开销敏感的多处理器最优节能实时调度算法

张冬松[1)] 吴 飞[2)] 陈芳园[1)] 吴 彤[3)] 郭得科[4)] 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国家重点实验室 长沙 410073)

[2)](上海工程技术大学电子电气工程学院 上海 201620)

[3)](国防科学技术大学国家安全与军事战略研究中心 长沙 410073)

[4)](国防科学技术大学信息与管理学院信息系统工程国家重点实验室 长沙 410073)

摘 要 嵌入式多处理器系统的能耗问题变得日益重要,如何减少能耗同时满足实时约束成为多处理器系统节能实时调度中的一个重要问题.目前绝大多数研究基于关键速度降低处理器的频率以减少动态能耗,采用关闭处理器的方法减少静态能耗.虽然这种方法可以实现节能,但是不能保证最小化能耗.而现有最优的节能实时调度未考虑处理器状态切换的时间和能量开销,因此在切换开销不可忽视的实际平台中不再是最优的.文中针对具有独立动态电压频率调节和动态功耗管理功能的多处理器系统,考虑处理器切换开销,提出一种基于帧任务模型的最优节能实时调度算法.该算法根据关键速度来判断系统负载情况,确定具有最低能耗值的活跃处理器个数,然后根据状态切换开销来确定最优调度序列.该算法允许实时任务在处理器之间任意迁移,计算复杂度小,易于实现.数学分析证明了该算法的最优性.

关键词 实时系统;多处理器;节能调度;动态电压频率调节;动态功耗管理;绿色计算

中图法分类号 TP316 **DOI号**: 10.3724/SP.J.1016.2012.01297

An Overhead-Aware Optimal Energy-Efficient Real-Time Scheduling Algorithm on Multiprocessors

ZHANG Dong-Song[1)] WU Fei[2)] CHEN Fang-Yuan[1)] WU Tong[3)] GUO De-Ke[4)] JIN Shi-Yao[1)]

[1)](*National Laboratory of Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha* 410073)

[2)](*College of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai* 201620)

[3)](*Center for National Security and Strategic Studies, National University of Defense Technology, Changsha* 410073)

[4)](*National Laboratory of Information System Engineering, College of Information System and Management, National University of Defense Technology, Changsha* 410073)

Abstract As the energy consumption of embedded multiprocessor systems becomes increasingly prominent, the energy-efficient real-time scheduling in multiprocessor systems becomes an urgent problem. Most research efforts are based on the critical speed to reduce frequencies of processors for reducing the dynamic power consumption. Meanwhile, off-processor approach is also used to reduce the static power consumption. However, such approaches cannot achieve the minimum energy savings. In the other hand, the optimal energy-efficient real-time scheduling ignores the time and energy overhead of switching the processor state and thus is not optimal in actual platforms. For multiprocessors with the independent dynamic voltage frequency and dynamic power manage-

收稿日期:2011-08-11;最终修改稿收到日期:2012-04-09.本课题得到国家自然科学基金(61170284,60903206)、国家教育部博士点基金(20104307110005)、博士后面上基金(20100480898,201104439)、上海市教委科研创新项目(12ZZ182)、国防科学技术大学优秀研究生创新项目(B100601)以及湖南省研究生科研创新项目(CX2010B026)资助.**张冬松**,男,1980年生,博士,中国计算机学会(CCF)会员,主要研究方向为实时系统、复杂系统仿真.E-mail: dszhang@nudt.edu.cn.**吴 飞**,男,1968年生,博士,副教授,主要研究方向为信息智能处理、分布计算.**陈芳园**,女,1982年生,博士,主要研究方向为计算机体系结构、实时系统.**吴 彤**,男,1979年生,博士,副研究员,主要研究方向为实时系统、军事高科技.**郭得科**,男,1980年生,博士,副研究员,主要研究方向为无线多跳网络、实时系统、对等计算、数据中心互联.**金士尧**,男,1937年生,教授,博士生导师,主要研究领域为实时系统、复杂系统仿真、分布式计算、虚拟现实等.

ment, this paper proposes an optimal energy-efficient real-time scheduling algorithm for the frame-based tasks. The proposed optimal algorithm determines the system workload cases and the number of active processor cores in terms of the critical speed. Then we can obtain the optimal scheduling according to the switching overhead. The algorithm allows tasks to arbitrarily migrate across processors during their executions at the cost of the small computational complexity. Furthermore, it is easy to be implemented. Mathematical analysis shows that the algorithm is optimal.

Keywords real-time systems; multiprocessor; energy-efficient scheduling; dynamic voltage frequency scaling; dynamic power management; green computing

1 引 言

随着芯片制造商正在推出多核芯片和片上多处理器系统,多处理器平台变得更加普遍,受到了智能嵌入式实时系统领域越来越多的关注.处理器的高性能会带来高能耗.能耗管理已经成为嵌入式实时系统中一个亟待解决的问题,尤其对于电池供电的嵌入式系统非常重要.

电容切换活动产生的动态功耗和泄露电流产生的静态功耗是 CMOS 处理器能耗的两个主要来源[1].动态电压频率调节(Dynamic Voltage Frequency Scaling,DVFS)技术和动态功耗管理(Dynamic Power Management,DPM)技术已经被应用于现代处理器系统中,DVFS 可以在运行时降低供应电压和执行频率以减少处理器的动态功耗,而 DPM 可以在运行时关闭处理器以减少处理器的静态功耗.由于动态功耗通常是速度(或称频率)的凸函数和递增函数,所以一般来说速度越低,动态能耗越少.但是,较低的执行速度会延长实时任务的执行时间,又因泄露电流的影响而导致更多的静态能耗.因此,节能实时调度研究需要兼顾动态能耗和静态能耗,在满足实时约束的条件下最小化能耗[2].

早期的多处理器节能实时调度技术研究表明:在只考虑动态功耗的条件下,为了最小化能耗,最优的任务分配必须使得任务集的总负载在多个处理器上均匀分配[3].典型的方法是基于多种装箱问题启发式方法如 FFD(First-Fit Decreasing)、NFD(Next-Fit Decreasing)、BFD(Best-Fit Decreasing)和 WFD(Worst-Fit Decreasing),先将任务分配到指定的处理器上,然后利用单处理器平台中提出的各种节能实时调度算法来独立调节每个处理器的电压和频率.这样便可以将复杂的多处理器节能实时调度问题转化为多个简单的单处理器节能实时调度问题.但是,Aydin 等人[3]已经证明若将任务分配到固定数目的处理器上使其既具有最优的负载均衡又保证任务集可调度性是一个 NP-hard 问题.还有一些研究者基于全局调度法提出最优的节能实时调度算法.Chen 等人[4-5]针对基于帧的任务模型,提出一种最优的节能实时调度算法 LTF-M. Funaoka 等人[6]则针对周期任务模型,提出了一种最优的节能实时调度算法.

但是,随着纳米制造工艺的不断进步,CMOS 电路中静态功耗相比动态功耗,在总功耗中所占比重越来越大[1].为了减少因泄露电流所产生的静态能耗,处理器可以利用 DPM 技术进入睡眠状态(又称关闭)以及从睡眠状态返回活跃状态(又称唤醒).一些研究者针对不可忽视的静态功耗提出了多处理器节能实时调度算法.这些研究成果[5,7-9]通过将处理器转入睡眠状态来减少静态功耗.Xu 等人[7]根据工作负载确定活跃处理器的个数,提出节能实时调度算法.de Langen 等人[8]则针对具有离散速度的系统提出启发式节能实时调度算法.Chen 等[5]针对处理器速度独立调节和统一调节的两种不同处理器模型,基于帧的任务模型提出具有不同近似比的静态节能调度算法,同时他们的成果还进一步扩展到周期任务模型[9].

以上所提出的绝大多数调度算法均假设处理器具有活跃状态和睡眠状态,基于关键速度来设定活跃状态时的执行速度,基于状态切换开销来设定何时将处理器转入睡眠状态.这里关键速度是指具有最小能耗的可用执行速度,并且乐观地假设在执行任务之后将处理器转入睡眠状态.但是,处理器关闭和唤醒需要一定的时间和能耗开销.例如,在 70 纳米技术中 Transmeta 处理器具有 483 微焦的能耗开销和至少 2 ms 的时间开销[1].如果状态转换的能耗

开销大于能耗节余，那么显然把处理器转入睡眠状态可能不会节能[10].

目前，多处理器系统中节能实时调度研究已经开始关注具有切换开销的实际平台. 文献[10]通过说明关键速度的非最优性以及负载均衡的非最优性，针对基于帧的任务模型和周期任务模型，提出了一种多项式时间的近似度为 1.21 的节能实时调度算法. 该算法显著地优于原有未考虑开销情况下近似度为 1.667 的近似算法[9]. 但是，现有的研究成果[4-5,7-10]主要存在三个方面的问题：(1) 大多数节能实时调度算法基于划分调度法或非最优的全局调度法，无法保证实时任务集的最优可调度性. (2) 现有节能算法常采取降频的方法来减少动态功耗，采用关闭处理器的方法来进一步减少静态功耗，虽然这种方法可以实现节能，但是不能保证最小化能耗. (3) 现有最优的节能实时调度算法假设没有处理器状态切换的时间和能耗开销，因此在切换开销不可忽视的实际平台中不一定最优.

本文针对具有独立 DVFS 的多处理器系统，在考虑处理器状态切换开销情况下，提出一种基于帧任务模型的最优节能实时调度算法(Largest Utilization task First based on Switching Overhead, LUF-SO). 该算法在调度过程中允许实时任务在任意处理器之间迁移，可以在离线调度中确定所有任务的执行过程和执行速度. 同时，LUF-SO 算法与 LTF-M 算法的主要区别在于：LUF-SO 算法将根据关键速度来判断系统的低负载情况，一旦确定系统处于低负载情况，首先确定具有最低能耗值的活跃处理器个数，然后根据状态切换时间和能量开销来确定最优调度序列；而 LTF-M 算法在系统处于低负载情况不再保持最优性. LUF-SO 算法实现简单且复杂度小. 系统的理论分析证明了该算法的最优性.

本文第 2 节介绍系统模型，并给出问题定义；第 3 节结合实际切换开销，对现有算法 LTF-M 的非最优性进行分析；LTF-SO 算法的详细描述以及最优性分析和证明在第 4 节中给出；第 5 节示例说明算法的最优节能效果，并与 LTF-M 算法进行比较；第 6 节对全文作简要总结.

2　系统模型与问题定义

本节提出本文所用的处理器模型、任务模型以及能耗模型，并基于模型给出问题定义.

2.1　处理器模型

假设多处理器系统由 m 个同构处理器构成，m 为常数，处理器命名为 $cpu_1, cpu_2, \cdots, cpu_m$. 处理器 cpu_k 以速度 s 运行时的功耗 $P_k(s)$ 为 $P_k(s)=P_k^{\mathrm{dep}}(s)+P_k^{\mathrm{ind}}$[11]. $P_k^{\mathrm{dep}}(s)$ 和 P_k^{ind} 分别表示与速度相关和与速度无关的功耗[11]. $P_k^{\mathrm{dep}}(s)$ 主要由门电路电容充放电引起的动态功耗和短路瞬态电流产生的短路功耗构成. 根据文献[11-13]可知，与速度相关的功耗 $P_k^{\mathrm{dep}}(s)$ 可表示为 $P_k^{\mathrm{dep}}(s)=\alpha s^3$. 与速度无关的功耗 P_k^{ind} 主要来自于漏电流产生的泄露功耗，可设为 $P_k^{\mathrm{ind}}=\beta$[20]，其中 β 是一个常量. 因此，功耗函数可表示为

$$P_k(s)=\alpha\cdot s^3+\beta\text{[11]},\ \alpha>0,\beta\geq 0.$$

这种功耗模型适用于多种 DVFS 处理器，例如 Intel Xscale 处理器[11]. 显然，$P(s)$ 是严格凸函数和递增函数. 假设同构多处理器可以在最低处理器速度 $s_{\min}$ 和最高处理器速度 $s_{\max}$ 之间连续调节.

处理器以速度 s 执行一个时钟周期的能耗为 $P(s)/s=\alpha\cdot s^2+\beta/s$. $P(s)/s$ 是凸函数[13]，且使得 $P(s)/s$ 具有最小值的速度即是关键速度 s^*. 这里关键速度定义为处理器执行一个时钟周期所用能耗最小时的速度[1,9-10]. 例如，当 $P_k^{\mathrm{dep}}(s)=\alpha s^\gamma+\lambda s$ 时，关键速度 s^* 等于 $\max\left\{\min\left\{\sqrt[\gamma]{\frac{P_k^{\mathrm{ind}}}{\alpha(\gamma-1)}}, s_{\max}\right\}, s_{\min}\right\}$[10]. 显然，$s^*$ 满足如下性质[10-11]：

$$\forall s, s_{\min}\leq s\leq s_{\max},\ P(s^*)/s^*\leq P(s)/s.$$

处理器有两种状态：睡眠状态和活跃状态. 当处理器处于睡眠状态时，DPM 技术可以关闭处理器时钟，将处理器电压降低到非常低的水平，以至于处理器功耗可以忽略不计，近似为 0[10,13]. 执行任务的处理器则处于活跃状态. 由于睡眠状态中处理器上下文等信息不被保存，所以返回活跃状态需要一定的时间和能量开销①. 本文将 E_{sw} 和 t_{sw} 分别定义为处理器在睡眠状态和活跃状态之间的切换能量和时间开销.

当处理器在活跃状态中空闲时，最小能耗处理器将以极低速度执行空操作指令. 设空闲时功耗为 P_{idle}，且不小于处理器在最低速度时的功耗 $P(s_{\min})$，即 $P_{\mathrm{idle}}\geq P(s_{\min})$. 而 break-even 时间(即 t_θ)是基于空闲时功耗所计算的最小空闲周期. 当处理器的空闲时间大于 break-even 时间时，将处理器转入睡眠状态

① Advanced Configuration and Power Interface, 2011. http://www.acpi.info

是节能的. break-even 时间长度依赖于空闲时功耗和切换开销,所以不妨设 $t_\theta = E_{sw}/P_{\text{idle}}$[10],通常假设时间开销 t_{sw} 不大于 t_θ[10].

2.2 任务模型

本文考虑流媒体应用中常见的基于帧任务集 $T=\{\tau_1,\tau_2,\cdots,\tau_n\}$. 所有基于帧的实时任务具有相同的到达时间(又称释放时间)和周期,帧大小为周期 D. 每个周期任务具有无穷序列的任务实例,又称为作业. 任务 τ_j 的每个作业都按照相同周期到达(或称释放到)就绪队列. τ_j 的截止期等于周期,其在最高处理器速度 $s_{\max}$ 下的最坏情况执行时钟数或执行时间 C_j 是预先已知的. τ_j 的利用率为 $u_j = C_j/D$,任务集的总利用率为 $U=\sum_{\tau_j\in T} u_j$. 不失一般性,本文只考虑 $C_j \le D$ 这种情况,在该情况下任务 τ_j 可以以速度 $s_{\max}$ 在任务截止期 D 之前完成.

假设在时间间隔中执行的 CPU 时钟数与处理器速度成线性正比关系. 任务 τ_j 在 t 时间单元内以速度 s 运行所需要的 CPU 时钟数可以近似表示为 s 和 t 的乘积.

2.3 能耗模型

当处理器在活跃状态与睡眠状态之间的切换能耗开销 E_{sw} 不为 0 时,将任务集 T 中所有任务分配到 m 个处理器上的最优节能实时调度一定满足下列两个性质.

引理 1. 第 k 个处理器上所分配的任务集 T_k 中每个任务 τ_i 都具有相同的处理器速度.

证明. 如果定义处理器速度分配列表 σ 为处理器速度 s_j 和对应的执行时间长度 t_j 所构成数对 (s_j,t_j) 的有序集合,那么根据功耗函数 $P(s)$ 是凸函数性质可得如下不等式[4]:

$$\sum_{(s_j,t_j)\in\sigma}(P(s_j)\cdot t_j)\ge P\left(\frac{\sum_{(s_j,t_j)\in\sigma}(s_j\cdot t_j)}{\sum_{(s_j,t_j)\in\sigma} t_j}\right)\cdot\sum_{(s_j,t_j)\in\sigma} t_j \tag{1}$$

命题得证. 证毕.

引理 2. 第 k 个处理器上所分配的任务集 T_k 具有最低能耗的节能实时调度一定满足:当 $U=\sum_{\tau_i\in T_k}\frac{C_i}{D}$ 且 $U\ge s^*$ 时,以 U 为速度值来执行 T_k 中所有任务;当 $U<s^*$ 时,最优执行速度值只能从关键速度 s^* 或 U 中选择.

证明. 当 $U=\sum_{\tau_i\in T_k}\frac{C_i}{D}$ 且 $U\ge s^*$ 时,根据功耗函数的凸函数性质直接可知 U 是最优执行速度值.

当 $U<s^*$ 时,不妨假设以 U 为速度值来执行任务集 T_k 中所有任务时的能耗为 $E(U)=P(U)\cdot D$,而处理器以 s^* 为速度值执行时的空闲时间为 $t^*_{\text{idle}}=D-UD/s^*$. 下面采用反证法,根据引理 1 假设此时任务集 T_k 中最优节能调度的执行速度值为 s,且满足 $s\ne s^*$,$s\ne U$.

(1) 如果 $s<U$,则所有任务的执行时间为 $UD/s>D$,显然此时不是最优节能实时调度.

(2) 如果 $s>s^*$,则所有任务的执行时间为 $\frac{UD}{s}<\frac{UD}{s^*}<D$,存在空闲时间 $t_{\text{idle}}=D-\frac{UD}{s}>t^*_{\text{idle}}$. 当 $t_{\text{idle}}\le t_\theta$ 时,以 s 为速度的处理器能耗为 $E(s)=P(s)\cdot\frac{UD}{s}+P_{\text{idle}}\cdot t_{\text{idle}}$,而以 s^* 为速度来执行时的能耗为 $E(s^*)=P(s^*)\cdot\frac{UD}{s^*}+P_{\text{idle}}\cdot t^*_{\text{idle}}$. 因为 $P(s)/s$ 函数在 $s>s^*$ 范围中属于单调递增函数,所以 $P(s)\cdot\frac{UD}{s}>P(s^*)\cdot\frac{UD}{s^*}$,进而 $E(s)>E(s^*)$. 当 $t_{\text{idle}}>t_\theta$ 时,以 s 为速度的处理器能耗为 $E(s)=P(s)\cdot\frac{UD}{s}+E_{sw}$. 此时如果 $t^*_{\text{idle}}\le t_\theta$,则以 s^* 为速度来执行时的能耗为 $E(s^*)=P(s^*)\cdot\frac{UD}{s^*}+P_{\text{idle}}\cdot t^*_{\text{idle}}\le P(s^*)\cdot\frac{UD}{s^*}+E_{sw}$;如果 $t_\theta<t^*_{\text{idle}}<t_{\text{idle}}$,则能耗 $E(s^*)=P(s^*)\cdot\frac{UD}{s^*}+E_{sw}$. 同理,根据 $P(s)/s$ 函数在 $s>s^*$ 范围中属于单调递增函数,可得 $E(s)>E(s^*)$.

(3) 如果 $U<s<s^*$,则所有任务的执行时间为 $\frac{UD}{s}\left(\frac{UD}{s^*}<\frac{UD}{s}<D\right)$,存在空闲时间 $t_{\text{idle}}=D-UD/s$,$t_{\text{idle}}<t^*_{\text{idle}}$. 当 $t_{\text{idle}}>t_\theta$ 时,$t^*_{\text{idle}}>t_\theta$,以 s 为速度的处理器能耗为 $E(s)=P(s)\cdot UD/s+E_{sw}$,而以 s^* 为速度执行时的能耗为 $E(s^*)=P(s^*)\cdot UD/s^*+E_{sw}$. 因为 $P(s)/s$ 函数在 $s<s^*$ 范围中属于单调递减函数,所以 $E(s)>E(s^*)$. 当 $t_{\text{idle}}\le t_\theta$ 时,以 s 为速度的处理器能耗为 $E(s)=P(s)\cdot UD/s+P_{\text{idle}}\cdot(D-UD/s)$. 因为 $P_{\text{idle}}\ge P(s_{\min})\ge P(0)$,且功耗函数 $P(s)$ 属于凸函数性质,所以 $E(s)\ge P(s)\cdot UD/s+P(0)\cdot(D-UD/s)>P(s\cdot U/s+0\cdot(1-U/s))\cdot D=E(U)$.

综上可知,与假设相矛盾. 命题得证. 证毕.

由引理 2 易得下面推论.

推论 1. 当 $U=\sum_{\tau_i\in T_k}\frac{C_i}{D}$ 且 $U<s^*$ 时,第 k 个处

理器以关键速度执行完成任务集 T_k 中所有任务，且空闲时执行空操作，此时能耗值一定不小于以 U 值为执行速度且没有空闲时间时的能耗值.

证明.　如果 $U=\sum_{\tau_i\in T_k}\frac{C_i}{D}$ 且 $U<s^*$，那么处理器以关键速度执行，而空闲时执行空操作，即不会将处理器转入睡眠状态. 此时能耗为 $P(s^*)\cdot UD/s^*+P_{\text{idle}}\cdot(D-UD/s^*)$. 而处理器以 U 值为执行速度且没有空闲时间时的能耗值为 $P(U)D$. 因为 $P(s)$ 是凸函数，且 $P_{\text{idle}}\geq P(0)$，所以 $U/s^*\cdot P(s^*)+(1-U/s^*)\cdot P_{\text{idle}}\geq U/s^*\cdot P(s^*)+(1-U/s^*)\cdot P(0)\geq P(U/s^*\cdot s^*+(1-U/s^*)\cdot 0)=P(U)$，则 $(P(s^*)-P_{\text{idle}})\cdot UD/s^*+P_{\text{idle}}\cdot D\geq P(U)\cdot D$.　证毕.

不妨令 $\phi(T_k)$ 为第 k 个处理器上在公共截止期 D 内完成 T_k 中所有任务的最小能耗. 因为切换开销 E_{sw} 不为 0，所以如果某个处理器为空闲时，最优调度不一定会将处理器转入睡眠状态. 根据引理 2，如果 $\sum_{\tau_i\in T_k}\frac{C_i}{D}\geq s^*$，那么 $\phi(T_k)=P\left(\sum_{\tau_i\in T_k}\frac{C_i}{D}\right)\cdot D$；否则，继续分情况进行判断，如果处理器在截止期 D 之前完全运行时的能耗小于以关键速度运行时的能耗，那么 $\phi(T_k)=P\left(\sum_{\tau_i\in T_k}\frac{C_i}{D}\right)\cdot D$；如果处理器以关键速度运行时的能耗小于完全运行时的能耗，那么根据推论 1 可知，处理器必然会转入睡眠状态，$\phi(T_k)=P(s^*)\cdot\left(\sum_{\tau_i\in T_k}\frac{C_i}{D}\right)\cdot\frac{D}{s^*}+E_{sw}$. 综上，当负载 $U=\sum_{\tau_i\in T_k}\frac{C_i}{D}$ 时，第 k 个处理器在公共截止期 D 内完成 T_k 中所有任务的能耗为 $E_k(U)$，即

$$E_k(U)\leftarrow\begin{cases}P(U)\cdot D, & U\geq s^*\\ \min\left\{P(U)\cdot D, P(s^*)\dfrac{U\cdot D}{s^*}+E_{sw}\right\}, & \text{其它}\end{cases}.$$

图 1 给出 $E_k(U)$ 函数的示例. 显然该函数为分段函数，不属于凸函数，但是其仍部分满足凸函数性质，所以针对该函数的分析需要分情况考虑.

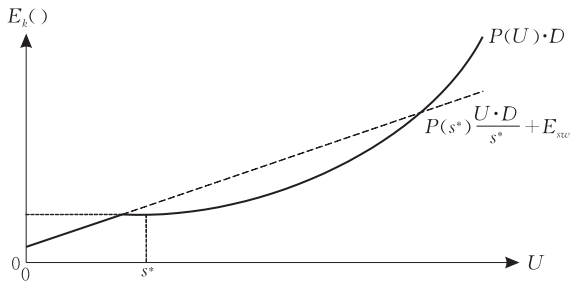


图 1　$E_k(U)$ 函数示例

2.4　问题定义

任务集的节能实时调度是指集合中所有任务的执行映射到系统中的处理器上，并且为任务的对应时间间隔分配处理器速度. 如果在时间间隔内分配的所有处理器速度都是有效的，且没有任务丢失截止期 D，那么这样一个节能实时调度是可行的. 一次调度 Sch 的能耗表示为 $\sum_{k=1}^{m}E_k(Sch)$（正如 2.3 小节中能耗的定义）. 如果一个调度是可行的，且能耗是所有可行的调度中最小能耗，那么它就是最优节能实时调度.

本文所考虑的问题定义如下：

给定基于帧的实时任务集 T，可以在 m 个同构多处理器上运行. 每个处理器具有相同的功耗函数 $P(s)=\alpha\cdot s^3+\beta,\alpha\geq 0,\beta\geq 0$. 假设每个处理器的执行速度可以在 $s_{\min}=0$ 和 $s_{\max}$ 之间连续调节. 所有任务具有相同的释放时间和截止期 D. 假设实时任务集 T 是可调度的，每个任务 τ_j 的最坏情况执行时间 C_j 都不超过其截止期 D，任务 τ_j 的实际执行按照最坏情况执行. 处理器在睡眠状态和活跃状态之间的切换能量开销为 E_{sw}，时间开销为 $t_{sw}\leq E_{sw}/P_{\text{idle}}$. 本文的目标是在允许任务迁移条件下，将 T 中所有任务分配到 m 个处理器上执行，得到最优的节能实时调度序列.

由于按照最坏情况执行的每个任务在运行时不会改变其速度，所以处理器在活跃状态时没有速度切换开销. 又因为任务迁移的次数不会超过处理器个数，所以开销敏感的多处理器最优节能实时调度问题的迁移代价可以忽略不计.

3　LTF-M 最优算法的非最优性限制

本节以 Intel Xscale 处理器的功耗函数为例，对最优节能实时调度算法 LTF-M 进行实际分析. 这里功耗函数被近似建模为 $P(s)=1.52\cdot s^3+0.08\,\text{W}$[10-12]. 不妨假设 $s_{\min}$ 为 0，$s_{\max}$ 为 1 GHz. 根据 2.1 节可知，以 $s_{\max}$ 为标准进行归一化，关键速度 s^* 约为 0.297 GHz，在关键速度 s^* 上的功耗 $P(s^*)=0.12\,\text{W}$. 假设切换开销 E_{sw} 为 0.8 mJ，处理器的空闲功耗为 0.08 W，则 break-even 时间 t_θ 为 10 ms. 已知基于帧的实时任务集 $T=\{\tau_1,\tau_2,\tau_3,\tau_4\}$ 在双处理器系统中执行，所有任务具有相同周期 $D=30$ ms. 每个任务在最高速度 $s_{\max}$ 下的最坏情况执行时钟数分别为 $C_1=0.012s^*$，$C_2=0.012s^*$，$C_3=0.006s^*$，$C_4=$

$0.006s^*$. 如果以关键速度 s^* 来执行这些任务，每个任务的执行时间分别为 12 ms，12 ms，6 ms，6 ms.

当不考虑泄漏功耗所引起的静态能耗时，Chen 等人[4-5]已经证明 LTF-M 算法针对同构多处理器系统，在保证基于帧实时任务集可调度性的条件下产生最低的能耗值. 当静态能耗不可忽视时，为了平衡动态能耗和静态能耗，如果将关键速度看作任务执行的最低速度，LTF-M 算法可以直接扩展为 LTF-M-CRITICAL 算法. 但是，当考虑处理器的状态切换开销时，LTF-M 算法及其 LTF-M-CRITICAL 扩展算法不再具有最优性. 以任务集 T 为例，图 2 给出了示例说明. 这里，x 轴表示时间，y 轴表示处理器执行速度，每个任务框的面积定义为执行该任务所占用的 CPU 时钟数.

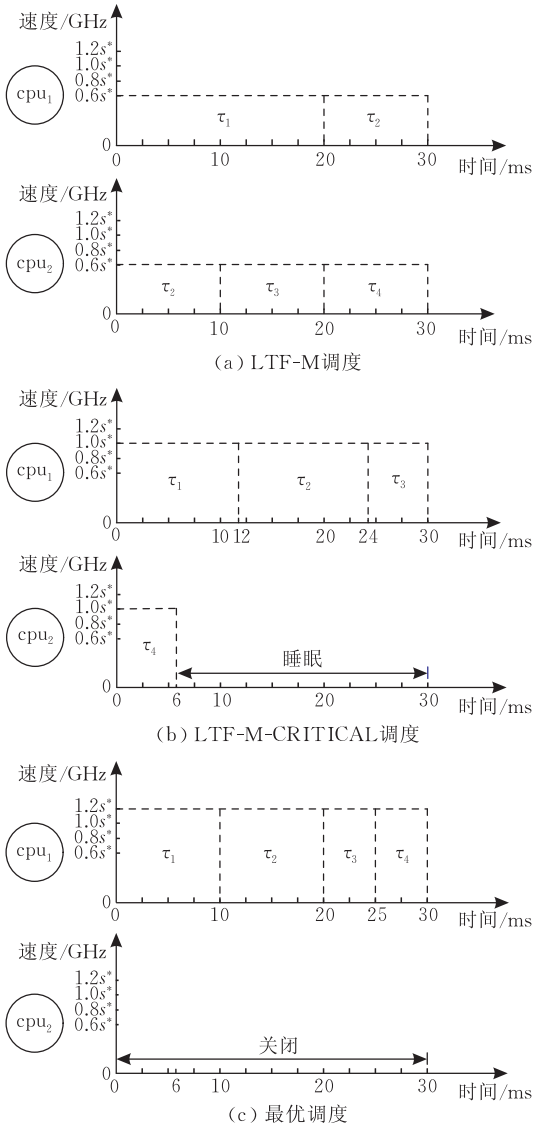


图 2 LTF-M 算法及其 LTF-M-CRITICAL 扩展算法的非最优性示例

图 2(a)表示任务集 T 在 LRE-TL 算法下的调度序列，可以保证 T 中所有任务在截止期 D 之前的可调度性. 已知每个任务在周期 $D=30$ ms 中的利用率分别为 $u_1=0.4s^*$，$u_2=0.4s^*$，$u_3=0.2s^*$，$u_4=0.2s^*$，平均利用率为 $\bar{u}=\sum_{\forall\tau_i\in T}u_i/2=0.6s^*$. 由于每个任务的利用率均小于平均利用率，所以 LTF-M 算法为每个任务分配执行速度 $\bar{u}$ GHz. 同时为保证所有任务的可调度性，指定任务 τ_2 从 20 ms 到 30 ms 时间段内在第 1 个处理器上执行，从 0 ms 到 10 ms 时间段内在第 2 个处理器上执行. LTF-M 算法能够使得所有处理器以完全利用整个时间段 $(0,30]$ ms 的速度 $\bar{u}$ GHz 执行任务，没有空闲时间存在，此时处理器总能耗 $E_2'=(0.04\cdot0.6^3+0.08)\cdot60=5.3184$ mJ.

LTF-M 算法为每个任务分配的执行速度 $\bar{u}$ 均小于关键速度，因此 LTF-M-CRITICAL 算法将以关键速度 s^* 来执行所有任务，总的执行时间为 $\sum_{\forall\tau_i\in T}C_i/s^*=36$ ms. 图 2(b)显示了 LTF-M- CRITICAL 算法的调度序列. 从图 2(b)中可知，第 2 个处理器将从 6 ms 开始进入睡眠状态，直到 30 ms 时刻. 由于睡眠时间超过 break-even 时间，切换能耗开销小于空闲能耗，所以第 2 个处理器在时间段 $(6,30]$ ms 内的能耗为 E_{sw}. 此时处理器总能耗 $E_2^*=0.12\cdot36+0.8=5.12$ mJ.

由以上分析可知，在考虑静态能耗条件下，LRE-TL 算法不再保证其最优性，而其扩展算法 LTF-M-CRITICAL 有可能进一步降低能耗. 在关键速度上执行可能会比在更低的速度级别上执行更节能的主要原因在于，它乐观地假设在任务执行之后将处理器转入睡眠状态. 同时这两种算法均假设系统负载在所有处理器上均衡分布具有最优性. 但是如果离线关闭某些处理器以减少活跃处理器的个数，可能会极大减少能耗. 例如，假设只有一个处理器处于活跃状态，而另一个处理器处于关闭状态，则可以得到一个最优节能实时调度序列，即所有任务以 $\sum_{\forall\tau_i\in T}C_i/D=1.2s^*$ GHz 速度在一个处理器上执行，没有空闲时间，如图 2(c)所示. 此时能耗 $E_1=(0.04\cdot1.2^3+0.08)\cdot30=4.4736$ mJ.

综上可述，LTF-M 算法及其扩展算法在考虑开销的实际环境中不是最优的，主要原因是在开销敏感的低负载情况下，基于关键速度以及负载均衡的节能调度不具有最优性[10].

4 LUF-SO 算法

由于开销问题越来越不容忽视，基于对 LTF-M 最优算法的非最优性分析，本文得到在实际环境中最优算法应该具备的三个条件：(1) 系统的负载情况的确定；(2) 具有最低能耗值的活跃处理器个数；(3) 开销敏感的处理器状态切换. 本文将基于以上分析，针对基于帧的实时任务模型，提出考虑实际切换开销情况下的多处理器最优节能实时调度算法 LUF-SO，如算法 1 所示.

算法 1. LUF-SO 算法.

输入：T, D, m, s^*

输出：具有最低能耗值的最优调度序列

1. 按照任务利用率 u_i 的非递增顺序排列所有任务 $\tau_i \in T$；
2. $U \leftarrow \sum_{\tau_i \in T} u_i$ and $m^* \leftarrow -1$；
3. IF $U > m$ or $\exists \tau_i \in T$ such that $u_i > 1$ THEN
4. return 不存在可行的调度；
5. $i \leftarrow 1$ and $M \leftarrow m$；
6. WHILE $i \leq |T|$ DO
7. IF $u_i < s^*$ and $U/M < s^*$ THEN
8. $m^* \leftarrow \lfloor U/s^* \rfloor$；
9. BREAK；
10. ELSE IF $u_i > U/M$ THEN
11. $\tau_i.speed \leftarrow u_i$；
12. $U \leftarrow U - u_i$，$i \leftarrow i+1$，and $M \leftarrow M-1$；
13. ELSE
14. $\tau_i.speed \leftarrow U/M$ and $i \leftarrow i+1$；
15. IF $m^* \geq 0$ THEN
16. $m^* \leftarrow$ Check_Switching_Overhead(T, U, D, i, m^*, s^*)；
17. IF $M > m^*$ THEN
18. $M \leftarrow m - (M - m^*)$；
19. ELSE
20. $M \leftarrow m$；
21. $i \leftarrow 1$ and $t \leftarrow 0$；
22. WHILE $i \leq |T|$ DO
23. IF $\tau_i.speed = u_i$ THEN
24. 调度任务 τ_i 以 $u_i \cdot s_{\max}$ 为速度值在 0 到 D 时间间隔内执行在第 M 个处理器上；
25. $M \leftarrow M-1$；
26. ELSE if $t + \frac{u_i \cdot D}{\tau_i.speed} > D$ THEN
27. 调度任务 τ_i 以 $\tau_i.speed \cdot s_{\max}$ 为速度值在 t 到 D 时间间隔内执行在第 M 个处理器上，在 0 到 $t + \frac{u_i \cdot D}{\tau_i.speed} - D$ 时间间隔内执行在第 $M-1$ 个处理器上；
28. $M \leftarrow M-1$；
29. ELSE
30. 调度任务 τ_i 以 $\tau_i.speed \cdot s_{\max}$ 为速度值在 t 到 $t + \frac{u_i \cdot D}{\tau_i.speed}$ 时间间隔内执行在第 M 个处理器上；
31. $i \leftarrow i+1$ and $t \leftarrow \left(t + \frac{u_i \cdot D}{\tau_i.speed}\right) \bmod D$；
32. return 所有任务的调度序列.

4.1 基本思想

本文提出的算法 LUF-SO 采用最大任务利用率优先策略，离线确定任务的调度过程和执行速度. 最大任务利用率优先策略是指所有任务按照利用率的非递增顺序来排列. 以任务集 T 为例，令 $U = \sum_{\tau_i \in T} u_i$. LUF-SO 算法的基本思想如下：

(1) 当 $u_i \geq s^*$ 时，如果 $u_i \leq U/M$，那么 LUF-SO 算法将剩余未分配的任务 $\forall \tau_j \in T \backslash \{\tau_1, \cdots, \tau_i\}$ 统一分配平均速度 $\tau_j.speed = U/M$（第 13 行到第 14 行）；否则，LUF-SO 算法将为任务 τ_i 分配速度 $\tau_i.speed = u_i$（第 10 行到第 12 行），且将可用处理器的个数减 1.

(2) 当 $u_i < s^*$ 时，如果 $U/M \geq s^*$，则易知 $u_i < U/M$，故 LUF-SO 算法将为剩余未分配速度的任务统一分配平均速度 U/M（第 13 行到第 14 行）；否则，这反映剩余未分配速度的任务的执行速度可能会小于关键速度，需要计算最优调度可能具有的处理器数 $m^* = \lfloor U/s^* \rfloor$（第 7 行到第 9 行）. 同时调用 Check_ Switching_Overhead 算法（第 15 行到第 16 行），不仅可以获得在考虑切换开销情况下最优的处理器数 m^*，而且将为所有剩余未分配速度的任务指定了对应的执行速度. 如果调用之后得到的 m^* 小于可用的处理器个数 M，则说明某些处理器可以被直接关闭，从而减少可用处理器个数（第 17 行到第 18 行），否则所有处理器全部打开，可用处理器个数 M 等于处理器个数 m（第 19 行到第 20 行）.

(3) 由于任务集 T 中所有任务都已分配了相关的执行速度，所以下面可以直接按照任务利用率非递增顺序将所有任务指派到对应的处理器上调度执行：如果某个任务 τ_i 的速度 $\tau_i.speed$ 等于其利用率 u_i，那么该任务将单独在一个处理器上从 0 到 D 时间间隔内调度执行，同时可用处理器个数减 1（第 23 行到第 25 行）；否则，因为 $u_i < \tau_i.speed$，所以 $t + \frac{u_i \cdot D}{\tau_i.speed} - D < t$. 如果任务 τ_i 被调度执行在两个处理器上（第 26 行到第 28 行），τ_i 将以速度 $\tau_i.speed \cdot s_{\max}$ 在 t 到 D 时间间隔内执行在第 M 个处理器上，在 0

到 $t+\frac{u_i \cdot D}{\tau_i.speed}-D$ 时间间隔内执行在第 $M-1$ 个处理器上；此外，如果任务 τ_i 被调度执行在一个处理器上(第 29 行到第 30 行)，τ_i 将以速度 $\tau_i.speed \cdot s_{\max}$，从 t 到 $t+\frac{u_i \cdot D}{\tau_i.speed}$ 时间间隔内执行在第 M 个处理器上.

为得到具有最低能耗值的解，Check_Switching_Overhead 算法针对所有未分配速度的后续任务子集 $T'=T\backslash\{\tau_1,\cdots,\tau_i\}$ 只分析下列 3 种情况，如算法 2 所示.

算法 2. Check_Switching_Overhead.

输入：T,U,D,i,m^*,s^*

输出：能够满足所有未分配速度任务 $T'=T\backslash\{\tau_1,\cdots,\tau_i\}$ 的截止期且具有最低能耗值的最优处理器数 m^*

1. $M'\leftarrow m^*+1$, $U'\leftarrow U$, $j\leftarrow i$, $E^l_{m^*+1}\leftarrow 0$, $E^*_{m^*+1}\leftarrow 0$, $E_{m^*}\leftarrow\infty$, $E_{\min}\leftarrow 0$;
2. WHILE $j\leq|T|$ DO
3. IF $u_j>U'/M'$ THEN
4. $\tau_j.speed\leftarrow u_j$ and $E^l_{m^*+1}\leftarrow E^l_{m^*+1}+P(u_j)\cdot D$;
5. $U'\leftarrow U'-u_j$, $j\leftarrow j+1$, and $M'\leftarrow M'-1$;
6. ELSE
7. $\tau_j.speed\leftarrow U'/M'$ and $j\leftarrow j+1$;
8. $E^l_{m^*+1}\leftarrow E^l_{m^*+1}+P\left(\frac{U'}{M'}\right)\cdot M'\cdot D$;
9. $t_{\text{idle}}\leftarrow(m^*+1)\cdot D-U\cdot D/s^*$;
10. IF $t_{\text{idle}}>t_\theta$ THEN
11. $E^*_{m^*+1}\leftarrow P(s^*)\cdot\frac{U\cdot D}{s^*}+P_{\text{idle}}\cdot t_\theta$;
12. ELSE
13. $E^*_{m^*+1}\leftarrow P(s^*)\cdot\frac{U\cdot D}{s^*}+P_{\text{idle}}\cdot t_{\text{idle}}$;
14. IF $m^*\neq 0$ THEN
15. $E^l_{m^*}\leftarrow P\left(\frac{U}{m^*}\right)\cdot D\cdot m^*$;
16. $E_{\min}\leftarrow\min\{E^l_{m^*+1},E^*_{m^*+1},E^l_{m^*}\}$;
17. IF $E_{\min}=E^l_{m^*}$ THEN
18. $\forall j\in[i,|T|]$, $\tau_j.speed\leftarrow U/m^*$;
19. IF $E_{\min}=E^*_{m^*+1}$ THEN
20. $\forall j\in[i,|T|]$, $\tau_j.speed\leftarrow s^*$, and $m^*\leftarrow m^*+1$;
21. IF $E_{\min}=E^l_{m^*+1}$ THEN
22. $m^*\leftarrow m^*+1$;
23. return 处理器个数 m^*;

情况 1. T' 中所有任务在 m^*+1 个处理器上以完全利用整个周期时间的速度运行，没有空闲时间存在. Check_Switching_Overhead 算法采取与 LTF 算法相同的方法分配所有任务的执行速度(第 2 行到第 7 行). 此时能耗值为 $E^l_{m^*+1}$(第 8 行).

情况 2. T' 中所有任务在 m^*+1 个处理器上以关键速度 s^* 运行，存在空闲时间 $t^*_{\text{idle}}=(m^*+1)\cdot D-U\cdot D/s^*$(第 9 行). Check_Switching_ Overhead 算法根据 t^*_{idle} 与 t_θ 大小关系判断是否进入睡眠状态，从而得到对应情况下不同的能耗值 $E^*_{m^*+1}$(第 10 行到第 13 行).

情况 3. T' 中所有任务在 m^* 个处理器上以完全利用整个周期时间的速度运行，没有空闲时间存在. 如果 $m^*\neq 0$，那么 Check_Switching_Overhead 算法可以得到在 m^* 个处理器上最低的能耗值 $E^l_{m^*}$(第 14 行到第 15 行).

基于上述 3 种情况的能耗值，Check_Switching_Overhead 算法确定最低能耗值 $\min\{E^l_{m^*+1},E^*_{m^*+1},E^l_{m^*}\}$，以此分配 T' 中所有任务的执行速度，得到最优处理器数 m^*(第 16 行到 23 行).

4.2 复杂度分析

设任务集 T 中任务总数为 n，处理器数为 m，每个任务状态数量为 w，每个处理器状态数量为 v.

时间复杂度：在算法 2 的第 2 步和第 7 步之间的循环以及算法 2 的第 18 步和第 20 步中，为 $T'\subseteq T$ 中所有未分配速度的任务指定执行速度的时间复杂度均为 $O(n)$. 由于算法 2 中计算各种情况下能耗值和最优处理器个数的时间复杂度均为常量，所以算法 2 的时间复杂度为 $O(n)$.

算法 1 中第 1 步针对任务集 T 的排序操作需要时间复杂度为 $O(n\cdot\log n)$. 从算法 1 的第 2 步到第 5 步，每步计算的时间复杂度为常量，记作 $O(1)$. 而在算法 1 的第 6 步和第 14 步之间的循环中，为每个任务分配执行速度的时间复杂度为 $O(n)$. 算法 1 中第 15 步到 18 步调用算法 2 的计算步骤的时间复杂度为 $O(n)$. 此外，在算法 1 的第 20 步和第 29 步之间的循环中，将所有任务分配到指定的处理器上调度执行的时间复杂度为 $O(n)$. 综上，算法 1 的时间复杂度为 $O(n\cdot\log n)+O(n)=O(n\cdot\log n)$.

空间复杂度：如果采用顺序表存储，任务集 T 所占用的空间为 $O(n\cdot w)$，处理器状态所占用的空间为 $O(m\cdot v)$.

4.3 最优性分析

当 $u_i\geq s^*$ 或者 $U/M\geq s^*$ 时，根据功耗函数的凸函数性质以及 LTF-M 算法的证明过程，显然可知 LUF-SO 算法得到的节能调度是最优的. 因此，为了证明 LUF-SO 算法在 $u_i<s^*$ 且 $U/M<s^*$ 的情况下也具有最优性，下面来证明算法 1 的 Check_ Switching_Overhead 可分析得到的 3 种情况，其能耗值也

是最低的.接下来,本文将根据活跃处理器个数的不同,分情况说明最优节能调度只能在 m^* 或 m^*+1 个处理器上产生.

4.3.1　m^* 个处理器($m^*>0$)

对于 $T'\subseteq T$ 中所有任务在 $m^*=\left\lfloor\frac{\sum_{\forall\tau_i\in T'}u_i}{s^*}\right\rfloor$ 个处理器上执行时,因为 T' 中所有任务的利用率 $u_i<s^*$,又因为 $U=\sum_{\forall\tau_i\in T'}u_i$,$U/s^*\geq m^*$,则 $U/m^*\geq s^*$,$U/m^*>u_i$.为了满足 T' 中所有任务的截止期,可以选择按照 U/m^* 为速度来执行,此时任务在 m^* 个处理器上运行没有产生空闲时间,能耗为

$$E^l_{m^*}\leftarrow P(U/m^*)\cdot D\cdot m^* \tag{2}$$

引理 3.　$T'\subseteq T$ 中所有任务按照任务利用率的非递增顺序排列,所有任务具有相同的截止期 D,如果令 $U=\sum_{\forall\tau_i\in T'}u_i$,那么 T' 中所有任务在 $m^*=\left\lfloor\frac{\sum_{\forall\tau_i\in T'}u_i}{s^*}\right\rfloor$ 个处理器上,以 U/m^* 为速度执行时所得到的能耗 $E^l_{m^*}$ 是最低的.

证明.　针对多处理器,由于具有最低能耗的速度分配方法必然保证单个处理器在运行时只有唯一的执行速度[4],所以不妨假设 m^* 个处理器 $\{P_1,P_2,\cdots,P_{m^*}\}$ 具有的执行速度列表为 $\{s_1,s_2,\cdots,s_{m^*}\}$,而每个处理器对应的运行时间列表为 $\{t_1,t_2,\cdots,t_{m^*}\}$.其中 $\forall s_j\in\{s_1,s_2,\cdots,s_{m^*}\}$,$0\leq s_j\leq s_{\max}$.因为 $\sum_{j=1}^{m^*}t_j\leq m^*\cdot D$,所以空闲时间 $t_{\text{idle}}=m^*\cdot D-\sum_{j=1}^{m^*}t_j$,且 $E_{sw}=P_{\text{idle}}\cdot t_\theta$.如果 $t_{\text{idle}}>t_\theta$,那么可以将处理器从活跃状态转入睡眠状态;否则,处理器仍处于空闲状态.能耗 E_{m^*} 表示如下:

$$E_{m^*}\leftarrow\begin{cases}\sum_{j=1}^{m^*}(P(s_j)\cdot t_j)+P_{\text{idle}}\cdot t_\theta, & t_{\text{idle}}>t_\theta\\ \sum_{j=1}^{m^*}(P(s_j)\cdot t_j)+P_{\text{idle}}\cdot t_{\text{idle}}, & t_{\text{idle}}\leq t_\theta\end{cases}.$$

因为 $\sum_{j=1}^{m^*}(s_j\cdot t_j)=U\cdot D$,根据功耗函数 $P(s)$ 的凸函数性质以及不等式(1)可知,

$$\sum_{j=1}^{m^*}(P(s_j)\cdot t_j)\geq P\left(\sum_{j=1}^{m^*}(s_j\cdot t_j)\Big/\sum_{j=1}^{m^*}t_j\right)\cdot\sum_{j=1}^{m^*}t_j=P\left(U\cdot D\Big/\sum_{j=1}^{m^*}t_j\right)\cdot\sum_{j=1}^{m^*}t_j=\frac{P\left(U\cdot D\Big/\sum_{j=1}^{m^*}t_j\right)}{U\cdot D\Big/\sum_{j=1}^{m^*}t_j}\cdot U\cdot D.$$

因为 $\sum_{j=1}^{m^*}t_j\leq m^*\cdot D$,所以 $UD\Big/\sum_{j=1}^{m^*}t_j\geq\frac{U}{m^*}$.又因为 $\frac{U}{m^*}\geq s^*$,所以 $UD\Big/\sum_{j=1}^{m^*}t_j\in\left[\frac{U}{m^*},s_{\max}\right]$.根据 $\frac{P(s)}{s}$ 的凸函数性质可知,当 $s\geq s^*$ 时,$P(s)/s$ 是 s 的严格单调递增函数.当 $\sum_{j=1}^{m^*}t_j=m^*\cdot D$ 时,$P\left(UD\Big/\sum_{j=1}^{m^*}t_j\right)\cdot\sum_{j=1}^{m^*}t_j$ 可以取最小值 $P\left(\frac{U}{m^*}\right)\cdot m^*\cdot D$.因此,对于 $\sum_{j=1}^{m^*}(s_j\cdot t_j)=U\cdot D$,$\sum_{j=1}^{m^*}(P(s_j)\cdot t_j)\geq P\left(\frac{U}{m^*}\right)\cdot m^*\cdot D$.显然,可得 $E_{m^*}>E^l_{m^*}$.

因为以 U/m^* 为速度在 m^* 个处理器上运行没有产生空闲时间,所以如果所有处理器以低于 U/m^* 的速度执行时,必然会导致某个任务丢失截止期.因此 U/m^* 是所有 m^* 个处理器完全执行时唯一速度.　　证毕.

定理 1.　$T'\subseteq T$ 中所有任务按照任务利用率的非递增顺序排列,所有任务具有相同的截止期 D,不妨令 $m^*=\left\lfloor\frac{\sum_{\forall\tau_i\in T'}u_i}{s^*}\right\rfloor$,$U=\sum_{\forall\tau_i\in T'}u_i$,$\lambda\in Z^+$,$1\leq\lambda\leq m^*-1$.如果 T' 中所有任务在 $m^*-\lambda$ 个处理器上可调度执行,那么在 $m^*-\lambda$ 个处理器上执行所得到的最低能耗 $\min\{E_{m^*-\lambda}\}$ 均不小于在 m^* 个处理器均以 U/m^* 为速度来执行所得到的能耗 $E^l_{m^*}$.

证明.　由题设可知 $U/s^*\geq m^*$,则 $U/m^*\geq s^*$.因为 $1\leq\lambda\leq m^*-1$,所以 $U/(m^*-\lambda)>U/m^*\geq s^*$.

(1) 如果 $m^*-\lceil U\rceil<\lambda\leq m^*-1$,则 $1\leq m^*-\lambda<\lceil U\rceil$,进而 $U>m^*-\lambda$,此时任务负载大于处理器处理能力,不存在满足可调度性的任务调度.

(2) 如果 $\forall\lambda\in Z^+$,$1\leq\lambda\leq m^*-\lceil U\rceil$,则根据引理 3 可知 T' 中所有任务在 $m^*-\lambda$ 个处理器均以 $U/(m^*-\lambda)$ 为速度在截止期 D 之前执行,没有空闲时间,此时所得到的能耗是最低的,即 $E_{m^*-\lambda}=E^l_{m^*-\lambda}\leftarrow P\left(\frac{U}{m^*-\lambda}\right)\cdot D\cdot(m^*-\lambda)$,等价于 $E^l_{m^*-\lambda}\leftarrow\frac{P(U/(m^*-\lambda))\cdot U\cdot D}{U/(m^*-\lambda)}$.而 $E^l_{m^*}\leftarrow\frac{P(U/m^*)\cdot U\cdot D}{U/m^*}$.又根据 $P(s)/s$ 的凸函数性质可知,当 $s\geq s^*$ 时,

$P(s)/s$ 是执行速度 s 的严格单调递增函数. 因此，$E^l_{m^*-\lambda} \geq E^l_{m^*}$. 证毕.

4.3.2 m^*+1 个处理器($m^* \geq 0$)

假设 $T' \subseteq T$ 中所有任务根据 LTF-M 算法为 m^*+1 个处理器所分配的执行速度为 $\{s^l_1, s^l_2, \cdots, s^l_{m^*+1}\}$，由于此时所有 m^*+1 个处理器以完全利用整个周期 D 时间的速度运行，不存在空闲时间，所以能耗为

$$E^l_{m^*+1} \leftarrow \sum_{j=1}^{m^*+1} p(s^l_j) \cdot D \tag{3}$$

假设 $\{s^*, s^*, \cdots, s^*\}$ 为 $T' \subseteq T$ 中所有任务按照关键速度 s^* 调度执行时 m^*+1 个处理器的执行速度，存在空闲时间为 $t^*_{\text{idle}} \leftarrow (m^*+1) \cdot D - U \cdot D/s^*$，且 $E_{sw} = P_{\text{idle}} \cdot t_\theta$. 如果 $t^*_{\text{idle}} > t_\theta$，那么可以将处理器从活跃状态转入睡眠状态；否则处理器仍处于空闲状态. 因为 T' 中所有任务的利用率 $u_i < s^*$ 且 $U/s^* \geq m^*$，所以可以将 m^*+1 个处理器上可能出现的空闲时间集中到一个处理器上，使得状态转换次数减少到最少，只有一次. 因此能耗形式化如下：

$$E^*_{m^*+1} \leftarrow \begin{cases} P(s^*) \cdot U \cdot D/s^* + P_{\text{idle}} \cdot t_\theta, & t^*_{\text{idle}} > t_\theta \\ P(s^*) \cdot U \cdot D/s^* + P_{\text{idle}} \cdot t_{\text{idle}}, & t^*_{\text{idle}} \leq t_\theta \end{cases} \tag{4}$$

考虑一般情况，针对 m^*+1 个处理器，由于具有最低能耗的速度分配方法必然保证单个处理器的执行速度是唯一的，所以不妨假设 m^*+1 个处理器 $\{P_1, P_2, \cdots, P_{m^*+1}\}$ 具有的执行速度列表为 $\{s_1, s_2, \cdots, s_{m^*+1}\}$，而每个处理器对应的运行时间列表为 $\{t_1, t_2, \cdots, t_{m^*+1}\}$. 其中 $\forall s_j \in \{s_1, s_2, \cdots, s_{m^*+1}\}$，$0 \leq s_j \leq s_{\max}$，$\{s_1, s_2, \cdots, s_{m^*+1}\} \neq \{s^l_1, s^l_2, \cdots, s^l_{m^*+1}\}$ 且 $\{s_1, s_2, \cdots, s_{m^*+1}\} \neq \{s^*, s^*, \cdots, s^*\}$，存在空闲时间为 $t_{\text{idle}} \leftarrow (m^*+1) \cdot D - \sum_{j=1}^{m^*+1} t_j$，且 $E_{sw} = P_{\text{idle}} \cdot t_\theta$. 如果 $t_{\text{idle}} > t_\theta$，那么可以将处理器从活跃状态转入睡眠状态；否则处理器仍处于空闲状态. 能耗形式化如下：

$$E_{m^*+1} \leftarrow \begin{cases} \sum_{j=1}^{m^*+1} (P(s_j) \cdot t_j) + P_{\text{idle}} \cdot t_\theta, & t_{\text{idle}} > t_\theta \\ \sum_{j=1}^{m^*+1} (P(s_j) \cdot t_j) + P_{\text{idle}} \cdot t_{\text{idle}}, & t_{\text{idle}} \leq t_\theta \end{cases} \tag{5}$$

接下来，本节需要证明当 $T' \subseteq T$ 中所有任务在 m^*+1 个处理器上执行时，在满足所有任务截止期条件下，最低能耗值为 $\min\{E^l_{m^*+1}, E^*_{m^*+1}\}$. 注意到得到 $E^l_{m^*+1}$ 的方法会使得处理器上没有空闲时间存在，而得到 $E^*_{m^*+1}$ 的方法使得处理器上出现空闲时间.

引理 4. $T' \subseteq T$ 中所有任务按照任务利用率的非递增顺序排列，所有任务具有相同的截止期 D，假设在 m^*+1 个处理器上执行时具有的执行速度列表为 $\{s_1, s_2, \cdots, s_{m^*+1}\}$，对应的运行时间列表为 $\{t_1, t_2, \cdots, t_{m^*+1}\}$. 如果任意第 j 个处理器的速度 $s^* < s_j \leq s_{\max}$，或者对于任意第 j 个处理器的速度 $0 < s_j < s^l_j$，那么这两种情况中所有处理器执行 T' 中所有任务的能耗不可能是在保证可调度性条件下的最低能耗.

证明.

(1) 假设任意处理器执行速度列表 $\{s_1, s_2, \cdots, s_{m^*+1}\}$ 中第 j 个处理器的速度 $s^* < s_j \leq s_{\max}$.

如果在不改变第 j 个处理器上任务负载 C_j(即任务执行时钟数)的条件下，将 s_j 的速度降低为 s^*，而保持其它剩余处理器的执行速度不变，则新的所有执行速度列表为 $\{s_1, s_2, \cdots, s_{j-1}, s^*, s_{j+1}, \cdots, s_{m^*+1}\}$. 显然，由于改变前后能耗的唯一不同在于第 j 个处理器上的能耗，所以下面仅仅比较第 j 个处理器执行速度改变前后的能耗值.

已知 $s_j \cdot t_j = C_j = s^* \cdot t^*_j$，速度为 s_j 的能耗值为 $E_j \leftarrow P(s_j) \cdot t_j + P_{\text{idle}} \cdot (D - t_j) = P(s_j)/s_j \cdot C_j + P_{\text{idle}} \cdot (D - t_j)$；执行速度为 s^* 的能耗值为 $E^*_j \leftarrow P(s^*) \cdot t^*_j + P_{\text{idle}} \cdot (D - t^*_j) = P(s^*)/s^* \cdot C_j + P_{\text{idle}} \cdot (D - t^*_j)$. 因为 $s_j > s^*$，所以 $t_j < t^*_j$. 又因为所有任务的截止期均为 D，所以 $D - t_j > D - t^*_j$. 也就是说，如果执行速度改变后处理器的空闲时间大于 break-even 时间，可以由活跃状态进入睡眠状态，那么执行速度改变前也一定可以由活跃状态进入睡眠状态. 又根据功耗函数 $P(s)$ 的凸函数性质以及 $P(s)/s$ 在 $s \geq s^*$ 时属于递增函数性质，可知 $P(s_j)/s_j > P(s^*)/s^*$，进而可得 $E_j > E^*_j$. 因此可知对于存在处理器执行速度大于 s^* 的任意处理器执行速度列表，总可以将其转换为能耗值更低且处理器执行速度不超过 s^* 的另一个处理器执行速度列表.

(2) 假设任意处理器执行速度列表 $\{s_1, s_2, \cdots, s_{m^*+1}\}$ 中任意第 j 个处理器的速度 $0 < s_j < s^l_j$，$0 \leq t_j \leq D$.

因为 $\{s^l_1, s^l_2, \cdots, s^l_{m^*+1}\}$ 使得所有 m^*+1 个处理器以完全利用整个周期时间的速度运行，没有空闲时间存在，所以 $\sum_{j=1}^{m^*+1} s^l_j \cdot D = U \cdot D$. 因为 $\sum_{j=1}^{m^*+1} s_j < \sum_{j=1}^{m^*+1} s^l_j = U$，所以 $\sum_{j=1}^{m^*+1} s_j \cdot t_j \leq \sum_{j=1}^{m^*+1} s_j \cdot D < \sum_{j=1}^{m^*+1} s^l_j \cdot D = U \cdot D$. 显然处理器按照 $\{s_1, s_2, \cdots, s_{m^*+1}\}$ 的执行速度不能完成

T'中所有任务的负载，必然存在某些任务丢失截止期. 证毕.

定理 2. $T'\subseteq T$ 中所有任务按照任务利用率的非递增顺序排列，所有任务具有相同的截止期 D，假设在 m^*+1 个处理器上执行时具有的执行速度列表为$\{s_1,s_2,\cdots,s_{m^*+1}\}$，对应的运行时间列表为$\{t_1,t_2,\cdots,t_{m^*+1}\}$. 那么在满足 T'中所有任务截止期的条件下，所有任务在 m^*+1 个处理器上执行的最小能耗值为 $\min\{E^l_{m^*+1},\ E^*_{m^*+1}\}$.

证明. 由引理 4 可知在分析具有最小能耗值的节能调度时，可以排除某个处理器执行速度大于 s^* 的情况以及全部处理器执行速度均小于 s^l_j 的情况. 因此，这里仅仅考虑 $\forall s_j\in\{s_1,s_2,\cdots,s_{m^*+1}\}$，$s_j\le s^*$，$\{s_1,s_2,\cdots,s_{m^*+1}\}\neq\{s^l_1,s^l_2,\cdots,s^l_{m^*+1}\}$ 且 $\{s_1,s_2,\cdots,s_{m^*+1}\}\neq\{s^*,s^*,\cdots,s^*\}$. 下面根据 t^*_{idle} 与 t_θ的大小关系分两种情况讨论：

(1) 如果所有 m^*+1 个处理器以关键速度 s^* 调度执行时出现的空闲时间 $t^*_{\text{idle}}\le t_\theta$，则处理器不会由活跃状态转入睡眠状态.

当处理器执行速度列表为$\{s^l_1,s^l_2,\cdots,s^l_{m^*+1}\}$且处理器没有空闲时间时，根据式(3)可知 $E^l_{m^*+1}\leftarrow\sum_{j=1}^{m^*+1}(p(s^l_j)-P_{\text{idle}})\cdot D+P_{\text{idle}}\cdot(m^*+1)\cdot D$. 当所有 m^*+1 处理器的执行速度均为 s^* 时，因为 $t^*_{\text{idle}}=(m^*+1)\cdot D-U\cdot D/s^*\le t_\theta$，根据式(4)可知$E^*_{m^*+1}\leftarrow P(s^*)\cdot\dfrac{U\cdot D}{s^*}+P_{\text{idle}}\cdot\left((m^*+1)\cdot D-\dfrac{U\cdot D}{s^*}\right)$，等价于$E^*_{m^*+1}\leftarrow(P(s^*)-P_{\text{idle}})\cdot\dfrac{U\cdot D}{s^*}+P_{\text{idle}}\cdot(m^*+1)\cdot D$. 当处理器执行速度列表为$\{s_1,s_2,\cdots,s_{m^*+1}\}$时，$\forall s_j\in\{s_1,s_2,\cdots,s_{m^*+1}\}$，$\{s_1,s_2,\cdots,s_{m^*+1}\}\neq\{s^*,s^*,\cdots,s^*\}$且 $s_j\le s^*$. 因为$\sum_{j=1}^{m^*+1}(s_j\cdot t_j)=U\cdot D$，所以$\sum_{j=1}^{m^*+1}(s_j\cdot t_j)=U\cdot D<\sum_{j=1}^{m^*+1}t_j\cdot s^*$，$U\cdot D/s^*<\sum_{j=1}^{m^*+1}t_j$，可知 $t_{\text{idle}}=(m^*+1)\cdot D-\sum_{j=1}^{m^*+1}t_j<(m^*+1)\cdot D-U\cdot D/s^*=t^*_{\text{idle}}$. 因此，由式(5)可得$E_{m^*+1}\leftarrow\sum_{j=1}^{m^*+1}(P(s_j)\cdot t_j)+P_{\text{idle}}\cdot\left((m^*+1)\cdot D-\sum_{j=1}^{m^*+1}t_j\right)$，等价于 $E_{m^*+1}\leftarrow\sum_{j=1}^{m^*+1}((P(s_j)-P_{\text{idle}})\cdot t_j)+P_{\text{idle}}\cdot(m^*+1)\cdot D$.

功耗函数 $P(s)$是凸函数和严格单调递增函数，而 P_{idle}是常量，因此 $P(s)-P_{\text{idle}}$仍然属于凸函数和严格单调递增函数. 令 $G(s)=P(s)-P_{\text{idle}}$. 根据本文的处理器模型可知，$P(s)=P^{\text{dep}}(s)+P^{\text{ind}}$，而 P_{idle}为处理器中没有指令执行且时钟关闭时的功耗，且 $P_{\text{idle}}\ge P^{\text{ind}}$. 此时 $G(s)=P(s)-P_{\text{idle}}=P^{\text{dep}}(s)+P^{\text{ind}}-P_{\text{idle}}$.

因为$\sum_{j=1}^{m^*+1}(s_j\cdot t_j)=U\cdot D$，根据函数 $G(s)$的凸函数性质以及式(1)可知，

$$\sum_{j=1}^{m^*+1}(G(s_j)\cdot t_j)\ge G\left(\frac{\sum_{j=1}^{m^*+1}(s_j\cdot t_j)}{\sum_{j=1}^{m^*+1}t_j}\right)\sum_{j=1}^{m^*+1}t_j=\frac{G\left(UD\Big/\sum_{j=1}^{m^*+1}t_j\right)}{UD\Big/\sum_{j=1}^{m^*+1}t_j}UD \tag{6}$$

由于 $\forall t_j\in\{t_1,t_2,\cdots,t_{m^*+1}\}$，$0\le t_j\le D$，则$\sum_{j=1}^{m^*+1}t_j\le\sum_{j=1}^{m^*+1}D=(m^*+1)\cdot D$，$UD\Big/\sum_{j=1}^{m^*+1}t_j\ge U/(m^*+1)$. 又因为$\dfrac{U}{m^*+1}<s^*$，所以 $UD\Big/\sum_{j=1}^{m^*+1}t_j\in\left[\dfrac{U}{m^*+1},s_{\max}\right]$. 不妨假设 $f(s)=\dfrac{G(s)}{s}=\dfrac{P^{\text{dep}}(s)+P^{\text{ind}}-P_{\text{idle}}}{s}$，则一阶导数$f'(s)=\dfrac{(P^{\text{dep}}(s))'\cdot s+(P_{\text{idle}}-P^{\text{ind}})}{s^2}$. 此时，当 $P_{\text{idle}}=P^{\text{ind}}$ 时，$f'(0)=0$；当 $P_{\text{idle}}>P^{\text{ind}}$ 时，$f'(s)>0$. 由此可知，$f(s)$是速度 s 的严格单调递增函数. 当$UD\Big/\sum_{j=1}^{m^*+1}t_j=U/(m^*+1)$时，式(6)右侧可以取最小值，同时 $\forall t_j\in\{t_1,t_2,\cdots,t_{m^*+1}\}$，$t_j=D$. 再由引理 2 可知，$\{s^l_1,s^l_2,\cdots,s^l_{m^*+1}\}$是使得所有$m^*+1$ 个处理器以完全利用整个周期时间的速度运行，没有空闲时间存在的唯一有效执行速度列表，所以式(6)左侧$\sum_{j=1}^{m^*+1}(G(s_j)\cdot t_j)=\sum_{j=1}^{m^*+1}G(s_j)\cdot D=\sum_{j=1}^{m^*+1}G(s^l_j)\cdot D$.

显然，可得 $E_{m^*+1}>E^l_{m^*+1}$. 此时最小能耗值为 $\min\{E^l_{m^*+1},\ E^*_{m^*+1}\}$.

(2) 如果所有 m^*+1 个处理器以关键速度 s^* 调度执行时出现的空闲时间 $t^*_{\text{idle}}>t_\theta$，则可能存在某些处理器由活跃状态转入睡眠状态.

当处理器执行速度列表为$\{s^l_1,s^l_2,\cdots,s^l_{m^*+1}\}$且处理器没有空闲时间时，根据式(3)可知 $E^l_{m^*+1}\leftarrow$

$\sum_{j=1}^{m^*+1} p(s_j^l)\cdot D$. 当所有 m^*+1 处理器的执行速度均为 s^* 时，因为 $t_{\text{idle}}^* = (m^*+1)\cdot D - \frac{U\cdot D}{s^*} > t_\theta$，根据式(4)可知 $E_{m^*+1}^* \leftarrow P(s^*)\cdot\frac{U\cdot D}{s^*} + P_{\text{idle}}\cdot t_\theta$. 当处理器执行速度列表为 $\{s_1, s_2, \cdots, s_{m^*+1}\}$ 时，因为 $\{s_1, s_2, \cdots, s_{m^*+1}\} \neq \{s^*, s^*, \cdots, s^*\}$ 且 $\sum_{j=1}^{m^*+1}(s_j\cdot t_j) = U\cdot D$，所以 $\sum_{j=1}^{m^*+1}(s_j\cdot t_j) = U\cdot D < \sum_{j=1}^{m^*+1} t_j\cdot s^*$，$\frac{U\cdot D}{s^*} < \sum_{j=1}^{m^*+1} t_j$，可知 $t_{\text{idle}} = (m^*+1)\cdot D - \sum_{j=1}^{m^*+1} t_j < (m^*+1)\cdot D - \frac{U\cdot D}{s^*} = t_{\text{idle}}^*$. 因此，由式(5)可得

$$E_{m^*+1} \leftarrow \begin{cases} \sum_{j=1}^{m^*+1}(P(s_j)\cdot t_j) + P_{\text{idle}}\cdot t_\theta, & t_{\text{idle}} > t_\theta \\ \sum_{j=1}^{m^*+1}(P(s_j)\cdot t_j) + P_{\text{idle}}\cdot \left((m^*+1)\cdot D - \sum_{j=1}^{m^*+1} t_j\right), & t_{\text{idle}} \le t_\theta \end{cases}$$

下面根据 t_{idle} 与 t_θ 的关系，分两种情况来讨论：

(2.1) $t_{\text{idle}} > t_\theta$.

此时 $E_{m^*+1} \leftarrow \sum_{j=1}^{m^*+1}(P(s_j)\cdot t_j) + P_{\text{idle}}\cdot t_\theta$. 因为 $t_{\text{idle}} < t_{\text{idle}}^*$，所以 $E_{m^*+1}^* \leftarrow P(s^*)\cdot\frac{U\cdot D}{s^*} + P_{\text{idle}}\cdot t_\theta$.

又因为 $\sum_{j=1}^{m^*+1}(s_j\cdot t_j) = U\cdot D$，根据功耗函数 $P(s)$ 的凸函数性质以及式(1)可知，

$$\sum_{j=1}^{m^*+1}(P(s_j)t_j) \ge P\left(\sum_{j=1}^{m^*+1}(s_j\cdot t_j)\Big/\sum_{j=1}^{m^*+1} t_j\right)\sum_{j=1}^{m^*+1} t_j = \frac{P\left(UD\Big/\sum_{j=1}^{m^*+1} t_j\right)}{UD\Big/\sum_{j=1}^{m^*+1} t_j} UD.$$

由于 $\forall t_j \in \{t_1, t_2, \cdots, t_{m^*+1}\}$，$0 \le t_j \le D$，则 $\sum_{j=1}^{m^*+1} t_j \le \sum_{j=1}^{m^*+1} D = (m^*+1)\cdot D$，$UD\Big/\sum_{j=1}^{m^*+1} t_j \ge U/(m^*+1)$.

又因为 $\frac{U}{m^*+1} < s^*$，所以 $UD\Big/\sum_{j=1}^{m^*+1} t_j \in \left[\frac{U}{m^*+1}, s_{\max}\right]$. 根据 $\frac{P(s)}{s}$ 的凸函数性质可知，当 $s = s^*$ 时，$\frac{P(s^*)}{s^*} = \min\left\{\frac{P(s)}{s}\right\}$. 当 $\sum_{j=1}^{m^*+1} t_j = \frac{U\cdot D}{s^*}$ 时，$P\left(UD\Big/\sum_{j=1}^{m^*+1} t_j\right)\cdot\sum_{j=1}^{m^*+1} t_j$ 可以取最小值 $UD\cdot P(s^*)/s^*$. 因为 $\forall s_j \in \{s_1, s_2, \cdots, s_{m^*+1}\}$，$0 \le s_j \le s^*$ 以及 $\{s_1, s_2, \cdots, s_{m^*+1}\} \neq \{s^*, s^*, \cdots, s^*\}$，所以 $\sum_{j=1}^{m^*+1}(P(s_j)\cdot t_j) > UD\cdot P(s^*)/s^*$.

显然，可得 $E_{m^*+1} > E_{m^*+1}^*$. 此时最小能耗值为 $\min\{E_{m^*+1}^l, E_{m^*+1}^*\}$.

(2.2) $t_{\text{idle}} \le t_\theta$.

此时 $t_{\text{idle}} = (m^*+1)\cdot D - \sum_{j=1}^{m^*+1} t_j$，因此 $E_{m^*+1} \leftarrow \sum_{j=1}^{m^*+1}(P(s_j)\cdot t_j) + P_{\text{idle}}\cdot\left((m^*+1)\cdot D - \sum_{j=1}^{m^*+1} t_j\right)$，等价于 $E_{m^*+1} \leftarrow \sum_{j=1}^{m^*+1}((P(s_j) - P_{\text{idle}})\cdot t_j) + P_{\text{idle}}\cdot(m^*+1)\cdot D$.

由于 $t_{\text{idle}}^* = (m^*+1)\cdot D - U\cdot D/s^*$，$t_{\text{idle}}^* > t_\theta$，$t_{\text{idle}} < t_{\text{idle}}^*$，所以 $E_{m^*+1}^* \leftarrow P(s^*)\cdot UD/s^* + P_{\text{idle}}\, t_\theta < (P(s^*) - P_{\text{idle}})\cdot UD/s^* + P_{\text{idle}}(m^*+1)D$. 而 $E_{m^*+1}^l \leftarrow \sum_{j=1}^{m^*+1} p(s_j^l)\cdot D$ 等价于 $E_{m^*+1}^l \leftarrow \sum_{j=1}^{m^*+1}(p(s_j^l) - P_{\text{idle}})\cdot D + P_{\text{idle}}\cdot(m^*+1)\cdot D$.

同理，采用与(1)中类似的证明过程可得 $E_{m^*+1} > E_{m^*+1}^l$. 此时最小能耗值为 $\min\{E_{m^*+1}^l, E_{m^*+1}^*\}$.

综上命题得证. 证毕.

定理 3. $T' \subseteq T$ 中所有任务按照任务利用率的非递增顺序排列，所有任务具有相同的截止期 D. 不妨令 $m^* = \lfloor \sum_{\forall\tau_i\in T'} u_i/s^* \rfloor$，$U = \sum_{\forall\tau_i\in T'} u_i$，$\exists m \in Z^+$，$m > m^*$，$\forall\lambda \in Z^+$，$1 < \lambda \le m - m^*$. 如果 T' 中所有任务在 $m^*+\lambda$ 个处理器上可调度执行，那么在 $m^*+\lambda$ 个处理器上执行所得到的最小能耗均不小于在 m^*+1 个处理器所得到的最小能耗 $\min\{E_{m^*+1}^l, E_{m^*+1}^*\}$.

证明. 由题设可知 $U/s^* < m^*+1$，则 $\frac{U}{m^*+1} < s^*$. 因为 $1 \le \lambda \le m - m^*$，所以 $\frac{U}{m^*+\lambda} < \frac{U}{m^*+1} < s^*$.

对于 $\forall\lambda \in Z^+$，$1 < \lambda \le m - m^*$，根据定理 2 可知，T' 中所有任务在 $m^*+\lambda$ 个处理器上可调度执行所得的最小能耗为 $\min\{E_{m^*+\lambda}^l, E_{m^*+\lambda}^*\}$.

(1) 如果 $\min\{E_{m^*+\lambda}^l, E_{m^*+\lambda}^*\} = E_{m^*+\lambda}^l$，则 $m^*+\lambda$ 个处理器以完全利用整个周期时间的速度执行，没有空闲时间时的执行速度列表为 $\{s_{1,\lambda}^l, s_{2,\lambda}^l, \cdots, s_{m^*+1,\lambda}^l\}$. 根据式(3)可知 $E_{m^*+\lambda}^l \leftarrow \sum_{j=1}^{m^*+\lambda} P(s_{j,\lambda}^l)\cdot D$. 因

为$\sum_{j=1}^{m^*+\lambda} s_{j,\lambda}=U$，根据功耗函数 $P(s)$的凸函数性质以及式(1)可知，$\sum_{j=1}^{m^*+\lambda} P(s_{j,\lambda})\geq P\left(\sum_{j=1}^{m^*+1} s_{j,\lambda}/(m^*+\lambda)\right)\cdot(m^*+\lambda)=\frac{P(U/(m^*+\lambda))}{U/(m^*+\lambda)}\cdot U$. 又根据 $P(s)/s$ 的凸函数性质可知，当 $s<s^*$ 时，$P(s)/s$ 是 s 的严格单调递减函数，可得 $\frac{P(U/(m^*+\lambda))}{U/(m^*+\lambda)}>\frac{P(U/(m^*+1))}{U/(m^*+1)}$. 因为 $E^l_{m^*+\lambda}\geq P\left(\frac{U}{m^*+\lambda}\right)(m^*+\lambda)D>P\left(\frac{U}{m^*+1}\right)\cdot(m^*+1)D=E^l_{m^*+1}$，所以 $\min\{E^l_{m^*+\lambda},E^*_{m^*+\lambda}\}>E^l_{m^*+1}\geq\min\{E^l_{m^*+1},E^*_{m^*+1}\}$.

(2) 如果 $\min\{E^l_{m^*+\lambda},E^*_{m^*+\lambda}\}=E^*_{m^*+\lambda}$，则 T'中所有任务的执行速度均为 s^*，$m^*+\lambda$ 个处理器上存在空闲时间. 因为 $1<\lambda\leq m-m^*$，所以 $t^*_{\mathrm{idle},\lambda}=(m^*+\lambda)\cdot D-\frac{UD}{s^*}>(m^*+1)\cdot D-\frac{UD}{s^*}=t^*_{\mathrm{idle},1}$. 根据式(4)可知，如果$t^*_{\mathrm{idle},\lambda}\leq t_\theta$，则$t^*_{\mathrm{idle},1}<t_\theta$，$E^*_{m^*+\lambda}\leftarrow P(s^*)\frac{UD}{s^*}+P_{\mathrm{idle}}t^*_{\mathrm{idle},\lambda}$，$E^*_{m^*+1}\leftarrow P(s^*)\frac{UD}{s^*}+P_{\mathrm{idle}}t^*_{\mathrm{idle},1}$. 故而 $E^*_{m^*+\lambda}>E^*_{m^*+1}$. 如果 $t^*_{\mathrm{idle},\lambda}>t_\theta$，则 $E^*_{m^*+\lambda}\leftarrow P(s^*)\cdot\frac{U\cdot D}{s^*}+P_{\mathrm{idle}}\cdot t_\theta$. 因为 $t^*_{\mathrm{idle},1}<t^*_{\mathrm{idle},\lambda}$，如果 $t^*_{\mathrm{idle},1}<t_\theta$，则 $E^*_{m^*+\lambda}>E^*_{m^*+1}$；如果 $t^*_{\mathrm{idle},1}\geq t_\theta$，则 $E^*_{m^*+\lambda}=E^*_{m^*+1}$. 由此可得，$E^*_{m^*+\lambda}\geq E^*_{m^*+1}$，所以 $\min\{E^l_{m^*+\lambda},E^*_{m^*+\lambda}\}\geq E^*_{m^*+1}\geq\min\{E^l_{m^*+1},E^*_{m^*+1}\}$.

综上可得 $\min\{E^l_{m^*+\lambda},E^*_{m^*+\lambda}\}\geq\min\{E^l_{m^*+1},E^*_{m^*+1}\}$.　　证毕.

定理 4.　Check_Switching_Overhead 算法可以得到具有最低能耗值的最优处理器数，同时 LUF-SO 算法可以得到最优的节能调度.

证明.　由引理 3 可知，Check_Switching_Overhead 算法确定的能耗值E_{m^*}是 $m^*=\left\lfloor\sum_{\forall\tau_i\in T'}u_i/s^*\right\rfloor$ 个处理器上能耗最低值；由定理 2 可知，Check_Switching_Overhead 算法确定的能耗值$\min\{E^l_{m^*+1},E^*_{m^*+1}\}$是 m^*+1 个处理器上能耗最低值；再根据定理 1 和定理 3 可知，Check_Switching_Overhead 算法确定的能耗值 $E_{\min}\leftarrow\min\{E^l_{m^*+1},E^*_{m^*+1},E_{m^*}\}$是所有 $m>m^*$ 个处理器上的最低能耗值，由此得到具有最低能耗值 $E_{\min}$的处理器数即是最优的.

已知 T 中所有任务按照任务利用率的非递增顺序排列，当单个任务的利用率 $u_i\geq s^*$ 时，或者当剩余未分配速度的任务集 $T'\subseteq T$ 中在 M 个处理器上的平均利用率$\frac{U}{M}\geq s^*$ 时，LUF-SO 算法对所有分配速度的任务的处理过程与 LTF-M 算法相同. 因此，直接根据功耗函数 $P(s)$的凸函数性质以及 LTF-M 算法的最优性证明过程，可知此时 LUF-SO 算法在 $s\geq s^*$ 得到的节能实时调度是最优的. 当$u_i<s^*$ 且$U/M<s^*$ 时，LUF-SO 算法调用 Check_Switching_ Overhead 算法可以获得最低的能耗值，同时保证 $T'\subseteq T$ 中所有任务的可调度性. 综上，LUF-SO 算法可以得到最优的节能调度.　　证毕.

5　示　例

本节通过举例说明 LUF-SO 算法的最优性. 以 Intel Xscale 处理器的功耗函数为例，每个处理器的功耗函数可以被近似建模为 $P(s)=1.52\cdot s^3+0.08\mathrm{W}$[10-12]. 假设处理器速度以 $s_{\max}$为标准归一化为[0,1]GHz($s_{\min}$为 0 GHz，$s_{\max}$为 1 GHz)之间任意值. 根据 2.1 节可知，关键速度 s^* 约为 0.297 GHz，在关键速度 s^* 上的功耗 $P(s^*)=0.12$ W. 假设切换开销 E_{sw}为 0.8 mJ，处理器的空闲功耗 $P_{\mathrm{idle}}=0.08$ W，则 break-even 时间 t_θ为 10 ms. 给定一个基于帧的实时任务集 $T=\{\tau_1,\tau_2,\tau_3,\tau_4,\tau_5,\tau_6\}$在多处理器系统中调度执行，处理器个数 $m=4$，所有任务具有相同周期 $D=30$ ms. 不妨假设所有任务已经按照利用率的非递增顺序排列，每个任务的最坏情况执行时钟数和利用率如表 1 所示.

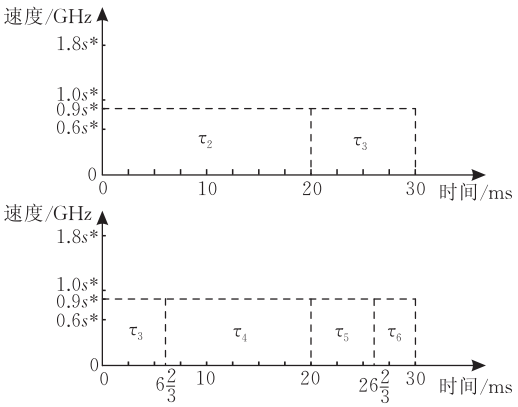
表 1　任务集举例

任务集	C_i	u_i
τ_1	$0.036s^*$	$1.2s^*$
τ_2	$0.018s^*$	$0.6s^*$
τ_3	$0.015s^*$	$0.5s^*$
τ_4	$0.012s^*$	$0.4s^*$
τ_5	$0.006s^*$	$0.2s^*$
τ_6	$0.003s^*$	$0.1s^*$

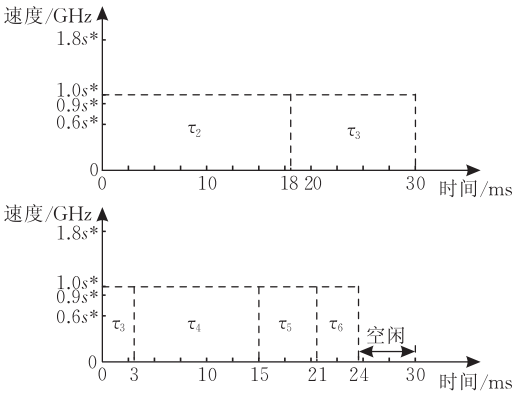
根据 LUF-SO 算法可知，首先需要为利用率最大的任务 τ_1分配执行速度 $\tau_1.speed$. 此时由于 $U=\sum_{\forall\tau_i\in T}u_i=3s^*$，$M=m$，所以 $U/M=0.75s^*<s^*$. 而 $u_1=1.2s^*>s^*$，且 $u_1>U/M$，因此 $\tau_1.speed=u_1$，$U=U-u_1=1.8s^*$，$M=m-1=3$. 然后为后续任务 τ_2分配执行速度，此时 $U/M=0.6s^*<s^*$ 且 $u_2=0.6s^*<s^*$，由 LUF-SO 算法可得 $m^*=\lfloor U/s^*\rfloor=1$，从而调用 Check_Switching_Overhead 算法. 针对所

有未分配速度的后续任务子集 $T'=\{\tau_2,\tau_3,\tau_4,\tau_5,\tau_6\}$，Check_Switching_Overhead 算法获得具有最低能耗值的最优处理器数 m^* 以及后续任务子集中所有任务相应的执行速度. 根据定理 4 可知，Check_Switching_Overhead 算法中具有最低能耗值的解只需要分析下列 3 种情况：

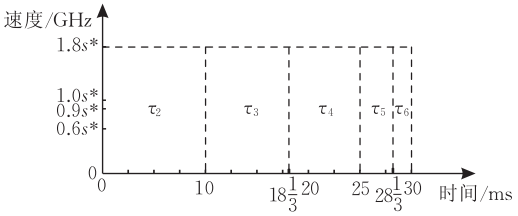
情况 1. $T'\subseteq T$ 中所有任务在 m^*+1 个处理器上以完全利用整个周期时间的速度运行，没有空闲时间存在. 由于 $u_2<U/(m^*+1)=0.9s^*$，所以 $\forall\tau_j\in T'$，任务 τ_j 的执行速度为 $\tau_j.speed=0.9s^*$ GHz. 因为此时 Check_Switching_Overhead 算法采取与 LTF 算法的相同方法分配所有任务的执行速度，所以对应的调度序列如图 3(a)所示. 根据式(3)可知能耗值 $E_2^l=P(0.9s^*)(m^*+1)D=(0.04\cdot 0.9^3+0.08)\cdot 60=6.5496$ mJ.



(a) 以完全利用整个周期时间的速度在两个处理器上执行



(b) 以关键速度在两个处理器上执行



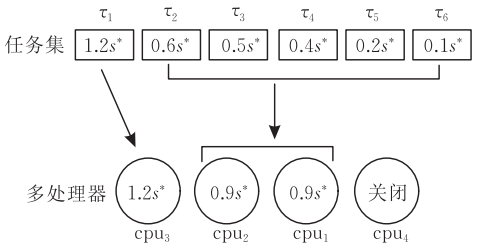
(c) 以完全利用整个周期时间的速度在一个处理器上执行

图 3 Check_Switching_Overhead 算法示例

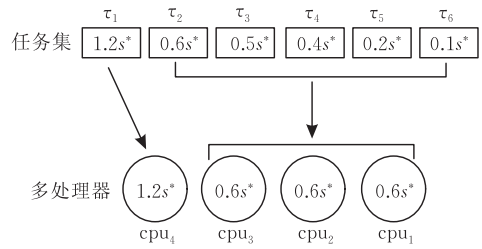
情况 2. $T'\subseteq T$ 中所有任务在 m^*+1 个处理器上以关键速度 s^* 运行，存在空闲时间 $t_{\text{idle}}^*=(m^*+1)\cdot D-U\cdot D/s^*$. 此时 Check_Switching_Overhead 算法分配 $T'\subseteq T$ 中所有任务的执行速度为 s^* GHz，对应的调度序列如图 3(b)所示. 由于 $t_{\text{idle}}^*=2\times 30-1.8\times 30=6$ ms，$t_\theta=10$ ms，所以 $t_{\text{idle}}<t_\theta$. 根据式(4)可知能耗值 $E_2^*=P(s^*)UD/s^*+P_{\text{idle}}t_{\text{idle}}=0.12\cdot 1.8\cdot 30+0.08\cdot 6=6.96$ mJ.

情况 3. $T'\subseteq T$ 中所有任务在 m^* 个处理器上以完全利用整个周期时间的速度运行，没有空闲时间存在. 因为 $m^*\neq 0$，所以根据引理 3 可知为得到在 m^* 个处理器上最低的能耗值，Check_Switching_Overhead 算法分配 $T'\subseteq T$ 中所有任务的执行速度为 $U/m^*=1.8s^*$ GHz，对应的调度序列如图 3(c)所示. 根据式(2)可知能耗值 $E_1^l=P(1.8s^*)\cdot m^*\cdot D=(0.04\cdot 1.8^3+0.08)\cdot 30=9.3984$ mJ.

当得到以上 3 种情况的能耗值后，Check_Switching_Overhead 算法确定的最低能耗值 $\min\{E_2^l,E_2^*,E_1^l\}=E_2^l$，最后返回最优处理器数 $m^*=2$. 当分配完 T 中所有任务的执行速度之后，LUF-SO 算法将所有任务指定到相应的 $M=3$ 个处理器上调度执行，如图 4(a)所示. 而如果采用 LTF-M 算法调度相同任务集 T，则 LTF-M 算法会将 T 中所有任务指定到 $m=4$ 个处理器上执行，如图 4(b)所示.



(a) LUF-SO算法在3个处理器上的调度序列



(b) LTF-M算法在4个处理器上的调度序列

图 4 LUF-SO 算法和 LTF-M 算法示例

图 4 中每个方框表示一个任务，方框中数值表示该任务的利用率. 每个圆表示一个处理器，圆中数值表示处理器的执行速度. 如果某个处理器没有被分配执行速度，则标记为“关闭”. 从图 4(a)可知，任

务 τ_1 在处理器 cpu_3 上以 $1.2s^*$ GHz 为速度从时刻 0 ms 到时刻 30 ms 执行，$T' \subseteq T$ 中所有任务分别在处理器 cpu_2 和 cpu_1 以 $0.9s^*$ GHz 为速度从时刻 0 ms 到时刻 30 ms 执行，因此 LUF-SO 算法得到的总能耗值为 $E_{\text{LUF-SO}} = P(1.2s^*)D + E_2^l = 11.0232$ mJ. 而从图 4(b)可知，除了任务 τ_1 在处理器 cpu_4 上以 $1.2s^*$ GHz 为速度从时刻 0 ms 到时刻 30 ms 执行以外，$T' \subseteq T$ 中所有任务分别在其它 3 个处理器上以 $0.6s^*$ GHz 为速度从时刻 0 ms 到时刻 30 ms 执行，因此 LTF-M 算法得到的总能耗值为 $E_{\text{LTF-M}} = P(1.2s^*) \cdot D + P(0.6s^*) \cdot 3 \cdot D = 12.4512$ mJ. 相比 LTF-M 算法，LUF-SO 最优算法能够节余能耗约 11%.

6　结束语

本文针对具有独立 DVFS 的多处理器系统，在考虑处理器状态切换开销情况下，提出一种基于帧任务模型的最优节能实时调度算法 LUF-SO. LUF-SO 算法允许实时任务在处理器之间的任意迁移，可以离线确定任务集的执行过程和执行速度. 同时，该算法根据关键速度来判断系统的低负载情况，一旦确定系统处于低负载情况，首先确定具有最低能耗值的活跃处理器个数，然后根据状态切换时间和能量开销来确定最优调度序列. LUF-SO 算法实现简单且复杂度小. 本文经过系统的理论分析证明了该算法的最优性.

参　考　文　献

[1] Jejurikar R, Pereira C, Gupta R. Leakage aware dynamic voltage scaling for real-time embedded systems//Proceedings of the Design Automation Conference. San Diego, USA, 2004: 275-280

[2] Chen J J, Kuo C F. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms//Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. Daegu, Korea, 2007: 28-38

[3] Aydin H, Yang Q. Energy-aware partitioning for multiprocessor real-time systems//Proceedings of the 17th IEEE International Parallel and Distributed Processing Symposium. Nice, France, 2003: 22-26

[4] Chen J J. Energy-efficient scheduling for real-time tasks in uniprocessor and homogeneous multiprocessor systems [Ph. D. dissertation]. National Taiwan University, Paipei, China, 2006

[5] Chen J J, Hsu H R, Chuang K H, Yang C L, Pang A C, Kuo T W. Multiprocessor energy-efficient scheduling with task migration considerations//Proceedings of the 16th Euromicro Conference Real-Time Systems. Catania, Italy, 2004: 101-108

[6] Funaoka K, Kato S, Yamasaki N. Energy-efficient optimal real-time scheduling on multiprocessors//Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing. Orlando, USA, 2008: 23-30

[7] Xu R, Zhu D, Rusu C, Melhem R, Mossé D. Energy-efficient policies for embedded clusters//Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems. Chicago, USA, 2005: 1-10

[8] de Langen P J, Juurlink B H H. Leakage-aware multiprocessor scheduling for low power//Proceedings of the 20nd IEEE International Parallel and Distributed Processing Symposium. Rhodes Island, Greece, 2006: 80-87

[9] Chen J J, Hsu H R, Kuo T W. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems//Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium. San Jose, USA, 2006: 408-417

[10] Chen J J, Thiele L. Energy-efficient scheduling on homogeneous multiprocessor platforms//Proceedings of the 25th ACM Symposium on Applied Computing. Sierre, Switzerland, 2010: 542-549

[11] Zhu D. Reliability-aware dynamic energy management in dependable embedded real-time systems//Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium. San Jose, USA, 2006: 397-407

[12] Bansal N, Kimbrel T, Pruhs K. Dynamic speed scaling to manage energy and temperature//Proceedings of the Symposium on Foundations of Computer Science. Rome, Italy, 2004: 520-529

[13] Irani S, Shukla S, Gupta R. Algorithms for power savings//Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, USA, 2003: 37-46

ZHANG Dong-Song, born in 1980, Ph. D.. His current research interests include real-time system and complex system simulation.

WU Fei, born in 1968, Ph. D., associate professor. His current research interests include information intelligent transaction and distributed computing.

CHEN Fang-Yuan, born in 1982, Ph. D.. Her current research interests focus on computer architecture and real-time system.

WU Tong, born in 1979, Ph. D., associate professor.

His current research interests focus on real-time system and military high-tech.

GUO De-Ke, born in 1980, Ph. D., associate professor. His current research interests focus on wireless multi-hop networks, real-time system, peer-to-peer computing, and datacenter networking.

JIN Shi-Yao, born in 1937, professor, Ph. D. supervisor. His current research interests include real-time system, complex system simulation, distributed computing, and virtual reality.

Background

With regard to dynamic power consumption, early research efforts show that the optimal scheduling fits well in workload balance among all processors to obtain minimum energy consumption. Some researchers propose the optimal energy-efficient real-time scheduling algorithms based on global scheduling. However, the optimal algorithm is not a perfect solution without any cost. This is because processor switching will be at a cost of time and energy overheads. Therefore, if energy savings is less than processor switching overheads, it is obviously not energy-efficient. Nowadays, researchers pay attention to the actual platforms with switching overheads. Furthermore, research does not take switching overheads into consideration.

As above mentioned, there are three main issues for recent research: (1) Most algorithms are based on non-optimal global and partitioning scheduling method. Thus the optimal feasibility can not be guaranteed. (2) Although recent methods could save energy, the minimum energy consumption can not be guaranteed. (3) The time and energy overheads are not taken into consideration in existing optimal energy-efficient real-time scheduling algorithms, which results that these algorithms are no more optimal in actual platforms.

This paper proposes an optimal multiprocessor energy-efficient real-time scheduling algorithm for frame-based tasks. The proposed optimal algorithm determines the system workload cases and the number of active processor cores in tems of critical speed. Then we can obtain the optimal scheduling according to the switching overhead. Systematic mathematical analysis shows that the algorithm is optimal.

This work has done research on energy-efficient real-time scheduling in RTOS on multiprocessors. It is a part of our projects mainly supported by the National Natural Science Foundation of China under Grant Nos. 61170284 and 60903206; the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20104307110005; the China Postdoctoral Science Foundation under Grant Nos. 20100480898 and 201104439; the Hunan Provincial Innovation Foundation for Postgraduate of China under Grant No. CX2010B026; and the Excellent Graduate Innovation of National University of Defense Technology under Grant No. B100601, and the Innovation Program of Shanghai Municipal Education Commission under Grant No. 12ZZ182.

Our past work includes energy-efficient real-time scheduling in single-core and multi-core systems. The works have been published in ACM Transactions on Architecture and Code Optimization, Journal of Software, Journal of Computers and so on. We will continue our work on RTOS scheduling on the actual platforms with unignorable overheads.