DOI:10.7544/issn1000-1239.2016.20150100
53(6):1238-1248,2016

面向监听一致性协议的并发内存竞争记录算法

朱素霞[1,2] 陈德运[2] 季振洲[3] 孙广路[2] 张 浩[4]
[1](哈尔滨理工大学计算机科学与技术学院博士后流动站 哈尔滨 150080)
[2](哈尔滨理工大学计算机科学与技术学院 哈尔滨 150080)
[3](哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)
[4](中国科学院计算技术研究所 北京 100190)
(zhusuxia@hrbust.edu.cn)

A Concurrent Memory Race Recording Algorithm for Snoop-Based Coherence

Zhu Suxia[1,2], Chen Deyun[2], Ji Zhenzhou[3], Sun Guanglu[2], and Zhang Hao[4]
[1](*Postdoctoral Research Station, School of Computer Science and Technology, Harbin University of Science and Technology, Harbin* 150080)
[2](*School of Computer Science and Technology, Harbin University of Science and Technology, Harbin* 150080)
[3](*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)
[4](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

Abstract Memory race record-replay is an important technology to resolve the nondeterminism of multi-core programs. Because of high hardware overhead, the existing memory race recorders based on point-to-point logging approach are difficult to be applied to the practical modern chip multiprocessors. In order to reduce the hardware overhead of point-to-point logging approach, a novel memory race recording algorithm implemented in concurrent logging strategy for chip multiprocessors adopting snoop-based cache coherence protocol is proposed. This algorithm records the current execution points of all threads concurrently when detecting a memory conflict. It extends the point-to-point memory race relationship between two threads to all threads in recording phase, reducing hardware overhead significantly. It also uses distributed logging mechanism to record memory races to reduce bandwidth overhead effectively in the premise of not increasing the memory race log. When replaying, this algorithm uses a simplified producer-consumer model and introduces a counting semaphore for each processor core to ensure deterministic replay, improving replay speed and reducing coherence bandwidth overhead. The simulation results on 8-core chip multiprocessor (CMP) system show that this concurrent recording algorithm based on point-to-point logging approach adds about 171 B hardware for each processor, and records about 2.3 B log per thousand memory instructions and adds less than 1.5% additional interconnection bandwidth overhead.

Key words chip multiprocessor (CMP); multi-core program; deterministic replay; memory race recording; memory conflict detection; snoop-based coherence protocol

收稿日期:2015-02-05;**修回日期**:2015-07-07
基金项目:国家自然科学青年基金项目(61502123);国家自然科学基金项目(61173024);国家"九七三"重点基础研究发展计划基金项目(2011CB302501);黑龙江省青年科学基金项目(QC2015084);中国博士后科学基金项目(2015M571429)
This work was supported by the National Natural Science Foundation of China for Youths (61502123), the National Natural Science Foundation of China (61173024), and the National Basic Research Program of China (973 Program) (2011CB302501), Heilongjiang Province Science Foundation for Youths (QC2015084), the China Postdoctoral Science Foundation (2015M571429).

摘 要 内存竞争记录是解决多核程序执行不确定性的关键技术,然而现有点到点的内存竞争记录机制带来的硬件开销大,难以应用到实际的片上多核处理器系统中.以降低点到点内存竞争记录方式的硬件开销为出发点,为采用监听一致性协议的片上多核处理器(chip multiprocessor, CMP)系统设计了基于并发记录策略的点到点内存竞争记录算法.该记录算法将两两线程间点到点的内存竞争关系扩展到所有线程,采用分布式记录方法为每个线程记录一个由内存竞争关系的一方构成的内存竞争日志;重演时采用简化的生产者消费者模型,确保了确定性重演的实现,有效降低了硬件消耗和带宽开销.在8核处理器系统中的仿真结果表明,该并发式点到点内存竞争记录算法为每个处理器核添加硬件资源约171 B,每千条内存操作指令记录日志大小约2.3 B,记录和重演阶段均添加不到1.5%的带宽开销.

关键词 片上多核处理器;多核程序;确定性重演;内存竞争记录;内存冲突检测;监听一致性协议

中图法分类号 TP303

在共享内存的片上多核处理器(chip multiprocessor, CMP)系统中,线程间内存访问的顺序不确定,导致多核程序的编写、调试、容错[1]和安全[2]等都变得异常困难.内存竞争记录和重演是解决这一问题的有效手段.内存竞争记录和重演机制通过在多核程序原始执行阶段记录下线程间的内存竞争顺序,在重演阶段依据所记录的内存竞争顺序复现原始的执行顺序,从而辅助实现多核程序的确定性重演.截至目前为止,研究者们已经提出了众多软件[3-4]和硬件[2,5-13]实现的内存竞争记录机制.硬件实现方案通过为原有CMP系统增加新的硬件资源实现内存竞争的记录和重演,具有效率高、开销低等优点,应用前景广阔.硬件实现的内存竞争记录方案中,主要有点到点记录方式[2,5,9-12]和chunk记录方式[6-8,13]2大类.点到点的记录方式以指令为粒度,记录线程间内存冲突对应的依赖关系,无论是记录还是重演,都具有良好的并行性,但硬件资源消耗较大,影响了它在实际CMP系统中的应用.chunk记录方式使用较少的硬件资源记录线程间无冲突的指令块,按照chunk时戳的顺序实现确定性重演.目前这2种记录方式针对基于目录一致性协议的CMP系统已经取得了丰富的研究成果[2,5-6,8-12],但是针对采用监听一致性协议系统的研究还有待进一步挖掘,因为实际的CMP系统大都采用了基于监听的cache一致性协议.因此,研究支持监听一致性协议的内存竞争记录和重演机制更具现实意义.

IMRR[7]针对监听一致性协议的CMP系统提出了一种基于chunk的内存竞争记录机制,该机制有效解决了监听协议下内存竞争的记录和重演,并且针对chunk方式重演速度受限问题,提出了并发chunk域来提高重演的速度.但是它在实现重演时需要引入同步计数器等更新机制,导致重演结构略显复杂,对原有一致性协议结构改造较大.文献[13]依据IMRR的思想采用FPGA实现了支持内存竞争记录和重演功能的Intel处理器架构原型.LReplay[14]通过记录指令或指令块的延迟时间信息,引入少量的硬件资源,记录了较小的日志,但它只能应用于一款特殊的体系结构Godson-3[15].其他基于chunk的记录机制[8]和采用点到点记录方式的内存竞争记录机制[2,5]均是针对基于目录的一致性协议提出的,虽然也可以应用到采用监听一致性协议的系统中,但都未给出该协议下记录算法的详细设计描述.而且,因为需要在处理器核间传送指令时戳等信息,点到点的记录方式还会导致监听协议下较大的带宽开销.

为了减小点到点记录方式的硬件资源消耗、提高其应用前景,本文面向采用监听一致性协议的CMP系统,提出了一种新颖的并发式点到点内存竞争记录算法(concurrent point-to-point memory race recording algorithm, CPMRR),相比已有的点到点记录策略,显著降低了硬件开销;相比面向监听一致性协议的chunk记录机制IMRR[7],该记录算法在未增大内存竞争日志的前提下,提高了重演速度,降低了带宽开销.

本文的贡献有3点:1)针对基于监听的cache一致性协议提出了第1个完整的点到点内存竞争记录算法;2)采用了并发式点到点记录策略,带来的硬件开销可以同chunk记录方式相媲美;3)提出了一种基于生产者消费者模型的高效的重演机制,硬件实现结构简单,重演速度快,核间互联带宽开销低.

1 记录内存竞争

2个或多个线程访问同一个内存,并且至少有

1 个是写操作,则发生内存冲突. 如果没有正确的使用同步操作,线程间内存冲突的顺序是不确定的,可能会引起内存竞争. 内存竞争检测是个 NP 困难问题[16],因此,点到点的内存竞争记录方式通过记录线程间内存冲突双方构成的依赖关系为每个线程记录 1 个内存竞争日志,如图 1 所示,便可以实现确定性重演. 该记录方式以指令为粒度,记录的是两两线程间的局部关系,要记录所有线程间的这种局部关系,需要 CMP 系统中的每个处理器核为其他所有处理器核都要存储信息. 假设 CMP 系统中共有 n 个处理器核,文献[2,5,9]中除了需要增加庞大的 cache 时戳外,每个处理器核还需要存储对应其他 $n-1$ 个处理器核的时戳. 文献[10-12]中设计的内存竞争记录器虽然采用占用较少资源的签名替换了时戳,但需要为每个处理器核添加 $n-1$ 对签名. 相比 chunk 记录方式,点到点记录方式带来的硬件资源消耗较高,这也是点到点记录方式难以应用到实际 CMP 系统的主要原因之一.

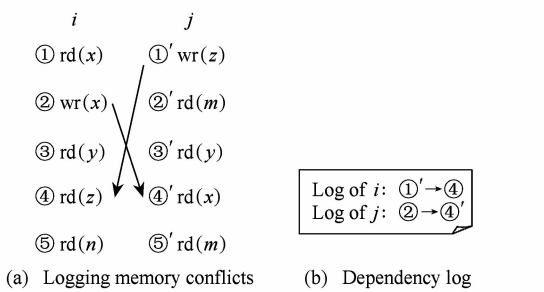

Fig. 1 Logging point-to-point memory races.

图 1 点到点内存竞争记录策略示意图

为了提高点到点记录方式的应用前景,本文面向监听一致性协议的 CMP 系统,提出了基于并发记录策略的点到点内存竞争记录算法. 该算法在检测到内存冲突后,记录下所有线程间内存操作指令的执行顺序,将线程间点到点内存竞争关系扩展到所有线程,减少了每个处理器核所需要存储的信息,降低了硬件资源消耗.

1.1 并发记录内存冲突

2 个线程发生冲突时,冲突双方对应的内存操作指令间存在执行的先后顺序. 虽然冲突先发生方所在线程同其他线程间没有发生冲突,但这些线程之间的内存操作指令也同样存在执行的先后顺序. 如图 2 所示,带箭头的虚线表示冲突双方的先后执行顺序,带箭头的点划线表示冲突先发生方所在线程同其他线程间内存操作指令的先后执行顺序,带箭头的实线表示并发记录方式中所要记录的冲突顺序. 图 2(a)中线程 i,j,k 分别运行在 CMP 系统中不同的处理器核上,当线程 i 执行到标记为③的内存操作指令 wr(z)后,线程 i,j 间会检测到 1 个内存冲突 j:①′→i:③,此时,线程 k 执行完标记为③″的内存操作指令 rd(y). 此时,线程 j,k 间虽然不存在冲突,但可以存在 j:①′→k:③″这样的执行顺序. 如果也记录下线程 j,k 间这种执行顺序,在重演时不会影响原有线程间内存操作指令执行顺序的复现,仅是增加了重演时检查点的次数. 鉴于此观察,本文在进行内存竞争记录时,将两两线程间的内存冲突扩展到所有线程,不只记录冲突双方的先后顺序(如图 2(a)中标记出的 j:①′→i:③),也记录下冲突先发生方所在线程同其他线程间内存操作指令的先后执行顺序(如图 2(a)中标记出的 j:①′→k:③″). 本文后面部分均称这 2 种先后顺序为冲突依赖关系.

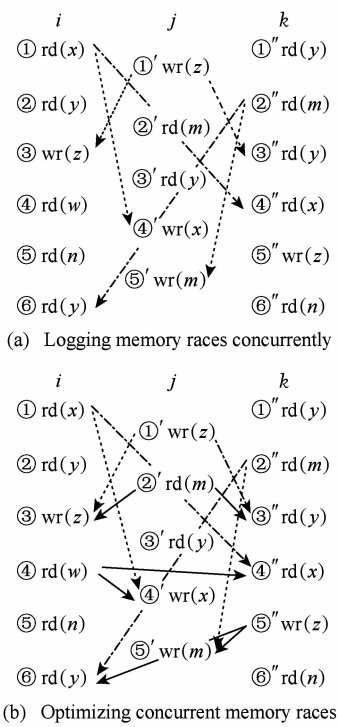

Fig. 2 An example of logging memory races concurrently.

图 2 并发记录内存竞争示意图

该记录方式可以在检测到冲突时,并发的记录下所有线程间内存操作指令的执行顺序,即变相的假定冲突先发生方同其他所有线程间都存在冲突. 如果只记录不能推导的内存冲突[17],则无需再检测先发生方所在线程已执行完的内存操作指令是否还会与其他线程发生冲突,因此,也就无需存储先发生方所在线程已执行完的内存操作指令的相关信息,从而可以在硬件实现时减少硬件资源消耗. 同时,为了

进一步减小硬件开销,本并发式记录策略采用冲突发生时冲突双方当前指令计数值构成的当前发生序[10]表示冲突依赖关系.如图 2(b)所示,带箭头的实线标记出了该并发记录方式所要记录的冲突依赖关系.

1.2 精简内存竞争日志

本文所提出的并发式内存竞争记录策略虽然通过记录冲突发生时所有线程间内存操作指令的先后执行顺序,减少了每个线程因检测内存冲突而需要存储的信息,但会导致记录的冲突数目过多,使得整个内存竞争日志变大.例如,该记录方式在检测到 1 个内存冲突时,需要记录 $n-1$ 个冲突依赖关系,所记录的日志数目是原有点到点记录方式的 $n-1$ 倍.并且,每次检测到冲突时所记录的 $n-1$ 个冲突依赖关系的先发生方都对应同一个内存操作指令,即冲突先发生方会被重复记录 $n-1$ 次,导致日志冗余过多.同样,随着系统中处理器核数目的增多,记录的日志也会成倍地增大.因此,本文采取了一系列措施来精简内存竞争日志.

首先,该并发式内存竞争记录策略不再记录冲突依赖关系的双方到同一线程的日志中,而是采用分布式的日志记录策略,每个线程仅记录冲突的一方,而且对于需要重复记录的冲突先发生方也只记录 1 次.如此以来,冲突先发生所在线程只记录先发生方,冲突后发生方所在线程和其他线程仅记录后发生方.为了区别线程记录的信息是先发生方还是后发生方,本文在记录格式中引入 1 个类型标志位,即记录格式包含 2 个域:1)类型域,它指出了该记录是先发生方还是后发生方,可以仅用 1 位来表示,0 代表先发生方,1 代表后发生方;2)大小域,它指出了该内存操作对应的指令计数值.如图 3 所示,带箭头的虚线表示冲突双方的先后执行顺序,带箭头的点划线表示冲突先发生方所在线程同其他线程间内存操作指令的先后执行顺序,带箭头的实线表示并发记录方式中所要记录的冲突顺序.图 3(b)给出了线程 i,j,k 所记录的分布式日志:当检测到内存冲突 j:①′→i:③时,线程 i,j,k 分别记录〈1,③〉,〈0,②′〉,〈1,③″〉到各自线程的日志中;当检测到内存冲突 i:1→j:④′时,线程 i,j,k 分别记录〈0,④〉,〈1,④′〉,〈1,④″〉到各自线程的日志中;当检测到内存冲突 k:②″→j:⑤′时,线程 i,j,k 分别记录〈1,⑥〉,〈1,⑤′〉,〈0,⑤″〉到各自线程的日志中.

为了进一步精简内存竞争日志,本并发式记录策略在每次记录下冲突依赖关系的先发生方和后发生方后将指令计数器复位,让其从 0 开始重新计数.同时,当每个线程的内存操作指令计数累计到 $2^{15}-1$ 时,也假定检测到内存冲突,在记录下先发生方和后发生方后,将指令计数器复位.当发生线程的上下文切换时,也假定检测到内存冲突,在记录下先发生方和后发生方后,将指令计数器复位.采用如此方法,不仅解决了线程切换带来的日志记录问题,还可以有效控制每个记录的大小不超过 $2^{15}-1$,从而进一步减小了内存竞争日志.如图 3(b)右侧日志部分指出了精简后的分布式内存竞争日志.

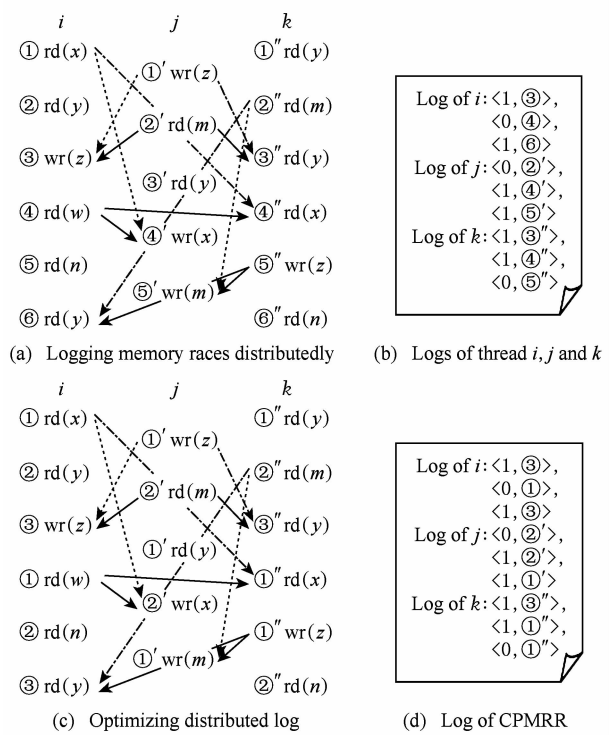

Fig. 3 Reducing memory race log.

图 3 精简内存竞争日志

这种分布式的记录策略,虽然将内存冲突的依赖关系分开记录,但仍然以指令为粒度记录线程间指令执行的先后顺序关系,没有改变点到点记录方式的本质.该记录方式不仅能够有效地减小日志,还可以减小核间互联网络的带宽开销.因为原来的点到点记录方式将冲突依赖关系记录到某个线程的日志中,需要处理器核间传送指令计数值,这样增加了带宽开销而分布式的记录策略却无需传送指令计数值,仅需在检测到内存冲突时冲突先发生方所在处理器核向其他处理器核广播 1 个检测到冲突的消息即可,其他处理器核收到此消息后,记录后发生方到内存竞争日志中.

2 检测内存冲突

并发记录策略在检测到内存冲突后会记录下此刻所有线程的执行点,即冲突依赖关系的先发生方和后发生方,先发生方之前的内存操作指令不再参与到后续的内存冲突检测中.因此,可以将先发生方所记录的有关冲突检测的所有信息清空.因此,本文采用签名技术,且仅需要为每个处理器核添加1对读写签名就可以实现内存冲突检测.当有内存访问请求时,应答方所在处理器核通过查找签名寄存器实现内存冲突检测,一旦检测到冲突,则记录下先发生方并广播1个检测到内存冲突的消息到所有处理器核,同时清空读写签名寄存器、复位指令计数器.

检测到内存冲突消息在实现时无需改变监听一致性协议的结构,而是仅在现有协议基础上增加了1个新型的请求类消息,取名为 *record*.因为采用分布式日志记录策略记录内存竞争,无需传送内存操作指令的时戳,该消息会带来较小的带宽开销.

3 重演内存冲突

为了能够依据所记录的内存竞争日志确定性的重演多核程序,本文采用了基于唤醒消息的重演策略.在重演时,处理器核所读取的日志信息只包含2种信息:先发生方和后发生方、先发生方所在处理器核会在先发生方对应指令执行完毕后向其他处理器核广播1个唤醒消息;后发生方所在处理器核在接收到先发生方发送的唤醒消息后才能继续执行.对某个线程来说,它所记录的后发生方只需要接收来自其他线程的唤醒消息.假设消息不会丢失,则每个后发生方都能等到相应的唤醒消息,从而可以实现确定性的重演.

在并发式记录策略中,所有线程所记录的先发生方之间存在1个全局的先后执行顺序,即连续的2个先发生方之间存在 happens-before 关系[18].图4是1个关于多核程序执行的简单示意图,a,b,c 等标识出了记录阶段所记录的先发生方和后发生方,带箭头的实线指出了它们之间的先后执行顺序.如图4(a)所示,连续的2个先发生方 a 和 d 位于同一线程,它们之间很自然地存在 happens-before 关系(hb),记为:$a \xrightarrow{\text{hb}} d$;如果新的先发生方位于某个后发生方之后,如图4(b)所示,因为 $b \xrightarrow{\text{hb}} e$,而 $a \xrightarrow{\text{hb}} b$,所以上一先发生方 a 和新的发生方 e 之间也一定存在关系:$a \xrightarrow{\text{hb}} e$.

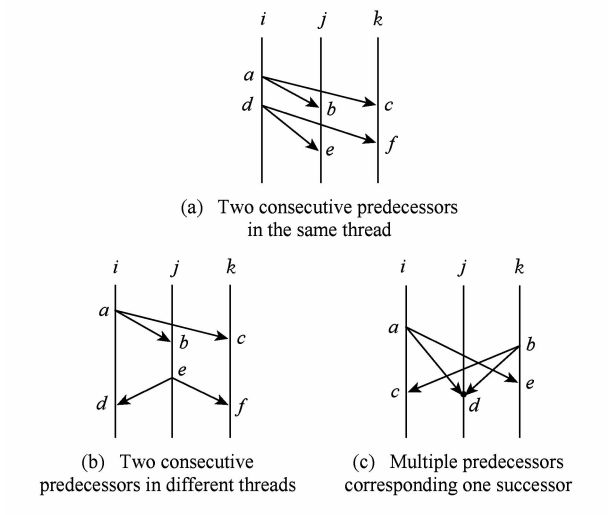

Fig. 4 Simple execution diagrams of a multi-core program.

图4 多核程序执行简单示意图

同时,位于同一线程的后发生方等待唤醒消息的顺序与先发生方所存在的全局的先后执行顺序也是一致的.在图4(a)中,位于线程 j 的后发生方 b 会先等待唤醒消息,e 会后等待唤醒消息,而 j 线程会先收到来自先发生方 a 的唤醒消息,后收到来自先发生方 b 的唤醒消息.对于其他线程也同样存在如此顺序.

如果1个后发生方需要等待多个来自其他线程的唤醒消息,则唤醒消息到来的先后顺序不会影响重演.如图4(c)所示,d 需要等待来自 a 和 b 的唤醒消息,重演时,只要2个消息都到达后 d 便可执行,无论 a,b 消息到来的先后顺序如何.

假设重演过程中不存在唤醒消息丢失现象,对于1个后发生方只对应1个先发生方的情况,重演时,如果1个后发生方(如图4(a)中的点 b)只等到了1个唤醒消息,则此唤醒肯定是来自于它对应的先发生方(如点 a),如果1个后发生方(如点 b)等到了2或多个唤醒消息,那其中有1个肯定是它要等待的唤醒消息.对于1个后发生方对应多个(假设 m 个)先发生方的情况,如图4(c)中所示,重演时,若点 d 等到了 m 个或大于 m 个唤醒消息,则肯定有 m 个唤醒消息是 d 所要等待的.因此,在重演时无需关心唤醒消息来自何方,只需要判断接收的唤醒消息是否达到后发生方所要求的唤醒消息数量即可.

鉴于此,本文在实现重演时,采用了简化的生产者和消费者模型:每个处理器核既充当生产者,又充

当消费者,当接收到唤醒消息后将唤醒消息计数值加 1,当执行到后发生方时将唤醒消息计数值减 1.为此,本文在实现重演时,为每个处理器核引入 1 个简化的唤醒消息计数信号量,用来记录接收到唤醒消息的数量.重演时,当处理器核收到唤醒消息后,唤醒消息计数信号量加 1,当线程执行到后发生方时,判断唤醒消息计数器的值是否为空,若不为空,则继续向前执行后发生方,执行完后发生方后,唤醒消息计数器减 1.随着程序的重放,生产的消息会不断地被消费,因此可用消息的数量是有限的,可以给唤醒消息计数信号量设定 1 个最大值,如 $2^{16}-1$.

唤醒消息的实现同记录阶段 *record* 消息的实现机制一样,无须改变现有的一致性协议结构,仅为监听一致性协议再增加 1 个新型的请求操作类型,名为 *wakeup*.当处理器核执行完先发生方后,向其他处理器核广播 1 个 *wakeup* 消息,同样因为无需标记唤醒消息的来源,该消息会为核间互联网络带来较小的带宽开销.

4 基于硬件的算法描述及具体实现结构

4.1 记录算法

本文提出的并发式点到点记录算法在检测到内存冲突后为每个线程记都记录 1 个日志,冲突的先发生方所在线程记录冲突对应的先发生方,其他线程记录冲突对应的后发生方.该记录算法基于硬件的描述如算法 1 所示,它详细描述了每个处理器核的状态和动作.

算法 1. 内存竞争记录算法.

每个处理器核的状态:

IC—指令计数器;

Pred—内存冲突的先发生方;

Succ—内存冲突的后发生方;

RF—读签名;

WF—写签名;

Log—内存竞争日志.

```
在内存指令提交时 memop{
  IC++;
  if (memop 是写指令)
    WF.insert (memop.address);
  if (memop 是读指令)
    RF.insert(memop.address);
  if (IC=2^15-1){
    Broadcast(record);
    Log.append(0,IC[14:0]);
    WF.clear();
    RF.clear();
    IC.clear();
  }
}
当收到来自其他处理器核关于块 b 的一致性请求消息 request 时 {
  if (WF.find(b.address) ‖ (RF.find(b.address) && (request=GETX))){
    Broadcast(record);
    Log.append(0,IC[14:0]);
    WF.clear();
    RF.clear();
    IC.clear();
  }
}
当收到来自其他处理器核的 record 一致性消息时 {
  Log.append(1,IC[14:0]);
  WF.clear();
  RF.clear();
  IC.clear();
}
```

该记录算法中每个处理器核做 4 个动作:

1) 每当处理器核执行内存操作指令时,指令计数器加 1;如果是写内存操作则将对应内存地址添加到写签名中,如果是读内存操作则将对应内存地址添加到读签名中.

2) 当收到来自其他处理器核的共享内存访问请求时,处理器核查找签名来检测是否发生内存冲突.

3) 如果检测到内存冲突或者指令计数器的值达到 $2^{15}-1$,则向所有处理器核广播 1 个 *record* 消息,记录冲突对应当前发生序的先发生方(指令计数器的值)到对应线程的日志中,并清空读写签名、复位指令计数器.

4) 当处理器核收到来自其他处理器核的 *record* 消息后,记录后发生方(指令计数器的值)到所在线程的日志中,并清空读写签名、复位指令计数器.

4.2 重演算法

本文提出的内存竞争记录算法在重演时采用了简化的生产者和消费者模型,为每个处理器核增加 1 个唤醒消息计数信号量辅助实现确定性重演.该重演算法基于硬件的描述如算法 2 所示,它详细描述了每个处理器核的状态和动作.

算法 2. 内存竞争重演算法.

每个处理器核的状态:

IC—指令计数器;

$WakeC$—唤醒消息计数器;

$WaitC$—等待消息计数器;

$Rflag$—读标志,初值为 false;

$Type$—日志类型;

Log—内存竞争日志.

```
当收到来自其他处理器核的 wakeup 一致性消
  息时 {
  WakeC++;
}
执行指令 {
  if (!Rflag)
    entry_old=Log.read(Type,IC);
  if (Type=0){
    for (i=1;i≤IC;i++) {
      执行下一条指令;
  }
    Broadcast(wakeup);
    Rflag=true;
} else if (Type=1) {
    WaitC++;
    for (i=1;i<IC;i++) {
      执行下一条指令;
  }
    entry_new=Log.read(Type,IC);
    while (!(Compare(entry_old,entry_new))){
      WaitC++;
      entry_old=entry_new;
      entry_new=Log.read(Type,IC);
  }
    if (WakeupC≥WaitC) {
      执行指令 IC;
      WakeC=WakeC-WaitC;
      IC.clear();
      WaitC.clear();
      }
    Rflag=false;
  }
}
```

该重演算法中每个处理器核所做的动作具体描述如下:

1) 当处理器核收到来自其他处理器核的唤醒消息 $wakeup$ 时,将唤醒消息计数信号量的值加 1.

2) 处理器核在执行指令前先读取 $Rflag$ 标志,判断是否需要从日志中读取 1 条记录,如果需要,则从日志中读取 1 条记录. 其中,$Rflag$ 的初始值为 false;如果处理器核读取的上一记录是后发生方类型,则 $Rflag$=true,否则 $Rflag$=false. 这是因为处理器核读取到后发生方类型的记录后,还需要继续从日志中读取记录,直到读到 1 个与上一记录不同的记录,因此,有些记录已经被提前读出.

3) 判断读出的记录是先发生方还是后发生方,如果是先发生方,则处理器执行指令,直至执行完毕先发生方对应的指令后,广播 $wakeup$ 消息;如果是后发生方,处理器则需要做 3 项处理.

① 执行指令到后发生方的前 1 条指令.

② 从日志中读取 1 条新记录,判断此条记录是否同上一记录相同,如果相同,说明此条记录是后发生方且对应多个先发生方,即它需要等待多个唤醒消息,这时,将等待消息计数器加 1,然后重复步骤②,直至读到不同的记录为止.

③ 比较唤醒消息信号量的值和等待消息计数器的值,如果唤醒消息信号量的值大于或等于等待消息计数器的值,则表明该后发生方已经等到所有的唤醒消息,处理器核继续执行后发生方对应的指令,并将唤醒消息信号量的值减去等待消息计数器的值,复位指令计数器和等待消息计数器,置 $Rflag$=true.

因为本文采用的并发式记录策略所记录的后发生方可能会被重复记录多次,如图 4(c)所示,点 d 会被记录 2 次,因此,在重演阶段引入计数信号量对这种特殊情况进行了处理,提高了重演效率.

4.3 重演速度分析

本文提出的并发式记录策略,同以往的点到点内存竞争记录方法一样,能够实现指令级的重演. 但因为该记录策略将许多的非内存冲突也假定为内存冲突,会增加重演时检查点的次数,一定程度上减缓了重演速度. 然而,相比采用 chunk 记录策略的 IMRR[7] 在重演速度方面具有一定的优势. 首先是因为 CPMRR 是以指令为单位,而 IMRR 则以指令块为单位;其次,后发生方仅在等到它所需要的消息后就会继续执行,而 IMRR 后续 chunk 要等到所有处理器核发出 chunk-end 消息后才能继续向前执行.

借助有向无环图(DAG),本文对并发点到点内存竞争记录策略的重演速度进行进一步地分析. 对

于 CPMRR 记录的内存竞争日志,可以映射成 1 个 DAG:每个顶点代表日志的记录点,即先发生方或后发生方;连接 2 个顶点的边的权值表示 2 个顶点间的内存操作指令的数目. 对于 IMRR,映射的 DAG 中每个顶点表示日志中记录的 chunk,因为后续 chunk 需要等到上一时戳的所有 chunk 都执行结束后才能开始执行. 因此,在所映射的 DAG 中,相同时戳的 chunk 可以映射到同一个顶点;连接 2 个顶点的边的权重表示 chunk 的大小,即 chunk 中包含的内存操作指令的数目. 如此,可以通过计算 DAG 的关键路径来比较两者的重演速度,关键路径越短,重演速度就更快. 图 5 给出了将内存竞争日志映射成 DAG 的 1 个示例. 该示例中,线程 i,j,k 各执行 30 条内存操作指令;图 5(a)和图 5(b)对应 CPMRR,图 5(c)和 5(d)对应 IMRR;在映射后的 DAG 中,均增加 1 个起始顶点 s 和 1 个结束顶点 e. CPMRR 中,线程 i,j,k 分别记录了 10,12,5 共 3 个记录点,线程 i 同线程 j 和 k 之间存在先后顺序关系. IMRR 中每个线程分别记录了时戳为 1($LC=1$)和 2($LC=2$)的 2 个 chunk,3 个线程中 $LC=1$ 的 chunk 分别含 10,12,5 条内存操作指令. 由 DAG 图可以计算出 2 种记录策略的关键路径长度分别是 35 和 37,CPMRR 的关键路径要短于 IMRR.

假设有 m 个线程,CPMRR 中所有后发生方都仅需要等待 1 个唤醒消息,各线程执行到上一记录点所需要的时间分别是$\{t_{11},t_{21},\cdots,t_{m1}\}$,各线程中上一记录点到当前记录点的时间间隔是$\{t_{12},t_{22},\cdots,t_{m2}\}$. 则在 CPMRR 中,DAG 的关键路径为 $t_{i1}+t_{j2}$,其中 $i,j\in\{1,2,\cdots,m\}$,而 IMRR 对应 DAG 的关键路径为 $\max\{t_{11},t_{21},\cdots,t_{m1}\}+\max\{t_{12},t_{22},\cdots,t_{m2}\}$. 因此,相比 IMRR,该并发式点到点内存竞争记录重演算法在重演阶段等待的时间更短,在重演速度方面要优于 IMRR.

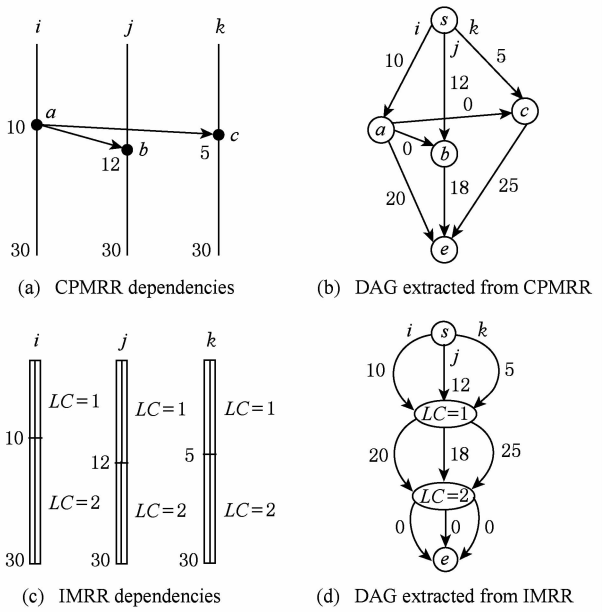

Fig. 5 An example of DAG extracted from memory race log.

图 5 内存竞争日志映射有向无环图示例

4.4 硬件结构

结合上述基于硬件的算法描述,图 6 给出了该并发式内存竞争记录算法基于 8 核 CMP 的硬件实现结构框图. 它为原有系统中的每个处理器核增加 1 个记录重演模块,用来实现内存竞争的记录和重演功能. 该记录重演模块共包含以下硬件:读签名寄存器 1 个(RF,256 b)、写签名寄存器 1 个(WF,1024 b)、指令计数器 1 个(IC,16 b)、先发生方寄存器 1 个($Pred$,16 b)、后发生方寄存器 1 个($Succ$,16 b)、等待消息计数器 1 个($WaitC$,16 b)、唤醒消息信号量寄存器 1 个($WakeC$,16 b)、记录类型寄存器 1 个($Type$,1 b)、读取标志寄存器 1 个($Rflag$,1 b). 其中 RF 和 WF 仅用于记录功能,IC 和 $Type$ 为记录和重演功能共用,其他部件仅用在重演阶段. 若不考虑运算器和日志缓冲部分,本记录算法为每个处理器核增加约 171 B 的硬件资源.

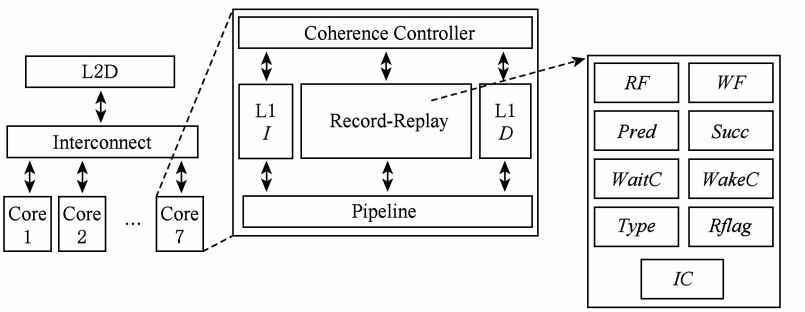

Fig. 6 The hardware configuration.

图 6 硬件结构框图

5 仿真结果

本文采用 GEMS[19] 对该并发式内存竞争记录算法(CPMRR)进行了仿真,并与同一配置下基于 chunk 记录策略的 IMRR[7] 的仿真结果进行了比较,同时,也给出了在硬件消耗方面 CPMRR 同已有的点到点记录算法的比较结果. 仿真配置如表 1 所示,测试负载选取典型的应用于多线程科学计算的 SPLASH2[20].

Table 1 Key Parameters in GEMS Configuration

表 1 GEMS 关键配置参数

Item	Description
Cores	8 cores, in-order, 3 GHz
Private L1 Cache	Split I&D, private, 64 KB 4-way set associative, write-back, 64 B lines, LRU replacement
Shared L2 Cache	Unified, shared, inclusive, 16 MB 4-way set associative, write-back, LRU replacement
Memory	4 GB DRAM
Coherence	Snoop-based MOSI
Consistency Model	Sequential consistency

下面给出 CPMRR 在硬件开销、日志记录和带宽开销方面的仿真结果:

1) 硬件开销. CPMRR 需要为每个处理器核添加 1 对读写签名,对于 8 核的 CMP 系统,仅增加约 171 B 的硬件资源,硬件开销同 IMRR 相当. 然而,相比已有采用签名实现的点到点内存竞争记录机制[9-12],硬件开销显著降低,约降低了 86%. 而且,CPMRR 为每个处理器核添加的硬件资源不再与系统中处理器核的数目相关,其硬件开销不会随着处理器核数目的增加而显著增加.

2) 日志记录. 图 7 给出了 CPMRR 同 IMRR 之间内存竞争记录次数和内存竞争日志的比较结果. 从图 7(a)可以看出,虽然 CPMRR 的记录次数平均约是 IMRR 的 2 倍,但其记录的日志总体上却与 IMRR 相当,如图 7(b)所示. 这是因为:IMRR 中不仅要记录 chunk 的大小,为了实现并行重演还需要记录 chunk 的时戳;而 CPMRR 采用分布式策略对日志进行了精简,仅记录了冲突双方的指令计数值,该计数值相当于 IMRR 中 chunk 的大小.

3) 带宽开销. 如图 8(a)所示,CPMRR 在记录阶段新增加的 *record* 消息仅占总一致性请求消息的 2.7%,因此,它为 8 核 CMP 系统的核间互联网络带来的带宽开销也较小,如图 8(b)所示,平均带宽开销不到 1.5%. 这是因为并发式内存竞争记录机制在检测到内存冲突后仅广播 *record* 消息,而该消息内容简单,无需添加指令时戳.

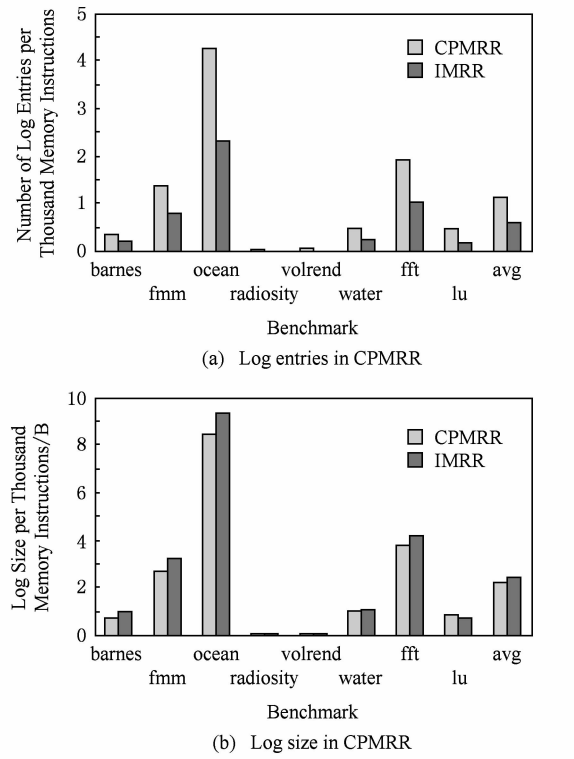

Fig. 7 Memory race recording performance in CPMRR.

图 7 CPMRR 日志记录性能统计

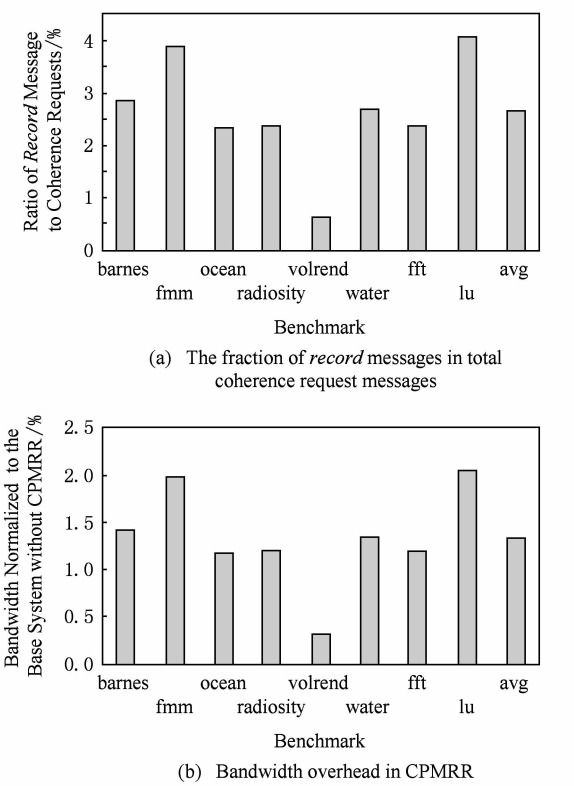

Fig. 8 The coherence overhead in CPMRR.

图 8 CPMRR 在一致性协议方面的开销

重演时,CPMRR 因为采用了计数信号量机制,仅在执行到发生方时广播唤醒消息,且该消息无需标记其对应的先发生方指令时戳,带来的带宽开销同记录阶段相当,约是 IMRR 重演阶段带宽开销的 20%,如图 9 所示. 因为在 CPMRR 中,只有先发生方执行完毕后才广播 *wakeup* 消息,后发生方收到对应的 *wakeup* 消息后方可继续向前执行,而 IMRR 在所有并发 chunk 执行完毕后都要广播 chunk-end 消息,直至来自所有线程的 chunk-end 消息聚齐后系统才能继续向前执行,相比 IMRR,CPMRR 很大程度上减少了重演时的消息数量.

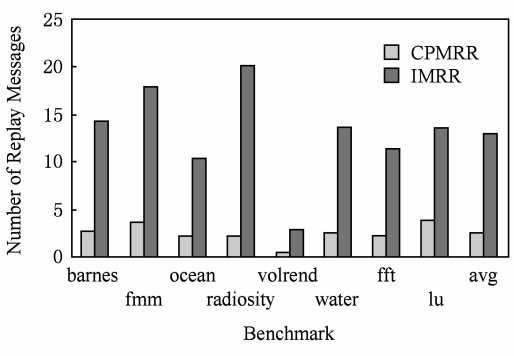

Fig. 9 The number of replay messages in CPMRR.

图 9 CPMRR 在重演阶段消息数量统计

6 结束语

本文为采用监听一致性协议的 CMP 系统设计了一种基于并发式记录策略的点到点内存竞争记录算法. 该算法中将原有的点到点记录策略扩展到了所有线程,解决了点到点记录方式硬件资源消耗较大的问题. 记录日志时采用分布式记录策略,有效降低了带宽开销. 重演时采用简单的生产者消费者模型,引入了计数信号量机制,硬件实现结构简单,添加的一致性消息少,重演速度快. 针对该记录算法在 8 核 CMP 系统中的仿真结果表明:该并发式记录算法在未增加日志的前提下,有效降低了点到点记录方式的硬件资源消耗和带宽开销.

参 考 文 献

[1] Aciicmez O, Seifert J. Cheap hardware parallelism implies cheap security [C] //Proc of the 4th Workshop on FDTC 2007. Los Alamitos, CA: IEEE Computer Society, 2007: 80-91

[2] Xu M, Bodik R, Hill M D. A "flight data recorder" for enabling full-system multiprocessor deterministic replay [C] //Proc of the 30th Int Symp on Computer Architecture (ISCA'03). New York: ACM, 2003: 122-135

[3] Montesinos P, Hicks M, King S T, et al. Capo: A software-hardware interface for practical deterministic multiprocessor replay [C] //Proc of the 14th Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09). New York: ACM, 2009: 73-84

[4] Nima H, Josep T. Replay debugging: Leveraging record and replay for program debugging [C] //Proc of the 41st Int Symp on Computer Architecture (ISCA'14). New York: ACM, 2014: 455-456

[5] Xu M, Bodik R, Hill M D. A regulated transitive reduction (RTR) for longer memory race recording [C] //Proc of the 12th Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06). New York: ACM, 2006: 49-60

[6] Hower D R, Hill M D. Rerun: Exploiting episodes for lightweight memory race recording [C] //Proc of the 35th Int Symp on Computer Architecture (ISCA'08). New York: ACM, 2008: 265-267

[7] Pokam G, Pereira C, Danne K, et al. Architecting a chunk-based memory race recorder in modern CMPs [C] //Proc of the 42nd Int Symp on Microarchitecture (MICRO'09). New York: ACM, 2009: 576-585

[8] Arkaprava B, Jayaram B, Hill M D. Karma: Scalable deterministic record-replay [C] //Proc of the Int Conf on Supercomputing (ICS'11). New York: ACM, 2011: 359-368

[9] Zhu Suxia, Ji Zhenzhou, Liu Tao, et al. Study on memory race recording mechanism in deterministic multi-core replay [J]. Acta Electronica Sinica, 2011, 39 (12): 2748-2754 (in Chinese)
(朱素霞, 季振洲, 刘涛, 等. 面向多核程序确定性重演的内存竞争记录机制研究[J]. 电子学报, 2011, 39(12): 2748-2754)

[10] Zhu Suxia, Ji Zhenzhou, Liu Tao, et al. CCTR: An efficient point-to-point memory race recorder implemented in chunks [J]. Microprocessors and Microsystems, 2012, 36(6): 510-519

[11] Zhu Suxia, Ji Zhenzhou, Wang Qing. An efficient deterministic record-replay with separate dependencies [J]. Computers & Electrical Engineering, 2013, 39(2): 175-189

[12] Zhu Suxia, Ji Zhenzhou, Li Dong, et al. A cyclic memory race recording algorithm implemented with hardware signatures [J]. Journal of Computer Research and Development, 2014, 51(5): 1149-1157 (in Chinese)
(朱素霞, 季振洲, 李东, 等. 基于硬件签名的循环式内存竞争记录算法[J]. 计算机研究与发展, 2014, 51(5): 1149-1157)

[13] Pokam G, Danne K, Pereira C, et al. QuickRec: Prototyping an Intel architecture extension for record and replay of multithreaded programs [C] //Proc of the 40th Int Symp on Computer Architecture (ISCA'13). New York: ACM, 2013: 643-654

[14] Chen Y, Hu W, Chen T, et al. LReplay: A pending period based deterministic replay scheme [C] //Proc of the 37th Int Symp on Computer Architecture (ISCA'10). New York: ACM, 2010: 187-197

[15] Gao X, Chen Y J, Wang H D, et al. System architecture of Godson-3 multi-core processors [J]. Journal of Computer Science and Technology, 2010, 25(2): 181-191

[16] Netzer R H B, Miller B P. What are race conditions?: Some issues and formalizations [J]. ACM Letters on Programming Languages and Systems (LOPLAS), 1992, 1(1): 74-88

[17] Netzer R H B. Optimal tracing and replay for debugging shared-memory parallel programs [C] //Proc of the ACM/ONR Workshop on Parallel and Distributed Debugging (PADD'93). New York: ACM, 1993: 1-11

[18] Lamport L. Time, clocks, and the ordering of events in a distributed system [J]. Communications of the ACM, 1978, 21(7): 558-565

[19] Martin M M K, Sorin D J, Beckmann B M, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset [J]. Computer Architecture News, 2005, 33(4): 92-99

[20] Woo S C, Ohara M, Torrie E, et al. The splash-2 programs: Characterization and methodological considerations [C] // Proc of the 22nd Int Symp on Computer Architecture (ISCA'95). New York: ACM, 1995: 24-36

Zhu Suxia, born in 1978. PhD. Associate professor in Harbin University of Science and Technology. Her research interests include cache coherence protocol, concurrent bug detection, and parallel computing.

Chen Deyun, born in 1962. Professor and PhD supervisor in Harbin University of Science and Technology. His main research interests include image processing, detection and imaging.

Ji Zhenzhou, born in 1965. Professor and PhD supervisor in Harbin Institute of Technology. His main research interests include parallel architecture, parallel computing and network security.

Sun Guanglu, born in 1979. PhD. Professor in Harbin University of Science and Technology. His main research interests include network security, pattern recognition and machine learning.

Zhang Hao, born in 1981. PhD and associate professor in Institute of Computing Technology, Chinese Academy of Sciences. His research interests include high throughput CPU microarchitectures.