一种有约束关系的实时周期任务调度算法研究

唐毓毅[1],朱怡安[1,2],黄姝娟[1,2],李凤彬[1]
(1.西北工业大学 计算机学院,陕西 西安 710072;
2.西北工业大学 软件与微电子学院,陕西 西安 710072)

摘 要:在多核嵌入式平台下,针对具有约束关系的实时周期任务,提出一种基于任务关键因子和截止时间的调度算法BVDS(Based on Value and Deadline Scheduling)。该算法以有效利用处理器为原则,根据每个处理器的实际运行情况,为有可能在截止时间前完成的任务分配处理器资源。算法实现分为两个阶段:第一阶段根据任务的到达时间、关键因子以及执行时间构建等待任务链表;第二阶段,在执行过程中,充分考虑不同任务的执行时间以及任务之间的约束关系进行优先级分配。实验结果表明,该算法在牺牲少量处理器利用率的前提下,有效地降低了任务的死限丢失率。

关键词:多核;实时系统;周期性任务;约束关系
中图分类号:TP273.5 **文献标识码**:A **文章编号**:1673-629X(2013)07-0001-05
doi:10.3969/j.issn.1673-629X.2013.07.001

Research on a Real-time Scheduling Algorithm for Periodic Task with Constraint Relation

TANG Yu-yi[1], ZHU Yi-an[1,2], HUANG Shu-juan[1,2], LI Feng-bin[1]
(1. College of Computer, Northwestern Polytechnical University, Xi'an 710072, China;
2. College of Software and Microelectronics, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: In this paper, propose a scheduling algorithm BVDS (based on value and deadline sc-heduling) for periodic task with constraint relation in mutli-core embedded platform. In principle of effective utilization of processor resources, only tasks that are possible to be done before deadline can be assigned to implement. The process of BVDS can be organized into two phrases. In the first phrase, a waiting job list is constructed according to the release time, value and execution time of each job. In the second phrase, dynamically adjust the priority of each job according to its predecessor job and the availability of each processor. Simulation results show that BVDS algorithm can effectively reduce the deadline miss-ratio of tasks.

Key words: multiprocessor; real-time system; periodic task; constraint relation

0 引 言

如今,单芯片多核芯处理器(Chip Multicore Processors,CMP)体系结构的研究已逐渐成熟,并开始应用于各种领域。随着集成电路工艺技术的发展,多核处理器的核数与硬件性能都在成倍的增加,但随之出现的任务调度问题却更为突出,成为系统整体性能提升的瓶颈。

目前研究实时系统的调度算法的文献较多,早期的研究主要集中在硬实时静态调度上,文献[1]证明了RMS是单处理器下的最优静态调度算法。随着实时系统开放、灵活多变的发展趋势,实时调度算法的研究逐渐转向了动态、分布式、软实时以及混合调度[2]。目前能够成功应用的动态调度算法是最早死限优先算法EDF(Earliest Deadline First)[3]和最短空闲时间优先算法LLF(Least Laxity First)[4]。虽然在单处理器下EDF和LLF是最优动态调度算法[5],但是由于二者在每个调度时刻都要计算任务的死限或空闲时间,并根据计算结构改变任务的优先级,系统开销大。因此在具有约束关系的任务上,EDF和LLF不是最优的算法。为任务找出最佳分配方案的问题在通常情况下

收稿日期:2012-10-15 修回日期:2013-01-25 网络出版时间:2013-04-08
基金项目:航天科技创新基金(2011XR60001);航空科学基金(20100753022);西北工业大学基础研究基金(JC20110283)
作者简介:唐毓毅(1987-),女,福建福州人,硕士研究生,研究方向为嵌入式计算;朱怡安,教授,博士生导师,研究方向为高性能计算、云计算、普适计算。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20130408.1600.026.html

是一个NP完全问题[6]。目前对于实时多核处理器任务调度算法方面的研究也较多,但针对具有约束关系的实时周期任务的相关研究却很少见。文献[7]研究了一种基于WFD的启发式方法来提高处理器利用率。文献[8]提出一种新的动态设置优先级的方法。文献[9]所提出的算法是一种基于RM算法的针对有依赖关系的多任务集的多核调度算法。文献[10]提出一种多核系统中基于Global EDF在线节能硬实时任务调度算法,结合动态功耗管理和动态电压/频率调节技术,降低多核系统中任务的执行速度,达到实时约束与能耗节余之间的合理折衷。

文中在考虑不同任务的关键因子、执行时间以及任务之间的约束关系的基础上,提出一种具有约束关系的实时周期任务的多核处理器任务调度算法——基于任务关键因子和完成截止时间的调度算法BVDS(Based on Value and Deadline Scheduling)。

1 基本理论

1.1 任务模型与基本假设

在描述具体的调度算法之前,首先给出任务模型及其参数的定义和表示。

处理器集 $C=\{C_1,C_2,\cdots,C_m\}$ 为含有 m 个具有相同处理能力的同构处理器集。

周期任务集 $T=\{T_1,T_2,\cdots,T_n\}$。每个任务仅有一个直接前驱任务,且任务 T_i 的前驱任务为 $T_{i-1}(i\in[2,n])$。

每个任务都是一个三元组 $T_i=(E_i,P_i,V_i)$,其中:

(1)E_i 表示任务的执行时间,即该任务的一次执行在无中断情况下执行所需的处理器时间;

(2)P_i 表示任务的执行周期,即该任务自启动起,每间隔 P_i 个处理器时间,需要执行一次,显然有 $P_i>E_i$;

(3)V_i 表示任务的关键因子,V_i 的值越大,表明该任务的重要性越高,应保证优先被执行。

任务的一次执行称为一个Job,任务 T_i 对应一个Job集 $J_i=\{J_{i,1},J_{i,2},\cdots\}$,$J_{i,j}=(E_{i,j},P_{i,j},V_{i,j},D_{i,j},S_{i,j})$,其中:

①$E_{i,j}$ 表示 $J_{i,j}$ 的执行时间,且 $E_{i,j}=E_i$;

②$P_{i,j}$ 表示 $J_{i,j}$ 的执行周期,且 $P_{i,j}=P_i$;

③$V_{i,j}$ 表示 $J_{i,j}$ 的关键因子,且 $V_{i,j}=V_i$;

④$D_{i,j}$ 表示 $J_{i,j}$ 的截止时间,若 $J_{i,j}$ 到达 $D_{i,j}$ 仍未执行完,则 $J_{i,j}$ 被认为产生死限丢失,且 $D_{i,j}=D_{i,j-1}+P_i$;

⑤$S_{i,j}$ 表示 $J_{i,j}$ 的最早到达时间,即 $J_{i,j}$ 允许被启动并准备执行的时间。

为了集中讨论所关心的问题,在该模型中,还做了以下假设:

1)调度时机均发生在整数时间,且对于 $T_i=(E_i,P_i,V_i)$,E_i,P_i 和 V_i 也均为整数;

2)任务调度和上下文切换所占用的处理器时间忽略不计;

3)任意 $J_{i,j}$ 在执行过程中不可被抢占;

4)任意 $J_{i,j}$ 只有其前驱Job执行完成后才能开始执行;

5)任意 $J_{i,j}$ 一旦到达截止时间,则 $V_{i,j}$ 立即降低为0。

1.2 约束条件

文中所讨论的任务模型,应满足性质:任意两个任务 T_m,T_n,若 $m<n$(即 T_m 为 T_n 的前驱任务),则有 $P_m\leqslant P_n$。

证明:

设 T_m 的第一个Job,即 $J_{m,1}$ 的最早到达时间 $S_{m,1}=a$,T_n 的第一个Job,即 $J_{n,1}$ 的最早到达时间 $S_{n,1}=b$。

则有 $a<b$,因为 $J_{m,1}$ 一定比 $J_{n,1}$ 先被执行。

假设:有 $P_m>P_n$,则此时存在 k,使得

$a+(k-1)P_m\geqslant b+kP_n$

推导得:$k\geqslant(b-a+P_m)/P_m-P_n$

又 $D_{n,k}=b+kP_n$,$S_{m,k}\geqslant a+(k-1)P_m$

故,当 $k\geqslant(b-a+P_m)/P_m-P_n$ 时,有 $D_{n,k}\geqslant S_{m,k}$

即 $J_{n,k}$ 在 $D_{n,k}$ 前始终无法被调度执行。

因此假设不成立,得证。

2 BVDS算法

2.1 算法描述

假设有任务集 $T=\{T_1,T_2,\cdots,T_n\}$,且符合文中的假设和约束条件,$C=\{C_1,C_2,\cdots,C_m\}$ 全部被用于执行任务,任务调度由另外的处理器执行。算法过程描述如下:

1)构建等待链表 Q 和完成链表 D,起始状态下,Q 与 D 均为空链表。

2)动态构建Job,计算属性值。其中 $S_{1,1}=0$,$S_{i,1}=\sum_{1}^{j-1}E_{i,j}$。

3)将2)中构建的Job插入链表 Q,按 $S_{i,j}$ 升序排序,当 S 相同时,按照 $V_{i,j}$ 降序排序。具体排序原则为:

若 $S_{i,j}<S_{m,n}$,$J_{i,j}$ 在 $J_{m,n}$ 之前;

若 $S_{i,j}=S_{m,n}$ 且 $V_{i,j}>V_{m,n}$,则 $J_{i,j}$ 在 $J_{m,n}$ 之前;

否则,$J_{i,j}$ 在 $J_{m,n}$ 之后。

4)检查 $C=\{C_1,C_2,\cdots,C_m\}$,若 C_k 空闲,则从 Q 的开头开始寻找符合调度条件的Job,Job符合调度条件的原则为:

到达最早开始时间,即 $S_{i,j} \leqslant$ 当前时间;

该 Job 的前驱 Job 已经完成执行,即前驱 Job 出现在完成链表 D 中;

若符合以上 2 个原则,则把该 Job 放入 C_k 执行,并将其从 Q 中删除。

5) 检查 $C = \{C_1, C_2, \cdots, C_m\}$,若 C_k 上的 Job 执行完成,则将 C_k 标志为空闲,并将该 Job 加入链表 D。

6) 重复步骤 3) 至 5)。

假设有 $C = \{C_1, C_2\}$,$T = \{T_1, T_2, T_3\}$,其中 $T_1 = (2, 3, 1)$,$T_2 = (3, 4, 3)$,$T_3 = (4, 5, 2)$。建立相应的等待任务链表 Q,如图 1 所示。

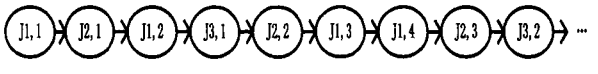

图 1　等待任务链表 Q

t 表示当前处理器执行时间。在执行过程中,当 $t = 6$ 时,C_1 忙碌,C_2 空闲,此时由于 $V_{2,2} > V_{1,3}$,因此优先执行 $J_{2,2}$;当 $t = 10$ 时,没有空闲的处理器可被调度,$J_{2,3}$ 和 $J_{3,2}$ 都无法被执行,则继续等待被调度;当 $t = 11$ 时,C_1 和 C_2 均空闲,分别调度执行 $J_{2,3}$ 和 $J_{3,2}$,此时 $J_{2,3}$ 和 $J_{3,2}$ 虽然延迟了一个时间片被执行,但仍然能够在截止时间之前完成,没有造成丢失。

最终调度结果如图 2 所示($t \in [0,30]$),其中,↑ 表示 Job 的最早到达时间,↓ 表示 Job 的截止时间 。

2.2　存在的问题以及改进方案

从图 2 中可以看出,有些 Job 的实际开始执行时间晚于其截止时间,如 $J_{1,3}$ 和 $J_{1,6}$,这样的任务必然会产生死限丢失,但是它们仍然占用了处理器资源,造成系统浪费。随着这类 Job 的增多,就有可能造成后续 Job 的延迟和丢失,例如图 2 中的 $J_{2,5}$,$J_{1,8}$。不难看出,由于延迟累积,处理器的利用率没有得到有效利用,同时任务的死限丢失率也很高。

另外,在实际应用中,周期任务集 T 中的任务数往往较多,但是任务的影响因子却在一个较小的区间内变动。因此仅仅根据 V_i 来确定一个 Job 的优先级是不足的。

基于以上两点存在的问题,对算法做出改进。改进后算法如下:

1) 构建 Job,计算每个 Job 的属性值。其中 $S_{1,1} = 0$,$S_{i,1} = \sum_{1}^{j-1} E_{i,j}$。

2) 构建等待链表 Q 和完成链表 D,起始状态下,Q 与 D 均为空链表。

3) 建立等待链表 Q,将 Job 插入链表 Q,按 $S_{i,j}$ 升序排序;当 $S_{i,j}$ 相同时,按照 $V_{i,j}$ 降序排序;当 $S_{i,j}$ 和 $V_{i,j}$ 均相同时,则采用 EDF 调度排序。具体排序原则为:

若 $S_{i,j} < S_{m,n}$,则 $J_{i,j}$ 在 $J_{m,n}$ 之前;

若 $S_{i,j} = S_{m,n}$ 且 $V_{i,j} > V_{m,n}$,则 $J_{i,j}$ 在 $J_{m,n}$ 之前;

若 $S_{i,j} = S_{m,n}$,$V_{i,j} = V_{m,n}$,且 $D_{i,j} < D_{m,n}$,则 $J_{i,j}$ 在 $J_{m,n}$ 之前;否则,$J_{i,j}$ 在 $J_{m,n}$ 之后。

4) 检查 $C = \{C_1, C_2, \cdots, C_m\}$,若 C_k 空闲,则从 Q 的开头开始寻找符合调度条件的 Job,Job 符合调度条件的原则为:

剩余时间不小于执行时间,即 $D_{i,j}$ - 当前时间 $\geqslant E_{i,j}$;

该 Job 的前驱 Job 已经完成执行,即前驱 Job 出现在完成链表 D 中;

若符合以上 2 个原则,则把该 Job 放入 C_k 执行,并将其从 Q 中删除;

若在寻找过程中发现存在 Job 不能在其截止时间之前完成,即 $D_{i,j}$ - 当前时间 $\leqslant E_{i,j}$,则该 Job 不再符合被调度条件,将其从 Q 中删除。

5) 检查 $C = \{C_1, C_2, \cdots, C_m\}$,若 C_k 上的 Job 执行完成,则将 C_k 标志为空闲,并将该 Job 加入链表 D。

6) 重复步骤 3) 至 5)。

针对同一例子,采用改进后方案的调度结果如图 3 所示($t \in [0,30]$)。可以看出,被调度给处理器执行的 Job 都不会产生死限丢失。在执行过程中,当 $t = 10$ 时,检查到 $D_{1,3} - t \geqslant E_{1,3}$,将 $J_{1,3}$ 从等待链表中删除,相应地,这也导致了后续的 $J_{2,3}$ 和 $J_{3,3}$ 不被执行。尽管 C_1 和 C_2 的利用率有一定降低,但任务的死限丢失率却大大降低,处理器资源得到更高效的利用。

2.3　算法的实现

本节给出 BVDS 调度算法的一种实现。基于算法的特点,首先需要建立用于存放等待被执行的 Job 的等待任务链表 Q 和用于记录执行完成的 Job 的完成任务链表 D。Q 与 D 都是带空闲头节点的链表,之所以使用链表,是为了便于节点的动态插入与移除。任务结点 JNode 的结构与链表的定义如下:

```
typedef struct JNode
{   int e;   //Job 的执行时间
    int p;   //Job 的周期
    int v;   //Job 的影响因子
    int d;   //Job 的截止时间
    int s;   //Job 的最早开始时间
    struct JNode * next;
}JNode;
```

首先要根据任务集的输入构造对应的 Job 集 J。一个 Job 构造完成后,要插入链表 Q 中合适的位置:

```
temp = Q->head->next;
while (temp ! = NULL) {
//a 为当前准备放入链表 Q 中的 Job
if ((temp->next == NULL) || (a.s < temp->next->job.s)
||( a.s == temp->next->job.s && a.v < temp->next->jo
```

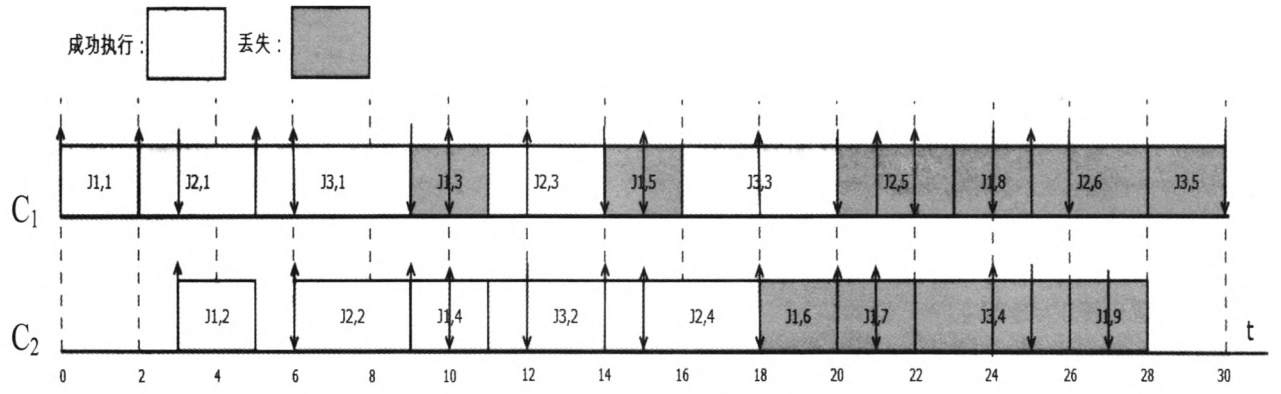

图2 C_1 和 C_2 的调度执行结果

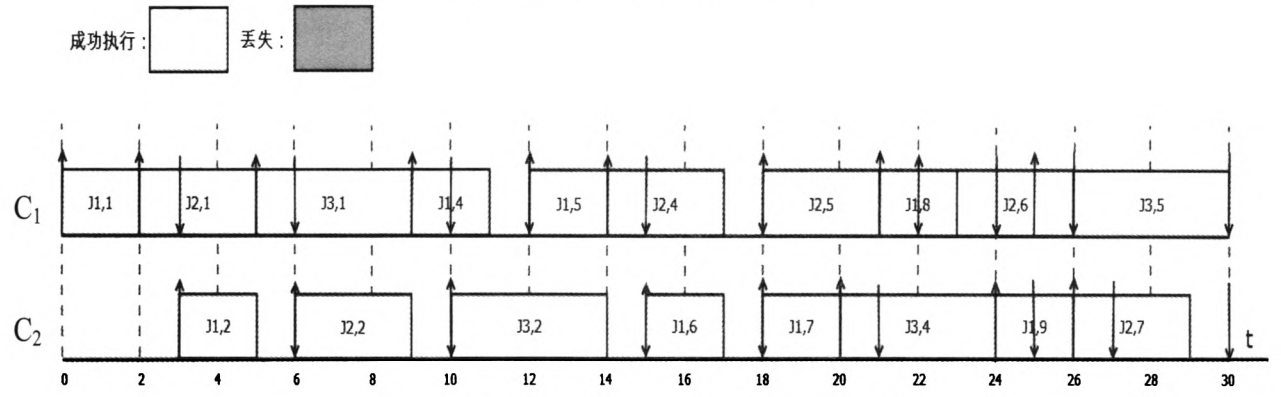

图3 采用改进方案后 C_1 和 C_2 的调度执行结果

```
b. v)
    ||( a. s == temp->next->job. s && a. v == temp->next->
job. v
    && a. d > temp->next->job. d))
    break;
    temp = temp->next; }//找到合适的位置
    insert_into_Q(a); //在 temp 之后插入 Job
```

系统运行过程中,调度模块要选择符合条件的Job调用执行,此时就要从链表 Q 中选取符合条件的Job。在此过程中,若是检查到存在无法在截止时间之前完成的Job,则将其从 Q 中删除。Job 选取/删除过程如下:

```
    temp = Q->head->next;
    while (temp ! = NULL){
    if (temp->job. d - time < temp->job. e) {
    //说明该 Job 已经不能在 deadline 之前完成,将其从 Q 中删
除
    delete_from_Q(temp);
    }else{
    //该 Job 时间上符合执行条件,并且该 Job 的前驱已经执行
完成
    if ((temp->job. s <= time)
    &&(find_pre_from_D(temp->job)) {
    return temp->job; }}//找到符合条件的 Job
    temp = temp->next;}
    return NULL;//找不到符合条件的 Job
```

调度过程中,系统寻找空闲的处理器并分配任务,对于完成执行的Job,将其放入完成链表 D,处理器标记为空闲。过程如下:

```
    time++;//时间推移
    for (i = 1; i <= m; i++) {//m 代表处理器的核数
    if (c[i] == 0) {//Ci空闲
    j = search_job();
    if (j ! = NULL) {//找到符合条件的 Job
    distribute(j,i);//将 j 分配给 Ci执行
    delete_from_Q(temp); }//将 j 从 Q 中删除
    else break;  }  }//当前没有可执行的 Job
    for (i = 1; i <= m; i++) {//m 代表处理器的核数
    if (is_done(i)) {//Ci上的任务执行完成
    insert_into_D(c[i]->job);//加入到完成队列
    c[i] == 0;  }  }//标志 Ci为空闲
```

3 仿真试验

为了研究 BVDS 算法,文中做一些模拟。在软实时多核处理器任务调度中,最关心的两个指标就是多处理器集中各个处理器的利用率以及任务的死限丢失率。

C_i 的利用率用 U_i 表示,在给定的一段时间 Δt 内,$U_{i,1}$ = (Δt - C_i 的空闲时间)/Δt,则处理器集的总利用率 $U_{i,1} = \dfrac{\sum_{i=1}^{n} U_n}{n}$。根据文献[11],$U_i$ 的上界为1。

任务集 T 的死限丢失率用 M 表示,在给定的一段

时间内,$U_{i,1} = \frac{\Delta t \text{ 时间内丢失的 Job 数}}{\Delta t \text{ 时间内到达的 Job 总数}}$。

在文中所做的仿真试验中,设计使用 10 核处理器作为试验对象,选取了 50 个具有约束关系的周期性任务对算法进行分析。为了更好地分析算法的通用性,任务的周期、执行时间和影响因子都由计算机函数随机产生,任务的参数根据以下方法产生:

任务的执行周期 P_i 的取值范围为[2,200],即在区间[2,200]内产生50个随机值,服从均匀分布,且满足 1.2 节中所述性质;

任务的执行时间 E_i 的产生采用文献[12]中的方法。为保证多数任务利用率小于0.4,先在0.4以内产生 50 个随机值作为任务利用率。任务执行时间根据 $E_i = P_i * U_i$计算, 其中 U_i 为任务利用率;

任务的影响因子 V_i 的取值范围为[1,10],即在区间[1,10]内产生 50 个随机值,服从正态分布 $N(5,2.55)$。

为满足各种不同情况的测试需要,对任务集设置了若干机动调整值,以保证任务集在一定的范围内达到边界情况下的要求。

为保证试验结果的一般性,结果数据均采用多次独立实验结果取平均值的方法产生。

结合图 4 与表 1,可以看出,改进后的算法在相同时间内,处理器的总利用率始终低于改进前的算法。在较短时间内,二者的死限丢失率相近。但随着时间延伸,改进前的方案受累积延迟影响,任务的死限丢失率迅速增大,但改进后的算法则能将任务的死限丢失率保持在一个较低的水平。

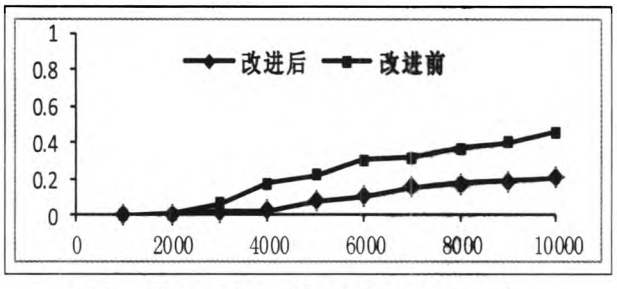

图 4　算法在改进前后的任务死限丢失率

4　结束语

文中通过对多处理器环境下实时调度问题进行分析研究,针对具有约束关系的实时周期任务,提出一种新的调度算法。该算法从有效利用处理器资源和降低任务死限丢失率的角度出发,在执行过程中动态分配处理器资源。实验结果表明,尽管该算法导致处理器的利用率部分降低,却大大减少了任务的死限丢失。其局限性是算法仅考虑了任务的简单约束关系。后续工作将逐渐放宽前提条件进一步进行相关研究。

参考文献:

[1] Liu C, Layland J. Scheduling algorithms for multiprogramming in real-time environment[J]. Journal of ACM, 1973, 20(1): 46-61.

[2] Leontyev H. Compositional Analysis Techniques for Multiprocessor Soft Real-time Scheduling[D]. Chapel Hill: the University of North Carolina, 2010.

[3] Chetto H, Chetto M. Some result of the earliest deadline scheduling algorithm[J]. IEEE Transactions on Software Engineering, 1989, 15(10): 1261-1269.

[4] Mok A K. Fundamental Design Problems of Distributed Systems for the Hard Real Time Environment[D]. Massachusetts, USA: Massachusetts Institute of Technology, 1993.

[5] Sha L, Abdelzaher T. Real Time Scheduling Theory: A Historical Perspective[J]. Real-time Systems, 2004, 28(2-3): 101-115.

[6] 袁　云,邵　时. 基于多核处理器并行系统的任务调度算法[J]. 计算机应用,2008,28(S2):280-282.

[7] George L, Hermant J F. A Norm Approach for the Partitioned EDF Scheduling of Sporadic Task Systems[C]//Proc. of 21st Euromicro Conference on Real-Time Systems. Dublin, Irish: [s. n.], 2009: 161-169.

[8] 宋　杰,檀林欣,曹竹冬,等. 一种新型的实时调度算法[J]. 计算机技术与发展,2010,20(12):73-76.

[9] 黄金贵,李荣珩. 独立多处理机任务静态调度问题的近似算法[J]. 软件学报,2010,21(22):3211-3219.

[10] 张冬松,吴　彤,陈芳园,等. 多核系统中基于 Global EDF 的在线节能实时调度算法[J]. 软件学报,2012,23(4):996-1009.

[11] 王志平,熊光泽. 实时调度算法研究[J]. 电子科技大学学报,2000,29(2):205-208.

[12] 黄姝娟,朱怡安,李兵哲,等. 基于利用率和负载均衡的多核实时调度算法研究[J]. 西北工业大学学报,2012,30(1):117-123.

表 1　算法在改进前后的处理器总利用率

	执行时间									
	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
改进前	0.464	0.635	0.739	0.803	0.842	0.869	0.887	0.902	0.912	0.918
改进后	0.412	0.552	0.552	0.732	0.786	0.821	0.847	0.865	0.871	0.865

一种有约束关系的实时周期任务调度算法研究

作者： 唐毓毅， 朱怡安， 黄姝娟， 李凤彬， TANG Yu-yi， ZHU Yi-an， HUANG Shu-juan， LI Feng-bin

作者单位： 唐毓毅,李凤彬,TANG Yu-yi,LI Feng-bin(西北工业大学计算机学院,陕西西安,710072)， 朱怡安,黄姝娟,ZHU Yi-an,HUANG Shu-juan(西北工业大学计算机学院,陕西西安710072;西北工业大学软件与微电子学院,陕西西安710072)

刊名： 计算机技术与发展 ISTIC

英文刊名： Computer Technology and Development

年，卷(期)： 2013,23(7)