维普资讯 http://www.cqvip.com

并发控制流检测技术综述

吴艳霞　顾国昌　付　岩　程立新
（哈尔滨工程大学计算机科学与技术学院　哈尔滨 150001）

摘　要　并发控制流检测技术是防止由于单粒子反转事件而导致运行系统发生故障的有效手段，目前主要采用以控制流图为基础的结点签名技术。本文首先介绍并发控制流检测技术的分类标准；然后按照技术发展的脉络，从软硬结合、纯软件两方面介绍控制流检测技术的典型方法，对其进行分析评价；最后提出基于目前的方法并发控制流检测技术还需要解决的问题及新的发展方向。

关键词　并发控制流检测，看门狗处理器，控制流图，分派签名，源签名

A Survey on Concurrent Control Flow Checking

WU Yan-Xia　GU Guo-Chang　FU Yan　CHENG Li-Xin
(College of Computer Science and Technology, Harbin Engineering University, Harbin 150001)

Abstract　The Concurrent Control Flow Checking (CCFC) is an effective way for running system to prevent from breaking down caused by Single Event Upsets. It mainly adopts current Node-Signature Technique based on the Control Flow Graph. The classification criteria of CCFC is presented. And the classic method of CFC is analyzed and evaluated as two ways: hardware-software combination and purely software-based, according to the sequence of technique development. At last, it points out the challenge problems according to the current method and new research directions.

Keywords　Concurrent control flow checking, Watchdog processors, Control flow graph, Assigned-signature, Derived-signature

1　引言

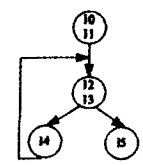
并发控制流检测技术是防止由于单粒子翻转事件而造成程序错误运行的有效手段之一[1]。其实现方法和处理器结构有着紧密的联系，因而处理器结构的不断发展，势必带来并发控制流检测技术的不断改进。

本文首先根据不同的标准对典型的并发控制流检测方法进行分类；接下来从软硬结合、纯软件两个方面根据技术发展的脉络介绍并发控制流检测方法，强调适应处理器结构发展的软件实现的编译辅助的控制流检测技术，重点分析这些方法存在的问题；最后提出新的发展方向。目前，在数字系统中，并发控制流检测技术一般分为3个层次：电路层、系统层及应用层。电路层检测主要采用数学编码的检测方法，根据芯片的输入输出管脚检测错误。系统层检测技术主要包括基于权力的寻址(Capability-based addressing)、看门狗定时器(watchdog timers)、容错性数据结构(fault-tolerant data structures)及复制的方法。应用层检测主要是在系统执行时应用专门的算法来检测，可通过如断言(assertions)、基于程序变量间的不变性关系等方法来实现。

2　分类标准

并发控制流检测技术从Yau首次提出至今发展已经近20年[2]。期间，主要采用基于签名的并发控制流检测技术，它主要通过程序运行时产生的签名同存储的签名信息进行比较，从而判断程序是否正确执行[3]。程序中的控制流可以通过编译时产生的控制流图(control flow graph, CFG)来表示，即 $P=\{V,E\}$，$V=\{v_i \mid v_i$ 为控制流结点，$1\leqslant i\leqslant n, n$ 为控制流结点数$\}$；$E=\{(v_i,v_j)\mid$表示控制流从 v_i 跳转到 $v_j\}$。结点划分为 $\delta:\lambda\rightarrow V$，$\lambda=\{I_k \mid I$ 为程序指令，k 为指令所在行号$\}$，即 $\delta(\lambda')=v_i$，$\lambda'\subset\lambda$，表示此结点除了入口和出口，其它位置不包含分支指令。图1给出了源程序与控制流图的对应关系，在控制流图的结点处可进行签名的比较。

```
I0: number1=2
I1: number2=3
I2: sum=number1+number2
I3: if(sum > 4) goto I5
I4: goto I2
I5: halt
```



(a)源程序图　　(b)对应的控制流图

图1　源程序与控制流图对应关系图

根据签名产生方式和引用信息的存储位置对签名控制流检测技术进行分类。表1列出了典型的基于签名的并发控制流检测技术。

根据签名产生方式可分为源签名(derived signature)和分派签名(assigned signature)。源签名方法的签名信息产生于控制流图的结点信息，它采用看门狗处理器监控总线信息，同时计算运行时的签名。分派签名的签名信息主要是任意分派的，同时由被检测的处理器激活检测操作。如果分派签名

吴艳霞　博士生，CCF会员，研究方向为可靠的嵌入式编译器；顾国昌　教授，博士生导师，主要研究领域：嵌入式系统；付　岩　博士生，研究方向：高级语言编译；程立新　讲师，博士，主要研究领域：EDA、高级语言编译。

维普资讯 http://www.cqvip.com

需要看门狗处理器进行签名比较，它可直接接收相应的运行时签名。

根据参数的存储位置又可分为：存储签名(Stored Signature Database)、引用程序(Reference Program)、嵌入式签名(Embedded Signature)及 Shambhu[4] 提出的无引用签名(No Reference Signature)方法。存储签名方法是将签名信息存储在看门狗处理器的本地内存中，看门狗处理器接收或计算运行时的签名，查找存储参数并且去判断此签名是否是有效的后继。存储签名可以在不损失性能的条件下并行监控签名。程序更新后，虽然需要产生新的程序图和分析现有的程序计算签名，但是不需要将新的签名嵌入到重新编译的程序中。存储签名方法的缺点是：它的监控复杂度高并且需要占用很大的内存空间，还要维持应用程序和存储签名之间的通讯。引用程序方法是将签名信息存储在寄存器中，主要是通过一个专门的看门狗处理器来执行参数程序。引用程序和待检测的程序有相同的 CFG 结构，通过并行执行程序来检测控制流。嵌入式签名是将签名信息存储在被检测的程序中，如果需要看门狗处理器参与校验，就在运行时将信息传给它，否则由被检测的处理器自行进行校验。无引用签名方法主要是基于编码的思想来实现的。

表 1　基于签名的并发控制流检测方法分类表

方法	签名产生方式	签名信息存储位置
CEDIMPS[5]	源签名(D)	存储签名(S)
Cerberus-16[6]	源签名(D)	引用程序(R)
Watchdog Direct Processing[7]	源签名(D)	引用程序(R)
ex-precision checksum[8]	源签名(D)	嵌入引用(E)
CSM[9]	源签名(D)	嵌入引用(E)
ESIC[10]	分派签名(A)	存储签名(S)
SIC[11]	分派签名(A)	引用程序(R)
CPMWNRS[4]	分派签名(A)	无引用(N)
BSSC/ECI[12]	分派签名(A)	嵌入引用(E)
SEIS[13]	分派签名(A)	嵌入引用(E)
CAOCAS[14]	分派签名(A)	嵌入引用(E)
ECCA[15]	分派签名(A)	嵌入引用(E)
CFCSS[16]	分派签名(A)	嵌入引用(E)
CFCVRE[17]	分派签名(A)	嵌入引用(E)
PECOS[18]	分派签名(A)	嵌入引用(E)
ACFC[19]	分派签名(A)	嵌入引用(E)
CAOCASC[20]	分派签名(A)	嵌入引用(E)
SWIFT[21]	分派签名(A)	嵌入引用(E)
Craft[22]	分派签名(A)	嵌入引用(E)

从图 2 可以看出，在几种签名分类中，目前研究人员最常采用嵌入签名机制的分派签名技术进行并发控制流检测。

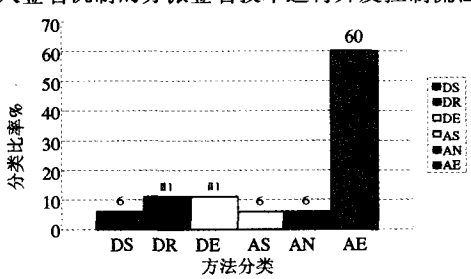


图 2　控制流检测分类使用频度图

图 2 中 DS 表示为存储签名机制的源签名，DR 表示为引用程序机制的源签名，DE 表示为嵌入签名机制的源签名，AS 表示为存储签名机制的分派签名，AR 表示为引用程序机制的分派签名，AE 表示为嵌入签名机制的分派签名。

基于签名的控制流检测技术，其检测能力一般从 3 个方面考虑：①检测结点是否属于非法(illegal)的分支跳转。②检测结点是否属于不正确的(incorrect)分支跳转。由于目前控制流检测技术的实现方法几乎是基于控制流图，但控制流图不能体现数据的依赖关系，所以目前的控制流检测能力几乎无法达到第 2 个方面的要求。③基本块的内部检测，即检测非控制流命令转换为控制流命令而导致在块内跳转现象。常常提到的由于单粒子翻转事件而造成的改变非控制流指令内容改变的现象，如加法指令被窜改为减法指令，在本文中没有将此划为控制流检测范围，对其检测可以采用其它技术，控制流检测仅仅指检测控制流的跳转方向。

基于签名的控制流检测技术的评价指标一般分为 5 个点：①错误检测覆盖率(error-detection coverage)，即检测出来的错误数目和总的可能发生错误数目的比值，它充分地体现了控制流检测技术的检测能力；②执行时间开销(execution time overhead)；③内存开销(memory overhead)；④错误检测潜伏期(error-detection latency)，即错误发生时，检查到错误的平均时间；⑤检测复杂度(monitor complexity)。但目前主要从前两点评价检测技术。

由于应用环境和处理器结构的不同，软硬结合、纯软件控制流检测方法各有利弊。同时，基于签名的控制流检测能力和对系统性能的影响常常是相互制约的。本文将依据控制流技术的评价指标中的前两方面分析目前的并发控制流检测技术，并提出需要改进的地方。

3　典型并发控制流检测技术的分析

并发控制流检测技术是从软硬结合、纯软件两方面充分利用电路层、系统层及应用层的检测方法来实现的。此部分首先分析评价基于硬件看门狗的控制流并发检测方法，其次分析评价纯软件实现的控制流并发检测方法。

3.1　软硬结合的实现方法

20 世纪 90 年代是并发控制流检测技术发展的高峰期，根据当时的主流的处理器结构，研究人员提出很多以硬件实现为主软硬结合的并发控制流检测技术，即，借助看门狗协处理器及相应电路协助完成并发控制流检测。伴随处理器结构和配置的发展，硬件实现为主软硬结合的并发控制流检测技术的签名方法从以源签名为主发展到以分派签名为主。

3.1.1　源签名

由于不同的处理器结构(CISC，带流水线技术的 RISC)和配置(单处理器、多处理器)，从 1982 年 Nam 发表的第一篇基于源签名的方法起，就不断产生其修改和改进方法[4,6,7,23~26]。这些方法的不同之处主要体现在控制流图结点的定义和引用签名信息的方式上。

为了降低检测复杂度，Sridhar[24] 将存储签名作为计算运行时签名的一部分，同时简化了错误信号的产生方式。Upadhyaya[4] 采用了 m-out-n 编码方式检测控制流；为了降低嵌入式签名的内存开销和提高运行性能，采用 GPSA[23] 等方法来减少签名数量；Ohlsson[26] 提出了增加错误覆盖率的方法，但是这些方法均只提高检测技术的某些指标，并没有提高其综合指标。

Cerberus-16[6] 是第一个采用引用程序的源签名并发检测控制流的签名方法，看门狗处理器执行一个和被检测的处

维普资讯 http://www.cqvip.com

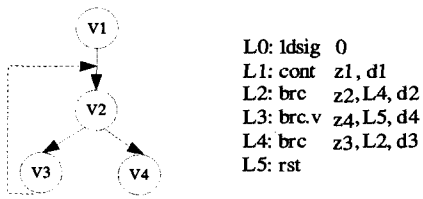
理器(此处理器执行的程序为 P)具有相同控制流图的程序 P',同时 P'始终与 P 保持相同的控制流,即同时执行相同的结点。程序 P'具有两种结构的指令,分别表示如下。

表示控制流结构的指令:OP Z,[L],[D]

用来初始化和主处理器之间通信的指令:OP or OP D

其中:Z 表示结点的序号,L 表示下个结点的地址,D 表示结点签名,L 和 D 是可选择的。

实现过程为:在程序运行时初始化程序压缩单元,当主处理器执行完指令 Z 后,看门狗处理器将从分支检测电路发出一个信号来表示主处理器是否选择一个分支。如果检测到执行一个分支指令,那么看门狗处理器执行 L 去计算下条指令的目的地址,在分支选择之后看门狗处理器和主处理器将执行相同的结点。如果没有执行分支,看门狗处理器则按顺序执行下一条指令。被看门狗处理器执行的每条指令 D 经过数据压缩处理后存入寄存器中。如图 3 如果主处理器执行的结点序列为(v1,v2,v3,v2,v4),那么看门狗处理器执行的指令地址为(L1,L2,L4,L2,L3),其中选择 XOR 操作作为压缩处理函数,这条路径的累计签名 h= d1⊕d2⊕d4⊕d2⊕d3,并将 h 存入寄存器中,在指令 brc. v 处将累加的签名和通过分支检测电路检测出来的签名进行比较,完成控制流的检测。



(a)主处理器执行的控制流图;(b)Cerberus-16 执行的相应程序图。

图 3 Cerberus-16 控制流检测图

在 Delord[7] 中对 Cerberus-16 进行优化,错误检测覆盖率和错误检测潜伏期的性能指标都取得明显的改善。但是由于处理器结构的发展,采用引用程序的这类方法的并行信息维护变的越来越复杂。

在传统的处理器结构中,基于源签名的方法通过检测指令总线即可完成。随着处理器结构的发展,芯片内开始采用缓存技术和预存取管道技术,这样虽然看门狗处理器继续监控总线上的指令序列,但是这并不是处理器要执行的准确的序列。在文[25]中提出了解决基于管道技术的监控问题,该文主要是在看门狗处理器中模仿管道技术中涉及的基本的函数,在取指令时计算引用签名。但是,该文仍然没有解决看门狗处理器在处理器外部监控内嵌缓存技术的处理器指令序列的技术。基于这些情况,分派签名越来越引起人们的关注。

3.1.2 分派签名

软硬结合的分派签名方法主要有 SIC[11] 及 ESIC[10]。SIC 是在计算机程序中重新组织高层的控制流结构,通过签名或标签标注这些结构,在运行时采用看门狗处理器检测这些结构的完整性。SIC 是基于形式语言和自动机的,而 ESIC 是在 SIC 的基础上扩展了检测能力。

采用分派签名并发检测控制流时,其预处理器主要完成 4 个步骤:①定义结点,构成 CFG 图;②对 CFG 图进行编码;③修改源程序,以便插入用来传送签名的语句;④从看门狗处理器中提取签名信息。不同的分派签名方法主要区分在第 2 步和第 4 步。但是,采用分派签名也存在一些问题,主要包括以下 3 个方面:①由于主处理器需要产生签名和将签名传送给看门狗处理器,这将引起系统性能退化。因为产生签名的速度远远大于传送它的速度,这将引起时间不同步的问题;②由于仅仅检查高层结点的执行顺序,忽略对其内容的检查,会导致更小的错误覆盖率;③容易造成更长的错误潜伏周期。

随着 VLSI 的不断发展,采用硬件冗余技术的并发控制流检测方法受到了很大的制约,加上并行系统的广泛应用,基于单处理器的传统的看门狗检测技术不能简单的应用,所以基于看门狗控制流检测技术的发展遇到很大的阻碍。

SEIS[13] 是通过软硬结合的方法实现多处理器多任务的控制流检测方法。它采用分派签名的方法,将签名信息嵌入到程序中。同时,此方法还可以检测执行一个程序的多处理器之间的交互,有效地利用看门狗的资源,减少硬件冗余。

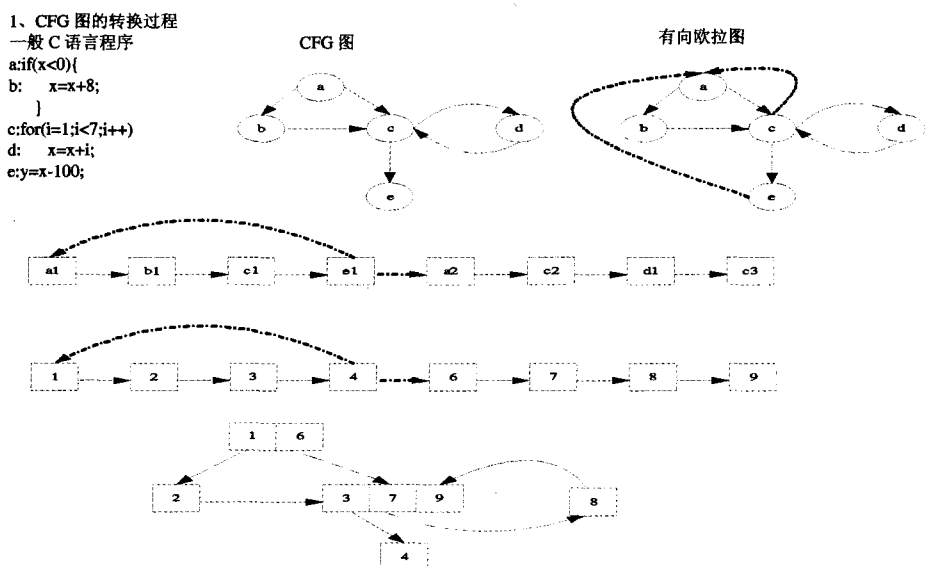


图 4 CFG 生成结点签名过程图

SEIS 方法是从 CFG 中提取有向边,通过增加额外边,保证每个结点的出度=入度,使之简化成欧拉(Eulerian)图。由 CFG 生成结点签名子标识的过程如图 4,该方法是按着子标识的顺序进行签名比较检测控制流跳转,图 4 为结点签名格式的定义。从图 4 中可以看出每个结点的子标签个数不同,在结构化 C 语言中,通过某些方法转换子标签个数,可使之

维普资讯 http://www.cqvip.com

确定在<=3的范围内。为了处理方便，在SEIS中将标签个数设置成固定数目，小于3的结点通过复制方法使其标签的个数等于3。但是，由于结点数目的增加，标识数的范围不可确定，此时就需要通过某些机制处理超过范围的数。可是，该文没有明确的提出子标识消减优化方法，同时没有详细的分析子标识经过优化后是否影响控制流检测能力。

Processor ID	Task ID	Procedure ID	Signature type	Sublabels

图5 结点签名格式图

通过实验表明，当基本块中包含3～5条指令时，程序的长度将超过原有的30%，执行时间将超过原有的100%，错误检测覆盖率为50%，内存开销超过原有的30%。如图5的格式定义方法可以支持多处理器多任务的处理器，同时降低了硬件开销，但此种方法检测速度慢，会造成很高的错误潜伏期。

另外，Yung-Yuan[27]提出的将看门狗处理器嵌入到RISC的32位5级流水线的处理器中的方法，采用水平和垂直两种混合的控制流检测方法，实现并行控制流检测。针对垂直签名该文提供一种可以容易改变签名长度的方法，此方法具有很好的优化效果。这种综合的控制流检测技术虽然在系统性能可接受的前提下提高了错误检测覆盖率及降低了错误潜伏周期，但增加了硬件实现的复杂度。

3.2 纯软件实现方法

随着处理器结构和集成电路的不断发展，采用硬件为主软硬结合的并行检测控制流的方法受到很大的挑战，纯软件实现的并发控制流检测技术越来越引起人们的注意。采用纯软件的并发控制流检测技术主要面临时间上的开销和对系统性能影响两方面的问题。纯软件实现的并发控制流检测技术的发展主要经历以下几个阶段：①以不同的设计概念并行检测控制流[28]；②从辨别系统运行状态这个角度出发，辨别系统内存空间的使用[29]，地址的使用[12]；③改变源程序去并发检测控制流，主要是采用编码方法[4,13,17]；④采用应用层断言(assertions)冗余检测技术，但这种技术对程序员来说不是完全透明的[15,18,30~32]；⑤采用不同的数据复制指令的方法来检测控制流。此方法是采用一定规则自动生成运行结果相同的代码，通过运行两套代码来检测控制流。这样对程序员来说是透明的，但是增加了代码的冗余，影响运行性能，最主要的是新增加的代码也有可能导致控制流错误，影响程序的正确执行[22,33,34]。

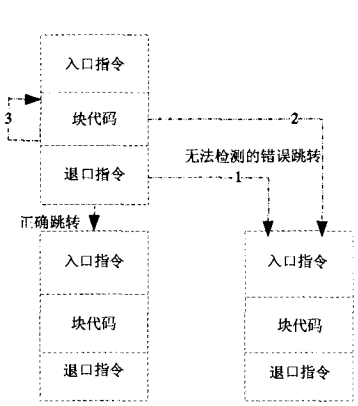


图6 BSSC检测能力图

3.2.1 BSSC/ECI

1992年的BSSC/ECI[12]是通过纯软件实现并行控制流检测的代表性方法。在BSSC中程序被分成块，并且每个块被分配一个签名，通过运行时的签名检测程序是否正确的进入和离开此块。在BSSC中采用基本块的第一条指令的地址作为签名，这种方法实现起来很简单，但是由于嵌入式签名是由绝对地址组成，所以代码变得和位置相关。同时此种方法的控制流检测能力存在局限性，无法检测出图6出现的情况，从而导致错误覆盖率只达到50%左右。

3.2.2 ECCA/PECOS

为了消除代码的依赖，检测出上述无法检测的控制流错误，研究人员开始采用ECCA[15]和PECOS[18]应用层断言冗余检测技术来并行检测控制流。

ECCA是将高层语言分割成块，每个块由两个断言保护，同时给每个块分配一个块标识符BID，设为大于2的整数(用数学符号表示)。第一条断言称作SET，以(1)式的形式插入到基本块首部：

$$id \leftarrow \frac{BID}{(id \bmod BID)\cdot(id \bmod 2)} \tag{1}$$

id为全局整数，在基本块进入和离开时赋值，在SET处如果出现除0，则表示控制流错误。

第二条断言称作TEST，以(2)式的形式插入到基本块尾部：

$$id \leftarrow NEXT + \overline{(id - BID)} \tag{2}$$

NEXT是个整数值，在预处理阶段由基本块的BID生成。如$NEXT = \prod IBD_{Permissible}$表示从目前的基本块可以访问的下一个基本块号。可以从公式看出在离开时会更新id值，如果进入下个基本块错误时，则会出现除0错误。此方法可以检测上述中第1、2种无法检测的情况。当基本块包含3至5条指令时，程序执行时间的开销为原有开销的150%，错误覆盖率达到92%以上。

PECOS严格说是检测内存错误，而不是检测控制流错误。但是在错误的分支地址被使用时内存错误也会变成控制流错误。但是，此方法存在使用的前提条件，为自从上次写入后，在分支执行时使用的数据没有改变。PECOS的计算公式为：

$$\frac{X_{out}}{!\,[(X_{out}-X_1)(X_{out}-X_2)\cdots]} \tag{3}$$

其中$\{X_1, X_2, \cdots\}$是有效的目的地址，X_{out}是运行时目的地址。

如果X_{out}是无效的地址将会抛出除0错误。此方法也只能检测上述中第1、2种无法检测的情况。当基本块包含3至5条指令时，程序执行时间的开销为原有开销的130%，错误覆盖率达到95%以上。

这两种方法均无法检测“不正确”控制流和块内控制流改变的情况，同时在使用断言技术检测控制流时，对程序员来说并不是透明的，因此，这种技术的发展受到了很大的制约。

3.2.3 ACFC

Rajesh[19]提出的ACFC方法是在预处理阶段将断言自动地插入程序，这样对程序员来说是透明的。该方法是将CFG图转化为有向无环图DAG。此方法是将控制流错误转化为跳转、重新执行及多路径错误，在每个基本块处执行一个奇偶校验，用此来检测控制流错误。图7为ACFC签名用例。

当控制流检测的覆盖率达到95%时，采用此方法的程序执行时间的开销仅为原有开销的105%。此方法不仅可以检测出检测图5中的3类控制流错误，而且也可以检测控制流

维普资讯 http://www.cqvip.com

的反向错误跳转。但是，该 Rajesh 没有阐述清楚 XOR 运算的执行状态字 ES 的分配方法，而这种分配方法的好坏直接影响控制流检错的覆盖率，同时他也没有明确的给出处理 goto 这类语句的方法。

```
1 number1=2;
2 if (sum>4)
3   New_Number = sum-3;
4 else
5   New_Number = sum-1;
```

```
1 ES_1 = ES_1 ^ 01;
2 number1=2;
3 if (sum > 4){
4   ES_1 = ES_1 ^010;
5   New_Number = sum -3;
6 }else{
7   ES_1 = ES_1 ^010 ;
8   New_Number = sum -1;
9 }
10 ES_1 = ES_1 ^0100;
11 if (ES_1 != 0111) error();
```

（a）源程序图　　（b）加入签名后的程序图

图 7　ACFC 签名用例图

3.2.4　CFCSS

Nahmsuk 提出的 CFCSS[16] 的分派签名并发控制流检测方法，是将分派签名作为计算运行时签名的一部份，这样可以检测出反向错误跳转，增加了控制流检测的能力。结点检测方法如图 8。

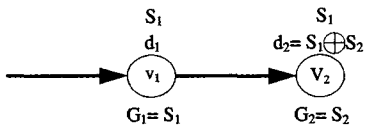


图 8　CFFS 结点签名原理图

G_n：在 V_n 点的运行时签名；S_n：在 V_n 点的分派签名，是独一无二的数；d_n：签名差。

签名函数定义为：$f(G,d)=G\oplus d_i$，运行时签名 G_2，$G_2=f(G_1,d_2)=G_1\oplus d_2$，同时 $d_2=S_1\oplus S_2$，$G_1=S_1$，所以新的运行时签名 $G_2=f(G_1,d_2)=G_1\oplus d_2=S_1(S_1\oplus S_2)=S_2$ 即，如果分支跳转正确时运行签名等于分派签名。

当基本块平均包含 7～8 条指令时，程序执行时间的开销为原有开销的 140%，代码错误覆盖达到 97%。由于在一个块中，CFCSS 需要插入三条断言，而 ACFC 只需要一条断言，所以 CFCSS 的路径检测方法相对 ACFC 复杂，但错误覆盖率也相对高。但此方法还需要考虑以下几个问题：①在签名长度受限情况下，如何分派签名方法。②此方法没有解决多扇入共享结点问题，算法中涉及的调整签名值的 D 的确定方法还需要优化。③算法中没有清晰地给出多分支结点中确定主结点的方法，在选择不恰当的主结点后会对控制流的检测能力产生很大影响。在文[22]中提出一种编译辅助的结合硬件实现的分派签名控制流检测方法来解决第 3 个问题，通过分析此种现象产生的具体环境，采用额外增加指令的方法解决此问题。但是，由于此方法需要增加 3 条指令和相应的硬件开销，所以降低了通用性。但是目前采用纯软件方法解决此问题的方法还没有提出。

最近提出的 SWIFT[21]，不是一个单纯的控制流检测方法，但是在该篇文章中提出的采用程序运行时应用指令层未使用的指令资源来检测由于单粒子反转造成的程序运行错误是一种很好的设计思路。为了减少性能的影响，ARC[35] 技术利用了处理器未使用的资源，但是原始资源的使用变得很低。AR-SMT[36] 和 SRT[37] 动态区分处理器资源应用不同的线程运行同一个程序的两个拷贝，比较每条冗余指令对，在一般情况下性能超负荷达到 5%～30%。这些方法均是应用处理器结构的特点进行控制流检测。

4　进一步研究的问题和发展方向

现有的基于签名的控制流检测技术力图达到综合指标的最优，但是它们之间始终存在相互制约的关系。随着处理器结构的发展，充分应用处理器结构的特点对控制流进行检测成为一个新的发展方向。如何将原有的一些并发控制流检测思想应用到新的处理器结构中也是值得我们探讨的问题。

为了提高控制流检测技术的综合指标，可从以下两个方面改进控制流的检测技术：

4.1　签名策略的改进

4.1.1　基于控制流图的签名

基于签名的分派控制流检测技术是在签名的入口[16]或出口[18]进行签名比较，这样就无法检测基本块内的由于单粒子翻转事件而导致的控制流跳转错误。通过实验表明当基本块的大小为 3～5 条指令时具有很高的控制流检测能力，当基本块的长度再大些时其检测能力就会下降，所以很多文献一直希望能通过修改结点定义来提高控制流检测能力。本文提出可采用某些方法使签名比较的位置对基本块划分策略的依赖性变小。如可考虑结合编译技术，当基本块的长度超出某个范围，在块内增加校验功能，进行简单的签名比较。这样有助于提高检测能力和检测技术指标。

当签名比较方法相同，但签名分配方法不同，也会影响控制流检测能力。大部分文章忽略了关于这部分的讨论，由于签名值本身也存在被打翻的情况，采用优化的签名分配方法和合理的对其长度控制的方法会提高控制流检测能力，如在相邻结点分配签名时尽量扩大其码距，即使签名被打翻也不会影响控制流的错误覆盖率。

SEIS 方法充分利用图的理论，将控制流图转化为有向欧拉图，再进行编码、签名。虽然此算法中关于签名分配部分需要优化，但是，由此提出的将程序语言的结构和图的理论相结合的控制流检测思路，可以简化签名校验方法，提高控制流检测技术指标。

以控制流图为基础的检测技术想要达到能够检测非法控制流序列的水平，就要通过某些方法关联控制流和数据流。由于控制流图不能体现数据之间的依赖关系，所以单纯的基于控制流图的控制流检测技术只能检测其跳转分支是否属于跳转序列，而不能检测出是否跳转到序列中正确的指令位置上。将数据流和控制流相结合的方法可提高控制流的检测能力。

4.1.2　非基于控制流图的签名

在文[7,13]中提到关于引用程序的控制流检测方法，它们大多需要硬件看门狗处理器。引用此控制流检测方法，采用纯软件的实现方法，将原程序采用某种算法映射成新程序，并行执行结构相同的程序，从而进行控制流检测。由于增加大量的冗余代码本身也存在错误的可能性，所以就要求有优秀的映射机制，减少代码冗余。

4.2　处理器结构的充分应用

随着处理器结构的发展，并发控制流检测技术也在不断的改进。根据处理器结构的特点协调硬件和软件各自完成的检测功能，充分利用处理器结构优化程序，利用空闲资源进行控制流检测。

结论　基于签名的并发控制流检测技术是解决单粒子翻

维普资讯 http://www.cqvip.com

转而导致控制流错误的有效手段，本文重点介绍软硬结合和纯软件的实现方法。这两个方面的实现方法有着不同的侧重点和各自的应用范围。基于签名的并发控制流检测技术虽然在航天等多方面的应用中取得了很显著的成效，但是也面临着在提高控制流检测能力的基础上，协调好各检测技术指标的关系。这些问题的难点，主要来自控制流图的定义、其使用方法及需要适应新的处理器结构的发展，这些问题的解决，是基于签名的并发控制流检测技术未来研究的主要方向。

参 考 文 献

1 Pradhan D K. Fault-Tolerant Computer System Design[D]. Texas A&M University, 1996

2 Yau S S, Chen F. An approach to concurrent control flow checking[J]. IEEE Trans on Software Engineering, 1980, 6(2):126～137

3 Mahmood A, McCluskey E J. Concurrent error detection using watchdog processors-a survey[J]. IEEE Transactions on Computers, 1988, 37(2):160～174

4 Upadhyaya S, Ramamurthy B. Concurrent process monitoring with no reference signatures[J]. IEEE Transactions on Computers, 1994, 43(4):475～480

5 Madeira H, Camoes J, Silva J G. A watchdog processor for concurrent error detection in multiple processor systems[J]. Microprocessors and Microsystems, 1991, 15(3):123～131

6 Namjoo N. Cerberus-16: An architecture of a general purpose watchdog processor[A]. In: Symposium on Fault Tolerant Computing[C]. Proceedings of 13rd Int, 1983. 216～219

7 Michel T, Leveugle R, Saucier G. A new approach to control flow checking without program modification[A]. In: Symposium on Fault Tolerant Computing[C]. Proceedings of 21st Int, 1991. 334～341

8 Saxena N R, McCluskey E J. Control-Flow Checking Using Watchdog Assists and Extended-Precision Checksums[J]. IEEE Trans. on Computers, 1990, 39(4):554～559

9 Wilken K, Shen J P. Continuous Signature Monitoring: Low-cost Concurrent-detection of Processor Control Errors[J]. IEEE Trans on Computer Aided Design, 1990, 9(6): 629～641

10 Michel E, Hohl W. Concurrent error detection using watchdog processors in the multiprocessor system MEMSY[A]. In: Fault Tolerant Computing Systems[C]. Proceedings of 283, 1991. 54～64

11 Lu D J. Watchdog processors and structural integrity checking [J]. IEEE Trans on Comp, 1982, 31(7):681～685

12 Miremadi G, Karlsson J, et al. Two Software Techniques for On-line Error Detection. In: Twenty-Second International Symposium on Miremadi[C]. FTCS-22[J], 1992. 328～335

13 Majzik I, Pataricza A. Control flow checking in multitasking systems[J]. Periodica Polytechnica Ser Electrical Engineering, 1995, 39(1):27～36

14 Li Xiaobin, Gaudiot L-L, et al. A Compiler-Assisted On-Chip Assigned-Signature. In: Proceedings of 9th Asia-Pacific Computer Systems Architecture Conference[C], Beijing, Control Flow Checking[J], 2004. 554～567

15 Alkhalifa Z, Nair V S S, Krishnamurthy N, et al. Design and Evaluation of System-level Checks for On-line Control Flow Error Detection[J]. IEEE Trans on Parallel and Distributed Systems, 1999, 10(6):627～641

16 Oh N, Shirvani P, McCluskey E J, et al. Control Flow Checking by Software Signatures[A]:[Technical Report]. Center for Reliable Computing [C]. In: Proceedings of 51, 2002. 111～122

17 Benso A, Di Carlo S, Di Natale G, et al. Control-Flow Checking via Regular Expressions[A]. In: Proceedings of the 10th Asian Test Symposium, IEEE Computer Society[C], 2001. 299～303

18 Winter M, Zeidler C, Stich C. The PECOS software process[A]. In: Conf. on Software Reuse[C]. Proceedings of Workshop on Components-based Software Development Processes 7th Int, 2002. 76～83

19 Venkatasubramanian R, Hayes J P, Murray B T. Low-cost on-line fault detection using control flow assertions[A]. In: Proceedings of 9th IEEE International On-Line Testing Symposium, ACM Press[C], 2003. 340～353

20 Li Xiaobin, Gaudiot J-L. A Compiler-assisted On-Chip Assigned-Signature Control Flow Checking[A]. In: Proceedings of Asia-Pacific Computer Systems Architecture Conference, Springer Berlin [C], 2004. 554～567

21 Huang Y, Kintala C. Software Implemented Fault Tolerance: Technologies and Experience[A]. In: Proceedings of the 23rd Fault-Tolerant Computing Symposium, IEEE Computer Society [C], 1993. 2～9

22 Reis G A, Chang Jonathan, Vachharajani N, et al. Design and evaluation of hybrid fault-detection systems[A]. In: Proceedings of the 32th Annual International Symposium on Computer Architecture, IEEE Computer Society[C], 2005. 148～159

23 Namjoo N. Techniques for concurrent testing of VLSI processor operation[A]. IEEE Trans Computers[C]. In: Proceedings of Int Test Conference, 1982. 461～468

24 Sridhar T, Thatte S M. Concurrent checking of program flow in VLSI processors[A]. In: Proceeding of Int Test Conference, 1982. 191～199

25 Delord X, Saucier G. Formalizing signature analysis for control flow testing of pipelined risc microprocessors[A]. In: Proceeding of Int Test Conference, 1991. 936～945

26 Ohlsson J, Rimen M. Implicit signature checking[A]. In: Proceedings of 22nd Int Symposium on Fault Tolerant Computing, 1995. 218～227

27 Chen Yung-Yuan. Concurrent Detection of Control Flow Errors by Hybrid Signature Monitoring[J]. IEEE Transactions on Computers, 2005, 54(10):1298～1313

28 Avizienis A. The N-Version Approach to Fault-Tolerant Software[J]. IEEE Trans on Software Engineering, 1985, 11(12): 1491～1501

29 Shirvani P P, Saxena N, McCluskey E J. Software-implemented EDAC protection against SEUs[A]. In: Proceedings of IEEE Transactions on Reliability 49, 2000. 273～284

30 Mcfearin L, Nair V S S. Control-flow Checking Using Assertions [A]. Computer science& engineering[C]. In: Proceedings of IFIP Int'l Working Conf. Dependable Computing for Critical Applications, 1995. 103～112

31 Kanawati K, Krishnamurthy N, Nair S, et al. Evaluation of Integrated System-level Checks for On-line Error Detection[A]. In: Proceedings of IEEE Int'l Symp, Parallel and Distributed Systems [C], 1996. 292～301

32 Nair V S S, Kim H, Krishnamurthy N, et al. Design and Evaluation of Automated High-level Checks for Signal Processing Applications[A]. In: Proceedings of SPIE Advanced Algorithms and Architectures for Signal Processing Conf., 1996. 292～301

33 Rebaudengo M, Reorda M S, Violante M, et al. A source-to-source compiler for generating dependable software[A]. In: Proceedings of SCAM, IEEE International Workshop on Source Code Analysis and Manipulation, 2001. 33～42

34 Oh N, Shirvani P P, McCluskey E J. ED4I: Error detection by diverse data and duplicated instructions[A]. In: Proceedings of IEEE Transactions on Computers 51, 2002. 180～199

35 Schuette M A, Shen J P. Exploiting instruction-level parallelism for integrated control-flow monitoring[J]. IEEE Transactions on Computers, 1994, 43(2):129～133

36 Rotenberg E. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors[A]. IEEE Computer Society[C]. In: Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, 1999. 84～91

37 Vijaykumar T N, Pomeranz I, et al. Transient-fault recovery using simultaneous multithreading[J]. In: Proceedings of the 29th annual international symposium on Computer architecture, IEEE Computer Society[C], 2002. 87～98