

基于简单反馈的混合静态/动态节能弱硬实时调度算法

吴彤^{1),2)} 张冬松¹⁾ 金士尧¹⁾

¹⁾(国防科学技术大学计算机学院并行与分布处理重点实验室 长沙 410073)

²⁾(国防科学技术大学继续教育学院 长沙 410073)

摘 要 随着能耗问题日益显著,节能实时调度成为实时调度领域研究的热点.由于混合静态/动态节能弱硬实时调度算法基于最坏情况执行时间计算任务的执行速度,因此限制了节能效果,文中针对这一问题,提出一种新算法,通过引入简单反馈机制,估计任务的实际执行时间,通过任务划分,降低任务的整体执行速度,延长执行时间,进而达到高效节能的目的.实验表明,当平均情况执行时间低于最坏情况执行时间较多时,新算法优于原始算法,最多可节能60%~70%,最少可节能约10%.算法的不足之处在于当平均情况执行时间接近最坏情况执行时间时,新算法比原算法更耗能.

关键词 实时调度; 能量有效; 反馈; 弱硬实时

中图法分类号 TP316

DOI号: 10.3724/SP.J.1016.2009.01140

A Hybrid Static/Dynamic Energy-Aware Weakly-Hard Real-Time Scheduling Algorithm Based on Simple Feedback

WU Tong^{1),2)} ZHANG Dong-Song¹⁾ JIN Shi-Yao¹⁾

¹⁾(National Laboratory for Parallel and Distributed Processing, Institute of Computer, National University of Defense Technology, Changsha 410073)

²⁾(School of Continuing Education, National University of Defense Technology, Changsha 410073)

Abstract With the emergence of the prominent problem of energy consumption, energy efficient real time scheduling is hot. Based on the Worst Condition Execution Time (WCET), the hybrid static/dynamic algorithm calculates the processor speeds offline, which results in the limited energy efficiency. To solve this problem, a new algorithm is proposed, in which, the simple feedback mechanism is introduced to estimate the actual execution time. The whole speed is decreased by splitting the job into two parts. The execution time is prolonged, and so the new algorithm is more energy efficient. The experimental results show that the proposed algorithm outperforms the original one when the Average Case Execution Time (ACET) is much less than the WCET, which can improve energy savings about 60% to 70% at most and about 10% at least. Unfortunately, when the ACET is close to the WCET, the proposed algorithm consumes more energy than the original one.

Keywords real-time scheduling; energy efficient; feedback; weakly hard real-time

1 引言

众所周知,能耗问题已成为系统设计中至关重

要的非功能性设计约束,因其不仅引发难以解决的散热问题,而且限制了处理器主频的进一步提高,制约系统性能的提高,而且能耗的急剧增加还会提高硬件的封装和制冷成本,导致系统可靠性下降.随着

动态电源管理 (Dynamic Power Manage, DPM)^[1]、动态电压调节 (Dynamic Voltage Scaling, DVS) 以及动态调制调节 (Dynamic Modulation Scaling, DMS)^[2] 等技术的出现,使得从软件角度上提出节能实时调度算法成为可能。

DVS 用于降低动态功耗,在节能实时调度中使用最多,影响最大. 在 DVS 处理器中,可以近似认为动态功率与处理器速度的关系如式(1)。

$$P \propto f^3 \text{ 或 } P \propto S^3 \quad (1)$$

当前节能实时调度的研究主要集中在硬实时系统中,但实际上,大量应用中更多地表现出弱实时系统的特性,具有更复杂的 QoS 需求. 为了在降低能耗的同时保证 QoS 需求,弱硬实时调度算法不失为一种良策. 弱硬实时系统的研究促进了许多实时应用,广泛用于多媒体处理、实时通信和嵌入式控制应用。

Hua 等^[3] 针对嵌入式系统中的多媒体应用,提出在线尽力能量最小化算法和混合离线/在线最小努力算法,但是算法不能提供带有保证的服务质量. AlEnawy 等^[4] 提出了离线计算满足 (m, k) -firm 截止期约束的 CPU 速度的方法,并通过在线速度调整算法来充分利用松弛时间,确保在能量预算范围内动态失效次数最小. Niu 等^[5] 研究了弱硬实时系统中能量最小化的问题,提出了一种混合静态/动态调度方法,在保证 (m, k) 约束的同时有效降低能耗。

混合静态/动态调度方法 MK_{E^R} 既提供了可保证的服务,又提供了较好的节能效果,但是由于任务执行的动态变化,通常实际执行时间远小于最坏情况执行时间,因此能否进一步延长任务执行时间以达到更好的节能效果值得深入的探讨。

幸运的是,反馈控制理论提供了对实际执行时间的估计方法^[6]. Zhu 等^[7-8] 针对硬实时系统,采用多种反馈控制技术,提出一系列优秀的反馈节能调度算法,算法可以有效地估计任务的实际执行时间,进而对任务的执行速度进行调整。

为此,本文在混合静态/动态调度算法中引入简单反馈控制机制,提出基于简单反馈的混合静态/动态节能弱硬实时调度算法 $MK_{E^R} - SF$ 。

2 基本思想

2.1 混合静态/动态调度算法

MK_{E^R} 算法^[5] 中的实时系统包含 n 个彼此独立

的周期任务, $\Gamma = \{\tau_0, \tau_1, \dots, \tau_{n-1}\}$, 每个任务包含无限数量的周期到达的实例,称为工作 (job). 任务 τ_i 使用 5 个参数描述 $(T_i, D_i, C_i, m_i, k_i)$, T_i, D_i ($D_i = T_i$) 和 C_i 分别表示任务 τ_i 的周期、截止期和最坏情况执行时间 (Worst Case Execution Time, WCET), (m_i, k_i) 表示任务 τ_i 的 QoS 需求,即任意 k_i 个连续工作中至少有 m_i 个满足截止期. DVS 处理器具有有限离散电压等级, $\Psi = \{V_1, \dots, V_{\max}\}$, 每种电压对应一个速度. 为了简化,按照处理器的最大速度 S_{\max} 归一化处理,得到速度集合 $\Omega = \{S_1, S_2, \dots, 1\}$. 假设 C_i 是任务 τ_i 在最高电压模式下的最坏情况执行时间. 因此如果 τ_i 在速度 S_j 下执行, τ_i 的最坏情况执行时间为 C_i/S_j 。

由于保证 (m, k) 约束是 NP-难的问题,通常通过研究 (m, k) -pattern 进行讨论. 由 (m, k) -pattern 产生的强制工作集能满足 (m, k) 约束,文献[5]中给出几种常用的 (m, k) -pattern: R -pattern、 E -pattern 和 E^R -pattern, 并证明了 E -pattern 和 E^R -pattern 的最优性,即对于一个任务集,如果存在可调度的 (m, k) -pattern, 则 E -pattern 和 E^R -pattern 一定可以被调度。

基于 (m, k) -pattern 的静态分析用于保证强制工作集合的可调度性. 然而,静态分析通常仅考虑最坏情况,比较悲观. 为了适应运行时的变化,需要在可选工作满足截止期时,将其它强制工作降级为可选工作,以更有效地节能. 为此, MK_{E^R} 算法根据 E^R -pattern 确定强制和可选工作,一旦可选工作满足截止期,就从下一工作开始重启 E^R -pattern。

MK_{E^R} 算法包含离线阶段和在线阶段. 离线阶段确定每个任务的处理器速度和 E^R -pattern 下的每个任务的最坏情况响应时间. 在线阶段,对强制工作采用双优先级模式,维护 3 个就绪队列:高强制队列 (High Mandatory Queue, HMQ)、可选队列 (Optional Queue, OPQ) 和低强制队列 (Low Mandatory Queue, LMQ). 当工作到达时,根据当前的 E^R -pattern 决定是强制工作还是可选工作. 可选工作直接放入 OPQ 中,强制工作首先放入 LMQ,在固定时间之后提升到 HMQ,这段固定时间称为提升时间 (promotion time), 记为 Y_i , $Y_i = D_i - R_i$, D_i 是任务 τ_i 的相对截止期, R_i 是在离线阶段计算得到的最坏情况响应时间. HMQ 中的工作具有最高优先级,按照 EDF 进行调度,以离线阶段确定的处理器速度执行. OPQ 中的工作的优先级高于 LMQ 中的工作,且可选工作彼此不能抢占,仅当其能够在最

早提升时间之前完成才进行调度. 通常 OPQ 中有多个工作可以调度, 采用以下方式选取, 当 HMQ 为空时, 首先计算完成每个可选工作所需要的速度 \hat{S}_i , 淘汰速度大于预定义速度 S_i 的工作, 然后定义每个工作的能量增益 $\Delta E_i = E(S_i) - E(\hat{S}_i)$, 选取拥有最大 ΔE_i 的工作进行调度. 当 HMQ 为空, OPQ 中没有可执行的工作时, 按照 EDF 策略以最低速度调度 LMQ 中的工作.

混合算法的节能高效性在于可以在运行过程中自适应地调整强制/可选的划分. 能否进一步降低强制工作的执行速度以达到更好的节能效果呢? 针对这一问题, 下面将展开讨论.

2.2 任务划分

Zhu 等^[8]提出将任务划分成两部分, 即将任务 τ_i 分为两个子任务 τ_A 和 τ_B , 并允许两个子任务以不同的速度执行, 前一子任务以较低速度执行, 而后一子任务以最高速度执行.

在 MK_{E^R} 算法中, 离线阶段预先计算出任务 τ_i 的强制工作的执行速度 S_i , 由此可以得出强制工作在最坏情况执行时间是 C_i/S_i , 如图 1, τ_B 总是执行在最高速度 S_{\max} , 而 τ_A 以较低速度执行; C_i 、 C_A 和 C_B 分别对应着任务 τ_i 、子任务 τ_A 和 τ_B 在 S_{\max} 下的最坏情况执行时间, α 为 τ_A 执行时的速度调节因子, t 为当前时间, $t + C_i/S_i$ 是 τ_i 的最大完成时间; 虚线部分为任务 τ_i 的强制工作的预定义速度 S_i . 通过这种方法, 希望任务 τ_i 能在 τ_A 内完成其实际执行, 但如果它具有最坏计算需求时, τ_B 会预留足够的时间以满足该任务的截止期要求.

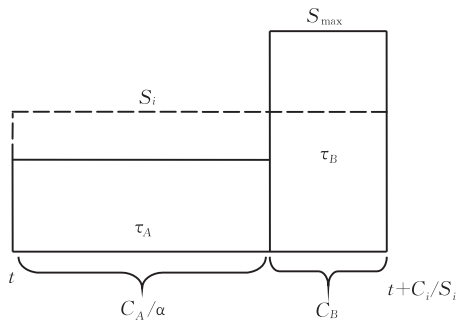


图 1 强制工作划分示意图

显然, 可选工作的执行时间也是确定的, 即从强制工作完成到最近的一次低强制工作提升之间的时间, 根据执行时间和 OPQ 中的各个工作可以计算出每个工作所需要的执行速度 \hat{S}_i , 采用 MK_{E^R} 中选择候选工作的方法, 即选取具有最大 ΔE_i 的工作. 类似地, 在执行该可选工作时也将其划分为两部分, 由

于 $\hat{S}_i < S_i$, 因此为了更高效地节能, 可选工作的 τ_B 以速度 S_i 执行, 显然 C_i/\hat{S}_i 是可选工作的最大执行时间, 如图 2 所示, 图中的 C_A 和 C_B 不再是 S_{\max} 下的最坏情况执行时间, 而是在 S_i 下的最坏情况执行时间, 同样 α 也是针对 S_i 的调节因子.

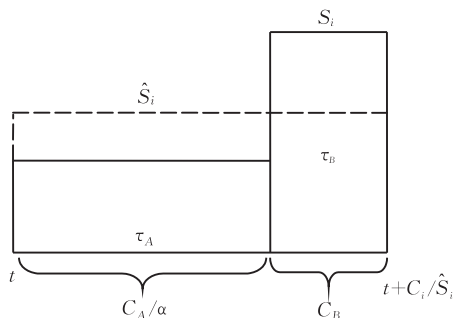


图 2 可选工作划分示意图

这种任务划分方法最多可以把每个任务分为两个子任务, 使得在任务执行过程中最多可能会增加一次额外的速度改变. 此时任务划分以对用户透明的方式, 即由操作系统来完成, 可以利用一个定时器来实现. 当调度执行 τ_A 时设置该定时器, 在 τ_A 完成时触发改变处理器电压和频率. 如果任务在 τ_A 内完成执行或发生抢占, 则定时器会被撤消, 没有额外开销. 只有当任务没有在 τ_A 内完成, 定时器才会触发电压和频率调节去执行 τ_B .

下面给出基于 MK_{E^R} 算法, 强制工作和可选工作的速度调节因子的计算方法.

强制工作, 已知 $C_i = C_A + C_B$, $\frac{C_A}{\alpha} + C_B = \frac{C_i}{S_i}$, 则

$$\alpha = \frac{C_A S_i}{C_i - C_i S_i + C_A S_i} \quad (2)$$

式(2)说明强制工作中 τ_A 的速度调节因子 α 不仅依赖于预定义速度 S_i , 还依赖于 τ_A 在最高速度下的最坏情况执行时间 C_A .

可选工作, 已知 $\frac{C_i}{S_i} = C_A + C_B$, $\frac{C_A}{\alpha} + C_B = \frac{C_i}{\hat{S}_i}$, 则

$$\alpha = \frac{C_A S_i \hat{S}_i}{C_i S_i - C_i \hat{S}_i + C_A S_i \hat{S}_i} \quad (3)$$

式(3)说明可选工作中 τ_A 的速度调节因子 α 不仅依赖于速度 S_i 和 \hat{S}_i , 还依赖于 τ_A 在速度 S_i 下的最坏情况执行时间.

由于 S_i 和 \hat{S}_i 在离线阶段已经可以直接或间接确定(\hat{S}_i 是通过离线阶段确定的提升时间而间接确定的), 因此下面主要分析如何确定 C_A 的值.

2.3 简单反馈

反馈控制器计算任务的实际执行时间与 C_A 的差值. 电压-频率选择器根据调整后的 C_A 值选择电压/频率等级, 从而确定速度等级. 速度等级确定后, 利用 MK_{E^R} 进行调度. 每个工作的实际执行时间进一步反馈到反馈控制器以进行下一步决策. 反馈框架如图 3 所示.

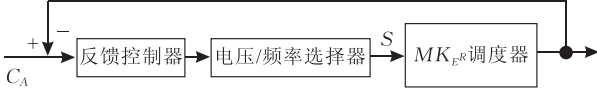


图 3 反馈框架

由于 C_A 是基于估计的子任务 τ_A 的执行时间的, 因此目标是让 C_A 近似等于 τ_{ij} 的实际执行时间 C_{ij} , 以使得 τ_{ij} 能在进入子任务 τ_B 之间完成. 如果 C_A 低于 τ_{ij} 的实际执行时间, 那么 τ_{ij} 完全执行在低速度等级上, 而不必切换到 S_{\max} . 这时的调度几乎是最佳的节能调度. 但是不幸的是, 实际应用中每个工作的实际执行时间是动态变化的, 所以在每次调度时, 要想使 C_A 恰好等于任务的实际执行时间 c_{ij} , 几乎是不可能的. 为此, 基于简单反馈控制机制, 采用以下方式估计 C_A 值.

$$C_{A_{i(j+1)}} = (C_{A_{ij}} \times N + c_{ij} - c_{i(j-N)}) / N, C_{A_{i0}} = C_i / 2 \quad (4)$$

一旦确定了 C_A , 可以根据式(2)和(3)获取强制工作和可选工作在 τ_A 内完成的理想速度调节因子 α .

3 算法描述

本节给出基于简单反馈的混合静态/动态弱硬实时调度算法 $MK_{E^R} - SF$. 算法在离线阶段, 保持 MK_{E^R} 的操作不变, 计算强制工作的最佳执行速度和任务的最坏情况响应时间. 在线阶段基于简单反馈机制分配任务的执行速度, 增加一个中断定时器, 当发生中断处理的进行速度切换. 算法中 $C_{A_{ij}}$ 是估计的子任务 τ_A 的执行时间, 而 C_{A_i} 是实际分配给子任务的执行时间. 伪代码如图 4.

4 算法特性

定理 1 给出 $MK_{E^R} - SF$ 算法的可调度性.

定理 1. 基于简单反馈的混合静态/动态算法 $MK_{E^R} - SF$, 不会影响混合静态/动态算法 MK_{E^R} 的可调度性.

$MK_{E^R} - SF$ Algorithm

```

if  $HMQ \neq \emptyset$  then
    根据 EDF 选取执行的工作  $\tau_{ij}$ ;
    Schedule_HMQ();
else if  $OPQ \neq \emptyset$  then
    if 可以选取出对节能最有利的工作  $\tau_{ij}$  then Schedule_OPQ();
    else Schedule_LMQ();
end if
else
    Schedule_LMQ();
end if
Schedule_HMQ();
    计算强制工作  $\tau_{ij}$  的速度调节因子,
     $\alpha_i = \text{Calculate\_Ratio}(\text{Mandatory})$ ;
    设定速度,  $\text{Select\_Speed}(\alpha_i)$ ;
    执行  $\tau_{ij}$ ;
Schedule_OPQ();
    计算可选工作  $\tau_{ij}$  的速度调节因子,
     $\alpha_i = \text{Calculate\_Ratio}(\text{Optional})$ ;
    设定速度,  $\text{Select\_Speed}(\alpha_i)$ ;
    执行  $\tau_{ij}$ ;
    if  $\tau_{ij}$  在截止期之前完成 then
        重新开始任务  $\tau_i$  的  $E^R$ -pattern;
    end if
Schedule_LMQ();
    选择具有最早提升时间的工作  $\tau_{ij}$ ;
    以最低的速度执行  $\tau_{ij}$ ;
Calculate_Ratio(type):
    if  $\tau_{ij}$  被定时器中断处理且未完成 then
        撤销定时器;
        if type = Mandatory then Return(1); // 强制工作
        else Return ( $S_i / S_{\max}$ ); // 可选工作
    end if
    end if
    if  $\tau_{ij}$  曾被别的任务抢占 then
        // 仅可能发生在 HMQ 或 LMQ 中的任务
        if  $\tau_{ij}$  的上次执行是在 HMQ 中 then
            if  $e_{ij} \geq C_{A_i}$  then Return(1);
            //  $e_{ij}$  是已完成部分在最高速度下的执行时间
        else
            Set_Interrupt( $\tau_{ij}, (C_{A_i} - e_{ij}) / \alpha_i$ );
            Return( $\alpha_i$ );
        end if
    else if  $\tau_{ij}$  的上次执行是在 LMQ 中 then
         $\alpha_i = \max \left\{ \frac{S_{\min}}{S_{\max}}, \frac{C_{A_{ij}} S_i}{C_i - C_i S_i + C_{A_{ij}} S_i} \cdot \frac{C_{A_{ij}}}{C_{A_{ij}} - e_{ij}} \right\}$ ,
         $C_{A_i} = \frac{(1 - S_i) \alpha_i C_i}{(1 - \alpha_i) S_i}$ ;
        Set_Interrupt( $\tau_{ij}, (C_{A_i} - e_{ij}) / \alpha_i$ );
        Return( $\alpha_i$ );
    end if
    end if
    if  $j = 1$  then
         $C_{A_{ij}} = C_i / 2$ ;
    end if
    if type = Mandatory then
         $\alpha_i = \max \left\{ \frac{S_{\min}}{S_{\max}}, \frac{C_{A_{ij}} S_i}{C_i - C_i S_i + C_{A_{ij}} S_i} \right\}$ ,  $C_{A_i} = \frac{(1 - S_i) \alpha_i C_i}{(1 - \alpha_i) S_i}$ ;
    else if type = Optional then
         $\alpha_i = \max \left\{ \frac{S_{\min}}{S_{\max}}, \frac{C_{A_{ij}} S_i \hat{S}_i}{C_i S_i - C_i \hat{S}_i + C_{A_{ij}} S_i \hat{S}_i} \right\}$ ,
         $C_{A_i} = \frac{(\hat{S}_i - S_i) \alpha_i C_i}{(1 - \alpha_i) S_i \hat{S}_i}$ ;
    end if
    Set_Interrupt( $\tau_{ij}, C_{A_i} / \alpha_i$ );
    Return( $\alpha_i$ ); // 返回速度调节因子  $\alpha_i$ 
Select_Speed( $\alpha_i$ ):
     $S = \alpha_i \cdot S_{\max}$ ;
    Set_Interrupt( $\tau_{ij}, t$ ):
        设置定时器为工作  $\tau_{ij}$  提供中断服务, 在  $t$  时间后触发;
    Upon Task_completion( $\tau_{ij}$ ):
         $C_{A_{i(j+1)}} = (C_{A_{ij}} \times N + c_{ij} - c_{i(j-N)}) / N$ ;

```

图 4 算法伪代码描述

证明. 只需证明 $MK_{E^R} - SF$ 算法不会导致强制工作的执行时间超过预定义的执行时间, 原命题则成立. 存在以下 3 种情况, 一是 HMQ 中任务直接完成、二是 HMQ 中任务曾被抢占、三是 LMQ 中任务在提升前曾被执行, 根据速度分配策略, 这 3 种情况都不会导致执行时间超过 C_i/S_i . 因此 $MK_{E^R} - SF$ 不会影响 MK_{E^R} 的可调度性. 证毕.

推论 1. 在静态节能调度算法中引入任务划分的反馈思想不会影响原算法的可调度性.

证明. 静态节能调度算法都定义了任务的最大完成时间, 而任务划分不会导致执行时间超过预定义的最大完成时间, 因此不会对原算法的可调度性产生影响. 证毕.

由于任务的实际执行时间通常都低于其最坏情况执行时间, 因此基于反馈的方法可以延长其执行时间, 进而达到进一步节能的效果, 定理 2 给出了最理想情况下的节能增益.

定理 2. 强制工作的执行速度为 S_1 , 其子任务 τ_A 的执行速度为 S_2 , 则最理想情况下 $MK_{E^R} - SF$ 可比 MK_{E^R} 节能 $1 - \frac{S_2^2}{S_1^2}$.

证明. 最理想的情况就是强制工作在 τ_A 内完成, 则有 $S_1 t_1 = S_2 t_2$, t_1 为强制工作在 S_1 下的执行时间, t_2 为强制工作在 S_2 下的执行时间.

根据式(1), 有 $E_1 = \beta S_1^3 t_1$, $E_2 = \beta S_2^3 t_2$, 两式相除, 并将 $S_1 t_1 = S_2 t_2$ 代入得 $\frac{E_2}{E_1} = \frac{\beta S_2^3 t_2}{\beta S_1^3 t_1} = \frac{S_2^2}{S_1^2}$. 即 $MK_{E^R} - SF$ 比 MK_{E^R} 节能 $1 - \frac{S_2^2}{S_1^2}$. 证毕.

尽管 $MK_{E^R} - SF$ 在多数情况下节能效果优于 MK_{E^R} , 但是当强制工作的实际执行时间等于最坏情况执行时间时, $MK_{E^R} - SF$ 劣于 MK_{E^R} , 见定理 3.

定理 3. 如果强制工作的实际执行时间等于最坏情况执行时间, $MK_{E^R} - SF$ 比 MK_{E^R} 更加耗能.

证明. 强制工作 τ_{ij} 的实际执行时间等于最坏情况执行时间, 在 MK_{E^R} 算法下, 强制工作 τ_{ij} 以 S_1 执行 t_1 时间, 在 $MK_{E^R} - SF$ 算法下, 强制工作 $MK_{E^R} - SF$ 以 S_2 执行 t_2 时间, 然后再以 S_{\max} ($S_{\max} = 1$) 执行 t_3 时间.

假设 $S_2 < S_1 < S_{\max}$, 否则必有 $S_2 = S_1 = S_{\max}$, 这时两种算法耗能相等.

存在方程组 $\begin{cases} t_1 = t_2 + t_3 \\ S_1 t_1 = S_2 t_2 + t_3 \end{cases}$,

$$\text{将 } t_1 \text{ 和 } t_3 \text{ 用 } t_2 \text{ 表示得到 } \begin{cases} t_1 = \frac{(1-S_2)t_2}{1-S_1} \\ t_3 = \frac{(S_1-S_2)t_2}{1-S_1} \end{cases}.$$

根据式(1), MK_{E^R} 算法耗能 $E_1 = \beta S_1^3 t_1$, $MK_{E^R} - SF$ 算法耗能 $E_2 = \beta(S_2^3 t_2 + t_3)$, 两式相除并将上式代入得到

$$\begin{aligned} \frac{E_1}{E_2} &= \frac{\beta S_1^3 t_1}{\beta(S_2^3 t_2 + t_3)} = \frac{S_1^3(1-S_2)}{S_2^3(1-S_1) + (S_1-S_2)} \\ &= \frac{S_1^3(1-S_2)}{S_1(1-S_2^2) - S_2(1-S_2^2)} \\ &= \frac{S_1^3}{S_1(1+S_2+S_2^2) - S_2(1+S_2)}. \end{aligned}$$

欲证 $\frac{E_1}{E_2} \leq 1$, 只需证

$$\frac{S_1^3}{S_1(1+S_2+S_2^2) - S_2(1+S_2)} \leq 1,$$

即

$$S_1^3 - S_1 - S_1 S_2 - S_1 S_2^2 + S_2 + S_2^2 \leq 0,$$

则有

$$S_1(S_1^2 - S_2^2) - (S_1 - S_2) - S_2(S_1 - S_2) \leq 0.$$

因为 $S_2 < S_1$, 则有 $S_1 - S_2 > 0$, 则有

$$S_1(S_1 + S_2) - 1 - S_2 < 0,$$

进一步有 $(S_1 - 1)(S_1 + 1) + S_2(S_1 - 1) < 0$, 因为 $S_1 - 1 < 0$, 则有 $S_1 + 1 + S_2 > 0$, 显然成立, 证得 $\frac{E_1}{E_2} \leq 1$. 证毕.

5 实 验

本节通过实验比较 MK_{E^R} 和 $MK_{E^R} - SF$ 的性能. 对于 MK_{E^R} 算法, 为了便于离线阶段的预先计算, 处理器具有 5 个离散电压等级, 对应地, 可以提供 5 个处理器速度, 归一化后得到 (0.2, 0.4, 0.6, 0.8, 1.0). 而对于 $MK_{E^R} - SF$, 考虑理想情况, 即电压/频率等级可以连续调节. 根据式(1)计算能耗, 忽略转变开销.

为了评估算法的性能, 引入两个环境参数:

(1) 平均情况执行时间 (Average Condition Execution Time) 与最坏情况执行时间 (Worst Condition Execution Time, WCET) 之比, 即 ACET/WCET. 由于 $MK_{E^R} - SF$ 算法的性能受实际执行时间的影响很大, 因此在实验中通过比较不同 ACET/WCET 下的执行情况进行性能比较.

(2) (m, k) 利用率. 所谓 (m, k) 利用率, 定义为

$$\sum_i \frac{m_i C_i}{k_i T_i}, \text{ 刻画了系统的实际负载情况.}$$

首先研究 ACET/WCET 对算法性能的影响,随机产生周期任务集,周期从 $[10, 50]$ 中随机选取,假设截止期等于周期.任务在最高电压条件下的最坏情况执行时间在 1 到截止期之间均匀分布,ACET/WCET 从 0.2 增加到 1,步长为 0.1,实际执行时间根据 ACET/WCET 的值确定,约束中的 k_i 在 3~10 之间均匀分布,而 $2 \leq m_i \leq k_i$,固定 (m, k) 利用率为 0.8.对产生的任务集进行可调度性测试,以保证在每个间隔内至少有 20 个可调度的任务集.仿真时间为 20000.以 MK_{ER} 的能耗作为标准进行归一化,结果如图 5,当 ACET 大幅度低于 WCET 时 $MK_{ER}-SF$ 比 MK_{ER} 节能将近 60%,随着比值的增大,节能效果逐渐降低,当 ACET 与 WCET 接近时, $MK_{ER}-SF$ 比 MK_{ER} 多耗能接近 10%.

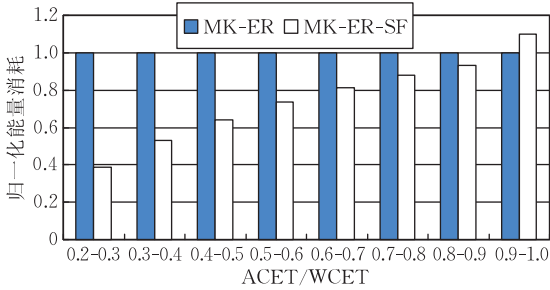


图 5 ACET/WCET 对性能的影响

然后研究 (m, k) 利用率对算法性能的影响,固定 ACET/WCET = 0.5, (m, k) 利用率从 0 增加到 1,步长为 0.1.任务参数产生的方式同前,依然保证在每个间隔内至少有 20 个可调度的任务集.仿真时间为 20000.将能耗以 MK_{ER} 的能耗进行归一化,以 MK_{ER} 的能耗作为标准进行归一化,结果如图 6,随着 (m, k) 利用率的增大,节能效果逐渐减弱,最少可节能约 10%,最多可节能近 70%.

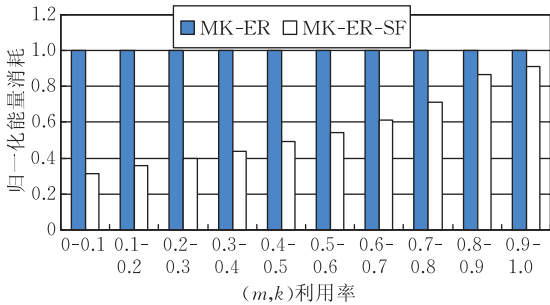


图 6 (m, k) 利用率对性能的影响

6 结束语

本文基于混合静态/动态节能弱硬实时调度算

法 MK_{ER} ,提出基于简单反馈的混合静态/动态节能弱硬实时调度算法 $MK_{ER}-SF$.算法针对任务的实际执行时间通常远小于最坏情况执行时间的实际情况,引入任务划分,并通过简单反馈控制机制尽量延长工作的执行时间,以获取更低的执行速度,达到进一步节能的目的.本文还分析了 $MK_{ER}-SF$ 的可调度性和节能特性.最后通过实验与原算法进行比较,结果表明当 ACET 低于 WCET 较多时可以取得最高 60%~70%,最低约 10%的节能增益.

由于在电压/频率的切换过程中会引入开销,下一步工作准备在调度中引入对切换开销的考虑.

参 考 文 献

- [1] Rele S, Pande S, Onder S et al. Optimizing static power dissipation by functional units in superscalar processors//Lecture Notes in Computer Science 2304, 2002: 85-100
- [2] Curt S, Olivier A, Mani S. Modulation scaling for energy aware communication systems//Proceedings of the 2001 International Symposium on Low Power Electronics and Design. New York, USA: ACM, 2001: 96-99
- [3] Hua S, Qu G, Bhattacharyya S S. Energy reduction techniques for multimedia applications with tolerance to deadline misses//Proceedings of the 40th Conference on Design Automation. New York, USA, 2003: 131-136
- [4] AlEnawy T A, Aydin H. Energy-constrained scheduling for weakly-hard real-time systems//Proceedings of the 26th IEEE International Real-Time Systems Symposium. Washington, DC, USA, 2005: 376-385
- [5] Niu Linwei, Quan Gang. A hybrid static/dynamic DVS scheduling for real-time systems with (m, k) -guarantee//Proceedings of the 26th IEEE International Real-Time Systems Symposium. Washington, DC, USA, 2005: 356-365
- [6] Sha L, Abdelzaher T, Karl Eric et al. Real time scheduling theory: A historical perspective. Real-Time System, 2004, 28 (2-3): 101-155
- [7] Zhu Y F, Mueller F. Feedback EDF scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling. ACM SIGPLAN Notices, 2005, 40(7): 203-212
- [8] Zhu Y F, Mueller F. Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling. Real-Time System, 2005, 31(1-3): 33-63
- [9] Chandrakasan A P, Sheng S, Brodersen R W. Low-power cmos digital design. IEEE Journal of Solid-State Circuits, 1992, 27(4): 473-484



WU Tong, born in 1979, Ph. D. .
His current research interests include real-time system.

ZHANG Dong-Song, born in 1980, Ph.D. candidate.
His current research interests focus on real-time system.

JIN Shi-Yao, born in 1937, professor, Ph. D. supervisor. His current research interests include real-time system, distributed computing, systems modeling and simulation, system performance estimation, fault-tolerant system.

Background

Nowadays, energy-aware scheduling is attracted in the research of real-time systems, in which Dynamic Voltage Scaling (DVS) technique is generated from hardware based on low-power design and has been one of key technologies in real-time system. DVS can be easily included into real-time scheduling methods to make energy savings by scaling the voltage and frequency while maintaining real-time deadline guarantees.

Currently, most existing energy-aware scheduling algorithms are focused on the hard real-time system. However, many practical real-time applications exhibit more complicated characteristics, which is called the Quality of Service (QoS) requirements. For example some applications may have soft deadlines where tasks which do not finish by their deadlines can still be completed with a reduced value; or they can simply be dropped without compromising the desired QoS levels. The techniques based on the traditional hard real-time systems become inefficient or inadequate when QoS requirements are imposed on the systems. And so the energy-aware weakly-hard real-time scheduling algorithms are introduced to

solve this problem.

In this paper, after analyzing the shortages of the hybrid static / dynamic algorithm, we proposed a new algorithm, in which, the simple feedback mechanism is introduced to estimate the actual execution time. The whole speed is decreased by splitting the job into two parts. The execution time is prolonged, and so the new algorithm is more energy efficient.

Authors have done research on weakly-hard real-time system for many years. By these studies, they have a deep understanding of weakly-hard real-time system and publish many papers in this area.

This paper contributes on the field of energy-aware weakly-hard real-time scheduling to get more energy savings. The experimental results show that the proposed algorithm outperforms the original one when the Average Case Execution Time (ACET) is much less than the WCET, which can improve energy savings about 60% to 70% at most and about 10% at least. Unfortunately, when the ACET is close to the WCET, the proposed algorithm consumes more energy than the original one.