基于 DSP 软加固下的功耗优化方法

姚天问 周海芳 方民权 申小龙

（国防科学技术大学计算机学院 长沙 410073）

摘 要 由于太空中存在各种宇宙射线辐射，导致星载设备产生可靠性问题，使得高性能数字信号处理器（DSP）在航空航天中的应用受到制约，因此需要采取容错措施来对其进行加固处理。但软加固算法会使程序复算而增加系统开销，文章基于 C6748 DSP 平台，详细说明了在不影响检错率的情况下的功耗优化方法。实验结果表明，通过此方法在保证检错率的条件下能显著降低系统开销，同时也提升了执行效率。

关键词 DSP；抗辐照；软加固；功耗优化

Power Optimization Method Based on DSP Software Fault Tolerance

YAO Tianwen ZHOU Haifang FANG Minquan SHEN Xiaolong

(*School of Computer Science*, *National University of Defense Technology*, *Changsha* 410073, *China*)

Abstract Because the various cosmic radiation in space leads the spaceborne equipment to produce some reliability problems, the application of high performance digital signal processor (DSP) is constrained in the aerospace. Therefore measures need to be taken for fault-tolerant processing. But the soft harden or reinforcement algorithm makes program to repeat calculation and increase the system energy cost. This paper, based on C6748 DSP platform, studies the power consumption optimization method in detail. The experimental results show that this method can significantly reduce the system energy cost and enhance the execution efficiency under the conditions of ensuring error detection rate.

Keywords DSP; anti radiation; software fault tolerance; power optimization

1 引 言

随着航天技术的迅猛发展，为了提升星载处理器的性能，需在设计上采用更小的工艺尺寸、更低的工作电压以及更高的时钟频率[1]。然而集成电路制造工艺的提升使得处理器对软错误也变得越来越敏感，可靠性问题则成为星载计算机领域研究的焦点问题之一。

软错误（Soft Error）也称为瞬态故障，通常是由宇宙环境中的高能粒子辐射、电磁干扰、电压干扰等电磁噪声诱发[10]。它具有瞬态、可重加电恢复、发生位置和时间随机等特点。虽然它不会导致处理器物理性能损坏，但会改变信号存储或传输的值。

为了解决软错误问题，通常采用的方法是冗余。而我们通过软件方法来实现冗余、复算的过程，对程序进行加固处理。相比使用硬件冗余来说，它具有成本低、应用灵活、独立于具体硬件、利于实现优化等优点。

星载设备是作为使用电池供电的嵌入式系统，它在功耗方面往往有着十分严苛的要求[2]。软加固能有效解决可靠性问题，但软加固处理会引起系统开销的增加。本文通过对 DSP 实现软加固并在保证检错率的情况下进行功耗优化处理，不仅能成功降低系统开销，同时提升了执行效率。

基金项目：国家自然科学基金（61272146）。

作者简介：姚天问（通讯作者），硕士研究生，研究方向为软件低功耗处理，E-mail：supermanytw@126.com ；周海芳，副研究员，研究方向为高性能图像处理；方民权，硕士研究生，研究方向为高性能图像处理；申小龙，硕士研究生，研究方向为图像模式识别。

本文结构安排如下：第一节引言；第二节简要介绍软件检错算法；第三节详细阐述基于 DSP 的软加固方法以及加固后的功耗优化方法；第四节通过实验，在模拟仿真的基础上对程序错误检测率、能量开销、执行效率进行分析；第五节对本文进行总结。

2 软件检错算法简介

软错误对计算机的影响可以分为控制流错误和数据流错误[8]。控制流错误是指故障改变了程序正确执行控制流的顺序；数据流错误是指内存和寄存器等存储部件中的数据发生单粒子翻转等软错误。下面简要介绍相应的几种软件检错算法。

控制流检错算法是采用基本块的标签分析法。在编译时为每个基本块分配唯一的静态标签，程序在运行过程中会根据当前的控制流产生动态标签，然后在每个基本块头部来比较两个标签，如果匹配则说明没有发生控制流错误，反之则说明发生了控制流错误。Stanford 大学提出的 CFCSS 就是基于前驱判定的控制流检测算法。

数据流检错算法常用的有 EDDI 和 ED^{4}I 等方法。EDDI 是将指令进行冗余复制，在程序发生跳转之前将两个执行的结果进行比较，若相符则说明无错误，反之说明发生数据流错误。ED^{4}I 是在 EDDI 的基础上引入了数据差异性的概念，以进一步提高错误检测率。

Princeton 大学提出的 SWIFT 方法是在 EDDI 和 CFCSS 的基础上进行了多项改进[9]。它的基本思想与 EDDI 一致，然而 SWIFT 并没有考虑指令调度、编译器优化等方面的问题，而且增加了基于标签签名的控制流检测机制 CFCSS。其主要改进为：假设内存部件已经被保护，SWIFT 针对内存读取数据的指令进行复制，然后插入副本指令进行冗余计算，在写入内存前插入比较指令。与 EDDI 相比，SWIFT 只使用了一半内存，访问操作也减少了一半，因此性能有了较大提高。

3 DSP 软加固方法及功耗优化

本节首先对 TMS320C6748 DSP 的结构和流水特点进行简要介绍。然后研究 DSP 的程序软加固方法以及加固后的功耗优化方法。

3.1 C6000 系类 DSP 结构和流水特点

美国德州仪器公司的 TMS320C6000 DSP 系列，采用VLIW（Advanced Very Long Instruction Word）体系结构，具有 8 个功能单元，一个时钟周期最多可以并行执行 8 条指令。在 VLIW 结构中，指令并行性和数据传送完全是在编译时确定的，因而 VLIW 结构处理器的代码效率在很大程度上取决于代码压缩的效率[6]。

VLIW 的另一个特点是指令获取、分配、执行，数据存储等阶段需要进行多级流水，而且不同指令执行的流水延迟时间也不相等。为了保证执行效率，各种指令的安排要尽量不破坏指令流水的执行。图 1 为 TMS320C6000+ 的流水结构示意图。

C6000 的 VLIW 采用类 RISC 指令集，使用较大的统一的寄存器堆，结构规整，具有潜在的易编程性和良好的编译性能。它与超标量等系统结构相比起来较为简单，在科学应用领域可以发挥良好的作用。

Fetch Packet	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
n	PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	
n+1		PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
n+2			PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9
n+3				PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8
n+4					PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7
n+5						PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6
n+6							PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5
n+7								PG	PS	PW	PR	DP	DC	E1	E2	E3	E4
n+8									PG	PS	PW	PR	DP	DC	E1	E2	E3
n+9										PG	PS	PW	PR	DP	DC	E1	E2
n+10											PG	PS	PW	PR	DP	DC	E1

图 1 TMS320C6000+ 的流水结构图

3.2　DSP 程序软加固方法

在 C6000 系列 DSP 中，可以用线性汇编对影响速度的关键 C 代码段进行重新编写。在 C 语言等高级语言级实现冗余计算的优点是独立于具体应用平台和编译器，容错策略的实现比较容易，而且用户还可以根据自己的需要自由配置和扩展具体的容错策略。但 C 语言等高级语言执行效率要低于汇编和线性汇编，而相较指令级容错来说，其缺点是产生太多的冗余代码，开销太大。由于线性汇编代码中不需要指定流水线延迟、寄存器的使用、功能单元的分配等信息，因此基于线性汇编代码进行软加固较基于汇编代码要简单得多，这有利于实现容错策略的自动化运行。同时，其通常程序效率均可达到标准汇编的 95% 以上[7]，性能与汇编语言基本相当。因此本文首先通过线性汇编实现 DSP 程序软加固算法，然后通过汇编优化器，实现算法的性能提升。

在检错算法的选取上，由于 SWIFT 检测算法同时具备控制流和数据流检测的功能，并且拥有较高的效率，所以本文使用 SWIFT 算法来对程序进行容错加固。实验代码使用的是数组累加乘，C 代码如下：

```
void Dot(int* x, int* y, int* z, int n)
{
    int i,j = 0;
    for(i = 0; i < n; i++)
    {
        j+= x[i] * y[i];
    }
    *z = j;
}
```

将C代码转换为线性汇编并用SWIFT算法软加固：

```
    .global _Dot
_Dot:.cproc x, y, z, n
.reg con, sum, a_r, b_r, apb_r
.reg con1, sum1, a_r1, b_r1, apb_r1
.reg S1,S2,S3,R,G,NE
        MVK     0001    ,S1
        MVK     0010    ,S2
        MVK     0011    ,S3
        MVK     0000    ,G
        MVK     0001    ,R
        XOR     G       ,R      ,G
        MV      n       ,con
        MV      n       ,con1
        MVK     0       ,sum
        MVK     0       ,sum1
        XOR     S1      ,S2     ,R
LOOP:   XOR     G       ,R      ,G
        LDW     *x      ,a_r
        LDW     *x++    ,a_r1
        LDW     *y      ,b_r
        LDW     *y++    ,b_r1
        SUB     con     ,1      ,con
        SUB     con1    ,1      ,con1
        MPY     a_r     ,b_r    ,apb_r
        MPY     a_r1    ,b_r1   ,apb_r1
        ADD     apb_r   ,sum    ,sum
        ADD     apb_r1  ,sum1   ,sum1
[con1]  XOR     S2      ,S2     ,R
[con]   B       LOOP
[!con1] XOR     S2      ,S3     ,R
        XOR     G       ,R      ,G
        CMPEQ   G       ,S3     ,NE
[NE]    CMPEQ   sum     ,sum1   ,NE
[!NE]   B       ERROR
        STW     sum     ,*z
ERROR:
[!NE]   MVK     -1      ,sum1
[!NE]   STW     sum1    ,*z
    .endproc
```

上面代码，是通过复算比较得出的计算结果。如果数据不一致，那么输出为－1；数据一致则输出正确结果。

3.3　软加固后功耗优化方法

软件的功耗是由软件运行的周期数乘以每周期消耗的能量来决定的[3]，其优化方法通常是降低程序执行的周期数，这样不仅可以降低功耗也能提高程序运行效率。

为了使得加固后的程序循环体流水线编排更加高效，需要消除加固后程序循环内部引入的跳转指令。本文提出一种对软加固后的程序循环体进行优化的方法，以消除循环体内部的控制代码，从而使得循环体可以编排出高效的流水线，在不降低错误检测率的同时降低软加固后程序的执行周期数。

软件容错加固技术的思想都是向程序中插入一部分冗余信息，并在程序中合适的位置设置检查点，进而比较这部分冗余信息与程序中的对应信息是否相

符。若不相符则检测到错误，并立即跳转到相应的错误处理程序进行错误处理和恢复。而加固后程序的循环体内部设有检查点，会给循环体内引入跳转指令，带来巨大性能的开销。

在程序中使用一个通用寄存器 Y 来保存循环中是否检查到错误这一信息。它并不会去消除程序循环体中设置的检查点，而且依然在每个检查点进行同步比较，但检测到错误之后并不立即跳转，而是将这一信息保存到 Y 寄存器中，等到循环体执行结束后，再根据 Y 寄存器的信息来判断是否需要跳转到错误处理程序上，如图 2 所示，左边为未优化的容错流程，右边为优化后的容错流程。我们可以给 Y 赋上初值为 1，每一次比较的结果都与 Y 的值相与，并以相与后的值重新赋给寄存器 Y，最后通过比较 Y 的值来判断是否有错误发生。这样，通过将循环内错误处理的跳转延迟到循环外执行，虽然增加了容错延迟，但可以使循环体得以高效编排流水线，从而大大降低了程序执行周期数。

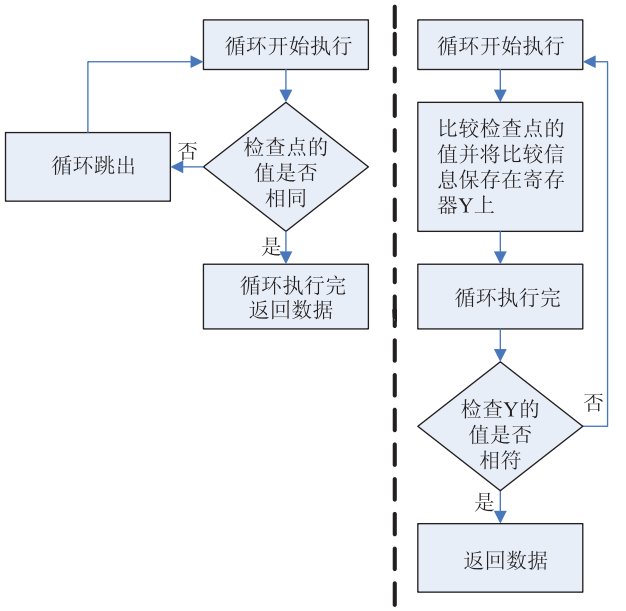

图 2 优化流程图

4 实验结果

实验首先分析错误检测率，要保证在错误检测率不变的条件下再分析能量开销。

4.1 故障注入实验

故障注入涵盖控制流和数据流两部分。注入是通过在程序单步执行的某个时刻，手动修改寄存器或者某个通用寄存器的值来实现。本文使用的数组累加乘程序(DOT)数组 X 和 Y 各有 50 个数字，故障注入 400 次，根据对程序的不同影响把注入的故障分为 4 类：

(1) Correct(CRC)：注入的故障没有影响程序的正确执行。

(2) Exception(EXP)：注入的故障被 DSP 系统检测出，或者造成程序异常终止。

(3) Wrong(WRG)：系统正常退出但结果错误。

(4) Detected(DTC)：检测算法检测到错误。

错误检测率的计算方法为：FailRate＝WRG/400。表 1 给出了程序故障注入的结果。

从表 1 可以看出，程序未加固经过故障注入实验后，失效率为 41.75%。通过 SWIFT 算法加固后，未优化的程序失效率为 1.75%，优化后的程序失效率为 2%。对比 SWIFT 线性汇编代码和优化后的线性汇编代码所取得的加固效果，二者失效率基本相当。

4.2 能量开销与性能分析

由表 1 能看出，原汇编代码优化后，检错率下降不明显，说明是能够保障容错率的。能量开销分析分为每周期能量消耗分析与程序周期数分析，通过分析这两组数据，来比较程序的能量开销与执行效率。

本文通过 Simple-wattch 来搭建能量模型，通过软件能量模型来计算每周期消耗的能量，通过 TI 公司的 CCS4.1.2 开发环境来得到程序的执行周期数。表 2 为在错误注入条件下的性能与开销。

从表 2 能够看出，未优化的程序，因为需要跳转出循环而不能排成流水线，效率太低。而优化后，程序完美地利用了软件流水线，执行周期数大大降

表 1 故障注入结果

影响 / 程序	*CRC*	*EXP*	*WRG*	*DTC*	*FailRate*（%）
线性汇编（未加固）	103	130	167	0	41.75
线性汇编（SWIFT）	110	40	7	243	1.75
优化后（SWIFT）	119	41	8	232	2

表 2　性能与开销

性能与开销 / 程序	周期数	每周期能量开销（nj）	总能量开销（j）
线性汇编（未加固）	144	1075283	0.1548
线性汇编（SWIFT）	1845	825133	1.5224
优化后（SWIFT）	150	1074824	0.1612

低，执行效率的提升非常明显。尽管每周期能量开销会增加，但随着周期数大大降低，总能量开销也有非常明显地下降，总能量开销为优化前总能量开销的 10.59%。

5　结束语

本文介绍了软件检错算法，提出了一种基于软加固的功耗优化方法，并在 DSP 平台上加以实现与验证。本方法的主要思想是尽量不破坏流水线，更加充分地利用起硬件的并行处理功能。实验结果表明，在不影响错误检测率的情况下，本软件功耗优化方法能显著降低程序的能量消耗，并且大大增加了程序执行效率，它的成功实现能够被应用于在恶劣环境下的各种嵌入式系统中。

参 考 文 献

[1] 陈娟. 低功耗软件优化技术研究 [D]. 长沙: 国防科大研究生院, 2007.

[2] 罗刚, 郭兵, 沈艳, 等. 源程序级和算法级嵌入式软件功耗特性的分析与优化方法研究 [J]. 计算机学报, 2009, 32(9): 1869-1876.

[3] Yang HB. Power-aware compilation techniques for high performance processors [D]. Newark: University of Delaware, 2004.

[4] Kandemir M, Vijaykrishnan N, Irwin MJ, et al. Influence of compiler optimizations on system power [C] // Proceedings of the 37th Annual Design Automation Conference, 2000: 304-307.

[5] Kim NS, Austin T, Mudge T, et al. Challenges for architectural level power modeling [C] // Robert Graybill, Rami Melhem. Power aware computing. USA: Kluwer Academic Publishers. 2002: 317-337.

[6] Texas Instruments, TMS320C6000 Programmer's Guide [M]. 2006.

[7] Texas Instruments, TMS320C6000 Optimizing Compiler User's Guide [M]. 2006.

[8] 邢克飞, 杨俊, 周永斌. 星载高性能 DSP 加固设计方法研究 [J]. 电子器件, 2007, 30(1): 206-209.

[9] Reis GA, Chang J, Vachharajani N. SWIFT: Software implemented fault tolerance [C] // Proceedings of International Symposium on Code Generation and Optimization, 2005: 243-254.

[10] 徐建军. 面向寄存器软错误的容错编译技术研究 [D]. 长沙：国防科大研究生院, 2010: 85-100.