

# 片上多核处理器容软错误执行模型

龚 锐 戴 葵 王志英

(国防科学技术大学计算机学院 长沙 410073)

**摘 要** 随着工艺的进步,微处理器将面临越来越严重的软错误威胁.文中提出了两种片上多核处理器容软错误执行模型:双核冗余执行模型 DCR 和三核冗余执行模型 TCR. DCR 在两个冗余的内核上以一定的时间间距运行两份相同的线程,store 指令只有在进行了结果比较以后才能提交.每个内核增加了硬件实现的现场保存与恢复机制,以实现对软错误的恢复.文中选择的现场保存点有利于隐藏现场保存带来的时间开销,并且采用了特殊的机制保证恢复执行和原始执行过程中 load 数据的一致性. TCR 执行模型通过在 3 个不同的内核上运行相同的线程实现对软错误的屏蔽.在检测到软错误以后,TCR 可以进行动态重构,屏蔽被软错误破坏的内核.实验结果表明,与传统的软错误恢复执行模型 CRTR 相比,DCR 和 TCR 对核间通信带宽的需求分别降低了 57.5% 和 54.2%.在检测到软错误的情况下,DCR 的恢复执行带来 5.2% 的性能开销,而 TCR 的重构带来的性能开销为 1.3%.错误注入实验表明,DCR 能够恢复 99.69% 的软错误,而 TCR 实现了对 SEU(Single Event Upset)型故障的全面屏蔽.

**关键词** 片上多核处理器;执行模型;软错误恢复;软错误屏蔽;双核冗余;三核冗余

**中图法分类号** TP302 **DOI 号:** 10.3724/SP.J.1016.2008.02047

## Chip Multiprocessor Execution Model for Soft Error Tolerance

GONG Rui DAI Kui WANG Zhi-Ying

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

**Abstract** With the development of integrated circuit, microprocessors are more and more susceptible to soft errors. Two chip multiprocessor execution models for soft error tolerance are proposed in this paper. Dual Core Redundancy (DCR) executes two redundant threads of a given program on separate cores with certain slack. The store instructions can not be committed until they are compared. The redundant cores are enhanced with hardware implemented context saving and recovery, so that the soft errors can be recovered by re-execution from the last context saving point. The context saving point chosen in this paper can efficiently hide the saving latency. The load coherence between original and re-executions is guaranteed by special mechanism to avoid undesirable fault. Triple Core Redundancy (TCR) applies triple modular redundancy on core level to exploit the core resources for soft error masking. Three redundant threads are executed in TCR on separate cores. Once detecting soft errors, TCR can be reconfigured to mask the wrong results of corrupted core. The experimental results demonstrate that, compared to traditional soft error recovery execution model CRTR, DCR and TCR can reduce 57.5% and 54.2% inter-core communication bandwidth demand respectively. The performance loss of DCR caused by re-execution is 5.2%, while reconfiguration on TCR brings 1.3% performance overheads. The fault in-

jection experiment shows that DCR can recover 99.69% soft errors, while TCR can mask all the SEU (Single Event Upset) faults.

**Keywords** chip multiprocessor; execution model; soft error recovery; soft error masking; dual core redundancy; triple core redundancy

1 引 言

集成电路受到宇宙射线或封装材料中的高能粒子辐射,将会引起局部充放电,可能产生瞬态故障(transient fault),进而导致微处理器功能错误.由于这些错误是瞬态且可恢复的,因而也被称为软错误(soft error).随着集成电路工艺的不断发 展,软错误将对微处理器的可靠性带来越来越严重的威胁.集成电路中单个器件发生软错误的概率与辐射剂量、电荷数目以及器件面积相关.随着供电电压的不断降低,器件中单个节点所存储的电荷数目也随之降低,这使得较低能量的粒子就可能引发软错误.但是随着特征尺寸的进一步减小,越来越小的器件面积使得单个器件受到粒子轰击的概率大大降低.这两种相反的趋势导致在相同的辐射剂量下,单个器件发生软错误的概率在未来的若干年中将保持不变,甚至略有下降<sup>[1-2]</sup>.但是由于单片集成电路能够集成的器件数目将呈指数增长,所以单个微处理器发生软错误的概率也将呈指数增长.因此,如何在越来越严重的软错误威胁下有效提高微处理器的可靠性成了亟待解决的问题. Intel 的研究报告表明,在未来几年内,必须采用有效的机制保证微处理器的可靠性,这些机制所消耗的晶体管数目将达单片能够集成的晶体管总数的 5%到 10%<sup>[3]</sup>.

在集成电路技术和体系结构技术的双重推动下,目前的微处理器已经全面进入片上多核处理器(Chip Multiprocessor, CMP)时代. CMP 中冗余的计算内核为提高微处理器的可靠性提供了新的解决途径. 目前典型的 CMP 软错误检测执行模型为片级冗余线程(Chip-level Redundant Threading, CRT)<sup>[4]</sup>. 典型的 CMP 软错误恢复执行模型为带恢复功能的片级冗余线程(CRT with Recovery, CRTR)<sup>[5]</sup>. 与 CRT 相比,CRTR 的性能开销较大,对核间通信通道的带宽要求也大大增加. 而高带宽的专用核间通信通道将带来额外的性能、面积和功耗开销,并增加微处理器的设计复杂度. 为了减少已有的多核冗余技术对核间通信通道的带宽需求,并且在合理的性能开销下实现对软错误的恢复与屏

蔽,本文提出了两种新的 CMP 容软错误执行模型,双核冗余(Dual Core Redundancy, DCR)模型与三核冗余(Triple Core Redundancy, TCR)模型.

DCR 在传统的软错误检测模型 CRT 的基础上,增加了硬件实现的现场保存与恢复. DCR 在不同的核上运行两份相同的线程,并在 store 指令提交前对其进行比较. 本文选择的现场保存点有利于隐藏现场保存带来的时间开销. 此外,本文还提出了特殊的机制以保证重新执行和原始执行过程中, load 数据的一致性. TCR 执行模型则将三模冗余应用到多核执行模型中,以实现对软错误的屏蔽. TCR 在 3 个不同的核上运行 3 份相同的线程,并且只对 store 指令进行比较. 当通过输出比较判断出哪个内核被软错误破坏后,TCR 可以将其隔离,并动态重构为 CRT 模型进行执行.

与传统的软错误恢复执行模型 CRTR 相比,本文提出的两种执行模型能有效降低对核间通信通道的带宽需求. DCR 在保持与软错误检测执行模型 CRT 相同的通信复杂度下,实现了软错误恢复. 而 TCR 执行模型通过使用更多的内核资源,在与 CRT 相同的通信复杂度和性能开销下,实现了对 SEU(Single Event Upset)型故障的有效屏蔽. 本文的实验结果表明,与 CRTR 相比,DCR 和 TCR 对核间通信带宽的需求分别降低了 57.5%和 54.2%. 没有发生软错误时,CRT、CRTR、DCR 和 TCR 相对于没有任何防护机制的基准处理器的平均性能损失分别为 30%,33%,32%和 30%. 在发生软错误的情况下,DCR 的恢复执行带来 5.2%的性能开销,而 TCR 的重构带来的性能开销为 1.3%. 错误注入实验表明,DCR 能够有效恢复 99.69%的软错误,而 TCR 能够对 SEU 进行全面的屏蔽.

2 软错误背景

高能粒子轰击集成电路,可能引发永久效应或瞬时效应<sup>[6]</sup>. 永久效应是由于高能粒子陷入集成电路内部硅氧界面引起的,一旦发生永久效应芯片就宣告报废,不能使用. 永久效应只有长期暴露于高辐射环境才有可能发生. 高能粒子引起的瞬时效应包

括单事件翻转 (Single Event Upset, SEU)、单事件瞬态 (Single Event Transient, SET) 和多位翻转 (Multi Bits Upset, MBU), 这些瞬时效应引起的错误也叫软错误。高能粒子轰击存储器件时, 可能引发 SEU, 单个存储器件的值将发生翻转, 错误的值将保持到下一个值写入。SEU 发生的频率很高, 最极端的情况下商用 MIPS R3000 微处理器在 500 公里地球近地轨道上大约每分钟发生一次 SEU<sup>[7]</sup>。高能粒子轰击组合逻辑电路, 将发生 SET, 产生一个宽 0.35ns 到 1.3ns 的毛刺<sup>[8]</sup>, 并且有可能沿组合逻辑通路传递。随着时钟频率的提高和两级寄存器之间组合逻辑深度的减少, SET 毛刺被采样到的概率大大增加<sup>[9]</sup>。在深亚微米工艺下, 对存储器的一次单个粒子轰击可能导致相邻的多个存储单元发生翻转, 即 MBU。由于 SEU 发生的概率远大于其他瞬时效应, 本文采用了 SEU 的故障模型。即假设在单个程序运行期间, 最多只有一位存储器或寄存器的值由于受到高能粒子轰击而被错误地改变。

### 3 相关工作

线程冗余是常用的容软错误技术之一。这种技术最早被用于高可靠的多处理器系统中, 通过两个处理器以锁步(lockstep)的方式运行相同的代码来实现对软错误的检测。在 IBM 的 Z900 高可靠服务器中, 两个冗余的 S/390 G5 处理器在每个周期内运行相同的指令, 通过比较指令的输出结果, 对可能发生的软错误进行检测<sup>[10]</sup>。

在同时多线程 (Simultaneous Multi Threading, SMT) 技术的支持下, 单个处理器就能运行多个线程, 利于线程冗余的实现。SMT 上的软错误检测执行模型包括 AR-SMT (Active-stream/Redundant-stream Simultaneous Multithreading)<sup>[11]</sup> 和 SRT (Simultaneously and Redundantly Threading)<sup>[12]</sup>。在这两种执行模型中, 单个 SMT 处理器就可以以松弛(slack)的方式运行两个冗余的可互相通信的线程。这两个线程被称为头线程和尾线程。与锁步执行不同, 松弛执行的头线程和尾线程在时间上保持一定的执行间距。头线程的 store 指令在提交前必须与尾线程的相应指令进行比较。需要通过专用的 LVQ (Load Value Queue) 和 StB (Store Buffer) 队列来实现头尾线程间的 load 数据复制和 store 数据比较。两个线程间还需要保持足够的时间间距, 以便尾线程能通过 BOQ (Branch Outcome Queue) 队列使用头线程的正确分支结果, 以避免尾线程中出现

分支预测失败。文献[12]还提出了复制范围 (Sphere Of Replication, SOR) 的概念。所有进入 SOR 的数据都必须进行复制, 以提供给冗余的线程使用; 所有离开 SOR 的数据都必须进行比较, 以检测软错误。AR-SMT 和 SRT 的 SOR 只包括流水线和寄存器文件, 而不包括 Cache, 所以这两种执行模型都只需对 store 指令进行比较。SRTR (SRT with Recovery)<sup>[13]</sup> 在 SRT 的基础上实现了软错误恢复。SRTR 的 SOR 不包括寄存器文件, 因此头线程的寄存器指令和 store 指令都必须在与尾线程进行比较后才能提交。而一旦通过比较发现错误, 则可以从该条错误指令开始恢复执行。SRTR 通过专用的 RVQ (Register Value Queue) 队列来实现头尾线程之间寄存器指令结果的传递。

随着技术的发展, 目前的主流处理器已经全面进入 CMP 时代, 在 CMP 的基础上实现线程冗余也逐步成为研究的热点。CMP 中一般包含多个 SMT 计算内核。为了实现负载均衡, 并防止单个软错误破坏同一 SMT 内核上的两个冗余线程, 基于 CMP 的容软错误执行模型并没有采用在单个 SMT 内核上运行多个冗余线程的方案, 而是在不同的 SMT 的内核上以松弛执行的方式运行相同的线程。文献[4]最早提出了基于 CMP 的软错误检测执行模型 CRT (Chip-level Redundant Threading)。CRT 的执行机制与 SRT 基本相同, 唯一不同的是 CRT 是在不同的内核上执行头尾线程。在 CRT 的基础上, 文献[5]提出了与 SRTR 类似的软错误恢复执行模型 CRTR (CRT with Recovery), 文献[14]更进一步发展, 提出了动态内核耦合 (dynamic core coupling) 的执行模型, 在核间通信通道动态可重构的基础上实现软错误恢复。在基于 SMT 的线程冗余中, 头尾线程间的通信是由核内的通信队列实现的。而在基于 CMP 的线程冗余中, 需要额外的核间专用队列实现冗余线程间的通信。这些核间通信队列由全局连线实现, 而全局布线必然带来严重的通信延迟和带宽限制, 进而影响整个冗余执行模型的性能、实现复杂度和可扩展性。由于只需要比较 store 指令, CRT 对核间通信的延迟和带宽需求都要小于 CRTR。而在 CRTR 中, RVQ 队列的延迟和带宽压力要远大于其他 3 个队列, 这是因为现代的多发射处理器在一个周期内提交的多条寄存器指令都必须进行比较而导致的。

### 4 研究思路

CMP 微处理器的可扩展性主要受限于核间互

连,复杂的核间互连不仅带来较大的面积、性能和功耗开销,还将影响微处理器的可扩展性与设计复杂度.目前基于 CMP 的软错误恢复执行模型在已有的内核互连的基础上,还需要增加额外的核间通信通道,且此核间通信通道对带宽的需求很大,这就严重限制了这种模型的可实现性.因此,本文着力解决基于 CMP 的容软错误执行模型中的核间带宽问题,并且希望能够在合理的性能开销下实现对软错误的恢复与屏蔽.

传统的解决核间带宽问题的方案,针对核间带宽需求最高的 RVQ 队列,采用特殊的技术减少进入 RVQ 队列的数据,或者对进入 RVQ 队列的数据进行压缩.本文采用了完全不同的解决思路,通过增加内核功能或者采用更多模的冗余,完全取消了 RVQ 队列.在目前基于 CMP 的容软错误执行模型中,对核间带宽需求最小的是软错误检测执行模型 CRT,其核间通信队列也是实现内核线程冗余所必需的最小配置.DCR 就是在保持此最小基本带宽需求的基础上,通过增加内核自身的复杂性,提供硬件实现的现场保存与恢复,在双核冗余的基础上实现软错误恢复.TCR 的思路与 DCR 不同,TCR 在保持最小基本带宽需求的基础上,利用更多模的线程冗余实现对 SEU 的屏蔽.业内的研究趋势表明,未来的多核处理器将发展到“many core”时代,单片内将集成几十甚至上百个内核<sup>[3]</sup>.在这种情况下,计算资源极其丰富,可以有效支持更多模的线程冗余.同时众核的情况下互连资源更加紧张,就更需要降低容软错误技术带来的核间互连需求.

目前,CMP 中的核间互连主要采用共享 Cache 或者片上网络(NOC)的方法实现.本文的研究工作并不改变已有的核间互连结构,所以提出的两种执行模型在这两种结构的 CMP 处理器上都可以实现.在接下来的具体描述中,本文采用的是基于共享 Cache 的结构.另外本文假设微处理器的指令集是标准 RISC 指令集,所有的存储器访问都通过 load、store 指令进行,而 IO 操作都通过 uncacheable 的 load 和 store 指令实现.指令都是乱序执行、顺序提交的.

5 DCR 执行模型

5.1 总体结构

DCR 执行模型在传统的软错误检测模型 CRT 的基础上增加了硬件实现的现场保存与恢复,以合

理的通信和性能开销实现了软错误恢复.在 DCR 执行模型中,一个执行节点的结构如图 1 所示.在采用 DCR 模型的 CMP 处理器中,可能包含一个或多个这样的执行节点,一个执行节点执行两份冗余的线程.DCR 执行模型的 SOR 包括冗余的流水线和寄存器文件.SOR 以外的 Cache 以及 SOR 以内的关键部件,如寄存器文件和关键队列,需要采用 EDAC<sup>[15]</sup>编码进行保护.DCR 节点中头尾线程保持一定的执行间距,在不同的内核上运行.同 CRT 一样,DCR 只需比较 store 指令,以防止错误的结果破坏 SOR 以外的模块.DCR 节点中的内核具有硬件现场保存与恢复功能.若检测到软错误,两个冗余的线程可以回退到上一现场保存点重新执行.

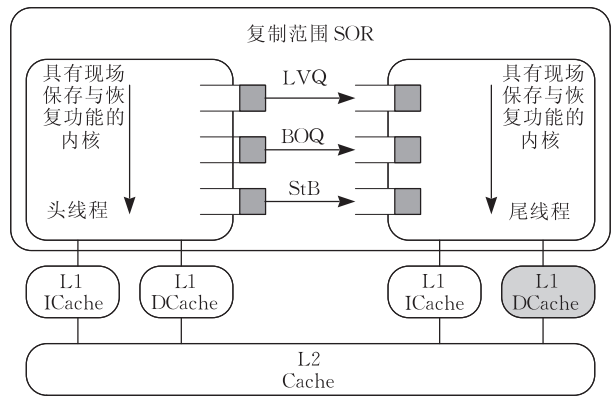


图 1 DCR 执行节点结构

5.2 输入复制与输出比较

对于进入 SOR 的数据,都必须进行输入复制,以保证两个冗余线程具有相同的输入;而对于离开 SOR 的数据,都必须进行输出比较,以防止软错误传递出 SOR.与 CRT 相同,DCR 节点中的输入复制与输出比较,是通过 3 个专用的核间通信队列实现的.

头尾线程都从自己的 L1 ICache 取指,假设程序是不可自修改的,且两个内核运行的是同一份程序的两个冗余线程,所以只要给出的取指地址相同,两个线程取到的指令也就相同.头线程通过 load 指令从自己的 L1 DCache 取数据,并将 load 指令的访存地址和数据放入 LVQ 队列.尾线程执行 load 指令时并不从自己的 L1 DCache 取数据,而是检测 LVQ 队列.如果队列头的访存地址与本条指令的访存地址相同,则使用队列头的数据作为 load 指令的执行结果;否则,两条冗余的 load 指令访存地址不同,必然是发生了软错误.因此采用 LVQ 队列可以实现输入复制,并且防止由于尾线程 DCache 失效

带来的性能损失。头线程的 store 指令在提交阶段,将访存地址和数据通过 StB 队列提供给尾线程,然后阻塞在 ROB (Reorder Buffer)中。尾线程的 store 指令提交时并不将数据写入自己的存储系统,而是与 StB 队列头的访存地址和数据进行比较。如果相同,给头线程一个反馈信号,收到反馈信号后头线程阻塞在 ROB 中的 store 指令才能完成提交,将数据写入自己的存储系统;如果 store 的访存地址和数据有任何一个不同,则认为发生了软错误。由于对数据的 load 和 store 操作实际都由头线程来完成,所以 DCR 执行模型并未使用尾线程的 L1 DCache。头线程的分支指令执行结果也通过 BOQ 队列提供给尾线程,作为分支预测的目标地址。如果没有发生软错误,尾线程不会遇到分支预测失败的情况。所以采用 BOQ 队列可以提高尾线程的执行性能。而一旦尾线程遇到分支预测失败,则认为发生了软错误。

本文对上述机制的完备性进行了分析。进入 SOR 的数据包括:

(1) 指令。指令的复制依赖于取指地址,若取指地址相同,两个线程可以通过各自的 L1 ICache 取到相同的指令。

(2) load 数据。load 数据的复制依赖于访存地址,若 load 指令访存地址相同,通过 LVQ 队列可以实现数据的复制。

离开 SOR 的数据包括:

(1) 取指地址。若为分支目标地址,可以通过 BOQ 队列进行比较;否则,不进行比较。

(2) load 访存地址。通过 LVQ 队列进行比较。

(3) store 访存地址。通过 StB 队列进行比较。

(4) store 数据。通过 StB 队列进行比较。

可以看到,只有取指地址没有进行全面的比较。因此可能发生头尾两个线程执行轨迹不同的情况。在此情况下,被软错误破坏的线程如果继续执行,遇到 store、load 或者分支指令,将通过比较机制发现错误。因此,上述的输入复制与输出比较机制可以对软错误进行完备的检测,保证软错误不会传递出 SOR 之外。

因为不用对寄存器指令进行比较,DCR 的专用核间通信队列与 CRT 一样,只有 LVQ、StB 和 BOQ。寄存器指令出现的频率远高于 load、store 和分支指令,所以 RVQ 队列对延迟和带宽的要求要远高于上述 3 个队列。由于彻底取消了 RVQ 队列,与 CRTR 相比,DCR 在大大降低核间通信设计实现复杂度的基础上,实现了对软错误的恢复。

### 5.3 现场保存与恢复

DCR 在 CRT 的基础上实现了现场保存与恢复。由于 DCR 的执行机制保证不会将软错误传递出 SOR,所以只须对 SOR 以内的执行现场进行保存和恢复。这些执行现场包括寄存器文件、程序计数器 PC 以及状态寄存器。本文采用了不同的策略,对这些现场进行保存与恢复。

对于集中的寄存器文件,本文采用了影子寄存器文件的办法进行保存与恢复。影子寄存器文件与实际的寄存器文件大小相等,并另外增加一个独立的 dirty 寄存器,该寄存器的每一位表示影子寄存器文件中对应的一项数据是否被写入过。当 DCR 执行寄存器指令时,不经比较就可以提交,将结果写入影子寄存器文件,并置上 dirty 寄存器的相应位。因此影子寄存器文件包含了被修改的现场。在执行以寄存器为源操作数的指令时,读寄存器文件阶段同时读取影子寄存器文件与实际寄存器文件,若影子寄存器文件命中(对应 dirty 位已置上),则使用从影子寄存器文件读出的值;否则使用从实际寄存器文件读出的值。在现场保存点,根据 dirty 寄存器的内容,将影子寄存器文件中所有被修改过的数据一一写入实际寄存器文件,并将 dirty 寄存器全部清零。因此现场保存所需的时间依赖于影子寄存器文件中被修改的数据的项数。需要恢复现场时,只需花费一个时钟周期将 dirty 寄存器清零即可。

对于分散的寄存器,如 PC 和状态寄存器,本文采用备份寄存器的方法进行保存与恢复。将 PC 和状态寄存器进行冗余备份,分为运行组和备份组。复位时两组都复位为相同的值。处理器在运行组上运行,在现场保存点,将运行组的数据写入备份组。检测到软错误以后,根据备份组数据恢复运行组现场。PC 和状态寄存器的保存与恢复都只需要一个时钟周期。

头线程流水线的提交段,若 ROB 队列头为 store 指令,必须等待尾线程相应指令的执行结果以进行比较,然后此 store 指令才能提交。当比较无误,且该条 store 指令在写入存储系统时引起 Cache 失效,触发一次现场保存。Cache 失效的处理过程可以有效地隐藏现场保存的时间,因此这种现场保存点的选择有利于减少 DCR 执行模型的性能开销。需要注意的是,当 store 指令进入提交阶段时,微处理器的 PC 和状态寄存器可能已被后续指令修改。因此,本文在 ROB 中增加了两个字段,即 PC 和状态寄存器字段。为指令分配 ROB 时,将其对应的 PC

和状态寄存器的值写入 ROB 中相应字段. 这样, 当 store 指令提交引起 Cache 失效, 并触发现场保存时, 将 ROB 中的 PC 和状态寄存器字段的值写入相应的备份寄存器即可.

在 DCR 模型的执行过程中, 如果通过输出比较检测到不一致, 则认为发生了软错误. 此时冗余的两个内核只需要恢复已保存的现场, 并清空流水线, 就可以从上一现场的现场保存点开始重新执行, 以消除软错误带来的影响. CRTR 的恢复是从发生错误的那条指令开始重新执行, 而本文提出的 DCR 执行模型则是从上一现场保存点开始重新执行.

5.4 load 数据一致性

在 DCR 执行模型中, 检测到软错误后微处理器将恢复到上一现场保存点重新执行. 如果 load 指令被再次执行, 有可能影响程序的正确性. 以下面一段代码为例:

```
I1: load R1, [addr1];
I2: add R1, R1, 1;
I3: store R1, [addr1];
```

如果在 I3 指令后检测到软错误, 且上一现场保存点在 I1 指令前, 则 I1 指令会被重新执行, 并取到与原始执行不同的数据, 这将会影响程序的正确性.

为了解决这个问题, 本文提出了一种特殊的机制, 以保证重新执行与原始执行时, load 指令的数据一致性. 本文在两个冗余的内核中, 各增加一个 CLV (Commit Load Value) 队列, 以保存各自内核从上一现场保存点开始已提交的 load 指令的读入数据. 当恢复到上一现场保存点重新执行时, 如果执行到 load 指令, 首先判断 CLV 队列是否为空, 不为空则采用 CLV 队列头的数据作为 load 数据; 否则才从存储系统 load 数据 (头线程) 或从 LVQ 队列取数据 (尾线程). 当进行现场保存时, 将 CLV 队列中的数据清空.

CLV 队列可以有效地保证 load 数据的一致性. 但是需要注意的是, 如果两次 store 导致的 DCache 失效之间执行了多条 load 指令, 可能发生 CLV 溢出. 在此情况下, 也需要进行现场保存, 以便将 CLV 清空. 因此 DCR 中的现场保存除了被 store 指令引起的 Cache 失效触发, 还会被 CLV 溢出触发. Cache 失效引起的现场保存能够被有效地隐藏, 但是 CLV 溢出时进行现场保存, 将造成流水线的不必要停顿, 引起额外的性能开销.

5.5 软错误恢复能力

要能进行正确的恢复, 必须能够保存正确的现

场. 本文的现场保存点包括 store 引起的 Cache 失效和 CLV 溢出. 在现场保存点, 可以保证软错误没有传递出 SOR, 但是并不能保证 SOR 内部没有发生瞬态故障. 瞬态故障可能已经破坏了某条寄存器指令的执行, 产生了错误的执行结果并写入影子寄存器文件. 这种寄存器错误是不可检测的错误. 如果这个影子寄存器文件的值没有进一步引发可检测错误, 在下一现场保存点, 该错误的值会被当成正确的现场写入实际寄存器文件. 如果该错误的值在后续的执行中引发可被检测到的错误, 那么微处理器将恢复到错误的现场, 整个执行将陷入“检测到错误—恢复到错误的现场—检测到同一错误”的死循环. 这种软错误不能通过 DCR 恢复, 因此也叫不可恢复错误. 可以采用其他的机制, 如 watchdog<sup>[16]</sup>, 检测程序的执行轨迹, 发现死循环后通过复位来消除不可恢复错误.

为了对这种不可恢复错误进行定性的描述, 本文对相关的概念进行了定义.

**定义 1.** 指令基本数据. 与指令  $i$  的执行直接相关的不可分割的数据的集合称为指令  $i$  的基本数据  $D(i)$ .

$D(i)$  包含三类数据, 指令  $i$  源操作数的值、目的操作数的值以及指令  $i$  的寻址操作使用的不可分割的数据. 以下面的指令为例说明指令的基本数据:

```
load R1, [R2]
```

该 load 指令采用了寄存器间接寻址的方式, 其基本数据包括: (1) 源操作数的值, 即从存储器中读出的值; (2) 目的操作数的值, 即写入 R1 寄存器的值; (3) 寻址操作使用的不可分割的数据, 即 R2 寄存器的值.

**定义 2.** 消费数据与生产数据. 指令  $i$  的消费数据  $C(i) = \{x | x \in D(i), \text{且指令 } i \text{ 读取 } x \text{ 的值}\}$ . 指令  $i$  的生产数据  $P(i) = \{x | x \in D(i), \text{且 } x \text{ 的值由指令 } i \text{ 产生}\}$ .

**定义 3.** 软错误传递关系. 指令  $i$  到指令  $j$  的软错误传递关系  $P(i, j)$  为真, 当且仅当  $\exists x \exists y, x \in P(i), x$  被软错误破坏,  $x \in C(j), y \in P(j)$  且  $y$  被软错误破坏.

需要注意的是, 如果指令  $i$  的生产数据被软错误破坏且该数据被指令  $j$  消费, 并不一定导致指令  $j$  的生产数据被破坏. 比如被破坏的寄存器数据与全 0 做按位与操作, 并不影响该指令的结果. 此时我们称该条按位与指令可以屏蔽软错误.

**定义 4.** 软错误传递链. 指令  $i_1$  到指令  $i_n$  存在



软错误传递链  $Chain(i_1, i_n)$ , 当且仅当  $\exists i_x (x=2, 3, \dots, n-1)$ , 使得  $P(i_y, i_{y+1})$  为真,  $y=1, 2, \dots, n-1$ .

软错误通过软错误传递链破坏其他指令, 也称为软错误扩散. 不可恢复的软错误可以描述为, 在某次现场保存点后由于发生瞬态故障, 使得某条寄存器指令的生产数据发生软错误, 在下一现场保存点前, 该软错误未经过任何软错误传递链传递到任何 load、store 或分支指令. 因此在运行到下一现场保存点的过程中, 不会发生任何可检测错误, 错误的执行现场会被保存. 今后的执行过程中如果软错误继续扩散, 并被 DCR 的输出比较机制检测到, 处理器也只能恢复到错误的现场, 不能恢复程序的正确执行.

需要注意的是, 不可恢复错误发生的概率是比较低的. 如果微处理器的 Cache 足够大且程序具有良好的数据局部性, 将很少发生由 store 指令引起 Cache 失效的情况. 如果 CLV 也足够大, 两个现场保存点之间的时间距离是比较长的. 因此软错误有足够的时间扩散到 load、store 或分支指令, 错误的值就不会被当作正确的现场进行保存, 也就不会导致不可恢复错误. 由以上分析可知, 要减少不可恢复错误的发生, 就要使两个现场保存点之间具有合理的时间间距, 以便于软错误充分扩散. 第 7 节的实验结果表明, 在典型的 L1 DCache 大小和 CLV 长度

配置下, 不可恢复错误的概率仅为 0.31%. 传统的 CRTR 执行模型能够恢复所有检测到的软错误. 本文在 CRTR 的基础上, 有效地提高了性能、降低了核间通信复杂度, 但同时也损失了一定的软错误恢复能力.

## 6 TCR 执行模型

### 6.1 总体结构

TCR 执行模型将三模冗余应用到 CMP 中, 以较低的性能开销和通信开销实现了对 SEU 的屏蔽. 在 TCR 执行模型中, 一个执行节点的结构如图 2 所示. 在采用 TCR 执行模型的 CMP 中, 可能包含一个或多个这样的执行节点, 一个执行节点执行 3 份冗余的线程. 同 DCR 一样, TCR 执行模型的 SOR 包含冗余的流水线和寄存器文件. Cache、寄存器文件以及关键队列需要采用信息冗余, 如 EDAC<sup>[15]</sup> 编码的方式进行防护. TCR 节点中, 头中尾 3 个线程保持一定的执行间距, 在不同的内核上运行. TCR 只比较 store 指令, 以防止错误的结果破坏 SOR 以外的模块. 通过比较可以判断出是哪个线程被软错误破坏, 然后将表决出的正确结果提交, 并将 TCR 执行节点动态重构为传统的软错误检测执行模型 CRT, 以屏蔽错误的线程.

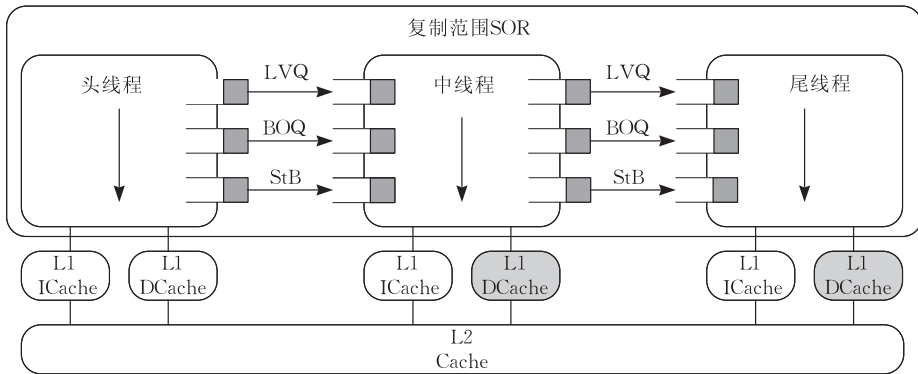


图 2 TCR 执行节点结构

### 6.2 输入复制与输出比较

TCR 将三模冗余应用于 CMP 的核间冗余. 其输入复制与输出比较由专用的核间通信队列 LVQ、StB 和 BOQ 实现.

一个执行节点内的三个冗余线程都分别从自己的 L1 ICache 取指. 与 DCR 一样, 头线程从自己的 L1 DCache 取数据, 并将 load 指令的访存地址和数据通过 LVQ 队列提供给中线程, 以实现输入复制.

中线程在执行 load 指令时与 LVQ 队列头的访存地址进行比较, 若不同, 则认为检测到软错误; 若相同, 则使用 LVQ 队列头的数据, 并将访存地址和数据再通过 LVQ 队列传递给尾线程. 尾线程执行与中线程相同的操作.

头线程在提交 store 指令时, 将 store 的访存地址和数据通过 StB 提供给中线程, 并将该条 store 指令阻塞在 ROB 中. 如果中线程对访存地址和数

据的比较没有发现不一致,头线程中的 store 指令可以立即提交,不用等待尾线程的执行结果,同时中线程再将 store 指令的访存地址与数据通过 StB 提供给尾线程.如果中线程的比较发现不一致,则认为头线程和中线程中有一个已被软错误破坏,此时中线程将 store 指令阻塞在自己的 ROB 中,头尾两个线程均停顿,等待尾线程的执行结果,以具体判断是哪个线程被破坏.由于实际的访存操作只由头线程执行,中线程和尾线程的 L1 DCache 并未使用.

同 DCR 一样,头线程的分支指令执行结果将通过 BOQ 提供给中线程和尾线程,作为分支预测的目标地址.在没有发生软错误时,中线程和尾线程不会出现分支预测失败的情况.

可以看出,TCR 的输入复制以及输出比较与 DCR 相同,只是中线程要将头线程传递过来的数据再通过专用核间队列提供给尾线程.这种输出比较机制对软错误检测的完备性在 5.2 节已经做了说明.需要注意的是,如果头线程和中线程的 store 指令比较结果一致,阻塞在头线程 ROB 中的 store 指令就可以立即提交,不用等待尾线程的执行,这有利于提高整个 TCR 执行模型的性能.由于 TCR 执行模型也不用对寄存器指令进行比较,所以完全取消了 CRTR 中对带宽需求最高的 RVQ 队列,大大简化了核间互连的设计实现难度.

### 6.3 重 构

如果微处理器发生了 SEU,TCR 执行模型可以通过比较 3 个冗余线程的执行结果确定是哪个线程被 SEU 破坏,并动态重构为传统的软错误检测执行模型 CRT,以屏蔽被 SEU 破坏的线程.

线程间的输出比较包括对 load 访存地址,store 访存地址和数据以及分支目标地址的比较.TCR 中中线程与尾线程间的数据队列增加了一位 verify 位,以标志相应的数据是否通过了中线程的比较.如果中线程在与头线程的任何比较中检测到软错误,则中线程立即阻塞,然后将头线程提供给中线程的数据通过相应核间通信队列提供给尾线程,并将该数据的 verify 位置为 unverified.尾线程执行与中线程相同的比较,如果也检测到错误,则证明是头线程被 SEU 破坏.头线程被破坏后的重构过程如图 3(a)所示.在此情况下,尾线程返回一个 unverified 信号给中线程,中线程接收到该信号后,向头线程发出 dump 命令,通知头线程停止执行并

将其 L1 DCache 中的数据全部写回 L2 Cache.头线程完成 L1 DCache 的写回后,给中线程回复一个 ack 信号,此时被阻塞的中线程可以开始重新执行,并从自己的 L1 DCache 取数据.余下的程序段由中线程和尾线程以 CRT 模型的机制进行执行.

中线程被破坏后的重构过程如图 3(b)所示.尾线程接收到中线程发来的 unverified 数据,进行相应的比较后没有发现不一致,则可以确定是中线程被 SEU 破坏.尾线程返回 verified 信号给中线程,此后中线程停止执行,并将头线程通过 LVQ,StB 和 BOQ 队列发来的数据一一转发给尾线程.余下的程序段由头线程和尾线程继续执行.

如果头线程和中线程都未被 SEU 破坏,则中线程的输出比较不会发生任何不一致的情况.中线程将相应的数据发给尾线程,并将其标志为 verified.如果尾线程发生 SEU,进行比较时将发现不一致.尾线程将向中线程发出 unverified 信号.TCR 执行节点将按照图 3(c)所示的过程进行重构.在余下的程序执行过程中,中线程将不再将任何数据传递至尾线程.

在没有发生任何故障的情况下,头线程的 store 指令在完成与中线程的比较后就能立即提交,而不用等待尾线程的执行.所以此时 TCR 的性能与软错误检测模型 CRT 相同.在检测到软错误以后,TCR 节点的重构过程将消耗一定的时钟周期.这其中,由于头线程被 SEU 破坏导致的重构消耗的时间最长.其性能开销包括头线程 L1 DCache 的写回,以及中线程刚启用自己的 L1 DCache 时发生的大量 Cache 失效.由中线程被破坏导致的性能开销次之,主要是由中线程被阻塞,以及中线程进行重构所引起的.而尾线程被破坏导致的重构,并不影响程序的正常执行,因此没有任何额外的性能开销.

本文假设的故障模型为 SEU,即一次程序的执行过程中最多有一位存储单元被错误的翻转.TCR 执行模型能够检测到由于 SEU 故障引发的软错误,并且在不进行任何程序回退的情况下,通过动态重构屏蔽被破坏的线程,保证程序的继续执行.因此 TCR 执行模型能够完全屏蔽 SEU 故障.在更为恶劣的环境中,一次程序的执行可能遭遇多次高能粒子轰击,导致多次故障.TCR 能够有效屏蔽第一次故障,然后重构为传统的软错误检测模型 CRT,以便对进一步发生的瞬态故障进行检测.





7.1 性能评估

本文首先对 DCR 和 TCR 执行模型在没有发生任何故障的正常执行状态下的性能进行了评估. 所使用的测试程序是 SPEC2000 标准测试程序集.

DCR 执行模型在正常执行时, 由于需要进行现场保存, 所以具有一定的性能开销. DCR 中现场保存由 store 导致的 Cache 失效和 CLV 溢出引起. Cache 失效引起的现场保存可以被失效处理过程有效隐藏, 所以不会有额外的性能开销. 而如果两次由 store 指令导致的 Cache 失效间出现 CLV 溢出, 将会导致流水线非正常停顿以进行现场保存, 这将增加整个程序的执行时间. 因此 DCR 正常执行时的性能与 CLV 溢出的次数相关, 也即依赖于 CLV 队列的长度. 图 4 表明了在不同的 CLV 队列长度配置下, DCR 执行模型在执行不同测试程序时的性能. 从图 4 中可知, DCR 的执行周期数随着 CLV 队列的增长不断降低, 但是降低趋势不断减缓. CLV 大到一定的程度, 将能够容纳绝大多数 Cache 失效间的 load 数据, 就可以有效地减少 CLV 队列溢出的情况. 在此基础上再增大 CLV 队列对性能的影响不大, 而且过长的 CLV 队列将导致严重的功耗和面积开销. 综合考虑, 本文采用的 CLV 队列的长度为 128.

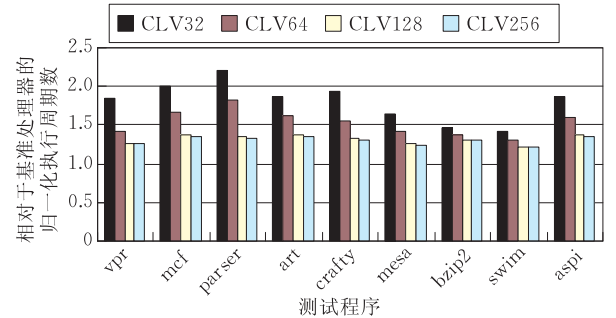


图 4 不同 CLV 队列长度配置下 DCR 执行模型的归一化性能

在 CLV 队列长度为 128 的情况下, 本文对 CRT、CRTR、DCR 和 TCR 4 种容软错误执行模型在正常执行时的性能进行了比较, 其结果如图 5 所示. 与无任何容错能力的基准处理器相比, CRT、CRTR、DCR 和 TCR 4 种执行模型的平均执行时间分别增加 30%、33%、32% 和 30%. 长度为 128 的 CLV 队列能够有效减少由 CLV 溢出导致的现场保存, 所以在此配置下, DCR 相对于 CRT 的额外性能开销只有 2%, 而传统的软错误恢复执行模型 CRTR 相对于 CRT 的性能开销为 3%. TCR 模型在正常执行时, 头线程的 store 指令在与中线程比

较无误后即可提交, 所以其性能与 CRT 相同. 需要注意的是, 本文为了获得不同执行模型对核间队列的带宽需求, 在实验中假设核间队列具有无限的带宽. 所以本文测得的性能只是最乐观的性能, 在带宽不能满足需求的情况下, 所有执行模型的性能都要差于图 5 中所示数据. 特别对于需要大量通信的 CRTR 模型来说, 有限的核间带宽将会产生大量由于寄存器指令无法及时比较和提交所导致的流水线停顿, 极大地影响正常执行时的性能.

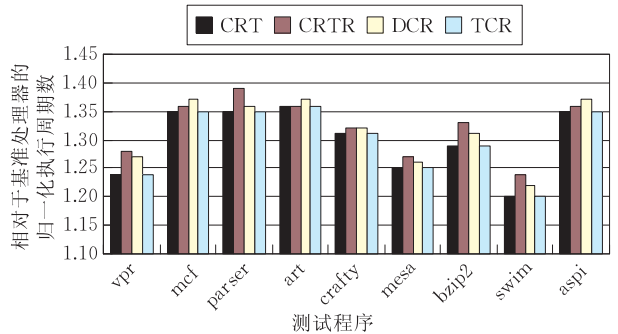


图 5 各种容软错误执行模型归一化性能比较

本文还对 DCR 和 TCR 执行模型在发生软错误情况下的性能进行了评估. 本文对 DCR 和 TCR 模型分别进行了故障注入, 每次测试都运行 10000 次 aspi 测试程序, 每次运行时随机选择一条指令并修改其运算结果, 以模拟一次 SEU 故障.

DCR 执行模型检测到软错误后, 将回退到上一现场保存点继续执行. 因此其发生软错误以后的性能依赖于两个现场保存点之间的时间间距. 如果两个现场保存点之间的时间间距过长, 可能导致一次回退过多, 重新执行时将有较大的性能开销. DCR 执行模型现场保存点之间的间距与 L1 DCache 大小以及 CLV 队列长度相关. 图 6 显示了在不同 DCache 大小和 CLV 队列长度配置下, DCR 执行模型在发生软错误的情况下的平均性能. 从图 6 中可知, L1 DCache 越大, CLV 队列越长, DCR 执行模型

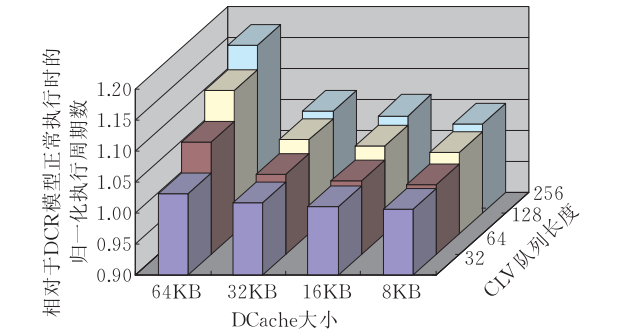


图 6 不同 DCache 和 CLV 配置下发生软错误时 DCR 平均归一化性能

中的现场保存点间距越远,发生软错误后需要回退的也就越多.在典型的配置下,即 L1 DCache 大小为 32KB,CLV 队列长度为 128 时,相对于正常执行,DCR 模型在发生软错误情况下进行回退所导致的平均性能开销为 5.2%.

TCR 执行模型在正常执行时的性能与 CRT 相同.但是在检测到软错误后将进行重构,而不同的内核被破坏引起的重构过程不同,所需的时间也不同.图 7 表明了不同的内核被破坏的情况下,TCR 执行模型的平均性能.从图 7 中可知,头线程被破坏引起的重构性能开销最大,为 3.1%.这是因为头线程被破坏将导致头线程的 L1 DCache 全部写回,并且中线程在继续执行时,将改由自己的 DCache 取数据,在重构后继续执行的早期,将发生大量 DCache 失效的情况.中线程被破坏后将导致中线程阻塞,以等待尾线程的结果,带来的额外性能开销为 0.8%.而如果尾线程被破坏,其重构过程将不会影响程序的正常执行.所以在发生软错误情况下,相对于正常执行,TCR 模型由于重构导致的平均性能损失为 1.3%.

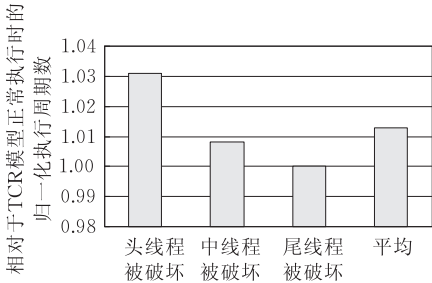


图 7 不同内核被破坏情况下 TCR 平均归一化性能

7.2 带宽评估

本文对 CRT、CRTR、DCR 和 TCR 4 种执行模型对核间队列带宽的需求进行了比较,比较结果见表 2.由于需要对频繁执行的寄存器指令进行比较,CRTR 执行模型对 RVQ 队列的带宽需求最高,为 8.8Byte/Cycle,比 LVQ、StB 和 BOQ 3 个队列带宽需求的总和还要高.由于 CRT、DCR 和 TCR 3 个执行模型都不需要对寄存器指令进行比较,可以完全取消 RVQ 队列,大大减少了对核间通信的带宽需求,从而简化了核间通信的设计. DCR 执行模型总的带宽需求与 CRT 相同,为 CRTR 的 42.5%.而 TCR 执行模型需要在队列中增加额外的 verify 位,标志该数据是否通过了中线程的比较,并且还需要额外的核间反馈信号来实现重构,所以其总的带宽需求略大于 CRT 和 DCR,但是也只有 CRTR 的 45.8%.

表 2 带宽需求比较

单位: Byte/Cycle					
	LVQ	StB	BOQ	RVQ	Total
CRT	3.0	2.7	0.8	0	6.5
CRTR	3.0	2.7	0.8	8.8	15.3
DCR	3.0	2.7	0.8	0	6.5
TCR	3.2	2.9	0.9	0	7.0

7.3 容软错误评估

本文对 DCR 和 TCR 两种执行模型的容软错误能力进行了评估.采用了故障注入方法,每次测试运行 10000 次 aspi 测试程序,每次运行时注入一次软错误.

DCR 执行模型存在软错误不可恢复的情况.影响不可恢复错误的因素是软错误的扩散速度和扩散时间.软错误的扩散速度与程序形态相关,无法通过硬件设计改变.软错误的扩散时间是指发生软错误到下一次现场保存点之间的时间间距,可以通过增加两次现场保存点之间的间距来有效增加扩散时间.如果两次现场保存点之间的时间间距足够长,绝大多数软错误都有足够的时间扩散到 load、store 或分支指令,就可以有效减少不可恢复软错误的数目. DCR 中两次现场保存点之间的时间间距,与 DCache 大小和 CLV 队列长度相关.图 8 显示了在不同 DCache 大小和 CLV 队列长度配置下,每注入 10000 次软错误所发生的不可恢复错误次数. DCache 越大,CLV 长度越大,两次现场保存点之间的时间间距就越长,相应的软错误扩散时间就越长.所以不可恢复软错误的数目随着 DCache 和 CLV 队列长度的增加而减少.但是,过大的 DCache 和 CLV 队列将带来额外的面积和功耗开销,并且在检测到软错误进行回退时,过长的现场保存点间距还会带来大量的由重新执行导致的性能开销.综合考虑,要在有效减少不可恢复错误的前提下,注意性

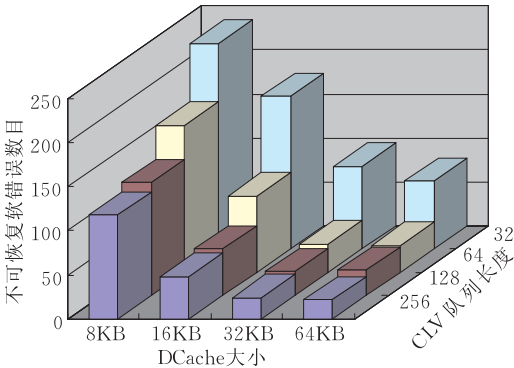


图 8 不同 DCache 和 CLV 配置下不可恢复软错误数目

能、面积和功耗的开销,可以选择 L1 DCache 大小为 32KB,CLV 队列长度为 128.此时可以在 5.2% 的回退性能开销下,恢复 99.69% 的软错误.

本文对 TCR 执行模型也进行了 SEU 故障注入,实验结果表明,TCR 没有发生任何执行错误,有效地证明了 TCR 执行模型对 SEU 故障的屏蔽特性.

## 8 与已有工作的比较

传统的软错误检测执行模型 CRT<sup>[4]</sup>对核间通信带宽要求低、性能高、易于实现和扩展,但是该模型只能对软错误进行检测. CRTR<sup>[5]</sup>执行模型在 CRT 的基础上实现了对软错误的恢复,更有利于提高微处理器的可靠性.但是由于 CRTR 需要通过 RVQ 队列对频繁执行的寄存器指令进行比较,所以其对核间通信队列的带宽要求高,在带宽不能满足的情况下其性能将显著降低,并且高带宽的核间通信队列也限制了 CRTR 模型的可实现性和可扩展性.研究人员提出了很多方案来解决 CRTR 模型中的核间通信问题,大量的工作集中于减小进入 RVQ 队列的数据,或对队列中的数据进行压缩. DBCE (Dependence-Based Checking Elision)<sup>[13]</sup>和 DDBCE (Death- and Dependence-Based Checking Elision)<sup>[5]</sup>技术寻找寄存器指令间的相关关系,只有相关链的最后一条寄存器指令的结果才进入 RVQ 队列.文献[17]采用 fingerprinting<sup>[18]</sup>的方法对进入 RVQ 队列的数据进行压缩,实现了能恢复软错误的 Reunion 执行模型.这些技术的目的在于减少 RVQ 队列的带宽需求.而本文提出的 DCR 和 TCR 执行模型完全取消了 RVQ 队列,在与传统的软错误检测执行模型 CRT 相同的核间通信复杂度下,以合理的性能开销分别实现了对软错误的恢复和屏蔽.

## 9 结 论

本文提出了基于双模冗余的 CMP 软错误恢复执行模型 DCR 和基于三模冗余的 CMP 软错误屏蔽执行模型 TCR. DCR 模型在传统的软错误检测模型 CRT 的基础上增加了硬件实现的现场保存与恢复,利用 Cache 失效处理过程隐藏现场保存的时间开销,并且采用特殊的机制实现原始执行与恢复执行之间的 load 数据一致性. TCR 执行模型将三模冗余应用于内核一级,在检测到软错误以后可以

动态重构,以实现 SEU 故障的屏蔽.

实验结果表明,相对于传统的软错误恢复模型 CRTR、DCR 和 TCR 对核间通信带宽的需求分别降低了 57.5% 和 54.2%,在较小的带宽需求下实现了对软错误的恢复和屏蔽,有利于降低设计和实现的复杂度. DCR 执行模型通过添加现场保存与恢复功能,可以在合理的性能开销下恢复 99.69% 的软错误.而 TCR 执行模型通过提高冗余度,以较小的性能开销实现了对 SEU 的全面屏蔽.因此,DCR 执行模型适合应用于内核数目较少的情况,通过增加单个内核的复杂度实现对软错误的有效恢复;而 TCR 执行模型更适合开发今后“many core”时代大量的计算内核资源,在保持性能的前提下有效提高对 SEU 故障的屏蔽能力.

## 参 考 文 献

- [1] Hareland S, Maiz J, Alavi M, Mistry K, Walstra S, Dai C. Impact of CMOS scaling and SOI on soft error rates of logic processes//Proceedings of the Symposium on VLSI Technology. Kyoto, Japan, 2001: 73-74
- [2] Karnik T, Bloechel B, Soumyanath K, De V, Borkar S. Scaling trends of cosmic rays induced soft errors in static latches beyond 0.18 $\mu$ //Proceedings of the Symposium on VLSI Circuits. Kyoto, Japan, 2001: 61-62
- [3] Borkar S Y, Dubey P, Kahn K C, Kuck D J, Mulder H, Pawlowski S S, Rattner J R. Platform 2015: Intel processor and platform evolution for the next decade. Technology@Intel Magazine, 2005: 1-10
- [4] Mukherjee S S, Kontz M, Reinhardt S K. Detailed design and evaluation of redundant multithreading alternatives//Proceedings of the International Symposium on Computer Architecture. Anchorage, AK, 2002: 99-110
- [5] Goma M, Scarbrough C, Vijaykumar T N, Pomeranz I. Transient-fault recovery for chip multiprocessors//Proceedings of the International Symposium on Computer Architecture. San Diego, CA, 2003: 98-109
- [6] Ma T, Dressendorfer P. Ionizing Radiation Effects in MOS Devices and Circuits. New York: Wiley-Interscience, 1989
- [7] Kaschmitter J L et al. Operation of commercial R3000 processors in the low earth orbit (LEO) space environment. IEEE Transactions on Nuclear Science, 1991, 38(6): 1415-1428
- [8] Eaton P et al. Single event transient pulsewidth measurements using a variable temporal latch technique. IEEE Transactions on Nuclear Science, 2004, 51(6): 3365-3368
- [9] Shivakumar P et al. Modeling the effect of technology trends on the soft error rate of combinational logic//Proceedings of the International Conference on Dependable Systems Networks. Bethesda, US, 2002: 389-398

- [10] Slegel T J, Averill R M, Check M A, Giamei B C, Krumm B W, Krygowski C A, Li W H, Liptay J S, MacDougall J D, McPherson T J, Navarro J A, Schwarz E M, Shum K, Webb C F. IBM's S/390 G5 microprocessor design. *IEEE Micro*, 1999, 19(2): 12-23
- [11] Rotenberg E. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors//*Proceedings of the International Symposium on Fault-Tolerant Computing*. Madison, WI, 1999: 84-91
- [12] Reinhardt S K, Mukherjee S S. Transient fault detection via simultaneous multithreading//*Proceedings of the International Symposium on Computer Architecture*. Vancouver, Canada, 2000: 25-36
- [13] Vijaykumar T, Pomeranz I, Cheng K. Transient-fault recovery via simultaneous multithreading//*Proceedings of the International Symposium on Computer Architecture*. Anchorage, AK, 2002: 87-98
- [14] LaFrieda C, Ipek E, Martinez J F, Manohar R. Utilizing dynamically coupled cores to form a resilient chip multiprocessor//*Proceedings of the International Conference on Dependable Systems Networks*. Edinburgh, UK, 2007: 317-326
- [15] Houghton A D. *The Engineer's Error Coding Handbook*. London: Chapman & Hall, 1997
- [16] Mahmood A, McCluskey E J. Concurrent error detection using watchdog processors — A survey. *IEEE Transactions on Computer*, 1988, 37(2): 160-174
- [17] Smolens J C, Gold B T, Falsafi B, Hoe J C. Reunion: complexity-effective multicore redundancy//*Proceedings of the International Symposium on Microarchitecture*. Orlando FL, 2006: 223-234
- [18] Smolens J C, Gold B T, Kim J, Falsafi B, Hoe J C, Nowatzky A G. Fingerprinting: Bounding soft-error-detection latency and bandwidth. *IEEE Micro*, 2004, 24(6): 22-29



**GONG Rui**, born in 1980, Ph.D. candidate. His research interests include high reliable microprocessor design and asynchronous integrated circuit design.

His research interests include microprocessor design, high performance computer architecture and asynchronous integrated circuit design.

**WANG Zhi-Ying**, born in 1956, Ph.D., professor, Ph.D. supervisor. His research interests include microprocessor design, high performance computer architecture and asynchronous integrated circuit design.

**DAI Kui**, born in 1968, Ph.D., associate professor.