基于WCET的多核共享资源冲突分析与约束研究

甘志华[1,2] 古志民[1] 安立奎[1] 赵 鑫[1]
(北京理工大学计算机学院 北京 100081)[1] (河南大学软件学院 开封 475004)[2]

摘 要 随着片上多核处理器在嵌入式实时系统中的应用,片上共享资源给任务的WCET分析带来诸多挑战,使得对多核共享资源冲突问题的研究变得非常重要。依据研究的目标,可以把目前已有的研究分为面向共享资源冲突分析和面向共享资源冲突约束两大类。对于面向共享资源冲突分析问题,探讨了不同共享资源冲突产生的原因,概括和比较了典型的冲突分析方法的优势和局限性;对于面向共享资源冲突约束问题,给出了其主要的研究内容,并评述和分析了几种主流的冲突约束方法。最后针对目前的研究状况指出了一些研究方向。

关键词 片上多核处理器,最坏执行时间,冲突分析,冲突约束
中图法分类号 TP302 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.004

Research on Constraint Conflicts and Analysis of Shared Resources in CMP Based on WCET

GAN Zhi-hua[1,2] GU Zhi-min[1] AN Li-kui[1] ZHAO Xin[1]
(School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)[1]
(Software School, Henan University, Kaifeng 475004, China)[2]

Abstract Chip multi-processor (CMP) is well suited to fulfill the increasing performance requirement of real-time embedded systems. However, the use of CMP introduces new challenges to WCET analysis due to conflicts of shared resources. In this paper, the newest research achievements about conflicts of shared resources were introduced and divided into two main categories, i.e., conflict analysis and constraint conflict, based on their research objective. For the conflict analysis of shared resources, we discussed the different causes of shared resources conflict, compared and surveyed the limitation and superiority of different types of conflict analysis approaches. For constraint conflict, the major research directions were presented, meanwhile, several mainstream method of constraint conflicts were investigated and compared respectively. At last, several major issues and research direction of conflict of shared resources for further exploration were pointed out.

Keywords Chip multi-processor (CMP), Worst case execution time (WCET), Conflict analysis, Constraint conflict

1 引言

随着嵌入式高端实时应用(如复杂医疗系统、汽车控制系统、电力线监控系统)的日益丰富,越来越多的嵌入式多核处理器(CMP, Chip Multi-processor)被采用。与单核处理器相比,片上多核处理器通常采用多个处理器核心共享片上总线、最后一级Cache、预取部件、DRAM控制器等设计方式。这种设计能保证共享资源被灵活有效地利用,确保数据通信与共享的高效性,但这也给CMP在实时系统中的应用带来诸多挑战:

(1)实时任务的可调度分析及时间特性验证(Timing Validation)[1]需要知道最坏情况下的执行时间WCET(Worst Case Execution Time),但CMP中任务对共享资源访问存在冲突[2],如总线等待延迟;并行任务(线程、进程)对共享Cache存在冲突访问,这给任务的WCET分析带来了困难。

(2)不同实时任务对共享资源的需求存在差异性,同时共享资源总是有限的,如实时任务的访存特征、时空局部性、核心工作集(Working Set)大小等不尽相同[3],因而对共享资源的争夺能力不同,这不仅会导致系统性能不可预测、产生多余的功耗、系统QoS无法保证等问题,也给任务的WCET计算带来了更多的复杂性。

为了分析共享资源对任务WCET的影响,解决其给系统带来的不确定性等问题,以满足实时系统性能需求,研究者们从各个层面和角度提出了分析方法与解决方案,以适用于不同类型的实时系统需求。目前的研究主要集中在片上共享总线和共享Cache上,因此,本文主要针对这两种共享资源的冲突问题进行阐述。

2 基于WCET的冲突分析与约束概述

实时系统中需要事先知道任务的WCET,以进一步判断任务是否违背截止期(Deadline),分析可调度性和验证时间特性。由于受程序控制流、处理器功能特性及执行环境等多种

到稿日期:2013-10-18 返修日期:2013-12-26 本文受国家自然科学基金(61370062,61070029)资助。
甘志华(1980-),男,博士生,讲师,主要研究方向为高性能嵌入式计算,E-mail:800521@bit.edu.cn;古志民(1964-),男,博士,教授,博士生导师,主要研究方向为高性能嵌入式计算、并行计算及缓存优化,E-mail:zmgu@x263.net(通信作者);安立奎(1978-),男,博士生,主要研究方向为高性能嵌入式计算;赵 鑫(1979-),男,博士生,主要研究方向为高性能嵌入式计算。

因素的影响[4]，任务的 WCET 分析存在困难。经过学者们的不懈努力，传统单核处理器的 WCET 分析技术取得了丰硕的成果：程序控制流分析技术从手工添加标注向自动化分析发展；处理器功能特性分析模型也从无流水线、无 Cache 的 CISC 指令集过渡到具有多级流水线、多级缓存的 RISC 指令集；出现基于抽象解释[5]技术、基于整数线性规划技术，基于模型检测[6]技术；WCET 的计算也出现基于语法树[7]技术、基于隐式路径枚举技术[8]、基于路径分析技术[9]等。同时也诞生了一批可用于实时系统 WCET 分析的工具，如 AiT、Bound-T Tool、RapiTime 等。

上述研究和工具主要针对单核处理器。片上多核处理器在实时系统中应用所带来的问题，吸引着学者们进行更深入的研究。就目前而言，从研究的目标可以分为共享资源冲突分析与共享资源冲突约束两大类：面向共享资源冲突分析的研究主要是在硬件给定的情况下，分析共享资源对任务 WCET 产生的影响，以获得精确地 WCET 估值；面向共享资源冲突约束的研究则是通过某种技术约束冲突的发生，在降低或消除共享资源对任务 WCET 分析的影响的同时，或追求系统性能的优化，或提供安全、相对精确的 WCET 估值。与面向共享资源冲突分析技术相比，共享资源冲突约束技术具有更多的灵活性。

3 片上总线冲突分析与约束

在嵌入式多核处理器上，为保证片上核心的互连与通信，总线成为最常用的互连方式之一。片上总线冲突分析技术主要在假定对共享 Cache 访问独立（不影响总线分析）的情况下讨论总线对任务 WCET 计算的影响，主要表现为精确地计算访存请求在等待总线过程所需要的延迟；片上总线冲突约束技术在基于共享 Cache 访问独立的情况下探讨如何尽可能合理高效地优化片上总线带宽的分配，以降低其对任务 WCET 的干扰，优化系统的整体性能。

3.1 片上总线延迟分析

目前常用的片上总线分配策略 TDMA 或 RoundRobin，其特点是总线被分成多个时间片，处理器核心只有在各自分配到的时间片内才可以访问总线，这种确定的时间片提供较好的可预测性，为实时系统所采用。然而对于任务 WCET 分析来说需要知道访存请求在总线上的等待延迟，最简单的方法是利用总线最大等待延迟的确定性来计算 WCET，图 1(a)为采用 TDMA 总线分配策略的 4 核处理器，每个核心被分配到的时间片长度为 Slot，假定 Core0 上的任务有 3 个访存请求 $R0$，$R1$，$R2$；可以看到这 3 个访存请求的总线等待延迟分别是 0，$R1_{delay}$（$2*\text{Slot}<R1_{delay}<3*\text{Slot}$），$R2_{delay}$（$1*\text{Slot}<R2_{delay}<2*\text{Slot}$）。事实上，我们无法获知这 3 个访存请求相对总线的偏移，上述的总线等待延迟也无法计算出来，WCET 分析只有悲观地假设每个访存请求都是总线最大等待延迟 $3*\text{Slot}+\text{LAT}$（LAT 表示访存请求缓存缺失代价）。

显然，利用这种保守的总线最大等待延迟计算得到的 WCET 很不精确，Chattopad 等人在文献[10]中提出使用相对偏移来计算总线等待延迟。假定任务中每个基本块第一个访存请求总是发生在其对应处理器核上总线时间片的起始位置；而计算基本块内其他访存请求的总线等待延迟 Wait(Δ)时，要先结合基于抽象解释分析得到的基本块内访存请求序列在 L1 Cache、L2 Cache 中的命中/缺失状态来计算出其相对第一个访存请求的偏移 Δ，再根据 Δ 与总线时间片相对关系分析该访存请求是否需要等待总线及等待的时间，如式(1)所示，其中 B 是总线的一个 Round Slot，LAT 是访存请求缺失延迟。

$$Wait(\Delta)=\begin{cases}0, & \text{if } (\lfloor\frac{\Delta}{B}\rfloor\times B+Slot-LAT)\geqslant\Delta\\ (\lfloor\frac{\Delta}{B}\rfloor+1)\times B-\Delta, & \text{否则}\end{cases} \tag{1}$$

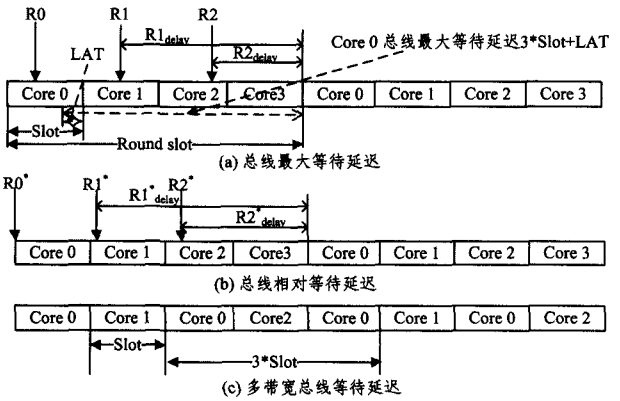

图 1 总线等待延迟分析

其理论依据在于“如果一个程序开始于其运行核上总线时间片，它的 WCET 对总线 Slot 的选择是独立的”，换句话说，上述假定第一个访存请求总是发生在其对应处理器核上总线时间片的起始位置所分析出来的 WCET 估值是安全的，如图 1(b)所示，相对于图(a)，假定 $R0$ 开始于总线时间片的起始位置。其他的访存请求也会被提前（不超过 1 个 Slot），由于第一个访存请求时序已知，能计算出其他访存请求 Δ，根据式(1)能计算出等待延迟 $R1^*_{delay}$ 和 $R2^*_{delay}$，计算得到的延迟 $R1^*_{delay}$ 和 $R2^*_{delay}$ 一定大于或等于实际 $R1_{delay}$、$R2_{delay}$，且小于总线最大等待延迟。该方法在保证 WCET 分析安全性的前提下，提高了总线等待延迟分析的精度，但在处理任务中的循环结构时，每次循环迭代都需要对齐，随着循环次数的增加，累计误差也随之变大。

针对上述不足，Timon Kelter 等人在文献[11]提出一种基于“数据流分析”技术的总线等待延迟分析方法，即通过扩展控制流图(CFG)中的基本块，把基本块中指令序列分为从不访问总线、总是访问总线及可能访问总线 3 种类型。根据序列及所属类型计算在总线的相对偏移，并把计算结果分为集合、区间两种偏移集，通过两种偏移集的数据流分析看是否能达到稳定状态，取最大值作为总线偏移。算法的最大优势在于对循环结构的处理上不需要循环对齐，而是通过对循环中集合、区间偏移集进行迭代收敛分析直到循环上限或偏移集出现不动点才结束，与文献[10]相比较，文献[11]方法的时间复杂度有所增加，但 WCET 分析的精度有所提高。

实际上，片上总线延迟分析的困难主要源于无法获得访存请求的时序。文献[12]则根据访存请求的执行顺序及基本块在时间上的先后顺序来建立时序，据此确定每个访存请求的时序范畴（上界和下界），再通过对时序范畴的分析来计算在总线上的等待延迟，分析结果比较精确。但随着任务规模

的增大，建立任务执行时序的复杂度也急剧上升，限制了该方法的应用范围。表 1 总结了这 3 种分析技术的主要优缺点。

表 1　不同总线等待延迟分析技术比较

	修改控制流	优点	缺点
基于相对偏移分析	否	易实现，基本块分析独立	循环需要对齐，分析误差相对较大
基于数据流技术分析	是	分析精度高，基本块分析独立	实现复杂，开销大
基于时序范畴分析	否	分析精度高，需根据基本块之间的关系依赖计算时序	实现较复杂，程序分析范围小

3.2　片上总线冲突约束

与总线延迟分析不同，总线冲突约束根据任务访存行为的多样性[13,14]及对总线带宽的敏感性，为不同任务分配不同的总线带宽来降低总线对任务访存的影响，优化任务的 WCET，以满足系统可调度性与时间特性。Man-Ki Yoon 等人在文献[15]中提出多带宽总线分配策略来优化任务在总线上的等待延迟。图 1(c)为多带宽总线分配策略，从 Core0 上发出的访存请求在总线最大只需要等待 1 个 slot，而从 Core1 上发出的请求则最大需要等待 3 个 Slot。这种方法本质上是 TDMA 分配技术的改进，但是能在满足实时系统中强时间约束任务需求的同时，又提供多种总线带宽。

类似地，文献[16]提出多级总线带宽的分配策略以满足多组硬实时任务集的需求，总线带宽被分为多级或组，组间采用 TDMA 方式，组内采用 TDMA 或 FCFS(First Come First Serve)方式，为任务总线带宽的分配提供更多的选择。

然而上述方法中任务一旦分配到某种总线，带宽在整个任务执行期间是不变的。实际上，任务在不同的执行阶段其总线带宽需求可能不一样，因此带宽的分配最好能根据任务不同执行阶段的需求而改变。针对这个问题 Jakob Rosen 等人在文献[17,18]提出存储总线分配策略以优化系统 WCET 的方案，其主要思想是：定义多种总线带宽类型，并依据任务的缓存缺失分布把其对总线带宽的需求分为多个密度区域；通过甘特图逆向分析任务在满足原有依赖关系的条件下不同核上任务的密度区域分配到某个总线带宽对整个系统 WCET 的影响，优先把最大带宽分给影响系统延迟最多的任务以实现整个任务集 WCET 的优化，最后把得到的带宽分配策略存储到总线仲裁缓冲区。该方案中需要增加总线缓冲区来存储任务的带宽分配结果，如果总线带宽类型或任务的密度区域多，会使带宽的分配更加精细，更能充分发挥总线性能，但同时会使存储总线带宽分配的总线缓冲区变大，导致总线仲裁时因频繁调用缓存中的带宽分配策略而影响性能，因此需要在二者之间做出平衡。

4　共享 Cache 冲突分析与约束

共享 Cache 冲突主要源于并行任务的访存请求在共享 Cache 中相互干扰，即运行在一个核上任务的 Cache 失效会替换共享 Cache 中的一些 Cache 块，而这些 Cache 块中数据可能是并行运行在另一个核上任务所需要的，同样另一个核上的任务也会破坏这个核上任务的 Cache 块，从而导致相互干扰。因此，在共享 Cache 多核架构下任务的执行时间会受到并行执行的其他任务的影响，WCET 分析也必须考虑并行任务间在共享 Cache 上的干扰。然而设计一种可以精确分析共享 Cache 冲突的方法非常困难。目前的研究主要集中在以下 3 个方面：①消除任务在共享 Cache 中的冲突；②降低任务在共享 Cache 中的冲突；③对任务在共享 Cache 中的冲突进行分析。

4.1　基于硬件结构冲突消除

为了避免任务在共享 Cache 空间的相互冲突，可以通过改变硬件结构或利用现有硬件支持的缓存划分技术来对任务 Cache 冲突进行约束以消除任务间的干扰，实现 WCET 计算的简单化。

4.1.1　改变硬件架构的冲突消除

MERASA[19,20]是欧盟第七框架计划中提出的一种基于硬件进行冲突约束、保证访存请求能被精确预测的系统架构。架构的主要部分如图 2 所示，灰色部分为增加的硬件单元，用于保证对共享资源的访存请求延迟都是有界的。新增加的硬件主要完成 3 个功能：①核内总线竞争的仲裁(ICBA)；②核间总线竞争的仲裁(XCBA)；③针对共享 L2 Cache 中不同 bank 请求的判断及选择。

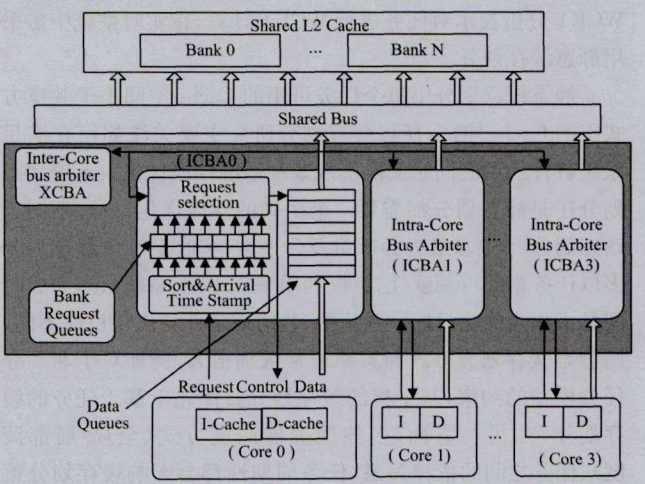

图 2　MERASA 架构

不同于传统多核架构中的总线仲裁策略，MERASA 采用 2 级总线仲裁架构，在 MERASA 架构上每个处理器核心上有一个 ICBA，当核心上的任务发出请求(Load/Store)时，对应的 ICBA 会依据任务的优先级(硬/软实时)及规定策略处理这些请求，并把结果提交给 XCBA。ICBA 的存在使得不同核的访存请求之间相互独立，任务的 WCET 分析不再依赖其它核上任务对总线的访问请求；整个架构上只有一个 XCBA，负责仲裁不同处理器核心对共享总线的请求，XCBA 自动识别不同类型的任务总线请求，硬实时任务的请求优先被响应，当有多个硬实时任务的请求时，XCBA 会采用 TDMA 和 Round-Robin仲裁策略，XCBA 保证了任务的总线最大等待延迟是由任务数来决定，而不像传统总线上的仲裁策略任务的总线最大延迟由处理器核心数决定；共享 Cache 被划分为多个 Bank，每个任务至少分到一个 Bank，Request Queues 队列主要用于保存对共享 L2 Cache 不同 Bank 的访存请求。硬件 CRU(Cache ReMap Unit)[20]根据任务 id 及访存请求的地址索引出对应的 Bank，并把请求加盖时间戳后插入对应的队列，不同 Bank 的请求可以并行，同一个 Bank 的请求由其时间戳来决定，使得访存请求对共享 L2 Cache 访问延迟也是有界的。

综上所述，MERASA 架构通过增加这 3 种硬件单元来实现冲突约束及对共享总线、共享缓存访问请求延时的有界性。但其局限性在于缓存中 bank 的数目要不少于系统中的任务数，同时尽管这种架构分别对共享总线和共享 Cache 冲突进行了约束，但对二者的约束是各自独立的。

4.1.2 基于缓存划分的冲突消除

缓存划分技术为每个核或任务分配独立的 Cache 空间，一旦某部分 Cache 空间被分配给某个核或任务，该核或任务则独享这部分 Cache 空间，其他核或任务无权替换这部分 Cache 空间中的内容，从而避免缓存竞争造成的 Cache 失效。缓存划分从类型上可分为动态划分和静态划分，动态缓存划分要实时收集任务的访存特征信息，划分机制也需周期性启动[21]，增加了系统的开销，且在不同划分周期内可能改变任务缓存空间的大小，导致无法进行有效的 WCET 分析，因此主要用于非实时领域。相对于动态划分而言，静态缓存划分一旦把缓存空间分给某个核或任务，整个任务或系统运行过程中分配的缓存空间就不会发生变化，可以利用单核的 WCET 分析技术对任务进行 WCET 计算，在实时系统中多采用静态缓存划分。

静态缓存划分中每个任务可用的 Cache 空间小于共享方式下的 Cache 空间，所以缓存划分研究主要关注如何在满足系统各种约束的同时又能实现某种系统性能优化。不论动态划分还是静态划分都需要一个明确的性能优化目标，不同于动态划分[22]追求最大吞吐量及公平性，实时系统中静态划分多以任务集的可调度性、系统利用率最小化、系统最迟完成时间最小化为优化目标。系统中，若约束条件与优化目标不同，则静态缓存划分方式与策略通常区别很大，例如对于非剥夺任务模型的约束，基于核的缓存划分要优先于基于任务的缓存划分[23]；另一方面，多核处理器调度方式（全局/局部调度）、任务之间的依赖关系、任务周期性都会影响缓存划分策略。

当前的研究为描述任务 WCET 与 Cache 空间的关系引入缓存变化收益[24-26]用于指导缓存划分，任务随着分配 Cache 空间的增加，WCET 会逐渐减小，由于任务对缓存空间变化的敏感程度不同，任务的 WCET 减少程度不同，图 3 示出实时任务 matmult 的 WCET 随缓存空间变化的曲线，可以看到在 0 至 k^{opt} 区间，WCET 值随着缓存空间的增加变化快，被称为 Cache 空间敏感区；在 k^{opt} 之后，任务的 WCET 随着缓存空间的增加变化很小，被称为 Cache 空间平坦区。显而易见在敏感区和平坦区给任务增加同样大小的缓存空间，任务的 WCET 变化不同。

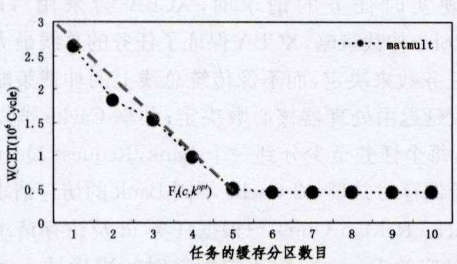

图 3 matmult 的 WCET 随缓存容量变化曲线

因此，在划分过程中要充分考虑任务分配到的缓存状态及缓存收益。典型地，Brice 等人在文献[23]中基于非剥夺 EDF 调度方式的任务模型进行缓存划分，以提高任务集的可调度性，通过任务分配使周期差值最大化与缓存分配时收益最大化达到优化目标。刘甜甜[25]针对非周期无依赖关系的任务模型进行研究，并把 Cache Lock 与 Cache 划分相结合，通过采用任务分配负载平衡与缓存分配收益的迭代分析，实现系统完成时间最小化的目标。Paolieri[26]把任务在不同运行环境下分析得到的 WCET 组织成 WCET 矩阵，结合任务的 WCET 敏感度，使用贪婪算法搜索最优划分方式，实现缓存分区的最佳化配置。

与要求每个任务必须有独占的 Cache 空间不同，Bach 在文献[24]中研究了既支持任务独占某块缓存空间，也支持多个任务共享一块缓存空间的系统模型。如果任务共享缓存划分空间，则要计算其重新加载延时（CRT，Cache Reload Time）。同时，Bach 还采用遗传算法，探讨分析了哪些任务分配独占缓存空间及分配多少缓存空间，哪些任务被放置在共享缓存空间及应该设置多大共享缓存空间才能达到系统利用率的最小化。

4.2 减少任务并行性降低 Cache 冲突

如前所述，任务的 Cache 访问行为呈现多样性，不同类型的任务其缓存行为存在差异性。计算密集型任务大部分时间用来计算，而访存密集型的任务执行时会有大量访存操作，因此如果任务调度时在满足系统约束的前提下，减少或避免访存密集型任务的并发执行，则可以在一定程度上降低任务在共享 Cache 中的相互冲突。Anderson[27,28]小组基于这种策略来提高 Cache 性能，然而这种方式并不能完全避免冲突，并且与多核处理器上执行任务的类型以及 Cache 访问行为密切相关。文献[29]提出了最优化任务调度顺序以降低冲突的方法，但其可能出现某些约束不被满足的情况。这些方法尽管都能降低 Cache 冲突，然而降低效果难以保证且无法被量化，因此多适用于软实时系统性能的优化。

4.3 共享缓存的冲突分析

与改变硬件架构和采用缓存划分技术以约束冲突不同，基于程序控制流的冲突分析不试图降低或消除任务在共享 Cache 中的冲突，而是通过分析冲突是否会发生并把分析结果反馈给 WCET 的计算，来获得精确的 WCET 估值。这类分析常用的技术框架[35]如图 4 所示。

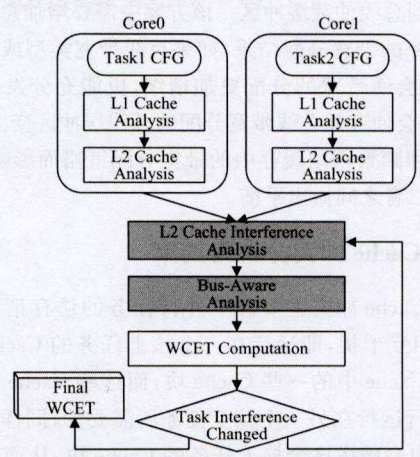

图 4 共享 Cache 的冲突分析

Cache 冲突发生主要取决于以下因素[30]：(1)访存请求在程序空间中的地址；(2)访存请求在共享 Cache 中所映射的

Cache Line;(3)访存请求发出的时间。3 个因素中的(1)和(2)通过现有的 WCET 分析技术很容易获得,与片上总线延迟分析一样,直接静态分析访存请求发出的时间存在困难,因此通常假定任务对共享 Cache 的访问都发生冲突,即任务对共享 Cache 的访问都是未命中的。以访问共享 Cache 缺失的代价来进行任务 WCET 的计算,该方法能获得安全的 WCET 估值,但极其不精确。

Yan 和 Zhang[30]首先针对这个问题进行研究,并提出一种基于地址映射的冲突分析方法,即采用抽象解释技术获得任务在私有 Cache(L1 Cache)和独占共享 Cache(L2 Cache)中的命中与缺失情况,再分析任务在 L2 Cache 中命中的访存请求是否被其他核上的任务干扰。假定另一个核上任务对 L2 Cache 的访存请求中只要有与待分析的任务在 L2 Cache 命中的访存请求映射到相同的 Cache Line 就会发生冲突,并进一步区分发生冲突的访存请求是处在顺序结构还是循环结构中,根据冲突分析结果修正任务在 L2 Cache 中的命中情况,最后通过隐式路径枚举技术分析得到任务在多核冲突情况下的 WCET。

实际上,两条指令能映射到同一 Cache Line 只是 Cache 冲突发生的必要条件。在文献[31]中,Zhang 通过对任务的逻辑顺序和 Cache 冲突图的分析发现,如果在共享 Cache 上的两个"冲突"请求之间存在逻辑访问交叉先后顺序,在任务实际执行过程中这两个冲突不会同时发生。如图 5 所示的两个任务的访存请求 ai,aj 和 bm,bn,假设{ai,bn}、{aj,bm}分别映射到相同的 Cache Line 中,根据文献[30]分析在共享 Cache 中会发生两次冲突。事实上,这两次冲突逻辑是互斥的,一个发生则另外一个必然不会发生,例如假定{ai,bn}在共享 Cache 中发生冲突,对于 Core1 上的访存请求 bn 而言,当它开始执行时,由于 bm 逻辑上的执行顺序在它之前,因此 bm 必然已经执行完毕,而 Core0 上的 aj 逻辑顺序在 ai 的后面,所以 aj 还未开始执行,因此{aj,bm}不存在冲突,从而排除了在 Cache 映射时存在冲突但是实际执行时并不会发生的互斥冲突,提高了分析的精度。

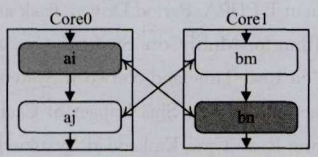

图 5　访存请求之间的逻辑干扰

与基于逻辑顺序分析不同,吕鸣松在文献[32]提出一种基于模型检测的时序建模分析 Cache 冲突的方法,陈芳园在文献[33,34]提出基于取指执行时序范畴的多核共享干扰分析方法。其主要根据是任务的执行总是需要时间的,即使访存请求在 Cache 中映射是相同的,而在实际执行过程中这些访存请求在时间上是可能完全错开的,在这种情况下这些访存请求之间不会发生冲突。

然而如何建立任务的执行时序模型成为关键问题,吕鸣松[32]采用基于时间自动机理论的 UPPAAL 模型检测器生成每个任务的自动机、L1 Cache 行为自动机及共享 L2 Cache 行为自动机,在任务自动机执行过程中通过维护一个全局时钟来更新 L2 Cache 自动机,通过任务在访问 L2 Cache 的时间消耗及建立的模拟时间分析来判断冲突是否发生,以获得任务的 WCET。陈芳园[33,34]则采用执行图(Execution Graph)分别建立任务在非干扰状态下及最大干扰状态(假定对共享 Cache 的访问请求都是未命中)下的时序,把在共享 Cache 命中的访存请求非干扰状态下的最早取指时间作为最早冲突时序 Earliest(IF_1),干扰状态下的最晚取指时间作为最晚冲突时序 Latest(IF_1),通过分析不同任务访存请求的 Earliest(IF_1)和 Latest(IF_1)之间关系判断是否存在冲突。如一个访存请求的最早冲突时序大于另一个访存请求的最晚冲突时序或一个访存请求最晚冲突时序小于另一个访存请求的最早冲突时序,冲突都不会发生。这两种通过建立时序模型来分析冲突的方法实际上是因素(3)的应用,对于解决共享 Cache 的冲突问题很有价值,但目前的研究尚不成熟,如吕的方法中当任务的执行路径很多或时钟重置次数变大时都会造成状态空间/时间爆炸问题;陈的方法中对于循环结构的时序范畴分析只是简单地将不同迭代层次的取指时序融合为一个时序范畴,导致对具有多循环结构的任务 WCET 分析过于保守。表 2 对 4 种分析方法的精确度和复杂度进行了对比。

表 2　不同共享 Cache 冲突分析技术比较

	基于地址映射分析	基于逻辑顺序分析	基于模型检测分析	基于时序范畴分析
分析精确度	低	较高	高	高
分析复杂度	低	高	高	较高
所用技术	地址映射分析	缓存冲突图及逻辑分析	时间自动机	基本块时序建模

除了微观上分析任务中的访存请求是否存在冲突外,也有学者从宏观上对实时系统任务集中任务的并行关系进行分析[35],分析具有依赖关系的任务集中某些"并行"任务执行时是否真正存在并行,通过分析任务独自运行情况的 BCET 及最大并行情况的 WCET,依据[BCET,WCET]来分析任务在执行过程中是否存在生命周期的重叠,消除一些在执行时不存在并行关系的任务之间的干扰,从而降低任务 WCET 分析的复杂度。

结束语　随着嵌入式实时系统领域一些应用的性能需求的提高,片上多核处理器被引入实时系统中,这给任务 WCET 分析带来新的挑战。为了降低共享资源的不利影响,获得安全、精确的 WCET 估值,学术界对此投入了大量的时间与精力,取得了一些成果,提出了各种降低冲突、消除冲突及分析冲突的方法。本文针对基于 WCET 的共享资源冲突与约束方法进行了综述,并分析和对比了总线及缓存冲突分析及约束技术。就目前而言,国内外关于多核架构下共享资源冲突分析及约束的问题,仍值得进一步展开研究:

1)目前的研究主要集中于共享总线和共享 Cache,对两者的研究都是独立进行的,实际上两者密切相关,访存请求在总线上的等待延迟不同,会直接影响其在共享缓存中的冲突,进而导致其在共享 Cache 中的命中状态发生变化,这种命中状态的变化又反过来影响其在总线上的等待时间,所以这两种共享资源的 WCET 分析技术在效果上可能互相影响,因此需要从全局出发,综合分析共享资源的冲突问题及约束问题。同时,系统中也包含其他的共享资源,各类共享资源之间也并非相互独立,也需要综合分析才能获得安全、精确的 WCET 估值。

2)功耗管理问题已成为实时系统中一个不得不关注的问题,尤其对电池供电的实时系统,功耗管理需要保证系统的总

功耗不能超过电池的功率或功耗预算，因此功耗也可以被视为一种重要的共享资源。随着嵌入式多核处理器中共享Cache的容量越来越大，Cache功耗在系统功耗所占的比例也越来越大，在满足系统约束的前提下，进行合理Cache划分，不仅可以有效地缓解因任务的访存请求相互冲突而产生的缓存缺失造成的动态功耗，还可以降低任务的静态功耗，即同时兼顾缓存功耗和访存请求冲突约束两方面的需求，这也是未来片上多核Cache划分机制的研究重点。

综上所述，解决多种共享资源的全局冲突分析问题及结合共享Cache功耗的冲突约束问题，有利于进一步对共享资源进行合理利用和高效优化，获得精确的WCET估值，充分发挥片上多核的潜力及优势。

参 考 文 献

[1] Quirk, William J. Verfication and Validation of Real-Time Software [M]. New York: Springer-Verlag, 1985

[2] Candra D, Guo Fei, Kim S, et al. Predicting inter-Thread Cache Contention on a Chip Multi-Processor Architecture [C]// Proceeding of the 11th International Symposium on High-Performance Computer Architecture. 2005: 340-351

[3] Subramanian R, Smaragdakis Y, Loh G H, Adaptive caches: Effective shaping of cache behavior to workload[C]// Proc of the 39th Annual IEEE/ACM into Symposium on Microarchitecture. 2006: 118-131

[4] Sha L, Abdelzaher T, arzen K-E, et al. Real-Time Scheduling Theory: A Historical Perspective [J]. Real-Time Systems, 2004, 28(2/3): 101-155

[5] Cousot P, Cousot R. Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fix points [C]// the Proceedings of ACM Symposium on Principles of Programming Language. 1997: 1-25

[6] Baier C, Katoen J-P. Principles of Model Checking [M]. Cambridge, Massaehusetts: The MIT Press, 2007

[7] Park C, Shaw A C. Experiment switches a Program Timing Tool Based on Source-Level Timing Schema [J]. IEEE Transactions on Computers, 1991, 24(5): 48-57

[8] Tsun Y, Li S, Malik S. Performance Analysis of Embedded Software Using Implicit Path Enumeration [C]// Workshop on Languages Compilers and Tools for Real-Time Systems. 1995: 456-461

[9] Stappert F, Ermedahl A, Engblom J. Efficient Longest Executable Path Search for Programs with Complex Flows and Pipeline Effects[R]. UPPsalaUniversity, 2001

[10] Chattopadhyay S, Roychoudhury A, et al. Modeling Shared Cache and Bus in Multi-cores for Timing Analysis[C]// 13th International Workshop on Software and Compilers for Embedded Systems. Association for Computing Machinery, 2010: 1042-1053

[11] Kelter T, Falk H, Marwede P. Bus-Aware Multi-core WCET Analysis through TDMA Offset Bounds [C]// 23rd Euromicro Conference on Real-Time Systems. 2011: 3-12

[12] 陈芳园，张冬松，王志英. 多核实时线程间干扰分析及WCET估值[J]. 电子学报，2012(7): 1372-1378

[13] Xie Yue-jian, Loh H. Dynamic classification of program memory behavior in CMPS[C]// Proc of CMP-MSI: 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects in Conjunction with 35th Symposium on Computer Architecture. 2008: 112-120

[14] Jia Xiao-min, Lu Ping-jing, Sun Cai-xia, et al. Dynamic Program Behavior Identification for High Performance CMPS with Private LLCS[J]. Transactions on Information and Systems, 2010, 93: 3211-3218

[15] Yoon M-K, Kim J-E, Lui Sha. Optimizing Tunable WCET with Shared Resource Allocation and Arbitration in Hard Real-Time Multicore Systems[C]// the 32nd IEEE Real-Time System Symposium(RTSS). 2011: 227-238

[16] Bourgade R, Rochange C, De Michie M, et al. MBBA: A Multi-Bandwidth Bus Arbiter for hard real-time[C]// The 5th Conference on Embedded and Multimedia Computing (EMC-10). 2011: 1101-1109

[17] Rosen J, Andrei A, Eles P, et al. Bus Access Optimization for Predictable Implementation of Real-Time Application on Multiprocessor Systems-on-Chip[C]// 28th IEEE International Real-Time Systems Symposium(RTSS). 2007: 49-60

[18] Rosen J, Neikter C-F, Eles P, et al. Bus Access Design for Combined Worst and Average Case Execution Time Optimization of Predictable Real-Time Application on Multiprocessor System-on-chip[C]// 17th IEEE Real-Time and Embedded Technology and Applications Symposium. 2011: 291-301

[19] MERASA(EU-FP7 Project)[EB]. www. merasa. org. 2007

[20] Paolieri M, Quinones E, Francisco J, et al. Hardware Support for WCET analysis of hard Real-Time Multi-core Systems[C]// Proceedings of International Symposium on Computer Architecture(ISCA2009). 2009: 56-68

[21] 贾小敏，张民选，齐树波，等. 片上多核Cache资源管理机制研究[J]. 计算机科学，2011，38(1)：295-301

[22] 王磊，刘道福，陈云雯，等. 片上多核处理器共享资源分配与调度策略研究综述[J]. 计算机研究与发展，2013，50(10)：21-32

[23] Suhendra V, Mitra T. Exploring Locking&Partitioning for predictable Shared Caches on Multi-Core[C]// Proceedings of the 45th annual Design Automation Conference. 2008: 300-303

[23] Berna B, Puaut I. PDPA: Period Driven Task and Cache Partitioning Algorithm for Multi-Core Systems[C]// 20th International Conference on Real-Time and Network Systems. 2012: 237-246

[24] Bui B D, Caccamo M, Lui Sha. Impact of Cache Partitioning on Multi-Tasking Real-Time Embedded Systems[C]// Proceeding of 14th International Conference on Embedded and Real-Time Computing Systems and Application. 2008: 101-110

[25] Liu Tian-tian, Zhao Ying-chao, Li Min-ming, et al. Joint Task Assignment and Cache Partitioning with Cache Locking for WCET minimization on MPSoC [J]. Journal of Parallel and Distributed Computing, 2011, 7(2): 1473-1483

[26] Paolieri M, Quinones E, Cazorla F J, et al. IA^3: An interference-aware allocation algorithm for Multi-core hard real-time systems [C]// 17th IEEE Real-time and Embedded Technology and Applications Symposium. 2011: 280-290

[27] Anderson J, Calandrio J, Devi U. Real-time scheduling on multicore platforms[C]// 2006 Proc of RTAS. 2006: 179-190

[28] Anderson J, Calandrino J. Parallel Real-time task scheduling on multi-core platforms[C]// the 27st IEEE Real-Time Systems Symposium(RTSS). 2006: 89-100

(下转第54页)

$def\ Saler(c(1),c(2))=$

$c(1).get()>x>c(2).put(x)>Rtime(t)>c(2).get()>y>c(1).put(y)>(Ift(y)>Rtime(t)|Iff(y)>STOP)>(Rtime(t)>STOP|c(1).get()>z)>(Ift(z)>k>c(2).put(k);STOP)$

京东仓储端 Site：

$def\ Store(c(2),c(3))=$

$c(2).get()>x>check\_store(x)>y>c(2).put(y)>(c(2).get()>z;STOP)>c(3).put(z)>delive(x)>STOP$

在 Internet 中，由于多个京东网购业务流程的实例并发执行，若简单并行执行 3 个泳道所对应的 Site：(*Saler*|*Client*|*Store*)，会导致业务流程出现死锁情况。其原因是在 BPMN 泳道分割，即水平分割，从控制流角度考虑业务运行，当业务流程跨越多个企业或控制域时，可执行业务流程则因交互顺序不匹配而导致死锁。

为了避免交互顺序不匹配情况，本文从 Web 服务交互模式设计角度，对多泳道的交互过程进行细化。换而言之，我们还考虑到服务交互模式垂直分割泳道的问题，这样不但可保证 Web 服务交互顺序的一致性，而且可利用已有的好的交互模式。另外，垂直分割将 Web 服务中计算部分和连接交互部分分离，因而利于设计开发并发性、扩展性的 Web 服务系统。因此，本文从 Web 服务交互思路重新分割和组织 Site，策略是在同一泳道中一个交互对应一个 Site。

1)下订单时，客户与销售的交互：

$def\ client_1(c(1))=find_good>x>c(1).put(x)$仓库

2)查询、销售与仓库管理人员的交互

$def\ sale_1(c(1),c(2))=c(1).get()>x>(c(2).put(x)|Rtime(t)>STOP)$

$def\ store_1(c(2))=c(2).get()>x>check\_order(x)>y>c(2).put(x)$

3)销售将是否有货的信息告知客户

$def\ sale2(c(1),c(2))=c(2).get()>x>c(1).put(x)>(Ift(x)>Rtime(t)|Iff(x)>STOP)$

4)客户自己判断是否购买或取消

$client_2(c(1))=c(1).get()>x>(Ift(x)>x;STOP)$

5)若购买，客户付款，并与销售联系，然后销售引荐仓储与客户直接联系收货事宜。由于该场景属于例 4 引用性请求模式，我们可以直接应用引用性请求模式的 Orc 表示。

结束语 并发分布式计算是云计算、服务计算的重要基石之一，如何描述和抽象 Web 服务的并发交互机制是当前一个重要的课题。本文应用 Orc 语言和 BPMN，讨论概念层次上 Web 服务交互模式，并从结构和功能层面分析和提炼出业务流程的各节点任务。下一步将从 BPMN[8] 和 Orc 语义层次细化工作。

参 考 文 献

[1] 游珍，薛锦云，应时. Apla 语言中并发分布式机制的研究[J]. 计算机科学，2012，39(1)：104-109

[2] Chinosi M，Trombetta A. Bpmn：An introduction to the standard [J]. Comput. Stand. Interfaces，2012，34(1)：124-134

[3] Decker G，Barros A. Interaction modeling using bpmn[C]// Proceedings of the 2007 International Conference on Business Process Management (BPM'07). Berlin，Heidelberg：Springer-Verlag，2008：208-219

[4] Decker G，Puhlmann F，Weske M. Formalizing service interactions[C]// Proceedings of the 4th international conference on Business Process Management (BPM'06). Berlin，Heidelberg：Springer-Verlag，2006：414-419

[5] Goel N，Shyamasundar R K. An executional framework for bpmn using orc[C]// Services Computing Conference (APSCC)，2011 IEEE Asia-Pacific. 2011：29-36

[6] Misra，Jayadev，Cook，et al. Computation orchestration：A basis for wide-area computing[J]. Software and Systems Modeling (SoSyM)，2007，6(1)：83-110

[7] Van Der Aalst W M P，Ter Hofstede A H M，Kiepuszewski B，et al. Workflow patterns[J]. Distrib. Parallel Databases，2003，14(1)：5-51

[8] Wong P Y，Gibbons J. A process semantics for bpmn[C]// Proceedings of the 10th International Conference on Formal Methods and Software Engineering (ICFEM '08). Berlin，Heidelberg：Springer-Verlag，2008：355-374

[9] Wong P Y H，Gibbons J. Formalisations and applications of bpmn[J]. Sci. Comput. Program.，2011，76(8)：633-650

(上接第 24 页)

[29] Wang Ying-xin，Cui Yan，Tao Pin，et al. Reducing Shared Cache Contention by Scheduling Order Adjustment on Commodity Multi-Cores[C]// 2011 IEEE International Parallel & Distributed Processing Symposium. 2011：984-992

[30] Yan J，Zhang W. WCET analysis for multi-core processors with shared L2 instruction caches[C]// Proc of the 14th IEEE Real-Time and Embedded Technology and Application Symposium. 2008：1179-1186

[31] Zhang W，Yan J. Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches[C]// Proc of the 15th IEEE Embedded and Real-time Computing Systems and Application. 2009：455-463

[32] Lv Ming-song，Wang Yi，Nan Guan，et al. Combining Interpretation with Model Checking for Timing Analysis of Multi-core Real-Time Software[C]// the 31st IEEE Real-Time Systems Symposium (RTSS). 2010：1176-1183

[33] Chen Fang-yuan，Zhang Dong-song，Wang Zhi-ying. Static analysis of run-time Inter-thread interference in shared Cache multi-core architecture based on instruction fetching timing[C]// 2011 Proceedings of IEEE International Conference on Computer Science and Automation Engineering. 2011：208-212

[34] 陈芳园，张东松，林聪，等. 基于取指执行时序范畴的多核共享 Cache 干扰分析[J]. 计算机研究与发展，2013，50(1)：206-217

[35] Liang Y，Suhendra V. Timing analysis of concurrent programs running on shared cache multi-cores[C]// Proc of the 30th IEEE Real-Time System Symposium. 2012：57-67