Task Scheduling Using Probabilistic Ant Colony Heuristics

Umarani Srikanth[1], Uma Maheswari[2], Shanthi Palaniswami[3], and Arul Siromoney[3]
[1]Computer Science and Engineering Department, S.A. Engineering College, India
[2]Department of Information Science and Technology, Anna University, India
[3]Department of Computer Science and Engineering, Anna University, India

Abstract: *The problem of determining whether a set of tasks can be assigned to a set of heterogeneous processors in general is NP-hard. Generating an efficient schedule of tasks for a given application is critical for achieving high performance in a heterogeneous computing environment. This paper presents a novel algorithm based on Ant Colony Optimization (ACO) for the scheduling problem. An attempt is made to arrive at a feasible schedule for a task set on heterogeneous processors ensuring fair load balancing across the processors within a reasonable amount of time. Three parameters: Average waiting time of tasks, utilization of individual processors and the scheduling time of tasks are computed. The results are compared with those of the First Come First Served (FCFS) algorithm and it is found that ACO performs better than FCFS with respect to the waiting time and individual processor utilization. On comparison with the FCFS approach, the ACO method balances the load fairly among the different processors with the standard deviation of processors utilization being 88.7% less than that of FCFS. The average waiting time of the tasks is also found to be 34.3% less than that of the FCFS algorithm. However, there is a 35.5% increase in the scheduling time for the ACO algorithm.*

Keywords: *ACO, tasks scheduling problem, processor utilization, heterogeneous processors.*

Received July 6, 2013; accepted September 19, 2013; published online August 22, 2015

1. Introduction

The heterogeneous computing platform meets the computational demands of diverse tasks. One of the key challenges of such heterogeneous processor systems is effective task scheduling [9]. The task scheduling problem is generally NP-Complete. Some of the parameters used for identifying a good schedule are reducing the make span, reducing the waiting time of the tasks and achieving other goals such as power reduction [7]. An efficient task scheduling avoids the situation in which some of the processors are overloaded while some others are idle.

Many real time applications are discrete optimization problems. For combinatorial optimization problems that are NP-hard, no polynomial time algorithm exists. This leads to computation time being too high for practical purposes. So, the development of approximate methods, in which the guarantee of finding optimal solutions is sacrificed for the sake of getting good solutions in a significantly reduced amount of time, has received more attention. Therefore, it seems reasonable looking for new computation paradigms that go beyond the limits of conventional computing.

Section 2 deals with the related work and section 3 discusses Ant Colony Optimization (ACO) based task scheduling algorithm. Section 4 deals with experimental illustration and section 5 show the conclusions.

2. Related Works

ACO has been formalized into a meta-heuristics for combinatorial optimization problems by Dorigo *et al*. [4, 5]. An approach based on ACO [1, 6] that explores different designs to determine an efficient hardware-software partitioning, to decide the task allocation and to establish the execution order of the tasks, dealing with different design constraints imposed by a reconfigurable heterogeneous MPSoC is proposed in [8]. The methodology determines the mapping and scheduling of the input application on the target architecture minimizing its overall execution time.

The work presented in [11] focuses on investigating multiprocessor system scheduling with precedence and resource constraints. A modified ACO approach called Dynamic Delay Ant Colony System (DDACS) is adopted for precedence and resource constraints multiprocessor scheduling problem. DDACS scheme provides an efficient method of finding the optimal schedule of the multi-constraints multiprocessor system. A two dimension matrix graph is adopted to represent the assigning jobs on processors. This graph is deployed to resolve the minimum make span schedule. The processors are homogeneous. An artificial immune system for heterogeneous multiprocessor scheduling with task duplication known as the Artificial Immune System with Duplication (AISD) is proposed in [10]. It first generates and refines a set of schedules using a modified clonal

selection algorithm and then improves the schedules with task duplication. The AISD algorithm schedules the tasks in a task graph via three carefully designed phases: Clonal selection, task duplication and ineffectual task removal.

An algorithm based on the ACO meta-heuristic is presented for solving the real time tasks scheduling in [2] using the task execution time and processor capacity. A local search heuristic is used in [3] in order to improve the assignment solution. The system assumes an arbitrary number of heterogeneous multiprocessors platforms with m pre-emptive processors. The task assignment problem addresses two objectives: The resource objective which searches a solution in which each of the task is assigned to a specific processor in such a way that the cumulative utilization of any processor does not exceed the utilization bound and the energy objective which minimizes the cumulative energy consumption of all the assigned tasks in a solution. Srikanth *et al*. [12] studied the task scheduling problem using ACO framework. The framework is used to generate schedules for independent tasks on heterogeneous processors. The ACO algorithm converges only when all the ants come up with the same schedule.

3. Materials and Methods

3.1. Ant Colony Optimization

ACO takes inspiration from the foraging behaviour of some ant species. These ants deposit pheromone on the ground in order to mark some favourable path that should be followed by other members of the colony. The above behaviour of real ants has inspired Ant System (AS), an algorithm in which a set of artificial ants cooperate for solving optimization problems by exchanging information via pheromones deposited on edges of a construction graph, whose model depends on the problem to be solved. A major advantage of ACO over other meta-heuristic algorithms is that the problem instance may change dynamically.

An algorithm based on ACO is proposed in this work. Here, an approximate solution is found out which allocates tasks to processors providing better load sharing among the processors in a reasonable amount of time.

3.2. Task Scheduling Problem

The characteristics of the processors and tasks assumed in the scheduling problem considered are given below:

1. Processor Characteristics:

- The processors are assumed to be heterogeneous.
- The heterogeneity of the processors is modelled by the varied proportional utilization of the same task on different processors.
- Many tasks can be scheduled on the same processor.

2. Tasks Characteristics:

- Tasks are assumed to be non-real time and independent.
- There are no precedence constraints among them.
- There is no inter-task communication.
- The utilization of a processor by a task is known apriori and it does not change with time.
- All the tasks are assumed to arrive at the same instant at 0 time units.
- There is no task migration.

Let the set $P=\{P_1, P_2, \ldots, P_m\}$ denote m arbitrary heterogeneous processors with each P_j running at variable speed. Let the set $T=\{T_1, T_2, \ldots, T_n\}$ denote n time tasks. Each task T_i is characterized by u_{ij} where u_{ij} is the worst case execution time of the task T_i on processor P_j. The worst case execution time of the tasks on the processors is given in a utilization matrix U. The number of rows of the matrix U equals the number of tasks and the number of columns equals the number of processors. In other words, the order of the matrix is $n \times m$. The elements of U are real numbers in the range (0, 1) and they specify the maximum proportional time of the processors used by the tasks. In other words, a typical entry u_{ij} denotes the fraction of the computing capacity of P_j required executing T_i. u_{ij} is also referred as utilization of T_i on P_j. A sample utilization matrix is shown in Table 1.

Table 1. Utilization matrix with 4 tasks and 3 processors.

	P_1	P_2	P_3
T_1	u_{11}	u_{12}	u_{13}
T_2	u_{21}	u_{22}	u_{23}
T_3	u_{31}	u_{32}	u_{33}
T_4	u_{41}	u_{42}	u_{43}

The task scheduling problem can be formally stated as follows: Given T and P, determine a schedule that assigns each of the tasks in T to a specific processor in P, in such a way that the cumulative utilization of the tasks on any processor is not greater than the utilization bound of that processor which is 1.0. This problem is represented by a bipartite graph with the two classes of nodes, T and P. A task is mapped to a T node and a processor is mapped to a P node. The graph is a directed graph with the edges leaving from the set of tasks nodes to the set of processor nodes. There is a directed edge from a T node to a P node, if and only if the corresponding task can be assigned to that processor, without exceeding its available computing capacity.

A schedule can be represented as an $n \times m$ binary matrix. A typical entry of this matrix is denoted as s_{ij}. The entry $s_{ij}= 1$ if task T_i is scheduled on processor P_j. It is noted that there are no two 1's in the same row as 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 corresponding processor. But the proportional utilization of all the tasks on a

processor should not exceed 1, as shown in Equations 1 and 2:

$$\sum_{j=1}^{m} s_{ij} = 1 \quad for\ i = 1,\ 2,\ \ldots,\ n \tag{1}$$

$$\sum_{i=1}^{n} u_{ij} * s_{ij} \leq 1\ for\ j = 1,\ 2,\ \ldots,\ m \tag{2}$$

3.3. Task Scheduling Algorithm

Given a set of T and P, the artificial ant stochastically assigns each task to one processor until each of the tasks is assigned to some specific processor. An artificial pheromone value τ_{ij} is introduced with an edge between T_i and P_j that indicates the favorability of assigning the task T_i to the processor P_j. Initially τ_{ij}= 0.9 and 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 behavior of evaporation of pheromone count over time. The fraction ρ= 0.7 specifies the percentage of the τ value after evaporation. i.e., 0.3 is the evaporation rate. The number of artificial ants is 30. Each ant behaves as follows: From a node i in T an ant choose a node j in P with a probability as shown in the Equation 3.

$$p(i,j) = \frac{\tau(i,j)}{\sum_{j=1}^{m} \tau(i,j)} \tag{3}$$

After all the tasks are considered for scheduling by an ant, the feasibility of the schedule is verified using the total utilization of each individual processor. If any processor's utilization exceeds 1.0, that schedule is infeasible. This procedure is repeated for all the ants. The quality q of a feasible schedule S generated by an ant in each iteration is computed by considering the total utilization of all the processors in that schedule. This quality is used in the pheromone update of the next iteration by all the ants. To begin with, the evaporation of the pheromones is taken care of, given in Equation 4.

$$\tau_{ij} = \rho * \tag{4}$$

Then, pheromone updating is done by all the ants that come up with feasible schedules in the following manner as shown in the Equation 5.

$$\tau_{ij} = \tau_{ij} + q(S)\ if\ \ T_i\ is\ assigned\ to\ P_j \\ in\ the\ schedule\ S\ of\ \ an\ ant_k. \tag{5}$$

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 which 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 that the ants have come up with schedules that approximately have the same quality and the iterations are terminated. i.e., the algorithm is assumed to converge. In that case the schedule with the best quality is chosen as the approximate best schedule.

The schedule with the maximum quality is chosen as the best approximate schedule and its parameters are computed.

Chen and Cheng [2] have used the following heuristics: A task is assigned to a processor on which it has the least execution time; and task assignment is done to minimize the laxity in the computing capacity of processor. Chen *et al.* [3] have used Max-Min AS (MMAS) for updating the pheromone trails. They have used local search heuristics that starts off with an initial assignment solution and then tries to find better solutions using the following neighborhood operations: Remove a task from a processor and then assign it to a different processor called 1-Opt; and remove two tasks from two processors and then assign each of them to the processor that the other task is assigned to initially, called 2-Exchange. This algorithm is designed to optimize assignment solutions in terms of both resource utilization and energy consumption.

In our work, we have used a model which is a combination of AS and ACS. We have not considered energy consumption. We have used the following heuristics: Negligible amount of pheromones deposited by ants is ignored to avoid the scattering of the solution trails generated by the ants; priority is given to arrive at feasible schedules within a reasonable amount of time; and approximation of the optimal schedule is admissible, which is decided by the quality of the solution based on the total utilization of the processors.

4. Results

Algorithm 1 is run for five problem sets with the number of tasks as 90, 100, 110, 120 and 130, number of processors as eight and the number of ants as 30. For the experiment, the utilization matrix is generated randomly for each problem set. The same utilization matrix is used for all the trials and the First Come First Served (FCFS) algorithm. The parameters considered in the experiment are the utilization of each processor, the average waiting time of all the tasks and the time taken for generating a feasible schedule. For each problem set, 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.

Algorithm 1: Task scheduling.

```
While ((iteration_no < max_iteration) and (std_dev (qual[] > threshold))
{
  for (k=1 to num_ants)
    {
      for (i=1 to num_tasks)
```

```
        {
          # select a processor stochastically
          # using τ matrix
            sched[k][i]= stoch_val(τ)
        }
      # compute quality of each schedule
      qual[k]= calc_qual(sched[k][])
    }
  # update τ using quality of each schedule
  τ= update(τ, qual[])
}
```

5. Discussions

Table 2 presents the individual processor utilization and standard deviation of each of the eight processors using ACO algorithm. Table 3 shows the same parameters for the FCFS algorithm. It is observed from Table 3 that the number of processors not utilized by the FCFS scheduling algorithm varies from 1 to 4, whereas ACO uniformly uses all the processors. Figure 1 pictorially represents the individual processor utilization using ACO and FCFS algorithms for these five problem sets. Hence, it is proved that the ACO algorithm results in fair processor utilization than the FCFS algorithm.

Table 2. Individual processor utilization for ACO for 5 problem sets.

Processor	Tasks=90	Tasks=100	Tasks=110	Tasks=120	Tasks=130
1	0.676442	0.685348	0.720643	0.604547	0.707257
2	0.703828	0.749016	0.650884	0.771987	0.759952
3	0.513967	0.508955	0.750640	0.739138	0.704041
4	0.562577	0.580108	0.649285	0.698303	0.690416
5	0.686510	0.709957	0.680012	0.664643	0.728576
6	0.603779	0.682561	0.681009	0.755928	0.707009
7	0.545770	0.683775	0.647264	0.678312	0.650390
8	0.575534	0.728833	0.677148	0.744772	0.730427
Standard Deviation	0.071592	0.080867	0.036797	0.056398	0.032173

Table 3. Individual processor utilization for FCFS for 5 problem sets.

Processor	Tasks=90	Tasks=100	Tasks=110	Tasks=120	Tasks=130
1	0.999140	0.994333	0.997119	0.999652	0.999192
2	0.999302	0.995411	0.998299	0.999039	0.999836
3	0.993370	0.999633	0.998961	0.999928	0.998118
4	0.964096	0.994504	0.999994	0.999764	0.999814
5	0.000000	0.712981	0.978659	0.993507	0.995617
6	0.000000	0.000000	0.216666	0.403263	0.971887
7	0.000000	0.000000	0.000000	0.000000	0.001559
8	0.000000	0.000000	0.000000	0.000000	0.000000
Standard Deviation	0.528745	0.495497	0.482090	0.464140	0.459901

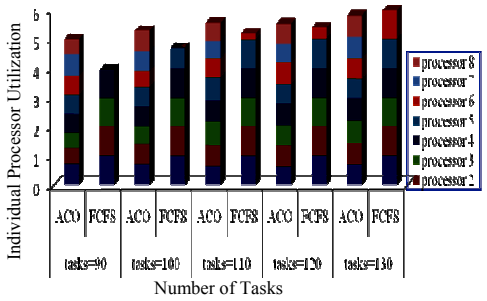

Figure 1. Comparison of the individual processor utilization using ACO and FCFS algorithms.

On comparison with the FCFS approach, the ACO method balances the load fairly among the different processors with the standard deviation of processor utilization being 88.7% less than that of FCFS which is proved in Figure 2. Table 4 provides a comparison of the parameters waiting time of tasks and scheduling time of tasks for ACO and FCFS algorithms. Figure 3 shows this comparison pictorially. It exhibits the fact that the average waiting time of the tasks is found to be 34.3% less than that of the FCFS algorithm. The tabulated experimental results in Table 4 prove our claim. However, there is a 35.5% increase in the scheduling time for the ACO.

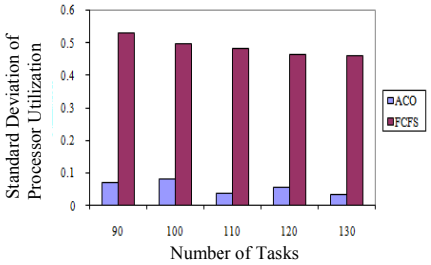

Figure 2. Comparison of standard deviation of the processor utilization using ACO and FCFS algorithms.

Table 4. Comparison of the waiting and scheduling time of tasks using ACO and FCFS algorithms.

Number of Tasks	Average Waiting Time of Tasks (Secs)		Average Scheduling Time of Tasks (Secs)	
	ACO	FCFS	ACO	FCFS
90	0.304094	0.507763	0.335165	0.274725
100	0.328272	0.464974	0.373626	0.302198
110	0.339830	0.526415	0.401099	0.329670
120	0.345808	0.512628	0.456044	0.329670
130	0.348974	0.528829	0.525900	0.329670

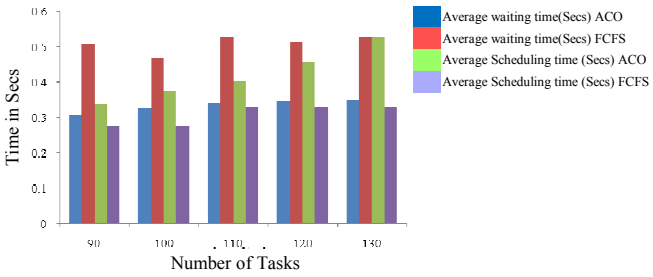

Figure 3. Comparison of waiting time and Schedule time of tasks using ACO and FCFS algorithms.

6. Conclusions

A variant of the ACO algorithm for the Task Scheduling Problem is given and the implementation results are presented. The algorithm finds a feasible task assignment with the objectives of keeping all the processors fairly loaded and obtaining the schedule within a reasonable amount of time. The parameters used for study are the average waiting time of the tasks, individual processor utilization and the time taken to arrive at a schedule. A comparative study is made with the FCFS scheduling algorithm and it is found that ACO performs better than the FCFS with respect to the waiting time and processor load balancing. But ACO method marginally takes more time to come up with a schedule compared to FCFS.

References

[1] Blum C., "Ant Colony Optimization: Introduction and Recent Trends," *Physics of Life Reviews*, vol. 2, no. 4, pp. 353-373, 2005.

[2] Chen H. and Cheng A., "Applying Ant Colony Optimization to the Partitioned Scheduling Problem for Heterogeneous Multiprocessors," *ACM Special Interest Group on Embedded Systems Review*, vol. 2, no. 2, pp. 1-14, 2005.

[3] Chen H., Cheng A., and Kuo Y., "Assigning Real Time Tasks to Heterogeneous Processors by Applying Ant Colony Optimization," *Journal of Parallel and Distributed Computing*, vol. 7, no. 11, pp. 32-42, 2011.

[4] Dorigo M. and Blum C., "Ant Colony Optimization Theory: A Survey," *Theoretical Computer Science*, vol. 344, no. 2, pp. 243-278, 2005.

[5] Dorigo M., Caro G., and Gambardella L., "Ant Algorithms for Discrete Optimization," *the Journal of Artificial Life*, vol. 5, no. 2, pp. 137-172, 1999.

[6] Dorigo M., Maniezzo V., and Colorni A., "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems*, *Man and Cybernetics*, vol. 26, no. 1, pp. 29-41, 1996.

[7] Elhossini A., Huissman J., Debowski B., Areibi S., and Dony R., "Efficient Scheduling Methodology for Heterogeneous Multi-core Processor Systems," *in Proceedings of the International Conference on Microelectronics*, Cairo, Egypt, pp. 475-478, 2010

[8] Ferrandi F., Pilato C., Tumeo A., and Sciuto D., "Mapping and Scheduling of Parallel C Applications with Ant Colony Optimization onto Heterogeneous Reconfigurable MPSoCs," *in Proceedings of 15th Asia and South Pacific Design Automation Conference*, Taipei, pp. 799-804, 2010.

[9] Ijaz S., Munir E., Anwar W., and Nasir W., "Efficient Scheduling Strategy for Task Graphs in Heterogeneous Computing Environment," *the International Arab Journal of Information Technology*, vol. 10, no. 5, pp. 486-492, 2013.

[10] Lee Y. and Zomaya A., "An Artificial Immune System for Heterogeneous Multiprocessor Scheduling with Task Duplication," *in Proceedings of IEEE International Parallel and Distributed Processing Symposium*, Long Beach, USA, pp. 1-8, 2007.

[11] Lo S., Chen R., Huang Y., and Wu C., "Multiprocessor System Scheduling with Precedence and Resource Constraints using an Enhanced Ant Colony System," *Expert Systems with Applications*, vol. 34, no. 3, pp. 2071-2081, 2008.

[12] Srikanth U., Maheswari U., Shanthi P., and Siromoney A., "Task Scheduling using Ant Colony Optimization," *the Journal of Computer Science*, vol. 8, no. 8, pp. 1541-1546, 2012.

Umarani Srikanth is currently working as an Associate Professor in PG Studies in Engineering Department, S.A. Engineering College, India. She has 18 years of teaching experience. Her areas of interests include compilers, theory of computation, data structures and soft computing.

Uma Maheswari is currently working as an Associate Professor in the Department of Information Science and Technology, Anna University, India. She has over 26 years of teaching experience. She has several publications and has served on several programme committees as reviewer. Her main area of research is data structures and algorithms, compilers, rough set theory, operating systems and soft computing.

Shanthi Palaniswami is currently working as an Associate Professor at Department of Computer Science and Engineering, Anna University, India. She has over 25 years of teaching experience. Her areas of interest include computer architecture, microprocessors, machine learning, evolvable hardware and soft computing.

Arul Siromoney is currently working as a Professor in the Department of Computer Science and Engineering, Anna University, India. He has over 22 years of teaching experience and 7 years of Industry experience. He has several publications and has served on several programme committees as reviewer. His main area of research is rough set theory and data mining. his interests include mobile computing, operating systems and software systems.