基于虚拟寄存器的控制流错误检测算法

高 星, 廖明宏, 吴翔虎, 黄振远
(哈尔滨工业大学 计算机科学与技术学院 哈尔滨 150001)

摘 要: 控制流故障是航天软件系统必须面对的一个重要故障类型。提出一个基于程序基本块模型的算法 CFCVR(Control Flow Checking Based on Virtual Register, 基于虚拟寄存器的控制流检测) 对程序控制流进行检测。它首先通过虚拟寄存器分配算法获得虚拟寄存器, 然后基于这些虚拟寄存器添加特定的控制流检测指令。这些指令可以检测模块间的控制流错误, 所有工作都是在汇编源程序上完成的。实验表明 CFCVR 会引入平均 28.7% 的性能代价和平均 31.3% 的存储代价, 而对于控制流错误检测率平均为 97.1%, 优于目前已存在的各种方法, 能够提高航天软件容错能力和可靠性。

关键词: 可靠性; 容错; 控制流检测; 虚拟寄存器
中图分类号: TP302.8 文献标识码: A 文章编号: 1000-1328(2007)01-0183-05

0 引言

航天软件系统工作环境非常苛刻, 空间环境会对软硬件系统的可靠性产生极大影响, 例如外层空间的高速中子流或者 α 粒子会引起飞行器软硬件瞬时故障。这些故障可能会导致灾难, 因此有高可信需求的航天软件在设计时要考虑尽量早地发现和处理这类故障。环境因素引起的程序故障可以按照故障类型进行分类: 对于一条指令, 由于瞬时故障使操作数或者操作码发生了错误, 这类错误都称为数据错误; 而程序运行时读取并执行下一条指令时发生的错误称为控制流错误。根据重粒子[1] 和模拟实验[2] 的结果, 控制流错误引起的控制流故障占由瞬时故障的大约一半。数据错误可以通过外接 EDAC (Error Detection and Correction, 错误检测和纠正) 模块等相对简单的策略来处理; 而对于发生在 CPU 内部的控制流错误, 其处理受到 CPU 架构限制, 而现有的 COTS(Commercial Off-the-Shelf, 商业货架) 处理器通常都不具备处理控制流错误的能力, 因此控制流检测需要额外的技术手段。

最初控制流检测是通过在一些编译器中使用断言(Assertion) 检查[3] 的方法, 但这些方法往往跟具体应用相关, 检错覆盖率也不能让人满意。还有一些检测技术利用一个看门狗处理器[4] 来实现系统级的同步错误检测, 但是现代带内部缓存的流水线处理器的内部指令的执行并不能被外部看门狗处理器观测[5], 而且大部分的 COTS 处理器也没有内建的看门狗。

为了使控制流检测技术突破具体硬件的限制, 学术界提出了一些纯软件的控制流检测理念, 其中有代表性的是: 文献[6] 提出了一种被称为 ECCA (Enhanced Control Flow Checking Using Assertion, 基于断言的增强控制流检测) 的技术, 它通过给程序的每个基本块指定一个独一无二的标识符, 然后程序在运行时检查这些标识符, 但这会带来 150% 的存储开销; CFCSS(Control Flow Checking by Software Signature, 基于数字签名的控制流检测)[7] 是另一种基于块签名的技术, 在作者的实验中检错覆盖率达到了 97%, 存储开销是 30- 60%, 但它需要额外的寄存器和高级语言写成的程序源码, 而且存在一些检错覆盖盲点; 文献[8] 提出的 ECFC(Enhanced Control Flow Checking, 增强控制流检测) 提高了 CFCSS 的检错覆盖率, 却进一步增加了性能负担; 文献[9] 提出了一种新的基于断言的程序控制流检测方法 ACFC(Assertions for Control Flow Checking, 基于断言的控制流检测), 使得存储开销在 30- 47% 的情况下检错覆盖率达到 95%, 但是该方法同样是基于源码的。还有一些方法, 如二次调用[10] 等, 但这些方法限制太多, 使用范围受限。

本文主要研究检测控制流错误的软件办法, 提

收稿日期: 2006-01-11; 修回日期: 2006-05-09
基金项目: 国家高技术研究发展计划(2005AA742013)

出了一种基于程序基本块的签名算法和插入检测指令的虚拟寄存器分配方法, 主要特点: 1) 不需要高级语言源码, 直接对汇编或伪汇编程序操作; 2) 不需要专门寄存器, 降低了程序运行开销; 3) 纯软件方法, 不需要任何额外的硬件支持。实验表明本文所述方法优于现已存在的其它方法, 能够提高航天软件容错能力和可靠性。

本文组织如下: 第一章介绍基于基本块的程序流图; 第二和第三章介绍具体的签名和寄存器分配原理; 第四章是详细的算法以及对算法的分析; 第五章给出故障注入方法和实验结果; 第六章给出结论和对未来工作的展望。

1 基于程序基本块的控制流图

本文使用一种源于文献[7, 8] 中的基于程序基本块的程序流图来描述程序的结构。如果一系列指令是从第一条开始到最后一条结束顺序执行的, 满足这一条件的指令的最大集合称为一个基本块。在基本块中除了最后一条指令不可能有其它分支指令, 一个基本块的最后一条指令要么是指向另外一个基本块的分支指令, 要么是接受从其它一个或者多个地方传入控制流的指令。如果一个基本块有超过两个其它基本块连入, 那么称该基本块为一个扇入节点。

定义集合 $V = \{v_1, v_2, v_3, \cdots, v_n\}$ 表示程序中的所有基本块, 另一个集合 $E = \{ b_{ij} \mid b_{ij}$ 是从 v_i 到 v_j 的分支$\}$ 表示程序中所有基本块之间的跳转关系, 那么程序就可以表示为程序流图 $P = \{V, E\}$。若 $b_{ij} \in E$, v_j 是 v_i 的后继, 记作 $v_j \in \mathrm{suc}(v_i)$; 若 $b_{ij} \in E$, v_i 为 v_j 的前驱, 记作 $v_i \in \mathrm{prev}(v_j)$。假如 v_i 到 v_j 的分支是非法的, 那么 $b_{ij} \notin E$。非法分支就表明了一个控制流错误。

2 基于虚拟寄存器的动态签名

文献[7] 提出了一种运行时签名算法, 这个算法首先给每个基本块指定一个唯一的静态签名, 然后使用在编译过程中保留的两个专用寄存器中保留动态签名的信息, 程序运行时通过比较静态和动态签名来判断程序控制流是否正确。这个算法是基于 XOR 运算的, 因为对于计算机而言 XOR 运算性能优于其它的运算, 本文提出的签名算法也是基于 XOR 运算的, 不同之处在于对扇入节点的特殊处理方式, 以及通过特殊设计的虚拟寄存器分配方法突破了在文献[7, 8] 中必须的两个保存动态签名的寄存器的限制, 同时消除了文献[7] 中的检错覆盖盲点。

对于程序流图中的每一个节点 v_i, 分配唯一的静态签名 s_i, 设程序运行时的动态签名为 G, G 在节点 v_i 的动态签名值为 G_i, 用来保存 G 值的是一个被称为 VRSR(Virtual Rurr time Signature Register, 虚拟运行时签名寄存器) 的临时分配的存储空间(可以是寄存器或内存, 实现方法在第三部分介绍)。

设 $G_{\mathrm{prev}(i)}$ 是 v_i 的一个前驱节点的动态签名, v_i 节点用来辅助验证动态签名的静态值为 d_i, d_i 用来表示 $G_{\mathrm{prev}(i)}$ 和 G_i 签名差异。在程序发生从 $\mathrm{prev}(v_i)$ 到 v_i 跳转时候, 按照一个功能函数 $f(G_{\mathrm{prev}(i)}, d_i)$ 计算 v_i 节点的动态签名 G_i, 比较 s_i 和 G_i 的值是否相等, 如果 s_i 和 G_i 的值相等, 则程序继续执行, 反之则认为程序发生了控制流的错误, 程序进入相应的错误处理。其中 $f(G_{\mathrm{prev}(i)}, d_i)$ 是基于 XOR 运算的, 定义为 $f(G_{\mathrm{prev}(i)}, d_i) = G_{\mathrm{prev}(i)} \oplus d_i$, 可以推导 d_i 由公式 $d_i = f^{-1}(s_{\mathrm{prev}(i)}, s_i) = s_{\mathrm{prev}(i)} \oplus s_i$ 决定的。图 1 描述了利用该方法检测控制流错误的过程。

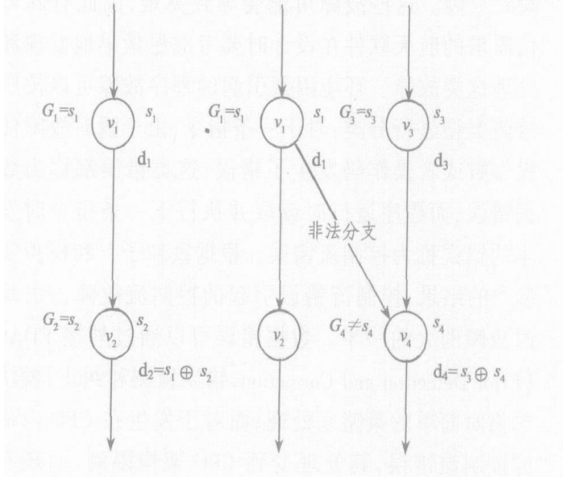


图 1 控制流错误检测示意图

Fig. 1 Sketch map of control flow checking

假设有 M 个合法分支 $b_{(1, \cdots, M)j}$ 进入扇入节点 v_j, 对于 v_j 不同的前驱节点组成的集合 $V_{\mathrm{prev}(j)}$ 中的每一个节点分别计算不同的 d_j 值组成集合 D_j, 这些值并不相等。对此, 本文尝试在进入节点 v_j 后对每一个可能的 d_j 值分别跟 G_j 进行匹配, 遇到匹配的则停止匹配并跳转到 v_j 内的原有代码进行执行, 反之如果在 D_j 中没有找到能匹配的 d_j 值, 则认为发生了

一个控制流错误。

3　虚拟寄存器分配方法

现有的控制流检测方法要么是对源代码的修改,要么是对源代码编译的过程进行修改,它们都需要额外的寄存器支持[3-8]。这个特点决定了现有技术都需要高级语言所写的程序源代码作为输入,这限制了控制流检测技术的使用范围。本文设计两种基于临时分配的寄存器或内存的 VRSR 分配方法,使得汇编语言源程序或者高级语言源程序编译生成的伪汇编程序也能被处理。

方法 1: 设程序流图中的一个节点 v_i 执行,那么在程序在 v_i 最后一条指令发生跳转时,程序将进入 v_i 某个后继节点的第一条指令,设程序能使用的所有通用寄存器的集合为 U,基本块 v_i 使用的寄存器的集合 R_i,其后继节点的集合为 $V_{suc(i)}$,假设其元素为 R_{vsi1} 到 R_{vsin},这些节点中的寄存器按照第一次访问是读或者写操作又可以分为 R_{vsi1W}, …, R_{vsinW} 和 R_{tsi1R}, …, R_{vsinR}。设其先写寄存器集合 $R_{vsiW} = R_{tsi1W} \cap R_{vsi2W} \cap$, …, R_{vsinW},若集合 $(U - R_i) \cap R_{tsiW} \neq$ NULL,则 VRSR 可以从 $(U - R_i) \cap R_{vsiW}$ 中任意一个元素为值。

方法 2: 统计表明汇编源程序并不是平均使用所有通用寄存器,在程序局部通常只使用相对较少的部分寄存器,例如在对 GNU C 编译器生成的 ARM 汇编源码进行统计中发现,平均每个基本块只使用约 3 个寄存器,平均相邻的 3 个基本块只使用约 5 个寄存器,在所有统计的 16632 行汇编源程序中 R_{10} 没有被使用, R_9 也只被使用了 3 次。可以利用这些极少使用的寄存器来作为 VRSR,在其前后插入相应的指令,用内存来暂时保存这些寄存器的值。

```
Load Ri ,[Addr]
Ope VRSR
Store [Addr] , Ri
/* 以下 Ri 作为VRSR* /
 ……
/* 以上 Ri 作为VRSR* /
Load Ri , [Addr]
Ope VRSR
Store [Addr] , Ri
```

方法 1 本质是查找块间可以临时使用的寄存器,不会与内存有任何关联,提高程序性能,缺点是 $(U - R_i) \cap R_{vsiW} \neq$ NULL 的情况平均只有 36%;方法 2 需使用内存临时转储寄存器中的内容,但其覆盖率约 100%,而且在实际应用中转储操作极少发生,实验表明访存引起的平均性能代价小于 1%。第四章给出的算法是基于方法 2) 的。

4　基于 VRSR 的 CFCVR 算法

4.1　算法描述

输入: 汇编语言源程序(汇编代码或者由高级语言源代码编译而成的伪汇编代码) P_s。

输出: 插入 VRSR 分配指令和 CFCVR 控制流检测指令之后的汇编程序 P_d。

(1) 输入 P_s,识别其中基本块并建立程序流图 P;

(2) 检查 P 中所有通用寄存器的使用情况,找出其中使用频率最低的寄存器 R_f;

(3) 保留地址为 Adds 的特定内存位置,在所有使用 R_f 的指令前插入“load R_f [Adds]”,其后插入“store[Adds] R_f”;

(4) 规定 P 中的任意节点 v_i 有唯一的静态签名 s_i,即 $s_i \neq s_j$,若 $i,j = 1,2, \ldots, N$, N 是 P 中节点个数;

(5) 规定 P 中的起始节点 v_1 的动态签名 $G_1 = s_1$,遍历 P 中所有的其它节点 $v_j, j = 2,3, \ldots, N$;

① 对节点 v_j, $v_{prev(j)} = \{v_{j1}, v_{j2}, \ldots, v_{jM}\}$,计算集合 $D_j = \{d_{jk} \mid d_{jk} = s_{jk} \oplus s_j, k = 1,2, \ldots, M, j \in 2,3, \ldots, N\}$, M 是其前驱个数;

② 计算集合 $D'_j = \{d'_{jk} \mid d'_{jk} = d'_{j1} \oplus d'_{j2}, \ldots, \oplus d_{jk}, k = 1,2, \ldots, N\}$;

③ 对于 D'_j 中除最后一个元素以外的其它每个元素依次在块 v_j 中插入两条指令“$G = G \oplus d'_{jk}$”和“$br\ G = s_j\ Lj$”;

④ 对于 D'_j 中的最后一个元素,插入指令“$G = G \oplus d'_{jk}$”和“br $G \neq s_j$ error”;

(6) 整理输出插入指令之后的程序 P' 即为所求。

图 2 以一个扇入节点为例说明插入指令的方式。

4.2　存储和性能代价分析

假设节点 v_i 具有 N 个前驱节点,为每个前驱节点需要插入的验证指令是两条,所以节点 v_i 需要我们需要插入控制流检测指令为 $2N$ 条,假设 v_i 有 N 个前驱的概率为 p_N,那么对于一段程序插入指令条数 N 的数学期望 $E(N) = p_1 * 2 + p_2 * 4 + \ldots, p_N * 2N$;根据对基准程序和实际项目源码的统计

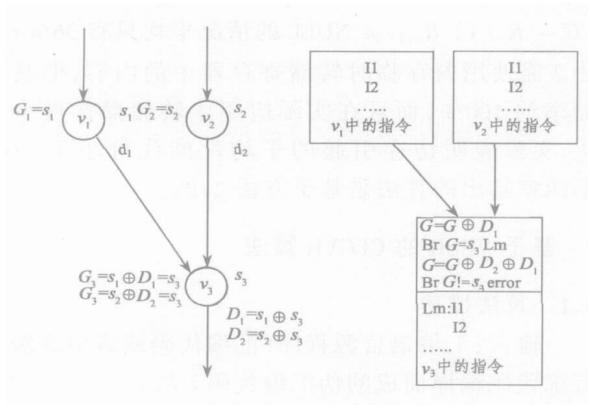


图 2 块内检测指令示意图

Fig. 2 Sketch map of checking instruction in block

结果 $E(N)=2.48$; 统计还表明每个基本块的平均指令条数是 8– 10 条, 那么采用 CFCVR 增加的存储代价大约为 24. 8– 31% 之间优于 CFCSS 的 25– 43% 和 ACFC 的 30– 47%; 至于性能代价, 因为插入指令都是内存操作指令, 其执行时间远小于访存指令, 因此性能代价应小于存储代价; 实测表明性能代价平均为 18– 35%。

5 实验及分析

实验分为两部分: 1) 采用了 LZW 压缩算法, 矩阵乘法, 快速排序, 汉诺塔等四个基准程序来测试 CFCVR 的可行性及性能; 2) 选择哈工大某重大航天课题星务管理程序(简称 A 课题) 来测试算法在实际项目中的表现。所有实验都是基于 GNU C 编译器编译生成的. s 文件进行。实验方案选用的故障注入工具是基于 GNU 调试器 gdb 的, 它通过直接修改相应跳转指令的数据段来达到注入不同控制流故障的目的, 其具体细节参见文献[11, 12]。

在应用了控制流检测技术之后, 程序执行结果可分为四种情况: 1) 程序未检测出控制流错误, 结果正确; 2) 程序未检测出控制流错误, 结果错误; 3) 程序检测出控制流错误, 结果正确; 4) 程序检测出控制流错误, 结果错误。在这四种情况中, 只有 2) 是对导致错误的扩散, 而对于 1) 3) 4) 无论对错, 都认为结果是可预测的, 不会影响程序的下一步运行。表 1 给出了 CFCVR 与其它两种典型技术对情况 2) 的实验结果比较。

可以看出 CFCVR 在检测效果上优于 CFCSS 和 ACFC, 在存储及性能代价上 CFCVR 也有了提升, 表 2 给出了程序源码的统计数据以及存储和性能代价的统计数据。

表 1 CFCVR 与其它方法检测结果比较

Table 1 Result comparison of CFCVR & others

未测出的错误输出	LZW	矩阵乘法	快速排序	汉诺塔	A 课题
(无措施)	26. 2%	33. 8%	33. 6%	21. 8%	30. 6%
CFCVR	1. 7%	2. 8%	2. 7%	1. 6%	2. 9%
CFCSS	2. 0%	3. 0%	3. 4%	1. 8%	3. 1%
ACFC	7. 6%	12. 0%	9. 3%	7. 2%	9. 1%

表 2 程序源码统计数据及存储和性能代价统计数据

Table 2 Statistics of source code & overhead

	指令	检查指令	基本块	存储代价	性能代价
LZW	452	252	105	55. 7%	44. 3%
矩阵乘法	763	232	102	30. 4%	27. 2%
快速排序	254	111	47	43. 7%	37. 8%
汉诺塔	179	60	27	33. 5%	30. 1%
A 课题	16632	5207	1851	31. 3%	28. 7%

在统计的 1851 个基本块中, 总前驱数目为 2280 个; 1423 个节点只有一个前驱节点; 有两个前驱节点的节点数目为 427; 有三个前驱节点的节点数目是 1 个; 没有发现超过 3 个前驱节点的节点。由此可见具有单个前驱的节点占节点总数的 77. 88%; 有两个前驱节点的节点数目为 23. 07%; 超过两个前驱节点的节点对算法性能的影响极小。除了在效率和性能上有了提高, CFCVR 还具有其它特性, 如表 3 所示。

表 3 CFCVR 和其它方法应用特性比较

Table 3 Application comparison of CFCVR & others

	高级语言	汇编	反汇编	手动修改	性能代价	存储代价
CFCSS	支持	部分	否	否	30– 70%	30– 60%
ACFC	支持	否	否	是	30– 48%	30– 47%
CFCVR	支持	是	是	否	27– 55%	27– 42%

6 结论

本文结合哈工大某重大航天课题星务管理系统的设计, 提出一种纯软件的控制流检测方法, 它在引入平均 28. 7% 的性能代价和平均 31. 3% 的存储代价的情况下, 对于控制流错误检测率达到平均 97. 1%, 表现优于目前已有其它方法, 能够提高航天软件容错能力和可靠性。CFCVR 算法突破了目前已经存在的其它控制流检测算法必须基于高级语言源程序的限制, 这为以后对于无源码程序实现控制流检测提供了手段。CFCVR 算法是对基本块间的

控制流错误进行检测, 块内错误却不能处理, 未来可以尝试与块内错误检测技术结合, 提高检错覆盖率; 此外未来还可以对 VRSR 的分配算法进行改进, 提高算法性能。还可以尝试将本算法引入无源码的环境中, 通过对反编译之后的代码进行适当处理, 然后添加控制流检测指令。这些技术对于将 COTS 组件引入航天软件领域都有着相当大的助益。

参考文献:

[1] Gunneflo U, Karlsson J, Torin J. Evaluation of error detection schemes using fault injection by heavy-ion radiation[C]// Proc of Intl Symposium on Fault Tolerant Computing, 1989: 340-347

[2] Ohlsson J, Rimen M, Gunneflol U. A study of the effects of transient fault injection into a 32-bit RISC with built-in watchdog[C]// Proc. of Intl. Symposium on Fault Tolerant Computing, 1992: 316-325

[3] Saib S H. Executable assertions-an aid to reliable software[C]// Proc. of 11th Asilomar Conference on Circuits. Systems and Computers, 1978: 277-281

[4] Kanawati G A, Kanawati N A, Abraham J A. FERRARI: a flexible software-based fault and error injection system. IEEE Trans on Computers, 1995, 44: 248-260

[5] Rela M Z, Madeira H. Experimental evaluation of the fail-silent behavior in programs with consistency checks[C]// Proc of Intl Symposium on Fault Tolerant Computing, 1996: 394-403

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

[7] Oh N, Shirvani P P, McCluskey E J. Control flow checking by software signatures. IEEE Trans on Reliability, 2002, 51: 111-122

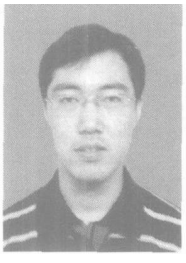
[8] Reis G A, Chang J. SWIFT: software implemented fault tolerance. International Symposium on 20-23, March 2005: 243-254

[9] Venkatasubramanian R, Hayes J P, Murray B T. Low-cost on-line fault detection using control flow assertions[C]// In Proceedings of the 9th IEEE International On-Line Testing Symposium, July 2003: 137-143

[10] Oh N, McCluskey E J. Low energy error detection technique using procedure call duplication[C]. Int Conf on Dependable Systems and Networks (DSN'01), 2001

[11] Reddi V J, Settle A, Connors D A, Cohn R S. PIN: A binary instrumentation tool for computer architecture research and education[C]. In Proceedings of the 2004 Workshop on Computer Architecture Education (WCAE), June 2004: 112--119

[12] M Lutz. Programming Python. O'Reilly. Sebastopol, CA, 1996



作者简介: 高星(1980-)男, 博士研究生, 计算机软件与理论专业, 研究方向为高可信系统, 实时嵌入式计算。

通信地址: 哈尔滨工业大学 316# 信箱 (150001)

电话: 13604887022

E-mail: hs_gaoxing@126.com

A Control Flow Checking Algorithm Based on Virtual Register

GAO Xing, LIAO Ming-hong, WU Xiang-hu, HUANG Zhen-yuan

(School Of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract: Control Flow Fault was an important fault type which should have been seriously treated in the high-confidence software systems such as aero-space system. An algorithm named CFCVR(Control Flow Checking Based on Virtual Register) was given which checks the control flow fault based on the Base-Block prototype. The control flow faults were detected by obtaining the virtual register and adding some control flow checking instructions into the program based on the virtual register. All these works were done on the assemble programs. Experiments show that CFCVR will introduce about 28.7% performance overhead and about 31.3% storage overhead and will increase the fault detection rate to 97.1% that was better than the existent methods.

Key words: Reliability; Fault-tolerance; Control-flow checking; Virtual register