基于 DVS 的多核实时系统节能调度

钟 萧, 齐 勇, 侯 迪, 苗 蕾, 郑晓梅
(西安交通大学电子与信息工程学院软件研究所, 陕西西安 710049)

摘 要: 动态电压调节是一种有效的节能技术. 本文提出了多核处理器平台上的一种近似最优的动态电压调节算法. 算法将电压调节问题转化为松弛时间分配问题, 由任务集结构找到存在的松弛时间, 针对不同类型的松弛时间, 使用了并行补偿等分配方法. 实验结果表明本文的算法能够有效的降低能量消耗且具有较低的时间复杂度.

关键词: 动态电压调节; 实时任务调度; 多核处理器
中图分类号: TP393 文献标识码: A 文章编号: 0372-2112 (2006) 12A-2481-04

Tasks Scheduling with Dynamic Voltage Scaling on Multi-Core Real-Time Systems

ZHONG Xiao, QI Yong, HOU Di, MIAO Lei, ZHENG Xiao-mei
(*Institute of Software, School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China*)

Abstract: The method of dynamic voltage scaling (DVS) is an efficient technology for saving energy. This paper put forwards a near-optimal strategy on multi-core processor using DVS. It changes the problem of voltage scaling to the problem of slack time allocation. It detects the slack time based on task structure firstly. Aiming at different kinds of slack times, it then uses relevant allocating method such as parallel compensate algorithm. Result of the experiment indicates that the strategy of this paper can decrease the energy consumption efficiently and has low complexity.

Key words: DVS; real-time task scheduling; chip multi-processor

1 引言

近来, 随着移动设备和嵌入式设备平台数目飞速的增长, 能量消耗问题越来越受到关注, 如何在不降低性能的情况下尽可能减少能量消耗成为了热点问题. 研究人员从软硬件方面提出了多种节能技术和机制, 其中动态电压调节(DVS)被认为是一种有效节能技术.

动态电压调节的核心思想是: 在满足任务完成时限要求的前提下, 在程序运行时动态调节处理器电压, 使得处理器不总以最高电压工作, 起到节能的作用. 文献[1]中提出了一个两阶段算法, 在离线阶段, 调度由每个任务的最差执行时间决定, 在运行阶段, 调度由执行时的松弛时间进行调整. 文献[2]利用拉格朗日乘子方法解决调度问题, 先找到任务电压之间的关系, 并用迭代的方式找到满足这个关系的解. 一种基于负载电流函数的调度方法在文献[3]中被提出. 而在文献[4]中提出了一种具有双电压调节处理器的最优 DVS 算法. 目前, 大部分的算法是基于单处理器平台的, 仅考虑了相互独立的任务, 而且算法的时间复杂度大多大于 $O(n^2)$.

本文提出了一种在多核处理器平台上基于任务结构分析的 DVS 算法, 通过分析实时任务间的依赖关系, 得到任务松弛时间 t_{idle}和系统松弛时间 t_{slack}, 而由松弛时间的分配, 动态调节处理器电压, 找到各任务运行频率的近似最优解, 而分配算法自身的时间复杂度接近 $O(n)$.

2 DVS 节能原理

对电压调节而言, 以数字 COMS 技术为基础的处理器功率消耗主要由动态开关功耗 P_d 决定[5], 即:

$$P_d = \alpha_T * C_{load} * V_{DD}^2 * f_{CLK} \quad (1)$$

其中, α_T 是节点转换因子, f_{CLK}是处理器时钟频率, V_{DD}是系统供给电压.

处理器速度f_{CLK}和系统供给电压 V_{DD}之间的关系由式(2)可知:

$$f_{CLK} = \eta * \frac{(V_{DD} - V_{th})^2}{V_{DD}} \quad (2)$$

其中, η 是一个常量, V_{th}是系统的门限电压. 由式(2)可以看出, 当 $V_{DD} >> V_{th}$的时候, f_{CLK}与 V_{DD}之间几乎是线性的关系.

对某个任务 T_i 而言, 在处理器速度f_{CLK}下, 执行时间为:

$$t_i = IC_i * CPI_i * f_{CLK}^{-1} \quad (3)$$

收稿日期: 2006-09-22; 修回日期: 2006-12-10
基金项目: 国家自然科学基金(No. 60473098); 微软合作项目

其中 IC_i 是指令数, CPI_i 是每条指令的平均 CPU 周期数. $IC_i * CPI_i$ 即为执行任务 T_i 所需的 CPU 周期数, 用 C_i 表示.

由式(1)~(3)联立可知, 执行任务 T_i 的能量消耗:

$$E_i = P_d * t_i = C_{load}\alpha_T V_{DD}^2 C_i \approx C_{load}\alpha_T \eta^{-2} f_{CLK}^2 C_i \qquad (4)$$

当降低处理器速度 f_{CLK}, 将平方级的减少任务能量消耗 E_i. 由式(3)可以看出, 当 f_{CLK} 降低时, 将会线性的增长任务的执行时间. 在满足任务时间的限制条件下, 以任务执行时间的线性增长来换取能量消耗的平方级降低对能源受限的实时系统相当重要.

3 模型建立

目前, 多核处理器芯片的体系结构没有统一的标准. 如图 1 中所示, 本文认为每个处理器核都是同构的, 拥有独立的 L1 缓存, 共享 L2 缓存. 每个处理器核能够独立的调节电压. 本文主要研究的是相关任务在多核处理器上的调度, 即要考虑任务之间的前置性约束. 且因为每片处理器上核的个数将快速增长[6], 因此本文认为处理器核的个数是不受限的.

定义 1 任务集 F 由子任务 $T_1, T_2, T_3, \cdots, T_n$ 组成, 任务集的周期为 t. 即任务运行的单次截止时间为 t. 任务集表示为:

$$F = \{T_1, T_2, T_3, \cdots, T_n, t, \prec\}$$

其中 $\prec$ 表示子任务间的前置约束关系, 如 $T_i \prec T_j$, 表示 T_j 的运行依赖于 T_i 的运行结果, 换句话说 T_i 必须在 T_j 开始执行前结束. 因为(共享二级缓存的作用)任务在多核平台上不同处理器核之间的通信代价比多处理器系统中要小得多, 在本文中认为任务间通信代价为 0. 考虑 CPU 核 α 上的时间代价函数 $\omega_\alpha(F) = \sum_{k=1}^{M} t_{\alpha k}$, 其中 M 是任务集 F 分派到 α 上的任务数, $t_{\alpha k}$ 是任务 T_k 在 α 上执行所耗费的时间. 完成整个任务集 F 的总时间代价函数 $\omega(F) = \max \omega_\alpha(F)$. 则任务集 F 的最短执行时间:

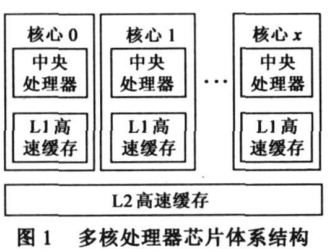

图 1 多核处理器芯片体系结构

$$t_{path} = \min(\omega(F)) = \min(\max \omega_\alpha(F)) \qquad (5)$$

将决定 t_{path} 的任务流程称为任务集 F 的关键路径, 关键路径上的任务集为 $Path = \{T_{p1}, T_{p2}, T_{p3}, \cdots, T_{pm}\}$. 即关键路径上的任务在处理器最高频率的执行时间为 t_{path}, 显然有 t_{path} 小于等于周期时间, 否则任务是不可能满足实时性要求的.

定义 2 适当的延长非关键路径上任务的执行时间, 并不会延长整个任务集的执行时间, 这类延长时间统称为任务松弛时间 t_{idle}. t_{idle} 的分散情况和大小视任务集的不同而变化.

定义 3 周期时间 t 与任务集 F 的最短执行时间 t_{path} 之间的时间差为最快执行时的系统松弛时间, 令其为 t_{slack}, $t_{slack} = t - t_{path}$.

4 算法描述

当处理器电压 V_{DD} 降低时, 处理器频率 f_{CLK} 将下降, 而由 $t_i = C_i * f_{CLK}^{-1}$, 因为 C_i 不变, 则任务执行时间 t_i 线性延长, 消耗能量 E_i 降低. 结合模型分析可知节能调度算法演变为如何将时间片 t_{idle} 和 t_{slack} 分配给各个任务, 以在保证任务集实时运行的情况下获取能量的最大收益.

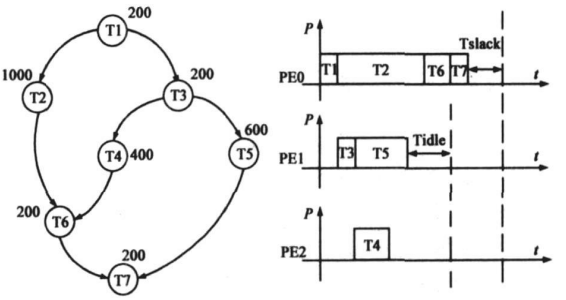

图 2 任务调度图及其在非 DVS 系统上的能量消耗

4.1 任务松弛时间分配算法

将任务集由一个有向无环图表示, 利用关键路径算法[7]找到在关键路径之后, 非关键路径上的任务就可以在不影响最短执行时间的情况下延长自己的执行时间, 所以也称为任务松弛时间(或缓冲时间)分配问题. 任务松弛时间的分布与任务集结构相关, 且较为分散, 很难找到最优的分配方案. 本文采用了贪心法分配.

由公式(4)可以推出能量增益公式

$$\Delta E_i = P_i * t_i - P_i' * t_i' = C_{load}\alpha_T \eta^{-2}\left(V_{DD}^3 t_i - V_{DD}'^3 t_i'\right)$$

$$= C_{load}\alpha_T \eta^{-2} C_i^3 \left(\frac{1}{t_i^2} - \frac{1}{(t_i + \Delta t)^2}\right) \qquad (6)$$

由能量增益公式可以推出, 在各任务 α_T 相等的情况下, 当 $C_1 > C_2$ 时, $\Delta E_1 > \Delta E_2$, 即相等的时间片 Δt 分给 C_i 值较大的任务获取的能量收益最大. 因此在分配过程中优先将时间片分给 C_i 值较大的任务, 但分配的时间片不应超出此任务的最早开始时间和 最晚完成时间.

算法描述如下:

(1)将所有非关键路径任务放入队列 Task;

(2)从 Task 中选取执行时间最长的任务 T_k 作为当前任务, 使任务开始时间 $TS(T_k)$ 等于任务最早开始时间 $ES(T_k)$, 任务结束时间 $TF(T_k)$ 等于任务最晚结束时间 $LF(T_k)$;

(3)根据任务 T_k 的开始时间, 修改任务 T_k 的所有前驱任务的最晚完成时间;

(4)根据任务 T_k 的结束时间, 修改任务 T_k 的所有后继任务的最早开始时间;

(5)将 T_k 从 Task 中删除, 如队列 Task 为空, 结束, 否则转 2.

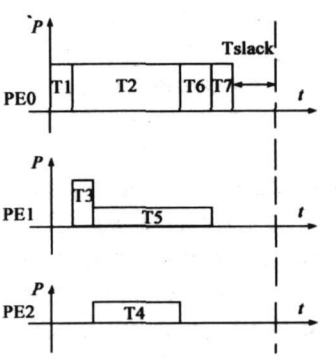

图 3 任务松弛时间分配后的能量消耗

分配后结果如图 3 所示:

4.2 系统松弛时间分配算法

关键路径上任务执行时间和就是系统最短执行时间 t.

由 $t_{slack}= t- t_{path}$, 可以计算出系统松弛时间 t_{slack}. 利用任务的时间片对任务进行分层, 当有新的任务生成时便设为一层, 每一层上的任务个数称为并行度, 意味着任务可以同时并行执行. 与任务松弛时间不同, 系统松弛时间的分配是以层为单位, 同层的任务分得相同的松弛时间, 若任务属于几个不同的层, 则最终分配的时间为各层松弛时间之和.

表 1　仿真实验结果

任务集	占用 CPU 核个数	能量消耗(非 DVS)	参考文献算法能量消耗	最优能量消耗	本文算法能量消耗	能量节省%	误差率%
1	4	8500.00	5325.66	3974.13	4086.06	48.07%	2.58%
2	4	6348.00	4056.63	3272.35	3467.62	54.62%	5.97%
3	3	12578.00	7127.94	5454.32	5702.38	45.34%	4.55%
4	4	7140.00	3597.08	2854.86	3060.69	42.86%	7.2%
5	5	8945.00	6025.73	4997.19	5282.78	59.06%	5.72%

考虑目标函数最大化能量增益:

$$\max\ \Delta E_{total}= \sum_{i=1}^{n}(E_i- E_i') = C_{load}\alpha_T\eta^{-2}\sum_{i=1}^{n}C_i^3\left(\frac{1}{t_i^2}-\frac{1}{(t_i+x_i)^2}\right) \quad (7)$$

因为任务 T_i 在最高 CPU 频率下运行的时间是定值, 所以问题转化为求

$$\min E'_{total}= \sum_{i=1}^{n}E_i'= C_{load}\alpha_T\eta^{-2}\sum_{i=1}^{n}C_i^3\left(\frac{1}{(t_i+x_i)^2}\right) = C_{load}\alpha_T\eta^{-2}\sum_{j=1}^{N}m_jC_j^3\left(\frac{1}{(t_j+x_j)^2}\right) \quad (8)$$

其中 N 为层数, m_j 表示第 j 层的并行度, C_j 表示执行第 j 层任务所需 CPU 周期数, t_j 表示执行第 j 层任务所用时间, α_T 是节点转换因子, 为了简化公式认为各任务 α_T 值相等, x_j 表示分配给第 j 层的系统松弛时间, 可见 $t_{slack}= \sum_{j=1}^{N}x_j$, 在这个约束条件下可以求出最优的 x_j 分配值:

$$x_j = \frac{C_j\sqrt[3]{m_j}}{\sum_{k=1}^{N}C_k\sqrt[3]{m_k}}\left(t_{slack}+\sum_{k=1}^{n}t_k- t_j-\beta t_j\right) \quad (9)$$

其中, $\beta = \frac{\sum_{\lambda=1}^{N}C_\lambda\sqrt[3]{m_\lambda}- C_j\sqrt[3]{m_j}}{C_j\sqrt[3]{m_j}}$ 在 $x_1, x_2, x_3, \cdots, x_n > 0$ 的约束条件下找到最优分配值是困难的, 因此我们取:

$x_j = \frac{C_j\sqrt[3]{m_j}}{\sum_{k=1}^{N}C_k\sqrt[3]{m_k}}t_{slack}$ 作为分配给第 j 层的系统松弛时间.

其中 $\sqrt[3]{m_j}$ 称为系统并行补偿参数. 可以看出任务的完成周期越长, 跨越的层次越多, 则分到的系统松弛时间越长.

算法描述如下:

(1) 从任务开始时间为 0 的任务开始, 为程序分层, 当有新的任务开始执行时, 即为一层. 记录下每层的任务编号.

(2) 将所有任务放入链表 List;

(3) 从第一层任务开始按 $x_j = \frac{C_j\sqrt[3]{m_j}}{\sum_{k=1}^{N}C_k\sqrt[3]{m_k}}t_{slack}$ 分配松弛时间片到各层.

(4) 从 List 中删除只属于这一层的任务.

(5) 如果 List 为空, 跳 6, 否则继续 3 直到所有层已经处理.

(6) 计算任务 T_j 总共分配到的松弛时间片, 作为其新的完成时间.

分配结果如图 4 所示:

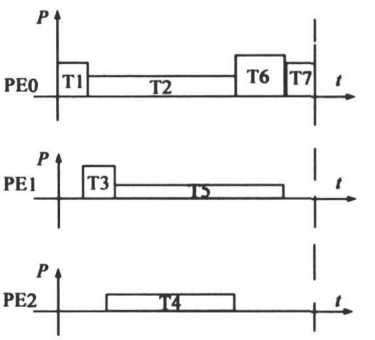

图 4　系统松弛时间分配后的能量消耗

5　实验结果

本文建立了一个仿真实验环境来对算法进行实验和验证. 仿真器分为两个部分: 第一部分是任务产生器, 随机产生符合需要的任务图. 第二部分的作用是实现算法, 将任务分配到不同的核上并动态调节处理器核的电压.

表 1 列出了产生的任务集的不采用 DVS 技术时的能量消耗, 使用文献[3]算法的能量消耗, 最优的能量消耗, 以及本文所提算法的能量消耗. 由实验结果可以看出, 在使用本文提出的算法后, 能量消耗比不使用 DVS 的系统下降了 45% ~ 60%. 与最优调度相比, 误差的范围在 2% ~ 10% 之间. 其中误差定义为 $(E- E_{best})/E_{best}$. 值得注意的是, 能量节省的比例和误差大小由系统松弛时间 t_{slack} 和任务集 F 的结构决定. 当松弛时间 t_{idle} 和 t_{slack} 的值越高, 能量节省的比例越高.

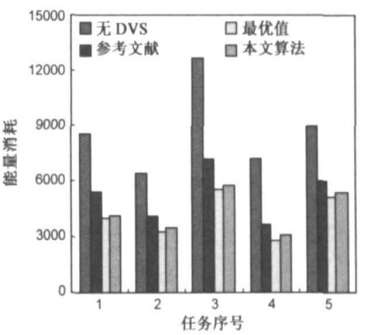

图 5　任务能量消耗直方图

6 结论

本文提出了一种近似最优的 DVS 节能调度算法, 将电压调节问题转化为松弛时间分配问题, 通过任务集的结构分析, 找到任务集中可能出现的不同类型的松弛时间. 针对任务松弛时间 t_{idle}和系统松弛时间 t_{slack}的不同特点, 分别利用并行补偿等机制予以分配. 仿真实验结果证明是一种简单而有效的方法. 将来的工作将进一步扩展到考虑处理器核关闭和限定处理器核个数时情况的研究.

参考文献:

[1] Ali Manzak, Chaitali Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy Low Power Electronics and Design[A]. Proceedings of the 2001 international symposium on Low power electronics and design[C]. US: ACM Press, 6-7 Aug, 2001. 279-282.

[2] Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, Mani B Srivastava. Power optimization of variable-voltage core-based systems[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1999, 18(12): 1702-1714.

[3] Yuan Cai, Reddy S M, Pomeranz, I, Al-Hashimi B M. Battery-aware dynamic voltage scaling in multiprocessor embedded system[A]. IEEE International Symposium on Circuits and Systems 2005[C]. 23-26 May 2005. 1. 616-619.

[4] 王华勇, 陈渝, 康烁, 戴一奇. 具有双电压调节处理器的最优 DVS 算法[J]. 清华大学学报(自然科学版), 2005, 45(10): 1405-1408.
Wang Hua yong, Chen Yu, Kang Shuo, Dai Yiqi. Optimal DVS algorithm for real-time systems with double voltage scalable processor[J]. Journal of Tsinghua University (Science & Technology), 2005, 45(10): 1405-1408. (in Chinese)

[5] Sung-Mo Kang, Yusuf Leblebici. CMOS Digital Integrated Circuits—Analysis and Design[M]. 北京: 清华大学出版社, 2002.

[6] Intel Multi-Core Processor Architecture Development Backgrounder[OL]. www. Intel. com.

[7] Xu Feng-sheng. A New algorithm for finding the critical paths[J]. Computer Engineering and Application, 2005, 41(24): 82-84. (in Chinese)

作者简介:

钟 男, 1982 年 8 月生, 西安交通大学在读硕士研究生, 主要研究方向: 分布式计算、多核计算平台.
E-mail: jiuhu2005@gmail. com

齐 勇 男, 教授, 博士生导师, 研究方向: 分布式系统、普适计算、移动计算、中间件技术和面向对象理论与技术等.