多核处理器平台资源管理的若干问题研究

刘宇芳
(惠州学院计算机系 惠州 516007)

摘 要 在多核处理器体系结构中,同一芯片上集成了多个处理器核心,它们共享片上多种硬件资源。介绍了多核处理器技术的发展;提出了多核处理器平台上资源管理的相关问题;针对处理器管理和共享Cache管理中的若干关键问题进行了较深入的探讨。

关键词 多核处理器,共享Cache,任务分配,处理机调度,Cache管理
中图法分类号 TP302 **文献标识码** A

Research of Some Problems about Resource Management on Multi-core Processor Platform

LIU Yu-fang
(Department of Computer Science, Huizhou University, Huizhou 516007, China)

Abstract In the architecture of multi-core processor, there are several cores integrated on a chip. These cores share some kind of hardware resources integrated on the same chip. This paper introduced the development of multi-core processor technology, proposed some problems about resource management on multi-core processor platform, discussed some problems of processor management and shared cache management deeply.

Keywords Multi-core processor, Shared cache, Tasks allocation, Processor scheduling, Cache management

1 多核处理器技术的发展

1.1 CPU进入多核时代

计算机发展的历史表明,推动处理器技术高速发展的原因有二:一是微电子技术的发展,二是处理器体系结构的进化。基于更加先进工艺技术的处理器可以使用更小、更快的晶体管,因而微处理器可以在更高的主频下运行;而更新处理器的体系结构,可以提高它对信息的处理能力,从而提高整个系统的性能和可靠性。

摩尔定律指出,当价格不变时,集成电路上可容纳的晶体管数目,约每隔18个月便会增加一倍,性能也将提升一倍。随后的实践证明了摩尔定律的正确性和有效性。然而,随着微电子技术逐步进入纳米时代,这是人类能掌控的微观极限,摩尔定律是否会继续有效呢?业界并不乐观。

首先,传统的单核处理器,其性能的提高主要依赖于主频的提高。但发展到现在,仅仅提高处理器的主频会产生过多热量,而且无法带来与之相应的性能改善。以先前产品中那种提高速率,处理器产生的热量很快会超过太阳表面。其次,即使是没有热量问题,其性价比也令人难以接受,速度稍快的处理器价格要高很多。

由此可见,无论从性价比还是从性能功耗比方面,传统的单核处理器都遭遇到令市场无法接受的发展瓶颈。因此,依靠体系结构的进化来改善处理器的性能,提高处理能力,是今后支撑计算机整体性能持续提高的核心。随着更多的微体系结构技术集成到处理器中,当前微处理器的发展已经由单核时代过渡到多核时代。

1.2 多核处理器介绍

多核处理器是指在一枚处理器中集成两个或多个完整的计算引擎(内核),整个处理器作为一个统一的结构对外提供服务、输出性能。多核处理器是单枚芯片,能够直接插入到单一的处理器插槽中,但操作系统会利用所有相关的信息,将它的每个执行内核作为分离的逻辑处理器来管理。

多核处理器首先通过集成多个单线程处理核心或者集成多个同时多线程处理核心,使得整个处理器可同时执行的线程数或任务数是单处理器的数倍,这极大地提升了处理器的并行性能。其次,多个核集成在片内,极大地缩短了核间的互连线,核间通信延迟变低,提高了通信效率,数据传输带宽也得到提高。再者,多核结构有效共享资源,片上资源的利用率得到了提高,功耗也随着器件的减少得到了降低。最后,多核结构简单,易于优化设计,扩展性强。这些优势最终推动了多核的发展并逐渐取代单处理器成为主流[1]。

当今主流的多核处理器采用片上多处理器结构(CMP, Chip Multi-Processor)。在这里,每个微处理器都是一个相对简单的单线程微处理器,而且这多个核心间联系非常紧密,甚至共享一级或二级缓存,其核间通过高速总线连接在一起。而更高端的多核处理器则为片上多核多线程结构(CMT, CMP of Multi-threading),它是同时多线程结构(SMT, Simultaneous Multi-Thread)以及CMP结构的结合体,性能非常卓越。现在高端的多核处理器一般都采用CMT结构。

在Intel高级副总裁帕特基辛格看来,从单核到双核,再

本文受广东省自然科学基金项目(9151008901000165)和惠州市科技计划项目(A507.0210)资助。
刘宇芳(1965—),女,硕士,副教授,主要研究方向为软件基础理论、分布式计算、软件工程,E-mail:llyf@hzu.edu.cn。

到多核的发展，证明了摩尔定律还是非常正确的，因为“从单核到双核，再到多核的发展，可能是摩尔定律问世以来，在芯片发展历史上速度最快的性能提升过程”，内核数量每 18 个月翻一倍[2]。

2 多核平台上的资源管理问题

与单核处理器相比，多核处理器内部拥有多于一个的计算核心，那么就存在任务分配、调度、仲裁以及负载均衡等问题。多核之间的任务调度是充分利用多处理器性能的关键，但现在的操作系统还无法有效地支持多核处理器的任务运行。

多核体系结构共享很多硬件资源，如最后一级 Cache、内存控制器、硬件预取单元等，而并行运行的线程对共享资源的竞争往往会造成系统整体性能的下降，这就给资源管理带来了挑战。现在的操作系统缺乏对程序关于片上共享资源使用情况的感知，对如何在相互竞争的线程之间合理分配片上资源缺乏有效的控制；而且，目前操作系统缺乏实际有效的机制管理线程运行时使用片上资源，大多数的管理技术存在额外的开销或者需要特殊硬件的支持，实用价值不大。

3 若干资源管理问题的研究

3.1 处理器管理

3.1.1 任务分配与调度

核的任务分配是多核时代提出的新概念。在单核时代，计算机中只有一个核的计算资源可以使用，因而没有核的任务分配问题。而在多核时代，有多个核的计算资源可以使用，这就要考虑核的任务分配问题。如果系统中有几个任务需要处理，要么一起分配到一个核，要么将它们均匀地分配到各个核，或者按照一定的算法进行分配。

任务分配结束后，接下来需要考虑任务调度。对于不同的核，每个核都可以有自己独立的调度算法来调度执行不同的任务，也可以整个系统使用一致的调度算法。此外，还可以考虑一个线程上一个时间片运行在一个核上，下一个时间片是否还继续在这个核上运行，还是进行线程迁移，让其他核来承载该线程剩下的执行活动？系统在核资源分配不平衡时是否要进行负载均衡调度？还有就是怎样分别调度实时任务和普通任务等等。

对于多核处理器的调度，目前还没有明确的标准与规范。关于多核的任务调度算法主要有全局队列调度、局部队列调度和共生队列调度算法。全局队列调度是指操作系统维护一个全局的任务就绪队列，当系统中有一个核空闲时，操作系统便从全局任务就绪队列中选取一个就绪任务并开始在此核上执行。它的优点是核的利用率较高，缺点是增加了进程上下文切换、锁转换的执行时间，降低了系统的性能。局部队列调度是指操作系统为每个核维护一个局部的任务就绪队列，当系统中有一个核空闲时，便从该核的任务就绪队列中选取恰当的任务执行。局部队列调度的优点是任务基本上无需在多个核间迁移，有利于提高核局部缓存的命中率，缺点是核利用率太低。共生调度方法的基本思想是将访问共享资源较多的任务和访问共享资源较少的任务调度到同一时刻执行，从而最大程度减少资源访问冲突。目前，多数多核操作系统采用了基于全局队列的任务调度算法。

另外一种考虑就是创建主从结构，选择一个处理器来为其他处理器调度。有的系统将主从结构进行扩展，采用单一处理器来处理所有调度的调度策略、I/O 处理和其他系统活动。只有一个处理器访问系统处理数据，从而减轻了数据共享需要，但它的执行效率并不高[3]。

3.1.2 关于负载均衡

在处理器管理中，负载均衡调度的一个重要目标就是缩短任务的平均响应时间，提高系统性能。负载均衡的另一个重要目标是均匀地、充分地利用整个系统的资源。这两个目标是一致的，任务的响应时间依赖于其所运行的系统上的负载，资源的使用越平衡，任务的响应时间就越短。

负载均衡问题是经典的组合优化难题之一，其难度与 Hamilton 问题相当，是一个 NP 完全问题。负载均衡调度的类型通常分为两类：静态调度和动态调度[4]。

静态调度是根据系统的先验知识做出决策，将任务均匀地分配给各个节点，每个节点的任务数都是定值，运行时负载不能重新分配，不随时间变化。这种方法的优点是可以得到最优负载均衡结果，缺点是在现实环境中很难做到。

动态调度通过收集并分析系统的实时负载信息，动态地将不平衡节点上的任务迁移到其他合适的节点，在执行过程中调整负载分布的不均衡性。它具有超过静态算法的执行潜力，能够适应系统负载变化情况，比静态算法更灵活、有效。但是由于必须收集、存储并分析状态信息，因此动态算法会比静态算法产生更多的系统开销[5]。

动态负载均衡算法的组成包括 4 个部分[6]：

①信息策略负责收集整个系统的状态信息；

②转移策略决定节点是否处于适合参加任务转移的状态，即判断某节点是任务发送者还是接收者；

③选择策略决定哪一个任务应该转移。选择一个任务进行转移的基本判别依据是：转移任务的开销比起它的响应时间的减少是划算的；

④定位策略决定把所选择的任务转移到哪个节点上。

3.1.3 中断处理与同步互斥

多核的中断处理和单核有很大不同。多核的各处理器之间需要通过中断方式进行通信，所以多个处理器之间的本地中断控制器和负责仲裁各核之间中断分配的全局中断控制器需要封装在芯片内部，协调工作。

多核处理器对应一个多任务系统，由于不同任务会竞争共享资源，因此需要系统提供同步与互斥机制。另外，一个任务可能由多个并行的、相互作用的进程（线程）组成，进程（线程）间的同步和通信显得更加重要。而传统的用于单核的解决机制已不能满足多核，需要利用硬件提供的“读-修改-写”的原语或其他同步互斥机制来保证，以减少系统在其上的开销，提高并行系统的性能。

3.2 共享 Cache 管理

3.2.1 片上共享 Cache

为了缓解处理器和主存之间巨大的速度差异，在处理器中通常集成高速缓存（Cache）。片上高速缓存一般用作主存储器的临时缓冲区，它可以减少对主存储器的访问次数。片上高速缓存一般是由小容量的高速静态存储器构成，对它的访问通常比对片外主存的访问速度快一个数量级。

目前主流的多核处理器，广泛采用共享最后一级高速缓

存的体系结构设计，即私有一级（或一级和二级）Cache，共享二级（或三级）Cache 的片上存储结构[7]。而传统的单核处理器，是基于完全私有的高速缓存体系结构，在这种结构中，高速缓存只服务于单一的处理器核心。在共享最后一级高速缓存的多核处理器中，最后一级高速缓存同时服务于多个处理器核心，这就与传统单核处理器的私有高速缓存在着本质差别。

共享高速缓存采用物理上分布、但是逻辑上共享的体系结构设计。这种结构具有较低的访问速度，较低的失效率，较高的片上整体容量利用率。另外，每个内存块都被映射到唯一的高速缓存块中，因此简化了 Cache 一致性问题。但是，随着线延迟的加剧，处理器访问不同物理位置的 Cache 延迟差别越来越大。当保存常用数据的高速缓存块距离目标处理器核心的距离相对较远时，平均访问延迟可能会加大。因此，为优化处理器性能，需要把不同位置的 Cache 访问延迟暴露给处理器设计者或者程序员，以便将数据存放在距离核心较近的位置，以降低访问开销。

3.2.2 共享 Cache 划分技术

多核心共享 Cache，这些核心各自占用多大一部分 Cache 对于系统性能影响会很大。Cache 划分就是通过软件或者硬件机制，找到一种在处理器核心之间合理分配 Cache 的方法，从而提高系统的性能。

①利于性能的 Cache 划分

Stone[8] 的研究在多个进程竞争使用的情况下共享 Cache 的分配问题，他认为当进程共享使用共享 Cache 时，LRU 替换策略能够产生近似最优的效果；当进程划分使用共享 Cache 时，各个进程失效率相同的共享 Cache 划分策略会产生最优效果。Qureshi[9] 在 Cache 中选择性地加入计数器，动态获取 Cache 失效率，计算效用度最优的 Cache 划分。对于失效率函数为非凹函数的情况，提出了基于前瞻分配的 Cache 分配算法，其提高了共享 Cache 分配的精度。Tam[10] 提出了一种基于软件 Cache 划分算法，其利用 OS 的页面分配机制来控制应用程序使用 Cache 的大小，在系统 Cache 采用组相连策略的情况下，通过给页面着色，使得页面分组。每次应用进程需要申请页面的时候，只分配同一种颜色的页面给它，那么即使在 Cache 中丢失，硬件也只会替换掉本进程以前的 Cache，而不会影响其他进程的 Cache。

②利于公平性的 Cache 划分

所谓公平性就是所有的线程有相同的机会获得其运行所要求的资源，并保证没有线程饿死。但公平并不意味着所有线程具有相同的吞吐量。通常情况下，公平和效率（其主要指标为系统吞吐量）相互影响和制约，一方的提高有时会带动另一方的改善，但有时会削弱另一方。在某些特殊情况下，则会严重影响另一方导致系统失衡。

Kim[11] 提出了基于公平性的共享 Cache 划分技术。他先提出了 5 个衡量共享 Cache 划分是否公平的度量指标。使用这些度量指标，他又提出了静态和动态两种二级 Cache 划分算法。静态划分算法使用离线的剖面信息获得应用的 Cache 失效率信息，动态划分算法在 Cache 中加入冗余的计数器获得失效率信息。评测结果表明：相对于传统的 LRU 算法，基于公平性的 Cache 划分通常会提高系统的性能；相对于失效率最优的 Cache 划分算法，基于公平性的 Cache 划分有时效果较低。

③利于服务质量的 Cache 划分

因为在多核处理平台上可以同时运行多个应用，而每个应用都可能具有不同的访存模式，比如流式应用和科学计算类应用。因此，传统的把所有的访存平等对待的 Cache 管理机制就会导致 Cache 空间的浪费和数据局部性好的应用的性能降低。对此 Iyer[12] 提出了新的 Cache 管理框架 CQoS。CQoS 能够区分不同的访存流，并且根据应用的需求、局部性和延迟敏感性的不同，赋予不同的访存流以不同的优先级。优先级可以由用户指定，也可以在编译时确定。

Moreto[13] 提出了另一个框架 FlexDCP，其通过不同的缓存配置直接估计应用的性能。这使得操作系统能够将服务质量需求转换成对资源分配的需求，并使得操作系统可以满足不同的服务质量要求，而不需要知道体系结构的内部信息。

3.2.3 Cache 的管理策略

当前，多核处理器通常采用 LRU 策略或者其近似策略作为 Cache 的管理策略。LRU 选择 Cache 中将来被访问的数据块作为淘汰块，是理想管理策略的目标，这样可以使 Cache 缺失率达到最小值。但是，这样的管理策略在实现上非常复杂，因此实现时大多采用近似策略。

Cache 管理策略分成插入策略、淘汰策略和提升策略。插入策略确定新加载进入 Cache 的数据块存放在什么位置上；淘汰策略确定有新数据块需要进入而没有空闲的存放位置时，淘汰哪个数据块；提升策略确定当数据被命中后会将该数据块提升到哪个位置上。

插入策略主要有：将新加载进入 Cache 的数据块插入到 MRU 位，还有一种就是双重插入策略（Bimodal Insertion Policy，BIP），这种方法在使用上述方法的同时，以一定的概率将新进入 Cache 的数据块插入到 LRU 位。而后来被提出的动态插入策略（Dynamic Insertion Policy，DIP）动态地根据应用负载在不同阶段的访存行为，利用锦标赛机制（Set Dueling Monitors，SDMs），从上述两种方法中选择性能较优的 Cache 管理策略。与插入策略对应的淘汰策略则主要是选择 LRU 位上的数据块淘汰出 Cache。

当前主流的提升策略主要有 MRU 提升策略（MRU Promotion，MP）和单步提升策略（Single Incremental Promotion，SIP）两种。前者将被命中的数据块直接提升到 MRU 位，而后者，则将被命中的数据块向前提升一位。与动态插入策略类似，最近也有学者提出动态提升（Dynamic Promotion Policy，DPP）策略，该策略是动态地根据负载访存行为从 SIP 和 MP 这两个策略中选择性能较优的策略作为提升策略。

结束语 较之传统的单核处理器，多核处理器为计算机系统的资源管理带来了新的问题。本文介绍了多核处理器技术的发展，针对多核处理器平台上的处理器分配和调度、负载均衡、中断处理和同步互斥、最后一级共享 Cache 的划分和管理策略等方面进行了研究分析。

参 考 文 献

[1] 黄国睿，张平，魏广博. 多核处理器的关键技术及其发展趋势[J]. 计算机工程与设计，2009，30(10)：2414-2418

[2] 百度百科. 多处理器[EB/OL]. http://baike.baidu.com/view/2797908.htm，2012-04-03

（下转第 463 页）

P(存款)=$p1*p14+p2*p24$=0.099

P(转账)=$p1*p15+p2*p25$=0.0852

P(系统启动)=$p2*p26+p3*p31$=0.010039

根据整个系统的用例使用概率，以软件失效率作为目标值并经专家评定给它们分配重要度权值，按照从小到大的顺序对用例进行排序，计算方式如下：

(1)记 w_i 表示第 i 个用例的重要度权值。p_i 表示第 i 个用例的出现频率。$ratio=p_1-p_2$。

按照软件系统用例出现的频率从大到小排列。设出现频率最高的用例的重要度权重值为 w_1(这里假设为 $w_1=1$)，出现频率次高的用例的重要度权重值为 w_2(这里假设为 $w_2=3$)。

(2)用例之间的相差重要度权值为：

$$\Delta w_i=\frac{(w_2-w_1)*(p_{i-1}-p_i)}{ratio},i=3,\cdots,n$$

(3)于是从第 3 个用例开始，各个用例的权重值为：

$w_i=w_{i-1}+\Delta w_i$

具体情况见表 1。

表 1　各个用例权重的确定

用例名	概率	权值
取款	0.594	1
查询余额	0.198	3
存款	0.099	3.5
转账	0.0852	3.57
修改密码	0.013761	3.92
系统启动	0.010039	3.94
总计		18.93

根据表 1 进行可靠性分配计算。假设该系统要求可靠性目标为失效率为 0.001 失效数/小时，则可以根据式(3)计算出各个用例对应的失效率分配值。

(1)取款用例的失效率分配值为：

$f1=(1/18.93)*0.001=0.0000528$

(2)查询余额用例的失效率分配值为：

$f2=(3/18.93)*0.001=0.0001584$

(3)存款用例的失效率分配值为：

$f3=(3.5/18.93)*0.001=0.0001848$

(4)转账用例的失效率分配值为：

$f4=(3.57/18.93)*0.001=0.000188496$

(5)修改密码用例的失效率分配值为：

$f5=(3.92/18.93)*0.001=0.000206976$

(6)系统启动用例的失效率分配值为：

$f6=(3.94/18.93)*0.001=0.000208032$

这表明要想满足整个软件系统的失效率为 0.001 失效数/小时的目标，必须要求取款用例的失效率控制在 0.0000528 失效数/小时范围内。其他用例都有类似要求，这种结果对软件测试人员具有很好的参考价值。

结束语　UML 是目前最流行的软件建模语言。一个正确的 UML 用例模型能如实地反映软件系统需求规格说明中所规定的各种需求。利用用例的出现频率来计算各个用例所应该承担的可靠性指标，可为后续阶段的软件开发和软件测试提供很好的参考价值。但是这种方法也有一定的缺陷，主要是确定重要度权值需要依赖专家的评审，而对具体软件开发成员该承担多少可靠性指标任务无法衡量。所以我们将利用 UML 用例图、顺序图、交互图、状态图、类图等模型图，依据 UML 模型图所具有的层次结构特点对软件可靠性指标逐级分解，从而确定每个类该承担多少可靠性指标任务，最终对每个软件开发成员在可靠性方面所应该承担的责任进行很明确的规定，以真正体现“软件的可靠性是设计出来的”精神。

参 考 文 献

[1] 孙志安，裴晓黎，宋昕，等. 软件可靠性工程[M]. 北京：北京航空航天大学出版社，2009，3：160-175

[2] Eriksson H-E, Penker M. UML Toolkit[M]. Publishing House of Electronics Industry, 2004, 10: 150-230

[3] Singh H, Cortellessa V, Cukic B, et al. A bayesian approach to reliability prediction and assessment of component based systems[C]// Proc. of 12th International Symposium on Software Reliability Engineering(ISSRE'01). 2001

[4] Corteliessa V, Singh H, Cukic B. Early reliability assessment of UML based software models[C]// WOSP'02. Rome, Italy, July 2002: 24-26

[5] Khalaj M, Makui A, Tavakkoli-Moghaddam R, et al. Systematic Approach to Calculate Risk and Reliability in Uncertainty Condition[J]. Journal of Basic and Applied Scientific Research, 2012, 2(1): 573-582

[6] Hu Weng-sheng, Deng Zhou-hui, Hong Yi. A Method of FTA Base on UML Use Case Diagram[C]// ICRMS' 2011. June 2011: 12-15

[7] Fan Lin-bo, Ma Zhi-gang. Tendency Analysis of Software Reliability Engineering[C]// ICRMS'2011. June 2011: 12-15

(上接第 443 页)

[3] 多核系列教材编写组. 多核程序设计[M]. 北京：清华大学出版社，2007(9)：201-203

[4] Kunz T. The influence of different workload description on a heuristic load balance scheme[J]. IEEE Trans on Software Engineering. 1991, 17(7): 725-730

[5] 王力生，毛昀波. 多处理机系统的负载平衡模型设计[J]. 单片机与嵌入式系统应用，2008(4)：10-13

[6] 胡亮，徐高潮，鞠九滨. 一个基于收益与开销的作业选择策略[J]. 软件学报，1998，9(4)：280-283

[7] 方娟，蒲江，张欣. 片上多核处理器共享 Cache 划分的公平性研究[J]. 计算机工程与设计，2010，31(15)：3413-3415，3517

[8] Stone HS, Turek J, Wolf J L. Optimal Partitioning of Cache Memory[J]. IEEE Trans on Computer, 1992, 41(9): 1054-1068

[9] 所光，杨学军. 多核处理机系统 Cache 管理技术研究现状[J]. 计算机工程与科学，2010，32(7)：65-68

[10] Tam D, Azimim L, Stamm M. Managing shared L2 Caches on multi-core systems in software[C]// Workshop on the Interaction between Operating Systems and Computer Architecture. 2007

[11] Kim S, Chandra D, Solihin Y. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture[C]// Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques. 2004: 111-122

[12] Iyer R. CQoS: a Framework for Enabling QoS in Shared Caches of CMP Platforms[C]// Proc of the 18th Annual Int' l Conf on Supercomputing. 2004: 257-266

[13] Moreto M. FlexDCP: a QoS framework for CMP architectures. ACM SIGOPS Operating Systems[J]. 2009, 43(2): 86-95