DVS系统硬实时周期任务动态调度算法

吴 琦，熊光泽，廖 勇

(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】与实时任务的可调度分析不同，实时DVS调度在保证任务截止时间限制同时，还要关注任务执行的处理器功耗。功耗研究一段时间的累积效果，传统基于最坏执行时间的任务调度模型不能满足实时DVS调度需要。该文采用实际执行时间(AET)概率分布建立实时任务调度模型，利用随机分析的方法分析AET分布对实时DVS调度算法性能的影响，提出了基于平均执行时间的实时DVS动态调度算法。实验结果表明，该算法在任务具有不同AET分布情况下可保持稳定的功耗性能。

关 键 词 动态电压调整; 感功计算; 功耗优化; 实时系统; 任务调度
中图分类号 TP316 文献标识码 A

Dynamic Scheduling Arithmetics for Hard Real-Time Period Tasks in Dynamic Voltage Scaling System

WU Qi，XIONG Guang-ze，LIAO Yong

(School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 610054)

Abstract Dynamic Voltage Scaling (DVS) has become a promising method for embedded real-time systems to reduce their power consumption. Unlike schedulability, energy consumption focus on the accumulative effect in a period of time. The traditional real-time scheduling model based on the worst execution time is not satisfied for the need of energy consumption analysis. In this paper the real-time task scheduling model is built according to the probability distribution of actual execution time, and the effect of the Actual Execute Time (AET) distribution on the energy consumption performance of real-time DVS scheduling algorithm is analyzed using stochastic method. The real-time DVS dynamic scheduling algorithm based on the average execution time is presented. The experimental results show that algorithm presented in this paper can hold stable power consumption performance under different AET distribution.

Key words dynamic voltage scaling; power-aware computing; power optimization; real-time systems; task scheduling

目前，嵌入式微处理器广泛采用动态电压调整(Dynamic Voltage Scaling，DVS) 技术，工作电压在运行时可动态调整。由于集成电路的动态功耗近似地与工作电压平方成正比[1]，DVS已成为嵌入式实时系统降低功耗、延长电池使用时间的有效技术手段。硬实时系统(特别是安全关键系统)中任务具有严格的截至时间要求，即使在最坏的情况下也必须保证任务在指定时间内完成。为了达到这个要求，设计实时系统时应按最坏情况进行负荷分配，既保证实时系统一般情况下能轻负荷运行，降低功耗，又保证实时系统不突破任务截至时间。

实时DVS任务调度(RTDVS)可分为静态(或离线)和动态(或在线)两类[2]。静态调度是依据任务周期、最坏执行时间(WCET)、截至时间等可事先掌握的信息，针对WCET与截至时间之间的静态松弛时间进行处理器速度调度。现实系统中，任务的实际执行时间(AET)是一个不确定量，不同任务具有不同的分布特性，有的接近于WCET，有的则比WCET小很多。动态调度就是在系统运行时根据实际松弛时间进行处理器速度调度。对于实时周期任务，有基于截至时间优先调度(EDF)的动态回收算法和投机算法、基于反馈技术的动态算法、基于整数线性规划的动态算法、基于单调速率调度(RMS)的动态算法，以及以空间换时间的半静态算法等[3-7]。以上调度算法多数采用WCET作为任务执行时间建立调度模型。采用WCET建模，在分析可调度性时是充

收稿日期：2005 – 10 – 20
基金项目：国家高技术研究发展计划；国家863计划资助项目(2003AA1Z2210)
作者简介：吴 琦(1969 –)，男，博士生，主要从事嵌入式实时系统的功耗管理技术方面的研究.

分而且必要的，因为可调度性要求在最坏情况下也要保证可调度。而功耗注重的是累计效果，仅仅采用WCET建模不能满足要求，需要考虑AET实际分布因素。本文采用AET分布建立实时任务调度模型，利用随机分析的方法讨论任务AET分布特性对RTDVS性能的影响，提出基于平均执行时间(ACET)的固定优先级实时DVS动态算法。

1　系统模型

1.1　实时DVS任务调度

DVS处理器的工作电压可以取区间 $[V_{\min},V_{\max}]$ 内的 N_v 个离散值，工作电压集合表示为：

$$V=\{V_{\mathrm{dd}\_i}\mid V_{\min}\le V_{\mathrm{dd}\_i}\le V_{\max}\}\qquad i=1,2,\cdots,N_v$$

工作电压决定处理器功率消耗和运行速度[1]，当工作电压为 V_{dd} 时，处理器动态功耗和工作频率分别为 $P_d(V_{\mathrm{dd}})=CaV_{\mathrm{dd}}{}^2 f$ 和 $f(V_{\mathrm{dd}})=k(V_{\mathrm{dd}}-V_t)^2/V_{\mathrm{dd}}$，其中 C 为输出电容；a 为门状态转变平均个数；k 为CMOS工艺相关常数；V_t 为阀值电压。

实时周期任务可用释放周期、截至时间和AET分布函数三元组描述为 $T_i=\{P_i,D_i,F_i(x)\}$。$F_i(x)$ 是指处理器最快处理速度时任务 T_i 的AET分布函数，根据 $F_i(x)$ 可以确定 T_i 的最佳执行时间 B_i、最坏执行时间 W_i 和平均执行时间 A_i。$T_{i,j}$ 表示 T_i 的第 j 实例；$X_{i,j}$ 表示 $T_{i,j}$ 的实际执行时间。系统中任务集合用 $T=\{T_i\mid i=1,2,\cdots,N\}$ 表示，N 为系统中的任务数。本文将任务按周期长短排序，T_1 具有最短周期。调度点包括任务释放和完成时刻，调度点集合用 $M=\{R_{i,j},E_{i,j}\mid T_i\in T\}$ 表示，其中 $R_{i,j}$ 和 $E_{i,j}$ 分别表示 $T_{i,j}$ 的释放和完成时刻。如果将 $|M|$ 个调度点按时间先后排序，在调度点 $M_i\in M$，$i=1,2,\cdots,|M|$，已过调度点的决策结果和执行时间构成调度历史，用 H_{i-1} 表示。算法根据 H_{i-1} 选择下一时间段处理器执行的任务和工作电压，用 $\delta_i\mid H_{i-1}=(\tau_i,v_i)\in T\times V$ 表示 M_i 的决策。整个决策序列构成一个算法 $\Delta=(\delta_1,\delta_2,\cdots,\delta_{|M|})$，所有可能的算法构成算法集合，用 D 表示。

为方便讨论，参照RMS算法，本文采用以下近似假设：(1) 忽略处理器工作电压切换的时间代价和能量代价；(2) 忽略调度算法本身的时间代价和能量代价；(3) 任务之间相互独立；(4) 周期任务截至时间等于释放周期；(5) 任务在0时刻同时释放；(6) 处理器空闲时以 $V_{\min}$ 运行Idle任务。

1.2　任务功耗

处理器最低功率为 $P_d(V_{\min})$，本文采用相对功率 $P_d(V_{\mathrm{dd}})-P_d(V_{\min})$ 计算任务功耗。对于 $T_{i,j}$，当处理器以 $v_{i,j}$ 电压工作时，则 $T_{i,j}$ 相对功耗的期望为：

$$E[\hat{g}(T_{i,j})]=\int_{B_i}^{W_i} x\frac{f(V_{\max})}{f(v_{i,j})}(P_d(v_{i,j})-P_d(V_{\min}))\mathrm{d}F_i(x)=$$

$$\frac{f(V_{\max})}{f(v_{i,j})}(P_d(v_{i,j})-P_d(V_{\min}))A_i=\phi(v_{i,j})A_i$$

$$\phi(v_{i,j})=\frac{f(V_{\max})}{f(v_{i,j})}(P_d(v_{i,j})-P_d(V_{\min}))\qquad(1)$$

式(1)是 $v_{i,j}$ 的单调增函数。单就 $T_{i,j}$ 而言，$v_{i,j}$ 越低功耗越小，但 $v_{i,j}$ 取值必须满足以下条件：

$$V_{\min}\le v_{i,j}\le V_{\max}$$

$$f(v_{i,j})/f(V_{\max})\le W_i/(D_i-u)$$

式中　u 表示 $T_{i,j}$ 等待调度的时间。从式中可以看出 $v_{i,j}$ 主要决定于松弛时间 D_i-W_i-u。松弛时间越长，$T_{i,j}$ 相对功耗越低。

当 $T_{i,j}$ 执行时有后续 $T_{k,l}$ 等待执行，则 $T_{i,j}$ 的执行时间影响 $T_{k,l}$ 的松弛时间，从而影响 $T_{k,l}$ 的调度。如用 $\hat{G}(T_{i,j})$ 表示 $T_{i,j}$ 和所有后续实例相对功耗之和的期望，用 $\hat{G}(T_{k,l}\mid X_{i,j}=x)$ 表示 $T_{i,j}$ 执行时间 $X_{i,j}=x$ 时紧接着调度执行 $T_{k,l}$ 的 $\hat{G}(T_{k,l})$，则有：

$$\hat{G}(T_{i,j})=\phi(v_{i,j})A_i+\int_{B_i}^{W_i}\hat{G}(T_{k,l}\mid X_{i,j}=x)\mathrm{d}F_i(x)\qquad(2)$$

当 $T_{i,j}$ 执行时被高优先级实例抢占，$T_{i,j}$ 执行中止等待高优先级实例完成后重新调度。可将 $T_{i,j}$ 分为两个实例计算：(1) 被抢占前，对应的AET分布为 $F_i'(x)=F_i(x)/F(t)$，$0\le x\le t$，其中 t 为被抢占时已执行部分折算为处理器最快处理速度的处理时间；(2) 被抢占后，对应的AET分布为 $F_i''(x)=(F_i(x)-F_i(t))/(1-F_i(t))$，$t\le x\le W_i$。

如此，对于给定任务集，已知实例释放时间，可计算系统在给定算法 S 下的相对功耗期望(用 $\hat{G}(S)$ 表示)，评价算法的功耗性能。完全计算系统运行期间的 $\hat{G}(S)$ 比较困难，对于周期任务集可仅计算周期最小公倍数范围内的实例。

1.3　问题的形式化描述

RTDVS问题是对给定任务集求得使总功耗最小的算法，同时保证满足每个任务截至时间，可用以下优化问题描述为：

$$\min_{\Delta\in D}\ \hat{G}(\Delta)$$

$$\text{s.t.}\ \ X_{i,j}\le D_i,\ j=1,2,\cdots\text{for}\ \forall T_i\in T$$

RTDVS问题是一个NP问题[8]，当已知每个实例的释放时间，静态(或离线)求解也许可以实现。但对于动态调度，要求算法本身的计算量很小，不能采用动态(或在线)求解，需要简单快速的近似算法。

2 RTDVS动态算法

RTDVS动态算法需要处理松弛时间计算和分配两个关键环节。松弛时间可分静态和动态两种：静态松弛时间是指系统设计时就可确定的WCET与截至时间之间的松弛时间，用 S_i 表示，根据可调度条件可以计算静态松弛时间；动态松弛时间是指系统运行时的松弛时间用 $s_i(t)$ 表示。为计算动态松弛时间系统需维护每个任务的剩余截至时间 $d_i(t)$、已完成部分 $x_i(t)$、最坏剩余执行时间 $w_i(t)$ 和下一实例释放时间 $n_i(t)$。$d_i(t)$、$x_i(t)$、$w_i(t)$ 和 $n_i(t)$ 的初值都为零。另外，系统还需保存当前执行任务 T_c（c 表示当前任务的任务号）、当前处理器工作电压 v、就绪任务集合 $T_{\mathrm{RD}}=\{T_i \mid T_i \in T, w_i(t)>0\}$、已完成任务集合 $T_{\mathrm{ED}}=T-T_{\mathrm{RD}}$，上次调度点时间为 m。

本文算法采用RMS优先级分配方法，RMS可调度条件为 $\forall T_i \in T, W_i+\sum_{j=1}^{i-1} W_j\left\lceil \frac{D_i}{D_j} \right\rceil \le D_i$，由此可计算任务 T_i 的静态松弛时间 $S_i=D_i-W_i-\sum_{j=1}^{i-1} W_j\left\lceil \frac{D_i}{D_j} \right\rceil$。

系统运行时在每个调度点动态计算任务的松弛时间。在任务(例如 T_i)释放调度点 t，T_i 的参数为 $d_i(t)=D_i$、$x_i(t)=0$、$w_i(t)=W_i$、$n_i(t)=P_i$ 和 $s_i(t)=S_i$。其他已完成任务 $T_j \in T_{\mathrm{ED}}$ 参数为 $d_j(t)=0$、$x_j(t)=0$、$w_j(t)=0$、$n_j(t)=n_j(m)-(t-m)$ 和 $s_j(t)=s_j(m)-(t-m)$。其他就绪任务 $T_j \in T_{\mathrm{RD}}-\{T_c\}$ 的参数 $n_j(t)$ 同上，$d_j(t)=d_j(m)-(t-m)$，$x_j(t)=x_j(m)$，$w_j(t)=w_j(m)$，$s_j(t)=s_j(m)-(t-m)\cdot(1-f(v)/f(V_{\max}))$。当前执行任务的参数 $n_c(t)$、$d_c(t)$、$n_c(t)$ 和 $s_c(t)$ 同上。

$$x_c(t)=x_c(m)+(t-m)f(v)/f(V_{\max})$$

$$w_c(t)=w_c(m)-(t-m)f(v)/f(V_{\max})$$

在任务(例如 T_i)完成调度点 t，T_i 的参数为 $d_i(t)=0$、$x_i(t)=x_i(m)+(t-m)f(v)/f(V_{\max})$、$w_i(t)=0$、$n_i(t)=n_i(m)-(t-m)$ 和 $s_i(t)=S_i+n_i(t)$。其他已完成任务 $T_j \in T_{\mathrm{ED}}$ 的参数为 $d_j(t)=0$、$x_j(t)=0$、$w_j(t)=0$、$n_j(t)=n_j(m)-(t-m)$ 和 $s_j(t)=s_j(m)-(t-m)$。其他就绪任务的 $T_j \in T_{\mathrm{RD}}-\{T_c\}$ 参数 $n_j(t)$ 同上，$d_j(t)=d_j(m)-(t-m)$，$x_j(t)=x_j(m)$，$w_j(t)=w_j(m)$，$s_j(t)=s_j(m)-(t-m)\cdot(1-f(v)/f(V_{\max}))+W_i-x_i(t)$。如此，可以计算所有实例的动态松弛时间。但当前可实际利用的松弛时间为 $s_{\mathrm{cur}}(t)=\min_{T_i \in T, i<c}(s_i(t))$，其对应的任务称为控制任务。

关于松弛时间分配，静态研究表明最优静态调度是保持单一执行速度。现有动态算法的松弛时间分配主要有两种方法[9]。(1) 以WCET为权平均分配的方法，当任务ACET接近于WCET时，能保证功耗近似最优化。但当任务ACET远小于WCET时，后面实例因前面实例提前完成可再降低执行速度，前面实例的执行速度偏高，不能保证执行速度稳定。(2) 与第一种方法相反，将松弛时间全部分配给当前实例，当任务ACET远小于WCET时，后面实例因前面实例提前完成也可降低执行速度，以保证功耗近似最优化。但当任务ACET接近于WCET时，因前面实例利用了大部分松弛时间，后面实例不得不以较高速度执行。如将式(2)右边后一项用 $\hat{G}(T_{k,l} \mid X_{i,j}=A_i)$ 近似，则 $\hat{G}(T_{i,j})$ 主要由 $T_{i,j}$ 及后面实例的ACET决定。基于以上分析，本文RTDVS算法依据ACET进行松弛时间分配。

假设控制任务为 T_j，则按ACET分配松弛时间 T_{cur} 的优化执行频率为：

$$f^{\mathrm{O}}=\frac{f(V_{\max})\left(\sum_{k=1}^{j} A_k\left\lfloor \frac{P_j}{P_k} \right\rfloor+w_k(t)\right)}{\left(\sum_{k=1}^{j} A_k\left\lfloor \frac{P_j}{P_k} \right\rfloor+w_k(t)\right)+s_{\mathrm{cur}}(t)}$$

其后任务在 $f(V_{\max})$ 频率下保证截止时间 $T_{i,j}$ 的最小执行频率 $f^{\mathrm{L}}=f(V_{\max})w_i(t)/(w_i(t)+s_{\mathrm{cur}}(t))$，确定 $T_{i,j}$ 实际执行频率为 $f^{\mathrm{D}}=\max(f^{\mathrm{O}}, f^{\mathrm{L}})$。

3 仿真实验

为分析算法性能，本文以Transmeta公司的Crusoe嵌入式处理器为例，利用Matlab进行仿真研究。Crusoe处理器工作电压为1.1～1.6 V，其功耗参数如表1所示。本文还仿真了已经发表的几个典型算法[8-9](如ccRMS、laRMS和静态调度)进行比较。

表1 处理器时钟频率与功耗的关系

频率/MHz	功率/w	比例/%	功率增量/W
600	6.00	100	4.60
566	5.00	83	3.60
533	4.20	70	2.80
500	3.55	59	2.15
466	3.00	50	1.60
433	2.55	43	1.15
400	2.20	37	0.80
366	1.90	32	0.50
333	1.70	28	0.30
300	1.55	26	0.15
266	1.40	24	0

本文为周期任务调度实验设置三个周期任务，

执行时间均匀分布表示为U。具体任务参数如表2所　　示。固定优先级调度按RMS方法分配优先级。

表2　周期任务调度实验的任务参数　　单位：s

	Test1	Test2	Test3	Test4	Test5	Test6
T_1 {D,F(x)}	{20,U(5,5)}	{20,U(0,5)}	{20,U(5,5)}	{20,U(5,5)}	{20,U(0,5)}	{20,U(0,5)}
T_2 (D,F(x))	{40,U(10,10)}	{40,U(0,10)}	{40,U(0,10)}	{40,U(10,10)}	{40,U(0,10)}	{40,U(10,10)}
T_3 (D,F(x))	{80,U(10,10)}	{80,U(0,10)}	{80,U(0,10)}	{80,U(0,10)}	{80,U(10,10)}	{80,U(10,10)}

	Test7	Test8	Test9	Test10	Test11	Test12
T_1 {D,F(x)}	{20,U(5,5)}	{20,U(0,5)}	{20,U(5,5)}	{20,U(5,5)}	{20,U(0,5)}	{20,U(0,5)}
T_2 (D,F(x))	{40,U(10,10)}	{40,U(0,10)}	{40,U(0,10)}	{40,U(10,10)}	{40,U(0,10)}	{40,U(10,10)}
T_3 (D,F(x))	{60,U(10,10)}	{60,U(0,10)}	{60,U(0,10)}	{60,U(0,10)}	{60,U(10,10)}	{60,U(10,10)}

周期任务的调度实验结果如图1所示。图中算法功耗相对于静态调度归一化。通过实验结果可以看出动态调度功耗与任务AET分布有关，A_i/W_i 越小，算法功耗越小，当 A_i/W_i 等于1时最优调度为静态调度。LaRMS算法由于当前实例独占松弛时间，当ACET远小于WCET时可获得最好效果，如Test2、Test5、Test8和Test10；但当ACET接近于WCET时其效果反比静态调度还差，如Test1、Test4和Test7。相反，ccRMS算法用WCET进行松弛时间分配，在任务ACET接近于WCET时可以获得很好效果，但当ACET远小于WCET时，效果不佳。baRMS算法按照ACET进行松弛时间分配，可克服以上算法的缺点，在各种情况下都可以获得很好的效果。

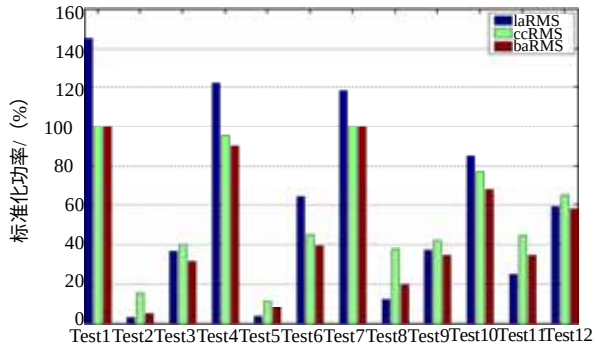

图1　周期任务固定优先级DVS调度算法功耗性能比较

实验中还发现先执行ACET远小于WCET任务，比先执行ACET接近于WCET可以获得更好节能效果。因此在按照RMS方法分配优先级时，对于相同周期的任务应该按照ACET分配优先级。

4　结 束 语

研究功耗的调度模型需要能描述任务的AET分布特性，只有考虑AET分布因素的调度算法才能实现在不同AET分布下保持稳定的功耗性能。本文提出的基于ACET的实时DVS动态算法可满足不同AET分布下功耗优化要求。本文算法是基于RMS固定优先级调度算法，但主要理念也适用于其他算法。

参　考　文　献

[1] WOLF W. Modern VLSI design[M]. 3rd ed. New Jersey: Prentice Hall Modern Semiconductor Design Series, 2002.

[2] ERNST R, YE W. Embedded program timing analysis based on path clustering and architecture classification[C]//In: ICCAD 97. San Jose: IEEE CS Press, 1997.

[3] AYDIN H, MELHEM R, MOSSE D, et al. Power-aware scheduling for periodic real-time tasks[J]. IEEE Trans. on Computers, 2004, 53(5): 584 - 600.

[4] ZHU Y, MUELLER F. Feedback EDF scheduling exploiting hardware-assisted asynchronous dynamic voltage scaling[J]. ACM SIGPLAN Notices, 2005, 40(7): 203 -212.

[5] SWAMINATHAN V, CHAKRABARTY K. Investigating the effect of voltage-switching on low-energy task scheduling in hard real-time systems[C]//In: ASP-DAC 2001. Yokohama: IEEE/ACM Press, 2001.

[6] KIM W, KIM J, MIN S L. Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis[C]//ISLPED 2003. Seoul: ACM Press, 2003.

[7] MEJIA A P, LEVNER E, MOSSE D. Adaptive scheduling server for power-aware real-time tasks[J]. ACM Trans. on Embedded Computing Systems, 2004, 3(2): 284-306.

[8] PILLAI P, SHIN K G. Real-time dynamic voltage scaling for low power embedded operating systems[C]//In Proceedings of SOSP 2001. Banff: ACM Press, 2001.

[9] KIM W, SHIN D, YUN H S, et al. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems[C]//In: RTAS 2002. San Jose: IEEE CS Press, 2002.

编　辑　熊思亮