Indonesian Journal of Electrical Engineering and Computer Science
Vol. 10, No. 1, April 2018, pp. 320~329
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v10.i1.ppa320-329

Task Scheduling in Heterogeneous Multiprocessor Environments–An Efficient ACO-Based Approach

Nekiesha Edward, Jeffrey Elcock
Department of Computer Science, Mathematics and Physics, University of the West Indies, Cave Hill Campus, Bridgetown, Barbados

Article Info

Article history:

Received Jan 9, 2018
Revised Mar 2, 2018
Accepted Mar 18, 2018

Keywords:

Task Scheduling
Heterogeneous
Ant Colony Optimization
Directed Acyclic Graphs

ABSTRACT

In heterogeneous computing environments, finding optimized solutions continues to be one of the most important and yet, very challenging problems. Task scheduling in such environments is NP-hard, so efficient mapping of tasks to the processors remains one of the most critical issues to be tackled. For several types of applications, the task scheduling problem is crucial, and across the literature, a number of algorithms with several different approaches have been proposed. One such effective approach is known as Ant Colony Optimization (ACO). This popular optimization technique is inspired by the capabilities of ant colonies to find the shortest paths between their nests and food sources. Consequently, we propose an ACO-based algorithm, called rACS, as a solution to the task scheduling problem. Our algorithm utilizes pheromone and a priority-based heuristic, known as the upward rank value, as well as an insertion-based policy and a pheromone aging mechanism to guide the ants to high quality solutions. To evaluate the performance of our algorithm, we compared our algorithm with the ACS algorithm and the ACO-TMS algorithm using randomly generated directed acyclic graphs (DAGs). The simulation results indicated that our algorithm experienced comparable or even better performance, than the selected algorithms.

Corresponding Author:

Jeffrey Elcock
Department of Computer Science, Mathematics and Physics,
The University of the West Indies, Cave Hill Campus,
P.O. Box 64 Bridgetown, Barbados.
Email: Jeffrey.elcock@cavehill.uwi.edu

1. INTRODUCTION

There are considerable improvements and advances in technology and computer architecture that have been achieved over the years and among these, are heterogeneous multiprocessor systems. Gaining increasing popularity for their diverse and incredible capabilities [1], these high performance environments continue to offer several benefits, including increased throughput and the potential for faster scheduling through increased parallelism. As such, task scheduling continues to be actively addressed in order to fully exploit and extract the benefits that these systems have to offer. Task scheduling, is defined as the assignment of tasks of a parallel application to different processors in a manner that minimizes the overall completion time or schedule length (SL) of the application while ensuring that all constraints are fully satisfied [2]. In a heterogeneous environment, scheduling of these interdependent tasks becomes even more challenging, because of the varying speeds associated with the different processors and hence the different computational cost associated with each task [3].

A program or parallel application may be modeled by a task graph in the form of a weighted directed acyclic graph (DAG), G = (V, E), where V denotes the set of nodes (n_i) which represent the tasks of the application and E denotes the set of edges that indicate the data dependencies between the various tasks.

The weight on each edge, denoted by *w*, represents the communication cost between two nodes and a computation cost matrix indicates the time it takes for the nodes (tasks) to execute on each of the processors [4].

In an instance where $(n_i, n_j) \in E$ then n_i is called the immediate predecessor or parent of n_j, and n_j is called the immediate successor or child of n_i. If a task n_a, has two or more immediate predecessors, then n_a is referred to as a joined task. The immediate successors of n_a is denoted by $isucc(n_a)$ and is defined as $\{ n_j \mid (n_a, n_j) \in E\}$, while its set of immediate predecessors, denoted by $ipred(n_a)$, is defined as $\{ n_j \mid (n_j, n_a) \in E\}$. Before task n_a, can be scheduled, all of its parent nodes must first be scheduled. Additionally, the critical path of a DAG is the longest path from the entry node to the exit node, considering both the computation and communication costs between the tasks [5]. It is assumed that there is one entry task (n_{entry}) which has no predecessor nodes, and one exit task (n_{exit}), which is a node with no successors for the DAG. If a DAG contains multiple entry or exit tasks, a dummy entry or exit node with zero computation cost, along with a zero-communication cost, can be connected, therefore making our algorithm applicable for DAGs of any kind.

The focus of our research is static task scheduling, where information about available resources is known before execution and scheduling may be done at compile time. Whether static or dynamic, task scheduling is classified as an NP-Hard problem [4, 6]. However, the task scheduling problem has been well studied and a number of suboptimal heuristic-based solutions have been proposed. These solutions may be categorized as list scheduling, clustering, task duplication and guided random search methods.

List scheduling algorithms [7], [8], [9], [10] first prioritize tasks, based on various criteria, into an ordered list before mapping them onto the processors. Generally, list scheduling algorithms have good performance-to-cost tradeoffs. Clustering algorithms [4], [11], [12], [13] attempt to reduce the communication cost associated with dependent tasks by grouping communicating nodes together in clusters for scheduling onto the same processor. The foundation of this approach is the fact that when two communicating tasks are placed on the same processor, the communication cost between them becomes negligible. The basis of task duplication algorithms [14], [15], [16], [17] is to, where possible, remove the cost associated with inter-processor communication, and thus reduce the overall schedule length, by duplicating predecessor nodes [4]. Task duplication, like clustering, takes advantage of the zero or negligible communication cost when two dependent tasks are placed on the same processor. Guided random search algorithms [5], [18], [19], [20], [21], [22], on the other hand, examine multiple solutions in the search space and converge to an efficient solution. Such algorithms, like Genetic Algorithms and Ant Colony Optimization (ACO) have been applied to the task scheduling problem producing some very good results. In particular, the ACO technique, which forms the basis of our algorithm, is considered an adaptable solution, and has been successfully applied to multiple problems [23], [24], [25].

1.1. The ACO Metaheuristic

Ant Colony Optimization (ACO) algorithms were first introduced by Dorigo and his colleagues in the early 1990s, and form part of a wider research area known as Swarm Intelligence, which models solutions to combinatorial, and optimization problems, based on the behavior and processes exhibited in nature [25]. ACO is inspired by the indirect communication of a foraging ant colony, where the survival of the entire colony governs the ants' behavior and not simply individual survival. This indirect communication, known as stigmergy, enables ants to find very short paths between food sources and their nest [26].

In the initial stages of foraging, the ants explore the area randomly, depositing chemical pheromone trails as they traverse. When food is encountered, the quality and quantity is assessed and pheromone, from the food source to the nest, is deposited. Subsequent foraging ants utilize these pheromone trails to guide them to the food, with the probability of utilizing paths marked by strong pheromone concentrations, which reinforces the pheromone density and thus increases their attractiveness for later ants. This reinforcement leads to convergence to the most attractive path. Evaporation of pheromones on the trails provides the limiting mechanism for this positive feedback, so less frequented paths have decreased pheromone concentration.

The ACO metaheuristic (Fig.1) applies the foraging behavior of natural ants in a computational environment and iteratively constructs candidate solutions using artificial pheromone and local heuristics to guide the artificial agents (ants) through the investigated search space. The pheromone trails bias future agents toward high quality solutions, until a termination condition is satisfied.

Contrary to foraging ants in nature which deposit a continuous trail of pheromone, ACO approaches have implemented various alternatives [27]. For example, in the original Ant System (AS) [28] ants deposit pheromone to only completed solutions. Alternatively, the Ant Colony System (ACS) [29] makes step-by-step online (local) pheromone deposits by every agent during the construction of solutions and introduces a further offline (global) update of pheromones to the best solution of the iteration. Additionally, some kind of evaporation mechanism is implemented, allowing the ants to consider new areas of the search space [27]. Furthermore, some ACO techniques employ local and global optimization strategies to further increase the quality of the solutions produced.

The ACO technique has been applied to various optimization, classification and scheduling problems [25]. It has been combined with other random search algorithms, for example, the Genetic Algorithm and Tabu Search. ACO has also been combined with list scheduling, for instance, the ANT-LS algorithm [27] and the ACO-TMS [30]. This combination of pheromone trails and list scheduling heuristics facilitates further guidance for the ants toward good quality schedules.

```
The ACO Metaheuristic

Set parameters, initialize pheromone trails
while termination condition not satisfied do
    Construct Ant Solutions
    Apply Local Optimization (optional)
    Pheromone Update
end while
```

Figure 1. The ACO Metaheuristic

Given the versatility of ACO algorithms, we present an ACO-based algorithm which uses the foundation of the ACO. Our proposed algorithm incorporates the upward ranking concept used in the HEFT algorithm [8] in our prioritization methodology, an insertion-based policy along with pheromone aging, to produce efficient schedules. Our research investigates the application of an efficient solution to the static task scheduling problem in a heterogeneous environment where dependencies between the tasks are taken into consideration.

For our scheduling system model, the target computing environment consists of a set of processors P, where P = { p_1, p_2, p_3, ...$p_{|P|}$ }, and |P| denotes the number of processors. Our model assumes heterogeneous non-preemptive processors that are connected in a fully connected topology and inter-processor communication is contention-free. The main objective of the task scheduling problem is to determine a mapping of tasks of a given application to processors that minimizes the schedule length.

The remainder of the paper is organized as follows: We describe our proposed algorithm in Section 2 and outline our methodology for performance evaluation in Section 3. Results obtained from a performance comparison of the ACS [29] and ACO-TMS [30] with our proposed work are presented and discussed in Section 4 and we summarize our conclusions and future works in Section V.

2. THE PROPOSED ALGORITHM

2.1. Overview of Our Algorithm

Our proposed algorithm (Fig. 2) is known as the ranking-Ant Colony System (rACS) and combines the foundation of the Ant Colony System (ACS) with the heuristic function which was inspired by the list scheduling algorithm HEFT. ACS exhibits flexibility with the utilization of the offline pheromone update and HEFT has yielded good performance as a list scheduling algorithm, with the use of the upward rank value for prioritization.

Firstly, we initialize our two matrices for our pheromone representation. $V \times P$, which we denote as τ, and $P \times V$ which we denote as τ_1. The entry $\tau(i, p)$ indicates the pheromone on the edge between task i and processor element p, whereas $\tau_1(p, j)$ indicates the pheromone on the edge between processor element p and task j. Therefore, if $n_{entry} \rightarrow p_2 \rightarrow n_3 \rightarrow p_1 \rightarrow n_2 \rightarrow p_{|P|} \rightarrow \ldots . \rightarrow n_{exit} \rightarrow p_1$ is a possible solution (a complete mapping of the task graph within the search space where an ant, starts at the entry node (n_{entry}) moves from task to processor and from processor to task, until a processor has been selected for the exit node (n_{exit})), then τ (n_3 , p_1) ϵ $V \times P$ and τ_1 (p_1, n_2) ϵ $V \times P$. Initially, a small pheromone deposit is made to all elements of each matrix and the ready list (RL) is initialized containing the entry node.

Our iterative ant colony algorithm then, executes as follows: for each ant, in each iteration, an ant list of length V that stores both a task and its selected processor is created. The ant selects a task from the ready list using the state transition (ST) rule (1) and a processor using the state transition (ST) rule (2) to construct a schedule. The selected task is removed from the ready list, and appended, along with the processor, to the ant list. The ready list is then updated to contain all the unscheduled children nodes of those parents who have already been scheduled. This process is repeated until all the tasks have been mapped. During the first iteration, our algorithm rACS, does not employ the state transition rules to select either task or processor – they are both selected in a random manner – thus mimicking the ants' natural environment. Throughout the execution of the algorithm, an insertion-based policy is employed whereby the task or node is checked to see if it can be scheduled earlier on the chosen processor, thus affording our approach, the opportunity to achieve shorter schedules.

After each iteration, an online or local pheromone update is applied to the best *q* ants ($q \leq K$, where K is the number of ants per iteration), according to the local pheromone updating (LPU) rule using (6), (7) and (8). Following this update, there is also an offline or global update to the best ant solution of the iteration according to the global pheromone updating (GPU) rule using (9), (10) and (11).

Our algorithm also attempts to alleviate stagnation by employing a pheromone aging mechanism (represented as Φ). This condition monitors how the best solution changes over the course of the execution of the algorithm. If the value for the best schedule length remains unchanged after a predetermined number of iterations, a deliberate evaporation of the pheromone trail of that schedule is invoked. When invoked, a random value is generated for Φ, which is always less than or equal to the initial pheromone, and applied. Repetition of this condition is dependent on the random generation of a value which is equal to, or less than the total number of iterations of the algorithm. This deliberate evaporation facilitates the increased probability of exploring new, possibly better-quality solutions.

```
BEGIN rACS
Initialize pheromone
while termination conditions are not satisfied do
 For each iteration do
    Repeat /*For each ant
        Initialise RL
        while RL not empty
          if first iteration
                Select task from RL randomly
                Select PE randomly
          else
                Select task from RL using ST Rule I
                Select PE using ST Rule II
          Update RL
         end while
     Until ant has built complete schedule
     Apply local pheromone update using LPU rules
     Apply global pheromone update using GPU rules

Aging Mechanism ( if invoked)
  end iteration
end while
END rACS
```

Figure 2. Our Proposed Algorithm

2.2. Criteria for Task and Processor Selection

With our proposed algorithm, the task and processor selections are governed by two state transition rules as follows:

Task Selection Rule: Each ant selects a task (*i*) from the ready list (RL) according to the probability calculated by

$$\Pr(i) = \frac{[\tau(i,p)]^{\delta} \cdot [\eta(i)]^{\theta}}{\sum_{r \in RL} [\tau(r,p)]^{\delta} \cdot [\eta(r)]^{\theta}} \quad (1)$$

Where $\tau(i,p)$ indicates the pheromone on the edge between task *i* and processor element *p* while δ and θ indicate the influence of the heuristic function and pheromone such that $\theta \in \{0,1,2\}$ and $\delta \in \{0,1,2\}$ and $\eta(i)$ is the list scheduling heuristic function which is calculated by

$$\eta(i) = \frac{1}{U_{rk}(i)} \qquad (2)$$

The upward rank, $U_{rk}(i)$ is the longest path from a particular node to the exit node, inclusive of the computation cost of node and is recursively defined as

$$U_{rk}(i) = \overline{w_i} + \max_{j \in succ(i)} [\overline{c_{i,j}} + U_{rk}(j)] \qquad (3)$$

Where $succ(i)$ denotes the immediate successors of node n_i, $\overline{w_i}$ is the average computation cost of n_i and $\overline{c_{i,j}}$ is the average communication on edge *(i,j)*

Processor Selection Rule: Each ant selects a processor element (*v*) for a chosen task (*j*) based on a probability calculated by

$$\Pr(v) = \frac{[\tau_1(v,j)] \cdot [\eta_*(v)]^{\Psi}}{\sum_{p \in P} [\tau_1(p,j)] \cdot [\eta_*(p)]^{\Psi}} \qquad (4)$$

where

$$\eta_*(v) = \frac{1}{avgCC(j)} \cdot \frac{1}{EST(j,v)} \qquad (5)$$

avgCC(j) is the average computation cost of the node *j* on the processors, and the *EST (j,v)* is the estimated start time of the node *j* on processor *v*. Ψ, such that $\Psi \in \{0,1,2\}$, is the influence on the probability and p is a component of the set of processors (P).

2.3. Criteria for Pheromone Update

The proposed algorithm (rACS), applies local pheromone update after construction of the ant solutions and a global pheromone update to the best ant solution of the iteration.

Local Pheromone Update: Pheromone is applied to edges *(i, p)* and *(v, j)* based on the functions

$$\tau(i,p) = (1-\rho) \cdot \tau(i,p) + \Delta\tau \qquad (6)$$

$$\tau_1(v,j) = (1-\rho) \cdot \tau_1(v,j) + \Delta\tau_1 \qquad (7)$$

where

$$\Delta\tau_1 = \Delta\tau = \frac{1}{FT_{(ant)}} \qquad (8)$$

and $FT_{(ant)}$ is the completion time of the ant while *ρ* is the pheromone decay parameter $(0 < \rho < 1)$.

Global Pheromone Update: Pheromone is applied to edges *(i, p)* and *(v, j)* of the solution of the best ant based on the functions:

$$\tau(i,p) = (1-a) \cdot \tau(i,p) + a \cdot \Delta\tau \qquad (9)$$

$$\tau_1(v,j) = (1-a) \cdot \tau_1(v,j) + a \cdot \Delta\tau_1 \qquad (10)$$

where

$$\Delta\tau_1 = \Delta\tau = \frac{1}{FT_{(best)}} \qquad (11)$$

and $FT_{(best)}$ is the finish time of the best ant of the iteration while a denotes the pheromone evaporation parameter $(0 < a < 1)$.

3. RESEARCH METHOD

We conducted a comprehensive performance evaluation of our algorithm by utilizing a two-pronged approach: (1) an evaluation of some of the attributes of our algorithm and (2) a comparison of our proposed work with two published ACO-based algorithms.

During the analysis of our proposed algorithm, we investigated the efficiency of the following properties:

- Utilization of Idle Processor Time
- Randomness with First Iteration
- Efficiency of Pheromone Aging Mechanism

With the experimentation of randomness on the first iteration, we investigated the influence of the guidance vs randomness during the first iteration. To test this, we searched the literature for DAG instances published in the literature [8], [22], [3]. This was done so as to avoid biasing any specific DAG. Table 1 shows the published makespans of the the selected DAGs.

Table 1. Published Makespans of the Selected DAGs

Graph	Published Makespan
DAG1 [8]	80
DAG2 [22]	31
DAG3 [3]	135

In the second phase of our evaluation, we compared our proposed work with the ACS [29] and ACO-TMS [30] algorithms by utilizing randomly generated task graphs. For this comparison, a total of 13,500 random graphs with the various characteristics were generated and then executed. The algorithms were then compared based on selected comparative metrics.

3.1. Attributes of Randomly Generated DAGs

In our experiment, the following input parameters were used for the generation of the task graph, which were also utilized in [8]:

- Number of tasks in the DAG ($|V|$).
- OutDegree of a node (O_{deg}). This is the maximum number of children of a node.
- Shape parameter of the graph (α).
- Communication to computation ratio (*CCR*). It is the ratio between the average communication cost and the average computation cost.
- Range percentage of computation costs on processors (*β*). It is the heterogeneity factor for processors. A higher percentage value indicates a significant difference in the computation cost across the processors, while lower values are indicative of more subtle differences in computation costs.

For each experiment, the values discussed above, were assigned from the sets given below.

- Set of Nodes (V) = {20, 40, 60, 80, 100}
- Set of CCR (CCR) = {0.1, 0.5, 1.0, 5.0, 10.0}
- Set of Alpha (α) = {0.5, 1.0, 2.0}
- Set of OutDegree (Odeg) = {1, 2, 3, 4, 5}
- Set of Beta (β) = { 0.25, 0.5, 0.75, 1.0}
- Number of Ants (K) = {min((avg(Odeg) × |V|, 100)}
- Number of Iterations = {100}
- No of Processors = {3}

3.2. Comparative Metrics

a. Speedup: The ratio between the sequential time and the parallel execution time of a process is defined as the speedup. The sequential time is calculated by adding, sequentially, the computational cost of each task in the graph. This is done for each processor and then the smallest value is used. The parallel execution time is the completion time of the graph, which is also referred to as the Makespan or Scheduled Length (SL). Therefore

$$Speedup = \frac{\min_{p_k \in P}\{\sum_{n_i \in V} \omega(n_{i,k})\}}{SL} \tag{12}$$

where $\omega(n_{i,k})$ denotes the computational cost of task n_i on processor p_k.

b. Schedule Length Ratio: The Schedule Length (SL) is the main performance measure of a scheduling algorithm. In our experiment, a large set of task graphs with varying properties is used and therefore it becomes necessary to normalize the schedule length to the lower bound. This is called the Schedule Length Ratio (SLR). The SLR is defined as follows:

$$SLR = \frac{SL}{\sum_{n_i \in CriP_{MIN}} \min_{p_k \in P}\{\omega(n_{i,k})\}} \tag{13}$$

The denominator is the summation of the minimum computation costs of the tasks on the *CriP*$_{MIN}$ (minimum Critical Path). The *CriP*$_{MIN}$ is derived by first setting each task (n_i) to its minimum computational cost and calculating the length of the Critical Path (|CP|) using these values.

4. RESULTS AND DISCUSSION

4.1. Preliminary Analysis of Our Proposed Work

ACO-based algorithms can obtain shorter schedules when they (i) incorporate functionality that ensures processor idle time is kept to a minimum and (ii) allow ants to randomly select schedules - thereby mimicking their natural environment. ACO-based algorithms found in the literature, generally, apply a local optimization strategy after generating a solution. The idea behind this strategy is normally to make adjustments, where feasible, to improve the solution obtained. One such strategy is to effectively utilize idle processor time [21], [22], [26], thus, reducing the overall schedule length. Given this basis, we designed our algorithm such that, as each ant constructs a solution, when a task is selected, the identified processor is searched for possible idle slots where the task is inserted, so that it can achieve the earliest possible finish time. While our utilization of idle slots is consistent with the literature; with our approach, the use of idle processor slots is determined as the schedules are built, not after.

We also experimented in the first iteration, with the ants selecting tasks from the ready list, and processor in a random manner. From Table 2, it is noticeable that shorter schedules were and can be produced when this approach is implemented.

Table 2. Makespans Attained by Our Algorithm When Randomness of 1st Iteration is varied

Graph	Published Makespan	Makespan Attained By Racs	
		1st Iteration, Not Random	1st Iteration, Random
DAG1 [8]	80	79	73
DAG2 [22]	31	29	27
DAG3 [3]	135	135	135

We also experimented with deliberate evaporation of pheromone so as to mitigate stagnation or escape local optima. The impact was not as significant as expected. We postulate that the value used to generate evaporation had minimal impact because of the timing of invocation and the amount. However, because of the randomness of this activity, when invocation occurred during the early iterations where the pheromone concentration was not high, newer opportunities were provided. We anticipate that a more impactful and useful approach would be to, at the beginning of each iteration, allow a random number of ants to randomly create solutions. These new schedules would be incorporated if they are worthy.

The performance of the proposed algorithm (rACS) was further evaluated by comparison with its progenitor, the ACS algorithm [29], and the ACO-TMS algorithm [30]. For this comparison, random directed acyclic graphs (DAGs) of varying attributes were generated and then executed by the algorithms.

4.2. Comparison of Proposed Work with Selected Algorithms

The rACS, ACS and ACO-TMS algorithms were first compared based on the average makespan attained with the varying shape parameters. For our first experiment, DAGs of varying degrees of parallelism were generated. From the results in Fig. 3, it was found that rACS outperformed the other two algorithms for short graphs of high parallelism (larger α values). When $\alpha \geq 1$, it was 18 percent better than ACO-TMS and 48 percent better than ACS. With $\alpha < 1$, where the graphs have greater depth and a low parallelism, the rACS experienced on par performance with ACO-TMS, and was 52 percent better than the ACS.

The next experiment examined the variation of the average SLR of the algorithms as the number of nodes of the DAGs was increased. Fig. 4 shows that as the number of nodes increases, the difference of the average SLR values when compared to our proposed algorithm and that of the ACO-TMS shows a steady increase. This is indicative of better performance from our proposed algorithm for large applications with more tasks when compared to smaller applications. rACS is better than ACO-TMS by 6 percent and the ACS by 24 percent.

Fig. 5 illustrates the behavior of the algorithms, from our next experiment, which investigated the average speedup as the DAG size was increased. Our proposed algorithm experienced a steady increase in the average speedup, outperforming both the ACS and the ACO-TMS algorithms. The ACS experienced minimal increase in the speedup throughout this experiment. The average speedup experienced by the ACO-TMS was steady, however, not as pronounced as rACS. Further, as the number of nodes of the DAG was increased from 80 to 100, our proposed algorithm yielded the most prominent outperformance of the other two algorithms. Overall, our proposed was better than ACS and ACO-TMS by 25 and 7.5 percent respectively. A larger speedup value is indicative of a smaller execution time in a parallel environment. Our results suggest that, generally, our parallel execution times were consistently smaller than the sequential execution times, even as the number of nodes increased.

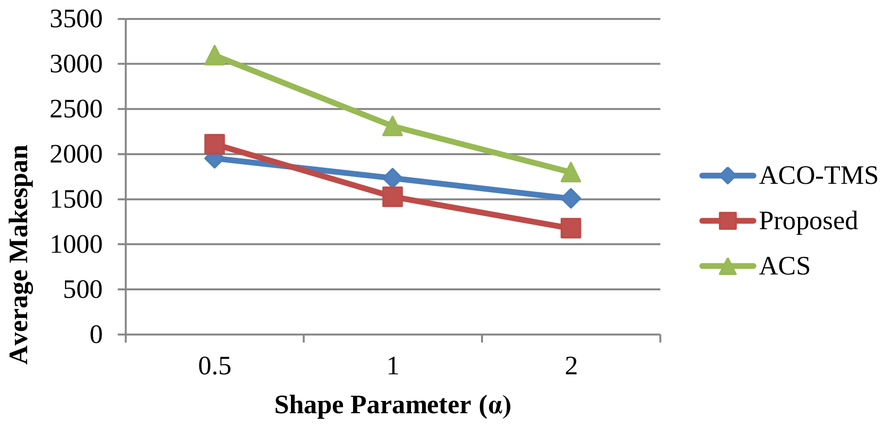

Figure 3. Results for Average Makespan for Varied DAG Structure

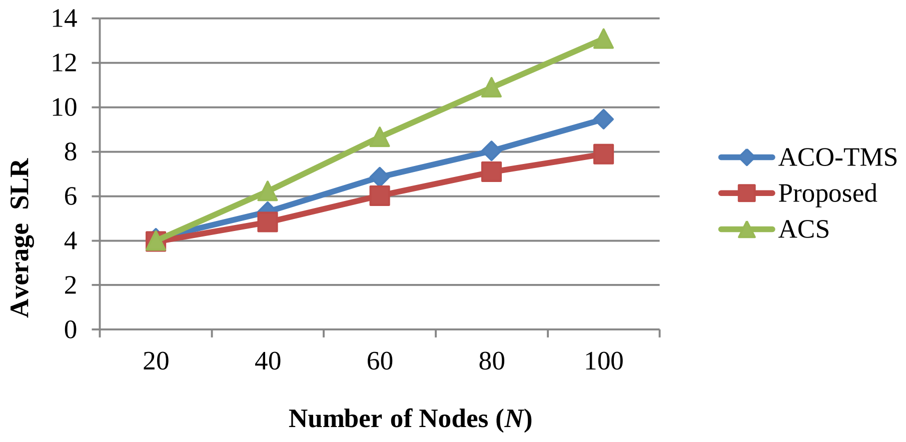

Figure 4. Results for Average SLR for Varying Number of Nodes

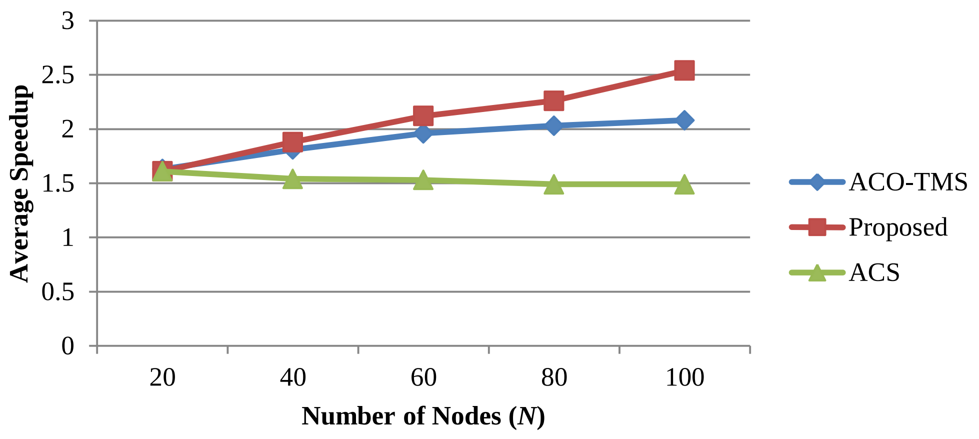

Figure 2. Results for Average Speedup for Varying Number of Nodes

5. CONCLUSION

In order to fully exploit the high performance of heterogeneous multiprocessor environments, versatile and robust scheduling strategies, which yield efficient results, are required. Our proposed algorithm is an ACO-based algorithm (rACS), which utilizes an upward rank value along with an insertion-based policy to further guide the ants toward quality solutions. In our experimental study we compared our proposed algorithm, rACS, with the ACS algorithm and the ACO-TMS algorithm using a set of various randomly generated task graphs. The rACS yielded better results, outperforming the algorithms in the various experiments such as average speedup and average SLR for increasing DAG size, as well as for varying DAG shape. Our planned future work is to investigate and add, to rACS, local optimization strategies to further increase its efficiency as an algorithm to tackle the static task scheduling problem.

REFERENCES

[1] Dogan, A. and F. Ozguner, *Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing.* IEEE Transactions on Parallel and Distributed Systems, 2002. 13(3): p. 308-323.

[2] Chaudhuri, P. and J. Elcock, *Scheduling DAG-based Applications In Multicluster Environments With Background Workload Using Task Duplication.* International Journal of Computer Mathematics, 2010. 87(11): p. 2387-2397.

[3] Ijaz, S., et al., *Efficient Scheduling Strategy For Task Graphs In Heterogeneous Computing Environment.* International Arab Journal of Information Technology, 2013. 10(5): p. 486-492.

[4] Lin, W.-M. and Q. Gu, *An Efficient Clustering-Based Task Scheduling Algorithm For Parallel Programs With Task Duplication.* Journal Of Information Science And Engineering, 2007. 23(2): p. 589-604.

[5] Manudhane, K.A. and A. Wadhe, *Comparative Study of Static Task Scheduling Algorithms for Heterogeneous Systems.* International Journal on Computer Science and Engineering, 2013. 5(3): p. 166-173.

[6] Kashani, M. and R. Sarvizadeh, *A Novel Method For Task Scheduling In Distributed Systems Using Max-Min Ant Colony Optimization*, in *2011 3rd International Conference on Advanced Computer Control (ICACC)*. 2011, IEEE: Harbin, China. p. 422-426.

[7] Iverson, M.A., F. Özgüner, and G.J. Follen. *Parallelizing Existing Applications In A Distributed Heterogeneous Environment*. in *4th Heterogeneous Computing Workshop* 1995. CA, USA: Citeseer.

[8] Topcuoglu, H., S. Hariri, and M.-y. Wu, *Performance-Effective And Low-Complexity Task Scheduling For Heterogeneous Computing.* IEEE Transactions On Parallel And Distributed Systems, 2002. 13(3): p. 260-274.

[9] Kwok, Y.-K. and I. Ahmad, *Benchmarking and Comparison of the Task Graph Scheduling Algorithms.* Journal of Parallel and Distributed Computing, 1999. 59(3): p. 381-422.

[10] Shirazi, B., M. Wang, and G. Pathak, *Analysis and Evaluation of Heuristic Methods for Static Task Scheduling.* Journal of Parallel and Distributed Computing, 1990. 10(3): p. 222-232.

[11] Yang, T. and A. Gerasoulis, *DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors.* IEEE Transactions on Parallel and Distributed Systems, 1994. 5(9): p. 951-967.

[12] Papadimitriou, C.H. and M. Yannakakis, *Towards an Architecture-independent Analysis of Parallel Algorithms.* SIAM journal on computing, 1990. 19(2): p. 322-328.

[13] Sakellariou, R. and H. Zhao. *A Hybrid Heuristic For Dag Scheduling On Heterogeneous Systems*. in *18th International Parallel and Distributed Processing Symposium, 2004.* . 2004. NM, USA: IEEE.

[14] Ahmad, I. and Y.-K. Kwok, *On Exploiting Task Duplication In Parallel Program Scheduling.* IEEE Transactions on Parallel and Distributed Systems, 1998. 9(9): p. 872-892.

[15] Ranaweera, S. and D.P. Agrawal. *A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems*. in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*. 2000. IEEE.
[16] Park, G.-L., B. Shirazi, and J. Marquis. *DFRN: A New Approach for Duplication based Scheduling for Distributed Memory Multiprocessor Systems*. in *Parallel Processing Symposium, 1997. Proceedings., 11th International*. 1997. IEEE.
[17] Nasr, A.A., N.A. EL-Bahnasawy, and E.-S. Ayman, *A New Duplication Task Scheduling Algorithm in Heterogeneous Distributed Computing Systems.* Bulletin of Electrical Engineering and Informatics, 2016. 5(3): p. 373-382.
[18] Abdeyazdan, M. and A. Rahmani. *Task Scheduling in Multiprocessor Systems Using a New Genetic Algorithm Priority Based on Number of Offspring*. in *Proc. 13th Iranian Int. CSI Computer Conf.* 2008.
[19] Jelodar, M.S., et al. *A Representation for Genetic-Algorithm-based Multiprocessor Task Scheduling*. in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. 2006. IEEE.
[20] Corrêa, R.C., A. Ferreira, and P. Rebreyend, *Scheduling Multiprocessor Tasks with Genetic Algorithms.* IEEE Transactions on Parallel and Distributed systems, 1999. 10(8): p. 825-837.
[21] Gupta, S., V. Kumar, and G. Agarwal. *Task Scheduling In Multiprocessor System Using Genetic Algorithm*. in *2010 Second International Conference on Machine Learning and Computing (ICMLC)*. 2010. Bangalore, India: IEEE.
[22] Chun, L., *Optimal Multi-Resource Scheduling Strategy Simulation Based on Improved Genetic Algorithm.* Indonesian Journal of Electrical Engineering and Computer Science, 2014. 12(4): p. 2898-2904.
[23] Abd-Allah, M., A. Said, and M.N. Ali, *Mitigation of lightning hazards at the more sensitive points in wind farms using ant-colony optimization technique.* Bulletin of Electrical Engineering and Informatics, 2016. 5(2): p. 144-159.
[24] Zhao, M. and D. Yong, *Robot Three Dimensional Space Path-planning Applying the Improved Ant Colony Optimization.* Indonesian Journal of Electrical Engineering and Computer Science, 2015. 14(2): p. 304-310.
[25] Jaiswal, U. and S. Aggarwal, *Ant Colony Optimization.* International Journal of Scientific & Engineering Research, 2011. 2(7): p. 1-7.
[26] Vassiliadis, V. and G. Dounias, *Nature–Inspired Intelligence: A Review Of Selected Methods And Applications.* International Journal on Artificial Intelligence Tools, 2009. 18(04): p. 487-516.
[27] Bank, M., U. Honig, and W. Schiffmann. *An ACO-based Approach for Scheduling Task Graphs with Communication Costs*. in *2005 International Conference on Parallel Processing (ICPP'05)*. 2005. Poland: IEEE.
[28] Dorigo, M., V. Maniezzo, and A. Colorni, *Ant System: Optimization by a Colony Of Cooperating Agents.* IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 1996. 26(1): p. 29-41.
[29] Dorigo, M. and L.M. Gambardella, *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.* IEEE Transactions On Evolutionary Computation, 1997. 1(1): p. 53-66.
[30] Chiang, C.-W., et al. *Ant Colony Optimisation for Task Matching and Scheduling*. in *IEEE Proceedings-Computers and Digital Techniques*. 2006. Institute of Engineering and Technology.