(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 112131032 B
(45)授权公告日 2022.02.11

(21)申请号 202010887119.0

(22)申请日 2020.08.28

(65)同一申请的已公布的文献号
申请公布号 CN 112131032 A

(43)申请公布日 2020.12.25

(73)专利权人 北京大学
地址 100871 北京市海淀区颐和园路5号

(72)发明人 孙广宇 肖侬 吕宝财 王晓阳

(74)专利代理机构 北京万象新悦知识产权代理有限公司 11360
代理人 黄凤茹

(51)Int.Cl.
G06F 11/07(2006.01)
G06F 8/41(2018.01)
G06F 11/14(2006.01)

(56)对比文件
CN 105183379 A,2015.12.23
CN 111459846 A,2020.07.28
US 2016062818 A1,2016.03.03
US 2015074339 A1,2015.03.12
李鑫 等.面向大数据应用的混合内存架构特征分析.《大数据》.2018,第61-80页.

审查员 王丹

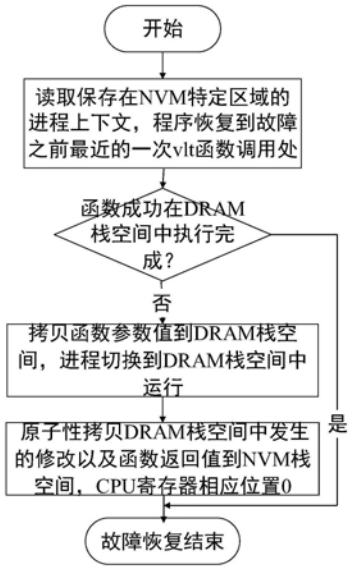
权利要求书2页 说明书5页 附图2页

(54)发明名称

一种基于混合内存系统的故障快速恢复方法

(57)摘要

本发明公布了一种基于混合内存系统的故障快速恢复方法,混合内存系统为动态随机存取存储器DRAM和非易失性存储器NVM的混合内存系统;在混合内存系统的DRAM和NVM中各自维护一个进程栈空间;设定新的函数限定符,针对使用该新的函数限定符的应用程序实现基于混合内存系统的故障快速恢复。本发明在DRAM和NVM混合内存系统中使用双栈结构运行进程的方法,减轻现有的日志和检查点技术中存在的由于引入大量额外NVM写入从而降低系统性能和设备寿命的问题,并实现系统故障后的快速恢复。



CN 112131032 B

1.一种基于混合内存系统的故障快速恢复方法,所述混合内存系统为动态随机存取存储器DRAM和非易失性存储器NVM的混合内存系统;在混合内存系统的DRAM和NVM中各自维护一个进程栈空间;设定新的函数限定符,针对使用该新的函数限定符的应用程序实现基于混合内存系统的故障快速恢复;包括如下步骤:

1)根据需要将应用程序中的一个或多个函数声明为带有新的函数限定符的函数;所述一个或多个函数可包括预计占据整个程序运行时间的很大比例的需要长时间运行或需要大量访存操作的函数;

2)将即将运行的应用程序的进程地址空间映射到NVM栈空间中;应用程序的进程地址空间包括代码段、数据段、运行时堆段和用户栈段;

3)当应用程序正常运行时,执行如下操作:

A1.混合内存系统在程序编译过程中识别应用程序中的每个函数是否带有设定的函数限定符;为带有设定的函数限定符的函数增加一条转移指令;

A2.执行应用程序,应用程序进程在NVM栈空间中启动运行;

A3.当应用程序执行到A1增加的转移指令时,设置CPU标志寄存器中的一个当前空闲的标志位为1,即表明函数将在DRAM栈空间中执行,继续执行操作A4;否则,

转到操作A7;

A4.保存进程上下文到该进程在NVM的特定区域;所述进程上下文包括页表、页表基址寄存器、程序计数器;所述特定区域是为每个进程分配的一段固定的NVM地址空间,进程结束后,该NVM地址空间可再次被别的进程利用为特定区域;

A5拷贝相应函数的参数值到DRAM栈空间,进程切换到DRAM栈空间中运行;

在运行过程中,应用程序的进程指令和数据仍从NVM中读取;当需要写入数据时,数据先写入DRAM栈空间中,并记录写入DRAM栈空间数据的地址和本应该写入NVM栈空间中的数据地址的对应关系;在函数执行结束后根据该对应关系将数据从DRAM栈空间中拷贝到NVM栈空间中;

A6.采用原子性拷贝方式将DRAM栈空间中发生的修改和DRAM栈空间中的函数返回值拷贝到NVM栈空间,将操作A3中置为1的CPU标志寄存器位置为零,完成函数运行;

A7.进程在NVM栈空间中继续运行;

A8.循环执行操作A3～A7,直到程序运行结束;

4)当系统发生故障需要重启恢复运行时,执行如下操作:

B1.读取保存在NVM特定区域的进程上下文,程序恢复到故障之前最近的一次带有设定函数限定符的函数的调用位置;

B2.判断该带有设定函数限定符函数是否成功在DRAM栈空间中运行完毕;如果是,则发生故障时DRAM栈空间中没有正在运行的函数,执行步骤B5,程序正常运行;否则执行步骤B3;

B3.拷贝该函数的参数值到DRAM栈空间,进程切换到DRAM栈空间中运行;

B4.采用原子性拷贝方式将DRAM栈空间中发生的修改及DRAM栈空间中的函数返回值到NVM栈空间,将CPU标志寄存器的相应位置0;故障恢复结束;

B5.进程继续在NVM栈空间中正常运行;

通过上述步骤,实现基于混合内存系统的故障快速恢复。

2.如权利要求1所述基于混合内存系统的故障快速恢复方法,其特征是,原子性拷贝方式包括采用写时复制方式或影子页方式。

3.如权利要求1所述基于混合内存系统的故障快速恢复方法,其特征是,B1中,如果程序中任何一个函数均未添加设定的函数限定符,则程序重新启动。

4.如权利要求1所述基于混合内存系统的故障快速恢复方法,其特征是,B2判断该带有设定函数限定符函数是否成功在DRAM栈空间中运行完毕;判断的方法包括:检验该进程对应的NVM特定区域是否为空,非空则表明未成功,反之则成功;或采用检查CPU标志寄存器的相应位,为1则表明未成功,为0则表明成功。

5.如权利要求1所述基于混合内存系统的故障快速恢复方法,其特征是,设定新的函数限定符为vlt。

一种基于混合内存系统的故障快速恢复方法

技术领域

[0001] 本发明属于高性能计算系统故障处理技术领域,涉及混合内存系统故障恢复的方法,具体涉及一种基于双栈运行的混合内存系统故障快速恢复方法,在混合内存系统的DRAM和NVM中各自维护一个进程栈空间加速程序运行,当系统故障发生之后能够快速恢复运行。

背景技术

[0002] 在高性能计算等其它需要大量节点的运行环境下,某些节点发生系统故障从而导致一些长期运行的计算任务在完成之前就被中断的可能性越来越大,因此需要在故障发生之后快速恢复到故障发生之前的状态以提高更好的可用性。在配备NVM(Non-Volatile Memory,非易失性存储器)和DRAM(Dynamic Random Access Memory,动态随机存取存储器)的混合内存系统中,实现故障恢复通常使用预写日志和周期性的检查点技术。

[0003] 预写日志需要在更新数据之前先创建日志,并且保证日志的持久化存储必须在数据持久化更新之前完成。预写日志根据内容可分为三类:撤销日志、重做日志、撤销+重做日志。日志项的内容通常包括要更新的数据地址以及数据内容,例如撤销日志的日志项包含(地址,数据旧值),重做日志的日志项包含(地址,数据新值),撤销+重做日志的日志项包含(地址,数据旧值,数据新值)。对于支持事务的系统,日志项的内容还包括事务编号等其它元数据。可以看出,预写日志会增加大量的对NVM额外的写入数据量,因为NVM比DRAM具有更高的写入延迟和有限的写入寿命,所以会降低系统性能以及设备使用时间。

[0004] 检查点需要周期性地存储应用程序状态的快照,通常分为系统级的检查点和应用级的检查点。系统级的检查点需要存储整个系统内存的快照,需要写入NVM的数据量十分巨大,如果检查点的周期比较短则对NVM十分不友好。应用级的检查点需要程序员决定什么时候建立检查点并且检查点应该存储哪些关键数据,并且写出针对特定应用的复杂的恢复代码,这种方法不但增加了程序员的负担,而且十分容易出错。

[0005] 因此,现有的日志和检查点技术中存在由于引入大量额外NVM写入从而降低系统性能和设备寿命的问题,同时也导致软件设计复杂化。

发明内容

[0006] 针对目前现有技术存在的问题,本发明提供一种新的基于混合内存系统的故障快速恢复方法,在DRAM和NVM混合内存系统中使用双栈结构运行进程的方法,减轻现有的日志和检查点技术中存在的由于引入大量额外NVM写入从而降低系统性能和设备寿命的问题,并实现系统故障后的快速恢复。

[0007] 本发明的原理是:进程的地址空间映射到NVM栈空间中,程序一直运行在NVM栈空间中。本发明为函数引入了新的限定符(如vlt,为volatile的缩写,表示易失性),该限定符是为程序员提供的选项,当程序执行到带有该限定符的函数时,程序转到DRAM栈空间中运行,充分利用DRAM的性能和写入寿命优势,由此利用以函数调用为分界极大降低了双栈代

价;如果程序员在编写程序时没有给任何函数添加该限定符,程序一直在NVM中运行。对于需要运行很长时间或者需要大量访存的函数,程序员可以在声明函数时加上限定符vlt。当程序执行到带有vlt声明的函数时,程序转到DRAM栈空间中执行,当函数执行完毕,程序再回到NVM栈空间中执行。当发生故障时,由于整个进程空间在NVM空间中,所以可以快速重启到最近一次在DRAM栈空间中运行的vlt函数,无须重复执行该函数之前所有的计算,只需要判断DRAM栈空间中运行的vlt函数是否成功运行完毕,如果该函数成功运行完毕,则在NVM栈空间中继续运行该函数后面的计算,如果该函数未成功运行完毕,则重新在DRAM栈空间中运行该函数。双栈运行的方法在本领域从未采用的原因是无法确定在转入DRAM栈空间中运行时应该拷贝哪些进程上下文,如果全栈拷贝则代价太大,严重影响程序性能,而本方法以函数为粒度,利用函数调用的特征,找到了拷贝数据最少的转换点,使得利用混合内存系统的双栈运行方法快速恢复系统故障得以高效实现。

[0008] 本发明的技术方案是:

[0009] 一种基于混合内存系统的故障快速恢复方法,所述混合内存系统为动态随机存取存储器(DRAM)和非易失性存储器(NVM)混合内存系统,在混合内存系统的DRAM和NVM中各自维护一个进程栈空间,设定一个新的函数限定符,针对使用该新的函数限定符的应用程序实现基于混合内存系统的故障快速恢复;包括如下步骤:

[0010] 1)在应用程序中,根据需要将一个或多个函数声明为带有新的函数限定符的函数;其中,一个或多个函数可以是预计占据整个程序运行时间的很大比例的需要长时间运行或需要大量访存操作的函数;

[0011] 2)将即将运行的应用程序的进程地址空间映射到NVM栈空间中;应用程序的进程地址空间包括代码段、数据段、运行时堆段和用户栈段;

[0012] 3)当应用程序正常运行时,执行如下操作:

[0013] A1.混合内存系统在程序编译过程中识别应用程序中的每个函数是否带有设定的函数限定符;为带有设定的函数限定符的函数增加一条转移指令;

[0014] A2.执行应用程序,应用程序进程在NVM栈空间中启动运行;

[0015] A3.当应用程序执行到步骤A1增加的转移指令时,设置CPU标志寄存器中的一个当前空闲的标志位为1,表明接下来的函数在DRAM栈空间中执行,继续执行操作A4;

[0016] 否则,转到操作A7;

[0017] 如果程序中任何一个函数均未添加设定的函数限定符,则不会执行到新增的转移指令;

[0018] A4.保存进程上下文到该进程在NVM的特定区域;进程上下文包括页表、页表基址寄存器、程序计数器等;

[0019] 特定区域是一段固定的NVM地址空间,每个进程分配一段空间,进程结束后则该段空间可再次被别的进程利用为特定区域;

[0020] A5拷贝相应函数的参数值到DRAM栈空间,进程切换到DRAM栈空间中运行;

[0021] 在运行过程中,应用程序的进程指令和数据仍然从NVM栈空间中读取;当需要写入数据时,为了减少NVM的写入磨损和提高写入速度,数据先写入DRAM栈空间中,并且记录写入DRAM栈空间数据的地址和本应该写入NVM栈空间中的数据地址的对应关系,方便函数在执行结束后数据从DRAM栈空间中拷贝到NVM栈空间中;

[0022] A6.原子性拷贝DRAM栈空间中发生的修改以及DRAM栈空间中的函数返回值到NVM栈空间,将操作A3中置为1的CPU标志寄存器位置为零;完成函数运行;原子性拷可采用写时复制方式或影子页方式;

[0023] A7.进程在NVM栈空间中继续运行;

[0024] A8.循环执行操作A3～A7,直到程序运行结束。

[0025] 4)当系统发生故障需要重启恢复运行时,执行如下操作:

[0026] B1.读取保存在NVM特定区域的进程上下文,程序恢复到故障之前最近的一次带有设定函数限定符的函数的调用位置;

[0027] 如果程序中任何一个函数均未添加设定的函数限定符,则程序重新启动;

[0028] B2.判断该带有设定函数限定符函数是否成功在DRAM栈空间中运行完毕,如果是,说明发生故障时,DRAM栈空间中没有正在运行的函数,所以故障没有影响DRAM栈空间,执行步骤B5,程序正常运行;否则执行步骤B3;

[0029] 判断的方法包括:检验该进程对应的NVM特定区域是否为空,非空则表明未成功,反之则成功;或采用检查CPU标志寄存器的相应位,为1则表明未成功,为0则成功;

[0030] B3.拷贝该函数的参数值到DRAM栈空间,进程切换到DRAM栈空间中运行;

[0031] B4.原子性拷贝DRAM中发生的修改以及函数返回值到NVM栈空间,将CPU标志寄存器的相应位置0;恢复结束;

[0032] B5.进程继续在NVM栈空间中正常运行。

[0033] 通过上述步骤,实现基于混合内存系统的故障快速恢复。

[0034] 本发明具体实施时,设定新的函数限定符vlt;在应用程序设计中,可根据需要选择将预计长时间运行或需要大访存操作的函数声明为带有该新限定符的函数。程序在编译过程中,如果遇到该限定符修饰的函数,系统会在带有限定符的函数之前插入一条转移指令,CPU读取执行该转移指令后,会将该函数的执行转入到DRAM栈空间中。程序员可以将预计长时间运行或需要大量访存操作的函数,该函数预计会占据整个程序运行时间的很大比例,例如带有多重循环的复杂的迭代计算的函数,或者对一大块数组内容进行排序的函数等,声明为带有新限定符的函数;当该函数运行完毕,程序再回到NVM栈空间中运行。

[0035] 本发明的有益效果:

[0036] 本发明提供一种新的基于混合内存系统的故障快速恢复方法,在DRAM和NVM混合内存系统中使用双栈结构运行进程的方法。利用本发明提供的技术方案,可以不需要程序员编写复杂易错的故障恢复代码就能实现系统故障快速恢复,同时减少对NVM的大量的额外写入,减轻现有的日志和检查点技术中存在的由于引入大量额外NVM写入从而降低系统性能和设备寿命的问题,并实现系统故障后的快速恢复,提高了系统性能和寿命。

附图说明

[0037] 图1为本发明具体实施时程序正常运行的流程框图。

[0038] 图2为本发明具体实施时系统故障恢复的流程框图。

具体实施方式

[0039] 下面结合附图,通过实施例,进一步阐述本发明,但不以任何方式限制本发明的范

围。

[0040] 本发明提供一种基于混合内存系统的故障快速恢复方法,所述混合内存系统为DRAM和NVM混合内存系统;进程主要在NVM栈空间中运行,因为NVM是断电非易失的,所以断电或者系统崩溃等故障在重启后才能快速恢复到进程运行的最新进度,不需要从头开始重新各种计算。DRAM则作为加速和减少NVM的寿命磨损。

[0041] 具体实施时,本发明为混合内存系统引入了新的函数限定符;将即将运行的应用程序的进程地址空间,包括代码段、数据段、运行时堆段和用户栈段映射到NVM栈空间中;提供给程序员一个新的函数限定符,程序在编译过程中,如果遇到该限定符修饰的函数,系统会在带有限定符的函数之前插入一条转移指令,CPU读取执行该转移指令后,会将该函数的执行转入到DRAM栈空间中。程序员可以将预计长时间运行或需要大量访存操作的函数,该函数预计会占据整个程序运行时间的很大比例,例如带有多重循环的复杂的迭代计算的函数,或者对一大块数组内容进行排序的函数等,声明为带有新限定符的函数;当该函数运行完毕,程序再回到NVM栈空间中运行

[0042] 即将运行的应用程序可以是一个图形渲染的程序,其中必然包含大量数据的矩阵乘法运算,程序员可以将完成该矩阵乘法的函数判定为长时间运行和需要大量访存操作的函数,将其声明为带有新限定符的函数;程序在编译过程中,如果遇到该限定符修饰的函数,系统会在带有限定符的函数之前插入一条转移指令,CPU读取执行该转移指令后,会将该函数的执行转入到DRAM栈空间中,当该函数执行完毕,程序再回到NVM栈空间中运行。

[0043] 本发明具体实施如下:

[0044] A.当程序正常运行时,执行如下操作(如图1所示):

[0045] A1.系统在编译过程中识别程序中的每个函数是否带有设定的限定符vlt,为带有设定的限定符的函数增加一条转移指令,该转移指令的作用是通知CPU将接下来的函数转到DRAM栈空间中执行,并设置CPU状态寄存器的某一空闲位;

[0046] A2.进程在NVM栈空间中启动运行,创建与进程相关的上下文,包括页表、页表基址寄存器、程序计数器等;

[0047] A3.当执行到新增的转移指令时,设置CPU标志寄存器中的一个当前空闲的标志位为1,比如PSW(Program Status Word,程序状态字)12～15位中的任意一位,表明接下来的函数在DRAM中执行;如果程序员没有为程序中的任何一个函数添加设定的限定符,则不会执行到新增的转移指令,执行步骤A7;

[0048] A4.保存进程上下文到NVM特定区域,特定区域是一段固定的NVM地址空间,每个进程分配一段,进程结束后则该段空间可再次被别的进程利用;

[0049] A5.拷贝函数参数值到DRAM栈空间,进程切换到DRAM栈空间中运行,在运行过程中,指令和数据仍然从NVM栈空间中读取;当需要写入数据时,数据先写入DRAM栈空间中,并且记录写入DRAM栈空间中数据的地址和本应写入NVM栈空间中数据地址的对应关系。

[0050] A6.采用原子性拷贝的方法拷贝DRAM栈空间中发生的修改以及函数返回值到NVM栈空间,将步骤A3中置1的CPU标志寄存器位置零,原子性拷贝可以保证该函数或者成功执行完毕,或者什么都没执行;实现原子性拷贝可以采用不同方式,比如写时复制、影子页等方式;拷贝的数据源地址和目的地址则通过A4中记录的映射对应关系确定;

[0051] A7.完成函数运行,进程在NVM栈空间中继续运行,完成vlt函数运行需要读取A4步

骤中保存的上下文,然后清理掉以备下一次vlt函数运行时所需;完成非vlt函数则不需要做任何事情,直接在NVM栈空间中继续运行;

[0052] A8.在NVM栈空间中运行如果再次遇到新增的转移指令则跳转至步骤A3,否则运行到进程结束,进程结束后清空保存该进程上下文的NVM特定区域。

[0053] B如果系统发生故障需要重启恢复运行时,执行如下操作(如图2所示):

[0054] B1.读取保存在NVM特定区域的进程上下文,程序恢复到故障之前最近的一次vlt函数调用处;

[0055] B2.判断该函数是否将成功在DRAM栈空间中运行完毕,如果是,则执行步骤B5,否则执行步骤B3;判断的方法可以有多种方式,比如检验该进程对应的NVM特定区域是否为空,非空则表明未成功,反之则成功,还可以通过检测新增指令设置的标志位,,该位为0说明成功,为1说明未成功;

[0056] B3.拷贝函数参数值到DRAM栈空间,进程切换到DRAM栈空间中运行,在运行过程中,指令和数据仍然从NVM栈空间中读取;当需要写入数据时,数据先写入DRAM栈空间中,并且记录写入DRAM栈空间中数据的地址和本应写入NVM栈空间中数据地址的对应关系;

[0057] B4.采用原子性拷贝方法拷贝DRAM栈空间中发生的修改以及函数返回值到NVM栈空间,将CPU标志寄存器的相应位置0,拷贝的数据源地址和目的地址则通过B3中记录的映射关系确定;

[0058] B5.恢复结束,程序继续正常运行。

[0059] 采用本发明方法,在运行带函数限定符的函数之前才会保存进程上下文,否则没有保存。因此,将一个程序中执行最慢的部分加入到一个带函数限定符的函数中,即可采用本发明提供的双栈运行方法,否则,没有双栈会降低系统性能和加快存储器的磨损。

[0060] 需要注意的是,公布实施例的目的在于帮助进一步理解本发明,但是本领域的技术人员可以理解:在不脱离本发明及所附权利要求的精神和范围内,各种替换和修改都是可能的。包括但不限于:为函数所引入的限定符、原子性拷贝的方式、判断DRAM中函数是否成功运行完毕的方式等。因此,本发明不应局限于实施例所公开的内容,本发明要求保护的范围以权利要求书界定的范围为准。

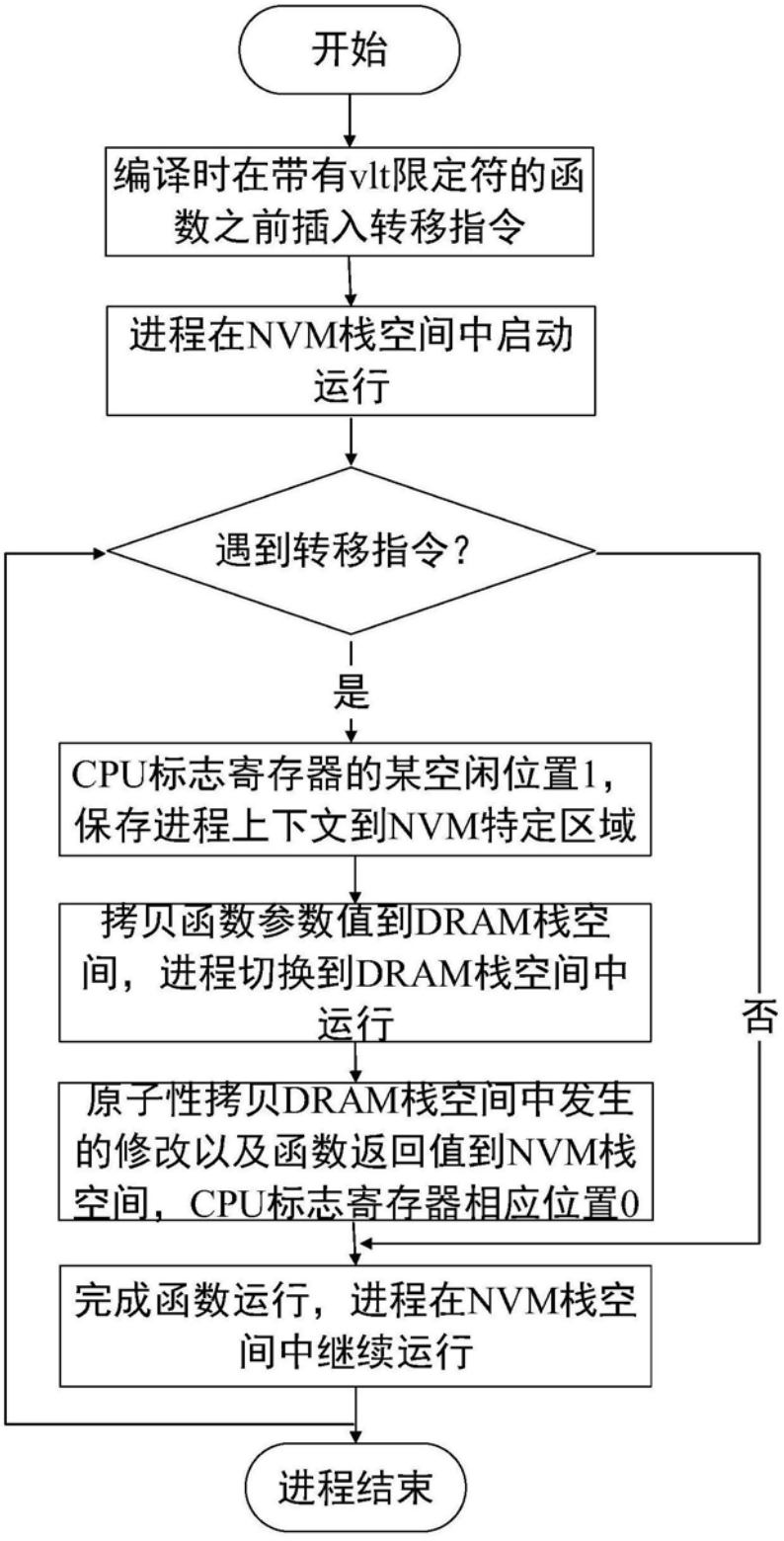

开始
编译时在带有vlt限定符的函数之前插入转移指令
进程在NVM栈空间中启动运行
遇到转移指令？
是
否
CPU标志寄存器的某空闲位置1，保存进程上下文到NVM特定区域
拷贝函数参数值到DRAM栈空间，进程切换到DRAM栈空间中运行
原子性拷贝DRAM栈空间中发生的修改以及函数返回值到NVM栈空间，CPU标志寄存器相应位置0
完成函数运行，进程在NVM栈空间中继续运行
进程结束

图1

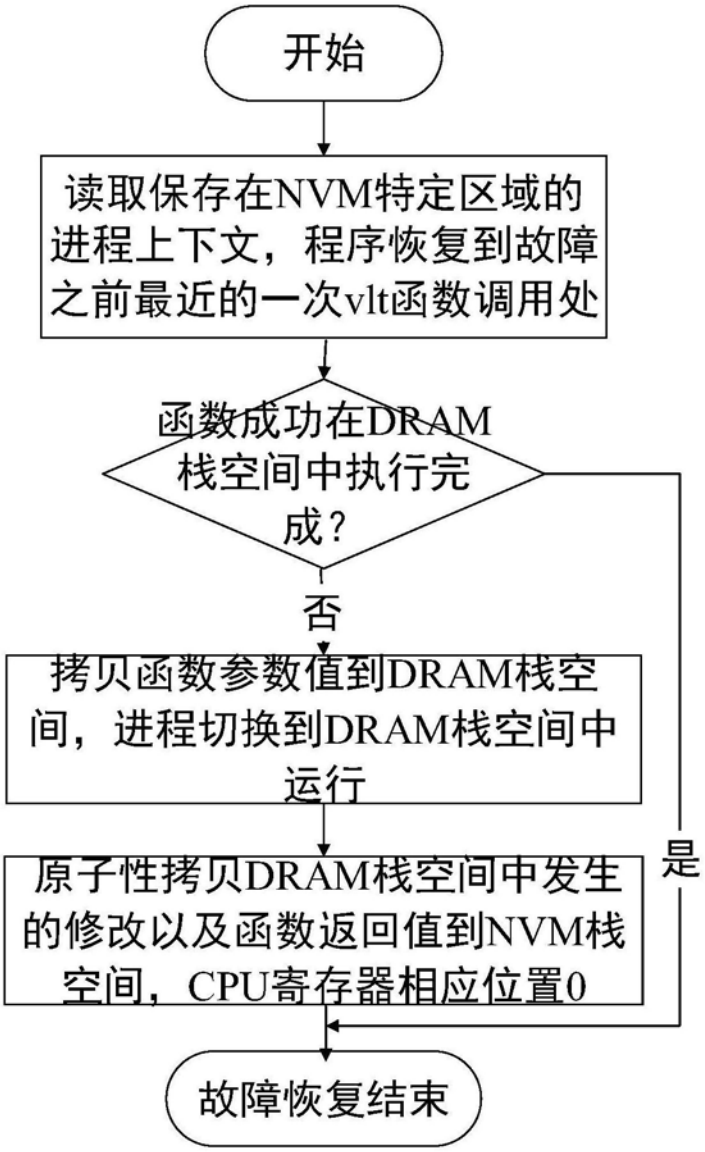

开始
读取保存在NVM特定区域的进程上下文，程序恢复到故障之前最近的一次vlt函数调用处
函数成功在DRAM栈空间中执行完成？
否
是
拷贝函数参数值到DRAM栈空间，进程切换到DRAM栈空间中运行
原子性拷贝DRAM栈空间中发生的修改以及函数返回值到NVM栈空间，CPU寄存器相应位置0
故障恢复结束

图2