

一种任意时间片长度对 CPU 上下文切换代价影响消除的新方法

张 洋^{1,2}, 张逸中^{1,2}

(1 华东师范大学 可信物联网产学研联合研发中心, 上海 200062; 2 华东师范大学 软件学院, 上海 200062)

摘 要: 分析并量化了操作系统任务调度时上下文切换对 CPU 性能影响, 得出了任务上下文切换代价对 CPU 效率影响关系. 在此影响关系的基础上, 提出了一种可行的 Hyper-Scheduling 方法. 此方法通过在 CPU 内部设置一条特殊的任务切换专用流水线数据通路, 并在通用寄存器堆上寄生一个相同的寄存器堆(影子寄存器堆)和监视访存状态的寄生逻辑, 来监视各硬件资源运行情况, 实现在 CPU 硬件资源闲置状态时进行预先数据准备或任务保存. 当任务切换时将这条特殊数据通路与 CPU 主数据通路流水线进行交换, 使上下文切换时间可趋近于零, 任务切换代价被消除. 上下文频繁切换或时间片长度缩短至近于零的情况下, CPU 效率仍能够保持性能最大化.

关键词: 时间片; 寄生逻辑; 上下文切换; 流水线

中图分类号: TN402

文献标识码: A

文章编号: 1000-7180(2015)02-0092-05

A New Method to Eliminate the CPU Context Switching Cost by the Effects of any Length of Time Slice

ZHANG Yang^{1,2}, ZHANG Yi-zhong^{1,2}

(1 Trusted Internet of Things Industry-Academy-Research Joint Research and Development Center, East China Normal University, Shanghai 200062, China; 2 Software Engineering Institute, East China Normal University, Shanghai 200062, China)

Abstract: This paper analyzes and quantifies the impact of context switching on CPU performance when operating system task scheduling, it concludes that the relation of the cost of mission context switching to the CPU efficiency. On the basis of the study on the influence relation, this paper proposes a feasible Hyper-Scheduling engineering method. By setting up a special task switching dedicated data path line in the CPU, and insert the parasitic logic in each key parts of CPU will be used to monitor the running status of the hardware resources, this method realizes the CPU hardware resources idle state for data preparation in advance or task. Exchange the special data path and main CPU data path line when task switching, which makes the context switch time can be close to zero, task switching cost can be eliminated. This means even in the situation of frequency context switching or time slice length is very short, CPU can remain in the maximized efficiency.

Key words: time slice; parasitic logic; context switching; pipeline

1 引言

上下文切换时间是评价操作系统最重要的技术指标之一. 上下文切换必将导致 CPU 流水线非用户工作态的打断, CPU 进入中断, 执行操作系统内核的指令的状态. 频繁切换或严重的颠簸现象将使

CPU 的性能额外损失, 并随着任务数量的增加而增加. 这个切换代价在理论上少有证明, 并被认为是无法避免的. 如果这个切换代价能够降低到趋近于零的话, 就可以得到在时间片缩短 N 倍情况下获得任务响应频率提高 N 倍的好处. 目前在软件上, 即使对各种调度算法进行优化, 其切换代价还是无法消

除.如最小空闲时间优先 LSF 算法,改进的 LSF 算法(improved least slack first,ILSF)^[1-4],减少了任务上下文之间的切换次数(context switching number,CSN),降低了任务的截止期错失率(missed deadline percentage,MDP).其目的也是减少上下文切换对 CPU 额外性能损失增加的影响,其本质就是减少 CPU 流水线中对用户来说的无用指令.

目前国内外的研究中,在硬件层面的优化方面,上下文切换的方法仅限于研究一种单纯的把软件调度算法用硬件调度器来替代的思路^[5-7],且都是作为 CPU 的协处理器来看待.但传统思路都没有考虑任务调度是由一连串指令流和数据流共同完成并且完整地在 CPU 流水线中运行的过程.硬件调度器只是在某种意义上将指令流过程转换成硬件处理过程,但并未考虑在数据流上的优化.

本文的工作主要分为两块:

(1)分析并量化了操作系统任务调度时上下文切换对 CPU 性能影响,得出了 T_{cs} (Timing Context Switch,上下文切换时间)任务上下文切换代价对 CPU 效率影响关系并从理论上寻找解决问题的思路.

(2)提出了一种新的 Hyper-Scheduling 方法,此方法实现了一种融合在 CPU 内部的硬件逻辑而非独立的单纯调度协处理器.其中包含了调度器、监控 CPU 各个关键部件运行状态的寄生逻辑、影子寄存器及调度用的特殊数据通路流水线.可以预先将任务调度所需数据流提前装在特殊数据通路流水线和影子寄存器中,任务切换时实现真正意义的零损耗任务切换.

2 可行性分析以及架构

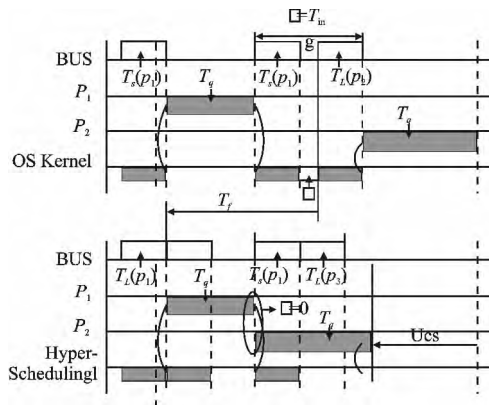


图1 线程切换的时间分析

2.1 可行性分析

从理论上提出可行性分析,图1中 $T_L(P_n)$ 表示从内存总线 Bus 恢复任务 P_n 现场的时间, $T_S(P_n)$ 表示从内存总线 Bus 中保存任务 P_n 现场的时间,当任务 P_1 进过了 T_q (时间片长度)后进入了长达 $\epsilon = T_{ing} = T_S(P_n) + T_L(P_{n+1}) + \theta$ (θ :临界区线程调度处理时间)的中断响应时间来进行任务切换.为了使任务 P_1 和 P_2 能够在 CPU 流水线中不停顿地执行,即 $\epsilon = 0$.由于保存和恢复现场的过程不能省去,因此需要一种硬件机制能够在 P_1 执行的过程中把线程 P_2 的现场预先从内存提取出来,即 $T_L(P_2)$ 的工作被提前大约 $T_f = T_q + T_S(P_{n-1}) + \theta$ 与 CPU 处理同步进行.在 P_2 运行的时候,同时将 P_1 的现场送回内存.它成立的原因是 CPU 流水线的各种资源不可能在同一时刻都被利用到.因此我们引入了 Hyper-Scheduling 任务调度方法,在 CPU 各种资源闲置的时候可以充分被利用到,使任务与任务间无缝连续运行.

2.2 零损耗任务切换的理论证明以及问题量化

以 Round-Robin(RR) 时间片调度算法来建立一个线程切换模型,以便找到上下文切换代价影响 CPU 性能关键点,并找出工程方法来解决该问题.

首先定义一个持续的任务系统由 $T(\tau_1, \tau_2, \dots, \tau_n)$ 组成, $\tau_i = (E_{\tau_i}, N_{\text{need}_{\tau_i}})$, 每一个 $\tau_i \in T, i = 1, \dots, n$ 称为一个任务.某一时刻,一个任务可能被挂起或恢复,因此某一时刻的任务数 m 为一随机值,为样本空间 S 中的某一样本 $m_i(t)$. E_{τ_i} 为任务 i 执行所需时间,则 $U_{pc} = \sum_{i=1}^m E_{\tau_i}$ 为存在 m 个任务时执行所需时间,而 $m = m(t)$, 某一时刻的任务数,是一个随机变量,它的一个样本空间 S 为 $\{m_1(t), m_2(t), \dots, m_i(t)\}, 1 \leq m_i(t) \leq M$, 每一时刻, m 为 S 中的一个样本.但任务在相互切换时必然存在上下文的切换时间,若把这个额外开销计算在内,对 CPU 将有一定的影响,其上下文切换的总延时 $U_{cs} = \sum_{i=1}^m T_{cs}$ 为 m 个任务之间切换的总和,而上下文的切换时间 $T_{cs} = T_l + T_s + T_{sch}$, 式中 T_l 为任务的恢复时间, T_s 为任务的保护时间, T_{sch} 为任务调度以及额外的开销, T_{cs} 则是三者之和.而 CPU 实际运行所需时间 $U_{total} = U_{pc} + U_{cs}$, 此时 CPU 的工作效率:

$$\eta = \frac{U_{pc}}{U_{total}} \quad (1)$$

可见 CPU 的效率明显受到 U_{cs} 的影响,若 U_{cs} 增大将极大制约 CPU 性能发挥.

现假设某一时刻系统中的任务为 $\tau_1, \tau_2, \dots, \tau_i, i \in m$ 对 $N_{ned_{\tau_i}}$ 进行排列, 为 $\{N_{ned_{\tau_1}}, N_{ned_{\tau_2}}, \dots, N_{ned_{\tau_i}}\}$, 取 $N'_{ned_{\tau_i}} = \min\{N_{ned_{\tau_1}}, N_{ned_{\tau_2}}, \dots, N_{ned_{\tau_i}}\}$ 此时上下文切换的总延时及任务执行所需的总时间分别为 U_{cs}

$$= T_{cs}(\sum_{i=1}^m N'_{ned_{\tau_{i-1}}}), U_{pc} = T_q \sum_{i=1}^m N'_{ned_{\tau_i}} \quad (2)$$

当任务系统中存在最多 M 个任务时, 取 $\max[U_{cs}] = T_{cs}(\sum_{i=1}^m N'_{ned_{\tau_i}} - 1)$, $\max[U_{pc}] = T_q \sum_{i=1}^m N'_{ned_{\tau_i}}$.

则 CPU 的工作效率:

$$\eta = \frac{\max[U_{pc}]}{\max[U_{pc}] + \max[U_{total}]} \quad (3)$$

现提出假设: 若上下文切换的总延时 $U_{cs} \approx 0$, 则 $\eta \approx 100\%$, 这样 CPU 将不再受上下文切换的制约, 它的执行效率才会达到最大值, 同时我们知道 CPU 的功耗 $P \propto T$, T 为运行时间, 此时将为 CPU 节省在时间 U_{cs} 内的消耗. 只有减少了切换延时才能有效地提升 CPU 的效率. 其次考虑上下文切换对任务 τ_i 的周转时间的影响, 定义 $T_{TRnd(\tau_i)}$: 任务 τ_i 的周转时间, 从任务第一次进入就绪状态时刻开始到任务最后一次结束运行状态的时刻, 这个期间的时间数目. 首先忽略上下文切换对 CPU 的影响, 将得出周转周期的理想值.

$$T_{TRnd(\tau_i)} = \begin{cases} (M-i+1)E_{\tau_i} + \sum_{k=1}^{i-1} E_{\tau_k} - T_q M + T_q, & i \geq 2 \\ M(E_{\tau_i} - T_q) + T_q, & i = 1 \end{cases} \quad (4)$$

若把上下文切换时间计算在内, 周转周期将增加为

$$\Delta T_{TRnd(\tau_i)} = (\frac{T_{TRnd(\tau_i)}}{T_q} - 1) \times T_{cs} \quad (5)$$

这样, 若 $\Delta T_{TRnd(\tau_i)}$ 接近于 0, 将有效减少进程的周转周期, 提高任务的相应频率. 现在以一个简单的模型定量分析上下文切换对 CPU 的影响, 令 $M = 5$, 此时任务为 $\tau_i (E_{\tau_i}, N_{ned_{\tau_i}})$, $1 \leq i \leq 5$, 现假设若 $i < j$, 则 $N_{ned_{\tau_i}} \leq N_{ned_{\tau_j}}$, $m(t)$ 的取值随着任务的完成依次从 5 递减到 1, 由式(5)可得

$$U_{csi} = \begin{cases} T_{cs}(\sum_{i=1}^i N_{ned_{\tau_i}} - 1), & i = 5, 4, 3, 2 \\ 0, & i = 1 \end{cases} \quad (6)$$

$$U_{pci} = \begin{cases} T_q i \times (N_{ned_{\tau(6-i)}} - N_{ned_{\tau(5-i)}} + 1), & i = 4, 3, 2, 1 \\ T_q i \times (N_{ned_{\tau(6-i)}} - 1) + 1, & i = 5 \end{cases} \quad (7)$$

因此上下文切换的总延时及任务执行时间

$$U_{cs} = \sum_{i=1}^5 U_{csi} + (5 - 1) \times T_{cs} =$$

$$T_{cs}(\sum_{i=1}^3 N_{ned_{\tau_i}} + 2N'_{ned_{\tau_4}}) \quad (8)$$

$$U_{pc} = \sum_{i=1}^5 U_{pci} = T_q \sum_{i=1}^5 N_{ned_{\tau_i}} \quad (9)$$

CPU 的工作效率:

$$\eta = \frac{U_{pc}}{U_{pc} + U_{cs}} = \frac{\sum_{i=1}^5 N_{ned_{\tau_i}}}{\sum_{i=1}^5 N_{ned_{\tau_i}} + \frac{T_{cs}}{T_q}(\sum_{i=1}^3 N_{ned_{\tau_i}} + 2N_{ned_{\tau_4}})} \quad (10)$$

若假设 $N_{ned_{\tau_4}} \approx N_{ned_{\tau_5}}$, 则式(10)化简可得

$$\eta \approx \frac{1}{1 + \frac{T_{cs}}{T_q}} \quad (11)$$

图 2 为 η 随 T_{cs}, T_q 的变化情况, 从图中可看出, $X \rightarrow 0 (T_{cs} \rightarrow 0)$, yz 平面 CPU 性能将趋近于 100%, Hyper-Scheduling 处理逻辑与 CPU 时间复用, U_{cs} 时间移出交由 HS 处理, CPU 中时间就能达到 $T_{cs} \rightarrow 0$, 验证了 Hyper-Scheduling 架构方法与上述推论结果的一致性.

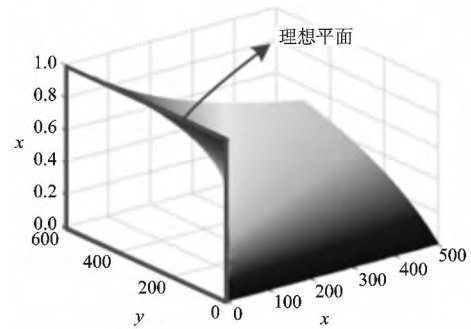


图 2 上下文切换延时对 CPU 性能影响

3 Hyper-Scheduling 消除上下文切换代价方法

3.1 辅助四级流水线(简记 HS 流水线)

HS 流水线包含一个 HS 状态机、一个与通用寄存器堆(GPR)和一个克隆的影子寄存器堆(HS_GPR). HS 流水线的四级分别是物理寄存器读、Hyper Scheduling 处理、访存和物理寄存器写. 访存逻辑是 CPU 共享资源, 但其访存的顺序是 CPU 正常流水线优先, 闲置时 HS 流水线才能使用. 从图 3 可以看到任务切换时流水线中的数据通路. 当保存现场时, 影子寄存器堆已经存有上文的数据, 该数据进入 HS 流水线中, HS 状态机检测到当访存逻辑空闲时将上文数据送入内存. 恢复现场时则是反向过

程.当上下文切换时,影子寄存器堆(存有预先读取的下文数据)和通用寄存器堆(存有上文数据)通路

交换,它们的名字也进行交换,这样处理器的正常流水线没有任何打断就完成了任务切换.

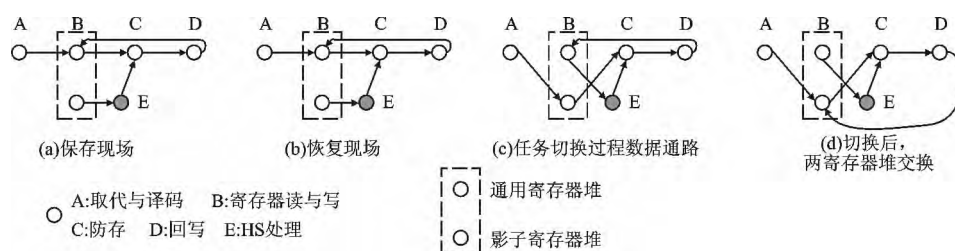


图3 流水线数据流图

3.2 影子寄存器堆的原理说明

CPU中有两套寄存器堆,正常流水线使用其中一套为GPR,另外一套为HS_GPR.任务切换时,两套寄存器堆将互换上下文数据进入不同的流水线(见图4).由于指令指针及一些特殊寄存器(如SR状态寄存器)都是不能在任务间共享的,取指级与译码级中HS状态机将负责预存储下一任务恢复运行时所需的第一条指令地址与其他特殊寄存器的值.

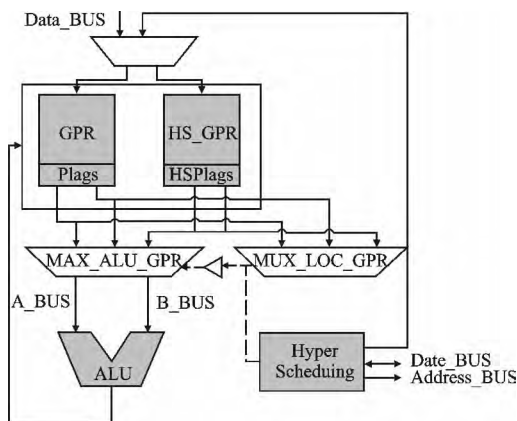


图4 寄生在GPR上的HS_GPR及数据通路

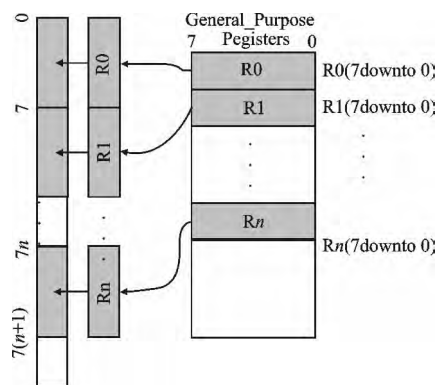


图5 GPR的上文数据折叠至HS_GPR

在进行快速通用寄存器堆切换时,过去的页面文件保存与装载方式都是通过CPU指令中PUSH

& POP操作完成,即当有 N 个寄存器需要保存与装载时需要执行 N 次PUSH&POP指令.通用寄存器堆是由一个长度为 $8N$ bit寄存器折叠成的 N 个8 bit寄存器(图5).这样数据就可以依次被HS流水线送回内存中了.

另外HS控制器会通过访问逻辑上的寄生逻辑监视访问空闲与否,并进行上下文数据切换.

这种流水线的GPR切换,可以提高在上下文切换时的开销,一个机器时钟周期就能使一个新任务立即运行,运行的同时,使用特殊寄生逻辑将前一个任务状态数据移入内存,做到了时间复用,提高了效率.

4 实验及性能分析

为了验证Hyper-Scheduling的方法在时间片取任意长度时,对CPU效率影响的消除效果.对开源CPU Openrisc1200进行Hyper-Scheduling改造.

并对此改造后的CPU进行了Dhrystone数学运算综合基准测试.测试中,操作系统使用缩减版的uclinux(只含有任务管理与调度),建立了2个计算线程,并规定时间片 T_q 的大小,测试时间为1s, T_q 将从10 000 ms长度不断缩小至0.1 ms,并同时计算MIPS值,从而得知上下文切换开销对CPU运算性能的影响及使用Hyper-Scheduling方法后CPU性能发挥的结果.进行了三大段时间片长度共46小组测试. T_q 是时间片长度, T_{cs}/T_q 是上下文切换的开销与时间片之比.随着 T_q 时间片长度不断缩小,任务切换的所占总时长的比重不断加重.图6显示了时间片从10 000 ms缩减至100 ms时使用HS方法和未使用HS方法基本无明显差异.图7中时间片从100 ms缩减至20 ms处开始出现CPU计算成绩下滑的趋势,即已经出现了CPU忙于应付上下文切换带来的性能损失,然而使用HS技术后则

CPU 性能损失几乎没有. 当进行到 T_q 缩减到 1 ms 以后, 性能差异达到了 12 倍之多, 如图 8 所示. 可以看出 HS 带来的 CPU 性能超低损失优势.

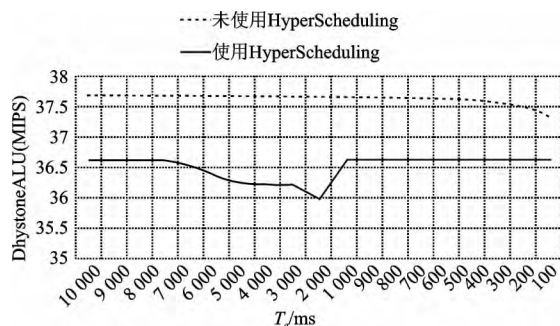


图 6 $T_q = [10\ 000, 100]$ CPU 任务切换性能比较

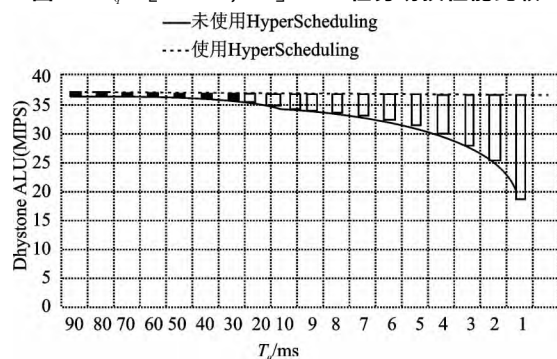


图 7 $T_q = [90, 1]$ CPU 任务切换性能比较

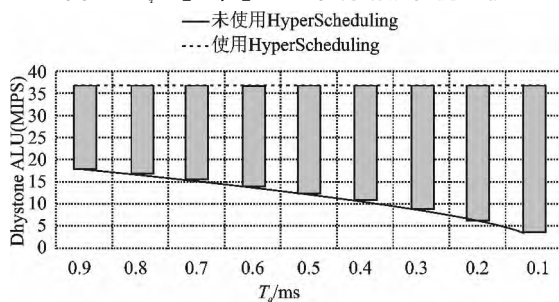


图 8 $T_q = [0.9, 0.1]$ CPU 任务切换性能比较

5 结束语

本文阐述了一种新的方法来消除任意时间片长度对 CPU 性能的影响, 在理论及实验都进行了论证. 目前的 Hyper-Scheduling 方法是在理想 Cache 情况下进行测试的, 即 CPU 完全执行在 Cache 中的指令, 内存总线完全让给 Hyper-Scheduling 使用. 我们在技术架构与方法上会继续改进, 下一步完善当 Cache 未命中时切换的代价影响.

参考文献:

- [1] Abbott R, Garcia-Molina H. Scheduling real-time transactions: A performance evaluation[C] // proceeding of the 14th VLDB conference. Los Alamitos: IEEE Computer Society Press, 1993:652-663.
- [2] Giorgio Buttazzo. Achieving scalability in real-time systems[J]. IEEE Compute, 2006, 39(5):54-59.
- [3] Juanjo Noguera, Rosa M Badia. System-level power-performance tradeoffs for reconfigurable computing [J]. IEEE Transactions on Vlsi Ssystems, 2006, 14(7):730-739.
- [4] G Wigley, D Kearney. The development of an operating system for reconfigurable computing[C] // Proc IEEE Symp FCCM'01. Australia:IEEE Press, 2001.
- [5] Pramote Kuacharoen, Mohamed A Shalan, vincent J Mooney. A configurable hardware scheduler for real-time systems[C] // Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, 2003.
- [6] Chetan Kumar N Ga, Sudhanshu Vyasa, Jonathan A. et al. Improving System Predictability and Performance via Hardware Accelerated Data Structures[C] // International Conference on Computational Science, ICCS 2012.
- [7] J Agron, D Andrews, M Finley, et al. FPGA Implementation of a priority scheduler module[C] // Proceedings of the 25th IEEE International Real-Time Systems Symposium, Works in Progress Session (RTSS WIP 2004), December 2004.
- [8] W Peck, J Agron, D Andrews, et al. Hardware/Software co-design of operating system services for thread management and scheduling[C] // Proceedings of the 25th IEEE International Real-Time Systems Symposium, Works in Progress Session (RTSS WIP 2004), 2004.

作者简介:

张 洋 男, (1988-), 硕士研究生. 研究方向为软硬件协同设计. E-mail: edisonzh@126.com

张逸中 男, (1986-), 博士研究生. IEEE 专业会员. 研究方向为可性物联网.