(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 104714848 B
(45)授权公告日 2018.01.02

(21)申请号 201410681945.4

(22)申请日 2014.11.24

(65)同一申请的已公布的文献号
申请公布号 CN 104714848 A

(43)申请公布日 2015.06.17

(30)优先权数据
14/104,228 2013.12.12 US

(73)专利权人 国际商业机器公司
地址 美国纽约阿芒克

(72)发明人 F.Y.巴萨巴 M.K.格施温德
V.萨拉普拉 岑中龙

(74)专利代理机构 北京市柳沈律师事务所 11105
代理人 邸万奎

(51)Int.Cl.
G06F 9/50(2006.01)

(56)对比文件
US 2006149913 A1,2006.07.06,
US 2010205408 A1,2010.08.12,
US 2010162247 A1,2010.06.24,
CN 101317160 A,2008.12.03,
CN 101410797 A,2009.04.15,

审查员 杨龙

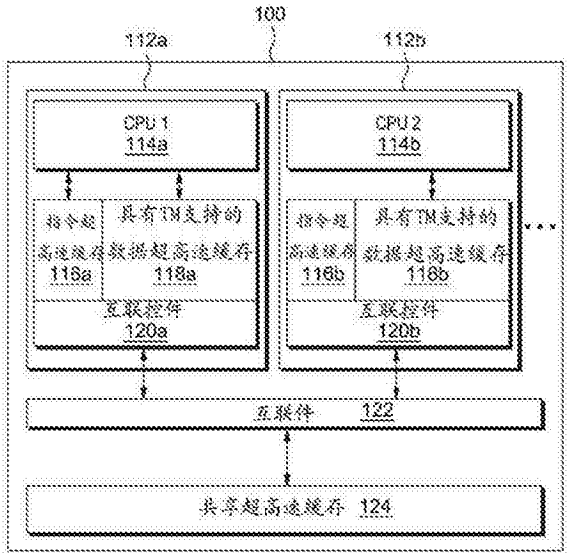
权利要求书2页 说明书42页 附图15页

(54)发明名称

利用用于聚结内存事务的指示的方法和系统

(57)摘要

本发明涉及一种事务内存系统，其利用用于聚结最外面事务的指示的计算机系统，该聚结使得针对第一事务的内存存储数据到内存的提交在第二事务执行(TX)末端处进行。事务内存系统的处理器执行一个或多个聚结指令，用于控制多个最外面事务的聚结。基于执行一个或多个聚结指令，处理器确定两个最外面事务是否将被聚结。基于确定两个最外面事务将被聚结，处理器将包括在多个最外面事务中的至少两个最外面事务聚结。



CN 104714848 B

1.一种利用用于聚结最外面事务的指示的计算机系统，该聚结使得针对第一事务的内存存储数据到内存的提交在第二事务的事务执行末端处进行，该计算机系统包括：

存储器；以及

处理器，与所述内存通信，其中该计算机系统被配置为执行一种方法，所述方法包括：

由处理器执行一个或多个聚结指令，用于控制多个最外面事务的聚结；

基于执行一个或多个聚结指令，由处理器至少基于被执行的所述一个或多个聚结指令设置的阈值以及基于两个最外面事务的聚结是否将超过所述阈值来确定两个最外面事务是否将被聚结，其中如果确定所述聚结将超过所述阈值，针对给定最外面事务禁止聚结具体最外面事务；以及

基于确定两个最外面事务将被聚结，由处理器将包括在多个最外面事务中的至少两个最外面事务聚结；

其中所述阈值表示以下之一：由将被聚结的最外面事务所展现的暂停、在将被聚结的两个最外面事务之间可能存在的指令的最大数量、处理被聚结的最外面事务所需的最大时间段、处理被聚结的最外面事务所需的资源量、能够被聚结最外面事务的最大数量、聚结导致中止状态的特定最外面事务的可容许实例的数量、以及先前经历过中止的被聚结事务的历史。

2.根据权利要求1所述的计算机系统，其中所述一个或多个聚结指令在被执行时表示哪个最外面事务能够被聚结。

3.根据权利要求1所述的计算机系统，其中所述一个或多个聚结指令包括与最外面事务的事务开始指令相关联的聚结前缀以及与最外面事务的事务开始指令相关联的聚结变量中的一个或两者。

4.根据权利要求1所述的计算机系统，其中所述一个或多个聚结指令包括与最外面事务的事务结束指令相关联的聚结前缀以及与最外面事务的事务结束指令相关联的聚结变量中的一个或两者。

5.根据权利要求1所述的计算机系统，所述方法还包括：

由处理器执行设置-事务-聚结-模式STCM指令，该STCM指令使得处理器进入STCM模式，其中进入STCM模式表示以下至少之一：(i)随后的聚结指令被启用以便使得事务将被聚结、(ii)能够存在于两个将被聚结的最外面事务之间的指令的最大数量、以及(iii)能够存在于两个将被聚结的最外面事务之间的指令的类型。

6.根据权利要求1所述的计算机系统，所述方法还包括：

执行重置-事务-聚结-模式RTCM指令，该RTCM指令使得处理器从STCM模式退出，其中从STCM模退出导致以下一个或多个：(i)停止执行随后的聚结指令、(ii)修改存在于两个最外面事务之间的指令的数量、以及(iii)修改存在于两个最外面事务之间的指令的类型。

7.一种利用用于聚结最外面内存事务的指示的方法，该聚结使得针对第一事务的内存存储数据到内存的提交在第二事务的事务执行末端处进行，该方法包括：

由处理器执行一个或多个聚结指令，用于控制多个最外面事务的聚结；

基于执行一个或多个聚结指令，由处理器至少基于被执行的所述一个或多个聚结指令设置的阈值以及基于两个最外面事务的聚结是否将超过所述阈值来确定两个最外面事务是否将被聚结，其中如果确定所述聚结将超过所述阈值，针对给定最外面事务禁止聚结具

体最外面事务;以及

基于确定两个最外面事务将被聚结,由处理器将包括在多个最外面事务中的至少两个最外面事务聚结;

其中所述阈值表示以下之一:由将被聚结的最外面事务所展现的暂停、在将被聚结的两个最外面事务之间可能存在的指令的最大数量、处理被聚结的最外面事务所需的最大时间段、处理被聚结的最外面事务所需的资源量、能够被聚结最外面事务的最大数量、聚结导致中止状态的特定最外面事务的可容许实例的数量、以及先前经历过中止的被聚结事务的历史。

8.根据权利要求7所述的方法,其中所述一个或多个聚结指令在被执行时表示哪个最外面事务能够被聚结。

9.根据权利要求7所述的方法,其中所述一个或多个聚结指令包括与最外面事务的事务开始指令相关联的聚结前缀以及与最外面事务的事务开始指令相关联的聚结变量中的一个或两者。

10.根据权利要求7所述的方法,其中所述一个或多个聚结指令包括与最外面事务的事务结束指令相关联的聚结前缀以及与最外面事务的事务结束指令相关联的聚结变量中的一个或两者。

11.根据权利要求7所述的方法,所述方法还包括:

由处理器执行设置-事务-聚结-模式STCM指令,该STCM指令使得处理器进入STCM模式,其中进入STCM模式表示以下至少之一:(i)随后的聚结指令被启用以便使得事务将被聚结、(ii)能够存在于两个将被聚结的最外面事务之间的指令的最大数量、以及(iii)能够存在于两个将被聚结的最外面事务之间的指令的类型。

12.根据权利要求7所述的方法,所述方法还包括:

执行重置-事务-聚结-模式RTCM指令,该RTCM指令使得处理器从STCM模式退出,其中从STCM模退出导致以下一个或多个:(i)停止执行随后的聚结指令、(ii)修改存在于两个最外面事务之间的指令的数量、以及(iii)修改存在于两个最外面事务之间的指令的类型。

利用用于聚结内存事务的指示的方法和系统

技术领域

[0001] 本公开总体涉及事务内存(TM)执行领域，并且更具体而言涉及修改用于将多个内存事务处理为单个事务的代码。

背景技术

[0002] 芯片上的中央处理单元(CPU)核的数量以及与共享内存相连的CPU核的数量继续显著增长以便支持日益增长的工作负载容量的需要。合作起来处理相同工作负载的CPU的日益增长的数量给软件可缩放性(scalability)带来的重大的负担，例如，由传统旗语(semaphore)保护的共享队列或数据结构成为热点并且导致子线性n路比例(scaling)曲线。

[0003] 传统上，这通过在软件中实现更好粒度锁定以及在硬件中采用更低延迟/更高带宽互联来对付。实现更好粒度锁定来改善软件可缩放性可能会非常复杂并且容易出错，并且在当今的CPU频率上，硬件互联的延迟受到芯片和系统的物理尺寸以及光速的限制。

[0004] 现在已经引入了硬件事务内存(HTM，或本讨论中简称为TM)的实现方式，其中一组指令—也称之为事务—以内存的数据结构上的原子方式运行(在其他文献中，原子操作也被称为“块并行(block concurrent)”或“串行化”)，如其它中央处理单元(CPU)以及I/O子系统所看到的。乐观地，事务可以在不获得锁定的情况下执行，但是，如果内存位置上的执行事务的操作与在相同内存位置上的另一个操作冲突，则可能需要中止或重试该事务执行。先前，已经提出了一些软件事务内存事项方式来支持人软件事务内存(TM)。但是，硬件TM也能够提供改善性能方案并且易于在软件TM上使用。

[0005] 在此通过引用整体包含在本申请中的、与2001年5月11日申请标题为“Nested Transactions in a File System(文件系统中的嵌套事务)”的美国专利US7,185,005教导了一些技术，其被提供用于在电子文件系统中执行作为嵌套事务的操作。根据一个方面，接收用于执行一个或多个文件系统操作的命令。响应于该命令，执行包括所述一个或多个文件系统操作的多个操作。执行所述多个操作包括(1)执行所述多个操作中的作为第一事务的部分的第一子集；以及(2)执行所述多个操作中的作为嵌套在第一事务中的第二事务的部分的第二子集。

[0006] 在此通过引用整体包含在本申请中的、与2005年12月30日申请标题为“software Assisted Nested Hardware Transactions(软件帮助的嵌套硬件事务)”的美国专利US7,730,286教导了一用于有效执行嵌套事务的方法和装置。提供了用于执行事务的硬件支持。此外，通过使用紧接于本地内存中的当前嵌套事务之前登录先前值以及存储与事务的层级相关联的处理器的堆栈，嵌套事务被潜在有效地执行。一旦在嵌套事务内有失败、中止、或无效事件/访问，在该嵌套事务的执行期间被写入的变脸或内存位置的状态则回滚到紧接于所述嵌套事务之前，而不是一直回到变量或内存位置在封入(enclosing)事务之前的原始状态。因此，可以在封入事务内重新执行嵌套事务，而无需弄平(flatten)所述封入的和嵌套的事务来重新执行所有事情。

发明内容

[0007] 本公开的实施例提供了一种用于事务内存系统的方法、计算机系统以及计算机程序产品，其用用于聚结最外面事务的指示。所述聚结使得针对第一事务的内存存储数据到内存的提交在第二事务的事务执行(TX)末端处进行。所述事务内存系统的处理器执行一个或多个聚结指令，用于控制多个最外面事务的聚结。基于执行一个或多个聚结指令，由处理器确定两个最外面事务是否将被聚结。基于确定两个最外面事务将被聚结，由处理器将包括在多个最外面事务中的至少两个最外面事务聚结。

附图说明

[0008] 本公开的实施例的一个或多个方面被具体指出并且被要求保护为在本说明书的结论处的权利要求书中的实例。本公开实施例的前述以及其它目的、特征以及优点根据结合附图所进行的下述描述而更清楚，其中：

[0009] 图1和2描绘了根据本公开的实施例的示例性多核事务内存环境的框图；

[0010] 图3描绘了包括根据本公开的实施例的实例CPU的实例组件的框图；

[0011] 图4图释了根据本公开的实施例的聚结控制器执行的操作活动，该聚结控制器控制一个最外面事务与另一个的聚结

[0012] 图5图释了根据本公开的实施例被执行用于聚结多个最外面事务的操作活动；

[0013] 图6图释了根据本公开的实施例的通过一种使用事务历史和标志来指导未来聚结活动的动态预测的操作活动；

[0014] 图7图释了根据本公开的实施例由识别和处理控制事务聚结活动的指示符的控制指示符执行的操作活动；

[0015] 图8图释了根据本公开的实施例由优化多个最外面事务的聚结以便使得性能增益最大化的聚结优化器执行操作活动；

[0016] 图9描绘了根据本公开的实施例支持聚结优化器的执行的Java®运行时间环境(JRE)的实例组件；

[0017] 图10描绘了根据本公开的实施例正在执行聚结控制、动态预测、控制指示符以及聚结优化器的计算设备的组件以及被执行用于最外面事务的聚结的操作活动的任何软件方面的框图；

[0018] 图11描绘了根据本公开的各个实施例的图10的一处理器的部分存在的各种硬件构件；

[0019] 图12描绘了根据本公开的实施例用于帮助读取器识别嵌套事务和聚结最外面事务之间的差别的事务嵌套的实例；

[0020] 图13图释了根据本公开的实施例由执行事务聚结处理的方面的动态编译器执行的操作活动的方法；

[0021] 图14描绘了根据本公开的实施例的流程图，其图释了利用用于聚结最外面事务指示的实施例；

[0022] 图15描绘了图释本公开的实施例的流程图；

[0023] 图16描绘了图释本公开的实施例的流程图；

[0024] 图17描绘了图释本公开的实施例的流程图；

[0025] 图18描绘了图释本公开的实施例的流程图；

[0026] 图19描绘了图释本公开的实施例的流程图；

[0027] 图20描绘了图释本公开的实施例的流程图；

[0028] 图21描绘了图释本公开的实施例的流程图；

[0029] 图22描绘了图释本公开的实施例的流程图。

具体实施方式

[0030] 从历史上看，计算机系统或处理其只具有单一处理器(也成为处理单元或中央处理单元)。处理器包括指令处理单元(IPU)、分支单元、内存控制单元等。这样的处理器能够一次执行程序的单个线程。现在开发了一些操作系统，其能够通过分派一个成组在一段时间内在处理器上执行而随后分派另一程序在另一段时间内在处理器上执行来对处理器进行时间共享(time-share)。随着技术的改进，经常将内存子系统超高速缓存器添加到处理器以及包括转换后备缓冲器(translation lookaside buffer(TLB))的复杂动态地址转换。IPU本身通常被称为处理器。随着技术的继续演进，整个处理器可以被包封在单一半导体芯片或裸片(die)中，诸如处理器可以被称为微处理器。随后开发的处理器包括多个IPU，这种处理器通常被称为多处理器。所述多处理器计算机系统(处理器)中的每个这种处理器可以包括单个或共享超高速缓冲器、内存接口、系统总线、地址转换机构等等。虚拟机和指令集架构(ISA)仿真器将一层软件添加给处理器，其按照单一硬件处理器中单一IPU的时间片的利用率为该虚拟机提供多个“虚拟处理器”(也称为处理器)。随着技术进一步演进，开发了多线程处理器，其使得具有单一多线程IPU的单个硬件处理器能够提供一种同时执行不同程序的线程的能力，因此，多线程处理器的每个线程对操作系统看起来像一个处理器。随着技术进一步演进，能够将多个处理器(每个都具有IPU)置于单个半导体芯片或裸片(die)上。这些处理器被称为处理器核或仅仅称为核。因此诸如例如处理器、中央处理单元、处理单元、微处理器、核、处理器核、处理器线程以及线程的术语通常被可互换地使用。在不脱离此处的教导的情况下，可以通过包括前面所示的那些术语的任何或所有处理器来实施此处实施例的方面。其中此处使用术语“线程”或“处理器线程”，期望的是，在处理器线程实施方式中可以具有该实施例的具体优点。

[0031] 基于Intel®的实施例中的事务执行

[0032] 在通过整体引用包含在此处的2012年2月披露的“Intel® Architecture Instruction Set Extensions Programming Reference(Intel®架构指令集扩展编程参考)”319433-012A中，第8章教导了多线程应用利用数量增加的CPU核来获得更高的性能。不过，多线程应用的写入需要编程者理解并考虑到在多线程之间的数据共享。对共享数据的存取通常需要同步机制。这些同步机制被用于通常通过使用受到锁定保护的关键部分确保多线程通过对被应用到共享数据的操作串行化更新共享数据。由于串行化限制并行性，程序员试图限制由于同步性导致的开销。

[0033] Intel®事务同步扩展(Intel® TSX)容许处理器动态确定线层是否必须通过锁定保护关键部分而被串行化，并且仅仅在需要的时候执行该船汉化。这使得处理器因为动态地

不必要的同步化而暴露和实施被隐藏在应用中的并行性。

[0034] 采用英特尔TSX，专用于程序员的代码区域(也称为“事务区域”或仅仅称为“事务”)被事务性地执行。如果事务执行成功完成，则在事务区域内执行的所有内存操作从其它处理器角度看将看起来已经即时发生。处理器使得在事务区域内执行的被执行事务的内存操作，仅仅在成功提交发生时，即，在所述事务成功地完成执行时，对其它处理器是可见的。该处理通常被成为原子提交(commit)。

[0035] 因特尔TSX提供两个软件接口来指定用于事务执行的代码的区域。硬件锁省略(Hardware Lock Elision(HLE))是一种老式(legacy)可兼容指令集扩展(包括XACQUIRE和XRELEASE前缀)，用于指定事务区域。受限事务内存(Restricted Transactional Memory(RTM)是一种用于程序员以比可能采用HLE更灵活的方式限定事务区域的新指令集接口(包括XBEGIN、XEND以及XABORT指令)。所述HLE用于更喜欢普通互斥编程模型的向后兼容性并且愿意在老式硬件上运行HLE使能软件并且也愿意在具有HLE支持的硬件上利用新锁定省略能力的程序员。RTM用于更喜欢灵活接口而不喜欢事务执行硬件的程序员。此外，因特尔TSX还提供了XTEST指令。该指令容许软件查询逻辑处理器是否正在由HLE或RTM所识别的事务区域中执行。

[0036] 由于成功的事务执行确保原子(atomic)提交，处理器在没有明确同步的情况下最有利地执行代码区域。如果对该具体执行而言同步没有必要，则执行可以在没有任何交叉线程(cross-thread)串行化的情况下提交。如果处理器不能原子化地提交，则该乐观(optimistic)执行失败。当发生这种情况时，处理器将回滚该执行，一种被称为事务中止的处理。在事务中止时，处理器将放弃在由该事务所使用的内存区域内执行的所有更新，恢复架构状态以便看起来就像从来没有发生过最有利执行一样，并且非事务性地继续开始执行。

[0037] 处理器可能会因为各种原因而执行事务中止。中止事务的主要原因是源于在正在事务地执行的逻辑处理器和另一个逻辑处理器之间的冲突内存访问。这种冲突内存访问会防止成功事务执行。在事务区域内的从其中读取的内存地址构成事务区域的读取集而事务区域内向其中写入的地址构成该事务区域的写入集。因特尔TSX以超高速缓存线的粒度维持所述读取集和写入集。如果另一个逻辑处理器读取作为事务区域的写入集的部分的位置或者写入作为事务区域的读取集或写入集的部分的位置，则出现冲突内存访问。冲突访问通常意味着对该代码区域需要串行化。由于英特尔TSX以超高速缓存线的粒度检测数据冲突，因此被置于相同超高速缓存线中的不相关数据位置将被检测为冲突，其将导致事务中止。事务中止也会由于受限的事务资源的出现。例如，在区域中的被访问的数据量可能被超过专用实施方式的容量。此外，指令和系统事件可能会导致事务中止。频繁的事务中止导致循环浪费和效率日益低下。

[0038] 硬件锁省略

[0039] 硬件锁省略(HLE)为程序员提供一种老式可兼容指令集接口以便使用事务执行。HLE提供两个新指令前缀提示(prefix hint)：XACQUIRE和XRELEASE.

[0040] 采用HLE，程序员将XACQUIRE前缀添加到指令的前部，其被用来获取正在保护关键部分的锁。处理器将所述前缀当作提示(hint)以便省略所述锁获取操作。即使所述锁获取具有对该锁的相关联写入操作，该处理器既不将所述锁的地址添加到所述事务区域的写入

集，其也不向该锁发出任何写入请求。相反，该所的地址被添加到所述读取集。逻辑处理器进入事务执行。如果在加有前缀XACQUIRE指令之前可以获得所述锁，那么所有其他处理器之后将继续将该锁视为可用。由于所述正事务地执行的逻辑处理器既不将所述锁的地址添加到其写入集也不对该锁外部地执行可视写入操作，因此其它逻辑处理器能够读取该锁而不导致数据冲突。这使得其它逻辑处理器也进入和并行执行受到所述锁保护的关键部分。处理器自动检测在事务执行期间发生的任何数据冲突并且将在必要时执行事务中止。

[0041] 尽管所述省略的处理器不对所述锁执行任何外部写入操作，但是该硬件确保在该锁上的操作的程序顺序。如果该省略的处理器自身读取所述锁在该关键(critical)部分中的值，其将显得就像该处理器已经获取了所述锁一样，即，该读取将返回非省略值。这种特性使得HLE执行在功能上等同于没有HLE前缀的执行。

[0042] XRELEASE前缀能够被加载指令的前部，其被用于释放保护关键部分的锁。释放所述锁包括对锁的写入。如果该指令将恢复该锁的值到该锁在对同一锁的加有前缀XACQUIRE的锁获取操作之前所具有的值，则处理器省略与该锁的释放相关联的外部写入请求，并且不将该锁的地址添加到所述写入集。随后处理器试图提交所述事务执行。

[0043] 采用HLE，如果多线程执行受到相同锁保护的关键部分但是它们没有对彼此的数据执行任何冲突操作，则这些线程能够并行执行并且无需串行化。即使软件对共用锁使用锁获取操作，硬件也会识别到这一点、省略该锁、并且对两个线程执行关键部分而不通过该锁要求任何通讯—如果动态地该通讯不是必要的话。

[0044] 如果处理器不能事务地执行该区域，则该处理器将非事务地执行该区域并且没有省略。能HLE的软件具有与底层基于非HLE锁的执行相同的转发进度(forward progress)保证。对于成功HLE执行，所述锁和所述关键部分必须遵循某些指南。这些指南仅仅影响性能；并且没有遵循这些指南将不会导致功能性故障。没有HLE支持的硬件将忽略XACQUIRE和XRELEASE前缀提示并且将不执行任何省略，因为这些前缀对应于REPNE/REPE IA-32前缀，其在其中XACQUIRE和XRELEASE有效的指令上被忽略。重要的是，HLE与现有基于锁的编程模型兼容。提示的适当使用将不会导致功能性程序错误(bug)，尽管可能暴露已经在代码中的潜在程序错误。

[0045] 受限的事务内存(RTM)提供用于事务执行的灵活软件接口。RTM为程序员提供三个新指令—XBEGIN、XEND以及XABORT—以便启动、提交以及中止事务执行。

[0046] 程序员使用XBEGIN指令来指定事务代码区域的开头以及使用XEND指令指定事务代码区域的结尾。如果RTM区域不能被事务性地成功执行，则XBEGIN指令获取向回落指令地址提供相对偏移的操作数。

[0047] 处理器可以因为多种原因而中止RTM事务执行。在许多情况下，硬件自动检测事务中止条件并从具有与在XBEGIN指令开始处呈现的架构状态对应的架构状态以及被更新以描述中止状况的EAX寄存器的回落指令地址重启执行。

[0048] XABORT指令使得程序员清楚地中止RTM区域的执行。XABORT指令采取8比特中间变元(immediate argument)，其被加载到EAX寄存器并因此可用于跟随在RTM中止之后的软件。RTM指令不具有任何与它们相关联的数据内存位置。尽管硬件不就RTM区域是否将在任何时候事务性地成功提交提供保证，但是期望跟随在推荐的指南之后的大部分事务将事务性地成功提交。不过，程序员必须总是在回落路径中提供可替代代码序列以便保证转发进

度。这可能与获取锁和非事务性地执行指定代码区域一样简单。而且，总是在给定实施方式上中止的事务可能在未来的实施方式上事务性地完成。因此，程序员必须确保用于事务区域的代码路径以及替代性代码序列在功能上得到测试。

[0049] HLE支持的检测。

[0050] 如果CPUID.07H.EBX.HLE[比特4]＝1，则处理器支持HLE执行。不过，应用可以使用HLE前缀(XACQUIRE和XRELEASE)而不核查处理器是否支持HLE。没有HLE支持的处理器忽略这些前缀并将执行所述代码而不进入事务执行。

[0051] RTM支持检测

[0052] 如果CPUID.07H.EBX.RTM[比特11]＝1，则处理器执行RTM执行。在处理器使用RTM指令(XBEGIN、XEND、XABORT)之前处理器必须核实该处理器是否支持RTM。这些指令在被用于不支持RTM的处理器上时将生成#UD例外。

[0053] XTEST指令的检测

[0054] 如果处理器支持HLE或RTM则处理器支持XTEST指令。应用在使用XTEST指令之前必须核实这些特征标志的任何一个。该指令在被用于不支持HLE或RTM的处理器上时将生成#UD例外。

[0055] 查询事务执行状况

[0056] XTEST指令能够被用于确定由HLE或RTM所指定的事务区域的事务状况。要注意的是，尽管HLE前缀在不支持HLE的处理器上被忽略，但是XTEST指令将在被用于不支持HLE或RTM的处理器上时将生成#UD例外。

[0057] HLE锁的要求

[0058] 对于要成功地事务性地提交的HLE执行，所述锁必须满足某些特性并且对该锁的访问必须遵循某些指南。

[0059] 带有前缀XRELEASE的指令必须将所省略的锁的值恢复到其在所述所获取之前所具有的值。这使得硬件能够通过不将他们添加到所述写入集而安全地省略锁。所述锁释放(带有前缀XRELEASE)指令的数据尺寸和数据地址必须匹配所述锁获取(带有前缀XACQUIRE)的数据尺寸和数据地址并且所述锁不必跨过超高速缓存线边界。

[0060] 软件不应采用有XRELEASE前缀的指令之外的任何指令向事务HLE区域内的所省略的锁写入，否则这种写入会导致事务中止。此外，递归锁(其中，线程在没有第一次释放所述锁的情况下多次获取同一锁)也可以导致事务中止。需要注意的是，软件能够观察到关键部分内所省略锁获取的结果。这种线程操作将所述写入的值返回到所述锁。

[0061] 所述处理器自动检测对这些指南的违反情况，并且在不省略的情况下安全地过度到非事务执行。由于英特尔TSX以超高缓冲线的粒度来检测冲突，因此可以通过省略同一锁的其他逻辑处理器将对共同位于与所省略锁相同的超高速缓冲线上的数据的写入检测为数据冲突。

[0062] 事务嵌套

[0063] HLE和RTM两者都支持嵌套事务区域。不过，事务中止将状态恢复到启动事务执行的操作：或者是最外面有前缀XACQUIRE的HLE合格指令或者是最外面XBEGIN指令。所述处理器将所有嵌套事务当作一个事务。

[0064] HLE嵌套和省略

[0065] 程序员可以嵌套HLE区域直到MAX_HLE_NEST_COUNT的具体实现深度。每个逻辑处理器内部地跟踪该嵌套计数，但是该计数不能用于软件。有前缀XACQUIRE的符合HLE的指令使得嵌套技术递增，并且有前缀XRELEASE的符合HLE的指令使其递减。逻辑处理器在该嵌套计数从零变为一时进入事务执行。该逻辑处理器仅仅在嵌套技术变成零时尝试提交。如果该嵌套计数超过MAX_HLE_NEST_COUNT。

[0066] 除了支持嵌套的HLE区域，该处理器还可以省略多个嵌套锁。该处理器跟踪用于省略的锁，所述省略对于该锁始于有前缀XACQUIRE的符合HLE的指令并且对于同一锁结束于有前缀XRELEASE的符合HLE的指令。在任何一次，该处理器能够跟踪直到MAX_HLE_ELIDED_LOCKS数量个锁定。例如，如果该实施方式支持的MAX_HLE_ELIDED_LOCKS值为二并且如果程序员嵌套三个HLE标识的关键部分(通过对三个不同锁执行有前缀XACQUIRE的符合HLE的指令而不对这些锁的任意一个执行插入的有前缀XRELEASE的符合HLE的指令)，那么前两个锁定将被省略，但是第三个锁定将不会被省略(但是将会被添加到事务的写入集)。不过，该执行依然事务性地继续。一旦遇到用于两个被省略的锁之一的XRELEASE，通过有前缀XACQUIRE的符合HLE的指令获得的随后的锁将被省略。

[0067] 当所有省略的成对XACQUIRE和XRELEASE已经得到匹配时该处理器尝试提交HLE执行，嵌套计数变为零，并且这些锁满足要求。如果执行不能原子地提交，那么执行过渡到非事务执行而没有省略，就像第一指令没有前缀XACQUIRE一样。

[0068] RTM嵌套

[0069] 程序员可以嵌套RTM区域直到具体实施方式MAX_RTM_NEST_COUNT。逻辑处理器内部地跟踪该嵌套计数，但是该计数对软件不可用。XBEGIN指令使得该嵌套计数递增而XEND指令使得该嵌套计数递减。仅仅在嵌套计数变为零时，该逻辑处理器尝试提交如果嵌套计数超过MAX_RTM_NEST_COUNT，则出现事务中止。

[0070] 嵌套的HLE和RTM

[0071] HLE和RTM提供两个替换软件接口给公共事务执行能力。当HLE和RTM被一起嵌套时，例如HLE在RTM内或者RTM在HLE内，事务处理性能为专用实施方式的。不过，在所有情况下，该实施方式将维持HLE和RTM语义。一种实施方式在被用于RTM区域内时可以选择忽略HLE提示，并且在RTM指令被用于HLE区域内时可以导致事务中止。在后一种情况下，因为处理器将重新执行该HLE区域而不实际进行省略，并且随后执行RTM指令，因此从事务性到非事务性执行的过渡无缝进行。

[0072] 中止状态定义

[0073] RTM使用EAX寄存器将中止状态通讯告知软件。在RTM中止之后，EAX寄存器具有如下定义。

[0074] 表1

[0075]

RTM 中止状态定义

EAX 寄存器比特位置	含义
0	如果中止由 XABORT 指令导致则设置
1	如果设置，则事务可以成功重试，如果比特 0 被设置，则该比特则总是清空
2	如果另一逻辑处理器与作为中止的事务的部分的内存地址冲突，则设置
3	如果内缓冲器溢出，则设置
4	如果命中调试断点，则设置
5	如果在执行嵌套事务期间出现中止，则设置
23:6	保留
31-24	XABORT 变量（如果比特 0 被设置则有效，否则，保留）

[0076] 用于RTM的EAX中止状态仅仅提供了中止的原因。其自身不编码对RTM区域是出现中止还是提交。在RTM中止之后,EAX的值可以为0。例如,CPUID指令在被用于RTM区域之内时导致事务中止并且不会满足用于设置任何EAX比特的要求。这可能导致EAX值为0。

[0077] RTM内存排序

[0078] 成功的RTM提交使得RTM区域内的所有内存操作看起来像原子地执行。由后面跟随有XEND的XGEGIN构成的所成功提交的RTM区域,即使在RTM区域内没有内存操作,具有与LOCK前缀指令相同的排序语义。

[0079] XBEGIN指令不具有屏蔽(fencing)语义。不过,如果RTM执行中止,则来自RTM区域内的所有内存更新被丢弃并且不会变得对任何其他逻辑处理器可见。

[0080] 启用RTM的调试器支持。

[0081] 在默认情况下,在RTM区域内部的任何调试异常(exception)都将导致事务中止,并且将将控制流重新定向到具有恢复的架构状态和EAX集中的比特4的回落指令地址。不过,为了使得软件调试器(debugger)截取对调试异常的执行,RTM架构提供附加能力。

[0082] 如果DR7的比特11和IA32_DEBUGCTL_MSR的比特15都是1,则由于调制异常(#DB)或断点异常(#BP)导致的RTM中止使得执行回滚并且从XBEGIN执行令而不是回落地址开始重新开始。在这种情形中,EAX寄存器也将被恢复回到XBEGIN指令的点。

[0083] 编程考虑

[0084] 通常程序员标识的区域被期望事务性地执行和成功提交。不过英特尔TSX不提供

任何这种保证。事务性执行科恩能够由于多种原因而中止。为了完全利用事务性能力。程序员应该遵循某些指南来增加他们事务性执行成功提交的概率。

[0085] 该部分阐述了可能导致事务性中止的各种事件。该架构确保在随后中止执行的事务内执行的更新将永不可见。只有被提交的事务性执行发起对架构状态的更新。事务性中止从来不导致功能故障并且仅仅影响性能。

[0086] 基于指令的考虑

[0087] 程序员可以在事务(HLE或RTM)内安全地使用任何指令，并且可以在任何特权级使用事务。不过，有些指令将总是中止事务性执行并且导致执行无缝地并且安全地过渡到非事务性路径。

[0088] 英特尔TSX考虑到将在事务内使用的最普通的指令而不会导致中止。事务内的下面的操作通常不会导致中止：

[0089] • 对执令指针寄存器、通用寄存器(GPR)以及状态标志(CF、OF、SF、PF、AF以及ZF)的操作；以及

[0090] • 对XMM和YMM寄存器以及MXCSR寄存器的操作register.

[0091] 不过，当在事务性区域内内部混用(intermixing)SSE和AVX操作时，程序员必须小心。混用存取XMM寄存器的SSE指令和存取YMM寄存器的AVX指令会导致事务中止。程序员可以在一些事务内使用有前缀REP/REPNE串(string)操作。不过，长的串会导致中止。而且，如果CLD和STD指令改变DF标志的值，则使用CLD和STD指令会导致中止。不过，如果DF为1，则STD指令将不会导致中止。类似地，如果DF为0，则CLD指令将不会导致中止。

[0092] 此处没有列举的在事务内被使用时导致中止的指令通常不导致事务中止(实例包括但不限于MFENCE、LFENCE、SFENCE、RDTSC、RDTSCP等)。

[0093] 下面的指令将使得位于任何实施方式上的事务性执行中止：

[0094] • XABORT

[0095] • CPUID

[0096] • PAUSE

[0097] 此外，在有些实施方式中，下面的指令总是可以导致事务性中止。这些指令不期望被通常使用在典型事务性区域。不过，程序员不必依赖这些指令来迫使事务性中止，因为它们是否导致事务中止是依赖于实施方式的。

[0098] • 对X87和MMX架构状态的操作。这包括所有MMX和X87指令，包括FXRSTOR和FXSAVE指令。

[0099] • 更新EFLAGS的非状态部分：CLI、STI、POPFD、POPFQ、CLTS。

[0100] • Instructions that update segment registers,debug registers and/or control registers:MOV to DS/ES/FS/GS/SS,POP DS/ES/FS/GS/SS,LDS,LES,LFS,LGS,LSS,SWAPGS,WRFSBASE,WRGSBASE,LGDT,SGDT,LIDT,SIDT,LLDT,SLDT,LTR,STR,Far CALL,Far JMP,Far RET,IRET,MOV to DRx,MOV to CR0/CR2/CR3/CR4/CR8and LMSW.

[0101] • 环形(Ring)过渡：SYSENTER、SYSCALL、SYSEXIT以及SYSRET.

[0102] • TLB和超高速缓存能力控制：CLFLUSH、INVD、WBINVD、INVLPG、INVPCID、以及具有非时间性提示(MOVNTDQA、MOVNTDQ、MOVNTI、MOVNTPD、MOVNTPS以及MOVNTQ)的内存指令。

[0103] • 处理器状态保存：XSAVE、XSAVEOPT以及XRSTOR。

[0104] • 中断：INTn、INTO。

[0105] • IO：IN、INS、REP INS、OUT、OUTS、REP OUTS以及它们的变化形式。

[0106] • VMX：VMPTRLD、VMPTRST、VMCLEAR、VMREAD、VMWRITE、VMCALL、VMLAUNCH、VMRESUME、VMXOFF、VMXON、INVEPT以及INVVPID。

[0107] • SMX：GETSEC。

[0108] • UD2、RSM、RDMSR、WRMSR、HLT、MONITOR、MWAIT、XSETBV、VZEROUPPER、MASKMOVQ以及V/MASKMOVDQU。运行时间考虑

[0109] 除了基于指令的考虑，运行时间事件可能导致事务性执行中止。这些可能由于数据存取模式或微架构实施方式特征导致。下面的列表不是所有中止原因的综合阐述。

[0110] 必须暴露给软件的事务中的所有缺陷或陷阱将得到抑制。事务性执行将中止并且执行将过渡到非事务性执行，就像该缺陷或陷阱从来未出现过一样。如果异常被遮掩，那么未被遮掩的一场将会导致事务性中止并且该状态将看起来就像该异常从来未出现过一样。

[0111] 在事务性执行期间出现的同步异常事件(#DE、#OF、#NP、#SS、#GP、#BR、#UD、#AC、#XF、#PF、#NM、#TS、#MF、#DB、#BP/INT3)可能导致执行非事务性地提交，并且要求非事务性执行。这些事件受到抑制就像它们从来没有出现过一样。采用HLE，由于非事务性代码路径与事务性代码路径一致，这些事件通常在导致异常的指令被非事务性地重新执行时将再次显现，使得相关联的同步时间在非事务性地执行中被适当地传送。在事务性执行期间发生的异步事件(NMI、SMI、INTR、IPI、PMI等等)可以使得所述事务性执行中止并且过渡到非事务性执行。异步事件将待决(pend)并且在所述事务中止被处理后被处理。

[0112] 事务仅支持写回可超高速缓存存储类型操作。事如果事务包括关于任何其他存储类型的操作，则事务可总是中止。这包括UC存储类型的指令取得。

[0113] 在事务性区域内的内存存取可以要求处理器设置所引用页表输入项的被存取和脏的标志。处理器如何处理这种情况的特性(behavior)是具体实现的。即使该事务性区域随后被中止，有些实施方式可以容许对这些标志更新以便可为外部所见。如果这些标志需要被更新，有些英特尔TSX实施方式可以选择中止事务性执行。而且，处理器的页表遍历(walk)可以生成对其自身事务性地被写入的但是非被提交状态的存取。有英特尔(Intel)TSX实施方式可以选择中止在这种情形中的事务性区域的执行。无论如何，该架构通过诸如TLB的结构的特性确保在事务性区域中止的情况下将不会使得该被事务性地写入的状态在架构上可见。

[0114] 事务性地执行自我修改的代码也可以导致事务性中止。即使在应用HLE和RTM时，程序员也必须继续遵循用于写入自我修改和交叉修改代码的英特尔推荐的指南。景观RTM和HLE的实施方式将通常提供足够的资源用于执行公共事务性区域，但是用于事务性区域的实施方式约束和估量尺寸可以导致事务性执行中止并过渡到非事务性执行。该架构不提供进行事务性执行的资源量表征并且不保证事务性执行一直成功。

[0115] 对在事务性区域内的被存取的超高速缓存线的冲突请求可以防止食物成功地执行。例如，如果逻辑处理器P0读取事务性区域中的线A并且另一个逻辑处理器P1写入线A(在事务性区域之内或之外)，那么，如果逻辑处理器P1的写入干扰处理器P0的事务性地执行的能力，逻辑处理器P0可中止。

[0116] 类似地，如果P0写入事务性区域中的线A并且P1读取或写入线A(在事务性区域之

内或之外)，那么，如果P1对线A的存取干扰P0的事务性地执行的能力，P0可以中止。此外，其他一致流通量(coherence traffic)可多次显现为冲突请求并且可以导致中止。尽管这些假冲突会发生，期望它们是非共同的。确定P0或P1中止的冲突解决策略是具体实现的(implementation specific)。

[0117] 一般性事务执行实施例：

[0118] 根据通过引用被整体包含在本公开中“ARCHITECTURES FOR TRANSACTIONAL MEMORY(用于事务性内存的架构)”，一份由Austen McDonald(奥斯丁.麦克唐纳)在部分申请博士学位的实现中于2009年6月递交给斯坦福大学的计算机科学部和毕业研究委员会(Department of Computer Science and the Committee on Graduate Studies of Stanford University)的论文，要实现原子和隔离事务性区域需要三种机制：版本控制(versioning)、冲突检测以及取数碰头(contention)管理。

[0119] 为了使得事务性代码区域显得原子(atomic)，由该事务性代码区域执行的修改必须被存储并且被保持与其他事务隔离直到提交时间为止。该系统通过实现版本控制策略来实现这一点。两个版本控制范例为：急切(eager)与懒惰(lazy)。急切版本控制系统就地存储新生成的事务性值并且在被称为取消-日志(undo-log)的一侧存储前一内存值。懒惰版本控制系统在被称为写入缓冲器的位置临时存储新值，紧在提交时将它们拷贝到内存。在另一个系统中，超高速缓存被用于优化新版本的存储。

[0120] 为了保全事务显现为被原子地执行，必须检测和解决冲突。连个系统，即，急切和懒惰版本控制系统，通过将冲突检测策略实现为悲观的或乐观的来检测冲突。乐观系统并行地执行事务，仅仅在事务提交时核实冲突。悲观系统在每次加载和存储时核实冲突。与版本控制类似，冲突检测也使用该超高速缓存，将每条线标记为读取集的部分或写入集的部分，或者标记为两者。两个系统通过实现取数碰撞管理策略来解决冲突。现有许多取数碰撞管理策略，有些更适用于乐观冲突检测而有些更适用于悲观冲突检测。下面描述一些实例策略。

[0121] 由于每个事务性内存(TM)系统需要版本控制检测和冲突检测，因此这些选择导致四个不同的TM设计：急切-悲观(EP)、急切-乐观(EO)、懒惰-悲观(LP)以及懒惰-乐观(LO)。表2简要描述了所有四种不同的TM设计。

[0122] 图1和2描绘了根据本公开的实施例的实例多核事务性内存环境的示意框图。图1显示了在一个裸片100上的多个启用TM的CPU(CPU1 114a、CPU2 114b等)，在互联控件120a、120b的管理下与互联件122连接。每个CPU 114a、114b(也成为处理器)可以具有分离(split)超高速缓存，其由用于高速缓存将被执行的来自内存的指令的指令超高速缓存116a、166b和用于高速缓存将由CPU 114a、114b(在图1中，每个CPU 114a、114b以及其相关的超高速缓存都被称为112a、112b)操作的内存位置的数据(操作数)的具有TM支持的数据超高速缓存118a，118b构成。在一种实施方式中，多个裸片的超高速缓存，诸如多裸片100c，相互连接以便支持裸片100的多个实例的超高速缓存之间的超高速缓存相关性。在一种实施例方式中，单个超高速缓存，而不是分离超高速缓存，被用来保持指令和数据。在一些实施方式中，CPU超高速缓存在层级超高速缓存结构中为一级超高速缓存。例如，每个裸片100可以使用共享超高速缓存124以便在裸片100上的所有CPU之间被共享。在另一个实施方式中，裸片100可以存取公共共享超高速缓存124，其在裸片100的多个实例的所有处理器之间

被共享。

[0123] 图2显示了具有CPU 114的实例事务性CPU环境112的细节，包括支持TM的一些附加部件。事务性CPU(处理器)114包括用于支持寄存器检验点126的硬件以及特定TM寄存器128。所述事务性CPU超高速缓存不仅可以具有传统超高速缓存的MESI比特130、标签140以及数据142，还还以具有例如显示线已经被CPU 114读取同时将执行事务的R比特132以及显示线已经被CPU 114写入同时将执行事务的W比特138。

[0124] 在TM系统中对于程序员而言的关键细节是非事务性存取如何与事务交互。通过设计，事务性存取使用上述机制被彼此筛选(screen)。不过，依然必须考虑在正规的非事务性载荷与含有用于该地址的新值的事务之间的交互。此外，也必须探究在非事务性存储与已经读取该地址的事务之间的交互。这些是数据库概念隔离的问题(issue)。

[0125] 当每个非事务性载荷和存储像原子事务一样动作时，据说TM系统实现强隔离、有时候被称为强原子性。因此非事务性载荷不能看见未被提交的数据并且在已经读取该地址的许多事务中非事务性存储导致原子性违反。据说不是这种情况的系统可以实现弱隔离，有时被称之为弱原子性。

[0126] 强隔离通常比弱隔离更理想，因为强隔离的概念化(conceptualization)和实现相对容易。此外，如果程序员已经忘记了要采用事务包围有些共享内存，导致程序缺陷，那么，采用强隔离，程序员通常将使用简单调试接口检测该疏漏，因为程序员将会看到导致原子性违反的非事务性区域。而且，以一种模型写出的程序可以在另一种模型上不同地工作。

[0127] 而且，强隔离通常比弱隔离更容易在硬件上支持TM。采用强隔离，由于相关协议已经管理在处理器之间的载荷和存储通讯，事务能够检测非事务性载荷和存储，并且适当地动作。为了在软件事务内存(TM)中实现强隔离，非事务性代码必须被修改以便包括读屏障和写屏障，潜在削弱(cripple)性能。尽管已经花费了巨大的努力来消除一些不必要的屏障，但是这种技术通常比较复杂并且通常性能远低于HTM的性能。

[0128] 表2

[0129]

事务性内存设计空间

冲突检测	版本控制：懒惰	版本控制：急切
乐观	在写缓冲器中存储更新；在提交时检测冲突。	未实践：等待更新内存直到提交时为止，但是在存取时检测冲突保证被浪费的工作并且不提供优点。
悲观	在写缓冲器中存储更新；在存取时检测冲突。	更新内存，在取消-日志中保持旧值；在存取时检测冲突。

[0130] 表2图释了事务性内存的基本设计空间(版本控制和冲突检测)

[0131] 急切-悲观(EP)

[0132] 下面描述的该第一TM设计被称为急切-悲观。EP系统“就地(in place)”(因此，名称为“急切”)存储其写入集，并且支持回滚，将被覆写线的旧值存储到“取消日志(undo

log)”。处理器使用W 138和R 132超高速缓存比特跟踪读取集和写入集并且在接收到所探听到的载荷请求时检测冲突。在已知的文献中EP系统的可能最著名实例是LogTM和UTM。

[0133] 在EP系统中开始事务与在其他系统中开始事务非常像:tm_begin()采取寄存器检查点并且初始化任何状态寄存器。EP系统还要求初始化取消日志,其细节依赖于该日志格式,但是通常涉及初始化日志基础指针到预分配的线程专有存储器的区域以及清除日志界限寄存器。

[0134] 版本控制(versioning):在EP中,由于急切版本控制被涉及为其作用的方式,多半让MESI 130状态过渡(与修改、独占、共享以及无效代码状态对应的超高速缓存线指示符)不改变。在事务之外,让MESI 130状态过渡完全改变。当读取事务之内的线时,标准相关性过渡不仅施加(S(共享)→S、I(无效)→S、或I→E(独占)),根据需要发出加载缺失(load miss),而且R 132比特也被设置。同样,写入线不仅施加标准过渡(S→M、E→II→M),根据需要发出缺失(miss),而且设置W 138(被写)比特。线被第一次写入,整个线的旧版本被加载,然后被写入取消日志,以便在当前事务中止的情况下保藏(preserve)它。新写入的数据随后被“就地”存储在旧数据之上。

[0135] 冲突检测:悲观冲突检测使用在缺失上交换的相关性消息、或者进行升级,以便寻找事务之间的冲突。当在事务内出现读取缺失时,其它处理器接收到记载请求;但是如果它们没有所需的线,它们将忽略该请求。如果其它处理器非推测地(non-speculatively)具有所需要的线或者具有线R 132(读取),则它们将该线降级到S,并且在某些情况下如果它们具有在MESI 130的M或E状态的线则发出超高速缓存-到-超高速缓存的传送。不过,如果该超高速缓存具有线W 138,则检测到在两个事务之间的冲突并采取附加动作。

[0136] 类似地,当事务试图将线从共享升级为修改(对第一写入),该事务发出独占加载请求,其也被用来检测冲突。如果接收超高速缓存非推测地具有该线,则该线被无效,并且在某些情况下,发出超高速缓存-到-超高速缓存的传送。但是,如果该线为R 132或W 138,则检测到冲突。

[0137] 验证:因为冲突检测针对每个加载进行,因此事务总是具有对其自身写入集的独占存取。因此,验证不需要任何额外工作。

[0138] 提交(commit):由于急切版本控制就地存储数据项的新本版,因此提交处理仅清除W 138和R 132比特并丢弃取消日志。

[0139] 中止(Abort):当事务回滚时,每个超高速缓存线在取消日志中的原始版本必须被恢复,一种被称为“展开(unrolling)”或“施加(applying)”该日志的处理。这可以在tm_discard()期间进行并且对其它事务而言必须是原子的。具体而言,写入集依然必须被用于检测冲突:该事务在其取消日志中具有唯一争取全额版本,并且请求事务必须等待将从该日志中恢复的正确版本。这种日志能够使用硬件状态机或软件中止处理器施加。

[0140] 急切-悲观具有如下特征:提交时简单的,并且由于其实就地进行因此非常快。类似地,验证是空操作(no-op)。悲观冲突检测较早检测冲突,由此减少了“被判定(doomed)”事务的数量。例如,如果两个事务涉及读后即写(Write-After-Read)相关性(dependency),那么在悲观冲突检测中立即检测该相关性(dependency)。不过,在悲观冲突检测中,这种冲突不会被检测直到作者提交为止。

[0141] 急切-悲观还具有如下特征:如上所述,超高速缓存线第一次被写入时,旧值必须

被写入日志，导致额外超高速缓存存取。中止由于它们要求取消(undoing)该日志而昂贵。对于该日志中的每个超高速缓存线，必须发出加载，也许在继续到下一线之前直到主存那么远。悲观冲突检测也防止某些可串行化的调度存在。

[0142] 此外，因为冲突随着它们的出现而被处理，因此存在活锁(livelock)的可能性并且必须应用小心的取数碰撞管理机制来保证转发进度。

[0143] 懒惰-乐观(LO)

[0144] 另一种流行TM设计为懒惰-乐观(LO)，其在“写入缓冲器”或“重做日志(redo log)”中存储器写入集并且在提交时检测冲突(依然使用R 132和W 138比特)。

[0145] 版本控制：正如在EP系统一样，LO设计的MESI协议被迫位于事务之外。一旦在事务之内，读取一条线不仅招致标准MESI过渡而且设置R 132比特。同样，写入一条线色设置该线的W 138比特，但是处理LO设计的该MESI过渡与处理EP设计的MESI过渡不同。首先，采用懒惰版本控制，被写数据的新版本被存储在超高速缓存层次中直到提交为止，同时其他事务具有对在内存或其他超高速缓存中可获得的旧版本的存取。为了使得旧版本可以获得，脏线(M线)在首次被事务写入时必须被驱逐(evict)。其次，由于下述乐观冲突检测特征而不需要升级缺失：如果事务具有处于S状态的线，则其能够简单向其写入并且将该线升级到M状态而无需将该变化与其他事务通讯，因为冲突检测在提交时进行。

[0146] 冲突检测和验证：为了验证事务和检测冲突，LO仅在其正准备提交时将推测修改的线的地址通讯给(communicate)其他事务。在验证时，处理器发送一个，可能更大的，网络数据包，其包含素数写入集中的所有地址。数据不被发送，而是留在提交器(committer)的超高速缓存中并被标记为脏(M)。为了在不在超高速缓存中搜索标记为W的线而构建这种数据包，使用一种简单比特矢量，被称为“存储缓冲器”，每个超高速缓存线一个比特用来跟踪这些被推测地修改的线。其它事务使用该地址数据包来检测冲突：如果在该超高速缓存中发现地址并且R 132和/或W 138比特被设置，则发起冲突。如果发现了该线并且R 132没有被设置W 138也没有被设置，则该线被简单地无效，这类似于处理独占加载。

[0147] 为了支持事务原子性，这些地址数据包必须原子化地被处理，即，没有两个地址数据包可以一次以相同的地址存在。在LO系统中，这可以通过在发送地址数据包之前简单获取全局提交令牌来实现。不过，可以通过首先发出该地址数据包、收集响应、施行排序协议(也许是最旧的事务第一)以及一次提交所有响应都是令人满意的来应用两相位提交方案。

[0148] 提交(Commit)：一旦已经进行验证，提交不需要特殊处理：简单清除W 138和R 132比特和所述存储缓冲器。所述事务的写入在该超高速缓存中已经被标记为脏并且这些线的其他超高速缓存的副本经由所述地址数据包已经被无效。其它处理器随后能够通过常规相关性协议来存取所提交的数据。

[0149] 中止：回滚同等方便：因为写入集被包含在本地超高速缓存中，因此这线能够被无效，随后清除W 138和R 132比特以及所述存储缓冲器。所述存储缓冲器使得W线被发现以便无需搜索该超高速缓存的情况下无效。

[0150] 懒惰-乐观具有如下特征：中止非常快，不需要附加加载或存储，并且仅仅标记本地变化。与在EP中发现的相比，可以存在更可串行化的调度，其使得LO系统更进取地推测事务时独立的，其能够产生更高的性能。最后，晚检测冲突能够增加转发进度的可能性。

[0151] 懒惰-乐观也具有如下特征：验证采取与写入集的尺寸成比例的全局通讯时间。被

判定的食物会浪费工作，因为仅仅在提交时间检测冲突。

[0152] 懒惰-悲观(LP)

[0153] 懒惰-悲观(LP)代表第三中TM涉及选项，位于EP和LO之间的某个位置：在写入缓冲器中存储新写入的线并基于每次存取检测冲突。

[0154] 版本控制：版本控制与LO的版本控制类似但又不一样：读取线设置其R比特132、写入线设置其W比特138，并且存储缓冲器被用于跟踪该超高速缓存中的W线。而且，在第一次被事务写入时脏(M)线必须被驱逐，就和在LO中一样。不过，由于冲突检测是悲观的，因此当从I、S→M升级事务线时加载独占必须被执行，这是与LO不同的。

[0155] 冲突检测：LP的冲突检测操作与EP的冲突检测操作相同：使用相关性消息来寻找事务之间的冲突。

[0156] 验证：与在EP中一样，悲观冲突检测确保在任何点处正运行的事务与其他正运行的事务没有冲突，因此验证为空操作。

[0157] 提交：提交不需要特殊处理：简单清除W 138和R 132比特以及所述存储缓冲器，与在LO中一样。

[0158] 中止：回滚也与LO的回滚一样：使用所述存储缓冲器简单地无效所述写入集并且清除W和R比特以及所述存储缓冲器。

[0159] 急切-乐观(EO)

[0160] LP具有如下特征：与LO一样，中止非常快。与EP一样，悲观冲突检测的使用减少了“被判定的”事务的数量。与EP一样，一些可串行化的调度不被容许并且必须对每个超高速缓存缺失执行冲突检测。

[0161] 版本控制和冲突检测的最终的组合是急切-乐观(EO)。EO可能比用于HTM系统的最优选择少一些：因为新事务性版本被就地写入，因此其它事务没有选择而是在冲突出现时(即，在超高速缓存缺失出现时)注意到冲突。但是由于EO在提交时之前一直等待以便检测冲突，因此这些事务是“僵的(zombies)”，继续执行、浪费资源，也被“判定”成中止。

[0162] EO已经被证明在STM中是有用的并且可以通过Bartok-STM和McRT来实现。懒惰版本控制STM需要对每次读取核实其写入缓冲器以便确保其正在读取最近的值。由于写入缓冲器不是非常昂贵的硬件构件，因此对就地写入急切版本控制为优选。此外，由于在STM中核实冲突也很昂贵，因此乐观冲突检测提供了批量执行该操作的优点。

[0163] 取数碰撞管理

[0164] 上面已经描述了一旦系统已经决定中止该事务则该事务如何回滚，但是，由于冲突涉及两个事务，因此需要探究如下主题：哪个事务应该中止、该中止应该怎样发起以及什么时候所中止的事务应该被重试。这些主题通过取数碰撞管理(CM)、事务性内存的关键组件来解决。下面描述的策略涉及该系统如何发起中止以及管理在冲突中哪些事务应该中止的各种确立的方法

[0165] 取数碰撞管理策略

[0166] 取数碰撞管理(CM)策略是一种机制，其确定卷入冲突的哪个事务应该中止以及所中止的事务何时应该被重试。例如，通常的情况是，立即重试被中止的事务不会导致最好的性能。相反，采用后退(back-off)机制能够产生更好的性能，该机制延迟了所中止事务的重试。STM首先设法解决了找到最好的取数碰撞管理策略并且下面概述的多个策略起初都是

为STM开发的

[0167] CM策略引出了的一些测量值以便做出决定，包括事务的年龄、读取集和写入集的尺寸、之前中止的数量等等。用于做出这些决定的测量值的组合是无穷的，但是下面粗略地按照增加的复杂性描述了某些组合。

[0168] 为了建立某些术语(nomenclature)，首先注意到，在冲突中存在两方：攻击方和防守方。攻击方是请求存取共享内存位置的事务。在悲观冲突检测中，攻击方是发出加载或加载独占的事务。在乐观冲突检测中，攻击方是试图验证的事务。防守方在两种情况下是接收攻击方的请求的事务。

[0169] 进取(aggressive)的CM策略立即和总是使得攻击方或防守方重试。在LO中，进取意味着攻击方总是赢，并且因此进取有时候被称为提交器赢。这种策略被用于最早的LO系统。在EP的情况下，进取可以是防守方赢或者进攻方赢。

[0170] 重启将理解经历另一个冲突的冲突中事务必然浪费工作—即互连带宽重新充满超高速缓存缺失。有礼的CM策略在重启冲突之前采用指数(exponential)后退(backoff)(但是也可以使用线性后退)。为了防止不足(starvation)，对于其中处理不具有由调度器分配给它的资源的情形，指数后退极大地增加了在许多n次重试之后事务成功的奇数(odd)。

[0171] 冲突解决的另一种途径是随机中止所述攻击方和防守方(称之为随机化的策略)。这种策略可以与一种随机化的后退方案结合以避免必要的取数碰撞。

[0172] 不过，采用随机选择，在选择事务中止时，会导致中止已经完成“很多工作”的事务，这会浪费一些资源。为了避免这种浪费，可以在确定哪个事务要中止时将事务上完成的工作量考虑在内。一个工作测量值可以是事务的年龄。其它方法包括：Oldest(最老的)、Bulk TM(批量TM)、Size Matters(大小问题)、Karma(卡马)以及Polka(波尔卡)。Oldest是一种简单时间戳方法，其中止冲突中的最年轻事务。Bulk TM使用该方案。Size Matters与Oldest一样，但是不同的是事务年龄，读取/写入字的数量被用作优先级，在固定数量的中止之后转回到Oldest。Karma类似，使用写入集作为优先级。在后退固定次数后回滚随后进行。在被中止后所中止的事务保持它们的优先级(因此命名为Karma)。Polka作用过程与Karma一样，不同是的后退预定的次数，其每次指数级地后退更多。

[0173] 由于中止浪费工作，因此逻辑上而言使得攻击方停顿直到防守方完成它们的事务会导致更好的性能。不幸的是，这种简单方案已与导致死锁(deadlock)。

[0174] 可以使用一些死锁避免技术来解决该问题。Greedy使用两个规则来避免死锁。第一个规则是，如果第一事务T1比第二事务T0的优先级低，或者如果T1正在等待另一个事务，则T1在与T0冲突时中止。第二个规则是，如果T1的优先级比T0高并且并不是在等待，则T0等待直到T1提交、中止或开始等待(在该情况下，施加第一规则)为止。Greedy提供一些关于时间界限的保证用于执行一组事务。一个EP设计(LogTM)使用与Greedy类似的CM策略以便采用保守的死锁避免实现停顿。

[0175] 实例MESI相关性规则提供了多处理器超高速缓存的超高速缓存线可以具有的四种可能状态：M、E、S以及I，其定义如下：

[0176] 修改(Modified)(M)：超高速缓存线仅仅位于当前超高速缓存中，并且是脏的；其已经修改于主存中的值。在允许(不再有效的)主存状态的任何其它读取之前，该超高速缓

存需要在未来某个时候将数据写回主存。所述写回将所述线改变为独占状态。

[0177] 独占(Exclusive)(E):超高速缓存线仅仅位于当前超高速缓存中,但是干净的;其匹配主存。其在任何时候可以响应于读取请求改变为共享状态。可替换地,其可以在写入它时被改变为Modified(修改)状态。

[0178] 共享(Shared)(S):表示该超高速缓存线可以存储在该机器的其他超高速缓存中并且是“干净的(clean)”;其匹配主存。该线可以在任何时候被丢弃(变为Invalid(无效)状态)。

[0179] 无效(Invalid)(I):表示该超高速缓存线是无效的(未被使用)。

[0180] TM相关性状态指示符(R 132、W 138)可以提供给每个超高速缓存线,此外,或者可以被编码到MESI相关性比特。R 132指示符表示当前事务已经从该超高速缓存线的数据中读取,并且W 138指示符表示当前事务已经写向该超高速缓存线的数据。

[0181] 在TM设计的另一个方面,使用事务性存储缓冲器设计系统。于2000年3月31提申请的题目为“Methods and Apparatus for Reordering and Renaming Memory References in a Multiprocessor Computer System(用于在多处理器计算机系统中记录和重命名内存引用的方法和装置)”的并且通过引用方式整体包含在本申请中的美国专利US6,349,361教导了一种在具有至少第一和第二处理器的多处理器计算机系统记录和重命名内存引用的方法。第一处理器具有第一专用(private)超高速缓存以及第一缓冲器,并且第二处理器具有第二专用超高速缓存和第二缓冲器。该方法包括如下步骤:对于第一处理器接收到的用于存储数据(datum)的多个门控存储请求的每一个,通过第一专门超高速缓存排它地获取含有所述数据的超高速缓存线,并且将该数据存储在第一缓冲器中。当第一缓冲器从第一处理器接收到加载特定数据的加载请求,按照加载和存储操作的排列顺序从存储在第一缓冲器中的数据间将该特定数据提供给第一处理器。一旦第一超高速缓存从第二超高速缓存接收到对所给定数据的加载请求,差错条件被指示并且当所述对所给定数据的加载请求对应于第一缓冲器中存储的数据时至少一个处理器的当前状态被重置为更早的状态。

[0182] 一个这样的事务性内存设施的主要实施组件为用于保持前事务GR(通用寄存器)内容的事务备份寄存器文件、用于跟踪在事务期间被存取的超高速缓存线的超高速缓存目录、用于在该事务结束前一直缓冲一些存储的存储超高速缓存、以及用于执行各种复杂功能的固件例程。在该部分,描述了详细实施方式。

[0183] IBM zEnterprise EC12企业服务器实施例

[0184] IBM zEnterprise EC12企业服务器介绍了事务性内存中的事务性执行(TX),并且部分在通过引用方式整体包含在本申请中的从IEEE计算机学会会议出版服务(Computer Society Conference Publishing Services)(CPS)可获得的出现在加拿大不列颠哥伦比亚省温哥华的2012年12月1-5日的MICRO-45上的会议汇编第25-36页(Proceedings Pages 25-36presented at MICRO-45,1-5December 2012,Vancouver,British Columbia,Canada,available from IEEE Computer Society Conference Publishing Services(CPS))的论文“用于IBM系统z的事务性内存架构和实施(Transactional Memory Architecture and Implementation for IBM System z)”中得以描述。

[0185] 表3表示一种实例事务。起始于TBEGIN的事务并不被确保总是以TEND成功完成,因为它们在每次被试图执行时会经历中止条件,例如由于重复与其他CPU的冲突。这要求程序

支持回落路径以便例如通过使用传统锁定方案非事务性地执行相同操作。尤其是在回落路径不由可靠的编译器自动生成时，这将重要的负担加在了编程和软件检验团队上。

[0186] 表3

[0187]

实例事务代码

```
        LHI     R0,0            *initialize retry count=0
loop    TBEGIN                  *begin transaction
        JNZ     abort           *go to abort code if CC1=0
        LT      R1, lock        *load and test the fallback lock
        JNZ     lckbzy          *branch if lock busy

        ... perform operation ...

        TEND                    *end transaction
        ...     ...     ...     ...

lckbzy  TABORT                  *abort if lock busy; this
                                *resumes after TBEGIN
abort   JO      fallback        *no retry if CC=3
        AHI     R0, 1           *increment retry count
        CIJNL   R0,6, fallback  *give up after 6 attempts
        PPA     R0, TX          *random delay based on retry count

        ... potentially wait for lock to become free ...

        J       loop            *jump back to retryfallback
        OBTAIN  lock            *using Compare&Swap

        ... perform operation ...

        RELEASE lock
        ...     ...     ...     ...
```

[0188] 要求为被中止的事务执行(TX)事务提供回落路径可能是繁重的(onerous)。在共享数据结构上运行的许多事务期望短一些、仅仅触到很少不同内存位置、并且仅仅使用简单的指令。对于这些事务，IBM zEnterprise EC12引入了受约束事务的概念；在正常条件下，即使不对必要重试的数量给出严格限制，CPU 114(图2)也确保该受约束的事务最终成功结束。受约束的事务起始于TBEGINC指令并结束语常规TEND。将一种任务实现为受约束或非受约束的事务通常导致非常可比较的(comparable)性能，但是受约束的事务通过去除对回落路径的需求而简化软件开发。IBM的事务执行架构还在此处通过引用整体包含在本申请中的IBM于2012年9月发布的SA22-7832-09第10版z/架构、操作原理(z/Architecture, Principles of Operation,Tenth Edition,SA22-7832-09)中被描述。

[0189] 受约束的事务起始于TBEGINC指令。起始于TBEGINC的事务必须遵循编程约束列

表;否则该程序采取非可过滤的违反约束的中断。示例性事务可以包括但不限于:事务能够执行最多32条指令,所有指令文本必须在内存的256的连续字节内;事务仅仅包含指向前(forward-pointing)的相关分支(即,无循环或子例程调用);该事务可以存取内存的最大4个对齐的八字(octoword)(一个八字为32字节);并且限制指令集以便排除像十进制或浮点运算的复杂指令。所述约束被选择以便能够执行像双链表插入/删除(doubly linked list-insert/delete)操作的多个普通操作,包括目标高达4个对齐八字的原子比较并交换(compare-and-swap)的非常强大的概念。同时,所述约束被保守地选择以便未来CPU实施方式能够保证事务成功而无需调整这些约束,因为否则那会导致软件不兼容。

[0190] TBEGINC大多表现得像TSX中的XBEGIN和IBM's zEC12服务器上的TBEGIN,不同的是浮点寄存器(FPR)控制和程序中断过滤字段不存在并且该控制被认为为零。在事务中止时,指令地址被直接设置回TBEGINC,而不是设置到之后的指令,这反映用于受约束事务的中间重试和中止路径的缺乏。

[0191] 在受约束的事务总不容许嵌套事务,但是如果在非受约束事务内出现TBEGINC,其被处理为正如TBEGIN那样开启新的非受约束的嵌套级别。例如,如果非受约束事务调用内部使用受约束事务的子例程,则这种情况就会出现。

[0192] 由于中断过滤隐含地被关闭,在受约束事务期间的所有异常导致操作系统(OS)的中断。该事务的最后成功完成依赖于该OS的能力以便页调入(page-in)由任何受约束事务所触及的最多4页。该OS也必须确保时间片(time-slice)长度足以容许该事务完成。

[0193] 表4

[0194]

事务代码实例

TBEGINC	*begin constrained transaction
...perform operation...	
TEND	*end transaction

[0195] 表4显示了表3中代码的受约束-事务性实施方式,假设该受约束的事务不与其他基于锁定的代码相互作用。因此没有显示锁定测试,但是如果受约束事务和基于锁定的代码被混合也可以添加锁定测试。

[0196] 当故障重复出现时,使用毫码(millicode)作为系统固件的部分来执行软件仿真。有利的是,受约束的事务具有理想的特性,因为该负担从程序员身上去掉。

[0197] 参见图3,IBM zEnterprise EC12处理器引入了事务性执行设施。该处理器能够每个时钟周期解码3个指令;简单指令被快速处理(dispatch)为单一微操作(micro-op),并且更复杂指令被分裂(crack)为多个微操作。所述微操作(Uops 232b)被写入统一发行(issue)队列216,从该统一发行队列可以不按照顺序地发出这些为操作。每个周期可以执行高达两个固定点、一个浮点、两个加载/存储以及两个分支指令。全局完成表(GCT)232保持每个微操作232b和事务嵌套深度(TND)232a。GCT 232在解码时按照顺序写入,跟踪每个

微操作232b的执行状态，以及在最旧指令组的所有微操作已经成功执行时完成这些指令。

[0198] 级别1(L1)240为数据超高速缓存。L1 240为96KB(千字节)6-路关联超高速缓存，其具有256字节超高速缓存线和4循环使用等待时间，其连接到专用(private)1MB(兆字节)8-路关联第二级别(L2)268。L2 268是一种数据超高速缓存，具有用于L1 240缺失的7循环使用等待时间损失。L1 240是与处理器最近的超高速缓存，并且Ln超高速缓存是处于进行超高速缓存的第n级别。L1 240和L2 268两者都是通过式存(store-through)超高速缓存。每个中央处理器(CP)芯片上的六核共享48MB第三级内在式存(store-in)超高速缓存，并且六CP芯片连接到一起包封在玻璃陶瓷多芯片模块(MCM)上的芯片外(off-chip)384MB低四级超高速缓存。高达4个的多芯片模块(MCM)可以连接到具有高达144个核(并不是所有的核都可以用于运行客户的工作负载)的相干对称多处理器(SMP)系统。

[0199] 相干性(Coherency)采用MESI协议的变化形式进行管理。超高速缓存线可以自我只读(共享)或独占；L1 240和L2 268为通过式存(store-through)超高速缓存并因此不含有脏dirty)线。L3 272和L4超高速缓存(未示出)为内在式存(store-in)并跟踪脏的状态。每个超高速缓存包括所有其连接的更低级别的超高速缓存。

[0200] 相干性请求被称为“交叉询问”(XI)并且被有层次低从高级别发送到较低级别的超高速缓存，并且在L4之间。当一个核丢失了L1 240和L2 268并且从其本地L3 272请求超高速缓存线时，L3 272核查其是否拥有该线，并且如果有必要则在其将该超高速缓存线返回给所述请求者之前，发送XI给当前正拥有的在该L3 272之下的L2 268/L1 240以便确保相干性。如果请求还错过L3 272，则L3 272发送请求给L4超高速缓存(未示出)，其通过发送XI到在L4之下的所有必要的L3以及相邻的L4来实施(enforce)相干性。随后，该L4对将响应转发到L2 268/L1 240的作出请求的L3作出响应。

[0201] 需要注意的是，由于超高速缓存层次的包含性(inclusivity)规则，有时候由于由来自请求的相关性(associativity)溢出导致的在较高级别的超高速缓存上的驱逐，该超高速缓存线从较低级别的超高速缓存被XI(XI’ed)到其他超高速缓存线。这些XI可以被称为“LRU XI”其中LRU表示最近被使用的。

[0202] 还可以参见另一个类型的XI请求，Demote-XI(降级-XI)将超高速缓存所有关系从独占转变成只读状态，而Exclusive-XI(独占-XI)将高速缓存所有关系从独占转变成无效状态。Demote-XI和Exclusive-XI需要响应返回到该XI发送者。目标超高速缓存可以“接受”该XI，或者如果其在接受该XI之前首先需要驱逐脏数据则发送“拒绝”响应。L1 240/L2 268超高速缓存为通过式存(store-through)，但是如果在将所述独占状态降级之前，它们在它们存储队列中具有需要被发送到L3的存储，则可能拒绝降级-XI和独占-XI。被拒绝的XI将会被发送者重复。只读-XI被发送给拥有只读线的超高速缓存；对这种XI不需要作出响应因为它们不会被拒绝。SMP协议的细节与P.Mak、C.Walters以及G.Strait在通过引用被整体包含在本申请中的2009年的IBM研究与开发期刊卷53：1的“IBM系统z10处理器超高速缓存子系统微架构(IBM System z10processor cache subsystem microarchitecture)”中针对IBM z10所描述的细节类似。

[0203] 事务性指令执行

[0204] 图3描绘了实例CPU环境112的实例组件，包括CPU 114和与其相互作用的超高速缓存/组件(诸如图1和2中所示的那些)。指令解码单元(IDU)208保持对当前事务嵌套深度

(TND)212的跟踪。当IDU 208接收到TBEGIN指令时,TND 212递增,并且相反地在接收到TEND指令时递减。对于每个被分派的指令,TND 212被写入GCT 232。当TBEGIN或TEND在后来被冲洗的推测(speculative)路径上被解码时,IDU 208的TND 212从没有被冲洗的最年轻的GCT 232输入项被刷新。该事务性状态也由执行单元,通常是由加载/存储单元(LSU)280,写入发行(issue)队列216用于消费,该执行单元也具有包含在LSU 280中的有效地址计算器236。如果该事务在到达TEND指令之前应该中止,则TBEGIN指令可以指定事务诊断块(TDB)来记录状态信息。

[0205] 与嵌套深度类似,IDU 208/GCT 232通过事务嵌套合作地跟踪存取寄存器/浮点寄存器(AR/FPR)修改掩码(mask);当AR/FPR-修改指令被解码并且该修改掩码阻挡该修改时IDU 208可以将中止请求置于GCT 232中。当该指令接近于完成时,完成被阻挡并且事务中止。包括TBEGIN的其它受限制指令,如果在受约束事务中时被解码或者超过最大嵌套深度时类似地被处理。

[0206] 最外面的TBEGIN基于GR-Save-Mask(GR-保存-掩码)而被分裂为多个微操作;每个微操作232b(包括,例如uop 0、uop 1以及uop2)将被两个定点单元(FXU)220执行以便将一对GR 228保存到特殊事务备份寄存器文件224中,其后来在事务中止情况下被用于恢复该对GR 228的内容。而且,如果一个被指定,则TBEGIN引起(spawn)微操作232b执行可存储性测试;地址被保存子啊专用寄存器中以便随后用于中止情况。在最外面TBEGIN的解码时,TBEGIN的指令地址和指令文本也被保存在专用寄存器中用于后来开始的潜在中止处理。

[0207] TEND和NTSTG为单个微操作232b指令;NTSTG(非事务性存储)像正常存储一样被处理,不同的是其在发行队列216中被标记为非事务性的,使得LSU 280能够适当地处理它。TEND在执行时为空操作,在TEND完成时执行该事务的结束。

[0208] 如所提到的,在事务内的指令在发行队列216中被这样标记,但是在另外情况下大多不变地执行;LSU 280执行如下面部分所描述的隔离跟踪。

[0209] 由于解码时按照顺序的,并且由于IDU 208保持跟踪当前事务性状态并且将其与来自该事务的每个指令一起写入发行队列216,因此TBEGIN、TEND以及该事务之前、之内以及之后的指令的执行可以不按照顺序进行。甚至有可能(尽管不可能)首先执行TEND,随后执行整个事务,并且最后执行TBEGIN。程序顺序在完成时通过GCT 232被恢复。事务的长度不受GCT 232的尺寸的限制,因为通用寄存器(GR)228可以根据备份寄存器文件224被恢复。

[0210] 在执行期间,程序事件记录(PER)事件基于Event Suppression Control(事件抑制控制)被过滤,并且PER TEND事件在被启用的请下被检测到。类似地,当在事务性模式下时,伪随机生成器可能由于被Transaction Diagnostics Control(事务诊断控制)所启用而正造成随机中止。

[0211] 事务性隔离的跟踪

[0212] 加载/存储单元280跟踪在事务执行期间被存取的超高速缓存线,并且在来自另一个CPU的XI(或LRU-XI)与占用(footprint)冲突的情况下触发中止。如果冲突的XI为独占或降级XI,在希望在L3 272重复XI之前完成该事务的情况下LSU 280拒绝XI回到L3 272。这种“伸手推开(stiff-arming)”在高度碰头(contended)的事务中是非常有效的。为了在两个CPU彼此伸手推开时防止悬而不决,采用一种XI-拒绝计数器,其可以在满足阈值时触发事务中止。

[0213] 传统上采用静态随机存取存储器(SRAM)来实现L1超高速缓存目录240。对于事务性内存实施方式，该目录的有效比特244(64行×6路)已经被移动到正常逻辑锁存器并且每超高速缓存线再多补充两个比特：TX-读取248和TX-脏252比特。

[0214] 在新的最外面TBEGIN被解码时(其与先前依然待决的事务互锁)TX-读取248比特被重置。TX-读取248比特在执行时由在发行队列中每个被标记为“事务性的”加载指令设置。需要注意的是，例如如果推测加载在错误预测分支路径上被执行，这能够导致过度标记(over-marking)。在加载完成时TX-读取248比特的设置的替代方式对于硅面积而言过于昂贵，因为多个加载会在同时完成，这需要在加载队列上的多个读取端口。

[0215] 存储以与非事务性模式相同方式执行，但是事务标记被置于所述存储指令的存储队列(STQ)260输入项中。在写回时间，当来自STQ 260的数据被写入L1 240时，L1-目录256中的TX-脏比特252被设置用于被写入的超高速缓存线。仅仅在所述存储指令已经完成之后才出现到L1 240的存储写回，并且每个周期最多一个存储被写回。在完成和写回之前，加载可以借助于存储-转发存取来自STQ 260的数据；在写回之后，CPU 114(图2)可以存取L1超高速缓存240中的被推测更新的数据。如果该事务成功结束，则所有超高速缓存线的相应TX-脏比特252被清除，并且在STQ 260中还没有被写入的存储的TX-标记也被清除，有效地将待决的存储转变成正常存储。

[0216] 在事务中止时，所有待决事务性存储从STQ 260被无效，即使这些存储已经完成。被L1超高速缓存240中的事务所修改的，即，具有TX-脏比特252开启的，所有超高速缓存线使得其有效比特关闭，有效地将它们从L1超高速缓存240立即去除。

[0217] 该架构要求在完成新指令之前维持该事务读取集和写入集的隔离。在XI待决时，通过在适当的时候使得指令完成停止来确保该隔离；推测的无序执行被允许，乐观地假设该待决的XI将到不同的地址并且不实际导致事务冲突。该设计非常自然地适于在现有系统上实现的XI-对-完成(XI-vs-completion)互锁，从而确保该架构所需要的强劲的内存排序。

[0218] 当L1超高速缓存240接收到XI，L1超高速缓存240存取该目录以便核实L1超高速缓存240中的被XI的地址的有效性，并且如果TX-读取比特248在被XI的线上是活跃的(active)并且XI没有被拒绝，则LSU 280触发中止。当具有活跃的TX-读取比特248的超高速缓存线从L1超高速缓存240中被LRU(LRU'ed)，则特定的LRU-扩展矢量针对L1超高速缓存240的64行的每一行记住TX-读取线存在于该行上。由于对于LRU扩展不存在精确地址跟踪，因此命中有效扩展行LSU 280的任何非被拒绝的XI触发中止。如果LRU-扩展将读取占用能力从L1-尺寸有效地增加到L2-尺寸以及有效地增加关联性，则对非精确LRU-扩展跟踪的与其他CPU 114(图1和2)没有冲突会导致中止。

[0219] 内存占用受到该存储超高速缓存尺寸(该存储超高速缓存在下面将更详细阐述)的限制，并且隐含地收到L2超高速缓存268的尺寸和关联性的限制。没有LRU-扩展动作需要在TX-脏252超高速缓存线从L1超高速缓存240中被LRU(LRU'ed)时被执行。

[0220] 存储超高速缓存

[0221] 在现有系统中，由于L1超高速缓存240和L2超高速缓存268为通过式存超高速缓存，因此每个存储指令导致L3 272存储访问；由于现在每个L3 272有6核并且每个核的改进的性能，因此用于L3 272的存储速率(以及对于L2超高速缓存268的更少的扩展)对每种工

作负载变成问题。为了避免存储排队延迟，必须添加一种聚集存储超高速缓存264，其在将一些存储发送到L3 272之前将一些存储组合到一些相邻的地址。

[0222] 对于事务性内存执行，可以接受的是在事务中止时根据L1超高速缓存240无效每个TX-脏252超高速缓存线，因为L2超高速缓存268超高速缓存是完全关闭的(7周期L1超高速缓存240缺失代价)以便带回到清洁线。不过，对性能(以及用于跟踪的硅面积)而言不可接受的是，在事务结束之前使得事务性存储写入L2超高速缓存268以及随后使得在中止的(或甚至更糟糕的在共享L3 272上的)所有脏L2超高速缓存268超高速缓存线无效。

[0223] 采用存储超高速缓存264的聚集可以解决存储带宽和事务性内存存储两个问题。存储超高速缓存264是一种环形队列的输入项64。每个输入项保持具有字节精确的有效比特的128字节的数据。在非事务性操作中，当从LSU280接收到存储时，所述存储超高速缓存264核实对于相同地址是否存在输入项，并且如果存在，则将新存储聚集到现有输入项中。如果没有输入项，则将新输入项写入队列中，并且如果自由输入项的数量落于阈值之下，则最老的输入项被写回到L2超高速缓存268和L3 272超高速缓存。

[0224] 当新的最外面事务开始时，在所述存储超高速缓存中的所有现有输入项都被标记为被关闭，以便没有新存储能够被聚集到它们中，并且开始将那些输入项驱逐到L2超高速缓存268和L3 272。从该点开始，出自于STQ 260、LSU 280的事务性存储分配新输入项，或者聚集到现有事务性输入项中。将那些存储写回到L2超高速缓存268和L3 272中被阻挡，直到该事务成功结束为止；在该点处，随后的(事务后的)存储可以继续聚集到现有输入项中，直到下一个事务再次关闭那些输入项。

[0225] 存储超高速缓存264被查询每个独占或降级XI，并且在XI与任何激活输入项对比的情况下导致XI拒绝。如果所述核不是正在进一步完成指令而是连续拒绝XI，则在一定阈值处中止该事务以便避免暂停(hang)。

[0226] LSU 280在存储超高速缓存264溢出时请求事务中止。LSU 280在其试图发送不能合并到现有输入项的新存储时检测该条件，并且整个存储超高速缓存264被充满来自当前事务的存储。存储超高速缓存264被管理为L2超高速缓存268的子集：在脏线事务性地从L1超高速缓存240中被驱逐时，它们必须在整个事务中保持驻留在L2超高速缓存268中。因此最大内存占用被限于64×128字节的存储超高速缓存尺寸，但是其还受到L2超高速缓存268的关联性的限制。由于L2超高速缓存268为8-路关联的并且具有512行，因此其通常大到足以不会导致事务中止。

[0227] 如果事务中止，则存储超高速缓存264被通知并且保持事务性数据的所有输入项被无效。存储超高速缓存264还每双字具有输入项是否被NTSTG指令写入的标记(8字节)—那些双字在事务中止上保持有效。

[0228] 实现毫码的功能

[0229] 传统上，IBM主机架服务器处理器含有一层被称为毫码的固件，其执行复杂功能，例如某些CISC指令指令、中断处理、系统同步以及RAS。毫码包括依赖于机器的指令以及指令集架构的指令，与应用程序和操作系统(OS)的指令类似，其从内存中被取出并执行。固件驻留在主存的客户程序不能访问的受限制区域。当硬件检测到需要调用毫码的情形，取指令单元204切换到“毫码模式”并且在该毫码内存区域中的适当位置处开始取指令。毫码可以以与指令集架构(ISA)的指令相同的方式被取出和执行，并且可以包括ISA指令，

[0230] 对于事务性内存，各种复杂情形下涉及到毫码。每个事务中止调用专用毫码子例程以便执行必要的中止步骤。该事务中止毫码通过读取专用寄存器(SPR)启动，该专用寄存器保持硬件内部中止原因、潜在异常原因以及被中止指令地址，在其中之一被指明时毫码随后使用其来存储TDB。TBEGIN指令文本从SPR加载以便获得GR-保存-掩码，该GR-保存-掩码被毫码所需以便了解哪个GR 228将恢复。

[0231] CPU(例如图1的CPU 114a、114b)支持特定的仅用于毫码的(millicode-only)指令以便读取TX备份-GR 224，并将其拷贝到主GR 228。TBEGIN指令地址也从SPR加载以便在PSW中设置新指令地址，从而一旦毫码中止子例程结束继续TBEGIN之后的执行。如果该中止是由非过滤的程序中断所导致的，该PSW后来被存储为旧程序(program-old)PSW。

[0232] TABORT指令可以一种所实现的毫码；当IDU 208解码TABORT时，其指令所述取指令单元分支到TABORT的毫码中，毫码从其中分支到普通(common)中止子例程。

[0233] 抽取事务嵌套深度(Extract Transaction Nesting Depth(ETND))指令还可以被毫码化，因为其不是性能关键；毫码加载出自于特定硬件寄存器的当前嵌套深度并将其置于GR 228中。PPA指令被毫码化；其基于作为操作数由软件提供给PPA的当前中止计数并且也基于其它硬件的内部状态来执行最佳延迟。

[0234] 对于受约束事务，毫码可以保持跟踪中止数量。在TEND成功完成时获取如果中断在OS中出现时(因为如果或当OS将返回到该程序是其并不被知晓)计数器被重置为零。依赖于该当前中止计数，毫码可以调用某些机制以便改善用于随后的事务输入项的成功机会。该机制包括例如成功增加在输入项之间的随机延迟、降低推测执行的数量以避免碰到由对该事务没有实际使用的数据的推测访问所导致的中止。作为最后的手段(resort)，毫码可以广播到其他CPU(例如图1的CPU 114a、114b)以便在释放其它CPU 114以便继续正常处理之前，停止所有冲突工作、重试本地事务。多个CPU(例如图1的CPU 114a、114b)必须协调起来以便不会导致死锁，因此需要在不同CPU(例如图1的CPU 114a、114b)上的毫码实例之间的某种串行化。

[0235] 嵌套事务时一种在另一个事务内执行的事务。其中嵌套有嵌套事务的事务被称为其“外部”事务。当被嵌套事务失败时，在该被嵌套的事务内做出的所有改变被回滚。不过，该被嵌套的事务的失败必然导致其外部事务失败。外部事务是否响应于被嵌套事务的失败而失败取决于外部事务的逻辑。

[0236] 事务可以被嵌套，并且可以分为开放式嵌套和闭合式嵌套。如果线程当前正在执行事务并且到达新事务的起点，则该原子块作为当前正在执行的父系事务的闭合式嵌套子事务被执行。该被嵌套的事务如包封的事务在相同隔离边界内执行，并且如该包封事务的其它内存访问一样，被嵌套事务的效果仅仅在该包封的事务提交时可见。换句话说，父系事务被有效挂起(suspended)，并且使得该封闭式被嵌套事务在该父系事务内的处理被恢复之前运行到完成。当被嵌套的事务回滚时，其临时效果被取消并且父系事务的状态被恢复到该被嵌套子事务的开始点。用于将子系事务和父系事务组合的两种已知技术为折叠式(collapsed)嵌套和闭合式嵌套。

[0237] 折叠式嵌套是一种用于组合事务嵌套的技术。折叠式嵌套包括将所有被嵌套事务归入其顶级祖先(ancestor)。在现有已知的实例方案中，除了更新嵌套深度计数器之外，嵌套的开始/结束事务指令并不被处理；仅仅在该计数器在末端事务之后为0时事务才提交。

[0238] 折叠式嵌套的技术的一种替换方式闭合式嵌套，其中，仅仅中止的事务和其子孙辈，而非其整个祖辈，易于被回滚。闭合式嵌套在例如寻求获取高度碰头(contended)资源的长事务情况下提供潜在的性能优势。在闭合式嵌套策略中，将锁定获取包封在其自己的被嵌套事务中在碰头锁上的冲突的成本限制为少数几个被回滚的指令。用于支持闭合式嵌套的其他性能理由(justification)可以被推进；不过，需要注意的是，在被嵌套的事务提交之后，该被嵌套的事务的整个读取和写入集被合并到其父系事务中，这使得父系事务招致潜在的冲突。

[0239] 父系事务与子系事务的合并通常带来一些固有的风险。例如，当子系事务与父系事务合并时，父系事务的占用变得更大。换句话说，处理父系事务所需的资源量随着子系事务与父系事务的合并而变得更大。这回导致几个复杂情况。首先，通常存在一些系统限制，其约束给定事务的最大可容许尺寸，其产生对合并事务的尺寸限制。第二，流行资源的竞争会剧烈，在合并事务需要该资源的情况下这导致事务处理的延迟。第三，由于合并事务的尺寸增加，该事务通常需要更长时间来处理。因此，由该事务所使用的值可能被另一个事务读取和写入的机会会增加，这导致合并事务的中止。第四，如果冲突随着合并事务的子系事务而上升，则父系事务自身可能经历导致合并事务中止的最终的冲突。

[0240] 如果两个最外面事务被组合形成单一事务的话，这些复杂情况很可能变得更显著。组合两个最外面事务的结果是产生用于该事务的更大的占用(例如超高速缓存内存占用)，这会导致事务中止的更高的风险。至少对于该原因，在现有技术中，不会将两个最外面事务组合形成单一事务。在现有技术中的这种组合会导致众多的冲突，其将使得很多已知事务处理技术不可实行或非常低效。不过，在下面的讨论中，将讨论一种将两个或多个最外面事务组合起来以便增加性能的技术，在此被称为“聚结(coalescing)”的处理。

[0241] 用于计算系统中的事务性内存的硬件支持试图保证硬件事务的内容被原子地执行。即，硬件事务在其整体中生效或根本不生效。硬件以超高速缓存线粒度保持跟踪在事务内遇到的取出和内存占用，即，内存占用。内存占用包括在事务处理期间内存中的被读出或被写入的地址。对超高速缓存在这些线中的内存地址的取出和存储看起来是被原子地执行的。此外，硬件保存了通用寄存器的内容(例如通用寄存器的数量，以及那个寄存器能被保存、能够在TBEGIN指令中被限定)、事务开始指令的指令地址、程序状态字(PSW)以及在事务开始之前的其他架构化和微-架构化寄存器。在中止情况下，硬件恢复回该值。

[0242] 为了限制超高速缓存线的数量以便观察超高速缓存占用的子集，以及为了减少在中止情况下将要恢复的备份寄存器的数量，硬件可以仙剑在管线中的“激活”外部事务的数量。在有些实例中，这是因为，在其他事情中，每个外部事务为防止其万一被中止而具有其自身的备份寄存器值以及其自身的超高速缓存占用。在有些系统中，硬件将从分派到完成的最外面事务的数量限制为仅仅一个。这种限制性管线可以限制事务被处理的速率并且会导致带处理事务的瓶颈。在这种情况下，一旦新事务被解码同时还有另一个还没有被完成的事务，硬件使得管线暂停(stall)在解码或分派区域。该新的外部事务在解码或分配阶段的暂停(在实施时，该外部事务开始指令和所有随后的(即，较年轻的)指令通常被保持)可以减少对附加备份寄存器和其超高速缓存占用的附加的需要。

[0243] 在某些情况下，最外面事务具有紧密接近度。由于紧密的接近度，会出现管线暂停或管线空隙(pipeline bubble)，其导致性能显著降低。最外面事务可以具有单一深度或者

能够替代地包括嵌套事务。通过将两个或更多最外面事务聚结到单一事务，可以降低管线暂停或管线空隙。

[0244] 现在专项图4-8，每个最外面事务具其自身的超高速缓存占用以及在事务被中止的情况下需要恢复的通用寄存器内容的依赖于事务的值。超高速缓存占用为该事务的内存操作数访问所触及的所有超高速缓存线的地图。不过，如果两个或更多最外面事务被聚结成单一事务，则这些事务的占用被合并成单一占用(聚结的占用)。不过，依然为原始，即非聚结的，事务配置备份寄存器。因此，在中止的情况下，硬件回滚到所聚结的集合的最老事务的起点并且不将所聚结的事务的任何事务存储数据存储到内存。如果所聚结的事务的集合完成，则所聚结的事务的所有存储数据在单一原子操作中被存储到内存，如其他处理器所观察到的那样。

[0245] 在一个实施例中，处理器监测事务的结束(即，最外面事务结束指令)并且在新事务开始(事务开始指令)被是识别时确定该事务是否应该被聚结到另一个事务中。当两个最外面事务被聚结时，先前已经被非事务性地执行过的两者之间的非事务性指令，如果有的话，反而可以被事务性地执行。存在多个可以被用于做出这种决定的因素。可以将多个这种因素例如分类为阈值极限或成功事务处理后聚结的历史或具体指令特征。例如，一种是因素可以是在第一事务的最外面事务结束指令和另一个事务的最外面事务开始指令之间的指令的数量。该因素可以表达为阈值极限。如果在第一事务的最外面事务结束指令和另一个事务的最外面TBEGIN之间有太多指令，则那些事务将不会被聚结。

[0246] 在第二实例中，一种因素可以为预计的(projected)的每周期指令(IPC)。及时在两个事务之间的指令的数量较低，这些指令也可以被微编码或者具有需要大量执行周期来完成的长等待时间操作。因此，在确定是否将给定事务与另一个事务聚结时所预计的IPC可能是有帮助的因素。如果所预计的IPC超过阈值，则这些事务将不会被聚结。

[0247] 第三因素可以是两个事务之间的指令所属的类别(class)。如果这种指令不属于某一类别，则聚结这两个事务会导致中止。因此，在两个事务之间的指令的类别可以是一种在确定是否聚结两个最外面事务时需要考虑的有用因素。如果该指令不属于所需的类别，则这些事务将不会被聚结。

[0248] 第四因素可以是所识别事务的先前出现的历史。表示事务已经被聚结而没有中止的历史可以是该事务可以被再次聚结并一起被处理为单一事务的好指示符。

[0249] 第五因素可以是关于能够被聚结的事务的数量的阈值极限。例如，由于随着聚结事务的数量增加而终止风险的增加，采用的阈值极限为5。因此，如果5个事务已经被聚结，则将不会添加第六个事务。

[0250] 第六个因素可以是可以利用的资源的限制。这可以与给定事务的历史性尺寸进行比较，诸如指令的尺寸、占用尺寸、存储缓冲器利用率、不同的超高速缓存线等等。通常，随着事务尺寸的增加，事务的占用尺寸会增加。该占用包括执行该事务的资源。这在事务正被聚结时可以成为限制因素。因此，被聚结事务的占用的尺寸被用来确定是否能够进行进一步的聚结。例如，三个事务已经被聚结并且依然有过量的资源可以利用。第四个事务被识别并且分析处理该事务所需的资源。可利用资源量与处理该事务所需的资源量之间的比较能够就第四个事务是否应该被聚结产生决定。被聚结事务所需要的资源应该不超过可利用的资源，因为超过的话会导致终止。这种资源的一个实例是被聚结的事务所需要的超高速缓

存线的数量。这种资源的另一个实例是被聚结事务所需要的同余类(congruence class)的超高速缓存线的数量(一种8路的组+关联性(set+associative)超高速缓存不能支持需要超过组内的8个超高速缓存线的事务)。多线程处理器可以具有进一步资源限制以便容许事务在多个线程上同时执行。

[0251] 第七个因素可以是给定事务先前已经被中止过的次数。如果该事务的被记录的中止的数量超过阈值,则将该事务与另一个事务聚结就会是一种浪费。因此,如果该事务的被记录的中止的数量超高该阈值,则这些事务将不会被聚结。事务可以通过例如启动该事务的事务开始指令的地址被或者与每个事务相关联的标识符值被识别。

[0252] 在一些实施例中,就能够被聚结的事务的数量存在限制。该数量可以被硬连线、编程或者通过硬件动态地选择。在一些实施例中,如果存在事务中止,则该聚结功能确定所中止的事务是否为被聚结事务。如果该中止的事务是被聚结事务,则关于能够被聚结的事务的数量的极限可以被临时设置为一并且这些事务被但各地重试,即不被聚结。如果中止的事务不是被聚结事务,则该事务被处理为正常终止。在有些实施例中,关于能够被聚结的事务的数量可以每次给定聚结事务被终止则减少一。例如,如果五个事务被聚结并且该被聚结事务导致终止,则关于能够被聚结的事务的数量的极限将被减少为四。新的被聚结事务随后将被创建并使用新的较低极限被处理。不过,在某些情形下,根据依赖于用于聚结的依赖于机器的协议,某些事务可不被聚结。例如,如果存在事务T,并且前一事务T-1的指令超出某中管线阶段,例如将要完成,那么事务T就不会与事务T-1聚结。

[0253] 在有些实施例中,在每个最外面事务的边界处定义检验点。使用检验点边界来识别被聚结事务的事务占用的哪个部分对应于被聚结的每个相应事务。在中止情况下,这些检验点可被用作返回点。例如,如果T1、T2、T3以及T4为被聚结的非嵌套事务并且存在由于T4所导致的中止,则可以分析占用并且将该占用的与事务T4对应的部分隔离以及提交该占用的与T1、T2和T3对应的其余部分。因此,仅有事务T4作为最外面事务需要进行重新执行而不是整个聚结事务被重新执行。不过,如果T1-T4为嵌套事务则情况就不是这样。如果T1-T4包括嵌套事务,则T1-T3不会被提交。

[0254] 为了使用这种检验点来处理中止,事务可以被隔离到某些检验点边界。例如,如果在M个事务被聚结而事务T小于或等于M时存在影响事务T的内存交叉询问,则提交事务1到T-1并在事务T处再次恢复回到起始点。标记检验点边界的一个方法是通过最外面事务开始指令的指令地址。通常,最这事务尺寸增加,完成该事务所花的时间也相应增加,这增加了中止出现的机会,因为其他处理器/线程具有更多时间生成冲突。不过,通过使用检验点边界,由于事务尺寸增加而导致的性能的潜在降低能够得到缓解,因为只有直接受到冲突影响的事务将需要重新执行。

[0255] 在一个实施例中,存在两个事务之间的非事务性指令需要考虑。在某些实施例中,不是任何事务的一部分的指令当在架构上比那些指令老或年轻的事务被聚结时可以标记为事务性指令。这在中止条件导致返回到被聚结的事务的集合的第一事务时会是有用的。某些指令可以被限制被包括作为事务的一部分。不是或不能是任何事务的一部分的一些指令可以标记为不是任何事务的一部分。例如,在已经采用检验点边界的情况下,有已经被标记为事务的一部分的加载指令和被标记为非事务性的另一个加载指令。如果出现中止,则非事务性加载值能够被保存和跟踪。这意味着未来需要做的工作更少,因为被加载的值依

然存在。不过，事务性加载指令将必须被重复。在另一个实例中，存储指令为事务性的。该存储值的地址被添加到该事务的占用并且该被存储值被观察用于冲突。

[0256] 在一些实施例中，动态预测器被用于帮助确定一些事务是否应该被聚结。存在两种启动状态，即，事务历史的两种初始状态，其被用于这种预测器。在两种情况下，由于事务的完成而对事务历史做出改变。

[0257] 第一中启动状态为其中假设所有事务不能被聚结的历史状态。在被提示(prompted)时，聚结动作开始并且历史被更新以便反映被聚结的事务成功执行，即被提交，或者被中止。所更新的历史随后可以用于引导未来的聚结动作。该历史可以包括能够被用来确定给定的事务是否能够被将聚结的事务标识符(例如，事务开始地址)、事务的占用尺寸、事务的指令的尺寸以及其他数据。

[0258] 第二中启动状态为其中假设所有事务能够被聚结的状态。如上所述，聚结动作开始并且更新历史以便反映被聚结事务是否成功执行或被中止。在这种情形下，只要可能，事务都被聚结并且该信息被保存为历史一部分。该历史因此表示给定事务已经基于被聚结事务的事务标识符，例如，诸如关于事务开始指令地址的信息，被聚结。

[0259] 在某些实施例中，为一些事务设置一种标志，一种硬件保护比特，作为一种指示，以便有助于指导聚结动作。第一种标志可以被用于表示给定事务已经成功与架构上更年轻的(即，后来的)事务聚结，并且第二种标志可以用来表示给定事务已经成功与架构上更老的事务聚结。打标志的使用可以扩展到进一步提供聚结能力以及增加被聚结事务的完成，即减少被中止的聚结事务的数量。如果被聚结事务被中止，则可以检查中止原因。如果确定中止的原因是由于两个或更多事务的聚结导致的，则为更年轻的事务打上标志以表示该更年轻的事务不能与架构上(architecturally)更老(即，在前的)的事务聚结。而且，更老的事务可以被打上标志以便表示该更老的事务不能与架构上更年轻的事务聚结。在有些实施例中，采用评级(rating)系统来表示给定事务有多大可能能够成狗与另一个事务聚结而不会导致中止。例如，一个事务可以有弱、中等或强的聚结标志来表示给定事务被成功聚结的潜能。优选的是，这些标志每次在给定事务被处理时被更新。

[0260] 这种为事务打上标志的凡是对区分单个事务时有用的，并且有助于指导将来的聚结动作。例如，当T1、T2、T3以及T4事务被聚结在一起并且最终的中止是由于T1和T4被聚结的事实所导致的时，则仅有T4被打上标志以便表示其不能与架构上更老的事务聚结。可选择地，T1可以打上标志以表示T1不能与架构上更年轻的事务聚结，或者T1可以打上标志以表示在更年轻的事务已经与一个或更多其他事务聚结的情况下T1不能与架构上更年轻的事务聚结。那么，在未来聚结动作中，T1和T4的标志能够被用于确定那些事务是否应该与其他事务聚结。最后，如果聚结事务成功完成，则设置标志来表示该事务能够与更年轻或更老的事务聚结。下面将在图6的讨论中详细描述使用事务历史和标志来指导未来聚结动作的动态硬件预测器的实例功能。在该实施例中，纯粹在硬件中处理聚结成功的预测。在其他实施例中，该功能可以纯粹采用软件或者硬件和软件的组合来处理。

[0261] 在有些实施例中，可以有一些指示符被用于控制事务聚结动作。为此，在图7的讨论中详细描述了用于添加控制事务聚结动作的通用指示符的控制指示符程序。这种通用指示符被添加以便在可能的情况下指导硬件聚结事务。这种通用指示符可以在无需了解诸如当前资源可用性的硬件约束的情况下被添加。在一个实施例中，用于聚集的通用指示符在

纯粹软件实施例中被处理。在其他实施例中，该功能可以纯粹以硬件或硬件与软件的组合来处理。为了将语境添加到图7的讨论中，在下面段落中提供了各种指示符的通用描述。

[0262] 第一指示符可以通过软件指令的执行来设置，其指示硬件在可能情况下从该点向前(point forward)开始聚结事务。第二指示符可以通过软件指令的执行来设置，其指示硬件停止从该点向前聚结事务。第三指示符可以基于事务开始(事务开始指令)的在前的前缀来设置，以便指示该事务开始指令之后的事务可以被聚结。第四指示符可以基于事务开始(事务开始指令)的在前的前缀来设置，以便指示该事务开始指令之后的事务不能被聚结。第五指示符可以通过与该事务开始指令自身相关联的操作数来设置，其指示该事务能够被聚结。还有另一个指示符可以通过设置阈值的指令来设置。该阈值例如为能够被聚结的事务的最大数量或在将被聚结的两个事务之间的指令的最大数量。

[0263] 在某些情况中，聚结内存事务会对新能带来负面影响，因为所组合的事务的超高速缓存占用大于单个事务的每个占用。在润兴多个并行工作负载的多处理器环境中，随着占用尺寸增加，工作负载数据产生干扰的机会会增加，由此导致更高百分比的事务中止。因此，由于事务中止的较高比率而导致源自不同组合不同事务的性能增益会降低，或者甚至消失。不过，在其他情况中，聚结事务产生较高的新能而不会见到在事务中止方面的有任何增加。硬件自身可能不会告知其何时受益于聚结事务以及其何时不会受益。通过使用工作负载的动态代码分析(profiling)，软件可以布置代码和插入提示信息，使得硬件能够就聚结不同内存事务是否有益而做出决定。

[0264] 在某些实施例中，硬件提供支持硬件的运行时间仪表环境，其能够被用于实时地获取对目标运行程序的了解。不能从纯粹软件实时分析(profiling)获得的信息能够通过硬件直接获得，例如关于事务内存以及它们特性的信息。像Java®运行时间(Run-time)环境(JRE))的动态代码生成环境能够引导(steer)该硬件基础结构确定收集什么数据以及多么经常地收集该数据，并且因此能够创建“自调谐(self-tuned)”运行时间生成代码。

[0265] 在有些实施例中，运行时间仪表分析(profiling)代码可以被用于执行许多掩码。运行时间仪表分析代码可以被用于给予硬件一些提示以便聚结事务或使得它们保持独立。其可以被用于改变代码以便通过去除第一事务的事务结束指令和第二事务的事务开始指令来自动合并不同事务以及将两个事务之间的指令改变成非事务性指令。运行时间仪表分析代码能够被用来添加前缀指令，即，在例如事务的事务开始指令的前头的指示符，指示该事务需要被独立地处理并且不能够聚结。其还可以被用于改变事务开始指令或事务结束指令自身使得这些事务不能被聚结。

[0266] 在一些实施例中，分析器(profiler)程序聚结一些数据，诸如事务中止的原因、被聚结事务之间的指令中的距离、事务之间的指令的类型、任何指令是否被限制在事务中运行、每个事务中动态指令的数量以及动态指令占用的尺寸，例如所需存储超高速缓存缓冲器的尺寸、所使用的超高速缓存线的数量等等。随后该信息可以被用于通过使用该工作负载的动态代码分析来优化事务的聚结。软件可以布置代码并插入提示信息，使得硬件就聚结不同内存事务是否有益做出决定。在图8的讨论中将详细描述优化最外面事务的聚结以便使得性能增益最大化的聚结优化器700。

[0267] 现在参见图1-20

[0268] 本公开可以是一种系统、方法和/或计算机程序产品。该计算机程序产品可以包括

计算机可读存储介质(或媒介),其上具有计算机可读程序指令,用于使得处理器执行本公开的一些方面。

[0269] 计算机可读存储介质为有形器件,其能够保存或存储由指令执行设备使用的指令。该计算机可读存储介质可以是例如但不限于电子存储器件、磁性存储器件、光学存储器件、电磁存储器件、a半导体存储器件、或任何前述器件的适当的组合。计算机可读存储介质的更具体实例的非穷尽列表包括如下:便携式计算机磁盘、硬盘、随机存取存储器(RAM、只读存储器(ROM、可擦除可编程只读存储器(EPROM或闪存)、静态随机存储器(SRAM)、便携式光盘只读存储器(CD-ROM)、数字通用盘(DVD)、记忆棒、软盘、诸如穿孔卡片或在其上记录有指令的槽中的上升构件的机械编码器件、以及前述任何适当的组合。此处所使用的计算机可读存储介质并不应认为是瞬时信号本身,诸如无线电波或其他自由传播电磁波、通过波导或其他传输介质传播的电磁波(例如,穿过光线缆的光脉冲)或通过导线发送的电信号。

[0270] 此处描述的计算机可读程序指令可以从计算机可读存储介质下载到相应的计算/处理设备或者经由网络,例如英特网、局域网、广域网和/或无线网络,下载到外部计算机或外部存储器件。该网络可以包括铜传输电缆、无线传输路由器、防火墙、交换器、网关计算机和/或边缘(edge)服务器。每个计算/处理设备中的网络适配卡或网络接口从网络接收程序并且转发该程序用于存储在相应计算/处理设备中的计算机可读存储介质中。

[0271] 用于执行本公开的操作的计算机可读程序指令可以是汇编指令、指令集架构(ISA)指令、机器指令、依赖于机器的指令、微码、固件指令、状态设置(state-setting)数据、或以一种或多种编程语言的任何组合编写的源代码或目标代码,包括面向对象的编程与杨,诸如Java、Smalltalk、C++等,以及普通过程编程语言,诸如“C”编程语言或类似变沉国语言。该程序指令可以整体地在用户计算机上执行、部分在用户计算机上执行、作为独立软件包部分在用户计算机执行以及部分在远程计算机上执行或者整体在远程计算机或服务器上执行。在后一种情况下,远程计算机可以通过任何类型的网络连接到用户计算机,该网络包括局域网(LAN)或广域网(WAN)、或者连接到外部计算机(例如通过使用英特网服务供应商的英特网)

[0272] 此处参照根据本公开的实施例的流程图图示和/或方法、装置(系统)以及计算机程序产品的框图描述了本公开的一些方面。将理解到,流程图图示和/或框图的每个框以及流程图图示和/或框图中框的组合可以通过计算机可读程序指令来实现。

[0273] 这些计算机可读程序指令可以提供给通用计算机、专用计算机或其他可编程数据处理装置的处理器,以便产生一种机器,使得经由计算机或其他可编程数据处理装置执行的该指令创建按用于实现在流程图和/或框图框中所指定的功能/行为的手段。这些计算机可读程序指令也可以存储在能够引导计算机、可编程数据处理装置、和/或以特定方式起作用的其他设备中计算机可读存储介质中,使得其中存储有指令的计算机可读存储介质包括一种制品,其包括指令,该指令实现在流程图和/或框图框中所指定的功能/行为的各个方面。

[0274] 计算机可读程序指令还可以加载到计算机、其他可编程数据处理装置或其他设备上,以便使得一系列操作步骤将在计算机、其他可编程数据处理装置或其他设备执行,从而产涩会能够计算机实现的处理,使得在计算机、其他可编程数据处理装置或其他设备上执行的指令实现在流程图和/或框图框中所指定的功能/行为。

[0275] 附图中的流程图和框图图示了根据本公开的各种实施例的系统、方法以及计算机程序产品的可能实现方式的架构、功能和操作。因此，流程图或框图的每个框代表指令的模块、分段或部分，其包括一个或多个可执行指令，用于是西安所指定的逻辑功能。还应该注意到，在某些可替换实现方式中，框中所示的功能可以不按照图中所示的顺序进行。例如，依赖于所涉及的到功能，按照顺序显示的两个框实际上可以基本上并行地执行，或者这些框可以有时候按照相反的顺序执行。还应该注意到，框图和/或流程图的每个框以及框图和/或流程图中的框的组合可以通过执行指定功能或实现专用硬件和计算机指令的组合的专用的基于硬件的系统实现。

[0276] 在第一实施例中，聚结控制300、事务聚结操作400、动态预测500、控制指示器600以及聚结优化器700的方面包括在图2的某种硬件中或者存储在作为根据实施例的图2的CPU环境112的部分所包括的计算机可读存储介质(未示出)中，图1的裸片100和图2和3的CPU环境112的物理结构的部分包括：聚结控制300、事务聚结操作400、动态预测500、控制指示器600以及聚结优化器700的基于硬件的方面。

[0277] 在某些实施例中，聚结控制300、事务聚结操作400、动态预测500、控制指示器600以及聚结优化器700的功能和处理可整体或部分地组合以便形成在下面图4-8的讨论中和上述针对基于硬件事务执行的讨论中描述的实施例的精神和范围内的各种程序或对应的硬件结构。因此，本公开的实施例可以包括在图4-8的讨论中和上述讨论中描述的各种方法、计算机程序产品以及硬件构件。此外某些特征和功能可以通过在图11的讨论中所描述的硬件构件来进行。

[0278] 现在转向图4。图4图示了由聚结控制300的第一实施例执行操作动作，其控制一个最外面事务与另一个事务的聚结。在第一实施例中，聚结动作的控制由软件处理；不过，实际聚结本身由硬件处理。在其它实施例中，聚结动作的控制可以纯粹以硬件处理或者以硬件和软件的组合处理。在还有一些实施例中，实际聚结本省可以纯粹以硬件或以硬件和软件的组合处理。

[0279] 在操作305中，聚结控制300识别事务和与事务相关联的各种标志。在判定操作310中，聚结控制300确定聚结是否为激活。聚结控制300通过识别代码中的指令该聚结处理的激活的指示符来确定聚结是否为激活。例如，聚结控制程序可以通过识别由设置-事务-聚结-模式(STCM)指令或重置-事务-聚结-模式(RTCM)(参见关于STCM和RTCM对模式设置器1135更详细的讨论)设置的状态信息来确定聚结是否为激活。在其他实施例中，可以不存在指示符。在这种情况下，硬件假设每个最外面事务具有被聚结的可能。在该实施例中，这些聚结指示符由控制指示器600和聚结优化器700添加。如果这种指示符不存在，则聚结控制300确定聚结不是激活的并且前进到操作360(判定操作310，否分支)。如果存在这种指示符，则聚结控制300确定聚结是激活的并且前进到操作320(判定操作310，是分支)。

[0280] 在操作320中，聚结控制300识别在确定两个最外面事务何时将被聚结时将被使用的聚结因素。这包括阈值极限，诸如能够被聚结的事务的数量、最大的可容许被投射的每周期指令(IPC)以及给定事务先前可能被中止的次数、或前面所列的七个因素的任意因素。该聚结因素还包括两个事务之间的指令所述的类别以及事务处理可利用的资源。所识别的聚结因素随后在操作330中被聚结控制300用来确定两个最外面事务如果被聚结是否违反任何阈值或超过事务执行可利用的资源。例如，如果可以被聚结到单一事务的极限是三个事

务并且在所聚结事务中已经存在三个事务，则聚结控制300确定第四个事务不能被聚结。

[0281] 在判定操作340中，聚结控制300确定，基于操作330的确定，两个最外面事务是否能够被聚结。如果两个最外面事务不能被聚结(判定操作340，否分支)，则聚结控制300前进到操作360。如果两个最外面事务能够被聚结(判定操作340，是分支)，则聚结控制300前进到操作350。在操作350中，聚结控制300使用事务聚结操作400来聚结最外面事务以及，如果可能的话，两者之间任何指令。随后被聚结的事务使用事务聚结操作400被处理。

[0282] 在操作360中，聚结控制300处理这些事务。所处理的事务可以包括非聚结的事务以及未被聚结的指令。

[0283] 图5图释了由用于最外面事务的聚结的第一实施例所执行的操作动作，此处称之为事务聚结操作400。在下述的第一实施例中，事务的聚结纯粹以硬件处理，即操作动作被包括作为诸如中央处理单元的硬件的一部分并被该硬件执行。在其它实施例中，事务的聚结可以纯粹以软件或者以软件和硬件组合来处理。

[0284] 在事务聚结操作400的操作405中，硬件识别聚结指示，诸如由控制指示器600或聚结优化器700所添加的聚结指示符。在其它实施例中，如果不使用指示符，则硬件会默认聚结，即假设存在该指示符并前进到操作410。在事务聚结操作400的操作410中，任何所需的指令由硬件执行。在事务聚结操作400的操作415中，硬件识别最外面事务的事务开始指令，诸如TBEGIN或XBEGIN。与先前事务聚结的最外面事务开始指令依然在考虑有效控制(在某些架构中，这些都是指令文本的部分)的意义上被执行以便将它们应用到嵌套事务。不过，最外面事务开始指令没有增加嵌套深度。嵌套深度为一并且保持一。类似地，被聚结事务的最外面事务结束指令没有将嵌套深度递减到0，除非其是最后的事务结束指令。

[0285] 在事务聚结操作400的操作420中，硬件识别硬件的资源极限。硬件资源可以包括：例如，不是事务的一部分的存储缓冲器的尺寸以及其中的空闲资源、被监测用于现存事务的超高速缓存线的数量、以及未使用线监测器的资源的计数。在事务聚结操作400的操作425中，硬件执行该事务的指令。在事务聚结操作400的操作430中，硬件识别在该事务之外的任何指令，例如，在将被聚结的事务之后的指令。随后，在事务聚结操作400的判定操作435中，硬件确定位于该事务之外的指令是否能够与该事务的指令一起而被包括，即被处理。换句话说，确定在该事务之后的指令是否能够被包括作为该事务的一部分并因此被处理。通常，除非在其他情况下该指令被打上标志，给定指令能够被包括作为该事务的一部分，只要这么做不会超过资源极限。如果该指令不能被包括(判定操作435，否分支)，则硬件操作400前进到事务聚结操作400.的操作455。如果该指令能够被包括(判定操作435，是分支)，则硬件进行的操作前进到事务聚结操作400的判定操作440。

[0286] 在事务聚结操作400的判定步骤440，硬件确定是否已经到达资源极限以及是否有足够的资源可用于使得第二事务与第一事务聚结。例如，如果可利用资源的百分之九十五正被利用，则尽管资源极限还没有达到，也确定可用于使得第二事务与第一事务聚结的可利用资源不足。通常，位于第一事务之外的指令包括那些在第一和第二事务之间的指令。不过，在某些情况和实施例中，这种在外部的指令可以包括哪些不是事务的一部分的指令，并且可以在事务之前或之后。因此，处理这些指令所需的资源也必须考虑在内。如果确定资源极限已经达到或者可利用资源不足以将第二事务和第一事务聚结(判定操作440，是分支)，则硬件前进到事务聚结操作400的操作455。如果确定资源极限还没有达到并且可利用资源

足以将第二事务和第一事务聚结(判定操作440,否分支),则硬件前进到事务聚结操作400的操作445。

[0287] 在事务聚结操作400的操作445中,硬件识别第二事务,并且将该事务的资源要求与可利用资源的量进行比较。在事务聚结操作400的判定操作450中,硬件确定第二事务是否能够聚结到第一事务中。该确定考虑了能够被附到第一和第二事务上的任何标志。例如,如果第一或第二事务包括表示其不能与其他事务进行聚结的标志,则确定该事务不能被聚结。如果这些事务不能被聚结(判定操作445,否分支),则硬件前进到事务聚结操作400的操作455。在事务聚结操作400的操作455中,硬件结束当前事务聚结动作并且该事务与已经包括在该事务中的任何指令一起被处理。

[0288] 如果确定该事务能够被聚结(判定操作450,是分支),则硬件前进到事务聚结操作400的操作460。在事务聚结操作400的操作460中,硬件事务性地执行第二事务的指令。例如在有些情况下,硬件不完全处理第一事务的事务结束指令和第二事务的事务开始指令。不过,第一事务的事务结束指令的或第二事务的事务开始指令的有些有效控制可以被处理,由此有效地处理该第二事务,就像其是第一事务的部分一样。

[0289] 在事务聚结操作400的判定操作465中,硬件确定是否因为处理被聚结的事务而出现中止条件。如果终止条件出现(判定操作465,是分支),则该硬件将被聚结的事务回滚到所中止的被聚结事务的第一事务,并且在操作470中由该硬件执行中止逻辑。如果终止条件未出现(判定操作465,否分支),则该硬件识别第二事务的事务结束指令,并且在步骤475中提交所聚结的事务。在有些情形下,可以在执行操作465之前重复进行操作405-460,由此将多个事务和指令聚结成所述聚结事务。

[0290] 图6图释了由动态预测500的第一实施例执行的操作动作,该动态预测500使用事务历史和标志来指导为了的聚结动作。

[0291] 在操作505中,动态预测500识别将被应用到所述历史上的启动状态。如上面所详细描述的,事务性内存系统能够村粗和启动状态和历史。启动状态为一种对所有事务都容许进行聚结的假设或对所有事务都不容许进行聚结。接着,在操作510中,动态预测500存取该历史本身并将启动状态应用到该输入项。在操作515中,动态预测500识别已经被处理过的聚结事务。

[0292] 在判定操作520中,动态预测500确定聚结事务是否经历了中止。如果聚结事务确实经历了中止(判定操作520,是分支),则动态预测500在操作525中确定中止的原因。如果聚结事务没有经历了中止(判定操作520,否分支),则动态预测500前进到操作530。在操作530中,动态预测500基于中止原因或所述聚结事务的成功执行应用一些标志。例如,没有中止和所述聚结事务被提交,因此,动态预测500相应地应用标志。在另一个实例中,被聚结的一对事务没有中止。因此,动态预测500将表示应用到被聚结的这些事务以便表示它们被成功聚结并且作为一个聚结事务被处理。在又一个实例中,包括四个被聚结的事务的聚结事务经历中止。动态预测500识别到该中止是由被聚结的第三和第四事务所导致。动态预测500对第四事务打上标志以表示其不能与更老的事务聚结但是能够与更年轻的事务聚结。动态预测500对第三事务打上标志以便表示其不能与更年轻事务聚结但是能够与更老的事务聚结。最后,动态预测500对第一和第二事务打上标志以便表示它们能够与更老和更年轻事务聚结。在有些实施例中,动态预测500可以对事务打上标志以便表示其不能与更年轻或

更老的事务聚结。该标志甚至可以采用能够被聚结的事务的数量来进一步限定(qualified)。

[0293] 在操作535中，动态预测500更新该历史以便表示各种事务已经被处理为被聚结的事务并且包括该处理的结果。例如，该历史被更新以便表示给定的一组五个事务被成功聚结并被处理。因此，对于聚结动作，那些事务的将来情况会被识别。

[0294] 在一些实施例中，动态预测500使用被更新的历史来预测聚结之前还没有被聚结过的最外面事务的结果。例如，第一和第二最外面事务被聚结并且所获得的聚结事务被提交。动态预测500随后识别与第一和第二最外面事务在尺寸、指令类型、占用等类似的第三和第四最外面事务。基于这种相似性，动态预测500确定如果被聚结的话第三和第四最外面事务将很可能提交并且因此添加一些标志和一些聚结指令。随后，基于聚结的结果，动态预测500相应地更新该历史。

[0295] 在有些实施例中，基于被更新的历史，动态预测500向聚结优化器700发送信号以便增加或减少某些阈值，由此控制聚结动作。

[0296] 图7图释了由控制指示器600的第一实施例执行的操作动作，控制指示器600用于识别和处理控制事务聚结动作的指示符。在一些实施例中，控制指示器600为纯粹的硬件。在有些实施例中，控制指示器600为纯粹的软件。还在其他实施例中，控制指示器600包括硬件和软件两者的组合。在某些实施例中，控制指示器600可以不在事务中插入指令，但是可以替代设置预测器状态

[0297] 在操作610中，控制指示器600识别将被处理器的事务。在控制指示器600的动态软件实施例中，一些在事务完成中展示过多暂停(stall)，即，导致延迟的事务被识别。因此，聚结指令或机器状态可以被添加，使得展现过多暂停的事务不与更年轻事务聚结。在控制指示器600的静态软件实施例中，一组因素，诸如在图4的操作320和330中使用的那些因素，可以被用来识别能够被聚结的事务。例如，指令的阈值数量可以用来识别那些彼此足够接近的事务以便容许聚结。

[0298] 在一个实施例中，一旦一些事务被识别，则如在操作620中所示，控制指示器600就将适当的聚结指令插入代码。在有些实施例中，控制指示器600设置状态信息。控制指示器600插入聚结指令(像，例如，事务开始指令的前部前缀，或者事务开始指令自身中的变量(argument))，并且随后使得硬件确定是否进行聚结。因此，如果没有聚结指令，则硬件就不会尝试进行聚结。如果有聚结指令，则硬件确定是否进行聚结。在多种情况下，如果可能，这导致硬件尝试聚结这些事务。随后这些事务与这些聚结指令一起被处理。然后在操作630中，控制指示器600从该事务处理中读出仪表(instrumentation)数据，例如关于在运行时间期间被采样的指令的数据。这包括在事务中止代码处理中的或在运行时间仪表逻辑中的仪表数据。

[0299] 在判定操作640中，控制指示器600确定是否存在过量的聚结事务中止。这通常基于预设的阈值；不过这还可能使用被调整到最大性能的动态阈值来确定。如果已经存在过量的聚结事务中止(判定操作640，是分支)，则在操作650中，控制指示器600去除该聚结指令以便减少未来中止的数量。在其它实施例中，如果已经存在过量的聚结事务中止，聚结指令会被控制指示器600留在代码中但是硬件会忽略它。如果还没有过量的聚结事务中止(判定操作640，否分支)，则控制指示器600会让该聚结指令保持原样。在有些实施例中，如果还

没有过量的聚结事务中止，则控制指示器600添加额外的聚结指令或发送信号给例如聚结优化器700，以便增加某些阈值，由此增加聚结动作或被聚结事务的尺寸。

[0300] 现在转向图8和9两者，将讨论聚结优化器和相关联环境。为了便于理解，参照图9中所包括的组件来描述聚结优化器700所利用的操作处理。在此被标记为JRE 900的、支持聚结优化器700的操作的Java®运行时间环境(JRE)的实例在图9中被显示为框图。

[0301] 根据第一实施例，图8图释了聚结优化器700的第一实施例执行的操作动作，聚结优化器700优化最外面事务的聚结，以便使得性能增益最大化。在第一实施例中，聚结优化器700配置为在诸如JRE 900的Java®运行时间环境(JRE)中操作。聚结优化器700具有对Just-In-Time(及时(JIT))分析器(profiler)910、优化器920、代码生成器930的接入口，并且与图9的诸如CPU 940以及预分配存储器950的硬件连接。

[0302] 在操作710中，使用Just-In-Time(JIT)分析器910、聚结优化器700在预分配存储器950中建立存储区域，其中硬件可以写入运行时间聚集(gathered)的仪表信息。在操作720中，聚结优化器700使用JIT分析器910将指令插入该应用中以便开启和关闭分析各种代码区域的运行时间，并且建立硬件中用于这种分析所需的环境。在操作730中，聚结优化器700使用代码生成器930将运行时间仪表目录命令(directive)并入其在时间硬件中运行的汇编代码中。

[0303] 响应于接收到运行时间仪表目录命令，硬件建立如这些新命令(directive)所指令的其内部仪表控制。在操作740中，目标信息，例如，超高速缓存确实统计，随后由CPU 940聚集，并且生成如由JIT分析器910建立的指令采样。硬件仪表控制，通过内部固件和特定硬件支持，将在运行时间期间所采样的指令的理想仪表数据写到预先分配存储器950中的由JIT分析器910在起初建立的存储区域。

[0304] 在步骤750中，聚结优化器700使用JIT分析器910分析硬件实时提供的聚集的仪表数据。所述聚集的仪表数据可以包括事务中止的原因、所聚结事务之间的指令中的距离、事务之间指令的类型、是否有人指令被限制在事务中运行、每个事务中动态指令的数量以及动态指令占用的尺寸，例如所需的存储超高速缓存的尺寸、所使用的超高速缓存线的数量等等。在操作760中，聚结优化器700利用算法来操纵(steer)优化器920来生成更多有效代码

[0305] 随后在操作770中，聚结优化器700使用JIT分析器910来周期性复查所聚集的仪表数据以及在汇编代码的运行时间环境中的任何改变，并且重新操纵优化器920以便根据任何新的运行时间仪表改变来调整聚结指令的优化。

[0306] 图10描绘了计算设备1000的组件的框图，该计算设备根据本公开的第一图示实施例执行聚结控制300、动态预测500、控制指示器600和聚结优化器700以及事务聚结操作400任何软件方面。应该理解的是，图10仅仅提供了一种实施方式的图释并且不意味着对可以实现不同实施例的环境的任何限制。可以对所示的环境进行多种修改。

[0307] 计算设备1000包括通信结构1002，其提供了在计算机处理器1004、内存1006、计算机刻度存储介质1008、通信单元1010以及输入/输出(I/O)接口1012之间的通讯。通信结构1002能够采用任何被设计用来使得数据和/或控制信息在系统内的处理器(诸如微处理器、通信和网络处理器等等)、系统内存、外围设备以及任何其他硬件组件之间通过的架构实现。例如，通信结构1002可以采用一条或多条总线实现。

[0308] 内存1006和计算机可读存储介质1008为计算机可读存储介质。在第一实施例中，内存1006包括随机存取存储器(RAM)1014和超高速缓存存储器1016。通常内存1006可以包括合适的易失性或非易失性计算机可读存储介质。

[0309] 根据本公开的第一实施例，聚结控制300、动态预测500、控制指示器600和聚结优化器700以及事务聚结操作400任何软件方面存储在计算机可读介质1008中用于由一个或多个相应计算机处理器1004经由内存1006的一个或多个存储区域来执行。在第一实施例中，计算机可读存储介质1008包括磁性硬盘驱动器。可替换地，或除了磁性硬盘驱动器之外，计算机可读存储介质1008还包括固态硬驱、半导体存储器件、只读存储器(ROM)、可擦除可编程只读存储器(EPROM)、闪存、或能够存储程序指令或数字信息的任何其他计算机可读存储介质。

[0310] 计算机可读存储介质1008所使用的介质还可以被移除。例如，可移除硬盘驱动器可用于计算机可读存储介质1008。其他实例包括能够被插入驱动器以便转换到还是计算机可读存储介质1008的一部分的另一个计算机可读存储介质上的光学和磁性盘、指状驱动器以及智能卡。

[0311] 在一些实例中，通信单元1010提供了与其他数据处理系统或设备的通信。在一些实例中，通信单元1010包括一个或多个网络接口卡。通信单元1010可以通过使用物理和无线通信连接之一或两者提供通信。聚结控制300、动态预测500、控制指示器600和聚结优化器700以及事务聚结操作400任何软件方面可以通过通信单元1010下载到计算机可读存储介质1008

[0312] I/O接口1012容许与可以连接到计算设备1000的其他设备的数据输入输出。例如，I/O接口1012可以提供到外被设备1018，诸如键盘、键区、触屏、和/或一些其他适当输入设备的外部设备1018的连接。外部设备1018还可以包括便携式计算机可读介质，诸如，例如，指状驱动器、便携式光学或磁性盘、以及存储卡。用来实现本公开实施例的软件和数据，例如聚结控制300、动态预测500、控制指示器600和聚结优化器700以及事务聚结操作400任何软件方面，可以存储在这种便携式计算机可读存储介质中并且可以经由I/O接口1012加载到计算机可读存储介质1008。I/O接口1012还连接到显示器1020。

[0313] 显示器1020提供了一种机构来向用户显示数据并且可以为例如计算机监视器。

[0314] 图11描绘了根据各种实施例的作为图10的处理器1004的一部分的各种硬件构件。在某些实施例中，图11的某些硬件构件可以整体或部分采用软件等同物来替换。而且，在有些实施例中，图11中描述的某些硬件构件作为的聚结控制300、动态预测500、控制指示器600聚结优化器700、事务聚结操作400一部分被包括或者被配置为支持它们的功能。

[0315] 事务处理器1105为硬件构件，被配置为执行最外面事务的事务开始指令、事务本身、事务结束指令、被聚结的事务、在事务之间的非事务性指令、聚集指令以及所执行事务的内存提交。此外，事务处理器1105基于执行事务开始指令处理与该事务开始指令相关联的相应事务。此外，如果通过聚结确定器1110确定最外面事务将不能与另一事务聚结，则事务处理器1105基于遇到该最外面事务的事务结束指令将最外面事务的第一存储数据提交个内存。

[0316] 聚结确定器1110是一种硬件构件，配置为基于事务处理器1105遇到第一最外面事务的第一事务结束指令来确定第一事务是否将与第二最外面事务聚结。聚结确定器1110基

于以下至少一个来确定第一最外面事务将与第二最外面事务聚结：所述第一最外面事务和第二最外面事务之间的指令的数量、处理所述第一最外面事务和第二最外面事务所需的时间段、处理所述第一最外面事务和第二最外面事务所需的资源量、能够被聚结的事务的最大数量、以及聚结导致中止状态的最外面事务的历史。基于一个或多个聚结指令的执行，聚结确定器1110确定两个最外面事务是否将被聚结。聚结确定器1110还基于所执行的仪表程序的结果确定两个最外面事务将是否被聚结。如果聚结确定器1110确定相关联程序的多个事务中的第一最外面事务和第二最外面事务应该被聚结，则那些事务使用事务聚结器1115将被聚结。聚结确定器1110还至少部分基于聚结历史来确定两个最外面事务将被聚结。

[0317] 阈值设定器1111是一种硬件，配置来至少部分基于由事务处理器1105执行的一个或多个聚结指令来为聚集事务设置阈值。这种阈值由阈值设定器1111至少部分基于预先存在的哦阈值、聚结历史以及用于处理事务的资源极限来确定。

[0318] 事务聚结器1115是一种硬件，配置为基于聚结确定器1110所作出的第一最外面事务将与第二最外面事务聚结的确定来聚结事务。为了聚结事务，事务聚结器1115执行一种方法，包括：(i)在处理第二最外面事务之前不将第一最外面事务的存储数据提交到内存；(ii)基于遇到第二最外面事务的第二事务开始指令，处理第二事务；(iii)基于所述处理第二事务遇到第二最外面事务的第二事务结束指令；以及(iv)将被聚结的第一最外面事务和第二最外面事务的存储数据提交给内存。如果指令处理器1120确定在第一最外面事务之后的一个或多个指令被处理为聚结事务的一部分，则事务聚结器1115将第一最外面事务之后的所述一个或多个指令执行成被聚结的第一最外面事务和第二最外面事务的一部分。随后作为将被聚结的第一最外面事务和第二最外面事务的存储数据提交给内存的一部分，事务处理器1105将所述一个或多个指令的存储数据提交给内存。

[0319] 指令识别器1120是一种硬件构件，配置为识别在第一最外面事务之后并且在第二最外面事务之前的一个或多个指令。指令识别器112还基于以下至少之一确定所述一个或多个指令是否可以处理成为该聚结事务的一部分：处理所述一个或多个指令所需的时间段、处理所述一个或多个指令所需的资源量以及所述一个或多个指令手哦书的类别。

[0320] 占用处理器1125是一种硬件构件，配置为将一个或多个标记添加到所聚结的第一最外面事务和第二最外面事务的内存占用以便指示内存的用于处理所述第一最外面事务和第二最外面事务的相应区域。内存占用包括在处理所述第一最外面事务和第二最外面事务期间从其读取和向其写入的内存地址，并且其中所述标记被用于(i)在中止情况下回滚所聚结的事务以及(ii)控制最外面事务的聚结。

[0321] 聚结指令处理器1130是硬件构件，配置为执行用于控制多个最外面事务的聚结的一个或多个聚结指令。所述一个或多个聚结指令在被执行时指示聚结确定器1110哪个最外面事务能够被聚结。所述一个或多个聚结指令可以包括与最外面事务的的事务开始指令相关联的聚结前缀和与最外面事务事务开始指令相关联的聚结变量中的一个或两者。所述一个或多个聚结指令可以包括与最外面事务的的事务结束指令相关联的聚结前缀和与最外面事务事务结束指令相关联的聚结变量中的一个或两者。

[0322] 模式设定器1135是一种硬件构件，配置为执行设置-事务-聚结-模式(STCM)指令和重置-事务-聚结-模式(RTCM)指令。执行STCM指令导致处理器进入STCM模式。进入STCM模式表示以下至少之一：(i)随后聚结指令被启用以便使得事务将被聚结、(ii)可在将被聚结

的两个最外面事务之间爱你存在的指令的最大数量、以及(iii)可在将被聚结的两个最外面事务之间爱你存在的指令的类型。相反，执行RTCM指令使得处理器从STCM模拟出来。退出STCM模式导致以下之一或多种结果：(i)停止随后聚结指令的执行、(ii)修改存在于两个最外面事务的指令的数量、以及(iii)修改存在于两个最外面事务的指令的类型以便基于由聚结确定器1110确定聚结将会超过或聚结先前已经超过由阈值设定器1111生成的阈值而禁止对给定的最外面事务进行聚结。其中超过阈值表示以下任意一个：通过将被聚结的最外面事务展现暂停、在将被聚结的两个最外面事务之间的指令的最大数量、处理将被聚结的最外面事务所需的最大时间段、处理将被聚结的最外面事务所需的资源量、能够被聚结的最外面事务最大数量、聚结导致中止状态的特定最外面事务的可容许实例的数量、以及先前经历中止的聚结事务的历史。

[0323] 运行时间监测器1145是一种硬件构件，配置为执行运行时间仪表程序，用于监测和修改据偶多个事务的相关联程序。基于执行该相关联程序的事务，运行时间仪表程序动态获取与该相关联程序的执行相关联的仪表信息。运行时间监测器1145还生成用于该相关联程序的连续执行的运行时间分析的环境以便获取仪表信息。运行时间监测器1145使用所获得的仪表信息以及该相关联程序的运行时间环境中的改变分析该相关联程序的执行。运行时间监测器1145处理运行时间仪表程序的运行实时仪表命令(directive)并至少部分基于所处理的运行实时仪表命令来配置该运行时间仪表的控制。该运行时间仪表包括一个或多个指令，用于获取与相关联程序的事务的执行相关的仪表信息。所聚集的仪表信息包括以下之一或多个：(i)被聚结的最外面事务中止的原因以及其已经接收到的中止的数量、(ii)在两个被聚结最外面事务之间的指令的数量、(iii)在两个被聚结的最外面事务之间指令的类型、(iv)在两个被聚结的最外面事务之间的任何指令是否被限制在给定最外面事务中运行、(v)在给定最外面事务中的动态指令的数量、以及(vi)在给定最外面事务中动态指令的占用的尺寸。

[0324] 事务执行优化器1150是一种硬件构件，配置为基于所获得仪表信息动态地修改该相关联程序的事务的连续执行，以便优化事务性的执行(TX)。运行时间仪表程序通过向该相关联程序添加一个或多个聚结指令来修改该相关联程序的连续执行，以便至少部分基于对所聚集仪表信息的分析来控制对多个事务中的一个或多个的聚集。所述一个或多个聚结指令包括以下之一或多个：(i)用于聚结最外面事务的指令、(ii)用于去除事务开始指令或事务结束指令或者不执行事务开始指令或事务结束指令的指令、(iii)用于在已经聚结的指令的数量大于阈值的情况下修改事务开始指令或事务结束指令使得该相关联的最外面事务不能与一种类型的最外面事务聚结的指令、(iv)用于在已经聚结的指令的数量大于阈值的情况下指示特定最外面事务将不与一种类型的最外面事务聚结的指令、(v)用于将非事务性指令处理为事务性指令的指令、(vi)一种停止聚结最外面事务的指令、以及(vii)一种指定被聚结的最外面事务最大可容许数量的指令。

[0325] 事务中止确定器1155是一种硬件构件，配置为确定来自该相关联程序的、被聚结了的第一多个最外面事务是否经历过中止。该第一多个最外面事务包括第一事务的第一实例(instance)。

[0326] 历史更新器1160是一种硬件构件，配置为通过处理器更新该相关联程序的历史以便反映该确定的结果。基于事务中止确定器1155确定该第一多个最外面事务经历或未经历

过中止，使用历史更新器1160来更新该相关联程序的历史，以便表示所述第一多个最外面事务是经历或是未经历过中止。

[0327] 事务标志器1165为一种硬件构件，配置为基于事务中止确定器1155确定所述多个最外面事务没有经历过中止，则为第一最外面事务打上标志以便反映所述多个最外面事务没有经过中止的确定。基于所述第一多个最外面事务确实经过中止的确定，通过事务标志器1165至少部分基于中止的原因为第一最外面事务打上标志。

[0328] 指令添加器1170是一种硬件构件，配置为至少部分基于包括第一最外面事务的第一实例的相关联程序的被更新的历史，来确定是否添加指令来聚结第一最外面事务的第二实例。基于表示用于聚结第一最外面事务的第二实例的指令将被添加的确定，指令添加器1170使得相关联程序的执行向第二多个最外面事务添加一个或多个指令以便聚结第一最外面事务的第二实例。

[0329] 聚结预测器1175是一种硬件构件，配置为至少部分基于相关联程序的更新历史，来预测最外面事务的聚结结果。这种预测可以基于被聚结的事务的先前实例的结果。这种预测还可以基于各种事务之间的相似性。

[0330] 历史1180是一种硬件构件，配置为存储数据和对数据进行通讯，该数据可用于预测聚结最外面事务的结局(outcome)。这种数据可以包括该相关联程序的更新历史以及被聚结事务的先前实例的结果。这种数据可以被聚结预测器1175用来预测聚结最外面事务的结果。包括在历史1180中的数据可以例如被诸如历史更新器1160的硬件构件更新。

[0331] 图12描绘了在一个实施例中嵌套的事务的实例。图12意图帮助读者认识到嵌套的事务与聚结的最外面事务之间的不同。嵌套的事务的实例是在1206内的事务。每次在遇到TBEGIN时，嵌套深度递增1，而每次在遇到TEND是，嵌套深度递减1。最外面TBEGIN为将深度从0递增到1的指令，而最外面TEND是将深度从1递减到0的指令。

[0332] 现有的方案被用于处理嵌套事务。相反，在图4-8中描述的方案解决最外面事务的聚结。在图9中，可以理解到的是将1206中的最外面事务TB1-TE1与1208中最外面事务TB1-TE1的组合。这些指令能够基于每个线程的程序指令顺序1202(即，程序发出这些指令的顺序)按顺序执行。对于每个事务开始(TBEGIN)指令1210，嵌套深度1204(i)增加1。对于每个事务结束(TEND)指令1212，嵌套深度1204减少1。当第一外面事务1206启动时，最外面事务计数(j)增加1。当第二外面事务1208启动时，j增加1。随着第一外部事务1206和第二外部事务1208中的每一个被完成，j递减1。

[0333] 图13图释了根据本公开的实施例的一种由动态编译器执行的操作动作的方法1300，该动态编译器执行事务聚结处理的一些方面。应该注意到的是，在某些实施例中这种动态编译器能够用于聚结最外面事务而不需要已经被配置用于聚结最外面事务的硬件。

[0334] 在有些实施例中，这种动态编译器包含了聚结控制300、动态预测500、控制指示器600和聚结优化器700的各个方面以及事务聚结操作400的软件方面，以便容许动态编译器能够做出决定和生成如在方法1300的讨论中描述的指令。在其他实施例中，聚结控制300、动态预测500、控制指示器600和聚结优化器700以及事务聚结操作400的方面与这种动态编译器结合并通讯，由此使得动态编译器执行方法1300的操作。

[0335] 方法1300图释了在没有提供对受控硬件事务聚结的硬件支持时事务聚结的软件实施方式。在一个实施例中，在每个代码片段的固定施加量之后发动代码生成和再优化。代

码片段可以是动态优化器可作用的多个已知区域中的任何一个。在先前的代码生成中，这些包括：跟踪文件、树区域、函数、热区域等或者动态代码优化和/或代码生成方法能够对其执行的任何其他指令组。

[0336] 在操作1310中，非事务性指令由动态编译器生成以便非事务性地执行。在判定操作1320中，确定是否已经到达非事务性指令代码的末端，即代码片段的所有指令是否已经由动态编译器生成以及是否已经到达代码片段的末端。如果确定已经达到代码的末端(判定操作1320，是分支)，则方法1300结束。如果确定还没有到达代码的末端(判定操作1320，否分支)，则方法1300前进到操作1330.

[0337] 在操作1330中，响应于识别事务的开始生成事务开始指令。在操作1340中，生曾事务指令。在判定操作1350中，确定是否容许该事务与随后的事务聚结。

[0338] 如果确定不容许聚结该事务的(判定操作1350，否分支)，则方法1300前进到操作1360。在操作1360中，响应于识别到该事务的结束，则生成事务结束指令。随后方法1300前进到1310。如果确定容许该事务的聚结(判定操作1350，是分支)，则方法1300前进到操作1370。

[0339] 在操作1370中，以事务性指令形式生成非事务性指令。这种指令在将被聚结的第一事务之后并在下一事务之前。换句话说，生成在将被聚结的第一和死而事务之间的非事务性指令作为聚结事务的一部分。在至少一个实施例中，用于确定是否容许进行事务聚结的分析包括插入指令的分析以确保所有插入指令能够被事务性地执行。在一种替换实施例中，在是否容许事务聚结的起初确定期间不进行这样的确定。在这样的替代实施例中，当遇到不能被事务性执行的指令时，则如在操作1360中所述，在生成非事务性指令之前，通过生成事务结束指令来结束该事务。

[0340] 在方法1300的判定操作1380中，确定是否已经到达下一个事务的开始。如果确定已经到达下一个事务的开始(判定操作1380，是分支)，则方法前进到操作1340。如果确定还没有到达下一个事务的开始(判定操作1380，否分支)，则方法1300前进到操作1370。

[0341] 本公开的实施例可以在适于存储和/或执行程序代码的数据处理系统中实现，其包括包括通过系统总线直接或间接连接到内存元件的至少一个处理器。该内存元件包括例如在程序代码的实际执行期间所采用的本地内存、大容量存储器以及提供至少一些程序代码的临时存储以便减少代码在执行期间从大容量存储器中被检索的次数的超高速缓冲存储器。

[0342] 图14描绘了根据本公开实施例的流程图，其图释了一种实施例1400，用于控制最外面内存实例的聚结。

[0343] 该实施例由处理器执行1410一个或多个聚结指令，用于控制多个最外面事务的聚结。该实施例由处理器发起1420相关联程序的执行。该实施例基于所述一个或多个聚结指令的执行，由处理器确定1420两个最外面事务是否将被聚结。该实施例基于两个最外面事务将被聚结的确定，通过处理器聚结包含在多个最外面事务中的至少两个最外面事务。

[0344] 如图15所示，作为执行1410所的一部分在一个实施例中包括，所述一个或多个聚结指令在被执行时表示1500哪个最外面事务能够被聚结。

[0345] 如图16所示，作为聚结1430的一部分在一个实施例中包括，所述聚结使得针对第一事务将内存存储数据到内存的提交将在第二事务的事务执行(TX)结束处进行。

[0346] 如图17所示，作为一个或多个聚结指令的一部分包括1500，所述一个或多个聚结指令包括1700与最外面事务的事务开始指令相关联的聚结前缀以及与最外面事务的事务开始指令相关联的聚结变量中的一个或两者。

[0347] 如图18所示，作为一个或多个聚结指令的一部分包括1500，所述一个或多个聚结指令包括1800与最外面事务的事务结束指令相关联的聚结前缀以及与最外面事务的事务结束指令相关联的聚结变量中的一个或两者。

[0348] 如图19所示，在1400之前，实施例由处理器执行1900执行设置-事务-聚结-模式(STCM)指令，该STCM指令使得处理器进入STCM模式，其中进入STCM模式表示以下至少之一：(i)随后的聚结指令被启用以便使得事务将被聚结、(ii)能够存在于两个将被聚结的最外面事务之间的指令的最大数量、以及(iii)能够存在于两个将被聚结的最外面事务之间的指令的类型。

[0349] 如图20所示，在1400之前，实施例由处理器执行2100执行重置-事务-聚结-模式(RTCM)指令，该RTCM指令使得处理器从STCM模式退出，其中从STCM模退出导致以下一个或多个：(i)停止执行随后的聚结指令、(ii)修改存在于两个最外面事务之间的指令的数量、以及(iii)修改存在于两个最外面事务之间的指令的类型。

[0350] 如图21所示，在执行1410之后，基于被执行的所述一个或多个聚结只指令，由处理器设置2210阈值。该实施例由处理器基于这种聚结是否将超过该阈值而确定2220两个最外面事务是否将被聚结。

[0351] 如图22所示，实施例通过处理器基于这种聚结是否将超过该阈值确定2220两个最外面事务是否将被聚结。在该实施例中，基于确定2300所述聚结将超过表示以下之一的所述阈值，针对给定最外面事务禁止聚结具体最外面事务：由将被聚结的最外面事务所展现的暂停、在将被聚结的两个最外面事务之间可能存在的指令的最大数量、处理被聚结的最外面事务所需的最大时间段、处理被聚结的最外面事务所需的资源量、能够被聚结最外面事务的最大数量、聚结导致中止状态的特定最外面事务的可容许实例的数量、以及先前经历过中止的被聚结事务的历史。

[0352] 输入/输出或I/O设备(包括但不限于，键盘、显示器、指点设备、DASD、磁带、CD、DVD、指状驱动器以及其他存储介质等等)可以直接或通过插入的I/O控制器连接到系统。网络适配器也可以连接到系统以便使得数据处理系统能够通过插入的专用或公用网络连接到其他数据处理系统或远程打印机或存储设备。调制解调器、线缆调制解调器以及以太网卡就是这种可利用类型的网络适配器中的少数几种。

[0353] 本公开的一个或几个能力可以以软件、固件、硬件或其组合来实现。而且一个或几个性能可以被模拟。

[0354] 本公开的一个或几个方面可以包括一种具有例如图10的计算机可读存储介质1008的制品(例如，一种或几种计算机程序产品)中。该介质中已经包含了例如计算机可读程序代码(指令)，以便提供和实现本公开的功能。该制品可以包括作为计算机系统的一部分或者作为独立产品。

[0355] 实施例可以是一种计算机程序产品，用于使得处理器电路能够执行本公开的要素，该计算机程序产品包括可由处理电路读取并能够存储由该处理电路执行以便执行一种方法的计算机可读存储介质。

[0356] 该计算机可读存储介质可以有形的、非瞬态的存储介质，其上级路由指令，用于使得处理器电路执行一种方法。该“计算机可读存储介质”为非瞬态的，至少因一旦该指令记录在该介质上，则所记录的指令能够随后在与记录时间无关的时间被处理器电路读取一次或多次。该“计算机可读存储介质”为非瞬态的包括一些仅在被通电时保留被记录信息的设备(已失性设备)以及一些与通电无关地保留被记录信息的设备(非易失性设备)。“非瞬态存储介质”的示例非穷尽列表包括但不限于例如：

[0357] 半导体存储器件，包括例如，诸如RAM或存储阵列，或存储电路，诸如其上记录有指令的锁存器；

[0358] 机械编码设备，诸如穿孔卡片或在其上记录有指令的槽中的上升构件；

[0359] 光学可读器件，诸如其上记录有指令的CD或DVD；以及

[0360] 磁性编码器件，诸如其上记录有指令的磁带或磁盘。

[0361] 计算机可读存储介质的实例的非穷尽性列表包括：便携式计算机盘、硬盘、随机存取存储器(RAM)、只读存储器(ROM)、可擦除可编程只读存储器(EPROM或闪存)、便携式紧凑盘只读存储器(CD-ROM)。

[0362] 尽管不此处已经给出了一个或多个实例，但是，这些都仅仅是举例。在不图例本实施例的精神的情况下可以有多种变化形式。例如，处理环境除了此处所给出的实例之外可以包括和/或受益于本发明的一个或多个方面。而且该环境不需要基于z/Architecture®，而是可替代地基于例如由IBM®、Intel®、Sun Microsystem(太阳微系统)以及其他公司其他架构。还有，该环境可以例如包括多个处理器，可以被分割开，和/或可以连接到其它系统。

[0363] 如此处所使用的，术语“获取”包括但不限于，取得、接收、具有、提供、被提供、创建、开发等等。

[0364] 本公开的一个或多个方面的能力可以采用软件、固件、硬件或其一些组合来实现。可以提供由机器读取的至少一个程序存储器件含有至少一个程序，其指令可以由该机器执行以便执行本公开的能力

[0365] 在此处所描绘的流程图仅仅是举例。在不脱离本公开的精神的情况下可以对这些示意图或步骤(或操作)有多种变化形式。例如，这些步骤可以以不同的顺序来执行，或者可以添加或修改步骤。所有这些变化形式都被认为是要求保护的本公开的一部分。

[0366] 尽管此处详细描绘和描述了一些优选实施例，但是本领域技术人员应该理解到，在不脱离本公开的精神的情况下可以做出各种修改、添加、替代等等，并因此这些修改、添加、替代也被认为在后面的权利要求所限定的本公开的范围之内。

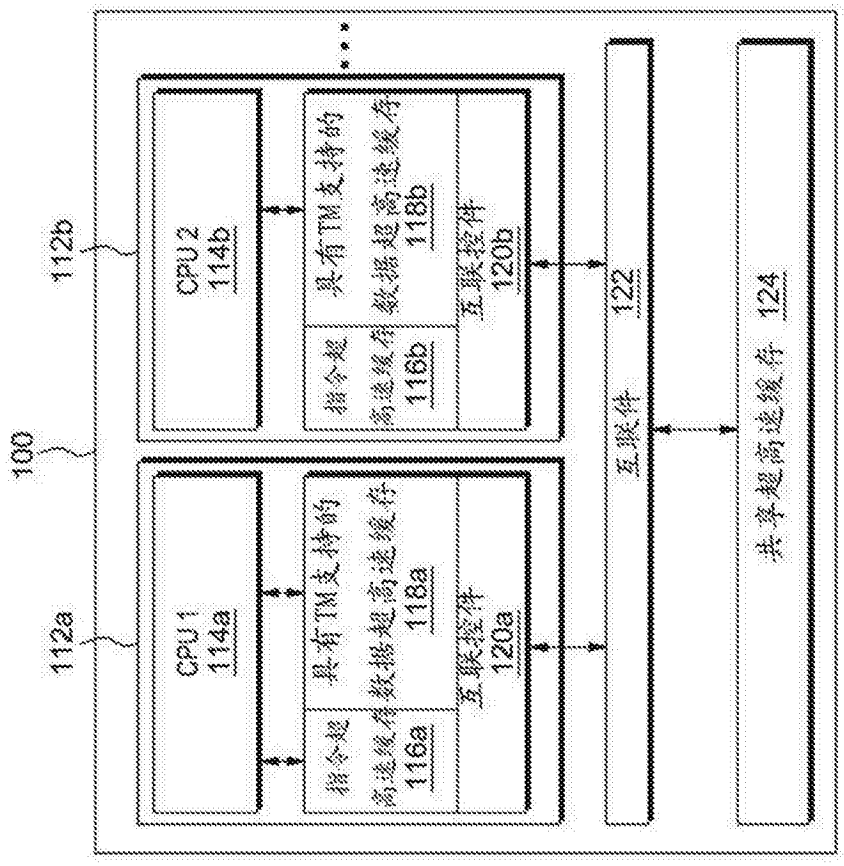

100
112b
112a
CPU2 114b
具有TM支持的数据缓存 118b
指令缓存 116b
互联控件 120b
CPU1 114a
具有TM支持的数据缓存 118a
指令缓存 116a
互联控件 120a
互联件 122
共享缓存 124

图1

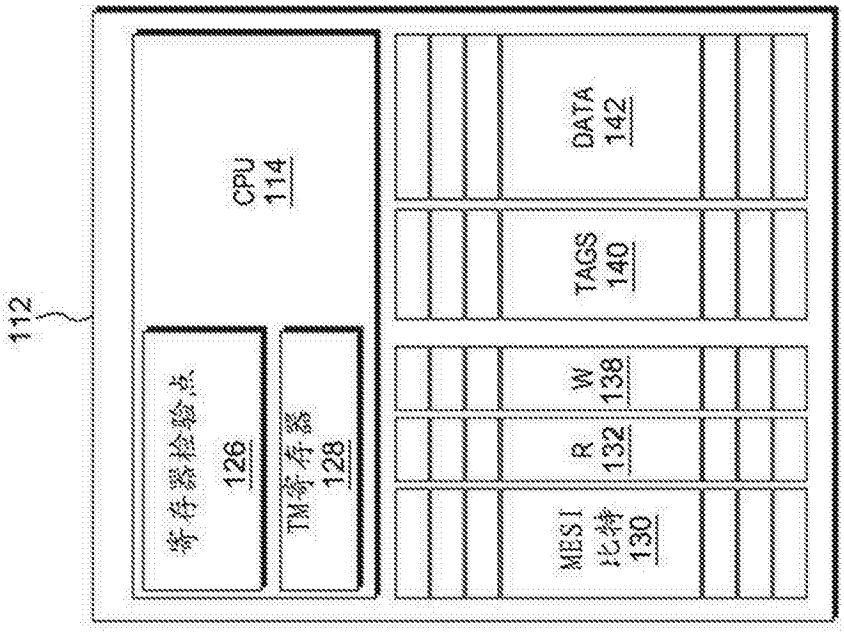

112
CPU 114
寄存器检验点 126
TM寄存器 128
DATA 142
TAGS 140
W 138
R 132
MESI 比特 130

图2

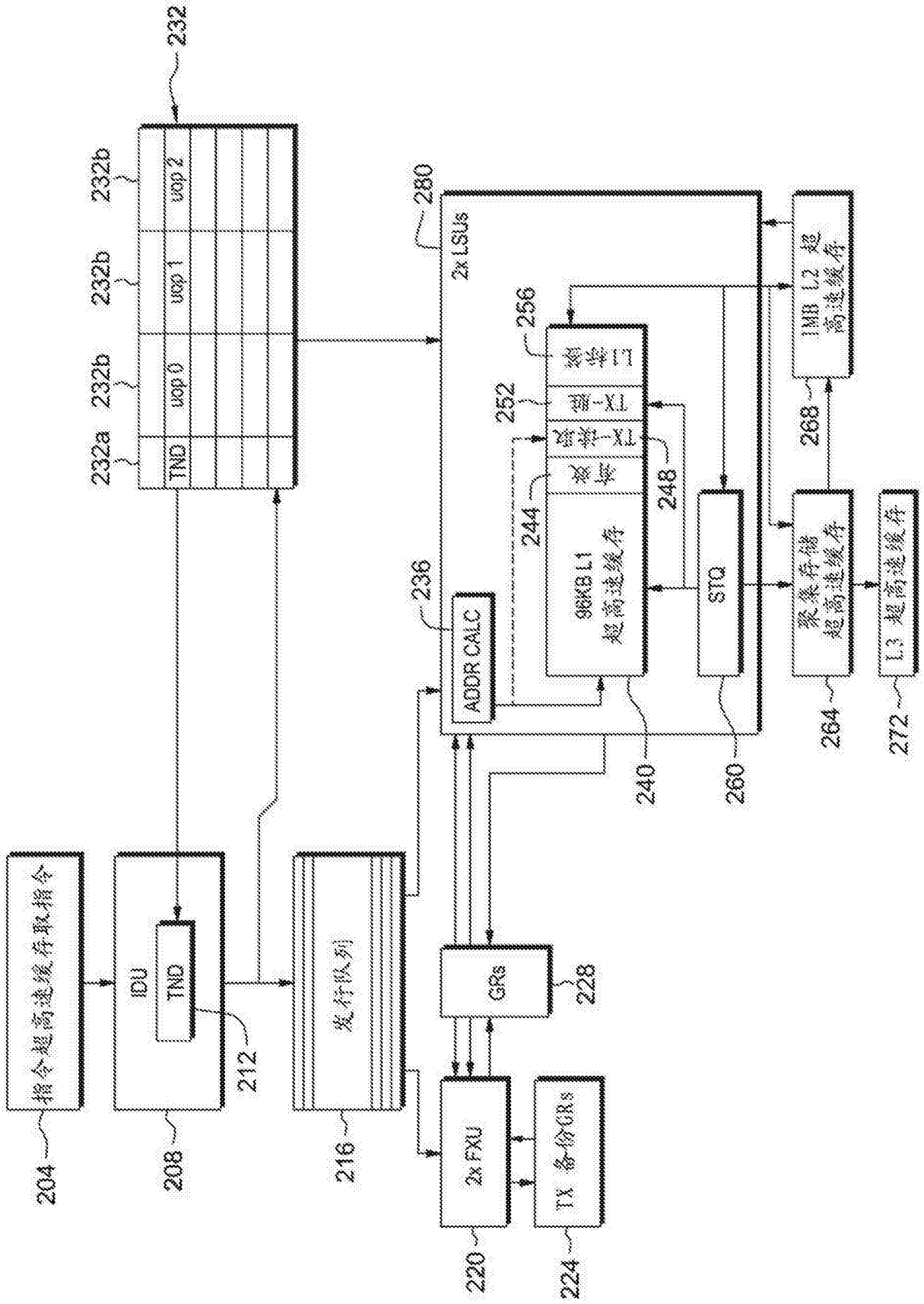

232
232b
232b
232b
232a
TND
uop 0
uop 1
uop 2
280
2x LSUs
256
L1 标签
TX-表
TX-状态
数据
252
244
248
96KB L1
超高速缓存
236
ADDR CALC
STQ
1MB L2 超
高速缓存
268
聚集存储
超高速缓存
264
L3 超高速缓存
272
240
260
指令超高速缓存取指令
204
IDU
TND
212
208
发行队列
216
GRs
228
2x FXU
220
TX 备份GRs
224

图3

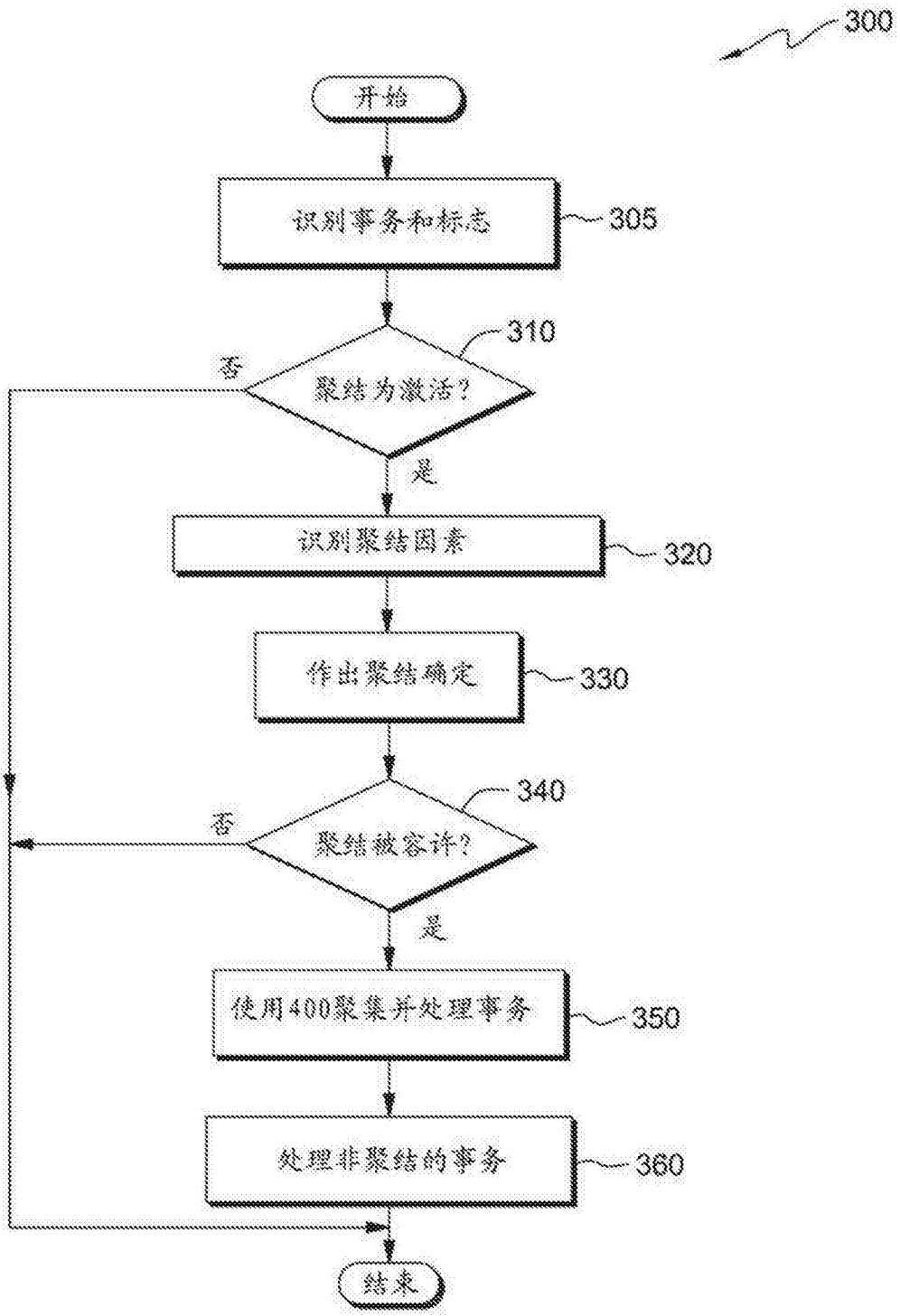

300
开始
识别事务和标志
305
310
聚结为激活?
否
是
识别聚结因素
320
作出聚结确定
330
340
聚结被容许?
否
是
使用400聚集并处理事务
350
处理非聚结的事务
360
结束

图4

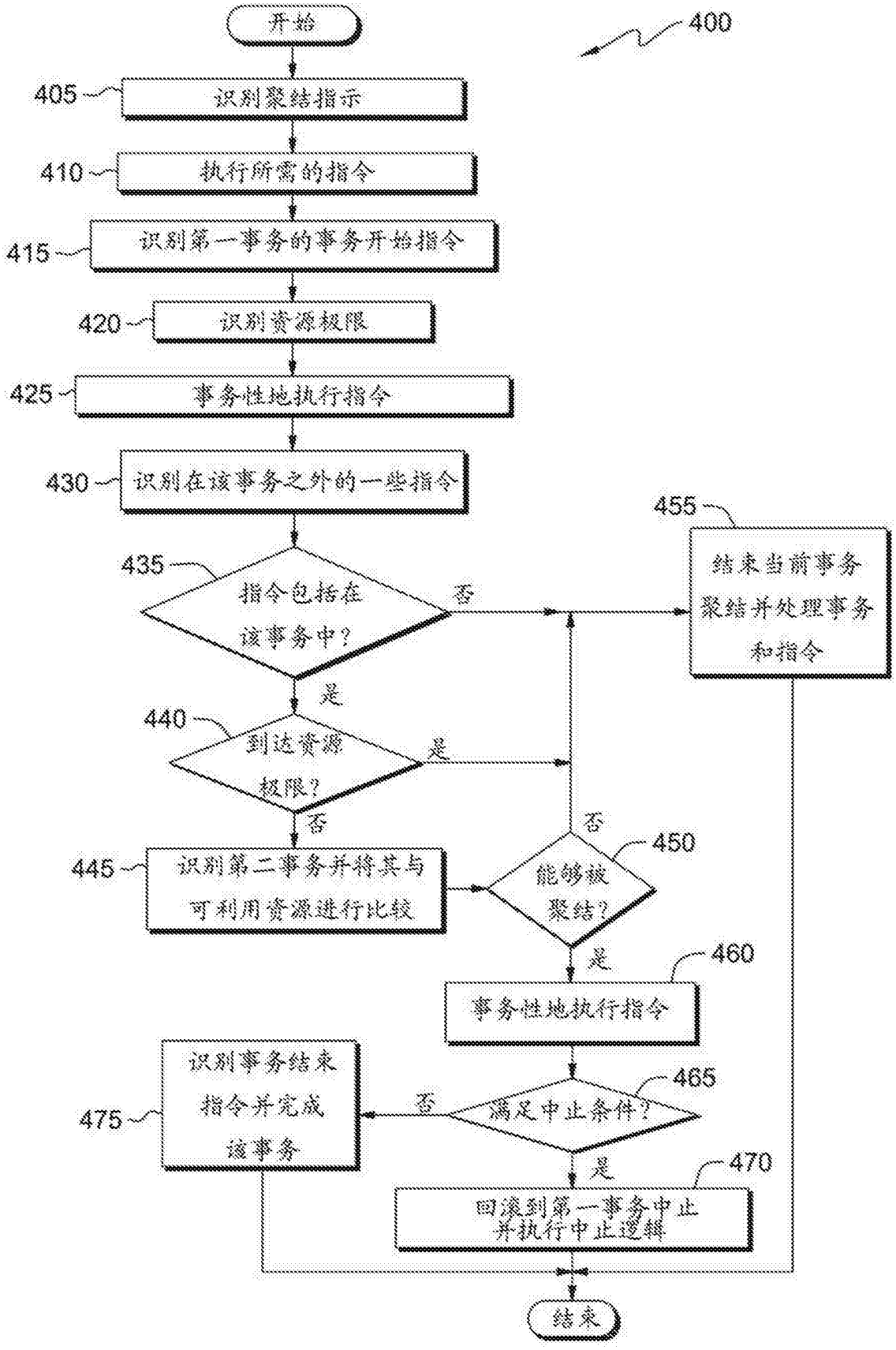

开始
400
405
识别聚结指示
410
执行所需的指令
415
识别第一事务的事务开始指令
420
识别资源权限
425
事务性地执行指令
430
识别在该事务之外的一些指令
435
指令包括在该事务中?
否
455
结束当前事务聚结并处理事务和指令
是
440
到达资源权限?
是
否
否
450
能够被聚结?
445
识别第二事务并将其与可利用资源进行比较
是
460
事务性地执行指令
465
满足中止条件?
否
475
识别事务结束指令并完成该事务
是
470
回滚到第一事务中止并执行中止逻辑
结束

图5

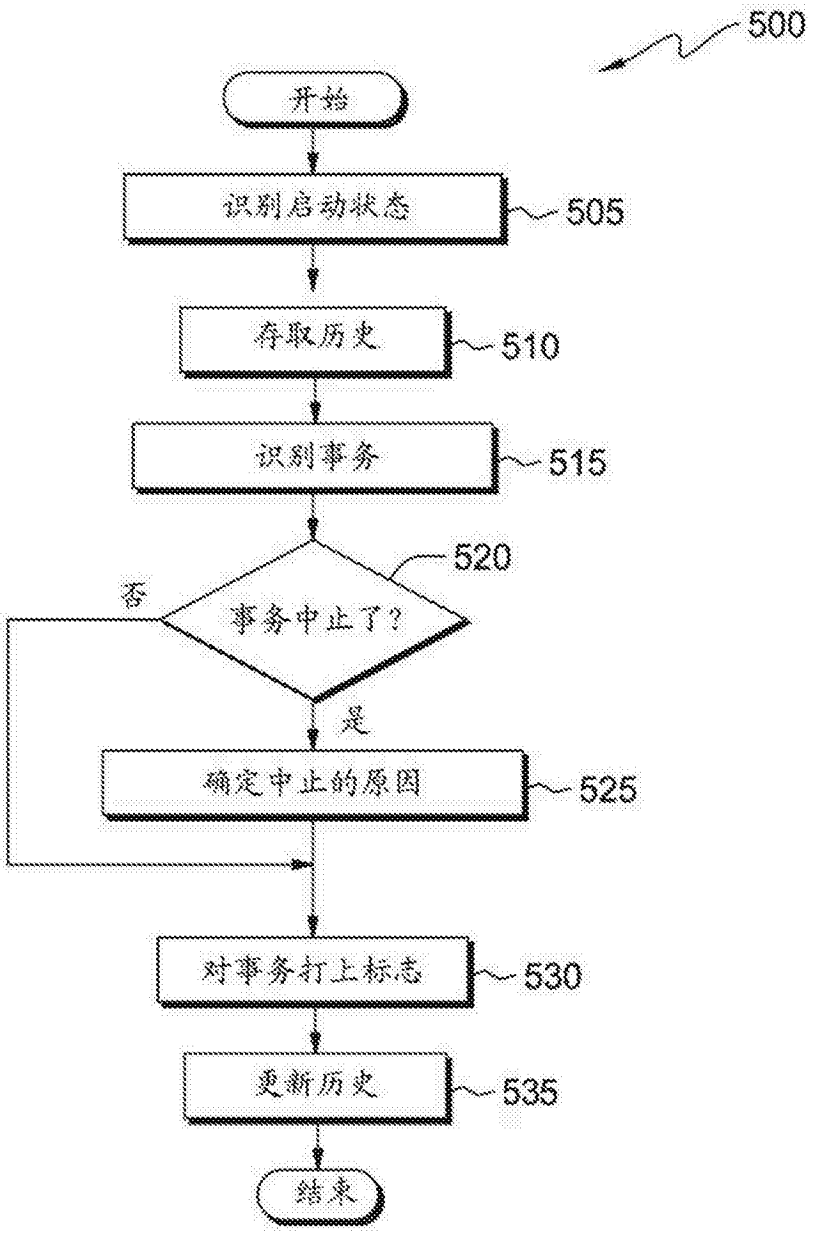

500
开始
识别启动状态
505
存取历史
510
识别事务
515
520
事务中止了?
否
是
确定中止的原因
525
对事务打上标志
530
更新历史
535
结束

图6

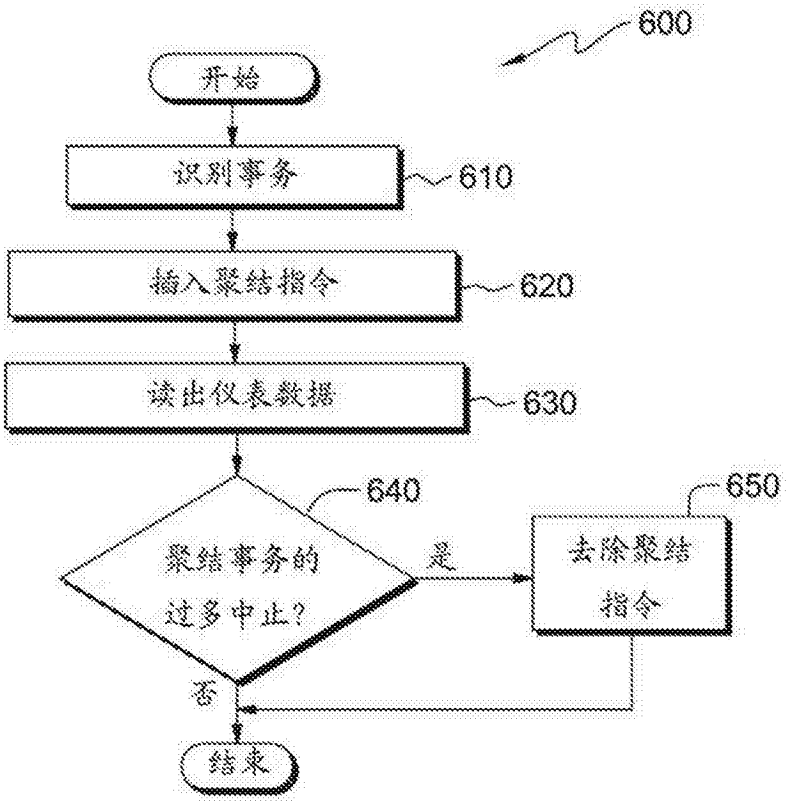

600
开始
识别事务
610
插入聚结指令
620
读出仪表数据
630
640
聚结事务的
过多中止?
是
650
去除聚结
指令
否
结束

图7

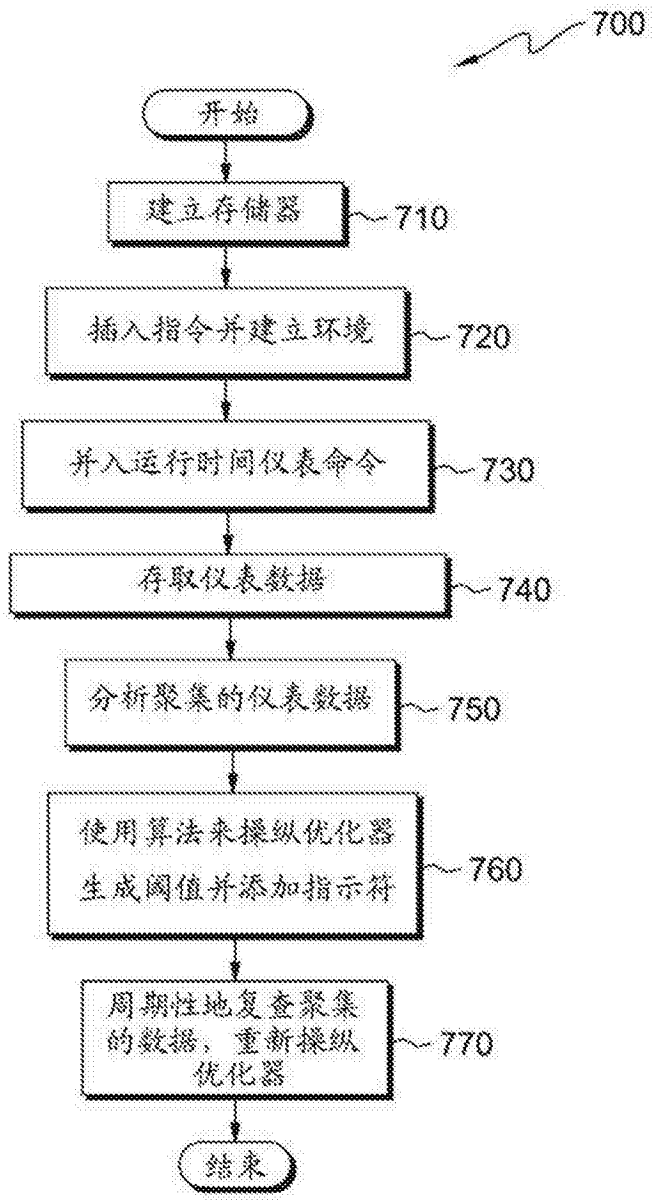

700
开始
建立存储器
710
插入指令并建立环境
720
并入运行时间仪表命令
730
存取仪表数据
740
分析聚集的仪表数据
750
使用算法来操纵优化器
生成阈值并添加指示符
760
周期性地复查聚集
的数据，重新操纵
优化器
770
结束

图8

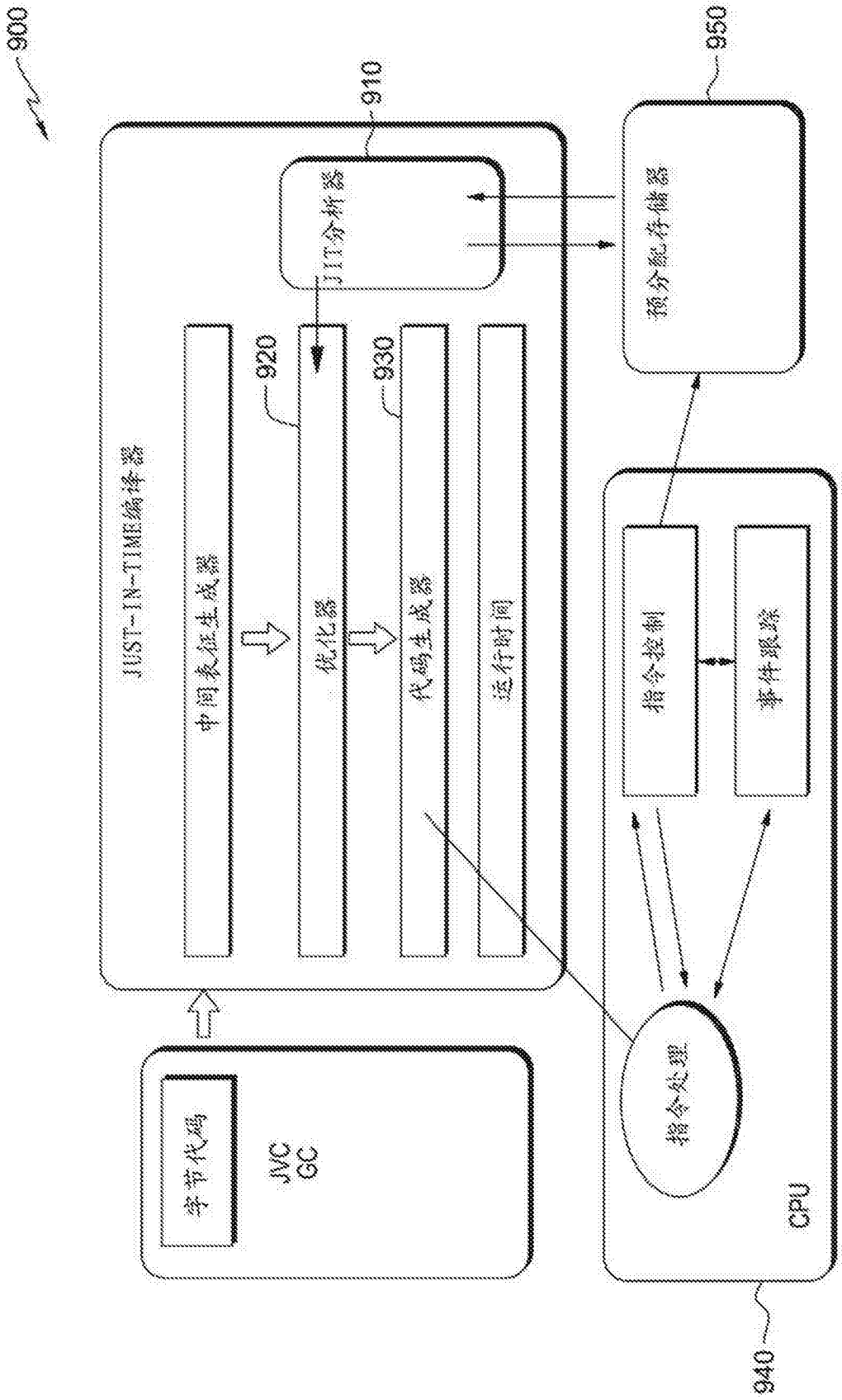

900
910
950
JIT分析器
预分配存储器
JUST-IN-TIME编译器
920
930
中间表征生成器
优化器
代码生成器
运行时间
指令控制
事件跟踪
字节代码
JVC
GC
指令处理
CPU
940

图9

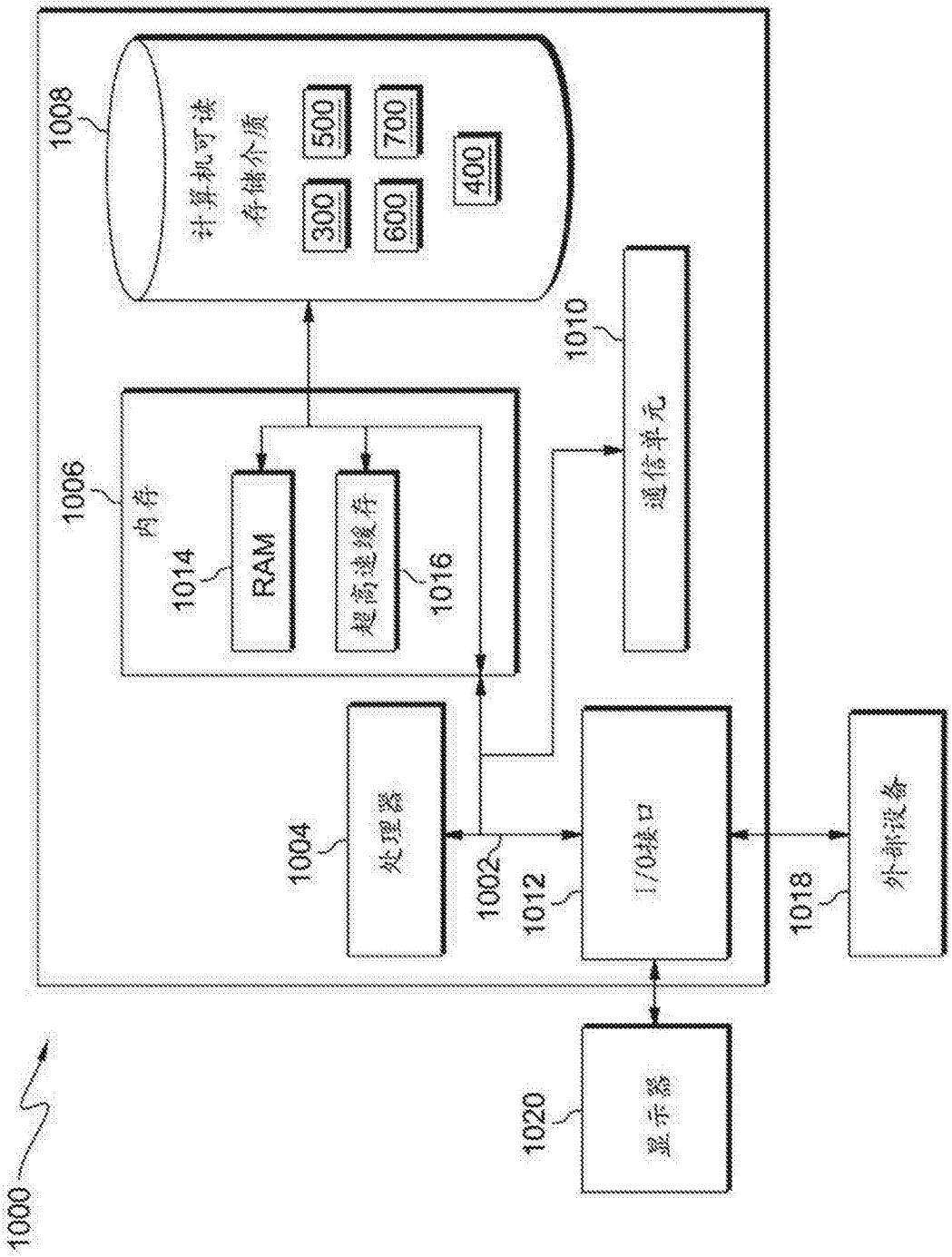

1000
1008
计算机可读存储介质
300
400
500
600
700
1004
处理器
1006
内存
1014
RAM
1016
超高速缓存
1010
通信单元
1002
1012
I/O接口
1018
外部设备
1020
显示器

图10


1004
事务处理器
1105
聚结确定
1110
阈值设置器
1111
事务聚结器
1115
指令识别器
1120
足迹处理器
1125
聚结指令
处理器
1130
模式设置器
1135
聚结控制器
1140
运行时间
监测器
1145
事务执行
优化器
1150
事务中止
确定器
1155
历史更新器
1160
事务标志器
1165
指令添加器
1170
聚结预测器
1175
历史
1180

图11

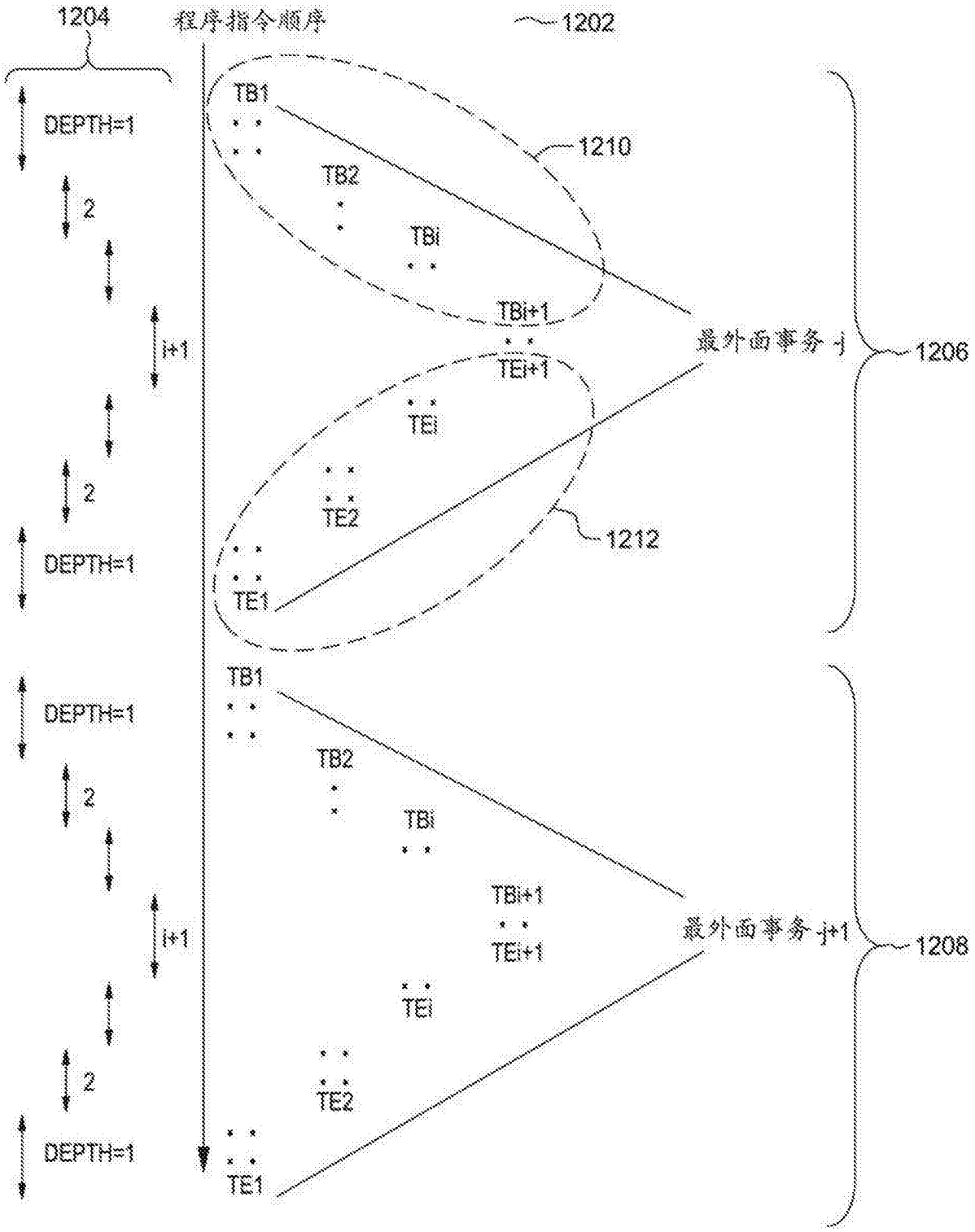

1204
程序指令顺序
1202
DEPTH=1
2
i+1
2
DEPTH=1
TB1
TB2
TBi
TBi+1
TEi+1
TEi
TE2
TE1
1210
1212
最外面事务 j
1206
DEPTH=1
2
i+1
2
DEPTH=1
TB1
TB2
TBi
TBi+1
TEi+1
TEi
TE2
TE1
最外面事务 j+1
1208

图12

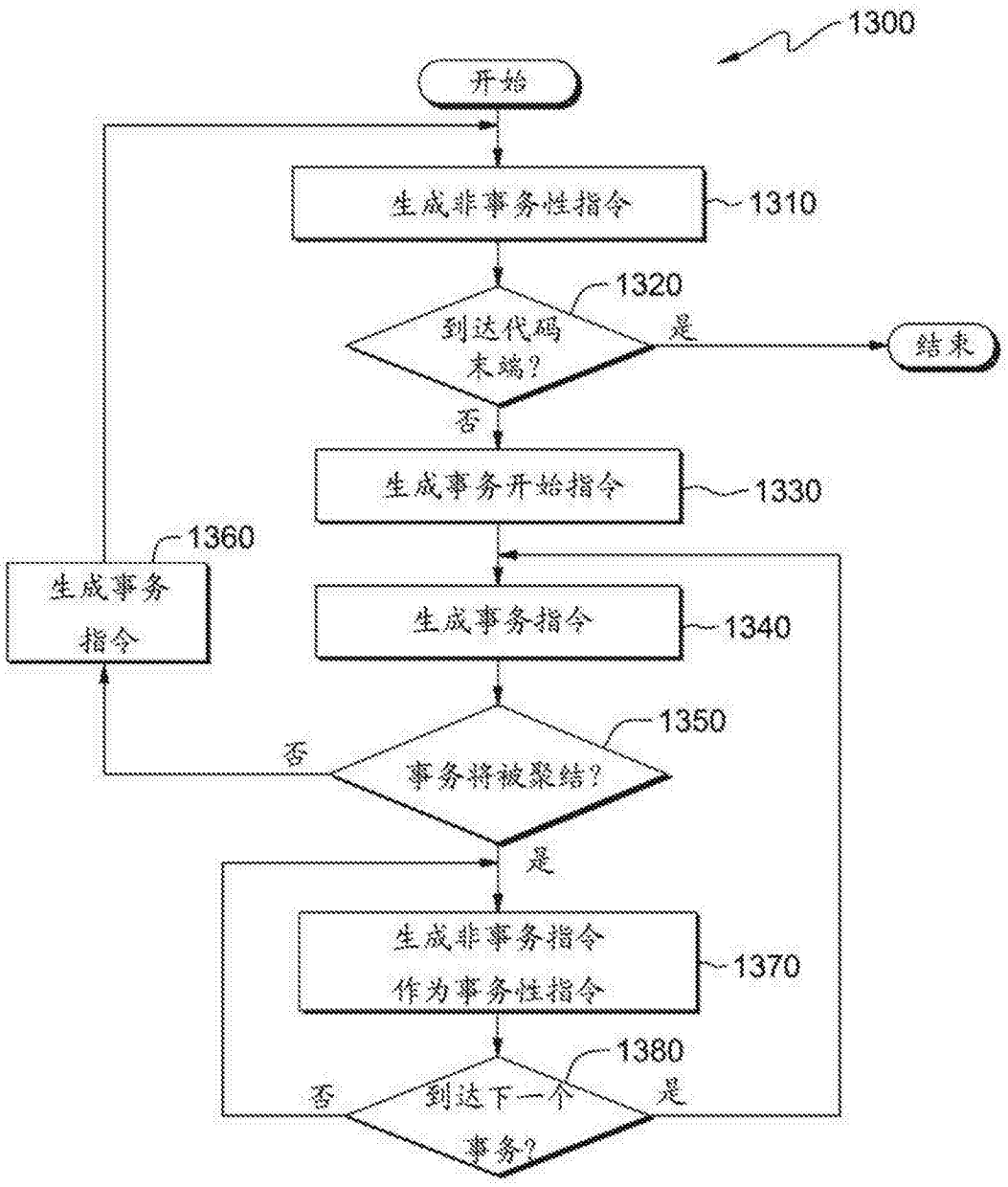

1300
开始
生成非事务性指令
1310
1320
到达代码末端?
是
结束
否
生成事务开始指令
1330
1360
生成事务指令
生成事务指令
1340
1350
事务将被聚结?
否
是
生成非事务指令作为事务性指令
1370
1380
到达下一个事务?
否
是

图13

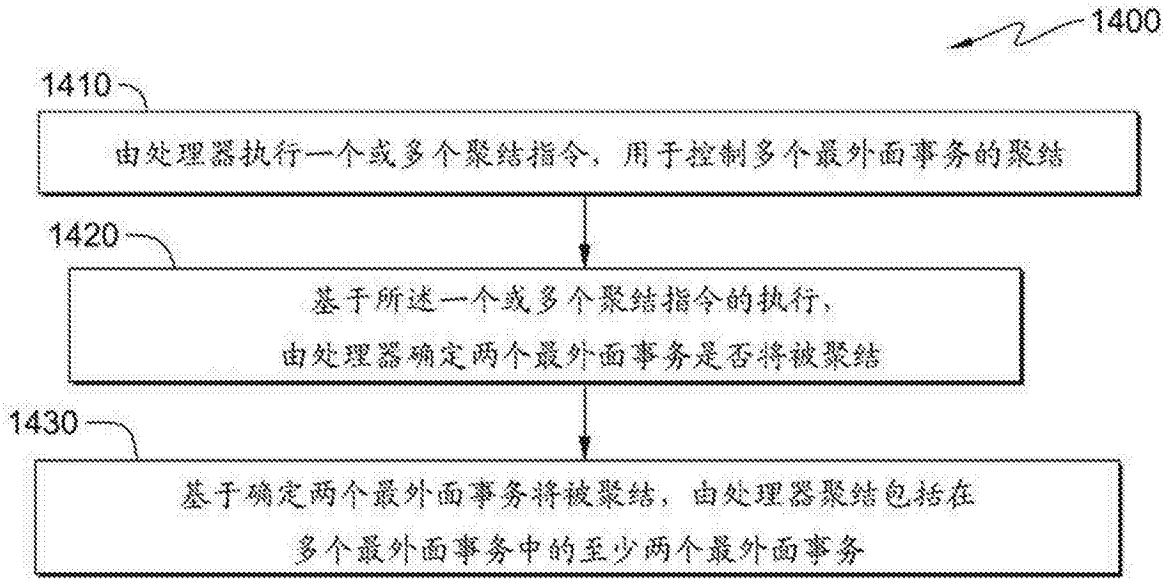

1400
1410
由处理器执行一个或多个聚结指令，用于控制多个最外面事务的聚结
1420
基于所述一个或多个聚结指令的执行，
由处理器确定两个最外面事务是否将被聚结
1430
基于确定两个最外面事务将被聚结，由处理器聚结包括在
多个最外面事务中的至少两个最外面事务

图14

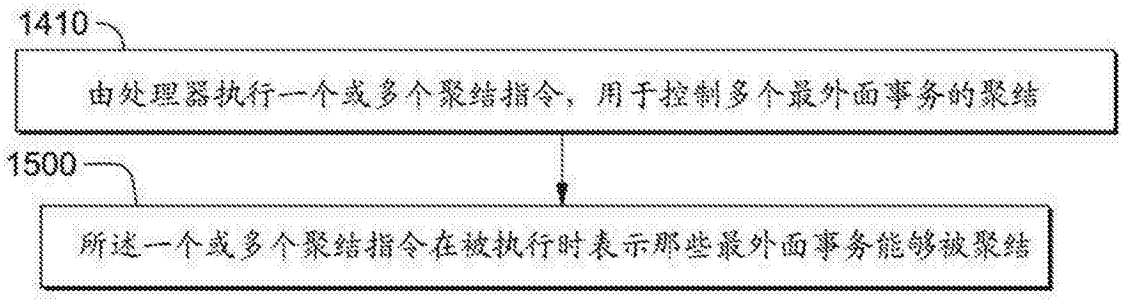

1410
由处理器执行一个或多个聚结指令，用于控制多个最外面事务的聚结
1500
所述一个或多个聚结指令在被执行时表示那些最外面事务能够被聚结

图15

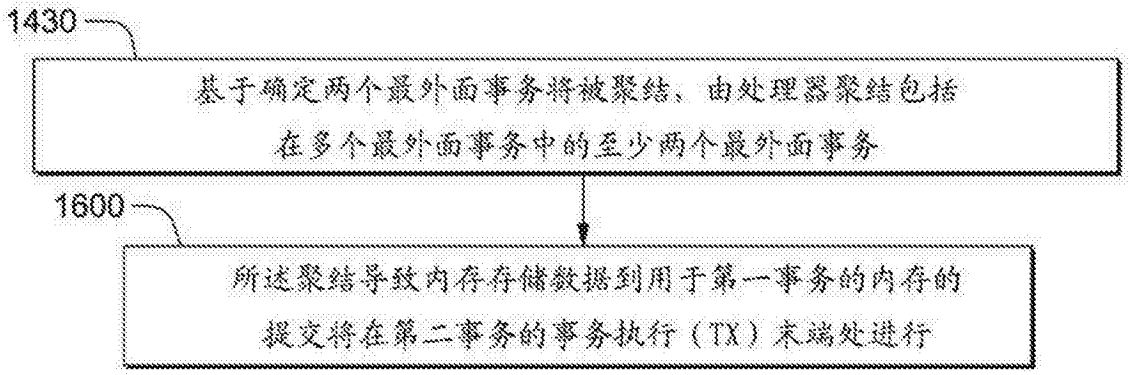

1430
基于确定两个最外面事务将被聚结，由处理器聚结包括
在多个最外面事务中的至少两个最外面事务
1600
所述聚结导致内存存储数据到用于第一事务的内存的
提交将在第二事务的事务执行（TX）末端处进行

图16

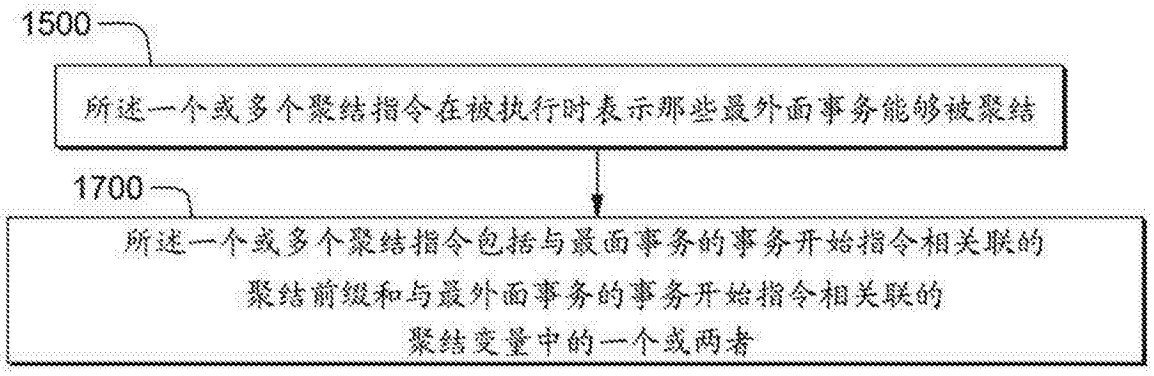

1500
所述一个或多个聚结指令在被执行时表示那些最外面事务能够被聚结
1700
所述一个或多个聚结指令包括与最面事务的事务开始指令相关联的
聚结前缀和与最外面事务的事务开始指令相关联的
聚结变量中的一个或两者

图17

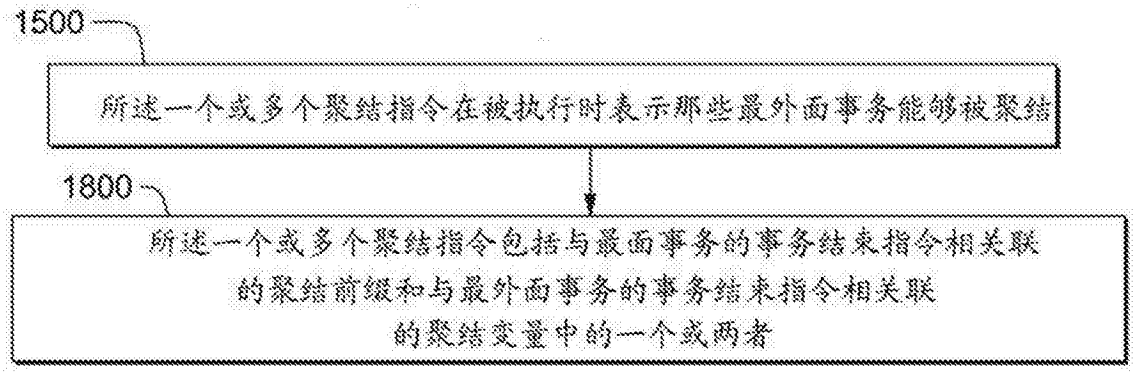

1500
所述一个或多个聚结指令在被执行时表示那些最外面事务能够被聚结
1800
所述一个或多个聚结指令包括与最面事务的事务结束指令相关联
的聚结前缀和与最外面事务的事务结束指令相关联
的聚结变量中的一个或两者

图18


1900
由处理器执行设置-事务-聚结-模式（STCM）指令，该STCM指令使得处理器进入STCM模式，其中进入STCM模式表示以下至少之一：（I）随后的聚结指令被启用以便使得事务将被聚结，（II）能够存在于两个将被聚结的最外面事务之间的指令的最大数量，以及（III）能够存在于两个将被聚结的最外面事务之间的指令的类型
1400

图19

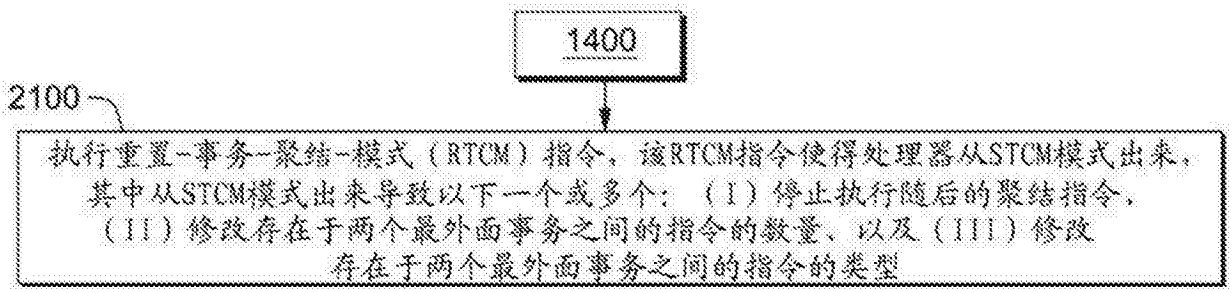

1400
2100
执行重置-事务-聚结-模式（RTCM）指令，该RTCM指令使得处理器从STCM模式出来，其中从STCM模式出来导致以下一个或多个：（I）停止执行随后的聚结指令，（II）修改存在于两个最外面事务之间的指令的数量，以及（III）修改存在于两个最外面事务之间的指令的类型

图20

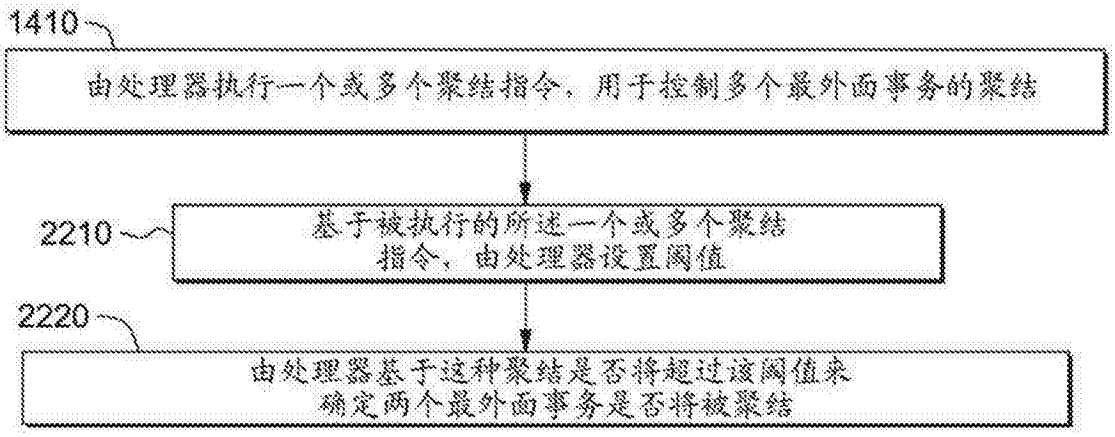

1410
由处理器执行一个或多个聚结指令，用于控制多个最外面事务的聚结
2210
基于被执行的所述一个或多个聚结指令，由处理器设置阈值
2220
由处理器基于这种聚结是否将超过该阈值来确定两个最外面事务是否将被聚结

图21

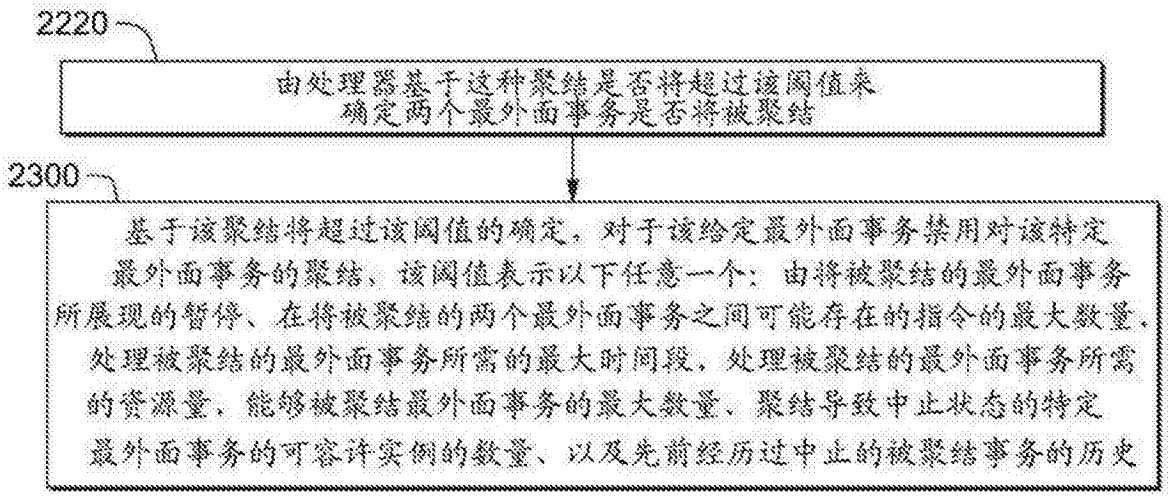

2220
由处理器基于这种聚结是否将超过该阈值来确定两个最外面事务是否将被聚结
2300
基于该聚结将超过该阈值的确定，对于该给定最外面事务禁用对该特定最外面事务的聚结，该阈值表示以下任意一个：由将被聚结的最外面事务所展现的暂停、在将被聚结的两个最外面事务之间可能存在的指令的最大数量，处理被聚结的最外面事务所需的最大时间段，处理被聚结的最外面事务所需的资源量，能够被聚结最外面事务的最大数量，聚结导致中止状态的特定最外面事务的可容许实例的数量、以及先前经历过中止的被聚结事务的历史

图22