Applying Ant Colony Optimization to the Partitioned Scheduling Problem for Heterogeneous Multiprocessors

Hua Chen and Albert M. K. Cheng

Real-Time Systems Laboratory
Department of Computer Science
University of Houston, TX 77204, USA

Abstract

The problem of determining whether a set of periodic tasks can be assigned to a set of heterogeneous processors in such a way that all timing constraints are met has been shown, in general, to be NP-hard. This paper presents a new algorithm based on Ant Colony Optimization (ACO) metaheuristic for solving this problem. Experimental results show that our ACO approach can outperform the major existing methods. In addition to being able to search for a feasible assignment solution, our ACO approach can further optimize the solution to reduce its energy consumption.

1. Introduction

The heterogeneous computing environment is well suited to meet the computational demands of diverse tasks. However, a direct issue of this computing environment is that the same code chunk may need different execution times upon different processors. For example, a routine responsible for intensive mathematical calculation may execute much faster on a floating-point coprocessor than on a traditional CPU (e.g. 80386). Therefore, the implementation of real-time systems upon such platforms requires more work than upon uniprocessor or homogeneous multiprocessor platforms.

In this paper, we study the partitioned scheduling where each task is exclusively assigned to a specific processor. In particular, this paper is focused on the off-line version of the task assignment problem (TAP), that is, determining *which* task goes *where* such that each of the tasks can be assigned to a specific processor without violating its computing capacity, based on the assumption that the problem instance will not change with time. For each processor, the Earliest-Deadline-First (EDF) algorithm is employed to schedule all tasks assigned to it. There are no precedence constraints among the tasks. The implications of inter-task communication are also completely ignored in this research.

The related work falls into two categories: heuristic methods and efficient approximation algorithms. In [6], Braun et al. compared 11 heuristics for matching (i.e. assigning) and scheduling a set of independent tasks onto heterogeneous computing environment, and the goal was to minimize the makespan. In [4] and [5], Baruah proposed a polynomial time algorithm which is guaranteed to find a feasible assignment solution only if the problem instance satisfies certain constraints. To the best of our knowledge, this is the first paper that applies the Ant Colony Optimization (ACO) algorithms to solving the partitioned scheduling problem for heterogeneous multiprocessors.

2. Models and Problem Statement

HMP = $\{P_1, P_2, ..., P_m\}$ denotes an arbitrary Heterogeneous Multiprocessor Platform with m preemptive processors based on CMOS technology. Each processor P_j runs at variable speed according to the type of task it is performing. $s_{i,j}$ denotes the clock frequency, i.e., the speed, of P_j for a particular task T_i. $e_{i,j}$ denotes the execution time for T_i on processor P_j. $e_{i,j}$ and $s_{i,j}$ are correlated by : $e_{i,j} = c_i / s_{i,j}$, where c_i is the number of cycles to execute T_i. The energy consumption of T_i on P_j, $E_{i,j}$, is given by:

$$E_{i,j} = Power_{i,j} \cdot e_{i,j} \approx (C_{ef} \cdot \frac{s_{i,j}^3}{k^2}) \cdot e_{i,j} = \frac{C_{ef}}{k^2} \cdot c_i \cdot s_{i,j}^2$$

where C_{ef} and k are constants. Therefore, the energy consumption incurred by executing T_i on P_j is almost linearly related to the product of c_i and $s_{i,j}$: $E_{i,j} \propto c_i \cdot s_{i,j}^2$.

A Periodic Task Set PTS = $\{T_1, T_2, ..., T_n\}$ is comprised of n real-time tasks. A real-time task T_i is represented by the tuple (e, p), where e is the estimated worst-case execution time (WCET) and p is the period. T_i generates an infinite sequence of task instances of execution time at most e time units each, with instances being generated exactly p time units apart, and each instance has a deadline p time units after its arrival. The utilization matrix U_{n*m} stores the real numbers in $(0,1) \cup +\infty$. The value of $u_{i,j}$ denotes the maximum fraction of the computing capacity of P_j required to execute T_i, and is equal to the product of $e_{i,j}/p_i$, where p_i is the period of T_i. $u_{i,j}$ is also referred to as *utilization* of T_i on P_j. If task T_i is not suitable to be executed on processor P_j, $u_{i,j}$ is set to $+\infty$.

The TAP can be formally described as follows. Given HMP and PTS, determine whether there is a solution to assigning each of the tasks in PTS to a specific processor in HMP in such a way that the accumulative utilization of all tasks on any processor is no greater than the utilization bound of the EDF algorithm for a preemptive uniprocessor. This problem is proven to be NP-hard in [4].

3. Applying ACO to the TAP

3.1 Construction Graph and Constraints

Intuitively, the TAP can be represented by a bipartite graph with two categories of nodes: PTS and HMP. A task is mapped to a PTS node; a processor is mapped to a HMP node. There is an edge between a PTS node and a HMP node if and only if the corresponding task can be assigned to the corresponding processor without exceeding its available computing capacity. The cost of assignment is directly related to the utilization of the task upon the processor.

To model the process in a more straightforward manner, we use the construction graph that is derived from the utilization matrix. Below is a sample construction graph.

	P_1	P_2	P_3
T_1	$u_{1,1}$	$u_{1,2}$	$u_{1,3}$
T_2	$u_{2,1}$	$u_{2,2}$	$u_{2,3}$
T_3	$u_{3,1}$	$u_{3,2}$	$u_{3,3}$
T_4	$u_{4,1}$	$u_{4,2}$	$u_{4,3}$

Figure 1: Construction graph of 3 processors and 4 tasks

In Figure 1, T_i ($1 \le i \le n = 4$) represents the *i*-th task, P_j ($1 \le j \le m = 3$) represents the *j*-th processor, and $u_{i,j}$ represents the utilization of the *i*-th task on the *j*-th processor. With this construction graph, we can transform the TAP into a traveling ant problem. Specifically, given a "chessboard" of *n* rows and *m* columns, and each of its cells is associated with a value in $(0,1) \cup +\infty$, an ant seeks to travel across the chessboard in such a way that *all* of the following constraints will be satisfied: (1) one and only one cell is visited for each of the rows; (2) the accumulative value of the visited cells on the same column is no greater than 1. In the rest of this paper, "*tour*" and "*solution*" are used interchangeably; a pair of (*Task*, *Processor*) means: *Task* is assigned to *Processor*.

3.2 Pheromone Trails

Due to the close relationship between the TAP and the bin-packing-problem (BPP), we examined the pheromone trails scheme proposed in [2], in which Levine et al. applied Max-Min Ant System (MMAS) framework to the BPP. In [2], the pheromone trial $\tau(i, j)$ encodes the favorability of having item of size *i* and item of size *j* in the same bin. This pheromone trial is changed because of the apparent differences between the TAP and the BPP. First, in practice, the utilization overlap among periodic tasks is much smaller than the size overlap among items. Hence, encoding utilization value instead of task identity will not give us a compact pheromone matrix. Second, because the utilization of a task is tightly coupled with a specific processor, it is straightforward to see that we need to establish a 1:1 mapping between a pheromone trial and a pair of (*Task*, *Processor*). In other words, we need the processor identity as well as the task identity to index the pheromone trial.

Based on preceding observations, we use the pheromone trial $\tau(i, j)$ to encode the favorability of assigning task T_i to processor P_j. From the construction graph point of view, $\tau(i, j)$ is used to encode the desirability of adding cell (T_i, P_j) to the tour of an ant.

3.3 Heuristic Information

In [2], the heuristic favorability of an item is directly related to item size. This is further adapted because the utilization value of a task can be different upon different processors. This nature opens another direction for artificial ants to find good solutions: The fraction of computing capacity for an individual task, T_i, can be reduced by assigning T_i to a processor on which the execution time of T_i is shorter.

Apparently, the fraction of computing capacity for a task is analogical to the item size. Because the First-Fit-Decreasing (FFD) heuristic cannot handle the items of changing sizes, we designed a more sophisticated heuristic which will bias task assignment toward those tasks that use more computing capacity, and bias the choice of processors in such a way that the processor needs a relatively small amount of its available computing capacity to perform the task. Basically, the new heuristic scheme extends FFD with the attempt to lower the cost of computing capacity incurred by a task. Starting with a set of processors with their accumulative utilization set to 0, the ant repeatedly chooses a pair of (*Task*, *Processor*) in such a way that the pairs with following properties will be favored: (1) the execution time of *Task* upon *Processor* is shorter than upon other processors; and (2) Assigning *Task* to *Processor* minimizes the laxity in the computing capacity of *Processor*. The first property is aimed at saving the *overall* computing capacity of HMP, and the second property is inspired by the FFD algorithm to make more use of the computing capacity of an *individual* processor. Formally, given the current partial solution s under construction, the heuristic desirability of adding cell (T_i, P_j) to an ant's tour is given by:

$$\eta(i,j,s) = \frac{m \cdot (1 + U_j(s) + u_{i,j})}{Rank(j,i,s)}$$

In this equation, m is the number of processors in HMP; $U_j(s)$ is the accumulative utilization of processor P_j ; $Rank(j,i,s)$ is an integer bounded in $[1,m]$, and it reflects the relative quickness of executing T_i on P_j among all eligible processors. If T_i is not suitable to be executed on P_j, or T_i will overflow the spare computing capacity of P_j after s is formed, $\eta(i,j,s)$ is undefined.

3.4 Solution Construction

Starting with a set of periodic tasks and a set of heterogeneous processors, the artificial ant will stochastically assign the tasks one by one to the processors, until either all tasks are assigned to some processor, or none of the remaining tasks can be assigned to any processor without exceeding its spare computing capacity. If an ant stops with at least one unassigned task, the resulted tour is called an *infeasible* tour. On the other hand, if an ant stops with all tasks being assigned, the resulted tour is called a *feasible* tour. If no stop condition is met, the current tour is called a *partial* tour. For the selection of the next pair of (*Task*, *Processor*) (i.e. the next cell to visit), the ant uses a weighted average of pheromone and heuristic distributions over the set of eligible pairs. Formally, given the current partial solution *s*, the probability that an ant will assign task T_i to processor P_j is given by:

$$p(s,i,j) = \begin{cases} \dfrac{[\tau(i,j)] \cdot [\eta(i,j,s)]^{\beta}}{\sum_{(i',j') \in N(s)} [\tau(i',j')] \cdot [\eta(i',j',s)]^{\beta}} & \text{if } (i,j) \in N(s) \\ 0 & \text{otherwise} \end{cases}$$

In this equation, $N(s)$ denotes the set of eligible pairs of (*Task*, *Processor*) after s is formed; $\tau(i,j)$ denotes the pheromone trial of (T_i, P_j) ; $\eta(i,j,s)$ denotes the heuristic desirability of adding (T_i, P_j) to *s*; Parameter β determines the relative influence of the heuristic information on the decision of an ant.

3.5 Pheromone Update

We followed the MAX-MIN Ant System (MMAS) in [3] for the update of pheromone trials. Regarding pheromone trial limits, an estimate of the upper bound, $f(s^{best})/\rho$ [1], is used to define $\tau_{\max}$, where s^{best} is the best-so-far solution, and ρ is the evaporation rate of pheromone trails. $\tau_{\min}$ is defined to be $\tau_{\max} \cdot \gamma$, where γ is a parameter that indicates the ratio of the lower bound to the upper bound of pheromone trails. $\tau(i,j)$ will be increased every time task T_i is assigned to processor P_j in the best solution. So, we get:

$$\tau(i,j) = \begin{cases} (1-\rho) \cdot \tau(i,j) + f(s^{best}) & \text{if } (i,j) \in s^{best} \\ (1-\rho) \cdot \tau(i,j) & \text{otherwise} \end{cases}$$

s^{best} here denotes the best-so-far solution or the iteration-best solution. $f(s)$ is a quality function which measures how good the solution *s* is, and is given by:

$$f(s) = Award(s) + (MaxEC - EC(s)) / MaxEC$$

Two factors are taken into consideration for the quality of a solution. The first one is the number of EDF-schedulable tasks along the time dimension, and the second one is the energy consumption along the resource dimension. Informally, $Award(s)$ is defined to be the number of assigned tasks in solution s. Because the assigned tasks never overflow the computing capacity of any processor, $Award(s)$ reflects the distance between solution s and a feasible solution. $EC(s)$ is an estimate of the energy consumption of solution s, and is normalized by the maximum possible energy consumption $MaxEC$. Because the normalized value is bounded in $[0,1)$, this equation implies that the best-so-far ant is closest to finding a feasible solution among all ants since the algorithm started.

4. Experimental Results

We tested our ACO algorithm on a variety of randomly generated problem sets, each of which is composed of numerous problem instances. A problem set is characterized by u[*m*]x[*n*], with the interpretation that each of its problem instances has *m* processors and *n* tasks. A problem instance is generated using the following method.

Initially, a $1 \times n$ period matrix, P , of integer values is created by setting its element p_i , the period of task T_i , to a uniform random number in $[100,300]$. Then, a $1 \times n$ clock cycle matrix, *C*, of integer values is created by setting its element c_i , the number of cycles to execute T_i , to a uniform random number in $[20000,100000]$. Next, a $n \times m$ speed matrix, *S*, of integer values is created by setting its element

$s_{i,j}$ to a uniform random number in $[1000,5000]$. Finally, a $n \times m$ utilization matrix, U, is generated by setting its elements $u_{i,j}$ to the product of $c_i/(s_{i,j} \cdot p_i)$.

Our ACO approach is compared to a Genetic Algorithm (GA) heuristic [6] and Baruah's approximation algorithm based on Integer Linear Programming (ILP) technologies [5]. Our choice of the Linear Programming (LP) implementation is the simplex algorithm available from GNU Linear Programming Kit 4.6. On the GA side, a chromosome sequence encodes an assignment solution. To simplify the implementation of crossover and mutation, we relax the utilization bound so each of the processors can be assigned with an arbitrary number of tasks. The fitness value of a chromosome is the number of processors whose computing capacities have not been violated. The ACO algorithm and the GA stop when any one of two conditions is met: (1) no improved best-so-far solution for 200 consecutive iterations; and (2) total 1000 iterations. The parameter settings for these two algorithms are as follows:

ACO	
β (heuristic weight)	4
ρ (pheromone evaporation rate)	0.02
γ ($Min(\tau)/Max(\tau)$)	0.02
number of ants	80

(a)

GA	
crossover rate	60%
mutation rate	40%
population size	200

(b)

Table 1: Parameter settings for (a) ACO and (b) GA

Upon each instance of every problem set, we run our ACO approach, Baruah's algorithm, and the GA heuristic 10 times each, and report per each approach its average execution time and the number of times that a feasible solution was found across the 10 runs. Figure 2 and Figure 3 show the side-by-side comparison results upon u6x120 composed of 9 problem instances. It is shown that our ACO approach is faster and more effective than Baruah's algorithm and the GA heuristic.

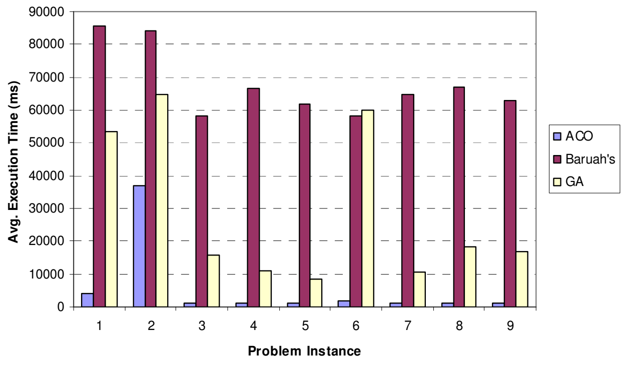

Figure 2: The average execution time of the respective algorithm across 10 consecutive runs is summarized per problem instance.

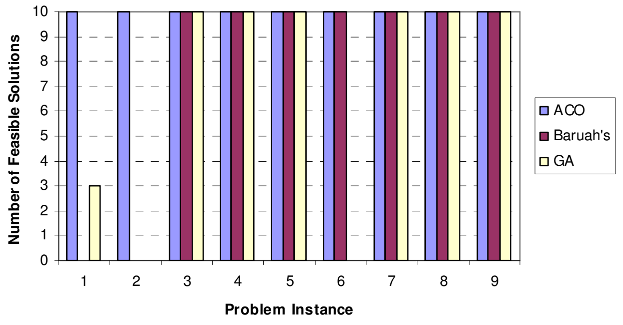

Figure 3: The number of times that a feasible solution was found by the respective algorithm across 10 consecutive runs is summarized per problem instance. Note that Baruah's didn't find any feasible solution for problem instance #1 and #2; and GA didn't find any feasible solution for problem instance #2 and #6.

5. Conclusion and Future Work

A new ACO approach to the task assignment problem is presented. Preliminary test shows that this approach has better performance than a GA heuristic and Baruah's approximation algorithm. Experimental results regarding the reduction of energy consumption will be presented in a future paper. We also plan to investigate the impact of heuristic to the performance of our ACO approach, as well as potentials of adding local search algorithms.

References

[1] M. Dorigo and T. Stützle, *Ant Colony Optimization*, MIT Press, 2004.

[2] J. Levine and F. Ducatelle, *Ant Colony Optimisation and Local Search for Bin Packing and Cutting Stock Problems*, Journal of the Operational Research Society (forthcoming), 2003.

[3] T. Stützle and H. Hoos, *MAX-MIN Ant System*, Future Generation Computer Systems, 16(8), 889-914, Nov. 1999.

[4] S. Baruah, *Task partitioning upon heterogeneous multiprocessor platforms*, RTAS, 2004.

[5] S. Baruah, *Partitioning real-time tasks among heterogeneous multiprocessors*, ICPP, 2004.

[6] T. Braun et al. *A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems*, Journal of Parallel and Distributed Computing 61, 810-837, 2001.