RESEARCH PAPER

International Journal of Recent Trends in Engineering, Vol. 1, No. 1, May 2009

Parallel Implementation of Task Scheduling using Ant Colony Optimization

T. Vetri Selvan[1], Mrs. P. Chitra [2], Dr. P. Venkatesh[3]

[1]Thiagarajar College of Engineering /Department of Computer Science, Madurai, India
Email:vetriselvan@tce.edu

[2] Thiagarajar College of Engineering /Department of Computer Science, Madurai, India
Email:pccse@tce.edu

[3] Thiagarajar College of Engineering /Department of Electrical & Electronics, Madurai, India

Abstract— **Efficient scheduling of tasks for an application is critical for achieving high performance in heterogeneous computing environment. The task scheduling has been shown to be NP complete in general case and also in several restricted cases. Because of its key importance on performance, the task scheduling problem has been studied and various heuristics are proposed in literature. This paper presents a novel framework for task scheduling problem based on Ant colony optimization (ACO). The inherent parallelism of this heuristics is exploited to be implemented effectively on multicore processors. The performance of the algorithm is demonstrated by the time taken for producing effective schedules for random task graphs.**

Index Terms— **Ant Colony Optimization, Directed Acyclic Graph (DAG), Parallel programming, Multi core processors, Task scheduling.**

I. Introduction

Parallel programming systems offer a promising and effective alternative choice for high performance computing. A parallel program is a collection of separate co-operating and communicating modules called tasks and processes. Tasks can execute in sequence or at the same time on two or more processors. Task mapping distributes the load of the system among its processors so that the overall processing objective according to given criteria is maximized. An efficient allocation strategy avoids the situation where some processors are idle while others have multiple jobs queued up. The task scheduling activity determines the execution order. To meet the computational requirements of a larger number of current and emerging applications, a satisfactory algorithm for task matching and scheduling is able to strengthen the parallelization functions.

One of the key challenges of such heterogeneous systems is the scheduling problem. Given an application modeled by a dependence graph, the scheduling problem deals with mapping each task of the application onto the available processors in order to minimize makespan.The task scheduling problem has been solved for years and is known to be NP complete [2]. Several heuristic algorithms are proposed in literature to solve this problem. These heuristics are classified into different categories such as list scheduling algorithms, clustering algorithms, duplication based algorithms [3].These algorithms for homogenous systems. These algorithms suffer from some limitations. For example, the solution quality is not guaranteed for large sized problems.

Reliability of the systems in the heterogeneous systems has a vital role in scheduling the application on to the processors. As heterogeneous systems become larger and larger, the issue of reliability of such systems needs to be addressed. This problem can be prevented by a constructive algorithm based approach, called Ant colony Optimization (ACO).

ACO is a meta heuristic method that is inspired from models of cooperative food retrieval in real ants. A set of agents, called artificial ants, are adopted to represent the behavior of real ant colonies. These ants work cooperatively and communicate indirectly through artificial pheromone trails. An ant constructs a problem solution iteratively by traveling a construction graph. Each edge of the construction graph represents the possible partial solution that the ant can take according to a probabilistic state transition rule. This rule provides a positive feedback mechanism to accelerate convergence and also keeps premature solution stagnation from happening. Due to the successful elaborate properties of ACO, numerous algorithms based on the ACO metaheuristic have been applied to many difficult combinatorial optimization problems from various fields of science and engineering. This work supports the ACO based algorithm for solving the task matching and scheduling problem.

II. Task scheduling Problem

Generally, a parallel program is decompose into a set of tasks in the parallel (cluster based) systems. A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor. These tasks can be characterized by a node and edge

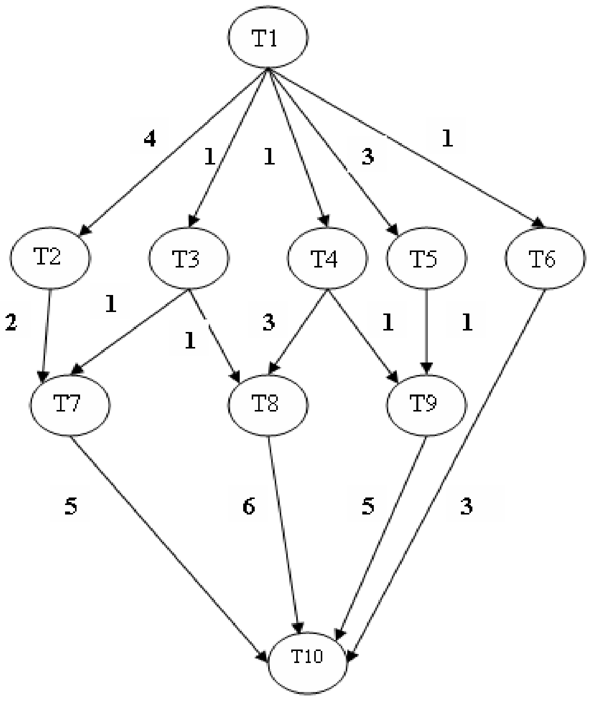

Fig.1 Example of DAG

weighted DAG, G=(V,E) where V denotes the set of tasks and E is the set of directed edges. The set V={t_1,t_2,……….t_n} consists of tasks to be executed. The size of the tasks in the application is represented by the symbol (n) that means the number of tasks. Each and every task are related by directed edge graph represents the communication directions between tasks and precedence constraints (i.e. data dependency) by $E_{ij.}$ A directed edge e_{ij} $\in$ E indicates the data dependency constraint exists between the tasks t_i and t_j so that t_i must be completed before t_j to start. The edge e_{ij} also represents that t_i denotes the immediate predecessor of t_j and t_j is the immediate successor of $t_{i.}$ .In a task graph, an entry task $t_{entry\ is}$ subjected to pred (t_{entry}) = $\varnothing$, and an exit task t_{exit} is subjected to respectively. Without loss of generality, each task graph is assumed to have exactly one entry task and one exit task for each task graph. As long as a task graph has multiple entry tasks, these tasks are connected to a pseudo entry-task that has zero load weight and zero capacity edges. Similarly, a pseudo exit task with zero load weight and zero capacity edges can be added to the graph to connect the multiple exit-tasks. Fig. 1 shows an example of a DAG comprising of 10 tasks to illustrate these definitions graphically. As revealed in Fig. 1, the immediate successors of t_2 are $t_7$and t_8; the immediate predecessors of t_9 are t_4 and t_5.

In the Directed Acyclic graph (DAG) model, each node label gives the execution time for the corresponding task and each edge

TABLE .I
Executed Time of Completion cost matrix (ETC)

	P1	P2	P3
T1	5	10	15
T2	10	5	25
T3	8	24	3
T4	2	5	8
T5	8	2	5
T6	10	8	4
T7	4	15	10
T8	6	8	4
T9	3	6	9
T10	0	0	0

label gives communication time required to pass data from one node to another if the two nodes are assigned to different processors during program execution. A task cannot start until all of its predecessor tasks are complete. During the task scheduling all tasks under go the non preemption scheduling method .i.e they doesn't make any interruption among them.

The heterogeneous computing system is a set P of p heterogeneous machines (processors) connected in a fully connected topology. It is also assumed that:

- any machine (processor) an execute the task and communicate with other machines at the same time.
- Once a machine (processor) has started task execution, it continues without any preemption, and after completing the execution it immediately sends the output data to all children tasks I parallel.

Fig.1 represents an example of an application modeled by a DAG G= (V, E) where V={t_1,t_2,……….t_{10}} represents the set of 10 tasks to be executed, and the set of e weighted, directed edges E represents the communication requirement between tasks. Table I represents the computation cost matrix of the heterogeneous computing system (composed of three processor P_1, P_2, and P_3).The communication cost, COMM (t_i, t_j, PE (t_i), PE (t_j)), can be adopted to evaluate the time taken by PE (ti) to transmit an essential relevant message to PE (tj). This cost is given by

$$\mathrm{COMM}\left(t_i,t_j,\mathrm{PE}(t_i),\mathrm{PE}(t)_j\right) = \mathrm{D}\left(t_i,t_j\right)*\mathrm{R}(\mathrm{PE}\left(t_i,\mathrm{PE}\left(t_j\right)\right) \quad (1)$$

Notably, if two tasks are scheduled on the same PE, then the data transmission rate can be neglected since the intra-communication rate is much less than the intercommunication rate; and therefore the communication cost becomes zero.

Let COMP (t_i, PE (t_i)) denote the computation cost that represents the time taken by PE (t_i) to calculate a given task t_i. Therefore the completion time of task t_j, FT (t_j), can be expressed as follows

$$\mathrm{FT}(t_j) = \max_{\forall t_i \in Pred(t_j)} (FT(t_i) + COMM(t_i, t_j, PE(t_i, t_j))) + COMP(t_j, PE(t_j)) \quad (2)$$

Consequently, the objective of task matching and scheduling is to minimize FT(t_{exit}).

III. Proposed Aco Algorithm

The proposed ACO algorithm is performed in following distinct steps: initialization, the selection of processors, the selection of tasks, local pheromone update and global pheromone update. The initialization step reads in the related information about amounts of tasks, the workload of each task, numbers of processors, and processing ability of processing elements in the task matching and scheduling problem, and the configuration of parameters in the ant colony algorithm.

According to the state transition rules, each artificial ant selects processors and tasks, respectively. After the selections of task and processor, the local pheromone updating rules are applied. Repeat the steps including the selection of processors, the selection of tasks and local pheromone update, until each artificial ant constructs a feasible solution. The process for all the artificial ants to obtain one solution is called iteration. The local search procedure can be adopted to improve the obtained solution once each iteration is finished. After the local search procedure, the global pheromone updating rules I and II are applied to proceed the global pheromone update. Finally, check whether the criterion of finishing the algorithm is satisfied or not. If it is satisfied, export the current optimal solution minimum execution time. Otherwise, continue the next iteration until the criterion is satisfied.

A. State-transition rules

The proposed algorithm applies two state-transition rules to processing element selection and task selection, respectively.

Rule I. Processing element rule: Each artificial ant moves from task i to processing element u according to the probability calculated by

$$\mathrm{Prob(i,u)} = \frac{(\tau(i,u)*[\eta(i,u)]^{\beta})}{\Sigma_{s\in p}(\tau(i,s)*[\eta(i,s)]^{\beta})}\ .$$

(3)

where $\tau(i,u)$ denotes the amount of pheromone on the edge i,u. Heuristic function $\eta(i,u)$= 1/FT(i), denotes the visibility from i to u. β is the parameter that determines the influence of the heuristic function. Here β= 1 + ln (number of iterations) since the effect of the heuristic function can be changed according to the number of iterations. Also, s belongs to the processing elements set P.

Rule II. Task selection: Each artificial ant moves from processing element u to task j according to the probability calculated by

$$\mathrm{Prob(u,j)} = \frac{(\tau'(u,j)*[\eta'(u,j)]^{\beta})}{\Sigma_{t\notin tabu}(\tau'(u,t)*[\eta(u,t)]^{\beta})}\ . \quad (4)$$

where t is a set that records two kinds of tasks: one is that the tasks have not been executed, and the other is the kind of the tasks whose immediate predecessors have been executed. Heuristic function

$$\eta'(\mathrm{u,j}) = \mathrm{BL(j)/BL}(t_{entry})\ . \quad (5)$$

BL (j) represents the bottom-level of task j and it is the maximum value of the sum of minimum computation cost and average communication cost in all paths that the task j takes to the exit task t_{exit}.

B.Pheromone updating rules

The proposed algorithm has two kinds of pheromone updating rules, namely local pheromone updating rules and global pheromone updating rules. The local pheromone updating rules are applied while the ants construct a solution.

The global pheromone updating rules are applied only to edges which belong to the best-found solution.

B.1 Local pheromone update:

When a solution is constructed, the artificial ants proceed to local pheromone update according to the following rules:

Rule I. Update the pheromone on the edge (i, u) chosen by the steps mentioned earlier according to the following function.

$$\tau'(\mathrm{u},\mathrm{j}) = (1-\varphi) * \tau'(\mathrm{u},\mathrm{j}) + \varphi.\tau_0 . \quad (6)$$

where φ denotes the pheromone decay parameter, and τ_0 is the initial value of pheromone on all edges. Here, τ_0 is set to be 0.5.

Rule II. Update the pheromone on the edge (u, j) chosen by the steps mentioned earlier according to the following function

$$\tau'(\mathrm{u},\mathrm{j}) = (1-\varphi) * \tau'(\mathrm{u},\mathrm{j}) + \varphi.\tau_0 . \quad (7)$$

B.2 Global pheromone update:

The global pheromone update can be proceeded according to the rules as follows.

Rule I. Update the pheromone on the path from task i to processing element u based on the following function

$$\tau(i,u) = (1-\rho) * \tau(i,u) + \Delta.\tau(i,u) \quad (8)$$

where ρ denotes the pheromone evaporation rate, and $\Delta.\tau(i,u)$

$$. \left\{ \frac{Q * S_{level}(t_{entry})}{FT(t_{exit})} * \frac{CCR}{DegofParallelism} \right. \quad (9)$$

$S_{level}(t_{entry})$ is the static level of task tentry, static level of task tentry, which is the largest sum of computations from the task tentry to the exit task texit. The CCR represents the communication to computation ratio. The CCR of a DAG is defined as its average communication cost divided by its average computation cost. The degree of parallelism is defined as the graph parallelism over the number of Processors. The graph parallelism can be computed by the ratio of the sum of all computation costs over the sum of computation costs along the longest path (critical path) [9]

Rule II. Update the pheromone on the path from processing element u to task j based on the following function

$$\tau'(u,j) = (1-\rho) * \tau'(u,j) + \Delta.\tau'(u,j) \quad (10)$$

Where $\Delta.\tau'(u,j)$ =

$$\left\{ \frac{Q * S_{level}(t_{entry})}{FT(t_{exit})} * \frac{CCR}{DegofParallelism} \right. . \quad (11)$$

if (u,j) belongs to t.

IV. PROPOSED FRAMEWORK

The proposed framework deals with the application which contains 10 kinds of tasks represented as nodes, dependant on each other and their dependencies are given by the edges linked with each other node. These tasks can be characterized by a node and edge weighted DAG, G=(V,E) where V denotes the set of tasks and E is the set of directed edges as on Fig.1. When not mentioned explicitly, the parameter values of the ACO algorithm used for the test runs are chosen as follows. The number of ants per generation is k=5. The number of ant generations per run of the algorithm is at most 10.We set τ (i,u)=0.1 initially, η(i,u)= i/FT(i) at every time from ETC matrix, β= 1 + ln(number of iterations) initially and is decreased linearly so that it becomes zero after 50% of the (maximal) number of generations.

V. SIMULATION RESULTS AND DISCUSSION

To test the effectiveness of the ACO algorithm for the task scheduling, the solutions were obtained for random task graph. Random task graph were generated using the method as proposed in [3].The input parameters of the directed acyclic graph generation algorithm were set with the following values:

- Number of nodes (tasks) in the graph , V.the value of V is assigned from the set of {10,20,50,100,200}
- Shape parameter of the graph, α . The height of a DAG is randomly generated from a uniform distribution with mean equal to $\sqrt{\frac{v}{\alpha}}$ .The width for each level in a DAG is randomly selected from a uniform distribution with mean equal to $\alpha \times \sqrt{v}$. α is assigned value from the set {0.5,1.0,2.0}
- Out degree value for all nodes are randomly generated from a uniform distribution with mean equal to out degree.
- Communication to computation ratio, CCR.CCR is the ratio of the average communication cost to the average computation cost.CCR={0.1,1,2,10}
- Average computation cost in the graph,ACC. This is the average time required to complete the task on all of the

available processors. Computation costs are generated randomly from a uniform distribution with mean equal to ACC.The value of ACC is from set {10,15,100,200}

- Average communication cost is derived as avg_comm=CCR*ACC.

- Heterogeneity factor, β .This value indicates the range percentage of compution cost on processors.The computation cost of each task v_i on each processor p_j in the system is randomly set from the range :

$$ACC \times \left(1 - \frac{\beta}{2}\right) \leq w_{ij} \leq ACC \times \left(1 + \frac{\beta}{2}\right)$$

The Fig.1 and Table .I are the output of the random task graph generator .

Then the parallelizing ACO and its implementation is done by C, were we made the task matching and scheduling for the given application to provide best results with low time complexity so far to the knowledge of us. We just made the comparison of execution time of each task with their appropriate processor and find the best matching solution through the Rule I of the state transition rule and the tasks are allocated and scheduled in their matched processor by the bottom level approach of topological sorting order through the Rule II.

VI. Conclusion And Future Work

As larger and larger infrastructures are available to execute distributed applications, scheduling becomes a crucial issue.In this work, we described a effective mapping and scheduling algorithm which is an essential component of heterogeneous computing, in order to efficiently utilize the available architectural features of a given heterogeneous processing system. In this paper, we presented a non-preemptive, compile time scheduling heuristic that uses dynamic priorities based on the bottom level. To map and schedule directed acyclic graphs onto heterogeneous processors, with the objective of minimizing the schedule length. This work presents a novel framework on ACO-based approach to find the optimal solution for the task matching and scheduling problem. The future work concentrates on dynamic scheduling and comparison with many other heuristic algorithms with more number of possible constraints in a possible way to parallelize the work in the multicore environment .

References

[1] C.W. Chiang, Y.C. Lee, C.N. Lee and T.Y. Chou "*Ant colony optimization for task matching and scheduling"*, IEE Proc.-Comput. Digit. Tech., Vol. 153, No. 6, November 2006

[2] Freund, R.F.Siegel, H.J. "*Heterogeneous processing"*, IEEE Comput.Vol.26, no.6.pp.13-17, 1993.

[3] Haulk Topcuoglu, Salim Hariri, and Min You Wu, "*Performance Effective and Low complexity Task Scheduling for Heterogeneous Compuitng*", IEEE Trans. On Parallel and Distributed Systems.

[4] A.K.Kulatunga, D.K.Liu, G.Dissanayake ARC Centre of excellence in Autonomous systems faculty of Engineering, University of Technology, Sydney.

[5] M.Dorgio, V.Maniezzo & A.colorni "*Ant System Optimization by a colony of Co operating agents*", IEEE Trans. Syst. Man Cybern. B, 1996, 26, (91), pp. 29–41

[6] Kwok, Y.K., and Ahmad, I.: "*Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors*",

IEEE Trans. Parallel Distrib. Syst., 1996, 7, pp.506–521.

[7] Dorigo, M., and Gambardella, L.M.: "*Ant colony system: a cooperative learning approach to the traveling salesman problem*", IEEE Trans. Evol. Comput. 1997, 1, (1), pp. 53–66

[8] Bank, M., Honig, U., and Schiffmann, W.: "*An ACO-based approach for scheduling task graphs with communication costs*". Proc. Int. Conf.on Parallel Processing (ICPP'05), 2005, pp. 623–629

[9] Wu, M.Y., Shu, W., and Gu, J.: "*Efficient local search for DAG scheduling*", IEEE Trans. Parallel Distrib. Syst., 2001, 12, (6),pp. 617–627

[10] T. Stützle and H. H. Hoos, "*Max-min ant system,*" Fut. Gener. Comput.Syst. vol. 16, no. 8, pp. 889–914, June 2000.