doi:10.3772/j.issn.1002-0470.2017.06.001

一种基于硬件的快速确定性重放方法①

章隆兵②[*,**,***,****] 李 磊[*****] 肖俊华[*,**,***,****] 贺 晓[*,**,***,****] 王 剑[*,**,***,****]

([*]计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

([**]中国科学院大学 北京 100049)

([***]中国科学院计算技术研究所 北京 100190)

([****]龙芯中科技术有限公司 北京 100190)

([*****]华为技术有限公司 深圳 518129)

摘 要 针对多核处理器上并行程序执行不确定性所造成的并行调试难问题,提出了一种基于硬件的快速确定性重放方法——时间切割者。该方法采用面向并行的记录机制来区分出原执行中并行执行的访存指令块和非并行执行的指令块,并在重放执行中避免串行执行那些在原执行中并行执行的访存指令块,从而使得重放执行的性能开销小。在多核模拟器 Sim-Godson 上的仿真实验结果表明:该方法的重放速度快,其性能开销仅为 2% 左右。此外,该方法还具有硬件支持简单特点,未来有望应用于国产多核处理器研制中。

关键词 多核处理器,并行调试,确定性重放,多核模拟器,全局时钟

0 引 言

随着多核处理器的发展,并行程序调试受到很大关注。并行程序执行具有不确定性,在某次执行中暴露出来的错误在下次执行中可能不会出现,这使得调试困难。学术界提出了用确定性重放(deterministic replay)来复现并行程序的执行。确定性重放在原执行中记录并行程序的执行轨迹作为日志,并在重放执行中根据记录的日志确定性重放出原执行的执行轨迹。除了用于并行程序调试,确定性重放还广泛用于其他应用,如网络入侵检测、故障容忍等。

已有确定性重放方法可分为纯软件的确定性重放和硬件的确定性重放。纯软件的确定性重放方法在并行程序中插入一些指令以检测并记录访存指令之间的序关系,给并行程序的原执行带来极高的性能代价。例如,确定性重放及再制造分析框架 PinPlay[1] 通过二进制注入工具在原程序中插入一系列检测的指令,使得原执行的性能仅为正常执行(不支持确定性重放的执行)时的 1/117。与之对应的,硬件的确定性重放方法通过在多核处理器上加入一定的硬件支持,可以大幅降低原执行时的性能开销。本文主要研究硬件确定性重放方法。

目前已有硬件确定性重放方法在原执行时几乎没有性能开销,但是其重放执行的性能开销往往很大,成为确定性重放系统的性能瓶颈。例如,对于网络入侵检测的应用,快速的重放执行可以快速地分析来自网络的入侵,并且快速追踪到入侵源[2,3]。对于故障容忍的应用,一个暂态故障可以通过比较原执行和重放执行时系统的状态以检测并容忍[4]。当原执行比较高效时,整个故障容忍系统的性能取决于重放执行时的性能。此外,即使对于传统的并行程序调试,高效重放执行也可以使一个错误更快

① 国家"核高基"科技重大专项课题(2009ZX01028-002-003, 2009ZX01029-001-003, 2010ZX01036-001-002, 2012ZX01029-001-002-002),国家自然科学基金(61221062, 61232009, 61222204)和 863 计划(2012AA010901)资助项目。

② 男,1974 年生,博士,副研究员;研究方向:计算机系统结构;E-mail: lbzhang@ict.ac.cn

(收稿日期:2016-12-09)

的复现出来。

本文提出了一种快速硬件确定性重放方法——时间切割者,它能够实现快速的重放执行。该方法采用面向并行的记录(concurrency oriented recording,COR)机制来区分出原执行中并行执行的访存指令块和非并行执行的指令块,对于原执行中的并行执行的访存指令块,允许其在重放执行中也并行执行,从而获得很高的重放执行速度。

1 相关工作

本节简介目前已有硬件确定性重放方法,即飞行数据记录器(flight data recorder,FDR)[5],它在条缓存块行中加入了额外的位来记录每一条访存指令之间的序关系。为了减少额外位的开销,Strata[6]在多核处理器中加入了一个全局的层,使得每条缓存行上只需要 1 位的额外位。但当处理器核数变大时,Strata 记录的日志规模会急剧变大[7]。此外,FDR 和 Strata 都需要在一级缓存中加入额外的存储空间,而一级缓存是多核处理器的核心部件且处于性能关键路径上,还易于出错。

为了消除对多核处理器上一级缓存的额外存储开销,研究者们提出用访存指令块之间的序关系来代替访存指令之间的序关系。Rerun[7]动态地将一个并行程序的执行分解为多个访存指令块。当一个访存指令块与其他线程中并行执行的访存指令块有交互时,该访存指令块就结束并开始下一个访存指令块。Rerun 记录了每个访存指令块的大小,并用 Lamport 逻辑时钟戳来刻画访存指令块之间的序关系。另一个有代表性的确定性重放工作是 Delorean[8]。Delorean 依赖于 BulkSC 架构[9]来记录访存指令块直接的序关系。在 BulkSC 架构上,当一个访存指令块与其他并行执行的访存指令块有交互时,其执行可以被终止。由于 Delorean 中所有访存指令块的大小都是固定的,所以 Delorean 只需要记录各个访存指令块之间的序关系,而不用记录每个访存指令块的大小。因此,Delorean 记录的日志大小比 Rerun 要小。但是,Delorean 的缺点是很难用于除 BulkSC 以外的架构,而 BulkSC 架构并不是传统的处理器架构。

为了得到更好的兼容性以及更小的日志,研究者提出了 Timetraveler[10]和 LReplay[11]。Timetraveler 通过找出非循环的访存指令竞争来扩大每个访存指令块的规模。由于访存指令块的规模变大,Timetraveler 中访存指令块的数量会变少,使得 Timetraveler 有更小的日志。LReplay 通过固定每个访存指令块的规模来减少需要记录的日志大小。在重放执行中,Timetraveler 和 LReplay 都将它们的实现放在单核处理器上,因为它们都无法记录原执行中并行执行的访存指令块之间的某些序关系,必须通过在单核处理器上的顺序重放来保证这些未记录的序关系。因此,对于一个 N 线程的并行程序,Timetraveler 和 LReplay 会给其重放执行带来将近 N 倍的性能损失。

虽然已有的硬件确定性重放方法在日志大小上做出了很大成绩,但是在重放速度上依然有较大的开销。主要原因在于,原执行中并行执行的访存指令块在重放执行中只能串行执行或者交替执行。与已有的所有硬件确定性重放方法不同的是,本文提出的时间切割者允许在原执行中的并行执行的访存指令块在重放执行中依然并行执行,从而大大的提高重放执行的性能。

2 时间切割者

2.1 基本思想

时间切割者的基本思想是:采用面向并行的记录(COR)机制来区分出原执行中并行执行的访存指令块和非并行执行的指令块,对并行执行的访存指令块,允许其在重放执行中也并行执行。在原执行中,对于任意一对非并行执行的访存指令块,COR 为其指派一条访存指令块之间的序关系;而对于任意一对并行执行的访存指令块,COR 将直接记录两个访存指令块内部指令之间的访存指令序关系。在重放执行中,由于并行执行的访存指令块之间并没有序关系,所以并行执行的访存指令块依然可以并行的被执行。与传统的硬件的确定性重放方法相比,COR 可以大大提高确定性重放时重放执行的性

能。

2.2 时间切割者的原执行

在原执行中,时间切割者采用 COR 机制区分并行执行的访存指令块和非并行执行的访存指令块,记录访存指令块的次序。

2.2.1 确定并行执行的访存指令块和非并行执行的访存指令块

采用 COR 机制的关键是需要一种高效方法来确定并行执行的访存指令块和非并行执行的访存指令块。本文利用全局时钟[11]来定义访存指令块以及访存指令块之间的序关系。在一个有全局时钟的多核处理器上,每条指令可以指定一个开始时间和一个结束时间,从开始时间到结束时间的时间区间内必须包括这条指令全局提交的时间点。采用一个简单的周期采样工具来得到每条指令的开始时间和结束时间。下面给出访存指令 u 的开始时间和结束时间、访存指令块的定义。

定义 1 指令的开始时间和结束时间:对于一条访存指令 u,其开始时间是指在 u 取指之前的最近的采样时间,其结束时间是指在 u 提交之后的第一个采样时间。

定义 2 访存指令块:每个访存指令块都由同一个线程上一串连续的访存指令 $u,\cdots,u'$ 组成,其中这些访存指令有着相同的结束时间。例如,线程 i 上的第 k 个访存指令块由线程 i 上所有结束时间是第 k 个采样时间的指令组成。

下面先给出程序原执行中一个访存指令块的并行区间的定义,然后给出任意两个访存指令块是否是并行执行的判断依据。

定义 3 并行区间:原执行中一个访存指令块的并行区间是一个时间区间,在原执行的这段时间区间内,该访存指令块中的指令可以与其他线程上的指令并行执行。例如,假设每两次采样之间的时间间隔足够长(如 $M=512$ 个时钟周期),那么每个线程第 k 个访存指令块的并行区间就是$[(k-2)M, kM](k>=2)$。

定义 4 并行执行的访存指令块:对任意两个访存指令块 A 和 B,如果 A 的并行区间和 B 的并行区间是没有交叠的,则 A 和 B 就是一对非并行执行的访存指令块;反之,如果 A 的并行区间和 B 的并行区间是有交叠的,则 A 和 B 就是一对并行执行的访存指令块。

在 COR 机制中,对两个并行执行的访存指令块不分配其访存指令块之间的序,而是直接记录这两个并行访存指令块中访存指令之间的序关系。而对两个非并行执行的访存指令块,分配其访存指令块之间的序,来涵盖这两个访存指令块之间的访存指令之间的序关系。图 1 给出了一个例子来解释并行

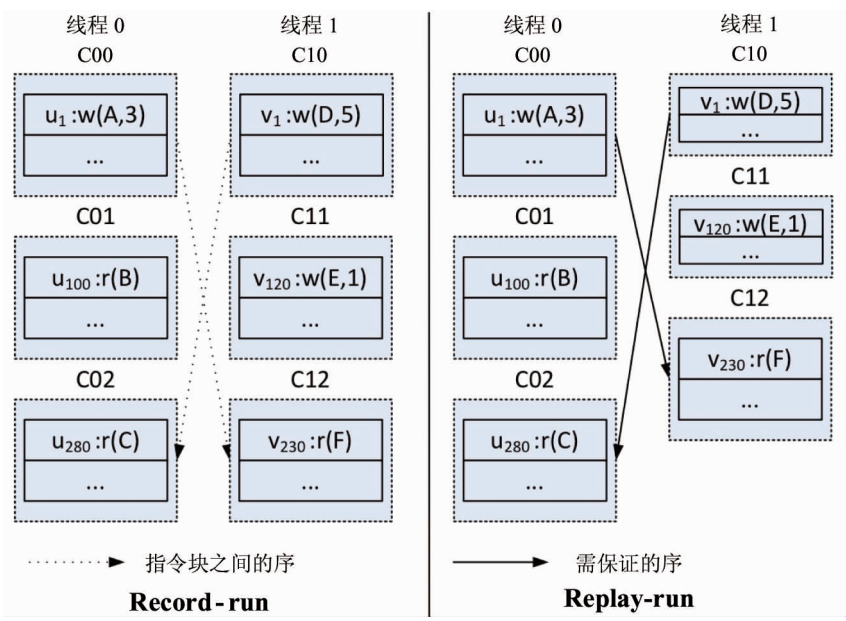

图 1 时间切割者的原执行和重放执行

执行的访存指令块。在原执行中,访存指令块 Cij 由线程 i 上的一串连续的访存指令组成,其中每个访存指令的结束时间都是第 $(j+1)M$ 拍(也就是说,这些指令的提交时间都是在第 jM 拍到第 $(j+1)M$ 拍之间的)。访存指令块 C00/C10 和访存指令块 $C12/C02$ 是非并行执行的访存指令块,因为它们的并行区间分别是 $[0,M]$ 和 $[M,3M]$。在原执行的记录中,COR 机制记录两条访存指令块之间的序:$C00 \rightarrow C12$ 和 $C10 \rightarrow C02$。在重放执行中,任何一对并行执行的访存指令块(如指令块 $C01$ 和 $C11$),都可以并行执行。实际上,只有少量的访存指令块需要被暂停来保证访存指令块之间的序。因此与已有的硬件确定性重放方法相比,COR 机制可以大大减少重放执行时的性能开销。

2.2.2 记录非并行执行访存指令块之间的序关系

其具体如图 2 所示。

对于非并行执行的访存指令块,比较并行区间就可以判定它们之间的序关系,因此记录非并行执行访存指令块之间序关系的主要任务就是记录每个访存指令块的并行区间。本文用一个简单采样工具来得到每个访存指令块的并行区间。该工具周期性采样每个线程在采样间隔内提交的访存指令数,从而将每个线程上的访存指令切割成访存指令块。线程 i 上的第 k 个访存指令块由线程 i 上所有提交时间在第 $(k-1)$ 次采样到第 k 次采样的所有访存指令组成。当两次采样的间隔 M 足够长时,每个线程第 k 个访存指令块的并行区间就是 $[(k-2)M,kM]$ (当 $k>=2$)。根据定义 4 可以得出以下结论:对于线程 i 的第 k 个指令块和线程 j 的第 k' 个指令块,它们之间有一个指令块的序关系当且仅当 $|k-k'| \geqslant 2$ 时成立。

因此需要记录的主要信息就是每个访存指令块的大小,即每个访存指令块中的访存指令数。如算法 1 所示,用一个计数器来记录每个采样间隔中提交的访存指令数量。每当进行一次采样时,计数器的值被记录为日志并将计数器清空为 0。通过记录的信息,可以判断出访存指令块分块的方法,以及每个访存指令块的并行区间。

```
Module record _ blocki begin
  Reg[8:0] commit _ mem _ increment;  /* 计数器 */
  If(commit _ meminst) then
      /* 当一个访存指令被提交时 */
      Commit _ mem _ increment + +;
  If (sampling) then
    /* 当进行一次采样时 */
    log(commit _ mem _ increment,i);
    /* 将计数器的值记为日志 */
    Commit _ mem _ increment = 0;
    /* 重新将计数器清零 */
```

图 2 记录非并行执行访存指令块之间的序关系

2.2.3 记录并行执行访存指令块之间的序关系

对于并行执行的访存指令块,COR 机制直接记录这些指令块中冲突的访存指令之间的序关系。当线程 i 的第 k 个访存指令块中的指令 u 正在执行时,时间切割者需要找到其并行执行的访存指令块中所有的冲突指令,也就是其他线程的第 k 和 $(k-1)$ 个访存指令块中的冲突指令。本文采用布隆过滤器(bloom filter)[12]来检测冲突访存指令,具体实现为每个处理器核分配 4 个布隆过滤器,其中 2 个写和 2 个读布隆过滤器分别保存最近一个(第 $(k-1)$ 个)和当前访存指令块(第 k 个)中所有写指令和读指令的访存地址集合。过程如下:当一个处理器核在执行写指令 u 时发生了一个本地的一级缓存失效,写指令 u 的地址就会在其他处理器核上的写和读布隆过滤器上查找。如果写指令 u 的地址命中了处理器核 i 中保存最近访存指令块的访存地址的布隆过滤器,那么处理器核 i 的最近访存指令块中的某个指令就与 u 冲突,则记录一条访存指令序 $v \rightarrow u$,其中 v 是处理器核 i 中最近的访存指令块最后的一个访存指令;如果写指令 u 的地址命中了处理器核 i 中保存当前访存指令块的访存地址的布隆过滤器,那么处理器核 i 的当前访存指令块中的某个指令就与 u 冲突,则记录一条访存指令序 $v \rightarrow u$,其中 v 是处理器核 i 中最近提交的访存指令。实际上由于绝大多数的访存指令序都已经被访存指令块之间的序涵盖,所以只有很少的访存指令序需要被记录。

2.3 时间切割者的重放执行

在原执行中,记录了以下两种信息:非并行执行访存指令块的块间序关系和并行执行访存指令块之间的访存指令序关系。重放执行需要将这两种信息重现出来,才能保证与原执行有一样的逻辑行为。

2.3.1 重现非并行执行访存块间的序关系

在重放执行中,所有原执行中记录的非并行执行访存指令块的块间序关系都需要被重现出来。具体来说,在重放执行中,对于任何一个访存指令块A,只有所有在A块间序关系之前的访存指令块都已经提交了之后,A才能够被执行。图3给出了重现非并行执行访存指令块的块间序关系的算法。其中,$grant(s)$是为每一次的采样时间s安排的计数器。[$curr\_start$,$curr\_end$]和[$next\_start$,$next\_end$]分别表示该线程上当前访存指令块和下一个访存指令块的并行区间。当一个访存指令块执行完成时,从$curr_end$到$next_end$之间的采样时间对应的计数器的值都会加1。另一方面,当一个新的访存指令块准备执行时,需要查看其开始时间$next_start$对应的计数器,也就是算法中的$grant(next_start)$。当且仅当该计数器的值等于程序线程的总数时,这个访存指令块才允许被执行。

```
module replay_p;
begin
    reg[7:0] curr_start, [7:0] curr_end, [7:0] next_start, [7:0] next_end;
    reg[7:0] next_start_new, [7:0] next_end_new;
    /* 当一个块结束时用来更新 next_start 和 next_ned*/
    if (end_a_block) then
        for (s=curr_end; s<next_end; s++) do
            grant(s)++;
        curr_end=next_end;
        next_end=next_end_new;
    if (start_a_block) then
        if (grant(next_start)==p) then
            /*p 是程序的线程总数 */
            start_a_new_block;
            /* 开始该访存指令块的执行 */
            next_start=next_start_new;
        else
            pause_the_progress;
        /* 暂停该访存指令块的执行 */
```

图3 重现非并行块执行访存指令块的块间序算法

2.3.2 重现并行执行访存指令块之间的访存指令序关系

在重放执行中,所有原执行中直接记录的并行执行访存指令块之间的访存指令序关系需要被重现出来。在实现中,给每个处理器核增加一个访存指令序的缓冲器来导入直接记录的访存指令序信息。根据实验结果,每执行10,000条指令,需要被直接记录的访存指令序平均不到一条。因此,只需要一个很小的缓冲器(如32个字节)就足够周期性的导入并缓存这些直接记录的访存指令序信息。

此外,还需要给每个处理器核安排一个一位的标志寄存器来暂停执行。假设对于一条访存指令序$u \rightarrow v$,其中u和v分别在处理器核0和1上执行。在重放执行时,处理器核1不能执行指令v,直到指令u已经被处理器核0提交了。如果指令u还没有被提交,标志寄存器就被置为1以暂停处理器核1的执行。一旦处理器核0提交了指令u,处理器核0就会通知处理器核1并将上述标志寄存器的值置为0。这样,所有直接记录的并行执行访存指令块之间的访存指令序关系都可以被如实的重现出来。

3 仿真实验及结果分析

本文将时间切割者在中科院计算所开发的分片式多核模拟器Sim-Godson[13]上进行了模拟实现。该模拟器模拟了基于MESH片上网络互连的同构

多核结构,逻辑上共享二级缓存,但物理上二级缓存是分布的,分体与每个处理器核连在一起。本文采用 splash2 并行测试程序集[14]进行了仿真实验,所采用的模拟器参数和测试集参数分别如表 1 和表 2 所示,实验过程中的周期采样间隔是 512 个时钟周期。

表 1 模拟器参数设置

参数	描述
核数	4
处理器核流水线	4 发射,9 级流水,乱序执行
L1 dcache	私有,大小 32kB,每个缓存行 32B
L1 icache	私有,大小 64kB,每个缓存行 32B
L2 cache	共享,4 个体,每个体为 512kB,每个缓存行 32B
布隆过滤器	每核 4 ×1024 位

表 2 splash2 benchmark 参数

cholesky	d750.0
fft	524288、双精度复数
radix	262144 键的数目
contiguous _ blocks lu	512 ×512
non _ contiguous ocean	258 ×258
barnes	16384
raytrace	精度 128 ×128、块大小 8 ×8

3.1 重放功能验证

在时间切割者的功能正确性验证方面,对比原执行和重放执行的所有访存指令执行轨迹是没有必要的。因为在重放执行过程中能够确保非并行块之间是串行执行的,其潜在的访存指令执行序能够得以保证,因此只需要保证并行块间的冲突访存指令序关系,确定原执行中冲突访存指令序是否在重放执行中重现。具体实现方法:对于每一条冲突访存指令序,如 $u -> v$; 找到 u 所在处理器核对应该条指令的提交时间,以及 v 所在核的提交时间。如果 u 的时间在 v 指令之前,则一致;反之,重放过程未能保证其冲突访存指令序关系。

表3为运行splash2测试集得出的结果,可以看出时间切割者的重放执行能够重现原执行中冲突访存指令序关系,正确实现重放功能,并且原执行具有较小的日志大小。在正常使用中,重放执行是不需要日志的,只是为了验证重放执行的正确性才需要进行记录。

表 3 功能测试结果

测试程序	并行程序原执行日志大小(单位:MB)	并行程序重放执行日志大小(单位:GB)	冲突访存指令序是否一致
Cholesky	8.8	3.5	是
FFT	4.425	3.21	是
Radix	1.98	0.83	是
LU	8.85	7.6	是
Ocean	1.45	0.23	是
Barnes	1.75	0.41	是
Raytrace	6.8	2.1	是

3.2 性能测试

为了测试确定性重放的性能,主要对比原执行和重放执行的运行周期。本文采用处理器核运行的指令拍数来表示并行程序的执行时间。由于并行程序重放过程中需要根据原执行日志记录来引导执行,相应的重放时间比原执行长。因此采用(重放执行处理器核拍数/原执行处理器核拍数 - 1) × 100% 来计算重放性能开销。其结果如表 4 所示,可见重放执行对原执行造成的性能开销很小,平均只有约 2%,几乎可以忽略。

表 4 确定性重放性能测试结果

测试程序	原执行周期(单位:万拍)	重放执行周期(单位:万拍)	重放性能开销
Cholesky	45936.3	46078.8	0.31%
FFT	40682.1	40928.3	0.61%
Radix	5251.6	5329.6	1.49%
LU	34057.6	34446.8	1.14%
Ocean	4319.8	4422.8	2.38%
Barnes	5652	5653.2	0.02%
Raytrace	36784	36785.2	0.003%

4 结 论

本文针对多核处理器上并行程序执行不确定性问题,提出一种快速的硬件确定性重放方法——时间切割者,并在多核模拟器上进行了模拟器实现。该方法通过在原执行中区分并行执行的访存指令块和非并行执行的访存指令块,并在重放执行中避免串行执行那些在原执行中并行执行的访存指令块,获得了高重放速度。仿真实验结果表明,该方法的重放速度很快,其性能开销仅为2%左右。本文方法的硬件开销也不大,未来工作主要是进行充分评估和验证,实现到某款国产多核处理器中。

参考文献

[1] Patil H, Pereira C, Stallcup M, et al. Pinplay: a framework for deterministic replay and reproducible analysis of parallel programs. In: Proceedings of the International Symposium on Code Generation and Optimization, Ontario, Canada, 2010. 2-11

[2] King S, Chen P. Backtracking intrusions. In: Proceedings of the 19th Symposium on Operating Systems Principles, New York, USA, 2003. 223-236

[3] Dunlap G, King S, Cinar S, et al. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. In: Proceedings of the 2002 Symposium on Operating System Design and Implementation, Massachusetts, USA, 2002. 211-224

[4] Lucchetti D, Reinhardt S, Chen P. ExtraVirt: detecting and recovering from transient processor faults. In: Proceedings of the 20th Symposium on Operating System Principles, Brighton, UK, 2005. 1-8

[5] Xu M, Bodik R, Hill M. A flight data recorder for enabling full-system multiprocessor deterministic replay. In: Proceedings of the 30th International Symposium on Computer Architecture, San Diego, USA, 2003. 122-135

[6] Narayanasamy S, Pereira C, Calder B. Recording shared memory dependencies using Strata. In: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, USA, 2006. 229-240

[7] Hower D, Hill M. Rerun: exploiting episodes for lightweight memory race recording. In: Proceedings of the 35th International Symposium on Computer Architecture, Beijing, China, 2008. 265-276

[8] Montesinos P, Ceze L, Torrellas J. DeLorean: recording and deterministically replaying shared-memory multiprocessor execution effciently. In: Proceedings of the 35th International Symposium on Computer Architecture, Beijing, China, 2008. 289-300

[9] Ceze L, Tuck J, Montesinos P, et al. BulkSC: bulk enforcement of sequential consistency. In: Proceedings of the 34th International Symposium on Computer Architecture, San Diego, USA, 2007. 278-289

[10] Voskuilen G, Ahmad F, Vijaykumar T. Timetraveler: exploiting acyclic races for optimizing memory race recording. In: Proceedings of the 37th International Symposium on Computer Architecture, Saint Malo, France, 2010. 198-209

[11] Chen Y, Chen T, Hu W. Global clock, physical time order and pending period analysis in multiprocessor systems. http://arxiv. org/pdf/0903. 4961: CoRR abs/0903.4961, 2009

[12] Bloom B. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970, 13(7):422-426

[13] 黄琨,马可,曾洪博等. 一种分片式多核处理器的用户级模拟器. 软件学报,2008, 19(04):1069-1080

[14] Woo S, Ohara M, Torrie E, et al. The SPLASH-2 programs: characterization and methodological considerations. In: Proceedings of the 22th International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, 1995. 24-36

A fast deterministic replay based on hardware

Zhang Longbing[*,**,***,****], Li Lei [*****], Xiao Junhua[*,**,***,****], He Xiao[*,**,***,****], Wang Jian [*,**,***,****]

([*] State Key Laboratory of Computer Architecture(Institute of Computing Technology, Chinese Academy of Sciences), Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

([***] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([****] Loongson Technology Corporation Limited, Beijing 100190)

([*****] Huawei Technologies Corporation Limited , Shenzhen, 518129)

Abstract

To solve multicore processors' difficulty in debug of parallel programs caused by the nondeterminacy in parallel program execution, a fast hardware based deterministic replay method, called Time Slicer, was proposed. Time slicer adopts the concurrency oriented recording mechanism to distinguish parallel execution memory instruction blocks(PEMIB) from serial execution memory instruction blocks(SEMIB) in the record execution period. In the replay execution period, time slicer runs the PEMIB in parallel, which makes the replay efficient. The time slice was implemented on Sim-Godson, a multiprocessor simulator, and the experiment results showed that the performance cost of replay execution was only 2%. Since this method requires little hardware-support, it is expected to be a promising method applicable for domestic multi-core processors.

Key words: multicore processor, parallel debug, deterministic replay, multicore simulator, global clock