软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(6):1390−1402 [doi: 10.3724/SP.J.1001.2013.04392]

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

多核处理器并行程序的确定性重放研究*

高 岚[1], 王 锐[1,2], 钱德沛[1]

[1](北京航空航天大学 计算机学院,北京 100191)

[2](北京市网络技术重点实验室 北京 100191)

通讯作者: 王锐, E-mail: rui.wang@jsi.buaa.edu.cn

摘 要: 多核处理器并行程序的确定性重放是实现并行程序调试的有效手段,对并行编程有重要意义.但由于多核架构下存在共享访存不同步问题,并行程序确定性重放的研究依然面临多方面的挑战,给并行程序的调试带来很大困难,严重影响了多核架构下并行程序的普及和发展.分析了多核处理器造成并行程序确定性重放难以实现的关键因素,总结了确定性重放的评价指标,综述了近年来学术界对并行程序确定性重放的研究.根据总结的评价指标,从纯软件方式和硬件支持方式对目前的确定性重放方法进行了分析与对比,并在此基础上对多核架构下并行程序的确定性重放未来的研究趋势和应用前景进行了展望.

关键词: 多核处理器;并行程序;确定性重放;多线程程序;数据竞争;调试

中图法分类号: TP368 **文献标识码**: A

中文引用格式: 高岚,王锐,钱德沛.多核处理器并行程序的确定性重放研究.软件学报,2013,24(6):1390−1402. http://www.jos.org.cn/1000-9825/4392.htm

英文引用格式: Gao L, Wang R, Qian DP. Deterministic replay for parallel programs in multi-core processors. Ruan Jian Xue Bao/Journal of Software, 2013,24(6):1390−1402 (in Chinese). http://www.jos.org.cn/1000-9825/4392.htm

Deterministic Replay for Parallel Programs in Multi-Core Processors

GAO Lan[1], WANG Rui[1,2], QIAN De-Pei[1]

[1](Department of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

[2](Beijing Key Laboratory of Network Technology, Beijing 100191, China)

Corresponding author: WANG Rui, E-mail: rui.wang@jsi.buaa.edu.cn

Abstract: The deterministic replay for parallel programs in multi-core processor systems is important for the debugging and dissemination of parallel programs, however, due to the difficulty in tackling unsynchronized accessing of shared memory in multiprocessors, industrial-level deterministic replay for parallel programs have not emerged yet. This paper analyzes non-deterministic events in multi-core processor systems and summarizes metrics of deterministic replay schemes. After studying the research for deterministic multi-core processor replay in recent years, this paper introduces the proposed deterministic replay schemes for parallel programs in multi-core processor systems, investigates characteristics of software-pure and hardware-assisted deterministic replay schemes, analyzes current researches and gives the prospects of deterministic replay for parallel programs in multi-core processor systems.

Key words: multi-core processor; parallel program; deterministic replay; multithread; data race; debugging

作为目前提高处理器性能的主要技术,在单个芯片上集成多个处理器核形成多核处理器(multi-core processor)已成为芯片工业的发展趋势[1].在这种架构下,硬件能力的提高是否可以转变成程序性能的提升,成为多核时代所面临的最严峻的挑战之一.与串行程序相比,并行程序由于能够更充分地利用多核处理器的资源,因

* 基金项目: 国家自然科学基金(61073011, 61133004); 国家高技术研究发展计划(863)(2012AA010902)
收稿时间: 2012-07-01; 定稿时间: 2013-02-26

此具有更大的潜力以匹配硬件资源的提升.然而面对多核架构,并行程序的开发和部署面临许多困难,而并行调试正是其中最突出的一个.与单核架构下的传统并行程序相比,在多核架构中,造成并行程序执行过程与结果不确定的因素更加普遍,导致其执行过程的重现更加困难,给调试工作带来了巨大挑战,成为多核架构下并行程序发展和普及的障碍.由于并行程序的确定性重放(deterministic replay)是实现调试的有效手段,所以近年来对该问题的研究受到学术界的广泛关注,ISCA,HPCA,Micro 和 ASPLOS 等体系结构研究领域的著名会议也出现了大量关于确定性重放的论文.除了并行程序调试以外,确定性重放技术还被应用于并行安全性和可靠性检查[2]、性能预测[3]等领域.

确定性重放的基本思想是,在程序首次或者某次执行中记录不确定性因素,而后在程序其他执行过程中索引这些记录,并强制其按照相同的方式或顺序执行,以确保并行程序的每次运行都与其首次或者某次运行相一致.因此,确定性重放将程序的执行分成两个阶段:记录阶段(recorded run)和重放阶段(replay run).其中,记录阶段是作为程序运行基准的某次运行,而程序后续的确定性执行则称为重放阶段.

确定性重放实现的关键,即是对不确定性因素的记录.然而,与单核架构下导致并行程序不确定的因素不同,多核架构中造成并行程序不确定的因素众多,而且某些因素出现的频率非常高,给确定性重放的实现带来了很多困难.针对这些问题,Respec[4],Rerun[5],Delorean[6]和 Lreplay[7]等研究人员从不同角度、采用不同方式研究了并行程序的确定性重放问题.这些方法虽然取得了一定进展,但也存在一系列问题,它们或者实现确定性重放的效率较低,或者只能实现部分确定性重放,或者对硬件设计有新的要求,目前还无法在实际处理器上应用.

总体而言,多核架构下并行程序的确定性重放仍然有很多问题亟待解决.本文对多核处理器造成并行程序不确定的关键因素进行了分析,总结了多核架构下确定性重放的评价指标,并以此为依据,分别对纯软件和有硬件支持的实现方法进行了详细介绍,剖析了国内外多核架构下并行程序确定性重放的研究现状.基于这些讨论,展望了多核架构下并行程序确定性重放未来的研究趋势和发展前景.

本文在第 1 节对多核架构下导致确定性重放难以实现的根本原因进行了分析.第 2 节~第 4 节分别总结提出了确定性重放程度、评价指标和实现方法分类.在此基础上,第 5 节对当前国内外对多核架构下确定性重放的实现方法和研究现状进行了综述,并在第 6 节对多核架构下并行程序确定性重放研究趋势进行了展望.

1 确定性重放实现障碍

确定性重放是在记录阶段对并行程序的不确定性因素进行记录,而后在重放阶段,根据记录的不确定性信息来保证并行程序能够确定性地重新执行.其关键在于设计某种方式,以简洁有效地记录不确定性因素.并行程序的不确定性因素在单核处理器中已经存在,比如并行程序在执行过程中,多个并行任务会被动态调度执行,并行任务间的交错运行顺序和任务执行调度也都是动态的,这些都会使得并行程序的执行过程难以被重现.然而,随着多核架构的出现,共享存储器的存在使得并行程序不确定的因素进一步增多,且出现频率大大增加,给确定性重放带来更大困难.本节在简单介绍传统多核并行程序不确定性因素的基础上,对多核架构新引入的不确定性因素——存储器竞争进行了详细的分析,并指出其导致多核架构下并行程序确定性重放难以实现的原因.

1.1 单核架构并行程序不确定性因素概述

在传统的单核架构下,造成并行程序不确定的因素可以概括为硬件和软件两个方面.在硬件方面,不确定性因素包括中断和陷阱、DMA fills、cache 的状态、预测表、总线优先级控制器以及其他微体系结构;在软件方面,线程间资源的竞争、内存页状态、操作系统中全局数据结构的状态、线程的调度策略、部分系统调用(比如 random 等)、访存的重新排序以及指令执行时间的差异等也会导致并行程序的不确定性.然而,在单核中造成并行程序不确定的因素虽然很多,但出现频率低,甚至有些不会在所有并行程序中出现,比如某些造成不确定的系统调用等.由于记录这些不确定性因素不会引起过大的开销,使得并行程序的确定性重放易于实现.目前,ReVirt[8]和 ReTrace[9]已经能够较好地支持单核架构下并行程序的确定性重放.

1.2 存储器竞争

图 1 所示为典型的多核架构,在一个芯片上集成多个处理器核,其中,这些处理器核与之前单核架构中的处理器相同;多个处理器核间共享 L2 cache,LLC(Last level cache)和主存.在这种架构下,并行程序会对共享存储器中的数据进行同步访问,通过共享存储器实现线程间的交互通信,协同工作.

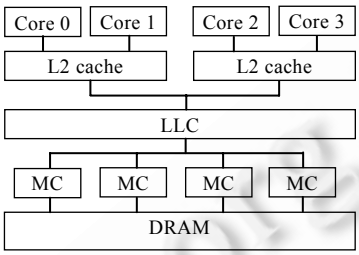


Fig.1 Multi-Core architecture

图 1 多核架构示意图

与单核架构相比,多核架构中共享存储器的存在导致了新的不确定性因素的产生.在多核架构下,并行程序的执行需要额外考虑对共享存储器的同步访问问题.因此,多核架构下并行程序的不确定性因素除需要考虑传统单核架构下不确定性因素外,如何对共享存储器进行访存同步是一个新的问题,同时也是使得多核架构下并行程序确定性重放难以实现的最大障碍和关键原因.由于多核架构中对共享存储器的访问频率很高,记录这些信息会极大地增加时间开销,占用过多的存储空间并严重减缓程序的执行速度,使得多核架构下并行程序的确定性重放问题尚未彻底解决.

不确定性因素直接影响确定性重放的设计和实现.Netzer 和 Miller 在文献[10]中对共享存储器的不同步访问进行了详细的剖析.在此基础上,近年来学术界使用存储器竞争(memory race)来表示由于共享存储器的存在而导致的线程间不同步访问,其定义是,访存指令 *i* 和 *j* 之间存在存储器竞争当且仅当:

(1) 访存指令 *i* 和 *j* 之间存在冲突(conflict);

(2) 访存指令 *i* 和 *j* 来自不同线程;

(3) 当程序执行过程满足顺序一致性(sequentially consistent execution)时,根据程序执行顺序无法确定指令 *i* 和 *j* 的交错执行顺序.

其中,访存指令 *i* 和 *j* 之间存在冲突是指 *i* 和 *j* 访问的为共享存储器中的同一地址,且至少有一条为写指令.

由于存储器竞争在多核架构下的并行程序中出现频率极高,成为了确定性重放难以实现的主要原因,也使得如何在记录阶段中简洁有效地记录存储器竞争从而降低记录时空开销成为了确定性重放的关键,近年来,学术界对于确定性重放的研究也主要集中于对存储器竞争的记录.

根据存储器竞争在程序代码中表现形式的不同,可将其分为同步(synchronization)和数据竞争(data race)两种.如图 2(a)所示,同步的出现会伴随有同步操作语句,比如锁、信号量等,其中,同步操作包括编程模型提供的标准同步操作语句和原子操作等;而数据竞争则没有显式或标准的表现形式,难以识别和发现,如图 2(b)所示.

1.2.1 同 步

同步会导致并行程序的不确定性,但正确的同步/互斥操作不会产生错误,这是程序员为了提高程序的性能或者为了平衡系统负载而有意设计的,比如不同线程对于临界区资源的竞争等,属于并行程序本身的特点和性质.记录同步操作能够保证同步部分的确定性重放[11],但由于并行程序中可能包含上百甚至几千条同步操作,使得单纯的记录会引起巨大开销.

同步在多核架构并行程序中出现频率相对较高,但易于检测和记录,虽然会给确定性重放的实现造成较大的时空开销,但可以通过优化软件实现等方法来提高系统性能.

```
P1                  P2
global M, N         global M, N
M=3                 M=3
N=6                 N=6
LOCK                LOCK
If (M>N)            If (M>N)
    M=M−N               M=M/N
else                else
    M=M+N               M=M*N
UNLOCK              UNLOCK
```

(a)

```
P1                  P2
global M            global M
M=1                 M=1
While (TRUE){       While (TRUE){
  If (M==0)           If (M==0)
    BREAK               BREAK
  M−−                 M−−
}                   }
```

(b)

Fig.2 Synchronization and data race

图 2 同步和数据竞争

1.2.2 数据竞争

与同步相比,数据竞争更为复杂.虽然没有显式或标准的同步操作,但并不是所有的数据竞争都会造成程序错误.根据数据竞争是否会引起程序错误,可将其分为有害数据竞争(harmful data race)和良性数据竞争(benign data race).有害数据竞争源于程序互斥/同步设置不当,直接影响并行程序执行的正确性,因而也称为真数据竞争(true data race),是并行程序调试的真正目标.如图 2(b)所示,数据竞争可能会使得 *P*1 或 *P*2 陷入 while 死循环,导致程序出错;与其相对,良性数据竞争也称为伪数据竞争(false data race),它不会引起程序错误,比如程序员有时会自己定义同步操作来实现线程间对共享存储器的同步访问,虽然没有标准的同步操作语句,但其功能相似,且不会造成程序错误.

虽然数据竞争在程序中出现频率相对较低,但其检测属于 NP 问题,检测和记录的实现复杂度高,对系统性能影响极大,且无法保证查找方法的完备和正确.鉴于此,目前的确定性重放方案根据其面向的应用不同,或者只实现包含同步的并行程序的确定性重放,或者通过专门硬件将所有存储器竞争不加区分地全部记录,或者只集中于对有害数据竞争的区分和检测,使得无法有效地解决因数据竞争而导致的多核架构下并行程序的不确定性.

2 确定性重放程度分类

确定性重放旨在使程序的每次执行与之前的执行相一致,然而这里所谓的确定性是一个笼统的概念,且有程度之分.鉴于不同的确定性对于确定性重放方案的设计及其应用范围有重要的影响,本节归类说明不同程度的确定性.

(1) 理想确定性

保证重放阶段中所有线程每条指令的执行顺序及线程间指令的执行顺序与记录阶段中的执行顺序严格一致,且其输入时间、中断产生时间和响应时间等也与记录阶段相同.它的确定性程度最高,也最为理想,但需记录大量信息,实现难度较高.除此之外,就目前而言,这种确定性程度对相关应用来说没有必要.因而,对于理想确定性程度的实现研究较少,有代表性的研究有文献[12].

(2) 严格确定性

保证重放阶段中所有线程每条指令的执行顺序与记录阶段中执行顺序相一致,且每个线程访问共享存储器所得到的结果与记录阶段相同.与理想确定性相比,它无需考虑除访存指令之外的其他指令的执行顺序及每条指令的执行时间,而只需保证与访问共享存储器相关的指令的执行顺序即可.由于严格确定性足以支持所有并行程序的确定性重放,满足应用需求,所以可认为是实际情况中确定性程度最高的确定性重放.在多核架构下,这种确定性重放在单核架构已实现确定性重放的前提下只要求线程间存在相互依赖的访存指令的顺序相同,即只要求重放阶段的存储器竞争顺序与记录阶段相一致.由于可以从指令层级有效地记录存储器竞争的顺序,一般有专门硬件支持的确定性重放方法都实现这种确定性,比如 Delorean[6],Lreplay[7],Strata[13],FDR[14],RTR[15]等,除此之外,纯软件实现方法[16]也支持这种确定性.

(3) 非严格确定性

理想确定性实现难度较大且没有重要的意义,严格确定性基本上能够满足绝大部分应用程序的需求,但对于纯软件实现方式来说仍会产生巨大的时间和空间开销.为了减少时空开销,研究者对严格确定性程度进行了不同方面的削弱,实现非严格确定性.具体包括 3 种:第 1 种是外部确定性,只保证重放阶段中外部可见的状态与记录阶段相同,要求重放阶段的寄存器文件状态、程序输出、系统调用的顺序及传递的参数应与记录阶段相一致,但并不要求线程间指令交错执行的顺序一致,而只保证程序运行中的外部可见状态相同,比如 ResPec[4];第 2 种是部分确定性,保证程序的一部分能够被确定性地重新执行,而对于其他部分则无法实现确定性重放,比如 PRES[17]只确定性重新执行那些与程序 bug 相关的代码部分,而 Kendo[18]等只对程序中的同步代码进行确定性重放;第 3 种是输出确定性,只保证重放阶段最终的输出结果与记录阶段相同.由于只要求输出结果相同,对某些线程间指令的交错运行顺序以及其他程序状态可以不予考虑和检测,因而实现较容易,比如 ODR[19].

3 确定性重放评价指标

确定性重放应用广泛,实现机制多样,不同应用侧重点不同,针对的评价指标也不同.比如:Montesinos 等人在 Delorean[6]和 Capo[20]中分别从确定性重放的效率和应用两方面评价确定性重放方法;Lreplay 以并行程序的调试为目标,将 DFD(design for debugging)作为评价指标.本节综合考虑,总结提出多核架构下并行程序确定性重放的评价指标.

(1) 记录阶段时间开销

即确定性重放方法对记录阶段执行速度的影响.程序执行时间为 T_E,而作为确定性重放的记录阶段时执行时间为 T_D,则 T_D/T_E 即为确定重放记录阶段时间开销,其值越大,时间开销越大.

有硬件支持的确定性重放可以简单采用该值作为记录阶段的时间开销,但在纯软件实现方法中,部分实现方法或者通过增加重放次数[17]或者采用重放失败后回滚的方法[4],无法单纯地采用该值来进行测量,而需要对重放次数和回滚次数进行统计,来说明系统时间开销.

在程序运行过程中,对于不确定性因素的记录应当不影响程序原本的运行速度,记录阶段应与程序原本的运行速度相一致.目前为止,对于有硬件支持的确定性重放的实现,记录阶段时间开销几乎可以忽略;但是对于纯软件的实现方法来说,记录阶段时间开销仍然较大.

(2) 空间开销

空间开销是指产生的日志文件所占空间的大小.空间开销越小越好,特别是对于有专门硬件支持的确定性重放来说,日志空间较小对工业成本及确定性重放所能支持的时间长度有重要意义.

对于纯软件实现方式,空间开销主要集中在其所生成的日志文件,因而可将日志文件的大小作为衡量空间开销的标准.但在一般情况下,由于在软件层面,存储空间并不受限,日志文件的大小不会影响系统及确定性重放方案的性能,因而不会将纯软件实现方式的日志文件大小作为主要评价指标.

与此不同,对于有硬件支持的实现方式来说,确定性重放方案生成的日志文件一般存储在添加的硬件寄存器中,因而日志文件的大小会严重影响到其系统性能及实现成本,是评价有硬件支持的实现方式优劣的主要指标,其空间开销通过添加的寄存器文件大小进行衡量.

(3) 专用硬件支持开销

专用硬件支持是针对有硬件支持的实现方式的评价指标,添加的多少对工业成本及方案的普及有重要影响,因而专用硬件添加的越少越好.在保证确定性重放性能的前提下,最好是能够不添加额外硬件;如果必须添加的话,鉴于工业成本及设计复杂度,添加的额外硬件应尽可能小而且不影响当前计算机中已经存在的体系架构,特别是某些经典架构,比如 cache 一致性协议等.

专用硬件支持的开销通过其占用的芯片面积来衡量,也可通过其增加的晶体管和逻辑单元的数量进行衡量,但同时也要考虑其是单纯的添加还是需要对当前的体系架构进行修改.

(4) 重放阶段执行速度

重放阶段执行速度应与记录阶段的执行速度相同或者相近,特别是对于面向在线应用的确定性重放方案,比如容错[4]等,重放阶段的执行速度对系统性能有很大影响.不仅如此,重放阶段的执行速度也会对用户体验造成影响.

重放阶段的执行速度通过重放阶段的执行时间进行统计.但到目前为止,该项评价指标并未被研究者广泛重视,而是主要关注如何实现不确定性因素的高效记录.由于在保证确定性的前提下,采用多线程方式执行重放阶段难度较大,多数当前已实现的确定性重放方案使用单线程的执行方式来执行重放阶段.当重放片段较小时,重放阶段执行速度对于系统性能影响不大;但当重放片段较大时,较慢的重放速度仍然会对用户体验造成影响,因而认为在后续的研究和实现中应引起重视.

(5) 存储器模型

为了增加其适用范围,确定性重放方案应能支持多种存储模型,比如 SC(sequential consistent),TSO(total store order)等.

(6) 迁移性和扩展性

应具有较好的迁移性和扩展性,包括支持多种软件环境、操作系统类型,且对于多核架构中核的数量有良好的扩展性.

(7) 确定性级别

确定性程度对其应用范围和应用效果有重要影响,确定性越好,应用越广泛,效果也就越好.

综上所述,由于不同的确定性重放所面向的具体应用不同,对上述评价指标的侧重点及具体要求也不尽相同,不存在完全定量而统一的标准.

4 确定性重放方法分类

确定性重放的研究有多种分类方式.首先,根据实现方式的不同可将确定性重放方法分成纯软件和有硬件支持两大类.纯软件的实现方式通过在编译器和操作系统中插入特定的代码来记录逻辑序,通常会带来极大的性能损失,时间开销大;而有硬件支持的实现方式通过加入专门的调试硬件来记录逻辑序,虽然不会带来很大的性能损失,但通常会导致较大的日志,空间开销大.其次,根据重放阶段是否与记录阶段同时执行,可将确定性重放方法分成在线和不在线两种.其中,在线确定性重放的重放阶段与记录阶段同时执行,主要面向容错等应用,比如 Respec[4];而不在线确定性重放的重放阶段无需与记录阶段同时执行,主要面向调试等应用,比如 Rerun[5],BugNet[21]等.根据确定性重放所面向的对象,可将其分成全系统和程序确定性重放.前者对所有执行的程序,包括系统调用、中断等都进行确定性重放[14];而后者只确定性的重放目标程序,比如 BugNet[21],RTR[15]等.除此之外,根据确定性重放所支持的确定性也可以将其分成不同的类别.

总体而言,目前绝大多数确定性重放方法的理论基础为逻辑序理论.其中,逻辑序是Lamport在1978年提出,用来刻画缺乏全局时钟的分布式系统中事件的顺序关系[22].在逻辑序理论的指导下,存储器竞争通过多个处理器核访存事件的逻辑序关系表示和索引,确定性重放方法或者通过在硬件层级记录核间访存指令的逻辑先后顺序,或者通过在软件层级记录同步操作的逻辑顺序来确保并行程序执行过程的重现.已提出的绝大多数确定性重放方法都基于这一理论,比如 FDR[14],RTR[15],Delorean[6],RecPlay[11]等.这些方法并不记录处理器核中指令执行的绝对时间,而是在逻辑序理论的基础上,采用不同的算法记录单个处理器核访存事件或者处理器核访存事件块的相对顺序,而后在重放阶段中重新索引这些信息,保证其逻辑序关系不变,从而实现确定性重放.由于片上多核处理器每秒钟都可能发生上百亿个逻辑事件,随之产生的信息量可以达到数十千兆每秒,将其完整记录下来根本不可能实现.所以,基于这一理论的确定性重放方案往往采用各种方式来减少所需记录的逻辑序,比如设计某些算法以通过记录部分逻辑关系来推导其他逻辑序,或者通过降低确定性程度来减少记录信息,或者通过对程序或者指令进行分块从而通过记录块之间的逻辑序关系来减少记录信息等.

为了简化对时间序的记录,Chen 等人提出了全局时钟理论[23],并将其应用于确定性重放的研究[7].全局时钟是指可以用其来度量片上多核处理器内所有事件先后的时钟.在全局时钟理论的基础上,以未决时间区间

(pending period)来代表访存操作的完成时间,并以此作为访存事件逻辑序的比较依据.

在对确定性重放的研究中,是否有硬件支持极大地影响着确定性重放方案的实现方式、所需考虑的问题、确定性等确定性重放的诸多性质,是确定性重放方案最重要的分类方式.本文将以这种分类方式分别对纯软件和有硬件支持的确定性重放方法进行详细的介绍和分析.

5 确定性重放研究分析

根据上述程度分类和评价指标,本节从纯软件和有硬件支持两种实现方式对当前国内外的实现方法和研究进程进行综述分析,对理解多核架构并行程序确定性重放研究现状及把握未来实现方法有重要意义.

5.1 纯软件确定性重放方法

鉴于专门硬件的添加会对现有体系架构进行修改,难以迅速在实际系统中应用和推广,很多研究采用纯软件的方式,力求在不改变现有多核架构的基础上实现并行程序的确定性重放.这种纯软件方式通过在编译器和操作系统中插入特定的代码来记录逻辑顺序,从而记录存储器竞争,以实现确定性重放.由于缺少专门硬件的支持,且存储器竞争出现频率很高,纯软件实现方法通常会造成极大的性能损失,记录时间开销过大是其面临的主要问题.为此,很多纯软件确定性重放方法从不同的编程模型和软件纵向层次结构出发,通过不同的记录方式,并采用改变确定性等方式来减少时间开销.

Instant Replay[16]是第 1 个并行程序的确定性重放方案,它采用纯软件方式实现,通过记录所有指令对共享存储器的访问顺序来保证确定性重放.后续的 SMP-ReVirt[24]和 PinSel[25]也通过记录访存顺序实现确定性重放.除此之外,微软的 iDNA[26]通过记录共享访存的值来实现确定性重放.

与此相对,为了进一步降低时间开销,其他纯软件实现方式的基本思想为折衷确定性和时间开销,以确定性的降低为代价来减小时间开销,有代表性的研究包括 Respec[4],PRES[17],Kendo[18],ODR[19]以及 RecPlay[11]等.

Respec 选择性地记录同步操作的顺序和系统调用顺序,通过回滚机制来确保重放阶段和记录阶段的一致性.它主要通过实现外部确定性和预测记录阶段两种方式来减少信息的记录,降低时间开销.对于两个线程,Respec 的运行时间开销增加 18%;4 个线程时增加 55%,扩展性较差.

PRES[17]通过降低确定性程度,并采用增加重放次数的方法来减少所需记录的信息,降低时间开销,将以前记录阶段中全部的开销分散成多次以使用户体验较好.

与 PRES 减弱确定性以提升记录性能相似,RecPlay[11]和 JaRec[27]只记录同步操作及其执行顺序,因此只能保证第 1 个数据竞争出现之前的确定性重放;ODR[19]选择性的记录部分信息(调度、程序输入、读序列中的一部分),然后通过搜索程序执行空间来实现确定性重放,但只保证输出确定性.

以上实现方式集中于对不确定信息的记录,与此不同,Kendo[18]引入确定性逻辑时间的概念,通过设计调度算法来确保并行程序执行时线程调度顺序的一致性,但只保证部分确定性重放.Kendo 的最大不足在于程序的确定性依赖于程序编写正确的假设,编译和运行时并没有提供任何保障,对于不是基于锁机制的程序以及存在数据竞争的程序无法实现确定性重放.

表 1 列出了已提出的主要纯软件方式的性能.相对于有硬件支持的确定性重放实现方式,其最大的优点是无需专门硬件的支持,有利于确定性重放技术的推广应用.但纯软件的实现方式目前还无法从根本上有效地解决多核架构下包括数据竞争在内的并行程序的确定性重放,时间开销过大是它们的最大缺陷.为了降低时间开销,纯软件方法采用了各种方式,比如降低确定性、增加重放阶段的次数等,但这些折衷的办法也在一定程度上限制了确定性重放的应用范围.除此之外,纯软件实现方式一般局限于某类特定的编程环境或者软件系统,迁移性较差.

5.2 有硬件支持的确定性重放方法

为了从根本上解决纯软件方案时间开销过大的问题,Bacon 和 Goldstein 提出了第 1 个有硬件支持的并行程序确定性重放方法[28],通过专门的硬件有效地实现处理器架构下并行程序的确定性重放.随着多核处理器的发

展,很多研究将这种思想应用到了多核处理器中,代表性的方法包括 FDR[14],RTR[15],BugNet[21],Rerun[5],DMP[29],CORD[30],ReEnact[31],Rcdc[32],Calvin[33],Timetraveler[34]等.

Table 1 Summary of characteristics of main software-pure schemes

表 1 主要纯软件确定性重放汇总

	记录阶段性能损失	重放阶段性能损失	迁移性和扩展性	确定性
Respec[4]	两线程 18%,四线程为 55%	两线程 18%,四线程 55%	在 Linux 中实现,线程数扩展性较差	外部确定性
ODR[19]	60%	20%~60%	以中间件形式实现,扩展性较好	输出确定性
Kendo[18]	16%	N/A	基于锁机制的程序,扩展性较好	部分确定性
RecPlay[11]	91%	N/A	Solaris 操作系统,扩展性较好	部分确定性
PRES[17]	7%~84%	N/A	在 Linux 中实现,扩展性较好	部分确定性

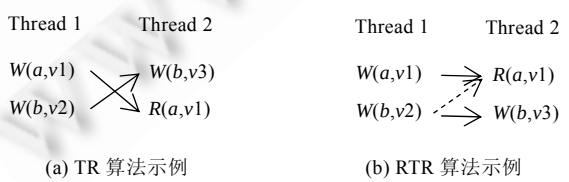
与纯软件实现方式相比,有硬件支持的确定性重放方案通过专门硬件记录逻辑序,实现对存储器竞争的记录,能够极大地降低时间开销,但通常会导致较大的日志,空间开销大.因此,如何减少日志体积是硬件方式所要解决的关键问题.除此之外,对于专门硬件的添加,特别是一些有硬件支持的确定性重放方案[14]对目前已有的经典体系架构修改较大,也为其在工业生产中的应用和推广造成很大障碍.

当前,绝大多数有硬件支持的确定性重放方案都是基于逻辑序的理论基础,集中于对访存指令逻辑序记录的研究,采用不同的机制和方式在保证确定性重放的基础上减少所需记录的线程间交错执行指令的数量,同时尽量减少对现有体系架构的修改.最初的确定性重放方案记录所有访存指令的逻辑序,一般基于多核的监听总线(snooping bus)架构,通过监听总线上的一致性消息来对数据竞争进行记录.但由于需要记录总线上所有的一致性事务,日志体积非常大.

Netzer 于 1993 年提出了传递优化算法 TR(transitivity reduction optimization)[35],它通过线程间指令依赖关系的传递性减少所需记录的指令逻辑序的数量,这种方法对有硬件支持的确定性重放访存指令逻辑序的记录产生了重要的影响.图 3(a)描述了 TR 算法如何通过传递关系减少所需记录的指令逻辑序数量,其中,→表示某一指令先于其他指令执行.$W(a,v1)\rightarrow R(a,v1)$;$W(b,v2)\rightarrow W(b,v3)$.同时,在线程 Thread 1 和 Thread 2 中,根据程序执行顺序分别有 $W(a,v1)\rightarrow W(b,v2)$及 $W(b,v3)\rightarrow R(a,v1)$,因此,只记录 $W(b,v2)\rightarrow W(b,v3)$这一条依赖就可以推导出 $W(a,v1)$和 $R(a,v1)$的依赖关系是 $W(a,v1)\rightarrow R(a,v1)$.

由于日志体积较大给应用带来很大的挑战,很多方案在 TR 算法基础上减少日志体积,包括 FDR[14],BugNet[21]和 RTR[15].FDR 以 cache 块为单位,用时间戳标记,并通过在 cache 一致性消息中添加该时间戳来检测并记录指令逻辑序,但只支持顺序连续性(SC)的内存模型.BugNet 和 RTR 分别从不同方面对 FDR 进行改进.BugNet 在 FDR 的基础上实现对用户代码和共享库的确定性重放.

RTR 进一步改进了 Netzer 的传递优化算法 TR,提出了新的记录局部逻辑顺序的算法 RTR(regulated transitive reduction),通过自行创建依赖关系来发现更多的传递优化机会,如图 3(b)所示.$W(a,v1)\rightarrow R(a,v1)$;$W(b,v2)\rightarrow W(b,v3)$,无法通过 TR 算法减少所需记录的依赖指令条数.根据 RTR 算法,通过创建依赖关系 $W(b,v2)\rightarrow R(a,v1)$,就可以根据 TR 算法来进行推导,进一步减小了所需记录的指令逻辑序数量.与 TR 相比, RTR 平均减少了 28%的日志体积,并且支持 TSO(total store order)的存储一致性模型.



(a) TR 算法示例 (b) RTR 算法示例

Fig.3 Key insight of TR algorithm and key insight of RTR algorithm

图 3 TR 算法和 RTR 算法示例

与上述方案针对单条指令间的依赖关系不同的是,很多研究如 Strata[13],Rerun[5]和 Delorean[6]通过记录指令

块之间的逻辑顺序来实现确定性重放.如图 4 所示,Strata 将程序的执行分成不同的层(stratums),当有处理器对有核间依赖关系的地址进行第 2 次访存操作时,一个新的层次就会被创建并被存储,所产生的日志是一个维数与处理器核的数量相同的向量,记录所有核在该层中访存指令的数量.通过这种方法,Strata 只需 1 条记录即可索引层间多条访存指令间的冲突,但是这种方法对于核的扩展性较差,且只支持顺序一致性存储模型.

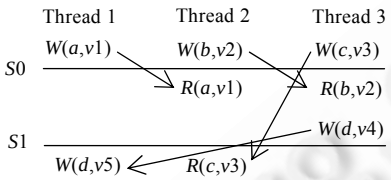


Fig.4 Key insight of Strata

图 4 Strata 关键设计示例

Rerun 将程序指令分成原子片段(atomic episode),用这种片段取代单条指令,作为记录逻辑序的基本单元.与其不同,Delorean 受事务存储[36]的启发,将程序的执行分成块(chunk),并通过记录块的提交顺序来实现确定性重放.

以上实现方式均以逻辑序理论为基础,然而在逻辑序的理论框架下,逻辑序信息量过大,这在很大程度上限制了确定性重放技术的发展.鉴于此,在当前集成电路工艺的支持下,Lreplay[7]提出了全局时钟理论,并在此基础上对确定性重放进行了研究,以期在实现并行程序确定性重放的同时,解决逻辑序给并行程序重放带来的困难.

全局时钟是指可以用其来度量片上多核处理器内所有事件先后的时钟.与逻辑序理论的相对顺序不同,全局时钟能够以较低的误差记录指令执行的物理顺序,并从指令执行的物理顺序推得其逻辑顺序,从而实现确定性重放.由于是对指令的物理顺序进行记录,因而在全局时钟的框架之下,通过有效的时间采样方法即可使用较小的日志记录大量信息,能够极大地减少空间开销.

Lreplay 以全局物理时钟(global physical time)为基础,通过未决时间区间(pending period)[23]抽象表示访存操作的精确完成时间,用未决时间区间的先后判断两个访存操作之间的时间序,并通过未决时间区间来推得处理器核访存事件的逻辑序,从而实现确定性重放.由于这个区间的开始时间和结束时间相对灵活,很容易进行观察,所以可以通过设计简洁的方法进行测量和采集,显著减少了记录的信息量.

图 5 为 Lreplay 的硬件实现,其中,主要添加的硬件为 LPU(log processing unit),Lreplay 通过该硬件来收集存储器指令信息并进行处理以生成日志,用于确定性的重放.

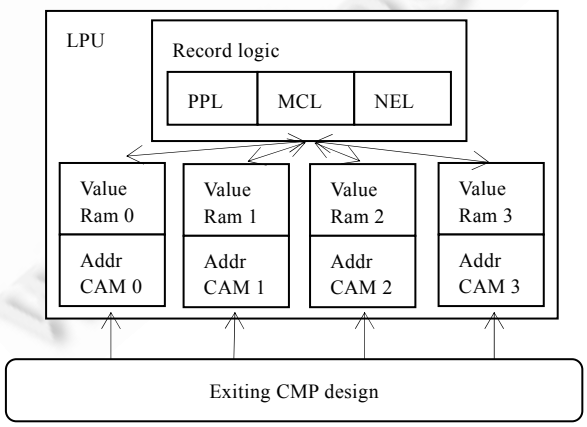


Fig.5 Hardware support of Lreplay

图 5 Lreplay 的硬件实现图

表 2 列出了目前有代表性的硬件支持确定性重放的方案.专用硬件的支持能够高效地记录存储器竞争,实现并行程序的确定性重放.虽然目前的研究已尽可能地减少了所需的硬件支持,但额外添加的硬件仍限制着这些方案的实际应用.

Table 2 Summary of characteristics of main hardware-assisted schemes

表 2 主要有硬件支持实现方式性能汇总

	记录阶段性能损失	重放阶段性能损失	日志体积	存储一致性	专用硬件体积	确定性
FDR[14]	少于 2%	N/A	2 MB per 1 GHz processor per second	SC	增加 Cache 体积 6.25%	严格确定性
BugNet[21]	N/A	N/A	255B per processor per kilo-instruction	SC	每个核增加 16KB 存储空间	严格确定性
RTR[15]	N/A	N/A	1 B per processor per kilo-instruction	SC, TSO	每个核 24KB 存储空间	严格确定性
Strata[13]	N/A	N/A	2.2B~2.3B per kilo-instruction	SC	L1cache 增加 64KB, L2cache 增加 2MB	严格确定性
Rerun[5]	N/A	N/A	4 B per kilo-instruction	N/A	每个核增加 166 bytes 存储空间	严格确定性
Delorean[6]	0~86%	72%~0.82%	3.7 bits per processor per kilo-instruction	RC	N/A	严格确定性
Lreplay[7]	N/A	N/A	0.55~0.85 B per kilo-instruction	RC	增加体积为 1.3%	严格确定性
Calvin[33]	20%	N/A	N/A	SC, TSO	N/A	严格或部分确定性

6 确定性重放研究趋势和发展前景

基于上述国内外研究现状,本文认为,对多核架构下并行程序确定性重放的研究应从以下几个方面寻求突破,并有如下的发展趋势:

6.1 软硬件协同

以纯软件方式实现确定性重放方法对现有的硬件平台没有特别的要求,因而较硬件实现方式有较强的通用性,但在软件层面记录存储器访问操作会严重影响程序的执行速度.虽然很多纯软件方式采用了改变确定性、增加重放次数等折衷方法,但它们在很大程度上影响了确定性重放系统的其他性能,限制了应用范围.纯软件的实现方式无法从根本上解决由存储器竞争引起的巨大时间开销,文献[18]中指出,专门硬件的缺失使得纯软件实现方式不太可能在真正的应用中被采用.

与此相对,有硬件支持的解决方式面临的最大问题就是对当前体系架构的修改以及硬件存储空间的开销.修改程度过高会极大限制其在工业界的应用推广,而过大的空间开销则对确定性重放实现的成本、支持重放时间的长度等都有严重的影响.除此之外,对有硬件支持的确定性重放的研究只专注于硬件层的设计,特别是只关注如何在硬件层面有效地记录存储器竞争,而忽略对造成多核架构下并行程序不确定性其他因素的记录和处理,与实际应用仍有很大的差距,需要进一步完善.

鉴于纯软件和有硬件支持两种实现方式的优缺点,我们认为采用软硬件协同的设计方式,将软件方法和硬件实现方式进行整合,并分别对软件和支持硬件的功能进行详细划分和设计,以充分利用各自的优势,有效地实现多核架构下并行程序的确定性重放.本文认为,软硬件协同是有效解决确定性重放问题的重要发展趋势之一.

6.2 存储器竞争的细化

随着处理器内核数量的增加,存储器竞争出现的频率也会随之增加,笼统地记录内存访问会给系统带来巨大的开销,严重影响程序的执行速度.因此,对存储器竞争中同步和数据竞争的区分以及对数据竞争中有害数据竞争的辨识,对确定性重放有重要影响,依据它们不同的特点和表现形式采用不同的记录和处理方式,对提高确定性重放性能有重要意义.

在指令层级,数据竞争和同步表现形式相同,单纯从硬件层面无法区分数据竞争和同步;然而在软件层级,虽然可以发现同步,但对于数据竞争却难以辨识,而是在程序出现错误时才能够检测或者标记数据竞争的出现.比如 Respec,将程序的运行分成不同的时间段(epoch),在每个时间段结束时设置检查点(checkpoint),用于检验重放阶段是否和记录阶段相一致.它分别使用内核级日志和用户级日志对记录阶段中的系统调用和用户级同步操作信息进行记录,但由于在软件层面无法对数据竞争进行检测和记录,因此只记录上述信息无法保证时间段的成功重放,Respec 必须使用预测执行来透明的重新执行重放阶段执行时失败的时间段,从而保证成功的确定性重放.虽然如此,多次的重放对系统的性能造成很大影响.

除此之外,文献[37]中指出,只有 10%的数据竞争属于有害数据竞争,其他 90%都不会造成程序错误.对于有害数据竞争的检测和区分,也有助于并行程序确定性重放的实现.比如 Rerun,将程序指令分成原子片段(atomic episode),通过定义原子片段间不存在存储器竞争的条件,在指令层级分别记录原子片段的读写集、长度和时间戳,将所有存储器竞争不加区分地统计和对比,以期通过这种实现方式来确保检验记录极少出现频率的数据竞争,造成了不必要的性能损失.

由此看出,将存储器竞争区分检测记录十分必要.本文认为,根据存储器竞争不同的表现形式,采用某种方式分别记录数据竞争和同步,并且将有害数据竞争和良性数据竞争区分开来进行记录和处理,将会极大地提升确定性重放的性能.

6.3 硬件整合

由于硬件成本较高,资源珍贵,为确定性重放而添加专门的硬件会增加生产商和用户的成本.同时,多核架构下所存在的原子性检查等其他问题与确定性重放问题相关联.将这些问题的解决方案整合到同一个硬件使其能够解决多个问题,是一种较为合适的方法.所以,对于多核架构下的确定性重放问题,未来很有可能从其他问题入手,利用其已经添加的硬件,经过不同的分析组合实现确定性重放.

6.4 新的编程模式和程序执行模式

多核架构下并行程序确定性重放问题难以有效实现的根本原因在于记录大量的共享存储器同步访问.而多线程的编程模式和程序执行模式是导致共享存储器访问出现的本质,因此,如果能够设计并实现新的编程模式或程序执行模式,使其避免对于共享存储器的同步访问,即可从根源上解决多核架构下并行程序确定性重放难于有效实现的问题.但该方法难度较大,如果能同时保证执行效率,其贡献将是巨大的.

6.5 其他技术的应用

目前,已经有确定性重放方案受到其他领域发展的影响,从而提出新方法,比如,Delorean 中使用了线程预测技术(thread-level speculation)和事务存储的思想,FDR 采用了 safetynet[38],Respec,ODR 等方案中利用的检查点技术和回滚机制等.这些技术和思想极大地促进了多核架构下并行程序确定性重放的研究,利用和借鉴其他领域的方法能够加快对于确定性重放问题的解决,也是未来确定性重放研究的主要趋势之一.

7 结束语

多核架构下并行程序的确定性重放对多核架构下并行编程的广泛普及有重要意义,但线程间存储器竞争的存在给确定性重放的实现带来很大挑战.本文的贡献在于分析了多核架构下并行程序确定性重放难以实现的关键原因,总结了确定性重放的评价指标,对确定性重放程度和实现方法进行了归类说明,并在综述国内外研究现状的基础上,对其未来发展方向和研究趋势进行了展望.对于当前多核架构下并行程序确定性重放的研究具有一定的指导意义.本文认为,鉴于当前多核架构的普及以及对计算性能提升的需求,多核架构下并行程序的确定性重放将会在并行编程中发挥越来越重要的作用.

致谢 在此,感谢中国科学院计算技术研究所陈云霁老师对本文给予的支持和建议.

References:

[1] Asanović K, Bodik R, Demmel J, Keaveny T, Keutzer K, Kubiatowicz J, Morgan N, Patterson D, Sen K, Wawrzynek J, Wessel D, Yelick K. A view of the parallel computing landscape. Communications of the ACM, 2009,52(10):56−67. [doi: 10.1145/1562764.1562783]

[2] Nightingale EB, Peek D, Chen PM, Flinn J. Parallelizing security checks on commodity hardware. In: Proc. of the 13th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2008. 308−318. [doi: 10.1145/1346281.1346321]

[3] Zhai JD, Chen WG, Zheng WM. Phantom: Predicting performance of parallel applications on large-scale parallel machines using a single node. In: Proc. of the 15th Annual Symp. on Principles and Practice of Parallel Programming. New York: ACM Press, 2010. 305−314. [doi: 10.1145/1693453.1693493]

[4] Lee DY, Wester B, Veeraraghavan K, Narayanasamy S, Chen PM, Flinn J. Respec: Efficient online multiprocessor replay via speculation and external determinism. In: Proc. of the 15th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2010. 77−90. [doi: 10.1145/1736020.1736031]

[5] Hower D, Hill MD. Rerun: Exploiting episodes for light weight memory race recording. In: Proc. of the 35th Int'l Symp. on Computer Architecture. Washington: IEEE Computer Society, 2008. 265−276. [doi: 10.1109/ISCA.2008.26]

[6] Montesinos P, Ceze L, Torrellas J. DeLorean: Recording and deterministically replaying shared-memory multiprocessor execution effciently. In: Proc. of the 35th Int'l Symp. on Computer Architecture. 2008. 289−300. [doi: 10.1109/ISCA.2008.36]

[7] Chen YJ, Hu WW, Chen TS, Wu RY. LReplay: A pending period based deterministic replay scheme. In: Proc. of the 37th Int'l Symp. on Computer Architecture. New York: ACM Press, 2010. 187−197. [doi: 10.1145/1815961.1815985]

[8] Dunlap G, King S, Cinar S, Basrai M, Chen P. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In: Proc. of the 5th USENIX Symp. on Operating System Design and Implementation. New York: ACM Press, 2002. 211−224. [doi: 10.1145/844128.844148]

[9] Xu M, Malyugin V, Sheldon J, Venkitachalam G, Weissman B. ReTrace: Collecting execution trace with virtual machine deterministic replay. In: Proc. of the 2007 Workshop on Modeling, Benchmarking and Simulation. 2007.

[10] Netzer RHB, Miller BP. What are race conditions?: Some issues and formalizations. ACM Letters on Programming Languages and Systems, 1992,1(1):74−88. [doi: 10.1145/130616.130623]

[11] Ronsse M, Bosschere KD. RecPlay: A full integrated practical record/replay system. ACM Trans. on Computer Systems, 1999, 17(2):133−152. [doi: 10.1145/312203.312214]

[12] Su M, Chen Y, Gao X. A general method to make multi-clock system deterministic. In: Proc. of the Conf. on Design, Automation and Test in Europe. 2010. 1480−1485. [doi: 10.1109/DATE.2010.5457045]

[13] Narayanasamy S, Pereira C, Calder B. Recording shared memory dependencies using strata. In: Proc. of the 12th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2006. 229−240. [doi: 10.1145/1168857.1168886]

[14] Xu M, Bodik R, Hill MD. A "flight data recorder" for enabling full-system multiprocessor deterministic replay. In: Proc. of the 30th Int'l Symp. on Computer Architecture. New York: ACM Press, 2003. 122−135. [doi: 10.1145/859618.859633]

[15] Xu M, Hill MD, Bodik R. A regulated transitive reduction (RTR) for longer memory race recording. In: Proc. of the 12th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2006. 49−60. [doi: 10.1145/1168857.1168865]

[16] LeBlanc TJ, Mellor-Crummey JM. Debugging parallel programs with instant replay. IEEE Trans. on Computers, 1987,36(4): 471−482. [doi: 10.1109/TC.1987.1676929]

[17] Park S, Xiong WW, Yin ZN, Kaushik R, Lee KH, Lu SH, Zhou YY. PRES: Probabilistic replay with execution sketching on multiprocessors. In: Proc. of the 22nd ACM Symp. Operating Systems Principles. New York: ACM Press, 2009. 177−192. [doi: 10.1145/1629575.1629593]

[18] Olszewski M, Ansel J, Amarasinghe S. Kendo: Efficient deterministic multithreading in software. In: Proc. of the 2009 Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2009, 97−108. [doi: 10.1145/1508244.1508256]

[19] Altekar G, Stoica I. ODR: Output-Deterministic replay for multicore debugging. In: Proc. of the 22nd ACM Symp. on Operating Systems Principles. New York: ACM Press, 2009. 193−206. [doi: 10.1145/1629575.1629594]

[20] Montesinos P, Hicks M, King S, Torrellas J. Capo: A software-hardware interface for practical deterministic multiprocessor replay. In: Proc. of the 14th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2009. [doi: 10.1145/1508244.1508254]

[21] Narayanasamy S, Pokam G, Calder B. BugNet: Continuously recording program execution for deterministic replay debugging. In: Proc. of the 31st Int'l Symp. on Computer Architecture. Washington: IEEE Computer Society, 2005. 284−295. [doi: 10.1109/ISCA.2005.16]

[22] Lamport L. Time, clocks and the ordering of events in a distributed system. Communications of the ACM, 1978,21(7):558−565. [doi: 10.1145/359545.359563]

[23] Chen Y, Chen T, Hu W. Global clock, physical time order and pending period analysis in multiprocessor systems. The Computing Research Repository, 2009.

[24] Dunlap GW, Lucchetti DG, Fetterman MA, Chen PM. Execution replay of multiprocessor virtual machines. In: Proc. of the 4th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual execution environments. New York: ACM Press, 2008. 121−130. [doi: 10.1145/1346256.1346273]

[25] Narayanasamy S, Pereira C, Patil H, Cohn R, Calder B. Automatic logging of operating system effects to guide application-level architecture simulation. In: Proc. the 2006 Joint Int'l Conf. on Measurement and Modeling of Computer Systems. New York: ACM Press, 2006. 216−227. [doi: 10.1145/1140277.1140303]

[26] Bhansali S, Chen WK, deJong S, Edwards A, Murray R, Drinić M, Mihŏcka D, Chau J. Framework for instruction-level tracing and analysis of program executions. In: Proc. of the 2nd Int'l Conf. on Virtual Execution Environments. New York: ACM Press, 2006. 154−163. [doi: 10.1145/1134760.1220164]

[27] Georges A, Christiaens M, Ronsse M, Bosschere KD. Jarec: A portable record/replay environment for multi-threaded Java applications. Software-Practice & Experience, 2004,34(6):523−547. [doi: 10.1002/spe.579]

[28] Bacon D, Goldstein S. Hardware-Assisted replay of multiprocessor programs. In: Proc. of the Workshop on Parallel and Distributed Debugging. New York: ACM Press, 1991. 194−206. [doi: 10.1145/122759.122777]

[29] Devietti J, Lucia B, Oskin M, Ceze L. DMP: Deterministic shared-memory multiprocessing. In: Proc. of the 14nd Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2009. 85−96. [doi: 10.1145/1508244.1508255]

[30] Prvulovic M. CORD: Cost-effective (and nearly overhead-free) order recording and data race detection. In: Proc. of the 12th IEEE Symp. on High-Performance Computer Architecture. 2006. 232−243. [doi: 10.1109/HPCA.2006.1598132]

[31] Prvulovic M, Torrellas J. ReEnact: Using thread-level speculation mechanisms to debug data races in multithreaded codes. In: Proc. of the 30th Annual Int'l Symp. on Computer Architecture. New York: ACM Press, 2003. 110−121. [doi: 10.1145/859618.859632]

[32] Devietti J, Nelson J, Bergan T, Ceze L, Grossman D. Rcdc: A relaxed consistency deterministic computer. In: Proc. of the 16th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2011. 67−78. [doi: 10.1145/1950365.1950376]

[33] Hower DR, Dudnik P, Hill MD, Wood DA. Calvin: Deterministic or not? Free will to choose. In: Proc. of the 17th IEEE Int'l Symp. on High Performance Computer Architecture. 2011. 333−334. [doi: 10.1109/HPCA.2011.5749741]

[34] Voskuilen G, Ahmad F, Vijaykumar TN. Timetraveler: Exploiting acyclic races for optimizing memory race recording. In: Proc. of the 37th Annual Int'l Symp. on Computer Architecture. New York: ACM Press, 2010. 198−209. [doi: 10.1145/1815961.1815986]

[35] Netzer R. Optimal tracing and replay for debugging shared-memory parallel programs. In: Proc. of the Workshop on Parallel and Distributed Debugging. New York: ACM Press, 1993. 1−11. [doi: 10.1145/174267.174268]

[36] Hammond L, Wong V, Chen M, Carlstrom BD, Davis JD, Hertzberg B, Prabhu MK, Wijaya H, Kozyrakis C, Olukotun K. Transactional memory coherence and consistency. In: Proc. of the 31st Annual Int'l Symp. on Computer Architecture. Washington: IEEE Computer Society, 2004. 102. [doi: http://doi.acm.org/10.1145/1028176.1006711]

[37] Narayanasamy S, Wang Z, Tigani J, Edwards A, Calder B. Automatically classifying benign and harmful data races using replay analysis. In: Proc. of the 2007 ACM SIGPLAN Conf. on Programming Language Design and Implementation. New York: ACM Press, 2007. 22−31. [doi: 10.1145/1250734.1250738]

[38] Sorin DJ, Martin MMK, Hill MD, Wood DA. SafetyNet: Improving the availability of shared memory multiprocessors with global checkpoint/recovery. In: Proc. of the 29th Annual Int'l Symp. on Computer Architecture. Washington: IEEE Computer Society, 2002. 123−134. [doi: 10.1109/ISCA.2002.1003568]

高岚(1987－),女,河北易县人,硕士生,CCF 学生会员,主要研究领域为多核体系结构.
E-mail: lan.gao@jsi.buaa.edu.cn

钱德沛(1952－),男,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算,多核体系结构.
E-mail: depeiq@buaa.edu.cn

王锐(1978－),男,博士,讲师,CCF 学生会员,主要研究领域为多核体系结构.
E-mail: rui.wang@jsi.buaa.edu.cn