Computer Science and Application 计算机科学与应用, 2015, 5, 133-140
Published Online April 2015 in Hans. http://www.hanspub.org/journal/csa
http://dx.doi.org/10.12677/csa.2015.54017

Hans汉斯

DSP Soft Error Detection Based on Program Control Flow Integrity Checking

Guochang Zhou[1], Xiangtao Wang[2], Xiang Gao[1], Xiaoling Lai[1], Yangming Guo[2], Dengyun Yu[3]

[1]Academy of Space Technology (Xi'an), Xi'an Shaanxi
[2]School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an Shaanxi
[3]The Science and Technology Committee, China Aerospace Science and Technology Corp., Beijing
Email: zhouguochang2000@163.com, yangming_g@nwpu.edu.cn

Received: May 1st, 2015; accepted: May 17th, 2015; published: May 22nd, 2015

Abstract

In the aerospace, the single particle soft errors are more and more frequently reported in DSP and the other memory devices, which seriously impact the system running safely and reliably. Aiming at the soft error occurring in DSP program storage areas, a soft error detection method based on program control flow integrity checking is presented. Firstly, the DSP program in assembly language is divided into some basic blocks whose structure information is stored in a partition table. And then setting up check points at the end of each basic block, the program control flow error will be fund by examining the application runtime information and blocking the consistency of the information recorded in the partition table. Compared with signature-based method, the proposed method can reach almost 100% of error detection coverage and has better cross platform portability under almost the same detection efficiency and detection overhead.

Keywords

DSP, Soft Error, Control-Flow Error, Integrality Checking

基于程序控制流完整性检查的DSP软错误检测

周国昌[1]，王向涛[2]，高　翔[1]，赖晓玲[1]，郭阳明[2]，于登云[3]

[1]中国空间技术研究院西安分院，陕西　西安
[2]西北工业大学计算机学院，陕西　西安

[3]中国航天科技集团公司科技委，北京
Email: zhouguochang2000@163.com, yangming_g@nwpu.edu.cn

收稿日期：2015年5月1日；录用日期：2015年5月17日；发布日期：2015年5月22日

摘　要

空间环境中DSP等器件频繁发生的单粒子软错误，严重影响了系统的安全可靠运行。针对DSP程序存储区的软错误，本文提出了基于程序控制流完整性检查的软错误检测方法。首先对DSP程序在汇编语言上将程序划分为若干个基本块并将每个基本块的结构信息存储在一个分块表中；然后在每个基本块的末尾设置检测点，通过检查程序运行时信息与分块表中记录的信息的一致性来判断程序是否发生控制流错误。该方法和基于签名的检测方法相比，在故障检测效率和系统开销大致相当的情况下，具有近乎100%的控制流错误检测覆盖率以及良好的跨平台移植性。

关键词

DSP，软错误，控制流错误，完整性检查

1. 引言

当前，高性能 DSP 被广泛应用于空间飞行器电子系统。然而，太空辐射环境中无时不刻存在的单粒子效应使得 DSP 容易发生瞬态错误或间歇性错误，这类被称为软错误的故障或失效严重影响了系统运行的可靠性和安全性[1]，已成为研究和工程开发人员关注的重点。公开资料表明，DSP 中超过 70%的软错误会引发软件发生控制流错误[1]。控制流错误将导致程序偏离正确的执行流程，导致输出结果错误，甚至完全崩溃[2]。因此，基于 DSP 程序控制流检查可以有效检测 DSP 程序存储区是否发生了软错误。

已有控制流错误检测的主要思路是签名监测[3] [4]，即将程序划分为若干个基本块，并为每个基本块分派一个静态签名，当程序运行时计算出基本块的运行时签名，并通过比较静态签名与动态签名是否一致。签名监测有硬件和控制流件两种实现方式。硬件实现方式是利用一个硬件看门狗，实时地检测程序运行过程中出现的错误。硬件看门狗的实现方式的最大特点是速度快，系统性能开销小。但是，该方案不可避免地引入了新的硬件逻辑，带来新的面积开销，增加了系统开发的困难[5]。

随着人们对低成本、高性能、短开发周期的嵌入式系统的需求不断的增长，使得 COTS 越来越广泛地使用在嵌入式系统开发中。COTS 的出现大大加快了系统的开发进度。但是，由于 COTS 的体系结构在出厂后就不能改变，人们不得不在硬件看门狗等手段之外寻求新的控制流错误检测技术。针对硬件看门狗方案的局限性，广大技术人员提出了多种基于软件实现的控制流错误检测方案，如 Enhanced Control-Flow Checking using Assertions [6]、Control-Flow Checking by Software Signatures [7]、Enhanced Control-Flow checking [8]、Edge Control-Flow Checking 和 Region Based Control-Flow Checking [9]等。基于软件实现的控制流检测方案在不依赖底层硬件的基础上，以一定的性能牺牲为代价，大大增强了基于 COTS 的系统的可靠性。虽然基于软件实现的签名监测方案独立于硬件实现，但是仍具有很强的平台依赖性。在不同的平台上，如 ARM、MIPS、X86 等，由于使用的指令集不同，使得在一个平台上的签名生成和监测技术不能直接应用到另一个平台上，这大大破坏了软件系统的可移植性。

针对上述不足，本文提出了基于完整性检查的控制流错误检测方法，即通过检查每个基本块的执行

完整性来判断程序是否发生控制流错误。本方案的最大特点是可移植性，即只需少量的修改就可从一个平台是移植到另一个平台上实现。在检测开销方面，基于完整性检查的控制流检测方案与基于签名的控制流检测方案所带来的开销在一个数量级上。同时，基于完整性检查的检测方案具有近乎 100%的控制流错误检测覆盖率。

2. 基于程序控制流完整性检查的 DSP 软错误检测原理

2.1. 检测原理分析

运行中的 DSP 状态检测，首先应分析单粒子效应形成的软错误故障模式及其特点，将 DSP 构建的软件代码进行模块的合理划分，进而在模块之间确定合理的检测点。

DSP 程序在其指令顺序执行的一段基本模块中，如果没有发生控制流错误，程序将从模块入口开始执行，并在模块的出口跳出。因此，分支指令出错是导致控制流错误的主要原因。在划分基本块后，所有的分支指令都位于基本块的末尾，当出现分支指令错误时，控制流可能转移到不正确的地址处开始执行，也可能导致一个本该发生的控制流转移没有发生或相反。

此外，DSP 的软错误可能引起存储单元单元中的信息发生跳变，如果这种跳变发生在程序存储区，则有可能改变指令的编码。公开的文献[10]和实际分析表明，发生在程序存储区的跳变可能导致一条非分支指令转变为分支指令；也可能导致一条分支指令转变为非分支指令。如果一条非分支指令转变为分支指令，则程序必然出现控制流错误。在划分基本块后，所有的非分支指令都在基本块内部(包括基本块的第一条指令)。此时，非分支指令转换为分支指令导致的控制流错误主要表现为，基本块内部到程序其他地方的控制流错误转移。

综上讨论，控制流错误主要表现为 1) 在该发生控制流转移的地方没有转移或转移地址出错；2) 在不该发生控制流转移的地方发生了控制流转移。如图 1 所示。

图 1 中，矩形表示基本块，实线箭头表示正确的控制流转移，虚线箭头表示错误的控制流转移。A 和 B 表示由于出现非分支指令转换为分支指令导致的控制流错误，其中 A 为控制流发生基本块内错误转移，B 为控制流发生基本块间错误转移。C 和 D 表示一条分支指令的两个分支项，C_1 表示本来应该发生

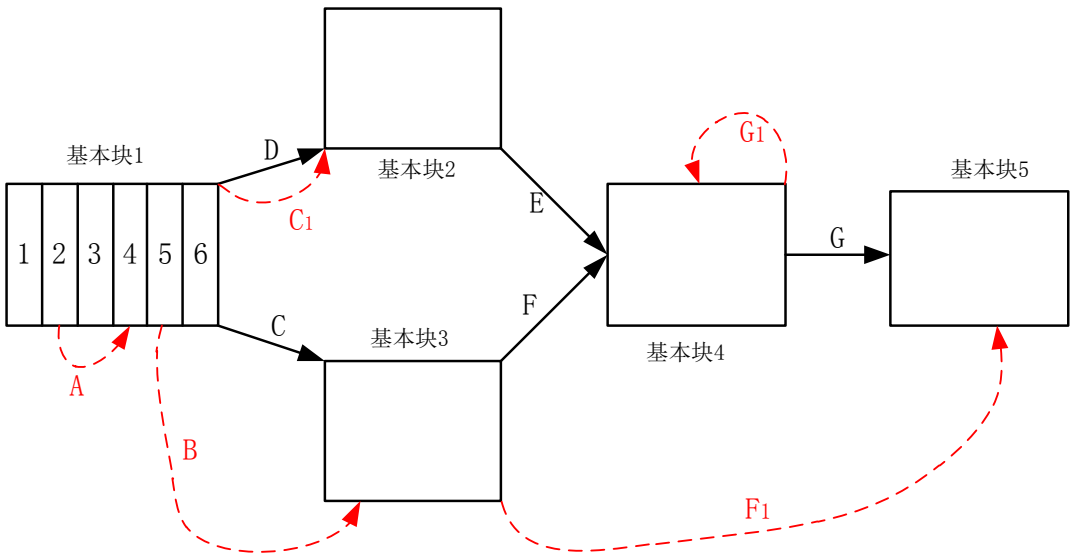

Figure 1. Control flow error diagram
图 1. 控制流错误示意图

控制流转移 C，但由于分支指令错误，发生了一个看似正确的控制流转移 C_1。F_1 表示由于分支指令错误，发生了一个到其它基本块的控制流错误转移。G_1 表示由于分支指令错误，发生了一个到基本块自身的控制流错误转移。

2.2. 延迟槽的处理

延迟槽是指控制流分支指令后的一条或几条指令，不管分支是否发生，延迟槽中指令总是被执行且结果先于分支指令提交。

延迟槽主要存在于早期在流水线中没有分支预测功能的平台上，用于提高流水线的效率。现代处理器已经广泛具备了分支预测功能，延迟槽的存在已经失去了意义，但为了实现软件系统的兼容性，MIPS、PowerPC 和 DSP 平台上仍然保留着延迟槽。有学者曾提出一种基于签名控制流检测方法，通过将签名存储在分支延迟槽中，从而减小系统的存储开销[1]。单纯从基本块的定义上，延迟槽中的指令应作为一个独立的基本块。但是，在功能层面上，延迟槽中的指令大多与前一个基本块中的指令具有强烈的相关性，所以应该将延迟槽中的指令并入延迟槽之前的一个基本块。此外，有些延迟槽完全由空操作指令构成，其中并没有包含有效的指令，将这种延迟槽划分为基本块是没有任何意义的，反而会增加检测开销。综上所述，在划分基本块的过程中，将延迟槽并入之前的一个基本块中。

2.3. 基本块的划分

划分基本块是实现控制流错误检测的基础。基本块是指一段顺序执行的代码。控制流只能从代码的第一句进入基本块，从最后一句退出基本块。即在正常情况下，基本块中除最后一句外不可能出现控制流分支的情况；基本块中除第一句外不可能出现控制流进入的情况。

划分基本块可在两个层次上进行：C 语言级和汇编语言级[11] [12]。由于 C 语言是一门高级语言，具有平台无关性。因此，在 C 语言级划分基本块并施加控制流检测，代码能运行在不同的平台上。但高级语言要经过编译、链接成可执行代码后才能在机器上运行。在高级语言向低级语言转换的过程中，C 语言一级的基本块并不能很好的映射到低级语言上。

目前，大多数基本块划分技术都是在汇编语言一级实现的。汇编指令用助记符对机器指令进行翻译，汇编指令与机器指令之间具有很好的对应关系。这使得汇编语言一级划出的基本块能很好地映射到由机器指令构成的可执行代码上。这样，在程序语言一级划分基本块的目的和诉求(添加签名等)能够在程序执行过程中很好地体现出来。此外，基于完整性检查的控制流错误检测方案要求获得每个基本块的结构信息，即基本块包含的指令数量和基本块最后一条语句的地址。因此，本文选择在汇编语言一级划分基本块。

在汇编语言级划分基本块，就是在 DSP 汇编程序代码中寻找程序控制指令，如函数调用指令、函数返回指令和跳转指令的位置，以此确定基本块的边界。这里，没有将系统函数调用和中断指令作为控制流指令来处理。因此，在划分基本块的过程中，将系统函数调用和中断指令视作普通指令[12]。虽然不同平台采用的指令集不同，但不同的指令集内部都有用于实现函数调用、函数返回和跳转的指令。

2.4. 分块表的设计

实现基于完整性检查的控制流检测的基础，是获得每个基本块的结构信息，其结构信息主要包括：基本块入口(基本块第一条指令在程序中的地址)、基本块出口(基本块最后一条指令在程序中的地址)、块长度(基本块中包含的指令条数)、下一跳地址(是指跳转指令的目的地址或函数调用时，子函数的第一条指令所在的地址)。此外，为了设置检测点，还需进行指令备份，将基本块的最后一条指令储存三份。为此，需要设计相应的数据结构。

由于 DSP 程序在划分基本块之前，无法确定具体能划分成多少个基本块。因此，设计的数据结构必须具有动态增长的特点。本文将分块表设计成一个链表的形式其结点结构如图 2 所示。

基于以上分析，基本块的划分规则如下：

1) 以跳转指令、函数调用指令和函数返回指令所在位置作为当前基本模块的出口；

2) 当前跳转(调用、调用返回)的后一指令(若存在延迟槽，则将延迟槽之后的第一条指令)所在位置下一个作为基本模块的入口；

3) 当前跳转(调用、调用返回)的目的指令所在位置作为另一个基本模块的入口。

事实上，划分基本块的过程就是扫描汇编程序中，不断寻找程序控制指令的位置来确定基本块的边界的过程。此外，出于检测的需要，本文为分块表设置一个工作指针。工作指针始终指向下一个即将进行完整性检查的基本块的信息所在的结点。

3. 基于完整性检查的检测方法

3.1. 基于完整性检查的检测机制

完整性检查是指在每个基本块的最后检查当前基本块有没有被完整的执行，如果当前基本块被完整执行，表明在之前的程序执行中没有发生控制流错误。基于完整性检查的控制流错误检测方法通过检查每个基本块是否被完整地执行来检测程序是否发生控制错误。

为了实现对每个基本块完整性的检查，需在每个基本块的出口处设置检测点。如图 3 所示为例，当程序在基本块 1 中执行时，如果发生一个控制流错误转移 B，则程序在执行完第 5 条指令后将转移到第 10 条指令处执行。由于发生上述控制流错误转移，程序将错过设置在第 6 条指令处的检测点，但分块表中的工作指针此时仍然指向基本块 1 的结构信息所在的结点。当程序执行到第 11 条指令时，碰到设置在基本块 2 处的检测点，开始进行基本块完整性检查。检测模块将当前检测点的地址“11”与分块表工作指针指向的结点中的基本块出口信息“6”比较，发现不一致。由此，检测模块判断程序发生了基本块间的控制流错误转移。通过比较检测点地址与工作指针指向的结点中的基本块出口信息是否一致，可以检查出绝大部分基本块间的控制流错误转移。

比较地址一致的方式无法检测出基本块内发生的控制流错误转移。在图 3 中，A 表示一个基本块内的控制流错误转移，程序在执行完第 2 条指令后，转移到第 4 条指令处继续执行。此时，设置在第 6 条

基本块入口	基本块出口	块长度	下一跳地址	指令备份1	指令备份2	指令备份3	链表下一结点地址

Figure 2. Nodes structure diagram of partitioned table
图 2. 分块表结点结构示意图

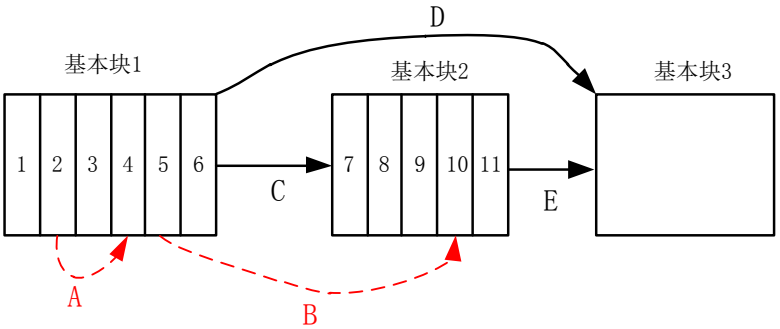

Figure 3. Sample of basic block integrity check
图 3. 基本块完整性检查示例

指令处的检测点无法检测出这种错误。为了检测出这种控制流错误转移，设置一个计数器 counter，当程序控制流进入一个基本块时，将 counter 值初始化为 0。此后，程序每执行一条指令，counter 的值加 1。进入检测模块后，先比较检测点地址与工作指针指向的结点中的基本块出口信息是否一致，再通过比较 counter 的值是否与工作指针指向的结点中的块长度信息一致。如果两个信息都一致，则可以判定当前基本块被完整执行，程序在当前基本块中没有发生控制流错误；若地址比较信息不一致，则判定程序在当前基本块中发生了块间的控制流错误转移；若 counter 值比较不一致，则判定程序发生了块内控制流错误转移。

基于完整性检查的方法能检测图 1 中 C_1 之外的所有控制流错误转移。对于 C_1 这一类控制流错误转移，本方案中采用关键指令三模冗余技术来避免其造成的影响[2]。关键指令三模冗余是值将跳转指令冗余执行三次，并根据三次执行结果表决出占优势的执行结果，将该结果作为指令的正确结果。关键指令三模冗余技术能极大地提高分支指令执行的正确率。

3.2. 检测点的设置

基于软件实现的控制流错误检测方案必须通过修改源代码来实现检测点的插入。在基于签名的控制流错误检测方法中，是直接将用于比较签名和更新签名的代码直接置于两个基本块之间，这种检测点设置方式必然会加大系统的存储开销。

实现完整性检测的关键步骤之一是将检测点的地址与分块表工作指针指向的结点中的基本块出口信息相比较。由于基本块划分操作是在插入检测点之前实现的，如果通过直接向源代码中插入检测代码的方式来插入检测点，插入的检测代码必然会引起源程序中的指令的地址发生变化。这种地址的变化将使得地址比较的检测方式失效。

基于以上讨论，本方案选择将检测代码设计成一个独立的模块——检测模块，并将检测模块置于程序代码的末尾。同时，将基本块的最后一条指令替换为一条跳向检测模块的跳转指令，将基本块的最后一条指令的三个备份存储在分块表中。图 4 为检测点设置和检测模块结构的示意图。图 4 中，黑色实线箭头表示没有施加检测措施时的控制流转移；黑色虚线箭头表示施加完整性检测措施后的控制流转移。在基本块 1 中，在原来的第 6 条指令处设置检测点，将原来的指令替换为一条跳向检测模块的跳转指令。在检测模块的最后，执行在当前检测点处被置换的指令，实现控制流的“真正”转移。在其他基本块中，检测点的设置方式与控制流转移过程与基本块 1 中的情形完全相同。

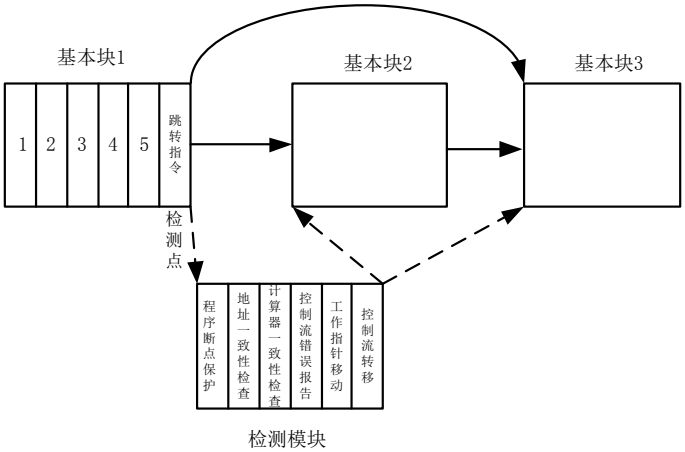

Figure 4. Structure diagram of testing point setting and testing module
图 4. 检测点设置和检测模块结构示意图

3.3. 检测模块的设计

为了实现基本块完整性检查，必须“中断”程序正常的执行，将控制流转移到检测模块内。为了使控制流能正确的回到原程序中，进入检测模块之后的第一件事就是将原程序的断点信息保存起来。然后在检测模块内部完成地址一致性检查和计算器一致性检查，并根据两次检查的结果判断当前基本块是否被完整执行，即程序在当前基本块内是否发生控制流错误。若当前基本块发生了控制流错误，则停止程序的执行，并向系统报告；若没有发生控制流错误，则开始初始化下一次完整性检测——即进行分块表工作指针的移动操作和 counter 值清零。当检测模块执行到这个步骤时，表明上一个基本块的完整性检查已经完成，此时，必须将分块表工作指针指向下一个即将进行完整性检查的基本块的信息所在的结点。但是，在检测模块内原程序的控制流并没有真正的发生转移，即此时并不知道接下来将执行哪个基本块。为了解决这个问题，在此引入了一个“预转移”过程。

“预转移”是指先恢复一次之前保存的程序断点信息，然后执行保存在分块表中的三条指令并对三条指令的执行结果做表决——即关键指令三模冗余技术，根据表决结果判断程序接下来将执行哪个基本块，并将分块表工作指针重新指向下一个将被执行的基本块的信息所在的结点。

完成上述步骤后的控制流仍然在检测模块中。因此，在检测模块的最后，还需将控制流转移回原程序。此时，只需再一次恢复之前保存的程序断点信息，然后对在当前检测点处被替换的指令执行三模冗余，控制流即回到原程序中。

4. 性能分析

1) 检测覆盖率

本文方案通过对由分支指令错误和非分支指令错误引起的控制流错误进行建模，将控制流错误归纳为基本块间错误跳转和基本块内错误跳转两种类别。这种建模方式完整地覆盖了所有控制流错误类型。根据 3.1 节的分析，基于完整性检查的检测方法有着近乎 100%的检测覆盖率，即通过检查出口地址一致性，能够检测出基本块间的错误跳转；通过检查基本块内的指令执行条数，能够检测出基本块内的错误跳转；通过关键指令三模冗余技术，能够检测出由于分支指令条件出错导致的错误跳转。而基于签名的控制流错误检测方法只能检测基本块间的控制流错误和分支指令条件出错的导致的控制流错误,公开文献表明，基于签名的控制流错误检测方法的检测覆盖率约为 74%~96% [2]。综上所述，基于完整性检查的检测方法的故障检测覆盖率高于绝大多数基于签名的检测方法。

2) 故障检测效率

故障检测效率是指从控制流错误转移发生到检测到这次错误转移之间所执行的指令条数。故障检测效率仅与基本块的长度和检测点的设置位置有关。在基于完整性检查的检测方案中，每个基本块后都设置了检测点。因此，对于所有控制流错误转移，都能在一个基本块的长度内即可检测出来。但是，由于高级语言的语义表达效率更高，使得在高级语言一级划分出的基本块的平均长度比在汇编语言级划分出的基本块的平均长度要小。所以，本文提出的方法的故障检测效率与其他汇编语言级的控制流错误检测方法大致相当，但比高级语言级的控制流检测方法低。

3) 性能开销

任何检测机制的引入都会导致原程序性能的下降，下降的程度用性能开销来表示。性能开销是指引入检测机制后所带来的额外的程序执行时间与原程序的执行时间的比值。但是，一种检测机制所具有的性能开销是很难定量评价的，它受所加固的程序本身的性质(如：矩阵乘法、FIR、FFT)，检测机制的实现层次(汇编级、高级语言级)，目标代码的执行平台(X86、MIPS、DSP、ARM 等)，检测机制的优化等级以及编译器的设置等因素的影响。在具体的实施过程中，通常是调整检测机制的实现层次、优化等级

以及编译器设置等选项，使检测机制的性能开销能控制在一个可接受的范围内。

在评价基于完整性检查的软错误检测方法的性能开销时，采用定性比较的方法，即通过分析完整性检测方法与签名监测方法在检测机制上的差异来判断完整性检测方法可能具有的性能开销。在基于完整性检查的检测方法中，引入了保护程序断点、实施检测、初始化下次检测(移动工作指针，counter 值清零)和返回原程序四个操作。在基于签名的检测方法中，也需要保护程序断点、签名检查、更新签名和返回原程序几个操作。因此，基于完整性检查的检测方法带来的性能开销与基于签名的方法的性能开销大致相当。

5. 总结

本文讨论了基于程序控制流完整性检查的 DSP 软错误检测方法。首先对 DSP 程序在汇编语言上将程序划分为若干个基本块并将每个基本块的结构信息存储在一个分块表中。然后在每个基本块的末尾设置检测点，通过检查程序运行时信息与分块表中记录的信息的一致性来判断程序是否发生控制流错误。该方法和基于签名的检测方法相比，在故障检测效率和系统开销大致相当的情况下，具有近乎 100%的控制流错误检测覆盖率和良好的跨平台移植性。

基金项目

国家自然科学基金(61371024)、航空科学基金(2013ZD53051)、航天支撑技术基金、中航产学研项目(cxy2013XGD14)。

参考文献 (References)

[1] Jafari-Nodoushan, M., Miremadi, S.G. and Ejlali, A. (2008) Control-flow checking using branch instructions. *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Shanghai, 17-20 December 2008, 66-72.

[2] 刑克飞 (2007) 星载信号处理平台单粒子效应检测与加固技术研究. 博士论文, 国防科技大学, 长沙.

[3] Sugihara, M. (2011) A dynamic continuous signature monitoring technique for reliable microprocessors. *IEICE Trans. on Electronics*, **94**, 477-486.

[4] Tan, L.F., Tan, Y. and Xun, J.J. (2013) CFEDE: Control-flow error detection and recovery using encoded signatures monitoring. 2013 *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems* (*DFTS*), New York, 2-4 October 2013, 25-32.

[5] 明月伟, 宁洪, 邓胜兰 (2014) 面向星载操作系统的控制流错误检测方法. *计算机应用*, **34**, 1418-1422.

[6] Alkhalifa, Z., Nair, V.S.S., Krishnamurthy, N., et al. (1999) Design and evaluation of system-level checks for on-line control-flow error detection. *IEEE Transactions on Parallel and Distributed Systems*, **10**, 627-641.

[7] Oh, N., Shirvani, P.P. and McCluskey, E.J. (2002) Control-flow checking by software signatures. *IEEE Transactions on Reliability*, **51**, 111-122.

[8] Reis, G.A., Chang, J., Vachharajani, N., et al. (2005) Software implemented fault tolerance. *The Proceedings of the Third International Symposium on Code Generation and Optimization* (*CGO*), San Jose, 20-23 March 2005, 243-254.

[9] Borin, E., Wang, C., Wu, Y.F., et al. (2006) Software-based transparent and comprehensive control-flow error detection. *The Proceedings of the International Symposium on Code Generation and Optimization* (*CGO*), New York, 26-29 March 2006, 333-345.

[10] Benso, A., Di Carlo, S., Di Natale, G. and Prinetto, P. (2002) Static analysis of SEU effects on software applications. 2013 *IEEE International Test Conference* (*ITC*), Baltimore, 7-10 October 2002, 500-508.

[11] Goloubeva, O., Rebaudengo, M., Reorda, M.S. and Violante, M. (2003) Soft-error detection using control flow assertions. *The Proceedings of the* 18*th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Boston, 3-5 November 2003, 581-588.

[12] 黄振远 (2006) 一种星载计算机控制流件检错技术的研究与实现. 硕士论文, 哈尔滨工业大学工学, 哈尔滨.