MULTIPROCESSOR SCHEDULING BASED ON GENETIC ALGORITHMS

Intisar A. Majied Al-Said*, Nedhal Al-Saiyd**, Firas Turki Attia***

* Department of Computer Science, Al-Isra Private University, Amman-Jordan
  intisaralsaid@yahoo.com, intisar@ipu.edu.jo

** Department of Computer Science, Applied Science University, Amman-Jordan
  nedhal_alsaid@asu.edu.jo

*** Department of Control and Computer Engineering, University of Technology, Baghdad-Iraq
  liverpoolfc_1984@yahoo.com

ABSTRACT

This paper presents the development of genetic algorithm approach to schedule tasks on a multiprocessor system. The objective is to minimize the make-span i.e. the completion time of all tasks while maintaining the precedence constraints within the task graph. No inter-processor communication overheads are assumed. The array data structure is employed for string representation and a hybrid selection method for reproduction is adopted. The ability of the genetic-based scheduler to deal with resource failures and aperiodic operations is also explored.

Keywords: task scheduler, genetic algorithms, multiprocessors, parallel computing.

1. INTRODUCTION:

Genetic algorithms (GAs) are stochastic search techniques that can perform optimization based on natural selection and evolution theories without relying on gradient information or becoming trapped in local minima [3, 6]. Such flexible techniques can be appropriate for a wide range of applications, including scheduling problems.

The problem of scheduling a set of dependent or independent tasks to be processed in a parallel fashion is a well-studied area. Examples of such problems include the scheduling of jobs onto a fixed set of machines in a manufacturing plant, the scheduling of aircraft takeoffs and landings onto one or more landing strips, and the scheduling of meeting rooms to multiple events of varying size and length. The GA involvement in solving these problems may vary considerably from one to another. For example, a simple sequencing problem may be resolved entirely by a simple GA, while a complicated job-shop problem may necessitate further assistance using hybridization methods [1, 8].

With the development of parallel computing, an increasing attention has been paid for this new version of the problem. A program can be decomposed into a set of smaller tasks. These tasks are likely to have dependencies and, consequently, precedence requirements. The benefits of parallelism, however, can be nullified if some issues are not handled properly. The task scheduling is the most important of these issues because inappropriate scheduling of tasks can fail to exploit the true potential of a distributed system and can offset the gains from parallelization. The goal of a scheduler is to assign tasks to available processors such that precedence requirements between tasks are satisfied and the overall length of time required to execute the entire program, the schedule length or make-span, is minimized [5, 8]. This problem of scheduling of tasks to be executed on a multiprocessor computer is one of the most challenging problems in parallel computing. The multiprocessor scheduling is known to be NP-hard problem even in the simplest forms [4, 5]. A large number of algorithms were proposed which represent various tradeoffs between the quality of the solution and the computational complexity and scalability of the algorithm. For realistic considerations, the scheduler must also have the ability to adapt to varying resource environments [5, 7].

Many heuristics have been proposed based on a wide spectrum of techniques, including branch-and-bound, integer programming, searching, graph theory, randomization, genetic algorithms, and evolutionary methods. A survey in [5], described various scheduling algorithms and their functionalities in a contrasting fashion as well as examine their relative merits in terms of performance and time-complexity. Recent comparison study is proposed with communication delays onto a homogeneous cluster of processing elements [4].
Scheduling approaches can be classified according to the arrival time of tasks into static and dynamic. In static scheduling, all tasks to be scheduled are ready to be dealt with. In other words, no tasks arrive later. On the other hand, dynamic scheduling contains tasks with unfixed arrival times. In other words, tasks can arrive at some known or unknown future time. Scheduling approaches can also be classified according to the availability of related information into deterministic or stochastic scheduling. In deterministic category, all related information is either known or with certainty, such that tasks, processing times, and any other information, are available at the beginning, or arrive at a known future time. The stochastic scheduling deals with situations where task ready times, processing times, and any other related information, are random variables.

We introduce a deterministic multiprocessor task scheduling based on GAs that takes into consideration the changing environment. The rest of the paper is as follows: Section 2 describes the proposed GA including its operators and requirements. Section 3 gives the scheduling function. Section 4 illustrates the proposed algorithm. Section 5 evaluates the results. Section 6 makes some concluding remarks and future work.

2. GENETIC ALGORITHMS
GAs is search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures, with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using pieces from the fittest of the old, an occasional new part is tried for good measure. Although randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance [3, 6].

To use a GA, five components are required [3, 8]:
1. A way of encoding solutions to the problem
2. A way of initializing a population of solutions
3. An evaluation function that returns a rating for each solution
4. Operators that may be applied to parents when they reproduce, to alter their genetic composition, such as crossover and mutation
5. Parameter settings for the algorithm, operators, and so on

A simple GA cycle is shown in Fig. (1).

2.1 ENCODING
Various encoding methods have been created for particular problems to provide effective implementation of GAs. According to what kind of symbol is used, the encoding methods can be using binary, real, integer or literal encoding.

For more complex real-world problems, as the task scheduling, an appropriate data structure is suggested for the chromosome representation, to capture the nature of the problem. In such cases, a gene may be an n-array or a more complex data structure [6, 8].

The proposed GA suggests that each string consists of a three-dimensional integer array, with dimensions corresponding to the number of processing units (processors), the total number of operations to be performed (tasks), and the number of the information fields presented to the user [1]. The latter is decomposed into three information fields occupied by the task index, and both the start and end time of same task. For example, if task (3) is scheduled to start at time T=4 and terminates at time T=7, the information fields will be as shown in Fig. (2).

The three-dimensional array representation provides flexibility regarding the number of processing elements and total number of operations to be scheduled.

2.2 INITIALIZATION
The initial population is randomly generated, i.e. a processing element and an operation to be processed on it are chosen randomly.

2.3 GA OPERATORS
The crossover operator generates a crossover site randomly which is ensured not to be located in the middle of the information fields belonging to the same operation. The two strings involved in the crossover procedure are selected on tournament basis.

The mutation operator adopted in this work is based on swapping two operations belonging to two consecutive processing elements in the same string so as to avoid repeated or missing possible solutions through GA progress [1].

2.4 FITNESS EVALUATION
Each GA string solution is evaluated according to the following fitness function:

\[
\text{Fitness} = \frac{F1}{a+F2} \quad \text{....................} (1)
\]

Where:
- \( F1 \): Scaling factor,
- \( a \): small numerical value, taken (0.01) in this work.
- \( F2 \): Completion time of the schedule.

3. SCHEDULING FUNCTION
To minimize the make-span F2, the scheduling function is performed after receiving legal strings from the GA containing the allocation of the available processing elements to the required operations. Thus, guarantees that all precedence constraints are met, and avoids any process overlapping as the case of execution of two program tasks on the same processor simultaneously.

This function begins with determining the tasks that have no pre-requisite precedent operations, in other words, those operations that are ready to enter the schedule without waiting for others. The next step is to determine the processing elements that have schedulable
operations assigned to them. The processing element with the earliest possible starting time is then chosen. Each time an operation is scheduled, it is marked, and other operations that are awaiting it become schedulable operations. The above procedure is repeated until all operations are covered by the schedule.

In this work, rescheduling is required with minimum changes in the case of processor malfunctioning. This is achieved by distributing the operations originally assigned to the failed processing among those processors capable of performing them as equally as possible so as to avoid unnecessary overloading.

Another type of unpredictable events that may affect the schedule currently being run is the arrival of new operations with no prior knowledge of their arrival time or their required processing time. These apperiodic operations are tackled directly, where the processing element capable of performing the operation is relinquished until the request for that operation ceases. During this period, the operations previously assigned to the relinquished processor are distributed among the other elements capable of performing them as equally as possible. In case of the existence of more than one processing element capable of performing the apperiodic operation, the processor with the lesser number of operations is chosen. In the case of failures occurring to the processing element performing the apperiodic operation, another capable element, if it exists, is relinquished with its original operations redistributed and so on.

4. THE PROPOSED GA TASK SCHEDULER
The GA approach of the task scheduling proceeds as:
1. Generation counter = 0
2. Generate the initial population
3. Calculate the objective and fitness function of each individual using the scheduling function for the legal strings. A zero fitness value is assigned for illegal solutions as a penalty to reduce its chance for reproduction.
4. Sort in a descending order all the population according to their fitness then save the best few individuals for elitism purposes
5. Perform crossover and mutation as described above
6. Sort the population and perform elitism
7. Generation counter = generation counter+1
8. If generation counter < generation size Go to 5

The scheduler deals with the cases of resource failure, repair or the presence of apperiodic operation as earlier mentioned in section 3.

5. EVALUATION OF THE GA-BASED SCHEDULER
As a case study, the problem taken from [2] is considered. The task put forward consists of scheduling nine non-preemptive operations (i.e. operations that are not allowed to be interrupted before their completion) on a number of identical microprocessors, with precedence constraints as given in Fig (3). The precedence graph shows that each operation (node) is associated with a number corresponding to the processing time required for that operation.

The optimal schedules are shown in Fig (4) assuming two processors. After few generations, more that one solution obtained, with final scheduler time of 15 as that obtained using heuristics in [2]. The maximum fitness values versus generation number for a number of runs are shown in Fig (5).

Concerning the number of processors used, estimation is obtained from the graph representation in Fig (3). It is noticed that the maximum number of operations with simultaneous processing periods, assuming an infinite number of processors, is three. Therefore using more than three processors is meaningless. Now, for a schedule to achieve the minimum completion time, operations (2) and (3) should not be placed on the same processor since they may start simultaneously. This leaves two possibilities of three tasks being processed simultaneously, these are (6,4,5) and (6,7,8). In either cases, the relatively large processing time of operation (7) makes using a third processor unbeneficial, since the other operations may be completed on a single processor before completing operation (7).

If the example is slightly modified by reducing the processing time of operation (7) to only 4 time units instead of 7, then, using a third processor will lead to a shorter completion time than that obtained using only two. Fig (6) shows that the make-span is 12 with three processors while it is 14 for two processors.

To investigate the ability of the presented GA-scheduler to deal changes as the case of resource failure and the presence of apperiodic operations, let us return to the task graph in Fig (3). This is done with four processors, as illustrated in Fig (7).

6. CONCLUDING REMARKS AND FUTURE WORK
A scheduling algorithm has been developed to schedule non-preemptive tasks onto identical multiprocessors computing system. It provides efficient schedules and adapts to varying resource availability (processing elements).

In the case of more than one optimal schedule existing with respect to specific objectives, genetic algorithms offer a good possibility of presenting more than one optimal solution as shown in Fig (4). This may be of great importance when facing unpredictable events, such as processor failures.

Increasing the number of processing elements may not necessarily offer a better solution with respect to time. As mentioned earlier, this factor is problem-dependent relating the corresponding precedence constraints among tasks and their required processing times.

Although the inter-processor communication overheads are not considered in this work, it represents an important factor for realistic multiprocessor
scheduling problem, that will be considered in future work.

REFERENCES:

Fig (1): A Simple GA Cycle [1].
Fig (2): Information fields for the task graph scheduler problem.

Fig (3): Task graph [2].
Fig. (4): Genetically obtained optimal schedules

Crossover probability=0.95, Mutation probability=0.5 (top)
Crossover probability=0.8, Mutation probability=0.3 (bottom)

Fig (5): Best fitness value vs. generation number
Fig (6): The scheduler with 2 and 3 processors if the duration of task 7 is set to 4.
Fig (7): Rescheduling the scheduler due to failure and the presence of apperiodic operation. (Top: optimal scheduler with all four processors functioning properly)