软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(2):179−199 [doi: 10.13328/j.cnki.jos.004529]

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

面向 WCET 估计的 Cache 分析研究综述*

吕鸣松, 关 楠, 王 义

(东北大学 信息科学与工程学院,辽宁 沈阳 110819)

通讯作者: 吕鸣松, E-mail: lvmingsong@ise.neu.edu.cn

摘 要: 实时系统时间分析的首要任务是估计程序的最坏情况执行时间(worst-case execution time,简称 WCET).程序的 WCET 通常受到硬件体系结构的影响,Cache 则是其中最为突出的因素之一.对面向 WCET 计算的 Cache 分析研究进行了综述,介绍了经典 Cache 分析框架与 Cache 分析核心技术,并从循环结构分析、数据 Cache 分析、多级 Cache 分析、多核共享 Cache 分析、非 LRU 替换策略分析等角度介绍了 Cache 分析在不同维度上的研究问题与主要挑战,总结了现有技术的优缺点,展望了 Cache 分析研究的未来发展方向.

关键词: 实时系统;WCET(worst-case execution time);Cache 分析;时间分析;抽象解释

中图法分类号: TP316 **文献标识码**: A

中文引用格式: 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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/4529.htm

Survey of Cache Analysis for Worst-Case Execution Time Estimation

LÜ Ming-Song, GUAN Nan, WANG Yi

(School of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

Corresponding author: LÜ Ming-Song, E-mail: lvmingsong@ise.neu.edu.cn

Abstract: The main task of real-time system design is to analyze the timing behaviors of a system at design time in order to guarantee that the given timing constraints are met at run time. The key issue is to estimate the Worst-Case Execution Time (WCET) of a program. Typically the WCET is heavily influenced by the hardware features of the target processor, among which Cache is the most influential factor. This article presents a survey on Cache analysis for WCET estimation. It introduces main research problems and challenges in different dimensions, such as the analysis of loops, data caches, multi-level caches, multi-core shared caches, non-LRU replacement policies, etc. The mainstream analysis techniques with their pros and cons are evaluated. An outlook for future research directions of Cache analysis is given in the end.

Key words: real-time system; WCET (worst-case execution time); cache analysis; timing analysis; abstract interpretation

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.大多数嵌入式系统都存在于安全关键的(safe critical)环境中,系统的正确性不仅取决于运算的逻辑正确性,更取决于系统的时间行为是否满足规定的时间约束.这类系统称为实时系统[1].在硬实时系统中,所有任务必须在规定的时间内完成,否则将引发灾难性后果.例如,在火箭发射过程中,点火任务必须在发射窗口内完成,否则可能导致箭体爆炸等不可挽回的后果.因此,实时系统的核心任务就是在设计阶段对系统的时间行为进行分析,以确保在运行过程中系统的时间约束能够得到满足.

一个实时系统(软件部分)通常由一系列的实时任务组成,这些任务共同完成预定的系统功能(在本文的讨

* 基金项目: 国家自然科学基金(61100023, 61300022); 中央高校基本科研业务费(N120404008)
收稿时间: 2013-05-07; 定稿时间: 2013-09-29

论中,不区分任务和程序这两个概念,它们具有相同的含义).由于任务的执行受到程序逻辑、体系结构特性、数据输入、处理器状态等复杂因素的影响,其执行时间通常呈现某种分布,如图 1 所示.在所有可能的执行时间中,存在客观上的最小值和最大值,分别称为最好情况执行时间(best-case execution time,简称 BCET)和最坏情况执行时间(worst-case execution time,简称 WCET).实时系统时间行为分析的首要任务就是估计任务的最坏情况执行时间[2].

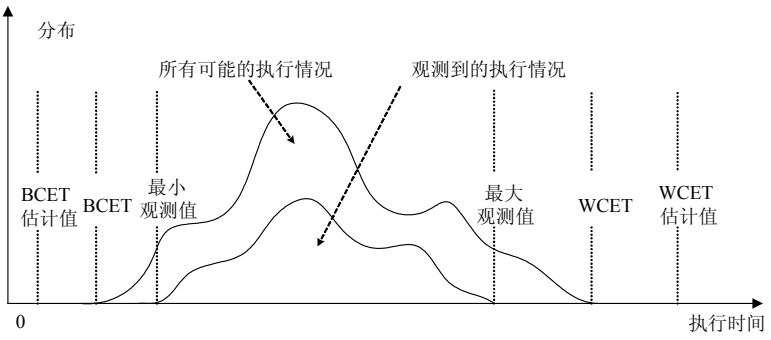


Fig.1 Basic concepts on the execution times of tasks

图 1 有关任务执行时间的基本概念

工业界通常的方法是,测量任务端到端的执行时间[3].为尽可能覆盖更多的执行情况,一般需要多次运行任务并记录执行时间.通过这种方法获得的最小值和最大值分别称为执行时间的最小观测值和最大观测值.该方法又称为动态时间分析(dynamic timing analysis),得到的最大观测值被当做任务的 WCET 使用.

动态时间分析存在一个根本问题:一般情况下,多次执行一个任务无法保证一定覆盖最坏执行情况,因此,测量得到的最大观测值通常小于任务客观上的 WCET.在实时系统中,称这样的分析结果是不安全的,或者是对客观 WCET 的低估(underestimation).如果在系统的后续分析(如可调度性分析)中使用了不安全的执行时间估计值,将可能导致时间分析的最终结果是错误的.

为了避免上述问题,就需要这样一种分析方法:它在不实际执行任务的前提下,确保分析能够覆盖任务所有可能的执行情况,包括所有可能的程序执行路径、程序输入以及处理器初始状态等.这种方法称为静态时间分析(static timing analysis).静态分析得到的最大执行时间称为 WCET 估计值,如图 1 所示.

在一般意义上,从所有可能的任务执行情况中找到执行时间最大者,是一个具有很高复杂度的问题.因此在实际分析过程中需要对任务的行为进行一定程度的抽象,从而提高分析效率.但是抽象会损失信息,进而导致过度估计(overestimation),即,分析得到的最大执行时间大于程序客观上的 WCET.显然,WCET 估计值越接近任务客观上的 WCET,分析结果就越精确.不同的静态分析技术通常是对分析效率和分析精确性的不同折中.好的分析技术能够以最小分析效率的牺牲,获得最大分析精确性的提高.

分析任务的 WCET,必须考虑任务在具体硬件体系结构下的行为及产生的时间延迟.在硬件体系结构中,Cache 对程序的执行时间影响最为显著[4].在大多数处理器中,访存如果在 Cache 中命中,则时间延迟通常为几个时钟周期;如果不命中,则需要从主存载入所需数据,时间延迟通常为 100~200 个时钟周期.因此,Cache 分析的精确性对任务的执行时间具有决定性的影响.Cache 分析是实时系统时间分析领域持续的研究热点.

面向 WCET 计算的 Cache 分析研究已经开展了 20 余年,并形成了相当数量的研究成果以及代表性的分析技术.本文将对现有 Cache 分析技术进行综述,介绍 Cache 分析在不同维度上的研究问题及其挑战,总结现有主流技术及其优缺点,并展望 Cache 分析的发展方向.WCET 分析是一个覆盖面很广的研究领域.文献[2]对 WCET 分析研究进行了综述,但是对于 Cache 分析这一核心领域没有进行深入、系统的阐述.本综述将弥补这一空白,为相关领域研究者进行 Cache 分析研究提供参考.

本文第 1 节介绍背景知识,包括 Cache 的工作原理和主流的 WCET 分析框架.第 2 节综述 Cache 分析的核心技术,分析各种不同技术的优缺点.第 3 节从循环结构分析、数据 Cache 分析、多级 Cache 分析、多核共享 Cache 分析、非 LRU 替换策略分析等多个不同维度介绍 Cache 分析问题的外延,并综述相关分析技术.第 4 节总结全文,并展望未来的研究方向.

1 背景知识

本节首先介绍 Cache 这一硬件部件的基本工作原理与特点,然后介绍实时系统 WCET 分析的基本框架.

1.1 Cache工作原理

Cache 是处理器片内的一个高速、小容量的存储器件.它在系统中的位置如图 2 所示.程序在执行过程中会按照控制流程依次执行一系列指令,并操作相应的数据,指令和数据需要从存储器中载入.在此过程中,处理器首先在 Cache 中查找所需要的数据是否存在:如果存在,则从 Cache 中取出数据送至 CPU;否则,将通过内存总线从主存中载入数据.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,因而,Cache 能够显著提高程序的执行性能.

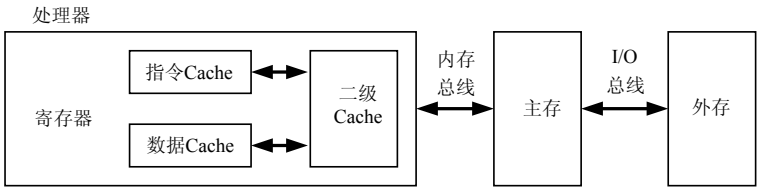


Fig.2 A common memory architecture

图 2 常见存储体系结构

多数处理器通常配置两级 Cache:第 1 级 Cache(又称 L1 Cache)又被分为独立的指令 Cache 和数据 Cache,分别用于缓存指令和数据.L1 Cache 的访问延迟通常为 1~3 个时钟周期,容量大约为 16KB~32KB.如果访问 L1 Cache 不命中,则需要到二级 Cache(L2 Cache)中查找所需数据.L2 Cache 的容量通常在几百 KB 到几 MB 不等,访问延迟通常在 10 个时钟周期左右.在一些高性能多核处理器中,通常还配备 L3 Cache,供所有处理核心共享.如果数据或指令在末级 Cache 中不命中,则需要从主存载入所需内容.由于这一过程需要通过内存总线,故访问延迟非常高,大约在 100~200 个时钟周期.

向 Cache 载入数据的最小单位称为内存块,大小一般为 8~64 字节.目前,大多数实际处理器采用组相联(set-associative)的方式组织 Cache 地址空间.在组相联 Cache 中采用二维编址:地址空间首先被划分为若干个组(set),每个组内部包含若干路(way),每路存放 1 个内存块.每组中路的数量称为相联度(associativity).将一个内存块载入 Cache 时,首先根据地址确定唯一对应的组,如果组内空间不足,则根据某种替换策略(replacement policy)选择一个内存块进行替换.替换策略对程序的时间行为会产生显著影响.本节将通过研究工作中最常使用的 LRU (least recently used)替换策略介绍替换策略的一般工作原理.在后面的第 3.5 节,将专门讨论对不同 Cache 替换策略的分析.由于组相联 Cache 中的各个组是相互独立的,因此后文仅针对一个 Cache 组讨论.

LRU 替换策略会为 Cache 组中的各路标记年龄.图 3 表示了一个相联度为 4 的 Cache 组,元素所在位置即为其年龄,其中,最上面的位置年龄最小(年龄为 1).如果访问内存块 x 在 Cache 中不命中,那么 Cache 组中年龄最大的元素 d 被替换出去,其他元素的年龄依次增加 1(向 Cache 组底端移动一个位置);如果访问 x 命中,那么比 x 年轻的所有元素的年龄都增加 1,而比 x 年龄大的元素位置不变.无论哪种情况,访问 x 之后,x 元素都将被放在 Cache 组中年龄最小的位置.在下文讨论中,用小写字母表示内存块,用大写字母 A 表示 Cache 的相联度,称 Cache 为 A 路组相联.

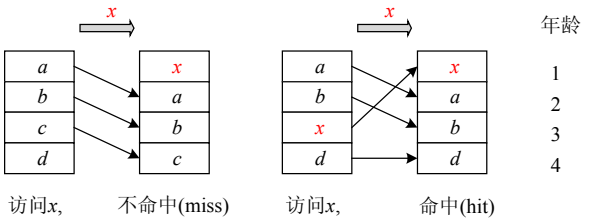


Fig.3 Principle of LRU replacement policy

图 3 LRU 替换策略工作原理

1.2 经典静态WCET分析框架

静态 WCET 分析的目标是:在不实际执行程序的情况下,通过数学手段分析得到程序的 WCET 估计值.本质上,程序的 WCET 对应于“在特定的初始状态和数据输入下,程序按照某条路径的一次执行情况”.如果程序的执行不被打断,所有可能的执行情况的集合 S 可粗略地表示为 $S=P\times D\times I$.其中,P 表示所有可能的程序执行路径的集合,D 表示所有可能的数据输入的集合,I 表示程序启动前所有可能的初始状态的集合.

一种直接的分析思路是,穷举程序所有可能的执行情况.由于程序大多包含循环结构,循环体内部通常又包含分支结构,故程序路径的数量随循环内部分支数量和循环次数呈指数关系.穷举所有执行情况的方法的可扩展性(scalability)必定很差.所以,多数静态分析方法并未采用上述思路,而采用如图 4 所示的分析框架.

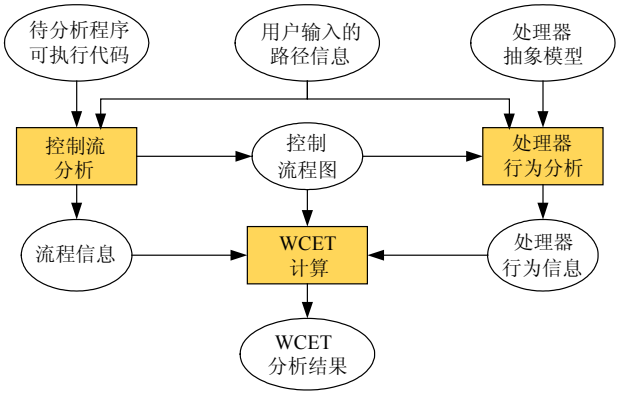


Fig.4 Work flow of static WCET analysis

图 4 静态 WCET 分析流程

首先对程序的可执行代码进行控制流分析.在对可执行代码进行反编译后,抽取出程序的控制流程图(control flow graph,简称 CFG).程序的 CFG 是一个有向带环图:图的节点为每一条指令**,图的边表示程序的控制流程.程序的 CFG 有唯一的起点和终点,分别表示程序的入口和出口.后续分析步骤都针对程序的 CFG 进行.

然后进行处理器行为分析.指令的执行时间受到流水线[5]、分支预测器[6]、Cache 等处理器部件的影响,呈现较大波动.处理器行为分析的目标就是在给定的处理器特性参数下,为 CFG 中的每条指令估计一个 WCET 的上限值.

最后一步工作就是找到导致整个程序执行时间最长的路径,称为 WCET 计算.目前,研究领域最常用的技术是隐式路径枚举(implicit path enumeration)[7].该技术的基本思想是:将 WCET 求解问题(或最长执行时间路径搜

** 执行指令引发对 Cache 的访问:首先,指令本身造成对指令 Cache 的访问;其次,执行指令会操作数据,造成对数据 Cache 的访问.通常,内存块的大小是指令大小的整数倍,所以程序中普遍存在多条指令访问同一内存块的情况.下文中,我们称指令执行引发了访存.

索问题)转化为求每条指令执行次数的问题,并将其建模成一个整数线性规划问题进行求解.该问题的目标函数如公式(1)所示:

$$WCET = \mathrm{MAX}\sum_{i=1}^{N} C_i \cdot X_i \tag{1}$$

其中,N 表示所有指令的个数;C_i 表示通过处理器行为分析得到的第 i 条指令的 WCET 估计值;X_i 是为分析引入的变量,表示第 i 条指令的执行次数.

显然,每条指令的执行次数并不可以任意取值,它们必须满足一系列的约束:

首先,需要满足程序结构的约束.定义 d_{i_j} 为从指令 i 到指令 j 的边的执行次数,那么对于任何一条指令,其执行次数等于所有入口边执行次数之和,也等于所有出口边执行次数之和.这一约束可以用公式(2)表达:

$$\forall i, X_i = \sum_{\text{所有入边}} d_{*\_i} = \sum_{\text{所有出边}} d_{i\_*} \tag{2}$$

在此基础上,还需要建立程序功能约束.例如,对于任何循环体,需要指定该循环体的最大循环次数.这一约束也可以通过不同节点执行次数之间的线性关系进行表达,并合并到上述线性规划问题中.通过求解公式(1)的最大值问题,即可得到程序的 WCET 估计值以及对应的每条指令的执行次数.多条连续执行(中间不存在分支)的指令组成基本块(basic block),多数分析以基本块为单位进行,能够提高分析的效率.

可以看出,图 4 所示的分析思路与穷举所有执行路径的分析思路存在本质区别.在这一框架中,最长路径搜索和程序行为分析这两步是分离的.对于一条多次执行的指令而言,每次执行所耗费的时间客观上都可能是不同的.而上述方法为每条指令计算 WCET 估计值,以此代表该指令每次执行的时间.在此基础上,再寻找导致程序最大执行时间的路径.此时找到的路径可能已经不是程序客观上的最坏情况执行路径.因此,这一分析框架已经对程序的行为进行了抽象,引入了一定的悲观性,但换来了分析效率的显著提高.在实际分析中,这种分析框架被验证是非常有效的,因此在 WCET 分析领域被广泛采用.

无论采用何种分析思路,客观上处理器特性是影响程序执行时间的关键因素.从下一节开始,将聚焦到处理器行为分析中的 Cache 分析,深入探讨 Cache 分析的目的与挑战,并综述 Cache 分析所采用的主要技术.

2 Cache 分析核心技术

给定一种 Cache 体系结构,程序在 Cache 上的行为特点是一种客观事实.无论何种分析技术,其根本目的都是试图挖掘程序的行为特征,并最终获得指令和数据在 Cache 中命中与否的信息.显然,程序在 Cache 中的行为是两个方面因素的结果:(1) 程序自身特性;(2) Cache 体系结构特性.其中,程序自身特性又包含两方面的含义:首先,程序执行不同的路径将会造成不同的访存序列,因而导致在 Cache 中的行为结果不同;其次,要区分指令和数据,它们的访存行为通常大相径庭.Cache 体系结构特性,如不同 Cache 替换策略、不同 Cache 级数、Cache 是否共享等因素,也都会对程序在 Cache 中的行为产生决定性影响.正是由于这些因素的存在,才产生了关于 Cache 分析的不同研究方向和侧重点.

为了达到 Cache 分析的目标,需要对程序的行为和 Cache 的工作原理分别进行建模,根据所建立的数学模型,采用适合的数学手段来分析程序的指令和数据在 Cache 上的访问特性.不同分析方法本质上对系统行为的抽象方法和抽象程度不同,造成的结果是分析效率和分析精确性上的差异.一般情况下,抽象程度越高,分析效率就越高,但是由于信息损失严重,分析的精确性也就越差.一种好的抽象,既能充分保留关键信息,又能有效地削减对分析结果影响不显著的系统状态.

本节将针对“采用 LRU 替换策略的单级指令 Cache”这种特定配置,综述当前研究领域的主流分析技术.该配置在实时系统 WCET 分析中广为采用.当前最具代表性的主流技术——基于抽象解释的 Cache 分析技术,就是在这一体系结构条件下发展起来的.通过这部分综述,将为读者建立 Cache 分析的基本概念与解决问题的典型思路.在第 3 节中,将从循环结构分析、数据 Cache 分析、多级 Cache 分析、多核共享 Cache 分析、非 LRU 替换策略分析等角度,探讨 Cache 分析在不同维度上的新问题和新挑战,并阐述本节将介绍的核心技术是如何向上述问题延伸的.最终建立 Cache 分析这一研究领域的全貌.

2.1 基于模型检测的Cache分析方法

模型检测(model checking)[8]是进行系统分析与验证的典型手段之一.在实时系统中,通常采用时间自动机(timed automata)[9]来建模系统行为,特别是时间行为.吕鸣松等人开发的 WCET 分析工具 McAiT[10]、Dalsgaard 等人开发的 METAMOC 框架[11]以及 Gustavsson 等人提出的多核 WCET 分析方法[12]中,都采用了 UPPAAL[13]模型检测器将系统建模成若干时间自动机,通过验证的手段得到程序的 WCET.

图 5 展示了 McAiT 分析工具的基于模型检测技术的分析流程,包括两个主要功能:首先,根据程序的控制流程图建立程序自动机(program automata)***.程序自动机是对程序行为的完整建模,既包含了程序的执行路径、控制流程信息,也建模了程序每条指令的执行对 Cache 的访问行为;其次,根据给定的 Cache 参数描述建立 Cache 自动机.该自动机完整地描述了 Cache 的工作原理,并对具体 Cache 状态进行建模.程序自动机因执行指令或访问数据发起 Cache 访问,通过 UPPAAL 提供的通道机制,向 Cache 自动机发送消息并传递 Cache 访问参数,Cache 自动机根据这一信息相应地更新 Cache 状态.在 UPPAAL 中,可以通过提取验证过程中时钟变量的上限来得到程序的 WCET 值.

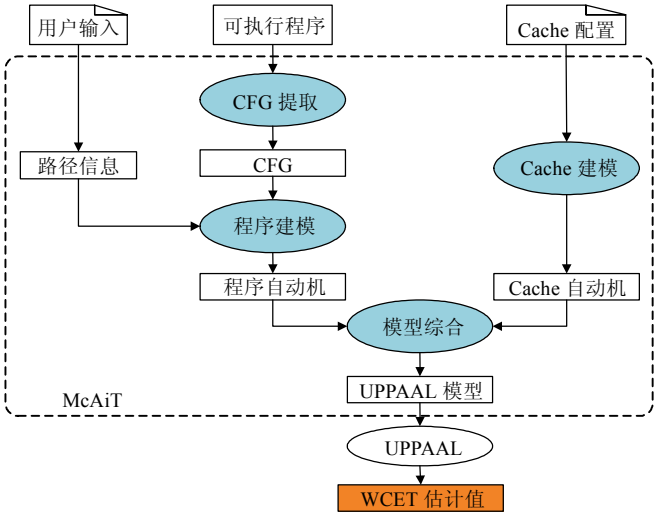


Fig.5 Analysis framework of McAiT

图 5 McAiT 分析框架

基于模型检测技术的 Cache 分析方法,本质上是一种搜索程序的最坏执行情况的手段.如果采用自动机模型对系统行为进行了完全精确的建模,那么最终找到的就是程序的最坏执行情况,得到的也是程序客观的 WCET.这类分析方法能够精确地告诉分析者,程序对 Cache 的每次访问是否命中.因此,基于模型检测的 Cache 分析方法具有最高的精确度.但是,这类方法的问题也是显而易见的.如前所述,程序路径的数量随循环内部的分支数量和循环次数呈指数关系,所以基于模型检测技术的分析方法的可扩展性通常很差.对于较大的程序,会遭遇状态空间爆炸问题,难以在合理的时间和空间范围内得到分析结果.

2.2 基于抽象解释的Cache分析方法

鉴于 Cache 分析这一问题存在自身的复杂性,在实际分析中,大多数分析技术通常需要对程序行为及 Cache 状态进行一定程度的抽象,通过适当地牺牲分析精度来换取分析效率的大幅度提高.

基于抽象解释(abstract interpretation)理论[14]的 Cache 分析方法[15,16]因兼具很高的分析效率和分析精度,已

*** 限于篇幅,这里仅提取了 McAiT 工具基于模型检测技术分析部分的功能.McAiT 还支持基于抽象解释的 Cache 分析.可参阅参考文献[10]来了解关于该工具的更多信息.

经成为 Cache 分析领域具有统治地位的分析手段.

客观上,不同访存在 Cache 中的行为特性是不一样的,有些访存的行为规律较为明显,而其他一些访存的行为规律则可能难于描述.基于抽象解释的 Cache 分析试图判定访存是否符合表 1 所示的 3 种访问特性,这些特性被称为“Cache 命中/失效分类(cache hit/miss classification,简称 CHMC)”.

Table 1 Objectives of cache analysis by abstract interpretation

表 1 抽象解释 Cache 分析方法的分析目标

Cache 命中/失效分类	所反映的 Cache 访问行为特征
一定命中(always hit,简称 AH)	每次访问该指令,一定在 Cache 中命中
一定失效(always miss,简称 AM)	每次访问该指令,一定在 Cache 中不命中
首次失效(first miss,简称 FM)	第 1 次访问该指令失效,之后各次访问都在 Cache 中命中
无法确定(not classified,简称 NC)	不能被归类为上述 3 种分类中的任何一类

需要说明的是,客观上,程序的行为特征并非仅有 AH,AM,FM 这 3 种,而是因为主流的基于抽象解释的分析方法[15,16]关注这 3 类行为特性.如果一个访存无法被 Cache 分析归类为上述任何一种,那么称该访存为“无法确定”.造成这一结果的原因有两个:首先,这个访存客观的行为特征就不符合上述 3 种分类;其次,即使符合上述 3 种分类的某一种,但也可能因为分析方法不够精确,最终未能有效地分析该访存的访问分类.

为保证分析的安全性,判定访存 x 对 Cache 的访问是否满足某种性质,需要检查从程序的初始点出发,经过所有可能的程序路径,最终到达 x 之前的所有具体 Cache 状态(concrete cache state,简称 CCS),确定对于每个具体状态该性质都成立.把每条指令访问前的状态定义为一个程序点(program point),那么在每个程序点上都存在这样的一个具体 Cache 状态集合.

显而易见,具体 Cache 状态集合中的状态数量是呈指数爆炸的(与程序的执行路径的数量相关).如果利用具体 Cache 状态进行分析,可扩展性将会非常差.为此,基于抽象解释的 Cache 分析技术在每个程序点维护一个抽象 Cache 状态(abstract cache state,简称 ACS).抽象表示该程序点上的所有具体 Cache 状态.分析的过程也是针对抽象状态进行的.经过抽象,Cache 状态空间被大大缩减,由此带来分析效率的显著提高.

下面以分析 AH 分类为例,说明基于抽象解释的 Cache 分析方法的基本思想.下文中针对一个 Cache 组进行讨论,并将其当做整个 Cache 看待.用来分析 AH 分类的方法称为 MUST 分析.客观上,一个访存 x 能够被判定为 AH,必须保证 x 紧前的程序点上的所有具体 Cache 状态中都包含 x,那么接下来对 x 的访问必然命中.根据这一性质,MUST 分析在抽象状态中为每个内存块维护年龄的上限值(upper bound).对于任何一个内存块,在访问它之前的程序点上,如果抽象状态中的年龄上限小于 Cache 的相联度 A,则能够断定该访存一定命中.MUST 分析抽象域的形式化定义详见文献[15],这里仅用图 6 的例子来粗略解释 MUST 抽象域的直观含义.

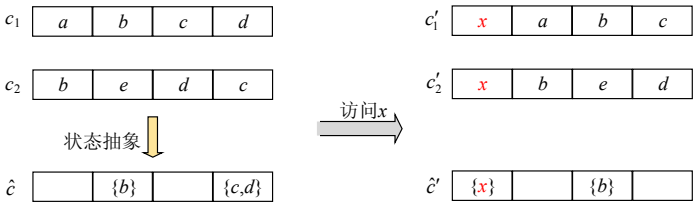


Fig.6 An example to show the MUST abstract domain

图 6 解释 MUST 抽象域的例子

假定 Cache 是 4 路组相联,讨论对 x 的访问.假定从程序起始点到 x 有两条路径,通过这两条路径到达 x 之前的具体 Cache 状态分别为 c_1 和 c_2.可以对这两个具体状态进行抽象,得到抽象状态 $\hat{c}$.仅当一个内存块出现在所有具体状态中时,它才会出现在抽象状态 $\hat{c}$ 中,且该内存块的年龄(即所在位置)为它在 c_1 和 c_2 中的最大年龄.因此,抽象 Cache 状态中的每一路维护的是内存块的集合.如果此时访问 x,根据 LRU 替换策略的工作原理,具体状态 c_1 和 c_2 将更新为 c_1' 和 c_2'.

与此同时,在抽象域中,访问 x 造成对抽象状态的更新,更新行为称为 MUST 分析的更新函数 $U_{MUST}(\hat{c},x)$,显然有 $\hat{c}'=U_{MUST}(\hat{c},x)$. 更新函数是安全的,仅当更新后抽象状态 $\hat{c}'$ 依然能够正确维护每个内存块的年龄上限.例如在图 6 中,访问 x 导致抽象状态中所有内存块的年龄都增加 1,$\hat{c}$ 中年龄为 4 的内存块 c 和 d 被替换出 Cache. 这一行为所表达的语义是:对于 c 或 d,可能存在某个被 $\hat{c}$ 所代表的具体状态,在这个具体状态中,c 或 d 的年龄为 4,访问 x 会把具体状态中的 c 或 d 替换出 Cache.因此在抽象域中,必须增加抽象状态中 c 和 d 的年龄,以保证分析的安全性.

基于抽象解释的分析在每个程序点上维护唯一的抽象状态,但一个程序点上可能有多条入口路径,每条路径上都可能携带一个抽象状态,所以需要一种操作,将多个抽象状态合并为一个抽象状态.这一操作称为合并函数,记作 $J_{Must}(\hat{c}_1,\hat{c}_2)$. 图 7 表示的是 MUST 分析合并抽象状态的例子.可以看出,仅当一个内存块出现在所有入口抽象状态中时,它会被保留在合并后的抽象状态中.内存块的年龄为它在所有入口抽象状态中年龄的最大值.

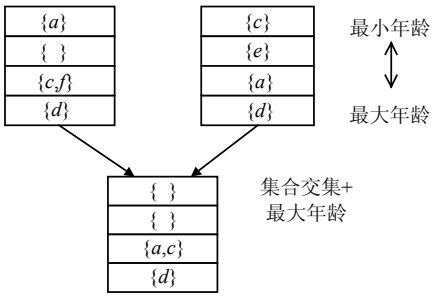


Fig.7 An example of the JOIN function of MUST analysis

图 7 MUST 分析的 JOIN 函数举例

在分析启动时,各个程序点上起始的抽象状态是不包含任何程序执行信息的.为了正确判定访存的 CHMC,必须为每个程序点构造一个能够包含所有可能具体 Cache 状态(即包含了程序的所有可能执行情况)的抽象状态.构造抽象状态并不是一个简单的过程.基于抽象解释的 Cache 分析通过不动点迭代(fixed point iteration)的方法来求得最终的抽象状态,其伪代码如图 8 所示.

算法 1. 基于抽象解释的 Cache 分析的不动点迭代(以 MUST 分析为例).

```
输入:(1) 程序的控制流程图;(2) Cache 配置.
输出:每个程序点上的最终抽象状态.
//迭代轮数
int k=0;                    //k 表示迭代的轮数,初始为 0

//分析初始化
for (i=1 to N)              //N 是待分析程序中程序点的个数
    ACS[i][k]=空状态;   //ACS[i][j]表示第 j 轮中第 i 个程序点上的抽象状态
endfor

//不动点迭代
while (不动点没有达到)
    k++;
    for (i=1 to N)
        for (j=1 to P)     //P 是该程序点所有入口(entry)的个数
            ACS[pre(i)][k−1]=J_MUST(ACS[pre(i)][k−1],ACS[entry[j]][k−1]);
        endfor
        ACS[post(i)][k]=U_MUST(ACS[pre(i)][k−1],m(i));
        //pre(i)和 post(i)分别表示第 i 条指令前面和后面两个程序点
        //m(i)表示第 i 个程序点上要访问的指令造成的访存
        endfor
endwhile
```

Fig.8 Fixed-Point iteration algorithm

图 8 不动点迭代算法

在每轮分析中,对于每个访存,根据该访存输入程序点第 i–1 轮的抽象状态,利用相应的抽象域更新函数,计算得到该访存输出程序点第 i 轮的抽象状态.若发现相邻的两轮中每个程序点上的抽象状态不变,则表明不动点已经达到,迭代结束.此后,根据每个访存之前的抽象状态,判定访存的 CHMC 分类.在 MUST 分析中,如果一个访存出现在它之前的 MUST 抽象状态中,则判定该访存为 AH,否则不是 AH.

以上是使用 MUST 分析判定 AH 分类的完整过程.在基于抽象解释的 Cache 分析技术中,判定 AM 分类是通过 MAY 分析来完成的.MAY 分析的工作过程与 MUST 分析非常类似,主要区别在于 MAY 抽象域的定义,以及该抽象域下的更新与合并函数语义.MAY 分析在抽象 Cache 状态中维护每个访存的年龄下限(lower bound).如果不动点达到后,访存的年龄下限大于 Cache 的相联度 A,则判定该访存在 Cache 中一定不命中(AM).MAY 分析抽象域的形式化定义详见文献[15].

除了 AH 和 AM 两种分类以外,还有一种更重要的分类是首次失效(FM).决定程序执行时间的主要因素是循环体结构,程序局部性原理告诉我们,大多数情况下,程序的循环体是能够被放入 Cache 中的.在第 1 次执行循环体时,需要将其载入 Cache,产生冷失效;在以后各轮循环执行过程中,循环体内部的访存将是命中.所以,这些访存的行为特性呈现为首次失效.在基于抽象解释的分析技术中,PERSISTENCE 分析用于判定 FM 分类,其抽象域类似于 MUST 抽象域和 MAY 抽象域的结合.类似于 MAY 抽象域,PERSISTENCE 抽象域记录所有可能被访问的元素的信息,以捕捉“元素被载入 Cache”这一行为.类似于 MUST 抽象域,PERSISTENCE 分析为每个访存维护年龄上限,用以判定元素被载入后是否一直驻留在 Cache 中.PERSISTENCE 分析首先由 Ferdinand 在文献[15]中提出,但其分析方法在提出 12 年后被发现是不安全的[17−19].Cullmann[17]和 Huynh 等人[18]分别提出了不同的 PERSISTENCE 分析方法,解决了上述问题.

MUST,MAY 和 PERSISTENCE 分析完整地构成了基于抽象解释的 Cache 分析技术.客观上,访存的行为特性可能并不仅限于 AH,AM,FM 这 3 种.但实际应用表明:基于抽象解释的 Cache 分析技术具有很好的精确性,这说明对于 LRU 替换策略下的指令 Cache 而言,这 3 种访问特性分类足以有效覆盖指令访存行为的大部分情况.基于抽象解释的 LRU Cache 分析技术在实时系统时间分析领域已建立起统治地位,很多分析方法都是在该分析技术基础上提出的.

Müller 等人曾提出了静态 Cache 模拟(static cache simulation)技术[20,21].其本质上也是基于抽象解释的 Cache 分析技术.该技术试图挖掘的 Cache 访问行为特性(CHMC 分类)以及分析框架与 Ferdinand 提出的分析技术是类似的,两种方法的主要区别在于抽象域的定义.静态 Cache 模拟技术利用统一的抽象 Cache 状态来同时判定 AH,AM,FM 这 3 种访问特性,其抽象域与抽象解释分析中的 MAY 抽象域很相似,记录在程序执行过程中所有可能的访存.在得到最终的抽象状态之后,根据一系列规则来判定访存的 CHMC 分类.静态 Cache 模拟的抽象域比 Ferdinand 的抽象域丢失更多信息,因此分析精度要差一些.

2.3 Cache状态转换图方法

实际上,在基于抽象解释的 Cache 分析技术出现之前,Li 等学者曾提出过一种静态 Cache 分析技术.该技术用 Cache 冲突图(cache conflict graph)/Cache 状态转换图(cache state transition graph,简称 CSTG)来建模 Cache 中的替换行为[22,23],简称为 CSTG 方法.在分析组相联 Cache 时,需要为每个 Cache 组建立 CSTG 图(本质上是一个有向带环图).图中节点为所有可能的具体 Cache 状态,节点之间的有向边表示存在程序的某种执行情况,可以导致从起点 Cache 状态迁移到终点 Cache 状态.根据 CSTG 图中信息,可以为每个访存的命中次数建立线性约束.把这些 Cache 行为的线性约束和程序结构的线性约束合并为一个整数线性规划问题,就可以求解程序的 WCET.同时,也能从此时的变量取值获得最坏情况下每个访存具体的 Cache 命中次数.

可以从不同角度对基于 CSTG 图的 Cache 分析技术进行评价.从抽象程度来讲,CSTG 方法近乎是对具体 Cache 状态的枚举,显然,其抽象程度要低于抽象解释分析技术.较低的抽象程度导致 CSTG 方法有较高的分析精度,但是分析效率低下及可扩展性差,是这种方法的根本问题.假定组相联 Cache 的相联度为 A,映射到某一个 Cache 组的内存块的个数为 M,理论上,针对这个 Cache 组的 CSTG 图中的 Cache 状态个数如公式(3)所示[23].如此庞大的节点数量将导致同样规模的线性约束条件的数量,因此,求解从 CSTG 方法得到的整数线性规划问题,

复杂度是非常高的.

$$\sum_{i=0}^{A}\frac{M!}{(M-i)!} \tag{3}$$

从分析结果的本质来看,CSTG 方法的分析目标也与抽象解释分析差别很大.CSTG 方法用线性约束建模 Cache 行为,最终得到的是每个访存在最坏执行情况下的命中次数.从某种角度讲,这是关于 Cache 命中行为的一种数量统计(quantitative)形式的表达;而抽象解释分析则试图挖掘诸如 AH,AM,FM 这些不变式(invariant)所描述的确定访问规律.显然,CSTG 对 Cache 访问特性的表达比基于抽象解释的方法更具有一般性.也可以从这个角度解释为什么 CSTG 方法在一般情况下能够获得更加精确的分析结果.但是分析效率低这一根本问题,导致 CSTG 方法在面向实际程序的 Cache 分析中很少被采用.

3 其他维度上的 Cache 分析问题与相关技术

上节主要针对“面向 LRU 替换策略的单级指令 Cache”这种特定的配置,综述了当前主流分析技术.这些技术是进行任何 Cache 分析的理论基础.实际系统中的 Cache 体系结构是多种多样的,不同的 Cache 体系结构为 Cache 分析带来了很多新的问题和挑战.本节将从循环结构分析、数据 Cache 分析、多级 Cache 分析、多核共享 Cache 分析、非 LRU 替换策略分析等不同维度探讨 Cache 分析问题的延伸及相关分析技术.

3.1 循环结构的分析

第 2.2 节介绍了基于抽象解释的 Cache 分析技术,其中一个隐含假设是:将一个程序看做整体进行分析,得到的结果自然也是从全局角度所刻画的 Cache 访问特性.但是当程序中存在复杂循环结构时,采用上述方法有可能得到过于悲观(甚至不可用)的分析结果.以图 9(a)的例子来说明这一问题.

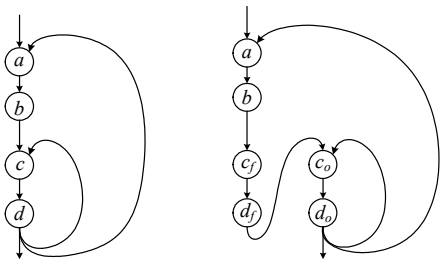
假定 Cache 采用 LRU 替换策略,2 路组相联.每个访存如果在 Cache 中命中,时间延迟为 1,不命中时间延迟为 10.各层循环的执行次数为 10.外层嵌套循环有 *a*,*b*,*c*,*d* 这 4 个内存块映射到同一 Cache 组中,其中,*c* 和 *d* 属于内层循环.假定外层循环入口处的 Cache 状态为空,如果采用抽象解释分析将这个嵌套循环当做一个整体来分析,则分析结果将是 *a* 和 *b* 被判定为 AM,而 *c* 和 *d* 被判定为 NC.在 WCET 计算过程中,NC 将被当做 AM 处理,因此这个嵌套循环最终计算得到的 WCET 值为 2 200.

但是很容易看出,每次进入内层循环时,*c* 和 *d* 一定不在 Cache 中,因为访问 *a* 和 *b* 会将这两个内存块替换出 Cache.但是一旦进入内层循环,则在剩余的 9 轮执行中,*c* 和 *d* 一定是命中的.也就是说,虽然从全局的角度 *c* 和 *d* 的访问特性不能归类为 FM,但是从内层循环的局部行为来看,*c* 和 *d* 呈现首次失效访问特性.据此计算得到的 WCET 值为 580.相比之下,将嵌套循环视为整体的分析方法得到的 WCET 值过于悲观.

因此,Cache 分析面临一个新挑战:如何挖掘访存在各层循环级别上的局部访问特性.如果不能提出有效的分析手段,那么 WCET 分析的结果将不可用.在研究领域,针对这一问题有两种分析技术:VIVU(virtual inlining and virtual unrolling)[24]与多层 PERSISTENCE 分析[25].

VIVU 技术的思路是:通过展开循环体(对应的 CFG)区分循环的首次执行和其他各次执行,对展开后的 CFG 进行分析,以发现循环体在首轮和其他各轮被访问时的不同访问特性.为表达简洁,仅展开图 9(a)所示 CFG 的内层循环体.展开后的 CFG 如图 9(b)所示,其中,c_f 和 d_f 表示内层循环的首轮执行;c_o 和 d_o 表示内层循环其他各轮执行,且 c_o 和 d_o 所在循环体的执行次数为 9 次.显而易见,图 9(a)和图 9(b)的程序语义等价.对图 9(b)的 CFG 进行 MUST 分析可以发现,c_o 和 d_o 都被判定为 AH.把 c_f 和 c_o 的分析结果结合起来能够说明,每次进入内层循环,“内存块 *c* 除首轮之外的执行都是命中”这一局部 FM 行为特性被有效地捕捉到,解决了前面所提出的问题.

Ballabriga 等人提出了多层 PERSISTENCE 分析技术[25]来解决这一问题.他们不对循环体进行展开,而是分别针对各层循环体进行 PERSISTENCE 分析,得到的结果就是在对应循环层次上的 FM 访问特性.以图 9(a)为例,首先对 *a*,*b*,*c*,*d* 所在的外层循环进行 PERSISTENCE 分析,没有任何一个访存能被判定为 FM.之后,对 *c*,*d* 所在的内层循环体的局部 CFG 单独进行 PERSISTENCE 分析,结果是 *c* 和 *d* 都能够被判定为 FM.这个结果的含义是:每次进入内层循环,*c* 和 *d* 都表现为 FM 的行为特性.

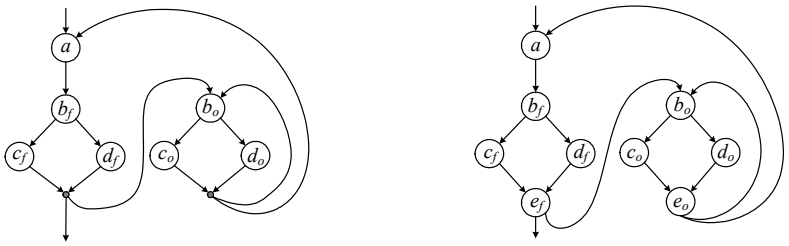


(a) 原 CFG 图 (b) 展开内层循环后的 CFG 图

Fig.9 An example of nested loop

图 9 嵌套循环的例子

对于图 9(a)这一程序,上述两种方法的分析结果是完全等价的.但是对于更复杂的程序,上述两种方法可能得到不同的分析结果.以图 10 的两个例子进行说明.限于篇幅,仅给出内层循环展开后的 CFG,原 CFG 可以很容易推出.假定 Cache 为 3 路组相联,所有出现的内存块都映射到同一 Cache 组.对图 10(a),采用 VIVU 的方法,c_o 和 d_o 因位于两个分支中,都无法被 MUST 分析判断为 AH,最终,c_f,d_f,c_o,d_o 都只能被判定为 NC.如果采用多层 PERSISTENCE 分析的办法,通过对 b,c,d 所在的内层循环体的局部分析,3 个内存块都能够被判定为 FM,内层循环的局部 FM 访问特性被分析出来.反之,对于图 10(b),运用多层 PERSISTENCE 分析,b,c,d,e 这 4 个内存块都无法被判定为 FM.如果采用 VIVU 的方法,尽管 MUST 分析不能将 c_o 和 d_o 判定为 AH,它却能够发现 b_o 和 e_o 是 AH.因此在 VIVU 分析下,b 和 e 的局部 FM 访问特性被分析出来.



(a) 多层 PERSISTENCE 分析好于 VIVU 的例子 (b) VIVU 好于多层 PERSISTENCE 分析的例子

Fig.10 Comparison of VIVU and multi-level PERSISTENCE analysis

图 10 VIVU 方法与多层 PERSISTENCE 分析方法的比较

因此,对于 VIVU 技术和多层 PERSISTENCE 分析,没有任何一种方法严格好于另外一种.两者精确性的优劣,本质上是 MUST 抽象域和 PERSISTENCE 抽象域[****]的精确性所决定的.对于图 10(a)所示的例子,MUST 分析比 PERSISTENCE 分析丢失更多信息;而对于图 10(b)所示的例子,PERSISTENCE 分析则比 MUST 分析丢失更多信息.在后面的讨论中,将不再重复探讨对多层循环的分析.

3.2 数据Cache分析

上文讨论主要针对指令 Cache.但是大多数处理器都会配备独立的指令 Cache 和数据 Cache,分别缓存程序指令和被操作的数据.程序对指令的访问规律性强,可预测性高,因此,现有技术分析指令 Cache 都能获得很高的分析精度.但是,程序对数据的访问行为要复杂得多:首先,对于指令 Cache 而言,执行一条指令引发对唯一内存块的访问;而对于数据 Cache,执行一条指令可能导致对多个数据块的访问.如图 11 所示,用一个嵌套循环实现矩阵乘法:在空间上,同一语句一次操作多个矩阵的数据(程序第 4 行);在时间上,同一语句在循环的不同轮次里操

[****] 目前,学术界有多种 PERSISTENCE 抽象域及相应分析技术存在,它们对于同一个程序(或循环体)分析得到的结果可能不同.因此,比较多层 PERSISTENCE 分析和 VIVU 分析技术,与采用哪种 PERSISTENCE 抽象域有直接关系.

作不同的数据(矩阵不同的行、列).正是因为程序对数据和指令的访问特性在空间和时间上有明显的不同,分析数据 Cache 要比分析指令 Cache 复杂得多.简单地将分析指令 Cache 的技术套用到数据 Cache 的分析上,将导致分析结果过于悲观.数据 Cache 分析的主要挑战,就是如何精确分析数据访问的时空特性.

```
1    for (i=0; i<N; i++)
2      for (k=0; k<N; k++)
3        for (j=0; j<N; j++)
4          C[j][i]+=A[k][i]*B[j][k]
```

Fig.11 An example of matrix multiplication

图 11 矩阵乘法运算的例子

为了分析数据 Cache,首先要确定执行每条指令时所有可能被访问的数据块的集合.这并不是一个简单的问题,主要原因有以下两个方面:

- 第一,从体系结构的角度,如果对数据的寻址方式仅有直接寻址一种,那么被访问数据的地址可以从指令中直接读到;但是主流处理器都采用了复杂的多级寻址机制,被访问数据的地址并不是直接写入指令,而是存于某些特定的寄存器中,这就给确定访问数据的物理地址造成了很大的困难.
- 第二,从程序语言的角度,通常在程序中使用指针访问数据.在实际运行过程中究竟会访问哪些数据,是受程序语义控制的.因此,作为 Cache 分析之前的准备工作,确定每条指令可能访问的数据集合是很复杂的.目前,分析数据访问地址的主要方法包括编译技术中常用的数据流分析技术[26]以及基于抽象解释的分析技术[27].

Ferdinand 等人和 Sen 等人分别将抽象解释框架下的面向指令 Cache 的 PERSISTENCE 分析[28]和 MUST 分析[29]扩展到了数据 Cache.但是这两种分析仅试图将指令 Cache 分析简单扩展到数据 Cache,对元素年龄上限的维护过于悲观,导致分析结果精确性很差,因此需要从数据 Cache 特有的行为特性入手,提出新的分析技术.

一种可能的思路就是从时间维度上发掘数据访问的局部性,并将这种时间局部性信息建模到分析所采用的抽象域中.代表性技术是 Huynh 等人提出的范围敏感(scope-aware)的数据 Cache 分析[18].他们发现:对于循环体中的指令而言,即使在一点上可能访问多个数据,它们也是在循环的不同轮次被访问的.针对这一规律,可以在传统的 PERSISTENCE 抽象域中为每个内存块标记访问时间范围(temporal scope)信息,即每个内存块可能被访问的循环轮次.在进行更新时,如果内存块 x 的时间范围与抽象状态中 y 的时间范围无交叠,那么访问 x 不会增加 y 在抽象状态中的年龄.通过上述手段,把每个内存块自身的行为特性及对其他内存块的影响都限定在了特定的循环轮次中,这是对数据访问时间局部性的更加精确的建模,因此能够有效地提高分析的精确性.

此外,Ernst 等人提出了一种分析技术,以解决 Ferdinand 提出的面向数据 Cache 的 PERSISTENCE 分析的悲观性问题[30].该分析方法的实质是:为访存建立失效计数器(miss counter),从程序全局的角度约束最多可能发生的失效的次数.例如,执行循环中的某条指令可能访问 a,b,c,d 这 4 个数据,但每次仅访问其中的 1 个.在文献[28,29]的分析方法中,在这个程序点上,由于不能确定具体访问 4 个数据中的哪一个,因此更新语义是:每个访问都会造成抽象状态中内存块的年龄增加.如果 Cache 相联度为 4,那么访问这 4 个内存块后,抽象状态中的所有其他内存块都已经被替换出 Cache.但实际上,每次指令执行只访问 1 个内存块,那么最多只能造成抽象状态中年龄最老的内存块被替换出去.失效计数器就是用来捕捉上述行为并约束失效数量上限的.通过这种手段,能在一定程度上弥补文献[28,29]方法的悲观性.

Huynh 等人的方法是更精确地描述单个访存的行为特性,而 Ernst 等人的方法则是从程序全局发掘整体的行为特性.无论哪种思路,精确分析数据 Cache 的根本,都是准确建模数据访问的时间局部性特征.

3.3 多级Cache分析

以上分析假设 Cache 只有 1 层,且指令 Cache 与数据 Cache 是相互独立的.对于这样的配置,程序对 Cache 的访问行为相对简单,因此对分析技术的挑战是有限的.但大多数实际处理器通常采用图 2 所示的多级 Cache

(multi-level cache)体系结构.以一个两级 Cache 为例,当程序需要访问数据或指令时,首先在 L1 Cache 上查找;如果不命中,再到 L2 Cache 查找;再不命中,才会访问内存.尽管各级 Cache 的访问速度不同,由于 Cache 位于 CPU 内部,而访问内存却需要通过片外内存总线,因此,末级 Cache 的访问速度总是远远高于内存的访问速度.为了获得精确的 WCET 估计值,必须对各级 Cache(而不仅仅是第一级 Cache)进行有效的分析.

分析多级 Cache 又会引发很多新问题:

- 首先,无论访问指令或数据,都一定首先访问 L1 Cache.如果在 L1 Cache 中命中,就不会造成对 L2 Cache 的访问.因此,需要在相邻两级 Cache 之间建立“是否访问下级 Cache”的访问关系模型.
- 其次,在多级 Cache 硬件设计中,相邻级别 Cache 之间的包含关系是一个重要的设计选项.典型的包含关系有两类:非包含结构(non-inclusive)与包含结构(inclusive).对于包含结构,要求第 i+1 级 Cache 必须严格包含第 i 级 Cache 的内容;对于非包含结构,则不作此要求.给分析带来更大困难的是包含结构,这是因为如果某次访存造成某级 Cache 中的某个元素被替换出去,那么在其他各个级别上都需要相应地将这个元素替换出去以保持包含特性.这种行为造成了相邻两级 Cache 上访问行为的相互依赖,给分析带来难度.
- 再次,访问数据 Cache 涉及对 Cache 的写操作.如果系统只有一级 Cache,那么无论采用直写(write through)方式还是写回(write back)方式,都不会对数据 Cache 的行为造成显著的影响.但是在多级 Cache 结构下,采用写回还是直写机制,甚至可能对分析的安全性产生影响[31].

Müller 最早进行了多级 Cache 分析的研究[32],将用于单级 Cache 分析的静态 Cache 模拟技术推广到了多级 Cache 的分析.这一工作开启了多级 Cache 分析的一类典型思路,即,从第 1 级开始逐层分析各级 Cache.根据第 i 级分析得到的 Cache 命中/失效分类(cache hit/miss classification,简称 CHMC),确定是否会访问第 i+1 级 Cache.

但是,Müller 的分析方法被 Hardy 发现是不安全的[33].这是因为在 Müller 的分析方法中,如果一个访存在第 i 级 Cache 中被判定为 NC,那么在讨论是否会访问 i+1 级 Cache 时,会被当做 AM 处理.显然,在单级 Cache 分析中,将 NC 视为 AM 是标准的做法;但是在多级 Cache 分析中,就会导致分析结果不安全.文献[33]给出了这一问题的具体例子,通过纠正 Müller 分析方法的错误,Hardy 等人奠定了多级 Cache 分析的理论基础.他们提出了针对多级非包含 Cache 的抽象解释分析框架,如图 12 所示.

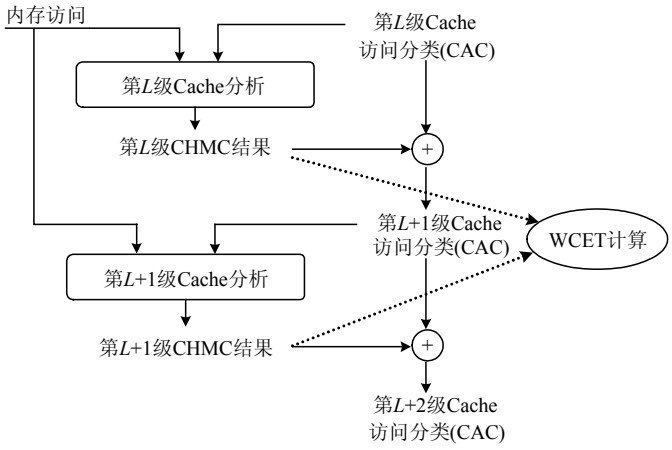


Fig.12 Framework of multi-level Cache analysis[33]

图 12 多级 Cache 分析框架[33]

首先,采用传统的抽象解释分析技术对第 L 级 Cache 进行分析,得到该级的 CHMC 分类;同时,对于第 L 级 Cache,定义 Cache 访问分类(cache access classification,简称 CAC).CAC 描述了对于一种特定的 CHMC 分类,是否会造成对下一级 Cache 的访问.根据第 L 级 Cache 的 CHMC 分析结果以及 CAC 分类,可以确定对 L+1 级 Cache

的 CAC 分类.第 L+1 级的 CAC 分类信息就像一个过滤器,它决定了程序的哪些访存对第 L+1 级 Cache 有效.根据以上结果,就可以继续使用抽象解释技术来分析第 L+1 级 Cache.利用第 L 级的 CHMC 结果和 CAC 分类计算第 L+1 级 CAC 分类的具体方法详见文献[33].

在 CAC 分类中会出现“不能确定一个访存是否一定会对第 L+1 级 Cache 造成访问”的情况.Hardy 等人的解决思路如下:当一个访存的 CAC 特性为不确定时,分别针对访问和不会访问两种情况进行抽象状态的更新,然后,使用对应抽象域的合并函数,将上述两种情况计算得到的抽象状态合并为一个,如图 13 所示.通过综合考虑两种可能性,有效地解决了 Müller 分析方法不安全的问题.

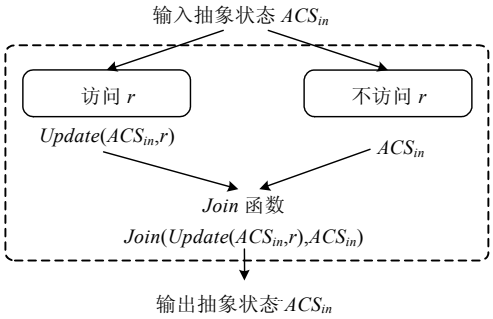


Fig.13 Update function in multi-level cache analysis[33]

图 13 多级 Cache 分析中的更新函数[33]

Hardy 等人的工作虽然仅针对非包含多级指令 Cache,但是却奠定了多级 Cache 逐层分析的理论框架.这一工作被 Lesage 简单地扩展到了对数据 Cache 的分析[34],被 Chattopadhyay 扩展到了对联合 Cache 的分析[35].

3.4 多核共享Cache分析

随着应用需求的提升,实时系统的复杂度也正在迅速提高.实时系统大多是嵌入式系统,因而对于系统的体积、重量、功耗等方面有着苛刻的要求.传统的单核处理器已经远远无法满足上述要求,多核处理器应用于实时系统已经成为不可逆转的发展趋势[36].

在多核处理器中,所有处理核心通常共享末级 Cache,其体系结构如图 14 所示.由于所有核心上的任务并行执行,因此在共享的末级 Cache 上,任务之间会发生冲突,也就是说,在一个任务执行完成之前,它在末级 Cache 中的数据或指令会被其他任务替换出 Cache.这一现象称为任务间干涉(inter-task interference).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 分析,其难度远远超过单核 Cache 分析.

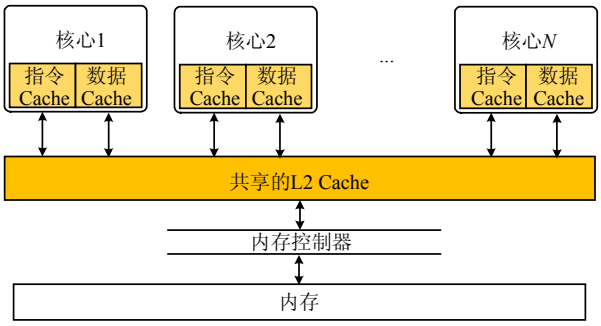


Fig.14 Architecture of shared Cache in multi-cores

图 14 多核共享 Cache 体系结构

多核共享 Cache 分析的关键是如何对任务间在共享 Cache 上的干涉行为进行有效的抽象,具体而言,就是要找到一种既能保证分析精确性,又能有效削减状态空间的干涉模型.目前,相关研究主要有以下几种技术路线:

第 1 类方法采用模型检测技术对基于多核共享 Cache 的体系结构进行 WCET 分析[10,12,37],其原理与此前介绍的基于模型检测的单核 Cache 分析是类似的.可以将每个核心上运行的程序建模成一个时间自动机,同时将共享 Cache 也建模为时间自动机.通过运行模型检测器搜索系统的全部状态空间,找到每个程序执行时间的最大值.这一方法的本质是,不对程序行为做任何抽象.在单核环境下,基于模型检测的分析方法的可扩展性已经是一个比较严重的问题;对于多核共享 Cache 分析,将出现更加严重的状态空间爆炸问题.

第 2 类分析方法主要是扩展和改造单核体系结构下基于抽象解释的分析技术.Yan 等人率先开展多核共享 Cache 的 WCET 分析[38].他们首先采用抽象解释技术对每个核心上的任务进行多级 Cache 分析,得到各级 Cache 上的 CHMC 分类结果.然后,分析程序之间在共享 Cache 上是否会产生干涉.如果一个程序的某个访存可能被其他任务干涉,其 CHMC 访问特性都将变为 NC.这一方法显然是安全的,但是精确性难以得到保证.此外,Yan 等人的方法仅能处理直接相联 Cache.

Liang 等人采用类似的思路,将面向组相联 Cache 的抽象解释分析方法扩展到了多核共享 Cache 分析[39].其分析过程仍旧是先对每个核心上的程序进行多级 Cache 分析,得到 CHMC 分类结果.之后,分析程序在共享 Cache 上的干涉行为.在第 2 步中,假定 A 程序与 B 程序并行执行,欲分析 A 程序的 WCET.那么对于 A 程序中的每个内存块,可以从抽象状态中提取它在末级 Cache 上的最大年龄(可能来自 MUST 分析,也可能来自 PERSISTENCE 分析).假定 Cache 相联度为 A,内存块 m 的最大年龄为 $age(m)$.显然,如果 B 程序向该 Cache 组中载入的内存块的个数不超过 $A–age(m)$,就能保证 m 不会因为程序 B 的执行而被替换出 Cache;否则,将 m 的 CHMC 分类变为 NC.此外,Liang 等人还分析了每个基本块的生命周期,如果两个基本块的生命周期不重叠,那么客观上它们之间是不会发生干涉的.通过这一信息,可以进一步提高分析的精确性.陈芳园等学者也采用类似的技术对多核共享 Cache 分析进行了研究[40].

将单核系统上的抽象解释分析扩展到多核共享 Cache,也势必将这一技术的缺点一并引入.基于抽象解释的 Cache 分析,其不精确性的一个主要来源就是路径分析与 Cache 行为分析的分离(在单核非共享 Cache 分析中,这种路径分析与 Cache 分析分离的框架在实践中被验证是非常精确的.在多核共享 Cache 分析中,这种抽象的悲观性被放大出来).采用第 1.2 节所介绍的方法,分析得到的最坏路径在实际执行过程中可能是不可行的(infeasible path).因此,如果能够发现程序的不可行路径,并将其从程序行为中剔除,那么无论对于单核 Cache 分析还是多核 Cache 分析,都将提高分析的精确性.Chattopadhyay 等人和 Banerjee 等人采用模型检测、SAT 等技术来分析不可行路径[41,42].通过在一个迭代过程中不断发现新的不可行路径并将其剔除,可以对分析结果逐步求精.

尽管存在上述分析方法,目前多核 WCET 分析领域也还没有较好的干涉行为抽象模型出现.鉴于多核共享 Cache 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.如果设计技术足够合理,甚至可能提高程序的性能.主要技术手段有如下几类:第 1 类方法通过 Cache 染色技术[43]将共享 Cache 分成若干区域,通过控制操作系统的内存分配机制,使得不同核心上的程序映射到共享 Cache 上不交叠地址空间,因而不会发生干涉;其次,实时领域常见的 Cache 锁定技术[44,45]等也可用于多核共享 Cache 以消除或减少任务间干涉.Cache 锁定的本质是把固定的内容锁定到 Cache 中,那么任何访存在 Cache 中是否命中就是确定的.如果被锁定的内容选择合理,还可能进一步提高程序的性能.此外,Hardy 采用 Cache 旁路(by-pass)技术来达到类似的目的[46].Cache 锁定是选择最常用的数据载入 Cache,而旁路技术会分析程序中不会被反复访问的访存,在执行过程中不将其载入 Cache,避免了这类访存对其他访存造成的干涉.

3.5 非LRU替换策略分析

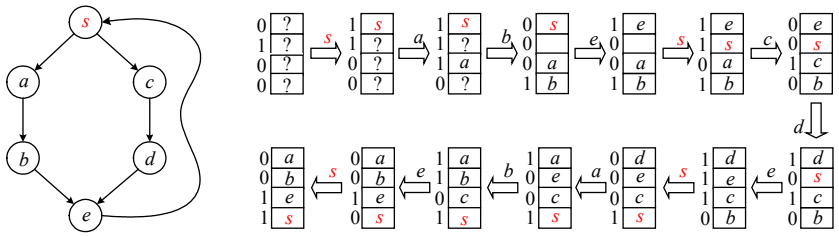
在以上讨论中,无论针对指令 Cache 还是数据 Cache、单级还是多级 Cache、独占还是共享 Cache,一个共同的假设是采用 LRU 替换策略.目前,研究领域普遍认为 LRU 替换策略的可预测性最好,但是由于 LRU 替换策

略的硬件实现代价很大,如硬件逻辑电路、功耗、散热等,实际处理器中通常使用类 LRU 的替换策略,如 MRU (most recently used),PLRU(pseudo-LRU),FIFO(first in first out)等.这些替换策略不仅易于硬件实现,且能够达到较高的平均性能[47].即便如此,多年来,绝大多数面向 WCET 估计的 Cache 分析技术都假设采用 LRU 替换策略.究其根本,是因为在非 LRU 替换策略下,程序在 Cache 中的行为特性非常复杂,难以建模.仅仅针对单级指令 Cache,分析访存的命中情况已经是非常困难的问题.

在非 LRU 替换策略分析领域,Reineke 等人和 Grund 等人提出了一种基于相对竞争性(relative competitiveness)的分析框架[48].这一分析框架的基本思想是:在两种替换策略 A 和 B 之间建立相对竞争关系,如果已知替换策略 A 下的 Cache 失效数量的上限,就可以根据分析得到的竞争关系,间接计算出程序在替换策略 B 下的 Cache 失效数量的上限.与抽象解释分析试图获取 AH,AM,FM 等分类不同,相对竞争性分析方法的本质不是针对具体的每个访存进行行为特性的分析,而是将整个程序看做一个整体,对 Cache 失效给出一种全局的数量统计表达.但是该分析方法也存在一些问题:首先,该方法不是针对每个访存的特性进行分析,因此得到的信息是比较粗糙的,一般情况下,分析结果不精确;其次,该方法随着 Cache 相联度的增加会遭遇可扩展性的问题.

此后,Grund 等人提出了一种针对 FIFO 替换策略的分析方法[49],基于抽象解释理论为 FIFO 替换策略下的 Cache 行为建立了一个较为复杂的 MUST 抽象域和 MAY 抽象域,采用与 LRU 替换策略类似的不动点迭代的方法对程序在 Cache 中的行为进行分析.但是,这种分析框架的本质还是试图分析哪些访存具备 AH 和 AM 类型的访问特性.结果表明,这两种分类并不能够有效地描述程序在 FIFO 替换策略下的行为特征,因此,分析结果很不精确[50].这说明,欲提高非 LRU 替换策略的分析精度,需要深入挖掘在给定替换策略下程序特有的行为特征.

Guan 等人在面向非 LRU 替换策略(MRU 和 FIFO)的 WCET 分析领域也开展了大量研究,发现了一些 Cache 访问行为特征的新规律[50,51].研究表明,在 MRU 替换策略下,循环中的访存存在如图 15 所示的规律.对于图 15(a)的循环体,观察内存块 s 的 Cache 命中特性可以发现:在经过若干次 Cache 失效之后,s 会驻留在 Cache 组中年龄较大的位置,在循环剩余的轮次中,访问都为命中.他们证明了内存块 s 失效的次数是有上限的,且这一上限与 Cache 的相联度有关.在 MRU 替换策略下,访存的这种行为特性被称为 K-MISS.这种全新的 Cache 访问特性分类是对 LRU 替换策略分析技术的一次重要扩充.



(a) 循环体 (b) 一个访存序列的例子

Fig.15 A demonstrating example to show the K-MISS behavior under MRU[51]

图 15 展示 MRU 替换策略下 K-MISS 行为的例子[51]

基于上述工作,Guan 等人研究了面向 FIFO 替换策略的 Cache 分析问题.他们发现:对于那些反复被访问的内存块,它的命中和失效的数量在满足一定条件的情况下符合某种比例关系.以图 16 所示的访存序列为例,假定 Cache 相联度为 4,如果在两次访问 δ 之间有至少 4 个其他访存,那么后一次访问 δ 一定失效.对于图 16 的访存序列,5 次对 δ 的访问中有 3 次为失效.通过分析对某个内存块的任意两次相邻访问之间的距离,并确定这个距离不超过某个固定值,可以得到命中与失效的比例.

这种分析方法比现有方法更加精确.与基于相对竞争性的分析方法相比,该方法的优势体现为两点:

- 首先,基于相对竞争性的方法发掘的是程序全局的行为特性,而 Guan 等人的方法分析的是具体每个访存的行为特性.

- 其次,基于相对竞争性的方法所给出的失效数量上限的比例,是针对任何程序的一个安全的比例,因此一定非常悲观;而 Guan 等人的方法针对具体的程序进行分析,得到的是反映程序个体行为特性的结果.与 Grund 的分析方法相比[49],由于采用了 Cache 命中/失效比例这种更反映 FIFO 替换策略下程序行为特性的描述(不是简单地采用 AH,AM 这些不反映 FIFO 下程序行为特征的访问分类),精确性更高.

$$\delta \to a \to b \to \delta \to c \to d \to \delta \to e \to f \to g \to \delta \to h \to i \to \delta$$

Fig.16 An example of memory accesses under FIFO[50]

图 16 FIFO 替换策略下访存的例子[50]

3.6 Cache与Cache分析对时间可预测性的影响

时间可预测性(timing predictability)是所有实时系统研究者所共同关注的根本问题.研究 Cache 分析,不仅要研究用于 WCET 估计的具体分析技术,而且需要将 Cache 分析放入实时系统时间特性分析的大框架中,深入探讨 Cache 对程序时间可预测性有何影响,以及不同 Cache 分析技术对结果判定的影响.本节从时间可预测性的基本概念出发,首先说明 Cache 是对时间可预测性有重要影响的硬件部件;其次,具体剖析各种 Cache 硬件特性中影响较大的因素;最后,通过实例阐述 Cache 分析技术对上述评价的影响.

在文献[52]中,Axer 提出了时间可预测性的一般性量化定义,如公式(4)所示:

$$Pr_p(Q,I) := \min_{q_1,q_2 \in Q} \min_{i_1,i_2 \in I} \frac{T_p(q_1,i_1)}{T_p(q_2,i_2)} \tag{4}$$

其中,Q 表示程序所有可能的初始硬件状态;I 表示程序所有可能的输入条件;Pr 表示程序的可预测性,其值越小,可预测性越差.显然,Pr 值等价于在各种可能的执行情况下,一个程序 BCET 和 WCET 的比值.换句话说,这里可预测性是从任务执行时间波动的角度来定义的.

虽然上述定义对时间可预测性进行了清晰的描述,但是找到客观上的 BCET 和 WCET 值还是非常困难的.在实际应用中,通常采用分析手段获得 BCET 和 WCET 的估计值.以 WCET 为例,分析得到的通常是程序客观 WCET 的悲观估计,将其代入公式(4)进行计算,获得的 Pr 值会更小,分析得到的可预测性就更差.在文献[53]中,Reineke 等人对这一问题进行了阐述.他们用不确定性(uncertainty)和惩罚(penalty)的乘积定义 WCET 估计值与程序客观 WCET 的差值,其中,不确定性是指那些在静态分析中不得不考虑而在程序实际执行中一定不会发生的时间行为;惩罚表示分析中每个这样的时间行为对 WCET 估计值造成的影响.可见,WCET(以及 BCET)分析的精确性对于准确评估程序的时间可预测性具有重要影响.

显然,Cache 是一个能够带来高惩罚的硬件部件.因为 Cache 命中和失效的时间延迟差距巨大,对命中行为的不精确分析会显著增加 WCET 估计的悲观性.因此,Cache 是影响程序时间可预测性的重要因素.前文所讨论过的每一种 Cache 体系结构特性都可能带来不同程度的惩罚,而 Cache 替换策略则是这些因素中影响最显著的之一.

评判一个替换策略的可预测性如何,本质上要看这种替换策略下程序的执行时间波动情况.为了简化讨论,我们假定程序的 BCET 不变,仅探讨 WCET 方面的影响(同样的分析思路可以用于讨论 BCET 对可预测性的影响).显然,有两个方面的因素会影响到最终的评判结果:首先,对于两种替换策略 A 和 B,程序在 A 策略下的客观 WCET 越大,显然,A 策略带来的时间可预测性就更差;其次,即使在两种替换策略下程序具有相同的 WCET,如果对某个替换策略的分析技术不精确,那么得到的 WCET 估计值也就会更悲观,最终计算得到的这个替换策略的时间可预测性就不好.

针对这一问题,Reineke 等人定义了两个概念:*evict* 和 *fill*,以具体描述不同 Cache 替换策略的可预测性[53].对于给定的替换策略,*evict* 的取值表示经过多少个两两不同的访存后,就可以安全地确定某些元素一定不会存在于 Cache 中;*fill* 的取值表示经过多少个两两不同的访存后,A 路组相联 Cache 中的元素一定是最后 A 个访问过的内存块.这两个参数描述了对于一种给定的替换策略,关于 Cache 命中与否的信息能够多快被获得,因此给出了针对不同替换策略进行 Cache 分析所能达到的精确性的极限.

根据这一可预测性定义,LRU 替换策略的这两个值较小,因此可预测性较好;MRU 替换策略的这两个值较大,被认为可预测性不好.但是 Guan 等人的研究结果表明[51]:对于相同的程序,MRU 替换策略下分析得到的 WCET 估计值与 LRU 替换策略下的很接近,这说明 MRU 替换策略的时间可预测性比较接近 LRU.这一差距存在的主要原因是,Reineke 等人的衡量标准是对一个访存在任何可能情况下的某种性质的描述.而 Guan 等人的分析结果是针对实际程序展开的,在实际程序中,由于程序行为具有一定的规律,所以在大多数情况下,程序在 Cache 中的实际命中/失效行为并不像 *evict* 和 *fill* 的极限值所描述的那样悲观.这是从不同角度、采用不同技术衡量程序在 Cache 上的行为特性的结果.这也表明,准确地衡量一种 Cache 体系结构特性对时间可预测性的影响,必须拥有更高质量的分析技术.

4 总结与未来的研究方向

实时系统的首要任务是分析程序的最坏情况执行时间.由于 Cache 是影响程序执行时间的最重要因素之一,因此,Cache 分析的质量直接决定了执行时间分析的质量,并进而影响到对系统可预测性的评价.本文对现有 Cache 分析技术进行了综述,深入剖析了 Cache 分析的核心技术,并探讨了循环结构分析、数据 Cache 分析、多级 Cache 分析、多核共享 Cache 分析、非 LRU 替换策略分析等不同维度上的研究问题及主要挑战,总结了现有技术的优缺点,并展望了 Cache 分析未来的发展方向.期望能够为相关领域学者进行 Cache 分析方面的研究提供参考.

近 20 年来,面向 WCET 计算的 Cache 分析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:

- 多核共享 Cache 分析研究

如前所述,多核共享 Cache 分析的关键点是对核间冲突行为的有效抽象.自 Yan 等人首次提出多核共享 Cache 分析的问题以来,已经有许多研究组开展了这方面的研究工作.但是到目前为止,还没有一种适合这一问题的抽象数学模型出现.此外,由于这一问题存在众所周知的难度,通过设计方法(软件层面[54]和硬件层面[55])来减少核间干涉,从而简化分析提高性能的技术,也可能在未来得到更多关注.

- 非 LRU 替换策略的分析以及 Cache 可预测性的评价

尽管 Reineke,Grund,Guan 等人对 FIFO,MRU 等替换策略已经开展了一些分析工作,并提出了精确性很好的分析方法,目前的研究大多还仅限于对单级指令 Cache 的分析.由于程序对指令的访问具有明显的规律性,现有技术能否用于其他 Cache 特性(如数据 Cache、多级 Cache、共享 Cache 等)的分析并依然保证分析结果的精确性,还值得深入探讨.也只有在对上述 Cache 特性进行了全面的分析和探索之后,才能准确地评价不同替换策略的时间可预测性.非 LRU 替换策略的行为比 LRU 替换策略更加复杂,因此,新的分析技术对 Cache 访问行为的刻画需更具一般性.由于实际系统大多采用非 LRU 替换策略,这一方向的研究除了理论意义以外,还将具有很大的应用价值.

- Cache 分析与其他分析技术的结合

本文所综述的大多数分析技术都假定系统中只有 Cache,而不考虑诸如流水线、分支预测器、内存控制器等其他硬件体系结构.而实际系统中,分析程序的 WCET 则不可避免地要综合考虑这些因素以及它们之间的相互影响.以流水线为例,当系统采用乱序流水线时,在分析过程中可能出现时间异常(timing anomaly)[56].为了解决这一问题,可能需要更加复杂的分析方法.Chattopadhyay 等人已经在 Cache 分析与其他分析技术的结合上开展了一些工作[57],但这方面的研究还远远有待深入.当更多的处理器特性和更复杂的 Cache 特性结合到一起后,可能会产生更多的新问题.能否找到同时保证精确性和分析效率的分析方法,将直接决定 WCET 分析在实际系统中的应用.

References:

[1] Liu JWS. Real-Time Systems. Upper Saddle River: Prentice Hall, 2000.

[2] Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenstrom P. The worst-case execution time problem—Overview of methods and survey of tools. ACM Trans. on Embedded Computing Systems, 2008,7(3):1−53. [doi: 10.1145/1347375.1347389]

[3] Kirner R, Puschner P, Wenzel I. Measurement-Based worst-case execution time analysis. In: Proc. of the 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems. 2005. 7−10. [doi: 10.1109/SEUS.2005.12]

[4] Patterson DA, Hennessy JL. Computer Architecture: A Quantitative Approach. 5th ed., Burlington: Morgan Kaufmann Publishers, 2011.

[5] Li XF, Roychoudhury A, Mitra T. Modeling out-of-order processors for WCET analysis. Real-Time Systems, 2006,34(3):195−227. [doi: 10.1007/s11241-006-9205-5]

[6] Mitra T, Roychoudhury A. A framework to model branch prediction for WCET analysis. In: Proc. of the 2nd Workshop on Worst Case Execution Time Analysis. 2002. 1−4.

[7] Li YTS, Malik S. Performance analysis of embedded software using implicit path enumeration. In: Proc. of the 32nd ACM/IEEE Design Automation Conf. 1995. 456−461. [doi: 10.1145/217474.217570]

[8] Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: The MIT Press, 1999.

[9] Bengtsson J, Yi W. Timed automata: Semantics, algorithms and tools. Lecture Notes in Computer Science, 2004,3098:87−124. [doi: 10.1007/978-3-540-27755-2_3]

[10] Lü MS, Guan N, Deng QX, Yu G, Yi W. McAiT: A timing analyzer for multicore real-time software. In: Proc. of the 9th Int'l Conf. on Automated Technology for Verification and Analysis. Berlin: Springer-Verlag, 2011. 414−417. [doi: 10.1007/978-3-642-24372-1_29]

[11] Dalsgaard AE, Olesen MC, Toft M, Rydhof HR, Guldstrand LK. METAMOC: Modular execution time analysis using model checking. In: Proc. of the 10th Int'l Workshop on Worst-Case Execution Time Analysis. 2010. 113−123. [doi: 10.4230/OASIcs.WCET.2010.113]

[12] Gustavsson A, Ermedahl A, Lisper B, Pettersson P. Towards WCET analysis of multicore aarchitectures using UPPAAL. In: Proc. of the 10th Int'l Workshop on Worst-Case Execution Time Analysis. 2010. 101−112. [doi: 0.4230/OASIcs.WCET.2010.101]

[13] Larsen KG, Pettersson P, Yi W. UPPAAL in a nutshell. Int'l Journal on Software Tools for Technology Transfer, 1997,1(2): 134−152. [doi: 10.1007/s100090050010]

[14] Cousot P, Cousot R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages. 1997. 238−252. [doi: 10.1145/512950.512973]

[15] Ferdinand C. Cache behavior prediction for real-time systems [Ph.D. Thesis]. Saarbrücken: Saarland University, 1997.

[16] Theiling H, Ferdinand C, Wilhelm R. Fast and precise WCET prediction by separated cache and path analyses. Real-Time Systems, 2000,18(2-3):157−179. [doi: 10.1023/A:1008141130870]

[17] Cullmann C. Cache persistence analysis: A novel approach theory and practice. In: Proc. of the 2011 SIGPLAN/SIGBED Conf. on Languages, Compilers and Tools for Embedded Systems. 2011. 121−130. [doi: 10.1145/1967677.1967695]

[18] Huynh BK, Ju L, Roychoudhury A. Scope-Aware data cache analysis for WCET estimation. In: Proc. of the 17th IEEE Real-Time and Embedded Technology and Applications Symp. 2011. 203−212. [doi: 10.1109/RTAS.2011.27]

[19] Lü MS, Yi W, 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. 2010. 339−349. [doi: 10.1109/RTSS.2010.30]

[20] White RT, Healy CA, Whalley DB, Mueller F, Harmon MG. Timing analysis for data caches and set-associative caches. In: Proc. of the 3rd IEEE Real-Time Technology and Applications Symp. 1997. 192−202. [doi: 10.1109/RTTAS.1997.601358]

[21] Patil K, Seth K, Mueller F. Compositional static instruction cache simulation. In: Proc. of the 2004 ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems. 2004. 136−145. [doi: 10.1145/998300.997183]

[22] Li YTS, Malik S, Wolfe A. Performance estimation of embedded software with instruction cache modeling. ACM Trans. on Design Automation of Electronic Systems, 1999,4(3):257−279. [doi: 10.1145/315773.315778]

[23] Li YTS, Malik S, Wolfe A. Cache modeling for real-time software: Beyond direct mapped instruction caches. In: Proc. of the 17th IEEE Real-Time Systems Symp. 1996. 254−263. [doi: 10.1109/REAL.1996.563722]

[24] Martin F, Alt M, Wilhelm R, Ferdinand C. Analysis of loops. In: Proc. of the 7th Int'l Conf. on Compiler Construction. 1998. 80−94.

[25] Ballabriga C, Casse H. Improving the first-miss computation in set-associative instruction caches. In: Proc. of the 2008 Euromicro Conf. on Real-Time Systems. 2008. 341−350. [doi: 10.1109/ECRTS.2008.34]

[26] White RT, Mueller F, Healy CA, Whalley DB, Harmon MG. Timing analysis for data and wrap-around fill caches. Real-Time Systems, 1999,17(2-3):209−233. [doi: 10.1023/A:1008190423977]

[27] Balakrishnan G, Reps T. Analyzing memory accesses in x86 executables. In: Proc. of 13th Int'l Conf. on Compiler Constrution. 2004. 5−23. [doi: 10.1007/978-3-540-24723-4_2]

[28] Ferdinand C, Wilhelm R. On predicting data cache behavior for real-time systems. In: Proc. of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems. 1998. 16−30. [doi: 10.1007/BFb0057777]

[29] Sen R, Srikant YN. WCET estimation for executables in the presence of data caches. In: Proc. of the 7th ACM & IEEE Int'l Conf. on Embedded Software. 2007. 203−212. [doi: 10.1145/1289927.1289960]

[30] Staschulat J, Ernst R. Worst case timing analysis of input dependent data cache behavior. In: Proc. of the 18th Euromicro Conf. on Real-Time Systems. 2006. 227−236. [doi: 10.1109/ECRTS.2006.33]

[31] Sondag T, Rajan H. A more precise abstract domain for multi-level caches for tighter WCET analysis. In: Proc. of the 31st IEEE Real-Time Systems Symp. 2011. 395−404. [doi: 10.1109/RTSS.2010.8]

[32] Mueller F. Timing predictions for multi-level caches. In: Proc. of the ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems. 1997. 29−36.

[33] Hardy D, Puaut I. WCET analysis of multi-level non-inclusive set-associative instruction caches. In: Proc. of the 29th Real-Time Systems Symp. 2008. 456−466. [doi: 10.1109/RTSS.2008.10]

[34] Lesage B, Hardy D, Puaut I. WCET analysis of multi-level set-associative data caches. In: Proc. of the 9th Int'l Workshop on Worst-Case Execution Time Analysis. 2009.

[35] Chattopadhyay S, Roychoudhury A. Unified cache modeling for WCET analysis and layout optimizations. In: Proc. of the 30th IEEE Real-Time Systems Symp. 2009. 47−56. [doi: 10.1109/RTSS.2009.20]

[36] Borkar S, Jouppi NP, Stenstrom P. Microprocessors in the era of terascale integration. In: Proc. of the Conf. on Design, Automation and Test in Europe. 2007. 237−242. [doi: 10.1145/1266366.1266417]

[37] Wu L, Zhang W. A model checking based approach to bounding worst-case execution time for multicore processors. ACM Trans. on Embedded Computing Systems, 2012,11(S2):56−75. [doi: 10.1145/2331147.2331166]

[38] 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. 2008. 80−89. [doi: 10.1109/RTAS.2008.6]

[39] Liang Y, Ding HP, Mitra T, Roychoudhury A, Li Y, Suhendra V. Timing analysis of concurrent programs running on shared cache multi-cores. In: Proc. of the 30th IEEE Real-Time Systems Symp. 2012. 638−680. [doi: 10.1007/s11241-012-9160-2]

[40] Chen FY, Zhang DS, Wang ZY. Inter-Thread inteference analysis for real-time WCET estimations of multi-core architectures. Acta Electronica Sinica, 2012,40(7):1372−1378 (in Chinese with English abstract).

[41] Chattopadhyay S, Roychoudhury A. Scalable and precise refinement of cache timing analysis via model checking. In: Proc. of the 32nd IEEE Real-Time Systems Symp. 2011. 193−203. [doi: 10.1109/RTSS.2011.25]

[42] Banerjee A, Chattopadhyay S, Roychoudhury A. Precise micro-architectural modeling for WCET analysis via AI+SAT. In: Proc. of the 19th IEEE Real-Time and Embedded Technology and Applications Symp. 2013. 87−96. [doi: 10.1109/RTAS.2013.6531082]

[43] 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. 2009. 89−102. [doi: 10.1145/1519065.1519076]

[44] Vera X, Lisper B, Xue JL. Data cache locking for tight timing calculations. ACM Trans. on Embedded Computing Systems, 2007, 7(1):4−42. [doi: 10.1145/1324969.1324973]

[45] Ding HP, Liang Y, Mitra T. Integrated instruction cache analysis and locking in multitasking real-time systems. In: Proc. of the 50th ACM/IEEE Design Automation Conf. 2013. 1−10.

[46] 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. 2009. 68−77. [doi: 10.1109/RTSS.2009.34]

[47] Al-Zoubi H, Milenkovic A, Milenkovic M. Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite. In: Proc. of the 42nd Annual Southeast Regional Conf. 2004. 267−272. [doi: 10.1145/986537.986601]

[48] Reineke J, Grund D. Relative competitive analysis of cache replacement policies. In: Proc. of the ACM SIGPLAN-SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems. 2008. 51−60. [doi: 10.1145/1375657.1375665]

[49] Grund D, Reineke J. Precise and efficient FIFO-replacement analysis based on static phase detection. In: Proc. of the 22nd Euromicro Conf. on Real-Time Systems. 2010. 155−164. [doi: 10.1109/ECRTS.2010.8]

[50] Guan N, Yang XP, Lü MS, Yi W. FIFO cache analysis for WCET estimation: A quantitative approach. In: Proc. of the Design, Automation and Test in Europe. 2013. 296−301. [doi: 10.7873/DATE.2013.073]

[51] Guan N, Lü MS, Yi W, Yu G. WCET analysis with MRU caches: Challenging LRU for predictability. In: Proc. of the 18th IEEE Real Time and Embedded Technology and Applications Symp. 2012. 55−64. [doi: 10.1109/RTAS.2012.31]

[52] Falk H, Girault A, Grund D, Guan N, Jonsson B, Marwedel P, Reineke J, Rochange C, Hanxleden RV, Wilhelm R, Yi W. Building timing predictable embedded systems. ACM Trans. on Embedded Computing Systems, 2012. http://www.artist-embedded.org/docs/Events/2012/ArtistDesign_Y4_Review/Deliverables/predictability_survey.pdf

[53] Reineke J, Grund D, Berg C, Wilhelm R. Timing predictability of cache replacement policies. Real-Time Systems, 2007,37(2): 99−122. [doi: 10.1007/s11241-007-9032-3]

[54] Ding YQ, Zhang W. Multicore-Aware code co-positioning to reduce WCET on dual-core processors with shared instruction caches. Journal of Computing Science and Engineering, 2012,6(1):12−25. [doi: 10.5626/JCSE.2012.6.1.12]

[55] Paolieri M, Quinones E, Cazorla FJ, Bernat G, Valero M. Hardware support for WCET analysis of hard real-time multicore systems. In: Proc. of the 36th Annual Int'l Symp. on Computer Architecture. 2009. 57−68. [doi: 10.1145/1555754.1555764]

[56] Reineke J, Wachter B, Thesing S, Wilhelm R, Polian I, Eisinger J, Becker B. A definition and classification of timing anomalies. In: Proc. of the 6th Int'l Workshop on Worst-Case Execution Time Analysis. 2006. [doi: 10.4230/OASIcs.WCET.2006.671]

[57] Chattopadhyay S, Kee CL, Roychoudhury A, Kelter T, Marwedel P, Falk H. A unified WCET analysis framework for multi-core platforms. In: Proc. of the 18th IEEE Real Time and Embedded Technology and Applications Symp. 2012. 99−108. [doi: 10.1109/RTAS.2012.26]

附中文参考文献:

[40] 陈芳园,张冬松,王志英.多核实时线程间干扰分析及 WCET 估值.电子学报,2012,40(7):1372−1378.

吕鸣松(1980－),男,辽宁沈阳人,博士,讲师,CCF 会员,主要研究领域为实时系统时间分析,多核嵌入式系统设计与验证.
E-mail: lvmingsong@ise.neu.edu.cn

关楠(1981－),男,博士,副教授,CCF 会员,主要研究领域为多核实时调度,实时系统时间分析,多核嵌入式系统设计与验证.
E-mail: guannan@ise.neu.edu.cn

王义(1961－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为模型检测,形式化方法,多核计算机系统,嵌入式系统设计与验证.
E-mail: wangyi@ise.neu.edu.cn