Indian Journal of Science and Technology, *Vol 8(S7), 33–42, April 2015*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Task Scheduling Model

G. Umarani Srikanth[1*], V. Uma Maheswari[2], A. P. Shanthi[3] and Arul Siromoney[4]

[1]S. A. Engineering College, Chennai, India; gmurani@saec.ac.in
[2]Department of Information Science and Technology, Anna University, Chennai, India
[3, 4]Department of Computer Science and Engineering, Anna University, Chennai, India

Abstract

To design and implement a task scheduling model which predicts a schedule for a new task set without actually running a task scheduling algorithm. Generating an optimal schedule of tasks for an application is critical for obtaining high performance in a heterogeneous computing environment and it is a hard problem. This work attempts to optimize on the scheduling time by designing a task scheduling model. The task scheduling algorithm used in this work is based on ACO, a swarm intelligence model. The prediction is done after the training phase of the model. The model is validated by comparing the predicted schedule with the actual schedule obtained by running the ACO scheduling algorithm on the new task set. The parameters used for comparison are waiting time of tasks, average processor utilization and the scheduling time. The predicted schedule is comparable to the actual schedule with respect to waiting time of tasks and average processor utilization. The scheduling time is significantly reduced and the reduction in the scheduling time increases with the increase in the task set size.

Keywords: ACO (Ant Colony Optimization), Ant Systems, Clustering, Heterogeneous Multiprocessors, Optimization Techniques, Task Scheduling

1. Introduction

The heterogeneous architecture[1] meets the computational demands of large number of emerging applications. One of the key challenges of such heterogeneous processor systems is effective tasks scheduling. The problem of scheduling tasks to processing units has a major impact on the performance of a system. The task scheduling problem[2] deals with mapping each task of the application onto the available processors in order to optimize certain parameters like the length of the makespan, utilization of the processor, cache performance, power consumption, workload balance, scheduling penalties and switch cost. The scheduling problem is a computationally hard problem and researchers have been working on applying non-conventional computing paradigms[11] for solving this problem.

Nature can be used as a source of inspiration for the development of new techniques for solving complex computational and engineering problems[3]. Among all natural computing approaches, bio-inspired systems[4] apply more heuristics or meta-heuristics to problems that could not be satisfactorily resolved by other more traditional techniques. The solutions they produce are approximate rather than precise, involve randomness for fault tolerance and unplanned change and provide probabilistic decision making. ACO is a swarm intelligence model[13] motivated by the foraging behaviour of real ants. The core of this behaviour is the indirect communication between the ants by their pheromone trails that they deposit on their way to search for food sources. A major advantage of ACO is that it can be adapted to the changing problem instances dynamically.

An approach based on ACO[5,14] 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

**Author for correspondence*

MPSoC was proposed by Ferrandi[6]. The methodology determines the mapping and scheduling of the input application on the target architecture minimizing its overall execution time.

An artificial immune system for heterogeneous multiprocessor scheduling with task duplication known as the Artificial Immune System with Duplication (AISD) was proposed by Lee[7]. 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 tasks in a task graph via three carefully designed phases: clonal selection, task duplication and ineffectual task removal. A heuristic algorithm for tasks scheduling based on an evolutionary method which embeds a fast technique called Elitism stepping to decrease the computation time for finding an optimal schedule was proposed by Rahmani[8].

An approach for mapping and scheduling task graphs for heterogeneous hardware/software computing systems[12] using heuristic search was proposed by Lam[9]. Two techniques were proposed to enhance the speedup of mapping/scheduling solutions: one is an integrated technique combining task clustering, mapping, and scheduling, and a multiple neighbourhood function strategy which starts with a feasible solution and attempts to improve it by searching its neighbours. This process is repeated until a local optimum or the termination condition is reached and search ends when neither neighbourhood function produces better solutions.

2. Task Scheduling Problem

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

2.1 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.

2.2 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.
- There is no task migration.

Let the set $P = \{P_1, P_2,\ldots, P_m\}$ denote m heterogeneous processors with each P_j running at variable speed. $T = \{T_1, T_2,\ldots, T_n\}$be the set that denote n 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 and 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^*m and the elements of U are real numbers in the range (0, 1). They specify the maximum proportional time of the processors used by the tasks. A sample utilization matrix is shown in Table 1.

Task Scheduling Problem can now 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 no 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 task 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 n^*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 specific processor. But the proportional utilization of all the tasks on a processor should not exceed 1 as shown by the equations (1) and (2).

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}

In other words,

$$\sum_{j=1}^{m} S_{ij} = 1 \text{ for } i = 1, 2, \ldots, n \quad (1)$$

$$\sum_{i=1}^{n} u_{ij} * S_{ij} \leq 1 \text{ for } j = 1, 2, \ldots, m \quad (2)$$

3. Task Scheduling Model using ACO

Ant System (AS) is the first ACO algorithm proposed in the literature[10]. Its main characteristic is that, at each iteration, the pheromone values are updated by all the *m* ants that have built a solution in the iteration itself. The algorithm is said to converge when all the ants arrive at the same solution and the iteration is terminated.

Given the sets T and P, an 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. 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 pheromone τ_{ij}, associated with the edge assigning task *i* to the processor *j*, is updated as given in Equation (3).

$$\tau_{ij} = (1-\rho).\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \quad (3)$$

where, ρ is the evaporation rate, *m* is the number of ants, and $\Delta\tau_{ij}^{k}$ is the quantity of pheromone laid on edge (i, j) by ant *k*. When ant *k* is trying to schedule task *i* and has so far constructed the partial solution s^p, the probability of choosing processor *j* is given by the equation (4).

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}}{\sum_{c_{il} \in N(s^p)} \tau_{il}} & \text{if } C_{ij} \in N(s^p) \\ 0 \text{ otherwise} \end{cases} \quad (4)$$

where, $N(s^p)$ is the set of feasible components, that is edges (*i*, *l*) where, *l* is a processor that can be assigned to task *i* by the ant *k*.

The feasibility of the schedule is verified after all the tasks are considered for scheduling by an ant. If the total utilization of a processor exceeds 1.0, the schedule generated by that ant is infeasible. If the schedule is feasible, its quality *q* 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 this ant indicated by the equation (5).

$$\Delta\tau_{ij}^{k} = \begin{cases} Q * u_k \text{ if ant } K \text{ is assigned task } i \text{ to processor } j \\ 0 \text{ otherwise} \end{cases} \quad (5)$$

where, Q is a constant, and u_k is the utilization of all the processors in the schedule generated by ant *k*. This procedure is repeated for all the ants. Then at the start of the next iteration, the pheromone update of the edges is done using the equation (3).

The ACO algorithm is adapted in this work to incorporate the following features:

- The algorithm should take only a reasonable amount of time to come up with a decisive answer, generating a feasible schedule if possible or indicating that no such schedule exists.
- Negligible amount of pheromones are not used for updation by an ant to avoid scattering of the search by the ants in that iteration.

To ensure the first feature, both a predefined upper bound for the number of iterations as well as an admissible deviation of quality between the multiple schedules generated by the ants are used and the second feature is implemented by using a simple rounding off procedure. The admissible deviation of quality between the multiple schedules is chosen heuristically to fit the experimental data. The ACO algorithm used in the proposed model is given below:

3.1 Procedure: Task Scheduling Algorithm

do while ((number of iterations is less than a predefined value) and (standard deviation of the quality of the schedules of all the ants is greater than a threshold))

```
{
  for each ant k
  {
    for each task i
    {
      Select the processor stochastically using the τ
        matrix
    }
```

If the schedule is feasible, compute its quality.
}
Update the pheromone based on the quality of each feasible schedule
Generate the τ matrix for the next iteration
}
end while

Thus 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. The initial value of the threshold is computed based on the quality of the solutions obtained by the ants in the first iteration. If the process terminates because the number of iterations exceeds a threshold value, it is assumed that a feasible schedule is not obtained. This ensures that if at all a schedule is arrived at, it is done so within a reasonable amount of time. But, if the process terminates because the standard deviation is below the threshold value, then it means that the ants have come up with schedules that approximately have the same quality. If the standard deviation is zero, it means that all the ants come up with the same schedule. The schedule with the maximum quality, when the algorithms terminates, is chosen as the best approximate schedule.

4. Task Scheduling Model

Given a set of tasks and a set of processors, this model predicts how the given set of tasks can be scheduled on the set of processors. Initially, a task set model that captures the features of a set of tasks is defined. Then two phases, namely a training phase and a testing phase are used. In the training phase, a sample set of task sets are used to generate a task set model. In the test phase, when a new task set arrives, this is fitted into the model and an appropriate schedule is predicted.

4.1 Task Set Model

The task set $T = \{T_1, T_2, \ldots, T_n\}$, the processor set $P = \{P_1, P_2, \ldots, P_m\}$ and the utilization matrix U are considered. Each task is modeled as a point in the *m*-dimensional space, where, each coordinate specifies the proportional utilization of the corresponding processor by that task. The features of a task set T of size *n* is obtained by clustering them using the Euclidean distance in the *m*-dimensional space, as the distance metric. Let $\{C_1, C_2, C_3, \ldots, C_k\}$ be the *k* clusters obtained after the convergence of the clustering algorithm and let $c_1, c_2, c_3, \ldots c_k$ be the respective cluster heads. A task T_i belongs to a cluster C_r, with $1 \le r \le k$, if and only if for $|T_i - c_r| < |T_i - c_s|$ for $1 \le s \le k$ and $s \ne r$. In other words, the task T_i is closer to the cluster head c_r in the *m*-dimensional space, when compared to the other cluster heads. Tasks that belong to a cluster C_r are represented by the cluster head c_r of C_r. Thus the task set T is now modeled by the k-cluster heads $c_1, c_2, c_3, \ldots, c_k$.

4.2 Training Phase

A problem set with a given number of tasks, say *n*, is considered. For each problem set, many different problem instances, say *p*, are used in the training phase. The different instances are obtained by generating different utilization matrices randomly. Clustering technique is applied on each problem instance to come up with *k* clusters. Thus, *p* problem instances result in *p* sets of cluster heads, c_{ij} for $1 \le j \le k$ and $1 \le i \le p$.

4.3 Testing Phase

When a new task set of the same size $R = \{R_1, R_2, \ldots, R_n\}$ arrives, its features are obtained by applying clustering again and this results in R being modeled by the *k* cluster heads, $h_1, h_2, \ldots, h_k$. The distance of this point $(h_1, h_2, \ldots, h_k)$ from the other *p* points $(c_{11}, c_{12}, \ldots, c_{1k})$, $(c_{21}, c_{22} \ldots, c_{2k})$, …, $(c_{p1}, c_{p2}, \ldots, c_{pk})$ in the *k*-dimensional space is computed and the closest point $(c_{q1}, c_{q2}, \ldots, c_{qk})$ is obtained. The schedule of that task set $T_q = \{T_{q1}, T_{q2}, \ldots, T_{qk}\}$ is used as the schedule for the new task set R.

5. Experimental Illustration

The artificial pheromone value τ_{ij} associated with the edge between T_i and P_j indicates the favorability of assigning the task T_i to the processor P_j. Initially $\tau_{ij} = 0.9$ 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 $\rho = 0.7$ specifies the percentage of the τ value after evaporation. (ie) 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 chooses a node *j* in P with a probability given by equation (6).

$$p(i,j) = \frac{\tau(i,j)}{\sum_{j=1}^{m} \tau(i,j)} \qquad (6)$$

After all the tasks are considered and scheduled by an ant, the feasibility of the schedule is verified using the utilization value of individual processors. 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.

A task set of size 100 is initially used in the experiment. Five different instances of this task set are obtained by randomly generating five different utilization matrices. The parameters considered in this experiment are utilization of each processor, waiting time of all the tasks and time taken to schedule all the tasks. For each instance of the problem set, ten trials are run using the ACO scheduling algorithm and the average values of the parameters are taken. Table 2–6 give the individual processor utilization for a task set of size 100 for all the five instances respectively. Table 7 and Table 8 show the waiting time of the tasks and time taken to generate the schedule in all the ten trials of each of the five problem instances

Table 2. Individual processor utilization of a task set of size = 100 first instance

Rocessor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.413703	0.527835	0.759492	0.368078	0.818287	0.531584	0.656709	0.556856	0.550655	0.60811	0.579131
2	0.465838	0.716416	0.606374	0.361955	0.414037	0.845718	0.79616	0.715078	0.690003	0.961006	0.657259
3	0.346468	0.764075	0.563849	0.754195	0.76058	0.321832	0.352593	0.33897	0.670561	0.620627	0.549375
4	0.800592	0.661120	0.726056	0.638663	0.771742	0.673483	0.763649	0.935698	0.746938	0.553215	0.727116
5	0.939105	0.666653	0.767261	0.793756	0.54394	0.350056	0.542122	0.561706	0.345391	0.440268	0.595026
6	0.699677	0.648393	0.479349	0.993661	0.631545	0.620272	0.674913	0.928157	0.585322	0.556859	0.681815
7	0.659984	0.499031	0.570098	0.755576	0.703374	0.832122	0.761722	0.654445	0.937608	0.603846	0.697781
8	0.933425	0.538215	0.578458	0.554701	0.510611	0.922294	0.61252	0.401758	0.718417	0.806388	0.657679

Table 3. Individual processor utilization of a task set of size = 100 second instance

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.731105	0.798559	0.734617	0.750054	0.726444	0.357452	0.516309	0.730192	0.50033	0.753086	0.659815
2	0.447411	0.715992	0.570651	0.359584	0.265511	0.924930	0.843706	0.749056	0.796321	0.737841	0.641101
3	0.572643	0.585837	0.719415	0.932287	0.627847	0.729305	0.573040	0.843833	0.888720	0.814273	0.728720
4	0.546675	0.722299	0.669631	0.304413	0.749396	0.997606	0.626001	0.520975	0.721247	0.499652	0.635790
5	0.713665	0.730978	0.563915	0.941972	0.874157	0.239742	0.631290	0.644438	0.490126	0.747638	0.657792
6	0.792016	0.253331	0.752405	0.673835	0.836839	0.533675	0.604147	0.412564	0.461671	0.501015	0.582150
7	0.815005	0.647274	0.622974	0.721718	0.641724	0.536861	0.912519	0.727163	0.694161	0.700773	0.702017
8	0.546834	0.640900	0.492352	0.461703	0.588611	0.773347	0.538767	0.715569	0.595334	0.614859	0.596828

Table 4. Individual processor utilization of a task set of size = 100 third instance

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.944845	0.867274	0.999289	0.803829	0.474314	0.936325	0.836471	0.912032	0.579059	0.861971	0.821541
2	0.804091	0.598781	0.485220	0.768768	0.713901	0.584034	0.704375	0.534443	0.482748	0.747376	0.642374
3	0.678477	0.643900	0.737542	0.855079	0.754509	0.471878	0.681918	0.935404	0.605228	0.565322	0.692926
4	0.35286	0.723364	0.873899	0.315845	0.524152	0.325829	0.416450	0.382965	0.762690	0.620151	0.529821
5	0.639758	0.597195	0.252950	0.438664	0.818726	0.997745	0.517425	0.274078	0.739661	0.415433	0.569164
6	0.525891	0.54989 2	0.451679	0.380050	0.647040	0.577496	0.513544	0.671255	0.810786	0.662634	0.579027
7	0.773149	0.516663	0.633433	0.612384	0.795906	0.741715	0.691628	0.580420	0.798279	0.571460	0.671504
8	0.432872	0.799162	0.715053	0.794089	0.445922	0.555776	0.779642	0.805213	0.651113	0.910634	0.688948

Table 5. Individual processor utilization of a task set of size = 100 fourth instance

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.612214	0.771018	0.625425	0.741519	0.699507	0.464626	0.790185	0.549219	0.810997	0.436568	0.650128
2	0.494135	0.669769	0.887355	0.774805	0.674145	0.284716	0.761862	0.596420	0.580299	0.873629	0.659714
3	0.671951	0.681593	0.736429	0.507989	0.478969	0.716511	0.605460	0.742051	0.619422	0.741878	0.650225
4	0.404957	0.674895	0.515171	0.329792	0.837037	0.603247	0.737038	0.836448	0.750020	0.659656	0.634826
5	0.861800	0.632520	0.956147	0.737529	0.508371	0.650479	0.667373	0.768223	0.683979	0.513034	0.697946
6	0.650784	0.546660	0.427914	0.683030	0.884145	0.941375	0.670779	0.436622	0.713392	0.619103	0.657380
7	0.512057	0.550765	0.556660	0.623390	0.554956	0.682246	0.617766	0.505315	0.463815	0.948338	0.601531
8	0.726130	0.764544	0.433677	0.615883	0.522790	0.730002	0.441989	0.662984	0.654819	0.512546	0.606536

Table 6. Individual processor utilization of a task set of size = 100 fifth instance

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.719749	0.510794	0.567254	0.700115	0.892775	0.892775	0.717473	0.617559	0.862472	0.53964	0.702061
2	0.883496	0.822381	0.630889	0.945138	0.832658	0.832658	0.690349	0.747878	0.476908	0.731456	0.759381
3	0.818752	0.417144	0.801810	0.80304	0.677994	0.677994	0.98596	0.401228	0.424293	0.79695	0.680517
4	0.761083	0.685494	0.994338	0.565567	0.853777	0.853777	0.646977	0.935643	0.56519	0.739703	0.760155
5	0.649709	0.835244	0.486924	0.782676	0.405674	0.405674	0.341426	0.587329	0.652891	0.842251	0.598980
6	0.536061	0.399638	0.811698	0.248848	0.188748	0.188748	0.706742	0.485563	0.945826	0.641349	0.515322
7	0.368148	0.717534	0.425266	0.734474	0.787811	0.787811	0.674983	0.493009	0.606300	0.321399	0.591674
8	0.474365	0.815519	0.650012	0.513540	0.752264	0.752264	0.563674	0.809527	0.768574	0.550777	0.665052

Table 7. Waiting time of a task set of size = 100 for all 5 instances

Instances	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	0.345898	0.290868	0.308291	0.350713	0.305333	0.314591	0.3098	0.338039	0.328943	0.330861
2	0.326957	0.333458	0.283039	0.333556	0.307403	0.328069	0.315484	0.323964	0.322055	0.333663
3	0.293618	0.298907	0.325333	0.307282	0.328464	0.308047	0.28521	0.316875	0.297353	0.288269
4	0.300422	0.320143	0.316751	0.297545	0.304444	0.320602	0.321953	0.322758	0.311316	0.331571
5	0.302223	0.312427	0.309568	0.320378	0.339915	0.339915	0.323188	0.309714	0.284431	0.302617

Table 8. Scheduling time of a task set of size = 100 for 5 instances

Instances	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
1	0.43956	0.824176	0.604396	0.494505	0.604396	0.549451	0.494505	0.549451	0.494505	0.549451
2	0.659341	0.604396	0.549451	0.549451	0.549451	0.604396	0.659341	0.604396	0.714286	0.549451
3	0.714286	0.604396	0.604396	0.714286	0.659341	0.549451	0.549451	0.604396	0.494505	0.494505
4	0.659341	0.494505	0.604396	0.714286	0.714286	0.824176	0.659341	0.714286	0.494505	0.494505
5	0.604396	0.604396	0.604396	0.549451	0.494505	0.494505	0.549451	0.659341	0.604396	0.714286

respectively. The schedule with maximum quality out of the ten approximate schedules generated in the ten trials is assumed to be the schedule for that problem instance.

The characteristics of each instance of the problem set is arrived at by applying *k*-means clustering algorithm with k = 4. The set of four cluster heads describe each instance of the problem set. So the five instances of the

problem set generate five sets of four cluster heads. This completes the training phase.

In the testing phase, a new problem instance of a set of 100 tasks is obtained by generating another utilization matrix randomly. Its characteristics are obtained by forming them into four clusters. Then the schedule of that instance of the training set, that resembles this task set most closely, is predicted as the schedule for this new task set. Then the ACO scheduling algorithm is run on the new problem instance and the parameter values and the schedule are obtained. A comparison between the schedule predicted and the actual schedule obtained by running ACO is done based on the parameters: time taken to arrive at the schedule and the quality of the schedule and the results are tabulated. ACO scheduling algorithm is run on the new task set of size 100 and the individual processor utilization of the actual schedule obtained is shown in Table 9. Table 10 shows the individual processor utilization for that problem instance of the task set of size 100 that most closely resembles the new task set. A comparison between the two tables illustrates how closely the predicted schedule is to the actual schedule based on individual processor utilization.

The above process is done for three more problem sets of sizes 120, 140 and 160 and the results are tabulated. Tables 11–16 show the individual processor utilization for the actual and predicted schedules for the task sets of sizes 120, 140 and 160 respectively. Table 17 compares all the three parameters of the actual and predicted schedules for all these 4 problem sets.

Figure 1 represents pictorially the individual processor utilization of the actual and predicted schedules for the task sets of sizes 100, 120, 140 and 160. It is found from Figure 1 that the total processor utilization is almost the same for the predicted and actual schedules. Figure 2 gives a comparison of utilization of each processor, the waiting time of all the tasks and the time taken to schedule all the tasks for the actual and the predicted schedules for all the four problem sets.

The results show that all the three parameters are almost equal for the predicted and actual schedules. The prediction of a schedule for a new instance of a problem

Table 9. Individual processor utilization of a task set of size = 100(actual)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.548633	0.524398	0.313916	0.648265	0.654898	0.640466	0.918396	0.760246	0.819619	0.377237	0.620607
2	0.627619	0.731921	0.822067	0.701193	0.353306	0.776318	0.242119	0.725384	0.375631	0.819094	0.617465
3	0.977234	0.543839	0.559451	0.450179	0.712966	0.293200	0.689102	0.434565	0.867999	0.566503	0.609504
4	0.760411	0.644109	0.826027	0.662171	0.406152	0.643019	0.642746	0.855257	0.780342	0.764999	0.698523
5	0.536226	0.448079	0.611148	0.588918	0.854119	0.690780	0.557881	0.743040	0.694269	0.595103	0.631956
6	0.592752	0.964651	0.613521	0.720135	0.510047	0.712633	0.931270	0.386495	0.714789	0.921940	0.706823
7	0.596330	0.994327	0.379002	0.478390	0.663218	0.853952	0.537079	0.770122	0.605968	0.866103	0.674449
8	0.767240	0.604499	0.979770	0.806445	0.794023	0.717985	0.469138	0.636168	0.421051	0.323712	0.652003

Table 10. Individual processor utilization of a task set of size = 100 (predicted)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.719749	0.510794	0.567254	0.700115	0.892775	0.892775	0.717473	0.617559	0.862472	0.539640	0.702061
2	0.883496	0.822381	0.630889	0.945138	0.832658	0.832658	0.690349	0.747878	0.476908	0.731456	0.759381
3	0.818752	0.417144	0.801810	0.803040	0.677994	0.677994	0.985960	0.401228	0.424293	0.796950	0.680517
4	0.761083	0.685494	0.994338	0.565567	0.853777	0.853777	0.646977	0.935643	0.565190	0.739703	0.760155
5	0.649709	0.835244	0.486924	0.782676	0.405674	0.405674	0.341426	0.587329	0.652891	0.842251	0.598980
6	0.536061	0.399638	0.811698	0.248848	0.188748	0.188748	0.706742	0.485563	0.945826	0.641349	0.515322
7	0.368148	0.717534	0.425266	0.734474	0.787811	0.787811	0.674983	0.493009	0.606300	0.321399	0.591674
8	0.474365	0.815519	0.650012	0.513540	0.752264	0.752264	0.563674	0.809527	0.768574	0.550777	0.665052

Table 11. Individual processor utilization of a task set of size = 120(actual)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.641667	0.728053	0.757178	0.680856	0.866961	0.674112	0.490531	0.809130	0.419784	0.743614	0.681188
2	0.879733	0.849649	0.403386	0.923163	0.500992	0.753723	0.561881	0.885043	0.971135	0.427677	0.715638
3	0.584729	0.764064	0.836533	0.552011	0.575890	0.516873	0.655009	0.540118	0.732240	0.807536	0.656500
4	0.636853	0.612431	0.441541	0.672805	0.767690	0.882250	0.504391	0.642926	0.970929	0.880296	0.701211
5	0.838034	0.605933	0.811058	0.631690	0.912801	0.852561	0.745007	0.796077	0.634340	0.333096	0.716059
6	0.958754	0.574961	0.481241	0.695692	0.355171	0.762218	0.860330	0.212076	0.506186	0.628586	0.603521
7	0.353472	0.725902	0.852056	0.483016	0.772710	0.623811	0.706372	0.872567	0.521190	0.951223	0.686231
8	0.566224	0.487517	0.726582	0.693374	0.825012	0.543440	0.994974	0.602885	0.747576	0.577707	0.676529

Table 12. Individual processor utilization of a task set of size = 120(predicted)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.614637	0.707766	0.317366	0.715138	0.500589	0.445023	0.830471	0.460145	0.475165	0.860045	0.592634
2	0.713264	0.451405	0.593324	0.827173	0.797708	0.851259	0.810266	0.889658	0.806706	0.847413	0.758817
3	0.607738	0.591765	0.656522	0.740733	0.641253	0.729219	0.635845	0.788455	0.426330	0.836795	0.665465
4	0.656924	0.719865	0.808505	0.708724	0.531950	0.761453	0.799918	0.727841	0.481148	0.813811	0.701013
5	0.526202	0.685226	0.691633	0.823143	0.624619	0.769752	0.585168	0.753602	0.800966	0.387502	0.664781
6	0.727964	0.595835	0.838054	0.726453	0.714116	0.938764	0.679191	0.568009	0.432616	0.661523	0.688252
7	0.673574	0.918848	0.745747	0.582955	0.857890	0.354774	0.435131	0.740297	0.977246	0.192114	0.647857
8	0.992352	0.730539	0.773438	0.428505	0.462343	0.530835	0.604835	0.762020	0.877954	0.608734	0.677155

Table 13. Individual processor utilization of a task set of size = 140(actual)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.592202	0.598846	0.425825	0.525398	0.929518	0.629092	0.444884	0.546656	0.578397	0.576656	0.584747
2	0.610233	0.651190	0.587524	0.655990	0.523561	0.813353	0.807617	0.670819	0.633364	0.862170	0.681582
3	0.752055	0.949435	0.499202	0.639077	0.845557	0.557578	0.516255	0.838723	0.818435	0.757629	0.717394
4	0.565612	0.669207	0.788387	0.880267	0.637155	0.543963	0.553181	0.608492	0.660573	0.787840	0.669467
5	0.433311	0.766383	0.842371	0.714930	0.501425	0.678371	0.760227	0.868792	0.626579	0.532654	0.672504
6	0.581491	0.577676	0.667474	0.617129	0.794543	0.565321	0.769178	0.671412	0.598780	0.564729	0.640773
7	0.970833	0.549449	0.829366	0.604235	0.621715	0.596417	0.827079	0.533776	0.555248	0.554661	0.664277
8	0.667174	0.686860	0.536585	0.722455	0.481705	0.847393	0.577933	0.485001	0.891952	0.796945	0.669400

Table 14. Individual processor utilization of a task set of size = 140 (predicted)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.676914	0.488895	0.497442	0.505466	0.788556	0.550622	0.718123	0.650181	0.905267	0.734003	0.651546
2	0.581312	0.696232	0.349132	0.487800	0.833656	0.703045	0.736042	0.887146	0.650243	0.772416	0.669702
3	0.647346	0.510642	0.759288	0.674998	0.547117	0.639196	0.475649	0.722762	0.617615	0.428162	0.602277
4	0.713803	0.929059	0.913572	0.570482	0.404193	0.826175	0.636084	0.622461	0.660488	0.810253	0.708657
5	0.761575	0.689142	0.710068	0.676610	0.884099	0.329650	0.482839	0.614852	0.544148	0.767019	0.646000
6	0.517670	0.607994	0.752379	0.797626	0.606368	0.856955	0.766532	0.864538	0.719311	0.742679	0.723205
7	0.535379	0.862057	0.571691	0.784520	0.827862	0.712601	0.565676	0.380452	0.681503	0.494952	0.641669
8	0.577659	0.382800	0.673831	0.709263	0.424488	0.641057	0.751971	0.432419	0.417778	0.519894	0.553116

Table 15. Individual processor utilization of a task set of size = 160(actual)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.988679	0.803435	0.791433	0.760569	0.697116	0.904263	0.805188	0.435572	0.831855	0.605004	0.762311
2	0.766847	0.828376	0.853372	0.868316	0.576013	0.774109	0.434449	0.608197	0.397727	0.829711	0.693711
3	0.312381	0.887356	0.519111	0.667715	0.747089	0.695413	0.680535	0.567489	0.557100	0.873245	0.650743
4	0.827140	0.588504	0.651463	0.543773	0.433338	0.676612	0.555132	0.939688	0.769847	0.679477	0.666497
5	0.855343	0.710790	0.863887	0.858270	0.754759	0.796293	0.885367	0.711493	0.804189	0.671282	0.791167
6	0.729636	0.542444	0.472638	0.713516	0.544405	0.688123	0.811834	0.580099	0.626074	0.519420	0.622819
7	0.294579	0.279820	0.700599	0.384375	0.659658	0.433324	0.655689	0.672560	0.512446	0.703858	0.529691
8	0.614691	0.652855	0.396222	0.543804	0.944256	0.511776	0.568580	0.661736	0.755204	0.512809	0.616193

Table 16. Individual processor utilization of a task set of size = 160 (predicted)

Processor	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average utilization
1	0.562295	0.478413	0.826563	0.878070	0.570400	0.880872	0.426639	0.607730	0.606602	0.594271	0.643186
2	0.619338	0.583407	0.573713	0.601034	0.579826	0.554614	0.728122	0.601737	0.624362	0.449276	0.591543
3	0.495760	0.676591	0.539877	0.885179	0.794528	0.564350	0.531969	0.559209	0.924024	0.599891	0.657138
4	0.675717	0.792475	0.792551	0.482393	0.841834	0.724846	0.787474	0.712459	0.797799	0.730787	0.733834
5	0.752332	0.893308	0.560355	0.738782	0.823378	0.730886	0.943927	0.572991	0.632883	0.723664	0.737251
6	0.926435	0.833910	0.568619	0.577955	0.653972	0.639156	0.851791	0.688707	0.624514	0.542977	0.690804
7	0.584015	0.691933	0.824436	0.539072	0.787497	0.858838	0.669141	0.677701	0.612726	0.692457	0.693782
8	0.730664	0.516451	0.531348	0.622979	0.536711	0.507571	0.754068	0.956100	0.548168	0.953799	0.665786

Table 17. Comparison of average processor utilization, waiting time of tasks and scheduling time of tasks in actual and predicted schedules

Task Parameters	Tasks = 100 (actual)	Tasks = 100 (predicted)	Tasks = 120 (actual)	Tasks = 120 (predicted)	Tasks = 140 (actual)	Tasks = 140 (predicted)	Tasks = 160 (actual)	Tasks = 160 (predicted)
Waiting time of tasks(secs)	0.319822	0.314437	0.332466	0.326674	0.310669	0.313634	0.332732	0.325319
Scheduling time of tasks(secs)	0.571429	0.362640	0.824176	0.362640	1.587912	0.362640	1.945055	0.362640
Average processor utilization	0.651416	0.659143	0.679610	0.674498	0.662518	0.649522	0.666642	0.676665

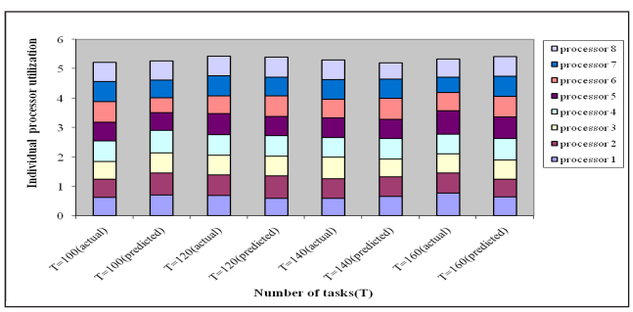

Figure 1. Comparison of average processor utilization of the actual and predicted schedules for 4 problem sets.

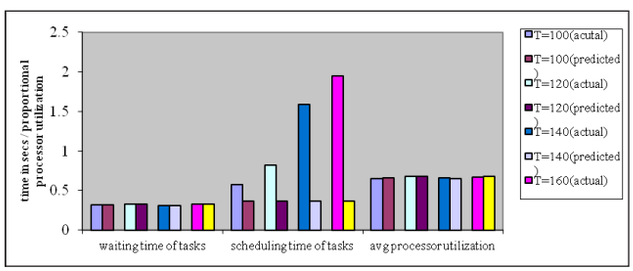

Figure 2. Comparison of average processor utilization, waiting and scheduling time of the actual and predicted schedules for 4 problem sets.

set reduces the time to come up with a feasible schedule by a considerable amount. It is shown experimentally that on the average, time taken to arrive at a feasible schedule for 100 tasks is reduced by 36.5%, 120 tasks is reduced by 56%, 140 tasks is reduced by 77.2% and 160 tasks is reduced by 81.4%. It is observed that the reduction in the scheduling time increases with the increase in the task set size since the convergence time increases in ACO with the increase in the task set size.

6. Conclusion

A task scheduling model that uses the bio-inspired paradigm ACO is designed and implemented. The proposed model predicts a schedule for a new task set after the training phase. The study shows that the scheduling time is considerably reduced without compromising much on the quality of the schedule generated. But the price that has to be paid is the time taken for training. This is not too much of an overhead considering the fact that the training is carried out only once.

7. References

1. Baruah SK. Partitioning real-time tasks among heterogeneous multiprocessors. International Conference on Parallel Processing; 2004; p. 467–74.
2. Ijaz S, Munir E, Anwar W, Nasir W. Efficient scheduling strategy for task graphs in heterogeneous computing environment. The Int Arab J Inform Tech, 2013; 10(5):386–492.
3. De Castro LN. Fundamentals of natural computing: an overview. Phys Life Rev. 2007; 4(1):1–36.
4. Forbes N. Biologically inspired computing. J Comput Sci Eng. 2000; 2(6):83–7.
5. Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the travelling salesman problem. IEEE Trans Evol Comput. 1997; 1(1):53–66.
6. Ferrandi F, Pilato C, Tumeo C, Sciuto D. Mapping and scheduling of parallel c applications with ant colony optimization onto heterogeneous reconfigurable MPSoCs. Proceedings of IEEE Asia and South Pacific Design Automation Conference; 2010; p. 799–804.
7. Lee YC, Zomaya AY. An artificial immune system for heterogeneous multiprocessor scheduling with task duplication. Proceedings of the International Parallel and Distributed Processing Symposium; 2007; p.1–8.
8. Rahmani AM, Vahedi MA. A novel task scheduling in multiprocessor systems with genetic algorithm by using elitism stepping method. IJCTE. 2009; 1(1): 1–6.
9. Lam YM. Integrated task clustering, mapping and scheduling for heterogeneous computing systems, Int J Comput Sci Inform Tech. 2012; 4(1):275–80.
10. Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B Cybern.1996; 23(1):29–41.
11. Ashwin Kumar Sarma V, Rahul Rajendra, Dheepan P, Sendhil Kumar KS. an optimal ant colony algorithm for efficient VM placement. Indian Journal of Science and Technology. 2015 Jan; 8(S2):156–59.
12. Rishita Kalyani. application of multi-core parallel programming to a combination of ant colony optimization and genetic algorithm. Indian J Sci Technol. 2015 Jan; 8(S2):138–42.
13. Swapna B. Sasi, Sivanandam N. A survey on cryptography using optimization algorithms in WSNs. Indian J Sci Technol. 2015 Feb; 8(3):216–21.
14. Dizaji ZA, Gharehchopogh FS. A hybrid of ant colony optimization and chaos optimization algorithms approach for software cost estimation. Indian J Sci Technol 2015 Jan; 8(2):128–33.