低代价锁步EDDI:处理器瞬时故障检测机制

王 超[1) 傅忠传[1) 陈红松[2) 崔 刚[1)

[1)(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

[2)(北京科技大学计算机与通信工程学院 北京 100083)

摘 要 随着ULSI工艺步入深亚微米时代,处理器内部组合逻辑的瞬时故障敏感性迅速提高,文中在设计初期将硬件寄存器纠检错能力和系统软件检错能力纳入考虑,兼顾处理器内组合逻辑、时序逻辑两类部件,设计应用级"低代价锁步EDDI(Error Detection by Duplicated Instructions)"机制.创新如下:(1)提出基于概率论的故障漏检率量化估计方法,为纠检错与性能折中进行指导.以往的应用级检错机制在设计过程中并没有考虑到下层操作系统的检错能力,这会造成可靠性估计不足而带来性能损失.文中依照指令流经的部件将故障划分为不同子类,并将操作系统纳入考虑,提出基于概率论的故障漏检率量化估计方法,理论估计与故障注入结果拟合良好.(2)低代价锁步EDDI机制,结合硬件纠检错能力,兼顾处理器内组合逻辑和时序逻辑两类部件,大幅降低了性能代价.提出独特的低代价锁步指令复制规则,并通过编译链前端的寄存器分配,大幅减少了寄存器预留数,有效缓解了寄存器压力,降低了访存代价,提高了寄存器的性能.寄存器预留也保证了本机制无需修改编译器传参规则,无需重新编译系统库,提高了通用性.(3)采用单比特故障模型,基于SPARC体系结构,选取处理器中代表性部件:解码(Decoder Unit)单元、地址生成(Address GEN Unit)单元、算逻单元(ALU)进行故障注入,对低代价锁步EDDI实现代价进行详细评测.与全复制EDDI相比,低代价锁步EDDI仅以故障漏检率SDC(Silent Data Corruption)平均升高0.8%的代价,换取了动态执行指令数平均减少36.1%,执行时间平均降低35.2%的性能优势.

关键词 EDDI;故障漏检率;组合逻辑;瞬时故障;SEU

中图法分类号 TP302 **DOI号**:10.3724/SP.J.1016.2012.02562

Cost-Effective Lock-Stepped EDDI: Transient Fault Detection Mechanism

WANG Chao[1) FU Zhong-Chuan[1) CHEN Hong-Song[2) CUI Gang[1)

[1)(*Department of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

[2)(*School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing* 100083)

Abstract As semiconductor technology scales into deep submicron regime, transient fault vulnerability of combinational logic increases rapidly, and will overtake that of the sequentials. Giving attention to both combinational logic and sequential logic components in a processor, especially taking aim at the hard to protect combinational logic, an instruction level transient fault detection mechanism, namely Cost-Effective Lock-stepped EDDI, is proposed. In the initial stage of design, the underling factors of both hardware detection and correction capability and system software fault detection capability are taken into account. The primary contributions of this paper are as follows: Firstly, a novel theoretical SDC(Silent Data Corruption) rate quantitative estimation methodology basing on probability theory is proposed for guidancing the tradeoff between fault tolerance and performance. It is the underestimation of the system reliability because of the ignorance of system software level fault tolerance capability, that incurs great performance over-

收稿日期:2011-01-11;最终修改稿收到日期:2011-10-25.本课题得到国家自然科学基金(90818016)、航空电子系统综合技术国防科技重点实验室和航空科学基金(20095577012)、哈尔滨工业大学创新基金(2009066)资助.**王 超**,男,1982年生,博士研究生,研究方向为容错计算、体系结构可靠性、大规模集成电路测试与验证.E-mail: wangchao1221@126.com.**傅忠传**,男,1970年生,副教授,研究方向为容错计算、计算机系统结构、多核.**陈红松**,男,1977年生,副教授,研究方向为高性能低功耗网络处理器、Ad hoc网络安全.**崔 刚**,男,1950年生,教授,博士生导师,中国容错计算机学会容错计算专业委员会委员,研究领域为可信计算等.

head of fault detection algorithm. Nevertheless, in the initial stage of design the underling factor of operating system is taken into account. Meanwhile, faults are classified into different categories according to the processor components utilized by an instruction. Fault injection experiment results prove that the theoretical quantitative estimations fit well. Secondly, giving attention to both combinational logic and sequential logic faults, the Cost-Effective Lock-stepped EDDI mechanism is designed, whilst a novel instruction duplication rule and a register allocation scheme are put forward. Coupling with EDAC(Error Detection And Correction) hardware for sequential logic, the Lock-Stepped EDDI especially effective for combinational logic greatly reduces performance overhead. Wherein, the novel register allocation scheme in compiler front-end results in a large reduction of reserved registers, alleviating register pressure and significantly reducing the numbers of load/store instructions. Likewise, with the introduction of register reservation, there is no need to modify function parameter passing rule, no need to recompile system libraries, which is of great importance to the generality of this mechanism. Thirdly and lastly, single-bit fault model is adopted. Representative components in SPARC processor such as decoder unit, address generation unit, and ALU are conducted for fault injection campaign. Comparing to the traditional Fully Replicated approach, Cost-Effective Lock-stepped EDDI achieves an average 35.2% speedup in execution time and 36.1% reduction in number of dynamic instructions at the modest cost of an average 0.8% increase in SDC rate.

Keywords　EDDI (Error Detection by Duplicated Instructions); SDC(Silent Data Corruption); combinational logic; transient fault; SEU (Single Event Upset)

1 引　言

瞬时故障(transient fault),常被称为软故障(soft error),主要由宇宙射线和封装中的 α 粒子引发,通常导致器件暂时性失效或状态改变,不会产生物理性损伤. 瞬时故障对于组合电路和时序电路的影响有所不同. α 粒子或宇宙射线进入内存单元或动态寄存器,引发比特翻转(bit-flip)或不确定故障(indetermination fault),一经锁存就会持续到重新写入或刷新. 除此之外,还会诱发组合逻辑产生干扰脉冲[1]. 组合逻辑对此类脉冲虽有屏蔽能力,但随着特征尺寸的缩小、供电电压的降低和工作频率的提升,故障易感性将越发提升. 文献[2]预计 2011 年组合逻辑故障易感性将会与时序逻辑持平,之后将超过后者.

瞬时故障并非由设备内因引起,受环境因素影响较大,具有随机性,这无疑对诊断提出了挑战. 2000 年美国在线和 ebay 曾暂停服务,原因在于所使用的 UltraSPARC-II 服务器中的 SRAM 对瞬时故障防护不足[3]. 2005 年 Los Alamos 国家实验室一台 2048-CPU 的惠普超级计算机因太空辐射引发的软故障频繁崩溃. Cypress 半导体公司也在 2004 年公布了一系列瞬时故障事件[4].

目前,硬件提供的瞬时故障纠检错机制主要有 ECC、校验和. 该类方法对时序逻辑瞬时故障行之有效,却无法覆盖组合逻辑. 高端机型一般采用多模冗余或多机备份的方式对组合逻辑进行保护,如惠普的 non-stop[5]、IBM 高端大型机[6]. 但硬件代价是一般商用机型无法承受的. 在这个方面,软件冗余以其灵活性和低硬件代价的特点格外具备竞争力. 软件冗余一般通过在编译的前、后端增加组件自动实现冗余功能. 主要包括增加控制流软件签名、复制指令备份数据以及软件 EDAC 等方式. 实现的过程中,处处存在着可靠性与性能的折中,例如软件签名代价较低但是只对控制流有效,指令复制会导致内存占用翻倍,访存代价过高等.

不同体系结构下,EDDI 机制的移植工作也存在亟待解决的问题. 例如,SPARC 属窗口寄存器体系结构,具有独特的寄存器环结构,这与在 ARM、MIPS 等平面寄存器体系结构下的移植工作相比,机制的移植工作面临完全不同的问题与挑战. 例如,编译器传参规则的修改,系统库的重新编译等工作,严重降低了机制的通用性.

基于以上分析，本文提出了一种软硬件混合的冗余方式——低代价锁步EDDI机制.期望最大化利用普遍使用的硬件纠检错机制，同时将处于应用层之下的系统软件纠检错能力(如段越界检查、非法指令检查等)纳入考虑，对指令复制机制进行裁剪和设计，以达到整个系统的可靠性与性能高效结合.本机制主指令与影子指令以锁步(lock-stepped)方式执行，以编译选项的形式集成在编译链后端，创新如下：

(1)本文依照指令流经的部件将故障划分为不同子类，经过对每种子类故障详细分析给出基于概率论的故障覆盖率量化估计方法，为故障覆盖率与性能的折中提出指导.经实验验证，理论估计与故障注入结果拟合良好.

(2)以往的应用级检错机制在设计过程中并没有考虑到下层操作系统的检错能力，这通常会造成可靠性估计不足而带来性能损失.在本机制设计初期将操作系统因素纳入考虑.

(3)提出低代价锁步的指令复制规则和寄存器分配方式.锁步规则结合硬件纠检错机制，既兼顾组合逻辑、时序逻辑两类部件，又大幅降低了性能代价.同时，编译前端实现的寄存器分配方式，大幅减少了寄存器预留数、有效缓解寄存器压力，降低了访存代价.寄存器预留机制的引入也保证了机制实现无需修改编译器传参规则，无需重新编译系统库，提高了本机制的通用性.

(4)实验验证阶段，选择单比特故障模型，在SPARC体系结构平台，选取处理器中代表性部件：解码(Decoder Unit)单元、地址生成(Address GEN Unit)单元和算逻单元(ALU)进行故障注入.

本文第2节介绍国内外相关工作；第3节阐述故障覆盖率理论估计方法；第4节对低代价锁步EDDI机制进行描述；第5节介绍故障注入实验和性能测试并对结果进行分析；最后得出结论.

2 相关工作

SIHFT(Software Implemented Hardware Fault Tolerance)采用软件手段实现硬件故障容错，源于美国斯坦福大学，并于1999年进行星载实验，引起广泛关注[7-9].SIHFT包括多种技术：针对存储系统故障提出的软件EDAC，针对处理器内瞬时故障引发控制流错误的CFCSS(Control Flow Checking by Software Signatures)、针对引发的数据流错误的EDDI(Error Detection by Duplicated Instructions)技术以及针对永久故障的ED^4I[10-13]等.

国内外本领域的相关工作一直较为活跃.普林斯敦大学提出了SWIFT(Software Implemented Fault Tolerance)来检测处理器内发生的数据流错误与控制流错误，并设计相关优化策略[14].法国TIMA实验室提出具有全覆盖能力的DSM检错技术[15].近年来国内多个课题组对此展开跟踪研究[16-19].

EDDI/CFCSS/ED^4I等技术通常在编译链后端实现，与体系结构和ISA紧密相关，可移植性受到制约.Yu等人[20]基于LLVM编译工具链，在SSA高级中间代码表示级别对应用进行加固，并提出多种优化机制降低实现代价，成功解决了可移植性问题.但是，该方法与编译链前端耦合紧密，开发周期长，且随编译器不同版本的发布，需频繁修改.

3 可靠性理论估计

提出锁步机制目的在于追求整体故障检测能力最大化，同时将机制间故障覆盖能力的重叠降到最低，以此降低性能代价.为实现这个目标，需要综合考虑操作系统、固件以及硬件具备的故障检测能力，并在此基础上设计指令复制规则.

本节提出了基于概率论的故障覆盖率估计方法，用于指导故障覆盖率与性能的折中策略.本文主要针对处理器内瞬时故障，故障发生后，经微结构级、结构级蔓延，传播至操作系统，最终在应用层外显.故障的传播路径涵盖多个层级，各层级的特点决定了它们在故障纠检错方面的不同角色.

据此，本文采用面向部件的故障分类方法，以紧密结合耦合路径的故障覆盖特性对覆盖率进行准确估计.本文选择了单比特故障模型，并以SPARC V9指令集为例进行验证，本文方法适用于所有RISC指令集.

3.1 故障分类

首先，将指令集分为计算指令(computational instruction)、访存指令(memory access instruction)、跳转指令(branch instruction)和其它类指令(other instruction).其中，计算指令包括所有算术运算、逻辑关系运算和赋值指令.访存指令包括取数、存数和内存栅栏指令.跳转指令包括无/有条件跳转和函数调用(call)指令.其它类指令是除了计算指令、访存指令、跳转指令以外的指令集合，同时为涵盖故障引

发的正常指令转为非法指令的情况，此处将非法指令纳入其它类指令中. 每种指令发生的故障仅包括两种情况：操作码故障和操作数故障. 据此，将故障类型细分为 14 子类，基于 SPARC V9 的指令分类及故障类型详见表 1. 表中各故障类型发生概率由故障源所在部件和指令集编码决定，例如，解码部件故障涵盖全部类型. 但是同类型（C→C、M→M、B→B 和 OT→OT）转化概率明显高于类型间转化，这一点由图 1 给出的指令集编码可知.

表 1　指令分类及故障类型

类别	符号	指令类型	故障类型
计算指令	C	arithmetic	**C→C**
		logical	C→M
		mov	C→B
		cmp	C→OT
访存指令	M	load	**M→M**
		store	M→C
		membar	M→B
跳转指令	B	branch	**B→B**
		fbranch	B→C
		call	B→M
			B→OT
其它指令	OT	sethi	**OT→OT**
		nop	OT→C
		illegal	OT→B

Format 1（op=0，1）：CALL && Branches(Bicc, BPcc, BPr, FBfcc, FBPfcc)

00 01	a	cond	op 2	disp 22

Format 2（op=2）：Arithmetic, Logical, MOVr

10	dst	opcode	src 1	0/1	src 2　or　simm 13

Format 3（op=3）：MEMBAR, Load, and Store

11	dst	opcode	src 1	0/1	src 2　or　simm 13

Format 4（op=0）：sethi,nop

00	dst	100	simm 22

图 1　SPARC V9 指令编码

如果故障源自算逻部件，只可能触发 C→C 一类，发生概率为 100%. 由此可知同类型故障能否良好覆盖，对系统的纠检错能力起关键作用，因此在表 1 中重点标示.

3.2　故障模型

瞬时故障在组合逻辑诱发后，在多级逻辑门传输路径中蔓延，被微结构级（锁存器、触发器）时间窗口捕获后，通常导致多比特故障. 但我们采用微结构级单比特故障模型，主要因为：(1) 目前尚无统一的多比特故障模型. 因为故障传播特性与设计细节，如结构/微结构级、门级，乃至设备级，耦合紧密；(2) 由于多比特模型增大了出错的概率，实验中会更容易被容错机制（如 EDDI 机制）检测到，从而导致过高地估计容错机制带来的可靠性提高. 出于对用户负责的角度，我们有理由选择后者.

单比特模型假设系统同一时刻只触发一次瞬时故障，故障经传播到达微结构/结构级引发单比特翻转. 不同部件发生故障的概率依部件复杂度服从均匀分布（本文假设各部件单比特故障触发概率相等）.

3.3　部件模型

地址生成部件

地址生成部件计算得出当前指令和下条指令地址（PC/NPC），由取指逻辑访存将指令取回. 该部件属组合逻辑部件，故障触发将直接影响 PC/NPC，发生概率为 100%.

此类故障引发控制流错误，包括 C→B、M→B、B→B 和 OT→B 类型. 这也是指令复制类容错机制覆盖能力相对薄弱的方面. 然而，操作系统对此提供了强大覆盖能力（具体数据见 5.2 节）. 地址翻转发生在低位，会导致字未对齐，引发总线故障；发生在高位，通常造成段越界访问，引发段错误. 对于后者，具体的故障覆盖能力取决于程序规模. 这些现象在故障注入过程中表现十分明显.

解码部件

解码部件负责解析存放于指令队列中，等待进入流水的指令操作码. 解码完成后负责将生成的控制信号送到相关执行部件，如算逻单元. 该部件故障包括全部 14 种类型. 由计算指令引发的 4 种类型，发生概率最大的是 C→C，该类故障无法由硬件和操作系统覆盖，因此需要在机制中将计算指令复制和锁步比较.

其余 10 类故障经估计和实验验证，会以较高概率被操作系统捕获（具体数据见 5.2 节）. 如对剩余部分有覆盖要求，则需复制全部指令，完全拷贝数据段、堆栈段等程序信息. 而在程序实际执行过程中相当一部分不会被执行到，却会造成访存次数增多、执行时间增长、内存空间浪费等后果. 关于覆盖率与性能折中的详细分析见 5.2、5.3 节.

算逻单元

计算指令经复制、锁步比较，绝大多数会被覆盖. 但需要指出，有一类计算指令引入的比较指令无法覆盖，否则会引发循环锁步的情况.

寄存器文件

本文假设寄存器文件有 ECC 或校验和保护，在该部件触发的单比特故障都会被检出，即系统对寄存器文件的故障漏检率为 0. 因此，有理由在后续实验中省去对该部件的故障注入，直接将其故障漏检

率带入统计.

其它部件

其它部件,如处理器核外的各级缓存、内存以及这些部件间的关键数据通路、胶合逻辑,它们的物理实现一般配有不同的校验机制. 虽然无法保证这些部件的原发性故障都被检出,但是对这些部件的瞬时故障易感性与性能、功耗的折中具有显著、积极的作用[20]. 由于处理器外的部件故障不属本文的研究范围,因此,本文假设核外的各级缓存、内存与之间的胶合逻辑是无故障的. 至于从处理器流出的漏检指令或者数据,可归类到以上几个部件触发的故障类型中.

3.4 估计方法

基于上述指令分类和详细的部件分析,下面进行理论估计.

我们将 A 类指令由于单比特翻转而转变为 B 类指令,定义为 A→B 类故障,其中 A 和 B 可能为同类指令. n 表示故障类型总数,m 表示故障部件总数,故障发生在部件 j 的事件用 U_j 表示,Y 表示故障漏检事件. SDC(Silent Data Corruption)表示故障未被硬件、操作系统和 EDDI 机制检测到,并且最终导致程序错误的故障比例. 则

$$P(SDC)=\sum_{j=1}^{m}P(SDC_j)=\sum_{j=1}^{m}P(SDC_j|U_j)P(U_j) \tag{1}$$

$$P(SDC_j|U_j)=\delta_j\times P(Y|U_j),\ \ \delta_j=\frac{N_{SDC_j}}{N_{SDC_j}+N_{CR_j}} \tag{2}$$

其中,N_{SDC_j} 为测试基准中结果为 SDC 的总数,N_{CR_j} 为测试基准中出现正确结果的总数.

解码部件故障漏检率估计

用 X_i 表示发生于解码部件的第 i 类故障事件,则

$$P(Y|U_j)=\sum_{i=1}^{n}P(Y|X_i)P(X_i),\ j \text{ is Decoder} \tag{3}$$

如果用 N 表示测试基准总指令数,C_i 表示由解码部件触发的第 i 类故障的指令集合,n_b 为指令位数,W_i 指第 i 类故障的故障比特所在的指令域宽度,则在解码部件发生第 i 类故障的概率为

$$P(X_i)=\frac{|C_i|}{N}\times\frac{W_i}{n_b} \tag{4}$$

$$P(Y|X_i)=P(O|X_i)P(Y|O|X_i) \tag{5}$$

其中,$P(O|X_i)$ 指第 i 类故障发生后操作系统检测不到的概率,则

$$P(O|X_i)=1-P(\bar{O}|X_i)=1-\{P(b)+P(s)+P(i)\} \tag{6}$$

式中,$P(b)$ 为引发总线错误的概率;$P(s)$ 为引发段错误的概率;$P(i)$ 为引发非法指令的概率.

$P(\bar{O}|X_i)$ 表现了系统软件纠检错的能力. 考虑其中的段越界(segmentation fault)和总线错误(bus error):段越界一般发生在指令/数据地址的高位,具体区域取决于应用程序编译后各个段大小(L_{seg});总线错误往往是由单比特故障引起的字不对齐引发,取决于指令/数据字长(L_{word}). 因此,可得

$$P(b)+P(s)=\frac{\log_2 L_{\text{seg}}-\log_2 L_{\text{word}}}{n_a}=\frac{\log_2 L_{\text{seg}}/L_{\text{word}}}{n_a} \tag{7}$$

其中,n_a 为地址域宽度.

对于 A→B 类故障,C_{ii} 表示合法的 A 指令单比特故障发生后没有对应的合法的 B 指令的集合,C_{ai} 表示合法的 A 指令的集合,得

$$P(i)=\frac{|C_{ii}|}{|C_{ai}|} \tag{8}$$

操作系统漏检指令会蔓延至应用层,这些指令都处于 C_i 集合内,其中部分指令一定不会被低代价锁步 EDDI 算法所覆盖,这些指令的集合称为 C_i';而在 C_i-C_i' 集合中,如果故障发生在特殊的指令比特,又不能被覆盖,这些特殊的指令比特总数称为 W_i'. $P(Y|O|X_i)$ 指第 i 类故障发生后操作系统检测不到的条件下,低代价锁步 EDDI 算法检测不到的概率,则

$$P(Y|O|X_i)=\alpha_1+\alpha_2\times\frac{|C_i'|}{|C_i|}+\alpha_3\times\frac{W_i'}{W_i} \tag{9}$$

其中,α_1、α_2 和 α_3 为 0 或者 1,取决于具体的故障类型.

对于 A→B 类故障,考虑下面一段代码:

```
1.  sethi %hi(.LL0),%g1   //未复制
2.  add %g1,%g0,%g2       //影子指令
3.  add %g1,%g0,%g1       //主指令
4.  cmp %g2,%g3           //比较
5.  bne error
```

1、3 为主指令,如果 A→B 类故障导致控制流出错,而跳转到指令 3、4 时,该故障可被算法检出;如果跳到指令 1、2、5 处,则检测不到. 此种情况下 W_i'/W_i 采用如下的估计方法:

令 N_{eddi} 表示经 EDDI 机制加固后的程序指令数,N_{copy} 表示未加固测试基准需要复制的指令数. 可以近似得到

$$\frac{W_i'}{W_i}\approx 1-\frac{2N_{\text{copy}}}{N_{\text{eddi}}} \tag{10}$$

综上，可得

$$P(SDC_j)=\delta_j \times P(Y|U_j)\times P(U_j),\ j \text{ is Decoder} \tag{11}$$

其中，$\delta_j=\frac{N_{SDC_j}}{N_{SDC_j}+N_{CR_j}}$.

地址生成部件故障漏检率估计

对于地址生成部件，估计方法如下：

$$P(Y|U_j)=\frac{W'}{W}\times P(O),\ j \text{ is AGEN} \tag{12}$$

其中，$P(O)$为地址生成部件发生故障后，操作系统漏检的概率，计算方法与式(5)相同.

综上，可得

$$P(SDC_j)=\delta_j \times P(Y|U_j)\times P(U_j),\ j \text{ is AGEN} \tag{13}$$

其中，$\delta_j=\frac{N_{SDC_j}}{N_{SDC_j}+N_{CR_j}}$.

算逻单元故障漏检率估计

算逻单元的估计方法与解码、地址生成部件稍有不同，部件的原发性故障无法被系统软件检出. 设未被覆盖的指令集合为 C_i'，C_i'为未处理的源程序中 cmp 指令集合，可得

$$P(Y|U_j)=\frac{|C_i'|}{N}\times P(O)=\frac{|C_i'|}{N},\ j \text{ is ALU/FPU} \tag{14}$$

$$P(SDC_j)=\delta_j \times P(Y|U_j)\times P(U_j),\ j \text{ is ALU/FPU} \tag{15}$$

其中，$\delta_j=\frac{N_{SDC_j}}{N_{SDC_j}+N_{CR_j}}$.

4　低代价锁步 EDDI 机制设计

本节给出了具体的指令复制原则，并以 SPARC 架构为目标平台提出了寄存器预留方法.

4.1　指令复制原则

低代价锁步 EDDI 影子指令生成原则如下：

(1) 计算指令(C 类指令)复制，即进入算逻单元的指令.

(2) 同步指令不复制. 同步指令包括：访存指令(M 类指令)，跳转指令(B 类指令).

(3) 主指令与影子指令按照严格锁步方式执行，在主指令与影子指令之间不能有其它指令，且在主指令与影子指令执行之后立即比较执行结果.

(4) 影子指令在主指令之前执行.

4.2　寄存器预留

基于上述的指令复制原则，本机制只需预留数目确定的少数几个寄存器，用于保存影子指令的执行结果和状态寄存器的内容. 本文以 SPARC 架构为目标平台，将寄存器预留方法说明如下：为整型应用预留两个全局寄存器，一个作为影子指令目的寄存器，一个用于保存整型状态寄存器(icc/xcc)的内容. 对于浮点运算，可能出现 128 位的结果，需要预留 4 个 32 位浮点寄存器. SPARC 有 4 个独立的浮点状态寄存器(fcc)，预留其中一个保存运算状态.

4.3　影子指令生成规则

本文针对整型和浮点运算分别提出了具体的影子指令生成规则，并成功解决了 SPARC V9 指令集中副作用影子指令的生成问题.

浮点运算

影子指令与主指令状态寄存器相互独立，使得浮点运算影子指令生成不涉及副作用问题，直接按指令复制原则操作即可.

摘自 MiBench 中 FFT 典型代码片段如下：

```
fdivd     %f10, %f8, %f28        //影子指令
fdivd     %f10, %f8, %f10        //主指令
fcmpd     %fcc3, %f10, %f28      //比较指令
fbne,pn   %fcc3, .error          //错误处理
```

整型运算

由于 SPARC 架构对于整型运算没有提供多份状态寄存器，因此我们单独预留了一个全局寄存器用于保存/恢复指令状态. 整型运算影子指令具体生成规则如下：

(1) 非复制指令

对于副作用指令，必须先保存状态，并在后续其它副作用指令执行之前恢复状态.

来自 spec2000 mcf 代码片段例子如下：

```
cmp    %o4, %i4              //主指令
rd     %ccr,%g3              //保存状态寄存器
st     %o2, [%l0+64]         //主指令
wr     %g3, %g0, %ccr        //恢复状态
movl   %icc, %i4, %o4        //主指令
```

(2) 复制指令

对于副作用指令，需将影子指令放在主指令前执行，在主指令后保存状态，并在后续使用 icc 的指令之前恢复 icc.

来自 spec2000 gzip 代码片段如下：

```
addcc      %o3, -1, %g2        //影子指令
addcc      %o3, -1, %o3        //主指令
rd         %ccr, %g3
cmp        %o3, %g2            //比较指令
bne, pn    %xcc, .error
nop
wr         %g3, %g0, %ccr      //恢复状态寄存器
```

bne,a,pt %icc, .LL53

综上,本文分别针对整型运算与浮点运算,提出了独特的指令复制规则,并成功解决了副作用指令生成问题.通过寄存器预留成功实现主指令与影子指令对寄存器的分割,寄存器预留比远小于全复制EDDI寄存器分半的50%.本机制通用性强,无需传参规则的修改,无需系统库的重新编译.

5 实 验

5.1 实验环境

故障注入系统

编译工具链选择GCC 4.2.1,通过对体系结构文件的重配置,实现编译器支持的寄存器预留、分配和后端优化.为避免编译器指令调度破坏主、影子指令间严格锁步,我们在编译、汇编之间插入指令复制规则的具体实现.故障注入通过对结构级全系统模拟器SAM[21]的修改实现,目标架构为UltraSPARC T2,指令集为SPARC V9.

评测环境

本文将低代价锁步EDDI实现代价与全复制EDDI进行比较[20],后者实现方法为:主指令、影子指令各占用一份寄存器;计算指令复制并在同步指令之前比较,同步指令包括load,store,跳转,函数调用,函数返回;跳转指令不复制;访存执行一次,若是取数指令需将取来的数据写入影子寄存器.

测试基准的选取

故障注入的目的是观察实验平台运行测试基准时,注入故障引发的种种表现.广泛使用的spec系列测试基准,主要用于性能测试,运行时间较长(从数分钟到数小时不等,在模拟器上更要高出几个数量级),而且应用个数较多(通常有20个左右).因此,研究人员通常采用运行指定代码片段、记录trace的方式进行试验.然而,这种方式与瞬时故障模型的随机性相悖[22],故本文没有采用spec系列测试基准.文献[23]将MiBench与SPEC进行了对比,MiBench的指令类型和吞吐量等方面不逊于SPEC,完全满足故障注入的要求.而且,MiBench输入集较小,运行时间大大缩短,有利于提高故障注入样本空间.因此,本文采用MiBench测试基准,应用包括FFT、stringsearch、dijkstra和basicmath,采用标准输入集.

5.2 可靠性评测

故障注入方式

在SAM模拟器进行故障注入,分别模拟处理器组合逻辑的代表性部件:解码单元、地址生成单元和算逻单元发生瞬时故障,蔓延至部件输入/输出锁存引发的单比特翻转.每种故障类型每个测试基准注入100次.

令T_{load}表示测试基准在SAM上运行时长,在$[0,T_{load}]$区间随机生成时刻T_{inject}.

地址生成单元故障

SPARC处理器除了包含程序计数器PC外,还具有NPC(Next Program Counter).此寄存器包含下一条要执行指令的地址.本文对NPC寄存器注入单比特故障来模拟地址生成单元故障.

解码单元故障

对SAM指令缓存中即将进入执行段的指令注入单比特故障,模拟解码单元瞬时故障经蔓延引发的微结构级单比特翻转.因为指令缓存中的故障指令可能被后续过程中执行到,所以在故障指令执行结束后需恢复.

算逻单元故障

算逻单元的故障模拟与上述两个部件不同.计算指令进入算逻单元后,流经部件均为故障蔓延的潜在路径.如图2所示,可能的故障源有5处,从寄存器文件读取操作数时,经①数据通路读入②输入锁存,经③组合逻辑运算生成结果,通过④输出锁存和⑤数据通路被写回寄存器文件.

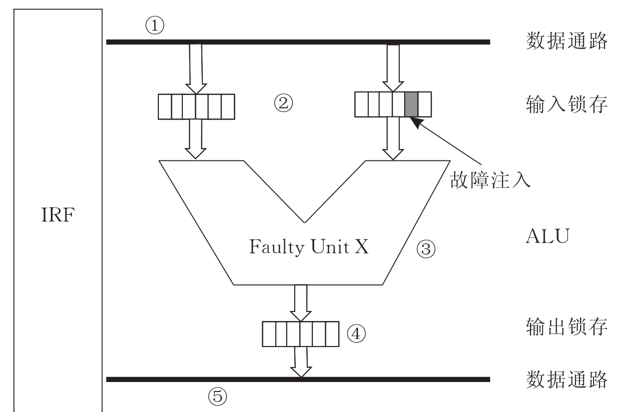

图2 算逻单元逻辑图

据此可将故障归为3类:(1)操作数在数据通路引发翻转和输入锁存受辐射翻转,最终都会导致输入锁存单比特翻转.(2)运算过程中组合逻辑故障,与输入锁存单比特翻转导致的故障表现类似.(3)同理,输出锁存故障和写回过程中数据通路故障,三者也可用输入锁存单翻转模拟.

由上述分析可知,本文在SAM执行过程中随机选择算术指令,并在此算术指令读取源操作数寄

存器时注入单比特故障.

故障漏检率估计

表2给出了解码单元(Decoder)、算逻单元(ALU)和地址生成单元(AGEN)故障漏检率的估计和实际值的对比. 其中，实际正确结果项为故障注入后，未经硬件、系统软件以及EDDI检出，而且程序结果仍然正确的比例. 本文假设寄存器文件有硬件保护，该部件故障漏检率为0. 因此，对4个部件故障漏检率依故障发生概率加权求和，可得低代价锁步机制对处理器(system)的总体故障漏检率(本文假设各部件单比特故障触发概率相等). 总体故障漏检率估计值为3.1%，实际漏检率为2.8%. 估计值与实际值拟合良好. 估计值偏高的主要原因在于间接寻址方式导致操作系统检错能力被低估，从而最终使总体故障漏检率估计值偏高. 但对于不同的部件其估计值表现不同，具体如下：

在SPARC体系结构中，解码部件的估计值略低于实际值，原因是测试基准中某些运算指令实现的特殊性，使得故障注入到这些指令后被屏蔽，例如mov指令(该指令将源寄存器值赋给目的寄存器)的一个源操作数为%g0(SPARC架构g0值恒为0)，若故障导致寄存器号改变为g1，恰好g1的值还是0，则不影响执行结果. 此种情况在故障注入试验中发生频率较高.

表2　故障漏检率估计

漏检率/%测试	漏检率/%											
	Decoder			ALU			AGEN			System		
	估计SDC	实际SDC	实际正确结果	估计SDC	实际SDC	实际正确结果	估计SDC	实际SDC	实际正确结果	估计SDC	实际SDC	实际正确结果
fft	5.1	5.3	39.1	0.1	0.9	22.4	14.5	12.0	7.3	4.9	4.6	17.2
dijkstra	0.3	0.4	54.7	0.1	0	29.2	3.1	2.1	12.8	0.9	0.6	24.2
stringsearch	0.8	0.9	54.0	0.1	0.5	31.5	2.3	2.3	18.2	0.8	0.9	25.9
basicmath	10.3	9.7	37.1	0.2	0.9	4.1	12.8	8.7	9.3	5.8	4.8	12.6
average	4.1	4.1	46.2	0.1	0.6	21.8	8.2	6.3	11.9	3.1	2.8	20.0

对于算逻单元，估计值低于实际值，主要原因是故障注入到运算指令操作数寄存器后可能被逻辑屏蔽. 例如：

```
and  %g1, 1, %g2  //影子指令
and  %g1, 1, %g1  //主指令
```

and指令将寄存器g1的值和立即数1逻辑与，并将结果赋给目的寄存器. 若故障发生在g1，仅当g1末位发生单比特翻转，才会影响执行结果，其它情况均导致逻辑屏蔽.

对于地址生成部件，估计漏检率高于实际值的主要原因是：测试基准中的部分访存指令，访问地址由计算得出(间接寻址等)，致使理论估计无法精确计算操作系统检错能力，导致估计的漏检率偏高.

故障注入结果与分析

测试基准在SAM注入故障之后，将故障实例的输出与标准输出进行对比. 根据故障表现的不同，可将注入结果分为以下5类：

(1)正确结果(Correct result)：运行结果的输出和标准输出一致.

(2)EDDI检测(EDDI)：故障被EDDI机制检测到.

(3)操作系统检测：包括段错误(Segmentation Fault)、非法指令(Illegal Instruction)、总线错误(Bus Error).

(4)操作系统崩溃(OS Panic、High OS)：故障导致操作系统重新启动.

(5)错误结果：运行结果和无故障结果不一致.

其中操作系统崩溃和错误结果之和为漏检比例，称为SDC(Silent Data Corruption).

图3显示原测试基准、全复制EDDI加固、低代价锁步EDDI加固后分别在解码单元、算逻单元和地址生成单元注入故障的结果. 与未加固测试基准相比，EDDI机制带来的容错性能提升十分明显操作系统的故障覆盖能力也相当可观. 全复制EDDI整体SDC为2.0%，低代价锁步EDDI为2.8%.

对于解码部件故障，未加固测试基准正确结果为60.2%，操作系统检测比例为24.9%，SDC为14.9%. 全复制EDDI加固后SDC平均值为1.9%，经本文锁步EDDI加固后SDC平均值为4.1%. 而锁步EDDI的SDC高于全复制EDDI的原因主要是前者没有对测试基准中cmp指令进行加固. 在SPARC体系结构中，cmp指令是由目的操作数为g0的subcc指令实现，如果注入的故障修改了目的操作数域，会导致g0以外的寄存器误写入. 若该寄存器影响程序数据流，就会引发错误结果. 但此种情况对整体检错性能贡献并不明显，而全复制EDDI为覆

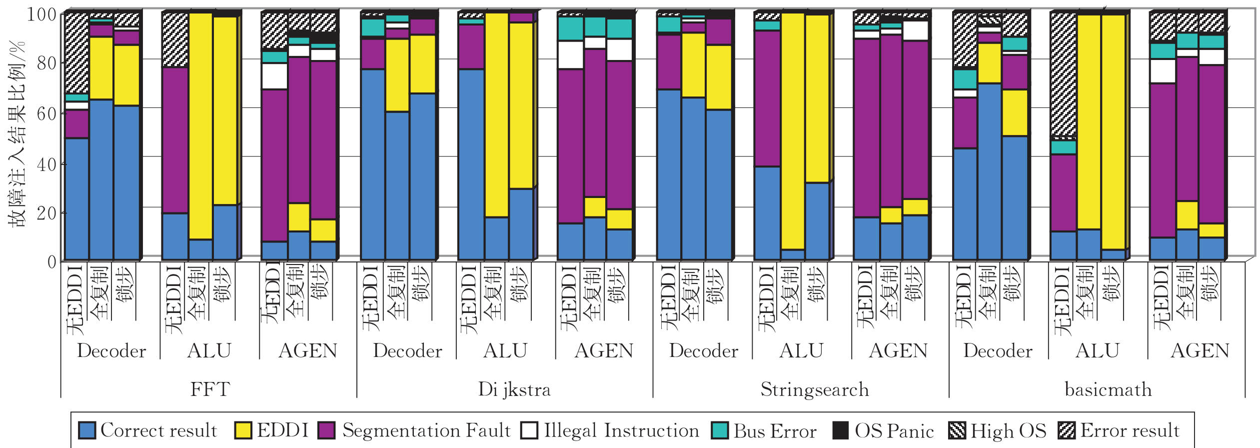

图 3　故障注入结果

盖 cmp 指令,也付出了寄存器分半的代价.由此带来的性能下降十分明显,具体分析见 5.3 节性能评测.

对于地址生成部件,没有 EDDI 加固的测试基准结果 SDC 平均值为 8.6%.全复制 EDDI 加固后 SDC 平均值为 6.1%.经本文低代价锁步 EDDI 加固后 SDC 平均值为 6.3%.其中略高于全复制 EDDI,主要原因是本机制复制指令原则是影子指令和主指令源操作数都来自同一个寄存器.

而对于算逻单元,经本文低代价锁步 EDDI 加固后,SDC 平均值从无 EDDI 加固的 19.6%降低为 0.6%.全复制 EDDI 为 0.1%.

5.3　性能评测

本节从静态指令数、动态指令数和执行时间 3 个方面对比了低代价锁步 EDDI 与全复制 EDDI 机制的性能表现.所列数据均为与未加固测试基准规格化的结果,如图 4 所示.

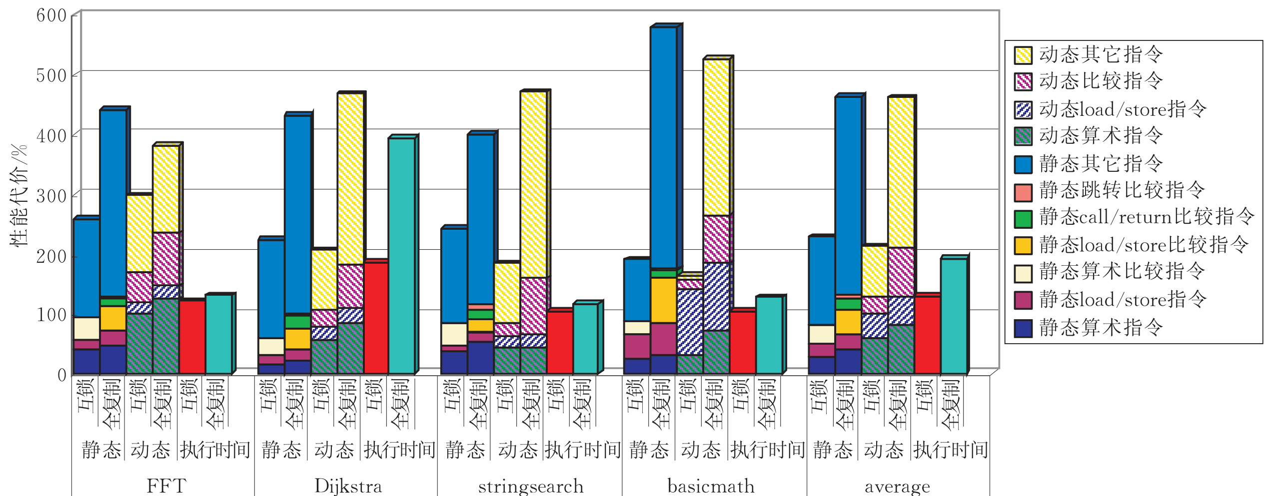

图 4　性能代价

静态指令代价

全复制 EDDI 静态指令总数平均值是原测试基准指令总数的 4.63 倍,而低代价锁步 EDDI 则是原测试基准指令总数的 2.29 倍.主要原因是:(1)比较原则.全复制 EDDI 在同步指令之前比较值和地址,store 带来至少两次的比较(值、地址)和对标志寄存器 ccr 的保存和恢复,函数调用会带来和参数个数相同数量的比较,跳转指令之前还需比较标志寄存器.而低代价锁步 EDDI 只比较运算指令结果,对于标志寄存器也只是在修改时保存,使用时恢复,此原则在寄存器有保护的情况下,不会带来容错性能的下降.与低代价锁步 EDDI 相比,全复制 EDDI 的比较指令高出 121.1%.(2)寄存器压力.全复制 EDDI 主指令、影子指令各占用一份寄存器,使程序中可分配的通用寄存器变少,这样就增加了程序中访存指令和运算指令的比例.与低代价锁步 EDDI 相比,全复制 EDDI 访存指令比例平均增加 33.8%,运算指令数平均增加 41.3%.其中 FFT 的访存指令增加比例达到 69.1%,可知低代价锁步 EDDI 对于诸如 FFT 这类寄存器负载较高的应用,

带来的性能提升尤其明显. 需要指出的是，访存指令的增多会进一步提高全复制 EDDI 的比较代价.

动态性能代价

全复制 EDDI 动态执行指令数是原测试基准的 2.37 倍，执行时间是原测试基准的 1.92 倍. 与低代价锁步 EDDI 相比，全复制 EDDI 平均执行的比较指令增加 253.5%，运算指令增加 50.1%，访存指令增加 8.7%，其中 FFT 的访存指令增加比例达到 18.3%.

总之，由于本文低代价锁步 EDDI 独特的复制原则和在编译前端实现的寄存器预留两个，减少了寄存器预留数、有效缓解寄存器压力，降低了访存代价，同时也降低了插入指令总数. 与全复制 EDDI 相比，低代价锁步 EDDI 在性能上提高显著：平均执行时间缩短 35.2%，平均动态执行指令则减少 36.1%. 这充分证明了本文代价互锁 EDDI 在性能上的优势.

6 结　论

本文在设计初期将硬件和系统软件的故障纠检错能力纳入考虑，针对处理器内瞬时故障，设计应用级“低代价锁步 EDDI”. 主要内容如下：

(1) 设计了低代价锁步 EDDI 机制，提出低代价锁步的指令复制规则和寄存器分配方式. 本机制结合硬件纠检错能力，兼顾处理器内组合逻辑、时序逻辑两类部件. 与传统 EDDI 将寄存器分半不同，通过编译链前端寄存器分配大幅减少了寄存器预留数，有效缓解寄存器压力，降低了访存代价，减少了性能损失. 无需修改编译器传参规则，无需重新编译系统库，提高了通用性.

(2) 基于概率论提出故障漏检率量化估计方法，为纠检错与性能折中进行指导. 通过对处理器中代表性部件：算逻单元、解码单元和地址生成部件进行故障注入，验证理论估计方法的有效性.

(3) 选择单比特瞬时故障模型，对机制故障注入实验结果、静态性能代价，与动态性能代价进行详细评测.

与全复制 EDDI 相比，低代价锁步 EDDI 仅以 SDC 平均升高 0.8% 的代价，换取了平均执行时间缩短 35.2%，平均动态执行指令则减少 36.1% 的性能优势.

参　考　文　献

[1] Gil P J et al. Fault representativeness. Deliverable ETIE2 of Dependability Benchmarking Project, IST- 2000-25245, 2002

[2] Shivakumar P, Kistler M, Keckler S, Burger D, Alvisi L. Modeling the effect of technology trends on the soft error rate of combinational logic//Proceedings of the DSTN. Washington DC, USA, 2002: 389-398

[3] Baumann R. Soft errors in commercial semiconductor technology: Overview and scaling trends//Proceedings of the International Reliability Physics Symposium. Dallas, Texas, USA, 2002: 121 01.1-121 01.14

[4] Zielger J F, Puchner H. SER—History, Trends and Challenges. Cypress Semiconductor Corporation, 2004

[5] McEvoy D. The architecture of tandem's nonstop system//Proceedings of the ACM'81 Conference. New York, USA, 1981: 245

[6] Slegel T, Averill R, Check M et al. IBM's S/390 G5 microprocessor design. IEEE Micro, 1999, 19(2): 12-23

[7] Oh N, Shirvani P P, McCluskey E J. Error detection by duplicated instruction in super-scalar processors. IEEE Transactions on Reliability, 2002, 51: 63-75

[8] Oh Nahmsuk. Software implemented hardware fault Tolerance[Ph. D. dissertation]. Stanford University, California, USA, 2000

[9] Shirvani P P, Saxena N, Oh N, Mitra S, Yu Shu-Yi, Huang Wei-Je, Fernandez-Gomez S, Touba N A, McCluskey E J. Fault-Tolerance Projects at Stanford CRC//Proceedings of the Military and Aerospace Applications of Programmable Devices and Technologies. Laurel, MD, 1999.

[10] Shirvani P P, Saxena N, McCluskey E J. Software implemented EDAC protection against SEUs. IEEE Transactions on Reliability, 2000, 49(3): 273-284

[11] Oh N, Shirvani P P, McCluskey E J. Control-flow checking by software signatures. IEEE Transactions on Reliability, 2002, 51(2): 111-122

[12] CRC-TR 00-5. Error detection by duplicated instructions. April 2000

[13] Oh N, Mitra S, McCluskey E J. ED^4I: Error detection by diverse data and duplicated instructions. IEEE Transactions on Computers, 2002, 51(2): 180-199

[14] Reis G A, Chang J, Vachharajani N, Rangan R et al. SWIFT: Software implemented fault tolerance//Proceedings of the International Symposium on Code Generation and Optimization (CGO). San Jose, CA, USA, 2005: 243-254

[15] Nicolescu B, Savaria Y, Velazco R. Software detection mechanisms providing full coverage against single bit-flip faults. IEEE Transactions on Nuclear Science, 2004, 51(6): 3510-3518

[16] Li Jian-Ming, Tan Qing-Ping, Xu Jian-Jun, Jiang Cheng. Control flow detection based on path tracking. Computer Engineering, 2009, 35(20): 68-70

[17] Wu Yanxia, Gu Guochang, Huang Shaobin, Ni Jun. Control flow checking algorithm using soft-based intra-/inter-block assigned signature//Proceedings of the 2nd International Multi-Symposium on Computer and Computational Sciences. Iowa, USA, 2007: 412-415

[18] Li Jian-Ming. Software implemented control flow error detection for transient failures in on-board computers[M. S. dissertation]. National University of Defence Technology, Changsha, 2009(in Chinese)
(李剑明. 面向星载计算机瞬时故障的软件控制流错误检测技术[硕士学位论文]. 国防科技大学, 长沙, 2009)
[19] Gao Long. Software implemented hardware fault tolerance [Ph. D. dissertation]. National University of Defence Technology, Changsha, 2006(in Chinese)
(高珑. 面向硬件故障的软件容错[博士学位论文]. 国防科技大学, 长沙, 2006)
[20] Yu Jing, Garzarán María Jesús, Snir Marc. Esoftcheck: Removal of non-vital checks for fault tolerance//Proceedings of the International Symposium on Code Generation and Optimization (CGO). Seattle, Washington, USA, 2009: 35-46
[21] Sun Microsystems, Inc. UltraSPARC architecture 2007. Santa Clara, CA, USA: Draft D0. 9. 2, 2008
[22] George N J, Elks C R, Johnson B W, Lach J. Transient fault models and AVF estimation revisited//Proceedings of the International Conference on Dependable Systems and Networks (DSN). Chicago, IL, 2010: 477-486
[23] Guthaus, Ringenberg J, Ernst D, Austin T et al. MiBench: Afree, commercially representative embedded benchmark suite//Proceedings of the 4th IEEE Workshop Workload Characterization. Austin, TX, USA, 2001: 3-14

WANG Chao, born in 1982, Ph. D. candidate. His research interests include design for fault tolerance, architecture reliability/availability, VLSI testing and verification.

FU Zhong-Chuan, born in 1970, associate professor. His current research interests include fault-tolerance computing, computer system architecture, multi-core etc.

CHEN Hong-Song, born in 1977, associate professor. His research interests include high performance and low power NP, Ad hoc network security.

CUI Gang, born in 1950, professor, Ph. D. supervisor. His research interests include dependable computing, etc.

Background

As semiconductor technology scales into nanometer regime, a revival of interests about intermittent fault is impelled by the following driving forces such as shrinking geometries, smaller interconnect dimensions, lower power voltages and decreased noise margins, etc.

In this paper, a compilation supported fault detection approach, namely cost-effective lock-stepped EDDI was proposed aiming at the hard to protect ALU components in SPARC processor. It was evaluated in great details in terms of fault coverage, fault detection capability, etc. And then a novel fault coverage estimation method was put forward and validated through thorough fault injection champions. Experiment results validated its efficacy.

This paper is part of our work supported by the National Natural Science Foundation of China under Grant No. 90818016, Heilongjiang Provincial Natural Science Foundation of China under Grant No. F200822, and National Key Laboratory of Science and Technology on Avionics System Integration & Foundation of Avionics Science under Grant No. 20095577012.

A whole fault tolerance framework across different levels, such as RTL, firmware, OS, and application, was proposed against intermittent faults in SPARC processor. Detailed investigations were made in the following aspects, such as SOF (Sources Of Failures) of intermittent faults, Symptom based high level fault detection & diagnosis approaches, fault propagation investigation, hypervisor based firm-ware level fault tolerance and compilation supported application level fault detection, etc.