软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(7):2152−2176 [doi: 10.13328/j.cnki.jos.005580]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

实时多核嵌入式系统研究综述*

陈 刚, 关 楠, 吕鸣松, 王 义

(东北大学 计算机科学与工程学院 智慧系统实验室,辽宁 沈阳 110819)

通信作者: 陈刚, E-mail: chengang@cse.neu.edu.cn

摘 要: 随着计算机系统与物理世界的结合越来越紧密,实时系统需要承担越来越复杂的运算任务.多核处理器的兴起为同时满足实时性约束和高性能这两方面的需求提供了可能.基于多核处理器的实时嵌入式系统的研究已成为近几年研究的热点.对现有的面向实时多核嵌入式系统的研究工作进行了综述,介绍了实时多核嵌入式系统的关键设计问题,从多核共享资源干扰及管理、多核实时调度、多核实时程序并行化、多核虚拟化技术、多核能耗管理和优化等几个方面对现有研究工作进行了分析和总结,并展望了实时多核系统领域进一步的研究方向.

关键词: 实时多核系统;共享资源干扰;实时调度;并行软件;虚拟化;能耗优化

中图法分类号: TP316

中文引用格式: 陈刚,关楠,吕鸣松,王义.实时多核嵌入式系统研究综述.软件学报,2018,29(7):2152−2176. http://www.jos.org.cn/1000-9825/5580.htm

英文引用格式: Chen G, Guan N, Lü MS, Wang Y. State-of-the-Art survey of real-time multicore system. Ruan Jian Xue Bao/ Journal of Software, 2018,29(7):2152−2176 (in Chinese). http://www.jos.org.cn/1000-9825/5580.htm

State-of-the-Art Survey of Real-Time Multicore System

CHEN Gang, GUAN Nan, LÜ Ming-Song, WANG Yi

(Smart Systems Laboratory, School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: As computer systems are more and more closely integrated into the physical world, real-time systems are required to perform more and more complex computation tasks. The development of multi-core processors makes it possible to fulfill the requirements of both real-time constraints and high computation demands. The research on real-time multicore system has attracted a lot of attention in recent years. This paper presents a survey on the research of real-time multicore system. The survey first introduces the main research problems and challenges. Then, a detailed review is provided covering the various topics, such as shared resource interference, real-time scheduling in multi-core system, parallel real-time software design, multicore virtualization, and power management under real-time constraints. Open issues and research directions are also identified in this survey.

Key words: real-time multi-core system; shared-resource interference; real-time scheduling; parallel software; virtualization; power optimization

随着信息技术的飞速发展,嵌入式系统正在以前所未有的速度渗透到人类生活的方方面面.调查数据表明,目前世界上超过 98%的微处理器都用于嵌入式系统[1].随着物联网(Internet-of-Things)时代的到来,嵌入式系统的地位还将进一步提升.嵌入式系统往往应用在过程控制、环境控制、关键基础设施控制系统等领域.这类系统本质上是复杂实时系统,系统的正确性不仅取决于逻辑计算结果,更取决于结果产生的时间,即系统必须在规

* 基金项目: 国家自然科学基金(61702085, 61532007, 61772123); 中央高校基本科研业务费(N161604002); 装备预研教育部联合基金青年人才基金(6141A020333)

Foundation item: National Natural Science Foundation of China (61702085, 61532007, 61772123); Fundamental Research Funds for the Central Universities (N161604002); Ministry of Education Joint Fundation for Equipment Pre-Research (6141A020333)

收稿时间: 2017-04-27; 修改时间: 2018-02-08; 采用时间: 2018-03-30; jos 在线出版时间: 2018-04-27

CNKI 网络优先出版: 2018-04-27 14:58:10, http://kns.cnki.net/kcms/detail/11.2560.TP.20180427.1457.008.html

定的时间范围内正确地响应外部物理过程的变化[2].如果系统的时间行为不能满足要求,则可能导致灾难性后果.因此,如何保障系统的实时性是可信系统设计中的关键问题.

同时,应用需求的提升导致嵌入式系统日益复杂,集成度不断提高.传统的嵌入式设计方法通常将不同的应用(或功能)部署到不同的独立硬件上执行.而对于高度复杂的未来嵌入式系统,这一方法学已经无法满足开发效率的需要,同时更难以满足嵌入式系统对性能、体积、重量以及功耗等方面的复杂需求.因此,实时系统将面临实时性约束和高性能集成设计需求的双重压力.多核处理器的出现,使得在同一平台上集成多个应用成为可能,为未来实时嵌入式系统设计带来了希望.例如,在汽车电子领域中,一台高端汽车上有超过 100 个电子控制单元(electronic control unit,简称 ECU),每一个 ECU 负责一个特定的功能(如加速、制动等).在未来的系统设计中,多个控制功能可能被集成在一个多核处理器上,从而降低系统开发和维护成本,减小电子系统的体积与功耗[3].因此,在可预见的未来,多核处理器将成为各类计算系统的标准硬件,实时系统在多核硬件上的集成也将成为必然趋势.

尽管多核处理器可以为实时系统带来诸多的好处,但是多核处理器在体系结构和系统管理上有不同于单核处理器的固有特性,在多核平台上高效地设计和部署实时系统仍然面临着一些挑战.到目前为止,面向可认证(certifiable)高可信实时系统的软件设计仍然停留在单核系统上[4].例如,当调度安全关键任务时,航空系统设计者往往会关掉其他核心,只留一个核心运行安全关键任务,以保证其安全性[4].这种保守的使用方式往往使得多核平台的高性能优势难以得到充分的体现和发挥.如何高效地将多核计算平台应用于实时系统设计中,在国内外工业界和学术界仍然是一个开放性的研究问题.

目前,基于多核处理器的实时嵌入式系统设计问题正得到各国学术界和工业界的密切关注.欧美许多国家已将多核嵌入式系统的研究纳入该国基础研究战略计划,并大力支持相关的研究中心,代表性例子有美国伊利诺伊大学(UIUC)的 UPCRC 研究中心[5]、加州大学伯克利分校的 ParLab 实验室[6]、瑞典的 UPMARC 研究中心[7]等.同时,实时多核系统也成为各国科研投入的重点,比如欧盟著名的 FP7 项目框架已经把实时多核嵌入式系统的设计与分析作为热点研究方向进行了立项(MultiPARTES[8]、ACROSS[9]、CERTAINTY[10]、PROXIMA[11]).国内在多核系统设计方面的工作主要集中在学术研究领域.相关高校、研究院所在嵌入式系统、普适计算、形式化验证、可信软件等方面积累了雄厚的研究基础.部分高校在嵌入式微处理器等方面拥有多年研究经验,开发了高性能嵌入式龙腾系列芯片.相关技术在汽车电子等领域实现了成果转化.

针对多核处理器在实时系统应用中产生的问题,国内及国际学术界、工业界从系统架构、调度策略、实时程序并行化、能耗管理等关键问题上提出了许多解决的途径和办法.基于多核处理器的实时嵌入式系统的研究是一个覆盖面很广的研究领域,需要解决多方面的问题.文献[12]对面向通用计算的多核系统研究进行了综述,但是针对实时计算的多核系统设计与分析的研究问题还没有进行深入、系统的阐述.针对上述问题,本综述介绍了多核系统实时系统设计在不同层面上所面临的问题以及挑战,总结现有主流技术以及其优缺点.并从多核共享资源管理技术、多核时间分析和实时调度、实时程序并行化、多核能耗管理等方面对多核实时嵌入式系统设计问题进行全面的分析总结,为相关领域研究者提供参考.

1 多核体系结构与实时多核关键问题

1.1 多核体系结构

从所包含的处理器核结构的角度来看,多核处理器分为同构多核处理器和异构多核处理器.同构多核处理器中处理器芯片内部的所有内核结构完全相同,各个内核具有等同的地位.异构多核处理器中异构多核处理器芯片内部采用多种功能不同的内核,例如,目前 ARM 公司推出的 Big-Little 架构处理器.在实际工业应用中,ARM 的 Cortex-R 与 Freescale 的 PowerPC 是两种比较常用的实时处理器架构,并且占有了比较大的市场份额[13].最近,NXP 半导体公司推出了针对汽车电子领域的实时多核处理器 Qorriva MPC5643L.该处理器是目前首次通过 ISO 26262 功能安全标准认证的多核处理器,是基于 PowerPC 架构的双核处理器[14].Wind River 公司

的 Parkinson 在文献[15]中对在 NXP QorIQ P4080 八核处理器上部署多独立安全等级(multiple independent levels of security,简称 MILS)的航空系统的过程进行了阐述.

目前,关于实时多核系统的研究工作重点集中在同构多核处理器架构上.因此,本文重点也主要放在同构多核处理器的相关研究工作上.图 1 是一个典型的嵌入式多核体系结构.多核芯片中所有的核心都有一个私有的 L1 缓存(L1-cache),并且共享 L2 缓存(L2-cache).整个多核芯片通过共享的存储器总线(bus)与主存储器控制器(memory controller)相连.每个核心与外部存储器(off-chip memory)之间的通信请求都是由主存储器控制器和总线总裁器来进行处理的.从图 1 中可以看出,多核处理器在体系结构和系统管理上有不同于单核处理器的固有特性.在单核处理器时代,微处理器的发展路线是在保证微体系结构对用户透明的基础上不断提高芯片的主频.因此,系统性能提升这一任务主要由硬件设计者来承担;软件设计者只需将现有软件不加修改地部署到频率更高、速度更快的 CPU 上,就可以直接获得性能上的提升.而随着多核处理器的出现,软件设计者将无法继续享受这种免费的系统加速,他们必须思考如何从根本上改变软件的体系结构,设计出可以同时运行在多个处理核心上的并行软件,以充分利用多核处理器提供的计算能力.因此,程序并行化是多核硬件和实际应用之间的第一个挑战.此外,在传统的单核系统中,共享资源主要是 CPU.一个任务一旦获得 CPU 执行权,即获得对系统大部分其他硬件资源的控制.而在多核系统中,资源的共享存在于多个维度.并行执行的任务除了需要共享多个处理核心外,还需要大量、细粒度的共享末级 Cache、总线、内存控制器、主存等硬件资源.多个维度上的共享行为造成了任务之间显式或隐式的相互依赖与干涉,而这种依赖关系的粒度之细,复杂度之高,远非单核软件所能比拟.任务间的复杂干涉导致系统整体性能下降、时间行为高度不可预测、逻辑正确性难以保证.随着集成在同一芯片上的处理核心数量的继续增长,上述问题将会进一步加剧.

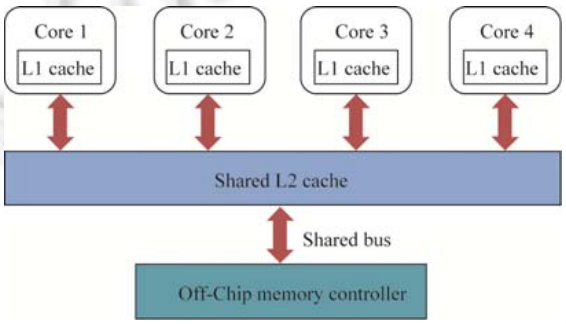

Fig.1 Multicore system architecture

图 1 多核体系结构

1.2 实时多核系统的关键问题

从上面的讨论可以看出,能否将并行应用成功地部署到多核硬件上,在确保程序逻辑正确的同时,使性能、实时性、功耗等非功能特性都得到保证,是计算机软件设计者,尤其是实时嵌入式软件设计者所面临的关键问题.问题具体体现在以下几个方面.

- 如何在多核平台上保证系统行为的可预测性(predictability).由于实时系统,尤其是硬实时系统,对系统的时间行为特性有严格的要求,通常要在系统实际运行之前能够对系统的时间特性进行完整的离线分析.但是,在多核平台上,并行执行的任务除了需要共享多个处理核心外,还需要大量、细粒度的共享多层 Cache、总线、内存控制器、主存等多个维度的硬件资源(如图 1 所示).多个维度上的共享行为造成了任务之间显式或隐式的相互依赖与干涉,而这种依赖关系的粒度之细,复杂度之高,远非单核的系统所能比拟.任务间的资源无序竞争所引起的复杂干涉会使时间行为高度不可预测,从而为实时分析带来困难.
- 如何设计面向多核系统的实时调度分析技术,从而在保证系统行为的时间正确性的同时,充分利用多核处理器所提供的强大计算能力.自 20 世纪 90 年代末实时调度问题出现以来,经过学术界近年的努力,面向单处理机系统的实时调度技术已经趋于成熟,并出现了一系列重要的理论成果.然而,单处理机模型下的绝大多数理

论成果目前还无法应用于多处理机器模型.

• 如何在多核上并行化地实时应用程序,在充分发挥多核处理器的运算能力的同时,保证实时应用程序的实时性.虽然并行程序设计在通用和高性能计算机系统领域中已经成为主流,但在实时系统领域还未得到足够的重视.现有的面向多核处理器实时系统设计与分析研究工作主要针对串行程序.目前,面向并行程序的多核实时系统方面的研究刚刚起步,还未有成型的理论体系与技术框架.

• 如何在实时性能约束下优化系统功耗.随着 VLSI 技术的迅猛发展,芯片工艺已经从 1995 年的 500nm 发展到目前的 7nm 工艺[16],晶体管尺寸的不断缩减以及日益复杂的电路结构导致芯片的功耗密度在急剧上升,随之带来的问题是 Power Wall 问题,能耗短板将严重制约多核系统整体性能的提升.处理器能耗问题不仅引发难以解决的散热问题,提高了硬件的封装和散热成本,导致系统可靠性的下降,限制了处理器速度的进一步提高,反过来也会影响系统的性能.因此,若要继续追求高性能系统,首先要解决系统的功耗问题.

从下一节开始,针对上面几个关键问题,本文将从多核共享资源管理技术、多核时间分析和实时调度技术、实时多核虚拟化技术、实时程序并行化技术、多核能耗管理技术等几个关键技术入手进行深入探讨,总结现有主流技术及其优缺点.

2 多核共享资源管理技术

虽然多核技术使得更多的任务可以并行进行,但是对于共享资源的访问是整个系统性能提升的瓶颈.例如,多内核系统中,内存带宽是共享的,这使得内存成为了临界资源,于是带来了存储器延迟、系统复杂性、饥饿等问题.另一方面,多核技术是一个整体架构,并行运行在不同核心上的任务在各个共享资源层次上相互交叉影响,对共享资源的无序抢占会带来不确定性干扰问题.这种复杂干涉会使时间行为高度不可预测,使得多核系统实时分析变得极其困难.近期,美国联邦航空管理局 FAA 以 NXP PowerPC MPC8572 双核处理器为例,研究了将多核技术用于安全关键实时系统有可能存在的问题以及可能产生的后果[17].研究结果表明,应用的执行性能会受到其他核心上并行运行的其他应用的严重影响,进而导致拒绝服务(DOS)等后果.针对上述问题,本节从 cache、总线、片外存储器等多个不同维度阐述不可预测性(unpredictability)干扰源,探讨多核共享资源所带来的相应的新问题和新挑战,阐述针对这些问题的最新的解决方案,并建立多核共享资源管理这一研究领域的全貌.

2.1 共享缓存干扰

处理器和主存储器之间的速度存在较大的差距,理论上一次访问主存储器行为需要花费 50~100 个时钟周期,而处理器在每个时钟周期都可能访问主存.Cache 是处理器片内的一个高速、小容量的存储器件,用来临时存储处理器经常访问的数据和指令,以缩小处理器和片外存储器的速度差异.在一些多核处理器中,通常末级 cache(last level cache,简称 LLC)供所有处理核心共享.如果数据或指令在末级 cache 中不命中(cache miss),则需要从片外存储器载入所需内容.由于这一过程需要通过内存总线,故访问延迟相对较高.如果命中(cache hit),则从 cache 中取出数据送至 CPU,故访问延迟相对较低.根据局部性原理,程序通常会在某个时间段内集中访问一部分指令或数据,相当数量的访存会在 cache 中命中.因此,cache 能够显著提高程序的执行性能.

同时,cache 的引入也会带来一些负面影响.在单核系统中,任务之间的相互抢占使得 cache 的行为分析问题变得错综复杂.被抢占的任务的缓存块(cache line)会被其他任务置换,导致被抢占的任务的 cache 不命中率发生变化,进一步影响了系统时序.因此,在系统分析过程中,需要对缓存相关抢占延迟(cache-related preemption delay,简称 CPRD)进行分析.在多核系统中,问题将变得更加复杂,除了需要考虑 CPRD 问题以外,共享 cache 还会带来多个核心之间对共享 cache 的无序竞争.图 2 阐述了多核系统中共享 cache 无序竞争的问题.在这个例子中,核心 1 需要访问数据元 b0.因此,数据元 b0 被核心 1 加载到共享 cache 中某个位置,如图 2 中蓝色框所示.需要注意的是,数据元 b0 可能在将来某个时刻会被核心 1 重用.在下一个时刻,核心 2 需要访问数据元 b1,而导入的数据元 b1 刚好被映射到数据元 b0 的位置.此时,数据元 b0 被数据元 b1 替换,并被逐出共享 cache.这种替换会

给后面核心 1 对数据元 b0 的再次访问造成 cache 不命中.在多核系统中,一种可能的 cache 干扰情况是一个核心上会持续驱赶对另一个核心有用的缓存块,而这种驱赶不仅不会给自身带来显著的性能提升,还会使得系统 cache 不命中率上升,导致系统性能下降.在文献[18]中,通过在四核处理器上运行实验,其结果数据表明,与任务单独运行时的完成时间相比,核心间的共享缓存干扰会将任务的完成时间增加 40%.更重要的问题是,核心之间这种无序的抢占时序导致系统行为分析极为困难.任务间干涉导致一个任务的 WCET 不再仅仅依赖任务自身,同时也受到与其并行执行的其他任务的影响.从状态空间的角度来讲,整个系统的状态空间本质上是所有并行任务状态空间的乘积,状态空间随着核心数量的增加呈指数爆炸.因此,在多核共享 cache 体系结构下进行 cache 行为分析,其难度远远超过单核 cache 行为分析.

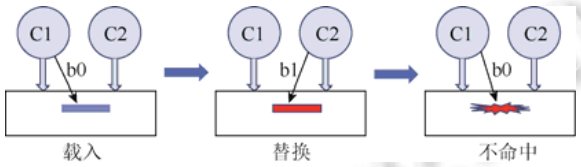

Fig.2 The interference of shared cache

图 2 共享缓存干扰

针对这一问题,文献[19–22]提出了一些静态分析方法,在有限的程度上解决了共享缓存对 WCET 的影响问题.考虑到问题的复杂性,这些解决方案都需要引入一些假设来简化问题.同时,这些简化的假设也限制了所提出解决方案的可用性.具体来说,这些解决方案的局限性主要表现在:(1) cache 结构过于简单,只考虑直接映射 cache 或者相联度(associativity)很小的 cache;(2) 忽略了数据 cache 的影响,假设需要存取的数据总是存在于数据 cache 里面,因此,大部分工作只分析指令 cache 的行为;(3) 没有考虑 cache 的写操作的影响;(4) 这些分析方法都没有在实际的多核计算平台上加以验证.

尽管存在上述分析方法,但目前多核 WCET 分析领域仍然还没有较好的干涉行为分析方法出现.鉴于多核共享 cache 静态分析的难度,一部分学者转向从设计的角度来避免或减少核间的干涉行为,这样就可以简化时间分析的难度.目前主要的方法是通过 cache 空间隔离来消除核间共享 cache 的干扰.Cache 空间隔离的主要技术手段有如下几类.

- 通过 cache 划分(partitioning)方式将共享 cache 分成若干区域,并将这些区域划分给不同的核心,核心之间的 cache 区域不会发生重叠,因而不会发生干涉.Cache 划分可以分别通过软件和硬件的方式来实现.

(a) 软件实现机制是利用操作系统的内存分配机制,通过 cache 染色技术[23]将共享 cache 在 cache set 维度上分成若干区域(如图 3(a)所示),使得不同核心上的程序映射到共享 cache 上不产生交叠地址空间,因而不会发生干涉.文献[24–26]通过使用 cache 染色技术实现了 cache 空间上的隔离,提高了多核 cache 的可预测性,而且还在一定程度上提高了系统的性能.然而,cache 染色技术在重新页染色时需要大量的时间开销.因此,cache 染色技术一般不能用于动态隔离.文献[27]通过实验对 cache 重染色的时间开销进行了实际测量,发现 cache 重染色的时间开销会占到整个任务执行时间的 7%.

(b) 硬件实现机制往往是在 cache way 维度上进行划分(如图 3(b)所示),通过处理器中特殊的硬件支持来实现 cache 的空间划分支持[28].相比于 cache 染色技术,cache 资源的分配和隔离完全是由硬件来完成的,cache 的重新划分所需要的时间开销比较小.因此,硬件实现机制适用于动态划分.基于任务的动态 cache 分配策略,文献[29–31]分别提出了一种 cache 感知的调度方法,根据任务对 cache 需求多样性的特点,为每个任务分配合适的 cache,使得系统在保障实时性的约束下性能最优.

- Cache 锁定技术[32]也可用于多核共享 cache 隔离,以消除或减少任务间干涉.cache 锁定技术通过处理器中相关硬件的支持,把经常访问的内容锁定到 cache 中.通过这种锁定机制,任务运行时 cache 访问是否命中就是确定的,可以离线分析.如果被锁定的内容选择合理,cache 锁定技术不仅可以保证任务的实时性,还可以进一步提高程序的性能.此外,cache 旁路(bypass)技术也可以达到类似的目的[33].Cache 锁定是选择最常用的数据载入

cache,而 cache 旁路技术则是通过分析程序中不会被反复访问的数据,在执行过程中禁止其载入 cache,从而避免这类访存对其他访存造成的干涉.

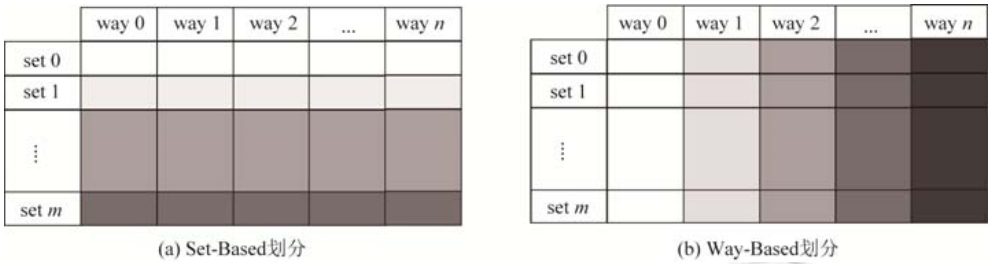

Fig.3 Two methods of cache partitioning

图 3 两种 cache 划分方式

2.2 共享总线干扰

如图 1 所示,每个核心需要经过共享总线访问片外存储器,会造成共享总线干扰,增加任务的响应时间.在多核计算平台上,如何在考虑共享总线干扰情况下进行响应时间分析仍然是一个具有挑战性的任务.其主要原因在于:

(a) 任务在哪个时刻访问总线是不受操作系统调度器控制的,而是由 cache 不命中来决定.

(b) 任务的总线访问请求没有携带任务的优先级属性,导致总线会根据任务的访问重新排序,有可能造成高优先级任务的访问请求服务时间晚于低优先级任务.

(c) 任务总线访问请求的延迟,不仅取决于总线的仲裁机制,即决定总线访问服务顺序,还取决于当前时刻并行任务的总线访问量.

为了解决这一问题,目前大部分的研究[34−36]主要集中在基于时分复用(time division multiple access,简称 TDMA)仲裁机制的总线时间分析.TDMA 仲裁机制将总线时间划分成固定时间大小的时间片,每个核心只允许在固定的时间片内使用总线进行访问.这种访问机制可以解决上面所提到的几个问题.Rosen 等人在文献[37]中描述了一种在多核计算平台上执行 TDMA 总线调度的方案.基于 TDMA 总线,该方案首先离线计算出总线调度表,使得所有的任务满足截止期约束.在系统运行之前,将总线调度表写进总线仲裁器上的存储器中.在系统运行时,根据该调度表协调并控制总线访问申请,从而避免总线干扰.

此外,一些研究者对非 TDMA 总线的时间分析技术展开了研究.基于通用的事件驱动模型,Schliecker 等人在文献[38]中提出了一种方法,能够解决多核共享资源负载界定问题.在该模型中,他们假设在一个时间窗口中共享资源的访问次数是可以界定的,即假设总线最大访问次数和最小访问次数是给定的.在固定优先级调度的情况下,通过分析高优先级任务的资源访问模式可以推导出最差共享资源干扰.文献[39]提出了基于时间自动机(timed automata,简称 TA)的总线分析方法,但是复杂的总线架构往往会导致时间自动机状态空间急剧增大.文献[40]提出了一种结合抽象解释和模型检测技术对分时总线和非分时总线进行分析的方法.该方法是对多核共享资源抽象方法与分析技术的一个有益探索,但是尚未彻底解决分析规模可伸缩性问题.

最近,文献[41]提出了内存访问截停技术,该方法可以通过软件来控制外存的访问.在该方法中,每个任务在固定的时间周期内其访问配额是固定的.通过监视末级 cache 不命中率,软件可以监控任务对总线的访问配额是否用完,一旦用完,立即将任务挂起,使得任务停止对总线的访问,从而避免总线干扰的产生.直到下一个时间周期的到来,任务的总线访问配额会重新分配.这种任务截停机制给任务之间的总线干扰上限的界定提供了一种有效的方法.同时,该文作者也给出基于该方法的 WCET 的分析方法.

2.3 共享片外存储器干扰

在现代的多核系统中,处理器性能与内存性能间的不均衡发展导致当前内存存储速度严重滞后于处理器的计算速度,从而形成 Memory Wall 问题.随着系统核心数目的增加,系统对内存带宽需求也在不断增加,

Memory Wall 问题变得越来越突出,并严重阻碍了多核系统的性能发挥.此外,并行实时任务共享内存干扰会降低 DRAM 内存系统的响应速度,进而拉大访存延迟,扩大处理器与内存之间在性能上的差距,使得 Memory Wall 问题越来越严重,从而导致系统性能下降,同时也使得任务的实时性能得不到保障.共享片外存储器的不可预测性主要来自于 DRAMs 存储器自身内部的体系结构、页策略、存储器调度的影响.下面,本节将从这 3 个方面分别加以阐述.

图 4 所示为一个典型的 DRAM 体系结构.在 DRAM 结构中,每一个 Bank 都包含一个行缓冲(row buffer),Bank 的 Row Buffer 是整个访存操作过程中的核心部件,只有将数据读入到行缓冲才能进行读写操作.访存时先通过行地址确定 Bank 中的一行,然后将选中行的所有内容都缓存到 Bank 的 Row Buffer 里,最后通过列译码器译出的列地址在 Row Buffer 选中某一列,读取单元(cell)里存储的内容.Row Buffer 对 Bank 中的数据起到了缓存的作用.DRAMs 在进行读取数据之前,数据必须以整行的形式从 Bank 中被读取到行缓冲之中,这一过程称为行激活(active row).但在行激活操作之前,需要保证此时行缓冲必须处于空闲状态,也就是需要进行预充电(precharge)操作.在行激活操作之后,内存控制器将发出读命令或者写命令,以及需要读取的列地址.当要访问一个新的行时,内存控制器必须首先发出预充电命令,将整个 Bank 置为 Precharge 状态.此过程所花费的时间被称为 tRP.此外,为了保证 DRAM 系统的正常运行,内存控制器发出的操作命令必须遵守一些严格的时序,否则,容易产生数据错误.

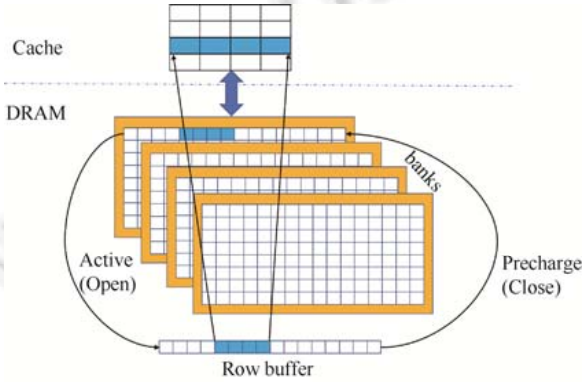

Fig.4 DRAM architecture

图 4 DRAM 体系结构

存储器 DRAM 的体系结构和复杂的时序使得存储器访问请求的响应时间变得不稳定,主要原因来自于以下 3 个方面.

(a) 响应时间取决于访问请求的数据是否在 open row 中,如果在,则访问请求会被立即响应,否则,首先需要关闭当前 row 并打开被访问的 row;

(b) DRAM 读写切换需要消耗一些时钟;

(c) 在 DRAM 中,为了防止数据的丢失,存储器偶尔需要在执行访问请求之前进行刷新操作,而这些刷新操作的时间可能要比执行访问请求还要长,因此会增加响应时间.

由于以上 3 个因素的影响,访问请求时间变得非常不稳定,可以从几个 cycles 变化到几十个 cycles.因此,共享 DRAM 干扰会在很大程度上影响任务的执行时间.

页策略决定了 DRAM 控制器应当什么时候执行 precharge 指令.目前,DRAM 控制器一般有 open page 和 close page 两种页策略.Open page 策略在存储器访问的时候一直保持 open row 状态,希望下一个访问请求也在当前的 row 中,从而可以降低访问延迟.当程序局部性较好时,open page 策略有助于产生 row 命中,从而能够提升系统的性能.同时,对于 row 不命中的情况,open page 策略也会带来额外的 precharge 和 activation 时间开销.与 open page 策略相反,close page 策略在存储器访问结束后会立即关闭已经激活的 row,以减小打开其他 row 的时间开销.Close page 策略比较适合于实时系统[42,43],因为实时系统往往需要保证在最差情况下的访问内容局部

性,特别是在多个程序并行运行的情况下.

存储器调度主要负责对存储器访问进行排序并产生符合 DRAM 时序约束的时序指令.文献[44]提出了可预测存储器模式(predictable memory pattern)这一概念,采用静态和动态调度相结合的手段来提高存储器的存储效率.在离线阶段,首先计算出存储器在各个模式下的指令序列,以保证每一个模式的时间可预测性.存储器有读模式、写模式、读/写切换模式、写/读切换模式、刷新模式这 5 种.在运行时阶段,这些模式会根据到来的访问请求调用不同的指令序列组合,完成相应的操作.通过将原本相互依赖的 DRAM 指令抽象成相互独立的模式,存储器调度和性能分析变得更加简单.

3 多核系统时间分析与实时调度技术

任务调度是实时多核系统设计的一个重要组成部分,它影响着整个系统的资源使用率.实时调度的本质是对 CPU 资源进行仲裁和分配,可调度性分析的目标是在给定的调度策略下,分析系统中的所有任务是否能在截止期之前完成.嵌入式系统实时调度分析方面的研究可以追溯到 20 世纪 60 年代末和 70 年代初.其中,Liu 与 Layland 在其经典论文[45]中奠定了系统实时性能分析的基础.文中假设一个由若干个相互独立的周期性任务组成的多任务系统运行在一个单处理器上.经过几十年的研究,面向单处理机系统的实时调度技术已经趋于成熟,并出现了一系列重要的理论成果.然而,单处理机模型下的绝大多数理论成果无法应用于多处理机器模型.例如,对于可抢占周期性任务,最早截止期优先调度算法是单处理机模型上的最优算法,其可以达到最大资源利用率界限[45],但在多核处理机模型上非但不是最优,而且会导致极低的资源利用率界限.因此,多核处理器的实时调度与分析面临着一些新的挑战.任务调度问题作为实时系统研究的一个经典问题,一些相关综述[46,47]从各个角度对其进行了详细的介绍.本节主要关注多核系统时间分析、实时系统建模和调度问题,将从时间分析、负载建模和调度算法分析等几个角度对近年来多核计算平台上任务调度的最新研究进展进行总结.

3.1 多核最坏情况执行时间分析

系统实时特性分析的第 1 个层次是分析每个软件任务的最坏情况执行时间(worst-case execution time,简称 WCET).这一信息将作为系统级时间分析,也就是可调度性分析的输入.从 20 世纪 90 年代末起,基于抽象解释(abstract interpretation,简称 AI)[48]和隐式路径枚举(IPET)[49]的方法逐渐成为程序执行时间分析领域的主流.虽然 AI+IPET 的分析框架对于单核处理器上程序的分析效果很好,但当应用此框架对多核处理器上程序进行分析时却遇到了极大的挑战.其主要原因是,各个处理核心共享许多软硬件资源,由于同时运行的各个程序对这些资源的竞争和交错访问,使系统的时间行为变得高度不可预测.研究者们围绕不同类型的共享资源对多核程序执行时间分析进行了研究.

主流多核处理器通常采用共享的末级缓存(cache),多个核心上的任务并行地访问末级缓存,造成大量细粒度的任务间访问冲突.因此,多核系统中程序执行时间分析的一个关键问题是能否对共享 cache 上的干涉情况进行有效分析.针对考虑 cache 行为影响的实时程序执行时间分析问题,文献[50]已经给出了全面的综述,本文对这一部分就不再详细展开.多核处理器上另一个重要的共享资源是存储器及相关接口硬件.许多研究者针对实时性需求设计了新型的存储器架构来帮助准确计算程序的执行时间[51].例如,欧盟的 MERASA 项目提出了一种“可分析”的内存控制器的设计方案[51]:该方案采用 Closed Page 机制,因此分析每个访存时不必关心其前面的访存;为系统中的每个任务单独设立访存队列,以消除任务间干涉;对于硬实时任务,采用轮询方式调度,以保证每个访存的延迟存在上限.这种设计的主要贡献是保证了访存延迟存在上限且可计算.最近,文献[52]针对商业化的内存控制器提出了一种可减少存储器干扰,并提供最差情况下的干扰延迟的界定方法.同时,该文作者在多核 Linix\RK 运行环境中验证了方法的有效性和高效性.

3.2 实时任务模型

目前,实时系统研究领域使用最广泛的时间模型是 Liu 和 Layland 等学者在 20 世纪 70 年代提出的 Liu & Layland Task Model[45].文献[45]中假设一个由若干个相互独立的周期性任务组成的多任务系统运行在一个单

处理器上,分析的目标是保证每个任务的每次执行都在其下一个周期开始之前结束.在这一模型中,每个实时任务的时间属性由最坏情况执行时间(WCET)、执行周期(period)以及截止期(deadline)这 3 个参数进行描述.Liu 与 Layland 为这样的系统设计了两种经典的调度算法 RM 和 EDF,并可以通过简单数学公式对上述问题进行分析.Liu 与 Layland 所假设的独立周期性任务模型捕捉了周期性控制系统的最基本特性,并因其理论简单、优美、易于应用而受到广泛关注.目前大多数的实时调度及可调度性分析方面的研究工作都是基于上述模型开展的.这一任务模型由于比较简单,相应的时间分析通常具有较高的效率.但其问题是难以对复杂的系统行为进行建模,因此只能描述非常有限的一类实时系统.

在另外一个极端上,研究者采用时间自动机(timed automata)对实时系统的时间行为进行建模[53].采用时间自动机的好处就是能够对一个复杂系统进行非常精确的描述,例如可以建模复杂的任务间依赖关系、约束更小的任务释放时间、任务间的同步行为等等.而精确建模所付出的代价就是导致可调度性分析的复杂度非常高,在某些实际分析中是不可行的.为了保证时间分析具有合理的效率,并同时尽可能地提高分析的精确性,研究者们将 Liu 与 Layland 独立周期性任务模型逐渐扩展到间歇性任务模型(sporadic task model)[54]、多帧任务模型(multiframe task model)[55]、有向图(digraph)任务模型[56]、fork-join 实时任务(FJRT)模型[57]、无环再生实时任务模型(non-cyclic recurring real-time task model)[58]、并行结构任务模型[59]等等.其中,无环再生实时任务模型[58]采用一个有向无环图 DAG(directed acyclic graph)来描述每个实时任务.尽管基于这一模型的实时调度分析是可行的,该模型的描述能力仍旧有限.例如,它不能够描述任务内部的无上界的局部循环,无法描述任务的模式切换(mode switch)等.而目前分析可行的时间模型中,描述能力比较强大的是最近提出的 Digraph 时间任务(digraph real-time task,简称 DRT)模型[56].在这一模型中,每一个实时任务可以通过一个有向带环图进行描述.基于该模型和 EDF 调度策略的可调度性分析具有伪多项式复杂度,基于固定优先级调度策略的可调度性分析是一个 Strongly Co-NP-Hard 问题.

另一个与实时系统可调度性分析相关的重要理论是实时演算(real-time calculus)[60].实时演算可以看作是经典的网络演算[61]针对嵌入式实时系统性能分析的扩展.实时演算通过使用时间区间域(time interval domain)来表示系统的负载以及可用的运算和通信资源.时间区间域表示在某个特定长度的时间区间内所到达的负载或所提供的资源的最大(最小)值.例如,图 5 所示为一系列需要处理的事件所到达的时间及其时间区间域的表示:$\alpha^u(t)$和$\alpha^l(t)$分别为长度为 t 的时间区间内所包含事件个数的最大值和最小值.实时演算在表达系统负载方面具有一定的能力,经典实时调度中研究过的大部分任务模型都可以看作实时演算的特例.

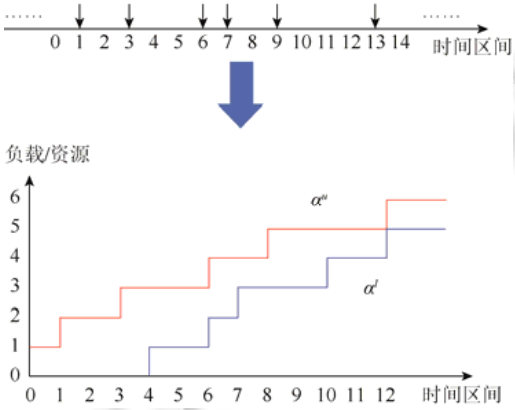

Fig.5 Timing abstraction

图 5 时间区间域抽象示意图

实时演算可以对具有网络结构的系统的实时性能进行建模和分析.近年来,基于实时演算的实时性能分析方法得到了迅速的发展.其先后被扩展到处理相关性负载的调度[62]、基于接口的系统设计方法[63]、能耗敏感调度的分析与设计[64]、温度敏感调度的分析与设计[65]、多媒体嵌入式系统的分析与设计[66]、具有循环依赖系

统的性能分析[67]等等.该理论直接从基于上述时间模型抽取系统的负载信息,并分别采用到达曲线和服务曲线来描述任务对处理时间的需求和系统在给定单位时间区间内所能提供的服务.这一大类模型的主要优点是对系统的时间语义具有清晰、简洁的描述,因此,非常适用于系统时间行为的分析(主要是可调度性分析).但是它们的突出问题是抽象程度过高,仅抽取了任务的时间特性,而无法表达复杂的程序语义与控制逻辑.因此,该类模型难以服务于复杂系统的具体设计.

3.3 多核实时调度技术

多处理机调度分为划分调度(partitioned scheduling)和全局调度(global scheduling)[68].全局调度允许任务或作业在所有处理核心之间自由迁移;划分调度的基本思想是把任务集进行划分,每个任务绑定到固定的处理核心上执行.全局调度能够提高系统的总体处理机利用率,但却可能带来较大的系统开销;划分调度则恰恰相反,可以充当从单处理机调度向多处理机调度的桥梁.划分调度的主要缺陷是处理机使用率低.造成这一问题的主要原因是任务被绑定在固定的处理机上,因此难以有效利用空闲处理器.为了解决这一问题,学术界开始对划分调度进行改进,提出一种称为半划分调度的算法,该算法能够很好地综合全局调度和划分调度的行为特性.在任务调度过程中,大部分任务采用划分调度的思想,被预先划分到一个固定的核心上运行,而另外一部分任务则采用全局调度的思想可以在多个核心上运行.这种策略可以增强系统的平衡负载,缓解空闲处理器难以利用的问题.

对多核调度领域产生重要影响的论文[69]是 Dhall 和 Liu 在 1978 年发表的.该论文指出了与单核调度相比,多核实时调度本质上更难解决.另一方面,该论文提到多核全局调度会遭遇 Dhall 效应问题,使得系统的调度性能反而不如划分调度算法.受到该结论的影响,学术界在此后的 20 年间基本停止了对全局调度算法的相关研究,将主要的研究精力放在划分调度的相关研究上.划分调度本质上可以看作一个装箱问题,该问题具有 NP-Hard 复杂度.因此,许多与装箱问题解法类似的近似算法被提出并加以应用,相关的研究工作可以划分为两类.其中一类的目标是得到与最优算法近似度最高的算法,该类方法通常通过证明一种算法的近似率(approximation ratio)来表明该算法的优劣[70,71].另外一类工作是针对单核的基于利用率的可调度性,将利用率的计算转化为装箱问题.由于装箱问题的限制,其利用率界限[72]不会超过 50%.在针对半划分调度问题的研究中,Andersson 在文献[73]中提出半划分调度算法 EKG,该文指出,EKG 的性能与任务的强占次数有很大关系,比如,当每个任务实例平均被抢占 4 次时,其资源利用率界限为 40%.针对基于固定优先级调度策略,Kato 等人在文献[74,75]中提出了半划分调度算法 RMDP 和 DMPM,并证明这两种算法的资源利用率界限都为 50%.针对 DMS 调度策略,Lakshmanan 等人在文献[76]中提出了一种半划分调度算法 PDMS_HPTS_DS,该算法的资源利用率界限为 65%.针对 RMS 调度策略,Guan 等人在文献[77]中提出了半划分调度算法 SPA1 和 SPA2,并首次将单个处理机上资源利用率界限提高到 Liu & Layland 使用率上限.

由于文献[69]关于全局调度算法的悲观结论,20 世纪 80 年代到 90 年代这 20 年间学术界关于实时调度的研究全部关注在划分调度方面.直到 Phillips 等人在文献[78]中指出,在全局调度算法中,高负载的任务被错误地赋予了低优先权,从而造成 Dhall 效应.该论著[78]研究结果的意义在于,首次正确阐述了产生 Dhall 效应的根源,指出了任务负载的重要性,通过赋予高负载任务正确的优先级,可以提升全局调度的性能.此后,学术界开始注重全局调度的研究工作.然而,对于全局调度,一些根本理论问题仍然还未得到很好的解决.其中一个主要问题是无法对算法的可调度性进行精确的分析,即无法找到系统的关键时刻(critical instant).关键时刻是指导致该任务响应时间最大的系统状态.具体来说,关键时刻是指调度中一个特定的任务释放模式,系统如果在这个释放模式下可被调度,则在任何任务释放模式下都保证可以被调度.从关键时刻出发,就可以计算任务的最坏情况响应时间.因此,对于系统的分析只需要考虑这样一个具体情况来代表系统的最坏行为.鉴于关键时刻的意义,一些研究者针对该问题展开了研究,其主要的研究思路是通过分析某个时间区域的任务负载总和上限并根据该上限进行可调度性分析.大部分的研究工作主要考虑如何构造出近似的系统任务负载状况,并通过排除不可能的系统行为来使上述计算变得更加精确[79−85].虽然这些工作对提升全局调度性能有一定的帮助,但是总的来说,全局调度的分析技术还不是很精确.

此外,基于时间片划分,Baruah 等人在文献[86]中提出了按比例公平使用资源的全局调度策略 Pfair.其核心思想是,每个周期任务可以划分成多个小的子任务,每个子任务执行一个小的时间片.在调度时,所有等待调度的子任务按照优先级次序组成全局等待队列,新的子任务到来时,按优先级插入到等待队列中.处理器在每个时间片检查等待队列,选择最高优先级任务调度.根据确定子任务优先级方式的不同,随后学术界又提出了 PD[87]和 PD2[88]调度算法.通过划分子任务,Pfair 全局调度策略可以达到满负载系统利用率,是一种理论上最优的全局调度.然而,由于子任务的划分,在调度过程中子任务频繁切换会带来巨大的系统开销,导致 Pfair 调度算法实际应用时比较困难.

上面大部分研究工作仅仅关注了周期任务模型和间歇性任务模型这两类简单任务模型.除了基于简单的任务模型,一些研究者也针对复杂任务模型多核可调度问题展开了深入的研究.在应对系统的并行结构复杂性方面,Leontyev 等人[89]考虑了在实时演算中对全局多处理器 EDF 调度策略进行分析.首先通过传统的多处理器 EDF 调度分析方法对任务的响应时间进行分析,再将分析结果应用到实时演算的框架中去,间接获得负载的输出到达曲线.此工作虽然朝着应对系统并行结构复杂性迈出了重要的一步,但其依然存在一定的局限性.首先,其只考虑了系统并行资源,而依然假设负载为一组相互独立的串行任务.其次,其仅仅局限于 EDF 调度这一特定的调度算法.再次,其分析精确度较差,会引起比较严重的资源浪费.总体来讲,其依然是在传统的实时演算框架之下,通过使用现有的近似响应时间分析方法针对全局 EDF 调度作了特定的改进.Schranzhofer 等人[90]研究了使用实时演算分析多核处理器上具有共享总线资源的任务系统.该工作虽然考虑了并行负载间由于共享总线资源而产生的相互作用关系,但对负载的结构及其在处理器上的调度行为作了简化的假设,从而使分析的焦点只集中在共享总线资源这一个维度.

3.4 多核混合关键性调度

传统实时系统的设计与分析只有一个目标,即满足系统的实时性需求.为此,从程序级别的分析到系统级别的调度都不得不通过牺牲资源利用率来换取最坏情况下的实时性.为了减少这种传统系统模型的保守近似(over-approximation)的影响,Vestal[91]最先提出了混合关键性系统中的实时任务调度问题,假设每个任务具有多个最坏情况执行时间并引入了混合关键性模型(mixed-criticality model).在这个系统模型中,系统具有多个关键性级别,系统的参数和实时性能需求在不同的关键性级别之下是不同的.通过这种方式,可以在一定程度上解决实时系统设计中各个分析环境的悲观性问题,提高系统的资源利用率.

随着应用需求的提升和硬件技术的发展,在多核平台上部署混合关键性应用将成为嵌入式系统发展的必然趋势.如何在满足不同关键性功能认证需求的同时提高多核系统资源利用率,对多核混合关键性系统的设计与应用至关重要,也为多核实时系统理论研究提出了新的挑战.为了解决这一问题,大量的研究工作投入到混合关键性系统实时调度问题中,文献[92]对混合关键性系统的相关研究进行了详细的、全面的综述.鉴于此,本文这里不再进行详细论述,只对多核混合关键任务实时调度的研究工作中存在的一些关键问题进行总结和归纳.

(1) 在关键性模式切换(mode-switch)时,为保证高关键性任务的可靠运行,所有低关键性(low-criticality)任务都被抛弃,系统只保证高关键性任务的执行.然而,在实际系统中,往往需要给低关键性任务保留一部分的服务,突然停止运行所有的低关键性任务往往是不可取的.

(2) 任何一个高关键性任务的运行时间超过当前关键级别的最差执行时间,系统会强制所有的高关键性任务都进入高关键级别,并分配更加悲观的系统设置,造成不必要的资源浪费.

(3) 在现有模型中,系统的模式切换往往只考虑系统进入高关键级别的过程.系统切换回正常模式的过程在多数的研究工作中还没有考虑.

对于上述问题,Ren 和 Pan 在文献[93]提出了一种基于 Pfair 调度策略的任务组多核调度算法.在该算法中,任务被划分成时间片长度(quantum-length)的子任务.每一个高关键性任务和几个低关键性任务捆绑在一起,形成一个任务组.任务组在调度时按照 EPDF 调度策略运行.然而,文献[93]给出的是一种基于 Pfair 的多核调度策略,大量子任务频繁切换会带来巨大的系统开销,导致算法实际应用时比较困难.此外,任务组之间的同步也是一个需要考虑的问题.最近,文献[94]提出了一种基于 EDF-VD 的灵活调度策略 FMC,在高关键性任务发生过载

时,系统只会触发过载任务本身进入高关键级别,同时低关键性任务会根据过载情况最大限度地保留其服务.但是,目前 FMC 还只支持单核调度,如何将其推广到多核处理器还需要进一步加以研究和探讨.

4 实时多核虚拟化技术

在实时多核系统设计中,为了避免系统资源的干扰,往往使用物理隔离技术对系统资源进行隔离.虚拟化技术可以将不同功能的子系统部署在运行于同一硬件平台的虚拟机上,为系统隔离和资源管理提供了一种更加灵活的方法.对于虚拟机而言,虚拟机监控器(hypervisor)负责管理虚拟机并屏蔽了底层硬件的实现细节,采用灵活的策略将底层硬件资源有效地分配给虚拟机,从而满足各个虚拟机对资源的需求.对于实时多核系统而言,虚拟化技术能够很好地满足实时多核系统设计中的一些迫切需求.

(a) 可移植性(potability):能够很容易地将软件部署在硬件上;

(b) 可隔离性(isolation):可以实现不同操作系统上的安全隔离;

(c) 可集成性(composition):能够很容易地将新的应用集成到目前的平台上,易于实现增量化设计(compositional design);

(d) 遗留软件的兼容性(legacy code):芯片厂商在推出一个新的多核芯片时,需要考虑遗留软件的兼容性问题.虚拟化技术可以很好地解决这一问题,在新的硬件平台上隔离出一个虚拟环境,用于运行遗留软件.

鉴于虚拟化技术的这些优势,在实时多核上应用和部署虚拟化技术是一个非常有前景的技术方案,能够解决实时多核系统设计中的一些关键问题.

近年来,虚拟化技术在云计算和服务器等通用计算领域已得到了广泛的应用,这些领域主要关注于虚拟机的公平性与系统的吞吐率.然而,虚拟化技术在实时嵌入式领域中仍然属于新的应用方向,相关研究工作还十分有限.其主要问题在于,当虚拟化技术被引入到实时嵌入式系统后,运行在虚拟机上的实时操作系统的响应性能易受到虚拟化软件层的负面影响.虚拟化层的引入所带来的语义缝隙问题使得虚拟机监视器难以感知上层虚拟机应用类型,妨碍了虚拟机监视器根据上层应用的需求进行有效的硬件资源分配,从而无法为对实时性要求较高的应用提供良好保障.在虚拟化环境下,如何动态地给实时虚拟机分配足够的资源并使实时虚拟机的实时性能得到保障是一个重要的课题.本节对实时多核系统虚拟化的研究工作进行综述,主要探寻虚拟监控器中虚拟机层次化实时调度机制以及实时性改造工作.本节首先对虚拟化技术进行一个总体的概述.然后,在此基础上对层次化调度算法和实时虚拟机监控器的设计相关研究工作进行综述.

4.1 虚拟化系统概述

本文主要讨论基于虚拟机监控器(hypervisor-based)的虚拟化系统.图 6 阐述了多核虚拟化系统的基本调度架构.如图6所示,虚拟化系统采用两层调度框架,虚拟化系统能在单个硬件平台上运行多个客户操作系统(guest OS).每个客户操作系统可以管理不同的应用程序,并将应用程序调度到不同的虚拟机(VCPUs)上运行;虚拟机监控器则负责调度VCPU到物理核(PCPUs)上运行.在多核虚拟化系统中,虚拟机监控器位于操作系统软件层与硬件层之间.作为虚拟化技术的核心,虚拟机监控器负责管理 VCPUs 并屏蔽了底层硬件的实现细节,采用灵活的策略将底层硬件资源有效地分配给 VCPUs,从而满足各个 VCPUs 对资源的需求.因此,虚拟机监控器直接决定了整个虚拟化平台的性能.

虚拟化技术可以分为两类:全虚拟化(full virtualization)和半虚拟化(para virtualization).全虚拟化技术能够完全抽象系统物理硬件平台,并且能够为客户操作系统提供一个完整的虚拟环境.在运行时,操作系统不会感知到自己运行在虚拟化环境中.因此,全虚拟化技术可以为虚拟机提供完整的隔离,保证系统的安全性.全虚拟化技术需要使用虚拟机协调客户操作系统和原始硬件,虚拟机监控器用于在客户操作系统和裸硬件之间进行工作协调,一些受保护指令必须由虚拟机监控器来捕获处理.在全虚拟化系统中,操作系统没有经过任何修改,唯一的限制是操作系统必须能够支持底层硬件.半虚拟化使用虚拟机监控器分享存取底层的硬件,但是它的客户操作系统集成了虚拟化方面的代码.该方法无需重新编译,因为操作系统自身能够与虚拟进程进行很好的协作.

半虚拟化需要客户操作系统做一些修改来配合虚拟机监控器.但是,半虚拟化提供了与原始系统相近的性能和运行时系统的灵活性.

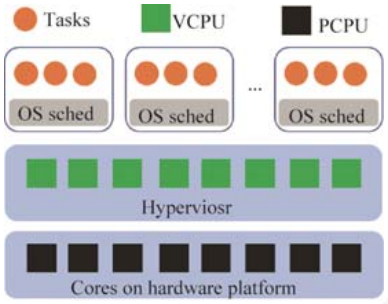

Fig.6 Virtualization hierarchical scheduling framework

图 6 虚拟化系统层次化调度架构

4.2 层次化实时调度

如图 6 所示,当在实时嵌入式系统中部署虚拟化环境时,虚拟化系统需要采用层次化调度框架.(1) VCPUs 在 PCPUs 上的调度;(2) 实时任务在 VCPUs 的调度.这两个层次上的调度级别必须满足实时性保障,从而实现整个系统的实时性保障.然而,经典的实时调度对于这种层次化的调度结构不再适用.层次化调度框架需要在不同的调度算法下为不同的调度服务提供层次化的资源共享和分配策略.层次化调度框架可以表述为一种带节点的树状(层次化)结构.每一个节点可以表述为调度模型以及从父母节点到子节点的待分配资源.从父母节点到子节点的资源分配可以看成是从父母节点到子节点的一个调度接口.因此,层次化实时调度问题可以转化为父母节点与子节点的调度接口的可调度性分析问题.

针对实时多核系统层次化可调度分析问题,文献[95]提出了多处理器周期资源模型(multiprocessor periodic resource model,简称 MPR).该模型基于单核层次化调度的周期资源模型(periodic resource model,简称 PRM)[96].PRM 模型严格限制了节点模块在一个周期内获取计算单元的个数.与单核 PRM 模型不同的是,多核 MPR 模型允许所申请的计算单元能够并行消耗,并假设任务集中的实时任务是独立的,单个任务不能被并行化.任务层级的调度采用全局 EDF 调度算法.调度接口可调度性问题可以看成是在调度接口周期内的多个周期性任务的服务问题.由于模型中整数型预算限制使得接口计算变得非常复杂.为了解决这些问题,文献[97]提出了通用多处理器周期资源模型,可以看作是 MPR 模型的扩充.另外,文献[98]提出了并行供给函数(parallel supply function,简称 PSF)这一概念,提出了一种通用的资源供给机制,允许节点模块所获取的计算单元可以有不同的周期.文献[99]提出了边界延迟多分区模型(bounded-delay multipartition,简称 BDM).其调度接口采用边界延迟函数(bounded-delay function)根据实时性保障需求计算节点模块所允许的最大空闲资源时间.在给定资源分数的情况下,通过计算节点模块的相关参数来达到资源使用和灵活性之间的平衡.

4.3 实时虚拟机监控器

将虚拟技术应用于实时嵌入式领域,除了需要对层次化实时调度理论进行研究外,还需要对目前存在的虚拟机监控器进行改造或者重新研发,使其能满足实时性保障.目前,针对实时虚拟机监控器设计实现的研究工作还处于起步阶段,在多核环境下,相关研究工作还十分有限.本节将对多核实时虚拟机监控器的实现与应用的相关研究进行梳理和介绍.

Xi 等人在文献[100]中首先提出了一种基于 Xen 的分层的单核实时调度框架 RT-Xen,在 Xen 上提出并实现了相关的分层调度理论.该框架对多种不同固定优先级的调度算法进行了实现,并对其实时性能进行了测试.实现的调度算法主要包括:Deferrable Server、Periodic Server、Polling Server、Sporadic Server.实验结果表明,Deferrable Server 在实时性能上要优于其他调度算法.随后,在文献[101]中,RT-Xen 被加以扩展,能够支持多核计算平台,并且支持 MPR、DMPR、MSF、GMPR 等多种模型.为了尽可能地降低 deadline miss,虚拟机监控器调

度采用的是全局 EDF 调度算法,任务层次的调度算法采用的是划分的 EDF 调度算法.

此外,还有一些研究者利用虚拟机监控器来实现多核资源的隔离,从而提升多核系统的时间可预测性.Jing 等人在文献[102]中提出了基于 Xen 虚拟机监控器的多核存储器管理技术.该技术主要采用内存访问截停机制,在运行之前给每一个虚拟机分配一定的内存访问份额,在运行时通过处理器内嵌的性能监视单元(performance monitoring unit,简称 PMU)来监控各个虚拟机对内存的访问情况,如果发现虚拟机达到访问上限,则截停当前访问,需要等到下一个周期分配新的访问份额.最近,Kim 等人在文献[103]中提出基于虚拟机监控器的实时多核缓存资源管理技术.缓存资源的隔离是通过 page coloring 来实现的,根据 guest OS 是否支持 page coloring 提出了 vLLC 和 vClolloring 缓存管理机制.同时,Xu 等人在文献[104]中提出了基于 Intel 缓存分配技术 CAT 动态缓存虚拟化管理技术,实验数据表明,CAT 虚拟缓存管理方法能够为并行实时任务提供很好的实时性能隔离.

5 多核实时程序并行化技术

为了充分发挥多核处理器的运算能力,需要将计算机程序并行化.并行程序设计虽然在通用和高性能计算机系统领域中已经成为主流,但在实时系统领域还未得到足够的重视.在过去的 10 年中,全局调度和划分调度这两种类型的调度与分析方法都得到了长足的发展.但是这些工作所采用的任务模型是从经典单核处理调度理论继承而来的,即假设每个任务的负载是一个必须串行执行的实体.因此,这些工作的结果不适用于面向并行实时软件的调度分析问题.目前,面向并行程序的多核实时系统方面的研究刚刚起步,还未有成型的理论体系与技术框架.

针对这一问题,近年来一些研究者开始关注并行任务的实时调度与分析问题.目前,多核实时程序并行化的研究大多围绕 DAG 任务模型展开.DAG 任务模型能够描述任务线程执行依赖关系.例如并行执行和串行执行.在并行 DAG 模型中,串行同步并行(sequential synchronous parallel,简称 SSP)任务模型是一种对任务行为更加严格的模型.在该模型中,任务可以划分为串行执行的 segment,每一个 segment 可以包含任意多个并行的 threads,所有的 threads 必须都在 segment 末尾同步.图 7 描述了一个串行同步并行任务模型,包含了 5 个串行执行的 segment,后面的 segment 只有等前面的 segment 执行完毕之后才能开始执行.从模型的定义来看,并行 DAG 模型可以很好地兼容 fork-join 语义,能够与一些成熟的并行编程语言兼容.最近一些研究者考虑了 OpenMP 编程模型在实时系统中的适用性问题,通过对 OpenMP 中的任务语义进行分析,试图将 OpenMP 与现有的并行实时任务调度研究工作建立起联系[105].然而这些工作只考虑了 OpenMP 中程序结构的一个子集,而忽略了循环结构、隐式并行结构等许多重要程序结构.此外,这些工作没有提出解决分析相关调度问题的新方法,所以其分析结果与上述研究工作仍然非常悲观.

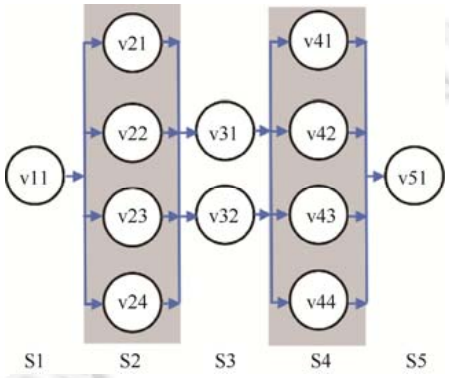

Fig.7 Sequential synchronous parallel task model

图 7 串行同步并行任务模型

针对这些问题,学术界开展了模型实时调度的相关研究,着重针对多核系统对并行任务调度问题提出的新需求开发高效算法,并引入新的计算方法和工具解决传统调度问题.总的来说,目前的相关研究可以分为基于变

换的调度技术和直接调度技术两大类.基于变换的调度技术的主要思想是通过某种转化手段,例如分配线程的截止期,将执行依赖的并行任务模型转化为独立非依赖的任务集,从而可以借助传统调度算法来优化并行任务.由于依赖于转化技术,基于变换的调度技术因此往往需要提前知道并行任务模型的内部结构细节信息.相反,直接调度技术不需要知道并行任务模型的内部结构细节信息,任务只有在所有执行约束条件满足之后才开始执行.

Saifullah 等人[59]考虑了并行任务模型在多核上的实时调度问题,并提出了任务分解的变换策略.针对该 DAG 并行任务模型,作者通过给每个子任务分配释放偏移和截止期,从而消除任务之间的执行依赖关系.他们所提出的任务分解方法将每个并行 DAG 任务分解到一组顺序任务中,并证明了该方法的资源增强的上界在全局 EDF 和 DMS 下可以分别达到 4 和 5.Li 等人[106]考虑同一个任务里各个执行单元间的依赖关系[107−109],并通过结合全局调度和划分调度两类算法的优点,提出了一种新型的调度算法,即联合调度(federated scheduling),以进一步提高系统的实时性能.在该方法中,根据任务利用率高低,任务被划分成不同任务组,每个任务组被调度在不同的核心上运行.此外,针对 DAG 并行任务模型,Li 等人[106]还证明了 GEDF 和 GRM 算法的容量增值界分别为 $\frac{3+\sqrt{5}}{2}$ 和 $2+\sqrt{3}$.在上述工作的基础上,有些工作考虑对并行任务模型进行扩展,包括在并行任务模型中加入条件分支语义[110−112],并给出了高效的任务响应时间分析方法.对于直接调度方法,Sliva 等人在文献[113]中为伙伴任务静态优先级调度提供了一种可调度性分析方法.在每一个调度点上,任务根据其优先级和空闲核心数量是否满足需求来进行调度.Goossens 等人在文献[114]中讨论了动态伙伴调度策略的可预测性问题,提供了一些方法来保证动态伙伴调度策略的可预测性,并提供可调度性分析方法.

6 多核能耗优化技术

目前,为了进一步提升计算系统的性能,芯片制造工艺已经从 20 世纪的亚微米水平向深亚微米方向发展,晶体管尺寸的不断缩减趋势使得 Dennard Scaling 不再有效.Dennard Scaling 是早期半导体工艺变化的规律,即单位面积晶体管数不断增加而功耗保持不变.Dennard Scaling 规律的结束意味着在能效比的约束下,增加核心数目很难再增加性能.晶体管尺寸的不断缩减以及芯片内计算核心的不断增加导致芯片的功耗密度显著增加.根据 Intel 公司的预测,到 2020 年,多核系统芯片的峰值功耗将会升至目前水平的 10 倍以上[115].急剧上升的功耗密度会导致芯片温度过热及系统可靠性下降.正是由于系统功耗的限制,无法支撑芯片所有晶体管同时正常工作,从而导致一部分核心必须关闭,即所谓的暗硅(dark silicon)[116]现象.这种现象将直接导致多核系统芯片实际所能达到的性能比理想性能差很多.目前,这种性能上的差距还在不断扩大[116].因此,为了进一步提升多核系统的性能,首先要解决系统的功耗问题,这就需要研究一些技术手段根据系统在线负载情况合理调度任务并调节系统状态,在满足实时动态任务截止期约束的前提下尽可能地降低多核系统能耗.

本节主要对实时多核系统的能耗管理的研究工作进行综述,对实时约束下的系统功耗管理方法和技术进行深入总结和探究.功耗管理的优化研究工作首先需要有功耗模型,用于描述系统的功耗特性.因此,本文首先阐述现有研究工作中比较常用的功耗模型.然后,在此基础上对不同的能耗管理方法进行综述.

6.1 功耗模型

通过对单个门电路的功耗建模,Martin 等人在文献[117]中对系统的功耗进行了建模,并通过 SPCIE 仿真验证了所提出的功耗模型的精确性.在该功耗模型中,系统的功耗来源于动态功耗 P_{dyn}、漏电功耗 P_{lkg} 和固有功耗 P_{on} 这 3 个方面.其中,动态功耗主要是由时钟网络产生;漏电功耗是由晶体管的栅漏电流产生,是静态功耗中的主导部分,有研究者直接将其称作静态功耗;固有功耗是处理器处于开启状态的固有代价.

动态功耗源于器件上电情况下的电路反转,受供电电压和时钟频率影响.因此,动态功耗 P_{dyn} 在给定电压/时钟(V_{dd},f)前提下可表示为

$$P_{dyn}=C_{eff}\cdot V_{dd}^{\ 2}\cdot f \tag{1}$$

其中,V_{dd}为供电电压,f是时钟频率,C_{eff}为有效切换电容.时钟长度t_{cycle}在α功耗模型中由下面公式可以计算得到:

$$t_{cycle}=\frac{L_d\cdot K_6}{(V_{dd}-V_{th})^{\alpha}} \tag{2}$$

其中,L_d是由处理器的最长逻辑深度决定,并与处理器的电路设计以及温度参数相关,K_6是技术参数常量,门限电压V_{th}由公式(3)计算得到:

$$V_{th}=V_{th1}-K_1\cdot V_{dd}-K_2\cdot V_{bs} \tag{3}$$

其中,V_{th1}、K_1、K_2为技术参数常量,V_{bs}为体偏压,与芯片工艺有关.

漏电功耗P_{lkg}是由漏电流产生的,只要能量流过电路就会产生漏电流,因此,漏电功耗是必然存在的.根据Martin等人[117]提出的功耗模型,漏电功耗P_{lkg}主要由漏电流I_{sub}、反偏节电流I_j以及芯片中存在的晶体管数量L_g来决定:

$$P_{lkg}=L_g\cdot(V_{dd}\cdot I_{sub}+|V_{bs}|\cdot I_j) \tag{4}$$

固有功耗P_{on}是处理器处于开启状态的固有代价,与漏电功耗一样,即使处理器处于空闲状态也会产生这部分功耗.与动态功耗和漏电功耗相比,固有功耗的值通常较低.固有功耗主要是由芯片的外设产生的,主要来源是芯片锁相环PLL和I/O系统.不同技术以及体系结构下的固有能耗值是不同的.在一些功耗管理的研究中,常常将固有功耗设置为常量[118].

此外,一些研究工作也在上述模型的基础上对系统功耗模型进行了简化,并假设处理器频率f可以在[$f_{\min}$,$f_{\max}$]范围内发生变化.定义系统速度变量$s=\frac{f}{f_{\max}}$,系统功耗可以通过系统速度s来表征,这些研究工作认为系统的功耗可以直接分为静态功耗P_{ind}和动态功耗P_{dyn}:

$$P(s)=P_{ind}+P_{dyn}(s) \tag{5}$$

其中,静态功耗P_{ind}不受系统速度s的影响,而动态功耗P_{dyn}是系统速度s的多项式函数.在一些文献[119]中,多项式函数被表述为$P_{dyn}(s)=\beta\cdot s^{\alpha}$,其中,$2\leqslant\alpha\leqslant3$.

6.2 能耗管理以及优化

多核系统的工作负载在通常情况下会随着时间发生变化,可以通过将核心切换到功耗更低的睡眠状态或者调节核心的工作频率来平衡系统负载和功耗之间的关系.目前,多核功耗管理技术主要分为两类:动态功耗管理(dynamic power management,简称DPM)和动态电压频率调节技术(dynamic voltage and frequency scaling,简称DVFS).实时多核功耗优化研究的目的在于,运用这些功耗管理技术并结合实时调度的相关理论,在实时约束的条件下优化系统功耗.

DPM技术的出发点是在满足系统实时性要求的前提下,根据系统的动态负载,尽可能地让系统进入睡眠模式,以减小系统静态功耗.一般而言,处理器可以有normal、idle、sleep这3个工作模式,3个工作模式对应的功耗分别为P_n、P_i、P_s,并且有$P_n>P_i>P_s$.当系统有任务需要处理时,系统处于normal状态并全速处理任务,否则,处于idle或者sleep状态.Idle状态和sleep状态的切换有一定的时间开销和能量消耗,因此,使用DPM策略将系统切换到sleep状态以达到节能目的,需要满足sleep状态下节约的能耗大于关闭延迟和唤醒延迟的额外能耗,否则,DPM的这种状态切换是失效的.状态之间的切换行为是通过对工作负载的观察来决定何时以及如何进行工作模式的转移的,并同时满足系统实时性约束和模式切换的时序约束.

DVFS技术是降低嵌入式系统功耗的另一种重要手段[120].其主要是基于一个这样的事实:处理器的能量消耗正比于电压的平方和时钟频率,通过动态监控系统负载,在满足实时约束的条件下,可以适当降低时钟频率并延长任务执行时间,从而达到降低系统功耗的目的.在实时系统中,DVFS技术是根据任务的紧迫程度来动态调节处理器运行电压和频率,在保证任务的完成时间不迟于其截止时间的前提下,使得处理器不总是以最高频率运行,从而有效地减少CPU的能量消耗.

通过对DVFS和DPM的基本原理的分析可以看出,DVFS与DPM在应用上有着明显的区别.DVFS技术主要是通过调节芯片的电压频率来降低系统的动态功耗,DPM技术是通过系统切换到休眠状态来降低系统的静

态功耗.从芯片制造工艺发展水平上来看,现有相关研究分为两个明显的阶段:① 在早期,工艺水平低时,处理器的动态功耗是系统能耗的主要贡献者.在这一阶段,人们主要研究降低系统动态功耗的方法.因此,这一时期的大部分功耗优化工作主要通过 DVFS 技术来实现能耗优化.② 随着近来芯片制造工艺水平的不断发展(例如,目前 Intel 公司已将工艺水平发展到 7nm[16]),功耗密度不断上升,导致漏电功耗的比重不断增加,最终超过动态功耗成为主导功耗,如何优化系统静态功耗的研究工作也越来越受到重视.在这一阶段,研究者们开始将重点转向静态功耗的优化,开始采用 DPM 技术(或者 DVFS-DPM 联合技术)来进行系统能耗优化.因此,本文根据能耗管理技术的不同,将实时多核系统能耗优化问题的研究分为两个类别:动态功耗优化算法和静态功耗优化算法.

(1) 动态功耗优化算法

在多核系统中,根据 DVFS 调节方式以及调节粒度的不同,已有的动态功耗管理研究工作可以分为基于核心(per-core)、基于任务(per-task)、基于电压频率岛(voltage frequency island,简称 VFI)的动态功耗优化方式.Per-Core 方式主要是在运行过程中给每一个核心分配一个固定的频率运行,而 per-task 方式是给每一个任务分配一个固定的运行频率.为了在多核系统中更加灵活地进行功耗管理,近年来,基于电压频率岛的功耗管理技术受到了广泛的关注.在 VFI 电压调节技术中,计算核心被划分为多个组,每个组内的核心共享相同的内核电压和频率,因此组内的核心构成了一个电压岛.通常将这种局部统一控制电源频率的方式称为 VFI 方式.表 1 对基于 DVFS 技术的动态能耗优化方法进行了总结和对比.

在 per-core DVFS 方式下,Aydin 和 Yang 研究了在多处理器系统中将实时任务集划分到各个核心上以最小化能耗为目标的问题[121].该工作发现,对于最早截止期任务优先(earliest deadline first,简称 EDF)的调度而言,平衡各个处理器的负载可以有效地优化系统的能耗.同时,该工作还指出,在多核环境下计算能耗最优的任务映射是一个 NP-hard 问题.通过比较 BFD(best-fit decreasing)、WFD(worst-fit decreasing)、FFD(best-fit decreasing)以及 NFD(next-fit decreasing)等一些常用的启发映射算法,发现 WFD 策略可以得到较均衡的映射,并且在能耗上要优于其他算法.Moreno 和 De Niz 在文献[122]中提出了增长最小频率(growing minimum frequency,简称 GMF)算法,该算法可以给每个核心分配最小的频率.在该方法中,考虑了周期性的实时任务,并采用 U-LLREF 算法进行调度.U-LLREF 算法是 LLREF 调度算法在同构多核平台上的扩充.GMF 算法在初始状态时,将实时任务根据利用率进行排序,并将所有核心的频率设置为最小.在迭代过程中,算法将 i 个任务分配在 i 个核心上,其中,i=1:m(m 为核心数目).在一次测试中,通过比较 i 个任务利用率与频率的大小决定是否增大最慢核心的频率.

Table 1 Approaches for dynamic power optimization

表 1 动态功耗优化方法对比

算法	DVFS 调节方式	调度策略	功耗模型	离散速度	复杂度
Reservation[121]	Per-Core	划分 EDF 调度	$\beta \cdot s^{\alpha}$	是	$O(nm)$
GMF[122]	Per-Core	全局 U-LLREF 调度	$\beta \cdot s^{\alpha}$	是	Polynomial
GSSR, FLSSR[123]	Per-Task	帧任务全局不可强占调度	$\beta \cdot s^{\alpha}$	是	Polynomial
FFDH[124]	Per-Task	帧任务划分调度	(s_i,P_i)	是	$O(kmn^2)$
LEET[125]	Per-Task	帧任务划分调度	$\beta \cdot s^{\alpha}$	否	$O(n\log(n))$
SPA2, PHD[126]	Per-Task	划分调度	$\beta \cdot s^{\alpha}$	否	Polynomial
VILCF[127]	VFI	划分 EDF 调度	$s^{\alpha}+\delta$	否	$O(n\log(n))$
BS+LTF[128]	VFI	划分调度	$\beta \cdot s^{\alpha}$	否	$O(n\log(n))$

在 per-task DVFS 方式下,Zhu 等人在文献[123]中,对于实时多处理器系统,充分考虑了系统在动态运行过程中的负载变化以及运行中动态产生的空闲时间,研究了基于全局不可抢占调度的能耗管理模式,并利用空闲共享技术来平衡负载,以达到更好的节能效果.Xu 等人在文献[124]中研究了多核计算平台上并行任务的能耗优化问题.在该方法中,主要考虑了帧任务模型,并将能耗优化问题转化为 ILP 问题来求解,为每一个并行任务分配频率使得能耗最优.此外,考虑到 ILP 问题的复杂性,作者还提出了一些二步式的启发式算法.首先利用层打包算法 FFDH 对任务进行映射,然后再反复迭代所有核心上允许的频率设置,直到找到最小能耗配置.Chen 等人在文献[125]中研究了多核平台中允许实时任务在核间迁移情况的能耗优化问题,并提出了频率分配的近视算法 LEET.文献[126]考虑了基于任务分割模型的多核能耗优化问题.在该问题中,任务可以被分割为子任务集,子任

务可以分配到各个核心上按照顺序执行.对于高利用率的实时任务,多核划分调度往往效率很低.利用任务分割模型对任务进行分割,很好地解决了这一问题.针对任务分割模型,作者提出了基于 SPA2[129]和 PHD[130]调度算法的相应的能耗调度策略.

此外,Liu 等人在文献[127]中考虑了周期性实时任务在电压岛调节方式多核计算平台的能效调度问题,推导出了电压岛能效调度问题最优解的下界,并发现,当负载在各个核心上达到均衡时,可以取得能耗调度问题的最优解.基于该发现,作者们提出了电压岛最大容量优先(voltage island largest capacity first,简称 VILCF)算法,并对该算法的 approximation ratio 进行了理论分析.分析结果表明,该算法的 approximation ratio 与多核的电压岛个数有关.文献[128]考虑实时任务到电压岛映射的能耗优化问题.该优化问题考虑了 3 个自由度:任务划分、频率分配、电压岛激活策略.作者发现能耗最优的频率分配方案不仅与所负载的平衡度有关,还与核心个数与电压岛激活个数有关.针对这一复杂问题,文献[128]首先给出多项式复杂度的 BS 算法,以确定在任务划分明确的情况下实时任务的频率分配.最后给出了一种综合算法来确定任务划分、频率分配、电压岛激活个数.

(2) 静态功耗优化算法

随着 VLSI 技术的迅猛发展,晶体管尺寸的不断缩减以及日益复杂的电路结构导致芯片的功耗密度在急剧上升.这一趋势导致漏电功耗的比重不断增加,最终超过动态功耗成为主导功耗.国际半导体科技蓝图(international technology roadmap for semiconductors,简称 ITRS)在其报告[131]中指出,当芯片制造工艺进展到 32 nm 时,芯片的静态功耗将成为芯片的主导功耗.因此,系统静态功耗优化的研究也越来越受到重视,开始采用 DPM 技术(或者 DVFS-DPM 联合技术)进行系统静态功耗优化.DPM 技术主要是在系统空闲的时候,将处理器切换到睡眠状态.如上所述,运用 DMP 技术所面临的一个主要问题是在做系统切换决策时,需要保证设备空闲时长满足一定的阈值(称为 break-even time),这样,使用 DPM 策略才能节约能量.为了计算允许的空闲时间,大部分的研究工作使用任务拖延技术,推迟处于 ready 状态的任务尽可能地创造空闲时间,使得系统能够满足空闲时间阈值进入休眠状态.同时,在实时系统中,任务拖延技术还需要保证系统的实时性约束得到满足.表 2 对基于 DPM 技术的静态功耗优化方法进行了总结和对比.

Table 2 Approaches for static power optimization

表 2 静态功耗优化方法

算法	负载模型	调度策略	分析方法	空闲	技术
PBOOA[132]	Arrival curve	先来先服务	RTC	静态 Slack	DPM
OPT[133]	Periodical DAG	Time-Triggered 调度	ILP	静态 Slack	DPM+DVFS
RDAG+ GeneS[134]	Periodical DAG	静态调度	Retiming	静态 Slack	DPM+DVFS
BWS[135]	Arrival curve	先来先服务	RTC	动态 Slack	DPM

文献[132]研究了流水线架构的多核系统的静态功耗优化问题.在该问题中,一个任务可以分解为顺序执行的子任务,分别映射到各个核心上执行.系统的任务负载被抽象成事件流,并采用 arrival curve 来描述.为了分析任务端到端的延迟,作者采用了 pay-burst-only-once 的原理对多核系统的 service curve 进行了计算和简化,并最终将问题转化为一个带有线性约束的二项式规划问题.通过对该问题的求解,可以得到核心开关的时间设置,达到优化系统静态功耗的目的.然而,该方法是一种离线的 DPM 技术,还不能完全挖掘系统在线运行时所产生的动态 slack.随后,文献[135]在此基础上对问题进行扩展,探究了运用 pay-burst-only-once 的原理在线计算多核系统的服务曲线的可能性.系统在运行时,考虑到核心之间的 FIFO 可能会存在还没有处理完毕的事件,作者将还没有完成的事件也抽象成事件流,并将 pay-burst-only-once 原理扩展到多事件流模型情况.除了使用纯粹的 DPM 技术外,还有一些研究者开展了基于 DPM+DVFS 联合技术的能耗优化的研究工作.Wang 等人在文献[134]中考虑了周期性的 DAG 任务模型的多核能耗优化问题.作者首先使用了 retiming 技术消除 DAG 任务模型中子任务之间的依赖性,并提出了 RDAG 算法将 DAG 任务转化为相互独立的任务.在此基础上,提出了 GeneS 调度算法优化系统能耗.所提出的 GeneS 方法采用了遗传算法来寻找能耗最优的调度算法.文献[133]主要考虑在任务映射给定的情况下如何以一种最优的方式结合 DVFS 和 DPM 技术来优化系统的全局能耗.在文献[133]中,每一个任务被抽象成一个 DAG,DAG 中子任务可以被映射到不同的核心上.基于 time-triggered 调度,所研究的

问题被转化为标准的混合整数线性规划(mixed integer linear programming,简称 MILP),并通过求解 MILP 问题可以找到能耗优化问题的最优解.

7 总结与展望

随着计算机系统与物理世界的结合越来越紧密,实时系统需要承担的运算任务越来越复杂.为了满足实时系统对高性能的需求,如何在多核硬件上高效集成和部署实时系统是目前学术界和工业界的一个热点问题.本文对现有的面向实时多核嵌入式系统的研究工作进行了综述,介绍了实时多核嵌入式系统的关键设计挑战,并从多核共享资源干扰及管理、多核实时调度、实时程序并行化、多核能耗管理等方面综述了近年来的研究进展,并展望了实时多核系统领域进一步的研究方向.期望本文的介绍能够为相关领域的同行学者提供一定参考.近 10 年来,面向实时多核系统的研究已经取得了一定的成果,但仍存在一些问题需要进一步研究和完善,具体总结如下.

- 资源访问分析技术的融合问题:针对多核共享资源管理和分析技术,现有工作的另一个根本问题是,每种技术通常只考虑一种硬件资源的分析,不同的技术之间通常不存在接口,它们难以被整合在一起以分析整个存储体系结构,也难以综合考虑不同存储层次之间的相互关联.例如,考虑两个共享资源的并行程序,如果一个程序分配较多的缓存资源,从而具有较多的缓存命中,那么这个程序对下一级共享总线的访问将会减少;对于另外一个程序而言,它在共享总线上获得的实际带宽将增加.如果不考虑这些关联,对每一层次的独立分析都不得不考虑最坏情况,那么最后综合得到的分析结果将可能非常悲观(分析值远大于程序客观上的最坏情况执行时间).因此,需要研究不同资源访问分析技术的融合,提高时间分析的准确性和安全性,最终将在实时调度中对各种共享资源的隔离和管理统一起来.这一方向的研究除了理论意义之外,还具有很大的应用价值.
- 功耗模型精度问题:在多核系统的功耗管理研究方面,还需要研究更精准的能耗模型,并需要考虑实时任务多方面的特性对功耗以及执行时间的影响.在目前的研究中,实时任务的功耗和执行时间还只与芯片的频率有关,远不够精确,还需研究更准确的模型以确保能耗计算和时间计算的准确性以及基于相应模型的优化效果.
- 任务抢占切换开销问题:目前,多核调度算法大部分还没有考虑任务之间抢占和任务迁移的开销问题.大部分研究工作都假设任务之间抢占和任务迁移的开销忽略不计或者已经计算入最差执行时间.然而,文献[136]已经发现,任务之间的抢占以及任务在处理器之间的迁移造成的开销是不可忽略的,会导致现在按照 WCET 进行的调度缺少可预测性.如何建立有效的任务抢占切换开销评价体系与方法,并将其无缝集成到当前调度算法中,还需要进一步加以研究和探讨.
- 异构多核调度问题:本文所综述的实时调度问题大多数考虑的是对等多核处理器模型思想,即各个处理器核心具有完全相同的处理能力.然而,现在多核处理器系统的一个重要的发展方向是采用异构处理器架构,其中一个重要的理论问题是如何将“Liu & Layland”资源利用率推广到异构多核处理器.

References:

[1] Turley J. The Essential Guide to Semiconductors. 4th ed., Englewood: Prentice Hall, 2002.

[2] Kopetz H. Real-Time Systems: Design Principles for Distributed Embedded Applications. 2nd ed., Springer-Verlag, 2011.

[3] Salloum CE, Elshuber M, Hoftberger O, Isakovic H, Wasicek A. The ACROSS MPSoC—A new generation of multi-core processors designed for safety-critical embedded systems. Microprocessors & Microsystems, 2013,37(8):1020–1032.

[4] Fisher S. Certifying applications in a multi-core environment: The World’s first multi-core certification to SIL 4. White Paper, SYSGO AG, 2014.

[5] http://www.upcrc.illinois.edu/

[6] http://parlab.eecs.berkeley.edu/

[7] http://www.it.uu.se/research/upmarc

[8] http://www.multipartes.eu/

[9] http://www.across-project.eu/

[10] http://www.certainty-project.eu/

[11] http://www.proxima-project.eu/

[12] Blake G, Dreslinski RG, Mudge T. A survey of multicore processors. IEEE Signal Processing Magazine, 2009,26(6):26–37.

[13] Shi W, Zhang M, Guo YF, Gong R. Technologies of real-time processor architecture: A survey. Computer Engineering and Science, 2015,37(5):857–864 (in Chinese with English abstract).

[14] https://www.nxp.com/docs/en/data-sheet/MPC5643L.pdf

[15] Parkinson PJ. Applying MILS to multicore avionics systems. In: Proc. of the Int'l Workshop on Mils: Architecture and Assurance for Secure Systems (HIPEAC). Stockholm: IEEE, 2016.

[16] Dutta S. Impact of spintronics transducers on the performance of spin wave logic circuit. In: Proc. of the 16th IEEE Int'l Conf. on Nanotechnology (IEEE-NANO). Sendai: IEEE, 2016. 990–993.

[17] Mahapatra R, Lee J, Gupta N, Manners R. Microprocessor evaluations for safety-critical, real-time applications: Authority for expenditure. Technical Report, No.43, DOT/FAA/AR-11/5, US Federal Aviation Administration, 2011.

[18] Kim H, Kandhalu A, Rajkumar R. A coordinated approach for practical OS-level cache management in multi-core real-time systems. Real-Time Systems, 2013,8114:80–89.

[19] Yan J, Zhang W. WCET analysis for multi-core processors with shared L2 instruction caches. In: Proc. of the 2008 IEEE Real-Time and Embedded Technology and Applications Symp. Missouri: IEEE, 2008. 80–89.

[20] Li Y, Suhendra V, Liang Y, Mitra T, Roychoudhury A. Timing analysis of concurrent programs running on shared cache multi-cores. In: Proc. of the IEEE Real-Time Systems Symp. Washington: IEEE, 2009. 56–57.

[21] Calandrino JM, Anderson JH. Cache-Aware real-time scheduling on multicore platforms: Heuristics and a case study. In: Proc. of the 25th Euromicro Conf. on Real-Time Systems (ECRTS). Prague: IEEE, 2008. 299–308.

[22] Cho S, Jin L, Lee K. Achieving predictable performance with on-chip shared l2 caches for many core-based real-time systems. In: Proc. of the 13th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications. IEEE, 2007. 3–11.

[23] Zhang X, Dwarkadas S, Shen K. Towards practical page coloring-based multicore cache management. In: Proc. of the 4th ACM European Conf. on Computer Systems. Nuremberg: ACM, 2009. 89–102.

[24] Ward BC, Herman JL, Kenna CJ, Anderson JH. Making shared caches more predictable on multicore platforms. In: Proc. of the 25th Euromicro Conf. on Real-Time Systems (ECRTS). Paris: IEEE, 2013. 157–167.

[25] Kim H, Kandhalu A, Rajkumar R. A coordinated approach for practical OS-level cache management in multi-core real-time systems. In: Proc. of the 25th Euromicro Conf. on Real-Time Systems. Paris: DSLP, 2013. 80–89.

[26] Suzuki N, Hyoseung Kim, de Niz D, Andersson B, Wrage L, Klein M, Rajkumar R. Coordinated bank and cache coloring for temporal protectionof memory accesses. In: Proc. of the 16th IEEE Int'l Conf. on Computational Science and Engineering (ICESS). Melbourne, 2013. 685–692.

[27] Lin J, Lu QD, Ding XN, Zhang Z, Zhang XD, Sadayappan P. Gaining insights into multicore cache partitioning: Bridgingthe gap between simulation and real systems. In: Proc. of the 14th IEEE Int'l Symp. on High Performance Computer Architecture (HPCA). Salt Lake City: IEEE, 2008. 367–378.

[28] Chen G, Huang K, Cheng L, Hu B, Knoll A. Dynamic partitioned cache memory for real-time MPSoCs with mixed criticality. Journal of Circuits, Systems and Computers, 2016,25(6):1650062.

[29] Chen G, Hu B, Huang K, Knoll A, Liu D, Stefanov T, Li F. Reconfigurable cache for real-time MPSOCs: Scheduling and implementation. In: Microprocessors and Microsystems. 2016. 200–214.

[30] Chen G. Automatic cache partitioning and time-triggered scheduling for real-time MPSoCs. In: Proc. of the 2014 Int'l Conf. on ReConFigurable Computing and FPGAs (ReConFig14). Cancun: IEEE, 2014. 1–8.

[31] Guan N, Stigge M, Yi W. Cache-Aware scheduling and analysis for multicores. In: Proc. of the Embedded Software, Grenoble. ACM, 2009. 245–254.

[32] Liu TT, Zhao YC, Li MM, Xue CJ. Joint task assignment and cache partitioning with cache locking for WCET minimization on MPSoC. Journal of Parallel and Distributed Computing, 2011,71(11):1473–1483.

[33] Hardy D, Piquet T, Puaut I. Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches. In: Proc. of the 30th IEEE Real-Time Systems Symp. Washington: IEEE, 2009. 68–77.

[34] Chattopadhyay S, Roychoudhury A, Mitra T. Modeling shared cache and bus in multicores for timing analysis. In: Proc. of the 13th Int'l Workshop on Software Compilers for Embedded Systems. Scottsdale: ACM, 2010.

[35] Schranzhofer A, Chen J, Thiele L. Timing analysis for TDMA arbitration in resource sharing systems. In: Proc. of the 16th IEEE Real-Time and Embedded Technology and Applications Symp. San Diego: IEEE, 2010. 215–224.

[36] Kelter T, Falk H, Marwedel P, Chattopadhyay S, Roychoudhury A. Bus-Aware multicore WCET analysis through TDMA offset bounds. Real-Time Systems, 2011,6794(29):3–12.

[37] Rosen J, Andrei A, Eles P, Peng Z. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In: Proc. of the 28th IEEE Int'l Real-Time Systems Symp. Tucson: IEEE, 2007. 48–60.

[38] Pellizzoni R, Schranzhofer A, Chen J, Caccamo M, Thiele L. Worst case delay analysis for memory interference in multicore systems. In: Proc. of the Design, Automation, and Test in Europe (DATE). Dresden: IEEE, 2010,170(1):741–746.

[39] Gustavsson A, Ermedahl A, Lisper B, Pettersson P. Towards WCET analysis of multicore architectures using UPPAAL. In: Proc. of the Int'l Workshop on Worst-Case Execution Time Analysis. Brussels, 2010. 101–112.

[40] Lü M, Wang Y, Guan N, Yu G. Combining abstract interpretation with model checking for timing analysis of multicore software. In: Proc. of the 31st IEEE Real-Time Systems Symp. San Diego: IEEE, 2010. 339–349.

[41] Yun H, Yao G, Pellizzoni R, Caccamo M, Sha L. Memory access control in multiprocessor for real-time systems with mixed criticality. In: Proc. of the 24th Euromicro Conf. on Real-Time Systems (ECRTS). Pisa: IEEE, 2012. 299–308.

[42] Paolieri M, Quiñones E, Cazorla FJ. Timing effects of DDR memory systems in hard real-time multicore architectures: Issues and solutions. ACM Trans. Embedded Computing System, 2013.

[43] Pao M. An analyzable memory controller for hard real-time CMPs. IEEE Embedded Systems Letters, 2010,1(4):86–90.

[44] Akesson B, Goossens K. Architectures and modeling of predictable memory controllers for improved system integration. In: Proc. of the 2011 Design, Automation Test in Europe Conf. Exhibition (DATE). Grenoble: IEEE, 2011. 1–6.

[45] Liu C, Layland J. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM, 1973.

[46] Davis PI, Burns A. A survey of hard real-time scheduling for multiprocessor systems. ACM Computer Surveys, 2011,43(4):1–44.

[47] Li RF, Liu Y, Xu C. A survey of task scheduling research progress on multiprocessor system-on-chip. Journal of Computer Research And Development, 2008,45(9):1620–1629 (in Chinese with English abstract).

[48] Cousot P, Cousot R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fix points. In: Proc. of the POPL. 1977. 238–252.

[49] Li YTS, Malik S. Performance analysis of embedded software using implicit path enumeration. In: Proc. of the 32nd Annual ACM/IEEE Design Automation Conf. (DAC). Brighton: IEEE, 1995. 450–460.

[50] Lü MS, Guan N, Wang Y. Survey of cache analysis for worst-case execution time estimation. Ruan Jian Xue Bao/Journal of Software, 2014,25(2):179–199 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4529.htm [doi: 10.13328/j.cnki.jos.004529]

[51] Ungerer T, Cazorla F, Sainrat P, *et al*. Merasa: Multicore execution of hard real-time applications supporting analyzability. IEEE Micro, 2010,30(5):66–75.

[52] Kim H, de Niz D, Andersson B, Klein M, Mutlu O, Rajkumar R. Bounding and reducing memory interference in COTS-based multi-core systems. Real-Time System, 2016,52(3):356–395.

[53] Waez MTB, Dingel J, Rudie K. A survey of timed automata for the development of real-time systems. Computer Science Review, 2013,9:1–26.

[54] Baruah SK, Mok AK, Rosier LE. Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proc. of the Real-Time Systems Symp. IEEE, 1990. 182–190.

[55] Baruah S, Chen D, Gorinsky S, Mok A. Generalized multiframe tasks. Real-Time Systems, 1999,17(1):5–22.

[56] Stigge M, Ekberg P, Guan N, Wang Y. The digraph real-time task model. In: Proc. of the 17th IEEE Real-Time and Embedded Technology and Applications Symp. Chicago: IEEE, 2011. 71–80.

[57] Stigge M, Ekberg P, Wang Y. The fork-join real-time task model. ACM SIGBED Review, 2013, 20.

[58] Baruah S. The non-cyclic recurring real-time task model. In: Proc. of the 31st IEEE Real-Time Systems Symp (RTSS). San Diego: IEEE, 2010. 173–182.

[59] Saifullah A, Agrawal K, Lu CY, Gill CD. Multi-Core real-time scheduling for generalized parallel task models. In: Proc. of the 31st IEEE Real-Time Systems Symp. (RTSS). San Diego: IEEE, 2010. 217–226.

[60] Chakraborty S, Künzli S, Thiele L. A general framework for analysing system properties in platform-based embedded system designs. In: Proc. of the Design, Automation and Test in Europe (DATE). Munich: IEEE, 2003. 10190.

[61] Cruz R. A calculus for network delay, parts I and II. IEEE Trans. on Information Theory, 1991,37:114–141.

[62] Wandeler E, Thiele L. Characterizing workload correlations in multi processor hard real-time systems. In: Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp. San Francisco: IEEE, 2005. 46–55.

[63] Stoimenov N, Chakraborty S, Thiele L. Interface-Based design of real-time systems. In: Advances in Real-Time Systems. 2012. 83–101.

[64] Chen G, Huang K, Buckl C, Knoll A. Applying pay-burst-only-once principle for periodic power management in hard real-time pipelined multiprocessor systems. ACM Trans. on Design Automation of Electronic Systems (TODAES), 2015,20(2):1–27.

[65] Cheng L, Huang K, Chen G, Hu B, Knoll A. Minimizing peak temperature for pipelined hard real-time systems. In: Proc. of the Design, Automation and Test in Europe, EDA Consortium. 2016. 1090–1095.

[66] Huang K, Thiele L. Performance analysis of multimedia applications using correlated streams. In: Proc. of the EDA Consortium. 2007. 912–917.

[67] Jonsson B, Perathoner S, Thiele L, Wang Y. Cyclic dependencies in modular performance analysis. In: Proc. of the 14th Int'l Conf. on Embedded Software (EMSOFT). Atlanta: ACM, 2008. 179–188.

[68] Carpenter J, Funk S, Holman P, Srinivasan A, Anderson J, Baruah S. A categorization of real-time multiprocessor scheduling problems and algorithms. In: Handbook of Scheduling Algorithms Methods & Models and Performance Analysis, Production Planning & Control. 2004.

[69] Liu C. Scheduling algorithms for multiprocessors in a hard real-time environment. In: Proc. of the Int'l Conf. on Information, Communications and Signal Processing. 1969. 1470–1474.

[70] Burchard A, Liebeherr J. New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. on Computers, 1995, 44(12):1429–1442.

[71] Oh Y, Sang HS. Allocating fixed-priority periodic tasks on multiprocessor systems. Real-Time Systems, 1995,9(3):207–239.

[72] Andersson B, Jonsson J. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In: Proc. of the IEEE Euromicro Conf. on Real-Time Systems (ECRTS). Porto: IEEE, 2003.

[73] Andersson B, Bletsas K. Sporadic multiprocessor scheduling with few preemptions. In: Proc. of the 2008 Euromicro Conf. on Real-Time Systems (ECRTS). Prague: IEEE, 2008. 322–334.

[74] Kato S, Yamasaki N. Semi-Partitioned fixed-priority scheduling on multiprocessors. In: Proc. of the IEEE Symp. on Real-Time and Embedded Technology and Applications. IEEE Computer Society, 2009. 23–32.

[75] Kato S, Yamasaki N. Portioned static-priority scheduling on multiprocessors. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing. Sydney: IEEE, 2008. 1–12.

[76] Lakshmanan K, Rajkumar R, Lehoczky JP. Partitioned fixed-priority preemptive scheduling for multi-core processors. In: Proc. of the 2008 Euromicro Conf. on Real-Time Systems (ECRTS). Dublin: IEEE, 2009. 239–248.

[77] Guan N, Stigge M, Wang Y, Yu G. Fixed-Priority multiprocessor scheduling with Liu and Layland's utilization bound. In: Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS). Stockholm: IEEE, 2010. 165–174.

[78] Phillips C, Stein C, Torng E, Wein J. Optimal time-critical scheduling via resource augmentation. In: Proc. of the ACM Symp. on theory of Computing, 2002,32(2):163–200.

[79] Baker T. An analysis of EDF schedulability on a multiprocessor. IEEE Trans. on Parallel Distribution System, 2005,16(8):760–768.

[80] Baker T. An analysis of fixed-priority schedulability on a multiprocessor. Real-Time Systems, 2006,32(1-2):49–71.

[81] Bertogna M, Cirinei M, Lipari G. Improved schedulability analysis of EDF on multiprocessor platforms. In: Proc. of the 17th Euromicro Conf. on Real-Time Systems (ECTRS). Palma de Mallorca: IEEE, 2005. 209–218.

[82] Bertogna M, Cirinei M. Response-Time analysis for globally scheduled symmetric multiprocessor platforms. In: Proc. of the 30th IEEE Int'l Real-Time Systems Symp. (RTSS). Washington: IEEE, 2007. 149–160.

[83] Bertogna M, Baruah S. Tests for global EDF schedulability analysis. Journal of Systems Architecture, 2011,57(5):487–497.

[84] Guan N, Stigge M, Wang Y, Yu G. New response time bounds for fixed priority multiprocessor scheduling. In: Proc. of the 30th IEEE Real-Time Systems Symp. (RTSS). Washington: IEEE, 2009. 387–397.

[85] Sun YC, Lipari G, Guan N, Wang Y. Improving the response time analysis of global fixed-priority multiprocessor scheduling. In: Proc. of the 20th IEEE Conf. on Embedded and Real-Time Computing Systems and Applications. Chongqing: IEEE, 2014. 1–9.

[86] Baruah SK, Cohen NK, Plaxton CG, Varel DA. Proportionate progress: A notion of fairness in resource allocation. Algorithmica, 1996,15(6):600–625.

[87] Baruah SK, Gehrke J, Plaxton CG. Fast scheduling of periodic tasks on multiple resources. In: Proc. of the Int'l Parallel Processing Symp. IEEE, 1995. 280–288.

[88] Anderson JH, Srinivasan A. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In: Proc. of the 13th Euromicro Conf. on Real-Time Systems (ECRTS). Delft: IEEE, 2001. 76–85.

[89] Leontyev H, Chakraborty S, Anderson JH. Multiprocessor extensions to real-time calculus. Real-Time Systems, 2011,47(6): 562–617.

[90] Schranzhofer A, Pellizzoni R, Chen JJ, Thiele L, Caccamo M. Timing analysis for resource access interference on adaptive resource arbiters. In: Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp (RTAS). Chicago: IEEE, 2011. 213–222.

[91] Vestal S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In: Proc. of the 30th IEEE Int'l Real-Time Systems Symp. (RTSS). Washington: IEEE, 2007. 239–243.

[92] Burns A, Davis R. Mixed criticality systems—A review. Technical Report, New York: University of York, Department of Computer Science, 2016.

[93] Ren JK, Phan LTX. Mixed-Criticality scheduling on multiprocessors using task grouping. In: Proc. of the 27th Euromicro Conf. on Real-Time Systems (ECRTS). Lund: IEEE, 2015. 25–34.

[94] Chen G, Guan N, Liu D, He Q, Huang K, Stefanov T, Wang Y. Utilization-Based scheduling of flexible mixed-criticality real-time tasks. IEEE Trans. on Computers, 2017,99:1.

[95] Shin I, Easwaran A, Lee I, Hierarchical scheduling framework for virtual clustering of multiprocessors. In: Proc. of the Euromicro Conf. on Real-Time Systems (ECRTS). Prague: IEEE, 2008. 181–190.

[96] Shin I, Lee I. Periodic resource model for compositional real-time guarantees. In: Proc. of the 24th IEEE Real-Time Systems Symp. (RTSS). Toronto: IEEE, 2003. 2–13.

[97] Burmyakov A, Bini E, Tovar E. Compositional multiprocessor scheduling: The GMPR interface. Real-Time Systems, 2014, 50(3):342–376.

[98] Bini E, Bertogna M, Baruah S. Virtual multiprocessor platforms: Specification and use. In: Proc. of the Real-Time Systems Symp. (RTSS). Washington: IEEE, 2009. 437–446.

[99] Lipari G, Bini E. A framework for hierarchical scheduling on multiprocessors—From application requirements to run-time allocation. In: Proc. of the 31st IEEE Real-Time Systems Symp. (RTSS). San Diego: IEEE, 2010. 249–258.

[100] Xi S, Wilson J, Lu C, Gill C. RT-Xen: Towards real-time hypervisor scheduling in Xen. In: Proc. of the 9th ACM Int'l Conf. on Embedded Software (EMSOFT). IEEE, 2011.

[101] Xi SS, Xu M, Lu CY, Phan LTX, Gill C, Sokolsky O, Lee I. Real-Time multi-core virtual machine scheduling in Xen. In: Proc. of the 14th Int'l Conf. on Embedded Software (EMSOFT). New Delhi: IEEE, 2014. 1–10.

[102] Jing W, Guan N, Wang Y. Performance isolation for real-time systems with Xen hypervisor on multi-cores. In: Proc. of the 20th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA). 2014. 1–7.

[103] Kim H, Rajkumar R. Real-Time cache management for multi-core virtualization. In: Proc. of the 2016 Int'l Conf. on Embedded Software (EMSOFT). Pittsburgh: ACM, 2016. 1–10.

[104] Xu M, Phan L, Choi HY, Lee I. vCAT: Dynamic cache management using CAT virtualization. In: Proc. of the 2017 IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS). Pittsburgh: IEEE, 2017. 211–221.

[105] Vargas R, Quiñones E, Marongi A. OpenMP and timing predictability: A possible union. In: Proc. of the Design, Automation, and Test in Europe (DATE). Grenoble: IEEE, 2015. 617–620.

[106] Li J, Chen JJ, Agrawal K, Lu CY, Gill C, Saifullah A. Analysis of federated and global scheduling for parallel real-time tasks. In: Proc. of the Euromicro Conf. on Real-Time Systems (ECRTS). Madrid: IEEE, 2014. 85–96.

[107] Li J, Luo Z, Ferry D, Agrawal K, Gill CD, Lu CY. Global EDF scheduling for parallel real-time tasks. Real-Time Systems, 2015, 51(4):395–439.

[108] Saifullah A, Ferry D, Li J, Agrawal K, Lu CY, Gill CD. Parallel real-time scheduling of DAGs. IEEE Trans. on Parallel & Distributed Systems, 2014,25(12):3242–3252.

[109] Lakshmanan K, Kato S, Rajkumar R. Scheduling parallel real-time tasks on multi-core processors. In: Proc. of the 31st IEEE Real-Time Systems Symp. (RTSS). San Diego: IEEE, 2010. 259–268.

[110] Melani A, Bertogna M, Bonifaci V, Marchetti-Spaccamela A, Buttazzo GC. Response-Time analysis of conditional DAG tasks in multiprocessor systems. In: Proc. of the IEEE Euromicro Conf. on Real-Time Systems (ECRTS). Lund: IEEE, 2015. 211–221.

[111] Fonseca JC, Nélis V, Raravi G, Pinho LM. A multi-DAG model for real-time parallel applications with conditional execution. In: Proc. of the 30th Annual ACM Symp. on Applied Computing (SAC). New York: ACM. 2015. 1925–1932.

[112] Baruah S, Bonifaci V, Marchetti-Spaccamela A. The global EDF scheduling of systems of conditional sporadic DAG tasks. In: Proc. of Euromicro Conf. on Real-Time Systems (ECRTS). Lund: IEEE, 2015. 222–231.

[113] Silva F, Lopes E, Aude E, Mendes F, Silveira J, Serdeira H, Martins M, Cirne W. Response time analysis of gang scheduling for real time systems. In: Proc. of the Int'l Symp. on Performance Evaluation of Computer and Telecommunication Systems. 2002.

[114] Goossens J, Berten V. Gang ftp scheduling of periodic and parallel rigid real-time tasks. Computer Science, 2012.

[115] Borkar S. Thousand core chips: A technology perspective. In: Proc. of the 44th Annual Design Automation Conf. (DAC). New York: ACM, 2007. 746–749.

[116] Esmaeilzadeh H, Blem E, Amant R, Sankaralingam K, Burger D. Dark silicon and the end of multicore scaling. In: Proc. of the SIGARCH Computer Architecture News. ACM, 2011. 365–376.

[117] Martin SM, Flautner K, Mudge T, Blaauw D. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In: Proc. of the IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD). ACM/ IEEE, 2002. 721–725.

[118] Jejurikar R, Pereira C, Gupta R. Leakage aware dynamic voltage scaling for real-time embedded systems. In: Proc. of the 41st Design Automation Conf. (DAC). San Diego: ACM, 2004.

[119] Pagani S, Chen J. Energy efficiency analysis for the single frequency approximation (SFA) scheme. In: Proc. of the 19th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications. IEEE, 2013. 1–25.

[120] Pouwelse J, Langendoen K, Sips HJ. Application-Directed voltage scaling. IEEE Trans. on VLSI Systems, 2003, 812–826.

[121] Aydin H, Yang Q. Energy-Aware partitioning for multiprocessor real-time systems. In: Proc. of the Int'l Parallel and Distributed Processing Symp. Nice: IEEE, 2003. 9.

[122] Moreno G, de Niz D. An optimal real-time voltage and frequency scaling for uniform multiprocessors. In: Proc. of the 2012 IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications. Seoul: IEEE, 2012. 21–30.

[123] Zhu D, Melhem R, Childers B. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. In: Proc. of the IEEE Transactions on Parallel and Distributed Systems. Nice: IEEE, 2003. 686–700.

[124] Xu H, Kong F, Deng Q. Energy minimizing for parallel real-time tasks based on level-packing. In: Proc. of the 2012 IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications. Seoul: IEEE, 2012. 98–103.

[125] Chen JJ, Kuo TW. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In: Proc. of the 2005 Int'l Conf. on Parallel Processing (ICPP). Oslo: IEEE, 2005. 13–20.

[126] Lu J, Guo Y. Energy-Aware fixed-priority multi-core scheduling for real-time systems. In: Proc. of the 17th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications. Toyama: IEEE, 2011. 277–281.

[127] Liu J, Guo JH. Energy efficient scheduling of real-time tasks on multi-core processors with voltage islands. In: Future Generation Computer System. 2016. 202–210.

[128] Kong F, Yi W, Deng Q. Energy-Efficient scheduling of real-time tasks on cluster-based multicores. In: Proc. of the 2011 Design, Automation & Test in Europe (DAC). Grenoble: ACM, 2011. 1135–1140.

[129] Guan N, Stigge M, Yi W, Yu G. Fixed-Priority multiprocessor scheduling with Liu and Layland's utilization bound. In: Proc. of the 16th IEEE Real-Time and Embedded Technology and Applications Symp. Stockholm: IEEE, 2010. 165–174.

[130] Lakshmanan K, Rajkumar R, Lehoczky J. Partitioned fixed-priority preemptive scheduling for multi-core processors. In: Proc. of the 21st Euromicro Conf. on Real-Time Systems. Dublin: IEEE, 2009. 239–248.

[131] ITRS 2011. Int'l Technology Roadmap for Semiconductors. 2011. http://www.itrs.net/reports.html

[132] Chen G, Huang K, Buckl C, Knoll A. Energy optimization with worst-case deadline guarantee for pipelined multiprocessor systems. In: Proc. of the Design, Automation and Test in Europe (DATE). Grenoble: ACM, 2013. 45–50.

[133] Chen G, Huang K, Knoll A. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. ACM Trans. on Embedded Computing Systems (TECS), 2014,13:40.

[134] Wang Y, Liu H, Liu D, Qin ZW, Shao ZL, Sha E. Overhead-Aware energy optimization for real-time streaming applications on multiprocessor system-on-chip. ACM Trans. on Design Automation of Electronic Systems (TODAES), 2011,16(2):1–32.

[135] Chen G, Huang K, Knoll A. Adaptive dynamic power management for hard real-time pipelined multiprocessor systems. In: Proc. of the 20th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA). Chongqing: IEEE, 2014. 1–10.

[136] Brandenburg BB, Calandrino JM, Anderson JH. On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In: Proc. of the Real-Time Systems Symp. Tennessee: IEEE, 2008. 157–169.

附中文参考文献:

[13] 石伟,张明,郭御风,龚锐.实时微处理器体系结构综述.计算机工程与科学,2015,37(5):857–864.

[47] 李仁发,刘彦,徐成.多处理器片上系统任务调度研究进展评述.计算机研究与发展,2008,45(9):1620–1629.

[50] 吕鸣松,关楠,王义.面向 WCET 估计的 Cache 分析研究综述.软件学报,2014,25(2):179–199. http://www.jos.org.cn/1000-9825/4529.htm [doi: 10.13328/j.cnki.jos.004529]

陈刚(1985－),男,湖北咸宁人,博士,副教授,CCF 会员,主要研究领域为嵌入式系统,实时系统,计算机体系结构.

吕鸣松(1980－),男,博士,副教授,CCF 会员,主要研究领域为实时系统时间分析,实时操作系统.

关楠(1981－),男,博士,教授,CCF 会员,主要研究领域为实时系统,嵌入式系统.

王义(1961－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为形式化方法,实时嵌入式系统.