Advanced Science and Technology Letters
Vol.54 (Networking and Communication 2014), pp.61-64
http://dx.doi.org/10.14257/astl.2014.54.16

Real-time tasks allocation for heterogeneous network platform

Weizhe Zhang , Xuehui Wang

School of Computer Science and Technology, Harbin Institute of Technology,
Harbin 150001
wzzhang@hit.edu.cn

Abstract. We formulate the energy-aware real-time task scheduling problem into a combination optimization problem. Then, a successful metaheuristic, Shuffled Frog Leaping Algorithm (SFLA) is introduced to reduce the energy consumption.

Keywords: real-time scheduling, heterogeneous multiprocessors, shuffled frog leaping algorithm.

1 Introduction

The problem of scheduling real-time tasks on heterogeneous platforms is a NP-hard problem. However, most of the algorithms do not take the energy consumption into consideration. Moreover, the optimization goals for the number of feasible solutions and the energy saving are conflicting. Current swarm intelligence algorithms are mired in to difficulties. New metaheuristics for energy-aware real time scheduling need further digging.

A metaheuristic designates a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. A particularly successful metaheuristic, Shuffled Frog Leaping Algorithm (SFLA) [1, 2] is inspired by the behavior of the evolution of the frog foraging. The frogs are divided into subgroups named as memeplexes, which have different cultures. Each memeplex performs a local search. The frogs in each memeplex can exchange the ideas. After some evolution steps, memeplexes exchanges the ideas in a shuffled process. Until convergence criteria are satisfied, the algorithm does not stop. In essence, the SFLA combines the benefits of particle swarm optimization (PSO) and the idea of mixing information from parallel local searches to move toward a global solution.

The main objective of this work is to design novel real-time scheduling algorithms not only to satisfy the hard deadline but also to reduce the energy usage. Thus, real-time computing can be a more sustainable and eco-friendly mainstream technology to drive commercial, scientific, and technological advancements for future generations

2 Real-time scheduling model

Real-time tasks are denoted by *T*. Each *T* is identified with its execution time and deadline. If *e* denotes *T*'s execution time in the worst case and *p* denotes *T*'s period, then *T* can be represented with a binary group *(e, p)*. *T*'s task instance arrives every *p* time, and after *p* time this instance meets its deadline. A periodic real-time task set with *n* real-time tasks can be denoted by $TS = \{T_1, T_2, \ldots, T_n\}$. Every T_i ($i = 1, 2, \ldots, n$) is a periodic real-time task and identified with (e_i, p_i).

The processor is denoted by *P*. The clock cycle of T_i on P_j is denoted by $t_{i,j}$ and the maximum execution time of T_i's instance on P_j is denoted by $e_{i,j}$. $e_{i,j}$ can be expressed as , where c_i indicates the number of instructions that T_i's execution needs.

Here *P* is limited to execute only one instruction in a clock cycle. Different processors have different task processing speeds. A processor platform with *m* processors can be denoted by $PP = \{P_1, P_2, \ldots, P_m\}$. Every P_j ($j = 1, 2, \ldots, m$) is a processor.

The schedule model can be described as a schedule matrix . Every element in denotes the relationship of T_i and P_j. As the tasks are indivisible, each has only two kinds of value, either 0 or 1. indicates that T_i is assigned to P_j and indicates that T_i is not assigned to P_j.

The utilization matrix is defined as . Every element in indicates the computing consummation when T_i executes on P_j and it can be calculated from u;,; = e4, A . is a real number in (0,1)u +00 . If T_i can't execute on P_j, then set to +00 . The constraint model contains the following three parts:

k *k*

1)

j□

, (i=1,2,...,n)

j □ 1

n

2) $\square\, x_{i,j}\, {}_{i,j}\ U,\ (j=1,2,\ldots,m)$

3) is either 0 or 1, (i=1,2,...,n;j=1,2,...,m)

where U is the maximum computing capacity that each processor can bear. Here in the experiment it is set to 1.

The Constraint 1) denotes that every task will be assigned completely. The Constraint 2) denotes that each processor's assigned computing capacity can't exceed its maximum computing capacity. The Constraint 3) limits the values of .

The energy consumption of T_i on P_j in a period is defined as $E_{i,j}$, its calculation formula is:

$$s^3$$

$$E_i = {}_{Power_{i,j}x}\, e_{i,j}\, {}^{n}{}_{(C_{ef}x} \qquad)x\ {}_{e_{i,j} = C_{ef}2\ x\ c_i} x\ S_i 2,$$

j

where C_{ef} and k are constants.

The schedule scheme's fitness is defined as follow:

(2)

where *nSchedulable* is the number of the processors whose assigned computing capacity haven't exceeded its maximum computing capacity.

The energy-aware fitness is defined as *Fitness_Energy*, its calculation formula is:

(1)

(3)

where *Fitness* is defined in Equation 2, *Max_Energy* is the theoretical maximum energy consumption, and *Energy* is the energy consumption of the schedule scheme. *Energy* and *Max_Energy* is given by:

(4)

(5)

Energy-aware Real-Time Schedule for Heterogeneous Multiprocessors: according to the mathematical model, the optimal schedule which minimizes energy consumption and satisfies the constraints is defined as follows:

Minimize Fitness_Energy (OptimalSchedule)

s.t.

1) , $(i = 1, 2, ..., n)$

2) , $(j=1,2,...,m)$

3) is either 0 or 1, $(i=1,2,...,n; j=1,2,...,m)$

3 Scheduling algorithm

According to the basic idea of SFLA, the whole process of applying SFLA to assign real-time tasks to heterogeneous processors is as follows:

Step 1. Initialize parameters. Initialize the number of schedule schemes *F* in the population, the number of sub-populations *m*, the number of the schedule schemes *n* in each sub-population. Initialize the iterations of the sub-population and the maximum iterations when P_x can't be improved. Initialize the parameters of ERTS, include the number of tasks, the number of processors, the heterogeneity of TS and PP and the utilization matrix.

Step 2. Generate the original schedule scheme set. Generate *F* schedule schemes randomly in the solution space of the real-time task schedule problem, namely generate *F* frogs.

Step 3. Divide the schedule scheme set into *m* sub-populations. Calculate the fitness of each frog and sort them in descending order according to their fitness. Divide them into *m* sub-populations. Determine P_x of the population, P_b and P_w of each sub-population.

Step 4. Transfer information in each sub-population. Conduct the information transfer mode in each sub-population according to the iterations of the sub-population. The information transfer process searches the solution space of the schedule problem and the frogs will jump to the better schedule schemes.

Step 5. Transfer information between sub-populations. Combine the frogs in each sub-population back to the population and sort them in descending order according to their fitness. Update P_x.

Step 6. Check the termination conditions. If SFLA finds a feasible schedule scheme or the iterations where P_x can't be improved reach the maximum, then stop the operation of SFLA. Else return to Step 3.

References

1. Muzaffar, M., Eusuff and Lansey, K.: Optimization of water distribution network design using the shuffled frog leaping algorithm. Presented at Journal of Water Resources Planning and Management, 2003, pp.210-225.
2. Luo, X., Yang, Y., and Li, X.: Solving TSP with Shuffled Frog-Leaping Algorithm. Proc. ISDA (3), 2008, pp.228-232