

一种向分支指令后插入冗余指令的容错微结构

张仕健 胡伟武

(中国科学院计算技术研究所计算机系统结构重点实验室 北京 100080)

(中国科学院研究生院 北京 100039)

摘 要 随着深亚微米工艺的广泛应用,瞬态故障已成为芯片失效的主要原因.文中提出了一种向分支指令后插入冗余指令的容错微结构,利用分支误预测浪费的处理带宽,降低了冗余执行导致的性能损失.实验结果表明,该技术的性能损失在6%~31%之间,平均为21%,明显低于MBI技术而和DIE技术的性能损失相当.该技术能够检测流水线上各阶段发生的瞬态故障并能恢复处理器状态,故障检测延时短,需要的硬件开销也较小,非常适合提高带有简单预测机制的嵌入式微处理器的容错能力.

关键词 瞬态故障;时间冗余;容错;分支预测;嵌入式微处理器

中图法分类号 TP303

Injecting Redundant Instructions Behind Branches for a Fault-Tolerant Architecture

ZHANG Shi-Jian HU Wei-Wu

(Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

(Graduate School of Chinese Academy of Sciences, Beijing 100039)

Abstract Since deep submicron manufacturing process is widely used in microprocessors, transient faults have become the main source of chip faults. A new fault-tolerant technique is proposed that inject redundant instructions behind primary branch instructions. It utilizes the wasted processing bandwidth during branch misprediction for redundant execution, hence the performance overhead is mitigated. The experiment results show that performance penalty resulted from the solution is ranging from 6% to 31%, with an average of 21%, which is much lower than that of MBI technique and almost equal to that of DIE technique. The proposal can detect and recovery faults occurring in the entire pipeline, provides short fault detection latency and requires modest hardware cost. It is well suited to realize a fault-tolerant embedded microprocessor which has a simple branch predictor.

Keywords transient fault; temporal redundancy; fault-tolerant; branch prediction; embedded microprocessor

1 引 言

随着集成电路制造工艺的进步,微处理器高性

能和高可靠的矛盾变得更加突出^[1].一方面,缩小器件尺寸、降低工作电压和提高工作主频,是提升微处理器性能的主要手段;另一方面,这些工艺技术使得微处理器计算结果的可信度受到瞬态故障的严重威

胁,甚至一些中、低端的嵌入式微处理器也不得不考虑容错性。瞬态故障是一种由粒子辐射、噪声干扰等原因导致的非物理性故障,在深亚微米工艺下,它已成为芯片失效的主要原因^[2],本文后继部分所述的故障均指瞬态故障。

在微结构层面,增强微处理器容错能力的技术分为三类。第一类为基于编码的信息冗余技术,它被广泛地用于保护一些基于内存单元的部件,如缓存、寄存器文件等,但是由于编、解码(如 ECC)涉及较大的硬件开销,流水线上的组合逻辑和锁存单元一般不采用该技术^[3-4]。第二类为空间冗余技术,它在多个处理通路上执行相同的指令流,比较通路之间执行状态的一致性。例如 IBM G5 处理器^[5]复制了 I 单元和 E 单元,通过检测指令在原单元和副本单元之间的执行状态来发现故障。该技术故障覆盖率高,但是硬件开销大,一些中、低端的微处理器无法承受。第三类为时间冗余技术,它在相同的处理通路上多次执行相同的指令流,通过比较指令前后的执行结果来发现故障。该技术硬件开销小,是目前研究的热点。Ray 等人提出重复使用超标量数据通路来检测故障(DIE)^[6]。他们在译码阶段向流水线插入多个指令副本,在指令修改寄存器状态之前比较它们的执行结果是否一致,如果不一致,则采用指令猜测执行时的回绕机制来恢复处理器状态。他们提出了一些时间冗余技术的共性问题,但是该技术故障覆盖率不高,不能检测流水线上取指阶段和译码阶段发生的故障。Qureshi 等人提出利用缓存失效损失的带宽执行冗余指令(MBI)^[7]。该技术故障覆盖率高,可以检测整个流水线上发生的故障,但故障恢复比较困难,在非访存密集型应用中性能损失高达 50%。

本文提出了一种向分支指令后插入冗余指令的时间冗余技术(IRIB)。它利用微处理器分支误预测浪费的带宽,降低了冗余执行导致的性能损失。实验结果表明,IRIB 的性能损失在 6%~31%之间,平均为 21%,显著低于 MBI 技术而和 DIE 技术的性能损失相当。IRIB 能够检测流水线上从取指令到提交结果各阶段发生的瞬态故障并能恢复处理器状态,故障检测延时短,同时需要的硬件开销也较小,非常适合于提高带有简单预测机制的嵌入式微处理器的容错能力。

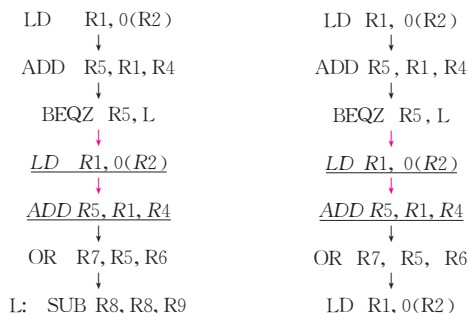
本文第 2 节介绍 IRIB 技术的总体思路;第 3 节论述 IRIB 的微结构及操作过程;第 4 节介绍实验配置;第 5 节给出实验结果并进行分析;最后总结全文。

2 总体思路

IRIB 技术是在处理器执行分支指令时插入冗余指令,通过比较主指令和它的冗余指令的计算结果来检测故障。考查一个程序段,如图 1(a)所示,图 1(b)是处理器在未出现故障时的一条执行路径,主指令 BEQZ 后插入了冗余指令(图中下划线的指令),主指令 LD 在执行结束时修改主结构寄存器 R1 并将运算结果保存在一个队列中,冗余指令 LD 在执行结束时和保存在结果队列中的值进行比较,由于结果一致,修改冗余结构寄存器 R1,当主指令 BEQZ 执行结束时,处理器切换到主指令流的分支路径 OR 指令处执行。图 1(c)是处理器在出现故障时的一条执行路径,假设主指令 LD 在执行过程中受故障影响,它产生的错误结果修改了主结构寄存器 R1 并保存在队列中,后继主指令 ADD 及 BEQZ 与 LD 指令直接或间接相关,采用出错的寄存器 R1 的值参与运算。冗余指令 LD 在提交阶段比较本次计算结果和先前保存在队列中的结果,发现它们不一致,于是就检测出该指令在执行过程中发生了故障。为了恢复处理器状态,处理器首先要废弃流水线正在执行的指令,如冗余指令 ADD 和主指令 OR;然后用冗余的结构寄存器恢复被污染的主结构寄存器 R1 及 R5,冗余的结构寄存器保存的是通过检测的正确的寄存器状态;最后处理器调整程序计数器从主指令 LD 处重新执行,从而克服了故障的影响。从以上的分析可知,只要主指令和冗余指令受故障影响表现出不同的运算结果,IRIB 就能检测到故障并恢复处理器状态,当主指令发生故障同时它的冗

```
LD    R1, 0(R2)
ADD   R5, R1, R4
BEQZ  R5, L
OR    R7, R5, R6
L: SUB R8, R8, R9
...
```

(a) 源程序段



(b) 无故障时的一条执行路径 (c) 出现故障时的一条执行路径

图 1

余指令也发生了同样的故障时,由于它们的运算结果一致,IRIB 就无法检测出故障,但是这种情况发生的可能性是极其微小的. IRIB 技术的具体实现机制将在本文第 3 节详细阐述.

3 设 计

本节首先介绍 IRIB 的总体微结构,然后讨论需要的额外部件和操作过程.

3.1 总体微结构

IRIB 在通用的超标量微处理器内核上增加一些部件以双线程的方式执行程序指令,见图 2. 图中阴影部分是新增的部件, P_ARF 是原结构寄存器文件, R_ARF 是冗余的结构寄存器文件, R_PC 是冗余指令程序计数器. 主指令流在提交阶段,如提交

的是写访存指令,则将写访存信息保存在写访存队列中,否则将操作结果保存在操作队列中,并修改 P_ARF 中寄存器的内容. 冗余指令流在提交指令时,如提交的是写访存指令,则写访存队列出队列并和提交的信息进行比较,否则操作队列出队列并和提交的信息进行比较. 当结果一致时,冗余指令流修改 R_ARF 中寄存器的内容或写数据缓存,否则出现异常. 主指令流在取指阶段,如果取回的是一条分支指令或发现操作队列和写缓存队列没有足够的空间保存后继指令的操作结果,IRIB 将根据 R_PC 取冗余指令执行. 当主指令流提交一条分支指令或操作队列和写缓存队列中的指令都已被冗余执行时,IRIB 根据 P_ARF 中的程序计数器切换到执行主指令流的状态.

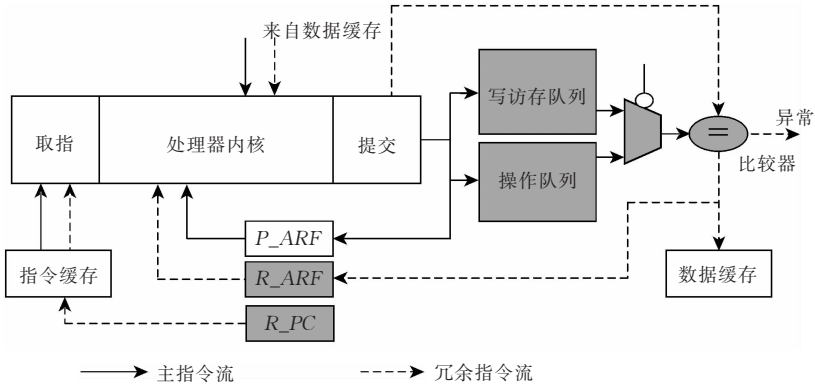


图 2 IRIB 的微结构

我们的方案可以检测流水线上各个流水级发生的故障,具备很高的故障覆盖面. 本文假定 IRIB 中的 P_ARF 、 R_ARF 、 R_PC 、指令缓存和数据缓存采用奇偶校验或 ECC 编码的方式进行保护.

3.2 额外的部件

在微结构层面,IRIB 需要以下额外的部件:

(1) 冗余的结构寄存器文件(R_ARF). 该部件保存冗余指令流执行时的结构寄存器状态. 它包括通用寄存器、控制寄存器和程序计数器,其中程序计数器记录下一条待检验的冗余指令的地址. 由于该部件中寄存器的结果通过了比较器的检验,因此当 IRIB 检测到故障时,可采用该部件恢复 P_ARF 的寄存器状态.

(2) 操作队列. 该队列保存主指令流的执行结果. 对于写访存指令,它保存一个指令标识,表示访存信息保存在写访存队列中;对于分支指令,它保存分支的转移地址;对于其它指令,它保存指令的运算结果. 该队列还有一个入队列的尾指针和两个头指

针. 一个头指针叫比较头指针,用于出队列比较运算结果;另一个叫取指头指针,指示 IRIB 取冗余指令时该指令在队列中的位置.

(3) 写访存队列. 该队列保存主指令流的写访存信息. 队列中每一项包括访存地址和待写入的值. 对于主指令流而言,该队列保存了最新的内存映像,因此主指令流的读访存指令需要从写访存队列和数据缓存中选择访存结果,只有当写访存队列没有命中时,才考虑选择数据缓存中的内容.

(4) 选择器和比较器. 当冗余指令流提交一条写访存指令时,选择器选取写缓存队列的输出信息和冗余指令的访存地址、访存值一一比较,如果结果匹配,则写数据缓存;当冗余指令流提交的不是写访存指令时,选择器选取操作队列中的输出信息和冗余指令的操作结果进行比较,如结果一致,则修改 R_ARF 中寄存器的内容. 当结果不匹配时,比较器输出一个异常状态,IRIB 通过异常处理来恢复处理器状态.

(5) 冗余指令程序计数器(R_PC): 用于保存待取的冗余指令的地址. 它的初始值等于 P_ARF 中的程序计数器; 在执行冗余指令的过程中, 对于非分支指令, 下一条指令的 R_PC 等于当前 R_PC+1 ; 对于分支指令, 下一条指令的 R_PC 等于操作队列中取指头指针所指的转移地址.

另外, IRIB 还需要一些判断逻辑和选择逻辑. 例如: IRIB 在取指阶段需要选择 P_ARF 中的程序计数器和 R_PC ; 在指令执行阶段, 需要判断指令是主指令还是冗余指令, 分别从 P_ARF 和 R_ARF 中读取操作数.

3.3 操作过程

在 IRIB 中, 主指令流和冗余指令流交替取指执行, 比较器通过比较指令前后两次的执行结果, 来检测流水线上发生的瞬态故障, 这个过程涉及如下操作.

(1) 执行主指令流: IRIB 执行主指令流, 有几点不同于通常的指令执行过程. (a) 取指阶段不必访问分支预测器, 对于非分支指令, 下一条主指令的 PC 等于当前 PC 加 1, 对于分支指令, 下一条主指令的 PC 等于该指令提交时的分支结果, 这期间空闲的处理带宽用于执行冗余指令; (b) 执行读访存操作时需要访问写访存队列, 得到最新的内存映像; (c) 在提交阶段, 指令的结果需要保存在操作队列或写访存队列中.

(2) 插入冗余指令: IRIB 在三种情况下选取 R_PC 插入冗余指令. (a) 主指令流取回一条分支指令; (b) 主指令流取回一条指令后发现流水线中的指令数等于操作队列中的空闲项数; (c) 主指令流取回一条 I/O 指令, 为保证外设得到正确的处理器机状态, 操作队列和写缓存队列中的指令结果必须在 I/O 指令执行前都通过检验. 第一种情况是主动插入, 可利用分支误预测浪费的带宽执行冗余指令; 后两种情况是被动插入, 保证所有指令的执行结果都经过了检验, 但会导致较大的性能损失.

(3) 执行冗余指令流: IRIB 根据 R_PC 访问指令缓存, 并根据操作队列的取指头指针所指的内容设置下一条指令的 R_PC ; 当指令取回后, 标示为冗余指令, 并移动取指头指令; 冗余指令在执行过程中从 R_ARF 读取操作数, 如执行读访存则直接访问数据缓存; 冗余指令的执行结果通过比较验证后修改 R_ARF 或写数据缓存.

(4) 切换到主指令: 对应于插入冗余指令的三

种情况, IRIB 有三种方式选取 P_ARF 中的程序计数器重新回到取主指令的状态. (a) 主指令流提交一条分支指令, 修改 P_ARF 中的程序计数器后切换; (b) 操作队列中的指令都已被冗余取指, 即取指头指针等于尾指针; (c) 操作队列和写访存队列都为空.

(5) 异常处理: 当 IRIB 检测到瞬态故障时, 需要通过处理器异常恢复到正确的状态, 并从出现故障的指令处重新执行. 首先, 它要废弃流水线上的所有指令, 清空操作队列和写访存队列; 接着, 根据 R_ARF 中各寄存器的内容恢复 P_ARF , 其中用 R_ARF 中的程序计数器恢复 P_ARF 中的程序计数器; 再次, 用 R_ARF 中的程序计数器的内容设置冗余指令程序计数器 R_PC ; 最后, 根据 P_ARF 中的程序计数器重新取主指令执行.

另外, 为了消除线程内指令间的伪相关, 增大指令间的并行度, IRIB 的主指令流和冗余指令流需要分别进行重命名^[6]. 我们采用的方法是对 R_ARF 的定点寄存器堆和浮点寄存器堆都增加一个状态有效位和一个指令窗口标识位, 将目标寄存器重命名到指令窗口上, 具体过程参考文献[8].

4 实验环境

我们采用龙芯 1 号微处理器的模拟器来模拟和评估 IRIB^[9-10]. 处理器的基准配置见表 1, 表中括号内的值表示功能部件操作延时或缓存的命中延时. IRIB 在基准配置的基础上增加了操作队列和写缓存队列, 它们的缺省值皆为 32 项. 为了和 IRIB 进行比较, 我们还在基准配置的基础上模拟了 DIE 和 MBI. 模拟的 DIE 每条指令插入一个副本. 模拟的 MBI 有 1024 项后备日志缓冲(BACKLOG BUFFER), 在一级指令缓存未命中时触发执行冗余指令. 这和原文的二级缓存未命中时触发有所不同, 原因是一方面嵌入式微处理器一般没有二级缓存; 另一方面, 一级数据缓存未命中时, 流水线上的数据通路已被阻塞, 冗余指令无法执行.

我们从 Mibench^[11] 中选用了 8 个典型的嵌入式程序作为运行负载. 每个测试程序编译成龙芯指令集, 详细模拟 8 亿条指令. 表 2 给出了测试程序及输入的数据集. 本文中处理器的性能是在处理器未注入故障时统计的, 用每个时钟周期的指令数(IPC)来表示, 性能损失用 IPC 的下降百分比表示.

表 1 处理器的基准配置

(a)

取指/译码/发射/提交带宽	预测器类型	分支误预测的延时	RUU 大小	整型 ALU 部件	整型乘法/除法部件	浮点 ALU 部件
1	taken 静态预测器	1 个时钟周期	8	1(1)	1(2,32)	1(3)

(b)

浮点乘法/除法/开方部件	一级缓存端口	每个部件的保留站	一级数据缓存	一级指令缓存	二级缓存	访存延时
1(2,23,112)	1(3)	2	8KB,2 路, 每路 32Byte (1)	8KB,2 路, 每路 32Byte (1)	无	头块 60 个时钟周期, 块间 3 个时钟周期

表 2 测试程序及输入数据集

测试程序	输入数据集
qsort	input_large.dat
susan	input_large.pgm
dijkstra	input.dat
stringsearch	无
patricia	large.udp
blowfish	encode, input_large.asc
adpcm	large.pcm
crc32	large.pcm

5 实验结果及分析

5.1 三种处理器模型的性能比较

为了测量分支误预测浪费的处理带宽,评估执行冗余指令导致的性能损失,我们模拟了三种微处理器模型,结果见图 3. 图中,BM 表示基准处理器,PBM 表示拥有理想分支预测器的基准处理器,PM 表示拥有理想分支预测器、理想指令缓存和理想数据缓存的基准处理器.PBM 和 BM 之间的性能差距显示了分支误预测导致的性能损失. 由于指令在第二次执行过程中可以利用第一次执行时已取入缓存中的指令和数据,利用已计算出的分支结果,因此执行冗余指令的处理器模型类似于 PM. 实验结果显示,在大部分测试程序中分支误预测导致的性能损失比较明显. 所有测试程序,除了 susan 和 patricia 外,性能损失都在 20%左右,平均性能损失为 16%.

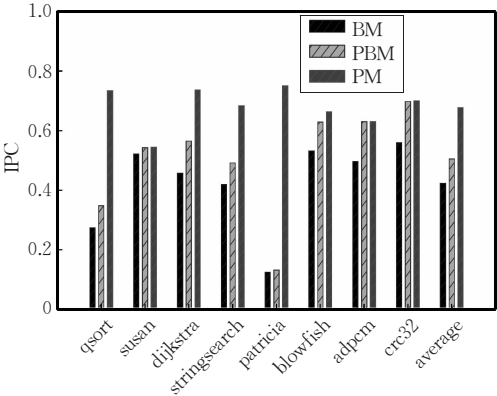


图 3 三种处理器模型的性能

实验结果还表明,PM 模型的性能显著优于 BM 模型. 因此,对于一些分支误预测率比较高的程序,可以利用浪费的处理带宽执行冗余指令,从而降低性能损失;对于一些分支误预测率比较低的程序,冗余指令较快的执行速度,可以减轻对原程序性能的影响.

5.2 冗余指令的分布

我们在 3.3 节阐述了执行冗余指令的 3 种触发方式,图 4 给出了这 3 种方式的比率,其中,BRANCH 表示由分支指令触发执行的冗余指令,OPQ 表示操作队列满时被迫执行的冗余指令,SYSCALL 表示要执行 I/O 指令时被迫执行的冗余指令. 结果显示,除了程序 susan,其余程序有 68%~85%的冗余指令是由分支指令触发执行的;平均而言,BRANCH 的比率约为 72%,OPQ 的比率约为 28%,SYSCALL 的比率仅为 0.04%. 分支触发的冗余指令可以利用基准处理器中分支误预测浪费的带宽,降低冗余执行导致的性能损失. 程序 susan 中分支指令比较少且分布很不均匀,因此被迫执行的冗余指令比率比较高.

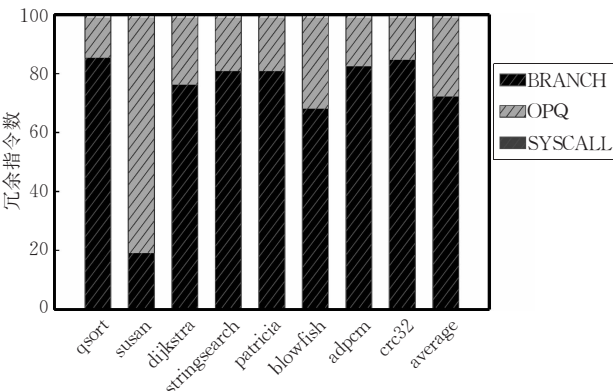


图 4 IRIB 中冗余指令的分布

5.3 性能损失

图 5 比较了 IRIB、DIE、MBI 3 种技术的性能损失. 结果显示,IRIB 的性能损失明显低于 MBI,而和 DIE 差别不大. 在程序 qsort,adpcm 中 IRIB 充分利用分支误预测损失的带宽,显著降低了性能损失;在程序 susan 中 IRIB 的性能损失相对较高,原因是该

程序中的冗余指令绝大多数是被迫执行的,同时该程序分支误预测率比较低;总体来看,IRIB 的性能损失为 6%~31%,平均为 21%。结果还反映出两个影响性能损失的共同要素。第一个要素为缓存命中率。当缓存命中率比较低时,三种技术方案都表现出较小的性能损失,如程序 patricia 和 qsort 所示,这是因为缓存失效导致的大量延时隐藏了冗余执行导致的性能损失,这和文献[7,12]的实验现象是一致的。第二个要素为功能部件的利用率。功能部件利用率越高,主指令和冗余指令竞争功能部件更加激烈,性能损失会加大。如程序 blowfish 和 crc32 整型 ALU 部件利用率较高,性能损失都比较突出,文献[13-14]表述了相似的实验现象。总之,虽然 IRIB 和 MBI 都可检测整个流水线上的瞬态故障,但是 IRIB 的性能损失显著低于 MBI;IRIB 的性能损失和 DIE 相差不大,而 DIE 仅能检测部分流水线上的瞬态故障。

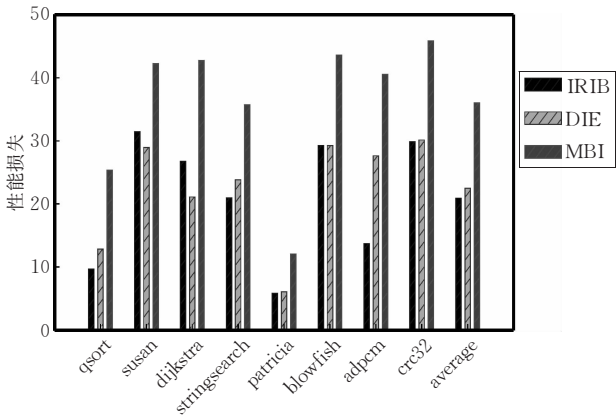


图 5 IRIB、DIE 及 MBI 的性能损失

5.4 操作队列的影响

操作队列的大小会影响 IRIB 的性能损失和硬件开销。一方面,较大的操作队列可以减少被迫执行的冗余指令数,从而降低性能损失;另一方面,操作队列过大会显著加大芯片面积,增加成本。我们模拟了 IRIB 中操作队列大小和平均性能损失的关系,见图 6。结果显示,加大操作队列,平均性能损失会逐

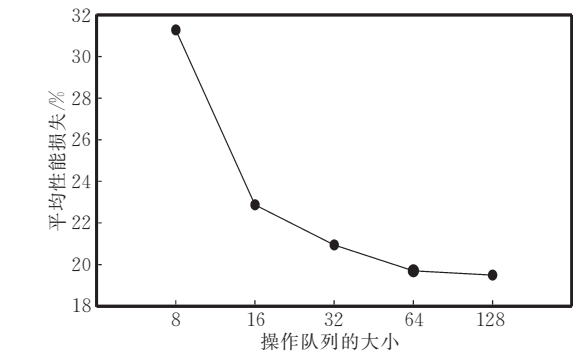


图 6 操作队列对性能损失的影响

渐下降,但是降低的幅度变化很大。当操作队列从 8 项增加到 32 项时,性能损失从 31%降低到 21%;当操作队列从 32 项变化到 128 项时,性能损失仅下降了 2%。因此,权衡性能损失和硬件开销,我们将 32 项作为操作队列的缺省配置。需要指出的是,本实验写访存队列和操作队列有相同数量的项配置。

5.5 硬件开销

我们已在 3.2 节讨论了 IRIB 需要的存储部件和组合逻辑。表 3 统计了它主要的存储开销,我们假设项中每个元素占用 4 个字节,不考虑校验码和重命名机制的开销。统计结果显示,IRIB 需要的存储开销为 656 个字节,相对基准处理器 16KB 的缓存配置来说,0.6KB 的开销是比较小的。组合逻辑的开销相对存储开销而言是次要的,本文不对它们作量化评估。

表 3 IRIB 的存储开销

部件	项数	项大小	合计/B
冗余的结构 寄存器文件	32 个整型寄存器, 32 个浮点寄存器, 2 个控制寄存器, 1 个程序计数器	4	268
操作队列	32	4	128
写缓存队列	32	4+4=8	256
冗余指令程序计数器	1	4	4
总计			656

5.6 故障检测及恢复延时

IRIB 通过比较主指令和冗余指令的执行结果来检测流水线上的瞬态故障,因此主指令和冗余指令的执行间隔决定了故障的检测延时。我们统计了 IRIB 技术和 MBI 技术的故障检测延时,见表 4。结果表明,IRIB 的检测延时远小于 MBI 的延时,表中 MBI 的延时和文献[7]的实验结果是接近的。IRIB 的冗余指令主要由分支指令和操作队列阻塞触发,MBI 的冗余指令主要由缓存失效和后备日志缓冲阻塞触发,相比较而言,前者触发的粒度更小,因此显著降低了故障检测延时。IRIB 具备的灵敏的故障检测机制,减少了长故障延时导致的各种潜在的故障。

表 4 故障检测延时(时钟周期)

测试程序	IRIB		MBI	
	平均值	最大值	平均值	最大值
qsort	42	1045	206	16482
susan	47	822	1675	6173
dijkstra	41	934	1739	9013
stringsearch	39	903	1008	11412
patricia	91	906	103	4319
blowfish	36	866	521	9907
adpcm	32	895	1745	4001
crc32	28	903	150	4162
总计	45	909	893	8184

IRIB 检测到故障后,通过 3.3 节的异常处理来恢复处理器状态,这个过程只需要一个时钟周期.

6 结 论

本文提出了一种向分支指令后插入冗余指令的容错微结构,能够检测流水线上各阶段发生的瞬态故障并能恢复处理器状态,同时利用分支误预测浪费的带宽,降低了冗余执行导致的性能损失.实验表明,该技术的性能损失在 6%~31%之间,平均为 21%,显著低于 MBI 技术的性能损失而和 DIE 技术相当.实验还表明,该技术需要的硬件开销比较小,能够提供较短的故障检测延时.该技术非常适用于提高带有简单分支预测机制的嵌入式微处理器的容错能力.

参 考 文 献

- [1] Ronen R, Mendelson A et al. Coming challenges in microarchitecture and architecture. *Proceedings of the IEEE*, 2001, 89(3): 325-340
- [2] Mitra S, Seifert N et al. Robust system design with built-in soft-error resilience. *IEEE Computer*, 2005, 38(2): 43-52
- [3] Quach N. High availability and reliability in the ITANIUM processor. *IEEE Micro*, 2000, 20(5): 61-69
- [4] Gaisler J. A portable and fault-tolerant microprocessor based on the SPARC v8 architecture//*Proceedings of the International Conference on Dependable Systems and Networks*. Washington, USA, 2002: 409-415
- [5] Slegel T J et al. IBM's S/390 G5 microprocessor design. *IEEE Micro*, 1999, 19(2): 12-23
- [6] Ray J, Hoe J C, Falsafi B. Dual use of superscalar datapath for transient-fault detection and recovery//*Proceedings of the 34th ACM/IEEE International Symposium on Micro- Archi-*

tecture. Austin, Texas, USA, 2001: 214-224

- [7] Qureshi M K, Mutlu O, Patt Y N. Micro-architecture-based introspection: A technique for transient-fault tolerance in microprocessors//*Proceedings of the International Conference on Dependable Systems and Networks*. Yokohama, Japan, 2005: 434-443
- [8] Hu Wei-Wu, Tang Zhi-Min, Feng Lei. A pipeline system and methodology using operation queue. The Institute of computing technology, CAS, Beijing: Technical Report 01141495.2, 2003(in Chinese)
(胡伟武,唐志敏,冯雷.基于操作队列复用的指令流水线系统和方法.中国科学院计算技术研究所,北京:技术报告 01141495.2, 2003)
- [9] Hu Wei-Wu, Tang Zhi-Min. Architecture of the Godson1 processor. *Chinese Journal of Computers*, 2003, 26(4): 385-396(in Chinese)
(胡伟武,唐志敏.龙芯 1 号处理器结构设计.计算机学报, 2003, 26(4): 385-396)
- [10] Zhang Shi-Jian, Zhang Fu-Xin, Tang Zhi-Min. Performance simulator for godson1 microprocessor. *Mini-Micro Systems*, 2006, 27(12): 2317-2320(in Chinese)
(张仕健,张福新,唐志敏.龙芯 1 号微处理器性能模拟器.小型微型计算机系统, 2006, 27(12): 2317-2320)
- [11] Guthaus M R, Ringenberg J S et al. Mibench: A free commercially representative embedded benchmark suite//*Proceedings of the International Workshop on Workload Characterization*. Austin, Texas, USA, 2001: 3-14
- [12] Lai S C, Lu S L et al. Ditto processor//*Proceedings of the International Conference on Dependable Systems and Networks*. Washington, USA, 2002: 525-534
- [13] Nickel J B, Somani A K. REESE: A method of soft error detection in microprocessors//*Proceedings of the International Conference on Dependable Systems and Networks*. Goteborg, Sweden, 2001: 401-410
- [14] Sato T. Exploiting instruction redundancy for transient fault tolerance//*Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. Boston, Massachusetts, USA, 2003: 547-554



ZHANG Shi-Jian, born in 1975, Ph.D. candidate. His main research interests include high reliable microarchitecture and high-performance computer architecture.

HU Wei-Wu, born in 1968, professor, Ph.D. supervisor. His main research interests include high-performance computer architecture, MPP system and VLSI design.

Background

With the widespread adoption of embedded microprocessor-based systems for safety critical application, fault-tolerant mechanisms have to be built into microprocessors. In this paper, a fault-tolerance microarchitecture is proposed that inject redundant instructions behind primary branch instructions. This work is an important component of the research on low

power and high reliability for the godson1 microprocessor. The project is supported by the National Basic Research Program of China(973 Program) under grant No. 2005CB321600 and the National Natural Science Foundation of China under grant No. 60325205 and No. 60603049 and Beijing Natural Science Foundation under grant No. 4072024.