Peña-Fernández, M.、Serrano-Cases, A.、Lindoso, A.、García-Valderas, M.、Entrena, L.、Martínez-Álvarez, A. 和 Cuenca-Asensi, S. (2019)。双核锁步增强了冗余多线程支持和控制流错误检测。微电子可靠性，卷。 100-101、113447。

DOI： 10.1016/j.microrel.2019.113447

使用冗余多线程增强双核锁步支持和控制流错误检测

M.Peña-Fernándeza , A. Serrano-Casesb , A. Lindosoc*、 M. García-Valderasc , L. Entrenac , A. Martínez-Álvarezb , S.昆卡-Asensib

至Arquimea Ingenieria SLU., 莱加内斯, 马德里, 西班牙
b 西班牙阿利坎特大学计算机技术系c西班牙马德里卡洛斯三世大学电子技术系

抽象的

这项工作提出了一种新的双核锁步方法来增强微处理器的容错能力。所提出的技术基于通过外部硬件模块将基于软件的数据检查和基于跟踪的控制流检查相结合。硬件模块连接到跟踪接口,能够观察架构中所有处理器的执行情况。所提出的方法已用于双核商用处理器。实验结果表明,该技术具有较高的错误检测能力,错误覆盖率高达 99.63%。

一、简介

微处理器是数字化的支柱
电子系统。制造技术的进步和晶体管特征尺寸的减小使微处理器变得便宜并适用于各种各样的应用。同时,主要由电离粒子引起的软错误的敏感性在越来越多的情况下已成为人们关注的问题[1]。对于高可靠性应用,要求微处理器具有容错能力,即在发生故障时能够继续运行。过去,在航空航天等恶劣环境中工作的系统需要容错微处理器,但如今,即使在地面上,对它们的需求也越来越大。

保护微处理器免受软错误影响的技术可以分为软件技术和硬件技术。软件技术非常灵活,但它们本质上是有限的[2]。

需要修改微处理器的硬件技术通常不可行,因为微处理器的设计和制造是一个昂贵的过程,只有大批量生产才能负担得起。相比之下,双模块冗余 (DMR) 是具有高错误检测能力的有吸引力的解决方案。作为微处理器
今天都比较便宜,复制不贵。
事实上,即使对于低端设备,多核设备也变得非常普遍,最先进的 MCU 开始引入安全功能,这些功能与自动控制越来越相关,例如在自动驾驶汽车行业和航空航天领域[3]。

双核锁步 (DCLS) [4-7] 是一种 DMR 容错技术,可以利用多核设备的可用性。它由两个处理器同时运行同一组操作组成,每个周期同步它们的输出,并在出现差异时触发恢复程序。

此架构在白皮书 ISO 26262 中进行了描述,其中 DCLS 处理器也称为“ASIL-D MCU”。尽管引入了安全功能,但 ASIL-D MCU 并没有消除在软件和系统级别实施其他安全措施的需要。一些 MCU 和处理器已成功实现此功能,例如,飞思卡尔 MPC5643L [8]、ML310 上的 PPC405 锁步系统以及 ARM Cortex-M33、Cortex-R4、Cortex-R5 和 Cortex-R7 [3、9]。 ARM Cortex R5 已集成在多个平台中,例如 TI Hercules TMS570 微控制器和 Xilinx UltraScale MPSoC。然而,据报道,恢复过程呈现高

* 通讯作者。 alindoso@ing.uc3m.es
电话:+34 (091) 6245964;传真:+34 (091) 6249430

与 Triple Core Lock-Step [9] 相比,开销约为 x1000。

软件 DLCS 将处理分为多个步骤,从单个指令到一组功能。在每个步骤之后,比较每个处理器产生的计算结果。如果它们不匹配,则会触发回滚机制以将系统恢复到一致状态。比较两个处理器提供的计算结果是 DCLS 的关键方面。通常,仅检查输出数据的错误 [5-6]。但是,控制流错误可能会导致其中一个处理器失去同步并最终挂起或丢失。

控制流错误不容易检测,因为它们在计算数据中可能不会立即产生可观察到的影响。此外,在双核中,一个处理器作为主处理器,另一个作为从处理器是很常见的。在这种情况下,master 的挂起会导致整个系统崩溃。这个问题的一个可能的解决方案是使用超时看门狗监视器来检测异常长的计算时间[6]。但是,这种方法很弱,并且会导致很高的错误检测延迟。

此外,即使输出数据正确,控制流错误也可能产生潜在影响,这些影响在系统恢复后仍可能保留在系统中。

在这项工作中,我们提出了一种增强的 DCLS 方法,该方法使用两种互补机制:观察跟踪子系统提供的信息以监控执行控制流,以及基于多线程软件的方案来检测数据不一致并从中恢复。

当今大多数微处理器都提供了一个用于调试目的的跟踪子系统,它能够以无缝和非侵入性的方式报告微处理器控制流,而不会影响执行。在正常操作下,跟踪子系统可以被重用来监控处理器的控制流[10-11]。可以通过在线解码相应的程序跟踪并检查获得的信息来检测影响任何处理器中控制流的错误[12]。

现代处理器架构和操作系统 (OS) 通常支持不同线程和进程的并行执行。通过在同一处理器(SMT-同时多线程) [13]、单独的内核(CMP 芯片级多处理器) [14]或混合使用它们 [15] 上执行代码的多个副本,这些功能已在不同的方法中得到利用]。所有这些方法都依赖于复杂的软件堆栈,除了操作系统之外,还包括不同的支持库,以减少开发时间并简化复制线程/进程的管理[16-18]。但是,添加的每个软件层都会引入新的漏洞,从而降低应用程序的整体可靠性。

在我们的方法中,CMP 方案已被用于裸机应用程序(无操作系统) ,与传统解决方案相比,该方案减少了竞争条件的数量并降低了控制开销。冗余线程在不同的内核上执行,并最终检查它们的输出。在出现差异的情况下,线程被迫重新执行关键区域作为恢复机制。所提出的方法可以被认为是一种宽松的锁步执行,其中可以以不同的粒度应用保护,从整个应用程序到代码的一些关键区域。

因此,可以在检查点数量和恢复时产生的时间开销之间建立适当的折衷。

所提出的方法已在双核 ARM Cortex-A9 [19] 上实施和评估。

微处理器是 Zynq FPGA [20] 中的硬核。建议的跟踪监视器已在可编程逻辑中实现。所提出的技术已经通过 871837 个错误的注入活动进行了测试,导致 43769 个错误。实验结果表明,该方法具有出色的错误检测能力,检测到的错误百分比高达 99.63%。

本文组织如下:第 2 节描述了所提出的锁步方法,第 3 节介绍了实验结果,最后第 4 节总结了这项工作的结论。

2. 建议的锁步方法

2.1。建筑学

与提出新硬件结构以扩展 CMP 处理器架构的其他方法相反,我们的解决方案旨在直接应用于现代多核处理器。该架构使用冗余多线程支持进行数据错误检测,并结合跟踪监控进行控制流错误检测。

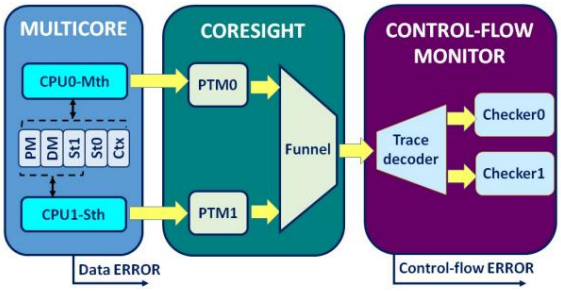

图 1. 双核 LockStep 架构

图 1 显示了所提出的 DCLS 方法的架构。它分为三个主要模块：多核微处理器（Multicore）、ARM Trace 子系统（Coresight)和控制流监视器。

控制流监视器是可以嵌入 FPGA 中的一小块硬件。数据错误检测在多核模块中实现，并在 2.2 小节中描述。其余块（Coresight 和控制流监视器)在 2.3 小节中描述。

2.2.数据错误检测

为数据错误检测和恢复开发了一种纯软件机制。它基于模块化冗余策略，依赖于冗余线程在不同内核上的并行执行。通过这种方式，它可以很容易地适应双或三模块冗余，具体取决于可用内核的数量 [21]。

类似于 Linux (POSIX Threads API) 或专用库 (OpenMP API) 等操作系统支持的传统线程，在我们的模型中，两个线程分别称为主线程 (MTh) 和影子线程 (STh)，共享指令（参见图 1 中的 PM) 和数据存储器（参见图 1 上的 DM）。然而，作为一个关键区别，每个线程都有自己的堆栈（即用于存储临时和自动变量以及函数输出变量的内存区域），它允许保存数据的副本。此外，还保留了第三个堆栈内存区域来存储上下文（图 1:Ctx）。由于 MTh 负责检查所考虑的每个区域的上下文和输出变量，因此该线程可以访问三个堆栈。

应用程序分为几个关键代码区域。每一个都由输入或上下文变量（即全局和局部变量)和输出变量来表征。为了完整起见，在这项工作中，上下文包含定义执行状态所需的所有信息，即区域的输入变量。作为一个

受限于我们方法的当前实现，我们无法保存处理器缓存的状态，因此我们假设它被禁用。作为这种选择的副作用，应用程序的编写方式应确保在区域内执行代码期间不修改输入变量，以便在恢复时保留这些变量的完整性。

关键区域由 C 或 C++ 中的注释原语分隔，并遵循 [22] 中建立的复制范围 (SoR) 的概念。为此，代码部分用于检查变量的正确性并使用屏障和互斥锁（互斥)同步执行。 SoR 定义了将发生线程复制和并行执行的代码区域。当指令流到达 SoR（临界区)时，由 MTh 执行上下文检查。如果没有差异，则上下文变量由 MTh 保存到上下文堆栈中。

每次指令流超出 SoR 边界时，MTh 都会自动对存储在两个堆栈上的数据执行一致性检查。如果发现任何差异，则执行恢复过程，即重新执行关键区域。

根据代码中关键区域的边界和情况，可以在不同的粒度级别应用保护，以在性能开销和从故障中恢复的延迟之间取得最佳平衡。

图 2 显示了矩阵乘法算法的带注释伪代码示例。

可以看出，SYNC 和 CHECK 注释包围了最里面的循环并定义了一个区域。
此外，该机制为每个线程提供同步，并分别自动检查上下文变量 i,j 和输出变量 acc。

该区域的本地化定义了具有 NxN 个检查点（步骤)的锁步执行，并且恢复延迟等于内部循环的执行时间（第 9-13 行)加上额外的分配（第 9 行）。通过在关键区域中仅包括内部循环体可以获得更精细的粒度。结果，恢复延迟减少到仅执行一行代码，代价是检查点的数量增加到 NxNxN。请注意，通过删除第 8 行和第 12 行（负责自动代码检测），我们获得了原始未受保护的代码。值得注意的是，SYNC 和 CHECK

函数可以与软件应用程序一起编译，因为它们是用 C 编写的。因此，为了保护

一个应用程序,用户只需要手动引入包含源代码中关键部分的两个函数。将来,计划开发一种工具来自动插入对这些功能的调用,以增强该解决方案的可扩展性。

算法1:

```
1:NT=2                          //线程数
2:A = 矩阵[N][N]
3:B = 矩阵[N][N]
4:C = 矩阵[N][N]
5:程序MxM
6:      对于 i = 0 到 N-1 做
7:          对于 j = 0 到 N-1 做
                //区域开始
8:              同步(i,j;NT)
9:              累积 = 0
10:             对于 k=0 到 N-1 做
11:                 acc += A[i][k] · B[k][j]
12:             检查(acc;NT)
                //区域结束
13:             C[i][j]=acc
```

图 2. 冗余线程矩阵乘法

除了代码注释之外,还实施了其他软件调整,以赋予双核系统在裸机上运行冗余线程的能力。首先,有必要修改板级支持包 (BSP) 以初始化平台并启动架构中提供的所有内核。通常,在裸机环境中,BSP 会提供最少的文件来启动平台。但是,BSP 只涵盖了引导系统的最常见方法的一小部分。默认启动顺序由 CPU0 控制,而另一个 CPU 进入无限忙等待循环。此默认初始化代码已更改为允许在 SPMD 模式（单程序多数据)下启动。其次,修改了内存映射和相关的链接器脚本,以支持每个内核的单独堆栈部分。最后,添加了自旋锁机制以允许内核同步。

2.3.控制流错误检测

通过连接到跟踪接口的外部硬件 IP（控制流监视器)观察两个内核的控制流。

所提出的方法是[12]中提出的技术的多核扩展。

所选器件中的双核 ARM Cortex-A9 包含一个基于 CoreSight 模块 [23]。在所选设备中可用的 CoreSight 模块中,我们的系统仅使用 PTM（程序跟踪宏单元)[24]。 PTM 是跟踪源 CoreSight 子类型。 PTM 单元与架构的单个核心相关联。因此,对于这项工作,使用了两个 PTM 实例 PTM0 和 PTM1,它们分别链接到核心 0 和核心 1（图 1）。两个跟踪源都经过多路复用并通过跟踪接口发送。外部 IP 对跟踪信息进行解码,并通过控制两者中执行指令的 PC 地址来检查执行流程的正确性

核心。

有两种技术用于检测不正确的执行:置信范围检查和地址看门狗检查。前者包括配置外部 IP 以将某些指令存储器地址范围视为有效。虽然内核的指令地址在其自己的置信范围内,但假定没有错误。后者也与核心指令地址有关,但这种情况下只配置了一个特定的地址进行周期性检查,即每一步的第一条指令。如果在可配置的时间内未收到所需的指令地址,则断言超时。如果在两个内核中的任何一个中检测到意外指令或超时,则会触发错误信号。 IP 以非常小的延迟在线工作。在执行之前不需要或存储其他信息,并且可以在编译时确定所需的配置寄存器值。外部 IP 可以通过软件配置为 AXI 外设。

为了简化外部 IP 的使用,所有用户定义的应用程序功能都针对程序员在链接描述文件中定义的特定内存区域,因此大多数代码都在相同的置信区间内。一些不能针对该区域的本机或库函数也已使用另外三个置信区间进行保护。代码主循环的第一条指令已被选为看门狗要检查的指令地址。由于应用程序使用 DCLS 进行了强化,因此外部 IP 已配置为使用完全相同的参数检查两个 CPU。

3. 实验结果

已执行故障注入活动以测试所提出的技术。仅在所选架构的两个核心之一中注入了故障。已选择第 M 个核心进行注入,因为考虑到它是最关键的

执行上下文和数据检查。

采用[12]中提出的技术,错误被注入到寄存器文件中。该技术在寄存器文件中随机生成位翻转。已使用外部控制器来确定、分类和收集观察到的和检测到的错误。在辐射环境中,错误也会影响所用技术未涵盖的存储器。暴露在辐射下的内存通常使用 EDAC 等冗余技术进行保护。在以前的辐射活动 [12] 中,我们已经验证了故障注入方法,即使我们只在寄存器文件中注入了故障,辐射和注入结果之间的错误检测匹配也非常准确。

控制流监视器位于 FPGA 中,辐射会影响其行为。 Xilinx SEM IP [25] 可用于保护 FPGA 及其包含的电路。

这些实验是在以 Xilinx Z7010 Zynq [20] 作为被测设备 (DUT) 为特色的商用 ZYBO 板上进行的。 DUT 中的两个 ARM Cortex-A9 内核的时钟频率均为 650MHz。在应用程序开始时,外部控制器生成一个随机种子,用于生成注入参数（时刻、寄存器编号和位索引)和程序数据的初始化值。

当 DUT 收到种子后,执行开始,注入器也开始执行。应用程序主循环的每五次迭代都会注入一个故障。在这五次迭代中没有出现错误的情况下,假设一个静默错误,并产生一个新的注入。如果发生错误,外部控制器会记录结果并重启 DUT 以开始新的注入。

故障注入活动的结果总结在表 1 中。使用两个版本的矩阵乘法基准完成了两个实验:未优化（-O0:表 1 的第 2 列)和以最大努力优化（-O3,表的第 3 列1）。两个基准测试都使用 32x32 32 位整数元素的矩阵。

对于 -O0 基准,注入了 591821 个故障,导致 20011 (3.38%) 错误。对于 -O3 基准测试,注入了 280016 个故障,导致 23758 (8.48%) 个错误。

表 1 中报告的错误类别是： · Det。挂起:故障已在系统中产生功能中断,已被外部 IP 检测到。 · 挂起:故障在系统中产生了功能中断,尚未检测到。

· 检测。仅 IP:外部 IP 已报告控制流错误,但未产生功能中断。 · 检测。数据:故障产生了软件检测到的数据错误。 · SDC:静默数据损坏,故障产生了一个未被检测到的数据错误。

· 通信:DUT 和外部控制器之间的通信故障,导致无法对错误进行分类。

· 总计:错误总数

表格1
注射活动结果

错误类型	-O0		-O3	
	# 错误	% 错误	# 错误	% 错误
这。恒行	15,462	77.27%	15,879	66.84%
挂起	23	0.11%	64	0.27%
来。仅有的。	75	0.37%	135	0.57%
这。数据	4,338	21.68%	6,725	28.31%
SDC	50	0.25%	940	3.96%
通讯	63	0.31%	15	0.06%
全部的	20,011	100.00%	23,758	100.00%

表 1 显示了所提出方法的高错误检测能力,未优化版本检测到 99.63% 的错误,优化版本检测到 95.77%。对于这个指标,我们没有考虑 Comm 错误,因为无法对它们进行分类。

关于控制流错误,值得注意的是,它们大多与系统功能中断有关,主要是由异常或锁步同步丢失引起的。然而,其中很少有人（Det. Only IP)没有产生这种效果。

这些可能与不会导致系统挂起的控制流错误相关联,例如,导致分支到错误但有效的代码区域的错误或循环索引中的错误导致意外更长的执行时间.尽管这些错误可能是误报,但出于预防原因,强烈建议将它们视为真正的错误。代码优化会略微降低控制流错误检测率。

在数据错误方面,存在一小部分 SDC。这些错误是由软件检查后注入的故障引起的。请注意,故障注入是不间断的,因此数据可能随时损坏,因此可能会出现错误

用于对结果进行分类的测试的最终检查。
然而,如果最终检查也是同步进行的,则可以检测到此类错误。无论如何,SDC 错误只占很小的一部分,尤其是在执行未优化的代码时。当引入优化时,这种效果会增加,因为编译器会更改某些操作的执行顺序以实现更高的吞吐量。

此外,随着优化代码变得更短,在软件检查后注入错误的可能性会更高。值得注意的是,在产生未优化版本的同一代码上引入了优化,这意味着没有进一步努力增强优化版本的错误检测,因此还有改进的空间。即便如此,数据检测惩罚是有限的,并且对于某些应用程序来说是可以承受的。

关于错误率,优化后的代码对错误的敏感性更高:8.48% 的注入错误产生错误,而在未优化的版本中,只有 3.38% 的注入错误导致错误。此外,数据已证明在优化版本中更容易出错,因为总错误的 32.27% 是数据错误,而未优化代码的 21.93%。在优化版本的情况下,这两个效果与寄存器的更高使用率有关。考虑到这些结果,有趣的是更深入地了解寄存器如何与错误相关,在错误发生时提取注入器信息。图 3 比较了两种代码版本中的寄存器灵敏度分布。

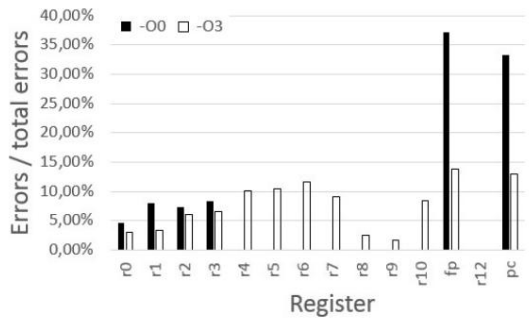

图 3. 寄存器对错误的敏感性

图 3 中的结果展示了在优化 (-O3) 代码的情况下寄存器的更广泛使用,因为除了 r12 之外的所有核心寄存器都会产生错误。
在未优化代码的情况下,仅使用 r0 到 r3 四个通用寄存器,因此注入到 r4 到 r10 上的故障对错误没有影响。帧指针 (fp) 和程序计数器 (pc) 与执行控制流密切相关,并且具有最高的错误两种情况下的费率。

4。结论

这项工作提出了一种双核锁步方法,增强了冗余多线程支持和控制流错误检测。数据错误检测和恢复基于独立内核上冗余线程的并行执行和检查数据正确性并同步执行检查的纯软件技术。

控制流保护是通过一个外部硬件 IP 来实现的,该 IP 以非侵入性的方式以小延迟监控两个内核的执行跟踪。

实验结果表明,控制流错误的可能性很大,因此需要数据和控制流检查来进行有效的错误检测。所提出的方法以低延迟实现了高错误检测率(高达 99.63% 的错误覆盖率)。

致谢

这项工作得到了西班牙经济和竞争力部项目 ESP2015-68245-C4-1-P、ESP2015-68245- 的部分支持

C4-3-P 和马德里社区在授予 IND2017/TIC-7776 下。

参考

[1] RC Baumann,先进半导体技术中辐射引起的软错误,IEEE Trans。关于设备和材料相关,第一卷。 5,没有。第 305-316 页,2005 年。

[2] JR Azambuja、S. Pagliarini、L. Rosa 和 FL Kastensmidt,探索 SEE 检测覆盖范围内仅软件技术的局限性,电子测试杂志,第 1 期。 27,(2011 年),第 541-550 页。

[3] X. Iturbe、B. Venu 和 E. Ozer,“实时安全相关 ARM Cortex-R5 CPU 的软错误漏洞评估”,2016 年 IEEE VLSI 和纳米技术系统中的缺陷和容错国际研讨会(DFT),康涅狄格州斯托斯,2016 年,第 91-96 页。

[4] NS Bowen 和 DK Pradham,基于处理器和内存的检查点和回滚恢复,计算机,第一卷。 26,没有。 2,第 22-31 页,1993 年 2 月。

[5] AB de Oliveira 等人,Lockstep 双核 ARM A9:重离子诱导软错误下的实现和弹性分析,IEEE Transactions on Nuclear Science,vol。 65,没有。 8,第 1783-1790 页,2018 年 8 月。

[6] F. Abate、L. Sterpone 和 M. Violante,一个新的

嵌入式处理器中软错误的缓解方法,IEEE Transactions on Nuclear Science,vol。 55,没有。 4,第 2063-2069 页,2008 年 8 月。

[7] M. Violante、C. Meinhardt、R. Reis 和 MS Reorda,在恶劣环境中部署处理器内核的低成本解决方案,IEEE Transactions on Industrial Electronics,vol。 58,没有。 7,第 2617-2626 页,2011 年 7 月。

[8] BERNON-ENJALBERT、瓦莱丽等人。支持 ASIL D 应用的安全集成硬件解决方案。 2013 年。

[9] X. Iturbe、B. Venu、E. Ozer 和 S. Das,“用于安全关键和超可靠应用的三核锁步 (TCLS) ARM® Cortex®-R5 处理器”,2016 年第 46 期IEEE/IFIP 国际可靠系统和网络会议研讨会 (DSN-W),图卢兹,2016 年,第 246-249 页。

[10] M. Portela-García 等人。关于在微处理器中使用嵌入式调试功能实现永久性和瞬态故障恢复。微处理器和微系统,36(5),第 334-343 页,2012 年。

[11] L. Entrena、A. Lindoso、M. Portela-García、L. Parra、B. Du、M. Sonza Reorda、L. Sterpone,使用跟踪接口的软核处理器的容错技术,在“用于航空航天应用的 FPGA 和并行架构。软错误和容错设计”,Springer,2015 年。

[12] M. Peña-Fernandez, A. Lindoso, L. Entrena, M. Garcia-Valderas、S. Philippe、Y. Morilla、P. Martin Holgado。用于 ARM 微处理器的基于 PTM 的混合错误检测架构。

微电子可靠性,88,第 925-930 页,2018 年。

[13] TN Vijaykumar、I. Pomeranz 和 K. Cheng,“使用同时多线程的瞬态故障恢复”,Proc。 29 年。诠释。症状。

计算。 Archit., Anchorage, AK, USA, 2002, pp. 38–87.

[14] M. Gomaa、C. Scarbrough、TN Vijaykumar 和 I。Pomeranz,“芯片多处理器的瞬态故障恢复”,第 30 届计算机体系结构国际研讨会,2003 年。

Proceedings., San Diego, CA, USA, 2003, pp. 98-109。

[15] K. Chen、G. v. der Bruggen 和 J. Chen,“具有多任务和冗余多线程的多核系统的可靠性优化”,IEEE Transactions on Computers,第一卷。 67,没有。 4,第 484-497 页,2018 年 4 月。

[16] A. Shye、J. Blomstedt、T. Moseley、VJ Reddi 和 DA Connors,“Plr:多核架构瞬态容错的软件方法”,IEEE Transactions on Dependable and Secure Computing,vol。 6,没有。 2,第 135-148 页,2009 年 4 月。

[17] H. Mushtaq、Z. Al-Ars 和 K. Bertels,“多核平台上基于软件的高效容错方法”,Proc。设计,汽车。测试欧元。会议。

展示。 （日期）,法国格勒诺布尔,2013 年,第 921-926.

[18] GS Rodrigues, F. Rosa, AB de Oliveira, FL Kastensmidt、L. Ost 和 R. Reis,“分析在运行 linux 和并行化的软错误下,ARM 处理器中容错方法的影响

apis,”IEEE 核科学汇刊,第一卷。 64,没有。 8,第 2196-2203 页,2017 年 8 月。

[19] ARM,Cortex-A9 MPCore 技术参考手册,2011 年。

[20] “Zynq-7000 All Programmable SoC:技术参考手册”,Xilinx Inc.,技术参考。手册 UG585,2016 年 9 月。

[21] A. Serrano-Cases、F. Restrepo-Calle、S. Cuenca Asensi1 和 A. Martínez-Álvarez,“基于线程复制的多核处理器的软错误缓解”第 20 届 IEEE 拉丁美洲测试研讨会论文集,智利， 2019 年 3 月

[22] SK Reinhardt 和 SS Mukherjee,“通过同时多线程进行瞬态故障检测”

第 27 届计算机体系结构国际研讨会论文集 (IEEE Cat. No.RS00201),温哥华,不列颠哥伦比亚省,加拿大,2000 年,第 25-36 页。

[23] “CoreSight 组件。技术参考手册”,ARM Ltd.,DDI0314H,2009 年。

[24] “CoreSight 程序流跟踪。架构规范”,ARM Ltd.,IHI 0035B，2011。

[25] “Soft Error Mitigation Controller v4.1”,产品指南,Xilinx Inc.，PG036,2014 年 11 月