(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 105190559 B
(45)授权公告日 2018.11.09

(21)申请号 201380072249.7
(22)申请日 2013.12.19
(65)同一申请的已公布的文献号
申请公布号 CN 105190559 A
(43)申请公布日 2015.12.23
(30)优先权数据
13/722,839 2012.12.20 US
(85)PCT国际申请进入国家阶段日
2015.08.04
(86)PCT国际申请的申请数据
PCT/US2013/076317 2013.12.19
(87)PCT国际申请的公布数据
WO2014/100296 EN 2014.06.26
(73)专利权人 甲骨文国际公司
地址 美国加利福尼亚
(72)发明人 D·戴斯 Y·列夫 M·S·莫尔
(74)专利代理机构 中国国际贸易促进委员会专利商标事务所 11038
代理人 罗亚男
(51)Int.Cl.
G06F 9/52(2006.01)
G06F 9/50(2006.01)
(56)对比文件
US 2011239299 A1,2011.09.29,
US 6940852 B1,2005.09.06,
US 7743003 B1,2010.06.22,
CN 101023417 A,2007.08.22,
Maurice Herlihy等.Scalable Concurrent Counting.《ACM Transactions on Computer Systems》.1995,第13卷(第4期),346-347.
Scott A. Mitchell等.Flexible Approximate Counting.《International Database Engineering & Applications Symposium Ideas》.2011,233-234.
审查员 彭莉

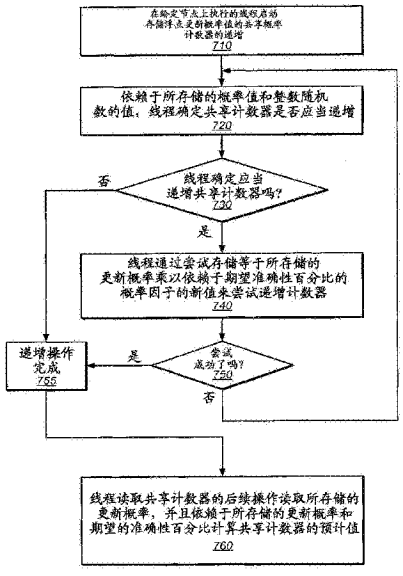
权利要求书5页 说明书31页 附图15页

(54)发明名称

用于实现存储更新概率值的共享概率计数器的系统和方法

(57)摘要

本文所述的系统和方法可以实现概率计数器和/或用于那些计数器的更新机制，使得它们依赖于可配置准确性参数的值。准确性参数值可以被调节，以便对计数器的准确性与访问它们的应用的性能之间的折中提供精细粒度的控制。计数器可以实现为包括共同表示更新概率值的尾数部分和指数部分的数据结构。当更新计数器时，可配置准确性参数的值会影响尾数部分和/或指数部分是否、何时、多经常或者按多少量被更新。更新概率计数器可以包括用依赖于可配置准确性参数的值的常量乘以其值。计数器可在事务内访问。计数器可以具有确定性更新策略。



CN 105190559 B

1.一种用于实现存储更新概率值的共享计数器的方法,包括:

由一个或多个计算节点执行以下操作,其中每个节点包括至少一个处理器核心以及存储器:

开始包括一个或多个递增共享计数器的操作的多线程应用的执行,其中共享计数器实现为能够由在一个或多个计算节点上执行的多线程应用的多个线程访问的数据结构,其中数据结构存储更新概率值的表示,其中更新概率值指示更新概率值的表示将响应于以共享计数器为目标的递增操作的启动而更新的概率,并且其中共享计数器的预计值能够至少部分地基于更新概率值的表示来计算;

由应用的给定线程启动以该共享计数器为目标的递增操作;

响应于所述启动,确定是否执行递增操作,其中所述确定至少部分地依赖于存储在数据结构中的更新概率的表示;及

响应于确定递增操作要被执行,更新存储在数据结构中的更新概率值的表示,其中更新的表示代表存储在数据结构中的更新概率值的更新表示将响应于后续递增操作的启动而进一步被更新的概率;

其中所述确定是否执行递增操作的步骤或者所述更新更新概率值的表示的步骤中的一个或多个依赖于可配置准确性参数的预定值。

2.如权利要求1所述的方法,还包括:

由所述多个线程之一执行以共享计数器为目标的读操作,其中执行读操作包括:

读取存储在数据结构中的更新概率值的表示;

从存储在数据结构中的更新概率值的表示计算共享计数器的预计值;及

返回共享计数器的预计值。

3.如权利要求1所述的方法,其中数据结构存储更新概率值的浮点表示。

4.如权利要求3所述的方法,其中所述更新包括用依赖于可配置准确性参数的预定值的常量乘以更新概率值的浮点表示。

5.如权利要求1所述的方法,其中数据结构存储概率计数器值的二进制浮点表示,所述二进制浮点表示包括尾数部分和指数部分。

6.如权利要求5所述的方法,其中所述更新包括依赖于可配置准确性参数的预定值更新尾数部分或指数部分中的一个或多个。

7.如权利要求1所述的方法,其中所述更新导致减小存储在数据结构中的更新概率值的更新表示将响应于后续递增操作的启动而进一步被更新的概率。

8.如权利要求1所述的方法,还包括:

响应于所述启动,生成随机数;

其中所述确定还依赖于所生成的随机数。

9.如权利要求1所述的方法,其中所述多线程应用的所述多个线程代表并发执行的事务,并且其中所述启动包括在并发执行的事务之一内启动递增操作。

10.如权利要求1所述的方法,

其中数据结构还存储精确的计数值;及

其中所述更新存储在数据结构中的更新概率值的表示是响应于递增所存储的精确计数值的一次或多次失败的尝试或者响应于争用管理策略的条件被满足而执行的。

11.如权利要求10所述的方法，还包括：

由所述多个线程之一启动以共享计数器为目标的读操作；及

响应于所述多个线程之一启动读操作：

读取存储在数据结构中的更新概率值的表示；

从存储在数据结构中的更新概率值的表示计算共享计数器的预计值；

把该预计值与所存储的精确计数值相加；及

返回所述相加的结果。

12.一种计算机系统，包括：

支持多线程的一个或多个处理器；

存储程序指令的存储器，当所述程序指令在一个或多个处理器上执行时，使这一个或多个处理器执行：

开始包括一个或多个递增共享计数器的操作的多线程应用的执行，其中共享计数器实现为能够由所述多线程应用的多个线程访问的数据结构，其中数据结构存储更新概率值的表示，其中更新概率值指示更新概率值的表示将响应于以共享计数器为目标的递增操作的启动而更新的概率，并且其中共享计数器的预计值能够至少部分地基于更新概率值的表示来计算；

由所述应用的给定线程启动以该共享计数器为目标的递增操作；

响应于所述启动，确定是否执行递增操作，其中所述确定至少部分地依赖于存储在数据结构中的更新概率的表示；及

响应于确定递增操作要被执行，更新存储在数据结构中的更新概率值的表示，其中更新的表示代表存储在数据结构中的更新概率值的更新表示将响应于后续递增操作的启动而进一步被更新的概率；

其中所述确定是否执行递增操作的步骤或者所述更新更新概率值的表示的步骤中的一个或多个依赖于可配置准确性参数的预定值。

13.如权利要求12所述的计算机系统，

其中数据结构存储更新概率值的浮点表示；及

其中所述更新包括用依赖于可配置准确性参数的预定值的常量乘以更新概率值的浮点表示。

14.如权利要求12所述的计算机系统，

其中数据结构存储概率计数器值的二进制浮点表示，所述二进制浮点表示包括尾数部分和指数部分；及

其中所述更新包括依赖于可配置准确性参数的预定值更新尾数部分或指数部分中的一个或多个。

15.如权利要求12所述的计算机系统，其中，当在一个或多个处理器上执行时，程序指令还使这一个或多个处理器执行：

响应于所述启动，生成随机数；

其中所述确定还依赖于所生成的随机数。

16.如权利要求12所述的计算机系统，

其中数据结构还存储精确的计数值；

其中精确计数值和预计值共同表示共享计数器的值;及

其中所述更新存储在数据结构中的更新概率值的表示是响应于递增所存储的精确计数值的一次或多次失败的尝试或者响应于争用管理策略的条件被满足而执行的。

17.一种存储程序指令的非暂态计算机可读存储介质,当程序指令在一个或多个计算机上执行时,使这一个或多个计算机执行:

开始包括一个或多个递增共享计数器的操作的多线程应用的执行,其中共享计数器实现为能够由多线程应用的多个线程访问的数据结构,其中数据结构存储更新概率值的表示,其中更新概率值指示更新概率值的表示将响应于以共享计数器为目标的递增操作的启动而更新的概率,并且其中共享计数器的预计值能够至少部分地基于更新概率值的表示来计算;

由所述应用的给定线程启动以该共享计数器为目标的递增操作;

响应于所述启动,确定是否执行递增操作,其中所述确定至少部分地依赖于存储在数据结构中的更新概率的表示;及

响应于确定递增操作要被执行,更新存储在数据结构中的更新概率值的表示,其中更新的表示代表存储在数据结构中的更新概率值的更新表示将响应于后续递增操作的启动而进一步被更新的概率;

其中所述确定是否执行递增操作的步骤或者所述更新更新概率值的表示的步骤中的一个或多个依赖于可配置准确性参数的预定值。

18.如权利要求17所述的非暂态计算机可读存储介质,

其中数据结构存储更新概率值的浮点表示;及

其中所述更新包括用依赖于可配置准确性参数的预定值的常量乘以更新概率值的浮点表示。

19.如权利要求17所述的非暂态计算机可读存储介质,

其中数据结构存储概率计数器值的二进制浮点表示,所述二进制浮点表示包括尾数部分和指数部分;及

其中所述更新包括依赖于可配置准确性参数的预定值更新尾数部分或指数部分中的一个或多个。

20.如权利要求17所述的非暂态计算机可读存储介质,其中,当在一个或多个计算机上执行时,程序指令还使这一个或多个计算机执行:

响应于所述启动,生成随机数;

其中所述确定还依赖于所生成的随机数。

21.一种计算机系统,包括:

一个或多个处理器核心(220a-220h),以及

互连结构(240),其被配置为将所述一个或多个处理器核心(220a-220h)耦合到系统存储器(260),

其中,由包括所述计算机系统在内的多个计算机系统开始包括一个或多个递增共享计数器的操作的多线程应用的执行,其中共享计数器实现为系统存储器(260)中的能够由所述多线程应用的多个线程访问的数据结构,其中数据结构存储更新概率值的表示,其中更新概率值指示更新概率值的表示将响应于以共享计数器为目标的递增操作的启动而更新

的概率，并且其中共享计数器的预计值能够至少部分地基于更新概率值的表示来计算；

其中，所述一个或多个处理器核心执行所述多线程应用的所述多个线程中的一个或多个线程，所述一个或多个线程中的每个线程被配置为

启动以该共享计数器为目标的递增操作；

响应于所述启动，确定是否执行递增操作，其中所述确定至少部分地依赖于存储在数据结构中的更新概率的表示；及

响应于确定递增操作要被执行，更新存储在数据结构中的更新概率值的表示，其中更新的表示代表存储在数据结构中的更新概率值的更新表示将响应于后续递增操作的启动而进一步被更新的概率；

其中所述确定是否执行递增操作的步骤或者所述更新更新概率值的表示的步骤中的一个或多个依赖于可配置准确性参数的预定值。

22.如权利要求21所述的计算机系统，所述线程还被配置为执行以共享计数器为目标的读操作，其中执行读操作包括：

读取存储在数据结构中的更新概率值的表示；

从存储在数据结构中的更新概率值的表示计算共享计数器的预计值；及

返回共享计数器的预计值。

23.如权利要求21所述的计算机系统，其中数据结构存储更新概率值的浮点表示。

24.如权利要求23所述的计算机系统，其中所述更新包括用依赖于可配置准确性参数的预定值的常量乘以更新概率值的浮点表示。

25.如权利要求21所述的计算机系统，其中数据结构存储概率计数器值的二进制浮点表示，所述二进制浮点表示包括尾数部分和指数部分。

26.如权利要求25所述的计算机系统，其中所述更新包括依赖于可配置准确性参数的预定值更新尾数部分或指数部分中的一个或多个。

27.如权利要求21所述的计算机系统，其中所述更新导致减小存储在数据结构中的更新概率值的更新表示将响应于后续递增操作的启动而进一步被更新的概率。

28.如权利要求21所述的计算机系统，所述线程还被配置为：

响应于所述启动，生成随机数；

其中所述确定还依赖于所生成的随机数。

29.如权利要求21所述的计算机系统，其中所述多线程应用的所述多个线程代表并发执行的事务，并且其中所述启动包括在并发执行的事务之一内启动递增操作。

30.如权利要求21所述的计算机系统，

其中数据结构还存储精确的计数值；及

其中所述更新存储在数据结构中的更新概率值的表示是响应于递增所存储的精确计数值的一次或多次失败的尝试或者响应于争用管理策略的条件被满足而执行的。

31.如权利要求30所述的计算机系统，所述线程还被配置为：

启动以共享计数器为目标的读操作；及

响应于所述线程启动读操作：

读取存储在数据结构中的更新概率值的表示；

从存储在数据结构中的更新概率值的表示计算共享计数器的预计值；

把该预计值与所存储的精确计数值相加;及

返回所述相加的结果。

用于实现存储更新概率值的共享概率计数器的系统和方法

技术领域

[0001] 本公开内容涉及共享的统计计数器，并且更具体而言涉及用于提高包括对共享统计计数器进行访问的应用的性能的技术。

背景技术

[0002] 多核体系架构设计中当前的趋势暗示，在未来几年内，将有从简单的基于总线的设计朝分布式非一致存储器访问(NUMA)和高速缓存固有NUMA(CC-NUMA)体系架构的加速偏移。根据NUMA，对任何给定访问的存储器访问时间依赖于被访问的存储器相对于处理器的位置。这种体系架构通常包括具有快速本地存储器(例如，紧密耦合到处理器和/或位于相同的单个多核芯片上的存储器)的计算核心的集合，其中这些存储器经由较慢的(芯片间)通信介质彼此通信。在这种系统中，处理器通常可以访问其自己的比非本地存储器快的本地存储器，诸如其自己的高速缓存存储器。在一些系统中，非本地存储器可以包括一个或多个在处理器之间共享的存储器组和/或在另一处理器本地的存储器组。一些系统，包括许多NUMA系统，提供非一致的通信体系架构(NUCA)属性，其中访问其它处理器核心的高速缓存的时间随其离发出请求的核心的物理距离而变。在这些系统中，核心对其本地存储器的访问，并且尤其是对共享的本地高速缓存的访问，可以比对远程存储器(例如，位于另一芯片上的高速缓存)的访问快若干(或者许多)倍。

[0003] 更大型的软件系统使用统计计数器来进行性能监视和诊断。例如，统计计数器对诸如检测各种系统事件的过高速率或者对基于事件频率而变的机制的目的具有实际重要性。虽然单线程的统计计数器微不足道，但是通常使用的单纯(naive)并发实现很快变得有问题，尤其是当线程计数增长时。例如，当系统增长并且当统计计数器在越来越多的非一致存储器访问(NUMA)系统中被使用时，常用的单纯计数器强加扩展性瓶颈和/或它们无用的此类不准确性。特别地，这些计数器(当在线程之间共享时)会对计数器的每次修改招致无效流量，这在NUMA机器上尤其昂贵。

[0004] 并行执行事务的能力是可扩展性能的关键。但是，当对计数器的访问发生在事务内部时，共享计数器用于收集统计数据(例如，关于一块代码执行得多频繁、多少元素在散列表中等等的统计数据)会不利地影响事务成功率(因为由不同事务或线程对共享计数器的任何两次更新都将潜在地彼此冲突)。解决这个问题的一些之前的方法涉及把更新计数器的操作移动到事务之外，由此改变程序的语义，或者实现对“事务提升”的复杂并且昂贵的支持，这不是在所有语境下都适用。

[0005] 出于这些和其它原因，应用设计者面对困难的折中，涉及强加于轻争用计数器之上的等待时间、可扩展性和(在一些情况下)重争用计数器的准确性，以及各种探针效应。

发明内容

[0006] 本文所述的系统和方法在各种实施例中可以用来实现可扩展的统计计数器。在一些实施例中，这些计数器，而不是标准计数器，的使用可以提高在采用NUMA风格存储器体系

架构和/或呈现NUCA属性的系统中执行的应用的性能。如在本文所使用的，术语“NUMA”和“NUMA风格存储器体系架构”可以参考呈现NUMA和/或NUCA属性的任何系统来使用。在一些实施例中，计数器可以实现为包括计数值部分和节点标识符部分的数据结构。节点标识符部分可以识别最近递增计数器的线程在其上执行的节点或者已经请求递增共享计数器的优先权的线程在其上执行的节点。在由计数器数据结构的节点标识符部分识别的节点上执行的线程可以比其它线程具有递增计数器的更高优先权。在一些实施例中，在除由计数器数据结构的节点标识符部分识别的节点之外的节点上执行的线程可以在重新尝试它们之前延迟其递增计数器的尝试。这可以鼓励从单个节点上线程的连续更新，从而减少高速缓存失中并提高整体性能。在一些实施例中，急不可耐的线程会尝试更新数据结构的节点标识符部分，或者可以更新单独的抗饥饿变量(例如，通过写它们在其上执行的节点的标识符)，以指示对递增共享计数器的优先权的请求。

[0007] 在一些实施例中，本文所描述的系统和方法可以实现概率计数器。如在本文更详细描述的，在各种实施例中，这些概率计数器可以直接存储概率值或者可以存储其它类型的概率计数器值(例如，它们可以存储代表更新概率和/或预计计数器值或者从其可以计算更新概率和/或预计计数器值的一个或多个值)。在一些实施例中，计数器和/或其更新机制的实现可以依赖于可配置准确性参数的值。在此类实施例中，可配置的准确性参数可以被调整，以便对计数器的准确性与访问它们的应用的性能之间的折中提供精细粒度的控制。例如，计数器可以实现为包括共同表示更新概率值的尾数部分和指数部分的数据结构。如在本文更详细描述的，当更新计数器时，可配置的准确性参数的值可以影响尾数部分和/或指数部分是否、何时、多频繁和/或按多少量更新。在另一个例子中，更新概率计数器可以包括用依赖于可配置准确性参数的值的常量乘以其值。

[0008] 在一些实施例中，本文所描述的系统和方法可以实现适应应用的多个线程对计数器的争用量的可扩展统计计数器。例如，用于确定是否响应于递增操作的启动而递增计数器的方法和/或用于更新计数器的方法可以依赖于当前、最近或历史争用量而从多种可用方法中选择。在一些实施例中，计数器可以从原子事务中访问。在各种实施例中，不同的争用管理策略和/或重试条件可以用来在多种方法中选择。例如，在一些实施例中，用于确定是否或者如何更新共享计数器的方法可以响应于对计数器的争用的增加或减少(例如，在利用初始或缺省方法递增或更新计数器的预定最大次数失败尝试之后)而动态(即，在执行期间)改变。在一些实施例中，共享的计数器可以包括在低争用情况下递增的精确计数器部分和在高争用情况下更新的概率计数器部分。在一些实施例中，概率计数器递增的量可以依赖于争用。在其它实施例中，计数器可以包括只有当计数器处于争用情况下时才鼓励单个节点上的线程连续递增的节点标识符部分。在还有其它实施例中，相对简单的计数器数据结构可以响应于对计数器的争用而膨胀(inflated)，如以下更详细描述的。

附图说明

[0009] 图1是示出如本文所述用于实现NUMA感知共享计数器的方法的一种实施例的流程图。

[0010] 图2是示出实现NUMA风格存储器体系架构的计算机系统的一部分的框图。

[0011] 图3A-3F是示出本文所述各种计数器结构的例子的框图。

[0012] 图4是示出根据一种实施例用于递增NUMA感知共享计数器的方法的流程图。

[0013] 图5是示出根据一种实施例用于递增依赖于对共享计数器的争用的NUMA感知共享计数器的方法的流程图。

[0014] 图6是示出根据一种实施例用于响应于争用而膨胀共享计数器的方法的流程图。

[0015] 图7是示出根据一种实施例用于递增存储浮点值的概率计数器的方法的流程图。

[0016] 图8是示出根据一种实施例用于递增依赖于对计数器的争用的混合计数器的方法的流程图。

[0017] 图9是示出根据一种实施例用于递增存储二进制浮点值的概率计数器的方法的流程图。

[0018] 图10是示出根据一种实施例用于递增包括多个更新选项的概率计数器的方法的流程图。

[0019] 图11是示出根据一种实施例用于依赖于可配置准确性参数而递增概率计数器的方法的流程图。

[0020] 图12是示出根据一种实施例用于依赖于对共享计数器的争用递增存储浮点值的概率计数器的方法的流程图。

[0021] 图13是示出根据一种实施例用于依赖于对共享计数器的争用确定是否递增共享计数器并用于递增共享计数器的方法的流程图。

[0022] 图14是示出配置为实现本文所述一个或多个共享计数器的计算系统的一种实施例的框图。

[0023] 虽然本公开内容作为例子对几种实施例和说明性附图进行了描述，但是本领域技术人员将认识到，本公开内容不限于所述实施例或附图。应当理解，附图和对其的详细描述不是要把本公开内容限定到所公开的特定形式，相反，本公开内容是要覆盖属于如由所附权利要求定义的精神和范围的所有修改、等价物和备选方案。本文所使用的任何标题都仅仅是为了组织而不是要限制描述或权利要求的范围。如本文所使用的，词“可以”是在允许的意义上(即，意味着具有...的潜能)而不是在强制的意义上(即，意味着必须)使用的。类似地，词“包括”意味着包括，但不限于。

具体实施方式

[0024] 如以上所指出的。统计计数器的使用在大部分大型软件系统中是非常常见的。对共享统计计数器的访问可以是多线程应用的并发线程，包括在支持硬件和/或软件事务存储器的系统中执行的那些应用，之间争用的来源。多核机器的尺寸增加，并且相应地从简单的基于总线的设计偏移到NUMA和CC-NUMA风格存储器体系架构。由于这种偏移，存在对可扩展统计计数器的增加的需求。本文所描述的系统和方法可以在一些实施例中被用来实现可扩展统计计数器。在不同的实施例中，本文所描述的可扩展统计计数器通过把计数器分成多个部分由此减少对每个部分的争用，或者通过采用计数器更不经常更新的技术，来实现这种扩展性。这两类技术可以显著减少使用它们的并发执行的原子事务之间的冲突，因此提高它们成功的机会并帮助实现可扩展性能。

[0025] 在不同的实施例中，与常用的单纯计数器相比，本文所描述的技术可以被用来实现不阻塞并且提供显著更好扩展性和准确性属性的精确和/或概率(统计)计数器。虽然概

率计数器不能提供确切的计数，但是它们可以具有统计属性，使得它们有高概率不会从精确计数偏离“太多”，例如，根据可配置的准确性参数。在一些实施例中，甚至当争用低时，本文所描述的计数器也可以与单纯计数器竞争。一般而言，本文所述的统计计数器可以适于其中它们被用来计数有可能以高频率发生的事件的应用，同时计数器的值可以不频繁地被读取，就像对性能监视和诊断常见的。虽然本文所描述的许多计数器被假设只递增一并且从不递减，但是本文所描述的技术在其它实施例中可以一般化为削弱和/或避免这些假设。

[0026] 在一些实施例中，在计数器更新在另一个节点上发生之前，本文所描述的技术可以鼓励争用的统计计数器在NUMA系统的一个节点上的多次连续递增。避免这些连续更新之间的跨节点通信在一些实施例中可以显著减少NUMA节点之间的昂贵通信流量，由此提高吞吐量和扩展性。在一些实施例中，这些技术可以提供比常用的单纯方法扩展更好的准确统计计数器，同时增加一点或不添加空间开销。

[0027] 一般而言，无需同步地简单递增共享计数器在多线程应用中工作得不好，因为一个线程所作的更新可以被另一个线程所作的更新重写，由此丧失一个或多个递增对计数器的影响。在一些情况下，这种计数器可以通过用锁保护来使得它们线程安全。但是，在大多数现代共享存储器多处理器中，利用诸如比较并交换(CAS)类型指令的原子指令递增计数器可能更好。如果CAS类型指令被用来递增计数器，则将只有计数器保持在递增它之前递增线程期望看到的值并且操作成功更新计数器值时才指示成功。否则，递增操作可以重试，有可能在某个退避(back-off)期之后。这种解决办法简单、正确并且不阻塞，但是它不扩展到更大和越来越多的NUMA系统。利用单线程递增操作(例如，利用单独的加载和存储指令来更新计数器)消除CAS类型指令的开销并且减小等待时间(例如，当不必知道计数器的精确值时)无法避免解决有可能在变量被NUMA系统中的线程修改时发生的远程高速缓存失中的主要成本。此外，这种方法不仅仅导致在争用情况下更新的偶然丢失，但是已经显示当被大量(例如，32个或更多)线程共享时导致绝大多数更新的丢失。具有讽刺意味的是，这个问题随着争用增加而变得更糟，这常常是计数器要被检测的场景。

[0028] 使计数器可扩展的一种方法是把它们分成每个线程的部分，每个线程递增其自己的部分，无需同步。但是，这种方法会有几个缺点。例如，如果计数器被动态线程集合使用，则线程可能需要注册和注销，并且可能需要有为了读取计数器而对线程的部分进行迭代的途径。此外，这种方法会按使用该计数器的线程数为因子增加空间需求。在各种实施例中，本文所述的技术可以以变化的程度缓解这些缺点。

[0029] 在其中附加空间开销不期望或者不可接受并且计数器必须精确的情况下，随机化的退避(RBO)技术可以被用来在重争用情况下至少避免完全的灾难。在一些实施例中，NUMA锁算法或队列(cohort)锁(可以在争用情况下通过在另一个节点上需要之前在给定的NUMA节点内交出锁多次来显著提高性能和扩展性)可以改善对采用RBO的计数器的争用管理。例如，当线程尝试利用CAS类型指令递增计数器失败时(例如，当争用发生时)，它只能在获取队列锁之后重试其递增计数器的尝试，由此鼓励在另一个节点更新之前在一个NUMA节点上的多次更新。这种技术已经显示对相对于RBO提高性能是有效的。但是，由于队列锁的空间开销，这种技术可以对本文所述的其它方法提供很小或者没有优点。

[0030] 在一些实施例中，类似于以上提到的NUMA锁但是不添加显著空间开销的方法可以使用计数器值的少数几位来识别NUMA风格存储器体系架构中的节点中哪个节点当前具有

优先权。在此类实施例中，当计数器递增时，这些位的值可以(作为正常操作的一部分)不时地变化，从而给予另一个节点更新计数器的优先权。换句话说，在几次递增操作之后(依赖于指示优先节点的位的位置)，由于这些递增操作而导致的这些位的值的变化会造成另一个节点变成优先节点。在此类实施例中，其它节点上的线程可以延迟其更新，从而使得优先节点上的线程更有可能会执行连续的更新。应当指出，一般而言，用来识别优先节点的位不可以包括最低位(即，最频繁改变的那些位)，但是可以被选择，使得优先权常常足够多地变化以避免不合理的延迟。这种方法是简单的、不添加空间开销并且已经显示在递增操作跨所有节点相对均衡地散布时执行良好。但是，对不太均匀的工作量不太适合。

[0031] 用于实现NUMA感知共享计数器(例如，NUMA感知RBO类型计数器或者另一种类型的NUMA感知计数器)的方法的一种实施例由图1中的流程图示出。如在110示出的，在这个例子中，该方法可以包括在实现NUMA风格存储器体系架构的系统的给定节点上执行的线程启动共享计数器的递增。该方法还可以包括线程确定在系统中另一个节点上执行的线程当前是否具有更新计数器的优先权或者已经请求更新计数器的优先权(就像在120中)。例如，在一些实施例中，计数器的少数几位可以被用来识别最近更新计数器的线程在其上执行的节点(因此，指示那个节点作为优先节点)，或者计数器的少数几位可以基于其它标准识别当前被指定为优先节点的节点。在一些实施例中，另一种方法可以被用来指示线程是否当前具有更新计数器的优先权(或者已经请求这种优先权)，并且这种方法可以包括抗饥饿变量的使用(如在本文更详细描述的)。

[0032] 如在这个例子中示出的，该方法可以包括至少部分地依赖于所述确定(即，依赖于在系统中另一个节点上执行的线程当前是否具有更新计数器的优先权或者已经请求更新计数器的优先权)尝试递增共享计数器或者延迟其递增计数器的尝试的线程，就像在130。例如，在一些实施例中，如果线程确定(例如，基于计数器中几个指定位的值，或者另一个优先节点指示符)在另一个节点上执行的线程具有(或者已经请求)更新计数器的优先权，则线程可以延迟其递增计数器的请求，并且随后可以重试其递增计数器的请求一次或多次，例如，直到尝试成功或者直到已达到预定的重试限制(例如，根据各种争用管理策略)。如果线程确定(例如，基于计数器中几个指定位的值，或者另一个优先节点指示符)没有设置(或请求)优先权，或者线程在其上执行的节点当前具有(或者已经请求)更新计数器的优先权，则线程可以以递增计数器的一次或多次尝试继续进行(例如，直到其中一次尝试成功或者直到已达到预定的重试限制)。应当指出，在一些实施例中，已经被指定为识别优先节点的计数器位(或者专用优先节点指示符)的预定默认或初始值可以指示还没有请求或设置优先权。

[0033] 在本文所述的许多例子中，可以假设计算机系统被组织成处理器核心的群集，每个群集具有在那个群集本地的核心之间共享的一个或多个高速缓存。在此类实施例中，群集间的通信可以比群集内的通信显著更昂贵。在本文所述的至少一些例子中，术语“群集”和“节点”可以被用来指处理器核心的集合，并且，依赖于在系统中实现的NUMA机器的尺寸，核心的这个集合可以包括单个多核芯片上的核心，或者接近同一存储器或高速缓存结构的多核芯片的集合。在这些例子中，还可以假设每个群集具有该群集上所有线程已知的唯一群集id。

[0034] 图2示出了实现NUMA风格存储器体系架构的计算机系统的一部分。在这个例子中，

计算机系统包括多个经互连250彼此通信的CPU板200(示为200a-200n)。这些CPU板之一(200a)说明得比其它更详细。在一些实施例中,CPU板200当中每一个可以包括与为CPU板200a所说明的相同或相似的体系架构。在其它实施例中,每个CPU板可以包括不同数量和/或布置的处理器芯片、处理器核心、高速缓存,等等。例如,在一些实施例中,可以有紧密耦合到每个处理器芯片的、为其处理器核心(未示出)充当"本地存储器"的一个或多个存储器芯片。如图2中所说明的,计算机系统还可以包括一个或多个系统存储器260和/或其它部件270。在这个例子中,CPU板200a包括经由互连240彼此通信的四个处理器芯片(示为210a-210d),其中之一更详细地说明。在这个例子中,假设处理器芯片210a-210d包括类似于处理器芯片210a的存储器体系架构。

[0035] 在图2所说明的例子中,处理器芯片210a包括八个处理器核心(示为220a-220h),并且每个处理器核心具有各自的(专用的)级别1(L1)高速缓存(示为230a-230h)。在一些实施例中,每个处理器核心可以是多线程核心。例如,在一种实施例中,每个处理器核心可以能够并发地执行八个硬件线程。在给定处理器核心220上执行的线程可以共享用于那个处理器核心220的级别1高速缓存230,并且对这个级别1高速缓存的访问可以极其快,该高速缓存可以被认为是处理器核心220及其硬件线程本地的。此外,八个处理器核心220可以共享用于处理器芯片210a的级别2(L2)高速缓存240,并且对这个级别2高速缓存的访问也可以快,但是不像每个处理器核心自己的级别1高速缓存那么快。在这个例子中,当与对那个硬件线程本地的级别1和级别2高速缓存和/或其它存储器的访问比较时,对同一CPU板200上不同处理器芯片210的高速缓存、对不同CPU板200上处理器芯片210的高速缓存以及对各种系统存储器260的访问(关于执行处理器芯片210a的特定处理器核心220的硬件线程,所有这些都可以被看作远程访问)可以呈现越来越高的等待时间。

[0036] 如前面所指出的,在一些实施例中,通过如下方式可以在NUMA体系架构上获得性能增益,即采用鼓励具有高相互存储器本地性的线程(例如,在相同处理器芯片上的处理器核心上执行的线程,或者在彼此附近的处理器核心上执行的线程)的共享计数器以连续递增计数器,由此在多个线程启动递增那些计数器的尝试时减小高速缓存失中的整体水平。本文所述用于实现NUMA感知共享计数器(例如,可以驻留在一个或多个系统存储器260中的计数器数据结构,并且其一部分在被系统中对应处理器核心上执行的线程更新和/或读取时可以被带入各种高速缓存)的系统和方法可以导致这种高存储器本地性,因为这些技术鼓励来自单个群集中的线程的递增这种计数器的请求的批(例如,共享级别1或级别2高速缓存的线程)被顺序执行。

[0037] 如上所述,NUMA感知RBO计数器的一种实施例可以进一步通过以下的示例伪代码说明。

[0038]

```
1  // 计数器类型 : 为存储节点id而保存的3位
2  // (在1开始), 用于实际计数器的29位
3  //
4  struct Counter {
5      unsigned int val : 29;
6      unsigned int nid : 3;
7  };
8  // 用来避免饥饿的全局变量
9   // 如果非零，则保持请求所有其它节点上的线程产出的节点的id
10  //
11   //
12   unsigned int g_deferReq = 0;
14   void Defer() {
15       for ( int i=0; i<YieldAmount; i++) Pause();
16   }
18   void Inc(Counter* cP) {
19       unsigned int myNid = getNodeId();
20       if (g_deferReq && g_deferReq != myNid) Defer();
22       bool deferAsked = false ;
23       int patience = InitPatience ;
24       int backoffTime = InitBackoff ;
25       int penalty = InitPenalty ;
```

[0039]

```
26      Counter seen = *cP;
27      while (true) {
28          Counter old = seen;
29          Counter newC = {old.val+1,myNid};
30          if (( seen = CAS(cP, old, newC)) == old) break;
31          if (seen. nid != myNid) {
32              if ( patience-- > 0) {
33                  // longer back-off, as long as g_deferReq is not set
34                  for ( int i=0; i<backoffTime + penalty; i++) {
35                      if (g_deferReq) break;
36                      Pause ();
37                  }
38                  if (g_deferReq && (g_deferReq != myNid)) Defer ();
39                  backoffTime *= 2; penalty *= 2;
40              } else {
41                  // 发出请求的节点失去耐心；试图请求产出
42                  if (! deferAsked)
43                      deferAsked = (CAS(&g_deferReq, 0, myNid) == 0);
44                  if (! deferAsked && g_deferReq != myNid) {
45                      // 另一个节点击败请求者，以设置deferReq
46                      Defer ();
47                      patience = InitPatience ;
```

[0040]

```
48                }
49                backoffTime = InitBackoff ;
50             }
51          } else {
52             if ( patience < InitPatience ) {
53             //现在提出到用于发出请求的节点的优先权的过渡；初
始化backoff
54             patience = InitPatience ; backoffTime = InitBackoff ;
55             }
56             for ( int i=0; i<backoffTime; i++) Pause ();
57             backoffTime *= 2;
58             }
59             seen = *cP;
60          }
61          if (deferAsked) g_deferReq = 0;
62       }
```

[0041] 如以上示例伪代码中所说明的，在一些实施例中，计数器可以增加几位(或者作为替代，可以从计数器偷几位，由此限制其范围)，这几位用来存储其线程当前具有更新计数器的优先权的节点的指示。这种方法可能只需要足够的附加位来存储NUMA节点的标识符(例如，节点ID)再多加一位。在所说明的例子中，该技术可以适应利用N位保持在从0到$2^{(N-1)-\lceil \log_2(\#NODES) \rceil}-1$范围内的值的计数器。例如，在一种实施例中，计数器可以包括32位，其中三位可以被偷，以存储NUMA节点ID，从而把计数器的范围限制到$2^{29}-1$。在这个例子中，这三位可以被用来存储计数器最后在其上连同计数器一起递增的节点的ID，由此为了鼓励识别出的节点上的连续递增而允许和/或请求其它节点上的线程推迟它们递增计数器的请求。在其它实施例中，计数器数据结构的不同数量的位可以被用来存储其线程当前具有更新计数器的优先权的节点的标识符。

[0042] 图3A-3F是示出本文所述的一些不同计数器数据结构的各种实施例的框图。例如，

图3A示出了计数器结构300，其中计数器305已经增加了存储节点ID的附加位310。图3B示出了计数器结构315，其中所存储计数值320的最高位子集(示为325)从计数值字段被“偷”并用来指示节点ID。在一些实施例中，这个位子集可以保留，以存储节点ID，并且各个节点ID值可以明确地写到计数器结构315的这个部分中(例如，当计数值320被更新时)。在其它实施例中，这些位的值可以仅仅反映存储在计数器结构315中的计数值320的对应位值。图3C说明了计数器结构330，其中计数值335中不包括最高位的位子集(示为340)代表节点ID。在这个例子中，这些位的值可以仅仅反映存储在计数器结构330中的计数值335的对应位值。一般而言，在不同的实施例中，所存储的计数值的任何位子集可以指定为指示节点ID，并且位子集的选择可以影响由单个节点上执行的线程的连续递增操作的次数。

[0043] 如以下更具体描述的，图3D-3E示出了在一些实施例中可以响应于某些条件而变得膨胀的计数器结构。例如，图3D示出了计数器结构345，其中保留的位355指示计数器部分350是否存储计数值或者到另一结构的指针。在这个例子中，由于所保留位355的值为零，因此计数器部分350存储计数值。类似地，图3E示出了计数器结构360，其中计数器值350的保留位370指示计数器部分365是否存储计数值或者到另一结构的指针。在这个例子中，由于保留位370的值为一，因此计数器部分365存储指针值并且这个指针值指向附加计数器结构375。在这个例子中，计数器结构375存储多个计数值，示为380a-380n。图3F示出了在一些实施例中可以用来实现概率计数器的数据结构。在这个例子中，图3F示出了包括尾部部分390和指数部分395的计数器结构385。

[0044] 应当指出，在上述NUMA感知计数器的一些实施例中，包括由以上示例伪代码表示的计数器，等待太长时间以至于不能尝试更新计数器的线程会变得不耐烦，在这个时候，它可以把它的节点ID存储到抗饥饿变量中。在此类实施例中，每个共享的计数器可以与这种抗饥饿变量关联，但是没必要每个计数器具有独立的抗饥饿变量。例如，在一些实施例中，单个抗饥饿变量可以被采用，以请求其它节点上的线程在尝试更新与抗饥饿变量关联的一个或多个共享计数器之前等待，由此使得具有不耐烦线程的节点上的线程能够把包含计数器的高速缓存线带到那个节点并递增计数器。但是，应当指出，这种方法不能防止其它线程(例如，同一节点上的其它线程)在不耐烦的线程之前递增计数器(由此保留计数器的非阻塞属性)。上述启发式方法已经显示在实践当中避免饥饿，甚至在重争用情况下。应当指出，在一些实施例中，包括在以上伪代码中说明的例子中，单个全局抗饥饿变量可以被用来请求其它节点上的线程在尝试更新多线程应用可访问的任何或全部共享计数器之前等待。

[0045] 在采用这种NUMA感知方法递增共享计数器的一些实施例中，在与已经变得不耐烦的线程在相同节点上的线程可以响应于不耐烦的线程设置抗饥饿变量而退出其延迟(例如，慢退避)并且可以尝试立即递增计数器。在此类实施例中，不管节点上的哪个线程递增计数器，这都可以具有把相关高速缓存线带到那个节点上的效果，这可以给予那个节点上的所有线程递增计数器的更好机会。在此类实施例中，不是尝试确保变得不耐烦的线程接下来递增计数器，而是其增量将帮助不耐烦线程的附近线程可以被允许在不耐烦的线程之前递增计数器。这种方法已经被发现导致比更具限制性的方法更好的性能。

[0046] 用于递增NUMA感知共享计数器的方法的一种实施例是由图4的流程图示出的。如在410说明的，在这个例子中，该方法可以包括在实现NUMA风格存储器体系架构的系统的给定节点上执行的线程启动共享计数器的递增。该方法可以包括确定(例如，作为递增计数器

的尝试的一部分)全局变量是否指示另一节点上的线程已经代表在该另一节点上执行的线程请求更新计数器的优先权,如在415中。例如,在各种实施例中,如果设置抗饥饿变量、保持特定的预定值或者保持另一节点的标识符,则抗饥饿变量可以指示另一节点上的线程已经请求更新计数器的优先权。如果全局变量指示另一节点上的线程已经为该另一节点上的线程请求优先权(示为从415肯定离开),则该方法可以包括线程延迟其递增计数器的尝试,就像在420中。例如,在不同的实施例中,线程可以延迟其尝试预定的或随机的量,在这之后,线程可以尝试递增共享计数器并且(例如,原子地,连同递增存储在计数器结构中的计数值),以更新计数器结构的节点ID部分,以反映线程在其上执行的节点(就像在425中)。应当指出,在一些实施例中,控制尝试延迟的时间量的一个或多个参数在节点ID字段指示最后的更新由同一节点上的线程执行时可以与在节点ID字段指示最后的更新由不同节点上的线程执行时具有不同的值。

[0047] 在这个例子中,如果全局变量指示另一节点上没有线程已经请求更新计数器的优先权(示为从415的否定离开),则该方法可以包括线程尝试递增共享计数器并且(例如,原子地,连同递增存储在计数器结构中的计数值),以更新计数器结构的节点ID部分,以反映线程在其上执行的节点(就像在425中)。在一些实施例中,递增计数值的尝试以及计数器结构的节点ID部分的更新可以利用单个CAS类型操作或者类似的同步操作执行。如在这个例子中所说明的,如果递增计数器和节点ID的尝试成功(示为从430肯定离开),则递增操作可以完成,就像在435中。另一方面,如果递增计数器和节点ID的尝试不成功(示为从430否定离开),并且如果计数器结构的节点ID部分不指示另一节点上的线程是更新计数器的最近线程(示为从440否定离开),则该方法可以包括线程延迟其递增计数器的尝试,就像在460中。例如,在不同的实施例中,线程可以延迟其尝试预定或随机的量,其后,线程可以重试其递增共享计数器并更新计数器结构的节点ID部分的尝试,以反映线程在其上执行的节点(示为从460到425的反馈)。

[0048] 如在这个例子中所说明的,如果递增计数器和节点ID的尝试不成功(示为从430否定离开),并且如果计数器结构的节点ID部分指示另一节点上的线程是更新计数器的最近线程(示为从440肯定离开),则该方法可以包括确定全局变量是否指示另一节点上的线程已经请求节点优先权(就像在445中)。如果是这样(示为从445肯定离开),则该方法可以包括线程延迟其递增计数器的尝试,就像在460中。例如,在不同的实施例中,线程可以延迟其尝试预定或随机的量,其后,线程可以重试其递增共享计数器并更新计数器结构的节点ID部分的尝试,以反映线程在其上执行的节点(示为从460到425的反馈)。如果全局变量不指示另一节点上的线程已经请求节点优先权(示为从445否定离开),但是线程的耐心已经耗尽(示为从450否定离开),则该方法可以包括线程延迟其递增计数器的尝试,就像在460中。例如,在不同的实施例中,线程可以延迟其尝试预定或随机的量,其后,线程可以重试其递增共享计数器并更新计数器结构的节点ID部分的尝试,以反映线程在其上执行的节点(示为从460到425的反馈)。否则(示为从445否定离开和从450否定离开),该方法可以包括线程更新为其节点请求优先权的全局变量(就像在455中),然后延迟其递增计数器的尝试,就像在460中。

[0049] 虽然上述递增共享计数器的NUMA感知方法可以在一些实施例中在重争用情况下产生比标准RBO方法更高数量级的吞吐量,但是它在低争用场景下会强加显著的开销。例

如，迄今为止所描述的方法包括在递增计数器的每次尝试之前测试抗饥饿标志。在其它实施例中，自适应NUMA感知方法可以被采用，其中递增操作依赖于计数器所经历的当前、最近或历史争用量。例如，在一些实施例中，自适应NUMA感知方法可以最初通过递增不记录最近递增计数器的线程的节点ID的定期计数器(regular counter)来响应递增计数器的请求。例如，计数器数据结构可以被初始化(例如，在多线程应用的初始化阶段期间)为指示任何线程都可以尝试递增计数器而不必还写到计数器的节点ID部分的初始或缺省值。在此类实施例中，虽然没有记录节点ID，但是可能不需要检查抗饥饿变量。

[0050] 在这种自适应NUMA感知方法中，在成功递增计数器之前，比预定次数更多地重试其递增计数器的尝试的线程(例如，快速相继地多于三次，之后是具有随机退避期的16次)可以(一旦其最后成功)在计数器中记录其节点ID。其后，上述较慢但更可扩展的NUMA感知计数可以响应于递增计数器的后续请求而应用。在一些实施例中，计数器可以偶尔复位(或返回)普通计数器(例如，周期性地或者根据各种策略，包括对共享计数器的争用的减少)，使得偶然争用的效果不永远存在。例如，计数器的节点ID部分可以偶然复位到指示任何节点上都没有线程具有(或已经请求)递增计数器的优先权的初始或缺省值，并且当这个初始或缺省值存储在计数器的节点ID部分中时尝试递增计数器的线程可以尝试递增计数器而不必还把值写到该计数器的节点ID部分。这种自适应NUMA感知方法已经显示在所有争用级别都与最好的现有RBO方法以及比以上描述的非自适应NUMA感知方法具有竞争力。

[0051] 用于递增依赖于对共享计数器的争用的NUMA感知共享计数器的方法的一种实施例由图5中的流程图示出。如在510说明的，在这个例子中，该方法可以包括在给定节点上执行的线程启动共享计数器的递增。如果共享计数器的节点ID部分识别出具有(或已经请求)优先权的节点(示为从515肯定离开)，则该方法可以包括继续其递增共享计数器的尝试，就像在图4中所示出的方法中，从元素415开始。如果共享计数器的节点ID部分没有识别出具有(或已经请求)优先权的节点(示为从515否定离开)，则该方法可以包括线程尝试递增共享计数器，就像在520中。在一些实施例中，递增共享计数器的尝试可以利用CAS类型操作或类似的同步操作执行。

[0052] 如在这个例子中说明的，如果递增共享计数器的尝试成功(示为从530肯定离开)，则递增操作可以完成(就像在535中)。另一方面，如果递增共享计数器的尝试不成功(示为从530否定离开)但是还没有到达重试限值(示为从540否定离开)，则该方法可以包括线程重试其递增共享计数器的尝试一次或多次，有或没有延迟，就像在545中。例如，线程可以重复其利用单个CAS类型操作或类似的同步操作递增共享计数器的尝试，有或没有其间的退避期。这在图5中是由从545到530的反馈来说明的。如果递增共享计数器的尝试不成功(示为从530否定离开)并且已经达到重试限值(示为从540肯定离开)，则该方法可以包括线程尝试递增共享计数器并且更新计数器结构的节点ID部分，以反映线程在其上执行的节点，有或没有延迟，就像在550中。如在这个例子中所说明的，如果这个尝试不成功(示为从555肯定离开)，则该方法可以包括重复递增共享计数器和更新计数器结构的节点ID部分的尝试一次或多次，直到其成功(或者直到由于各种适用的重试或争用管理策略而中止)。这在图5中通过从555到550的反馈来表示。一旦递增共享计数器和更新计数器结构的节点ID部分的尝试成功(示为从555肯定离开)，递增操作就可以完成，就像在560中。

[0053] 在各种实施例中，迄今为止所述的计数器在重争用情况下可以实现良好的单线程

性能和扩展性。但是，其优于简单RBO类型计数器的优点会在中等负载下减小，因为会有更少在同一节点上执行连续递增的机会。此外，这些计数器会对特定于系统的调谐敏感，这会使它们比一些其它方法更不稳定。在其它实施例中，使用稍多空间的计数器可以减小或消除这些效果，其中一些计数器在以下描述。

[0054] 在一些实施例中，被称为“多行(multiline)”方法的方法可以被用来避免昂贵的跨节点通信，而不会引入上述每线程计数器部件的缺点。例如，在一些实施例中，多行方法可以采用每NUMA节点的单独计数器部件。在此类实施例中，每节点部件的同步可以利用CAS类型指令实现，以递增每个计数器部件，有或没有递增计数器的尝试之间的随机化退避期。应当指出，当在这种情况下利用CAS类型指令同步时，不需要担心跨节点争用。当采用多行方法时，读取计数器可以涉及依次读取每个部件，没有同步，并且返回所读取的值之和。应当指出，这种方法的正确性可以依赖于递增操作只向计数加一的递增操作。但是，在其中这种假设不适用的实施例中，其它技术可以被用于相同的效果。

[0055] 虽然当采用多行方法时空间的增加受节点个数的限制，但是优选的是对只很少递增的计数器完全避免空间的增加。在一些实施例中，在本文被称为“多行-适应”方法的自适应方法可以被采用，其中递增操作依赖于对计数器的当前、最近或历史争用量。例如，在一些实施例中，多行-适应方法可以最初采用并递增标准计数器，并且可以“膨胀”其，以便只有多于预定次数(例如，在一种实施例中是四次)的递增标准计数器的尝试失败时才使用上述多线技术。其它策略可以在其它实施例中使用，例如，如果频繁地造成远程高速缓存失中就膨胀计数器。在一些实施例中，膨胀计数器可以包括分配包括每个节点一个计数器的附加结构并且用指向那个结构的指针代替标准计数器。在一些此类实施例中，初始(定期)计数器结构的一位可以保留，以区分初始结构是存储到附加结构的指针还是计数器值。这种计数器的一个例子在图3D-3E中示出并且在以上描述。

[0056] 在一些实施例中，用于采用多行-适应方法的低争用计数器的空间开销可以仅仅是保留的位(在实践当中，这将减小计数器的范围一半)并且更高的空间开销只能适用于经历更高争用的计数器(根据各种预定的争用管理策略)。在一些实施例中，多行-适应方法引入用于争用计数器的额外间接性水平，这会减慢计数器的递增操作。但是，在实践中，当计数器有争用时，这不会导致显著的性能问题，因为它可以减小初始计数器结构上CAS类型递增尝试的速率(由此减小多线程应用经历的整体争用)。

[0057] 用于响应于争用而膨胀共享计数器的方法的一种实施例由图6中的流程图示出。如在610说明的，在这个例子中，该方法可以包括在给定节点上执行的线程启动共享计数器的递增。在一些实施例中，线程可以尝试递增共享计数器(就像在620中)，例如，利用CAS类型的操作，有或没有居间的退避期。如果递增共享计数器的尝试成功(示为从630的肯定退出)，则递增操作可以完成(就像在635中)。如果递增共享计数器的尝试不成功(示为从630的否定退出)但是适用的重试限制条件还未满足(示为从640的否定退出)，则该方法可以包括线程重复其递增计数器的尝试一次或多次，直到其成功或者直到重试限制条件已经满足。这在图6中由从640到620的反馈来说明。在各种实施例中，重试限制条件可以基于未成功尝试的次数、高速缓存失中的次数或者基于另一适用重试或争用管理策略。

[0058] 如在这个例子中说明的，如果递增共享计数器的尝试不成功(示为从630的否定退出)并且适用的重试限制条件已经满足(示为从640的肯定退出)，则该方法可以包括利用到

包括每个节点一个计数器的结构的指针(即,一个或多个节点本地的计数器)代替共享计数器(或者其计数部分),就像在650中。例如,在一些实施例中,计数器的一位可以用来指示计数器部分的值当前是代表计数值还是代表到多个计数器结构的指针。该方法可以包括线程尝试递增其节点本地计数器一次或多次直到成功,如660中那样。例如,线程可以尝试使用CAS类型操作或类似的同步操作来递增其节点本地计数器,而有或没有居间的退避时段。如在这个例子中所说明的,在一些实施例中,在其中一个节点上的线程的读取共享计数器的值的后续操作可以通过读取所有节点本地的计数器并且返回其计数器值之和来做这件事。

[0059] 在一些实施例中,本文所述的多行-适应方法可以提供在低争用水平在空间开销和吞吐量方面都比上述基本RBO计数器有竞争性、随着增加的争用扩展良好并且在高争用情况下产生比基本RBO计数器高得多的吞吐量(例如,在一些实验中,多于700x吞吐量)的计数器。应当指出,在一些实施例中,由于利用单一部件的同一节点上的线程之间的争用,采用多行和多行-适应方法的计数器会遭受高争用水平。在一些此类实施例中,这种类型的争用可以通过使用每节点更多部件来缓解。例如,虽然每节点的部件必须在单独的高速缓存行中,以避免节点之间的错误共享,但是,如果每个节点采用多于一个部件,则在同一高速缓存行中为单个节点定位多个部件不是不合理。虽然错误共享在这种情况下仍然会强加一些开销,但是可以只在一个NUMA节点中。此外,仍然会有使用多个部件的好处,因为更少CAS错误将在这种情况下发生。因此,在一些实施例中,有可能在不增加空间使用的情况下利用这种方法提高性能。

[0060] 应当指出,在一些实施例中,由于多行方法造成的附加空间开销在具有大量统计计数器的系统中会是不可接受的,其中大部分统计计数器不是重争用的。虽然上述多行-适应方法会招致只对有争用计数器的这种空间开销,但是,如果不同的计数器在不同时间发生争用,则这会随时间推移导致过多开销。此外,在一些实施例中,这些方法增加读取计数器的操作的等待时间和/或,在一些语境下,它们会由于其动态分配的存储器的使用而不可接受。如以下更详细描述的,在一些实施例中,如果并且当计数器不需要精确时,则这些问题中一些或全部可以被避免。

[0061] 如前面所指出的,甚至在中等争用水平,简单的不同步计数器通常也丧失计数器更新的显著部分。因为计数器常常被用来检测各种系统事件的过高比率,所以这些单纯实现(具有讽刺意味的是)在它们应当提供的数据最重要的时候最低效。然而,在一些语境下并且对于一些应用,可以不需要精确的计数。如以下更详细描述的,在一些实施例中,计数器可以在仍然旨在维持未由单纯计数器实现实现的规定准确性水平的同时采用这种灵活性。

[0062] 一种现有的概率计数器(有时候被称为“Morris计数器”)可以代表比它所包含的位数(例如,八位)通常表示的更大的值范围。Morris计数器通过存储计数值的概率近似来做这件事,其中计数值的概率近似值在本文被称为v(n),其中n是根据以下的精确计数(即,对应的递增操作被调用多少次):

[0063] $v(n) = \log(1+n/a)/\log(1+1/a)$.

[0064] 在这个例子中,a代表其值控制计数器准确性的参数,如以下解释的。在这个例子中,向n/a添加一(就像在分母中一样)确保函数被良好定义并且当n=0时等于零。此外,除以log(1+1/a)确保当n=1时函数为一。换句话说,这种近似确保计数器至少对于值零和一

包含准确的值。根据这个定义，当存储在计数器中的值为v时，它所代表的精确计数是：

[0065] $n(v)=a((1+1/a)^{v}-1)$.

[0066] 在本文各种描述中，物理存储在概率计数器中的值v可以被称为“存储的值”，并且它所代表的值n(v)可以被称为由该概率计数器“计数”的所发生类型的事件个数的“预计值”或“估计值”。换句话说，Morris计数器存储概率近似v(n)，其中n是精确计数。在这个例子中，所存储的值必须是整数，就像在这个例子中假设只使用八位。因此，精确计数不能从所存储的值确定。因此，没有确定性的途径来知道何时递增存储在计数器中的值，以反映已发生足够的递增，使得计数器的值现在应当由更高的所存储的值来表示。为了解决这个问题，当计数器包含值v时，Morris计数器算法如下按概率p(v)递增所存储的值：

[0067] $p(v)=1/(n(v+1)-n(v))$

[0068] 直观地，这意味着，平均而言，存储在Morris计数器中的值将在存储给定的值v之后每n(v+1)-n(v)个递增操作递增一次。这确保由所存储的值预计的值是其预期值等于精确计数的随机变量。为了避免对每次递增计算概率，用于实现这种概率计数器的现有算法预先为a的给定值计算所有256个概率，并且把它们存储在查找表中。在这个例子中，查找表不需要为每个计数器复制，而是只为每个准确性类(即，a的每种选择)复制。

[0069] 在这个例子中，参数a可以既确定Morris计数器可以代表的范围又可以确定预计和实际计数之间预期的误差，作为预计值与实际计数的标准偏差(STDV)之比(有时候被称为相对STDV，或者RSTDV)。当精确计数为n时预计值的方差由$\sigma^2=n(n-1)/2a$给出，据此，当n变大时，RSTDV粗略地为$1/\sqrt{2a}$。在一个例子中，选择a=30的准确性参数值产生大约1/8的RSTDV。在这个例子中，a的这种选择允许计数器代表n(255)，这是大约130000。虽然这对于只使用八位的计数器结构是优秀的，但是对于在现代计算机系统中使用的许多类型的统计计数器这不是令人满意的(关于范围和/或准确性)。如以下更详细描述的，在一些实施例中，这种方法可以被修改，以便实现具有大得多的范围和更高准确性的可扩展计数器。

[0070] 应当指出，因为n(v)在n中是指数的，所以对Morris计数器的更新随着精确计数增长而变得更不频繁。在一些实施例中，概率计数器可以采用这个属性，以便减小对频繁更新的共享计数器的争用，同时界定预期的误差。在一些实施例中，可以实现提供比利用上述Morris计数器方法可能的更大范围和更高准确性的概率计数器。应当指出，仅仅把上述方法扩展到使用更多位的Morris计数器在一些语境下可能不是可接受的，因为当使用更多计数器位时，为所有可能的存储值预先计算更新概率会变得显著不期望。在一些实施例中，概率计数器和以下描述的对应的递增操作可以以避免这种需求的方式扩展上述技术。例如，已经观察到，从v到v+1递增所存储的值是具有因子a/(a+1)的v中的几何级数，如下所示：

[0071] $n(v+1)-n(v)=a((1+1/a)^{v+1}-(1+1/a)^{v})$

[0072] $=a((1+1/a)^{v}(1+1/a-1))$

[0073] $=(1+1/a)^{v}$

[0074] $\Rightarrow p(v)=1/(1+1/a)^{v}=(a/(a+1))^{v}$

[0075] 因此，在一些实施例中，对于给定的值p(v)，p(v+1)的值可以简单地通过用a/(a+1)乘以p(v)来计算。在一些实施例中，这个常量可以预先计算，以避免重复执行这个浮点除法运算。还已经观察到(例如，给定以上)$n(v)=a(1/p(v)-1)$。因此，在一些实施例中，概率

计数器的所存储计数器值v的预计值n(v)可以直接从p(v)计算，而无需知道v。实际上，在一些实施例中，这么做可以比直接从v计算n(v)快五倍多。因此，在一些实施例中，不是在概率计数器中存储v，就像在以上的Morris计数器例子中，而是用于概率计数器的计数器结构可以代替地存储浮点值p(v)。在一个例子中，这种计数器结构可以存储p(v)的32位浮点表示，但是在其它实施例中，范围和/或准确性可以通过利用64位双字存储p(v)的值来进一步扩展。在一些实施例中，利用这种方法，对于以该计数器为目标的每个被调用的递增操作，存储在计数器中的值p可以被读取，并且利用概率p，可以用等于p*a/(a+1)的值代替。当与上述Morris计数器方法比较时，这种方法可以提供对预计计数器值的更快评估，并且当利用b位表示计数器时可以避免预先计算并存储所有2^b位的值的需求。作为替代，只有产生期望RSTDV的a的值和a/(a+1)的对应值可能需要预先计算。

[0076] 在这种概率计数器的各种实施例中，在以该计数器为目标的每个递增操作期间，所存储的值可以以概率p更新，其中p可以等于所存储的概率值本身(即，最近存储的值)(或者可以依赖于其来确定)。例如，在一种实施例中，递增操作可以采用线程本地的、具有参数(6,21,7)的XOR偏移伪随机数发生器，该发生器可以返回整数i，i具有在1和最大整数值，MaxInt(在这个例子中，将等于$2^{32}-1$)之间的值。在这个例子中，如果i/MaxInt≤p，则所存储的值可以被更新。在一些实施例中，概率计数器结构可以存储(MaxInt*p)(例如，作为浮点数)，因此递增操作只需要比较i与所存储的值，以确定是否更新所存储的值。这个存储的值在本文可以被称为“阈值”。在这个例子中，初始阈值T_0=MaxInt，并且当所存储的值被更新T_i时，如果并且仅当伪随机数发生器返回的数字至多为T_i，当前值T_i被值$T_{i+1}=T_i*a/(a+1)$代替。根据一种实施例，可以用来实现这种技术的示例伪随机代码在下面给出。

```
1  // 对于RSTDV，准确性作为百分比数给出
2  //
```

[0077]

```
3  template <int Accuracy>
4  class ProbCounter {
5  private :
```

```
6       float threshold ;


8     // 静态（每个准确性类全局的）信息
9     //
10        static float s_a ;
11        static float s_ probFactor ;     // a/(a+1)


13    public:


15        static StaticInit () {
16            // a = 1/(2*err ^2)
[0078]
17            //
18            float tmp = (( float )Accuracy/100.0);
19            s_a = 1/(2*tmp*tmp);
20            s_probFactor = s_a /( s_a +1.0);
21        }


23        ProbCounter() {
24            threshold = (double)MaxInt;
25        }


27        unsigned int GetVal() {
```

```
28          float pr = threshold /MaxInt;
29          float val = (1.0/ pr - 1.0)*s_a;
30          return lroundf ( val );
31      }


33      void Inc () {
34          unsigned int r = rand ();
35          float seenT = threshold ;


37          while(true) {
38              if ( r > (unsigned int)seenT) return;
```

[0079]

```
40              bool overflow = (seenT < s_a + 1.0);
41              float newT = seenT * s_probFactor ;
42              if (overflow) newT = ( float )MaxInt;


44              float expected = seenT;
45              seenT = CAS(&threshold, seenT, newT));
46            if (seenT == expected) return;
47          }
48      }
49  }
```

[0080] 用于递增存储浮点值的概率计数器的方法的一种实施例由图7中的流程图示出。如在710说明的，在这个例子中，该方法可以包括在给定节点上执行的线程启动存储浮点更

新概率值(诸如本文所描述的那些)的共享概率计数器的递增。该方法还可以包括线程依赖于所存储的概率值和整数随机数的值来确定共享计数器是否应当递增,就像在720中。例如,在一些实施例中,确定可以依赖于其值在0和预定最大值(例如,maxint)之间的整数随机变量的值。应当指出,在一些实施例中,确定可以涉及浮点运算的使用,以比较更新概率的浮点表示与这个整数随机数。如果线程确定它不应当递增共享计数器(示为从730否定离开),则递增操作可以完成(即,不递增共享计数器),就像在755中。

[0081] 如在这个例子中说明的,如果线程确定它应当递增共享计数器(示为从730肯定离开),则该方法可以包括线程通过尝试在共享计数器中存储等于所存储更新概率乘以依赖于期望的准确性百分比的概率因子的新值来尝试递增计数器,就像在740中。例如,在一些实施例中,线程可以尝试利用单个CAS类型操作(或者类似的同步操作)在共享计数器中存储新值,有或没有居间的退避期。如果递增共享计数器的尝试不成功(示为从750否定离开),则该方法可以包括线程重复其递增共享计数器的尝试一次或多次,直到其成功(或者直到尝试由于各种适用的重试或争用管理策略而退出-未示出)。这在图7中由从750到720的反馈说明。应当指出,在这种情况下,该方法可以包括(基于更新概率)重复确定是否更新所存储的值,因为,如果递增共享计数器的尝试由于冲突而失败,则这可以指示另一个操作(例如,另一线程的递增操作)已经从进行前一确定开始修改了更新概率。一旦递增共享计数器的尝试成功(示为从750肯定离开),则递增操作可以完成,就像在755中。如在这个例子中说明的,在一些实施例中,线程读取共享计数器的后续操作可以通过读取所存储的更新概率并依赖于所存储的更新概率和期望的准确性百分比计算共享计数器的预计值来做这件事,就像在760中。

[0082] 在上述概率计数器的一些实施例中,需要注意避免在T_i变得太小时更新其,因为这会造成计数器的属性丢失。特别地,可以指出,因为这种方法适用整数伪随机数发生器,所以,如果更新不减小所存储阈值的整数部分,则这可以真正影响更新的概率。

[0083] 在一些实施例中,已经观察到,至少当$T_i \geq a+1$时$T_i - T_{i+1} \geq 1$。因此,在一些实施例中,当这不再为真时,概率计数器可以复位。在其它实施例中,如果这在给定语境中和/或对于给定的多线程应用是优选的,则在这种情况下会发生误差。在其中选择a=5000(例如,为了实现1%的RSTDV)并且使用32位计数器的例子中,当预计值低于MaxInt值大约0.02%时,这个阈值可以被越过。因此,当与单纯32位计数器比较时,概率计数器可以实现低相对误差和好得多的扩展性,而不显著减小所实现的计数器的范围。

[0084] 在一些实施例中,迄今为止所描述的概率计数器方法可以在计数器变得有争用并且达到更高的值时执行得非常好,但是当争用低并且预计计数器值低时会比标准的基于CAS的计数器显著更慢。在一些实施例中,这种概率计数器的混合版本(在本文被称为"概率-适应"计数器)可以被采用,其中递增操作依赖于对计数器的当前、最近或历史争用量。例如,在一些实施例中,这种自适应概率计数器可以最初通过递增标准并发计数器(例如,利用CAS类型指令)对递增计数器的请求作出响应,但是,如果CAS操作失败多次(例如,根据预定的重试限值或者其它争用管理策略),则它可以切换到上述概率计数方案。例如,在一种实施例中,概率计数器结构可以在64位字的一半中存储标准计数器,并且在另一半中存储概率计数器。当遇到争用时,递增操作可以从更新结构的标准计数器部分切换到更新概率计数器部分。在这个例子中,读取计数器可以包括把通过计数器结构的概率计数器部分

预计的值添加到由结构的标准计数器部分存储的值。这种自适应方法可以尤其适于用在访问数千计数器的多行程应用中，其中只有几个常常(或者曾经)被争用。

[0085] 用于依赖于对计时器的争用来递增混合计数器的方法的一种实施例由图8中的流程图示出。如在810说明的，在这个例子中，该方法可以包括启动混合共享计数器(例如，包括标准计数器部分和概率计数器部分的计数器)的递增的多线程应用的线程。应当指出，在这个和其它例子中，启动共享计数器递增的线程可以是共同表示多个并发执行的原子事务的多个线程之一，并且该共享计数器可以从这些事务当中一个或多个中访问。该方法还可以包括线程尝试递增共享计数器的标准计数器部分，就像在820中(例如，利用CAS类型操作或者类似的同步操作)。如果尝试成功(示为从830肯定离开)，则递增操作可以完成，就像在870中。如果递增混合共享计数器的标准计数器部分的尝试不成功(示为从830否定离开)但是还没有达到重试限制条件(示为从840否定离开)，则该方法可以包括线程重试其递增共享计数器的标准计数器部分的尝试一次或多次，有或没有延迟，就像在845中，并且确定这些尝试是否成功(示为从845到840的反馈)。应当指出，在各种实施例中，重试限制条件可以是一个或多个之前的CAS类型操作未能递增计数器的标准部分和/或指示对共享计数器的争用的一个或多个其它因素。

[0086] 如果递增混合共享计数器的标准计数器部分的尝试不成功(示为从830否定离开)并且已经达到重试限制条件(示为从840肯定离开)，则该方法还可以包括线程通过尝试递增共享计数器的概率计数器部分来尝试递增混合共享计数器(就像在850中)。如果这种尝试不成功(示为从860否定离开)，则该方法可以包括线程重复其递增混合共享计数器的概率计数器部分的尝试，直到其成功(或者直到尝试由于各种适用的重试或争用管理策略而退出-未示出)。这在图8中由从860到850的反馈说明。如果递增混合共享计数器的概率计数器部分的尝试成功(示为从860肯定离开)，则递增操作可以完成，就像在870中。如在这个例子中所说明的，在一些实施例中，线程读取混合共享计数器的后续操作可以通过读取标准计数器部分的值和概率计数器部分的值并且返回其和来做这件事，就像在880中。

[0087] 关于其准确性、低争用情况下的性能，高争用情况下的扩展性以及空间使用，上述概率计数器可以适于用在许多语境下并且用于许多类型的多线程应用。但是，在其它语境下，在不使用浮点运算的情况下提供相似属性的计数器可能更合适。因此，在一些实施例中，更新概率可以限制到总是二的非正幂。这可以使得决定是否更新计数器相对容易(具有适当的概率)，并且如果是这样，就计算下一个更新概率，而不使用任何浮点运算。以下描述两个这种计数器(连同对应的递增和读取操作)。

[0088] 在其中只有二的非正幂用作更新概率的实施例中，响应于对递增计数器的请求，通过确定整数随机数的低k位是否全为零，递增操作可以决定是否以概率$1/2^k$更新计数器(而不需要执行任何浮点计算)。应当指出，这种方法采用比上述方法更粗粒度的更新概率，因为，相对于按因子a/(a+1)减小其，每个更新可以只把更新概率减半。减小更新概率对于性能和扩展性是重要的(至少在一定程度上)。但是，如果更新概率在每次更新后减半，则它会太快变小，这会减小计数器的准确性。因此，在一些实施例中，根据用于管理这种折中的策略，相同的更新概率可以在最终减小其之前被重复使用，策略的例子在本文描述。

[0089] 在以下描述的例子中，计数器值可以利用二进制浮点(BFP)表示。例如，计数器可以存储对(m,e)，该对代表预计值$m*2^e$(即，m是尾数，而e是指数)。计数器变量中的不同位字

段被用来存储m和e。例如，如果四位被用来存储用于e的值并且28位被用来存储用于m的值，则计数器结构可以代表多达$(2^{28}-1)*2^{15}$的计数器值，或者大约2K倍的MaxInt。

[0090] 在以下所描述的例子中，当指数为e时，计数器可以以概率2^{-e}更新。就像在前面的例子中，为了保持计数器的预期预计值等于至今所执行的递增的总数，2^e可以在利用概率2^{-e}递增计数器时添加到预计值。注意，在各种实施例中，2^e可以以至少两种途径添加到由(m，e)表示的计数器的预计值。例如，一种途径是把所存储的值更新为(m+1，e)。只能在m为奇数并且指数字段不饱和的时候应用的另一种途径是把计数器更新为((m+1)/2，e+1)。在这两种情况下，添加到预计值的量都很容易看到是2^e。以下基于这种通用方法描述的实施例可以在当更新计数器时控制使用哪种方法的一种或多种策略中不同。

[0091] 用于递增存储二进制浮点值的概率计数器的方法的一种实施例由图9的流程图示出。如在910说明的，在这个例子中，该方法可以包括在给定节点上执行的线程启动存储概率计数器值作为二进制浮点数的共享概率计数器的递增，其中更新概率可从概率计数器值的指数部分计算并且限制到二的非正幂。例如，在一些实施例中，计数器结构可以包括尾数部分和指数部分，这两部分一起用来表示$m*2^e$的预计(或预期)值。该方法还可以包括线程确定共享概率计数器是否应当递增(就像在920中)。例如，在一些实施例中，共享概率计数器可以以概率$1/2^e$更新。

[0092] 在这个例子中，如果线程确定不应当递增共享概率计数器(示为从930否定离开)，则递增操作可以完成(即，不递增共享概率计数器)，就像在955中。另一方面，如果线程确定应当递增共享概率计数器(示为从930肯定离开)，则该方法可以包括线程通过尝试在共享概率计数器中存储新值来尝试递增计数器，使得其新预计值等于其之前的预计值与2^e之和，就像在940中。例如，递增计数器的尝试可以利用CAS类型操作执行，有或没有退避。应当指出，以这种途径(用((m+1)/2，e+1)代替(m，e))递增计数器减小更新计数器的概率一半。如果递增共享概率计数器的尝试成功(示为从950肯定离开)，则递增操作可以完成(就像在955中)。如果递增共享概率计数器的尝试不成功(示为从950否定离开)，则该方法可以包括重复递增共享概率计数器的尝试一次或多次，直到其成功(或者直到尝试由于各种适用的重试或争用管理策略而退出一未示出)。应当指出，在这个例子中，重复递增所存储概率计数器的尝试可以包括重复确定是否执行递增。这在图9中由从950到920的反馈说明。如在这个例子中所说明的，在一些实施例中，线程读取共享概率计数器的后续操作可以通过读取所存储的概率计数器值并计算预计值来做这件事(即，在这个例子中，返回左移指数值的尾数值)，就像在960中。应当指出，在这个例子中，这等效于计算$m*2^e$。

[0093] 在一些实施例中，存储概率计数器值作为二进制浮点数的概率计数器可以采用确定性更新策略，其中更新概率可从概率计数器值的指数部分计算。这种计数器的一个例子(在本文被称为BFP-DUP计数器)可以呈现类似于上述概率计数器的属性，例如，关于RSTDV的期望界限可以被指定，并且对应的更新操作可以尽快减小更新概率，以便在确保期望RSTDV界限的同时提高扩展性。在一些实施例中，确保指定的界限可以涉及确保更新概率不会太快减小。在一些实施例中，更新策略可以造成对缺省地递增尾数的计数器的更新。但是，如果递增尾数将使其达到预定的限值(在本文被称为“尾数-阈值”)，其中阈值可能需要是偶数，则递增操作可以代替地把尾数减半(在递增其之后)并且递增指数。利用这种方法，第一尾数-阈值数的递增可以以概率$2^0=1$更新计数器，由此确保计数器达到尾数-阈值，而

不引入任何误差。其后，指数可以在计数器更新的每个尾数-阈值/2次递增(并且尾数减半)。在一些实施例中，尾数-阈值的选择可以确定指数增长多快(并且因此确定更新概率减小多快)。本文描述用于选择尾数-阈值的各种方法。

[0094] 如上所述，BFP-DUP计数器的一种实施例可以由以下给出的示例伪代码说明。

```
1  // 对于RSTDV，准确性是作为百分比数给出的
2  //
3  template <int Accuracy>
4  class BFPCounter {
5  private :
6      // BFP计数器类型：指数4位，尾数28位
7      //
```

```
8   //
9   struct Counter {
10      int mantissa : 28;
11      int exp: 4;
12      enum {MaxExp = (1<<4) - 1 , MaxMantissa = (1<<28) – 1};
13  };

15  Counter bfpData;

17  enum {
18      MantissaThreshold = 2*((30000/(Accuracy_Accuracy) + 3)/8)
19  };

21  public:

23  BFPCounter() {
24      bfpData = {0,0};
25  }

27  //注意: 所表示的值可以大于MaxInt,
28  // 因此使用64位返回值
29  //
```

[0096]

```
30      unsigned long long GetVal() {
31          Counter data = bfpData;
32          return (unsigned long long)( data . mantissa << data.exp);
33      }


35      void Inc () {
36       int r = rand ();
37          int numFailures = 0;
38          while (true) {
39              ExpBackoff(numFailures);
40              Counter oldData = bfpData;
```

[0097]

```
41              int e = oldData.exp;
42              int m = oldData. mantissa ;


44              // 选择以概率1/2^e更新计数器
45              //
46              if (( r & ((1<<e)-1)) != 0) return;


48              //假设尾数字段大到足以保持MantissaThreshold-1，
49              //因此，除非指数饱和，否则不检查尾数溢出
50              //
51              //
```

```
52              bool overflow = (e == Counter::MaxExp &&
53                               m == Counter::MaxMantissa);
54              Counter newData = {0,0};
55              if (! overflow) {
56                if ((m == MantissaThreshold - 1) &&
57                                (e < Counter::MaxExp)) {
58                    newData = {e+1, (m+1)>>1};
59                } else {
60                    newData = {e, m+1};
61                }
62              }
63              if (CAS(&bfpData, oldData, newData) == oldDdata) return;
64              numFailures++;
65          }
66      }
```

[0098]

[0099] 用于递增包括多个更新选项的概率计数器的方法的一种实施例由图10的流程图示出。如在1010说明的，在这个例子中，该方法可以包括在给定节点上执行的线程启动存储概率计数器值作为二进制浮点数的共享概率计数器的递增，其中可从概率计数器值的指数部分计算的更新概率被限制到二的非正幂。例如，在一些实施例中，计数器结构可以包括尾数部分和指数部分，这两部分一起表示$m*2^e$的预计(预期)值。该方法还可以包括线程依赖于所存储的概率计数器值和整数随机数的值确定共享概率计数器是否应当递增(就像在1020中)。例如，在一种实施例中，为了以概率$1/2^e$更新计数器，该方法可以包括确定整数随机数的低e位是否全为零(浮点运算是不必要的)。如果是这样，则以这里所描述的方式更新计数器可以减小更新计数器的概率一半。在一些实施例中，用来执行计数器更新的方法可以依赖于如果计数器的尾部部分递增则其是否将溢出和/或计数器的指数部分是否饱和。

[0100] 如在这个例子中所说明的，如果线程确定不应当递增共享概率计数器(示为从1030否定离开)，则递增操作可以完成(即，不递增共享概率计数器)，就像在1080中。另一方面，如果线程确定应当递增共享概率计数器(示为从1030肯定离开)，并且如果递增尾数将不使其等于其依赖于准确性的阈值(示为从1040否定离开)，则该方法可以包括线程通过尝试递增共享计数器的尾部部分来尝试递增计数器(就像在1070中)，在这个时候，递增操作

可以完成(就像在1080中)。在一些实施例中,尝试递增共享计数器的尾数部分可以利用CAS类型操作(或者类似的同步操作)执行一次或多次,直到其成功(或者直到由于各种适用的重试或争用管理策略而退出),有或没有居间退避期(未示出)。就像在其它例子中,如果递增共享计数器的尾数部分的尝试失败,则该方法可以包括重复图10中所示的至少一些操作,从元素1020(未示出)开始。

[0101] 如在这个例子中说明的,如果线程确定应当递增共享概率计数器(示为从1030肯定离开),但是递增尾数将使其等于其依赖于准确性的阈值(示为从1040肯定离开),并且共享概率计数器的指数部分已经处于其最大值(示为从1050肯定离开),则该方法可以包括线程把计数器复位到零(就像在1055中),在这个时候,递增操作可以完成(就像在1080中)。换句话说,该方法可以包括把(尾数,指数)对复位成值(0,0)。如果递增尾数将使其等于其依赖于准确性的阈值(示为从1040肯定离开),但是共享概率计数器的指数部分还没有处于其最大值(示为从1050否定离开),则该方法可以包括线程通过尝试递增尾数、把递增后的尾数减半并且递增指数来尝试递增计数器(就像在1060中),之后,递增操作可以完成(就像在1080中)。在一些实施例中,更新共享计数器的尝试可以利用单个CAS类型的操作或者类似的同步操作来执行,所述操作可以重复(如果必要的话),直到其成功(或者直到尝试由于各种适用的重试或争用管理策略而退出-未示出)。就像在其它例子中,如果更新共享计数器的尝试失败,则该方法可以包括从元素1020(未示出)开始重复图10中所示的至少一些操作。

[0102] 如在以上的示例伪代码中所说明的,BFPCounter类在一些实施例中可以接受(作为模板自变量)对RSTDV的期望界限,作为百分比(例如,准确性参数值1可以对应于对RSTDV的期望界限1%)。在一些实施例中,尾数-阈值参数的值可以基于期望的准确性来确定,如以下解释的。在这个例子中,递增操作(示为Inc)可以以概率$1-1/2^{e}$决定,而不更新计数器,其中e是当前存储在计数器中的指数值(就像在以上伪代码的行36-46)。在这个例子中,如果对更新计数器作出决定,则递增操作可以首先检查,以查看计数器是否已经达到其最大值(就像在行52中),在这种情况下,可以尝试把计数器更新为零。应当指出,在其它实施例中,例如,如果这在给定语境下或者对于给定的应用是优选的,则递增操作可以代替地在这种情况下发信号通知错误。否则,新对可以基于当前对来确定(如在以上行56-61中所示)。最后,递增操作可以尝试把新对存储到计数器,例如,利用CAS类型指令来确认计数器还没有改变(就像在行63中)。在这个例子中,如果CAS操作失败,则操作可以重试,以确定是否更新计数器开始。在其它实施例中,其它争用管理策略可以适用。

[0103] 在一些实施例中,本文所述递增操作的各种优化可以提高整体性能。例如,在一些实施例中,实现递增操作的代码可以“内联(inline)”公共更新情况(即,其中更新计数器的CAS类型操作成功的情况),并且在重试递增操作之前使用失败的CAS类型操作的返回值避免再读计数器数据(例如,在以上的示例代码中,bfp Data)的需求。在一些实施例中,当CAS类型的操作由于与并发更新(例如,由相同多线程应用的另一线程尝试的更新)的冲突而失败时,基于新值确定更新是否应当应用的测试可以在退避之前执行,因为情况几乎从不会如此。在一些实施例中,上述对这种计数器的所有计算都可以利用位移位和屏蔽操作执行(即,无浮点运算)。

[0104] 应当指出,类似于以上所述的现有顺序近似计数算法不支持并发更新,并且不如

上述方法灵活。在这种现有算法中，不是在计数器更新的任何时候明确地更新尾数和指数，而是仅仅通过递增所存储的值来执行更新。在这种现有算法中，当计数器的尾数部分递增超过其最大值时，溢出可以自然递增指数字段(指数字段可以被适当地放置，以确保溢出可以自然递增指数字段)。由于这种选择，使用现有算法的更新函数可以比以上所述的稍简单。但是，这可以具有很小的性能影响，因为计数器随时间推移越来越不频繁地被更新。现有算法的另一含义是更新递增指数(并且由此减小对后续操作的更新概率)的频率需要是二的幂。此外，现有算法必须实现从计数器中所存储的数据计算预计值的不同途径，因为所存储数据的尾数部分在指数递增时变成零。

[0105] 在一些实施例中，本文所述的BFP-DUP计数器在第一次递增指数之前对尾数执行的递增可以是在指数的后续递增之间对尾数执行的递增的两倍，而现有算法在指数的每次递增之前对尾数执行相同次数的递增。因此，用来建模BFP-DUP计数器的Markov链在以别的方式类似于由现有算法使用的链之前包括长度为尾数-阈值/2的确定性链。但是，应当指出，这不会改变限制内的结果，因为尾数的这些确定性递增以概率1发生，并且因此不增加计数器的不准确性。

[0106] 与和上述概率-适应计数器关联的操作形成对比，在这种BFP-DUP计数器中，对RSTDV的界定不能独立于所执行的递增操作的次数。更确切地说，当递增次数n接近无穷大时，这些技术可以在限值内提供对预期RSTDV的界定。更精确地说，这可以如下描述：

[0107] $$\limsup_{n\to\infty} A_n \le \sqrt{\frac{3}{8M-3}}$$

[0108] 在这个例子中，A_n代表在n个递增操作之后预期的RSTDV，并且M代表指数的递增之间尾数递增的次数(在这个例子中，等于尾数-阈值/2)。在一些实施例中，这个公式可以用来确定M的选择，以便实现期望的界限。例如，因为以上伪代码中的BFPCounter类接受其准确性自变量作为百分比(如上所述)，所以以上等式可以暗示以下：

[0109] $M \leqslant ((30{,}000/\text{Accuracy}^2)+3)/8$

[0110] 在这个例子中，用于尾数-阈值的对应公式在以上伪代码的行18找到(并且尾数-阈值＝2M)。应当指出，在一些实施例中，因为BFP-DUP计数器不把递增次数限制到指数递增之间的尾数，所以这种方法的使用可以提供基于这种计算选择尾数-阈值的灵活性，从而导致对准确性性能折中的更精细粒度的控制。在一些实施例中(包括在本文所述各种实验中建模的那些实施例)，准确性参数值被设置为反映对RSTDV的1％界定，从而导致尾数-阈值被设置为7500。

[0111] 用于依赖于可配置准确性参数递增概率计数器的方法的一种实施例由图11中的流程图示出。如在1110所说明的，在这个例子中，该方法可以包括多线程应用的线程启动存储概率计数器的多值表示的共享计数器的递增，其中预计计数可以从所存储的概率计数器值计算。如果线程确定它应当更新所存储的概率计数器值(示为从1120肯定离开)，则该方法可以包括线程尝试更新所存储的概率计数器值，其中更新所存储的概率计数器值的尝试依赖于可配置准确性参数的值(就像在1130中)。应当指出，在一些实施例中，更新所存储的概率计数器值的尝试(和/或这么做的确定)还可以基于所存储的概率计数器值本身(即，当前所存储的值)。

[0112] 如在这个例子中所说明的，如果更新所存储的概率计数器值的尝试成功(示为从

1140肯定离开)，则递增操作可以完成(就像在1150中)。另一方面，如果更新所存储的概率计数器值的尝试不成功(示为从1140否定离开)，则该方法可以包括重复更新所存储概率计数器值的尝试，直到其成功，或者直到尝试由于各种适用的重试或争用管理策略而退出。应当指出，在这个例子中，重复更新所存储的概率计数器值的尝试可以包括重复确定是否执行更新。这在图11中说明为从1140到1120的反馈。如在这个例子中所说明的，在一些实施例中，线程读取共享计数器的后续操作可以通过读取所存储的概率计数器并且依赖于所存储的概率计数器值计算预计计数值来做这件事，就像在1160中。

[0113] 在各种实施例中，由BFP-DUP计数器使用的确定性更新策略可以对在各种语境下和在各种多线程应用中使用有吸引力。但是，虽然对于扩展性和性能而言随着计数器增长减小更新概率是重要的，但是有时对于给定系统和工作负荷，对计数器变量的争用可以减小到几乎为零，并且偶尔更新计数器的开销将变得不明显。过了这个点，减小更新概率可以进一步用来仅仅增加计数器的不准确性。因此，一些实施例采用自适应和/或争用敏感更新策略，诸如争用敏感更新策略。例如，在一些实施例中，递增操作可以选择只有当存在(或者曾经存在)对计数器的争用时更新指数(由此减小更新概率)。换句话说，自适应BFP计数器可以采用依赖于计数器的当前、最近或历史争用数量的递增操作。例如，在一些实施例中，递增操作可以首先尝试利用CAS类型指令递增尾数(例如，无条件地，或者除非其将溢出)一次(或者另一预定次数)，并且只有其失败时，才可以决定是否利用与在上述BFP-DUP计数器中所使用的相似的策略更新指数并把尾数减半。采用这种争用敏感更新策略的BFP计数器可以在本文被称为BFP-DUP计数器。在各种实验中，显示BFP-DUP计数器可以产生类似于上述BFP-DUP计数器的性能的性能，同时在实践中实现更高的准确性。

[0114] 用于递增存储代表依赖于对共享计数器的争用的计数器值的二进制浮点值的概率计数器的方法的一种实施例由图12中的流程图示出。如在1210所说明的，在这个例子中，该方法可以包括在给定节点上执行的线程启动存储概率计数器值作为二进制浮点数的共享概率计数器的递增，其中可从概率计数器的指数部分计算的更新概率受限于二的非正幂。例如，在一些实施例中，计数器结构可以包括尾数部分和指数部分，这两部分一起代表$m*2^e$的预计(预期)值。如果递增计数器的尾数部分将使其值等于其依赖于准确性的阈值(示为从1220肯定离开)，则该方法可以包括继续递增共享概率计数器的尝试，就像在图10中所说明的方法中，以元素1040开始。

[0115] 如在这个例子中所说明的，如果递增计数器的尾数部分将不使其值等于其依赖于准确性的阈值(示为从1220否定离开)，则该方法可以包括线程通过执行递增尾数的一次或多次尝试来尝试递增共享概率计数器(就像在1230中)。在各种实施例中，线程可以重试其尝试的次数可以依赖于一个或多个适用的重试或争用管理策略，并且可以执行多个重试尝试，有或没有居间的退避期。如果线程成功递增共享概率计数器(示为从1240肯定离开)，则递增操作可以完成(就像在1250中)。如果线程没有成功递增共享概率计数器(示为从1240否定离开)，则该方法可以包括继续递增共享概率计数器的尝试，就像在图10中所说明的方法中，以元素1040开始。

[0116] 在不同的实施例中，各种争用敏感方法可以用于确定是否、何时和/或如何基于当前、最近或历史争用来更新统计计数。用于确定是否(和/或何时)递增共享计数器以及用于依赖于对共享计数器的争用递增共享计数器的方法的一种实施例由图13中的流程图说明。

如在1310所说明的，在这个例子中，该方法可以包括多线程应用的多个并发执行的线程之一启动共享计数器的递增。在一些实施例中，线程可以是共同实现多个并发执行的原子事务的多个线程之一。该方法可以包括线程确定是否或者何时更新共享计数器，其中用于确定是否或何时更新共享计数器的方法依赖于并发执行的线程之间对共享计数器的争用量(就像在1320中)。例如，该方法可以至少部分地依赖于对共享计数器的当前、最近或历史争用。

[0117] 如果线程确定它不应当更新共享计数器(示为从1330否定离开)，则递增操作可以完成(即，不更新共享计数器)，就像在1360中。另一方面，如果线程确定它应当更新共享计数器(示为从1330肯定离开)，则该方法可以包括线程尝试更新共享计数器，其中用于尝试更新共享计数器的方法依赖于并发执行的线程之间对共享计数器的争用量(就像在1340中)。再次，该方法可以至少部分地依赖于共享计数器的当前、最近或历史争用。如在这个例子中所说明的，如果更新共享计数器的尝试成功(示为从1350肯定离开)，则递增操作可以完成(就像在1360中)。另一方面，如果更新共享计数器的尝试不成功(示为从1350否定离开)，则该方法可以包括线程重试去更新共享计数器的尝试一次或多次，直到其成功(或者直到尝试由于各种适用的重试或争用管理策略而退出-未示出)。应当指出，在这个例子中，重复更新共享计数器的尝试可以包括重复确定是否或何时更新计数器。这在图13中由从1350到1320的反馈说明。应当指出，在一些实施例中，重试尝试的执行(和/或尝试可以重试的次数)还可以依赖于对计数器的争用量(包括可以造成这种最近失败的争用)。还应当指出，虽然在图13中说明的例子中在1320中说明的操作和在1340中说明的操作都被描述为依赖于对计数器的争用量，但是在其它实施例中，这些操作中只有一个可以依赖于对计数器的当前、最近或历史争用量。

[0118] 本文所描述的例子大大集中在用于实现用在其中可以存在许多计数器的语境下的统计计数器的技术，其中一些可以频繁地递增。因此，例子包括呈现低空间开销、在缺少争用的情况下有低开销以及在重争用情况下有良好扩展性的技术。虽然这些技术已经对读执行(例如，对于目标在于计数器的读取操作)必要地进行了优化，但是在一些实施例中，与这些读操作关联的成本可以对本文所述的大部分技术合理地低。

[0119] 应当指出，一般而言，可以存在检索计数器值的成本的两个主要部分。一个部分是与读必要数据的成本关联的成本，并且另一个部分是与从被读的数据计算返回值关联的成本。在本文所述的许多场景中，这些成本当中第一个有可能主导检索与给定计数器关联的计数值的成本，因为作为计数器基础的数据不可能在用于执行读操作的线程的高速缓存中。因此，数据可以需要从存储器、或者从可以在系统中不同NUMA节点上的另一个高速缓存提取。

[0120] 应当指出，读取现有单纯计数器的值可以简单地涉及读取计数器本身当中所存储的数据并且返回被读取的值。由此，它们的读成本(最多)是单个高速缓存失中的成本。在一些实施例中，本文所描述的NUMA感知RBO类型计数器或者自适应NUMA感知RBO类型计数器，或者采用在本文被称为BFP-DUP和BFP-CSUP的方法，还会招致单个高速缓存失中的成本，但是这些计数器还会招致各种屏蔽和/或移位操作以确定计数器的预计值的成本。读取本文所述的多行计数器可以需要作为计数器基础的每个高速缓存行被读取。但是，这些可以是独立的读，因此高速缓存失中可以在很大程度上并行地在大部分现代体系架构上解决。采

用本文所述的多行-适应方法的计数器上的读操作可以类似于那些现有的简单计数器，除非计数器经历要膨胀的充分更新争用，在这种情况下，读操作必须不仅读为计数器分配的多个高速缓存行，而且读确定它们在哪里的指针。所分配的高速缓存行的读操作依赖于指针的值，并且因此读操作的等待时间可以有可能包括至少顺序的两个高速缓存失中的成本，即使所有分配的行都被并行读取。以Morris和“概率-适应”计数器为目标的读操作可以都包括如果频繁执行则有可能添加显著开销的多个浮点运算。因此，基于BFP的计数器在这些场景下可以是优选的。作为替代，给定(最终)计数器的所存储值只能不频繁地改变(例如，在更新概率已经充分减小之后)，记录从所存储值计算的预计值的优化可以是合算的。

[0121] 在一些实施例中，本文所描述的可扩展统计计数器可以在包括事务存储器支持的系统中使用的时候尤其有价值，不管事务存储器支持是在硬件中、在软件中还是利用硬件和软件的组合实现的。例如，统计计数器可以在此类系统中用于各种目的，诸如在散列表中记录条目的数量，或者维护关于某块代码多经常执行的统计数据。常见的经历是计数器在原子事务中的使用使得所有事务对都冲突，因为它们全都更新计数器。如在本文所描述的，在一些实施例中，可以通过减少对其的争用来使得计数器更可扩展，或者通过分割它们，使得多个更新可以并行发生(就像在以上所述的多线方法中)或者通过减小更新的频率(就像在本文所述的概率计数器中)。在一些实施例中，与采用通常扩展很差和/或产生高度不准确计数的单纯不可扩展计数器的事务相比，这些技术可以具有显著减小利用这些计数器的原子事务将彼此冲突的经常性的副作用。

[0122] 在各种实施例中，各种计数器技术(其中一些计数器技提供精确计数，并且其它的计数器技旨在合理的相对误差，使得它们仍然对检测递增许多次的计数值有用)可以关于扩展性和/或准确性产生比单纯并发计数器更好的结果。关于吞吐量和准确性，本文所述的几个计数器都可以大大超过通常使用的统计计数器，尤其是在NUMA系统中，同时保持空间开销低。

[0123] 本文所述的许多计数器技术都很容易被看到是无锁的。此外，当采用本文所述的概率计数器技术时，重试递增计数器的尝试的需求会随时间推移变得不太有可能，因为用于计数器的更新概率随时间推移变得更小(尤其是当存在对采用上述BFP-DUP技术的计数器的争用时)。在一些实施例中，本文所述的计数器可以被修改，使得它们无需等待，在一些情况下这会添加开销和/或复杂性。把计数器修改为无需等待还会引入附加的限制(诸如事先知道最大线程数的需求)，或者会导致为了避免这种限制而对更多开销和复杂性的需求。但是，在实践当中，假定一些类型的退避方案可以在存在对计数器的争用时应用，无锁在一些实施例中会是足够强的属性，以确保对多线程应用的并发线程的进度。

[0124] 图14示出了根据各种实施例被配置为实现本文所述方法的计算系统。计算机系统1400可以是任何各种类型的设备，包括，但不限于，个人计算机系统、台式计算机、膝上型或笔记本计算机、大型计算机系统、手持式计算机、工作站、网络计算机、消费者设备、应用服务器、存储设备、诸如交换机、调制解调器、路由器等等的外围设备，或者一般而言任何类型的计算设备。在一些实施例中，计算机系统1400可以是采用NUMA型存储器体系架构和/或NUCA属性的系统中的多个节点之一，或者一般而言包括至少一个耦合到某种类型存储器(例如，高速缓存、本地存储器、远程存储器，等等)的处理器核心的任何类型的计算节点。

[0125] 用于实现本文所述的任何或全部可扩展统计计数器的机制可以作为计算机程序

产品或软件提供，该产品或软件可以包括其上存储指令的非暂态计算机可读存储介质，指令可被用来编程计算机系统(或其它电子设备)，以执行根据各种实施例的过程。计算机可读存储介质可以包括用于以机器(例如，计算机)可读的形式(例如，软件、处理应用)存储信息的任何机制。机器可读存储介质可以包括，但不限于，磁性存储介质(例如，软盘)；光学存储介质(例如，CD-ROM)；磁-光存储介质；只读存储器(ROM)；随机存取存储器(RAM)；可擦可编程存储器(例如，EPROM和EEPROM)；闪存存储器；适于存储程序指令的电或其它类型介质。此外，程序指令可以利用光学、声学或其它形式的传播信号(例如，载波、红外线信号、数字信号，等等)传送。

[0126] 在各种实施例中，计算机系统1400可以包括一个或多个处理器1470；每个处理器可以包括多个核心，任何核心都可以是单线程或多线程的。例如，如图2中所说明的，多个处理器核心可以包括在单个处理器芯片(例如，单个处理器1470)中，并且多个处理器芯片可以包括在CPU板上，两个或更多个CPU可以包括在计算机系统1400中。在各种实施例中，每个处理器1470可以包括高速缓存的层次结构。例如，如在图2中所说明的，每个处理器芯片1470可以包括多个L1高速缓存(例如，每个处理器核心一个)以及单个L2高速缓存(这可以由处理器芯片上的处理器核心共享)。计算机系统1400还可以包括一个或多个持久性存储设备1450(例如，光学储存器、磁性储存器、硬驱、带式驱动器、固态存储器，等等)以及一个或多个系统存储器1410(例如，高速缓存、SRAM、DRAM、RDRAM、EDO RAM、DDR 10 RAM、SDRAM、Rambus RAM、EEPROM，等等)。各种实施例可以包括更少或者未在图14中说明的附加部件(例如，视频卡、音频卡、附加的网络接口、外围设备、网络接口，例如ATM接口、以太网接口、帧中继接口，等等)。

[0127] 一个或多个处理器1470、(一个或多个)存储设备1450，以及系统存储器1410可以耦合到系统互连1440。一个或多个系统存储器1410可以包含程序指令1420。程序指令1420可以是可执行的，以实现一个或多个应用1422(这可以包括对共享统计计数器的一个或多个访问，如本文所述)、共享库1424或操作系统1426。在一些实施例中，程序指令1420可以是可执行的，以实现争用管理器(未示出)。程序指令1420可以以平台本地的二进制、诸如Java™字节码的任何解释语言或者以诸如C/C++、Java™的任何其它语言等等或者以它们的任意组合来编码。程序指令1420可以包括功能、操作和/或其它过程，用于实现可扩展的统计计数器和相关联的功能(例如，以可扩展统计计数器为目标的增量操作和/或读操作)，如本文所述。在各种实施例中，这样的支持和功能可以存在于共享库1424、操作系统1426或应用程序1422当中的一个或多个中。系统存储器1410还可以包括其中数据可以被存储的私人存储位置1430和/或共享存储位置1435。例如，在各种实施例中，共享存储位置1435可以存储并发执行的线程、进程或原子事务可访问的数据，这可以包括存储在实现共享统计计数器的一个或多个结构中的数据(例如，本文所述的精确计数器或概率计数器之一)。

[0128] 虽然以上已经相当详细地描述了实施例，但是，一旦以上公开内容被完全理解，各种变化和修改就将对本领域技术人员变得显然。预期以下权利要求要被解释为包含所有此类变化和修改。

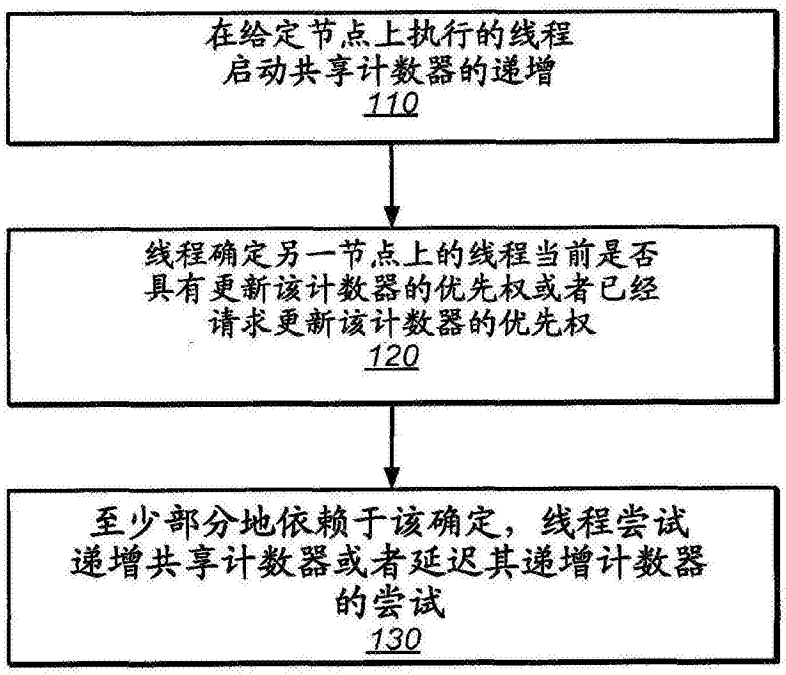

在给定节点上执行的线程
启动共享计数器的递增
110
线程确定另一节点上的线程当前是否
具有更新该计数器的优先权或者已经
请求更新该计数器的优先权
120
至少部分地依赖于该确定，线程尝试
递增共享计数器或者延迟其递增计数器
的尝试
130

图1

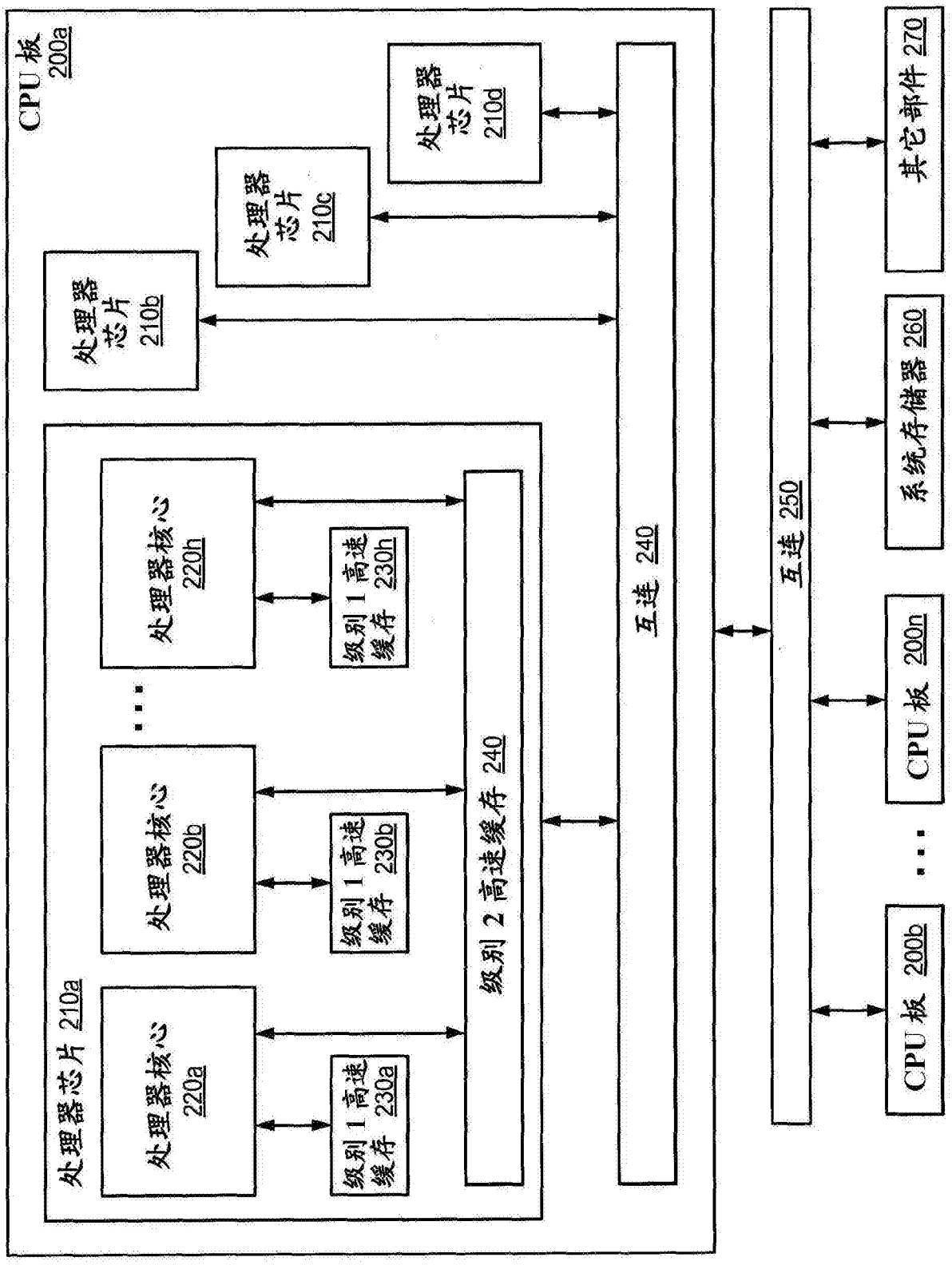

CPU 板 200a
处理器芯片 210a
处理器核心 220a
处理器核心 220b
处理器核心 220h
级别 1 高速缓存 230a
级别 1 高速缓存 230b
级别 1 高速缓存 230h
级别 2 高速缓存 240
处理器芯片 210b
处理器芯片 210c
处理器芯片 210d
互连 240
互连 250
系统存储器 260
其它部件 270
CPU 板 200b
CPU 板 200n

图2

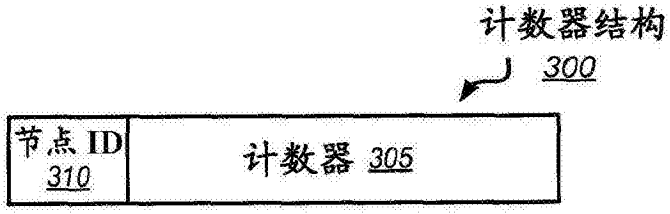

计数器结构
300
节点 ID
310
计数器 305

图3A

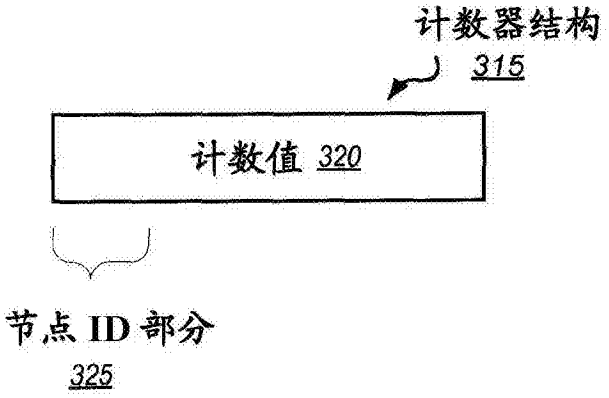

计数器结构
315
计数值 320
节点 ID 部分
325

图3B

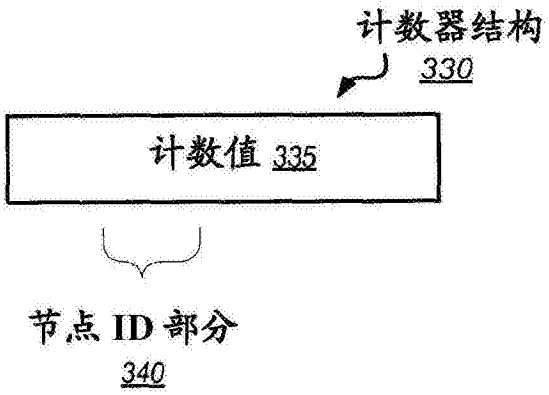

计数器结构
330
计数值 335
节点 ID 部分
340

图3C

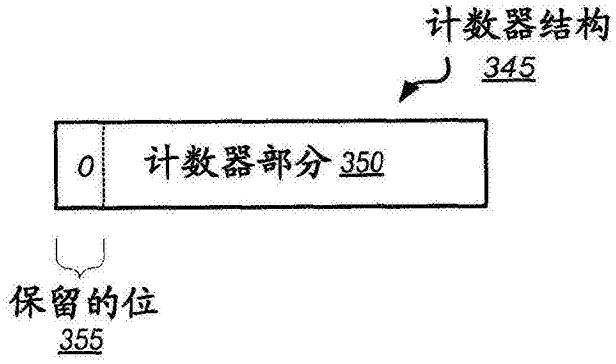

计数器结构
345
0
计数器部分 350
保留的位
355

图3D

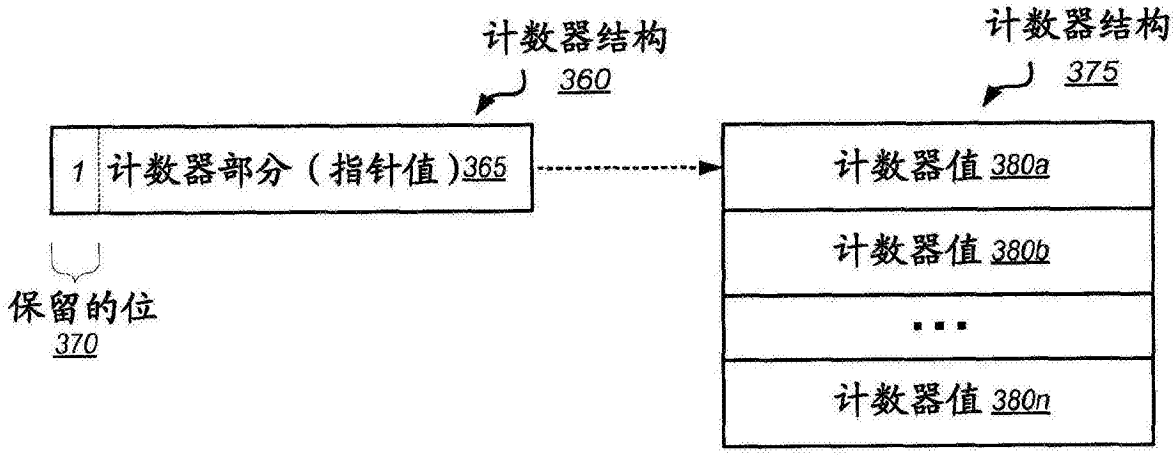

计数器结构
360
计数器结构
375
1
计数器部分（指针值）365
计数器值 380a
计数器值 380b
...
计数器值 380n
保留的位
370

图3E

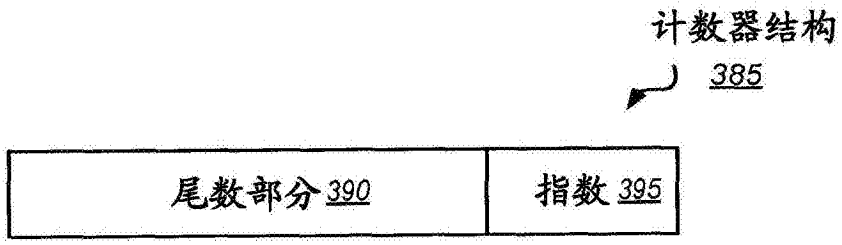

计数器结构
385
尾数部分 390
指数 395

图3F

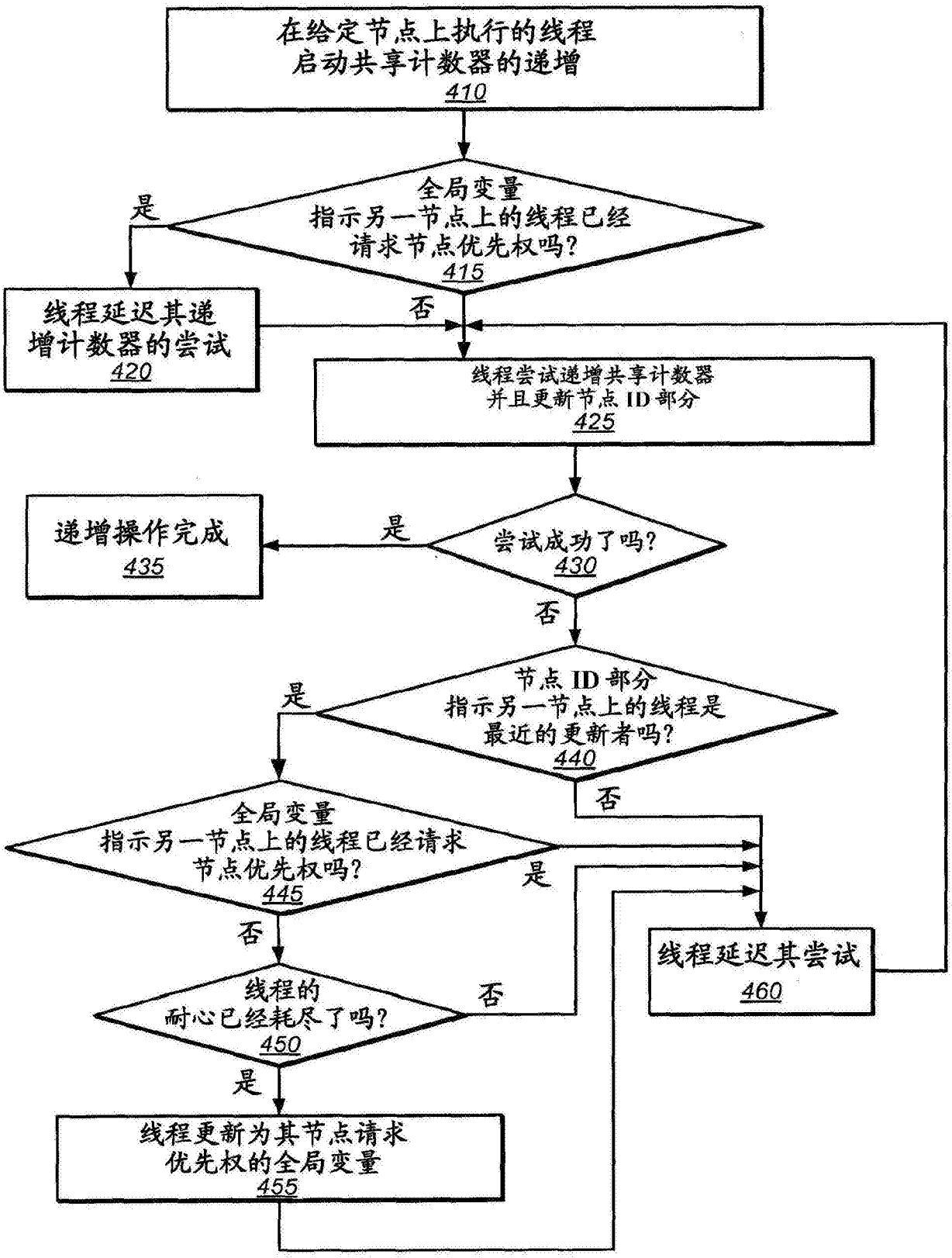

在给定节点上执行的线程
启动共享计数器的递增
410
全局变量
指示另一节点上的线程已经
请求节点优先权吗?
415
是
否
线程延迟其递
增计数器的尝试
420
线程尝试递增共享计数器
并且更新节点 ID 部分
425
尝试成功了吗?
430
是
递增操作完成
435
否
节点 ID 部分
指示另一节点上的线程是
最近的更新者吗?
440
是
否
全局变量
指示另一节点上的线程已经请求
节点优先权吗?
445
是
否
线程的
耐心已经耗尽了吗?
450
否
是
线程更新为其节点请求
优先权的全局变量
455
线程延迟其尝试
460

图4

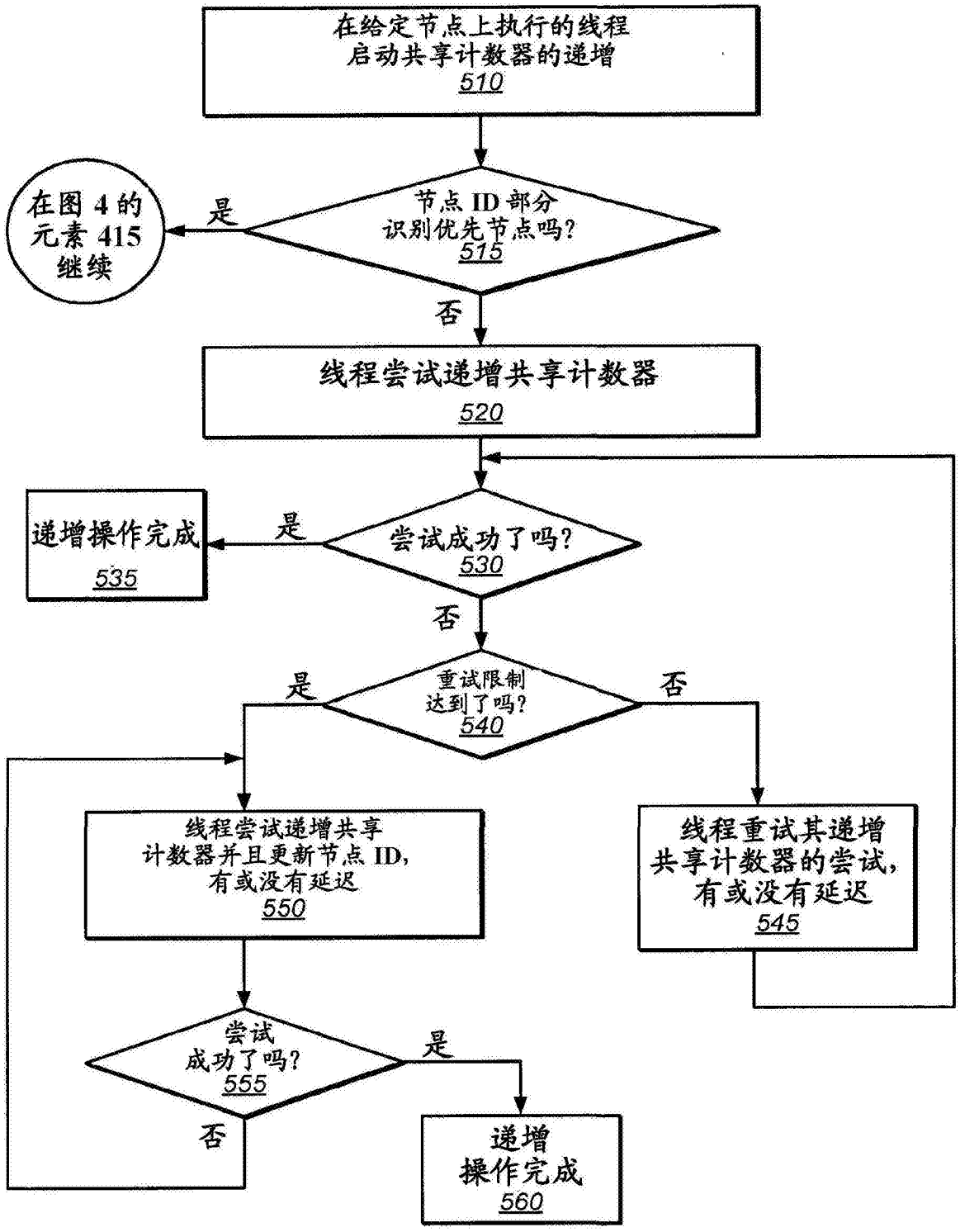

在给定节点上执行的线程
启动共享计数器的递增
510
节点 ID 部分
识别优先节点吗?
515
是
在图 4 的
元素 415
继续
否
线程尝试递增共享计数器
520
尝试成功了吗?
530
是
递增操作完成
535
否
重试限制
达到了吗?
540
是
否
线程尝试递增共享
计数器并且更新节点 ID,
有或没有延迟
550
线程重试其递增
共享计数器的尝试,
有或没有延迟
545
尝试
成功了吗?
555
是
否
递增
操作完成
560

图5

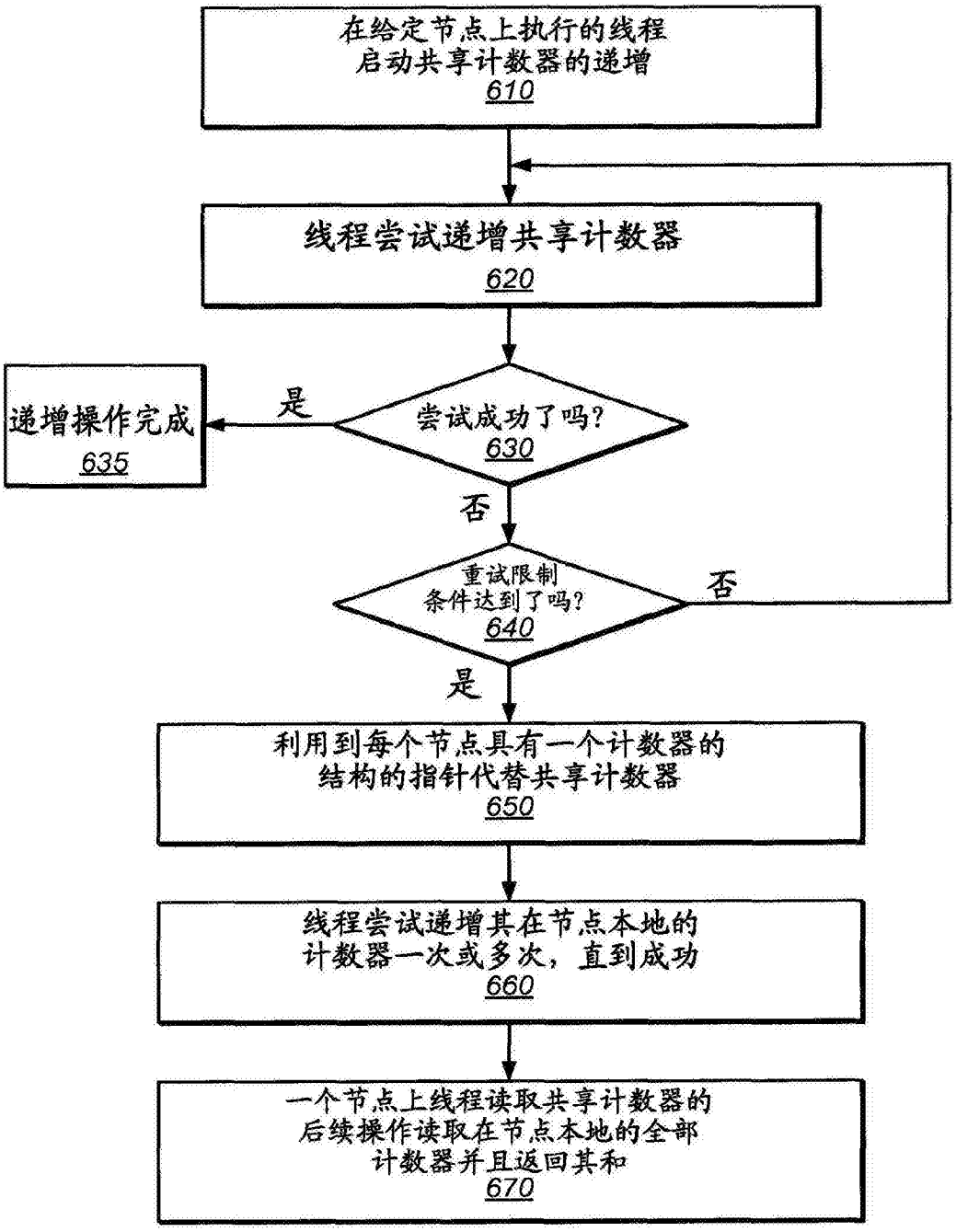

在给定节点上执行的线程
启动共享计数器的递增
610
线程尝试递增共享计数器
620
尝试成功了吗?
630
是
递增操作完成
635
否
重试限制
条件达到了吗?
640
否
是
利用到每个节点具有一个计数器的
结构的指针代替共享计数器
650
线程尝试递增其在节点本地的
计数器一次或多次，直到成功
660
一个节点上线程读取共享计数器的
后续操作读取在节点本地的全部
计数器并且返回其和
670

图6

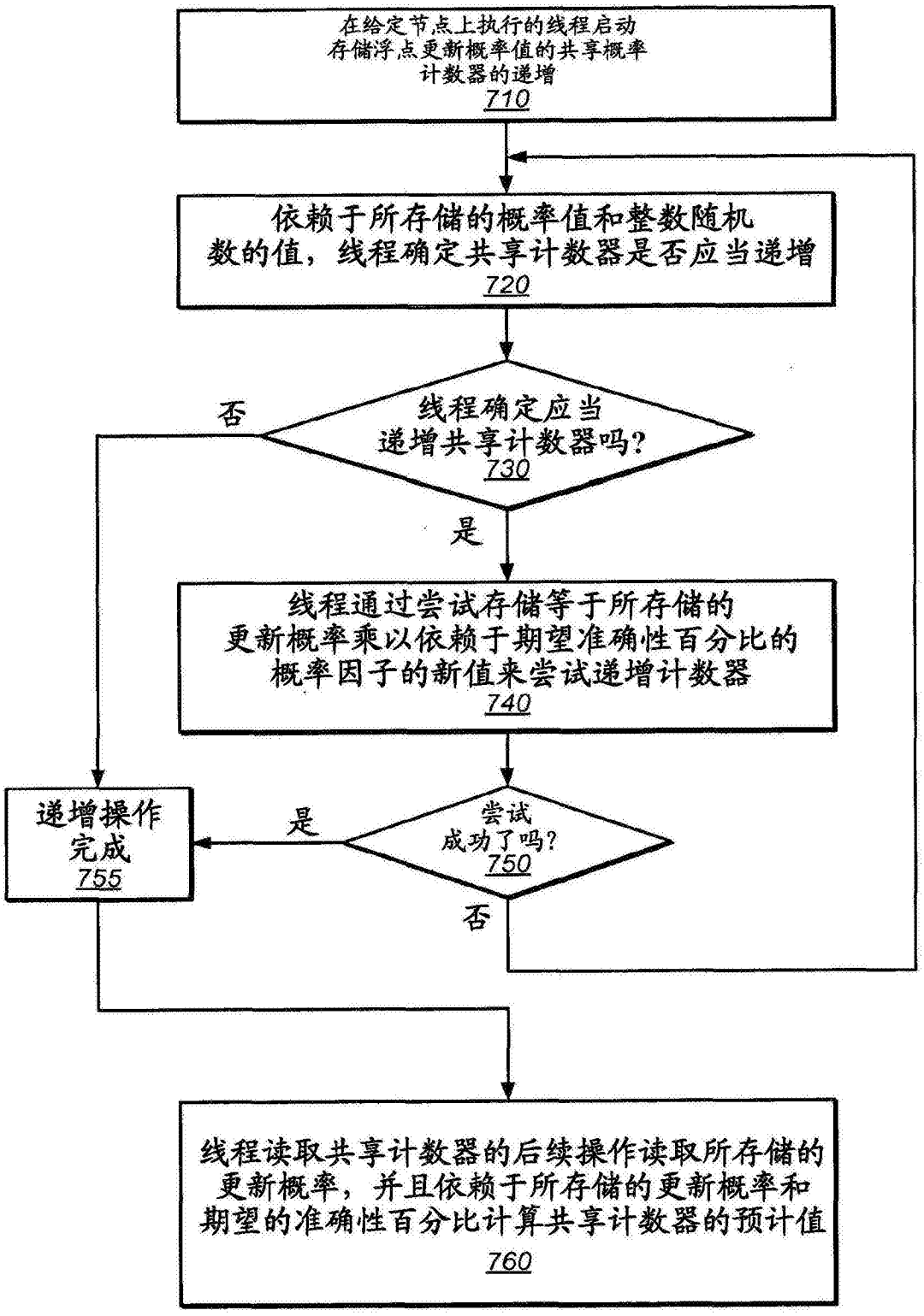

在给定节点上执行的线程启动
存储浮点更新概率值的共享概率
计数器的递增
710
依赖于所存储的概率值和整数随机
数的值，线程确定共享计数器是否应当递增
720
线程确定应当
递增共享计数器吗?
730
否
是
线程通过尝试存储等于所存储的
更新概率乘以依赖于期望准确性百分比的
概率因子的新值来尝试递增计数器
740
尝试
成功了吗?
750
是
否
递增操作
完成
755
线程读取共享计数器的后续操作读取所存储的
更新概率，并且依赖于所存储的更新概率和
期望的准确性百分比计算共享计数器的预计值
760

图7

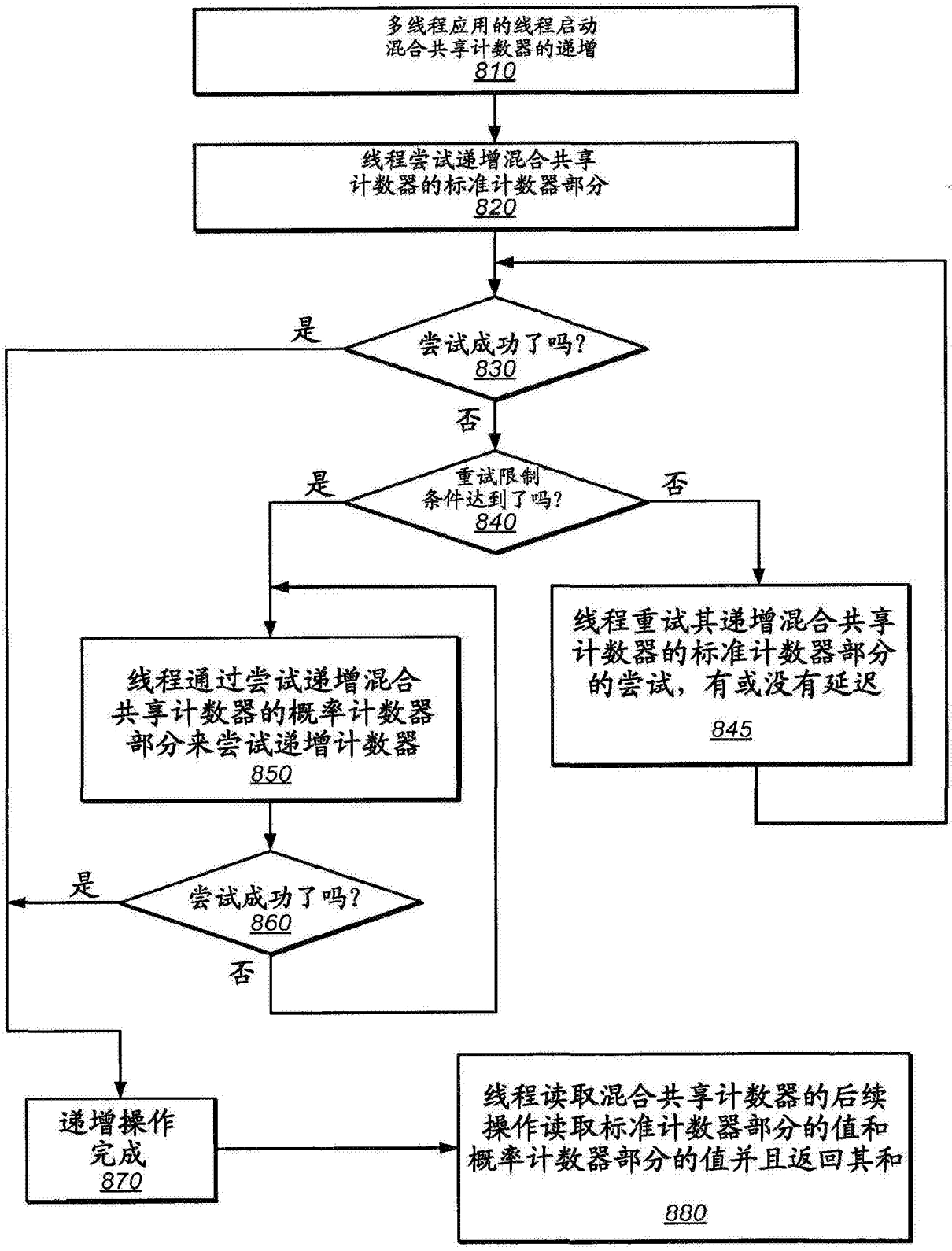

多线程应用的线程启动
混合共享计数器的递增
810
线程尝试递增混合共享
计数器的标准计数器部分
820
尝试成功了吗?
830
是
否
重试限制
条件达到了吗?
840
是
否
线程通过尝试递增混合
共享计数器的概率计数器
部分来尝试递增计数器
850
线程重试其递增混合共享
计数器的标准计数器部分
的尝试，有或没有延迟
845
尝试成功了吗?
860
是
否
递增操作
完成
870
线程读取混合共享计数器的后续
操作读取标准计数器部分的值和
概率计数器部分的值并且返回其和
880

图8

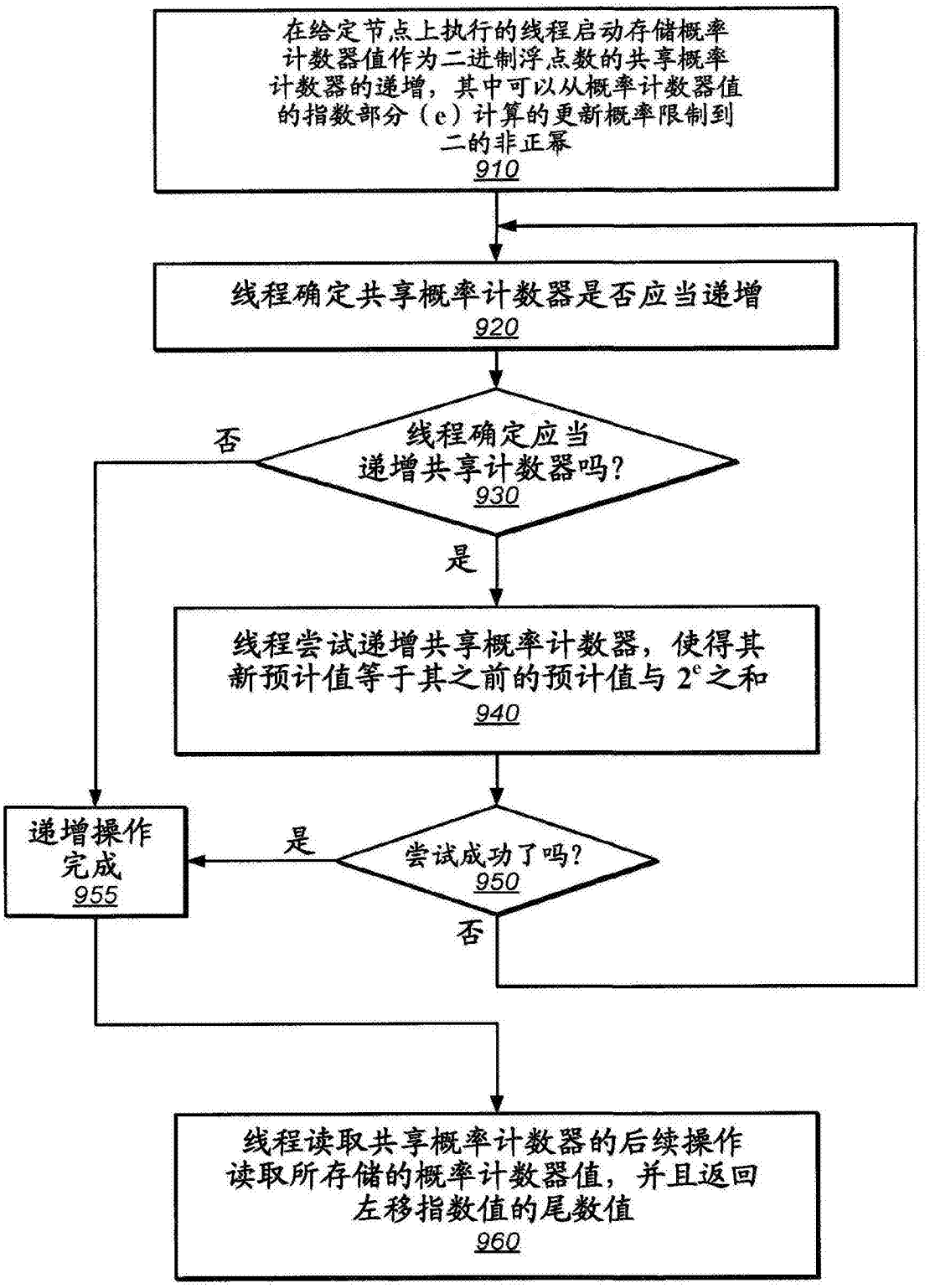

在给定节点上执行的线程启动存储概率计数器值作为二进制浮点数的共享概率计数器的递增，其中可以从概率计数器值的指数部分（e）计算的更新概率限制到二的非正幂
910
线程确定共享概率计数器是否应当递增
920
线程确定应当递增共享计数器吗?
930
否
是
线程尝试递增共享概率计数器，使得其新预计值等于其之前的预计值与 2^e 之和
940
尝试成功了吗?
950
是
否
递增操作完成
955
线程读取共享概率计数器的后续操作读取所存储的概率计数器值，并且返回左移指数值的尾数值
960

图9

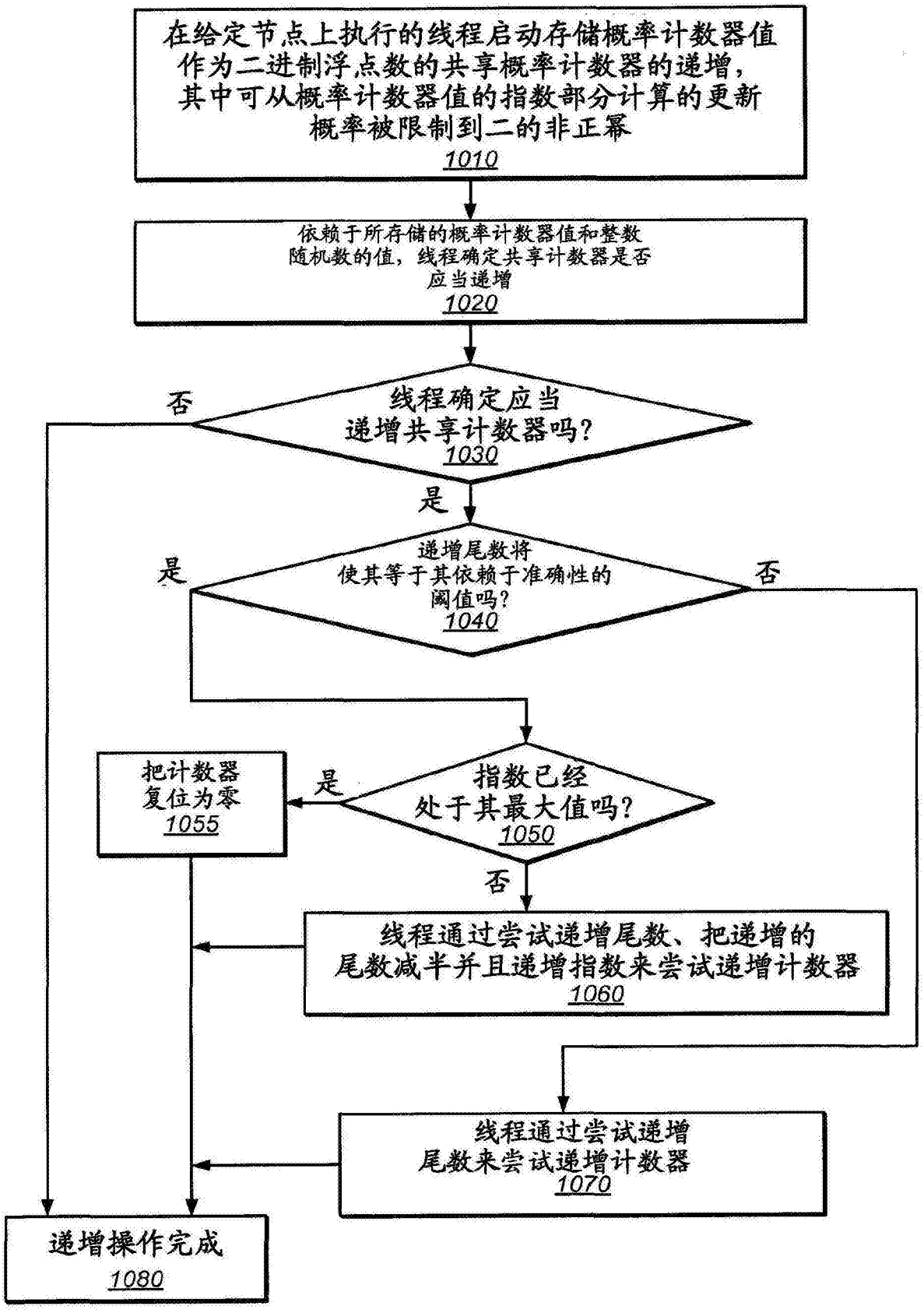

在给定节点上执行的线程启动存储概率计数器值作为二进制浮点数的共享概率计数器的递增，其中可从概率计数器值的指数部分计算的更新概率被限制到二的非正幂
1010
依赖于所存储的概率计数器值和整数随机数的值，线程确定共享计数器是否应当递增
1020
线程确定应当递增共享计数器吗？
1030
否
是
递增尾数将使其等于其依赖于准确性的阈值吗？
1040
是
否
指数已经处于其最大值吗？
1050
是
把计数器复位为零
1055
否
线程通过尝试递增尾数、把递增的尾数减半并且递增指数来尝试递增计数器
1060
线程通过尝试递增尾数来尝试递增计数器
1070
递增操作完成
1080

图10

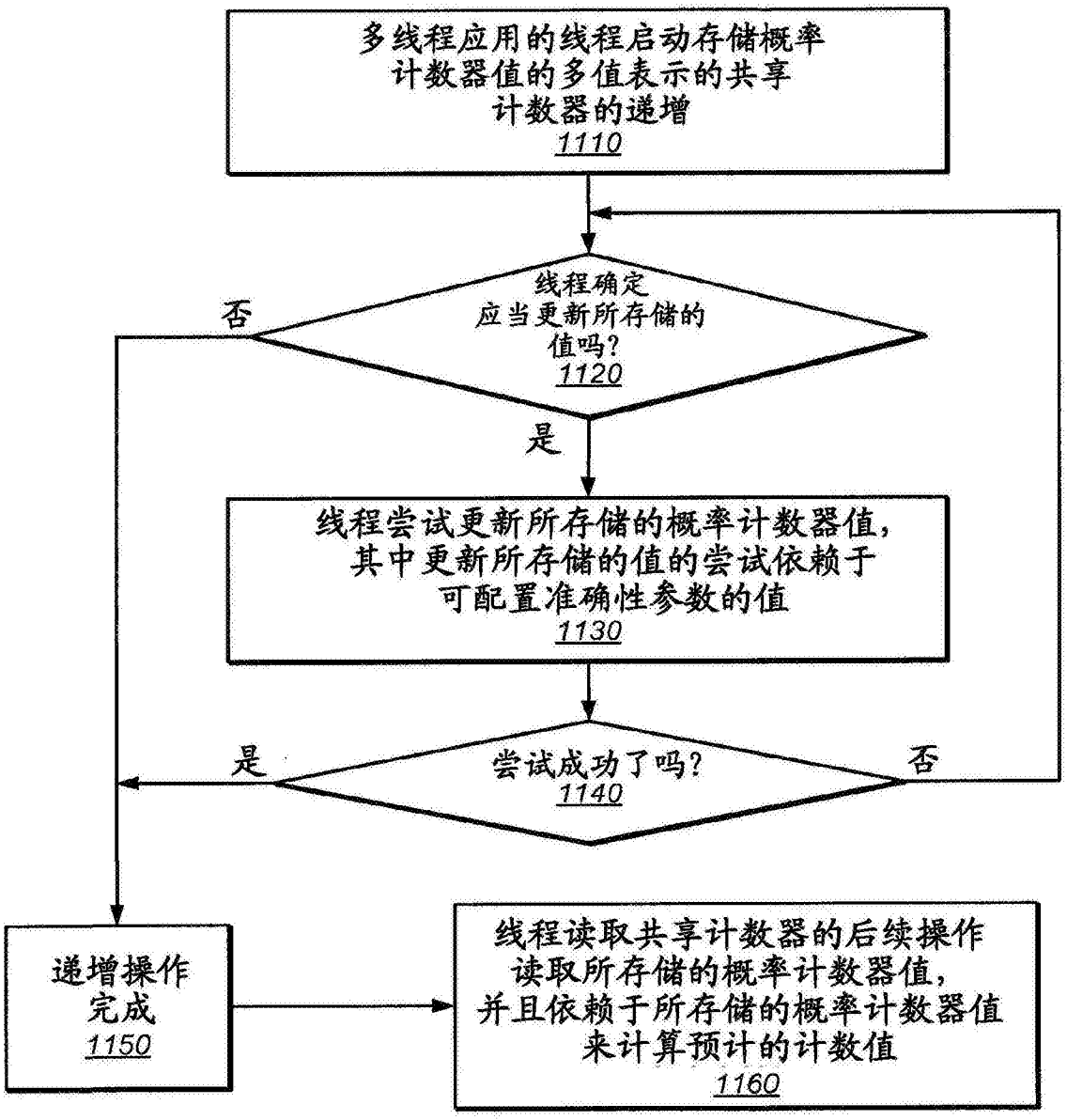

多线程应用的线程启动存储概率
计数器值的多值表示的共享
计数器的递增
1110
线程确定
应当更新所存储的
值吗?
1120
否
是
线程尝试更新所存储的概率计数器值,
其中更新所存储的值的尝试依赖于
可配置准确性参数的值
1130
尝试成功了吗?
1140
是
否
递增操作
完成
1150
线程读取共享计数器的后续操作
读取所存储的概率计数器值,
并且依赖于所存储的概率计数器值
来计算预计的计数值
1160

图11

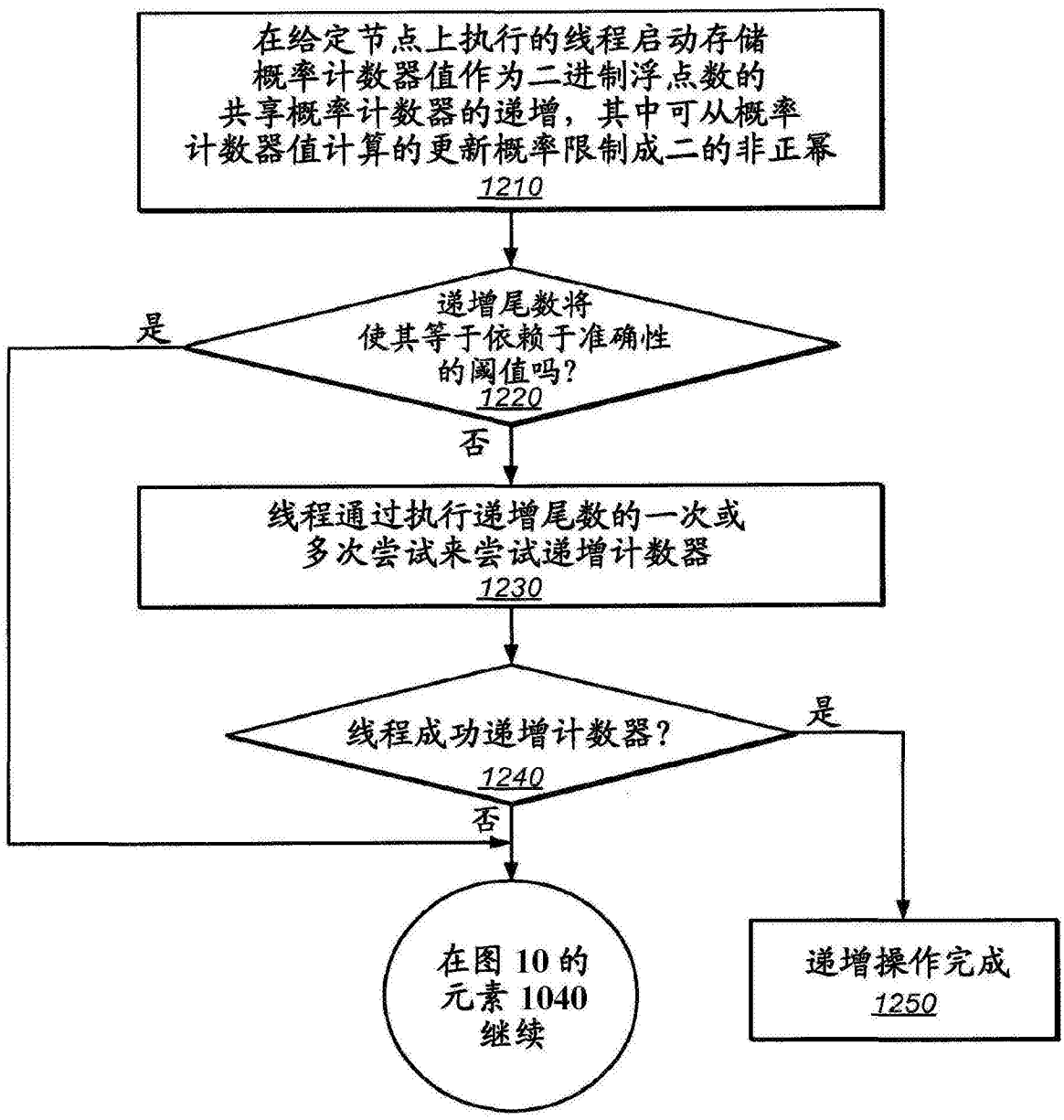

在给定节点上执行的线程启动存储
概率计数器值作为二进制浮点数的
共享概率计数器的递增，其中可从概率
计数器值计算的更新概率限制成二的非正幂
1210
递增尾数将
使其等于依赖于准确性
的阈值吗?
1220
是
否
线程通过执行递增尾数的一次或
多次尝试来尝试递增计数器
1230
线程成功递增计数器?
1240
是
否
在图 10 的
元素 1040
继续
递增操作完成
1250

图12

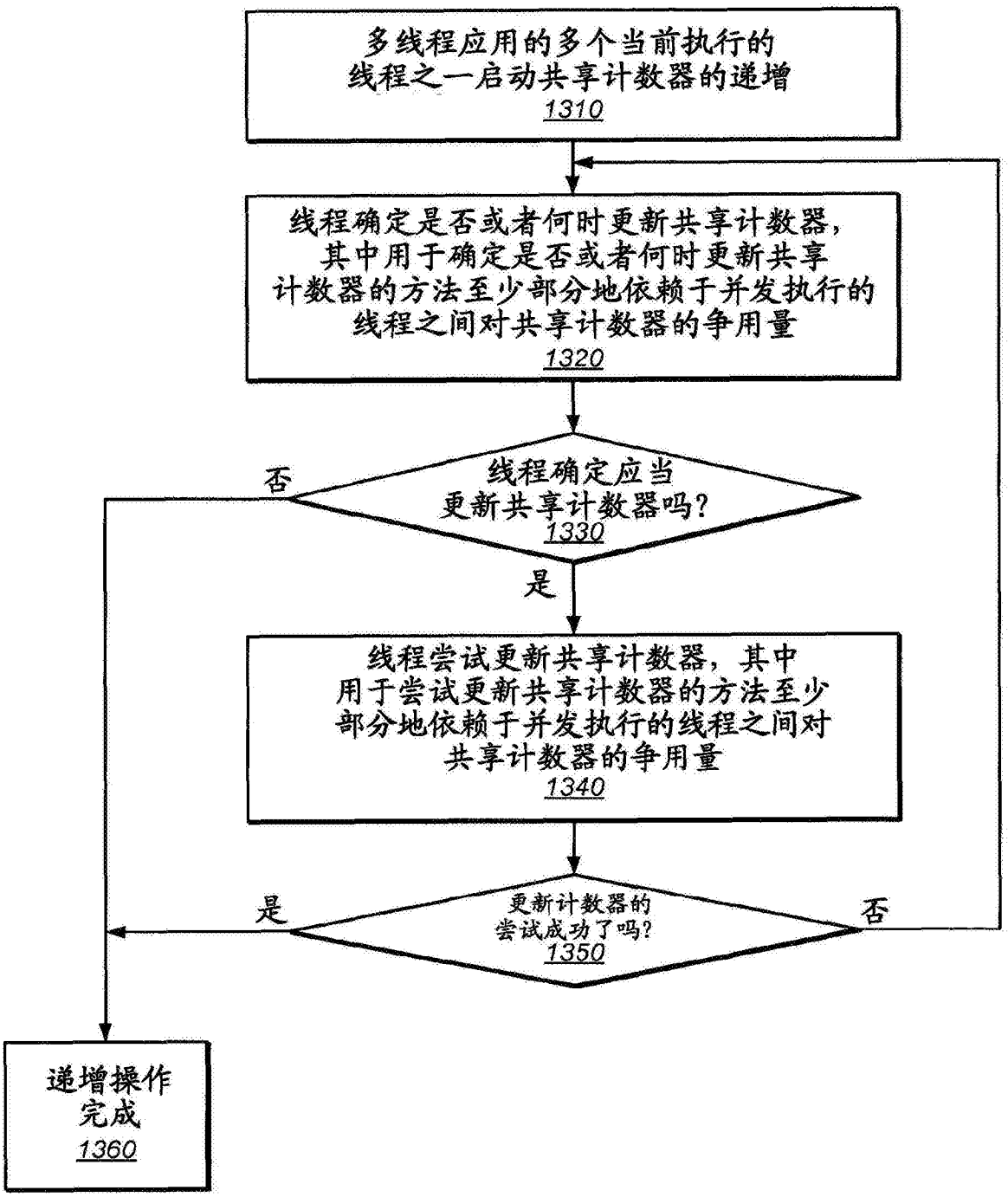

多线程应用的多个当前执行的
线程之一启动共享计数器的递增
1310
线程确定是否或者何时更新共享计数器，
其中用于确定是否或者何时更新共享
计数器的方法至少部分地依赖于并发执行的
线程之间对共享计数器的争用量
1320
线程确定应当
更新共享计数器吗？
1330
否
是
线程尝试更新共享计数器，其中
用于尝试更新共享计数器的方法至少
部分地依赖于并发执行的线程之间对
共享计数器的争用量
1340
更新计数器的
尝试成功了吗？
1350
是
否
递增操作
完成
1360

图13

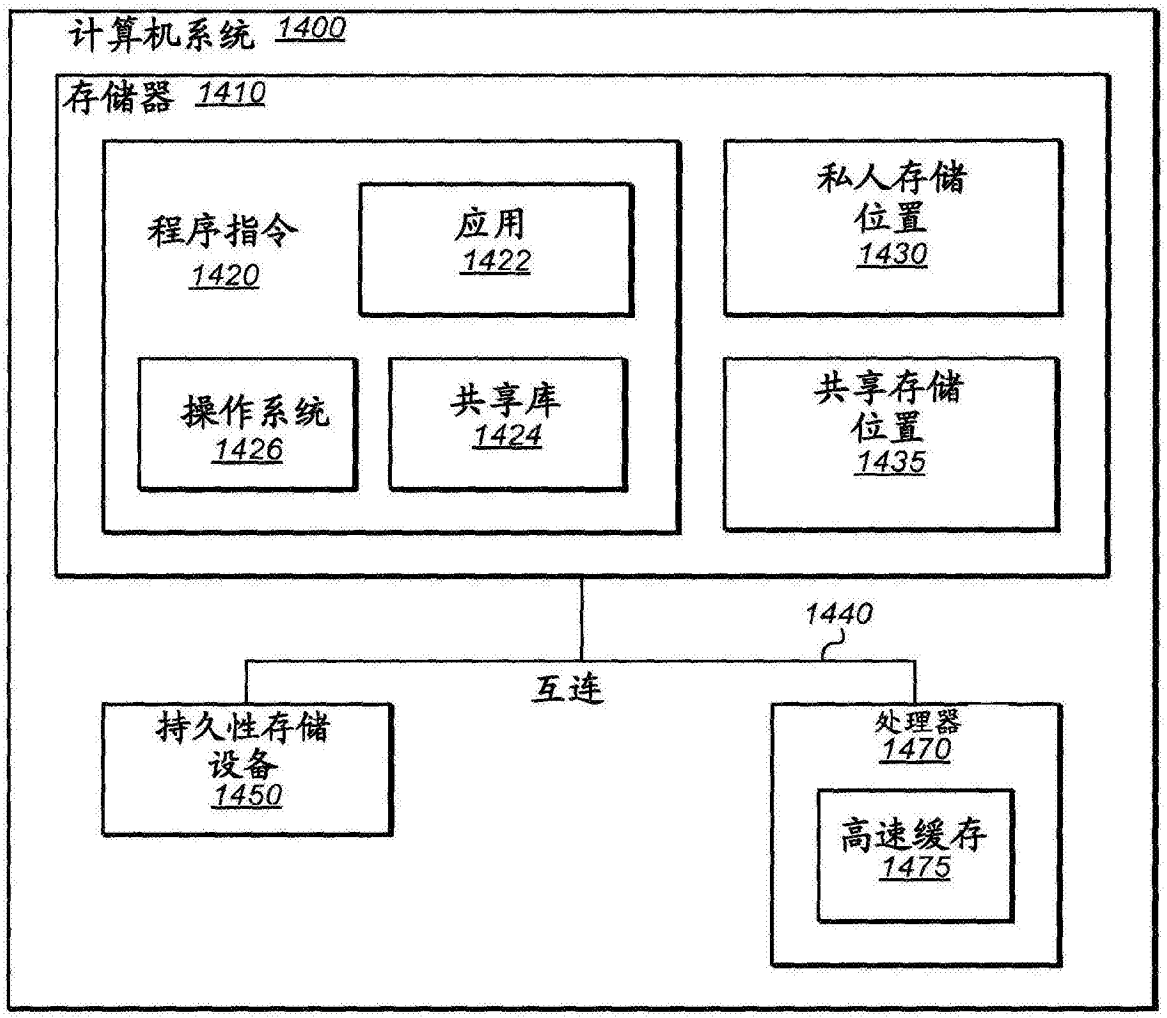

计算机系统 1400
存储器 1410
程序指令
1420
应用
1422
操作系统
1426
共享库
1424
私人存储
位置
1430
共享存储
位置
1435
1440
互连
持久性存储
设备
1450
处理器
1470
高速缓存
1475

图14