A MIXED INTEGER LINEAR PROGRAMMING MODEL FOR REAL-TIME TASK SCHEDULING IN MULTIPROCESSOR COMPUTER SYSTEM

Samuel Adebayo Oluwadare

Department of Computer Science
Federal University of Technology
Akure, Nigeria

oluwadaresam@yahoo.com

Basil Oluwafemi Akinnuli

Department of Mechanical Engineering
Federal University of Technology
Akure, Nigeria

boakinnuli@yahoo.com

ABSTRACT

There has been an upsurge in real-time multimedia applications in recent time. On a network, the ability of an average uni-processor computer to handle such data may be limited due to the large size of such data. Also, there may be a high number of concurrent users who may want to retrieve data and the need to process them in real-time; and in continuous stream. This may lead to low quality service and deadline misses. The advent of multi-processor systems offers a more efficient way of processing multimedia data in real-time. With the development of appropriate scheduling algorithm, another challenge is the mode of assigning tasks in multi-processor systems. This calls for the use of an appropriate mathematical model that will take cognizance of the nature of variables involved. In this research work, a Mixed Integer Linear Programming Model was developed to assign tasks in a multiprocessor system. The model was used to assign tasks to multi-processor systems ranging between 5 and 10 homogenous processors. The result of the simulation runs shows that with the appropriate scheduling algorithm, a high success rate ratio and guaranteed number of deadlines met could be achieved.

Keywords: Task scheduling, multiprocessor systems, multimedia, genetic algorithms, simulation.

INTRODUCTION

The reduction in physical size and corresponding increase in storage capacity, as well as the speed of processing of computers have led to the computer being used to store, process and output a large volume of different types data. The introduction of computer networking which permits the sharing of computing resources, both hardware and software, further broadened the horizon of the use of computers. The advent of the internet has also expanded the horizon of the types of data that can be transmitted from one end to another. This ranges from simple numeric data to multimedia data, which is usually large in size. Due to this development, there has been an increase in multimedia applications as well as the number of users. The term multimedia literally means more than one medium. However, technically speaking, multimedia refers to a document containing two or more continuous media that must be played back over some time interval. Multimedia data includes audio and video clips and live web casts which may be delivered to personal computers, handheld devices such as Portable Digital Assistants and smart phones. Multimedia data is stored in the file system just like ordinary data. However, multimedia data must be accessed with specific timing requirements.

Multimedia applications are often characterized as soft real-time applications, because they require support for timely correct behaviour. However, deadline misses do not naturally lead to catastrophic consequences even though the Quality of Service degrades, perhaps making the user annoyed. For instance, video frames are generated in a fixed frequency (or period), but the size of the frames and execution times to handle these frames are not constant (Goyal, Quo & Vin, 1996). Furthermore, the degree of user interactivity is much higher in recent multimedia applications, for example, interactive distance learning, than in earlier applications, like Video-on-Demand. This trend has continued, thus making resource requirements even harder to predict.

Although, most average personal computers are capable of handling the load that such multimedia applications impose on the client system, the potentially high number of concurrent users retrieving data presents a problem. With regard to the processing of multimedia data, the uni-processor systems seem to have been stretched to their limit. The advent of multiprocessor systems seems to offer a solution to this problem. However, with a very efficient scheduling algorithm the issue of an appropriate mathematical model to be used in assigning tasks to constituent processors in a multiprocessor system remains a challenge. Some of the challenges include the nature of variables involved. For instance, the Linear Programming model could only handle integer variables. Hence, in this paper, a Mixed Integer Linear Programming (MILP) model was used in order to accommodate both integer and real-valued variables.

LITERATURE REVIEW

The problem of task scheduling had received considerable attention in the literature. The Early Deadline First (EDF) algorithm which was proposed by (Liu & Layland, 1973) always chose the task with the earliest deadline. However, it had been proved that while the algorithm was optimal in a uni-processor system, it could not consider priority nor analyze it. This caused the algorithm to fail under overloading conditions (Tanenbaum, 2007; Thai, 2002). In Lee and Yen (1994), it was proposed that an algorithm that used task laxity and criticality as system parameters be adopted. The simulation model contained a small number of tasks on a uni-processor system and did not consider system overloads. Therefore, all tasks were seen as real-time and fairness was not considered. A real-time distributed system in which the task with higher computation time was assigned to bottleneck processor and the system's worst case processing time computed was also proposed in Thai (2002). The proposed algorithm had acceptable resistance to system overload, especially when the number of processors was increased. The algorithm needed communication time between processors, and assumed tasks processing times were different in real-time. The algorithm did not consider heterogeneous tasks and fairness.

A fuzzy inference for scheduling non-preemptive periodic tasks in soft real-time multiprocessor systems was presented in Sabeghi, Naghibzadeh and Taghavi (2006). Priorities and deadlines were used as tasks parameters, and a fuzzy inference engine was used to compute each task's priority and to select the task with maximum priority to process. All tasks were assumed to be periodic and it was not clear whether the multiprocessor that was proposed was homogeneous or heterogeneous. The proposed model did not consider a task's processing time. Therefore, results are more similar to EDF and not suitable for multiprocessing systems. Also, a scheduling model and a related algorithm that were suitable for both uni-processor and multiprocessor systems was discussed in Chen, Ozturk and Kandemir (2005). The model provided a method to detect work overloading and tried to balance load with task dispatching. It is however, doubtful if the proposed model could handle multimedia data efficiently. Also, dynamic integrated scheduling of hard real-time, soft real-time and non-real-time tasks was proposed in Brandt, Banachowski, Caixue and Bisson (2003). The model could generate feasible schedules, but the model was restricted to periodic tasks and the task periods were changed dynamically when overloading occurred.

Some researchers had used Genetic Algorithms (GA) to schedule tasks. For instance, a dynamic scheduling of computer tasks using GA was proposed in Alberto et al. (1994). The scheduling algorithm which was non-preemptive hard real-time was aimed at dynamically scheduling as many tasks as possible such that each task met its execution deadline while minimizing the total delay time of all the tasks. A sequential MicroGA that used a small population size of 10 chromosomes running for 10 trials and using a rather high mutation rate with a sliding window of 10 tasks had also been developed. A parallel MicroGA model designed for parallel processors was also developed. The performance of the sequential MicroGA model and the parallel MicroGA model were compared with other algorithms, namely First-In-First-Out (FIFO) and EDF for solving similar problems. The results showed that the sequential MicroGA and the parallel MicroGA models produced superior task scheduling compared to the other algorithms tested. However, the work was limited due to the fact that it was meant to handle hard real-time tasks. Also, it used a small population size of 10 chromosomes, and only 10 trials were conducted.

Similarly, a memetic algorithm for task scheduling for multiprocessor systems was presented in Sutar, Sawant and Jadhav (2006). The memetic algorithm which was aimed at reducing the shortcomings of Genetic Algorithms combined GA with another optimization technique called Simulated Annealing (SA). SA transversed the search space by testing random mutations on an individual. A mutation that increased fitness was always accepted. The memetic algorithms allocated a set of tasks such that optimum performance was obtained. Tasks were distributed among the processors in such a way that the precedence constraints were preserved, and total execution time was minimized. It also defined an order of processing tasks that were ready to run on a given processor. The memetic algorithms represented tasks in a task graph. The task graphs were then mapped onto a multiprocessor system in a way that maintained precedence relations, and ensured that all tasks were completed in the shortest possible time. The paper also developed a coding scheme and algorithms for generating initial population of chromosomes; and genetic operators such as crossover, reproduction and mutation. Even though the memetic algorithms seemed promising in being able to mitigate the shortcomings of GA, it was not implemented. Hence, the efficiency of the algorithms could not be ascertained.

Also, in Hamzeh, Fakhraie and Lucas (2007) a soft real-time fuzzy task scheduling for multiprocessor systems was proposed. The algorithm arranged real-time periodic and non-periodic tasks in multiprocessor systems. Since most static and dynamic optimal scheduling algorithms would fail with a non-critical overload, the fuzzy approach proposed in the study was an attempt to balance task loads of processors successfully, prevent starvation and ensure fairness which would cause higher priority tasks to have higher running probability. Experimental results showed that the proposed fuzzy scheduler created feasible schedules for homogeneous and heterogeneous tasks. It also, considered tasks priorities which caused higher system utilization and lowered deadline misses. However, the model was deficient because it did not consider scheduler processing time, and it was independent of the number of system processors.

Apart from the scheduling of tasks, some researchers had also applied genetic-fuzzy methods to some real life problems. For instance, Chin and Lan (2007) used genetic-fuzzy based Generalized Dimension Exchange (GDE) to uniformly distribute the unprecedented web cluster workload. This approach provided non-trivial information and techniques which were useful for network administrators who intend to manage their network capacity.

Some researches had also been carried out in the scheduling of hard real-time tasks. An example of such research is found in Mahmood (2000) in which a hybrid scheduling algorithm for task scheduling in multiprocessor real-time systems was developed. The system was an attempt to overcome the shortcomings of pure genetic algorithm, and recorded a significant improvement in guarantee ratio of tasks that arrived in the system. However, the system was not designed to handle multimedia tasks which had both hard real-time and soft real-time components.

With regard to the scheduling of multimedia tasks, a study reported in Oluwadare (2009) developed a Hybrid Genetic Algorithms for Scheduling Multimedia Tasks in a Multiprocessor System. It employed heuristic knowledge of the problem domain to model a hybrid genetic algorithm in a multiprocessor environment. The system model was made up of the scheduler model and the task model. The scheduler model involved a centralized dynamic scheduling scheme in which all tasks would arrive at a central processor called the scheduler. The scheduler model consisted of a minimum of 5 and a maximum of 10 identical processors. In genetic algorithms, tasks are mapped (coded) on chromosomes just as genes are mapped on chromosomes in living organisms. Each chromosome in hybrid genetic algorithms in a multiprocessor system represents a particular schedule of tasks. Determining the chromosome that gives optimal schedule of tasks is a major motivation for using hybrid genetic algorithms. Therefore, the thrust of this paper is to present a Mixed-Integer Linear Programming (MILP) model for finding the optimal schedule (chromosome) for implementation in a multi-processor system.

MILP MODEL FOR TASK SCHEDULING

In traditional linear programming (LP), the goal is to maximize or minimize a linear function, subject to linear constraints. The constraints may be equalities or inequalities. The function to be maximized or minimized is called the objective function. The constraints on the other hand, may be technical (functional) constraints and non-negativity constraints. Integer programming (IP) is employed when all the variables in the objective function and the constraints can only take on integer values.

Mixed-Integer Programming (MIP) is employed when some of the variables in the model are real-valued (can take on fractional values), and some of the variables are integer-valued. The model, therefore, is described as "mixed". When the objective function and constraints are all in linear form, then it is a Mixed-Integer Linear Program (MILP). Although in common parlance, MIP is often taken to mean MILP, it should be noted that Mixed-Integer Non-linear Programs (MINLP) also exist, and are much harder to solve. The rest of this section describes the proposed MILP model.

Let I be the number of processors available for processing J jobs. The time taken to finish a job j is called the makespan. The makespan of a job j assigned to processor P_i working for a given period of time is

a_{ij}, for $I = 1, \ldots, I$, and $j = 1, \ldots, J$.

The problem is to choose an assignment of jobs to processors to minimize the total finish time (total makespan) of jobs in a schedule. An assignment is a choice of numbers.

Let $x_{ij}\ 1 \le i \le J, 1 \le j \le I$ be a binary variable as follows:

$$x_{ij} = \begin{cases} 1 \cdot if \cdot Task \cdot is \cdot assigned \cdot to \cdot processor, \\ 0 \cdot otherwise \end{cases}$$

Where X_{ij} represents the proportion of the processor i's time to be spent on job j A task is assigned to a processor. The following constraints, therefore, is introduced:

$$\sum x_{ij} \le 1, j = 1, 1 \le i \le J \qquad (1)$$

x_{ij} for $i = 1, \ldots, I$, and $j = 1, \ldots, J$, where x_{ij} represents the proportion of the processor *i's* time to be spent on job *j*. Thus,

$$\sum_{j=i}^{J} a_{ij} \leq 1 fori = 1, \ldots, I \tag{2}$$

Equation (2) reflects the fact that a processor *i* cannot spend more than 100% of its resources (time) working.

$$\sum_{i=1}^{I} x_{ij} \leq 1 fori = 1, \ldots, J \tag{3}$$

Equation (3) means that only one job can be processed at a time.

$$\text{and } x_{ij} \geq 0 \text{ for } i = 1, \ldots, I, \text{ and } j = 1, \ldots, J. \tag{4}$$

Equation (4) means that no processor can spend a negative amount of time to process a job.

$$a_{ij} \geq 0 \text{ for } i = 1, \ldots, I, \text{ and } j = 1, \ldots, J. \tag{5}$$

Equation (5) means that no job can have negative makespan. Task *j* starts between its arrival time a_i and its deadline time d_i. The start time s_i is, therefore bounded as follows:

$$a_j \leq s_j \leq d_j, 1 \leq J$$

Since, Task *j* must finish by its deadline time *dj*, a constraint on the deadline time of a task is introduced as follows:

$$\sum a_{ij} x_j \leq d_j ; j = 1, 2, \ldots, J \tag{6}$$

The model also assumes that we have precedence constraints as follows:

If two tasks are assigned to the same processor they must be examined and executed based on the following criteria:

Let l_j = criticality of task *j*
m_j = dynamic priority of task *j*
n_j = user priority of task *j*
a_j = arrival time of task *j*

Task with the highest criticality should be scheduled first that is, execute task j_1 before j_2 if

$l_{j1} > l_{j2}$ where l_j denotes criticality of task j.
if $l_{j1} = l_{j2}$ then

consider dynamic priority m and execute task j_1 first if $m_{j1} > m_{j2}$

if $l_{j1} = l_{j2}$ and $m_{j1} = m_{j2}$ then consider user priority n and schedule $_{j1}$ first if $n_{j1} > n_{j2}$
if $l_{j1} = l_{j2}$, $m_{j1} = m_{j2}$ and $n_{j1} = n_{j2}$ then consider arrival time and execute on first come first serve basis and execute j_1 first if and only if $a_{j1} > a_{j2}$, otherwise execute j_2 before j_1.

The precedence constraints can be put together as follows, execute task $_{j1}$ before task j_2 if and only if

$$l_{j1} > l_{j2},\ m_{j1} > m_{j2},\ n_{j1} > n_{j2} \text{ and } a_{j1} > a_{j2} \tag{7}$$

Also, there is the constraint that the number of the processors cannot be less than 5 or greater than 10.

$$5 \leq \sum_{i=1}^{I} P_i \leq 10 \tag{8}$$

Finally, the number of tasks to be scheduled cannot be less than 1

$$\sum_{j=1}^{J} T_j \geq 1 \tag{9}$$

The full MILP model for scheduling tasks in a multiprocessor system can be stated as follows:

Minimize makespan

$$Z = \sum_{i=1}^{I} \sum_{j=1}^{J} a_{ij} x_j$$

Subject to:

(i) $\sum x_{ij}, j = 1, 1 \leq i \leq J$

(ii) $\sum_{j=i}^{J} a_{ij} \leq 1 \, for i = 1, \ldots, I$

(iii) $\sum_{i=1}^{I} x_{ij} \leq 1 fori = 1,...,J$

(iv) $x_{ij} \geq 0$ for $i = 1, \ldots, I$, and $j = 1, \ldots, J$.

(v) $a_{ij} \geq 0$ for $i = 1, \ldots, I$, and $j = 1, \ldots, J$.

(vi) $\sum a_{ij} x_j \leq d_j ; j = 1,2,...,J$

(vii) $l_{j1} > l_{j2}$, $m_{j1} > m_{j2}$, $n_{j1} > n_{j2}$ and $a_{j1} > a_{j2}$

(viii) $5 \leq \sum_{i=1}^{I} P_i \leq 10$

(ix) $\sum_{j=1}^{J} T_j \geq 1$

Variables

(i) x_{ij}, a_{ij} are binary integer variables

(ii) $l_j, m_j, n_j, a_j, p_i, I_j, J$ are integer variables

(iii) s_j, d_j, a_j are real-valued variables

MODEL IMPLEMENTATION

In the model implementation, we have adopted a hybrid genetic algorithm in which tasks are mapped (represented) as genes on a chromosome. Traditionally, in genetic programming, chromosomes are represented as binary vectors. However, the binary coding scheme may not be suitable in certain situations. This is because in certain situations, the type of the problem to be solved dictates the coding scheme to be adopted (Mahmood, 2000). Experiments have shown that other representations and operators can perform in the same way that binary vectors do in traditional GAs (Syswerda, 1991). For a multiprocessor system, there are a number of issues to be addressed. First, there should be the list of tasks to be scheduled. The second is the order in which these tasks should be executed on a given processor. The third is the list of processors which these tasks should be assigned to. The chromosome representation scheme used in this study is such that each gene is a pair of decimal values (T_i, P_i) which indicates that task T_i is assigned to processor P_i. The position of genes in the chromosome indicates the order in which tasks should be executed. For instance, the chromosome representation shown in Fig. 1 indicates that task 1 should be executed on processor 4, task 5 on processor 1, task 2 on processor 3, and task 3 on processor 1. It also indicates that if two tasks are assigned to the same processor, for instance, tasks 5 and 3, task 5 is executed first followed by task 3.

(1,4)	**(5,1)**	**(2,3)**	**(3,1)**

Figure 1. Chromosome representation in HGAMTS.

One of the advantages of this multiprocessor system is that two tasks assigned to two different processors may execute in parallel provided they do not require the same resource in exclusive mode. Tasks assigned to the same processor must execute in the specified order. In the chromosome representation scheme, each chromosome has a fixed length/size. This means that the maximum number of tasks that may be considered for scheduling at a time is bounded by the chromosome size. The remaining tasks together with the newly arriving tasks are kept in the task queue (Mahmood, 2000). When one set of task is scheduled, a new set of tasks from the task queue is selected for scheduling. If the number of tasks in the task queue is less than the chromosome size, then only part of the chromosomes is used, and the application of genetic operators is restricted to that part only. The part of a chromosome being used is called the active part. It should be noted that the maximum size of the active part is equal to the chromosome size.

Genetic Operators

In order to construct the appropriate genetic operators, it is necessary to do a thorough analysis of the chromosome syntax. In this study, the position of a task on the chromosome which is a greedy consideration based on the domain-specific knowledge, determines the order in which the task will be executed. The closer the task is to the front of the chromosome, the greater are the chances of it being scheduled. Also, the nature of the task that precedes a particular task may impose some constraints on where it can be placed. For instance, if two tasks require a resource, at least one task in exclusive mode, the first task in the list may prevent the second from being scheduled. This implies that the relative order of the tasks is also important. Another important parameter is the processor on which a task is scheduled for execution. It may not be feasible to schedule a task on a processor already heavily loaded. Therefore, one has to consider not only the order of tasks in the chromosomes but also the processor on which a task is finally executed. The three genetic operators employed in this study are crossover, reproduction and mutation.

a. Crossover

The crossover operator randomly selects two chromosomes from the population, and swaps the second part of each gene after a randomly selected point. This is equivalent to assigning a subset of tasks to different processors. The crossover point is indicated by ↻ arrows.

Parent 1	(1,3)	4,2)	(3,6)	(5,6)	(2,5) ↻	(6,2) ↻	(7,2) ↻	(1,2) ↻	(3,4) ↻
Parent 2	(3,4)	(5,2)	(2,5)	(6,2)	(7,3) ↑	(5,4)	(4,1)	(4,3)	(6,7)

Figure 2. Two parents involved in crossover.

After crossover, the two parents produce these two children.

Child 1	(1,3)	4,2)	(3,6)	(5,6)	(2,3)	(6,4)	(7,1)	(1,3)	(3,7)
Child 2	(3,4)	(5,2)	(2,5)	(6,2)	(7,5) ↑	(5,2)	(4,2)	(4,2)	(6,4)

Figure 3. Two children produced from the crossover.

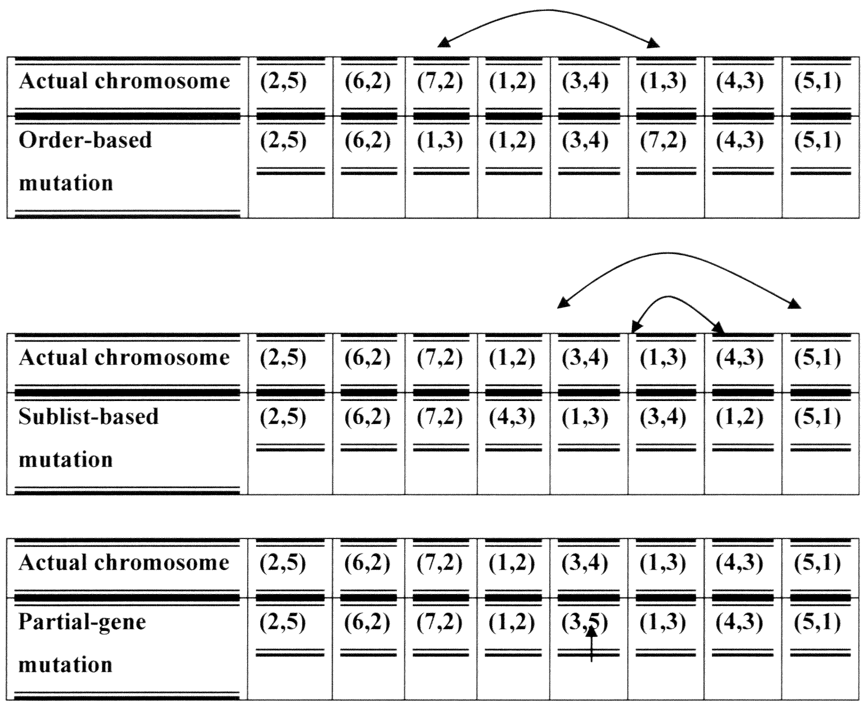
b. Mutation

Mutation provides and maintains diversity in a population, most especially when used in conjunction with other operators. On its own, it also serves as a search engine. In this study, three mutation operators used are order-based mutation, sublist-based mutation and partial-gene mutation.

Order-based mutation: It randomly selects two genes of a chromosome (the chromosome is also randomly selected) and interchange their positions.

Sublist-based mutation (also known as inversion): It randomly selects two points in a chromosome and reverses the order of the genes between these two points.

Partial-gene mutation: It randomly selects a chromosome and changes a randomly selected gene (T_i, P_i) to (T_i, P_j) for which available time (P_j) is minimum over all processors for task T_i. This mutation operator is based on the heuristic that a task should be assigned to a processor where it has the earliest start time. Fig. 4 illustrates the three mutation operators.



Actual chromosome	(2,5)	(6,2)	(7,2)	(1,2)	(3,4)	(1,3)	(4,3)	(5,1)
Order-based mutation	(2,5)	(6,2)	(1,3)	(1,2)	(3,4)	(7,2)	(4,3)	(5,1)

Actual chromosome	(2,5)	(6,2)	(7,2)	(1,2)	(3,4)	(1,3)	(4,3)	(5,1)
Sublist-based mutation	(2,5)	(6,2)	(7,2)	(4,3)	(1,3)	(3,4)	(1,2)	(5,1)

Actual chromosome	(2,5)	(6,2)	(7,2)	(1,2)	(3,4)	(1,3)	(4,3)	(5,1)
Partial-gene mutation	(2,5)	(6,2)	(7,2)	(1,2)	(3,5)	(1,3)	(4,3)	(5,1)

Figure 4. Illustration of mutation operators.

Hint: Mutated genes are shown in boldface.

Syswerda (1991) showed through several experiments that the application of mutation and crossover operators with varying probabilities produces better results than those obtained with fixed probabilities. Hence, in this study, the probability of applying partial-gene mutation and crossover is not fixed.

c. Reproduction

The reproduction operator kills the bottom x% of chromosomes of a population sorted in an ascending order of their fitness values, and then produces a clone of the fittest chromosomes. In this way, only the fittest offsprings passes into the next generation. As we go from one generation to another, fitter offspring are produced.

d. Fitness Function

In genetic algorithms, the evaluation of chromosomes is a central issue. For task scheduling problem, the fitness value of a chromosome is determined by the number of tasks in the chromosome that meet their deadlines. In this study, the fitness value of the chromosomes are generated, and are arranged in a descending order, and the bottom 10% are killed before we go into the next generation. In this way, we are able to get the best chromosome that produces the most efficient schedule based on criticality and time constraint. Once the chromosome that represents the optimal schedule of tasks has been determined by the MILP model, it is then passed on to a multi-processor system depicted in Fig. 5.

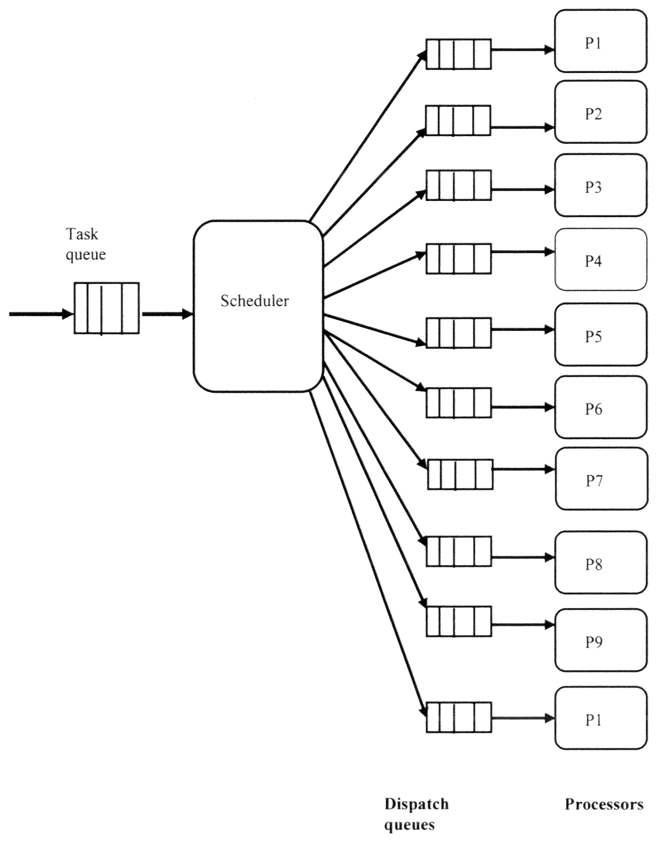


Figure 5. The scheduler model.

The scheduler model depicted in Fig. 5 involves a centralized dynamic scheduling scheme in which all tasks arrive at a central processor, called the scheduler. Each scheduler has a task queue attached to it. The task queue holds newly arriving tasks. The role of the central scheduler is to distribute the tasks to other processors in the system for execution. There is a dispatch queue associated with each processor. The communication between the scheduler and the processors is through the dispatch queues.

The scheduler makes sure that each dispatch queue is filled with a minimum number of tasks so that a processor could always find a task in its dispatch queue when it finishes execution of a task. The scheduler determines a feasible schedule based on the worst case computation times of tasks satisfying their timing and resource constraints. The scheduling algorithm has full knowledge about the currently active set of tasks, but not about the new set of tasks that may arrive while scheduling the current task set (Mahmood, 2000). The objective of the dynamic scheduling is to minimize the makespan, thereby improving the guarantee ratio (Shen et al., 1993). The guarantee ratio is the percentage of tasks that arrived in the system whose deadlines are met. The scheduler must also guarantee that the tasks already scheduled are going to meet their deadlines. The scheduler model consists of a minimum of 5 processors and a maximum of 10 processors.

The performance of the system is measured in terms of the success ratio, which is the percentage of tasks that arrived in the system whose deadlines are met. Simulation experiments were carried out in order to test the efficiency of the system. Software simulation was adopted instead of analytical or hardware prototyping because analytical modeling and hardware prototyping are costly and inflexible. Software simulations, on the other hand, are easier to work with, and are less expensive than their hardware counterparts. They are more flexible, allowing enhancements with new measuring features and real-world behavior capturing. Also, many simulations can be run simultaneously. The simulation environment is an execution-driven simulator which simulates operations of a multiprocessor system on a uni-processor host machine. The host machine is Pentium IV 3.0GHz, 160GB Hard disk and 1.0GB RAM with Windows XP Operating System.

Also, the scheduling and simulation were carried out using a partitioning scheme as against the global scheme. In the partitioning scheme, all the instances (or tasks) of a job are executed on the same processors, which are partitioned into between 5 and 10 processors during the different simulation runs. In the experiment, parallelism is prohibited, that is, no task of any job can be executed at the same time on more than one processor.

EXPERIMENTAL RESULTS

In the simulation experiment, 300 tasks were randomly generated for each simulation run, and the maximum iteration was set to 700. The chromosome size (maximum number of tasks on a chromosome) was varied between 5 and 20. The task arrival rate at the central processor (the scheduler) which also handles admission control was varied between 0.2 and 0.6 seconds. The dispatch queues were of length 2 each, that is, only two tasks could wait in the queue associated with a particular processor while waiting for a vacant slot. The results presented in Table 1 show that the success ratio increases as chromosome size increases.

Table 1

Average Success Ratio for Different Chromosome Sizes

Chromosome Size	Success Ratio
5	75
10	81
15	92
20	97

Source: Simulation studies, 2010

Also, the success ratio was computed at different task arrival rates and number of iterations. The results are presented in Tables 2 and 3, and the performance curves in Figs. 6 and 7.

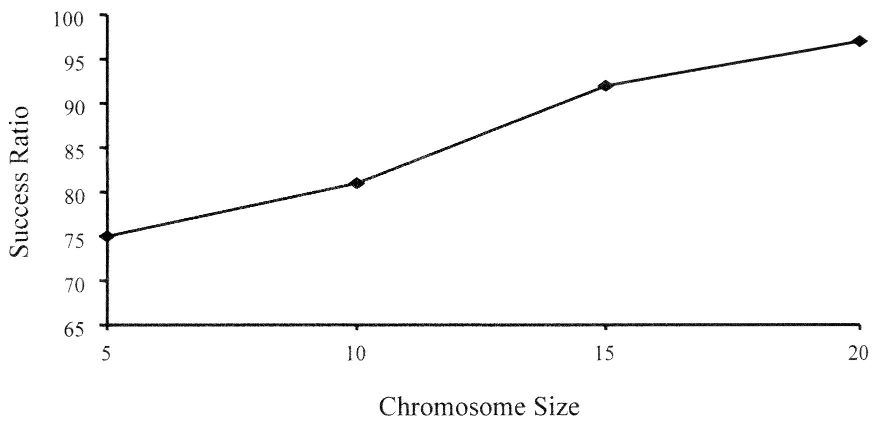


Figure 6. Average success ratio for different chromosome sizes.

Table 2

Success Ratio at Different Task Arrival Rates and Chromosome Sizes

Task Arrival Rate	Success Ratio			
	l = 5	l = 10	l = 15	l = 20
0.2	95	95	100	100
0.25	91	93	100	100
0.3	87	90	98	100
0.35	80	86	98	100
0.4	77	84	95	98
0.45	70	83	93	97
0.5	68	80	89	97
0.55	65	76	85	95
0.6	60	72	80	95

l = Chromosome size
Source: Simulation studies, 2010

Table 2 reveals that chromosome size of length 20 gives a success ratio from 95% to 100% for different task arrival rates. For chromosome size 15, the success ratio ranges between 80% and 100%. The success ratio for chromosome size of length 10 and length 5 ranges from 72% to 95% and 60% to 95% respectively. This clearly shows that chromosome size 20 has the highest success ratio with chromosome size 5 having the least success ratio. However, for small task arrival rates, the success ratio of chromosome size 15 is at par with that of chromosome size 20. The gap widens as the task arrival rates increases. Generally, Table 2 also reveals that success ratio decreases as task arrival rate increases. This result could be explained by the fact that small arrival rates minimize the amount of time spent in the dispatch queue, thus improving the success ratio (percentage of tasks arriving in the system that meets their deadlines). The result is also depicted in Fig. 7. Each point on the performance curve is an average of 10 simulation runs.

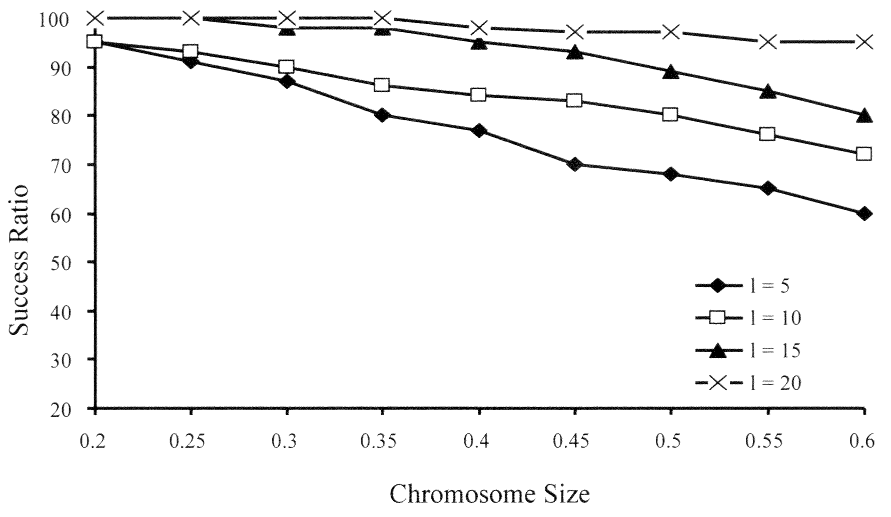


Figure 7. Success ratio vs task arrival rate for different chromosome sizes.

Table 3

Success Ratio at Different Number of Iterations

Number of	Success Ratio			
Iterations	l = 5	l = 10	l = 15	l = 20
100	41	43	47	48
200	46	47	58	59
300	48	51	62	65
400	52	58	69	72
500	57	63	83	85
600	60	66	89	96
700	61	67	92	98

l = Chromosome size
Source: Simulation studies, 2010

The result displayed in Table 3 shows that for the different chromosome sizes, the success ratio increases with the increase in the number of iterations. However, the success ratio for chromosome size 20 is the highest, reaching up to 98 with 700 iterations. The result is also shown in Fig. 8.

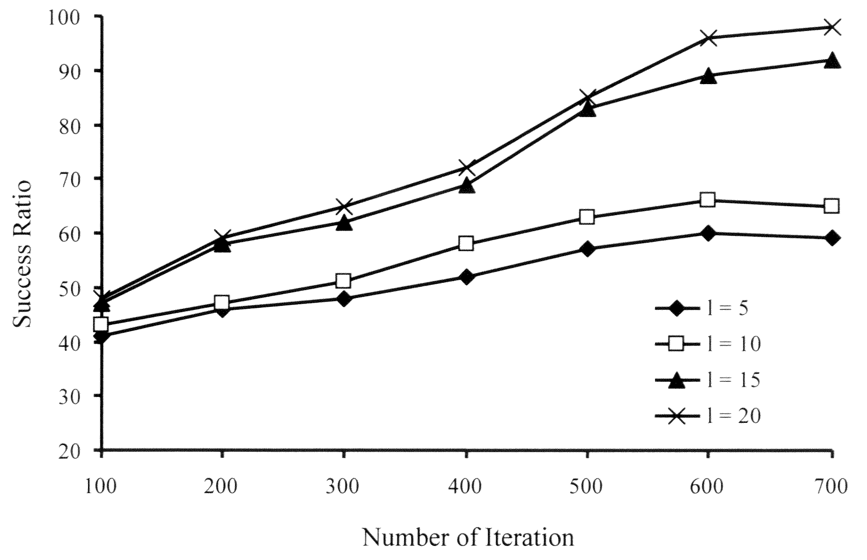


Figure 8. Success ratio vs number of iterations.

CONCLUSION

In recent times, there has been an upsurge in multimedia applications due to the advancement in electronic technology. This has led to the production of computer hardware smaller in size, but with a higher computing power. The storage space which used to be a major limiting factor in processing in the early days of computing is virtually becoming a non-critical issue. However, multimedia data are usually large in size, and some of them have to be processed in continuous stream in real-time. This scenario often leads to loss of quality and deadline misses. Uni-processor systems have been stretched to their limit, and multiprocessor systems have been designed in order to cope with the ever increasing need for higher speed and reduced deadline misses in real-time critical applications. However, the problem of determining an optimal schedule of tasks in a multiprocessor system is a major challenge in the use of multiprocessor systems.

Various models have been proposed in the literature with varying degrees of success. Finding an optimal schedule involves extensive search in the vast solution space. Some of the solutions may even be trapped in local optima due to in-exhaustive search, thus leading to inefficiency in the system. The problem is more cumbersome when there are precedent constraints to be fulfilled. The constraints for optimal schedule of tasks are often many and varied. The linear programming approach has been suggested, as it minimizes the objective function which is subject to constraints. However, it is also observed that some of the variables in a linear programming model may be real-valued (can take on fractional values) while others are integer-valued. This calls for a model that can handle both kinds of variables.

This paper therefore, presents the use of the Mixed-Integer Linear Programming (MILP) model to optimize task schedule in a multi-processor system. In implementing the model, a hybrid genetic algorithm was used in which the optimal schedule derived from the MILP model was mapped onto chromosomes. The chromosome that corresponds to the optimal schedule is then processed on a multiprocessor system with a minimum of 5 and a maximum of 10 homogeneous processors. The results of the simulation runs show that the model generates a high success ratio for the tasks that are scheduled by the system. In our future research, we will combine genetic algorithm with artificial neural network and or fuzzy logic to design a powerful hybrid genetic algorithm for multimedia task scheduling.

REFERENCES

Alberto, C., Pico, G., & Wainwright. (1994). Dynamic Scheduling of Computer Tasks using Genetic Algorithms. *Proceedings of the first IEEE Conference on Evolutionary Computation, IEEE World Congress on Computation Intelligence,* 829–833, Orlando, Florida.

Brandt, S. A., Banachowski, S., Caixue, L., & Bisson, T. (2003). Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. *Proceeding 24th IEEE International Real-Time Systems Symposium,* 396 – 407 .Cancun, Mexico.

Chen, G., Ozturk, O., & Kandemir, M. (2005). *An adaptive locally-conscious process scheduler for embedded systems. Proceding 11th IEEE Real-Time and Embedded Technology and Applications Symposium,* 354 – 364. San Francisco, CA.

Chin, C. W., & Lan, A. L. H. (2007). Web load balancing via genetic-fuzzy based algorithm. *Journal of Information and Communications Technology, 6,* 73 – 86.

Goyal, P., Guo, X., & Vin, H. M. (1996). A hierarchical CPU scheduler for multimedia operating systems. *Proceeding of USENIX on Operating Systems Design and Implementation (OSDI'96),* 107-121, Seattle, Symposium WA, USA.

Hamzeh, M., Fakhraie, S. M., & Lucas, C. (2007). Soft real-time fuzzy task scheduling for multiprocessor systems. *International Journal of Intelligent Technology*, *2* (4), 211 – 236.

Lee, J., Tiao, A., & Yen, J. (1994). A fuzzy rule-based approach to real-time scheduling. Proceeding *3rd IEEE Conf. Fuzzy Systems, IEEE World Congress Computational Intelligence.*, *2*, 1394 – 1399, Florida.

Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard real time environment. *Journal of the ACM*, *20,* (1), 46 – 61.

Mahmood, A. (2000). *A hybrid scheduling algorithm for task scheduling in multiprocessor real-time systems*. Technical Paper. Department of Computer Science, University of Bahrain.

Oluwadare, S. A. (2009). *A scheduling algorithm for enhancing operating system support forhigh-speed multimedia systems* (Unpublished doctoral dissertation). Department of Computer Science, The Federal University of Technology, Akure, Nigeria.

Shen, C., Ramamritham, K., & Stankovic, J. A. (1993). Resource reclaiming in multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, *4*, 382-397.

Sutar, S. R,, Sawant, J. P., & Jadhav, J. R. (2006). *Task scheduling for multiprocessor systems using memetic algorithms*, 27/1 – 27/9.

Syswerda, G. (1991). Schedule optimization using genetic algorithms. In L. Davis (Ed.) *Handbook of genetic algorithms* (332-349). New York: Van Nostrand Reinhold.

Tanenbaum, A. S. (2007). *Modern operating systems* (2nd ed.). Prentice Hall. pp. 250 – 275.

Thai, N. D. (2002). Real-time scheduling in distributed systems. Proceeding *International Conference Parallel Computing in Electrical Engineering.* Warsaw, Poland, 165 –170.

Sabeghi, M., Naghibzadeh, M., & Taghavi. (2006) Scheduling non-preemptive periodic tasks in soft real-time systems using fuzzy inference. In *Proceeding 9th IEEE International Symposium Object and Component Oriented Real-time Distributed Computing*. Gyeongju, Korea, 27-32.