A Task Migration Constrained Energy-Efficient Scheduling Algorithm for Multiprocessor Real-time Systems

Liu Zheng
School of Computer Science and Technology
Shandong Economic University
Jinan, Shandong, 250014, China
Email: lzh_48@126.com

***Abstract*—This paper presents a multiprocessor energy-efficient scheduling algorithm for the real-time periodic tasks with task migration constrained policies. We separate periodic tasks into fixed tasks and migration ones, and constrain the number of migration tasks and destination processors executing migration tasks. The algorithm is composed of two phases. Firstly, choosing a processor to sort all of the periodic tasks in a non-increasing order of task utilization, then distributing them to other processors. Secondly, scheduling the migration tasks with the virtual execution windows in the first place, and scheduling the fixed tasks with EDF algorithm in the next place. The experiment results show that compared with arbitrary task migration and no task migration allowed, the performance of energy consumption saving in multiprocessor scheduling is improved greatly with our algorithm.**

Keywords-Multiprocessor scheduling; Energy-Efficient Scheduling; Task Migration; Real-time Periodic Task; Dynamic Voltage Scaling

I. INTRODUCTION

Multiprocessor system has become an effective method to solve these complex real-time applications, because of its high performance. However, as the processor performance enhancing, the energy consumption increases rapidly, and the life-span of the system supported by battery shortens evidently and the system reliability declines fast as well. Therefore, it is important to reduce the energy consumption for multiprocessor systems. In the past decades with the development of VLSI technology, the processors supporting Dynamic Voltage Scaling (DVS) appeared, which can save energy consumption effectively. In order to reduce energy consumption, the supply voltage should be lowered properly assuring all the real-time tasks could be scheduled without missing deadlines.

There have been lots of efforts for energy-efficient scheduling, but most of the previous studies concentrated on the uniprocessor environment. In the field of multiprocessor energy-efficient scheduling, Dakai Zhu, et al. proposed an algorithm with the considerations of shared slack time reclamation [3]. In particular, there are several papers discussing task migration policies for multiprocessor energy-efficient scheduling. In [4], Jian-Jia Chen and Tei-Wei Kuo presented the algorithms in the conditions of arbitrary task migration and no task migration allowed. Shelby Funk and Sanjoy Baruah introduced semi-partition and virtual processor concepts to solve the problem of task migration among different processors [5].

On the basis of the researches above, we design an off-line task assignment algorithm based on a kind of approximate algorithm for the bin-packing problem, and present a virtual task execution windows policy to schedule the migration task executed between two processors.

II. SYSTEM MODEL

A. Energy Model

Most of the processors have a linear relationship between the operating frequency and the supply voltage, and the energy consumption is a strict increasing convex function of supply voltage. According to the characteristics of CMOS logic [1], the relationship between processor energy consumption and supply voltage is: $E \propto V^2$ [2]. So energy consumption saving can be achieved in multiprocessor scheduling with DVS. The mechanism of energy-efficient multiprocessor scheduling is to reduce the supply voltage properly when the workload lowering.

B. Task Model

We assume the task set T consists of n independent periodic tasks ($T=\{\tau_1,\tau_2,.....,\tau_n\}$), and each task τ_i is defined by a two-tuple (c_i,d_i), where c_i is the execution requirement of τ_i, and d_i is the deadline of τ_i. c_i and d_i are integer times to CPU cycle. Each task's relative deadline d_i is equal to its period, and all tasks arrive at time 0. Utilization of τ_i is defined as u_i ($u_i=\frac{c_i}{d_i}$). Each task is invoked repeatedly, and each such invocation is referred to as a job. The tasks we discussed in this paper conform to two principles: 1) the processor can only execute one task at one time, 2) the task can't be executed on different processors in parallel.

This work was supported by the National Natural Science Foundation of China (Grant No.60673151)

C. Multiprocessor Real-time Scheduling System

We assume there are *M+1* homogeneous processors in this system, and select one (which is denoted P) to sort tasks and distribute the sorted tasks to other *M* processors ($P_1, P_2, \ldots\ldots, P_M$), on which the tasks are scheduled. As shown in Fig.1, the functions of the processor P are different from the others.

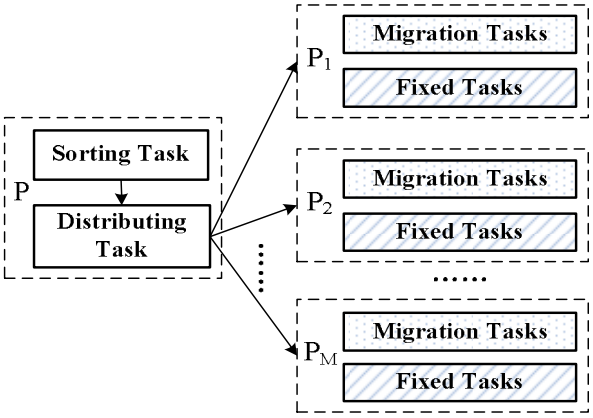

Figure 1. Multiprocessor real-time scheduling system architecture

III. OFF-LINE TASK ASSIGNMENT

A. Off-line Task Assignment Overview

The main characteristic of our off-line task assignment algorithm is that each task is assigned to either one or two processors. Tasks assigned to only one processor are called fixed tasks, while tasks assigned to two processors are defined as migration tasks. Each invocation of a task is called a job. Furthermore, jobs of a migration task can't be scheduled on different processors simultaneously. In the task assignment phase, processor P sorts the tasks by the task utilization firstly, and then decide which task should be distributed as a migration task or a fixed one. For the migration task, the migrating destination should also be assured in the distributing phase. We consider some rules in Best Fit Decreasing (BFD) algorithm[7] (a approximate algorithm for bin-packing problem) to be appropriate for task assignment in this paper.

B. Parameters Explaination

At present, processors only provide a few number of discrete speeds, and the continuous speed assumption does not exist in fact. For the discrete speed processor used in this system, the processor runs at one of the *S* discrete operating frequencies: $f_1, f_2, \ldots\ldots, f_S$ ($f_{min} = f_1 \le f_2 \le \ldots\ldots \le f_S = f_{max}$). We define the sum of all the tasks' utilization as U_{sum} ($U_{sum} = \sum_{\tau_i \in T} \frac{c_i}{d_i}$), then the average utilization (which is denoted U) of all the processors' is equal to $\frac{U_{sum}}{M}$. We select the lowest available frequency (which is denoted f_{sel}) higher than or equal to Uf_{max} as the operating frequency of the processors, that is $f_{sel} = min\{f \mid f \ge Uf_{max}, f \in \{f_1, f_2, \ldots\ldots, f_p\}\}$.

Let $CU[i]$ be the i[th] processor's rest processing capacity that can be allocated, and all the processors' initializing processing capacity is $\frac{f_{sel}}{f_{max}}$, then we have $0 \le CU[i] \le \frac{f_{sel}}{f_{max}}$. $tp_{i,j}$ denotes the utilization of j[th] processor allocated to τ_i. a and b are the number of fixed tasks and migration tasks respectively, with the initialized value 1. Let ft[a] and mt[b] be the a[th] fixed task and the b[th] migration task respectively. With the considerations of the fixed task allocated to only one processor and the migration task allocated to two different processors, we define pft[x] as the processor to which the x[th] fixed task is allocated, and define pmt[y][1], pmt[y][2] as the two processors to which the y[th] migration task is allocated.

C. Off-line Task Assignment Algorithm

Algorithm 1: Off-line Task Assignment

Input: (M, T, $\{f_1, f_2, \ldots\ldots, f_S\}$)

Output: (pft[], pmt[][])

1) sorting all the periodic tasks in a non-increasing order of utilization of each task;
2) **if** $\frac{1}{M}\sum_{\tau_i \in T} \frac{c_i}{d_i} > 1$ or $\exists \tau_i \in T \ \ \frac{c_i}{d_i} > 1$
3) **return** non-existence of any feasible schedule;
4) **for** i=1 to *n* **do** {
5) p = select_processor(u_i);
6) **if** $CU[p] \ge u_i$ **then** //the task is categorized as fixed task
7) $\{tp_{i,p} = u_i;\ \ CU[p] = CU[p] - u_i;$
8) ft[a]=i; pft[ft[a]]=p; a++;}
else { //the rest utilization of this processor can't contain τ_i, this task should be migrated
9) mt[b]=i;
10) **while**(u_i) do {
11) **if** count_mt[p]<2 **then** {
12) **if** $CU[p] \le u_i$ and count_mt[p]=0 **then**
13) $\{tp_{i,p} = CU[p];\ CU[p] = 0;\}$
14) **else if** $CU[p] \le u_i$ and count_mt[p]=1 then
$\{f_{sel}^{\dagger} = \min\{f \mid f > f_{sel}, f \in \{f_1, f_2, \ldots\ldots, f_p\}\};$
$CU[p] = CU[p] + \frac{f_{sel}^{\dagger} - f_{sel}}{f_{max}}$; continue;}
15) **else**
16) $\{tp_{i,p} = u_i;\ CU[p] = CU[p] - u_i;\}$
17) pmt[mt[b]][++count_mt[p]]=p;
18) $u_i = u_i - tp_{i,p}$;
19) p = select_processor(u_i);
else
20) {seek_other(u_i);}
21) b++; }}}}

Functions of select_processor(u_i): It returns the processor which has the most minimal processing capacity more than or

equal to u_i. However, if there are no such processors, it returns the processor which has the most maximal processing capacity.

Functions of seek_other( ): When the processor chosen by select_processor( ) has already been allocated with two migration tasks, then recall select_processor( ) to select another processor. If all processors have hold two migration tasks, then the task should be classified as a fixed one. Next, we choose the processor with the most maximal processing capacity to hold it. If the chosen processor hasn't not enough capacity, the operating frequency of it should be enhanced.

IV. TASK SCHEDULING WITH TASK MIGRATION CONSTRAINED

A. Task Scheduling Overview

According to algorithm 1, let $\tau_i(c_i,d_i)$ be any migration periodic task that is assigned shares $tp_{i,pmt[i][1]}$ and $tp_{i,pmt[i][2]}$ on processors pmt[i][1] and pmt[i][2], respectively. $h_{i,1}$, $h_{i,2}$ denote the fractions of the total execution requirement of the migration task τ_i that should be executed on pmt[i][1] and pmt[i][2], that is $h_{i,1}=\frac{tp_{i,pmt[i][1]}}{u_i}$, $h_{i,2}=\frac{tp_{i,pmt[i][2]}}{u_i}$.We assume that processor px executes the bigger share of migration task τ_i, and the vth job executed on px is denoted x_v. Similarly, py represents the processor executing the smaller share of τ_i, and the wth job executed on py is denoted y_w.

We transform $h_{i,px}$ into the form of fraction in lowest terms, that can be expressed as $\frac{E}{F}$, consequently, we have the equation $h_{i,py}=\frac{F-E}{F}$. τ_i is scheduled F times in the interval $[0,Fd_i]$, and the times τ_i being executed on px and py are E and $F-E$. In order to let the workload proportion between px and py be close to $\frac{h_{i,1}}{h_{i,2}}$ in a great degree at any time, we introduce virtual task execution windows policy in this section.

B. Virtual Task Execution Windows Policy

We assume that the g+1th job of the migration task τ_i is executed on processor s，and the virtual release time and the virtual deadline of the g+1th job are $r(j_{g+1})$ and $d(j_{g+1})$ respectively. Accordingly, the virtual task execution window is $[r(j_{g+1}),d(j_{g+1})]$. $r(j_{g+1})$ and $d(j_{g+1})$ are defined as follows:

$$r(j_{g+1})=\left\lfloor \frac{g}{h_{i,s}} \right\rfloor, d(j_{g+1})=\left\lceil \frac{g+1}{h_{i,s}} \right\rceil$$

The migration task τ_i must be scheduled in the interval of its period, which is equal to τ_i's deadline d_i, so that τ_i should be scheduled on either px or py in $[kd_i,(k+1)d_i]$, where k is a non-negative integer. We design a migration task scheduling algorithm to choose the processor at any given interval ($[td_i,(t+1)d_i]$), t is an arbitrary non-negative integer. For the job of task τ_i, supposing there has been v invocations on px, and w invocations on py.

Algorithm 2: Task Scheduling With Task Migration Constrained

Input: (M, T, $\{f_1,f_2,\cdots\cdots,f_S\}$)
Output: The schedule of all migration tasks and fixed tasks
/*Migration task scheduling phase*/
1) for the migration task τ_i, execute 2)-7) to choose its destination processor;
2) **if** $r(x_{v+1})=t$ **then**
3) the v+1th job invocation of τ_i should be executed on px in $[td_i,(t+1)d_i]$;
4) v++;
5) **else if** $d(y_{w+1})=t+1$ **then**
6) the w+1th job invocation of τ_i should be executed on py in $[td_i,(t+1)d_i]$;
7) w++;
/*Fixed task scheduling phase*/
8) scheduling the fixed tasks according to EDF algorithm in the intervals which not yet allocated to any migration task in the migration task scheduling phase;

V. EXAMPLE

Combining all the algorithms illustrated above, we now turn to a description of the entire algorithms. Supposing there are 4 homogeneous processors（one of them is used only to assign tasks）, with the frequency set {0.3, 0.5, 0.7, 0.8, 1.0}, and 7 periodic tasks all arriving at time 0. The parameters of periodic tasks $\tau_1-\tau_7$ are (1,2), (2,5), (2,5), (7,20), (1,4), (1,10), (2,25) in turn. Because $U=\frac{1}{3}\sum_{i=1}^{7}\frac{c_{\tau i}}{d_{\tau i}}\approx 0.693$ holds, the operating frequency selected is 0.7 .

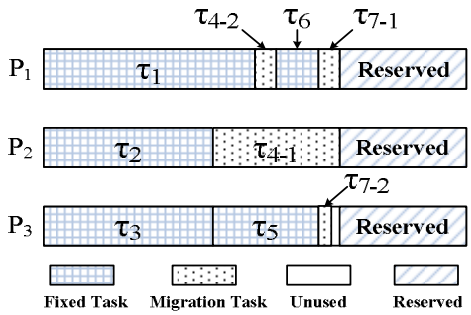

Figure 2. Tasks assignment

Based on the algorithm 1, we can get task assignment result (shown in Fig.2). The processing capacity allocated to tasks and other sections are

$u_{\tau 1}=\frac{1}{2}$, $u_{\tau 2}=\frac{2}{5}$, $u_{\tau 3}=\frac{2}{5}$, $u_{\tau_{4-1}}=\frac{3}{10}$, $u_{\tau_{4-2}}=\frac{1}{20}$, $u_{\tau 5}=\frac{1}{4}$, $u_{\tau 6}=\frac{1}{10}$, $u_{\tau 7-1}=\frac{1}{20}$, $u_{\tau 7-2}=\frac{3}{100}$, $u_{reserved}=\frac{3}{10}$, $u_{unused}=\frac{1}{50}$.

From the example discussed above, τ_7 is distributed to two processors named px and py, on which the utilization share allocated are $h_{7,px}$ (its value is $\frac{5}{8}$) and $h_{7,py}$ (its value is $\frac{3}{8}$). Therefore, for migration task τ_7, the virtual task execution windows and the processor distributions in the interval [0, $8d_i$] are shown in Fig.3.

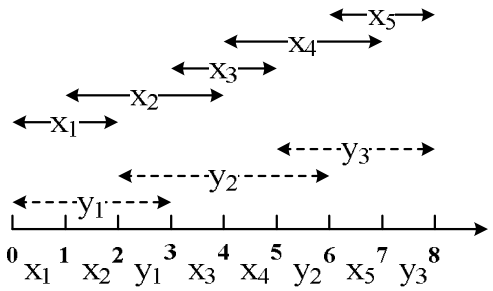

Figure 3. Virtual task execution windows for migration task τ_7

VI. Experimental Results

We provide the performance evaluation of energy consumption using our proposed algorithms compared to the case of arbitrary task migration and no task migration allowed presented in [4].We use intel Xscale processor supporting DVS[6].The relationship between supply voltage and operating frequency is shown in Table I.

TABLE I. Intel Xscale Supply Voltage-operating Frequency Relationship

Frequency(MHz)	1000	800	600	400	150
Voltage(V)	1.80	1.60	1.30	1.00	0.75

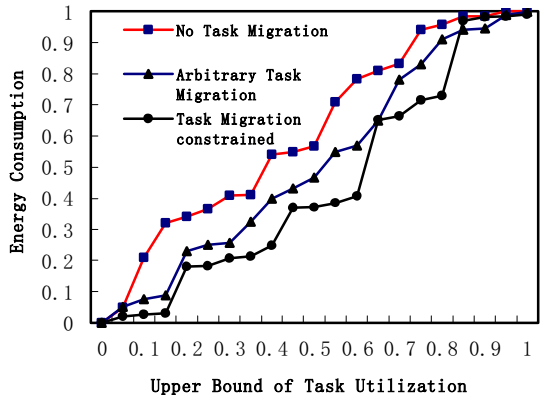

Figure 4. Energy consumption VS tasks utilization

The experiment environments consist of five homogeneous multiprocessors system. We implement the algorithms and a simulator with GNU C in Linux. The upper bound of total tasks utilization ratio varying from 0.0 to 1.0 in increments of 0.05. We suppose the time unit is a CPU cycle. For every given upper bound of total tasks utilization, the experiment program generates randomly 50 task sets, in which periodic task's deadline ranges randomly between 2 and 100. The execution time of periodic task τ_i ranges randomly between 1 and $\lfloor \frac{d_i}{2} \rfloor$. Then we compute the proportion between average energy consumption of the total 50 task sets (i.e. $\frac{1}{50}\sum_{i=1}^{50} E_{Ti}$) and the processor's maximal energy consumption running on the max operating frequency (Shown in Fig.4) . Next, we compute the proportion between average energy consumption of the total 50 task sets and the number of the processors (Shown in Fig.5). We evaluate the case that average total task utilization is equal to 0.5.

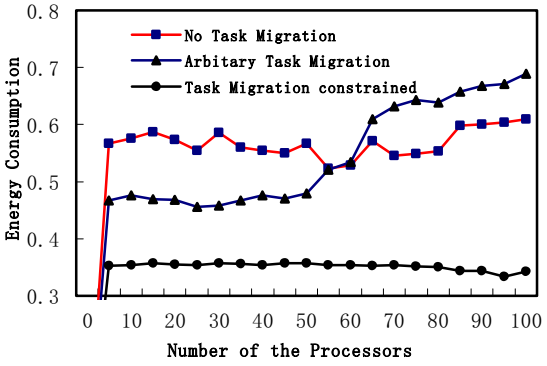

Figure 5. Energy consumption VS number of the processors

VII. Conclusions

This paper targets energy-efficient scheduling of periodic tasks over multiprocessors, where tasks share a common arriving time. Distinct from the past work, this paper proposes task migration constrained policy to lower processors' operating frequency without missing deadline. Experiment results show that our new multiprocessor scheduling algorithms effectively save the energy consumption with task migration constrained considerations.

References

[1] A. Chandrakasan, S. Sheng, and R. Broderson. Lower-power CMOS digital design. IEEE Journal of Solid-State Circuit, 27(4), pp. 473-484, 1992.

[2] Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In Proc. of 18th ACM Symposium on Operating Systems Principles (SOSP'01), Oct. 2001.

[3] Dakai Zhu, Rami Melhem, and Bruce R. Childers. Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems. IEEE Transaction on Parallel and Distributed Systems, vol. 14, no. 7, pp. 686-700, 2003.

[4] Jian-Jia Chen and Tei-Wei Kuo, Multiprocessor Energy-Efficient Scheduling for Real-Time Tasks with Different Power Characteristics, in the 2005 International Conference on Parallel Processing (ICPP05) pp.13-20 in Oslo, Norway, June 14-17, 2005.

[5] Shelby Funk and Sanjoy Baruah. Restricting EDF migration on uniform multiprocessors. Proceedings of the 12th International Conference on Real-Time Systems, Nancy, France, March 2004.

[6] http://developer.intel.com/design/intelxscale/.

[7] Xiaodong GU. Performance Analysis and Improvement for Some Linear On-Line Bin-Packing Algorithms. Journal of Combinatorial Optimization, Vol. 6, No. 4, pp. 455-471, December, 2002.