# A Genetic Algorithm for Multiprocessor Task Scheduling

Tashniba Kaiser, Olawale Jegede, Ken Ferens, Douglas Buchanan
Dept. of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada
Ken.Ferens@ad.umanitoba.ca

*Abstract—The goal of task scheduling in a multiprocessor system is to schedule dependent tasks on processors such that the processing time is minimized. This ensures optimal usage of the processing systems. However this problem is NP-hard in nature and heuristic based techniques are used to obtain a good schedule in polynomial time. Genetic Algorithms (GA) have been proposed over other heuristics because it can use its genetic processes to find multiple solutions faster. The GA proposed is based on a non-pre-emptive precedence relation between tasks in the task graph. Tasks assignment is prioritized based on the number of tasks dependencies (NTD) and the earliest start time (EST) of each task. For tasks with multiple possible earliest start times, the minimum earliest start time is chosen for such tasks. Java simulations compared the results obtained using the minimum EST and the maximum EST. Our simulation shows that the proposed algorithm with minimum EST achieves faster processing periods compared with the maximum EST.*

Keywords—Genetic Algorithm, Number of Task Dependencies, Total Finishing Time, Multiprocessor Scheduling.

## I. INTRODUCTION

The need to achieve optimal usage of a multiprocessing system for task allocation cannot be overemphasized. The aim is to ensure that the processing period is minimized by scheduling tasks on time. However this problem of obtaining optimal task scheduling in the multiprocessing system is reported to be NP hard [1]. There are different scheduling algorithms such as First-in-First-Out, Shortest-Job-First, Priority based scheduling, Round-robin scheduling, Multilevel Queue scheduling etc. It is important that any algorithm chosen is able to address the scheduling problem in polynomial time. Due to the computational complexity however, different types of machine learning techniques have been proposed. Some of the heuristics that have been widely used for this problem are simulated annealing, tabu search, ant colony optimization, and genetic algorithms among others. Genetic algorithms are efficient in solving NP hard problems especially in parallel computing such as any multiprocessing system.

The common approach to this problem has been to use the precedence-relations between tasks to prioritize task assignment on the processors. This is also known as the height-based tasks assignment, somewhat similar to the first-in-first-out method as tasks with higher heights are given priority than those down the tasks-graph. However to further improve the optimization process (by further reducing the makespan), a new approach is to assign tasks based on the number of task dependencies (NTD) of each task. Which means a task must be executed before all the other tasks that depends on it can be executed. Thus, irrespective of a task's height in a task graph, priority is given to tasks with higher number of task dependencies. This ensures a decrease in the total finishing time (TFT) of the schedule.

The remaining sections of this paper are organised as follows. Section 2 represents related work on multiprocessor task allocation problem and our contribution to this work. Section 3 discusses the GA approach to the multiprocessor task allocation problem and methodology. In Section 4, we discuss the simulations and results obtained using genetic algorithm. Section 5 concludes the paper and gives future work.

## II. RELATED WORK

There have been several approaches to the tasks allocation problem in a multiprocessing systems. Most of the approaches have been based on non-pre-emptive precedence relations between tasks in the task graph. Jin et al [2] carried out a comprehensive survey of nine scheduling algorithms which are frequently used to solve the multiprocessor task scheduling problem and compared the performance of each of the algorithms. The nine algorithms considered were min-min, chaining, A*, genetic algorithms, simulated annealing, tabu search, Highest Level First Known Execution Times (HLFET), Insertion Scheduling Heuristic (ISH), and Duplication Scheduling Heuristics (DSH) with task duplication. The performance of the nine algorithms was benchmarked against two widely used algorithms in

linear algebra which are the LU decomposition and the Gauaa-Jordan elimination. With task duplication, the DSH performed best while the ISH performed best without task duplication. It was also reported that the GA and tabu search obtained the best solution out of all the iterative search algorithms considered. However, in this work task duplication was not considered. Other works have been done reporting the performance of the GA. Majority of the works [1] [3] [4] [5] [6] assume non-pre-emptive precedence relations between tasks in the task graph as well as non-duplication of tasks. Wu et al [7] however assumed duplication of tasks. The non-preemptive characteristics of tasks in the task graph ensures that precedence relations are adhered to. This necessitates the priority-based task scheduling. The most common scheduling method is to prioritise task based on their height [1]. The height is used to denote the precedence relations between tasks in a task graph. The higher a task is in the task graph, the higher the priority given to it in allocating it to the multiprocessing systems. However, as a result of tasks dependencies, the height-based task scheduling can be inefficient. Tasks at a higher height with no task dependencies will be scheduled ahead of tasks at a lower height with task dependencies. This increases the makespan of the processor. Abdeyazdan and Rahmani [8] proposed a new algorithm which prioritizes tasks scheduling based on the number of task dependencies of each task and the earliest start time of each task. This ensures that tasks having higher task dependencies are given higher priority irrespective of their height on the precedence graph thereby resulting in a further decrease in the makespan of the processing system.

In this work, we have considered the algorithm proposed by [8]. We observed that in the task graph, there may be tasks with multiple possible earliest start times. Our contribution is that for tasks with multiple possible earliest start times, our algorithm choses the minimum earliest start time for such tasks as against the maximum earliest start time used by [8]. This is akin to choosing the shortest path as against the longest path in a routing problem. Our algorithm ensures a further decrease in the makespan.

## III. GA MULTIPROCESSOR TASK SCHEDULING

The main goal of a scheduling problem is to reduce the schedule length (makespan) of the processor. For a multiprocessing systems N processors, the time it takes for the last on a processor to finish executing is termed the *finishing time* FT. The maximum finishing time among the *m* processors in any schedule is termed the Total Finishing Time TFT of that schedule. For *k* number of schedules, the TFT can be represented as in (1) below.

$$TFT_k = \sum_{1}^{k} Max\,\{FT\ of\ Processor_i\} \quad for\ m \geq i \geq 1 \tag{1}$$

### *Height-Based Scheduling*

Hou and Ansari [1] based their task priority-scheduling in a multiprocessor system on the task height of each task. In this model, tasks that are higher up the task graph are given priority compared to tasks on lower levels. In a task graph where there is a sequence of directed edges from task, say $t_i$ to $t_j$, then $t_i$ is higher up the graph while $t_j$ is lower down the graph. This precedence relation implies that task $t_i$ has to be executed before tasks $t_j$ and other tasks that precede that task $t_i$. According to [1], if PRED($t_i$) is a set of preceded tasks of $t_i$, then we can obtain the height of any of the preceding tasks using equation (2).

$$height\ (t_i) \begin{cases} 0 & if\ PRED(t_i) = \emptyset, \\ 1 + \max\{height(t_j)\}\ otherwise.. \\ t_j \in PRED(t_i) \end{cases} \tag{2}$$

The height function given above is a mathematical representation of the precedence relations between the tasks in the task graph. Since the task height increases from 0 to a finite length, the preceding tasks to the task(s) at height 0 will have heights greater than 0. Therefore the lower the task height of any task, the higher up the task graph the task is. In other words, if task $t_i$ is preceded by tasks $t_j$, then $t_i$ will be executed before $t_j$ and height($t_i$) < height($t_j$). However if there is no precedence relation between any two tasks, the order of execution can be arbitrary.

### *Problem with Height-Based Scheduling*

To explain the drawback of a height-based scheduling algorithm, we have used the task graph below [8]. Each of the tasks t0 to t15 will have a height and an execution time. Task t0 is at height 0 (highest), t1 and t2 have height 1, etc. as shown in the table 1 below. The execution time of each task is assigned randomly ranging from 0 to 15. The height-based scheduling is such that tasks at higher heights are scheduled before task at lower heights. However for tasks at same heights, any of them is randomly chosen to be scheduled on the processor.

The implication is that: “tasks such as t7 will be scheduled before tasks such as t14.”

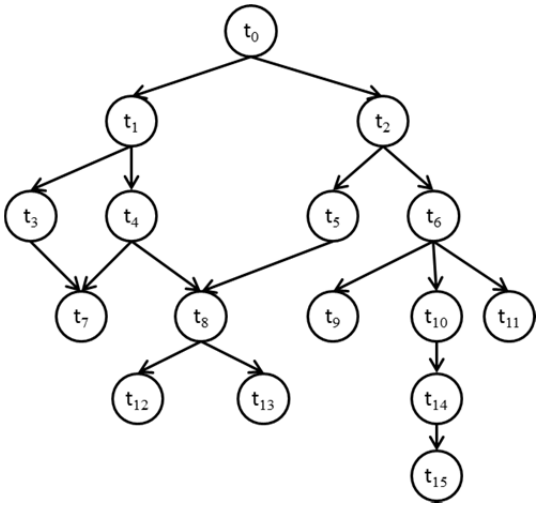


Fig. 1 A task graph, from [8].

Table 1. Height and Execution time of tasks in Fig. 1.

| Task | Height | Execution time |
|---|---|---|
| $t_0$ | 0 | 3 |
| $t_1$ | 1 | 2 |
| $t_2$ | 1 | 4 |
| $t_3$ | 2 | 1 |
| $t_4$ | 2 | 10 |
| $t_5$ | 2 | 3 |
| $t_6$ | 2 | 6 |
| $t_7$ | 3 | 9 |
| $t_8$ | 3 | 7 |
| $t_9$ | 3 | 11 |
| $t_{10}$ | 3 | 5 |
| $t_{11}$ | 3 | 5 |
| $t_{12}$ | 4 | 8 |
| $t_{13}$ | 4 | 10 |
| $t_{14}$ | 4 | 15 |
| $t_{15}$ | 5 | 2 |

It is observed that task $t_7$ has no task dependency, meaning there is no task that needs task $t_7$ to complete before it can start. However, tasks $t_{14}$ has task dependency; task $t_{15}$ cannot be scheduled unless task $t_{14}$ finishes execution. Therefore, scheduling based on height increases the FT of a processor and consequently the TFT. For a task graph with a high percentage of task dependencies, the height-based scheduling will not be suitable to achieve optimal scheduling. This is why a new algorithm is needed for optimal task scheduling in a multiprocessing systems.

*Task Dependency-based Scheduling*

The main goal of a scheduling problem is to reduce the TFT (makespan) of the processor. To further reduce the TFT, Abdeyazdan and Rahmani [8] proposed a new algorithm which prioritizes tasks scheduling based on the number of task dependencies and earliest start time (EST) of each task. This ensures that tasks having higher task dependencies are given higher priority irrespective of their height on the precedence graph. For any task $t_i$ with a $j$ number of outgoing edge, i.e. tasks that directly depends on $t_i$, the Number of Task Dependency (NTD) for such task $t_i$ is mathematially obtained by equation (3).

$$NTD(t_i) = \begin{cases} 0 & ;if\ NTD(t_i) = \emptyset, \\ \sum_{j=1} \{1 + NTD(t_j)\}\ otherwise. \\ t_j\ directly\ depends\ on\ t_i; \end{cases} \quad (3)$$

The NTD function above represents the total number of tasks that depends on a task $t_i$ whether directly or indirectly, this [8] termed as number of children. With this, a task with more number of tasks dependencies will be scheduled earlier than one with lower number of task dependencies. The concept of Earlier Start Time was introduced to ensure that tasks are scheduled with respect to the earliest time for which they are available to be scheduled. The EST of any task is a function of the summation of the execution time of all the tasks that precedes such tasks. However, because there could be one or more path on the task graph along which a task could be executed, this implies that there will be multiple EST for such task. Our algorithm selects the minimum EST. An algorithm to produce the schedule based on the number of task dependencies is as below:

1. Arrange the tasks in descending order based on the number of task dependencies of each task.
2. Put tasks with the same NTD in a single group and perform steps a and b for all the groups in order of higher NPD until every group is empty.
   a. Randomly select a task from the group and then delete it from the group.
   b. Allocate the selected task to one of the processors based on the EST method such that the starting time of the task on that processor is less than other processors.
      i. For tasks with multiple EST, choose the minimum EST in performing *b*.
3. Repeat steps 'a' and 'b' until all the tasks have been selected.

The algorithm is such that every task is assigned only once to a processor as there is no repetition of same task on the task graph. From the task graph in figure 1, we can arrange the tasks according to the NTD of each tasks. The execution time of each task is used to compute the EST of each task. Table 2 shows the EST of each tasks arranged according to the descending order of the NTD of each tasks. From Table 2, we see that each of tasks $t_{8,}$ $t_{7,}$ $t_{12,}$ and $t_{13}$ have two ESTs because there are two possible path from which

the EST can be obtained. For instance task $t_8$ have an EST of 10 along the path $t_4 \rightarrow t_1 \rightarrow t_0$ and an EST of 15 along the path $t_5 \rightarrow t_2 \rightarrow t_{10}$. Our algorithm always chooses the minimum EST.

*Scheduling using GA*

The goal of the combinatorial optimization problem in this work is to find optimal task schedule in a very short time. Since this problem is NP-complete, we have chosen the GA to do the task scheduling. GA is a subset of evolutionary algorithms that models biological processes to optimize highly complex functions.

Table 2. Task Order based on NTD.

| Task | NTD | EST |
|---|---|---|
| $t_0$ | 15 | 0 |
| $t_2$ | 10 | 3 |
| $t_1$ | 6 | 3 |
| $t_6$ | 5 | 7 |
| $t_4$ | 4 | 5 |
| $t_5$ | 3 | 7 |
| $t_8$ | 2 | 10,15 |
| $t_{10}$ | 2 | 13 |
| $t_3$ | 1 | 5 |
| $t_{14}$ | 1 | 18 |
| $t_7$ | 0 | 6,15 |
| $t_9$ | 0 | 13 |
| $t_{11}$ | 0 | 13 |
| $t_{12}$ | 0 | 17,22 |
| $t_{13}$ | 0 | 17,22 |
| $t_{15}$ | 0 | 33 |

The GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness" (i.e. minimize the objective function). It is important that a solution is found in good time because time plays an important role in real time applications for task scheduling in multiprocessing system. The main advantage of using GA over other stochastic techniques is its parallelism which enables faster convergence. GA therefore outsmarts all other meta-heuristic techniques in terms of the time it takes to arrive at a good solution. The GA is able to provide a list of optimum solutions at a single iteration; this is particularly good for our application because we have multiple processors on which the scheduling is done. GA's are also less likely to get stuck in local minima because of its crossover and mutation processes. GA is therefore well suited to our problem. The GA procedure is shown in Table 3.

The initial population consists of solutions in the search space. A solution represents a schedule generated every time the algorithm in section 3.3 is run. In GA term a solution is termed a chromosome. A particular population size is chosen depending on the problem size. Each of the solutions in the initial population is examined using the objective function in Equation (1). In GA terms, the objective function represents the fitness function. The goal is to minimize the function. The lower the TFT of a schedule, the better it satisfies the objective function.

Table 3. Standard Genetic Algorithm.

| Step | Action |
|---|---|
| 1 | Generate a random initial population of *n* schedules, where *n* is the population size. |
| 2 | Evaluate the fitness of each of the schedule in the initial population. |
| 3 | Generate new populations using processes in steps 4-6 |
| 4 | Selects two schedules among the current population using the roulette wheel method based on fitness of each schedule. |
| 5 | Crossover the two selected schedules considering the crossover probability, to form the schedules for the next generation |
| 6 | Mutate the one of the selected schedules at each defined mutation point, considering the mutation probability and place it in the new population. |
| 7 | Evaluate the fitness of each of the schedules in the new population |
| 8 | Repeat steps 3-7 until the stopping criteria have been met. |

GA uses selection, crossover and mutation processes to generate new solutions (schedules) in the search space.

• **Selection** deals with the probabilistic survival of the fittest, in that the fittest schedules are chosen to survive. Fitness is a comparable measurement of how well a schedule satisfies the objective function. Once the schedules with the better fittest are chosen, others will be eliminated. Simply, the probability of a chromosome to be selected is proportional to the quality value/fitness; this is also called the roulette wheel selection method. There are various selection methods but we propose to use the roulette wheel selection algorithm because it gives every chromosome a chance of survival. The lower the TFT of a schedule, the larger the slot it occupies in the roulette wheel and consequently the higher the chances of being selected for every spin.

• **Crossover** is a technique considered to be the most important step in the context of GAs. At a certain crossover rate, GA selects two schedules from the population based on roulette wheel method. After selecting these two schedules, using the roulette wheel, a task is randomly selected from the ordered set of tasks based on their NTD. In one of the schedule (first schedule), the algorithm will choose all the tasks that have equal or lower number of NTD to the selected task. For each processor in the first schedule, the

chosen tasks are exchanged with the other tasks on corresponding processor in the second schedule. This produces two new schedules with most likely varying TFT to the initial schedules.

• **Mutation** is a genetic operator used to maintain genetic diversity, at a certain mutation rate, from one generation of a population of schedules to the next. Mutation alters one or more gene (task) values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better or worst solution by using mutation. Consequently, mutation aids GA to avoid getting stuck in local minimal. To do mutation, the two different schedules and task selected for the crossover process are used. In the first schedule, the selected task is exchanged with another task with equal NTD on another processor in the same schedule. This same mutation process is done for the second schedule. Like the crossover, this procedure also produces two new schedules with most likely varying TFT to the initial schedules.

After each cycle of selection, crossover and mutation, the newly generated sets of solutions (schedules) are termed *new generation*. Every *generation* is evaluated based on the fitness function to determine if they represent a good enough solution to satisfy the fitness function. This determines if the GA can stop searching, or if otherwise, for the GA to continue searching until the set stopping criteria is met. The stopping criteria could be *the number of generations, or evolution time, or fitness threshold, or fitness convergence, or population convergence.* In our case, the *number of generations* was set as the stopping criteria. The schedule obtained after the stopping criteria will be the optimal or near optimal schedule.

## IV. Experimental Results

The GA Simulation was done in Java to evaluate our algorithm. Task graphs were created with number of tasks in the graph ranging from 16, 21, and 30. The task dependency percentage range between 20 and 60 and the execution time for each task is random between 1 and 15 s. The task graphs are scheduled on a multiprocessor system with 3 processors for the two genetic-based algorithms with maximum earliest start time and our algorithm minimum earliest start time. The genetic algorithm parameters chosen are population size of 40, crossover rate of 0.8 and the mutation rate of 0.1, number of generations of 50.

Table 4 shows the schedules and the TFT for each of the Max-EST and the Min-EST algorithm. The results shows that the Min-EST can schedule tasks either with same TFT or lower TFT compared to the Max-EST. The computation time is however the same for both algorithms.

Table 4. Schedules for the 2 Algorithms.

| Algorithms | Total Finish Time (seconds) | | |
|---|---|---|---|
| | Number of Processors = 3 | | |
| | Number of Tasks | | |
| | 16 | 21 | 30 |
| Max-EST | 40 | 76 | 146 |
| Min-EST | 40 | 75 | 134 |

Table 5. Schedule with varying NTD of tasks.

| Algorithms | Total Finish Time (seconds) | | |
|---|---|---|---|
| | Number of Processors = 3 | | |
| | Number of Tasks = 30 | | |
| | Increase NTD of Tasks with Multiple ESTs | | |
| | 20% | 40% | 60% |
| Max-EST | 146 | 187 | 209 |
| Min-EST | 134 | 162 | 176 |

Table 5 shows the results obtained when the number of task dependencies (NTD) of tasks with multiple ESTs is considerably large compared with other tasks in the graph with single EST. With a task graph containing 30 tasks, our algorithm (Min-EST) outperforms the Max-ESTs algorithms with increasing NPD of tasks with multiple ESTs. This occurs because since both algorithms are based on NTD, tasks with higher NTDs are given priority than those with lower NTDs. In effect, if a task with multiple ESTs have a higher NTD, it will be scheduled earlier and the minimum EST of such task is likely to be less than the current available start time on any of the processor. In the same vein, if the NTD of a task with multiple ESTs is considerably small compared to other tasks with single EST, then such tasks will be scheduled late at which time the minimum ESTs will be insignificant because the current available start time on any of the processors would have exceeded the minimum EST. Therefore our algorithm outperforms the Max-EST algorithm only when the NTD of tasks with multiple ESTs is considerably high compared to that for tasks with single EST.

## V. Conclusions and Future Work

This paper presents a simulation of multiprocessor tasks scheduling based on the number of task dependencies using GA. GA was used because this problem is NP_Hard and a an optimal-or near-optimal schedule is needed in good time. Tasks with higher number of task dependencies were given priority independent of the height of such tasks. This helps to further ensure that all the tasks in the tasks graph are scheduled on time using the earliest start time of each task. It was observed that some tasks can have more than one EST as a result of multiple path of reaching such tasks in the tasks graph. Our idea ensures that the minimum of the multiple ESTs is chosen. Choosing the minimum ESTs is only significant when the tasks with multiple ESTs are given

priority in scheduling which occurs only when the number of task dependency is considerably high compared to tasks with single ESTs. Simulation shows that our algorithm will outperform the Max-EST algorithm only when the tasks with multiple ESTs have higher task dependency compared to other task with single EST in the task graph. For future work, an adaptive adjustment of the algorithm parameters (crossover and mutation rate) proposed by Yun-Xiao [9], can be implemented in order to reduce the vector distance between individual schedules. This should reduce the convergence time for our proposed GA.

## REFERENCES


[1] E. S. Hou, N. Ansari and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," in *IEEE Transactions on Parallel and Distributed Systems*, 1994.

[2] S. Jin, G. Schiavone and D. Turgut, "A Performance study of multiprocessor task scheduling algorithms," *Journal of Supercomputing,* vol. 43, no. 1, pp. 77-97, January 2008.

[3] M. U, C. Ho, S. Funk and K. Rasheed, "GART: A Genetic Algorithm based Real-time System Scheduler," in *IEEE Congress on Evolutionary Computation*, 2011.

[4] D. Montana, G. Bidwell and S. Moore, "Using Genetic Algorithms for Complex Real Time Scheduling Applications," in *IEEE Network Operations and Management Symposium*, 1998.

[5] R. M. Miryani and M. Naghibzadeh, "Hard Real-Time Multiobjective Scheduling in Heterogenous Systems Using Genetic Algorithms," in *International CSI Computer Conference*, 2009.

[6] Y. Monnier, J.-P. Beauvais and A.-M. Deplanche, "A Genetic Algorithm for Scheduling Tasks in a Real-Time Distributed System," in *Euromicro Conference*, 1998.

[7] A. S. Wu, H. Yu, S. Jin, K.-C. Lin and G. Schiavone, "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling," in *IEEE Transactions on Parallel and Distributed Systems*, 2004.

[8] M. Abdeyazdan and A. M. Rahmani, "Multiprocessor Task Scheduling using a new Prioritizing Genetic Algorithm based on number of Task Children," in *International Conference on Distributed and Parrallel Systems*, 2008.

[9] Z. Yun-Xiao, Z. Jie and Z. Chang-Chang, "Cognitive Radio Resource Allocation based on Coupled Chaotic Genetic Algorithm," in *IOP Science Chinese Physics B*, 2010.