软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(6):1663−1681 [doi: 10.13328/j.cnki.jos.006243]

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

面向 AADL 模型的存储资源约束可调度性分析*

陆 寅[1], 秦树东[2], 习乐琪[1], 董云卫[1]

[1](西北工业大学 计算机学院,陕西 西安 710072)

[2](西北工业大学 软件学院,陕西 西安 710072)

通讯作者: 董云卫, E-mail: yunweidong@nwpu.edu.cn

摘 要: 嵌入式实时系统在安全关键领域变得越来越重要,其广泛应用于航空航天、汽车电子等具有严格时间约束的实时系统中.随着嵌入式系统的复杂度越来越高,在系统开发的早期设计阶段就需要对其可调度性进行分析评估.系统中的存储资源会对可调度性产生一定影响,在抢占式实时嵌入式系统引入缓存后,任务的最坏执行时间可能发生变化.因此,分析缓存相关抢占延迟对实时嵌入式系统的可调度性影响,一直以来是困扰大规模复杂系统架构设计的一个技术难题.提出一种面向软件架构级别、基于抢占调度序列的缓存相关抢占延迟计算方法,用来分析缓存相关抢占延迟约束下 AADL(架构分析和设计语言)模型的可调度性.扩展了 AADL 关于存储资源架构设计的模型元素,来支持对缓存属性进行建模,提出一种基于模型构件进行抢占序列排序、缓存相关抢占延迟时间计算和被抢占任务最坏执行时间的估算方法,来对系统架构各功能构件在共享系统存储资源下系统的可调度性进行分析.还实现了分析缓存相关抢占延迟约束下的系统任务可调度性分析工具原型,并以某型飞机机载开放式智能信息系统为例,在航空电子系统架构设计中进行尝试,验证了该方法的在复杂系统设计中的对实时性分析的可行性.

关键词: 软件架构分析与设计语言 AADL;复杂嵌入式系统;缓存相关抢占延迟;资源约束的可调度性

中图法分类号: TP311

中文引用格式: 陆寅,秦树东,习乐琪,董云卫.面向 AADL 模型的存储资源约束可调度性分析.软件学报,2021,32(6):1663−1681. http://www.jos.org. cn/1000-9825/6243.htm

英文引用格式: Lu Y, Qin SD, Xi LQ, Dong YW. On schedulability analysis of AADL architecture with storage resource constraint. Ruan Jian Xue Bao/Journal of Software, 2021,32(6):1663−1681 (in Chinese). http://www.jos.org.cn/1000-9825/6243.htm

On Schedulability Analysis of AADL Architecture with Storage Resource Constraint

LU Yin[1], QIN Shu-Dong[2], XI Le-Qi[1], DONG Yun-Wei[1]

[1](School of Computer Science, Northwest Polytechnical University, Xi'an 710072, China)

[2](School of Software, Northwest Polytechnical University, Xi'an 710072, China)

Abstract: The embedded system has been wildly applied in real-time automatic control systems, and most of these systems are safety-critical. For example, the engine control systems in an automobile, and the avionics in an airplane. It is very important to verify the schedulability property of such real-time embedded system in its early design stages, so that to avoid unexpected loss for the debugging of architecture design frictions. However, it has been proved to be a tough challenge to evaluate the schedulability of a PSRT (preemptive-scheduling real-time) system, especially when taking the constraints of system resources into consideration. The cache memory build inside the processor is such a kind of exclusive-accessing resource that is shared by all the tasks deployed on the processor. In addition, the CPRD (cache-related preemption delay) caused by preemptive task scheduling will bring extra time to the execution time to all the tasks. Thus, the CPRD should be taken into consideration when estimating the WCET (worst case executing time) of tasks in a

* 基金项目: 国家自然科学基金(61772423)
Foundation item: National Natural Science Foundation of China (61772423)
本文由"形式化方法与应用"专题特约编辑田聪教授推荐.
收稿时间: 2020-08-31; 修改时间: 2020-10-26; 采用时间: 2020-12-19; jos 在线出版时间: 2021-02-07

real-time system. A model-based architecture level schedulability evaluate and verification method, which is designed for priority based PSRT system, is proposed in this study, in order to do cache resource constrained, and CPRD related schedulability evaluation based on AADL system architecture model. In the first step, the study enhances the property set of AADL storage elements, so that to be compatible with cache memory properties in system architecture model constructing. Secondly, the study proposes a set or algorithms to: estimate the CPRDs of a task before it is completed; do system schedule simulation and construct the schedule sequence with the constraint of Cache resource and CPRDs involved; and WCET estimation of the tasks in such a CPRD considered, preemptive-scheduling execution sequence. Finally, methods mentioned above are implemented within a prototype software toolkit, which is designed to do system level schedulability evaluation and verification with CPRD constraints considered. The toolkit is tested with a use case of aircraft airborne open-architecture intelligent information system. The result shows that, compared with schedule sequence constructed without cache memory resource constraints, the WCET estimated for most tasks are extended, and sequence order is changed. In some extreme cases, when CPRD is taken into consideration, some tasks are evaluated to be incompletable. The test shows that the method and algorithms proposed in this study are feasible.

Key words: AADL; complex embedded system; cache related preemption delay; resource constraint schedulability

复杂嵌入式实时系统广泛地应用于航空航天、汽车电子、远程医疗等安全攸关应用领域,它对系统可靠性、安全性和实时性有着严格的要求,特别对实时性要求具有严格时间约束,通过使用截止期来对系统中任务的执行时间进行描述,表示任务必须在此时间之前完成执行.因此,这类系统在设计初期就要求面向软件架构进行实时性分析,根据分析结果来指导系统设计和实现,这对保障系统实时性有着积极的作用.

传统嵌入式系统可调度性判定是基于系统中每个任务的最坏执行时间(worst case execution time,简称 WCET)以及特定的调度算法,判断系统中各个任务的最坏响应时间能否满足截止期约束.WCET 可以表示为任务在处理器上执行时间的上界[1],因此,在系统模型设计阶段进行实时性分析需估算任务的 WCET.然而,最坏的执行时间又往往依赖于系统计算平台的多种资源约束,目前,在程序 WCET 估算的静态计算和动态计算中大多只考虑处理器一种资源[2].静态计算主要是通过获取到任务的控制流图,分析程序在执行过程中可能的最大路径,然后得出任务的 WCET[2].动态计算则是通过多次运行程序,在每次运行过程中测试程序的执行时间,选取这些测试结果中最大的执行时间作为该任务的 WCET.但是该过程是面向软件程序代码来计算 WCET,并不能在软件需求规划和模型设计阶段对软件的可调度性进行分析和评价.

随着嵌入式系统复杂度越来越高,软件开发采用逐步求精、层次化建模的过程来保障软件非功能属性的实现,软件非功能属性的保障是制约软件质量的一个重要挑战.近年来,人们提出一种新的架构设计与分析语言(architecture analysis and design language,简称 AADL)[3,4]来解决复杂系统非功能属性设计与分析困难的问题.AADL 支持文本化和图形化的软件架构设计,采用自顶向下的系统工程思想,可以大大缩短开发周期并节约成本.AADL 模型的可调度性验证是 AADL 非功能属性设计与验证的一个重要研究方向.AADL 模型可调度性验证一般情况下是基于给定的某种调度算法,比如单调速率调度算法(rate monotonic schedule,简称 RMS)、最早截止期优先调度算法(earliest deadline frist,简称 EDF)、固定优先级调度算法(fixed priority scheduling,简称 FPS)等.一般情况下,应用程序执行过程中,任务的优先级会随着时间动态变化,那么在调度过程中则会根据任务优先级的高低,动态进行调度.EDF 调度算法就是典型的优先级动态变化的调度策略,离截止期越近的任务优先级越高.基于 EDF 调度策略,处理器利用率的可调度性计算方法也被提出来,即:只要任务集中的所有任务处理器利用率之和小于 1,那么任务集在 EDF 调度策略下就是可调度的任务集[5].

基于时间自动机模型的 AADL 模型可调度性分析逐渐成为一个研究复杂系统实时性保障的一种方法,它利用时间自动机模型来模拟系统架构中各模型构件的动态行为,进而依据系统的实时性需求和行为规约对设计的系统模型进行可调度性分析,这对于复杂系统的功能划分和早期设计的性能指标分解具有重要的指导意义.一些国内外研究机构进行了卓有成效的尝试,例如:宾夕法尼亚大学开发的可调度性分析工具 VERSA (verification execution and rewrite system for ACSR)将 AADL 模型自动转化为 ACSR 模型,并在 VERSA 工具中建立线程的时间自动机模型,将处理器与访问连接建模为资源,数据、事件端口建模为通信通道[6];伊利诺大学香槟分校开发的一种支持对实时系统进行形式化描述、仿真和分析的语言和工具 Real-Time Maude[7,8],其首先

对系统进行形式化建模,然后进行推理分析和仿真运行,进而分析系统的可调度性;除此之外,aadl2sync 工具集支持 AADL 模型可调度性的形式化验证,但是 aadl2sync 并不支持 AADL 建模,并且在转换过程中存在缺陷,如将异步概念转换为同步概念,导致 AADL 模型不能完全转换[9];Brest 大学开发的开源的实时调度计算框架 Cheddar 工具可支持多种调度策略,支持 AADL 端口延迟、共享资源、消息队列的分析[10];国内研究者李振松等人结合 AADL 中的行为模型附录(behavior annex,简称 BA),根据 AADL 中行为模型的语法语义提出了行为模型到 UPPAL 工具的时间自动机模型转换规则,并且证明转换的正确性与有效性,在 UPPAL 工具下对系统的可调度性进行仿真验证[11].然而这些工具只考虑处理器单一资源下的可调度性分析,而在系统规划阶段,系统的架构设计需要综合分析评估多种资源配置下的系统可调度性.

复杂嵌入式实时系统通常采用抢占式调度策略来保障对关键任务处理的及时响应,抢占会对系统中任务的执行时间造成不确定性,从而增加计算成本.例如,抢占可能会导致高速缓存被干扰,被抢占任务的缓存块替换出高速缓存,当被抢占任务重新执行时则需要重新加载这些被替换出去的缓存块,这重复加载的时间就称为缓存相关抢占延迟(cache related preemption delay,简称 CRPD).在国内,有研究者提出了 CRPD 的计算方法,例如 UCB-ECB Union 算法,同时,作者也给出了该方法与其他算法之间的比较结果[12].Hai Nam Tran 等人通过建立任务的控制流图(control flow graph,简称 CFG),提出一种抢占任务路径分析的 CRPD 计算方法,为 CFG 中的每一个节点定义有用缓存块(useful cache block,简称 UCB)以及抢占缓存块(evicting cache block,简称 ECB),用 UCB-Union 计算方法对任务的 *CRPD* 进行计算[13].该方法提供了很好的思路去计算可执行构件动态运行时抢占行为所造成的抢占代价.同时,Hai Nam Tran 提出了 AADL 缓存建模方法,使用 AADL 中的 Subprogram 构件对程序控制流图进行建模,然后使用 Ada 语言扩展 Cheddar 工具,使其能够根据程序控制流图对 CRPD 进行计算[17].另外,为了保障复杂嵌入式系统任务之间的隔离,减少错误传播和干扰,通常需要采用分区调度策略,实现运行于同一计算平台中的不同安全性等级应用之间相互隔离.通过保障各分区任务的实时性,进而提高整个系统的可靠性和安全性.然而在架构设计层面,AADL 不能刻画程序执行的具体控制流图,故无法使用 AADL 现有的建模与分析方法计算出构件交互的抢占延迟.所以在对 CRPD 进行计算时,为了克服在模型层面上没有具体代码、具体实现控制流和数据流信息的问题,本文提出一种基于抢占序列的 CRPD 计算方法,该方法可以不用关心模型构件的具体代码实现,能够普适地分析 AADL 调度模型在 CRPD 约束下的可调度性.

本文面向复杂嵌入式系统,针对系统架构设计的早期阶段,研究在缓存相关抢占延迟约束下 AADL 模型的可调度性,通过扩展 AADL 存储资源建模技术,建立包含基本调度元素的分区内架构模型以及符合 ARINC653 标准的分区系统架构模型[15],研究 AADL 架构模型在缓存相关抢占延迟约束下的可调度性问题,并提出了基于抢占序列的 CRPD 计算方法以及 CRPD 约束下的 WCET 计算方法.该计算方法不关心 AADL 线程构件的具体代码实现,更适用于软件开发前期 AADL 架构模型实时性分析场景.在嵌入式软件模型设计期,提供缓存抢占延迟约束可调度性理论指导,避免在嵌入式软件开发过程中,由于任务之间抢占代价过高而导致系统实时性不达标进而造成重复开发的问题.

本文第 1 节针对 AADL 在缓存资源建模不足的方面进行资源建模能力扩展.第 2 节首先介绍缓存以及缓存相关抢占代价的计算方法,结合特定调度策略,分析系统架构中各构件运行行为造成的缓存相关延迟,提出一种基于抢占序列评估延迟的计算方法,并制定存储资源约束的任务集可调度性的判断方法和调度策略.第 3 节设计一种 AADL 架构可调度性分析工具原型架构,开展缓存约束下的某机载开放式智能信息系统可调度性分析案例实验.最后对本文的工作进行总结和展望未来的研究工作.

1 AADL 缓存资源建模

AADL 是一种用来设计和分析复杂实时嵌入式系统架构模型和分析非功能属性的建模语言,提供了存储资源 Memory 构件对系统存储资源进行建模,并且通过绑定机制刻画软件构件所使用的存储资源属性.AADL 模型支持层次化存储建模[13],使用 Memory 构件作为 Memory 构件的子组件来描述 Memory 的层次结构,并通过软件绑定到 Processor 构件关系来对软硬件视角下的存储资源形成一个映射,将存储器以段为单位进行划分.但

是由于 AADL 中缺少描述进程动态执行时的属性,不能进一步描述 AADL 构件进程、线程使用存储构件的分段属性.因此,需要利用 Memory 构件作为 Processor 的子构件对 Cache 进行建模,并通过 AADL 中 Memory 内部结构刻画出一级 Cache、两级 Cache 等 Cache 结构,其中,多核系统可采用两级 Cache 建模方式[16].

为了能在 AADL 模型层对构件的 *CRPD* 进行分析,本文对 AADL 语言 Memory 构件属性扩充,对 AADL 模型中存储层次结构进行细分.在 AADL 中使用 Memory 构件作为 Processor 构件的子构件,对高速缓存 Cache 进行建模,增加了调度资源竞争建模需要的 Memory 属性,见表 1.其中,*CacheSize* 刻画 Cache 的容量,*LineSize* 描述一个缓存块的容量,而 *CacheMissTime* 表示在一次未命中时从内存加载一个缓存块的时间延迟.一般地,针对具体型号 CPU 发生资源抢占时,从缓存中加载一个缓存块到 Cache 的时间是一个常量.当 Cache 未命中时, Cache 缺失发生,需要将缺失的数据以及相关联的块从主存中写入 Cache,然后在 Cache 中访问该数据,Cache 容量大小会影响这个过程的时延,进而对系统任务的可调度性带影响.

Table 1 Extended properties of memory component in cache modeling

表 1 Cache 模型属性

构件	属性	类型	单位
Memory	*CacheSize*	aadlinteger	Byte,KB,MB,GB
Memory	*LineSize*	aadlinteger	Byte,KB,MB,GB
Memory	*CacheMissTime*	aadlinteger	ps,ns,μs,ms,s

扩展后的 AADL Cache 模型可以刻画系统架构设计所需要的高速缓存和主存构件资源访问行为相关的属性,如 *CacheSize* 和 *CacheMissTime*,建模时可使用 applies to 关键字定义 Memory 构件的 Cache 块容量属性 *Line_Size*,也可以利用 AADL 提供的 Memory 构件属性进行主存资源建模.在设计系统架构时,可以用 AADL 构件 thread 定义系统任务及其调度执行相关的属性和行为,如调度协议、执行时间、任务周期、截止期以及存储资源访问需求等.当系统采用虚拟技术进行任务分区资源划分绑定及分区调度执行时,建模时需为每个任务分区定义调度分区时间片轮转的大小.在系统可调度性分析时,可通过提取 AADL 架构中分区与构件的行为属性及存储构件属性,见表 2,就可以计算模型中各构件执行中的访存时延,进而分析系统的可调度性.

Table 2 Component properties required for CRPD calculation

表 2 计算 CRPD 需要提取的构件模型属性

属性	所属构件
CacheSize	Memory
WordSize	Memory
LineSize	Memory
CacheMissTime	Memory
RomBudget	Thread,Thread Group
RomActual	Thread,Thread Group
RamBudget	Thread,Thread Group
RamActual	Thread,Thread Group
Source_Data_Size	Thread,Thread Group
Source_Code_Size	Thread,Thread Group
Source_Heap_Size	Thread,Thread Group
Source_Stack_Size	Thread,Thread Group
Dispatch_Protocol	Thread,Thread Group
ComputeExecutionTime	Thread,Thread Group
Period	Thread,Thread Group
Deadline	Thread,Thread Group

如果 CPU 以一种当缓存块加载完成后再从缓存中提取 CPU 所需字的方式去加载资源时,需要对 Cache 缺失情况进行分析,本文案例的 CacheMissTime 为了分析方便把它按做常数处理.

2 缓存相关延迟计算

2.1 缓存相关抢占代价

在嵌入式系统中,缓存资源对系统实时调度会带来缓存抢占延迟和任务执行时序异常两类影响.时序异常

往往会导致系统行为失效,但是不一定导致最坏的执行时间,因此时序异常不在本文不考虑范围.在抢占式系统中,当任务之间抢占发生时,被抢占任务已经加载到缓存中的内容由于缓存冲突而被替换出缓存,当被抢占任务再次执行时,需要将替换出去的内容再次加载到缓存[17],重复加载的时间就是任务调度因为缓存相关抢占造成的任务延迟代价.造成抢占延迟的根本原因是由于任务之间存储资源映射发生冲突.

在抢占式调度环境中,任务的执行效率以及系统实时性所受的影响往往来自于任务之间的抢占,任务的抢占不仅会造成上下文切换代价(context switch overhead,简称 CSH),并且也会造成缓存相关抢占延迟.研究成果表明:任务调度过程中,上下文切换代价大约在 5μs~10μs 之间,而缓存相关导致的任务抢占带来的计算延迟可以到达 1μs~10000μs,每一次计算任务抢占的延迟大小取决于系统结构及其 Cache 容量[18,19].因此,系统的实时性分析需要更多地考虑系统架构中存储资源结构及其缓存容量可能导致相关抢占引起的延迟.

由于系统模型中特定的软硬件环境下,CPU 从缓存中加载一个缓存块的时间 *CacheMissTime* 可以认为是常数,也可以用它来表示从主存中加载一个块大小的数据内容到缓存中的时间上限,所以计算缓存相关抢占代价时,只需要知道被抢占任务在恢复执行时需要重复加载缓存块的个数,就可以采用公式(1)计算出系统资源抢占延迟 CRPD[13]:

$$CRPD=g\times CacheMissTime \tag{1}$$

其中,(1) g 为被抢占任务恢复执行时需要加载的缓存块个数;(2) *CacheMissTime* 表示从主存中加载一个缓存块大小内容到缓存中所需要时间的上界,在特定硬件体系结构中可以将其看成一个常数.

在系统架构层级分析模型中,各构件的可调度性的关键是计算构件的 CRPD.由于架构模型中没有系统代码实现的信息,也就没有系统执行时真实的运行剖面信息可用来计算 CRPD,所以如何确定被抢占构件恢复执行时需要加载缓存块的个数,是系统架构可调度分析的难点.如果使用被抢占任务所有缓存块个数参与计算 CRPD,显然会高估 CRPD 的值,因为被替换出去的缓存块可能在被抢占任务恢复执行后不会再使用到.本文基于有用缓存块(useful cache block,简称 UCB)和抢占缓存块(evicting cache block,简称 ECB)的大小来估算 CRPD 的值.有用缓存块是指在程序执行的过程中不被替换出缓存的任务数据块,而抢占缓存块是指在抢占任务的执行过程中需要替换进入缓存的任务数据块.基于 UCB,ECB 的思想,人们提出了两种 CRPD 计算方法为 UCB-Union 和 ECB-Union[13].UCB-Union 方法是用来计算任务 T_k 对 T_i 抢占代价的开销,计算方法如式(2)所示[14]:

$$CRPD_{(i,k)}^{UCB\text{-}Union}=CacheMissTime\times\left|\left(\bigcup_{\forall k\in bet(i,k)}UCBs_i\right)\cap ECBs_k\right| \tag{2}$$

其中,(1) $CRPD_{(i,k)}^{UCB\text{-}Union}$ 为 UCB-Union 方法计算出的抢占任务 T_k 抢占任务 T_i 所产生的缓存相关抢占延迟;(2) $UCBs_i$ 为任务 T_i 的有用缓存块集合;(3) $ECBs_k$ 为任务 T_k 的抢占缓存块集合;(4) $bet(i,k)$ 表示被调度的任务集中优先级大于等于被抢占任务 T_i 且优先级低于任务 T_k 的任务集合.

ECB-Union 方法用来计算任务 T_k 对 T_i 抢占代价的开销,计算方法如公式(3)所示:

$$CRPD_{(i,k)}^{ECB\text{-}Union}=CacheMissTime\times\max\nolimits_{\forall k\in bet(i,k)}\left\{\left|UCBs_i\cap\left(\bigcup_{t\in hp(k)\cup T_k}ECBs_k\right)\right|\right\} \tag{3}$$

其中,(1) $CRPD_{(i,k)}^{ECB\text{-}Union}$ 为 ECB-Union 方法所计算出的任务 T_i 抢占任务 T_k 的 *CRPD*;(2) $UCBs_i$,$ECBs_k$ 的含义与公式(2)中一致;3) $hp(k)$ 表示在调度任务集中任务优先级大于任务 T_k 的任务集合.

UCB-Union 和 ECB-Union 方法都会高估缓存抢占代价,其原因是它们在 CRPD 计算时,把没有参与抢占的高优先级任务也加入了被抢占任务的 CRPD 计算中.为此,本文针对可能的缓存抢占代价高估问题提出基于抢占序列的 CRPD 计算方法,重点解决抢占发生时,如何准确计算被抢占任务的 CRPD,准确地判定系统架构中各构件的可调度性.

2.2 线程的时间自动机模型

为了刻画系统架构中各构件的运行行为,模拟构件运行的调度行为,AADL 把系统中最小执行单元线程构件的调度执行行为抽象为一个时间自动机模型.AADL 线程状态机中任务状态分为 Ready,Running,Awaiting

Resource,Awaiting Return,Awaiting Reentry 等 5 个状态,为了表示方便,本文不考虑 Awaiting Return, Awaiting Reentry 这两种状态,因为这两个状态与系统资源约束关系无关.因此,简化后的 AADL 线程执行模型可以用 3 个状态——Ready 状态、Running 状态和 Awaiting 状态以及状态之间的转移描述,如图 1 所示.其中,c 为线程在处理器上执行的时间累积,w 为线程由于资源受限阻塞的时间累积.∂c 表示该线程是否在处理器上处于计算状态并进行时间累积的标志量,当∂c=1 时,表示该线程处于执行态,并进行时间累积;否则,该线程不处于执行态,不需要时间累积.同样地,需要定义由于资源阻塞而导致的时间累积的标志量∂w,当∂w=1 时,线程处于阻塞状态,并进行时间累积;否则,线程不处于阻塞态,时间不进行累计.3 个状态间存在 4 种变迁情况:(1) Ready 状态下获得处理器时间,进入 Running 状态;(2) Running 状态下发生 Cache Miss 后会导致处理器阻塞,线程进入 Awaiting Resource 状态;(3) Awaiting Resource 状态下,当该线程得到执行所需的资源后,线程进入 Ready 状态;(4) Running 状态下的线程被更高优先级的线程抢占以后进入 Ready 状态.

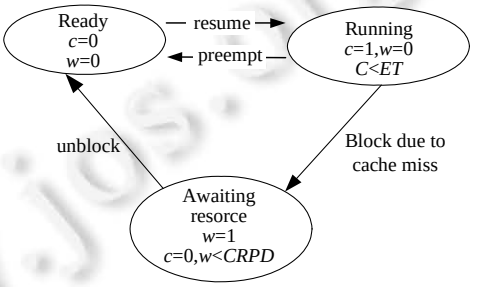


Fig.1 Simplified timed automata model of AADL thread component

图 1 简化的 AADL 线程时间自动机模型

复杂嵌入式系统的调度往往采用分区调度的方式对系统构件及其资源进行运行管理.例如在机载嵌入式系统中,就采用集成化、模块化航空电子体系结构 IMA,这种软件体系结构的嵌入式系统采用两级调度模式管理任务的运行:分区间调度和分区内调度.AADL 定义了一个附录模型 ARINC653 来支持对分区建模,分区间调度采用时间片轮转调度策略,因此,系统运行过程中还要考虑分区之间切换带来的延迟.根据 AADL ARINC653 附录模型的操作语义,我们也可以用时间自动机来描述带分区的复杂系统任务调度行为,如图 2 所示.

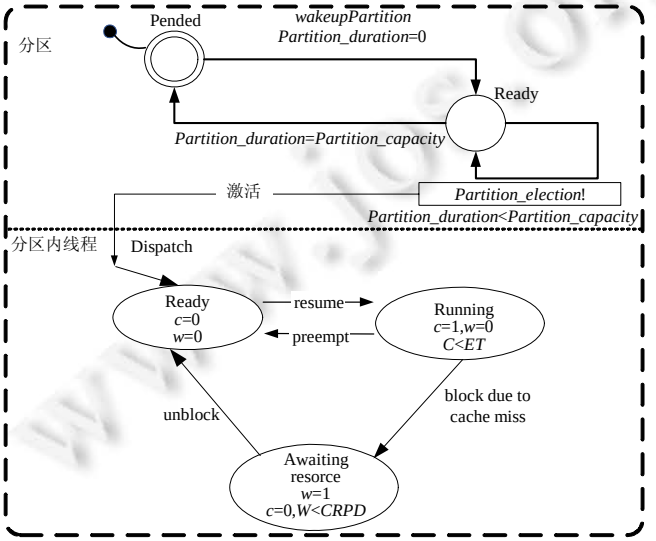


Fig.2 Timed automata model of ARINC653 partitioned system

图 2 ARINC653 分区系统时间自动机模型

系统根据分区配置表给每一个分区分配处理器时间片,分区内调度是根据各个分区得到时间片之后按照分区内指定的调度策略去调度分区内的线程.在图 2 所示的分区调度模型中,每一个系统分区状态机有两个状态:Ready 状态和 Pended 状态.Pended 表示该分区处于阻塞状态,并未得到处理器时间片信息.当 *wakeupPartition* 激活该分区时,记录分区已执行的时间 *Partition_duration*=0,并进入 Ready 状态.Ready 状态表示该分区分配到处理器时间片并激活分区内线程,选择该分区内优先级最高的线程执行.当分区实际执行时间与分区获得时间片大小(*Partition_capacity*)相等时,表明该分区的计算时间耗完,该分区中所有任务都进入阻塞状态.

2.3 抢占序列的计算

由于系统架构中不同的构件执行周期不尽相同,为了能准确、一致性地计算各构件执行的抢占代价,需要把任务集的运行基准定义在一个超周期内.任务集的超周期是任务集中各个构件任务执行周期的最小公倍数.为了更准确地描述任务集在一个超周期内的任务抢占发生时的时间开销,就需要对任务在一个超周期内发生的所有抢占,即抢占序列进行计算.为了更加准确地完成抢占序列的计算,可以将抢占方式分为显式抢占与隐式抢占两种情形.显式抢占是指抢占发生时,没有内存资源访问冲突导致 Cache 块切换的任务调度.此时,任务在特定调度策略下所产生的抢占不需要考虑每个任务抢占点的 CRPD 代价.而隐式抢占是指抢占发生,由于系统 Cache 资源有限,任务执行过程中内存资源访问冲突导致 Cache 块切换的任务调度.此时,任务在特定调度策略下发生抢占时,就需要考虑每个任务抢占点的 CRPD 代价.例如:系统中有 3 个线程构件,对应的执行任务集合为 $\{T_1,T_2,T_3\}$,各任务的调度行为属性见表 3.

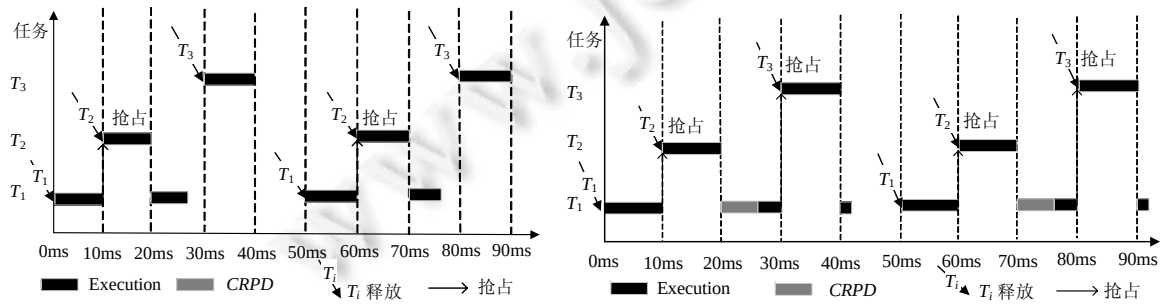
Table 3 Task properties used in scheduling simulation

表 3 任务调度模型属性

Task	Priority	φ(释放相位)	ET	Period	DeadLine
T_3	3	30	10ms	50ms	50ms
T_2	2	10	10ms	50ms	50ms
T_1	1	0	17ms	50ms	50ms

图 3(a)列出了该任务集在两个超周期内的调度序列(调度算法为固定优先级调度算法).在调度过程中,任务 T_1 在 10ms,60ms 处被任务 T_2 显示抢占.显然,任务的调度没有考虑 CRPD 代价对调度的影响.

然而,当考虑了 CRPD 代价并且任务调度过程中每个抢占点的 CRPD 代价较大时,基于显式抢占点计算出的整个任务集的 WCET 则有可能过低.如图 3(b)所示的调度过程中,假设任务 T_2 在 10ms 处对任务 T_1 所产生的抢占代价 CRPD 为 6ms,那么任务 T_1 基于 CRPD 的 WCET 为 23ms,因此会额外增加 T_3 对 T_1 的抢占.当 T_1 再次被 T_3 抢占时,如果 30ms,80ms 的抢占点中,任务的抢占代价大于 10ms 处与 60ms 处 T_2 对 T_1 的抢占代价 6ms,那么再以图 3(a)中的调度方式计算调度延迟的话,显然会低估任务 T_1 的调度延迟.因此,在计算任务集 WCET 的时候,需要考虑到这些隐式抢占点的 CRPD.



(a) 显式抢占调度序列 (b) 隐式抢占调度序列

Fig.3 Comparation of task scheduling sequence in different context

图 3 不同调度情形的任务集调度序列比较

在给定调度策略下,一个抢占序列是指任务集中的一个任务从被抢占开始到该任务再次执行之间高优先

级任务的执行序列.用 $preemptionSequence(T_i)$表示任务 T_i 从被抢占到恢复执行这一段时间内高优先级任务的执行序列,则 $preemptionSequence(T_i)=(T_x,T_y,T_z)$表示任务 T_i 被任务 T_x 抢占到任务 T_i 恢复执行之间的任务集的执行序列为 $T_x \rightarrow T_y \rightarrow T_z$.因此,我们在计算每一个执行任务的 CRPD 前,需要确定该任务的抢占序列,并把序列中每一次可能抢占的 CRPD 计算出来,并对所有抢占序列 CRPD 计算的结果进行求和,得到该任务被抢占的 CRPD.

对于RMS和EDF调度策略,系统任务在一个超周期内的抢占只能发生高优先级任务到来的时刻,我们称这些点为任务执行断点,任务抢占点是一类特殊的执行断点.所以分析任务的可调度性时,首先要计算出在一个超周期内所有任务的执行断点.假设有一个任务集 $Taskset=\{K_1,K_2,K_3\}$,其中,K_1 的周期为 2ms,K_2 的周期为 3ms,K_3 的周期为6ms,那么该任务集的超周期为其3个任务的最小公倍数,即为6ms,分别为(0ms,2ms,3ms,4ms,6ms,8ms,9ms,10ms)时刻.对于分区调度情形,分区调度时间片切换点也被认为是一种任务执行断点(如图 4 所示).

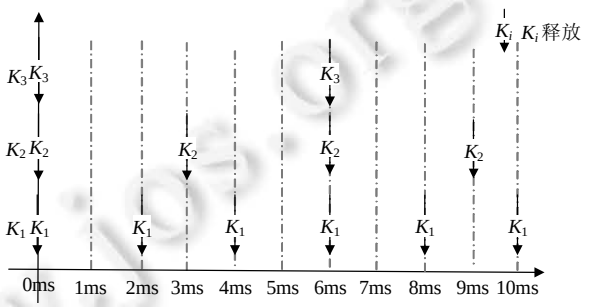


Fig.4 Dispatch of tasks in the task group within a superperiod

图 4 任务集中任务在一个超周期内的释放情况

算法 1 给出了任务集所有执行断点的计算过程.在算法 1 中有 Do-while 循环和 for 循环(设分区个数为 m,任务集中任务数量为 n),前者循环的次数为 m,后者循环次数为 n,循环之间无嵌套关系,且 m 与 n 之间满足关系:$m \leqslant n$,故算法 1 的时间复杂度为 $O(n)$.

算法 1. 计算任务集执行断点序列的过程.

Input:AADL 调度模型实例化文件;

Output:调度模型的执行断点序列 *ExecutionPointSet*;

1. 解析 AADL 调度模型文件,提取任务集合 *SM*;
2. 断点序列初始化:*ExecutionPointSet=null*;
3. 计算 *SM* 中任务的超周期 *Superperiod*;
4. **if** (采用分区调度) **then**
5. 获取分区配置信息 //包括分区调度顺序、时间片大小;
6. *temp*=1 //*temp* 标志当前的执行分区;
7. **Do**:
8. 将该分区切换点加入 *ExecutionPointSet* 集合;
9. *temp*=*temp*+1;
10. **while** (*temp*≤分区个数)
11. **end if**
12. **for** (*SM* 中的所有任务 *T*)
13. {令 *k*=0; //*k* 代表一个超周期内任务的最大执行次数
14. P_{T_i} 为任务 T_i 的周期;
15. **while** ($k^* P_{T_i}$ <*Superperiod*) //当 *k* 倍任务周期小于超周期
16. {**if** (*k* 倍周期不属于 *ExecutionPointSet*)

17. 将 *k* 倍任务周期对应的时间作为一个执行断点;
18. *k*=*k*+1;}
19. **end while**;}
20. **end for**
21. 输出执行断点集 *ExecutionPointSet*;

由于在 RMS 和 EDF 调度策略下,高优先级任务对低优先级任务的抢占一定会发生在高优先级任务到来的时刻,那么判断一个执行断点是否是抢占点,可以根据当前任务在执行断点到来时其剩余执行时间是否为零,并且是否有高优先级的任务到达来判断.所以在任务执行断点序列计算完成后,还要计算每个任务的剩余执行时间来判断执行断点是否为抢占点.如图 5 中任务 T_1 和 T_2 的执行序列为例,从第 *K*ms~*K*+5ms 处的执行断点序列为{*K*ms,*K*+1ms,*K*+2ms,*K*+4ms},假设当前执行断点为 *K*+1ms,那么上一个执行断点则为 *K*ms.*K*ms 处释放任务 T_1,且 T_1 所需要的执行时间为 1.5ms,那么在当前执行断点处,任务 T_1 并未执行完成,并且在当前执行断点 *K*+1ms 时刻,任务 T_2 到达,T_1 剩余执行时间为 0.5ms,任务 T_1 被抢占,当前执行断点即为一个抢占断点.因此,把一个任务执行过程中的所有抢占断点计算出来后,即可得到该任务调度过程中的抢占序列.

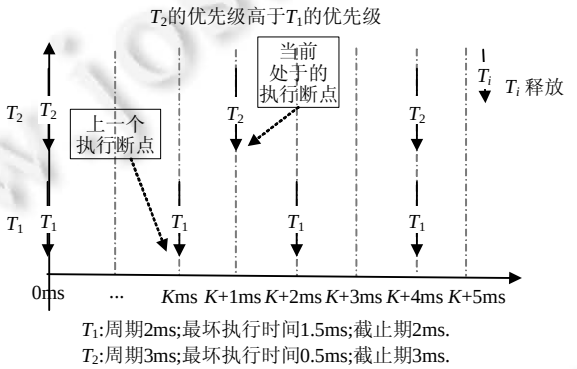


Fig.5 Calculation of preemption points

图 5 任务抢占点的判定

算法 2 给出了通过判断抢占点生成抢占序列的过程.算法 2 包含一层 Do-while 循环,循环变量为执行断点序列,该序列大小与每个任务周期大小相关,而任务周期由设计人员提供,故算法 3 的时间复杂度无法根据任务数量 *n* 进行估算.假设执行断点序列中的元素为 *k* 个,则算法时间复杂度为 *O*(*k*).

算法 2. 求解一个任务的抢占序列.

Input:任务集 *SM* 及其执行断点序列 *ExecutionPointSet*、分区配置信息表;

Output:任务集中任务在一个超周期内的抢占序列 *preemptionSequence*.

1. 任务集初始化:所有任务实例化标志 *State*=0; //*State*=0 表示任务实例未就绪或执行完成被销毁
2. 所有任务剩余执行时间 *LaveExecutionTime*=0;
3. 时钟变量 *clock*=0;
4. 完成预计时间:*time*=0;
5. 执行断点初始化:*CurPt*←*ExecutionPointSet*[0]; //设置当前执行点为执行断点序列中首元素
6. *Unfinished*=*null*; //设定上个执行断点处未完成的任务记录初始值为空
7. 抢占序列初始化:*preemptionSequence*=*null*;
8. 初始化分区标志:*r*=1; //分区初始值为第 1 个分区
9. 初始化任务预估;
10. **Do**:
11. **if** (执行断点处是否有任务要释放)

```
12.          if (当前任务状态为 Ready 状态)
13.            {判定当前任务在一个周期内未完成执行;
14.            任务集不可调度,输出抢占序列 preemptionSequence;}
15.          else
16.            将该任务状态设置为 Ready 状态;
17.          endif;
18.        endif;
19.        选择 ExecutionPointSet 序列的第 1 个元素 K_i;
20.          if (K_i 为分区断点)   //K_0,K_1,…,K_n 为执行断点序列 ExecutionPointSet 中的元素
21.            {r=r+1;
22.            选择分区列表中的下一个分区为当前分区;}
23.          endif;
24.          根据当前分区调度策略,选择优先级最高的任务 T_i;
25.          设置 T_i 任务状态 state=Running;
26.            LaveExecutionTime_i=任务 T_i 的执行时间;
27.          if (Unfinished≠null)   //执行断点处有任务 T_j 未完成,则抢占发生
28.            {添加抢占任务 T_i 到抢占序列 preemptionSequence 中;
29.            记录被抢占任务 T_j 的名称;
30.            记录抢占发生时的被抢占任务的周期 T_j;}
31.          end if
32.          time=Clock+LaveExecutionTime_i;   //time 为任务 T_i 预估完成时的时间,Clock 为当前时间
33.          if (time<ExecutionPointSet[1])
34.            {Remove(ExecutionPointSet[0];   //删除执行断点序列中的第 1 个元素
35.            ExecutionPointSet[0]=time;
36.            LaveExecutionTime_i=0;
37.            State(T_i)=0;
38.            CurPt←ExecutionPointSet[0];}
39.          else   //否则,当前任务 T_i 并未完成执行
40.            {将任务 T_i 加入 Unfinished 集合中;
41.            LaveExecutionTime_i=LaveExecutionTime_i–(ExecutionPointSet[1]–ExecutionPointSet[0]);
42.              //更新该任务的剩余执行时间
43.            Remove(ExecutionPointSet[0];
44.            State(T_i)=Ready;
45.            CurPt←ExecutionPointSet[0];}
46.        end if;
47.      While (执行断点序列 ExecutionPointSet 不空)
48.      print preemptionSequence   //输出任务集在一个超周期内的抢占序列
```

在算法 2 中,如果某个任务需要获得处理器资源时,其为 Ready 状态,并处于就绪队列中;若该任务的优先级在就绪队列中最高,则获得处理器处理时间.在任务执行断点处表示会有新的任务到来,如果新任务加入处于 Ready 状态的任务集中,此时,调度器选择任务集中优先级最高的任务执行,并将其状态改变为 Running 状态.当执行的任务被抢占时,任务状态切换为 Ready 状态并加入就绪队列.当正在执行的任务完成执行或者发生阻塞后,该任务进入 Awaiting 状态,因此,任务在一个周期内完成执行就进入 Awaiting 状态,当任务在下一个周期被释

放时就进入 Ready 状态.

2.4 基于抢占序列的CRPD计算方法

AADL 架构中,线程构件是一种基本可执行任务单元,然而,建立 AADL 系统模型时并不会直接给出线程模型的代码实现,因此无法依据模型给出任务调度时所需的 *UCBs*(*UCB* 集合)与 *ECBs*(*ECB* 集合).为了能够支持系统 AADL 模型中各线程构件的 CRPD 计算,就需要在建模时刻画出被抢占任务的 *UCBs* 与 *ECBs* 属性,因此我们需要对任务的缓存利用率进行定义.

定义 1. 线程构件缓存利用率(cache utilization(T_i),简称 CU_i)是指线程任务 T_i 运行时占用缓存中缓存块的数量与缓存资源块总数量的比例.任务 T_i 的缓存利用率 CU_i 计算可通过公式(4)得到:

$$CU_i=B_i/C_i \tag{4}$$

其中,

1) B_i 为任务 T_i 所使用内存空间映射到缓存中的块数,其大小是由 AADL 架构模型中线程任务 T_i 的 ROM, RAM 资源使用量来决定;
2) C_i 为缓存中缓存块总数.

这样,一个模型中所有线程的缓存利用率之和就称为系统缓存利用率(cache utilization,简称 *CU*).如果一个系统的 AADL 模型中包含了多个线程构件:$S=\{T_1,T_2,\ldots,T_n\}$,系统的缓存利用率可以利用公式(5)得到:

$$CU=\sum_{i=1}^{n}CU_i \tag{5}$$

当多个线程同时使用一个缓存块时,系统缓存利用率大于 1.

在 AADL 中,存储器构件 Memory 中具有 *Memory_Size* 来描述 Memory 的存储容量,Memory 构件源属性及其描述见表 4.

Table 4 Properties of AADL memory component

表 4 存储资源属性及其含义

属性名	含义
Source_Code_Size	表示程序编译、链接后产生的静态代码和只读程序,在软件构件、系统、处理器、外设构件中声明
Source_Data_Size	表示原文本中的程序编译链接后产生的可读可写的数据,在软件构件、系统、处理器、外设构件中声明
Source_Stack_Size	处理器、外设驱动、线程、子程序所需栈空间的最大容量.
Source_Heap_Size	表示线程和子程序需要堆资源的大小,是一个区间值

存储资源模型中,存储资源类型归类为 RAM,ROM 这两种.AADL 将 Memory 构件属性 *Source_Code_Size* 归为 ROM 类型,将 Memory 构件属性 *Source_Data_Size*,*Source_Stack_Size*,*Source_Heap_Size* 归为 RAM 类型.同时还可以定义 Memory 构件属性 *RomBudget*,*RamBudget*,它们表示构件的 ROM,RAM 资源预估容量,属性 *RomActual*,*RamActual* 用来定义 Memory 构件实际使用的 ROM,RAM 资源容量.因此,任务映射到缓存中的缓存块数 B_i 计算方法如公式(6)所示:

$$B_i=[ROM(T_i)+RAM(T_i)/Line\_Size] \tag{6}$$

其中,(1) $ROM(T_i)$为任务 T_i 的 ROM 资源占有量;(2) $RAM(T_i)$为任务 T_i 的 RAM 资源占有量;(3) *Line_Size* 为缓存块大小,由 AADL 架构模型属性 *Line_Size* 确定.

根据缓存利用率,在缓存中以均匀分布的方式来产生任务的 *ECBs*,*UCBs* 的产生依赖于 *ECBs*,对每个任务的 *UCBs*,可以通过 *ECBs* 乘以一个重用因子ρ得到,因此,*UCBs* 是 *ECBs* 的一个子集,集合的秩满足关系$|UCBs|=\rho\cdot|ECBs|$.ρ用来说明任务在执行时缓存块的重用率.影响重用因子ρ大小的因素有许多,比如程序执行流程存在循环路径时ρ就较大,数据处理类线程的程序局部性没有指令控制类线程的程序局部性明显,因而控制类线程的重用因子相较于数据处理类线程更高.在模型设计时,ρ是一个随机变量,其取值空间在[0,1]范围内,且服从均匀分布[20].

利用算法 2 计算出任务集在一个超周期内的抢占序列,结合任务 T_i 被抢占的时间以及 T_i 的周期大小,就可

得到任务 T_i 实例在一个执行周期内的抢占序列.遍历该序列,就能得到每个抢占任务的抢占缓存块集合 $ECBs_{T_n}$,所有集合 $ECBs_{T_n}$ 的并就是 T_i 实例执行过程中被抢占导致的可能访问缓存块总数,它与被抢占任务 T_i 的有用缓存块集合 $UCBs_{T_i}$ 的交集就是被替换出缓存块的集合.这样就可以得到任务 T_i 在第 k 个子周期(k=1,2,...,m)被任务 n 抢占后,基于第 j 个抢占序列任务的缓存相关抢占代价 $CRPD_{i,k}^{j}$,利用公式(7)计算得到:

$$CRPD_{i,k}^{j} = CacheMissTime \times \left| UCBs_{T_i} \cap \left(\bigcup_{T_n \in preemtingSequence(T_i)} ECBs_{T_n} \right) \right| \tag{7}$$

其中,$preemtingSequence(T_i)$为任务 T_i 从被抢占到恢复执行这一时间段内高优先级任务的执行序列.

公式(7)用来计算在一次抢占序列下的被抢占任务的抢占代价,但是我们需要计算在一个任务周期内发生多次抢占的被抢占任务的抢占代价.因为在被抢占任务的一个任务周期内可能出现多个抢占序列,那么被抢占任务的一个周期内的 $CRPD$ 为所有抢占序列所计算出的 $CRPD$ 值之和.因此,在一个周期内,一个抢占任务 T_i 的 $CRPD_{i,k}$ 可以用公式(8)计算:

$$CRPD_{i,k} = \sum_{j=1}^{N} CRPD_{i,k}^{j} \tag{8}$$

进而,利用公式(9)可以计算任务 T_i 在一个超周期内的最大延迟时间 $CRPD_i$:

$$CRPD_i = \max\{CRPD_{i,1}, CRPD_{i,2}, \ldots, CRPD_{i,k}\} \tag{9}$$

2.5 最坏执行时间计算及任务集可调度性判定

更新任务 T_i 的最坏执行时间为 $WCET_i^{new}$,根据公式(10)可以计算出任务缓存相关抢占延迟约束下的最坏执行时间:

$$WCET_i^{new} = WCET_i^{old} + CRPD_i \tag{10}$$

其中,

1) $WCET_i^{old}$ 为在 AADL 架构模型中 T_i 的执行时间;
2) $CRPD_i$ 为任务 T_i 在一个超周期内的缓存相关抢占最大延迟.

AADL 架构模型中,任务的执行时间具体大小是架构模型中线程构件的 *Execution_Time* 属性确定一个任务执行范围值,表示线程的最好执行时间和最坏执行时间.公式(10)中计算出来的最坏执行时间是基于超周期内任务集调度中产生的抢占序列所计算出的每个任务的 CRPD,但是在不同的调度序列会产生不同的抢占点,修改了任务的 WCET 后,再次执行任务集则会产生不同的调度序列,因此在基于抢占代价分析系统可调度性的时候,要不断地更新 WCET 的值,直到其不再发生变化的时候,说明该任务的 WCET 计算完成.

要对系统的实时性进行分析,则要引入最坏响应时间(worst case response time,简称 WCRT),最坏响应时间是实时调度理论研究的重点问题.可调度性的基本思想就是判断任务的最坏响应时间是否满足截止期约束:如果一个任务的最坏响应时间满足截止期约束,那么该任务就是可调度的.

系统的调度策略直接影响到任务执行的最坏响应时间,这是由于不同的调度策略下任务最坏响应时间的计算方法不同.例如,分区内调度系统采用固定优先级调度策略时,缓存抢占延迟的情况下任务 T_i 的最坏响应时间由任务 T_i 的执行时间 $WCET_i^{new}$ 和高优先级任务对 T_i 的干扰 I 确定,其干扰 I 的计算方法如公式(11)所示:

$$I = \sum_{\forall T_j \in hp(T_i)} n \times WCET_j^{new} \tag{11}$$

其中,

1) $hp(T_i)$为调度任务集中任务的优先级高于任务 T_i 的任务集合;
2) $WCET_j^{new}$ 为任务 T_j 的最坏执行时间;
3) n 为高优先级任务在任务 T_i 的响应时间内所释放的次数的上界,n 大小的计算如公式(12)所示:

$$n = \left\lceil \frac{WCRT_i}{P_j} \right\rceil \tag{12}$$

其中,

1) $WCRT_i$ 为任务 T_i 的最坏响应时间;

2) P_j 为任务 T_j 的周期.

固定优先级抢占式系统中任务的响应时间 $WCRT_i$ 可用公式(13)计算得到:

$$WCRT_i = WCET_i^{new} + \sum_{\forall j \in hp(i)} \left\lceil \frac{WCRT_i}{P_j} \right\rceil \times WCET_j^{new} \tag{13}$$

为了判断任务集的可调度性,需要判断在一个超周期内,任务集中任务在每一个周期的响应时间是否超过截止期.所以在 AADL 线程构件模型中添加剩余执行时间属性,当下一个任务执行断点来临并且下一个任务执行断点处释放的是当前正在调度的任务,根据任务当前的剩余执行时间,判断该任务在下一次执行断点来临时是否能够执行完成.如果当前正在执行任务的剩余执行时间小于下一个执行断点与当前时间点的差,则表示任务能够完成执行;否则,表示该任务在下一个任务释放来临前未执行完毕,任务集不可调度.

AADL 架构任务集的可调度性判定可以通过计算每一个任务基于抢占序列的最坏响应时间,并分别比较每一个构件任务的最坏响应时间与构件截止期的大小来进行判断.即,系统 AADL 架构是可调度的当且仅当架构中任何一个构件任务的最坏响应时间都小于其截止期.任务 WCET 的计算如算法 3 所示.算法 3 中调用了算法 2,并对每个任务的 WCET 求解,任务集中任务的数量 n,执行断点序列中断点数量为 k,则 n 与 k 之间满足关系:$n \leqslant k$.所以算法 3 的时间复杂度为 $O(k)$.

算法 3. 每个任务 *WCET* 的计算.

Input:AADL 的 XML 实例化文件;

Output:各个任务的 CRPD 约束下的 WCET 以及任务集可调度性.

1. 解析 AADL 调度模型文件,提取任务调度属性; //提取中所需相关属性
2. 初始化线程任务:*CRPD*=0;
3. 根据公式(6)计算线程使用的缓存块数;
4. 根据公式(4)计算缓存利用率;
5. 求解抢占缓存块集合 *ECBs*; //假设缓存利用率 CU_{T_i} 离散分布随机过程为均匀分布;
6. 计算|*UCBs*|=ρ·|*ECBs*|;
7. 调用算法 1 求解执行断点集 *ExecutionPointSet*;
8. 调用算法 2 计算任务集在一个超周期内的抢占序列 *preemptionSequence*;
9. **if** (任务集可调度)
   {**for** (对任务集中的所有任务)
10. {根据公式(7)计算一次抢占序列下任务的缓存相关抢占代价 $CRPD_{i,k}^{j}$;
11. 根据公式(8)计算任务在一个周期内的 $CRPD_{i,k}$;
12. 根据公式(9)计算任务在一个超周期内的 $CRPD_i$;
13. 根据公式(10)更新任务的 WCET;}
14. 输出任务集的可调度序列及其 WCET;} //在算法 2 的第 11 行~第 18 行中得到调度的结果
15. end if

3 工具实现与案例分析

基于前面建模与分析理论,采用 Eclipse 插件开发技术,开发了 AADL 模型的可调度性分析工具原型系统,系统架构如图 6 所示.该工具支持 ARINC653 分区操作系统平台上的多分区、多任务信息系统的 AADL 架构模型建模、存储资源建模和模型可调度性分析.

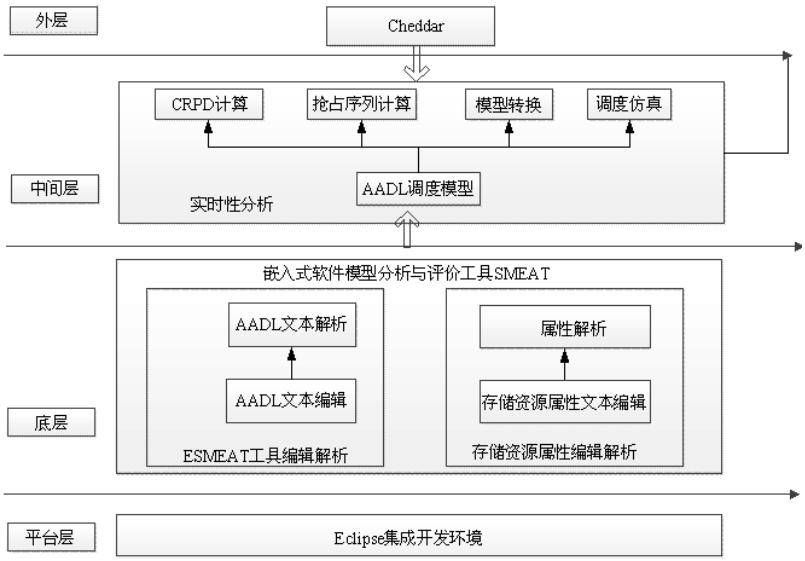


Fig.6 Architecture of schedulbility analyzation software toolkit

图 6 可调度性分析工具架构图

3.1 智能信息系统的架构建模

为了验证本文理论成果的正确性和原型工具的实用性,我们以航空智能信息系统为例开展实验.航空智能信息系统由 4 个计算节点,通过双环高速光纤网络连接形成高可靠航空电子系统,其中一个计算节点 *GPM_A* 子系统的 AADL 架构模型如图 7 所示.

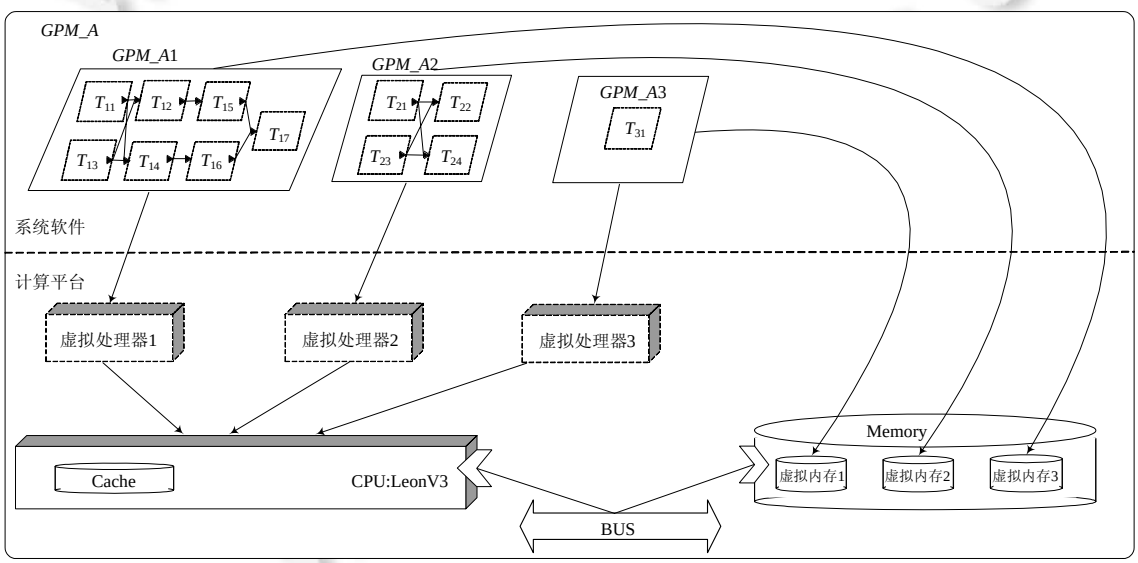


Fig.7 AADL architecture model of a single processor system

图 7 单处理器系统的 AADL 架构模型

表 5 列出了 *GPM_A* 的分区信息和各分区内的任务集合.各分区调度相关属性见表 6,各分区内线程调度属性值见表 7.其中,*GPM_A*1 包含了 T_{11}~T_{17} 线程构件,其绑定在虚拟处理器 virtual processor 上,它与其他两个分区共享处理器构件 LeonV3 和 memory 构件.

Table 5 Components inside the three partitions of GPM_A

表 5 GPM_A 构件含义

分区名	线程	线程含义
*GPM_A*1	T_{11}	数据采集任务
	T_{12}	驾驶员命令处理任务
	T_{13}	故障告警任务
	T_{14}	数据输出任务
	T_{15}	飞行参数处理任务 1
	T_{16}	状态机任务 1
	T_{17}	错误处理任务 1
	T_{21}	飞行参数记录任务 2
*GPM_A*2	T_{22}	错误处理任务 2
	T_{23}	消息派发任务
	T_{24}	状态机任务 2
*GPM_A*3	T_{31}	连续 BIT 任务

Table 6 Length of time period and intra-partition scheduling strategy of each GPM_A partitions

表 6 GPM_A 节点各分区分配时间片长度及内部调度策略

分区名	分区时间片	分区内调度策略
*GPM_A*1	8ms	RMS
*GPM_A*2	7ms	EDF
*GPM_A*3	3ms	RMS

Table 7 Properties of tasks related to scheduling within each partition of GPM_A

表 7 GPM_A 节点各分区内任务与调度相关的任务属性

任务名	所属分区	存储资源使用量(KB)	执行时间(ms)	周期(ms)	截止期(ms)
T_{11}	*GPM_A*1	400	3	15	15
T_{12}	*GPM_A*1	440	5	22	22
T_{13}	*GPM_A*1	300	5	60	60
T_{14}	*GPM_A*1	240	3	60	60
T_{15}	*GPM_A*1	330	4	75	75
T_{16}	*GPM_A*1	320	4	80	80
T_{17}	*GPM_A*1	260	5	100	100
T_{21}	*GPM_A*2	280	3	40	40
T_{22}	*GPM_A*2	300	4	80	80
T_{23}	*GPM_A*2	200	3	200	200
T_{24}	*GPM_A*2	330	3	100	100
T_{31}	*GPM_A*3	14	3	80	80

3.2 分区内调度实验分析

在航空智能信息系统的 *GPM_A* 模型中,其中,分区 *GPM_A*1 的 Cache 容量为 200KB,Cache 块容量为 32bytes,处理器 Leon 的主频为 250MHz,一次 Cache Miss 耗时 40ns.实验采用 RMS 调度策略,分区 *GPM_A*1 内任务集的超周期为 13200ms,所有任务在 13188ms 的时候全部执行完成,且满足每个任务的截止期约束,因此,子系统分区 *GPM_A*1 在缓存相关抢占延迟约束下是可调度的.如图 8 所示,系统调度执行迭代 20 次后,线程 T_{11} 的最坏执行时间对比没差别.这是由于 T_{11} 优先级最高,抢占代价为 0.然而,任务 T_{16} 与 T_{17} 的缓存相关抢占代价差距明显,因为它们在一个超周期内被抢占次数较多,缓存相关抢占延迟较高.

同理,图 9 给出了基于抢占序列调度算法与 UCB-Union,ECB-Union 方法调度延迟 CRPD 分析结果对比:基于抢占序列计算的 CRPD 比 UCB-Union 与 ECB-Union 更准确.实验结果表明:基于抢占序列调度方法相较于 UCB-Union 方法 CRPD 最大下降 73μs,相较于 ECB-Union 方法 CRPD 值最大下降为 23μs.当采用分区内调度方式时,从总体上来看,任务的缓存相关抢占代价基本与任务的优先级呈负相关.

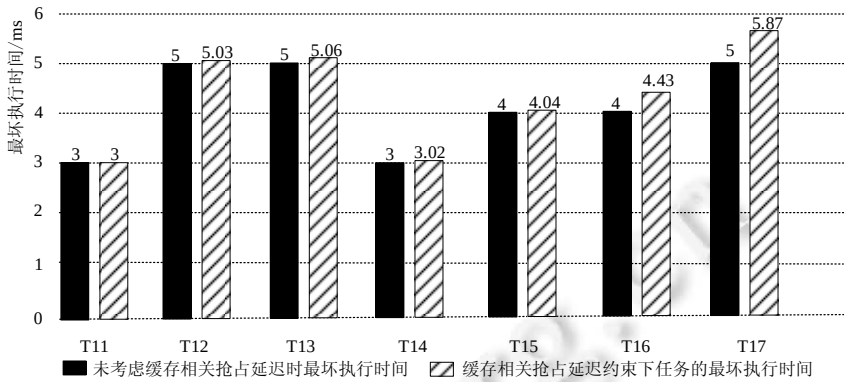


Fig.8 Comparation of WCET calculated with or without CRPD of tasks within partition GPM_A1

图 8 GPM_A1 分区内各任务在是/否考虑 CRPD 约束情况下 WCET 计算结果对比图

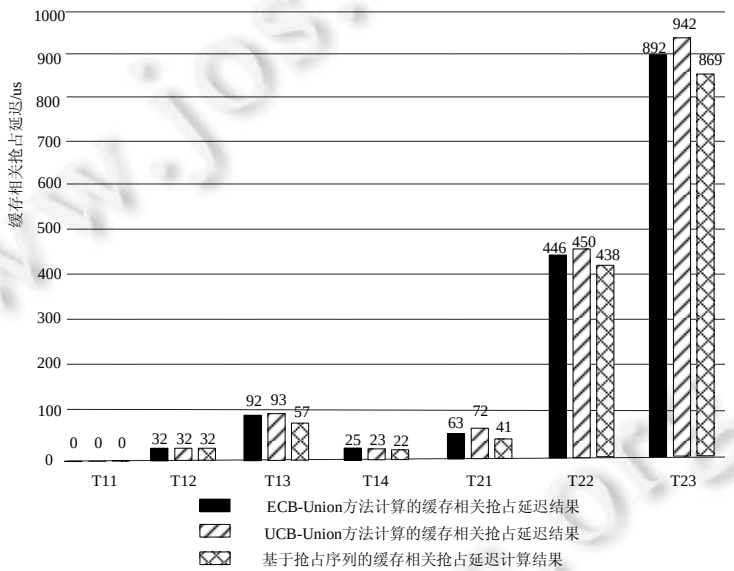


Fig.9 Comparation of CRPD calculated with different algorithms

图 9 分区内调度模型的 CRPD 计算结果与传统计算方法结果对比柱状图

3.3 系统分区调度实验分析

航空智能信息系统 *GPM_A* 采用分区调度时,Cache 大小为 500KB,Cache 块大小为 32bytes,处理器其主频为 250MHz,一次 Cache Miss 耗时 40ns.分区任务调度信息见表 7,任务的超周期为 13 200ms.当任务按着执行序列执行到 60ms 时,任务 T_{14} 被任务 T_{11} 发生抢占,但是任务 T_{14} 的截止期已到,任务 T_{14} 不可调度,从而使得该任务集不可调度.具体的调度序列为 *Sequence*:$T_{11}\rightarrow T_{12}\rightarrow T_{21}\rightarrow T_{22}\rightarrow T_{31}\rightarrow T_{12}\rightarrow T_{11}\rightarrow T_{12}\rightarrow T_{13}\rightarrow T_{22}\rightarrow T_{24}\rightarrow T_{12}\rightarrow T_{11}\rightarrow T_{12}\rightarrow T_{23}\rightarrow T_{12}\rightarrow T_{12}\rightarrow T_{11}\rightarrow T_{21}\rightarrow T_{12}\rightarrow T_{14}$(任务 T_{14} 截止期前未能完成执行).

由于 *GPM_A*1 分区内的任务周期过短,采用时间片轮转调度时,导致 *GPM_A*1 分区内短周期任务不断抢占其他任务,导致长周期任务超过截止期而不可调度.在系统架构设计过程中,我们需要对系统模型进行优化设计,调整 *GPM_A*1 分区配置及其任务调度属性,调整后的系统调度属性见表 8.

Table 8 Adjusting of task scheduling properties in every partition of GPM_A, when taking account of CPRD

表 8 考虑 CPRD 情况下 GPM_A 各分区内任务调度属性的调整结果

任务名	所属分区	存储资源使用量(KB)	执行时间(ms)	周期(ms)	截止期(ms)
T_{11}	*GPM_A*1	400	3	40	40
T_{12}	*GPM_A*1	440	5	80	80
T_{13}	*GPM_A*1	300	5	80	80
T_{14}	*GPM_A*1	240	3	100	100
T_{15}	*GPM_A*1	330	4	100	100
T_{16}	*GPM_A*1	320	4	80	80
T_{17}	*GPM_A*1	260	5	100	100
T_{21}	*GPM_A*2	280	3	40	40
T_{22}	*GPM_A*2	300	4	80	80
T_{23}	*GPM_A*2	200	3	200	200
T_{24}	*GPM_A*2	330	3	100	100
T_{31}	*GPM_A*3	14	3	80	80

优化系统的 3 个分区任务的超周期为 400ms,并且每个分区采用不同调度策略,所有任务在 381ms 的时候全部执行完成,且满足每个任务的截止期约束.因此,在缓存相关抢占延迟约束下是可调度的.优化系统各分区调度策略相同,是否考虑了 CRPD 约束下,系统的最坏执行时间对比如图 10 所示.

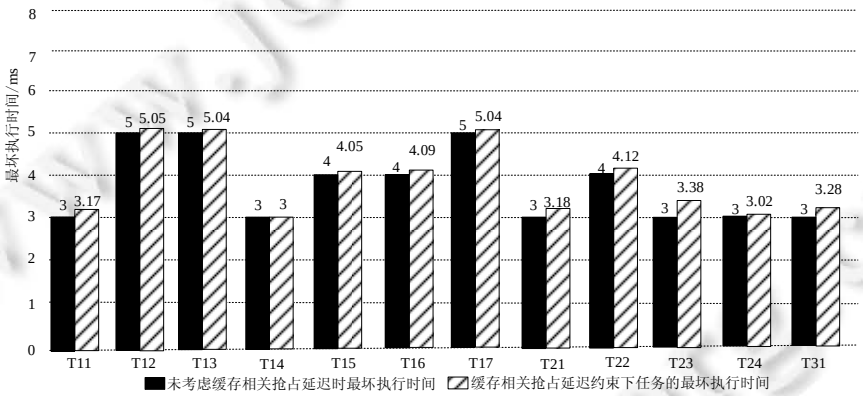


Fig.10 WCET calculated with or without CRPD of all tasks inside each partition of GPM_A

图 10 GPM_A 系统模型 CRPD 约束的最坏执行时间对比图

系统架构中,Cache 的容量影响系统调度过程中 Cache Miss 的发生频率,对系统可调度性也是有影响的.当 Cache Miss 时间不变,仅考虑 Cache 容量变化,系统任务的缓存相关抢占延迟见表 9,每个任务缓存相关抢占延迟随 Cache 大小变化的折线图如图 11 所示.随着 Cache 容量的增大,CRPD 的值减小.

Table 9 CRPD calculated with different size of Cache equipped for each task

表 9 不同 Cache 大小下任务计算的 CRPD 值

Cache 大小 / 任务名	500KB	750KB	1MB	1.25MB	1.5MB	1.75MB	2MB
T_{11}	156μs	118μs	94μs	54μs	48μs	48μs	35μs
T_{12}	51μs	53μs	38μs	32μs	22s	24μs	17μs
T_{13}	43μs	34μs	27μs	21μs	13μs	14μs	12μs
T_{14}	6μs	6μs	4μs	5μs	2μs	2μs	1μs
T_{15}	51μs	37μs	42μs	33μs	27μs	25μs	25μs
T_{16}	85μs	54μs	34μs	32μs	19μs	18μs	18μs
T_{17}	41μs	45μs	31μs	28μs	22μs	22μs	20μs
T_{21}	183μs	150μs	87μs	52μs	43μs	44μs	30μs
T_{22}	121μs	84μs	51μs	28μs	21μs	24μs	26μs
T_{23}	383μs	314μs	256μs	231μs	161μs	160μs	153μs
T_{24}	24μs	19μs	18μs	13μs	4μs	4μs	3μs
T_{31}	281μs	134μs	89μs	49μs	42μs	43μs	32μs

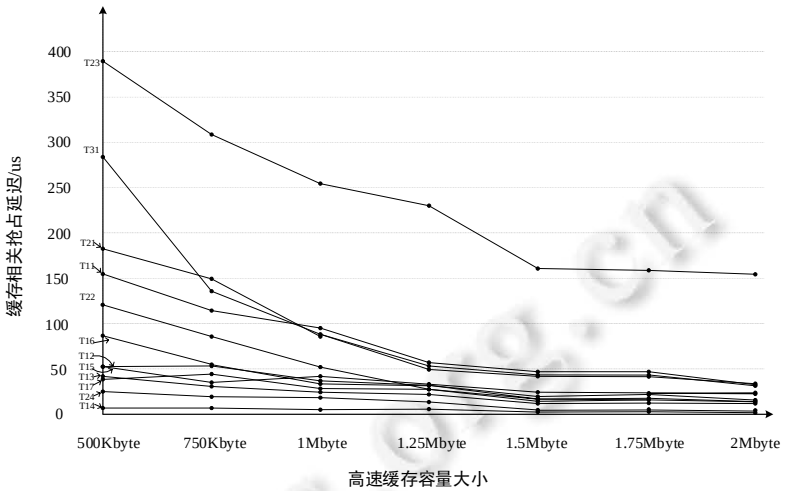


Fig.11 Line chart shows the impact of cache size on CRPD of tasks

图 11 Cache 大小对 CRPD 估计的影响

4 总结与展望

本文面向复杂嵌入式系统,研究在缓存相关抢占延迟约束下 AADL 模型的可调度性,采用属性扩展的方式对 AADL 属性进行扩展,使其能够对 Cache 进行建模,在此基础上,提出了 CRPD 约束下 AADL 模型可调度性分析方法,通过计算任务集在特定调度策略下的抢占序列,进而计算出被抢占任务的 CRPD.该方法从系统架构层面考虑了任务之间的抢占行为对系统可调度性的影响,并且具有良好的通用性以及有效性,保证了缓存相关抢占延迟约束的最坏执行时间计算值的安全性.本文设计并实现了面向 AADL 存储资源约束的可调度性分析工具原型,并结合航空智能信息系统开展实验分析,通过实验可知:基于抢占序列的计算方法比 UCB-Union,ECB-Union 方法得到的 CRPD 结果更加精确;当系统的任务集不可调度时,通过分析不可调度原因并给出合理优化意见,使得优化后的系统变得可调度;实验最后针对 Cache 容量大小对任务 CRPD 的影响进行评估,供系统设计人员参考使用.

基于抢占序列的 CRPD 计算方法是通过获取特定调度算法下的抢占序列,进而对任务之间抢占所产生的 CRPD 进行计算,并综合考虑加入最坏执行时间来分析系统任务抢占发生时系统调度行为.但是影响任务可调度性的因素较多,如不同调度策略下计算任务的抢占序列方法的差异性、模型中任务间同步关系等都会对系统调度性产生影响.因此,如何考虑到调度策略以及同步关系两个因素得出抢占序列,仍是亟待解决的问题.

References:

[1] Liu YF, Zhang LC. Worst-Case execution time analysis for real-time systems. Application Research of Computers, 2005,22(11): 16–18 (in Chinese with English abstract).

[2] Zhou GC, Guo BL, Gao X, *et al*. Fast estimation of WCET based on distribution function. Computer Science, 2016,43:157–161 (in Chinese with English abstract).

[3] SAE. Architecture analysis and design language (AADL) AS-5506A. The Engineering Society for Advancing Mobility and Sea Air and Space, Aerospace lnfonnation Report. Version 2.0. Tech. Rep. 2009.

[4] Yang ZB, Pi L, Hu K, Gu ZH, Ma DF. AADL: An architecture design and analysis language for complex embedded real-time systems. Ruan Jian Xue Bao/Journal of Software, 2010,21(5):899–915 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3700.htm

[5] Zhang J. Research on the earliest deadline priority real-time scheduling algorithm [Ph.D. Thesis]. Wuhan: Huazhong University of Science and Technology, 2009 (in Chinese with English abstract).

[6] Clarke D, Lee I, Xie H L. VERSA: A tool for the specification and analysis of resource-bound real-time systems. Nonlinear Dynamics, 2016,87(1):1–7.

[7] Clavel M, *et al*. Maude Manuel. Version 2.1. Menlo Park: SRI Int'l, 2004. http://maude.cs.uiuc.edu

[8] Jerad C, Barkaoui K, Grissa-Touzi A. On the use of real-time Maude for architecture description and verification: A case study. In: Proc. of the Visions of Computer Science-bcs Int'l Academic Conf. DBLP, 2015.

[9] Hu K, Duan ZB, Wang JY, Ga LC, Shang LH. Template-Based AADL automatic code generation. Frontiers of Computer Science in China, 2019,13(4):698−714.

[10] Singhoff F, Legrand J, Nana L, *et al*. Scheduling and memory requirements analysis with AADL. In: Proc. of the ACM Sigada Int'l Conf. on Ada: The Engineering of Correct & Reliable Software for Real-time & Distributed Systems Using Ada & Related Technologies. ACM, 2005. 1−10.

[11] Zhen-Song LI. Research on verification method of AADL behavior model based on UPPAAL. Computer Science, 2012.

[12] Ma BZ. The worst response time analysis considering the impact of cache replacement [Ph.D. Thesis]. Changsha: Hu'nan University, 2013 (in Chinese with English abstract).

[13] Tran HN, Singhoff F. Instruction cache in hard real-time systems: Modeling and integration in scheduling analysistools with AADL. In: Proc. of the Embedded and Ubiquitous Computing. IEEE, 2014. 104-111.

[14] Lee CG, Hahn H, Seo YM. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. In: Proc. of the IEEE Computer Society. 1998. 700−713.

[15] Delange J, Pautet L, Plantec A, *et al*. Validate, simulate, and implement ARINC653 systems using the AADL. ACM SIGAda Ada Letters, 2009,29(3):31−44.

[16] Dimpsey RT, Iyer RK. Modeling and measuring multiprogramming and system overheads on a shared-memory multiprocessor: Case study. Journal of Parallel and Distributed Computing, 1991,12(4):402−414.

[17] Tran HN, Singhoff F, Rubini S. Cache-Aware real-time scheduling simulator: Implementation and return of experience. ACM SIGBED Review, 2016,13(1):22−28.

[18] Li C, Ding C, Shen K. Quantifying the cost of context switch. In: Proc. of the Workshop on Experimental Computer Science, Part of ACM FCRC. San Diego: ACM, 2007. 1−4.

[19] Bastoni A, Brandenburg B, Anderson JH. Is semi-partitioned scheduling practical. In: Proc. of the Euromicro Conf. on Real-time Systems. IEEE Computer Society, 2011. 1−11.

[20] Bini E, Buttazzo GC. Measuring the performance of schedulability tests. Real-time Systems, 2005,30(1-2):129−154.

附中文参考文献:

[1] 刘育芳,张立臣.实时系统最坏执行时间分析.计算机应用研究,2005,22(11):16−18.

[2] 周国昌,郭宝龙, 高翔等. 基于分布函数的 WCET 快速估计.计算机科学,2016,43:157−161.

[4] 杨志斌,皮磊,胡凯,等.复杂嵌入式实时系统体系结构设计与分析语言:AADL.软件学报,2010,21(5):899−915. http://www.jos.org.cn/1000-9825/3700.htm [doi: 10.3724/SP.J.1001.2010.03700]

[5] 张杰.最早截止期优先实时调度算法研究[博士学位论文].武汉:华中科技大学,2009.

[12] 马炳周.考虑缓存替换影响的最坏响应时间分析研究[博士学位论文].长沙:湖南大学,2013.

陆寅(1975－),男,博士,讲师,CCF 专业会员,主要研究领域为嵌入式系统架构,嵌入式系统可靠性工程.

习乐琪(1996－),男,硕士,主要研究领域为嵌入式系统资源调度,嵌入式可信属性分析.

秦树东(1995－),男,硕士,CCF 学生会员,主要研究领域为嵌入式系统分功能属性分析,嵌入式系统可靠性工程.

董云卫(1968－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为嵌入式软件设计与验证,信息物理融合,系统建模与分析.