第 37 卷 第 3 期
Vol.37 No.3

计 算 机 工 程
Computer Engineering

2011 年 2 月
February 2011

• 工程应用技术与实现 • 文章编号：1000—3428(2011)03—0230—03 文献标识码：A 中图分类号：TP311

# 安全编码预编译器的设计与实现

李 刚[a]，丁 佳[a]，梁盟磊[b]，薛小平[a]，王小平[b]

(同济大学 a. 信息与通信工程系；b. 计算机科学与技术系，上海 201804)

**摘 要：**针对轨道交通车载装备的安全性问题，基于安全编码处理器(VCP)的编码思想，设计并实现安全编码预编译器(VCPC)。VCPC 能将没有安全性的源代码转换成具有验证能力的安全代码，生成的安全代码可应用于 VCP 中的单处理器，实现对处理器各种故障的监测和保护。测试结果表明，该安全代码的剩余错误率可以达到 1/*A*。

**关键词：**安全编码处理器；预编译器；编码；故障检测

# Design and Implementation of Vital Coded Pre-Compiler

**LI Gang[a], DING Jia[a], LIANG Meng-lei[b], XUE Xiao-ping[a], WANG Xiao-ping[b]**

(a. Department of Information and Communication Engineering;

b. Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

**【Abstract】** Aiming at the safety problem of rail transit equipment, this paper designs and implements Vital Coded Pre-Compiler(VCPC) based on the coding principle of Vital Coded Processor(VCP). VCPC can transform the source codes into codes which are safe and can be verified. Faults are detected by the safe codes running on VCP's single processor. Test results show that the non detection probability of an error can be 1/*A*.

**【Key words】** Vital Coded Processor(VCP); pre-compiler; coding; fault detection

**DOI:** 10.3969/j.issn.1000-3428.2011.03.081

## 1 概述

计算机技术在安全关键系统中的应用大幅度提高了自动化程度，增强了系统功能，给系统设计和维护带来很多便利。但由于集成电路的复杂性、随机错误等，计算机中的处理器可能会出现永久性故障和瞬时性故障[1]，导致处理器无法正确运行，从而引起各种运算错误。处理器中的运算错误若不能被正确地检测出来，会给安全关键系统带来灾难性后果，如航天、轨道交通、核工业。

安全编码处理器(Vital Coded Processor, VCP)[2-3]是一种利用编码技术来检测处理器的错误，以保障硬件可靠和安全的理论和方法。该方法最初应用于法国巴黎地铁 A 号线的 SACEM 项目，每辆列车配备一个安全编码处理器，对车速控制进行安全防护，使行车间隔缩短了 30 s，大大提升了运输能力。该技术现已广泛应用于世界各地的轨道交通信号系统，如墨西哥地铁 A 号线和 8 号线、圣地亚哥地铁 5 号线和 1 号线、香港地铁 3 号线、上海轨道交通 3 号线和 10 号线。实践证明，这一方法能够有效地保障轨道交通信号系统的可靠性。

本文在讨论安全编码处理器技术的基础上，研究，设计并实现了对其关键核心部件——安全编码预编译器。测试结果表明，本文设计的安全编码处理器预编译工具可直接应用于安全关键系统的开发，如轨道交通的车载列车自动防护(Automatic Train Protection, ATP)。

## 2 安全编码处理器

VCP 将检错编码的思想应用在处理器中，对输入数据进行编码，使输出结果具有可验证性，在高层(应用层)实现对处理器各种故障的防护。

VCP 采用改进的算术码，它是在算术码的基础上附加数据签名和时间标签编码而成[2]的。例如，对数值变量 $x$，其编码后的形式可表示为：

$$X=A\cdot x+Bx+D \tag{1}$$

其中：

(1)$A$ 为素数，通过乘 $A$ 来构造 AN 码[4]。$A$ 又称为编码强度，保证剩余错误率(不可检测的错误率)为 $1/A$[5]。$A$ 越大，剩余错误率越小。

(2)$Bx$ 是编码时为变量 $x$ 分配的数据签名，其值为 1~$A$-1 之间随机选取的任意整数。不同的变量对应不同的数据签名，以保证码字的随机性。

(3)$D$ 为时间标签，用于检测循环中变量内存未更新的错误。对处于同一层循环内的所有变量，其码字中 $D$ 的值是相同的，每循环一次，$D$ 的值会增加一个 $\Delta D$，如果循环次数发生错误，该错误可以通过 $D$ 的值反映出来。

为便于程序实现，可将编码形式转化为分离码的形式，即将编码后的码字分解为 2 个部分：高 $n$ 位和低 $k$ 位。其中，高 $n$ 位为信息位，表征原变量的数据值；低 $k$ 位为校验位，用于处理器的检错。

令 $r_{kx}=2^k\cdot x \bmod A$，则 $2^k\cdot x-r_{kx}$ 是 $A$ 的倍数，以此替代 $A\cdot x$，式(1)可转化为：

$$X=2^k\cdot x-r_{kx}+Bx+D \tag{2}$$

为便于描述，将码字写成 $X=(xh, xl)$，其中，信息位占据高位，即 $xh=x$；校验位占据低 $k$ 位，即 $xl=-r_{kx}+Bx+D$。

图 1 描述了原始编码变量 $x$ 编码成最终分离码形式的编码变量 $X$ 的转化过程。基于这种思想编码的处理器可以检测运算过程中的各种错误。处理器中可能的错误主要有 3 种类

**作者简介：**李 刚(1986－)，男，硕士研究生，主研方向：可靠计算，故障虚拟化；丁 佳、梁盟磊，硕士研究生；薛小平，副教授、博士；王小平，教授、博士

**收稿日期：**2010-09-17 **E-mail：**ligang_sh@yahoo.com

型：运算错误、操作数错误和操作符错误[2]，(1)通过算术码检测 CPU 中的运算错误(如进位和移位错误)及计算过程中操作数被修改的错误。(2)通过数据签名 *B* 检测操作符错误(如加法运算中误用了乘法运算)和寻址出错导致的操作数错误(即取错了操作数)。(3)通过时间标签 *D* 检测循环过程中因内存未更新导致的操作数错误。

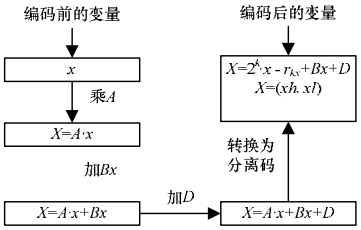


**图 1 VCP 中变量编码过程**

## 3 安全编码预编译器

从源程序到安全代码的转换是通过一个基于编码的预编译工具实现的。该工具基于 VCP 中的编码思想，对源程序进行转换，生成具有验证能力的安全代码，称为安全编码预编译器。为了保证数据经过程序处理后仍保持码字特征，VCP 需要对源程序进行编码，包括变量声明、运算语句和控制结构。与源程序相比，安全代码主要增加了针对数据校验位的操作，并输出用于校验的预计算签名。例如，对加法运算语句 *z*=*x*+*y* 编码，生成的安全代码如下：

```
{z_h=x_h+y_h;
z_l=x_l+y_l-D;
z_l=z_l%A;}
```

变量 *z* 的预计算签名为 *Bz*=(*Bx*+*By*)%*A*。

从源程序转换而来的安全代码再经过一般的编译器编译链接，生成安全的二进制程序，在处理器中执行。整个工作流程如图 2 所示。

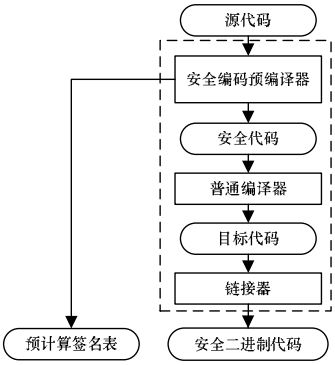


**图 2 安全二进制代码的生成过程**

为了实现 VCPC，本文将系统设计为 2 层：核心层和转换层，其结构如图 3 所示。

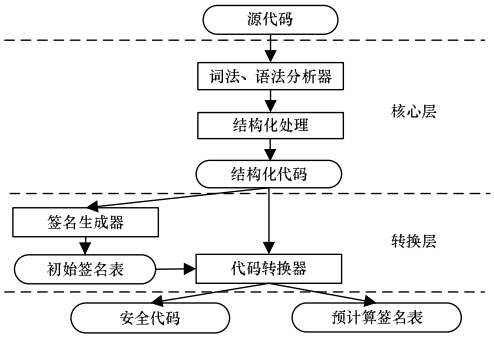


**图 3 VCPC 系统结构**

### 3.1 核心层

核心层也是系统的最底层，主要是对源程序进行词法、语法分析，并将分析所得的结果以结构的形式存储。该层是整个软件体系结构中的基础和核心环节，转换层建立在对该层分析的基础上。

核心层包括 3 个模块：词法分析器，语法分析器及结构化存储。

词法分析器的主要功能是检查源代码中的单词是否符合规范，并且将扫描到的合法单词转化成相应的标识符，便于后续阶段处理。语法分析器在词法分析识别出单词字符串的基础上，分析并判定程序的语法结构是否符合语法规则，对匹配语法规则的字符串组合进行规约和压栈。语法分析之后，将栈中的内容存入特定的数据结构中，称为结构化存储。

### 3.2 转换层

转换层主要实现签名的分配、预计算和安全代码的生成。

该层是软件体系结构中的功能部分，根据核心层的输出，按语法规范对结构递归遍历。从结构最顶层的语法单位(即声明)开始遍历，直至最基础的语法单位(即标识符)。遍历的目的是获取源程序中的变量声明、运算语句和控制语句等相关信息。当遍历到相应的语法单位时，按照编码算法进行一系列加工，生成所需的安全代码，如生成变量的校验位、输出冗余运算。

转换层主要由签名生成器和代码转换器 2 个模块组成。

在对结构遍历时，签名生成器根据签名的分配原则为遍历到的变量、常数分配数据签名，并将签名存储在表中，遍历结束时会生成一个与源程序相对应的初始签名表。

代码转换器在遍历结构的过程中，基于已分配的初始签名对变量和语句进行编码，生成安全代码。在编码的同时，对输出数据的签名进行预计算。因为源程序控制流程已知，所以可计算出输出数据的最终签名，这些签名存储在预计算签名表，预计算签名表将参与校验。

## 4 安全编码预编译器的实现

### 4.1 词法语法分析

本文基于开源的编译工具 Lex 和 Yacc[6]开发 VCPC 的词法和语法分析器。

根据 Lex 和 Yacc 规范，分别写出 VCPC 中的词法规则文件 vcp_scanner.l 和语法规则文件 vcp_parser.y。Lex 和 Yacc 通过编译词法规则文件 vcp_scanner.l 和语法规则文件 vcp_parser.y，能自动生成 C 语言格式的词法分析程序 vcp_scanner.c 和语法分析程序 vcp_parser.c。

如图 4 所示，当 VCPC 分析源代码时，首先进行词法扫描，并根据定义的词法规则将输入的源代码分解成标记(tokens)的形式；然后语法分析程序按照语法规则(BNF 范式)匹配来自 Lex 的标记组合并规约，同时将规约结果压栈。

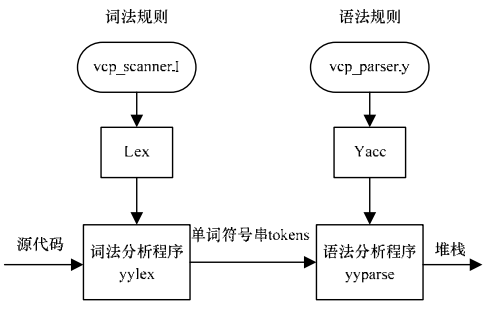


**图 4 词法语法分析示意图**

### 4.2 结构化处理

在语法分析之后，必须将堆栈中的内容存入特定的数据结构中，如二叉树、三元式、四元式。本文没有采用一般的方法，而是定义了新的数据存储方式来存储栈中的内容，称这种存储方式为结构。

结构化处理的关键是如何定义这种数据结构。数据结构的定义是根据 BNF 范式得出的，由于考虑重用机制以及存储和遍历的高效率，因此采用类来定义，并采用 C++标准模板库 STL 来提高效率。

以语法单位变量为例，其 BNF 范式[6]为：

```
var
:identifier /*普通变量*/
  |identifier exp_list /*数组变量*/
  |identifier . var /*结构体变量*/
  |identifier exp_list . var /*结构体数组变量*/
  ;
```

可见，变量的规约有 4 种情形，因此，在结构中定义变量类时，应设计相应的 4 种构造函数来存储变量的相关信息。变量类定义如下：

```
class Var
{private:
  Var* var_;
  string name_;
  list<Exp>* exp_list_;
public:
  Var(string);
  Var(string, list<Exp>*);
  Var(Var*, string );
  Var(Var*,string, list<Exp>*);
};
```

采用类似的方法，定义 BNF 范式中的所有语法单位，存储源程序的所有信息，实现结构化处理。

### 4.3 签名生成器

在对结构遍历的基础上，签名生成器会为源程序中的每个变量和常数分配签名，所有签名存储在初始签名表中。分配原则是保证不同变量、不同常数的签名互不相同。

签名生成算法如下：

(1)遍历源程序，识别程序中的变量及常数。

(2)根据随机算法为每一个变量和常数生成 1~A 之间的签名。

(3)如果当前的标识符为变量，则将(2)中分配的签名与已分配的签名进行比较，如果有重复，转(2)，如果没有重复，则将该变量及其签名存入初始签名表，转(5)。

(4)如果当前的标识符为常数，则查找初始签名表中是否已经存在该常数，如果存在，转(5)，否则，将该常数及其签名存入初始签名表，转(5)。

(5)检查遍历是否结束，若是，算法结束，否则，转(1)。

### 4.4 代码转换器

代码转换器根据遍历的运算类型进行签名的预计算和安全代码的生成。其内部维护一动态链表，记录所有变量的签名变化信息，称为动态签名表。每步运算编码后，更新该表中的签名。该模块的具体算法如下：

(1)将初始签名表中的内容拷贝到动态签名表。

(2)遍历源程序结构，识别运算或者语句，例如加法运算 z=a+b。

(3)从动态签名表中取出操作数的当前签名，如 Ba 和 Bb，并根据运算类型调用相应的签名计算函数计算左值变量的签名，如 Bz=sigAdd(Ba,Bb)。

(4)根据运算类型，调用相应的编码函数，输出对应的安全代码。对于加法运算则调用加法编码函数 vitalAdd 输出加法运算的安全代码。

(5)更新动态签名表中左值变量的当前签名。

(6)判断遍历是否结束，若结束，则将动态签名表中所有变量的当前签名写入离线签名表，否则，转(2)。

## 5 测试运行

本文使用一个约 8 000 行代码的 ATP 应用程序对 VCPC 进行测试。在测试中注意了 4 个方面：

(1)为便于编程，同时不影响处理器速度，选取 A 为 97 进行编码。

(2)开发了专门的界面程序，能够导入 ATP 工程文件及配置信息，进行多文件编码，显示编码过程中的错误提示信息。

(3)为模拟 VCP 中的硬件校验，编写了相应的校验驱动程序。

(4)对编码之后的 ATP 程序进行故障注入，测试安全代码的故障防护能力。

为了便于软件实现，采用文献[7]提出的软件故障注入方法，故障的注入通过 Linux 的 ptrace()系统调用实现。在编码的 ATP 应用程序执行过程中，通过一个脚本，连续执行故障注入程序，直到获得统计所需的足够信息。

所注入故障的种类为第 2 节提到的 3 种错误类型，即运算错误、操作数错误和操作符错误，因为从数据处理的角度，处理器中所有可能的错误可归为上述 3 种错误。

故障注入后，逐渐增加 ATP 程序的运行次数，得到剩余错误率关于运行次数的折线图，如图 5 所示。

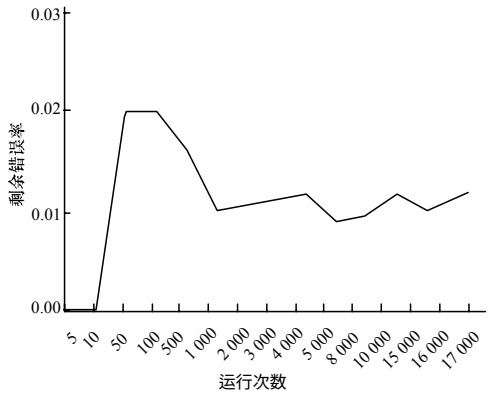


**图 5 剩余错误率与运行次数关系**

从图 5 可以看出，在运行次数较少时，剩余错误率波动范围比较大；运行次数增多时，剩余错误率趋于稳定，为 0.01。因为 VCP 选取的编码方法是一种概率性编码，只有通过大量的运行才能反映出其用于检错时的剩余错误率。在本文测试中，A 选 97，剩余错误率理论值为 1/97≈0.01。可见实际测试中得到的剩余错误率基本符合理论值。

## 6 结束语

本文基于安全编码处理器冗余编码的思想，设计并实现了安全编码预编译器工具，实验结果表明，该安全处理器预编译器完全可以满足目前轨道交通车载设备的应用要求。

(下转第 235 页)