MDCI:基于多粒度动态控制流不变式的硬件故障局部化

郑衍松[1,2],佟 冬[1],王克义[1],程 旭[1]
(1.北京大学微处理器研发中心,北京 100871;2.北京大学深圳研究生院,广东深圳 518055)

摘 要: 本文提出了一种基于多粒度动态控制流不变式的硬件故障局部化方法 MDCI.该方法基于预先提取的置信度较高的各种粒度动态控制流不变式,多粒度逐级迭代地检验控制流不变式程序点是否可达,从而将与硬件故障相关的代码范围局部化.实验结果表明 MDCI 只需检验少量的控制流程序点,就能准确地将与故障相关的代码范围局部化.

关键词: 多粒度;动态控制流不变式;硬件故障;故障局部化
中图分类号: TP306+.3 **文献标识码:** A **文章编号:** 0372-2112(2010)11-2465-06

MDCI:Hardware Fault Localization Based on Multi-Granularity Dynamic Control Flow Invariants

ZHENG Yan-song[1,2],TONG Dong[1],WANG Ke-yi[1],CHENG Xu[1]
(1.*Microprocessor Research and Development Center*,*Peking University*,*Beijing* 100871,*China*;
2.*Shenzhen Graduate School*,*Peking University*,*Shenzhen*,*Guangdong* 518055,*China*)

Abstract: This paper proposes a hardware fault localization approach,MDCI,which is based on multi-granularity dynamic control flow invariants.The approach first extracts high confidence dynamic control flow invariants in different granularity.It then checks gradually and iteratively in multi-granularity whether the dynamic control flow invariants are accessible,and localizes the code scope related to hardware faults.The experimental result shows that MDCI can localize the code scope related to hardware faults with high accuracy by checking only a few of control flow program points.

Key words: multi-granularity;dynamic control flow invariants;hardware faults;fault localization

1 引言

大规模系统芯片(System-on-Chip,简称 SoC)设计验证已成为系统开发的重要挑战[1].在硬件系统上运行已有大型商业软件,是有效的系统级验证手段[2].虽然通过运行真实商业软件能够有效地验证硬件系统,但是由于系统开发初期缺乏踪迹输出等高级调试功能,一旦硬件故障导致大型软件运行失效[3],则硬件故障局部化(fault localization[4])效率很低.本文针对该问题展开研究.

研究人员针对硬件故障诊断提出了诸多相关工作,但是现有的技术和研究难以有效地支持导致商业软件运行失效的硬件故障局部化.由于商业软件的状态空间巨大,同时现有的软硬件调试工具只能观测有限的硬件系统状态,所以这些调试工具无法有效地支持导致大型商业软件运行失效的硬件故障局部化.在研究方面,基于专门硬件设计的硬件故障检测诊断研究需要专门硬件的支持,其应用成本较高.基于异常软件行为分析的硬件故障诊断研究[5,6],虽然应用成本低、使用灵活,但是它依赖于特定的硬件环境,只针对规模较小的评测程序,难以适用导致大型商业软件运行失效的硬件故障局部化.此外,这些研究只针对处理器核故障,而本文研究主要针对 SoC 故障,旨在提高当多个 IP(Intellectual Property)核集成时出现的硬件故障局部化效率.

现有的故障局部化手段需要检验大量的控制流程序点,来缩小故障可疑范围,并且其效率和准确率较低.为了提高导致大型商业软件运行失效的硬件故障局部化效率,本文提出一种基于多粒度动态控制流不变式的硬件故障局部化方法 MDCI(Multi-granularity Dynamic Control flow Invariants extractor and checker).

收稿日期:2009-09-09;修回日期:2010-01-10
基金项目:国家高技术研究发展计划(863计划)课题(No.2006AA010202);国际科技合作项目(No.2008DFB10010)

2 MDCI 方法

本节介绍 MDCI 方法的设计思想、流程,以及多粒度动态控制流不变式提取和检验的实现.

2.1 设计思想

硬件故障局部化是将与硬件故障相关的代码或模块缩小到较小范围的过程,是提高故障定位效率的关键环节.由于大型商业软件的代码规模和各种粒度控制流信息庞大,所以故障局部化过程非常困难.动态控制流不变式能够有效地缩小与故障相关的代码范围.虽然商业软件的运行存在不确定因素,如系统初始状态和外部输入等,都会影响每次系统运行控制流的一致性,但是商业软件本身存在一些在程序正常运行时必需经历、且不因系统运行的不确定性而改变的控制流程序点,即程序关键点.关键点具有在每次程序运行时有且唯一、次序固定的特点:有且唯一是指关键点每次都被执行,且每次只被执行一次;次序固定是指每次运行时,关键点与其相邻程序点的位置次序固定.动态控制流不变式是指在多次参考运行中都成立的控制流关键点[7].由于动态控制流不变式也具备有且唯一的特点,所以一旦检测到某动态控制流不变式程序点没被执行则认为该程序点之前存在错误,一旦检测到该程序点被执行则认为之前的执行路径是正确的.而动态控制流不变式次序固定的特点,则使得只要依次检验动态控制流不变式程序点是否可达,就能得知哪个区间存在错误.为了进一步减少动态控制流不变式程序点的检验次数,更快地缩小可疑范围,更准确地逼近故障位置,我们提出了基于多粒度动态控制流不变式逐级迭代局部化的方法 MDCI.如图 1 所示,MDCI 逐级迭代地检验已知关键点、代码段切换(x86 体系结构)、函数调用、条件跳转四种粒度的控制流不变式,将与硬件故障相关的代码范围局部化.

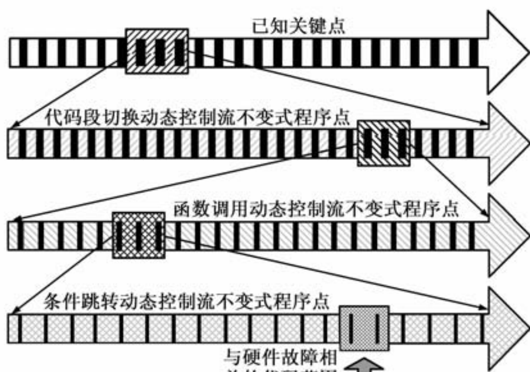

图1 多粒度逐级故障局部化

虽然大型商业软件中的各种粒度控制流数量非常多,以本文研究的典型商业操作系统 Windows XP 为例,它启动过程中动态执行的代码段切换约达 50 万次,函数调用数超过 5 千万个,但是,其动态控制流不变式数量却较少.通过实验可知,Windows XP 的代码段切换动态控制流不变式个数只有 35 个,是原始所有代码段切换个数的 1.5 万分之一,而函数调用动态控制流不变式个数为 1659 个,只占所有函数调用个数的 3 万分之一.并且,因为 MDCI 采用逐级迭代局部化的方法,所以 MDCI 方法所需检验的程序点数量将远小于已知关键点、代码段切换、函数调用、条件跳转(可疑范围间)四种粒度动态控制流不变式数量的总和.除了已知关键点检验外,其余各级的动态控制流不变式检验只需在上一级已确定的可疑范围中进行,这避免了从头开始进行所有粒度的动态控制流不变式检验.

多粒度逐级迭代局部化方法可以大大减少动态控制流不变式检验数量,能够缩减动态不变式检验的开销,从而提高效率. MDCI 是现有调试技术的有力补充,尤其是针对在商业操作系统启动过程中调试手段匮乏的硬件故障局部化.MDCI 也存在不足之处:主要针对导致控制流异常的硬件故障,而不考虑导致隐式数据破坏(silent data corruption)的硬件故障;动态控制流不变式检验需要多次故障运行;且只能将故障范围缩小到引发错误执行路径的代码范围,至于更详细的硬件故障根源,则需借助更底层的硬件调试手段.

2.2 MDCI 描述

MDCI 基于两个假设条件.其一,存在一个与目标测试系统规格说明相同的模拟参考系统,用于构建正确软件行为的参考运行.其二,目标测试系统支持检查点/重放(checkpoint/replay)功能,该功能可以快速地重新启动用于更细粒度动态控制流不变式检验的系统运行.

MDCI 包括多粒度动态控制流不变式的提取和检验两部分.在故障调试之前,我们预先完成多粒度动态控制流不变式的提取;然后再利用这些离线动态控制流不变式信息,在目标系统的故障运行中进行动态控制流不变式检验,实现硬件故障局部化.如图 2 所示,MDCI 的故障局部化过程包括如下步骤.

步骤 1:预先在参考运行中提取大型商业软件运行时的多粒度动态控制流不变式,包括已知关键点、代码段切换、函数调用三种粒度的动态控制流不变式.已知关键点是根据程序行为分析得出,是程序执行过程必经的程序点.代码段切换粒度较大,数量较少,一般发生在启动阶段或者内核态/用户态切换时刻.函数调用使用频繁,是分析程序执行路径较为常用的控制流.

步骤 2:根据置信度的要求,判断提取的动态控制流不变式是否已经完备.如果已经完备,则继续后续步骤,否则重复步骤 1,继续提取动态控制流不变式,提高

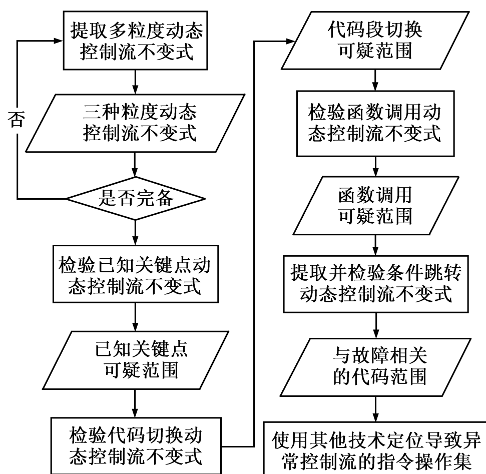

图2　MDCI流程

其置信度.

步骤 3:从故障运行开始处检验已知关键点是否可达,找到异常的执行路径区间,即已知关键点可疑范围.

步骤 4:控制流的粒度越细,则定位的故障可疑范围越小.将故障运行的执行回滚到最后可达已知关键点,在步骤 3 得出的已知关键点可疑范围中,继续检验代码段切换动态控制流不变式,找到导致执行路径异常的代码段切换动态控制流不变式程序点,得到更细粒度的代码段切换可疑范围.

步骤 5:将故障运行的执行回滚到最后可达的代码段切换动态控制流不变式程序点,在步骤 4 得出的代码段切换可疑范围中,继续检验粒度更细的函数调用动态控制流不变式,获得最后可达的函数调试动态控制流不变式程序点及其应有后继,得到函数调用可疑范围.

步骤 6:由于条件跳转粒度较小,数量巨大,所以我们只在步骤 5 得出的函数调用可疑范围中提取条件跳转动态控制流不变式.接着,将故障运行的执行回滚到最后可达的函数调用动态控制流不变式程序点,在函数调用可疑范围中,继续检验条件跳转动态控制流不变式程序点是否可达,获得最后可达的条件跳转动态控制流不变式程序点及其应有后继,即导致异常条件跳转的与硬件故障相关的可疑代码范围.

步骤 7:为了进一步细化故障根源,可以采用现有的动态切片技术[8],找到导致异常控制流的指令操作集.

2.3　多粒度动态控制流不变式提取和检验的实现

在动态控制流不变式的提取阶段,我们基于全系统模拟器的插桩功能提取多粒度动态控制流不变式.由于大多商业软件是非开源的,或者开源商业软件过大导致源代码分析困难,所以我们只基于二进制可执行文件来提取目标软件动态控制流不变式.商业软件本身存在一些系统必须执行的关键点,如本文针对的 Windows XP 商业操作系统中,引导扇区的加载位置和执行入口 0x0:0x7c00 等,因此,我们通过分析目标商业软件的程序行为获得已知关键点.在代码段切换和函数调用动态控制流不变式的提取中,我们记录每次参考运行中所有代码段切换和函数调用位置,形成相应踪迹文件;然后在多个参考运行的踪迹文件中提取有且唯一、次序固定的代码段切换控制流和函数调用控制流.大型商业软件执行过程中,动态执行的条件跳转指令非常多,以 Windows XP 为例,Windows XP 正常启动过程所执行的条件跳转指令数约达 4.1 亿.如果将它们全部提取,那对它们的存储和踪迹数据处理将十分低效.因此,我们只针对给定可疑范围,提取条件跳转动态控制流不变式.

由于我们针对的是真实大型商业软件,所以其运行产生的不变式踪迹非常大.为了解决踪迹过大的问题,我们不提取被频繁执行的中断处理程序中的控制流踪迹,同时忽略用户态空间的函数调用,从而减少与硬件故障相关性较小的控制流数量.为了提高动态控制流不变式的提取和检验效率,我们采用简单的动态控制流不变式表示.如图 3 所示,动态控制流不变式表示只包含控制流的位置信息,即代码段寄存器 CS 和指令指针寄存器 EIP.为了支持多粒度动态控制流不变式逐级地缩小故障范围,我们在保存动态控制流不变式程序点位置时,标识每个动态控制流不变式程序点的控制流粒度信息,包括 POINT、CS、CALL 和 JCC.

```
……
0xc000:0x3          POINT
0x3000:0x8000       CS
0xf000:0xb580       CALL
……                  JCC
0x10:0xfb5af        CS
0x0:0x7c00          POINT
……
```

图 3　动态不变式踪迹格式

在动态控制流不变式检验阶段,我们利用现有调试工具,如 WinDbg 和 FS2 调试器,检验动态控制流不变式程序点是否可达.MDCI 先以粗粒度动态控制流不变式程序点作为断点,检验程序点是否可达,如可达则继续下一个动态控制流不变式程序点,否则记录最后可达的程序点地址.然后采用检查点/重放(checkpoint/replay)技术将执行自动回滚到最后可达的正确程序点,从最后可达程序点处开始进行更细粒度的动态控制流不变式检验,从而将与故障相关的异常代码范围局部化.为了进一步细化故障根源,可以使用动态切片技术进一步诊断导致异常控制流的相关系统状态,并找到导致异常控制流的指令操作集.

3 实验

本节介绍实验环境,MDCI 所需检验程序点数量和故障局部化准确度,MDCI 在 PKUnity-86 系统[※]中的真实应用,动态控制流不变式的完备性与参考运行次数的关系,以及 MDCI 的开销.

3.1 实验环境

• 运行环境:

表 1 实验环境

实验资源	配置或版本
宿主机	处理器:Intel Core2 Quad 2.83GHz 内存:8GB 操作系统:Fedora Core 9
全系统模拟器 Bochs-P86	处理器:100MHz 内存:256MB
硬件环境	PKUnity-86 FPGA 原型:33MHz
客户操作系统	Redhat 9 Windows XP SP2

本文采用指令级的全系统模拟器 Bochs-P86 作为模拟参考系统.Bochs-P86 基于 Bochs 2.3.7 实现,遵循 PKUnity-86 系统的规格说明,速度可以达到 100 MIPS (Millions of Instructions Per Second)[9].Bochs-P86 可以模拟一个完整的计算机系统,包括 x86 兼容处理器和 Windows 操作系统兼容的外设,并能够支持微软公司的各类操作系统和应用软件.如表 1 所示,本文实验环境基于 Linux/x86 主机,并在主机上运行全系统模拟器 Bochs-P86,Bochs-P86 上运行 Bochs BIOS 以及 Redhat 9 或 Windows XP 操作系统.真实硬件系统则采用 33MHz 的 PKUnity-86 FPGA 原型系统.

• 故障模型:

本文主要研究功能故障和永久性故障(permanent fault),原因有二:其一,由于芯片规模、磨损和老化不足等原因,功能故障和永久性故障变得越来越重要[10];其二,相对于瞬时故障(transient fault),功能故障和永久性故障有着不同的挑战.它们被屏蔽的可能性更小,能持续地影响软件行为,导致软件运行失效.因此,一旦发现功能故障或永久性故障,开发人员必须快速诊断故障根源并修复它们[3].

故障注入技术是故障诊断研究的有效手段[11],因此我们通过在模拟器中注入故障来评测 MDCI 方法.Goswami 等人[12]的研究表明,硬件故障都可以映射为存储故障,比如,总线中的故障会导致写入存储的位置或值错误、I/O 设备中的故障会体现为存储错误等.另一方面,SoC 系统通常采用成熟的处理器核,而处理器核方面的故障诊断研究已比较成熟.因此,我们选择注入与外设存储相关的 500 个 SoC 硬件故障,并针对其中的 296 个不可屏蔽 SoC 硬件故障,即不会被体系结构和软件屏蔽的硬件故障,进行 MDCI 实验.所注入的故障包括设备寄存器读取故障和指令读取故障两种故障类型.设备寄存器读取故障是在设备寄存器读取操作处,将读取返回值中随机选择的位值修改.这里,我们选择在系统启动过程中具有关键作用的设备,包括 RTC、IDE、Host/PCI 桥、PCI/ISA 和 ACPI,作为故障注入对象.指令读取故障是在指令读取操作处,将读取指令中随机选择的位值修改.这些故障的注入点地址和注入内容都是由系统运行时动态生成的随机数产生,因此它们具有代表性.参照相关研究[3,5,6,13]的故障注入方法,我们在每次故障运行中只注入一个故障.

3.2 MDCI 所需检验程序点数量和准确度

MDCI 是否只检验少量的动态控制流不变式程序,就能快速、准确地将与 SoC 硬件故障相关的代码范围局部化呢?本节对 MDCI 中所需检验程序点数量和准确度进行评测.MDCI 所需检验程序点数量是指完成一次硬件故障局部化过程所需检验的所有程序点数量.MDCI 的故障局部化准确度,是指所报告的可疑故障范围的与实际故障位置的距离,即可疑范围起始位置与实际故障位置之间的动态执行指令数.

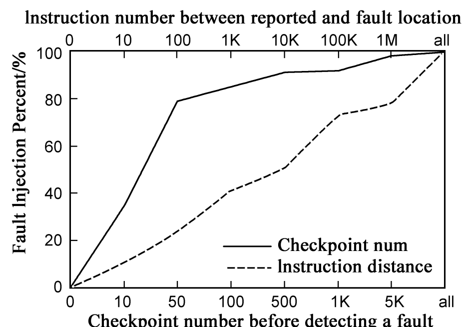

图4 MDCI所需检验程序点数量和故障局部化准确度的累积分布图

基于通过 300 次参考运行提取获得的动态控制流不变式,我们采用 MDCI 方法,对 296 个不可屏蔽的硬件故障进行故障局部化实验,MDCI 所需检验程序点数量和准确度的评测结果如图 4 所示.79%的故障只需检验 50 个程序点就能实现故障局部化,92%的故障需 1000 个程序检查点.典型商业操作系统 Windows XP 包含超过 4 千万行的程序代码.相对于庞大的商业软件,MDCI 只需检验几百个程序点,就能将大部分硬件故障局部化在准确度为几千条指令的可疑范围内.这大大减少了故障搜索范围,加快了故障诊断过程.

同时,所报告的 41%可疑故障位置与实际故障发生位置的距离在 1000 条动态执行指令以内,78%则在 1

※ PKUnity-86 系统结合 x86 处理器和 AMBA 总线架构,并且支持 Windows XP

百万条动态执行指令以内.其中，准确度较差的故障检验运行主要来源于隐式数据破坏故障.这种故障会隐式地破坏系统状态，并在很长一段后才破坏执行路径控制流，因此它会导致可疑故障位置与实际故障发生位置的距离变大.对于 1000 条指令以内的可疑范围，我们可以人工代码分析逐条排查与硬件故障相关的操作.对于百万条指令的可疑范围，可以采用检查点/重放机制和动态切片技术进一步诊断故障根源.

MDCI 所需检验程序点数量和故障局部化准确度依赖于动态控制流不变式.如果所检验的动态控制流不变式中粒度大的数量越多，两个动态控制流不变式程序点间的范围跨度越大，那么 MDCI 所需检验程序点数量越少.当故障位置越靠前，则 MDCI 所需检验程序点数量越少.如果动态控制流不变式的完备性和置信度越高，两个条件跳转动态控制流不变式程序点间的范围跨度越小，那么 MDCI 故障局部化准确度越高.

3.3　故障实例分析

采用 MDCI，我们成功地将 PKUnity-86 FPGA 原型系统中的真实硬件故障局部化.在 PKUnity-86 系统的开发初期，故障导致在 Windows XP 启动时文件系统加载中止，无法正确运行 Windows XP.

由于我们使用微软公司发布的 Windows XP 标准版本，所以可以判断，该问题的根源为 PKUnity-86 系统中的硬件故障.为了实现该故障的范围局部化，我们基于 MDCI 方法在 FPGA 原型系统上从程序开始处依次检验已知关键点、代码段切换、函数调用和条件跳转动态控制流不变式程序点是否可达，得出执行路径在条件跳转 0x8：0x804E50F2 及其应有后继 0x8：0x804DA829 间出错了，即故障可疑范围.最后，通过动态向后切片技术，我们找到影响异常控制流的原因，即中断状态中缺少中断请求 IRQ8.经过进一步对中断请求信号的观测，找到故障根源是 RTC/CMOS 状态寄存器 D 中 CMOS 内容有效位的表示与规格说明相反，CMOS 内容有效位抑制了 IRQ8 的触发，从而导致 Windows XP 无法通过中断跳出预设的时钟闹钟.

在该故障实例的局部化过程中，MDCI 方法只需检验 93 个动态控制流不变式程序点，包括 13 个已知关键点，1 个代码段切换程序点，45 个函数调用程序点和 34 个条件跳转程序点.MDCI 的局部化准确度为 539 条动态执行指令.该实例说明 MDCI 只需检验少量程序点，将与故障相关的可疑代码缩小到较小范围.

3.4　动态控制流不变式的完备性与参考运行次数的关系

为了分析动态控制流不变式的完备性与参考运行次数的关系，本节选用典型操作系统 Windows XP 和 Linux(Redhat 9)为例进行实验分析.动态控制流不变式的完备性体现为动态控制流不变式的置信度.如果动态控制流不变式的置信度达到 100%，那么我们认为它们是最完备的.这里我们只提取代码段切换和函数调用动态控制流不变式.由于已知关键点是通过程序行为分析所得，所以已知关键点在正确的系统运行中必定被执行，具有 100% 的置信度，我们不对其进行分析.而商业操作系统运行过程中的条件跳转动态控制流的程序点数量庞大，我们难以预先提取条件跳转动态控制流不变式，而是在不变式检验阶段，在指定范围中提取它.这能减少条件跳转动态控制流不变式的提取开销.

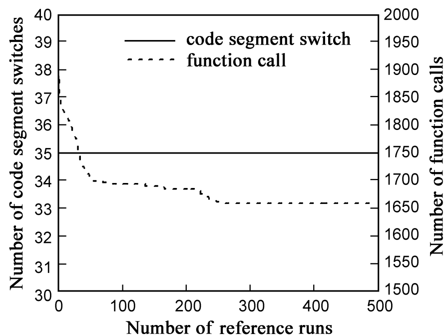

图5　Windows XP代码切换和函数调用动态控制流程序点数量随参考运行数的变化

如图 5 所示，我们进行 500 次的 Windows XP 参考运行(运行至桌面系统，但不包含用户态部分的控制流).Windows XP 的代码段切换动态控制流的程序点数量一直不变.当参考运行次数达到 250 次时，Windows XP 的函数调用动态控制流的程序点数量收敛为 1659.此外，经过 100 次的 Redhat 9 参考运行(运行至用户登录界面)，Redhat 9 的代码段切换和函数调用动态控制流的程序点数量的变化也是收敛的，代码段切换动态控制流的程序点数量收敛为 435，函数调用动态控制流的程序点数量收敛为 1072.这些实验数据说明，当系统环境保持一致的情况下，经过一定次数的参考运行后，我们可以获得置信度较高和完备性较高的动态控制流不变式.当参考运行次数达到 250 次时，Windows XP 的代码段切换和函数调用动态控制流不变式具有较高的完备性.

3.5　开销

动态控制流不变式的提取需要多次参考运行，因此它所需时间较长.为了减少系统模拟运行的时间，MDCI 将离线数据处理过程和踪迹数据收集过程分开进行.动态控制流不变式的提取时间依赖于为达到动态控制流不变式高置信度所需的参考运行次数.单次 Windows XP 参考运行的运行时间约为 3 分钟，产生的所有原始代码段切换控制流踪迹大小是 11 兆字节，函数

调用数据大小约为 650 兆字节.由于动态控制流不变式踪迹可以提前提取,所以相对于复杂的调试过程,其较大的开销是可以接受的.

MDCI 时间开销的关键是在故障运行中进行故障局部化所需的时间,即单次系统运行的性能开销.一次检验过程所需的时间开销包括各粒度不变式检验的故障运行时间和可疑范围内条件跳转动态控制流不变式的提取时间.实验表明,由于采用多粒度逐级迭代局部化方法和检查点/重放技术,一次检验过程所需时间约是一次参考运行时间的 1.5 倍.相对于复杂的调试过程,故障局部化过程所需的时间较少.

4 结论

本文针对导致大型商业软件运行失效的硬件故障局部化效率低的问题,提出了一种基于多粒度动态控制流不变式的硬件故障局部化方法 MDCI.实验表明,对于 79% 的不可屏蔽故障,MDCI 只需检验 50 个程序点就能完成故障局部化,并且 MDCI 所报告的 41% 可疑故障位置与实际故障发生位置的距离在 1000 条指令以内.因此,MDCI 有助于加快导致大型商业软件运行失效的硬件故障诊断过程.后续工作中,我们将基于 MDCI 方法,针对任一款支持特定大型商业软件的 SoC 系统,建立一个硬件故障和软件动态控制流不变式数据库,并将这些数据应用于后续芯片的故障局部化.

参考文献:

[1] Rashinkar P, Paterson P, Singh L. System-on-a-Chip Verification Methodology and Techniques[M]. Boston: Kluwer Academic Publisher, 2001.5 – 6.

[2] Chang Y S, Lee S, Park I C, et al. Verification of a microprocessor using real world applications[A]. Proc of the 36th Design Automation Conference (DAC)[C]. Altanta: ACM Press, 1999.181 – 184.

[3] Li M, Ramachandran P, Sahoo S K, et al. Understanding the propagation of hard errors to software and implications for resilient system design[A]. Proc. of the 13th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)[C]. Seattle: ACM Press, 2008.265 – 276.

[4] 郝丹.一种基于测试信息的交互式错误定位方法[D].北京:北京大学,2007.
Hao Dan. A Testing-Based Interactive Approach to Locating Faults[D]. Beijing: Peking University, 2007. (in Chinese)

[5] Racunas P, Constantinides K, Manne S, et al. Perturbation-based fault screening[A]. Proc. of the 13th IEEE International Symposium on High Performance Computer Architecture (HPCA)[C]. Scottsdale: IEEE Computer Society, 2007.169 – 180.

[6] Li M, Ramachandran P, Sahoo S K, et al. Trace-based microarchitecture-level diagnosis of permanent hardware faults[A]. Proc of the International Conference on Dependable Systems and Networks (DSN)[C]. Alaska: IEEE Computer Society, 2008.22 – 31.

[7] Ernst M. Dynamically Discovering Likely Program Invariants[D]. Seattle: Washington University, 2000.

[8] Zhang X, Gupta N, Gupta R. A study of effectiveness of dynamic slicing in locating real faults[J]. Empirical Software Engineering Journal, 2007, 12(2): 143 – 160.

[9] Mihocka D, Shwartsman S. Virtualization without direct execution or jitting: Designing a portable virtual machine infrastructure[A]. Proc of the 1st Workshop on Architectural and Microarchitectural Support for Binary Translation[C]. Beijing: ACM Press, 2008.

[10] Srinivasan J, Adve S V, Bose P, et al. The impact of technology scaling on lifetime reliability[A]. Proc of the International Conference on Dependable Systems and Networks (DSN)[C]. Florence: IEEE Computer Society, 2004.177.

[11] Peng J J, Hong B R, Yuan C J, et al. Study on software fault injection based on onboard system[J]. Acta Electronica Sinica. 2005, 14(3): 434 – 437.

[12] Goswami K K, Iyer R K. Simulation of software behavior under hardware faults[A]. Proc of the 23rd International Symposium on Fault-Tolerant Computing (FTCS)[C]. Toulouse: IEEE Computer Society, 1993.218 – 227.

[13] Li M L, Ramachandran P, Karpuzcu U R, et al. Accurate microarchitecture-level fault modeling for studying hardware faults[A]. Proc of the 15th International Symposium on High-Performance Computer Architecture (HPCA)[C]. Raleigh: IEEE Computer Society, 2009.105 – 116.

作者简介:

郑衍松 男,1980 年 4 月出生于广东省汕头市.现为北京大学信息科学技术学院博士后.目前主要的研究方向为软硬件协同设计和系统软件.E-mail: zhengyansong@mprc.pku.edu.cn

佟 冬 男,1971 年出生于吉林省长春市.现为北京大学计算机系副教授,主要研究领域为高性能微处理器结构及设计、软硬件协同设计.
E-mail: tongdong@mprc.pku.edu.cn