

基于抢占阈值调度的动态电压分配算法

郝嘉磊^{1,2}, 王东辉¹, 张铁军¹, 王雷欧^{1,2}

(1 中国科学院 声学研究所, 北京 100190; 2 中国科学院大学, 北京 100049)

摘 要: 为了将实时操作系统应用于功耗敏感的嵌入式场景中, 提出一种基于抢占阈值调度的动态电压分配算法。仅使用动态电压调节会导致更多无用的任务抢占, 任务集可调度性下降。针对上述的问题, 将抢占阈值调度方法与动态电压调节结合, 在保证任务集可调度的前提下进一步降低处理器供电电压和运行频率。通过仿真, 验证了该算法在降低系统功耗方面的优势。

关键词: 抢占阈值; 动态电压调节; 调度算法; 实时操作系统

中图分类号: TP316

文献标识码: A

文章编号: 1000—7180(2016)07—0054—06

Dynamic Voltage Allocation Algorithm with Preemption Threshold Scheduling

HAO Jia-lei^{1,2}, WANG Dong-hui¹, ZHANG Tie-jun¹, WANG Lei-ou^{1,2}

(1 Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China;

2 University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In order to implement real-time operation system on power sensitive embedded systems, a dynamic voltage allocation algorithm with preemption threshold is proposed. The implementation of dynamic voltage scaling(DVS) will cause more useless scheduling and make the task set hard to be scheduled. To solve the problems mentioned above, DVS can be used with preemption threshold scheduling(PTS). The voltage and frequency of processor can be further reduced with schedulable task set. The simulation results verified the advantage in reducing system power of the proposed algorithm.

Key words: preemption threshold scheduling; dynamic voltage scaling; scheduling algorithm; real-time operation system

1 引言

实时操作系统在嵌入式领域的广泛使用, 保障了系统的稳定性, 降低了软件开发人员的开发难度。但是, 实时操作系统自身的开销也在一定程度上影响了它在功耗敏感型应用场景中的普及。

动态电压调节(DVS)技术是降低处理器功耗的有效方法^[1]。在各种 DVS 调度算法中, 核心思想是: 在能够满足任务时限的前提下, 尽可能地降低任务工作频率, 缩短空闲时间。但是, 在传统基于任务优先级的完全抢占调度算法中, 更长的工作时间也意味着将会有更多的任务抢占, 文献[2]指出, 在使

用 DVS 调度算法后, 相比于原有系统而言, 任务切换次数增加了近 400%。任务切换次数增加意味着更高的内存能耗、DVS 调节硬件功耗开销和任务切换的时间开销, 从而增加了处理器功耗。

抢占阈值调度算法(PTS)最早由文献[3]提出, 核心思想是: 每个任务有双优先级——任务优先级和抢占阈值优先级, 就绪态任务以任务优先级高低来竞争处理器使用权, 当任务占有处理器使用权后, 将自身优先级提高到抢占阈值优先级, 从而阻止其他任务优先级高于其任务优先级但是小于其抢占阈值优先级的任务的抢占, 减少了不必要的任务切换, 提高了系统的可调度性。文献[3]提出了基于已知任

务优先级,设置抢占阈值优先级的算法.文献[4]理论分析了抢占阈值分配的解集.

所以,将抢占阈值调度算法和 DVS 技术结合,可以有效地减少因为降低工作频率、增长任务执行时间而导致的过多的任务抢占.文献[5]提出了一种已知任务优先级,分配任务抢占阈值和任务工作频率的算法.文献[6]提出了基于抢占阈值调度算法,在线设置任务工作频率的算法.

本文提出一种基于确定任务优先级的抢占阈值与动态电压分配算法(FP-PTDVS).该算法针对任务优先级和时间相关参数确定的周期任务任务集,离线计算出使系统功耗最低的最优抢占阈值和工作频率分配方式.本文从理论上分析了算法的功耗最优特性,并且通过仿真实验对比验证了 FP-PTDVS 算法相比文献[5]提出的 EE-FPPT 算法和速率贪心 DVS 算法,在降低功耗和控制无用任务调度方面的改进.

2 基本概念

2.1 任务模型

本文研究的系统模型是硬实时操作系统,使用周期性任务模型,任务集合 $T = \{\tau_1, \tau_2, \dots, \tau_n\}$, τ_i 表示第 i 个任务.周期任务 τ_i 由 6 元组 $(C_i, T_i, D_i, F_i, \pi_i, \gamma_i)$ 表示.其中 C_i 表示任务执行需要的系统时钟周期数; T_i 表示任务相邻两个任务实例到来的时间间隔; D_i 表示任务的截止时限; F_i 表示任务的工作频率,我们定义任务工作频率在离散的频率点上 $F_i \in \{F_{CLK_1}, F_{CLK_2}, \dots, F_{CLK_s}\}$; π_i 表示任务的优先级; γ_i 表示任务的抢占阈值,其中 $\gamma_i \geq \pi_i$,并且声明 π_i 与 γ_i 均为正整数,数值越大表示其优先级越高.假定任务集合 T 中各个任务之间都是相互独立的,不存在共享资源,任务也不会在执行完成之前主动放弃处理器使用权.

2.2 处理器功耗模型

本文研究的系统动态充放电功耗模型为

$$P_d = \alpha C_f V_{dd}^2 f \quad (1)$$

式中, α 是电路翻转频度; C_f 是负载电容; V_{dd} 是供电电压; f 是工作频率.处理器供电电压 V_{dd} 与处理器工作频率 f 存在如下关系:

$$f = \kappa \frac{(V_{dd} - V_T)^\alpha}{V_{dd}} \quad (2)$$

式中, κ 与 α 均为工艺相关的常数; V_T 为门限电压.所以, DVS 技术的实现就是将处理器工作频率设定在几个离散的频率点上,然后硬件将处理器工作电

压 V_{dd} 降低至频率对应的最小允许值.在文中提到调整处理器供电电压或者工作频率,均指同时调整供电电压和工作频率.

2.3 可调度性判定

在文献[5]给出了基于 Level- i 忙周期分析方法,任务优先级和抢占阈值确定的任务集计算任务 τ_i 最长响应时间 R_i 的公式,如下:

$$B_i = \max_{\forall j, \gamma_j \geq \pi_i > \pi_j} C'_j \quad (3)$$

$$S_i^k = B_i + (k-1) \cdot C'_i +$$

$$\sum_{\forall j, \pi_j > \pi_i} (1 + \lceil \frac{S_i^k}{T_j} \rceil) \cdot C'_j \quad (4)$$

$$F_i^k = S_i^k + C'_i +$$

$$\sum_{\forall j, \pi_j > \gamma_i} (\lceil \frac{F_i^k}{T_j} \rceil - (1 + \lceil \frac{S_i^k}{T_j} \rceil)) \cdot C'_j \quad (5)$$

$$R_i = \max_{k=0,1,\dots,\lceil \frac{L_i}{T_i} \rceil} (F_i^k - k \cdot T_i) \quad (6)$$

式中, C'_i 是 τ_i 任务实例执行一次时间; B_i 是 τ_i 的阻塞时间; S_i^k 是 τ_i 第 k 个任务实例的开始时间; F_i^k 是 τ_i 第 k 个任务实例的结束时间; R_i 是任务 τ_i 的最大相对响应时间.

如果任务 τ_i 的 R_i 小于 D_i ,则定义 τ_i 是可调度的.如果 $\forall \tau_i, \tau_i \in T$ 是可调度的,则定义任务集 T 是可调度的.

式(3)~(6)定义了无 DVS 参与的抢占阈值调度下任务集 T 的可调度性判定方法.而对于支持 DVS 的处理器上,降低处理器的实际运行频率,会影响到 τ_i 的实际任务实例计算时间 C'_i .当 τ_i 任务实例执行时处理器工作频率已知,则可通过公式 $C'_i = \lceil \frac{C_i}{F_i} \rceil$ 计算出任务实例实际执行时间 C'_i ,进一步可利用式(3)~(6)进行可调度性分析.

2.4 研究问题定义

本文研究的问题是:给定任务集 T ,其 C, T, D, Π 参数已知,计算得到 F 与 Γ ,使在 T 可调度的前提下,功耗最小.

3 理论分析

将式(2)带入式(1)后可以得到处理器动态功耗 $P_d(f) \propto f^3$,也就是说 $P_d(f)$ 是关于参数 f 的凸函数.所以,在可压缩的空闲时间一定的前提下,整体降低任务的频率将会得到比最小化某些任务的频率更好地低功耗表现.

但是,由于现阶段处理器 DVS 仅提供离散的频率点,所以在任务集整体频率达到最低值时,可能存

在个别任务仍然有降低频率的空间. 这时, 所有可能的任务频率的解 F 之间是偏序关系, 而且偏序集 F 并不一定存在最小元. 如何在解集 F 找出使 P_d 最低的一个解, 将转化为动态规划求最优解的问题.

当考虑 DVS 硬件开销时, 系统功耗公式为

$$P = P_d + P_n \quad (7)$$

式中, P_n 为 DVS 调节硬件功耗开销. $P_n = k \cdot n$, 其中 k 为进行一次 DVS 调节的功耗, n 为调节次数.

定理 1 如果 (F, Γ_j) 与 (F, Γ_k) 均为 T 可调度的解, 且 $\Gamma_j > \Gamma_k$, 则 $P_n(F, \Gamma_j) \leq P_n(F, \Gamma_k)$.

证明: 不失一般性地, 假设 Π 递增, $\Gamma_j = \{\gamma_1, \gamma_2, \dots, \gamma_m^j, \dots, \gamma_n\}$, $\Gamma_k = \{\gamma_1, \gamma_2, \dots, \gamma_m^k, \dots, \gamma_n\}$ 且 $\gamma_m^j > \gamma_m^k$. 根据抢占阈值调度算法的定义, 在解 (F, Γ_j) 条件下, τ_m 仅有可能被 $\tau_i \in T_j$, $T_j = \{\tau_{\gamma_m^j} \dots \tau_n\}$ 的任务抢占, 设抢占次数为 n_j . 而在解 (F, Γ_k) 条件下, τ_m 可被 $\tau_i \in T_k$, $T_k = \{\tau_{\gamma_m^k} \dots \tau_n\}$ 的任务抢占, 设抢占次数为 n_k . 因为 $\gamma_m^j > \gamma_m^k$, 所以可得出 $T_j \subset T_k$, 进而得到 $n_j \leq n_k$. 也就是说, 在解 (F, Γ_j) 条件下, 将发生不多于在解 (F, Γ_k) 条件下的任务抢占次数, 所以 DVS 硬件开销更小. 综上 $P_n(F, \Gamma_j) \leq P_n(F, \Gamma_k)$. 证毕.

推论 1 在 F 参数相同情况下, 最大抢占阈值 (MPTA) 可以获得最小功耗.

证明: 因为 $\forall \Gamma_i \in \Gamma, \Gamma_{\text{MPTA}} > \Gamma_i$, 由定理 1 可知 $P_n(F, \Gamma_{\text{MPTA}}) \leq P_n(F, \Gamma_i)$, 也就是在 MPTA 条件下, P_n 最小. F 参数相同则 P_d 相同. 根据式 (7) 得出 $P(F, \Gamma_{\text{MPTA}})$ 最小. 证毕.

综上所述, 解决 2.4 节中提出的问题的过程可分为两步: 首先在抢占阈值调度下, 找到满足调度且使 P 最小的任务频率解 $F_{\min}$, 然后确定此时的抢占阈值解 Γ_i 是否为 $\Gamma_{\max}$, 如果不是则将任务抢占阈值设为 $\Gamma_{\max}$, 此时得到的 $P(F_{\min}, \Gamma_{\text{MPTA}})$ 即为处理器最小功耗.

4 FP-PTDVS 算法

4.1 算法基本思想

FP-PTDVS 算法主要分为三步.

(1) 设置所有任务为最高频率, 计算得到此时任务集的 MPTA.

(2) 使用二分查找方式, 整体降低所有任务频率到保持可调度性的最低值. 在降低频率时如果遇到不可调度情况, 尝试降低任务的抢占阈值, 若可恢复调度则得到新的 MPTA. 重复此步骤直到不可调度. 输出最后一次可调度的最低的频率和 MPTA.

(3) 使用动态规划方法, 进一步降低个别任务的

频率, 同时按照步骤 2 中方法调节 MPTA 以适应任务频率的降低, 最终计算出处理器功耗最优解以及此时任务抢占阈值和频率的分配方法.

4.2 子函数算法

由 4.1 节描述, 在 FP-PTDVS 算法中有一个关键且重复次数很多的子函数: 尝试降低任务频率并使任务集可调度. 此算法具体实现如算法 1.

算法 1: Slowdown Task And Make Schedulable()

输入: (T, i) 输出: (Γ)

/* 任务集已经按照优先级升序排序所有任务 */

```
begin
1   $f_i = f_i - 1$ 
2  for( $j = \gamma_i$ ;  $j > \pi_i$ ;  $j--$ ) do
3       $t = \text{Get Task By Priority}(j)$ 
4      if(Task Schedulability Test( $t$ )) then
5          continue
6      else
7           $\gamma_i = \pi_i - 1$ 
8          if(! Task Schedulability Test( $t$ )) then
9              return NULL
10         end if
11     end if
12 end for
13 for( $j = \pi_i$ ;  $j \geq 1$ ;  $j--$ ) do
14      $t = \text{Get Task By Priority}(j)$ 
15     if(Task Schedulability Test( $t$ )) then
16         continue
17     else
18         return NULL
19     end if
20 end for
21 return  $\Gamma$ 
end
```

算法 1 的输入参数为任务集 T 和希望降低频率的任务号 i , 如果降低频率后任务集仍可调度, 则输出抢占阈值解 Γ , 如果降低频率后, 无法通过修改抢占阈值使任务集可调度, 则输出 NULL.

算法 1 分三步进行: 第一步将 τ_i 的频率降低一级 (1 行). 第二步检测降低任务频率后对任务优先级处于被调整频率任务的任务优先级和抢占阈值之间的任务的可调度性 (2~12 行), 这一步从高任务优先级的任务向低任务优先级的任务逐个检测, 如果因为降低频率使这一部分任务不可调度 (7~10 行), 则尝试将被调整频率任务的抢占阈值降低到比不可调度的任务更低一级的等级上 (7 行) 后再检查原不可调度的任务的可调度性, 如果不可以调度则

返回 NULL. 第三步检测降低任务频率后对任务优先级低于被调整频率任务的任务优先级的任务和被调整频率任务本身的可调度性(13~20 行),如果不可调度则返回 NULL. 检测完所有任务集后,返回此时可调度的抢占阈值解 Γ .

4.3 子函数正确性证明

算法 1 修改了任务集 T 的第 i 个任务的频率,对任务集可调度性的影响可以分三部分分析,得到两条定理:

定理 2 若任务集 T 的第 i 个任务 τ_i 的任务优先级和抢占阈值分别为 π_i 和 γ_i ,且 $\gamma_i \geq \pi_i$. 改变 τ_i 的 f_i ,不会对 $\forall \tau_j \in T', T' = \{\tau_k \mid \pi_k > \gamma_i\}$ 的可调度性有影响.

证明:任务 τ_j 的可调度性是由最长响应时间 R_j 是否满足截止时限 D_j 决定的,而 R_j 由 B_j, S_j^k 和 F_j^k 决定. 对于任务优先级大于 γ_i 的任务 τ_j ,根据式(3)~(6), τ_i 的 C_i' 改变不会影响到 B_j, S_j^k 和 F_j^k ,也就是说 R_j 不会改变,则 τ_j 的可调度性也不会受到影响. 证毕.

定理 3 若任务集 T 的第 i 个任务 τ_i 的任务优先级和抢占阈值分别为 π_i 和 γ_i ,且 $\gamma_i \geq \pi_i$. 降低 τ_i 的 γ_i ,可能会使 $\forall \tau_j \in T', T' = \{\tau_k \mid \pi_k > \pi_i \wedge \pi_k < \gamma_i\}$ 的可调度性提高,不会降低.

证明:若 τ_i 的 γ_i 改变为 $\gamma_i', \gamma_i' < \gamma_i$,根据文献[3]的引理:任务 τ_i 的抢占阈值从 γ_1 变为 γ_2 ,仅仅优先级在 γ_1, γ_2 之间的任务的最长响应时间受到影响. 所以,仅需要分析任务优先级处于 γ_i' 和 γ_i 之间的任务的可调度性. 根据式(3)~(6),降低 τ_i 的抢占阈值,仅可能使 B_j 减小,不会使 B_j 增加, S_j^k 和 F_j^k 仅因为 B_j 的变化而变化,其他部分不受影响,也就是说 R_j 仅有可能减小,不会增加,故 τ_j 的可调度性只会提高,不会降低. 证毕.

分析算法 1 的正确性:根据定理 2,不需要考虑任务优先级大于 γ_i 的任务的可调度性. 根据定理 3,如果因为 f_i 的降低导致任务优先级在 π_i 和 γ_i 之间的任务不可调度,则尝试降低 γ_i 到满足提高该不可调度任务的可调度性的最大值(也就是该不可调度任务的任务优先级减一),且该操作不会导致已可调度的任务变为不可调度,如果该任务还是不可调度,则因为任务集不可调度而退出. 根据文献[3]的推论:任务 τ_i 的最长响应时间不受任务优先级大于 π_i 的任务的抢占阈值变化的影响. 也就是说,如果因为 f_i 的降低导致优先级小于 π_i 的任务不可调度,无论如何设置 γ_i 都不会使该不可调度的任务再可调度,

所以因为任务集不可调度而退出.

4.4 算法 MPTA 特性证明

定理 4 若算法 1 的输入参数 T 中抢占阈值为 MPTA,且经过算法 1 计算可以得到保证任务集可调度性的解 Γ ,则 Γ 仍为降低频率后的任务集的 MPTA.

证明:设原 MPTA 为 $\Gamma_{\max} = \{\gamma_1^{\max}, \gamma_2^{\max}, \dots, \gamma_i^{\max}, \dots, \gamma_n^{\max}\}$,算法 1 计算后, $\Gamma = \{\gamma_1^{\max}, \gamma_2^{\max}, \dots, \gamma_i, \dots, \gamma_n^{\max}\}$. 若在执行算法 1 的(2~21 行)时,所有待检测任务集均仍可保持调度,则 $\gamma_i = \gamma_i^{\max}$,所以 Γ 仍为 MPTA. 若有任务是通过修改 γ_i 才能够保证任务集的可调度性(7 行),因为在分析算法正确性时说明,每次对 γ_i 的调整都是设置其为满足提高可调度性的最大值,所以在每一次调整时,仍可保证 Γ 是 MPTA. 而且在逐个检测任务可调度性时,是从范围内最大任务优先级的任务向最小任务优先级的任务递减(2 行),所以在执行完(2~12 行)循环后,可以保证 Γ 是 MPTA. 证毕.

4.5 FP-PTDVS 算法

在 4.1 节对 FP-PTDVS 算法第三步的描述中,说明使用动态规划方法求解,可以将该问题套用 0-1 背包问题求解. 具体说明:计算所有可进一步降低频率的任务在降低频率后执行时间总和作为最长延时时间 $D_{\max}$,将可进一步降低频率的任务数作为阶数 k ,将系统超周期内平均功耗作为过程指标,将降低某任务频率后系统仍可调度且平均功耗更低作为接受条件. 算法具体实现如算法 2.

算法 2: FP-PTDVS()

输入: (T) 输出: $(F, \Gamma, P_{\min})$

begin

1 Order Tasks As Ascending Priority(T)

2 Set Taskset Frequency($T, f_{\max}$)

3 Set Taskset MPTA(T)

4 $f_{\text{low}} = f_{\min}$

5 $f_{\text{high}} = f_{\max}$

6 while($f_{\text{low}} \leq f_{\text{high}}$) do

7 $f_{\text{mid}} = \text{Get Middle Frequency}(f_{\text{low}}, f_{\text{high}})$

8 if(Try Taskset Frequency(T, f_{mid})) then

9 $f_{\max} = f_{\text{mid}}$

10 else

11 $f_{\min} = f_{\text{mid}}$

12 end if

13 end while

14 Set Taskset Frequency($T, f_{\min}$)

15 do

```

16   $f = \text{Get Taskset Lowest Frequency}(T)$ 
17   $T_{\text{sub}} = \text{Get Sub Taskset By Frequency}(T, f)$ 
18   $d_{\text{max}} = \text{Calculate Exe Time}(T_{\text{sub}}, c, f-1)$ 
19  for( $i=1$ ;  $i \leq T_{\text{sub}}.n$ ;  $i++$ ) do
20      task = Get Task From Taskset ( $T_{\text{sub}}, i$ )
21       $d = \text{Calculate Exe Time}(\text{task}, c, f-1)$ 
22       $\text{tab}[i][] = \text{tab}[i-1][]$ 
23      for( $j=d$ ;  $j \leq d_{\text{max}}$ ;  $j++$ ) do
24          if(Accept( $\text{tab}[i][j]$ ,  $\text{tab}[i-1][j-d]$ , task))
25              Do Accept();
26          end if
27      end for
28  end for
29   $T = \text{Find Lowest Power Taskset}(\text{tab})$ 
30  while(Get Tasks Num By Frequency( $T, -f$ ))
31       $P_{\text{min}} = \text{Cal Power Of Taskset}(T)$ 
32  return  $F, \Gamma, P_{\text{min}}$ 
end

```

FP-PTDVS 算法输入为给定任务优先级的任务集 T , 输出为系统最小功耗以及 F 与 Γ 分配方式. 算法分 5 步进行. 第一步, 将任务集按任务优先级升序排列(1 行). 第二步, 将任务集频率初始化为最高频率并且计算出保证任务集可调度的 MPTA 分配(2~3 行). 第三步, 通过二分法将任务集设置到能达到的最低频率(4~14 行). 第四步, 循环使用动态规划方法, 进一步降低部分能够降低频率的任务, 直到没有任务可以进一步降低频率(15~30 行). 第五步, 得到的任务集就是功耗最优的解, 计算返回平均功耗和此时 F, Γ 分配方式(31~32 行).

算法 FP-PTDVS 前两步给出最高频率、MPTA 分配的任务集(1~3 行). 在第三步尝试降低任务集频率时(8 行)会循环使用算法 1 尝试降低任务集频率, 无论是否可满足调度, 均不会实际执行修改任务集的操作. 当找到合适的最低频率时, 才设置任务集工作在此频率下(14 行), 且设置任务集频率也是循环使用算法 1 实现, 以保证设置完任务集频率后任务集仍可调度且抢占阈值依旧是 MPTA 分配. 第四步循环使用动态规划方法找满足功耗最小的频率分配, 单次循环的目的是: 获得所有处于现阶段任务集最低频率的任务构成的子任务集(16~17 行), 尝试降低子任务集中部分任务频率, 得到使系统整体功耗最低的任务频率分配方法(18~30 行). 在获得频率分配时, 使用类似于 0-1 背包算法的方式计算, 将子任务集全部任务在更低一级频率时的执行时间总和作为背包容量(18 行), 将子任务集中任务数作为物品数, 判定降低任务频率(24 行)的依据是: 调用

算法 1 检测在 $\text{tab}[i][j-d]$ 条件下降低任务频率后任务集可调度且降低频率后系统功耗比 $\text{tab}[i][j]$ 条件下系统功耗更低. 经过判定如果认为可以降低任务频率, 则调用算法 1 降低任务频率并且更新 $\text{tab}[i][j]$ 内容(25 行). 循环完成后, 从得到的表格中找到使系统工作在最低功耗下的任务集, 并更新任务集 T (29 行). 第五步, 计算第四步得到的任务集 T 的功耗并返回(31~32 行).

4.6 FP-PTDVS 算法正确性证明

由于在算法 FP-PTDVS 计算过程中, 首先设置 T 的为 MPTA, 而且后续始终调用算法 1 进行降低频率和调整抢占阈值的操作, 由定理 4 可知计算得到的抢占阈值分配方式一定是 MPTA 分配. 而且算法 FP-PTDVS 是通过多次循环计算, 直到没有任务可以降低频率为止, 而每次循环都使用动态规划方法保证得到的是使系统整体功耗最低的分配方式, 故算法得到的频率分配方式一定是离线计算下使系统功耗最低的频率分配. 综上, 算法 FP-PTDVS 得到的任务集抢占阈值和频率分配方式, 是使系统功耗最优的分配方式.

5 仿真与分析

5.1 评价指标

由式(1)~(2)可得到系统动态功耗与频率可以由 $P_d = kf^3$ 表示, 为简化计算, 在仿真时设 $k=1$, 将系统在超周期内平均功耗作为评价指标之一. 如果考虑 DVS 的硬件开销, 设系统是在任务切换时完成的 DVS 切换, 则开销与任务切换次数成正比关系, 将任务切换数作为评价指标之一.

5.2 测试任务集

本文仿真平台基于 SPAK 仿真平台, 对其增加了一些功能用于测试算法. 为了保证仿真测试能够覆盖更多的任务集可能性, 仿真测试随机产生了 1 900 个任务集合, 每个任务集合都包含 10 个任务, 任务优先级随机分配. 1 900 个任务集合按照处理器使用率分为 19 组, 每组处理器使用率从 5% 到 95% 以 5% 为步长递增, 每个任务集中任务周期是均值为 1 000 遵循正态分布的随机变量, 为简单考虑, 任务截止时限与任务周期相同, 并且保证所有任务集在完全抢占式调度下均是可调度的.

5.3 对比算法

仿真对比本文提出的 FP-PTDVS 算法和文献[5]提出的 EE-FPPT 算法、速率贪心 DVS 算法, 在不同处理器使用率下动态功耗和任务切换次数.

5.4 结果分析

图 1 和图 2 分别是三种算法在不同处理器使用率下平均功耗和任务调度次数对比图。

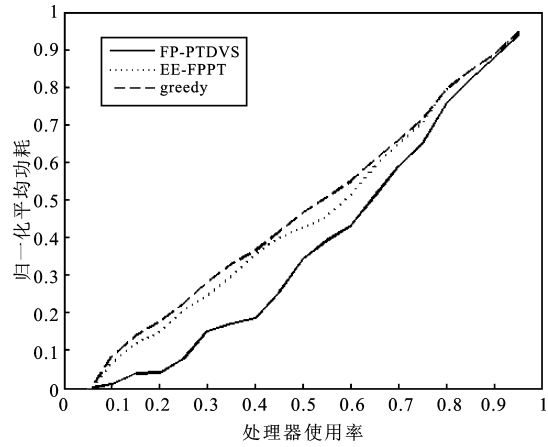


图 1 平均功耗对比图

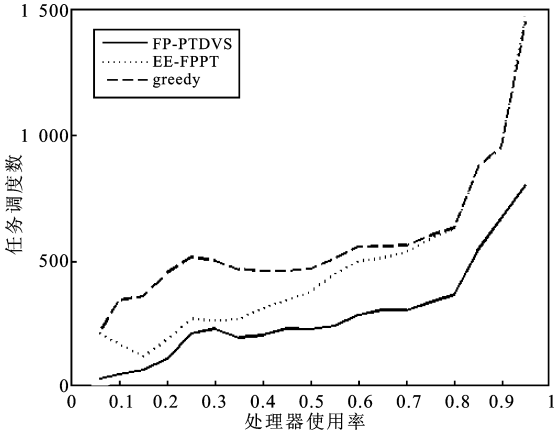


图 2 任务调度数对比图

从图 1 中可看出,FP-PTDVS 算法在动态功耗表现上是优于另外两种算法的. 另一方面,FP-PTDVS 算法积极调整抢占阈值以配合任务频率降低,并且保证抢占阈值是任务集可调度情况下最大抢占阈值分配,所以最大限度地减少了任务之间不必要的抢占,从图 2 可看出,FP-PTDVS 算法在任务调度数的控制上同样优于另外两种算法. 当处理器使用率进一步提高到最大值附近时,由于可压缩的空闲时间很小,几乎没法降低任何一个任务频率,所以三种算法功耗表现趋于一致,但是本文提出的 FP-PTDVS 算法在任务切换次数表现仍要优于另外两种算法,也就是说总体任务切换开销更小,功耗表现会更优。

6 结束语

本文从系统整体功耗角度出发,提出在支持 DVS 技术的处理器上,综合考虑任务抢占阈值和频率的 FP-PTDVS 算法. 理论证明了算法的功耗最优特性,仿真对比了 FP-PTDVS 算法与现有相关算法在不同处理器使用率下,功耗和任务调度数控制两方面表现的优势. 基于 FP-PTDVS 算法,可以将实时操作系统应用于更多功耗敏感的嵌入式工作场景中,以提高系统整体的可维护性和稳定性。

参考文献:

[1] Gang Q, Xiaobo H. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors [C] // Design Automation Conference, 2001. USA, Las Vegas, 2001: 828-833.

[2] Woonseok K. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems [C] // Real-Time and Embedded Technology and Applications Symposium, 2002. USA, San Jose, IEEE, 2002.

[3] Yun W, Saksena M. Scheduling fixed-priority tasks with preemption threshold [C] // Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on. China, Hongkong, 1999: 328-335.

[4] Chen J X, Harji A, Buhr P. Solution space for fixed-priority with preemption threshold [C] // Rtas 2005: 11th Ieee Real Time and Embedded Technology and Applications Symposium. USA, San Francisco, 2005: 385-394.

[5] 贺小川, 单芯片系统(SoC)中的实时任务调度算法研究[D]. 长沙:国防科学技术大学, 2009.

[6] Yang L, Lin M. On-line and off-line DVS for fixed priority with preemption threshold scheduling [C] // 2009 International Conference on Embedded Software and Systems. China, Hangzhou, 2009: 273-280.

作者简介:

郝嘉磊 男, (1989-), 硕士研究生. 研究方向为嵌入式系统. E-mail: bob-hjl@126.com.

王雷欧 男, (1983-), 博士研究生, 助理研究员. 研究方向为数字信号处理、嵌入式低功耗技术.

张铁军 男, (1974-), 博士研究生, 研究员. 研究方向为 VL-SI 信号处理、处理器体系结构.

王东辉 男, (1973-), 博士研究生, 研究员. 研究方向为信号与信息处理.