International Journal of Soft Computing 8 (1): 50-55, 2013
ISSN: 1816-9503

Scheduling of Real Time Tasks Using Ant Colony Optimisation

[1]G. Umarani Srikanth, [2]V. Uma Maheswari, [3]A.P. Shanthi and [3]Arul Siromoney
[1]Department of PG Studies, S.A. Engineering College, Chennai, Tamilnadu, India
[2]Department of Information Science and Technology,
[3]Department of Computer Science and Engineering, College of Engineering,
Anna University, Chennai, Tamilnadu, India

Abstract: Multiprocessor task scheduling is a key research area in high performance computing. The problem of allocating a set of real time tasks to heterogeneous processors without violating the deadlines is in general a NP-complete problem. Researchers present an approach for generating a feasible schedule on real-time systems. This research uses a paradigm based on Ant Colony Optimisation (ACO). An attempt is made to ensure load balancing across the processors with deadline compliance for all the tasks. The heterogeneity of the processors is modelled by assuming different utilisation times for the same task on different processors. For a given instance of the problem, ten runs are conducted based on an ACO algorithm and the average wait time of all tasks is computed. Also, the average utilisation of each processor is calculated. For the same instance, the two parameters: average wait time of the tasks and utilisation of processors are computed using the First Come First Served (FCFS).

Key words: ACO, heterogeneous multiprocessors, real time tasks, tasks scheduling problem, heuristics, FCFS, processor utilisation

INTRODUCTION

The heterogeneous computing platform meets the computational demands of various problems. In such platforms the tasks can be executed in sequence or in parallel on two or more processors (Taylor *et al*., 2002; Umarani *et al*., 2012a, b). One of the key challenges of such heterogeneous processor system is effective tasks scheduling. The performance of the heterogeneous multiprocessing systems highly relies upon the scheduling of real time parallel tasks. The problem of scheduling of real time tasks has been proven to be NP-complete (Graham *et al*., 1979; Cassavant and Kuhl, 1998) for which optimal solutions can be found only after an exhaustive search. The problem of assigning real time tasks to heterogeneous processors deals with finding proper assignment of tasks to processors in order to optimize some performance metric such as the system utilization and the turnaround time (Shih-Tang *et al*., 2008).

Literature review: An efficient task scheduling avoids the situation in which some of the processors are overloaded while some others are idle (Mao, 2010; Umarani *et al*., 2012a, b). The goal is usually represented as some cost function which may consider the combination of several criteria: fair load sharing between the processors, maximizing the degree of parallelism, reducing the average execution times of the program, minimizing the amount of communication among the processors, minimizing the makespan of the path and so on. In order to be of use in achieving a satisfactory solution, this cost function would include the constraints like tasks execution time, deadline of the tasks, inter-task communication time, precedence between tasks, speed of the processors, memory system properties and so on.

Several heuristic algorithms are proposed to solve the same problem. For a homogeneous multiprocessor platform (Davis and Burns, 2011) where all the processors are identical assigning the tasks to the processors requires to solve the Bin Packing Problem (BPP) (Kang and Park, 2001). But in a heterogeneous platform where a task may need different execution times on different processors, BPP cannot be applied for scheduling these tasks. The researcher has compared eleven heuristics for mapping and scheduling a set of tasks onto the heterogeneous processors and the final goal is to minimize the makespan (Braun *et al*., 2001). Other categories of algorithms are also defined such as list-based scheduling (Radulescu and Van Gemund, 2002), cluster based (Aguilar and Gelenbe, 1997) and duplication based scheduling heuristics. List-based heuristics assign

Corresponding Author: G. Umarani Srikanth, Department of PG Studies, S.A. Engineering College, Chennai, Tamilnadu, India

priority level to the tasks and map the highest priority task to the best fitting processors. Cluster based heuristics form groups consisting of all tightly coupled tasks and assign them onto the same processor whereas the duplication based heuristics combine the above two heuristics. ACO, a meta-heuristic, inspired by the foraging behaviour of the ants is an adaptive algorithmic framework that can be used to define heuristic methods for real time tasks scheduling. ACO is a class of constructive meta-heuristic algorithms that share the common approach of building a solution on the basis of information provided by both a standard constructive heuristic function and previously constructed solutions. It is also a popular technique for solving approximate combinatorial optimization (Blum and Roli, 2003) problems. The core of this behaviour is the indirect communication between the ants by means of chemical pheromone trails which enables them to find short paths between their nest and food sources (Blum, 2005).

In ACO, a set of ant-like agents solve the problem cooperatively to find a feasible solution. The positive feedback of pheromones deposited on arcs comprising more optimal node-arc tours allows the next cycle to move towards the direction of an optimal solution. A major advantage of ACO over other meta-heuristic algorithms like Simulated Annealing (SA) or Genetic Algorithms (GA) is that the problem instance may change dynamically (Chen *et al*., 2011) as in many real time systems, the tasks may vary dynamically. In this framework, the decisions made by all ants are purposeful and the experiences of all ants are utilised in each iteration to construct the new optimal solution.

MATERIALS AND METHODS

A well known strategy behind efficient execution of a huge application on a heterogeneous computing environment is to partition it into multiple independent tasks and schedule such tasks over a set of available processors.

Processor characteristics: The processors are assumed to be heterogeneous. The heterogeneity of the processors is defined by the varied proportional utilisation of the same task on different processors.

Tasks characteristics: Tasks are assumed to be real time and independent. There are no precedence constraints among them. Also there is no inter-task communication. The utilisation of a processor by a task is known a priori and it does not change with time. All the tasks are assumed to arrive at the same instant at 0 time units. They do not share any resources. All the tasks have specific deadlines by which they must complete their execution.

Scheduling problem: Let HMP (Heterogeneous Multi Processors) = $\{P_1, P_2,\ldots, P_m\}$ denote m processors and each processor P_j run at variable speed. Let TS (Tasks Set) = $\{T_1, T_2,\ldots, T_n\}$ denote n real time tasks (Chen and Cheng, 2005).

Each task T_i is characterized by $\{u_{i,j}, d_i\}$ where, u_{ij} is the worst case execution time of the task T_i on processor P_j and d_i is the deadline of the task i.

The utilisation matrix U of size n×m where, n is the number of tasks and m is the number of processors and whose elements are real numbers in (0, 1) gives the proportional utilisation of a processor by a task. In other words, a typical entry $u_{i,j}$ denotes the fraction of the computing capacity of P_j required to execute T_i. $u_{i,j}$ is also referred as utilisation of T_i on P_j. The number of rows is equal to the number of tasks and the number of columns is equal to the number of processors.

The TSP (Task Scheduling Problem) can be formally described as follows: Given HMP and TS, determine a schedule that assigns each of the tasks in TS to a specific processor in HMP in such a way that the cumulative utilisation of the tasks on any processor is no greater than the utilisation bound of that processor which is 1.0 and all the tasks are deadline compliant. That is they are able to complete their execution on the respective processor within their deadline constraint.

TSP can be represented by a bipartite graph with two classes of nodes: TS and HMP. A task is mapped to a TS node and a processor is mapped to a HMP node. The graph is a directed graph with the edges leaving from the class of tasks nodes to the class of processor nodes. There is a directed edge from a TS node to a HMP node if and only if the corresponding task can be assigned to that processor without exceeding its available computing capacity and can be executed before its deadline. More than one task can be scheduled on the same processor.

A schedule can be represented as an n×m binary matrix where n represents the number of tasks and m denotes the number of processors. A typical entry of this matrix is denoted as $s_{i,j}$. The entry $s_{i,j} = 1$ if task T_i is scheduled on processor P_j. Note that there are no two 1's in the same row. This means that a task is assigned to only one processor. A column can have many 1's indicating that all the corresponding tasks are scheduled on that processor. But the proportional utilisation of all the tasks on a processor should not exceed 1. In other words:

$$\sum_{j=1}^{m} s_{i,j} = 1 \quad \text{for } i = 1, n$$

$$\sum_{i=1}^{m} u_{i,j} \times s_{i,j} \leq 1 \quad \text{for } j = 1, m$$

The primary concern is scheduling the real time tasks satisfying the deadline compliance.

Applying ACO to TSP: Given a set of HMP and TS, the artificial ants stochastically assign each task to one processor until each of the tasks is assigned to some specific processor. Researchers introduce an artificial pheromone value $\tau_{i,j}$ with an edge between T_i and P_j that indicates the favourability of assigning the task T_i to the processor P_j. Initially $\tau_{i,j}$ is the same for all i, j. After each iteration, the pheromone value of each edge is reduced by a certain percentage to emulate the real-life behaviour of evaporation of pheromone count over time. The fraction ρ specifies the percentage of the τ value after evaporation. That is $1-\tau$ is the evaporation rate. Researchers use n_a artificial ants. Each ant behaves as follows: from a node i in TS an ant choose a node j in HMP with a probability given by:

$$P_{(i,j)} = \frac{\tau_{(i,j)}}{\sum_{j=1}^{m} \tau_{(i,j)}}$$

After all the tasks are considered and scheduled by an ant, the feasibility of the schedule is verified using the utilisation of individual processors and the completion time of each task. If any processor's utilisation exceeds 1.0 or any one of the task's completion time exceeds its deadline, that schedule is said to be infeasible and is ignored. This procedure is repeated for all n_a ants. The quality q of a feasible schedule S generated by an ant is computed by considering the total utilisation of all the processors and deadline compliance of the tasks. This quality is used in the pheromone update for the next iteration. This is given by:

$$\tau_{i,j} = \rho \times \tau_{i,j} + q(S)$$

if T_i is assigned to P_j in the schedule:

$$S = \rho \times \tau_{i,j}, \text{ otherwise}$$

Experimental illustration: Researchers have used FCFS algorithm for comparing the metrics used for the scheduling problem. FCFS assigns tasks serially to the processors 1 to m, ensuring that the deadline constraints are satisfied. If any task cannot be assigned to a processor because of constraint violation, it is considered for scheduling on the next processor.

For ACO implementation, researchers have considered ρ as 0.7 and the utilization matrix is generated randomly. The same utilisation matrix is used for all the trials of ACO and the FCFS algorithm. The parameters considered in the experiment are the utilisation of each processor, the average waiting time of all the tasks and the time taken for generating a feasible schedule. For each problem instance, ten trials are run for ACO and the average values of the parameters are taken which are then compared with those of the FCFS scheduling algorithm and the results are tabulated.

In ACO, the iterations continue till the number of iterations exceed a particular predefined value or the standard deviation of the quality of the solutions obtained by the ants is less than a small threshold value. This value is computed based on the quality of the solutions obtained by the ants in the first iteration. If the number of iterations exceeds a threshold value, it is assumed that a feasible schedule is not obtained. This is basically to ensure that a schedule is arrived at within a reasonable amount of time.

If all the ants come up with the same schedule, the standard deviation is zero. If the standard deviation is non-zero then it means we stop the iterations when all the ants come up with schedules that approximately have the same quality. In that case the schedule with the best quality is chosen as the approximately best schedule.

Procedure

Real time tasks scheduling algorithm: Do while ((solution is not converged) and (reasonable amount of time not elapsed)). Update the pheromone based on the quality of each feasible schedule generated in the previous iteration:

```
    Generate the τ matrix for the current
iteration
    For each ant k
        For each task i
    Select the processor stochastically
    using the τ matrix
    If the schedule obtained by an ant is
    feasible (individual processor
    utilisation and deadline compliance of
    tasks are satisfied) then compute its quality.
    For infeasible schedules the quality is zero.
    Calculate the standard deviation using
    the quality of all the schedules.
End while
```

Repeat the above procedure until the standard deviation is less than a threshold value or the number of iterations exceed a particular predefined value.

If the number of iterations exceeds a particular predefined value, the solution did not converge and an optimal schedule is not obtained. When the solution has converged, the parameters are computed based on the schedule which has the best quality. This schedule is assumed to be the optimal among the approximate schedules generated.

RESULTS AND DISCUSSION

A scheduling algorithm based on ACO is implemented and the algorithm is run for 6 problem instances with the number of processors as 8 and number of tasks as 80, 90, 100, 110, 120 and 130. The number of ants used for ACO is 30 and $\rho = 0.7$. Ten trials are done for each problem instance with ACO and the average waiting time of tasks and utilisation of each processor are obtained. For each problem instance, FCFS is run with the utilisation matrix used by ACO algorithm. The wait time of tasks and utilisation of each processor are computed and compared with that of ACO and the results are tabulated.

Table 1 and 2 show the individual processor utilisation obtained for ten trial runs for tasks sets of size 90 and 130, respectively. Similar results are generated for the other tasks sets but are not shown here. The average utilisation of each of the eight processors using ACO is shown in Table 3 for all problem instances. Table 4 shows the same for the FCFS algorithm. It is seen from that table that the number of processors not utilised by the FCFS scheduling algorithm varies from 1-2. But all the

Table 1: Individual processor utilisation for a task set of size = 90 using ACO

Processors	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilisation
1	0.499137	0.64661	0.38277	0.38255	0.48560	0.50963	0.47233	0.51385	0.44114	0.38914	0.47227
2	0.530376	0.68620	0.61224	0.79270	0.64861	0.62582	0.33457	0.65957	0.61216	0.35926	0.58615
3	0.687221	0.63864	0.74814	0.57912	0.56462	0.42453	0.56049	0.27815	0.58411	0.50260	0.55676
4	0.511604	0.56982	0.65982	0.57370	0.57227	0.56826	0.65407	0.48141	0.43001	0.45902	0.54800
5	0.518084	0.51335	0.48217	0.68394	0.46059	0.35717	0.51732	0.78031	0.53922	0.65521	0.55074
6	0.433095	0.57115	0.34472	0.29863	0.49355	0.46606	0.54286	0.49407	0.37460	0.52977	0.45485
7	0.449340	0.32600	0.48832	0.42434	0.55194	0.46422	0.66832	0.61369	0.58520	0.60854	0.51799
8	0.520985	0.40010	0.37716	0.40032	0.39607	0.63052	0.47966	0.46378	0.58619	0.49857	0.47534

Table 2: Individual processor utilisation for a task set of size = 130 using ACO

Processors	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilisation
1	0.877338	0.78992	0.75804	0.71280	0.59197	0.65574	0.85527	0.75010	0.67160	0.76106	0.74238
2	0.486135	0.56138	0.73880	0.85573	0.70382	0.76326	0.69843	0.79054	0.50839	0.59283	0.66993
3	0.650168	0.90577	0.62469	0.91373	0.88171	0.62762	0.54795	0.88272	0.98144	0.74009	0.77559
4	0.557843	0.75817	0.54691	0.50448	0.91741	0.92674	0.69059	0.68704	0.49127	0.67981	0.67603
5	0.771267	0.66110	0.60940	0.57826	0.62991	0.71926	0.46271	0.52919	0.87867	0.71246	0.65522
6	0.640102	0.66996	0.66475	0.62277	0.42644	0.67369	0.80178	0.72649	0.39194	0.59224	0.62102
7	0.618985	0.61915	0.80141	0.72528	0.68740	0.73737	0.40623	0.42827	0.66546	0.53512	0.62247
8	0.974411	0.45971	0.75174	0.45640	0.71756	0.38271	0.82244	0.84184	0.79904	0.85498	0.70608

Table 3: Average processor utilisation table for ACO for all 8 problem instances

Processors	Task = 80	Tasks = 90	Tasks = 100	Tasks = 110	Tasks = 120	Tasks = 130
1	0.5160821	0.472275	0.548295	0.718464	0.628331	0.742383
2	0.5152305	0.586150	0.513384	0.637610	0.629064	0.669933
3	0.4389294	0.556762	0.584743	0.713713	0.702465	0.775588
4	0.5055490	0.547998	0.585821	0.538903	0.692712	0.676026
5	0.5194804	0.550737	0.496787	0.483998	0.701493	0.655220
6	0.5253921	0.454850	0.537452	0.600773	0.730153	0.621016
7	0.5182347	0.517992	0.519209	0.608399	0.583008	0.622468
8	0.5511994	0.475335	0.496989	0.541249	0.664689	0.706084
Standard deviation	0.0321035	0.047774	0.035578	0.083545	0.049410	0.054937

Table 4: Average processor utilisation for FCFS for all 8 problem instances

Processors	Task = 80	Tasks = 90	Tasks = 100	Tasks = 110	Tasks = 120	Tasks = 130
1	0.763884	0.87006	0.94763	0.99912	0.99588	0.99841
2	0.744719	0.88575	0.91573	0.99725	0.9985	0.9974
3	0.733493	0.83417	0.91881	0.99846	0.99307	0.99523
4	0.751468	0.83428	0.91513	0.96537	0.99576	0.99909
5	0.697284	0.79901	0.84164	0.86478	0.99221	0.99784
6	0.172778	0.19321	0.03406	0.04814	0.63135	0.98962
7	0	0	0	0	0	0.04687
8	0	0	0	0	0	0
Standard deviation	0.3567608	0.409005	0.465022	0.493291	0.450413	0.450517

Table 5: Average waiting time of all the tasks for ACO for all the 8 problem instances

Tasks	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average waiting time (sec)
80	0.244334	0.21993	0.22593	0.20470	0.23855	0.23941	0.23892	0.22470	0.25213	0.24081	0.23294
90	0.244068	0.24460	0.23899	0.25011	0.22398	0.23839	0.27361	0.25813	0.24648	0.25857	0.24769
100	0.258541	0.24397	0.24297	0.26100	0.23408	0.25447	0.25569	0.25532	0.25617	0.27221	0.25344
110	0.287423	0.28358	0.29192	0.31200	0.29199	0.31200	0.30288	0.30619	0.29437	0.26223	0.29446
120	0.337754	0.35478	0.33835	0.30877	0.33908	0.30565	0.34154	0.30508	0.31409	0.33214	0.32772
130	0.335316	0.31542	0.31300	0.33083	0.34616	0.33407	0.31943	0.35693	0.34215	0.36003	0.33533

Table 6: Average time to generate a schedule using ACO for all the 8 problem instances

Tasks	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average scheduling time (sec)
80	1.318681	0.76923	1.09890	1.26374	1.53846	1.20879	1.15385	1.26374	1.37363	1.15385	1.21429
90	1.043956	1.04396	1.20879	1.09890	1.09890	1.04396	1.97802	0.93407	1.18132	1.81319	1.24451
100	1.428571	1.15385	1.70330	1.04396	1.15385	1.04396	1.20879	1.42857	1.04396	1.42857	1.26374
110	1.318681	1.48352	1.81319	1.09890	1.26374	1.09890	1.09890	1.04396	1.09890	1.64835	1.29670
120	1.483516	1.75824	1.09890	1.09890	1.15385	1.09890	1.15385	1.15385	1.15385	1.86813	1.30220
130	1.318681	1.20879	1.37363	1.42857	1.15385	1.26374	1.42857	1.37363	1.15385	1.37363	1.30769

Table 7: Comparison of the waiting time and the scheduling time of both algorithms

	Average waiting time of tasks (sec)		Average scheduling time of tasks (sec)	
Tasks	ACO	FCFS	ACO	FCFS
80	0.232940	0.391446	1.214286	0.769231
90	0.247693	0.403239	1.244505	0.824176
100	0.253442	0.440738	1.263736	0.934066
110	0.294457	0.513665	1.296703	0.934066
120	0.327721	0.488400	1.302198	0.934066
130	0.335333	0.487377	1.307692	0.989011

processors are utilised by the ACO algorithm. The last row of Table 3 and 4 gives the standard deviation of the utilisation across all the 8 processors for the respective algorithms. This substantiates the claim that the load is fairly shared among all the processors in ACO. Figure 1 and 2 pictorially depict this fact.

From Fig. 1, it is also observed that the total processor utilisation is more for FCFS for the problem instances for task sets of 90, 100, 110, 120 and 130. This is because the ACO explores different schedules for each iteration and comes up with the optimal schedule. Each iteration refines the schedule generated by the previous iteration.

Table 5 shows the average waiting time of all the tasks and Table 6, the average time taken for generating the schedule using ACO for all the eight problem instances. Table 7 gives a comparison of the twoparameters, average waiting time and scheduling time of both the algorithms. It is shown clearly that the average waiting time for the tasks is more in the case of FCFS. There is however, a marginal increase in the scheduling time in case of ACO. Figure 3 pictorially depict these facts.

A different approach to real time task scheduling on heterogeneous processors based on ACO is presented. The approach attempts to find a feasible task assignment with the objective of keeping all the processors more or

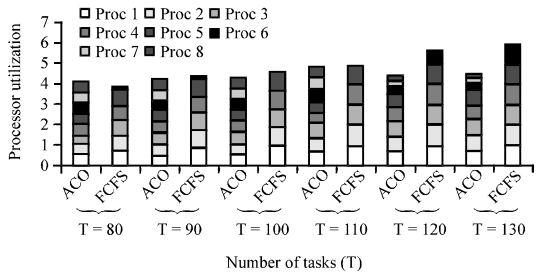

Fig. 1: Comparison of individual processor utilisation in both algorithms

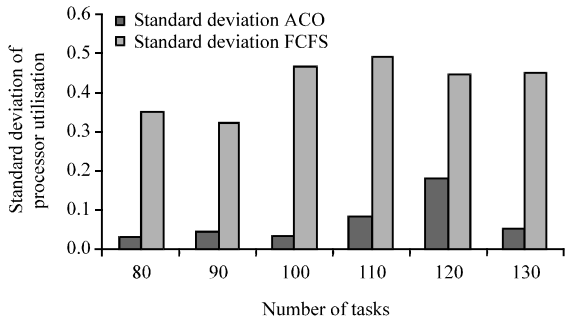

Fig. 2: Standard deviation of processor utilisation in both algorithms

less equally loaded. On comparison with the FCFS approach, the ACO Method balances the load fairly among the different processors. The average waiting time of the tasks is also found to be less than that of the FCFS algorithm.

Also, it is found that the total utilisation of the processors in ACO is less than that of the FCFS for 83% of the instances considered. It is noted that the instances are randomly generated. However, there is a slight increase in the scheduling time for the ACO algorithm.

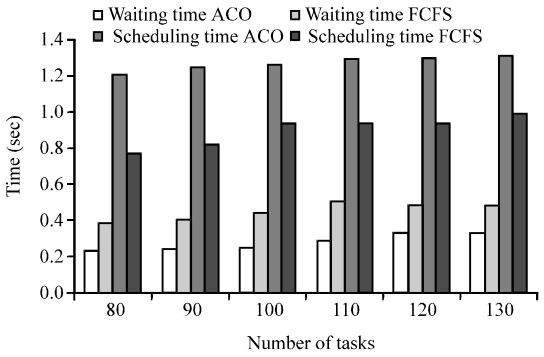

Fig. 3: Comparison of the waiting time and scheduling time of both algorithms

CONCLUSION

The results are tabulated and compared and it is found that ACO performs better than the FCFS with respect to the wait time. Although, the processor utilisation is more for some processors using FCFS algorithm, it is found that the load is better balanced among the processors in ACO. Also, it is found that the total utilisation of the processors in ACO is less than that of the FCFS for 83% of the instances considered. But there is a marginal increase in the time for arriving at a schedule in ACO compared to FCFS algorithm.

REFERENCES

Aguilar, J. and E. Gelenbe, 1997. Task assignment and transaction clustering heuristics for distributed systems. Inf. Sci., 97: 199-219.

Blum, C. and A. Roli, 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surveys, 35: 268-308.

Blum, C., 2005. Ant colony optimization: Introduction and recent trends. Physics Life Rev. J., 2: 353-373.

Braun, T.D., H.J. Siegel, N. Beck, L.L. Boloni and M. Maheswaran *et al*., 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput., 61: 810-837.

Cassavant, T.L. and J.G. Kuhl, 1998. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans. Software Eng., 14: 141-154.

Chen, H. and A.M.K. Cheng, 2005. Applying ant colony optimization to the partitioned scheduling problem for heterogeneous multiprocessors. ACM SIGBED Rev., 2: 11-14.

Chen, H., A.M.K. Cheng and K. Ying-Wei, 2011. Assigning real-time tasks to heterogeneous processors by applying ant colony optimization. J. Parallel Distrib. Comput., 71: 132-142.

Davis, R.I. and A. Burns, 2011. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv., Vol. 43. 10.1145/1978802.1978814.

Graham, R.L., E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, 1979. Optimization and approximation in deterministic sequencing and scheduling theory: A survey. Ann. Discrete Math., 5: 287-326.

Kang, J. and S. Park, 2001. Discrete optimization algorithms for the variable sized bin packing problem. Eur. J. Oper. Res., 147: 365-372.

Mao, J., 2010. Task scheduling of parallel programming systems using ant colony optimization. Proceedings of the 3rd International Symposium on Computer Science and Computational Technology, August 14-15, 2005, Jiaozuo, China, pp: 179-182.

Radulescu, A. and A.J.C. Van Gemund, 2002. Low-cost task scheduling for distributed-memory machines. IEEE Trans. Parallel Distrib. Syst., 13: 648-658.

Shih-Tang, L., C. Ruey-Maw, H. Yueh-Min and W. Chung-Lun, 2008. Multiprocessor system, scheduling with precedence and resource constraints using an enhanced ant colony system. Expert. Syst. Appl., 34: 2071-2081.

Taylor, M.B., J. Kim, J. Miller, D. Wentzlaff and F. Ghodrat *et al*., 2002. The raw microprocessor: A computational fabric for software circuits and general-purpose programs. IEEE Micro, 22: 25-35.

Umarani, S.G., V. Uma Maheswari, A. Shanthi and A. Siromoney, 2012a. A survey on real time task scheduling. Eur. J. Sci. Res., 69: 33-41.

Umarani, S.G., V. Uma Maheswari, A. Shanthi and A. Siromoney, 2012b. Task scheduling using ant colony optimization. J. Comput. Sci., 8: 1541-1546.