具有依赖关系的周期任务实时调度方法

黄姝娟[1),2)] 朱怡安[1)] 李兵哲[1)] 陆 伟[2)]

[1)](西北工业大学计算机学院 西安 710072)

[2)](西北工业大学软件与微电子学院 西安 710072)

摘 要 随着多核技术在嵌入式领域的快速发展,越来越多的功能被集成在同一个平台上,任务之间的关系越来越复杂.而当前大多数的实时周期任务的调度模型都是不考虑任务之间关系的相互独立的任务模型.文中则针对具有依赖关系的周期任务,提出了一种基于 ST(Simple-Tree)的实时周期任务调度模型,通过该模型来维护任务之间的依赖关系.此外,为了有效地提高系统利用率以及降低死限丢失率,文中还提出了可延迟时间越短越优先的调度方法并和 RM 算法、EDF 算法进行仿真实验比较,结果表明该方法具有较高的核利用率和较低的死限丢失率.

关键词 多核;实时调度;周期任务;调度模型;调度算法

中图法分类号 TP302 **DOI 号** 10.3724/SP.J.1016.2015.00999

Real-Time Scheduling Method for Dependency Period Tasks

HUANG Shu-Juan[1),2)] ZHU Yi-An[1)] LI Bing-Zhe[1)] LU Wei[2)]

[1)](*School of Computer Science, Northwestern Polytechnical University, Xi'an* 710072)

[2)](*School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an* 710072)

Abstract With the rapid development of multi-core technologies in embedded systems, more and more functions has been integrated in the same platform, the relationship between tasks has becoming more complex. Much more work on the real-time scheduling methods for the periodic tasks has been focus on independent tasks model without considering the dependencies between tasks. In this paper, we considered the dependencies of tasks in the context of the periodic task systems and proposed a new scheduling model—ST (Simple-Tree) model which could maintain the dependencies between tasks. In order to effectively improve the system utilization and reduce the miss deadline ratio we also proposed a new scheduling algorithm—PLTSF(Probably Lag Time Shortest First) based on the ST model and compared with the RM and EDF algorithm. The experimental results show that the PLTSF algorithm not only increased the utilization of the cores but also decreased the number of the miss deadline tasks.

Keywords multicore; real-time scheduling; period task; scheduling model; scheduling algorithm

1 引 言

随着多核技术在嵌入式领域的快速发展,围绕片上多处理器实时调度问题产生了大量的研究工作[1-5],所提出的调度模型和调度算法大都基于相互独立的实时任务模型.而未来嵌入式多核平台则要求能够支持更加复杂的应用,任务之间必定存在相

收稿日期:2013-08-27;最终修改稿收到日期:2014-12-11.本课题得到航空基金(20130753006)、航天科技创新基金(2011XR160001)、西北工业大学基础研究基金(JC20110283)资助.**黄姝娟**,女,1975 年生,博士研究生,讲师,中国计算机学会(CCF)会员,主要研究方向为嵌入式系统与普适计算.E-mail:hhsjj@nwpu.edu.cn.**朱怡安**,男,1961 年生,教授,博士生导师,主要研究领域为高性能计算、云计算和普适计算.**李兵哲**,男,1975 年生,博士研究生,讲师,主要研究方向为计算机体系结构和普适计算.**陆 伟**,男,1975 年生,博士研究生,讲师,主要研究方向为高性能计算、自主计算.

应的依赖关系,那么针对这种具有依赖关系的实时任务如何进行调度就成为了研究的热点问题[6].

当前片上多处理器任务调度方法中主要考虑两种模型[7].一种是相互独立的实时周期任务模型.其中典型的是 Liu 和 Layland[8]提出的,其周期任务模型是很多实时任务模型的基础,相应的调度算法有 RM(Rate-Monotonous)算法[9],EDF(Earliest Deadline First)算法[2]等;另一种则是考虑任务之间存在约束关系的任务图模型.该模型中的任务之间有很强的制约关系,但却不考虑任务实时性和周期性问题[10-11].本文则针对具有依赖关系的周期性任务提出一种能体现部分依赖关系的 ST 任务树模型,并针对该任务树模型特征提出基于可延迟时间越短越优先(Probably Lag Time Shortest First,PLTSF)的调度方法.该方法在考虑任务的实时性以及任务之间的依赖关系的基础上进行相应的优先关系定义,并对可并行执行的工作分别采用了 PLTSF、RM 以及 EDF 算法.其实验结果表明 PLTSF 算法在系统整体利用率以及死限丢失率方面都比 RM、EDF 算法先进.

本文第 2 节介绍 ST 任务树模型;第 3 节讲述 ST 任务树模型中各种优先关系定义以及可延迟时间、延迟界限的计算方法以及处理器预分配算法;第 4 节讲述仿真实验的方法和性能评测的指标以及实验结果分析;第 5 节是总结.

2 ST(Simple-Tree)模型

2.1 相关定义

定义 1. ST 模型的树型数据结构 $Tree=(D,R)$ 满足树的基本特征即在任意一棵树中:(1)有且仅有一个特定的称为根的节点;(2)当 $n>1$ 时,其余节点可分为 $n(n>0)$ 个互不相交的有限集 $T_1,T_2,\cdots,T_n$,其中每一个集合本身又是一棵树,并且称为根的子树.

定义 2. 对于任何一棵树而言,每个节点都有层次,根为第 1 层,根的孩子为第 2 层.若某个节点在第 K 层,其孩子就在第 $K+1$ 层,树的深度为树中节点的最大层次.树 T 深度为 H,用 $layer(T)=H$ 表示;节点 P_i 层次为 K,用 $layer(P_i)=K$ 来表示.

定义 3. 对于任意一棵树 T,除了叶子节点外,所有节点都只有一个孩子节点的树称为单行树 ST(Simple Tree),单行树是一棵每个父节点只有一个孩子节点的树.如图 1 所示为一棵深度为 4 的单行树,树中第 K 层的节点数用 $Number(K)$ 来表示.

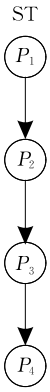

图 1 深度为 4 的单行树

ST 具有如下特征:ST 每层只有一个节点即 $Number(T_i)=layer(T_i)$.节点个数就是层数即 $K=Number(T_i)$.节点之间都是具有一定的顺序关系的.

定义 4. ST 任务模型中的节点都是暗含死限的实时周期任务节点,即该任务节点为一个二元组 $\{e_i,p_i\}$,其中 e_i 表示该任务一个最坏执行时间,p_i 表示该周期任务的周期或死限.以下所提到的 ST 都是此类性质的树.另外,本文假设 ST 任务模型中树与树之间都是相互独立的.

2.2 ST 模型

考虑一个任务系统 Γ 的调度集 $\Gamma=\{ST_1,ST_2,\cdots,ST_n\}$ 为 n 个相互独立的 ST 被调度在 $m\geq 2$ 的处理器核上运行.其中 $ST_i(1\leq i\leq n)$ 为一棵深度为 h 的任务树,当 $h=1$ 时,该树就是一个独立的实时周期任务节点.树中每个节点我们称其为该树根节点的子任务即 $ST_i=\{T_i^1,T_i^2,\cdots,T_i^k\}$ 是 h 个具有相互依赖关系的任务,其中 T_i^1 为该任务树 ST_i 的根任务.$T_i^2,\cdots,T_i^h$ 为 T_i^1 的子任务.为了统一,我们都将其称为该 ST_i 的任务.而其中任何一个任务 $T_i^k(1\leq i\leq n,1\leq k\leq h)$ 都是周期任务节点即 $T_i^k=\{e_i^k,p_i^k\}$.对于其中任何一个任务 T_i^k,因为是周期任务,那么该任务会被周期性地执行和释放,对于其每次执行我们称为一次工作(job).本文假设该工作不可以再划分为更小的可并行执行的粒度.我们把 T_i^k 的第 j 次执行记为 $T_{i,j}^k$,其中 $j\geq 0$.该工作开始执行时刻记为 $r_{i,j}^k$,绝对死限为 $d_{i,j}^k$.由于 T_i^k 的周期为 p_i^k,故 $d_{i,j}^k=r_{i,j}^k+p_i^k$.任务 T_i^k 的利用率为 $u_i^k=\frac{e_i^k}{p_i^k}$;对于任意一棵 $ST_i(1\leq i\leq n)$ 树,其利用率记为 $U_{ST_i}=\sum_{k=1}^{h}u_i^k$;任务系统 Γ 的总利用率为 $U_{sum}=\sum_{i=1}^{n}U_{ST_i}$.

对于同一棵 ST 而言，由于任务之间有相互依赖关系，那么在执行过程中必然有执行顺序，例如 $T_{i,j}^{k+1}$ 一定在 $T_{i,j}^{k}(k\geq 1)$ 执行完毕之后才能执行，而 $T_{i,j+1}^{k}$ 也必须在 $T_{i,j}^{k}(j\geq 1)$ 执行完毕之后执行，因此，为了区分它们，将同一周期子任务的上次工作 $T_{i,j-1}^{k}$ 称为 $T_{i,j}^{k}$ 工作前驱而下次工作 $T_{i,j+1}^{k}$ 称为 $T_{i,j}^{k}$ 工作后继(job)，并将与其有依赖关系的上层工作 $T_{i,j}^{k-1}$ 称为 $T_{i,j}^{k}$ 依赖前驱而下层工作后继 $T_{i,j}^{k+1}$ 称为 $T_{i,j}^{k}$ 依赖(depend)后继.

3　可延迟时间以及优先关系确定

3.1　可延迟时间确定

定义 5.　如果一个工作 $T_{i,j}^{k}$ 在时刻 t 之前已经开始却没有在时刻 t 完成，那么该时刻 t 称为子任务 $T_{i,j}^{k}$ 的延用时刻，该工作称为时刻 t 未完成工作. 如果 t 为该工作的绝对死限，则称该工作死限未满足或死限丢失.

定义 6.　在调度算法 S 下，定义一个时间分配函数为 $r_s(x)$，该函数表示在一个很小的单位时间间隔 Δt 内，一个工作 x 如果被执行，该函数值为 1，否则函数值为 0.

定义 7.　一个工作 $T_{i,j}^{k}$ 在一个时间间隔 $[t_1,t_2)$ 内通过调度算法 S 所能给分配的时间值定义为 $Runtime(T_{i,j}^{k},t_1,t_2,S)=\int_{t_1}^{t_2} r_s(T_{i,j}^{k})\mathrm{d}t$.

推论 1.　在调度算法 S 下，对于一棵 ST_i 树中任何一个暗含死限的周期任务 $T_i^k=\{e_i^k,p_i^k\}$，在没有任何外界干扰且 ST_i 的根节点 T_i^1 发布时间为 0 的情况下，该周期任务的任何一个工作 $T_{i,j}^{k}$ 理论上可被调度的最早发布时刻记为 $O_{i,j}^{k}$，则

$$O_{i,j}^{k}=\max\left\{\left[(j-1)\times p_i^n+\sum_{v=n}^{k-1}e_i^v\,\middle|\,n=1,2,3,\cdots,k-1\right],\ (j-1)\times p_i^k\right\}.$$

证明.　第 1 个任务 T_i^1 的第 j 个 job 的最早可以发布时刻为 $O_{i,j}^{1}=(j-1)\times p_i^1$.

由于第 2 个任务在第 1 个任务完成之后才能开始执行. 因此，第 2 个任务 T_i^2 的第 j 个 job 的最早可以发布的时刻为

$$O_{i,j}^{2}=\max\{O_{i,j}^{1}+e_i^1,(j-1)\times p_i^2\}=\max\{(j-1)\times p_i^1+e_i^1,(j-1)\times p_i^2\};$$

同样第 3 个任务在第 2 个任务的完成之后才能执行，第 3 个任务 T_i^3 的第 j 个 job 的最早可以发布的时刻为

$$O_{i,j}^{3}=\max\{O_{i,j}^{2}+e_i^2,(j-1)\times p_i^3\}=\max\left\{\left((j-1)\times p_i^1+\sum_{n=1}^{2}e_i^n\right),(j-1)\times p_i^2+e_i^2,(j-1)\times p_i^3\right\};$$

以此类推，第 k 个任务 T_i^k 的第 j 个 job 的最早可以发布的时刻为

$$\begin{aligned}O_{i,j}^{k}&=\max\{O_{i,j}^{k-1}+e_i^{k-1},(j-1)\times p_i^k\}\\&=\max\left\{\left((j-1)\times p_i^1+\sum_{n=1}^{k-1}e_i^n\right),(j-1)\times p_i^2+\sum_{n=2}^{k-1}e_i^n,\cdots,(j-1)\times p_i^{k-1}+e_i^{k-1},(j-1)\times p_i^k\right\}\\&=\max\left\{\left[(j-1)\times p_i^n+\sum_{v=n}^{k-1}e_i^v\,\middle|\,n=1,2,3,\cdots,k-1\right],(j-1)\times p_i^k\right\}\end{aligned}$$

证毕.

定理 1.　在调度算法 S 下，由 ST 模型可知第 k 层节点 $T_i^k=\{e_i^k,p_i^k\}$ 的前驱节点集合为 $V=\{T_i^1,T_i^2,\cdots,T_i^{k-1}\}$. 在忽略其他额外开销的情况下，从 0 时刻开始调度根任务，则判断其能否与其前驱结点在同一个处理器上可被调度的必要条件是 $p_i^k\geq\max\{p_i^n\,|\,n=1,2,\cdots,k-1\}$.

证明.　如果算法 S 能成功调度，那么该节点的任何一个工作 $T_{i,j}^{k}$ 的执行不能超过其死限，那么它的最早开始发布时刻一定小于最迟开始执行时刻 $d_{i,j}^{k}-e_i^k$，也就是说如果它的最早开始发布时刻大于最迟开始执行时刻，则它就不可能在其绝对死限 $d_{i,j}^{k}$ 之前完成，那么必定存在丢失死限情况. 因此必须有 $O_{i,j}^{k}\leq(d_{i,j}^{k}-e_i^k)$ 成立. 所以

$$\begin{aligned}&O_{i,j}^{k}\leq(d_{i,j}^{k}-e_i^k)\\&\Rightarrow\max\left\{\left[(j-1)\times p_i^n+\sum_{v=n}^{k-1}e_i^v\,\middle|\,n=1,2,3,\cdots,k-1\right],(j-1)\times p_i^k\right\}\leq(d_{i,j}^{k}-e_i^k)\\&\Rightarrow(d_{i,j}^{k}-e_i^k)\geq\max\left\{(j-1)\times p_i^k,(j-1)\times p_i^{k-1}+e_i^{k-1},\cdots,(j-1)\times p_i^1+\sum_{n=1}^{k-1}e_i^n\right\}\\&\Rightarrow(j\times p_i^k-e_i^k)\geq\max\left\{(j-1)\times p_i^k,(j-1)\times p_i^{k-1}+e_i^{k-1},\cdots,(j-1)\times p_i^1+\sum_{n=1}^{k-1}e_i^n\right\},\end{aligned}$$

也就是说，$j\times p_i^k-e_i^k$ 要大于后者其中任意一项. 则有

$$j\times p_i^k-e_i^k\geq(j-1)\times p_i^k\Rightarrow p_i^k\geq e_i^k$$

$$j\times p_i^k-e_i^k\geq(j-1)\times p_i^{k-1}+e_i^{k-1}$$

$$\Rightarrow p_i^k\geq\frac{j-1}{j}p_i^{k-1}+\frac{e_i^k+e_i^{k-1}}{j}$$

$$j\times p_i^k-e_i^k\geq(j-1)\times p_i^{k-2}+e_i^{k-1}+e_i^{k-2}$$

$$\Rightarrow p_i^k\geq\frac{j-1}{j}p_i^{k-1}+\frac{(e_i^k+e_i^{k-1}+e_i^{k-2})}{j}$$

$$\cdots,$$

$$j\times p_i^k-e_i^k\geq(j-1)\times p_i^1+\sum_{n=1}^{k-1}e_i^n$$

$$\Rightarrow p_i^k\geq\frac{j-1}{j}p_i^1+\frac{\sum_{n=1}^{k}e_i^n}{j},$$

因此,可以得到

$$p_i^k\geq\max\left\{\frac{j-1}{j}p_i^n+\frac{\sum_{v=n}^{k}e_i^v}{j}\,\middle|\,n=1,2,\cdots,k\right\}.$$

当 j 趋于无穷大时,即

$$p_i^k\geq\lim_{j\to\infty}\max\left\{\frac{j-1}{j}p_i^n+\frac{\sum_{v=n}^{k}e_i^v}{j}\,\middle|\,n=1,2,\cdots,k\right\}$$

$$\Rightarrow p_i^k\geq\max\{p_i^n\mid n=1,2,\cdots,k-1\}.$$ 证毕.

定理 2. 在调度算法 S 下,对于一个单行树模型 ST_i 从 0 时刻开始调度根任务,如果其中一个暗含死限的周期任务 $T_i^k=\{e_i^k,p_i^k\}$ 被成功调度,那么在任意一个时间间隔$[t_1,t_2)$内,该任务应该被调度的时间值为

$$R(T_i^k,t_1,t_2)=(t_2-t_1)u_i^k \tag{1}$$

该周期任务的任何一个工作 $T_{i,j}^k$ 在该时间间隔内应该被执行的时间值为

$$r(T_{i,j}^k,t_1,t_2)=$$

$$\begin{cases}0, & t_1\leq d_{i,j}^k\wedge t_2<O_{i,j}^k\\ e_i^k, & t_2-t_1\geq e_i^k\wedge[t_2,t_1)\in[O_{i,j}^k,d_{i,j}^k]\\ (t_2-t_1)\times u_i^k, & t_2-t_1<e_i^k\wedge[t_2,t_1]\in[O_{i,j}^k,d_{i,j}^k]\\ \dfrac{d_{i,j}^k-t_1}{p_i^k}\times e_i^k, & O_{i,j}^k\leq t_1<d_{i,j}^k\wedge t_2>d_{i,j}^k\\ \left(1-\dfrac{d_{i,j}^k-t_2}{p_i^k}\right)\times e_i^k, & t_1<O_{i,j}^k\wedge O_{i,j}^k\leq t_2\leq d_{i,j}^k\end{cases} \tag{2}$$

其中 $O_{i,j}^k$ 为 T_i^k 的最早发布时刻且

$$R(T_i^k,t_1,t_2)=\sum_{j\geq1}r(T_{i,j}^k,t_1,t_2).$$

证明. 根据推论 1 和定理 1 知,一个周期任务的一次作业 $T_{i,j}^k$ 的最早发布的时刻为 $O_{i,j}^k$,最迟发布时刻为 $d_{i,j}^k-e_i^k$,如果可以被成功调度,则根据$[t_1,t_2)$的时间段 $r(T_{i,j}^k,t_1,t_2)$ 相应的值分为 4 种情况:

第 1 种情况. 如果 $t_1\geq d_{i,j}^k$ 或者 $t_2<O_{i,j}^k$,那么 $T_{i,j}^k$ 的执行不在该范围内则 $r(T_{i,j}^k,t_1,t_2)$ 的值为 0;

第 2 种情况. 如果 $t_2-t_1\geq e_i^k$ 且$[t_1,t_2)\in[O_{i,j}^k,d_{i,j}^k]$,那么调度执行时间为 e_i^k 个时间单元. 否则在一个时间间隔$[t_1,t_2)$内包含有该任务的周期个数为 $\dfrac{(t_2-t_1)}{p_i^k}$,在每个周期内需要执行的时间为 e_i^k,因此 $T_{i,j}^k$ 在该时间间隔$[t_1,t_2)$内被调度的时间值为

$$r(T_{i,j}^k,t_1,t_2)=(t_2-t_1)\frac{e_i^k}{p_i^k}=(t_2-t_1)u_i^k;$$

第 3 种情况. 如果 $O_{i,j}^k\leq t_1<d_{i,j}^k$ 且 $t_2>d_{i,j}^k$,那么 $T_{i,j}^k$ 在该时间间隔$[t_1,t_2)$内被调度的时间值为

$$r(T_{i,j}^k,t_1,t_2)=\frac{d_{i,j}^k-t_1}{p_i^k}\times e_i^k;$$

第 4 种情况. 如果 $t_1<O_{i,j}^k$ 且 $O_{i,j}^k\leq t_2<d_{i,j}^k$,那么 $T_{i,j}^k$ 在该时间间隔$[t_1,t_2)$内被成功调度的时间值为

$$r(T_{i,j}^k,t_1,t_2)=e_i^k-\frac{d_{i,j}^k-t_2}{p_i^k}\times e_i^k=\left(1-\frac{d_{i,j}^k-t_2}{p_i^k}\right)\times e_i^k.$$

由此得证式(2)成立.

假设周期任务 T_i^k 在时间间隔$[t_1,t_2)$内被调度的第一个工作为 $T_{i,j}^k$,最后一个工作为 $T_{i,h}^k(h\geq j)$,则 $j-1$ 之前的工作(包括第 $j-1$ 个工作)和 $h+1$ 之后的工作(包括第 $h+1$ 个工作)都分别在时间间隔$[0,t_1)$和$[t_2,\infty)$之间执行,而没有在该时间间隔$[t_1,t_2)$内执行,所以它们在该时间间隔内被分配的时间值为 0. 由于第 j 个任务和第 h 个任务都是在该时间间隔内最早和最后执行,所以 $h-j-1$ 个任务在该时间间隔内都执行完毕则执行的时间值为 e_i^k,第 j 个任务和第 h 个任务执行的时间值分别为 $\left(\dfrac{d_{i,j}^k-t_1}{p_i^k}\right)\times e_i^k$ 和 $\left(1-\dfrac{d_{i,h}^k-t_2}{p_i^k}\right)\times e_i^k$. 那么

$$R(T_i^k,t_1,t_2)=\sum_{j\leq l\leq h}r(T_{i,l}^k,t_1,t_2)$$

$$=(h-j-1)e_i^k+\left(\frac{d_{i,j}^k-t_1}{p_i^k}\right)\times e_i^k+\left(1-\frac{d_{i,h}^k-t_2}{p_i^k}\right)\times e_i^k$$

$$=(h-j-1)e_i^k+\left(\frac{j\times p_i^k-t_1}{p_i^k}\right)\times e_i^k+\left(1-h+\frac{t_2}{p_i^k}\right)\times e_i^k$$

$$=(h-j-1)e_i^k+je_i^k-t_1\times\frac{e_i^k}{p_i^k}+t_2\times\frac{e_i^k}{p_i^k}-he_i^k+e_i^k$$

$$=\frac{t_2}{p_i^k}e_i^k-\frac{t_1}{p_i^k}e_i^k$$

$$=(t_2-t_1)u_i^k.$$

由此得证式(1)成立. 证毕.

定义 8. 在任意一个时间间隔$[t_1,t_2)$内，一个工作$T_{i,j}^k$在调度算法S的调度下被推迟执行的时间记为

$$lag(T_{i,j}^k,t_1,t_2,S)=r(T_{i,j}^k,t_1,t_2)-Runtime(T_{i,j}^k,t_1,t_2,S) \quad (3)$$

其中$Runtime(T_{i,j}^k,t_1,t_2,S)$用来表示在$[t_1,t_2)$时间间隔内，$S$调度算法下实际分配给该工作的时间片. 若该工作在$[t_1,t_2)$时间间隔内没有被调度执行则$Runtime(T_{i,j}^k,t_1,t_2,S)=0$.

定义 9. 一个子任务被推迟执行的时间记为

$$\begin{aligned}lag(T_i^k,t_1,t_2,S)&=\sum_{j\geq 1}lag(T_{i,j}^k,t_1,t_2,S)\\&=\sum_{j\geq 1}r(T_{i,j}^k,t_1,t_2)-\sum_{j\geq 1}Runtime(T_{i,j}^k,t_1,t_2,S)\\&=R(T_i^k,t_1,t_2)-\sum_{j\geq 1}Runtime(T_{i,j}^k,t_1,t_2,S)\\&=(t_2-t_1)\times u_i^k-\sum_{j\geq 1}\int_{t_1}^{t_2}r_s(T_{i,j}^k)\mathrm{d}t \quad (4)\end{aligned}$$

定理 3. 如果某时刻t在调度算法S下是子任务T_i^k的延用时刻，那么T_i^k的延迟界限是其利用率u_i^k和最坏执行时间e_i^k的线性函数.

证明. 由于t时刻为延用时刻，则一定存在T_i^k的某个工作$T_{i,j}^k$在t时刻已经开始执行，但却没有结束. 考察时间间隔$[0,t)$，假定T_i^k的第一个在t时刻之前开始执行却没有在t时刻完成的工作为第j个工作，那么$lag(T_i^k,0,t,S)=\sum_{h\geq j}lag(T_{i,h}^k,0,t,S)$. 假设第$j$个工作释放时刻为$r_{i,j}^k$，那么在该时刻之前即在时间段$[0,r_{i,j}^k)$内，该子任务所有第$i(i<j)$个工作在$t$时刻之前都已经完成. 由于从第$j$个工作开始才出现未完成的工作，因此$r_{i,j}^k<t$. 令$\lambda_{i,j}^k$为截至$t$时刻分配给工作$T_{i,j}^k$的时间值即$Runtime(T_{i,j}^k,r_{i,j}^k,t,S)=\lambda_{i,j}^k$，那么

$$\begin{aligned}lag(T_i^k,0,t,S)&=\sum_{h\geq j}lag(T_{i,h}^k,r_{i,j}^k,t,S)\\&=\sum_{h>j}lag(T_{i,h}^k,r_{i,j}^k,t,S)+lag(T_{i,j}^k,r_{i,j}^k,t,S)\\&=\sum_{h>j}r(T_{i,h}^k,r_{i,j}^k,t)-\sum_{h>j}Runtime(T_{i,h}^k,r_{i,j}^k,t,S)+\\&\quad r(T_{i,j}^k,r_{i,j}^k,t)-\lambda_{i,j}^k,\end{aligned}$$

因为第j个工作在t时刻还没有执行，那么在时间间隔$[0,t)$内不可能执行j以后的工作，所以当$h>j$时又有$Runtime(T_{i,h}^k,r_{i,j}^k,t,S)=0$，

$\sum_{h>j}r(T_{i,h}^k,r_{i,j}^k,t)=r(T_i^k,r_{i,j}^k,t)\leq r(T_i^k,0,t)=tu_i^k$，

而$r(T_{i,j}^k,r_{i,j}^k,t)\leq e_i^k$，

故$lag(T_i^k,0,t,S)\leq tu_i^k+e_i^k-\lambda_{i,j}^k$.

由此得证并记录$lag^{up}(T_i^k,0,t,S)=tu_i^k+e_i^k-\lambda_{i,j}^k$. 证毕.

定义 10. 一个工作在时刻t的可延迟时间(Probably Lag Time，PLT)定义为该工作的绝对死限与该工作在$[0,t)$延迟时间的差值，结合定义 9 得到

$$\begin{aligned}PLT(T_{i,j}^k,0,t)&=d_{i,j}^k-lag(T_{i,j}^k,0,t,S)\\&=d_{i,j}^k-(r(T_{i,j}^k,0,t)-Runtime(T_{i,j}^k,0,t,S)) \quad (5)\end{aligned}$$

其中$d_{i,j}^k$为该工作的绝对死限.

3.2 优先级定义

定义 11. 给定属于同一棵任务树的任何两个工作$T_{i,v}^w$和$T_{i,j}^k$，定义一个执行顺序优先关系$\prec$，如果$(1\leq v<j)\wedge(w=k)$或$(v=j\wedge 1\leq w<k)$，则认为$T_{i,v}^w$顺序执行优先于$T_{i,j}^k$，记为$T_{i,v}^w\prec T_{i,j}^k$. 在$(v\neq j)\wedge(w\neq k)$情况下，如果两个工作的所有前驱都执行完毕则可以并行执行，具体执行顺序可根据不同的调度算法来执行. 如果采用 PLTSF 算法调度并行执行工作则计算$PLT(T_{i,v}^w,0,t)$和$PLT(T_{i,j}^k,0,t)$，若$PLT(T_{i,v}^w,0,t)<PLT(T_{i,j}^k,0,t)$，则$T_{i,v}^w\prec T_{i,j}^k$.

定义 12. 对于不同的两棵单行树中的两个工作$T_{u,v}^w$和$T_{i,j}^k$，若$PLT(T_{u,v}^w,0,t)<PLT(T_{i,j}^k,0,t)$，则$T_{u,v}^w\prec T_{i,j}^k$. 如果$PLT(T_{u,v}^w,0,t)=PLT(T_{i,j}^k,0,t)$，则比较两个子任务的延迟界限；若$lag^{up}(T_u^w,0,t,S)<lag^{up}(T_i^k,0,t,S)$，则$T_{u,v}^w\prec T_{i,j}^k$.

3.3 处理器预分配算法

首先将 ST 模型中的每个 ST 的任务节点按照定理 1 进行判断是否可以满足可被调度的必要条件. 如果满足，则根据任务之间的依赖关系，在m个处理器核上进行分配，预分配算法如下：

(1) 将任务系统中n个 ST 树中的第一个工作放入到就绪队列中，根据定义 11 和定义 12 进行优先级判定，选择m个高优先级的工作分配到m个核上去执行. 如果工作个数n小于m个，则选择n个核来执行.

(2) 当执行一个单位时间之后，将可并行的任务放到就绪队列中，计算当前核上每个任务的工作所执行的时间值$\lambda_{i,j}^k$，如果$\lambda_{i,j}^k=e_i^k$则表明该工作已执行完毕，则将其在所对应的任务扩展图和就绪队列中删除. 如果$\lambda_{i,j}^k<e_i^k$，说明该工作没有执行完毕，则更改该工作需要执行的时间片为$e_i^k-\lambda_{i,j}^k$，并把该工作重新放到就绪队列中等待和其他工作一起调度.

(3) 重新计算就绪队列中的每个任务的优先

级，选取 m 个任务分别分配到 m 个核上运行. 如果工作数 n 小于 m 个核则选择 n 个核去执行.

(4) 重复步骤(2)～(3)，直到所有的工作都能分配到核上去执行.

4 实验及结果分析

本文的仿真实验平台是在 4 核 Intel core(TM)2 Quad CPU 2.66 GHz 内存为 3.4GB 的硬件环境下运行 Ubuntu Linux10.04 2.6.33-29 实时内核，采用 Codeblocks-v10.05 编写仿真测试程序. 根据文献[8]中的定理，在实验设计时，选择了处理器核数 m 与 ST 树总利用率 $\lceil U_{sum} \rceil$ 之间的 3 种不同的关系：$\lceil U_{sum} \rceil > m$，$\lceil U_{sum} \rceil = m$，$\lceil U_{sum} \rceil < m$，针对随机输入的具有依赖关系的 ST 树进行了多组实验求得平均值，为了进行实验结果对比分析，定义如下的性能评价指标.

定义 13. 为了在给定时间段内分析上述 3 种情况下不同调度算法的吞吐量，定义在相同的时间段内某调度算法所能够完成的工作总数为该调度算法的吞吐量.

定义 14. 在算法分配过程中存在某时间段内核处于空闲状态，为了描述核利用情况，定义核总利用率为所有核执行工作的时间片数量与全部时间片数量的一个比值. 该比值是 ST 模型对核利用情况的一个反映. 该值越大说明该模型和调度方法对核的利用越充分.

定义 15. 由于 ST 模型中的工作在调度过程中存在死限不满足的情况，为了考察死限不满足的严重程度，定义死限丢失率为所有丢失死限的工作数量与总工作数量的比值.

根据以上的评价指标，对可并行执行的工作分别采用 EDF 算法、RM 算法以及 PLTSF 算法进行调度，在 500 个时间片内对系统的吞吐量、核利用率以及丢失死限数进行了统计. 图 2 展示了在 3 种情况下，每种情况随机输入 10 组数据得出的系统吞吐量的平均值. 从图 2 中可以看出在负载量较大情况下即 $\lceil U_{sum} \rceil > m$ 时，PLTSF 算法系统吞吐量明显大于其他两个算法，而在负载量较小的情况下对比不是很明显，因此本文仅针对在负载量较大的情况下，统计 3 种算法的核总利用率和死限丢失率的情况，分别如图 3 和表 1 所示.

图 3 的结果显示出采用 PLTSF 算法的核利用率效果很好. 从表 1 中可以看出任务的死限不满足率随着时间的持续而增长，但 PLTSF 算法下增长

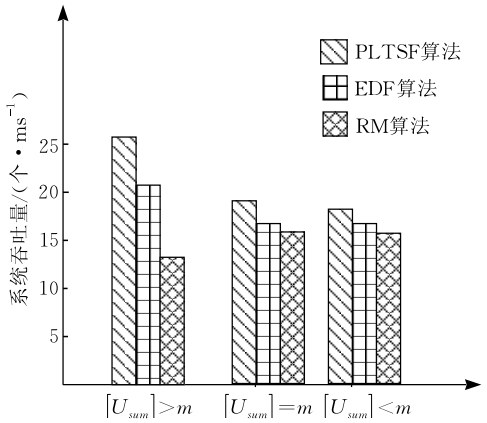

图 2 3 种情况下系统吞吐量测试

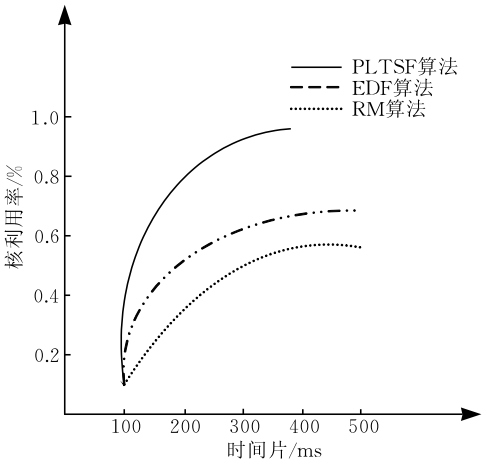

图 3 3 种算法核总利用率增长情况

的趋势较低. 分析原因发现由于 EDF 算法和 RM 算法都是针对独立任务模型而设置的，优先级是根据自身的参数(死限和利用率)来决定的，并没有考虑任务的实际执行情况，有时会导致一些后面的任务优先级高而没有办法执行的情况(因为前驱没有执行完毕). 而 PLTSF 算法从一开始就考虑任务在实际执行过程中出现的延迟时间以及任务之间前后执行顺序问题，所以效果较好. 这说明在 ST 模型下，PLTSF 调度算法比 EDF 算法和 RM 算法更加合适.

表 1 固定时间片内 3 个算法的死限丢失率统计

(单位：%)

固定时间片/ms	PLTSF 算法	EDF 算法	RM 算法
100	0.00120	0.00180	0.00220
200	0.01012	0.05032	0.04267
300	0.02310	0.06160	0.07120
400	0.03250	0.12500	0.13200
500	0.03260	0.15000	0.20000

5 结束语

本文考虑具有依赖关系的实时周期任务在多核

处理器上的调度模型和调度算法，该模型不仅维持了任务之间的部分依赖关系，而且采用了基于可延迟时间越短越优先的调度策略，提高了系统利用率又降低了死限丢失的任务数量. 文章首先描述了 ST 任务树的模型，接着定义了延迟时间、延迟界限以及优先关系，通过将模型图中的 ST 树进行扩展，根据相应的优先执行关系利用 PLTSF 调度算法将其映射到相应的核上去执行. 仿真实验表明在 ST 模型下的 PLTSF 调度算法比 RM、EDF 算法在吞吐量、核利用率以及死限丢失率方面都较优. 不足之处是本文假设 ST 树与树之间无任何关联关系且所选算法还是存在任务死限丢失情况. 未来方向就是研究如何将 ST 树之间的关联性加入到调度考虑的范围当中以及如何将死限丢失率降到最低限度.

参 考 文 献

[1] Zhou Benhai, Qing Jianzhong, Lin Shukuan. Research on synthesis parameter real-time scheduling algorithm on multi-core architecture//Proceedings of the IEEE Control and Decision Conference. Guilin, China, 2009: 5116-5120

[2] Bastoni A, Brandenburg B B, Anderson J H. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers//Proceedings of the 31st IEEE Real-Time Systems Symposium. San Diego, USA, 2010: 14-24

[3] Andersson B, Pinho L M. Implementing multicore real-time scheduling algorithms based on task splitting using Ada 2012//Proceedings of the 15th Reliable Software Technologies — Ada-Europe 2010. Valencia, Spain, 2010: 54-67

[4] Liu Jia-Hai, Yang Mao-Lin. Task scheduling of real-time systems on multi-core embedded processor//Proceedings of the International Conference on Intelligent Systems and Knowledge Engineering. Hangzhou, China, 2010: 580-583

[5] Davis R, Burns A. Improved priority assignment for global fixed priority preemptive scheduling in multiprocessor real-time systems. Journal of Real-Time Systems, 2011, 47(1): 1-40

[6] Cong L, Anderson J H. Scheduling suspendable, pipelined tasks with non-preemptive sections in soft real-time multiprocessor systems//Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium. Stockholm, Sweden, 2010: 23-32

[7] Li Ren-Fa, Liu Yan, Xu Cheng. A survey of task scheduling research progress on multiprocessor system-on-chip. Journal of Computer Research and Development, 2008, 45(9): 1620-1629(in Chinese)
(李仁发，刘彦，徐成. 多处理器片上系统任务调度研究进展评述. 计算机研究与发展，2008，45(9)：1620-1629)

[8] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of Association for Computing Machinery, 1973, 20(1): 46-61

[9] Bertossi A A, Fusiello A. Rate-monotonic scheduling for hard-real-time systems. Journal of European Operational Research, 1997, 96(3): 429-443

[10] Bozdag D, Ozguner F, Catalyurek U V. Compaction of schedules and a two-stage approach for duplication-based DAG scheduling. Journal of IEEE Transactions on Parallel and Distributed Systems, 2009, 20(6): 857-871

[11] Zhou Lan, Sun Shixin. A genetic scheduling algorithm based on knowledge for multiprocessor system//Proceedings of the International Conference on Communications, Circuits and Systems(ICCCAS 2007). Fukuoka, Japan, 2007: 900-904

HUANG Shu-Juan, born in 1975, Ph. D. candidate, lecturer. Her research interests focus on embedded system and ubiquitous computing.

ZHU Yi-An, born in 1961, professor, Ph. D. supervisor. His main research interests include HPC, cloud computing and ubiquitous computing.

LI Bing-Zhe, born in 1975, Ph. D. candidate. His main research interests focus on computer architecture and ubiquitous computing.

LU Wei, born in 1975, Ph. D. candidate, lecturer. His main research interests focus on HPC and autonomic computing.

Background

This paper discussed the content belongs to the multi-core real-time scheduling problems in embedded systems. In recent years, with the rapid development of the multi-core technologies in all fields, it is an active research area in the real-time embedded systems and such a research has both significant theoretic values and wide potential applications. But most of previous work on multiprocessor real-time scheduling focused on independent tasks, they did not analyze the

dependencies between the tasks and the performance index in this complex situation.

This paper investigated the impact on the interdependent tasks and proposed a new model and an algorithm to schedule these periodic tasks. Compared with the RM and EDF algorithms, the new algorithm could improve the performance of this system.

The research in this paper is supported by the Aeronautical Science Foundation of China (20130753006) and the Aerospace Science and Technology Innovate Foundation of China (2011XR160001) and the Northwestern Polytechnical University Basic Research Foundation of China (JC20110283). The leader of the projects is Zhu Yi-An. The first author Huang Shu-Juan of this paper is a Ph. D. candidate, and her supervisor is Zhu Yi-An.

There also has to face many challenges such as how to enhance efficiency, how to schedule the mixed criticality tasks, how to solve the synchronization and task immigration between the cores and etc. Some strategies are given to solve these problems in these projects and some key technologies are proposed to achieve the software reliability and effectiveness such as the ST model and the PLTSF algorithm and the evaluation methods in this paper.

Now, the research group has made some achievements. Several papers have been published in Journal of Northwestern Polytechnical University, Journal of Computer Applications, Journal of Computer Research and Development, and other Journals, and CSAE, PASS, and other conferences. Multi-core scheduling is an important issue for improving the efficiency in the embedded systems especially in the aviation and aeronautics field, and the content of this paper is an important part for the multi-core platform in embedded systems.