J. Parallel Distrib. Comput. 71 (2011) 132–142

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Assigning real-time tasks to heterogeneous processors by applying ant colony optimization[☆]

Hua Chen[a], Albert Mo Kim Cheng[b,*], Ying-Wei Kuo[b]

[a] Planning, Scheduling & Blending Group, Aspen Technology Inc., Houston, TX 77042, USA

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

ARTICLE INFO

Article history:
Received 16 March 2010
Received in revised form
16 August 2010
Accepted 25 September 2010
Available online 15 October 2010

Keywords:
Scheduling
Real-time systems
Multiprocessors
Heterogeneous processors
Ant colony optimization
Power-aware computing
Periodic tasks

ABSTRACT

The problem of determining whether a set of periodic tasks can be assigned to a set of heterogeneous processors without deadline violations 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. A local search heuristic that can be used by various metaheuristics to improve the assignment solution is proposed and its time and space complexity is analyzed. In addition to being able to search for a feasible assignment solution, our extended ACO algorithm can optimize the solution by lowering its energy consumption. Experimental results show that both the prototype and the extended version of our ACO algorithm outperform major existing methods; furthermore, the extended version achieves an average of 15.8% energy saving over its prototype.

1. Introduction

The heterogeneous computing environment is well suited to meet the computational demands of diverse tasks. However, implementing real-time applications upon multiprocessors has been shown to be difficult. A major challenge is that scheduling algorithms need to not only specify an execution order of tasks, but also determine the specific processor to be used. For a homogeneous multiprocessor platform where all processors are identical, assigning tasks to the processors requires solving the Bin Packing Problem (BPP) [14]. The assignment problem becomes more difficult when the computing environment consists of heterogeneous multiprocessors. The additional complexity arises from the fact that a piece of code 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 digital signal processor. Consequently, a periodic task is allowed to have different utilization requirements on different processors. This extension makes BPP an invalid simulation model, with the Generalized Assignment Problem (GAP) taking over the role. While efficient heuristics and polynomial time approximation schemes have been developed for GAP [25,21], their special application to assigning real-time tasks to heterogeneous processors has not been investigated before.

As in [17], here we study the partitioned scheduling problem where each task is assigned to a specific processor and there is no inter-processor migration. Here we consider the off-line version of the task assignment problem (TAP), that is, determining each task's processor assignment without exceeding each processor's computing capacity and violating task deadlines. The computation times and deadlines of the given tasks are known a *priori* and will not change over time. The tasks are independent and thus there are no precedence constraints among them. Also, there is no inter-task communication. We employ the Earliest-Deadline-First (EDF) algorithm [11] to schedule all tasks in each processor.

A metaheuristic is an adaptive algorithmic framework that can be used to define heuristic methods applicable to a wide range of different problems. A particularly successful metaheuristic, Ant Colony Optimization (ACO) [13,22], is inspired by the behavior of real ants, which are able to find the shortest path between food sources and their nest. Living with poor vision (some ant species are completely blind), real ants communicate with each

☆ Supported in part by the National Science Foundation under Award No. 0720856, a 2006 grant from the Institute for Space Systems Operations, and GEAR Grant No. I092831-38963. Parts of this work have appeared in their preliminary forms in [10], a 4-page work-in-progress article published in the WIP session of IEEE RTAS 2005.

* Corresponding author.
E-mail addresses: birch.chen@aspentech.com (H. Chen), cheng@cs.uh.edu (A.M.K. Cheng), ykuo@cs.uh.edu (Y.-W. Kuo).

other or the environment via chemical substances produced by themselves. These chemicals are called *pheromones*. The likelihood of choosing a specific path by an ant depends on the intensity of the pheromones in that path [23]. The path's intensity increases if this path is chosen by many ants and thus more pheromones are deposited, attracting even more ants. On the other hand, if few or no ants choose a path, then little or no pheromone is deposited in this path and thus this pheromone trail evaporates as time passes. ACO is motivated by this collective behavior of ants.

ACO employs simple agents called *artificial ants* to imitate the behavior of ant colonies. These artificial ants communicate with each other through environmental variables modeling the pheromone trails, making it possible to apply this search metaphor to solve combinatorial optimization problems. Starting with the Ant System, a number of algorithmic approaches based on the very same ideas were developed and applied with considerable success to a variety of combinatorial optimization problems from academic as well as real-world applications.

A major advantage of ACO over other metaheuristics such as Simulated Annealing (SA) [5] and Genetic Algorithms (GA) [5] is that the problem instance may change dynamically. In other words, ACO can adapt to the changing problem instance while running continuously. This feature is of great interest in TAP. In many real-time systems, the set of real-time tasks may change dynamically. For example, the Cell processor developed by IBM, Sony and Toshiba supports real-time audio processing and 3D graphic tasks, which may be initiated at arbitrary times. Motivated by this observation, our solution to TAP is based upon ACO. It is new that our algorithm is designed to optimize assignment solutions in terms of both resource utilization and energy consumption. Ritchie [20] has also applied ACO to solve the multiprocessor scheduling problem. However, he only considers single-instance tasks, whereas we tackle periodic tasks with deadline constraints.

There are projects to minimize energy consumption and meet scheduling objectives, including the work in [6,1,26], but they do not use the ACO heuristic. Bunde [6] describes a non-ACO approximation algorithm for multiprocessor scheduling to minimize energy consumption and makespan but the tasks must have equal computation times. Aydin and Yang [1] present a Worst-Fit-Decreasing heuristic for partitioning the tasks in a multiprocessor system to reduce energy consumption while meeting task deadlines but this heuristic is not ACO-based. Zhu et al. [26] also study multiprocessor scheduling of tasks in AND/OR graphs to reduce energy consumption and meet task deadlines but the presented approach is not ACO-based.

2. Models and problem statement

HMP $= \{P_1, P_2, \ldots, P_m\}$ denotes an arbitrary Heterogeneous Multiprocessor Platform with m preemptive processors based on CMOS technology. Each processor P_j is limited to operating only one instruction per clock cycle, and 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. Hence, $e_{i,j}$ and $s_{i,j}$ are correlated by: $e_{i,j} = c_i/s_{i,j}$, where c_i is the number of clock cycles to execute T_i. The energy consumption of T_i on P_j per period, $E_{i,j}$, is given by:

$$E_{i,j} = \text{Power}_{i,j} \cdot e_{i,j} \approx \left(C_{ef} \cdot \frac{s_{i,j}^3}{k^2}\right) \cdot e_{i,j} = \frac{C_{ef}}{k^2} \cdot c_i \cdot s_{i,j}^2$$

where C_{ef} and k are constants [7,27]. Therefore, the energy consumption incurred by executing T_i on P_j is almost linearly related to the product of c_i and $s_{i,j}^2$: $E_{i,j} \propto c_i \cdot s_{i,j}^2$.

A Periodic Task Set PTS $= \{T_1, T_2, \ldots, T_n\}$ 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 $U_{n\times m}$ 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 quotient of $e_{i,j}/p_i$, where p_i is the period of T_i. $u_{i,j}$ is also referred to as the *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 $+\infty$.

Given HMP and PTS, the TAP is a combinatorial optimization problem consisting of two objectives. The first one is called *resource objective*; that is, searching for a solution in which each of the tasks is assigned to a specific processor in such a way that the cumulative utilization of any processor does not exceed the utilization bound of the EDF algorithm [11]. This objective represents a decision problem which is proven to be NP-hard in [2]. The second one is called *energy objective*; that is, minimizing the cumulative energy consumption of all assigned tasks in a solution. In this research, the resource objective takes precedence over the energy objective. This hierarchical relationship is widely applicable to those time critical systems in which the participating tasks have hard deadlines.

3. Related work

The related work falls into two categories: heuristic methods and efficient approximation algorithms that solve various special cases of the general problem. In [5], Braun et al. compared 11 heuristics for matching (i.e. assigning) and scheduling a set of independent tasks onto a heterogeneous computing environment, and the goal was to minimize the makespan. In [2,3], Baruah proposed a polynomial time algorithm which is guaranteed to find a feasible assignment solution if the problem instance meets certain constraints.

3.1. Heuristic methods

Braun et al. [5] compared 11 heuristics for solving the problem of assigning tasks to heterogeneous processors in such a way that the total execution time of the tasks is minimized. The simulation model of their research is rather simple. It is assumed that an accurate estimate of the execution time for each task on each machine is known and contained within an ETC (expected time to compute) matrix. There is no timing constraint for each of the tasks. Some preliminary terms defined in [5] are as follows:

1. *Machine availability time,* mat(m_j)—The earliest time machine m_j can complete the execution of all the tasks that have previously been assigned to it.

2. *Completion time for task t_i on machine m_j*, ct(t_i, m_j)—The machine availability time for m_j plus the execution time of t_i on machine m_j. Hence ct$(t_i, m_j) =$ mat$(m_j) +$ ETC(t_i, m_j). The maximum ct(t_i, m_j) value is called the *makespan*.

The performance criterion used to compare the performances of the heuristics is the makespan. Each heuristic is designed to minimize the makespan. According to their test results, the GA consistently gave the best performance. Motivated by this conclusion, we implemented GA described in the paper with some modifications for our problem domain, and compared it to our ACO approach in our simulation experiments.

3.2. Theoretical analysis

Baruah [3] proposed an efficient (i.e., polynomial-time) algorithm for solving the TAP. His algorithm transforms a TAP instance into an Integer Linear Programming (ILP) problem, and then applies an approximation technique based on the idea of LP *relaxations* to solve the ILP in polynomial time.