文章编号:1001-2486(2011)06-0007-10

多核系统中基于动态松弛时间回收的节能实时调度算法*

张冬松[1],郭得科[2],陈芳园[1],吴 彤[3],吴 飞[4],金士尧[1]
(1. 国防科技大学 计算机学院,湖南 长沙 410073;
2. 国防科技大学 信息系统工程重点实验室,湖南 长沙 410073;
3. 国防科技大学 国家安全与军事战略研究中心,湖南 长沙 410073;
4. 上海工程技术大学 电子电气工程学院,上海 201620)

摘 要:许多嵌入式实时任务的实际执行时间往往小于最坏情况执行时间,因此可以产生大量的动态松弛时间。针对时限等于周期的偶发任务集,提出一种基于动态松弛时间回收的多核系统节能实时调度算法DSREM。该算法基于最优在线调度算法LRE-TL,利用TL面内节能实时调度思想,在每个TL面的提前完成时刻实现动态松弛时间回收,降低后续偶发任务的执行频率,实现更多的节能。同时该算法还在每个TL面的初始时刻、偶发任务的释放时刻实现动态电压和频率调节,保证了偶发任务集最优可调度性。经过系统的数学分析和仿真,结果表明,DSREM算法不仅保证了偶发任务集的最优可调度性,而且当任务集总负载超过某一个值后,其节能效果始终优于现有方法,尤其随着总负载的增加,节能效果会更好。

关键词:实时系统;节能调度;多核;动态松弛时间
中图分类号:TP316 **文献标识码**:A

An Energy-efficient Multi-core Real-time Scheduling Algorithm Based on Dynamic Slack Reclamation

ZHANG Dong-song[1], *GUO De-ke*[2], *CHEN Fang-yuan*[1], *WU Tong*[3], *Wu Fei*[4], *JIN Shi-yao*[1]
(1. College of Computer, National Univ. of Defense Technology, Changsha 410073, China;
2. Science and Technology on Information Systems Engineering Laboratory,
National Univ. of Defense Technology, Changsha 410073, China;
3. Center for National Security and Strategic Studies, National Univ. of Defense Technology, Changsha 410073, China;
4. College of Electronic and Electrical Engineering, Shanghai Univ. of Engineering Science, Shanghai 201620, China)

Abstract: In many embedded real-time systems, the actual execution time of tasks is usually less than their worst-case execution time (WCET), which produces lots of dynamic slack time. Based on this fact, we proposed an energy-efficient real-time scheduling algorithm DSREM for sporadic tasks deployed on multi-cores, which is based on optimal online scheduling algorithm LRE-TL. The main idea of the algorithm is to reclaim dynamic slack time, during which DVFS techniques can be used to reduce the execution frequency of future tasks to eliminate the energy consumption. Meanwhile, our algorithm also dynamically scales the voltage and frequency at the initial time of each TL plane and the release time of a sporadic task in each TL plane to guarantee the optimal schedulability of sporadic tasks. Systematic mathematical analysis and extensive simulation results show that DSREM can always save more energy than the existing algorithms when the total workload of the system exceeds a threshold, and it can also guarantees the optimal schedulability at the same time.

Key words: real-time system; energy-efficient scheduling; multi-core; dynamic slack time

为有效减少能耗,现代多核处理器系统广泛采用了各种硬件节能技术,如动态电压频率调节(Dynamic Voltage Frequency Scaling, DVFS)[1]和动态功耗管理(Dynamic Power Management, DPM)[2],但是这两种硬件节能技术并没有考虑任务的实时性,在进行节能调度时很可能会因为降频节能而导致执行时间增加,从而错失截止期。因此,为了满足嵌入式实时应用的进一步发展,随

* **收稿日期**:2011-06-01
基金项目:国家自然科学基金资助项目(60903206, 60803152, 60972166);国家教育部博士点基金资助项目(20104307110005);国家部委资助项目;国防科技大学资助项目;国防科技大学优秀研究生创新资助项目(B100601);湖南省优秀研究生创新资助项目(CX2010B026)
作者简介:张冬松(1980—),男,博士生。

着“绿色计算”需求的提出,节能实时调度技术成为研究的急需。

目前,控制简单、实现代价小的片上全局 DVFS[3]已被一些商业多核处理器[4]采用,基于这种多核平台的实时节能调度也吸引了越来越多的研究者的关注。在全局电压和频率的约束下,在调度时刻具有最大负载的处理器核成为多核系统中节能实时调度中起到主导的因素[5-6]。当前的研究[5-9]主要关注在多个处理器上均衡负载以实现能耗最小化。文献[6]首次针对具有片上全局 DVFS 的多核处理器系统,提出了基于帧任务模型的静态节能实时调度算法。文献[9]针对执行软实时任务模型的多核系统提出一种动态节能调度算法。文献[8]针对周期任务模型,提出了一种基于 Global EDF 的多核系统动态节能实时调度算法。文献[7]考虑执行周期任务的多核系统,提出基于各种启发式划分法的动态节能实时调度算法。该算法基于划分调度方法实现任务调度,在不允许任务迁移的条件下动态降低处理器频率以及将空闲核转入低功耗状态,可以在低负载情况下减少系统能耗,但是同样没有考虑高负载情况下节能效果。文献[5]针对执行周期任务模型的多核系统,基于启发式任务划分方法,提出了两种动态节能实时调度算法 Dynamic Repartitioning (DR) 和 Dynamic Core Scaling (DCS)。前者能够在任务执行过程中利用任务提前完成而产生的动态松弛时间(Dynamic Slack Time)进行部分任务迁移,通过平衡多个核的负载以降低能耗,而后者能够在前者的基础上,通过在低负载情况下调节活跃处理器核的数目以进一步减少泄漏功耗,从而获得更多的能耗节余。

现有算法的不足主要表现在:1)这些算法大多基于划分调度法或非最优的全局调度法,无法保证实时任务的最优可调度性;2)在高负载情况下节能效果差;3)很少考虑到现实世界中更普遍的偶发任务模型;4)不适应任务执行的动态变化,不能有效分析和利用动态松弛时间。

当任务实际执行时间小于最坏情况执行时间时,任务提前完成可以产生动态松弛时间。本文通过有效回收这一动态松弛时间,从而降低后续偶发任务的执行频率,实现更多的节能。由于现有考虑动态松弛时间的动态节能调度算法[5,7]都是针对周期任务这些简单模型,需要事先已知所有任务的释放时间,而偶发任务模型具有任意的释放时间,所以直接应用现有算法不一定有效。因此,本文针对时限等于周期的偶发任务集,提出一种基于动态松弛时间回收的多核系统节能实时调度算法 DSREM (Dynamic Slack Reclamation based Energy-efficient Multi-core real-time scheduling)。该算法基于最优在线调度算法 LRE-TL,利用 TL 面内节能实时调度思想,在每个 TL 面的提前完成时刻实现动态松弛时间回收,进一步降低系统能耗。为了保证偶发任务集最优可调度性,该算法还在每个 TL 面的初始时刻、偶发任务的释放时刻实现动态电压和频率调节,达到实时约束与能耗节余之间的合理折中。经过系统的数学分析和仿真,结果表明,DSREM 算法不仅保证了偶发任务集的最优可调度性,而且当任务集总负载超过某一个值后,其节能效果始终优于现有方法,尤其随着总负载的增加,节能效果会更好。

1　系统模型

1.1　处理器模型

本文考虑具有片上 DVFS 的多核处理器,假设多核处理器拥有 m 个同构处理器核 $\{Core_1, Core_2, \cdots, Core_m\}$。通常,基于 CMOS 电路的处理器功耗 P_{tot} 由动态功耗 P_d 和泄漏功耗 P_l 组成,即 $P_{tot} = P_d + P_l$[5]。其中,动态功耗 P_d 是指电路充电和放电时电容的切换功耗,是处理器在执行指令时总功耗的主要部分。P_l 是泄漏功耗,由泄漏电流所引起。P_d 可以表示为供应电压 V_{dd}、时钟频率 f 以及切换电容 c_l 的函数,即 $P_d = c_l \cdot V_{dd}^2 \cdot f$[10]。这里时钟频率 f 可以表示为 $f = \frac{(V_{dd} - V_{th})^{\varepsilon}}{L_d \cdot K_6}$[10],其中阈值电压 V_{th} 是反转偏移电压 V_{bs} 的函数,表示为 $V_{th} = V_{th1} - K_1 \cdot V_{dd} - K_2 \cdot V_{bs}$,而 ε、V_{th1}、K_1、K_2、K_6 都是依赖于处理器制造工艺的常量。一般来说,ε 是介于 1 与 2 之间的常数,所以在阈值电压之上增加 V_{dd} 会使得处理器增加时钟频率 f。P_l 主要由低于阈值的泄漏电流 I_{subn} 和反转偏移电流 I_j 组成,可以表示为 $P_l = L_g \cdot (V_{dd} \cdot I_{subn} + |V_{bs}| \cdot I_j)$[11],其中 $I_{subn} = K_3 \cdot e^{K_4 V_{dd}} \cdot e^{K_5 V_{bs}}$,$L_g$ 定义为电路中组件的个数,K_3、K_4、K_5 都是由处理器制造工艺决定的常量。

假设每个处理器核有两种模式:1)活跃状态,即执行指令;2)睡眠状态,即没有指令需要执行且维持着最小功耗值。与文献[5]相同,本文假设处理器核在睡眠状态只有泄漏功耗,泄漏功耗为当前活跃状态处理器核所用泄漏功耗的 3%,同时状态转换没有开销。此外,为了模拟以

上描述的功耗模型,本文采用与文献[5]相同的实际常量值[11],假设$f_{\min}$和$f_{\max}$分别表示电压和频率连续调节的最低和最高处理器频率。

1.2　任务模型

本文考虑时限等于周期的偶发任务集 $\Gamma=\{T_1,T_2,\cdots,T_n\}$。每个任务 T_i 具有多个任务实例 $T_{i1},T_{i2},\cdots$。任务 T_i 可以用四元组$(\varphi_i,a_{ij},P_i,C_i)$来描述,其中 φ_i 是任务 T_i 首次释放时的偏移量,a_{ij}是任务 T_i 在第 j 次调用时的释放时间,P_i 是 T_i 的周期或最小释放间隔时间,C_i 是 T_i 在最高处理器频率 f_m 下的最坏情况执行时间或执行时钟数。任务 T_i 在时刻 φ_i 释放第一个任务实例之后,后续任务实例 T_{ij}的释放时间间隔最小不少于 P_i 个时间单位,即 $a_{i1}=\varphi_i\geqslant 0$,且 $a_{ij}\geqslant a_{i(j-1)}+P_i$。$T_{ij}$ 的时限等于其周期 P_i。当 T_{ij}在 a_{ij}时刻释放,T_{ij}的绝对时限为 $d_{ij}=a_{ij}+P_i$;如果 T_{ij}在处理器核 $Core_k$ 上以频率 α_k 执行,那么为了满足时限约束,该任务实例必须在$[a_{ij},a_{ij}+P_i]$内完成 C_i/α_k 执行时间。所有的任务实例必须在绝对时限之前完成。任务 T_i 的利用率为 $u_i=C_i/P_i$,任务集的总利用率为 $U=\sum_{T_i\in\Gamma}u_i$,而其中最大的任务利用率为 $u_{\max}=\max\{u_i\mid T_i\in\Gamma\}$。本文还假设所有任务都是独立的,可以在任意时刻在处理器核之间抢占和迁移。

任务调度算法采用基于 TL 面的最优在线调度算法 LRE-TL[12]。TL 面是实时调度中的一种抽象概念[14-15]。简单地说,TL 面是一个二维平面,水平轴表示时间,垂直轴表示任务的局部剩余执行时间(Local Remaining Execution Time)。如果任务 T_i 的局部剩余执行时间 $l_{i,t}>0$,则称 T_i 处于活跃状态[12]。为了便于区分时刻 t 的活跃任务与非活跃任务,设 $Active(t)$为在 t 时刻处于活跃状态的所有任务构成的集合[12]。在任意时刻 t,任务 T_i 的局部利用率 $u_{i,t}$可以定义为 T_i 的局部剩余执行时间在当前 TL 面的剩余时间内所占的比例,即 $u_{i,t}=l_{i,t}/(t_f-t)$[13]。为了更好地描述任务在 TL 面内的节能调度,本文还定义任务的频率调节因子 α,任务 T_i 在时刻 t 的有效局部剩余执行时间 $l'_{i,t}=l_{i,t}/\alpha$,任务 T_i 的有效局部利用率 $r'_{i,t}=l'_{i,t}/(t_{f_t}-t)$以及 $Active(t)$任务集的 总利用率 $U_t=\sum_{\forall T_i\in Active(t)}u_{i,t}$ 和最大任务利用率 $u^t_{\max}=\max\{u_{i,t}\mid\forall T_i\in Active(t)\}$。

2　DSREM 算法思想

由于许多嵌入式实时任务的实际执行时间相比事先测得的最坏情况执行时间会有 87% 的变化量[14],任务在实际执行过程中会产生大量的动态松弛时间,因此 DSREM 算法在 TL 面内引入提前完成事件,通过回收动态松弛时间,在保证可调度性的前提下降低后续任务的执行频率以实现节能。

DSREM 算法将动态电压和频率调节操作引入到最优在线实时调度 LRE-TL 算法中,考虑属于同一 TL 面中多个偶发任务的节能实时调度,如图 1 所示。图中 token 标记表示 TL 面中每个偶发任务的状态。token 标记的位置用水平轴的时间和垂直轴的任务局部剩余执行时间来表示。图中 TL 面内的 α 斜边定义为从初始时刻 t_0 到结束时刻 t_f 之间斜率为 $-\alpha$ 的线段,表示处理器核以频率为 $\alpha\cdot f_m$ 执行任务。从图中可知任务 T_1 的局部剩余执行时间 $l_1=\alpha\cdot t_f$,其有效局部松弛时间为 $t_f-t_0-l_n/\alpha$。从图 1 中可知,由于任务执行频率可以在不超过最高频率的范围内变化,所以 TL 面的 α 斜边是不固定的。

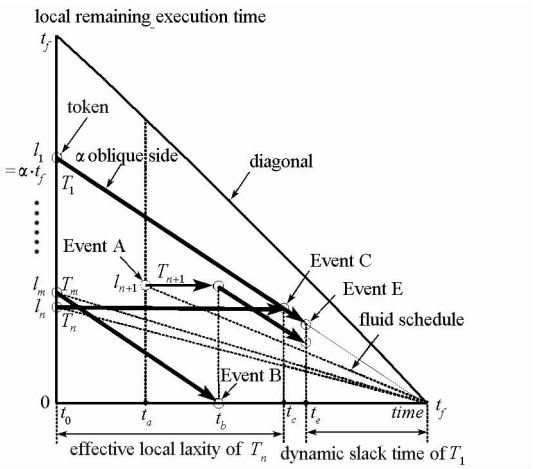

图 1　TL 面内节能实时调度

Fig. 1　Energy-efficient real-time scheduling in a TL plane

DSREM 算法基于 TL 面的初始时刻以及 TL 面内的两类事件进行频率和电压调节:1)首先在每个 TL 面的初始时刻根据当前活跃偶发任务个数,调用 TL 面初始化程序来进行初始频率设置以及初始化操作;2)TL 面内第一类事件发生在偶发任务在 TL 面内释放的时刻,此时,TL 面内增加一个任务的 token 标记,如图 1 中 T_{n+1}在时刻 t_a 释放,称为释放事件 A。此时任务集发生改变,系统需要重新进行频率调节和实时调度;3)第二类事件发生在任务已经完成,但其局部剩余执行时间不为 0 的时刻,此时正沿着 α 斜边移动的任务 token 标记会消失,如图 1 中 T_1 移动到时刻 t_e,称为提前完成事件 E。这表明任务 T_1 的实际执行

时间小于最坏情况执行时间,从而产生动态松弛时间($t_f - t_e$),系统可以回收这一动态松弛时间实现更多节能。

同时,为了保证任务的可调度性,DSREM 算法还需要保证任务正常完成(即事件 B)和任务抢占(即事件 C)时刻对任务的处理不会影响 TL 面内所有任务的局部可调度性。正常完成事件 B 发生在任务局部剩余执行时间为 0 时刻,任务的 token 标记会击中 TL 面的水平轴,如图 1 中 T_m 移动到时刻 t_b。注意到事件 B 不会产生动态松弛时间。另外,任务抢占事件 C 发生在任务有效局部松弛时间为 0 的时刻,此时任务 token 标记击中 α 斜边,如图 1 中 T_n 移动到时刻 t_c。此时任务 T_n 必须马上被选中以频率 $\alpha \cdot f_m$ 调度执行,否则就会远离 α 斜边,导致不可能到达 TL 面的结束时刻。

如果在 TL 面的结束时刻 t_f,所有任务的局部剩余执行时间均为 0,则称所有任务是局部可调度的[13]。如果所有任务在每个 TL 面内都是局部可调度的,那么它们在任意连续多个 TL 面内一定是可调度的[13]。为了分析局部可调度性,本文提出有效关键时刻(effective critical moment)用来描述在动态电压和频率调节下所有任务在 TL 面内局部不可调度的充要条件。

定义 1 有效关键时刻是指在频率调节因子 α 下,第一次出现的有多于 m(即处理器核数)的任务同时击中 TL 面的 α 斜边。

对于 TL 面内所有任务,只允许沿水平或 α 斜边方向移动:1)若任务被选中执行,该任务的 token 标记沿 α 斜边方向移动,如图 1 中 T_1 和 T_m 的移动所示;2)若任务没有被选中执行,该任务的 token 标记沿水平方向移动,如图 1 中 T_n 移动所示。在有效关键时刻的右侧,最多只有 m 个任务被选中沿着 α 斜边继续执行,如图 2 所示。未选中的任务会远离 α 斜边,在到达 $t_f - l_{m+1}$ 时刻会离开 TL 面,不可能到达 TL 面的结束点 t_f。

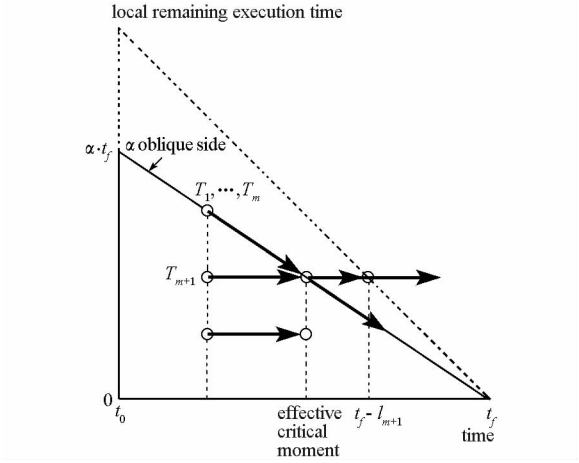

图 2 有效关键时刻

Fig. 2 Effective critical moment

定理 1 在频率调节因子 α 下,任务集在 TL 面内是局部不可调度的当且仅当至少出现一次有效关键时刻。

证明 必要性:采用反证法,假设任务集是局部不可调度时,没有出现有效关键时刻。如果没有一次有效关键时刻出现,则根据定义 1 可知任何时刻同时到达 TL 面的 α 斜边的任务个数不会超过 m。因此所有在斜边上的任务都会被选中执行到 TL 面的结束时刻 t_f。这就与之前任务集的局部不可调度性假设相矛盾。

充分性:假设出现有效关键时刻,那么未选中的任务必定会离开 TL 面。由于在频率调节 α 下所有任务只允许沿水平或 α 斜边方向移动,即这些任务移动路径的斜率只能是 0 或者 $-\alpha$,所以离开 TL 面的任务不可能达到 TL 面的最右顶点即结束时刻 t_f。□

本文定义任务集在时刻 t 的总有效局部利用率为 $R'_t = \sum_{i=1}^{n} r'_{i,t}$,据此可得推论 1。

推论 1 在频率调节因子 α 下,当有效关键时刻 t 出现时,一定存在 $R'_t > m$。

证明 因为在频率调节因子 α 下所有任务 $T_i \in Active$ 只允许沿水平或 α 斜边方向移动,所以在有效关键时刻 t,在 TL 面的 α 斜边上出现的 m 个任务的局部剩余执行时间 $l_{i,t}$ 显然等于 $\alpha \cdot (t_{f_t} - t)$。因此,$R'_t = \sum_{i=1}^{N} r'_{i,t} = \sum_{i=1}^{m} \frac{l_{i,t}/\alpha}{t_{f_t} - t} + \sum_{i=m+1}^{N} \frac{l_{i,t}/\alpha}{t_{f_t} - t} = \sum_{i=1}^{m} \frac{t_{f_t} - t}{t_{f_t} - t} + \sum_{i=m+1}^{N} \frac{l_{i,t}/\alpha}{t_{f_t} - t} > m$,其中 $N = |Active(t)|$,$N \leqslant n$。□

3 DSREM 算法描述

DSREM 算法在每个 TL 面的初始时刻、偶发任务的释放时刻以及任务提前完成时刻进行多核处理器系统的动态电压频率调节,在不违反偶发任务集可调度性的基础上,达到实时约束与能耗节余之间的合理折中。该算法的主程序与文献[12]中算法 1 类似。DSREM 算法在每个 TL 面的初始时刻实现 TL 面初始化和频率调节因子计算。在每个 TL 面内,DSREM 算法将处理事件 A 和事件 E 以进行动态松弛时间回收,从而动态调节电压和频率。DSREM 采用与 LRE-TL 同样的

方法[12]来确定下一个 TL 面的初始时刻。一旦初始化或相关事件处理程序被完成,该算法将确定处理器核的执行频率,指导每个处理器核按照分配的频率来执行被指派的任务。由于事件 B 和事件 C 的发生不会产生任务的动态松弛时间,因此本文对事件 B 和 C 的处理仍采取与文献[12]相同的方法。

由于处理器核 $Core_k$ 的电压和频率可以由其频率调节因子 α_k 确定,本文通过频率调节因子 α_k 的动态选择来动态调节电压和频率。本文考虑具有片上全局 DVFS 和 DPM 的多核处理器,进行电压和频率调节时,所有正在运行的处理器核的频率调节因子必须相同,即 $\alpha = \alpha_1 = \cdots = \alpha_k$。接下来,本文详细说明 TL 面初始化、事件 E 和事件 A 的处理过程。

3.1　TL 面初始化

在每个 TL 面的初始时刻并非所有的偶发任务都处于活跃状态,处理器核的执行频率只要保证当前活跃任务的可调度性即可,无需采用最高执行频率。因此,在每个 TL 面的初始时刻,本文根据当前所有处于活跃状态的任务来设置频率调节因子,据此来确定任务在处理器核上的执行频率。

Algorithm 1 TL-Initialize-Frequency-Selection

1: $U = 0$;
2: $u_{\max} = 0$;
3: for $i = 1$ to $|T_i \in Active|$ do
4: $\quad u_i = C_i / P_i$;
5: $\quad U = U + u_i$;
6: $\quad$ if$(u_{\max} < u_i)$ then
7: $\quad\quad u_{\max} = u_i$;
8: return $\alpha = \max\{u_{\max}, U/m\}$;

根据任务在 TL 面的流调度[14-15]可知任务的执行频率不会小于其利用率,否则会错失截止期。对于所有任务,本文在设置初始频率调节因子 α 时必须保证该频率不小于所有任务的最高利用率 $u_{\max}$(见算法 1 第 3 行到第 7 行),否则,具有最高利用率的任务以该频率执行时不可能达到 TL 面的结束时刻。同时,根据 EDF 算法易知,为了保证每个活跃任务以频率调节因子 α 执行时均满足局部可调度性,还必须满足 $\alpha \geqslant U_0/m$。因此 $\alpha = \alpha_1 = \cdots = \alpha_k = \max\{u_{\max}, U_0/m\}$(见算法 1 第 8 行)。证明过程请见定理 2。

定理 2　在每个 TL 面初始时刻 t_0,当 DSREM 算法通过算法 1 来确定频率调节因子 α 时,所有任务 $T_i \in Active$ 的有效局部利用率 $r'_{i,0} \leqslant 1$,总利用率 $U_0 \leqslant m$,以及总有效局部利用率 $R'_{t_0} \leqslant m$。

证明　在每个 TL 面初始时刻 t_0, $\forall T_i \in Active, u_{i,0} \leqslant u^0_{\max}$,且从算法 1 可知 $u^0_{\max} \leqslant \alpha \leqslant 1$,所以由定理 1 可得 $r'_{i,0} \leqslant 1$。不妨设 $N = |Active(t_0)|$, $N \leqslant |\Gamma|$,因为偶发任务集 Γ 满足 $U \leqslant m$,且 $u_{\max} \leqslant 1$,所以 $U_0 = \sum_{T_i \in Active} u_{i,0} \leqslant \sum_{T_i \in \Gamma} u_i = U \leqslant m$。由算法 1 已知 $U_0/m \leqslant \alpha \leqslant 1$,可得 $R'_{t_0} = \sum_{i=1}^{N} r'_{i,0} = \sum_{i=1}^{N} u_{i,0}/\alpha = U_0/\alpha \leqslant m$。 □

3.2　事件 *E*

事件 E 是偶发任务实例的一次提前完成,这使得当前 TL 面内产生动态松弛时间,此时通过重新确定频率调节因子可以回收动态松弛时间,达到进一步节能的目的。每当 TL 面内一个偶发任务 T_s 在 t_s 时刻完成执行离开处理器核时,DSREM 算法判断任务是否提前完成,即任务完成时间早于最坏情况下完成时间($t_s < T_s.key$),然后通过算法 2 来重新计算事件 E 触发后新的频率调节因子 α_{t_s}。

为了尽可能降低处理器电压和频率,算法 2 首先获得每个活跃任务 T_i 的有效局部剩余执行时间 l'_{i,t_s}(第 4 行到第 7 行),然后重新计算 T_i 的局部利用率 u_{i,t_s},得到当前 TL 面内最大利用率 $u^{t_s}_{\max}$ 和总利用率 U_{t_s}(第 8 行到第 11 行),最后确定新的频率调节因子 $\alpha_{t_s} \geqslant U_{t_s}/m$ 且 $\alpha_{t_s} \geqslant u^{t_s}_{\max}$(第 12 行)。

Algorithm 2 TL-E-Event-Frequency-Selection
Require: A sporadic task T_s completion in the kth processor core at time t and $Active = H_B \cup H_C - \{T_s\}$

1: $U = 0$;
2: $u_{\max} = 0$;
3: for $i = 1$ to $|T_i \in Active|$ do
4: $\quad$ if $T_i \in H_B$ then
5: $\quad\quad l' = T_i.key - t$;
6: $\quad$ if $T_i \in H_C$ then
7: $\quad\quad l' = t_f - T_i.key$;
8: $\quad u_i = \dfrac{l' \times T_i.speed}{t_f - t}$;
9: $\quad U = U + u_i$;
10: $\quad$ if $u_{\max} < u_i$ then
11: $\quad\quad u_{\max} = u_i$;
12: return $\alpha = \max\{u_{\max}, U/m\}$;

本文通过定理 3 证明当存在某个偶发任务 T_s

提前完成而产生的动态松弛时间被后续任务所利用,不会影响当前 TL 面内其他任务的局部可调度性。

定理 3 设 Γ 是在 m 个多处理器核上执行的偶发任务集,满足 $U \leqslant m$,且 $u_{\max} \leqslant 1$。假设$[t_0, t_f]$为采用 DSREM 算法调度 Γ 的任意 TL 面。不妨设任务 T_s 在时刻 t_s 完成执行,且 $t_s < T_s.key$,此时 t_s 为下一个事件 E 发生时刻,而事件 E 发生时原有的频率调节因子为 $\alpha \leqslant 1$。如果 $\forall t \in (t_0, t_s)$,满足 $\forall \Delta, 0 \leqslant \Delta \leqslant t_s - t, U_{t+\Delta} \leqslant m, R'_{t+\Delta} \leqslant m$,那么在时刻 t_s 需要将任务 T_s 从 *Active* 中去掉并通过算法 2 重新确定频率调节因子为 α_{t_s} 之后,$\alpha_{t_s} \leqslant \alpha, r'_{i,t_s} \leqslant 1, R'_{t_s} \leqslant m$ 以及 $U_{t_s} \leqslant U_{t+\Delta} \leqslant m$,这里 $\Delta = t_s - t$。

证明 由题设已知任务 T_s 在时刻 t_s 提前完成执行,即 $t_s < T_s.key$,此时原有的频率调节因子为 $\alpha \leqslant 1$,而 $\exists t \in [t_0, t_s)$,使得 $\Delta = t_s - t \geqslant 0$, $U_{t+\Delta} = \sum_{\forall T_i \in H_B \cup H_C} u_{i,t+\Delta} = \sum_{\forall T_i \in H_B \cup H_C} \frac{l'_{i,t+\Delta} \cdot \alpha}{(t_f - t - \Delta)} \leqslant m$, $R'_{t+\Delta} = \sum_{\forall T_i \in H_B \cup H_C} \frac{l'_{i,t+\Delta}}{(t_f - t - \Delta)} \leqslant m$。再由题设可知在时刻 t_s 需要将任务 T_s 从 *Active* 中去掉,即 $\forall T_i \in Active = H_B \cup H_C - \{T_s\}$,通过算法 2 易得 $u_{i,t_s} = \frac{l'_{i,t_s} \cdot \alpha}{t_f - t_s}, U_{t_s} = \sum_{\forall T_i \in Active} u_{i,t_s}$,重新确定频率调节因子 $\alpha_{t_s} = \max\{u_{\max}^{t_s}, U_{t_s}/m\}, r'_{i,t_s} = \frac{u_{i,t_s}}{\alpha_{t_s}} \leqslant \frac{u_{i,t_s}}{u_{\max}^{t_s}} \leqslant 1$。这里分两种情况判断:

(1) 如果 $U_{t_s}/m \geqslant u_{\max}^{t_s}$,则 $\alpha_{t_s} = U_{t_s}/m$。因为 $U_{t_s} = \sum_{\forall T_i \in H_B \cup H_C - \{T_s\}} \frac{l'_{i,t_s} \cdot \alpha}{t_f - t_s} = \sum_{\forall T_i \in H_B \cup H_C} \frac{l'_{i,t+\Delta} \cdot \alpha}{(t_f - t - \Delta)} - \frac{l'_{s,t_s} \cdot \alpha}{t_f - t_s} = (R'_{t+\Delta} - \frac{T_s.key - t_s}{t_f - t_s}) \cdot \alpha$,且 $\Delta = t_s - t$, $T_s.key \leqslant t_f, R'_{t+\Delta} \leqslant m$,所以 $\alpha_{t_s} = U_{t_s}/m \leqslant (R'_{t+\Delta} \cdot \alpha)/m \leqslant \alpha$。

(2) 如果 $u_{\max}^{t_s} > U_{t_s}/m$,则 $\alpha_{t_s} = u_{\max}^{t_s}$。因为 $T_{\max} \neq T_s, T_{\max} \in H_B$ or $H_C, t_s \leqslant T_{\max}.key \leqslant t_f$,所以 $l'_{\max,t_s} = (T_{\max}.key - t_s)$ or $(t_f - T_{\max}.key) \leqslant (t_f - t_s)$, $\alpha_{t_s} = u_{\max}^{t_s} = \frac{l'_{\max,t_s} \cdot \alpha}{t_f - t_s} \leqslant \alpha$。

综上可得 $u_{\max}^{t_s} \leqslant \alpha_{t_s} \leqslant \alpha \leqslant 1$,且 $U_{t_s} = \sum_{\forall T_i \in H_B \cup H_C - \{T_s\}} \frac{l'_{i,t_s} \cdot \alpha}{t_f - t_s} = \sum_{\forall T_i \in H_B \cup H_C} \frac{l'_{i,t+\Delta} \cdot \alpha}{(t_f - t - \Delta)} - \frac{l'_{s,t_s} \cdot \alpha}{t_f - t_s} \leqslant U_{t+\Delta} \leqslant m$。同时,$R'_{t_s} = \sum_{\forall T_i \in Active} \frac{l'_{i,t_s} \cdot \alpha}{(t_f - t_s) \cdot \alpha_{t_s}} = \sum_{\forall T_i \in Active} \frac{u_{i,t_s}}{\alpha_{t_s}} = \frac{U_{t_s}}{\alpha_{t_s}} \leqslant m$。 □

由定理 3 的结论,再结合定理 1 和推论 1 易知本文可保证在事件 E 发生之前任务集的局部可调度性,而 DSREM 算法对提前完成事件 E 的处理,包括重新计算频率调节因子,仍然可以保证在事件 E 发生之后所有任务的局部可调度性。

3.3 事件 *A*

事件 A 是偶发任务实例的一次释放,这使得多核系统的负载增加,此时活跃任务的个数以及属性都发生改变。本文在事件 A 发生时刻通过重新确定频率调节因子以实现保证局部可调度性的前提下达到节能的目的。

设偶发任务 T_s 在 TL 面内的 t_s 时刻释放其任务实例。根据定理 4 可知,在 T_s 释放之前活跃任务集的总有效局部利用率最多是 $m - u_s$,此时可以设 T_s 的局部剩余执行时间与它的利用率成正比,即 $l_{s,t_s} = u_s \cdot (t_{f_{t_s}} - t_s)$。由于 t_s 时刻活跃任务集中增加了新的任务 T_s,则当前的频率调节因子 α 需要随之变化以满足负载增加的需求。为了保证当前 TL 面内所有活跃任务的可调度性,算法 3 首先将 T_s 的利用率 u_s 加入所有任务的总利用率以及最大利用率中,即 $U_{t_s} = u_s$ 和 $u_{\max}^{t_s} = u_s$(第 1 行到第 2 行),然后获得每个活跃任务 T_i 的有效局部剩余执行时间 l'_{i,t_s},以及重新计算 T_i 的局部利用率 u_{i,t_s},同时得到当前 TL 面内最大利用率 $u_{\max}^{t_s} = \max\{u_{i,t_s}, u_{\max}^{t_s}\}$(第 3 行到第 11 行),从而重新确定当事件 A 发生时新的频率调节因子 $\alpha_{t_s} = \max\{u_{\max}^{t_s}, U_{t_s}/m\}$(第 12 行)。

Algorithm 3 TL-A-Event-Frequency-Selection

Require: A sporadic task T_s not in *Active* is released at time t, then it triggers the event A, whose utilization is u_s.

```
1: U = u_s;
2: u_max = u_s;
3: for i = 1 to |T_i ∈ Active = H_B ∪ H_C| do
4:     if T_i ∈ H_B then
5:         l' = T_i.key − t;
6:     if T_i ∈ H_C then
7:         l' = t_f − T_i.key;
8:     u_i = (l' × T_i.speed) / (t_f − t);
9:     U = U + u_i;
10:    if u_max < u_i then
11:        u_max = u_i;
12: return α = max{u_max, U/m};
```

下面首先给出一个引理,然后通过两个定理说明对事件 A 的处理不会引起 TL 面内其他任务丢失时限,同时可以保证 T_s 的时限。

引理 1　设 s 为 TL 面中任意时间,$s \in [t_0, t_f]$,且满足 $\forall T_i \in Active, R'_s \leqslant m$ 和 $r'_{i,s} \leqslant 1$。令 X 为 s 时刻在 $Active$ 中被调度执行的任意 x_s 个任务的集合。如果设 t 为从 s 时刻开始,下一个事件 B 或 C 发生的时刻,那么 $\forall \Delta, 0 \leqslant \Delta \leqslant t - s, R'_t \leqslant R'_{s+\Delta} \leqslant R'_s$ 且 $U_t \leqslant U_{s+\Delta} \leqslant U_s$。此外,如果 $R'_s < m$,那么 $R'_t < R'_{s+\Delta} < R'_s$;如果 $R'_s = m$,那么 $R'_t = R'_{s+\Delta} = R'_s$。

证明　因篇幅所限,具体证明省略,可参考文献[12]中定理 1。 □

定理 4　设 Γ 是在 m 个多处理器核上执行的偶发任务集,满足 $U \leqslant m$,且 $u_{\max} \leqslant 1$。DSREM 算法调度 Γ 的任意 TL 面设为 $[t_0, t_f]$,且每个 TL 面的初始时刻 t_0 均满足 $U_0 \leqslant m$,且 $u^0_{\max} \leqslant 1$。假设任务 T_s 是当前 TL 面 $[t_0, t_f]$ 内第 $s(s \geqslant 1)$ 个释放的任务,T_s 在初始时刻 t_0 不是活跃的,但在时刻 $t_s \in (t_0, t_f)$ 变成活跃的,则在 T_s 释放之前活跃任务集的总利用率 U_t 最多是 $m - u_s$,即 $\forall t \in [t_0, t_s), U_t \leqslant U_0 + \sum_{j=0}^{s-1} u_j \leqslant m - u_s$,其中 $\{u_j \mid j = 1, \cdots, s-1\}$ 表示当前 TL 面内第 j 个释放的任务,且 $u_0 = 0$。

证明　采用归纳法证明该定理。

初始情况:首先假设 T_s 是当前 TL 面 $[t_0, t_f]$ 内第一个释放的任务,释放时刻为 $t_s \in (t_0, t_f)$。因为任务 T_s 在初始时刻 t_0 不是活跃的,即 $T_s \notin Active, Active \cup \{T_s\} \subseteq \Gamma$,所以 $U_0 = \sum_{\forall T_i \in Active} u_i \leqslant \sum_{\forall T_i \in \Gamma} u_i - u_s \leqslant m - u_s$。又因为在 (t_0, t_s) 之间没有其他任务释放,所以只可能有事件 B、事件 C 以及事件 E 发生,或者没有任何事件发生,下面分三种情况判断:

(1)当事件 B 或事件 C 在时刻 $t \in (t_0, t_s)$ 发生时,根据引理 1 可知 $\forall \Delta, 0 \leqslant \Delta \leqslant t - t_0, U_t \leqslant U_{t_0+\Delta} \leqslant U_0$;

(2)当事件 E 在时刻 $t \in (t_0, t_s)$ 发生时,根据定理 3 可知 $U_t \leqslant U_{t_0+\Delta}$,这里 $\Delta = t - t_0$;

(3)当在 $t \in (t_0, t_s)$ 之间没有任何事件发生时,这说明在此期间 H_B 堆中所有任务均执行中,必然会在将来时刻发生事件 B 或 C,根据引理 1 仍可知 $\forall \Delta, 0 \leqslant \Delta \leqslant t - t_0, U_t \leqslant U_{t_0+\Delta} \leqslant U_0$;

综上可得 $\forall t \in [t_0, t_s), U_t \leqslant U_0 \leqslant m - u_s$。

归纳证明:假设 T_s 是当前 TL 面内第 $s(s \geqslant 1)$ 个释放的任务,释放时刻为 $t_s \in (t_0, t_f)$,满足题设结论即 $\forall t_1 \in [t_0, t_s), U_{t_1} \leqslant U_0 + \sum_{j=0}^{s-1} u_j \leqslant m - u_s$,其中 $\{u_j \mid j = 1, \cdots, s-1\}$ 表示当前 TL 面内第 j 个释放的任务,且 $u_0 = 0$。设 T_{s+1} 是在时刻 t_s 之后下一个释放的任务,释放时间为 $t_{s+1} \in (t_s, t_f)$。下面证明 t_{s+1} 时刻释放的任务 T_{s+1} 也满足题设结论即 $\forall t_2 \in [t_0, t_{s+1}), U_{t_2} \leqslant U_0 + \sum_{j=0}^{s} u_j \leqslant m - u_{s+1}$。

由题设可知当前 TL 面内有 $s+1$ 个任务 T_j 在初始时刻 t_0 不是活跃的,即 $\forall j = 1, \cdots, s, s+1$, $T_j \notin Active$,且 $Active \cup \{T_j\} \subseteq \Gamma$,所以此时 $U_0 = \sum_{\forall T_i \in Active} u_i \leqslant \sum_{\forall T_i \in \Gamma} u_i - \sum_{j=0}^{s+1} u_j \leqslant m - u_s - u_{s+1}$。根据归纳假设可得 $\forall t_2 \in [t_0, t_s), U_{t_2} \leqslant U_0 + \sum_{j=0}^{s-1} u_j \leqslant U_0 + \sum_{j=0}^{s} u_j \leqslant \sum_{\forall T_i \in \Gamma} u_i - u_{s+1} \leqslant m - u_{s+1}$。因为任务 T_s 释放之后,增加了当前活跃任务集中任务个数,所以 $U_{t_s} = \sum_{\forall T_i \in Active \cap \{T_s\}} u_{i,t_s} = \sum_{\forall T_i \in Active} u_{i,t_s} + u_s \leqslant U_{t_1} + u_s \leqslant U_0 + \sum_{j=0}^{s} u_j \leqslant m - u_{s+1}$。又因为在 $\forall t_2 \in (t_s, t_{s+1})$ 之间没有其他任务释放,所以只可能有事件 B、事件 C 和事件 E 发生。同理,根据引理 1 和定理 3 可得,$\forall \Delta, 0 \leqslant \Delta \leqslant t_2 - t_s, U_{t_2} \leqslant U_{t_s+\Delta} \leqslant U_{t_s}$。综上 $\forall t_2 \in [t_0, t_{s+1}), U_{t_2} \leqslant U_0 + \sum_{j=0}^{s} u_j \leqslant m - u_{s+1}$。 □

定理 5　设 Γ 是在 m 个多处理器核上执行的偶发任务集,满足 $U \leqslant m$,且 $u_{\max} \leqslant 1$。DSREM 算法调度 Γ 的任意 TL 面设为 $[t_0, t_f]$,且每个 TL 面的初始时刻 t_0 均满足 $U_0 \leqslant m$,且 $u^0_{\max} \leqslant 1$。假设任务 T_s 是当前 TL 面 $[t_0, t_f]$ 内第 $s(s \geqslant 1)$ 个释放的任务,T_s 在初始时刻 t_0 不是活跃的,但在时刻 $t_s \in (t_0, t_f)$ 变成活跃的,此时 t_s 为下一事件 A 发生时刻,设在任务 T_s 释放时原有的频率调节因子为 α。如果 $\forall t \in [t_0, t_s)$,满足 $\forall \Delta, 0 \leqslant \Delta \leqslant t_s - t$, $U_{t+\Delta} \leqslant m, R'_{t+\Delta} \leqslant m$,那么在时刻 t_s 设置 $l_{s,t_s} = u_s \cdot (t_f - t_s)$ 并通过算法 3 重新确定频率调节因子为 α_{t_s} 之后,$U_{t_s} \leqslant m, r'_{i,t_s} \leqslant 1$ 以及 $R'_{t_s} \leqslant m$。

证明　由题设已知任务 T_s 是当前 TL 面 $[t_0, t_f]$ 内第 $s(s \geqslant 1)$ 个释放的任务,T_s 在初始时刻 t_0 不是活跃的,但在时刻 $t_s \in (t_0, t_f)$ 变成活跃的,所以根据定理 4 可知在 T_s 释放之前活跃任务集的总利用率 U_t 最多是 $m - u_s$,即对于 $t \in [t_0, t_s)$, $\forall \Delta, 0 \leqslant \Delta \leqslant t_s - t$, $U_{t+\Delta} = \sum_{\forall T_i \in H_B \cap H_C} u_{i,t+\Delta} \leqslant U_0 + \sum_{j=0}^{s-1} u_j \leqslant m - u_s \leqslant m$,这里 $R'_{t+\Delta} \leqslant m$ $\{u_j \mid j = 1,$

…, $s-1$} 表示当前 TL 面内第 j 个释放的任务, $u_0 = 0$。因此,当任务 T_s 在时刻 t_s 释放时,需要将 T_s 加入 *Active* 中,设置 $l_{s,t_s} = u_s \cdot (t_f - t_s)$,通过算法 3 可得 $\forall T_i \in H_B \cup H_C, u_{i,t_s} = \frac{l'_{i,t_s} \cdot \alpha}{t_f - t_s}, U_{t_s} = \sum_{\forall T_i \in H_B \cap H_C} u_{i,t+\Delta} + u_s \leqslant U_0 + \sum_{j=0}^{s-1} u_j + u_s \leqslant m$。对于 $\forall T_i \in H_B \cup H_C$,因为 DSREM 算法保证 $t_s \leqslant T_i.key \leqslant t_f$,所以 $l'_{i,t_s} = (T_i.key - t_s) \text{or} (t_f - T_i.key) \leqslant (t_f - t_s), u_{i,t_s} \leqslant \alpha \leqslant 1$。又因为 $0 \leqslant u_s \leqslant 1$,从而 $u^{t_s}_{\max} = \max\{u_s, u_{i,t_s} \mid \forall T_i \in H_B \cup H_C\} \leqslant 1$。因此算法 3 重新确定频率调节因子为 $\alpha_{t_s} = \max\{u^{t_s}_{\max}, U_{t_s}/m\} \leqslant 1$,且 $r'_{i,t_s} = \frac{u_{i,t_s}}{\alpha_{t_s}} \leqslant \frac{u_{i,t_s}}{u^{t_s}_{\max}} \leqslant 1$ 和 $r'_{s,t_s} = \frac{u_s}{\alpha_{t_s}} \leqslant \frac{u_s}{u^{t_s}_{\max}} \leqslant 1$。因此,$R'_{t_s} = \sum_{\forall T_i \in H_B \cap H_C} r'_{i,t_s} + r'_{s,t_s} = \sum_{\forall T_i \in H_B \cap H_C} \frac{u_{i,t_s}}{\alpha_{t_s}} + \frac{u_s}{\alpha_{t_s}} = \frac{U_{t_s}}{\alpha_{t_s}} \leqslant m$。 □

由定理 5 的结论,再结合定理 1 和推论 1 易知如果在事件 A 发生之前任务集是满足局部可调度性的,那么 DSREM 算法对偶发任务释放事件 A 的处理,包括重新计算频率调节因子,仍然可以保证新任务 T_s 释放后不会引起其他任务丢失时限,从而满足当前 TL 面内所有任务的局部可调度性。

4 实 验

本文实现了多种节能实时调度算法,包括非节能的 LRE-TL 算法[12]、DCS 算法[5]以及 DSREM 算法。由于 Worst-Fit Decreasing(WFD)方法是最节能的启发式划分方法[5],实验中 DCS 算法采用 WFD 方法实现任务划分。本文以 LRE-TL 算法的能耗作为标准进行归一化,比较了这些算法的节能效果。能耗(Energy Consumption, EC)是指所有可调度的任务集的平均能耗值。

由于 DCS 算法是基于 EDF 实现多处理器任务调度[15],不能保证在系统高负载条件下的可调度性,而 DSREM 算法和 LRE-TL 算法具有偶发任务集的最优可调度性,所以单纯比较能耗并不能全面衡量算法性能,还需要比较可调度性。节能调度算法的可调度性越高且能耗越低,算法性能越高。

4.1 实验设计

为了具有可比性,本文采用与文献[5]相同的多核处理器模型(见本文第 1.1 小节),考虑电压和频率连续调节的理想情况,用以评价算法在调度偶发任务集时的能耗。仿真实验设置了下列参数:u_{sys}, m, $u_{\min}$ 和 $u_{\max}$,以及 AET/WCET。u_{sys} 表示系统利用率,m 表示处理器核个数,$u_{\min}$ 和 $u_{\max}$ 分别表示每个任务的最小和最大利用率。而 AET/WCET 表示任务平均执行时间(Average Execution Time, AET)与最坏情况执行时间(Worst-Case Execution Time, WCET)之比。

假设每个任务的时限等于最小释放间隔时间。系统利用率 u_{sys} 从 10% 开始增加到 100%,步长是 10%。处理器核数 m 分别设为 4、8 和 16。整个任务集的总负载即 Workloads = $u_{sys} \cdot m$。而每个任务利用率 u_i 在 $[u_{\min}, u_{\max}]$ 之间均匀分布。$[u_{\min}, u_{\max}]$ 设为 [0.1, 1.0],可以产生平均利用率为 0.5 的任务集。每个任务的最小释放间隔时间 P_i 在 [1,1000] ms 之间均匀分布。对于每个任务 T_i,一旦任务利用率 u_i 和最小释放间隔时间 P_i 确定,那么 T_i 的最坏情况执行时间 $C_i = u_i \times P_i$。AET/WCET 从 0.1 增加到 1,步长为 0.1,任务实际执行时间根据 AET/WCET 的值确定。对于每种参数设置,本文测试 100 000 组任务集。对于每个偶发任务集,模拟时间为 6 000 000 ms (100min),评价各种算法在该时间内的平均能耗值。

4.2 实验结果分析

图 3 显示了 LRE-TL、DSREM 以及 DCS 算法在不同参数设置下针对偶发任务模型的归一化能耗。从图 3(a)中可知,在处理器核数为 4 时,当任务集总负载 Workloads 超过某一个值后,DSREM 算法的节能效果始终优于 DCS 算法。同时,随着 Workloads 的增加,两者的性能差别会逐渐增大。这是因为 DCS 算法主要关注低负载情况下动态关闭处理器核以及回收动态松弛时间实现部分任务迁移,当 Workloads 增加时可关闭的处理器核减少,其节能效果必然会削弱。另一方面,DSREM 算法能够充分利用任务实例级迁移来实现处理器核之间负载均衡,因为它是基于最优偶发任务实时调度算法而提出的,在高负载情况下仍可以利用更多松弛时间实现节能,因此能够计算出更"真实"的动态松弛时间。当处理器核数为 8(图 3(b))和 16(图 3(c))时,可以得到与图 3(a)相一致的结论,这反映出在不同处理器核数条件下,当任务集总负载超过某一个值后,DSREM 算法的节能效果始终优于 DCS 算法,且在高负载情况下能耗节余稳定在 20% 左右。

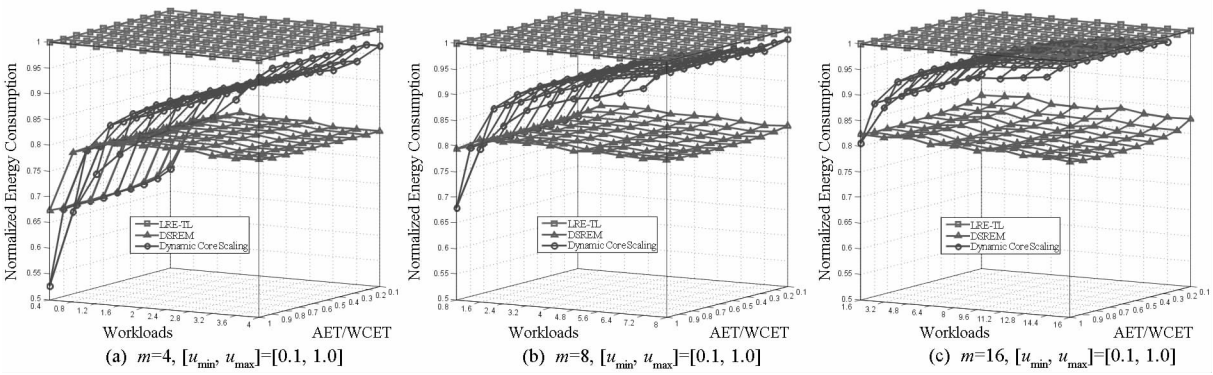

图 3　归一化能耗

Fig. 3　Normalized energy consumption

由图 3(a)、(b)和(c)可知,随着处理器核数的变化,虽然 DCS 算法和 DSREM 算法的节能效果都随着系统总负载的增加而有所降低,但是相比 DSREM 算法,DCS 算法的节能效果受到处理器核数变化的影响更大。DCS 算法在 4 个处理器核中归一化后能耗值的范围从 0.5 增加到 1.0,8 个处理器核时从 0.65 到 1.0,16 个处理器核时从 0.8 到 1.0。而 DSREM 算法在 4 个处理器核中平均能耗值的范围从 0.65 增加到 0.8,8 个处理器核时从 0.75 到 0.8,16 个处理器核时从 0.8 到 0.85。这是因为 DCS 算法主要关注周期任务,没有考虑到偶发任务具有任意的释放间隔时间,只能保守地假设每个偶发任务按照其最小释放间隔时间来释放,其节能效果必然随着偶发任务集总负载的增加而削弱。相比之下,DSREM 算法更适合系统总负载的动态变化,可以充分利用偶发任务提前完成而产生的动态松弛时间,降低处理器核执行频率,实现更多的能耗节余。

同时,在图 3(c)中还可以注意到,在处理器核数为 16 而任务集总负载为 16 的高负载情况下,DCS 算法的归一化能耗值不存在。这是因为 DCS 算法在高负载情况下通常很难保证可调度性,如图 4 所示。而节能实时调度只考虑在可调度任务集下的平均能耗值,当可调度性为 0 时平均能耗值不存在。

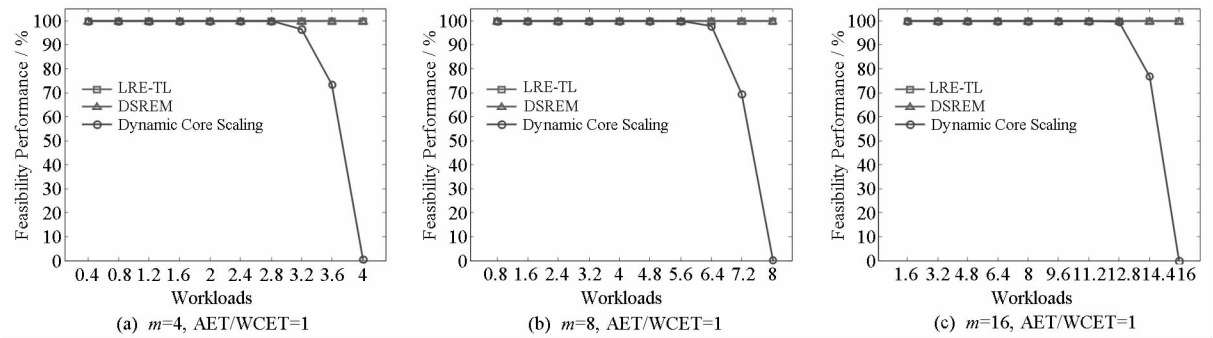

图 4　可调度性

Fig. 4　Feasibility performance

由于可调度性的测试通常基于任务的 WCET 进行分析,所以基于与图 3 相同的实验参数设置,图 4 显示了在 AET/WCET = 1 条件下算法的可调度性。从图 4(a)、(b)和(c)中可知,DSREM 算法在不同处理器核数设置下均能保证偶发任务集的最优可调度性,但 DCS 算法的可调度性却随着总负载的增加而不断降低,直至近似为 0。这是因为 DCS 算法是基于 WFD 启发式实现任务划分,任务调度仍基于 EDF 算法,不具有最优可调度性[15]。

对图 3 和图 4 统一比较可以全面衡量每个节能调度算法的性能。从图 3(a)和图 4(a)中可知:1)在处理器核数为 4 和 AET/WCET = 1 条件下,当总负载 Workloads 小于 2 时,DCS 算法不仅具有最优可调度性,而且能耗节余多于 DSREM 算法;2)但是当总负载 Workloads 超过 2 时,随着总负载的增加,DCS 算法的可调度性和能耗节余均不断减小,而 DSREM 算法是动态电压和频率

调节方法,不仅具有最优可调度性,而且能耗节余在高负载情况较稳定。另外,对图 3 和图 4 其他子图的分析可以得到与上述分析相一致的结论,这反映在不同处理器核数条件下,当总负载 Workloads 超过某一值后,DSREM 算法会超过 DCS 算法的性能,在高负载情况下可以获得更好的节能效果。

5 结束语

本文针对时限等于周期的偶发任务集,提出一种基于动态松弛时间回收的多核系统节能实时调度算法 DSREM。该算法基于最优在线调度算法 LRE-TL,利用 TL 面内节能实时调度思想,在每个 TL 面的提前完成时刻实现动态松弛时间回收,实现更多的节能,同时在每个 TL 面的初始时刻、偶发任务的释放时刻实现动态电压和频率调节,保证了偶发任务集最优可调度性。实验结果表明,DSREM 算法不仅保证了偶发任务集的最优可调度性,而且当任务集总负载超过某一个值后,其节能效果始终优于现有方法,且随着总负载的增加,能耗节余稳定在 20% 左右。

参 考 文 献:

[1] Chandrakasan A, Sheng S, Brodersen R. Low-power CMOS Digital Design[J]. IEEE Journal of Solid-state Circuit, 1992, 27(4):473 – 484.

[2] Rele S, Pande S, Onder S, et al. Optimizing Static Power Dissipation by Functional Units in Superscalar Processors [G]// Lecture Notes in Computer Science 2304, Grenoble, France, 2002: 85 – 100.

[3] Herbert S, Marculescu D. Analysis of Dynamic Voltage/ Frequency Scaling in Chip-multiprocessors[C]//Proc. of the Intl. Symp. on Low Power Electronics and Design (ISLPED), Portland, Oregon, USA, 2007:38 – 43.

[4] McGowen R, Poirier C A, Bostak C, et al. Power and Temperature Control on a 90nm Itanium Family Processor[J]. Journal of Solid-State Circuits, 2006.

[5] Seo E, Jeong J, Park S, et al. Energy Efficient Scheduling of Real-time Tasks on Multicore Processors[J]. IEEE Trans. on Parallel and Distributed Systems, 2008, 19 (11): 1540 – 1552.

[6] Yang C, Chen J J, Kuo T W. An Approximation Algorithm for Energy-efficient Scheduling on A Chip Multiprocessor[C]// Proc. of the Conf. on Design, Automation and Test in Europe, Munich, Germany, 2005.

[7] Devadas V, Aydin H. Coordinated Power Management of Periodic Real-time Tasks on Chip Multiprocessors [C]// Greencomp' 10 Proceedings of the International Green Computing Conference, Chicago, USA, 2010:61 – 72.

[8] Huang X, Li K, Li R. A Energy Efficient Scheduling Base on Dynamic Voltage and Frequency Scaling for Multi-core Embedded Real-time System [G]//ICA3PP 2009, LNCS 5574, Taipei, Taiwan, 2009:137 – 145.

[9] Bautista D, Sahuquillo J, Hassan H, et al. A Simple Power-aware Scheduling for Multicore Systems When Running Real-time Applications [C]//Proc of International Parallel and Distributed Processing Symposium, Florida, USA, 2008.

[10] Burd T D, Brodersen R W. Energy Efficient CMOS Microprocessor Design[C]//Proc. 28th Hawaii Int'l Conf. System Sciences, Hawaii, USA, 1995:288 – 297.

[11] Jerjurikar R, Pereira C, Gupta R. Leakage Aware Dynamic Voltage Scaling for Real-time Embedded Systems[C]//Proc. 41st Ann. Technical Conf. Design Automation (DAC'04), San Diego, CA, USA, 2004:275 – 280.

[12] Funk S, Nadadur V. LRE-TL: An Optimal Multiprocessor Algorithm for Sporadic Task Sets[C]// Conference on Real-time and Network Systems (RTNS), Paris, France, 2009: 159 – 168.

[13] Cho H, Ravindran B, Jensen E D. An Optimal Real-time Scheduling Algorithm for Multiprocessors[C]//Proceedings the 27th IEEE Real-time System Symposium (RTSS), Riode Janeiro, Brazil, 2006:101 – 110.

[14] Wegener J, Mueller F. A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints[J]. Real-time Systems, 2001, 21 (3): 241 – 268.

[15] Baker T P. An Analysis of EDF Schedulability on a Multiprocessor [J]. IEEE Transactions on Parallel and Distributed Systems, 2005, 16(8):760 – 768.