文章编号: 0253 2778(2006) 02 0219 06

# 一种新的静态优先级在线节能调度算法*

雷 霆, 胡 潇, 周学海

(中国科学技术大学计算机科学技术系, 安徽合肥 230027)

摘要: 合理运用动态电压调整技术可有效降低嵌入式实时系统能耗. 针对静态优先级实时调度, 提出了一种能够有效分析松弛时间并尽可能平衡分配松弛时间的在线节能调度算法 TPSRM. 设计了一种两段式频率执行策略来改变任务执行时间的分配, 能充分在线分析各种形式的松弛时间. 通过尽可能合理降低高优先级任务的处理器执行频率来实现有效的在线频率调整. 实验结果表明 TPSRM 算法可实现较好的节能效果.

关键词: 低功耗; 实时系统; 调度算法; 动态电压调整; 静态优先级

中图分类号: TP316 文献标识码: A

# A new energy efficient online scheduling algorithm for static priority real time systems

LEI Ting, HU Xiao, ZHOU Xue hai

(*Department of Computer Science and Technology, USTC, HeFei* 230027, *China*)

**Abstract**: Power is a valuable resource in embedded real time systems as the lifetime of many such systems is constrained by their battery capacity. Recent advances in processor design have added support for dynamic frequency/voltage scaling for saving power and energy. Static priority scheduling algorithms is widely used in real time systems and energy efficient scheduling algorithms for static priority real time systems are urgently needed to be designed. The limitations of energy efficient scheduling were discussed and a new energy efficient voltage scaling algorithm was proposed based on the rate monotonic algorithms. The algorithm can analyze slack time more effectively and try to balance the distribution of slack time among tasks of different priorities. A two phase frequency scaling strategy was designed in order to change the execution time of real time tasks. The proposed algorithm tried to lower the frequency of real time tasks of higher priority by analyzing all the slack times. Experimental results demonstrate that this algorithm can save up to 26.2% more energy than the DPM algorithm.

**Key words**: low power; real time system; scheduling algorithm; dynamic voltage scaling; static priority scheduling

* 收稿日期: 2004 08 10; 修回日期: 2005 02 25
基金项目: 国家自然科学基金(60273042)资助.
作者简介: 雷霆, 男, 1977 年生, 博士生. 研究方向: 计算机体系结构, 实时低功耗系统. E mail: tlei@mail.ustc.edu.cn
通讯作者: 周学海, 教授. E mail: xhzhou@ustc.edu.cn

## 0 引言

动态电压调整(dynamic voltage scaling, DVS)技术是通过降低处理器的电压和频率来降低功耗,可以有效运用于实时低功耗系统设计中. 节能调度算法是在已有实时调度算法中加入电压/频率调整策略,在满足实时性要求的前提下,分析并使用系统中的松弛(slack)时间,通过合理的电压/频率调整来降低系统能耗(以下把电压/频率调整统称为频率调整).

静态优先级调度算法实现简单,调度开销较低,系统过载时可预测性好,有着较为广泛的实际应用,因此,设计高效节能的静态优先级节能调度算法就显得尤为重要. Shin 等[1,2] 最早提出了静态优先级在线节能调度算法. Pillai 等[3] 针对动态优先级调度,提出了 CCEDF 节能算法,并且又提出节能效果更佳的 LAEDF 节能算法,但未改进所提出的 CCRM 静态优先级节能调度算法. Saewong 等[4] 针对静态优先级调度提出了 PM 算法和 DPM 算法,其中,PM 算法能够较充分地利用静态松弛时间,DPM 算法则进一步使用了基于优先级的松弛时间在线分析策略. 由于静态优先级调度与动态优先级调度相比,可调度利用率较低并且缺乏可调度性判定的简单充要条件[5],因而,设计高效节能的静态优先级节能调度算法更为困难,并且任务优先级在运行中始终保持不变会给在线频率调整带来不利影响.

静态优先级在线节能调度需要能够对松弛时间更有效地分析,并且要在不同优先级实时任务之间进行合理分配. 本文提出了一种能够有效分析并且尽可能平衡分配松弛时间的在线节能调度算法 TPSRM(two phase scaling rate monotonic). 该算法对系统运行中的松弛时间进行了巧妙而有效的分析和评估,实现对不同优先级的实时任务进行合理有效的在线频率调整,有效地降低了系统能耗.

## 1 问题描述

实时任务集 $\Gamma=\{\tau_i \mid 1\leqslant i\leqslant n\}$ 为周期性独立的可抢占任务集合. 任务 $\tau_i$ 用三元组 $(T_i, C_i, D_i)$ 表示,其中,$T_i$ 是执行周期,$C_i$ 是任务的最坏执行时间,$D_i$ 表示任务的时限,有 $T_i=D_i$. $\tau_i$ 的系统利用率表示为 $U_i$,有 $U_i=C_i/T_i$,任务集利用率为所有任务利用率的总和. 任务调度采用速率单调(rate monotonic, RM)调度算法[6]. RM 算法中,周期越短的任务优先级越高. 针对本文中的任务集 $\Gamma$,设定 $\tau_1$、$\tau_2$、…、$\tau_n$ 按照优先级由高到低的顺序排列.

节能调度是在原有调度算法中加入频率调整策略,不改变实时任务的优先级,通过合理的频率调整来降低系统能耗,并且需要保证实时系统可调度不变. 算法需要分析和利用系统中的松弛时间进行频率调整,松弛时间可分为静态松弛时间和动态松弛时间两种类型. 如果实时任务集的利用率和系统可调度性判定的利用率存在一定差异,则可以利用静态松弛时间进行频率调整. 实时任务的实际执行时间往往会小于最坏执行时间,故可以进一步利用动态松弛时间进行频率调整.

处理器功耗可表示为 $P\propto V^2 f$,其中,$V$ 是供电电压,$f$ 是运行频率. 电压和频率可近似认为是线性关系($V\propto f$),因此有 $P\propto f^3$,即功耗和频率三次方为线性关系,可表示为 $P=kf^3$[4],$k$ 为系统常数. 能耗计算公式为 $E=PT$,$T$ 为任务执行时间. 故有 $T=C/f$,$C$ 为周期数. 我们用能耗优化率来表示节能调度算法实现能耗节省的百分比.

## 2 TPSRM 算法的基本思路

从能耗最优的角度分析一般静态情况下的频率调整策略,可发现,只有各个任务的执行频率保持一致时,才能实现最佳的节能效果. 但任务的实际执行时间是无法预知的,并且还存在时限要求. 因此,需要从两个方面来考虑如何提高在线节能调度算法的节能效果. 一方面,算法是否充分地实现了对松弛时间的分析和尽可能地降低了处理器频率,表现为是否尽可能地延长了实时任务的处理器相对执行时间;另一方面,算法是否能够实现松弛时间的合理在线分配,表现为不同优先级任务的处理器执行频率是否能够尽量保持一致.

DPM 算法在 PM 算法的基础上,对在线分析系统中的动态松弛时间,采用了基于任务优先级的松弛时间分析策略,即高优先级任务的动态松弛时间可以被低优先级任务所利用. 但不同优先级实时任务会获得不同数量的动态松弛时间,最终不同优先级实时任务的执行频率存在较大差距,表现为高优先级任务执行频率偏高,低优先级任务执行频率偏低.

为设计较好的静态优先级在线节能调度算法,不仅需要能够有效分析松弛时间,使得高优先级任务能够使用更多松弛时间,同时也要尽可能实现松

弛时间的平衡分配,也不能过分减少低优先级任务可使用的松弛时间,以保持不同优先级任务的频率尽量趋于一致. 为此,本文利用动态松弛时间的产生特点,通过猜测任务的实际执行时间会低于最坏执行时间的情形[3,4,8],来进行合理的频率调整.

TPSRM 算法使用一种两段式频率执行策略,巧妙利用频率调整来改变任务执行时间的分配. 两段式频率执行策略是把一段执行过程 $R$ 划分为两个相邻部分 $R_A$ 和 $R_B$,通过升高 $R_B$ 部分的处理器频率来降低 $R_A$ 部分的处理器频率. 这相当于实现了松弛时间在 $R_A$ 和 $R_B$ 之间的移动,同时仍然能够保证执行过程 $R$ 的总执行时间不变. 那么在静态优先级在线节能调度中,虽然 $R_B$ 部分的处理器频率被升高,但由于动态松弛时间的存在,从平均概率的角度来看,$R_B$ 部分仍然能够以较低的频率执行或者 $R_B$ 部分的实际执行时间较少,这样可降低执行过程 $R$ 的整体平均能耗.

TPSRM 算法依次在线分析 SH、SI、SL 和 SS 等四种类型的松弛时间:

SH 是分析高优先级任务执行情况而得到的松弛时间. 依据高优先级任务的实际执行时间和最坏执行时间,按照基于优先级的松弛时间使用策略,可计算得到 SH.

SI 是在下一个任务到达时刻(next arrive time, NTA)之前的空闲时间. 由于 RM 算法无法完全利用处理器资源,因而在设定实时任务集静态执行频率后,仍然可能在 NTA 之前存在空闲时间 SI.

SL 是通过改变在 NTA 之前的其他低优先级就绪任务的执行频率而得到的松弛时间. 由于所有任务均存在静态执行频率,如果将低优先级就绪任务的静态执行频率提高到最高频率 $f_{max}$,则可以减少低优先级就绪任务的处理器相对执行时间,所节省时间用于降低当前高优先级任务的执行频率. 这里是把 NTA 之前的执行过程划分为两段式,把当前任务视为 $R_A$,把 NTA 之前的其他低优先级就绪任务视为 $R_B$,从而可计算 SL.

SS 是通过改变任务内部的处理器频率执行方式而得到的松弛时间. 把任务的执行划分为 $T_A$ 和 $T_B$,可提高 $T_B$ 部分的执行频率到最高频率 $f_{max}$,得到SS,从而可降低 $T_A$ 部分的处理器频率. 通过分析任务实际执行时间的概率分布,合理设定 $T_B$ 部分可以保证多数情况下能够有效降低能耗. 为实现 SS 的分析,我们借鉴随机单任务节能调度方法的设计思路[7],采用简单的启发式策略来设定任务内的 $T_A$ 和 $T_B$ 部分. 可得到概率值 $\xi$,取 $P(X\leqslant\xi)$ 部分为 $T_A$,剩余部分为 $T_B$,从而可实现对任务内松弛时间 SS 的分析. 如果使用反馈机制来动态预测任务的实际执行时间并作为 $T_A$ 部分,则可以进一步提高 TPSRM 算法的节能效果,但这需要增加一定的调度开销.

使用两段式频率有可能会过度降低高优先级任务的执行频率,损害松弛时间的平衡分配,因此,TPSRM 算法通过设置频率调整下界来防止部分高优先级任务使用过多松弛时间,一旦任务频率被降低到频率调整下界以下,则限定频率调整的力度并停止对后续松弛时间的分析.

## 3　TPSRM 算法的详细介绍

当实时任务 $\tau_i$ 被调度执行时,需调用 TPSRM 算法设置任务 $\tau_i$ 的处理器执行频率.

算法 **3.1**　TPSRM 算法. 代码如下:

```
(1) //upon task completion(τ_i)
(2) //or task preemption(τ_i)
(3) If (no other ready task)        //Setting f_i^l
(4)    f_i^l = f_min
(5) Else
(6)    f_i^l = Max(f_i^s × AETR, f_min)
(7) EndIf
(8) //Analyze SH
(9) //τ_k is the latest completed or running task
(10) If (k < i)
(11)    If (T_k^R > T_k^E && HaveInterrupt(τ_k))
(12)      SH = SH + (T_k^R - T_k^E)/f_k^c + T_k^E/f_k^c - T_k^E
(13)      CancelInterrupt(τ_k)
(14) Else
(15)      SH = SH + T_k^R/f_k^c
(16) EndIF
(17)   f_i^s = T_i^R/((T_i^R/f_i^c) + SH)
(18) SH = 0
(19) If (f_i^s < f_i^l)
(20)      SH = T_i^R/f_i^c - T_i^R/f_i^l
(21)      f_i^c = f_i^l
(22)      RETURN
(23) EndIf
(24) Else
(25)   SH = 0
(26) EndIf
(27) //Analyze SI
```

(28) $T_i^R/f_i^c - \sum_{x\in\psi}(T_x^R/f_x^c)$
(29) $f_i^c = T_i^R/((T_i^R/f_i^c)+SI)$
(30) If ($f_i^c < f_i^l$)
(31) $f_i^c = f_i^l$
(32) RETURN
(33) EndIF
(34) //Analyze SL
(35) For each $\tau_j \in \psi$
(36) $SL_j = T_j^R/(1/f_j^c - 1/f_{max})$
(37) If (HaveInterrupt($\tau_j$))
(38) $SL_j = SL_j - (T_j^E/f_j^c - T_j^E)$
(39) CancelInterrupt($\tau_k$)
(40) EndIf
(41) If ($\tau_j$ exceed the NTA)
(42) $T_j^N = NTA - current\ time, - T_i^R/f_i^c - \sum_{x\in\psi x\neq j}(T_x^R/f_x^c)$
(43) $SL_j = Min(SL_j, T_j^N)$
(44) EndIf
(45) $f_i^c = T_i^R/((T_i^R/f_i^c)+SL_j)$
(46) If ($f_i^c < f_i^l$)
(47) $SL_j = T_j^R/(1/f_i^c - 1/f_i^l) - SL_j$
(48) $f_j^c = T_j^R/(T_j^R/f_j^c + SL_j)$
(49) $f_i^c = f_i^l$
(50) RETURN
(51) ElseIf ($\tau_j$ exceed the NTA)
(52) $f_j^c = T_j^R/((T_j^R/f_j^c)+SL_j)$
(53) Else
(54) $f_j^c = f_{max}$
(55) EndIf
(56) EndFor
(57) //Analyze SS
(58) If ($T_i^R > T_i^E$)
(59) $SS = T_i^E/f_i^c - T_i^E$
(60) EndIf
(61) $f_i^c = (T_i^R - T_i^E)/((T_i^R - T_i^E)/f_i^c + SS)$
(62) If ($f_i^c < f_i^l$)
(63) $f_i^c = f_i^l$
(64) $f_i^c = T_i^E/((T_i^R - T_i^E)/f_i^l - (T_i^R - T_i^E)/f_i^c) + T_i^E)$
(65) Else
(66) $f_j^c = f_{max}$
(67) EndIf
(68) //trigger after $(T_i^R - T_i^E)/f_i^c$ time units
(69) SetInterrupt($\tau_i$, $(T_i^R - T_i^E)/f_i^c$)

其中,算法中各参数含义分别为:

$f_i^s$ 是任务 $\tau_i$ 的静态频率. 使用 PM 算法得到 $f_i^s$ 值.

$f_i^c$ 是 $\tau_i$ 的当前频率. $\tau_i$ 释放时, $f_i^c$ 设置为 $f_i^s$. 注意到, $\tau_i$ 在未被调度执行之前, $f_i^c$ 就有可能不等于 $f_i^s$, 因为算法会提高部分低优先级就绪任务的频率, 或者 $\tau_i$ 被抢占后再次恢复运行.

$T_i^E$ 等于 $\tau_i$ 的最坏执行时间减去 $T_B$ 部分的处理器绝对执行时间, $T_i^A$ 是 $\tau_i$ 的当前剩余最坏执行时间减去 $T_i^E$ 部分.

当 $\tau_i$ 自身采用两段式频率执行策略时, $f_i^c$ 是 $T_i^A$ 部分的执行频率, $f_i^e$ 是 $T_i^E$ 部分的执行频率.

$f_i^l$ 是 $\tau_i$ 的频率调整下界. 算法通过设置频率调整下界来防止当执行两段式频率执行策略时, 可能会出现的松弛时间过度分配现象, 即高优先级任务使用了过多松弛时间.

从概率角度分析, 任务的实际执行时间最有可能等于任务的平均执行时间, 故依据任务的平均执行情况来计算 $f_i^l$, 有

$$f_i^l = f_i^s \times (EW_i/W_i)$$

这里, $EW_i$ 为 $\tau_i$ 的平均执行时间, $W_i$ 为 $\tau_i$ 的最坏执行时间.

$T_i^R$ 是任务的剩余最坏执行时间(以 $f_{max}$ 衡量). $T_i^R$ 的设置规则如下:

(Ⅰ) $\tau_i$ 释放时, $T_i^R$ 被设置为 $\tau_i$ 的最坏执行时间.

(Ⅱ) $T_i^R$ 在 $\tau_i$ 执行过程中依据 $f_i^s$ 和执行时间逐渐递减.

(Ⅲ) $\tau_i$ 执行结束后, 如果后续执行任务为低优先级任务, $T_i^R$ 可以被低优先级任务所使用, 否则不能被使用. 然后 $T_i^R$ 清零.

(Ⅳ) $\tau_i$ 执行结束后, 如果系统中无就绪任务需要继续执行, $T_i^R$ 依据 $f_i^c$ 和系统空闲时间不断递减直至为零. 当有新任务需要执行时, 按照规则(Ⅲ)使用和设置 $T_i^R$.

$T_i^N$ 是 $\tau_i$ 在 NTA 之前的执行时间. 在计算 $SL_i$ 时, 如果 $\tau_i$ 的执行时间跨越 NTA, 此时需要计算 $T_i^N$ 来分析可以安全使用的松弛时间.

算法按次序依次分析计算 SH、SI、SL 和 SS. 一旦 $f_i^c$ 调整到 $f_i^l$, 则松弛时间分析和频率计算过程提前结束.

算法的执行过程介绍如下:

(Ⅰ) 确定 $\tau_i$ 的频率调整下界 $f_i^l$. 此外, 在没有其他就绪任务等待执行的情况下, 可更充分地降低频率, 有 $f_i^l = f_{min}$. 在确定频率调整下界后, 就可以实现对松弛时间的合理使用.

(Ⅱ)分析并使用松弛时间 SH. 如果最近结束执行的任务 $\tau_k$ 的优先级高于 $\tau_i$, 则根据 $T_k^R$ 来计算新的松弛时间, 并和上次剩余 SH 值累加, 从而得到新的 SH. 否则 SH 置为零. 基于频率调整下界来控制 SH 的分配, 如果有剩余松弛时间, 则计算剩余 SH, 供下次使用, 并且频率计算过程结束.

(Ⅲ)分析和使用松弛时间 SI. 当使用 SH 后, 如果在 NTA 之前仍然存在空闲时间, 则把该部分空闲时间计算为 SI.

(Ⅳ)依据 $\tau_i$ 的当前执行结束时间, 按优先级从高到低的顺序, 访问将在 NTA 之前开始执行的低优先级就绪任务集合 $\psi$ 中的所有任务, 分析并使用松弛时间 SL 来调整频率, 在 $\tau_i$ 和其他任务之间实现两段式频率执行. $SL_j$ 的计算通过升高 $f_j^c$ 到 $f_{max}$ 而得到. 需考虑集合 $\psi$ 中的最高优先级任务自身执行被设置为两段式频率执行的情形. 此外, 还需考虑集合 $\psi$ 中的最低优先级任务 $\tau_j$ 的执行可能会跨越 NTA 的情形, 此时为保证实时性, $SL_j$ 的计算要更为小心. 在根据升高 $f_i^c$ 到 $f_{max}$ 计算得出 $SL_j$ 后, 还需要和 $\tau_j$ 在 NTA 之前的执行时间 $T_j^N$ 相比较, 取两者之中的最小值作为最终的 $SL_j$, 这样可以保证所有将在 NTA 之前执行结束的任务仍然能够在 NTA 之前执行结束. 如果 $SL_j$ 大于 $\tau_i$ 所需要的松弛时间, 则 $f_j^c$ 不需要设置为 $f_{max}$.

(Ⅴ)如果 $f_i^c$ 未被调整到 $f_i^l$, 则基于 $T_i^A$ 和 $T_i^E$ 实现任务内两段式频率执行方式, 通过分析计算 SS, 进一步降低 $f_i^c$, 此时需要设置相应的定时器.

我们通过对松弛时间的使用来说明 TPSRM 算法能够满足实时性要求.

(Ⅰ)由于 SH 的使用始终遵循任务优先级顺序, 因而满足实时性要求.

(Ⅱ)SI 是 NTA 之前的空闲时间, SI 的分析和使用没有影响任务集在 NTA 之前的调度结果. 故 SI 的使用仍然保证实时任务集可调度.

(Ⅲ)SL 的使用分两种情况具体讨论:

(ⅰ)若不存在跨越 NTA 的执行任务, 则通过保证将在 NTA 之前执行结束的任务仍然能够在 NTA 之前执行结束, 则 SL 的使用仍可保证实时任务集可调度.

(ⅱ)若低优先级就绪任务集合 $\psi$ 中的最低优先级就绪任务 $\tau_j$ 的执行跨越 NTA 边界, 为区分起见, 设当前 NTA 为 $NTA_{now}$, 下一个 NTA 为 $NTA_{next}$. 由算法 3.1 可知, 首先将在 NTA 之前执行结束的任务仍然能够在 NTA 之前执行结束, 进而算法 1 仍然保证 $\tau_j$ 的执行结束时间保持不变, 即未抢占能够在 $NTA_{next}$ 之前执行结束任务的处理器资源, 因此, 将在 $NTA_{next}$ 之前执行结束的任务仍可能够在 $NTA_{next}$ 之前执行结束. 以此递推, 有 SL 的使用仍然保证实时任务集可调度.

(Ⅳ)由于 SS 的使用只是把当前任务 $\tau_i$ 的执行过程划分成两段, 分别执行不同的处理器频率, 并没有改变 $\tau_i$ 的执行完成时刻. 故 SS 的使用仍然保证实时任务集可调度.

## 4　仿真实验

我们实现了一个分析节能调度算法的仿真设计环境. 实验中, 每个实时任务的执行周期在 20~100 ms 之间均匀分布, 最坏执行时间(WCET)在 0~20 ms 之间均匀分布. 实时任务的实际执行时间在最快执行时间(BCET)和最坏执行时间之间服从均匀分布. 离散频率的个数为 10 个, 有 $f_{max}=1$ 和 $f_{min}=0.1$. 实验通过改变任务个数、系统利用率 $U$ 和 BCET/WCET 比值来分析各种节能调度算法的平均性能. 实验比较了 TPSRM 和 DPM 这两种在线节能调度算法, 基准能耗结果采用了 PM 算法的能耗值.

图 1 给出了 DPM 算法和 TPSRM 算法在不同利用率和不同实际执行时间下的能耗值, 实时任务集包含 20 个实时任务, 计算在任务集公共调度周期内的能耗值. 图 1(a)到(d)分别给出了在不同实际执行时间分布情况下的能耗值, 每个子图都给出了在不同系统利用率下的能耗值, 从而全面比较了 DPM 算法和 TPSRM 算法的平均性能.

实验结果表明, 在不同的利用率和实际执行时间下, TPSRM 算法都能够比 DPM 算法节省更多能耗. 在图 1(a)中, TPSRM 算法的能耗优化率平均比 DPM 算法高出 22.3%, 在系统利用率等于 0.2 的情况下, TPSRM 算法的能耗优化率比 DPM 算法高出 26.2%.

从图 1 可知, 随着 BCET/WCET 比值的增加, 系统中的动态松弛时间逐步减少, 因此, DPM 和 TPSRM 算法的能耗优化率都逐渐降低. 在图 1(d)中, 当 BCET/WCET=1 时, 系统无动态松弛时间, 此时, DPM 算法的节能效果等同于 PM 算法. 而 TPSRM 算法通过对在 NTA 之前的松弛时间 SI 的分析, 能耗优化率平均能够达到 9.06%.

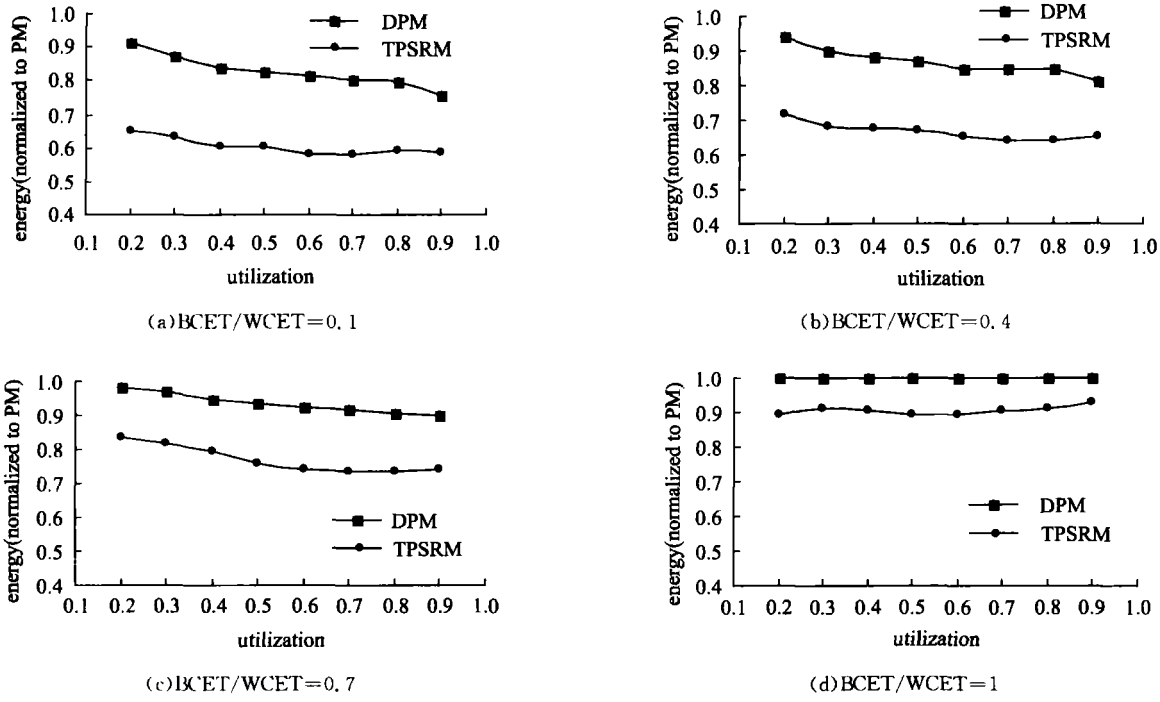


图 1 DPM 和 TPSRM 算法的节能效果

**Fig. 1** Energy consumption of DPM and BSDRM

我们还分别使用包含 10、30 和 50 等不同个数任务的任务集进行实验，分析任务个数对算法性能的影响，实验结果表明实时任务个数几乎不影响节能调度算法的节能效果. 另外，随着离散频率个数的减少，节能调度算法的能耗优化率逐渐降低.

## 5 结论

本文提出了一种新的静态优先级在线节能调度算法 TPSRM. 该算法使用两段式频率执行策略来改变任务执行时间的分配，分析了四种类型的松弛时间，能够对不同优先级的实时任务进行合理有效的处理器频率调整. 实验结果表明，TPSRM 算法实现了较高的能耗优化率.

参考文献(**References**)

[1] Shin Y, Choi K. Power conscious fixed priority scheduling for hard real time systems[C] // Proc. Design Automat. Conf., NY: ACM Press, 1999: 134 139.

[2] Shin Y, Choi K, Sakurai T. Power optimization of real time embedded systems on variable speed processors[C] // Proceedings of the International Conference on Computer Aided Design. NY: ACM Press, 2000: 365 368.

[3] Pillai P, Shin K G. Real time dynamic voltage scaling for low power embedded operating systems[C] // Proceedings of 18th ACM Symposium on Operating Systems Principles. NY: ACM Press, 2001: 89 102.

[4] Saewong S, Rajkumar R. Practical voltage scaling for fixed priority real time systems[C] // Proceedings of the 9th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'03), Washington: IEEE Press, 2003: 106 115.

[5] LIU W S. Real Time Systems[M]. NJ: Prentice Hall, 2000.

[6] LIU C L, Layland J W. Scheduling algorithms for multiprogramming in a hard real time environment[J]. Journal of ACM, 1973, 20(1): 174 189.

[7] YUAN W, Nahrstedt K. Energy efficient soft real time CPU scheduling for mobile multimedia systems [C] // Proc. of 19th Symposium on Operating Systems Principles. NY: ACM Press, 2003: 168 174.

[8] Aydin H, Melhem R, Mosse D, et al. Dynamic and aggressive scheduling techniques for power aware real time systems[C] // Proceedings of IEEE Real Time Systems Symposium. London: IEEE Press, 2001: 95 105.