span based on the List Scheduling (LS) algorithm. The fundamental LS algorithm consists of steps as shown in Figure 3.1. LS is an iterative algorithm, in which, in each iteration, a list of *ready vertices* is constructed. A ready vertex is one whose precedence constraints have been met. The list is then prioritized according to a scheme and then the ready vertex with the highest priority is scheduled on the processor that allows the earliest execution of the task associated with that vertex. Prioritization of the vertices is performed according to a variety of heuristics, and the heuristic used has a profound impact not only on the length of the schedule, but also on the amount of time that is required to construct the schedule.

1. Construct a *ready list* of vertices with no preceding vertices.
2. Loop while vertices remain in the ready list:
3. Prioritize the ready list.
4. Remove the highest priority vertex from the ready list and schedule it on the processor that will allow the earliest start time for this vertex.
5. Add the newly readied vertices to the ready list.

Figure 3.1

The Fundamental LS Algorithm [6]

Kwok and Ahmad [15] have proposed an effective technique for combining Genetic Algorithms (GAs) with the LS technique. Such a combination of GA and LS is known as *Genetic List Scheduling* (GLS). In GLS, chromosomes contain information that is used to decide the order in which the tasks are scheduled. Genetic operators are applied on individuals in a population to obtain better individuals. This iterative process is repeated till an optimal schedule is obtained. Figure 3.2 shows the steps in the fundamental GLS algorithm.

In [5], GLS has been successfully applied to scheduling DAGs with multicast edges in the presence of precedence constraints. The fundamental algorithm shown in Figure 3.2 is easily parallelizable. The advantage of using GAs is that they are able to search a large search space easily, without the need of having sophisticated models to describe the problem.

1. Generate the initial population.
2. Use LS to construct a schedule in order to evaluate each of the initial chromosomes.
3. Loop while the termination criteria are not satisfied:
    4. Select a genetic operator.
    5. Select chromosome(s) from the local population and apply the operator to produce the offspring chromosome.
    6. Use LS to construct a schedule in order to evaluate the offspring chromosome.
    7. Select chromosome from the local population to be replaced by the offspring chromosome.
8. Use the fittest chromosome to construct the solution schedule.

Figure 3.2

The Fundamental GLS Algorithm [6]

3.6 Genetic Representation

Most existing LS and GLS algorithms focus on prioritizing vertices in the ready list and can schedule the incoming edges of a vertex in an arbitrary manner because communication contention is ignored. However, when communication contention is allowed, the order in which edges are scheduled also impacts schedule length. Therefore, the genetic representation in this research is used for prioritizing the DAG's vertices and equally importantly, its edges. The following three distinct priority encoding schemes are used in this research.

3.6.1 Positional with No Thresholds (PosNT)

In the Positional with No Thresholds (PosNT) encoding scheme, each chromosome in the GLS has two vectors of genes. The vertex vector contains a gene for each vertex in the

DAG and the edge vector contains a gene for each edge in the DAG (*i.e.* there are $|V| + |E|$ genes in each chromosome). Each gene is a 32 bit value identifying the corresponding task (vertex or edge) in the DAG. The position of the vertex and edge genes in their respective vectors determines the priority of the corresponding vertices and edges used by the list scheduler. For example, consider two ready vertices v_x and v_y appearing at indices i_x and i_y respectively in the vertex gene vector. If $i_x < i_y$, the pointer for v_x appears before the pointer for v_y. In this case, v_x is given a higher priority than v_y. Edge priorities are similarly determined by the ordering of edge genes in the edge gene vector. The PostNT-based GA searches for an optimal ordering of tasks in the chromosomes.

3.6.2 Positional with Customized Thresholds (PosCT)

In the Positional with Customized Thresholds (PosCT) encoding scheme, in addition to the positional vector and edge genes as in PosNT, there is a third vector of genes making the total number of genes in a chromosome equal to 2 * ($|V| + |E|$). The vertex and edge vectors are identical in structure and function as in the PosNT representation described previously. The third vector contains *overlap threshold* genes, one for each vertex and edge in the DAG. The threshold gene specifes the overlap threshold value for the corresponding task represented as an 8-bit unsigned integer. It is a fractional value in the interval [0,1] computed by dividing the gene value by 255. The overlap threshold is used to determine if a task (vertex or edge), T_C, to be scheduled on processor P_S such that another task, T_S, already in the schedule for P_S, is delayed in order to allow task T_C to

execute. (Section 3.10 contains additional details on thresholds which are used for decreasing schedule length.) Unlike the positional genes, the threshold genes occur at fx ed locations in the gene vector (*i.e.*, the threshold for vertex $v_{\frac{x}{n}}$ is located at position x in the overlap threshold gene vector and that for edge $e_{\frac{y}{n}}$ is located at position $|V|_{\frac{n}{n}} + y$. In addition to searching for optimal vertex and edge positions, the PosCT-based GA also searches for optimal threshold assignments for tasks. There are two variations of the PosCT approach: the *PosCT-Fixed* and the *PosCT-Variable*. PosCT-Fixed uses a fx ed estimate of the start and end time PDFs of the tasks rather than using the PDFs themselves. This is done so as to decrease the time required to actually create a schedule. PDF manipulations and PDF operators are computationally expensive, and thus, using entire PDFs require prohibitively large amounts of time. The PosCT-Variable uses the entire start and end time PDFs instead of the fx ed estimate in constructing schedules. Schedules using PosCT-Variable were constructed for less than 12% of the total number of DAGs used for testing the fx ed estimate approaches. Moreover, in PosCT-Fixed, the value of the overlap gene is compared to the ratio of overlap versus task weights, whereas in PosCT-Variable, the value of the overlap gene is compared to the probability that a ready task fnishes before an already existing task. This is explained in more detail in Section 3.10.

3.6.3 Priority with No Thresholds (PriNT)

In the Priority with No Thresholds (PriNT) encoding scheme, the genetic representation is identical in structure to that of PosNT. The difference is that in this scheme the genes

directly encode the priority of the vertices and edges (recall that in PosNT, the priority of tasks was determined indirectly from their relative positions in the chromosome). There are $|V| + |E|$ genes in each chromosome. The priority of each task in the DAG is represented by a 32-bit integer value at a unique (and fx ed) offset in the chromosome (*i.e.*, the priority gene for vertex v_x is located at position x in the priority gene vector and that for edge e_y is located at position $|V| + y$.) The PriNT-based GA searches for an optimal prioritization of vertices and edges in the chromosomes.

3.7 Schedule Construction and Fitness Computation

This research adopts the schedule construction technique developed by Dandass in [5]. Given a chromosome, the LS portion of the GLS assigns vertices and edges in the DAG to the processors and processor-to-network links in the parallel machine. During each iteration, the algorithm schedules the highest priority ready vertex. The list of ready vertices is initially populated with vertices that do not have any preceding edges. During each iteration, the ready vertices are prioritized according to the genetic encoding scheme (*i.e.*, PosNT, PosCT, or PriNT) in use. In PosNT and PosCT, the ready vertices are prioritized by their relative positions in *C*. In PriNT, the priority information directly encoded in the chromosome is used to select the highest priority vertex, v_r. Conceptually, the start time of v_r is determined by temporarily scheduling the vertex on a processor in the parallel machine, *M*. Then, this temporary scheduling is reversed before scheduling v_r on the next processor in *M*. After attempting the scheduling operation on each processor in *M*, the

algorithm greedily selects the processor that allowed the earliest completion of v_r, and permanently schedules v_r on that processor. Function *Schedule_Vertex* is used for scheduling v_r on a processor and function *Remove_Schedule* is used for reversing the scheduling of v_r. The key operations in *Schedule_Vertex* are the searches for idle time slots on the processor and the communication channels in which the vertex and incident edges can be scheduled. Before v_r is scheduled, its incident edges must be scheduled. Scheduling an edge requires the algorithm to fnd overlapping time slots on the communication channels of the source and destination processors during which the source processor's outgoing network link and the destination processor's incoming link are simultaneously idle. Vertices only need to be assigned to a single processor.

3.8 Genetic Operators

This research uses the genetic operators used by Dandass in [5]. Three different operators are used which are described as follows:

3.8.1 Selection Operator

In genetic algorithms, the fttest individuals dominate the population and can cause premature convergence. In order to reduce this, the following ftness function proposed by Grajcar [11] is used in this research:

$$f(n) = \frac{raw\_fitness(n)}{|offspring(n)| + 1} \tag{3.1}$$

where, the rank of chromosome *c* is the number of chromosomes in population $\underset{n}{\Omega}$ that produce poorer schedules than *c*. The chromosome with the largest ftness value in a random subset of size 2% to 10% of $\underset{n}{\Omega}$ is selected for reproduction. Similarly, the chromosome with the least ftness value from another random subset from $\underset{n}{\Omega}$ is selected for replacement.

3.8.2 Recombination Operators

Three different crossover operators, *standard crossover* (SX), *ordered crossover* (OX) [8] and *vector crossover* (VX), and a mutation operator are used in this research.

SX is used for recombining genes in the threshold vectors in the PosCT approaches, and the priority vectors in PriNT. The vectors are treated as a sequence of bits. A random bit position is selected in the parent chrosomoes. The child chromosome contains the sequence of bits prior to the crossover bit from one of the parent, and the sequence following the crossover bit from the other parent.

The OX operator recombines the genes in the positional vertex and edge vectors in the PosNT and PosCT representation schemes. A random crossover point is selected, similar to that as in SX. The genes prior to the crossover point are copied from one of the parent to the child. The remaining part of the child chromosome contains the remaining genes in the order in which they appear in the other parent chromosome.

In VX, the child chromosome receives a copy of the entire vertex vector from one parent, and a copy of the entire edge vector from the other parent.

The mutation operator randomly swaps the location of a pair of genes within the vertex, edge, threshold, and priority vectors.

3.9 PDF Operators

In deterministic scheduling, since the execution time of a task is fx ed, the start time and the end time of a task are fx ed values. This research deals with stochastic scheduling where the tasks have stochastic execution times, and hence the start time and end time requirements need to be specifed as PDFs. Figure 3.3 depicts a schedule in Gantt-chart form for the DAG in Figure 1.3. In this fgure, the shaded rectangular regions indicate the times when the vertices and edges may potentially be executing. For example, $v_{\underset{n}{0}}$ begins executing on processor $p_{\underset{n}{0}}$ at time instance 1, and completes at the end of time instances 4, 5, or 6 with a probability of 1/3 each. Similarly, edge $(v_{\underset{n}{0}}, v_{\underset{n}{3}})$ begins execution immediately after $v_{\underset{n}{0}}$ completes at time instances 5, 6, or 7 with a probability of 1/3 each. The edge completes execution at time instances 5, 6, 7, 8, or 9 with probabilities 1/9, 2/9, 3/9, 2/9, and 1/9, respectively.

In deterministic scheduling, the end time is calculated by summing the execution time requirement with the starting time. In stochastic scheduling, the summation is replaced by convolution; the start time and the execution time PDFs of a task are convoluted in order to obtain the PDF of the end time of that task. Convolution of discrete PDFs $s(x)$ and $w(x)$ is defned as follows:

$$f_{X_n}(X) = \sum_{t=l_{s_n}}^{u_{s_n}} s_n(t) \cdot w_n(X - t + 1) \tag{3.2}$$

where $[l_{s_n}, u_{s_n}]$ and $[l_{w_n}, u_{w_n}]$ are the intervals over which $s(x)$ and $w(x)$ are non-zero, respectively, and $X \in [l_{s_n}+l_{w_n}-1, u_{s_n}+u_{w_n}-1]$.

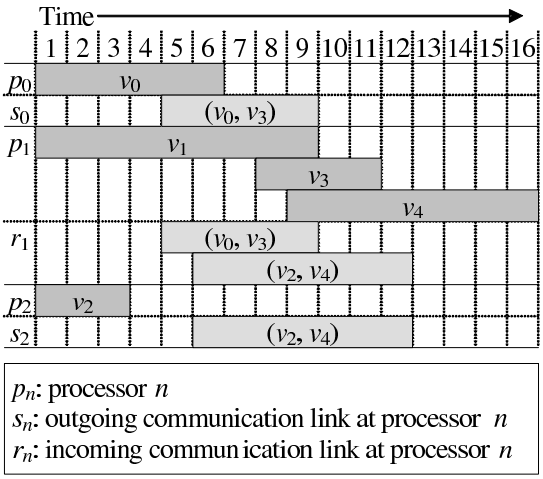


Figure 3.3

Stochastic Schedule for the DAG in Figure 1.3 [5]

A ready vertex, v_{r_n}, may need to be scheduled on a processor after a previously scheduled vertex, v_n, has completed and a previously scheduled incident edge, e_{r_n}, on the ready vertex v_{r_n} has also completed. For example, in Figure 3.3, v_{3_n} can start executing only after vertex v_{1_n} and edge (v_{0_n}, v_{3_n}) complete. Note that edge (v_{1_n}, v_{3_n}) has an effective weight of 0 because v_{1_n} and v_{3_n} are scheduled on the same processor, and therefore, does not factor in v_{3_n}'s start time computation. In such cases, the start time PDF for the vertex is determined

from the maximum of the completion PDFs of the two preceding tasks. The maximum of two independent PDFs π_1 and π_2 defned over intervals $[l_1, u_1]$ and $[l_2, u_2]$, respectively, is computed as follows [6]: $\forall x \in [max(l_1, l_2), max(u_1, u_2)]$,

$$\pi_{max(\pi 1,\pi 2)}(x) = \pi_1(x)\pi_2(x) + \pi_1(x)\Pi_2(x-1) + \Pi_1(x-1)\pi_2(x) \tag{3.3}$$

where Π_1 and Π_2 are the cumulative distribution functions (CDFs) corresponding to the PDFs π_1 and π_2 respectively. The CDF, $\Pi(x)$ associated with a PDF, $\pi(x)$, is derived as follows:

$$\Pi(X) = \sum_{k=0}^{X} \pi(X), \forall X \in [0, \infty] \tag{3.4}$$

To schedule an edge, a common time slot has to be found in the send and receive channels at the source and destination processors that starts after the source vertex has completed. Hence, the start time of an edge will be computed from the maximum of the end time PDFs of the source vertex, the previously scheduled edge (if any) in the communication channel of the source processor, and the previously scheduled edge (if any) in the communication channel of the destination processor. For example, suppose that edge e is to be scheduled after vertex v completes, and that the source communication channel has edge e_{src} scheduled to complete after e can begin executing. Similarly, assume that the destination communication channel has edge e_{st} scheduled to complete after e can begin executing. In this case, the starting PDF of e can be computed from the maximum of the end time PDFs of v, e_{src}, e_{st}The maximum of three independent PDFs π_1, π_2 and π_3 is computed as follows [6]:

$$\pi_{max(\pi 1,\pi 2,\pi 3)}(x) = max[max(\pi_1, \pi_2), \pi_3] \tag{3.5}$$

Equation 3.2 and Equation 3.3 only apply to independent PDFs. Situations with dependent PDFs must be handled separately. In the example above, if the previously scheduled edges in the source and destination communication channels are the same edge (*i.e.*, $e_{src_n} = e_{dst_n}$), then the starting time PDF of e must be computed from the maximum of the end time PDFs of v and e_{src_n} only; taking the maximum of v, e_{src_n}, and e_{dst_n} in this situation will be incorrect.

The end time of the entire schedule is given by the maximum of the PDFs of the end time of the terminal tasks, which are tasks that do not have any other tasks scheduled after them.

3.10 Thresholds

Every processor and communication channel has a list of *idle* time-slots. An idle time-slot is an interval of time in which no tasks have been scheduled as yet. A task is allocated to a slot on a processor if the slot begins at, or before the task can begin execution. The ending time of a slot is given by the minimum starting time of all tasks previously allocated to the same processor with starting times greater than the starting time of the slot. PDF operators developed in [5] have been used to fnd the minimum and maximum of sets of PDFs to fnd the ending time of a slot.

There are occasions when a ready task, T_{R_n}'s, ready time is less than or equal to the idle slot's start time, however, the idle slot, S_{I_n}, is not suffciently large in order to allow T_{R_n} to complete (*i.e.* T_{R_n} can be assigned to begin within S_{I_n} but the previously scheduled task, T_{S_n},

that appears at the end of $S_{\underset{n}{I}}$, is scheduled to begin before $T_{\underset{n}{R}}$ will complete if scheduled to start in $S_{\underset{n}{I}}$). In the PosNT approach (*i.e.*, the no threshold approach initially used by [6]), $T_{\underset{n}{R}}$ is inserted into $S_{\underset{n}{I}}$ only if $T_{\underset{n}{R}}$ does not overlap $T_{\underset{n}{S}}$. If there is overlap then $T_{\underset{n}{R}}$ is scheduled in another interval that occurs after $S_{\underset{n}{I}}$. However, inspection of the schedules produced by PosNT revealed several instances in which delaying $T_{\underset{n}{S}}$ by a small amount of time would have resulted in reduced schedule lengths. This is because allocating $T_{\underset{n}{R}}$ in a later time slot resulted in a signifcant delay of tasks dependant on the completion of $T_{\underset{n}{R}}$, as compared with the delay incurred by $T_{\underset{n}{S}}$ and its dependent tasks if $T_{\underset{n}{R}}$ was allowed to complete before $T_{\underset{n}{S}}$ began. Figure 3.4 depicts the schedule for the DAG in Figure 1.3 in which edge ($v_{\underset{n}{2}}$, $v_{\underset{n}{4}}$) is allowed to execute before ($v_{\underset{n}{0}}$, $v_{\underset{n}{3}}$). This results in a schedule that is shorter than the schedule shown in Figure 3.3 by one time unit. The schedule in Figure 3.3 was constructed using the PosNT approach.

However, arbitrarily delaying tasks can potentially perturb the scheduling power of the GLS algorithm. Therefore, previously allocated tasks should only be delayed by relatively small amounts. In order to determine when $T_{\underset{n}{S}}$ should be delayed, $T_{\underset{n}{R}}$ is tentatively assigned to begin in $S_{\underset{n}{I}}$ and the completion time of $T_{\underset{n}{R}}$ is computed. The two variations of PosCT, *viz.* PosCT-Fixed and PosCT-Variable, calculate the overlap between $T_{\underset{n}{S}}$ and $T_{\underset{n}{R}}$ in slightly different ways. This overlap is then compared to the threshold gene value of $T_{\underset{n}{S}}$ to decide if $T_{\underset{n}{R}}$ can be scheduled in $S_{\underset{n}{I}}$ or not.

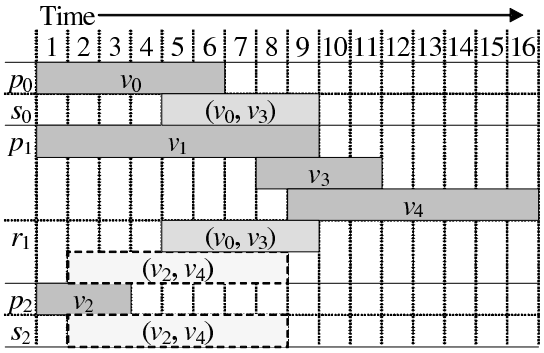


Figure 3.4

Shorter Schedule Produced by PosCT

In the PosCT-Fixed approach, the amount of overlap between tasks T_{R_n} and T_{S_n}, δ_n, is calculated as follows:

$$\delta(T_{S_n}, T_{R_n}) = F(T_{R_n}) - S(T_{S_n}), \tag{3.6}$$

where $F(T_{R_n})$ is the end time of T_{R_n} and $S(T_{S_n})$ is the start time of T_{S_n}. If $\delta_n \leq 0$, it implies that the two tasks do not overlap, and task T_{R_n} can be scheduled in S_{I_n}. If there is an overlap, meaning the difference in the end and start time is positive non-zero, then a decision has to be made as to whether T_{R_n} can be scheduled in S_{I_n} and T_{S_n} has to be delayed, or T_{R_n} has to be allocated at a later time. For this, the overlap ratio, ω_n, is calculated as follows:

$$\omega_n(T_{S_n}, T_{R_n}) = \frac{\delta(T_{S_n}, T_{R_n})}{t(T_{S_n})}, \tag{3.7}$$

where $t(T_{\underset{n}{S}})$ is the expected value of the weight of task $T_{\underset{n}{S}}$. The overlap ratio is a ratio of the amount of overlap to the weight of $T_{\underset{n}{S}}$. This overlap ratio is compared with the overlap threshold gene value for $T_{\underset{n}{S}}$, as obtained from its corresponding position in the overlap threshold vector in the chromosome. If $\underset{n}{\omega}(T_{\underset{n}{S}}, T_{\underset{n}{R}})$ is less than or equal to the threshold value, then $T_{\underset{n}{R}}$ is scheduled in $S_{\underset{n}{I}}$ and $T_{\underset{n}{S}}$ is delayed; otherwise $T_{\underset{n}{R}}$ is not scheduled in $S_{\underset{n}{I}}$ and the algorithm searches for the next available slot. $\underset{n}{\omega}(T_{\underset{n}{S}}, T_{\underset{n}{R}})$ represents the overlap between $T_{\underset{n}{S}}$ and $T_{\underset{n}{R}}$ as a percentage of the total weight of $T_{\underset{n}{S}}$. The threshold specifes the maximum allowable value of overlap as a percentage of the weight of the corresponding task. Only if the overlap percentage is less than the threshold will $T_{\underset{n}{R}}$ be scheduled in $S_{\underset{n}{I}}$. If the overlap percentage is more than the threshold value, it implies that the overlap is signifcant in comparison to the weight of $T_{\underset{n}{S}}$, and hence a later starting time slot for scheduling $T_{\underset{n}{R}}$ should be utilized.

The PosCT-Variable approach calculates the probability that task $T_{\underset{n}{R}}$ fnishes before $T_{\underset{n}{S}}$ if $T_{\underset{n}{R}}$ is scheduled in slot $S_{\underset{n}{I}}$. This probability is compared with the overlap threshold value of $T_{\underset{n}{S}}$. The threshold value in this case signifes the minimum probability with which $T_{\underset{n}{R}}$ should fnish before $T_{\underset{n}{S}}$ starts executing. If the probability of completion is at least as much as the threshold, $T_{\underset{n}{S}}$ is delayed and $T_{\underset{n}{R}}$ is scheduled in $S_{\underset{n}{I}}$. If the probability of completion is less than the threshold, then a new slot is searched to schedule $T_{\underset{n}{R}}$. This technique is adopted from [5].

Note that a delay in the start time of T_{S_n} also delays the start times of any previously scheduled tasks that depend on the completion of T_{S_n}. Therefore, a delay in the start time of T_{S_n} causes a "ripple" effect of delays in the partial schedule.

3.11 Parallel Implementation

A parallel implementation of the GLS algorithm based on the *synchronous connected island model* [21] was used to evaluate the scheduling approach described above. In the island model, the GA population is distributed between the parallel processes. Each process evolves its share of the global population independently from the other processes. After a predetermined number of iterations, all processes synchronously broadcast the fttest chromosomes in their local populations to all other processes.

Varying the GA control parameters in each of the *N* parallel GLS processes can lead to increased genetic diversity [12]. Therefore, the probabilities with which mutation, VX, and OX operators are selected in a parallel GLS process, $_i \in \{0, 1, ..., N - 1\}$ are given as:

$$\pi_{m}(n_i) = 0.05 + 0.05 \times _i/N, \tag{3.8}$$

$$\pi_{vx}(n_i) = 0.20 + 0.10 \times _i/N, and \tag{3.9}$$

$$\pi_{ox}(n_i) = 0.75 - 0.15 \times _i/N \tag{3.10}$$

respectively. This implies that the mutation rate ranges between 0.05 and 0.10, the VX rate ranges between 0.20 and 0.30, and the OX rate varies between 0.75 and 0.60. These ranges appear to work well for the sample scheduling problems solved in the research effort.

Evolving different populations separately from each other enhances “exploration” (and prevents premature convergence on a local optimum.) Periodically exchanging the fttest chromosomes introduces “exploitation” of good genetic information that can lead to higher-quality solutions than those that can be found by the individual populations alone.

CHAPTER IV

EXPERIMENT DESIGN AND METRICS

This chapter desrcribes the experiments that were conducted as part of this thesis, presents experimental data, and provides an analysis of the results.

4.1 Directed Acyclic Graphs

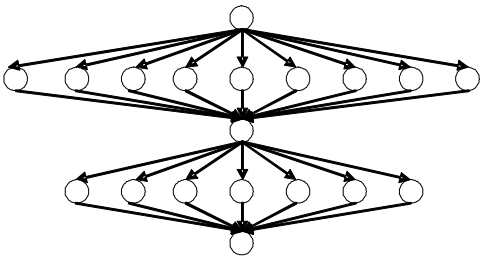
In this research, simulated DAGs were used for evaluating the three different GLS representation schemes. DAGs represent tasks of real-time applications, task interactions and precedence relationsips. Experimental DAGs differed from each other with regards to the structure, size, weight distributions, and computation vs. communication requirements.

4.1.1 DAG Structure

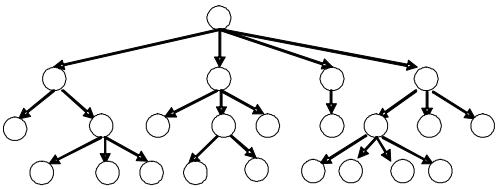
The characteristic structure of the DAGs were one of the following:

- *Hierarchical Fork-Join* (HFJ)
- *Mean Value Analysis* (MVA)
- *Out Tree* (OUT)
- *Random* (RND)
- *Simple Fork-Join* (SFJ)

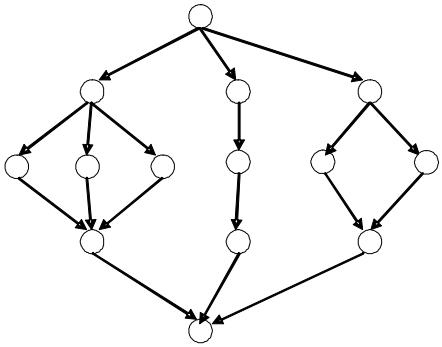
The structures are as shown in Figure 4.1.



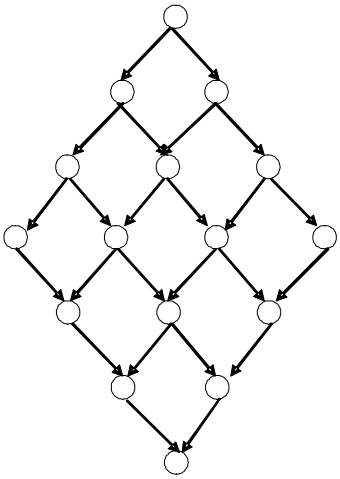
(a) Simple Fork-Join (SFJ)



(b) Out Tree (OUT)



(c) Hierarchical Fork-Join (HFJ)



(d) Mean Value Analysis (MVA)

Figure 4.1

Structure of Experimental DAGs [6]

The fork-join type of DAGs (namely HFJ and SFJ) represent trivially parallel applications. In these types of applications, a single task completes an operation, which gives rise to a number of tasks that can be executed in parallel; and the results from these parallel tasks are gathered by another single task. The MVA structure is as shown in Figure 4.1 (d). It represents a parallel application with several branching and joining fo ws. The OUT structure is similar to a tree structure with branches that keep splitting as we go down the tree hierarchy. The random DAG structure, RND, represents an application with no predetermined branching pattern, and is signifcantly irregular in structure than the other DAGs.

4.1.2 DAG Size

Each of the DAGs are constructed in such a way that the total number of vertices and edges in each DAG are in the ranges [290, 325], [390, 425], and [490, 525]. This number represents the total number of tasks in the application. These DAG sizes were selected in order to provide an additional degree of variability in the DAGs, and in order to investigate the effectiveness of each GLS representation for DAGs of different sizes.

4.1.3 Weight Distributions

Each of the tasks in a DAG have a variable weight value. These values are based on one of three distributions, namely, *beta*, *exponential*, and *random*. The process of selecting the weight probability distributions and assigning them to a task is adopted from [5]. Beta distributions are defned only over a fnite interval, which makes them suitable for modeling real-time applications since real-time tasks need to complete within a narrow range of

time. The beta and exponential distributions have relatively smooth curves as compared to the randomized PDFs which have irregularities in probability values.

4.1.4 Computation to Communication Ratio

The *computation to communication ratio* (CCR) for an application is the ratio of the total time spent performing computation tasks to the total amount of time spent performing communication operations. For a DAG representing an application, the CCR is defned as the ratio of the average vertex weight to the average edge weight. The CCR of DAG, *G*, is computed as follows:

$$CCR(G) = \frac{\frac{\sum_{v_i \in V} E[w(v_i)]}{|V|}}{\frac{\sum_{i \in E} E[w(e_i)]}{|E|}} \tag{4.1}$$

The DAGs in this research have CCR ratios as follows:

1. CCR = 0.5 represent applications whose computation tasks complete in 50% of the time that it takes for the communication tasks to complete.
2. CCR = 0.6 represent applications whose computation tasks complete in 60% of the time that it takes for the communication tasks to complete.
3. CCR = 1.0 represent applications whose computation tasks complete in the same amount of time as it takes for the communication tasks to complete.
4. CCR = 1.5 represent applications whose computation tasks take 50% more time to complete than it takes for the communication tasks to complete.
5. CCR = 2.0 represent applications whose computation tasks take twice the amount of time to complete than it takes for the communication tasks to complete.

4.2 DAG Instances

The combination of DAG structures, sizes, weights, and CCR options resulted in a total of 225 DAGs as summarized in Table 4.1.

Table 4.1

DAG Structure Combinations

Structure	Number of Size Options	Number of CCR Options	Number of Weight Distribution Options	Total DAGs
HFJ	3	5	3	45
MVA	3	5	3	45
OUT	3	5	3	45
RND	3	5	3	45
SFJ	3	5	3	45
			TOTAL	225

Schedules were created for all of the 225 DAGs with PosNT, PosCT-Fixed, and PriNT. The PosCT-Variable approach involves PDF manipulations which are computationally expensive. The amount of time to construct a schedule with the PosCT-Variable is prohibitively large. Hence, for testing the PosCT-Variable approach, only 25 out of the 225 DAGs were chosen. The 25 DAGs were randomly selected; there are 5 instances of each of the 5 DAG structures. Schedules constructed for these 25 DAGs by PosCT-Variable were compared to the schedules constructed for the same 25 DAGs with each of the PosNT, PosCT-Fixed, and PriNT approaches.

4.3 Metrics for Experimental Analysis

This section describes the various metrics used for comparing the schedules created by the different representation schemes. The metrics used in this research are as discussed by Dandass in [5]. Since this research deals with schedule construction, the length of the schedule is an important factor for comparing the representation schemes. Improvements in the result in terms of schedule lengths as compared to the List Scheuling technique are reported.

4.3.1 Schedule Length

In this research, the length of the schedule is measured as the amount of time that is required for all the tasks in the application to execute so that the end-to-end deadline is met with a probability of 100%. The length is calculated from the maximum of the end time PDFs of the terminal vertices. Let V_T be the set of terminal vertices in the DAG. Let f_i be the end time PDF of vertex $v_i \in V_T$. The end time PDF, f_{end} that is used for obtaining the length of the schedule is given by the following expression:

$$f_{end} = max\{f_i : v_i \in V_T\} \tag{4.2}$$

The PDF f_{end} is defned over the interval $[l_{end}, u_{end}]$. u_{end} is the maximum schedule length when the probability of meeting end-to-end deadlines is 100%. This value will be used as the schedule length metric for comparing the representation schemes.

4.3.2 Relative Improvement

This metric is used to compare the length of the schedules produced by each of the genetic representation schemes with each other, and with those produced by the List Scheduling technique-HLFET. The schedule length used in this comparitive analysis is the same as described above. This metric shows how better or worse each technique is as compared to the other. If L_1 and L_2 are maximum lengths of two schedules $schedule_1$ and $schedule_2$ respectively, then the relative improvement in the schedule length is given as following:

$$\lambda(schedule_1, schedule_2) = \frac{L_1 - L_2}{L_1} \tag{4.3}$$

If $schedule_2$ is worse than $schedule_1$, then $L_2 > L_1$ and λ will be negative.

4.3.3 Schedule Compression

The *schedule compression* metric, adopted from [5], gives the relative reduction in the width of the end time PDF when the probability of meeting end-to-end deadlines being less than 100% is acceptable. Let f_{nd} be the end time PDF, defned over the interval [l_{nd} u_{nd}]. As stated above, if the probability of meeting end-to-end deadlines has to be 100%, the maximum schedule length will be u_{nd} Allocation of any more time units will result in a waste of resources. Also, if less than l_{nd} time units are allocated, then the probability of meeting deadlines is 0. Therefore, the schedule length can be reduced in the range [l_{nd} u_{nd}]. However, it should be noted that although a 100% reduction is possible, it would require reducing the probability of meeting deadlines to 0. This research deals with soft real-time systems, in which occasional deadline misses are tolerable, but missing all

deadlines is not. Therefore, a 100% reduction will be inappropriate. Let $L(x)$ be the length of the schedule that results when the required probability of meeting end-to-end deadlines is x%. The schedule compression metric is computed as follows:

$$\zeta(x) = \frac{L(1.0) - L(x)}{L(1.0) - L(0.0)} = \frac{n_{nd} - L(x)}{n_{nd} - n_{end}}, \tag{4.4}$$

where $0 \leq x \leq 1$.

4.4 Platform Description

The experiments involved constructing schedules for each of the 225 DAGs using the HLFET List Scheduling technique and each of PosNT, PosCT-Fixed, and PriNT. The second set of experiments constructed schedules for the 25 DAGs by the PosCT-Variable scheme. The DAGs were evaluated on the basis of the metrics as mentioned above.

Schedules were constructed on a cluster with eight compute nodes and one head node. Each compute node had two Intel Xeon processors, operating at 3.06GHz with 2GB of DDR RAM and 80GB SATA HDD. The head node was also dual-processor with 4GB DDR RAM and 120GB SATA HDD. Interconnect technology was 100Mbps Ethernet, capable of Gigabit Ethernet, not used in this research. The cluster used LAM-MPI Version 7.0.6 with Linux Kernel 2.6.5, based on Fedora Core 2.

CHAPTER V

EXPERIMENTAL RESULTS AND ANALYSIS

the This chapter presents the results and analyses of the experiments performed in this research. In the frst series of experiments, schedules were created for the 225 experimental DAGs by each of PosNT, PosCT-Fixed, and PriNT. In the second series of experiments, schedules were created by the PosCT-Variable approach for 25 randomly selected DAGs. In each experiment, each of the 8 processors maintained independent populations of 1,000 chromosomes and computed 24,000 iterations. The processes exchanged the fttest chromosomes with each other at every 1,000$^{th}_{n}$ iteration, beginning with the 12,000th iteration and at every 100$^{th}_{n}$ iteration after the 23,000th iteration. The schedule length was computed as described in the previous chapter. The schedule lengths obtained by HLFET and those by the genetic representation schemes were compared with each other.

5.1 Schedule Lengths

The pairwise performance of the fx ed estimate genetic representation schemes, namely PosNT, PosCT-Fixed, and PriNT, is shown in Table 5.1. The results are grouped according to the structure of DAGs. Each row shows the number of DAGs, out of 45 DAGs of each of the f ve types, that each approach produced better results than the other in the pairwise

comparison. The last row shows the total number of DAGs for each approach out of the 225 DAGs tested.

Table 5.1

Pairwise Comparison of PosNT, PosCT-Fixed, and PriNT

Structure	PosNT	PriNT	PosNT	PosCT-Fixed	PriNT	PosCT-Fixed
HFJ	27	18	18	27	15	30
MVA	23	22	17	28	18	27
RND	20	25	10	35	11	34
OUT	22	23	20	25	22	23
SFJ	21	24	10	35	6	39
All	113	112	75	150	72	153

For example, the frst row contains the breakup of the 45 HFJ structured DAGs as follows:

- Of the 45 HFJ DAGs, scheduling with the PosNT approach produced shorter schedule lengths for 27 DAGs as compared to scheduling with PriNT. The remaining 18 DAGs had shorter schedule lengths with the PriNT approach as compared to PosNT.
- Of the 45 HFJ DAGs, scheduling with the PosNT-Fixed approach produced shorter schedule lengths for 18 DAGs as compared to PosCT. The remaining 27 DAGs had shorter schedule lengths with the PosCT-Fixed approach as compared to PosNT.
- Of the 45 HFJ DAGs, scheduling with the PriNT approach produced shorter schedule lengths for 15 DAGs as compared to PosCT. The remaining 30 DAGs had shorter schedule lengths with the PosCT-Fixed approach as compared to PriNT.

The last row of the above table gives the total number of the DAGs out of the 225 DAGs for which each approach created better schedules as compared to the other. For

example, out of 225 DAGs, the PosNT approach created schedules with shorter schedule lengths for 113 DAGs as compared to 112 as created by PriNT.

The results summarized in Table 5.1 clearly show that for the given experimental DAGs, PosNT and PriNT approaches have almost similar performance - 113 DAGs with PosNT and 112 DAGs with PriNT. PosCT-Fixed clearly outperformed both PosNT and PriNT by a ratio of 2:1. PosCT-Fixed produced better results for 150 DAGs out of 225 DAGs as compared to 75 by PosNT, and 153 DAGs out of 225 DAGs as compared to 72 by PriNT. All the three genetic representation schemes outperformed the HLFET LS scheduling technique.

Table 5.2

Comparison of PosNT, PosCT-Fixed, and PriNT

Structure	PosNT	PriNT	PosCT-Fixed	Total
HFJ	16	7	22	45
MVA	10	16	19	45
RND	5	9	31	45
OUT	11	12	22	45
SFJ	9	2	34	45
All	51	46	128	225

Whereas Table 5.1 provides a pairwise comparison of each of the three genetic representation schemes, Table 5.2 shows the performance of the three schemes together. It shows the number of DAGs of each structure that each of the approach produced the best

result for. For example, the frst row shows that of the 45 HFJ DAGs, 16 were the best with the PosNT approach, 7 with PriNT, and 22 with PosCT-Fixed. The last row shows that of the 225 DAGs, the PosCT-Fixed approach yielded 128 DAGs with the best results, whereas the remaining 97 DAGs were almost equally divided between PosNT and PriNT. Thus, more than half the DAGs had favorable results with the PosCT-Fixed approach. This reinfores the fact that the performance of PosCT-Fixed is much better than the other two approaches and also the HLFET LS technique.

Table 5.3 shows the pairwise comparison of the PosCT-Variable approach with PosNT, PosCT-Fixed, and PriNT scheduling approaches. The table can be read in the same manner as 5.1. It can be seen that PosCT-Variable outperformed all of the fx ed-estimate approaches.

Table 5.3

Pairwise Comparison of PosCT-Variable with Fixed-Estimate Schemes

Structure	PosNT	PosCT-Variable	PosCT-Fixed	PosCT-Variable	PriNT	PosCT-Variable
HFJ	1	4	2	3	0	5
MVA	2	3	1	4	0	5
OUT	0	5	1	4	1	4
RND	1	4	3	2	1	4
SFJ	1	4	3	2	0	5
All	5	20	10	15	2	23

It can be observed from Table 5.4 below that out of the 25 tested DAGs, 10 DAGs had shorter schedule lengths by the PosCT-Variable approach, which is greater than those by any of the other fx ed-estimate approaches. The performance of PosCT-Fixed and PosCT-Variable seem almost equivalent. Their relative schedule improvements are presented in the following section.

Table 5.4

Comparison of PosNT, PosCT-Fixed, PriNT, and PosCT-Variable

Approach	Number of DAGs	Out of
PosNT	5	25
PosCT-Fixed	8	25
PriNT	2	25
PosCT-Variable	10	25

5.2 Relative Schedule Length Improvements

Table 5.5 shows the relative schedule length improvements of schedules created by the three fx ed-estimate genetic representation schemes over those generated by the HLFET LS technique. Not only are all the schedules created by the genetic representation schemes better than those created by HLFET, but it can be seen from Table 5.5 that the relative improvements obtained are signifcant.

Table 5.6, Table 5.7, and Table 5.8 show the pairwise relative schedule improvements of the fx ed-estimate representation schemes with respect to each other. The results are

grouped according to DAG structures. For each pair of representation schemes for a particular DAG structure, the "Average" column is the average improvement of one scheme over the other, averaged over the 45 instances of that DAG structure, whereas the "Maximum" column is the maximum improvement of one scheme over the other for the 45 instances of DAG structure. The last row gives the average of the average and maximum improvements of one scheme over the other for all the 225 DAGs.

Table 5.5

Relative Schedule Improvements of Fixed-Estimate Schemes and HLFET

Structure	PosNT over HLFET		PosCT-Fixed over HLFET		PriNT over HLFET	
	Average	Maximum	Average	Maximum	Average	Maximum
HFJ	21.85	39.20	25.78	39.05	22.06	34.56
MVA	21.38	36.38	22.33	35.71	21.41	32.56
OUT	46.06	65.74	54.90	71.64	47.22	71.00
RND	28.83	45.29	29.53	46.74	29.38	44.66
SFJ	18.10	36.46	24.65	40.54	18.37	35.84
Average	27.24	44.61	31.44	46.73	27.69	43.72

From Table 5.6 and Table 5.7, it can be seen that the average relative improvements for PosCT-Fixed over PosNT, and PosCT-Fixed over PriNT for the 225 DAGs are 6.12% and 5.21% respectively; and the average maximum relative improvement for PosCT-Fixed over PosNT, and PosCT-Fixed over PriNT are 23.95% and 24.98% respectively.

Thus, not only has PosCT-Fixed outperformed PosNT and PriNT in terms of producing the largest number of DAGs with better schedules, but the improvement in schedule

lengths obtained over the other schemes is signifcant too. Table 5.8 shows that the relative improvements of PosNT over PriNT and vice versa are almost similar.

Table 5.6

Relative Schedule Improvements of PosCT-Fixed and PosNT Schemes

Structure	PosCT-Fixed over PosNT		PosNT over PosCT-Fixed	
	Average	Maximum	Average	Maximum
HFJ	4.99	15.16	-5.55	8.49
MVA	1.13	8.83	-1.22	3.05
OUT	15.93	41.15	-21.09	-0.13
RND	0.99	15.98	-1.20	6.43
SFJ	7.56	38.65	-10.10	5.64
Average	6.12	23.95	-7.83	4.70

Table 5.9 shows the pairwise relative schedule improvements of the two variations of the PosCT scheme. This table can be read in the same manner as Table 5.6. It was earlier established that PosCT-Variable performed better then PosCT-Fixed, in terms of the number of DAGs with shorter schedules. Table 5.9 compares the two schemes to evaluate the difference in the schedule lengths created by the two approaches. It can be seen that the maximum schedule length improvement of PosCT-Variable over PosCT-Fixed obtained is 4.07%. Although PosCT-Variable produced the maximum number of best schedules as compared to the fx ed-estimate approaches, the quality of schedules is not much better than that of PosCT-Fixed over the other two fx ed-estimate approaches.

Table 5.7

Relative Schedule Improvements of PosCT-Fixed and PriNT Schemes

Structure	PosCT-Fixed over PriNT		PriNT over PosCT-Fixed	
	Average	Maximum	Average	Maximum
HFJ	4.75	15.22	-5.29	2.16
MVA	1.08	23.29	-1.29	2.71
OUT	12.91	38.90	-18.36	11.87
RND	0.18	8.72	-0.27	7.10
SFJ	7.13	38.78	-9.41	1.88
Average	5.21	24.98	-6.93	5.14

Table 5.8

Relative Schedule Improvements of PosNT and PriNT Schemes

Structure	PosNT over PriNT		PriNT over PosNT	
	Average	Maximum	Average	Maximum
HFJ	-0.27	7.09	0.23	4.63
MVA	-0.10	24.09	-0.13	7.64
OUT	-3.21	13.81	2.62	16.33
RND	-0.95	4.07	0.79	18.30
SFJ	-0.85	31.42	0.21	20.77
Average	-1.08	16.10	0.74	13.53

Table 5.9

Relative Schedule Improvements of the PosCT Schemes

Structure	PosCT-Variable over PosCT-Fixed		PosCT-Fixed over PosCT-Variable	
	Average	Maximum	Average	Maximum
HFJ	0.52	1.67	-0.54	0.50
MVA	0.98	3.76	-1.12	5.60
OUT	4.48	8.23	-4.79	0.68
RND	-0.34	4.48	0.22	3.49
SFJ	-0.01	2.22	-0.01	2.04
Average	1.13	4.07	-1.25	2.46

These results imply that if a single genetic representation scheme has to be chosen, given the constraints of time and computational resources, PosCT-Fixed is the clear choice. As far as PosCT-Variable is concerned, it does create the best schedules in terms of short schedule lengths; but it requires an investment of a prohibitively large amount of time to construct a schedule which is only marginally better than PosCT-Fixed.

From the results, considering only the fx ed-estimate approaches, it can be observed that of the 50% DAGs that had shorter schedule lengths with PosNT and PriNT, both of these approaches proved almost equivalent in performance.

Moreover, the maximum relative schedule length improvement of PosNT over PriNT and vice versa are very similar. Thus, if there are no constraints on time and computational resources, then all the three fx ed-estimate representation schemes should be used to obtain the best schedule with the shortest schedule length.

5.3 Compression

For the compression metric, 4 DAGs of each DAG structure, each of different sizes and weight distributions were selected, and schedules were created using the PosCT-Fixed scheme. Schedule lengths were found out for probabilities of meeting end-to-end deadlines being 100%, and for values ranging from 99.9999999% to 70%. Table 5.10 shows the average compression obtained for each type of DAG structure.

The graph in Figure 5.1 is plotted from the data in Table 5.10. It can be seen from Figure 5.1 that for all types of DAGs, a reduction of the probability to meet end-to-end deadlines to 70% results in a schedule compression of about 70% for all the 5 types of DAGs. As the probability of meeting end-to-end deadlines is increased, the amount of compression obtained is reduced till 0 compression is obtained when the probability is 100%.

The scheduling technique developed in this research are meant for soft real-time applications where occasional deadlines misses are tolerable. Figure 5.1 shows that if a less than 100% probability of meeting end-to-end deadlines is acceptable, then signifcant reductions in schedule length can be obtained.

Table 5.10

Schedule Compression Grouped by DAG Structure

Probability of meeting end-to-end deadline	HFJ	MVA	OUT	RND	SFJ	All
0.7	68.76	68.76	69.38	69.38	69.27	69.76
0.8	68.05	68.05	68.66	68.58	68.43	68.81
0.95	66.24	66.21	66.73	66.43	66.21	66.35
0.96	65.99	65.96	66.44	66.12	65.88	65.96
0.97	65.7	65.66	66.16	65.79	65.53	65.6
0.98	65.29	65.25	65.71	65.32	65.05	65.02
0.99	64.63	64.58	65.00	64.53	64.23	64.14
0.999	62.71	62.61	62.93	62.28	61.93	61.58
0.9999	61.1	60.98	61.19	60.36	59.97	59.45
0.99999	59.69	59.53	59.66	58.68	58.24	57.53
0.999999	58.41	58.21	58.24	57.12	56.65	55.84
0.9999999	57.23	57.01	56.97	55.74	55.24	54.31
0.99999999	56.13	55.89	55.78	54.43	53.92	52.83
0.999999999	55.09	54.83	54.67	53.21	52.66	51.48
1	0	0	0	0	0	0

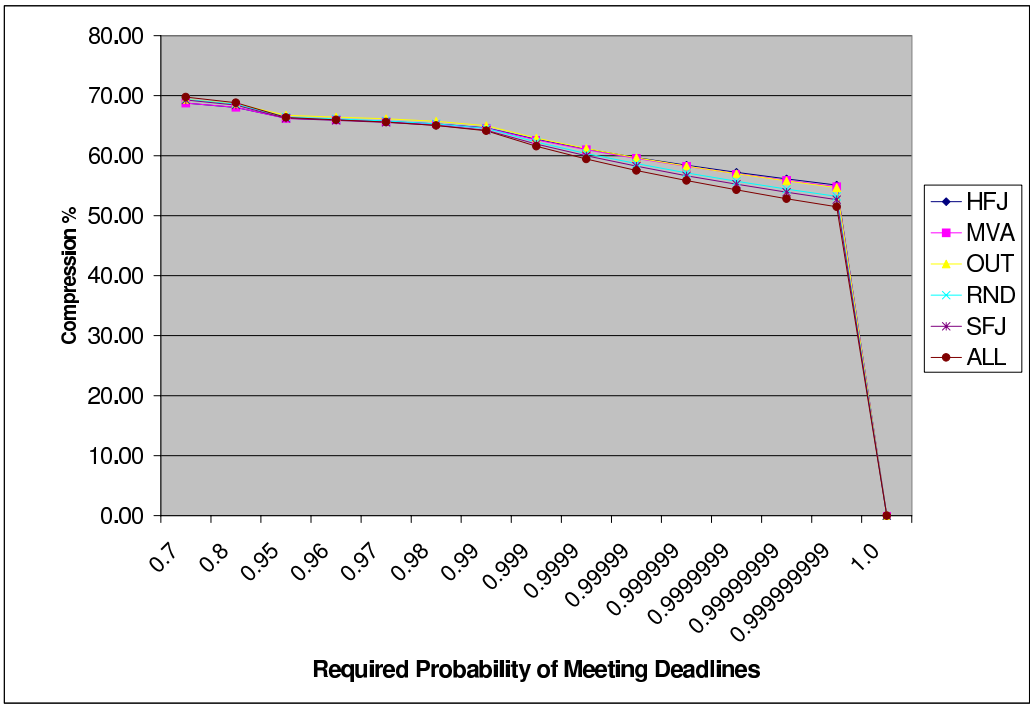


Figure 5.1

Compression grouped by DAG Structues

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the contributions and results of this research and presents potential extensions to the research.

6.1 Contributions

This research furthers the contributions made by Dandass in [5]. It generalizes and extends the traditional LS and GLS approaches for stochastic scheduling. Two new genetic representation schemes were developed in this research for constructing non-preemptive schedules for soft real-time parallel applications. The application was represented as a DAG with each vertex and edge representing computation and communication operations respectively. Each task had variable weights that refect the uncertainty in the time to complete these tasks. The weights are modeled as independent random variables, and are expressed in the form of probability distribution functions. The problem is to create stochastic schedules for the DAGs so as to minimize the schedule length.

This research investigated three different genetic representation schemes that were used for prioritizing ready tasks and constructing a schedule. 225 DAGs with varying structure, size, weight distribution, and computation to communication ratio were con-

structed. Schedules were created for each of them with the HLFET LS technique and the three genetic representation schemes.

The PosCT approach developed in this research was similar to PosNT, developed by Dandass in [5]; it has positional vector and edge genes that serve as the prioritization criteria. In addition to this, there is a third vector of genes that contains overlap threshold values for each vertex and edge in the DAG. After selecting a task to be scheduled, its overlap threshold value is used to determine when to delay the execution of certain previously scheduled tasks in order to allow that task to be scheduled. The genetic algorithm operates on the threshold gene vector alongwith the positional vertex and edge gene vectors to determine the best threshold for a task.

In the PriNT approach that was developed in this research, the genes in a chromosome directly encode the priority of vertices and edges. There is a single priority value for each task in the DAG, and the task with the highest priority gene is selected for scheduling.

The schedules were analyzed and compared on the basis of schedule lengths. It was observed that all the three genetic representation schemes outperformed the LS technique for each of the 225 DAGs tested. Of the three genetic schemes, the performance of PosCT was the best as compared to PosNT and PriNT. Using PosCT resulted in shorter schedules for more than 50% of the experimental DAGs as compared to the schedules produced using the other schemes.

It can be seen from the compression graph that decreasing the required probability of meeting end-to-end deadlines results in an increased amount of reduction in schedule

length. With the probability of meeting end-to-end deadlines reduced to 70%, a reduction of almost 70% in schedule length is obtained. Soft real-time systems can tolerate occasional deadlines misses, and this itself can be used to create schedules with shorter schedule lengths than if deadlines are to be met 100% of the time. This is a tradeoff between the quality-of-service and improved performance.

A paper based on the fndings of this research will appear in the proceedings of the ACM Genetic and Evolutionary Computation Science Conference (GECCO-2006).

6.2 Future Work

An immediate extension is to measure and compare the timing requirements for schedule creation for each genetic scheme. It is expected that the HLFET LS technique will require the least amount of time, since genetic algorithms, by nature, are exploratory. However, since this research presents an offine scheduling technique, the increased program execution time of the genetic representation schemes can be traded off for their ability to generate better quality schedules than LS. It will be interesting to compare the timing requirements of the positional and priority approaches and determine if any timing advantages can be achieved of either one over the other.

The PriNT approach used in this research does not use the thresholding technique. A hybrid version of the PriNT with customized thresholds would be another prioritization scheme that can be tested for its effcac y for generating good schedules.

This research deals with DAGs that are generated for experimental purposes. The next step is to use an actual real-time application and observe the performance of the LS and genetic representation approaches for scheduling. A prospective application can be analyzing video or GPS information. Then the scheduling techniques can be implemented for a real-time system, such as the Mobile Augmented Reality system that needs video and GPS information as input.

An important extension as mentioned in [5] is that this research assumes task execution time requirement PDFs to be independent of each other. However, in practical applications, this may not always be the case. Task behavior depends on the characteristics of the data which causes tasks to be dependent on other tasks, which was disregarded in this research. To account for such inter-task dependencies, new PDF manipulation algebra is required which can be used in schedule construction.

REFERENCES

[1] L. Abeni and G. Buttazzo, "Qos Guarantee Using Probabilistic Deadlines," *Proceedings: 11th Euromicro Conference on Real-Time Systems*. IEEE, June 1999, pp. 242–249.

[2] T. L. Adam, K. M. Chandy, and J. R. A. Dickson, "Comparison of List Schedules for Parallel Processing Systems," *Communications of the ACM*. ACM, 1974, vol. 17, pp. 685–690.

[3] S. J. Beaty, "List Scheduling: Alone, with Foresight, and with Lookahead," *Proceedings: 1st International Conference on Massively Parallel Computing Systems*. IEEE, May 1994, pp. 343–347.

[4] G. C. Buttazo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, Boston, Massachusetts, 1997.

[5] Y. S. Dandass, *Stochastic Scheduling for Soft Real-Time Parallel Applications to Tradeoff Quality-of-Service for Improved Performance*, doctoral dissertation, Department of Computer Science and Engineering, Mississippi State University, Mississippi State, Mississippi, 2003.

[6] Y. S. Dandass, "Genetic List Scheduling for Soft Real-Time Parallel Applications," *Congress on Evolutionary Computation*. IEEE, June 2004, pp. 1164–1171.

[7] G. B. David Montana and S. Moore, "Using Genetic Algorithms for Complex, Real-Time Scheduling Applications," *Network Operations and Management Symposium*. IEEE, February 1998, vol. 1, pp. 245–248.

[8] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains," *Proceedings: 9th International Joint Conference on Aritifcial Intelligence*, 1985, pp. 162–164.

[9] A. Dogan and F. Ozguner, "Stochastic Scheduling of a Meta-task in Heterogeneous Distributed Computing," *International Conference: Parallel Processing Workshops*. IEEE, September 2001, pp. 369–374.

[10] M. Grajcar, "Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System," *Proceedings: 36th Design Automation Conference*. ACM, June 1999, pp. 280–285.

[11] M. Grajcar, “Strengths and Weaknesses of Genetic List Scheduling for Heterogenous Systems,” *Proceedings: International Conference on Application of Concurrency to System Design*. IEEE, June 2001, pp. 123–132.

[12] S. B. Y. J. C. Potts, T. D. Giddens, “The Development and Evolution of an Improved Genetic Algorithm Based on Migration and Artifcial Selection,” *IEEE Transactions on Systems, Man, and Cybernetics*. IEEE, 1994, vol. 24, pp. 73–86.

[13] I.-J. Jeong, G. Papavassilopoulos, and D. S. Bayard, “Task Scheduling on Spacecraft by Hybrid Genetic Algorithms,” *Proceedings: IEEE International Conference on Robotics and Automation*. IEEE, May 1999, vol. 1, pp. 441–446.

[14] G. H. Kim and C. S. G. Lee, “Genetic Reinforcement Learning Approach to the Machine Scheduling Problem,” *Proceedings: International Conference on Robotics and Automation*. IEEE, May 1995, pp. 196–201.

[15] Y.-K. Kwok and I. Ahmad, “Effcient Scheduling Algorithms of Arbitrary Task Graphs to Multiprocessors using a Parallel Genetic Algorithm,” *Journal of Parallel and Distributed Computing*. IEEE, 1997, vol. 47, pp. 58–77.

[16] J. W. S. Liu, *Real-Time Systems*, Prentice-Hall, Upper Saddle River, New Jersey, 2000.

[17] S. Mingsheng, S. Shixin, and W. Qingxian, “An Effcient Parallel Scheduling Algorithm of Dependant Task Graphs,” *Proceedings: 4th International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, August 2003, pp. 595–598.

[18] Y. Monnier, J.-P. Beauvais, and A.-M. Deplanche, “A Genetic Algorithm for Scheduling Tasks in a Real-Time Distributed System,” *Proceedings: 24th Euromicro Conference*. IEEE, August 1998, vol. 2, pp. 708–714.

[19] K. Ramamritham, J. A. Stankovic, and P.-F. Shiah, “Effcient Scheduling Algorithms for Real-Time Multiprocessor Systems,” *Transactions on Parallel and Distributed Systems*. IEEE, April 1990, vol. 1, pp. 184–194.

[20] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.-S. Liu, “Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times,” *Proceedings: Real-Time Technology and Applications Symposium*. IEEE, May 1995, pp. 164–173.

[21] D. W. V. S. Gordon, “Serial and Parallel Genetic Algorithms as Function Optimizers,” *Technical Report CS-93-114*. Colorado State University, 1993.

[22] F. Wang, K. Ramamritham, and J. A. Stankovic, "Bounds on the Performance of Heuristic Algorithms for Multiprocessor Scheduling of Hard Real-Time Tasks," *Real-Time Systems Symposium*. IEEE, December 1992, pp. 136–145.

[23] P.-C. Wang and W. Korfhage, "Process Scheduling Using Genetic Algorithms," *Proceedings: 7th IEEE Symposium on Parallel and Distributed Processing*. IEEE, October 1995, pp. 638–641.