(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 101681292 B
(45)授权公告日 2012.10.10

(21)申请号 200880018391.2

(22)申请日 2008.05.30

(30)优先权数据
11/810,111 2007.06.04 US

(85)PCT申请进入国家阶段日
2009.12.01

(86)PCT申请的申请数据
PCT/US2008/065363 2008.05.30

(87)PCT申请的公布数据
WO2008/151046 EN 2008.12.11

(73)专利权人 微软公司
地址 美国华盛顿州

(72)发明人 J·J·达菲 J·J·格雷 Y·莱瓦诺尼

(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 顾嘉运 钱静芳

(51)Int.Cl.
G06F 12/00(2006.01)

(56)对比文件
US 2004148150 A1,2004.07.29, 全文.
US 2005283769 A1,2005.12.22, 全文.
US 6016399 A,2000.01.18, 全文.
US 6014741 A,2000.01.11, 全文.

审查员 王可

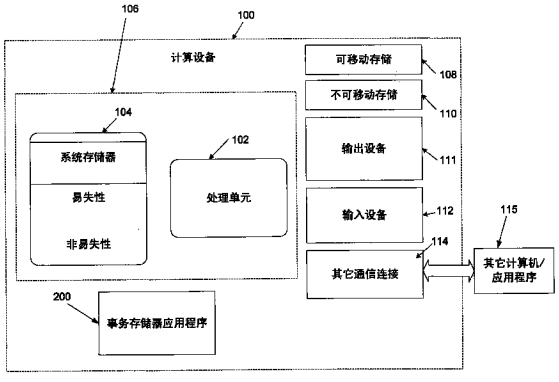
权利要求书 2 页 说明书 11 页 附图 25 页

(54)发明名称

使用事务来并行化顺序框架的方法

(57)摘要

公开了用于将顺序循环转换成并行循环以与事务存储器系统一起使用的各种技术和方法。可以将开放和/或封闭顺序循环转换成并行循环。例如,分析包含原始顺序循环的一部分代码以确定该原始顺序循环的固定迭代次数。将原始顺序循环转换成可生成数量等于该固定迭代次数的事务的并行循环。作为另一示例,可以将开放顺序循环转换成生成包含推测流水线的每一迭代的相应工作项的单独事务的并行循环。并行循环随后使用事务存储器系统来执行,其中各单独事务中的至少某一些在不同的线程上执行。



CN 101681292 B

1. 一种用于将封闭顺序循环转换成并行循环的方法，所述方法包括以下步骤：

提供事务存储器系统；

分析包含原始顺序循环的第一部分代码以确定所述原始顺序循环将要执行的固定迭代次数；

将包含所述原始顺序循环的所述第一部分代码转换成包含并行循环的第二部分代码，所述并行循环可操作来生成数量上等于所述固定迭代次数的多个事务，这些事务允许该并行循环的至少一部分并行执行；以及

使用所述事务存储器系统执行所述第二部分代码，其中所述多个事务中的至少某一些在不同的线程上执行。

2. 如权利要求 1 所述的方法，其特征在于，所述固定迭代次数是通过检索所述原始顺序循环与其相比以确定循环终止的常数值来确定的。

3. 如权利要求 1 所述的方法，其特征在于，所述事务中的每一个使用相应归纳变量计数器作为提交顺序号，所述提交顺序号由预定提交次序过程用来确保所述事务中的每一个都以正确次序提交。

4. 一种用于将封闭顺序循环转换成并行循环的方法，所述方法包括：

提供事务存储器系统；

将包含开放顺序循环的第一部分代码转换成包含并行循环的第二部分代码，所述并行循环可操作来生成包含推测流水线的每一迭代的相应工作项的单独事务；以及

使用所述事务存储器系统执行所述第二部分代码，其中所述单独事务中的至少某一些在不同的线程上执行。

5. 如权利要求 4 所述的方法，其特征在于，生成所述第二部分代码以使所述事务中的至少某一些并行执行。

6. 如权利要求 4 所述的方法，其特征在于，在不执行对所述开放顺序循环的编译器分析的情况下生成所述第二部分代码。

7. 如权利要求 4 所述的方法，其特征在于，通过以预定提交次序提交所述事务来保留原始的输入到输出映射。

8. 如权利要求 7 所述的方法，其特征在于，所述预定提交次序与所述开放顺序循环的执行次序相一致。

9. 一种用于执行从开放顺序循环生成的并行循环的方法，所述方法包括以下步骤：

生成估计在并行循环中执行的迭代次数的推测流水线，所述并行循环是从开放顺序循环中生成的；

取所述推测流水线的每一迭代并生成包含相应工作项的单独事务；

在不同的线程上执行所述单独事务中的至少某一些；

对每一相应工作项评估终止条件；以及

在所述相应工作项中的特定一个确定到达了终止所述并行循环的时间时，提交所述相应工作项中的所述特定一个的前导事务并丢弃其后续事务。

10. 如权利要求 9 所述的方法，其特征在于，在每一相应工作项执行时，检索当前迭代值。

11. 如权利要求 10 所述的方法，其特征在于，每一相应工作项的所述当前迭代值被用

作预定提交次序过程中的提交顺序号。

12. 如权利要求 10 所述的方法，其特征在于，所述当前迭代值是通过对可由每一相应工作项访问的值执行原子递增来检索的。

13. 如权利要求 9 所述的方法，其特征在于，实现与所述开放顺序循环的原始执行相一致的提交次序。

14. 如权利要求 9 所述的方法，其特征在于，所述开放顺序循环是“while”循环。

15. 如权利要求 9 所述的方法，其特征在于，所述开放顺序循环是“do while”循环。

16. 如权利要求 9 所述的方法，其特征在于，所述开放顺序循环是“for”循环。

17. 如权利要求 9 所述的方法，其特征在于，所述推测流水线的初始值至少部分地基于执行所述并行循环的计算机上的可用处理器的数量来计算。

18. 如权利要求 9 所述的方法，其特征在于，使用自适应统计量来调整所述推测流水线以供所述并行循环的之后执行。

使用事务来并行化顺序框架的方法

[0001] 背景

[0002] 软件事务存储器 (STM) 是类似于数据库事务的、用于在并发计算中控制对共享存储器的访问的并发控制机制。事务存储器的上下文中的事务是执行对共享存储器的一系列读取和写入的一段代码。STM 用作传统锁定机制的替换。程序员在代码块周围放置声明性注释（例如，原子的）以指示这些代码块所需要的安全特性，并且系统自动保证该块相对于其它受保护的代码区域原子地执行。软件事务存储器编程模型防止了基于锁的优先级倒置和死锁问题。

[0003] 虽然典型的 STM 系统具有许多优点，但它们仍然需要程序员仔细地避免非预期的存储器访问排序。例如，在典型的 STM 环境中提交事务（即，提交处理）的次序是不受约束的。各事务彼此竞争提交，这意味着事务 1 是在事务 2 之前还是之后提交通常是程序的动态调度的产物（并且通常也由程序专用逻辑来调度）。此外，如果两个事务冲突，诸如通过试图向同一段存储器写入等，则它们的提交次序可基于多个可能的争用管理策略中的一个来任意决定。在这两种情况下，不保证任何特定提交次序；因此确保程序员的程序按任一次序都正确地运作的负担就落在程序员身上。这使得并行编程非常困难。

[0004] 其中执行次序可能很重要并且其中并行性很有吸引力的一个场景是在并行地执行一循环的多次迭代时。以典型的“for...each”循环为例，如下所示：

[0005]

```
ForEach (string s in List<string>)
{
    S;
}
```

[0006] 在该循环的每一迭代期间，都将执行该循环主体中的语句 S。这一循环被编写来顺序地执行，其中该循环的第一次迭代在第二次迭代开始之前结束，以此类推。如果在没有处理可能的副作用或次序依赖性的额外预防措施的情况下并行地执行这一顺序循环，则可能发生意外结果。

[0007] 概述

[0008] 公开了用于对事务存储器系统中的事务应用排序的各种技术和方法。事务存储器系统具备允许为多个事务指定预定提交次序的特征。在运行时使用该预定提交次序来帮助确定提交事务存储器系统中的事务的次序。在一个实现中，预定提交次序可以或者是总体排序或者是部分排序。在总体排序的情况下，迫使事务以线性次序来提交。在部分排序的情况下，允许在多个可接受场景中的一个中提交事务。在一个实现中，提交仲裁器跟踪表示应被允许接下来提交的事务的下一个提交 (next-to-commit) 值，并且当特定事务准备好提交时，该事务在其提交序号匹配提交仲裁器的下一个提交值的情况下被允许提交。

[0009] 当在第一事务和第二事务之间发生冲突时调用争用管理过程。在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续。

[0010] 公开了用于将顺序循环转换成并行循环以与事务存储器系统一起使用的技术。提

供了一种基于事务存储器的系统。将包含原始顺序循环的第一部分代码转换成包含使用事务来保留原始的输入到输出映射的并行循环的第二部分代码。例如，可以通过取原始顺序循环的每一迭代并生成遵循预定提交次序过程的单独事务，并随后将这些事务分配到不同的线程以使它们并行执行，来将原始顺序循环转换成并行循环。万一在执行并行循环时从特定事务中检测到未经处理的异常，则提交该特定事务和任何前导事务所作出的状态修改并丢弃任何后续事务所作出的状态修改。否则，所有事务提交。

[0011] 在一个实现中，可以将开放和 / 或封闭顺序循环转换成并行循环。例如，分析包含原始顺序循环的一部分代码以确定该原始顺序循环的固定迭代次数。将原始顺序循环转换成可生成数量等于固定迭代次数的事务的并行循环。作为另一示例，可以将开放顺序循环转换成生成包含推测流水线的每 一迭代的相应工作项的单独事务的并行循环。将这些事务分配到不同的线程以允许并行循环的至少一部分并行执行。并行循环随后在使用预定提交排序的好处的事务存储器系统的保护之下执行。

[0012] 在一个实现中，提供了一种用于执行从开放顺序循环生成的并行循环的方法。生成估计在并行循环中执行的迭代次数的推测流水线，该并行循环是从开放顺序循环中生成的。系统取推测流水线的每一迭代并生成包含相应工作项的单独事务。随后将这些单独事务分配到不同的线程以使它们最终可以并行执行。对每一相应工作项评估终止条件。在相应工作项中的特定一个确定到达了终止并行循环的时间时，提交前导事务并丢弃后续事务。

[0013] 提供本概述以便以简化形式介绍将在以下详细描述中进一步描述的一些概念。本概述不旨在标识所要求保护的主题的关键特征或必要特征，也不旨在用于帮助确定所要求保护的主题的范围。

[0014] 附图简述

[0015] 图 1 是一个实现的计算机系统的图示。

[0016] 图 2 是在图 1 的计算机系统上操作的一个实现的事务存储器应用程序的图示。

[0017] 图 3 是图 1 的系统的一个实现的高级处理流程图。

[0018] 图 4 是图 1 的系统的一个实现的处理流程图，其示出在使用提交仲裁器来强制实施预定提交次序时所涉及的各阶段。

[0019] 图 5 是图 1 的系统的一个实现的处理流程图，其示出在使用提交仲裁器来强制实施对多个事务的总体排序时所涉及的各阶段。

[0020] 图 6 是图 1 的系统的一个实现的处理流程图，其示出在使用提交仲裁器来强制实施对多个事务的部分排序时所涉及的各阶段。

[0021] 图 7 是图 1 的系统的一个实现的处理流程图，其示出在提供使用预定提交次序信息来管理冲突的争用管理过程时所涉及的各阶段。

[0022] 图 8 是图 1 的系统的一个实现的处理流程图，其示出在提供使用预定提交次序信息来管理关于嵌套事务的冲突的争用管理过程时所涉及的各阶 段。

[0023] 图 9 是示出顶层先辈具有公共先辈的示例性先辈树的逻辑图。

[0024] 图 10 是示出顶层先辈不具有公共先辈的示例性先辈树的逻辑图。

[0025] 图 11 是图 1 的系统的一个实现的处理流程图，其示出在通过在事务存储器系统中使用提交仲裁器来减少浪费的工作量时所涉及的各阶段。

[0026] 图 12 是图 1 的系统的一个实现的处理流程图，其示出在争用管理过程中分析整个先辈链以确定适当的冲突解决方案时所涉及的各阶段。

[0027] 图 13 是在图 1 的计算机系统上操作的一个实现的事务存储器应用程序的图示。

[0028] 图 14 是图 1 的系统的一个实现的处理流程图，其示出在将原始顺序循环转换成并行循环时所涉及的各阶段。

[0029] 图 15 是图 1 的一个实现的处理流程图，其示出在使用预定提交次序过程来确保并行循环中的事务以正确次序提交时所涉及的各阶段。

[0030] 图 16 是图 1 的系统的一个实现的处理流程图，其示出在使用提交仲裁器来检测并处理执行并行循环时所发生的冲突时所涉及的各阶段。

[0031] 图 17 是图 1 的系统的一个实现的处理流程图，其示出在检测并处理执行并行循环时所发生的未经处理的异常时所涉及的各阶段。

[0032] 图 18A-18B 示出从原始顺序循环到并行循环的示例性转换的假想源代码。

[0033] 图 19 是图 1 的系统的一个实现的处理流程图，其示出在将封闭顺序循环转换成并行循环时所涉及的各阶段。

[0034] 图 20 是图 1 的系统的一个实现的处理流程图，其示出在使用推测流水线将开放顺序循环转换成并行循环时所涉及的各阶段。

[0035] 图 21 是图 1 的系统的一个实现的处理流程图，其示出在执行从开放顺序循环生成的并行循环时所涉及的各阶段。

[0036] 图 22 是图 1 的系统的一个实现的处理流程图，其示出在确保从开放顺序循环生成的并行循环中的每一工作项以正确次序提交时所涉及的各阶段。

[0037] 图 23 是图 1 的系统的一个实现的处理流程图，其示出在计算推测流水 线以确定要在并行循环中包括多少次迭代时所涉及的各阶段。

[0038] 图 24A-24B 示出从原始开放顺序循环到并行循环的示例性转换的假想源代码。

[0039] 详细描述

[0040] 为促进对本发明的原理的理解，现将对附图中所示的各实施例加以参考，同时也将用具体语言描述它们。不过，需要理解的是，并无意由此作出范围上的限制。对所述实施例的任何改变和进一步更改，以及在此所述的原理的进一步应用都可以预期将是本领域技术人员通常能想到的。

[0041] 本系统可以在一般上下文中被描述为事务存储器系统，但是本系统还用于除此之外的其它目的。在一个实现中，此处所描述的一个或多个技术可被实现为诸如微软 ®.NET 框架等框架程序内的、或来自为开发者提供开发软件应用程序的平台任何其它类型的程序或服务的特征。在另一实现中，此处所描述的一个或多个技术被实现为涉及开发在并发环境中执行的应用程序的其它应用程序的特征。

[0042] 在一个实现中，在事务存储器系统中提供允许为多个事务指定预定提交次序的特征。该预定提交次序用于帮助确定提交事务的次序。在一个实现中，当在第一事务和第二事务之间发生冲突时调用争用管理过程。然后在争用管理过程中使用该预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续。

[0043] 在另一实现中，在事务存储器系统中提供将原始顺序循环转换成并行循环的特征。原始顺序循环以确保原始的输入到输出映射得到保留的方式来转换成并行循环。此处

使用的术语“原始的输入到输出映射得到保留”意味着在执行并行化循环之后程序的状态是相同的，如同改为运行了顺序循环一样。在一个实现中，通过将原始顺序循环的每一次迭代置于事务中并随后使用此处描述的预定提交次序过程来确保以正确次序提交事务，来在并行循环中保留原始的输入到输出映射。

[0044] 尽管此处讨论的多个示例是在软件事务存储器系统的上下文中描述的，但可以理解，在其它实现中，此处所描述的特征 / 技术中的一些、全部、 或附加特征和 / 或技术可以用硬件事务存储器系统来与软件事务存储器系统分开或相结合地实现。

[0045] 如图 1 所示，用于实现本系统的一个或多个部分的示例性计算机系统包括诸如计算设备 100 等计算设备。在其最基本的配置中，计算设备 100 通常包括至少一个处理单元 102 和存储器 104。取决于计算设备的确切配置和类型，存储器 104 可以是易失性的（如 RAM）、非易失性的（如 ROM、闪存等）或是两者的某种组合。该最基本配置在图 1 中由虚线 106 来示出。

[0046] 另外，设备 100 还可具有附加特征 / 功能。例如，设备 100 还可包含附加存储（可移动和 / 或不可移动），包括但不限于磁盘、光盘或磁带。这样的附加存储在图 1 中由可移动存储 108 和不可移动存储 110 示出。计算机存储介质包括以用于存储诸如计算机可读指令、数据结构、程序模块或其它数据等信息的任何方法或技术来实现的易失性和非易失性、可移动和不可移动介质。存储器 104、可移动存储 108 和不可移动存储 110 都是计算机存储介质的示例。计算机存储介质包括但不限于，RAM、ROM、EEPROM、闪存或其它存储器技术、CD-ROM、数字多功能盘 (DVD) 或其它光存储、磁带盒、磁带、磁盘存储或其它磁存储设备、或者可用于存储所需信息并且可由设备 100 访问的任何其它介质。任何这样的计算机存储介质都可以是设备 100 的一部分。

[0047] 计算设备 100 包括允许计算设备 100 与其它计算机 / 应用程序 114 进行通信的一个或多个通信连接 115。设备 100 还可以具有诸如键盘、鼠标、笔、语音输入设备、触摸输入设备等输入设备 112。还可以包括诸如显示器、扬声器、打印机等输出设备 111。这些设备在本领域中公知且无需在此处详细讨论。在一个实现中，计算设备 100 包括事务存储器应用程序 200。事务存储器应用程序 200 将在图 2 中更详细地描述。

[0048] 现在转向图 2 并继续参考图 1，示出了在计算设备 100 上操作的事务存储器应用程序 200。事务存储器应用程序 200 是驻留在计算设备 100 上的应用程序中的一个。然而，可以理解，事务存储器应用程序 200 可另选地或另外地被具体化为一个或多个计算机上的计算机可执行指令和 / 或与图 1 所示的不同的变型。另选地或另外地，事务存储器应用程序 200 的一个或 多个部分可以是系统存储器 104 的一部分、可以在其它计算机和 / 或应用程序 115 上、或可以是计算机软件领域的技术人员能想到的其它此类变型。

[0049] 事务存储器应用程序 200 包括负责执行在此描述的技术中的一些或全部的程序逻辑 204。程序逻辑 204 包括用于提供事务存储器 (STM) 系统的逻辑 206 ；用于提供允许为 STM 系统中的多个事务静态或动态地指定预定提交次序的提交仲裁器的逻辑 208 ；用于允许提交仲裁器在运行时使用预定提交次序来帮助确定提交事务存储器系统中的多个事务的次序的逻辑 210 ；用于提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程的逻辑 212 ；用于在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续（例如，取决于同一事务组中的两个事务中的哪个事务具有

较低提交序号）的逻辑 214；用于允许提交仲裁器可操作来使用预定提交排序来跟踪一个或多个排序值（例如，在总体排序中，表示多个事务中应被允许提交的下一事务的下一个提交字段）以及用于将该一个或多个排序值与给定事务的特定提交序号进行比较以查看该给定事务的提交是否是适当的（给定应强制实施的排序）的逻辑 216；以及用于操作该应用程序的其它逻辑 220。在一个实现中，程序逻辑 204 可操作来通过编程，如使用对程序逻辑 204 中的一个过程的单一调用而从另一程序调用。

[0050] 现在转向图 3-10 并继续参考图 1-2，更详细地描述了用于实现事务存储器应用程序 200 的一个或多个实现的各阶段。图 3 是事务存储器应用程序 200 的高级处理流程图。在一种形式中，图 3 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 240 处开始，在那里提供事务存储器系统（例如，软件事务存储器系统）（阶段 242）。提供允许为多个事务指定（例如，动态或静态地分配）预定提交次序（例如，总体排序或部分排序）的特征（阶段 244）。此处所使用的术语“预定提交次序”旨在包括在事务开始运行之前的任何时刻所确定的特定相关事务组应被提交的特定次序。此处所使用的术语事务“组”包括由同一个提交仲裁器管理的特定的一组（例如，多个）事务，以及这些事务的嵌套子事务。

[0051] 在运行时使用预定提交次序来帮助确定提交事务存储器系统中的多个 事务的次序（阶段 246）。使用预定提交次序来帮助解决在多个事务中的两个或多个之间发生的冲突（阶段 248）。该过程在结束点 250 处结束。

[0052] 图 4 示出了在使用提交仲裁器来强制实施预定提交次序时所涉及的各阶段的一个实现。在一种形式中，图 4 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 270 处开始，在那里为事务存储器系统提供一个或多个提交仲裁器，该提交仲裁器可操作来允许为多个事务指定预定提交次序（阶段 272）。此处所使用的术语“提交仲裁器”旨在包括负责管理应当相对于彼此来排序的一个或多个事务组的任何类型的程序、特征或进程。在一个实现中，在任何给定时刻可能存在在程序中活动的一个或多个提交仲裁器。例如，可创建如所需要的那样多的提交仲裁器以管理不同的事务组。提交仲裁器跟踪并更新用于确定事务相对于彼此的正确排序的一个或多个排序值（阶段 274）。在总体排序的情况下，可使用下一个提交字段来表示多个事务中应被下一个提交的下一事务（阶段 274）。在部分排序的情况下，使用排序值来跟踪不同的可能次序的有向图。在适当时，提交仲裁器使用预定提交次序来为多个事务中的每一个提供提交序号（阶段 276）。

[0053] 当多个事务中的一特定事务准备提交时，如果该特定事务的提交序号在与一个或多个排序值进行比较时揭示该提交是适当的，则提交仲裁器允许该事务提交（阶段 278）。在总体排序的情况下，该情况在下一个提交字段与该特定事务的提交序号具有相同的值时发生。在这种情况下，提交仲裁器允许该事务提交并且然后在提交成功的情况下将下一个提交字段递增至序列中的下一个数字（例如，下一个较高的数字）（阶段 278）。当该多个事务中的特定事务准备提交时，如果该特定事务的提交序号在与排序值进行比较时揭示该提交不是适当的，则将该特定事务置于挂起模式中直到其在前导事务提交后的稍后时刻被唤醒（阶段 280）。在总体排序的情况下，当下一个提交字段和该特定事务的序号不具有相同的值时进入该挂起模式。

[0054] 在一个实现中，系统可在一事务的直接前导者被提交后唤醒该事务，在这种情况

下该事务可尝试立即提交。或者，系统可选择在某一非直接前 导者被提交后唤醒事务，即使该事务的直接前导者可能尚未提交。在被唤醒之后，系统检查以查看对于该事务而言真正提交是否适当。如果是，则提交该事务。该过程在结束点 282 处结束。

[0055] 图 5 示出了在使用提交仲裁器来强制实施对多个事务的总体排序时所涉及的各阶段的一个实现。在一种形式中，图 5 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 290 处开始，在那里提供可操作来允许为多个事务指定预定总体排序的一个或多个提交仲裁器（例如，指定应提交多个事务的确切次序的提交仲裁器）（阶段 292）。当多个事务中的一特定事务到达其提交点时，为了强制实施提交次序，将该特定事务的提交次序与提交仲裁器的下一个提交字段进行比较（阶段 296）。在一个实现中，如果系统确定总体排序的强制实施不是必要的（例如，诸如因为肯定不存在冲突），则可在适当时违反总体排序要求（阶段 294），然后该过程在结束点 302 处结束。

[0056] 如果要强制实施提交排序，并且如果该特定事务的提交次序具有与提交仲裁器的下一个提交字段相同的值（判定点 296），则提交该特定事务，并且如果提交成功，则递增该下一个提交字段并唤醒下一后继者（如果有的话）（阶段 298）。如果该特定事务的提交次序不具有与提交仲裁器的下一个提交字段相同的值（判定点 296），则将该特定事务置于挂起 / 休眠模式中直到其在前导事务提交后的稍后时刻被唤醒（阶段 300）。在一个实现中，在稍后时刻，如果与前导者发生冲突，则可要求该特定事务中止并回退以使得前导者可向前进。否则，如果未发生这样的冲突，则一旦满足此处所描述的提交次序要求，该特定事务就应能够提交。该过程在结束点 302 处结束。

[0057] 图 6 示出了在使用提交仲裁器来强制实施对多个事务的部分排序时所涉及的各阶段的一个实现。在一种形式中，图 6 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 310 处开始，在那里提供可操作来允许为多个事务指定预定部分排序的一个或多个提交仲裁器（例如，指定应提交多个事务的多个可接受次序（例如，以有向图的形式）的提交仲裁器）（阶段 312）。当该多个事务中的一特定事务到达其提交点时，为了强制实施提交次序，参考前导事务的状态（例如，一个或多个排序值）来寻找特定提交事务（例如，由提交仲裁器来跟踪的）（阶段 314）。如果该特定事务的所有前导者都已提交（判定点 316），则提交该特定事务（阶段 318）。如果提交成功，则在适当时更新由提交仲裁器跟踪的一个或多个值，并且唤醒所有可能的下一后继者（如果有的话）（阶段 318）。

[0058] 如果该特定事务的所有前导者尚未提交（判定点 316），则将该特定事务置于挂起 / 休眠模式中直到其在前导事务提交后的稍后时刻被唤醒（阶段 320）。该过程在结束点 322 处结束。

[0059] 图 7 示出了在提供使用预定提交次序信息来管理冲突的争用管理过程时所涉及的各阶段的一个实现。在一种形式中，图 7 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 340 处开始，在那里提供支持用于一个或多个事务组的预定提交次序的事务存储器系统（阶段 342）。提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程（阶段 344）。在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续（阶段 346）。如果第一事务和第二事务不是同一事务组的一部分（判定点 348），则在这两个事务之间不强制实施预定提交次序（因为不存在冲突）（阶段 350）。在这种情况下，因为这两个事务不在同一事务组中，所以不使用排序因素

来帮助解决冲突（阶段 350）。

[0060] 如果第一事务和第二事务是同一事务组的一部分（判定点 348），则系统将第一事务的第一序号与第二事务的第二序号进行比较（阶段 352）。允许具有较低序号的事务继续（或用另一合适的优先级排序）（阶段 354）。该过程在结束点 356 处结束。

[0061] 图 8 示出了在提供使用预定提交次序信息来管理关于嵌套事务的冲突的争用管理过程时所涉及的各阶段的一个实现。在一种形式中，图 8 的过程至少部分地在计算设备 100 的操作逻辑中实现。在一个实现中，在提交每一个事务之前对于该特定事务考虑整个先辈链，以便强制实施存在于该链中的任何排序。该过程在起始点 370 处开始，在那里提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程（阶段 372）。在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续（阶段 374）。如果第一和第二事务不是同一事务组的一部分（判定点 376），则在这两个事务之间不强制实施预定提交次序（因为不存在冲突）（阶段 378）并且该过程在结束点 388 处结束。如果第一和第二事务是同一事务组的一部分（判定点 376），则系统检查以查看是否涉及嵌套事务（判定点 380）。

[0062] 如果不涉及嵌套事务（判定点 380），则将第一事务的序号（或其它排序指示符）与第二事务的序号（或其它排序指示符）进行比较（阶段 384）。允许具有较低序号的事务继续（或者通过使用其它合适的排序准则被确定为按次序的下一个的事务）（阶段 386）。

[0063] 如果涉及嵌套事务（判定点 380），则将第一事务的顶层先辈的序号（或其它排序指示符）与第二事务的顶层先辈的序号（或其它排序指示符）进行比较（阶段 382）。此处所使用的术语“顶层先辈”在涉及公共先辈的情况下旨在包括公共先辈的直接子事务，而在不涉及公共先辈的情况下旨在包括每一个事务的顶层先辈。涉及公共和非公共先辈的这些情况在图 9 和 10 中更详细地示出。允许具有较低序号的事务继续（例如，与具有较低序号的先辈相关的事务或其它合适的准则）（阶段 386）。该过程在结束点 388 处结束。

[0064] 图 9 是示出顶层先辈具有公共先辈的示例性先辈树的逻辑图。在所示示例中，事务 A 是 D 和 E 的公共先辈。当在 D 和 E 之间发生冲突时，分析事务 B 和 C（公共先辈 A 的直接子事务）的序号以确定哪个事务，即，是 D 还是 E 应被允许继续（图 8 中的阶段 382）。

[0065] 图 10 是示出顶层先辈不具有公共先辈的示例性先辈树的逻辑图。在所示示例中，事务 A 是事务 C 的先辈。事务 D 是事务 F 的先辈。当在事务 C 和 F 之间发生冲突时，则将事务 A 和 D（各自的顶层先辈）的序号进行比较以确定哪个事务，即，是 C 还是 F 应被允许继续（图 8 中的阶段 382）。

[0066] 图 11 示出了在事务存储器系统中通过使用提交仲裁器来减少浪费的工作量时所涉及的各阶段的一个实现。在一种形式中，图 11 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 400 处开始，在 那里为事务存储器系统提供一个或多个提交仲裁器，该提交仲裁器可操作来允许为多个事务指定预定提交次序（阶段 402）。提交仲裁器可操作来将事务置于休眠 / 挂起模式中以便阻塞该事务在前导事务仍在执行时重新执行（例如，通过分析预定提交次序来确定正确次序）（阶段 404）。提交仲裁器还可操作来一旦前导事务完成则唤醒被置于挂起模式的事务（例如，通过再次分析预定提交次序来确定正确次序）（阶段 406）。通过提供这些阻塞和唤醒机制，提交仲裁器通过阻止将在稍后不得不撤销的操作的执行而帮助减少浪费的工作量（阶段 408）。该过程在结束点 410 处结束。

[0067] 图 12 示出了在争用管理过程中分析整个先辈链以确定适当的冲突解决方案时所涉及的各阶段的一个实现。在一种形式中，图 12 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 430 处开始，在那里提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程（阶段 432）。在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续（阶段 434）。分析预定提交次序的整个先辈链以帮助确定适当的冲突管理（阶段 436）。例如，如果存在四个事务，两个父事务和两个子事务，其中 B 嵌套在 A 中而 D 嵌套在 C 中。假设在 A 和 C 之间存在 A 应在 C 之前提交的排序关系。如果 B 和 D 冲突，则争用管理过程应偏向 B，因为在给定 A 必须在 C 之前提交的情况下偏向 D 是无用的（阶段 436）。该过程在结束点 438 处结束。

[0068] 现在转向图 13 并继续参考图 1，示出了在计算设备 100 上操作的具有并行循环支持的事务存储器应用程序 500。在一个实现中，具有并行循环支持的事务存储器应用程序 500 是驻留在计算设备 100 上的应用程序之一。然而，可以理解，具有并行循环支持的事务存储器应用程序 500 可另选地或另外地被具体化为一个或多个计算机上的计算机可执行指令和 / 或与图 1 所示的不同的变型。另选地或另外地，具有并行循环支持的事务存储器应用程序 500 的一个或多个部分可以是系统存储器 104 的一部分、可以在其它计算机和 / 或应用程序 115 上、或可以是计算机软件领域的技术人员能想到的其它此类变型。

[0069] 具有并行循环支持的事务存储器应用程序 500 包括负责执行在此描述 的技术中的一些或全部的程序逻辑 504。程序逻辑 504 包括用于提供事务存储器系统的逻辑 506 ；用于将包含原始顺序循环的第一部分代码转换成包含使用事务来保留原始的输入到输出映射并提高安全性的并行循环的第二部分代码的逻辑 508 ；用于将原始顺序循环的迭代中的一个或多个置于并行循环中的各事务中的单独事务中的逻辑 510 ；用于通过使用与原始顺序循环的执行次序相一致的预定提交次序来提交事务以保留原始的输入到输出映射的逻辑 512 ；用于在原始顺序循环包含修改数据的操作的情况下使用提交仲裁器来检测并处理并行循环中的冲突的逻辑 514 ；用于在不执行对原始顺序循环的编译器分析的情况下生成第二部分代码的逻辑 515 ；用于在确定原始顺序循环对重新排序（使用试探法、第一部分代码中的用户定义的注释等）免疫的情况下以允许各事务按不取决于原始顺序循环的执行次序的次序来提交的方式来创建第二部分代码的逻辑 516 ；用于生成第二部分代码以便并行地执行事务中的至少某一些的逻辑 517 ；用于以各单独事务中的至少某一些在不同的线程上执行的状态来使用事务存储器系统来执行第二部分代码的逻辑 518 ；以及用于操作该应用程序的其它逻辑 520。在一个实现中，程序逻辑 504 可操作来通过编程，如使用对程序逻辑 504 中的过程的单一调用而从另一程序调用。

[0070] 现在转向图 14，示出在将原始顺序循环转换成并行循环时所涉及的高级阶段的一个实现。在一种形式中，图 14 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 550 处开始，在那里通过取原始顺序循环的每一迭代并生成遵循预定提交次序过程的单独事务（例如，包括相应的工作项），来将顺序循环转换成并行循环，从而尊重与原始顺序循环的原始执行相一致的提交次序（阶段 552）。在另一实现中，在认为每一迭代创建一个事务代价过大的情况下，可以将邻接的迭代带（例如相邻的迭代）一起分组在一事务中（阶段 552）。系统在不执行对原始顺序循环的编译器分析的情况下生成并行循环（阶段 554）。随后执行并行循环，其中各单独事务中的至少某一些被分配到不同的线程以

使它们并行执行（阶段 556）。该过程在结束点 558 处结束。

[0071] 图 15 示出在使用预定提交次序过程来确保并行循环中的事务以正确 次序提交时所涉及的各阶段的一个实现。在一种形式中，图 15 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 570 处开始，在那里将原始顺序循环转换成遵循预定提交次序过程的并行循环（阶段 572）。系统向并行循环中的每一事务分配提交序号（或使用跟踪提交事务的次序的另一合适的方式）（阶段 574）。在执行并行循环时，系统使用预定提交次序过程来确保每一相应事务只可以在并行循环的在前迭代成功完成之后才能完成（例如，使事务等待直到其提交次序揭示它可以提交）（阶段 576）。该过程在结束点 578 处结束。

[0072] 图 16 示出在使用提交仲裁器来检测并处理在执行并行循环时所发生的冲突时所涉及的各阶段的一个实现。在一种形式中，图 16 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 600 处开始，在那里将原始顺序循环转换成使用预定提交次序过程来确保正确排序的并行循环（阶段 602）。系统执行并行循环（阶段 604）。系统随后检测并行循环包含超过一个将修改同一数据元素（例如，因为缺少线程安全性，因为排序要求等）的单独事务（例如，循环迭代）（阶段 606）。使用提交仲裁器来检测并处理在执行并行循环时所发生的冲突，如通过检测后续事务的乱序执行并且一旦前导事务完成则安排后续事务的重新执行（阶段 608）。该过程在结束点 610 处结束。

[0073] 图 17 示出在检测并处理在执行并行循环时所发生的未经处理的异常时所涉及的各阶段的一个实现。在一种形式中，图 17 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 630 处开始，在那里将原始顺序循环转换成使用事务来保留原始的输入到输出映射并提高安全性的并行循环（阶段 632）。系统执行并行循环（阶段 634）并检测在执行并行循环时特定事务中所发生的未经处理的异常（阶段 636）。提交特定事务以及该特定事务的任何前导事务所作出的状态修改（阶段 638）。通过回退它们的事务来丢弃任何后续事务对该特定事务所作出的推测性状态修改（阶段 640）。该过程在结束点 642 处结束。

[0074] 图 18A-18B 示出从原始顺序循环到并行循环的示例性转换的假想源代码。尽管图 18A 示出包含“for...each”循环 652 的原始顺序循环 650，但 可以理解，也可以使用其它形式的循环构造。对于该循环中的每一迭代，执行一个或多个语句 654。图 18B 示出在使用此处讨论的技术中的某一种将顺序循环转换成并行循环 660 之后其看来如何的假想示例。在所示示例中，通过对原始顺序循环 664 的每一迭代生成单独事务来创建并行循环。在另一实现中，在认为每一迭代创建一个事务代价过大的情况下，可以将邻接的迭代带（例如相邻的迭代）一起分组在一事务中。每一单独事务随后创建新工作项以执行作为原始循环中的语句 667 来包括的工作。可使用单独的类 662 来声明工作项迭代。随后将各单独事务分配到不同的线程以使它们可以并行执行。

[0075] 图 19 示出将封闭顺序循环转换成并行循环时所涉及的各阶段的一个实现。在一种形式中，图 19 的过程至少部分地在计算设备 100 的操作逻辑中实现。该过程在起始点 670 处开始，在那里提供事务存储器系统（阶段 672）。系统分析包含原始顺序循环的第一部分代码以确定原始顺序循环将执行的固定迭代次数（例如，通过检索用于确定循环终点的常数值）（阶段 674）。将包含原始顺序循环的第一部分代码转换成包含可以生成等于固

定迭代次数的事务的并行循环的第二部分代码（阶段674）。系统使用事务存储器系统来执行第二部分代码，其中将这些事务中的至少某一些分配到不同的线程以使它们可以并行执行（阶段678）。系统使用预定提交次序过程来以正确次序提交事务（例如，其中每一事务使用相应的归纳变量计数器作为提交顺序号）（阶段680）。该过程在结束点682处结束。

[0076] 在一个实现中，图19中描述的事务过程只用于在循环主体本身中从不对归纳变量进行写入的循环。换言之，可通过在循环主体中对归纳变量进行写入或通过取归纳变量的地址并对其进行可导致写如的某些操作（将其传递给函数、生成它的别名、等等）来使循环无法执行。

[0077] 图20示出使用推测流水线来将开放顺序循环转换成并行循环时所涉及的各阶段的一个实现。在一种形式中，图20的过程至少部分地在计算设备100的操作逻辑中实现。该过程在起始点700处开始，在那里提供事务存储器系统（阶段702）。系统将包含开放顺序循环的第一部分代码转换成包含可操作来生成单独事务的并行循环的第二部分代码（例如，使得至少 某些事务并行执行），其中该单独事务包含推测流水线的每一迭代的相应工作项（阶段704）。在不执行对开放顺序循环的编译器分析的情况下生成第二部分代码（阶段706）。系统使用事务存储器系统来执行第二部分代码，其中将这些单独事务中的至少某一些分配到不同的线程以使它们并行执行（阶段708）。通过以预定提交次序（例如，与开放顺序循环的执行次序相一致）提交事务来保留原始的输入到输出映射（阶段710）。该过程在结束点712处结束。

[0078] 图21示出在执行从开放顺序循环生成的并行循环时所涉及的各阶段的一个实现。在一种形式中，图21的过程至少部分地在计算设备100的操作逻辑中实现。该过程在起始点730处开始，在那里生成估计要在从开放顺序循环（例如，“while”循环、“do while”循环、“for”循环等）生成的并行循环中执行的迭代次数的推测流水线（阶段732）。在一个实现中，系统取推测流水线的每一迭代并生成包含相应工作项的单独事务（阶段734）。在另一实现中，如在认为每一迭代创建一个事务代价过大的情况下，系统取邻接的迭代带（例如相邻的迭代）并将它们一起分组在一事务中（阶段734）。系统将各单独事务分配到不同的线程以使它们并行执行（阶段735）。系统对每一相应工作项评估终止条件（阶段736）。在相应工作项中的特定一个确定到达了终止并行循环的时间时，提交前导事务并丢弃后续事务（阶段738）。该过程在结束点740处结束。

[0079] 图22示出在确保从开放顺序循环生成的并行循环中的每一工作项都以正确次序提交时所涉及的各阶段的一个实现。在一种形式中，图22的过程至少部分地在计算设备100的操作逻辑中实现。该过程在起始点760处开始，在那里当相应事务中的每一相应工作项执行时检索当前迭代值（阶段762）。在一个实现中，当前迭代值是通过对可由每一相应工作项访问的值执行原子递增来检索的（阶段762）。系统使用每一相应工作项的当前迭代值作为预定提交次序过程中的提交顺序号（阶段764）。系统实现与开放顺序循环的原始执行相一致的提交次序（阶段766）。该过程在结束点768处结束。

[0080] 图23示出在计算推测流水线以确定并行循环中要包括多少循环时所 涉及的各阶段的一个实现。在一种形式中，图23的过程至少部分地在计算设备100的操作逻辑中实现。该过程在起始点790处开始，在那里系统至少部分地基于执行并行循环的计算机上的可用处理器的数量来生成推测流水线的初始值（阶段792）。在一个实现中，推测流水线的

初始值是基于处理器的数量除以工作负载花费在进行绑定 CPU(CPU-bound) 的工作的时间百分比来计算的（阶段 792)。也可使用多种其它计算。使用初始值来确定要为并行循环的特定执行创建并行循环的多少迭代（阶段 794)。系统可以使用自适应统计量来调整推测流水线以供并行循环的稍后执行（例如使用历史来更好地确定循环的预期持续时间，自适应地调整何时阻塞工作项，等等）（阶段 796)。该过程在结束点 798 处结束。

[0081] 图 24A-24B 示出从原始开放顺序循环到并行循环的示例性转换的假想源代码。术语“开放顺序循环”旨在包括其迭代次数未知的顺序循环。如图 24A 所示，示出了原始开放顺序循环 810。该循环是在条件为真时（例如，在所示示例中在 P ＝ true（真）时）执行某些语句的“while”循环。图 24B 示出原始开放顺序循环如何被转换成并行循环 820。如在图 24B 的假想代码中所示，对于推测流水线的每一迭代，生成可以并行运行的工作项。在一个实现中，可以针对这一点来使用标准工作窃用队列。称为 currentIteration（当前迭代）的共享变量可由每一工作项访问。在每一工作项执行时，其对 currentIteration 执行原子递增，如通过标准比较并交换硬件指令或另一机制，来读取其自己的迭代值。这确保任何一个迭代只由单个工作者处理并且可以确定事务开始执行循环的各迭代中的一个的次序。这随后成为事务的提交顺序号，并确保该迭代能以正确次序在前导事务和后续事务之间串行。每一工作项在该工作之前或之后评估 P 或适用的任何其它终止条件，视循环构造而定（例如在图 24B 所示的“while”的情况中是之前，但在“do-while”的情况中是“之后”）。在工作者中的一个认识到到达了终止时间时，所有前导事务必须提交并且随后所有后续事务必须丢弃。

[0082] 尽管此处所讨论的各示例探讨了使用各种技术和方法来强制实施提交排序，但应当注意，事务可能根本不具有提交仲裁器。在事务不具有提交 仲裁器的情况下，普通的无序提交将会发生。

[0083] 尽管用对结构特征和 / 或方法动作专用的语言描述了本主题，但可以理解，所附权利要求书中定义的主题不必限于上述具体特征或动作。相反，上述具体特征和动作是作为实现权利要求的示例形式公开的。落入在此所述和 / 或所附权利要求所描述的实现的精神的范围内的所有等效方案、更改和修正都期望受到保护。

[0084] 例如，计算机软件领域普通技术人员会认识到在此讨论的示例中所述的客户机和 / 或服务器布置、用户界面屏幕内容、和 / 或数据布局可在一台或多台计算机上不同地组织，以包括比示例中所描绘的更少或更多的选项或特征。

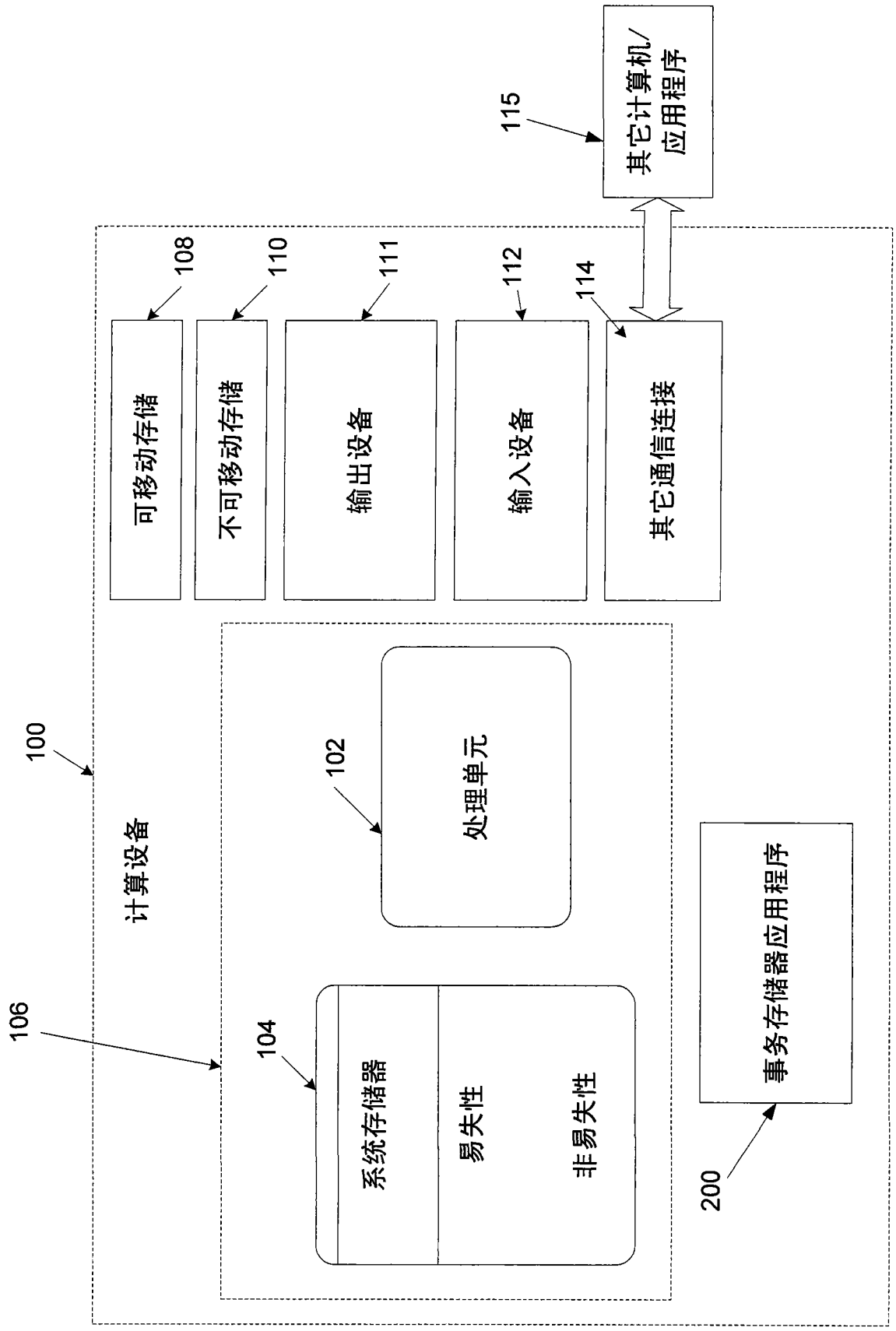

其它计算机/
应用程序
115
108
110
111
112
114
可移动存储
不可移动存储
输出设备
输入设备
其它通信连接
100
计算设备
102
处理单元
106
104
系统存储器
易失性
非易失性
事务存储器应用程序
200

图 1

事务存储器应用程序
200

程序逻辑 204

用于提供事务存储器（STM）系统的逻辑 206

用于提供允许为STM系统中的多个事务（静态或动态地）指定预定提交次序的提交仲裁器的逻辑 208

用于允许提交仲裁器在运行时使用预定提交次序来帮助确定提交事务存储器系统中的多个事务的次序的逻辑 210

用于提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程的逻辑 212

用于在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续（例如，取决于同一事务组中的两个事务中的哪个事务具有较低提交序号）的逻辑 214

用于允许提交仲裁器可操作来使用预定排序来跟踪一个或多个排序值（例如，在总体排序中，表示多个事务中应被允许提交的下一事务的下一个提交字段）以及用于将该一个或多个排序值与给定事务的特定提交序号进行比较以查看该给定事务的提交是否是适当的（给定应强制实施的排序）的逻辑 216

用于操作该应用程序的其它逻辑 220

图 2

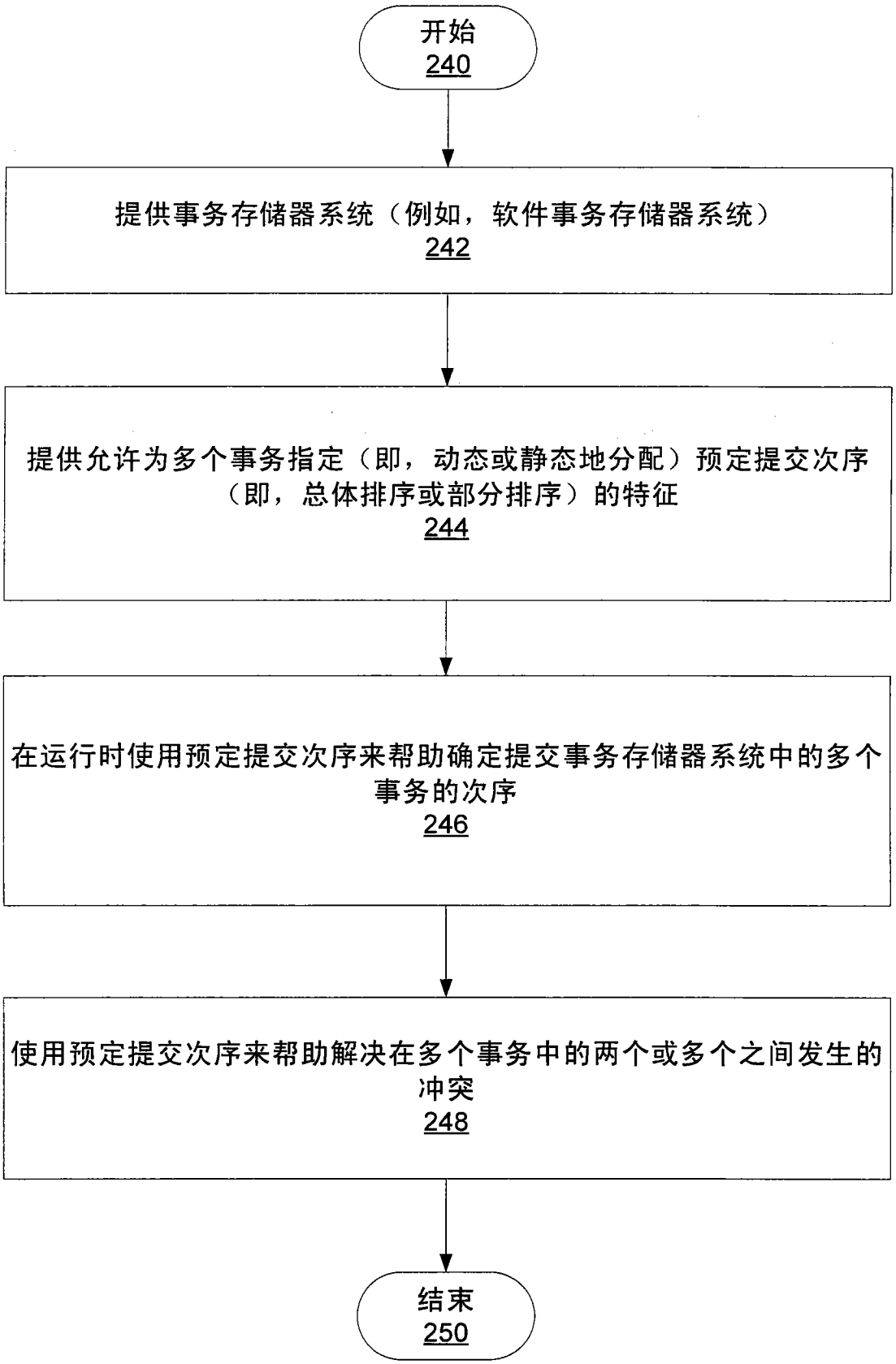

开始
240
提供事务存储器系统（例如，软件事务存储器系统）
242
提供允许为多个事务指定（即，动态或静态地分配）预定提交次序（即，总体排序或部分排序）的特征
244
在运行时使用预定提交次序来帮助确定提交事务存储器系统中的多个事务的次序
246
使用预定提交次序来帮助解决在多个事务中的两个或多个之间发生的冲突
248
结束
250

图 3

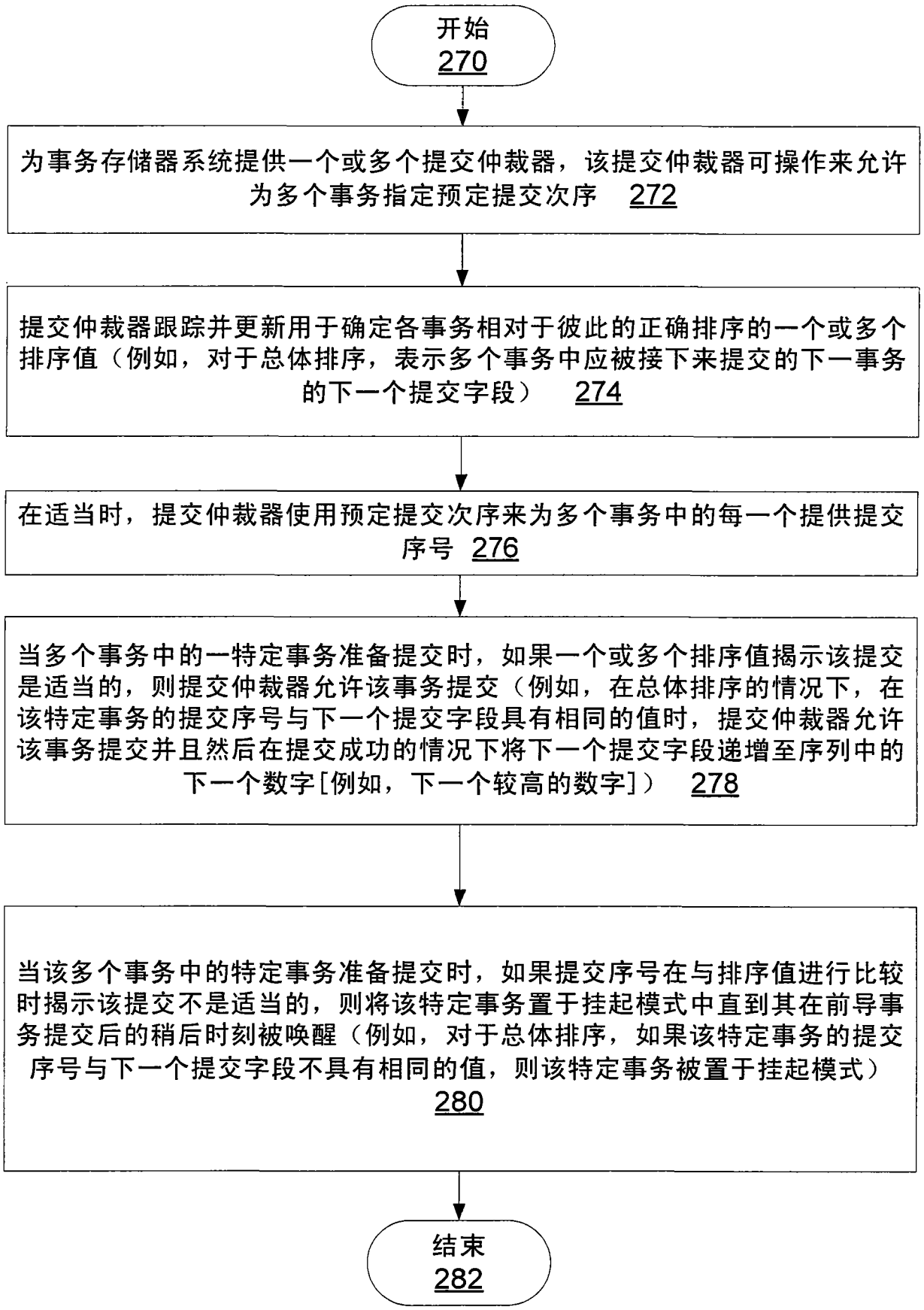

开始
270
为事务存储器系统提供一个或多个提交仲裁器，该提交仲裁器可操作来允许为多个事务指定预定提交次序 272
提交仲裁器跟踪并更新用于确定各事务相对于彼此的正确排序的一个或多个排序值（例如，对于总体排序，表示多个事务中应被接下来提交的下一事务的下一个提交字段） 274
在适当时，提交仲裁器使用预定提交次序来为多个事务中的每一个提供提交序号 276
当多个事务中的一特定事务准备提交时，如果一个或多个排序值揭示该提交是适当的，则提交仲裁器允许该事务提交（例如，在总体排序的情况下，在该特定事务的提交序号与下一个提交字段具有相同的值时，提交仲裁器允许该事务提交并且然后在提交成功的情况下将下一个提交字段递增至序列中的下一个数字[例如，下一个较高的数字]） 278
当该多个事务中的特定事务准备提交时，如果提交序号在与排序值进行比较时揭示该提交不是适当的，则将该特定事务置于挂起模式中直到其在前导事务提交后的稍后时刻被唤醒（例如，对于总体排序，如果该特定事务的提交序号与下一个提交字段不具有相同的值，则该特定事务被置于挂起模式） 280
结束
282

图 4

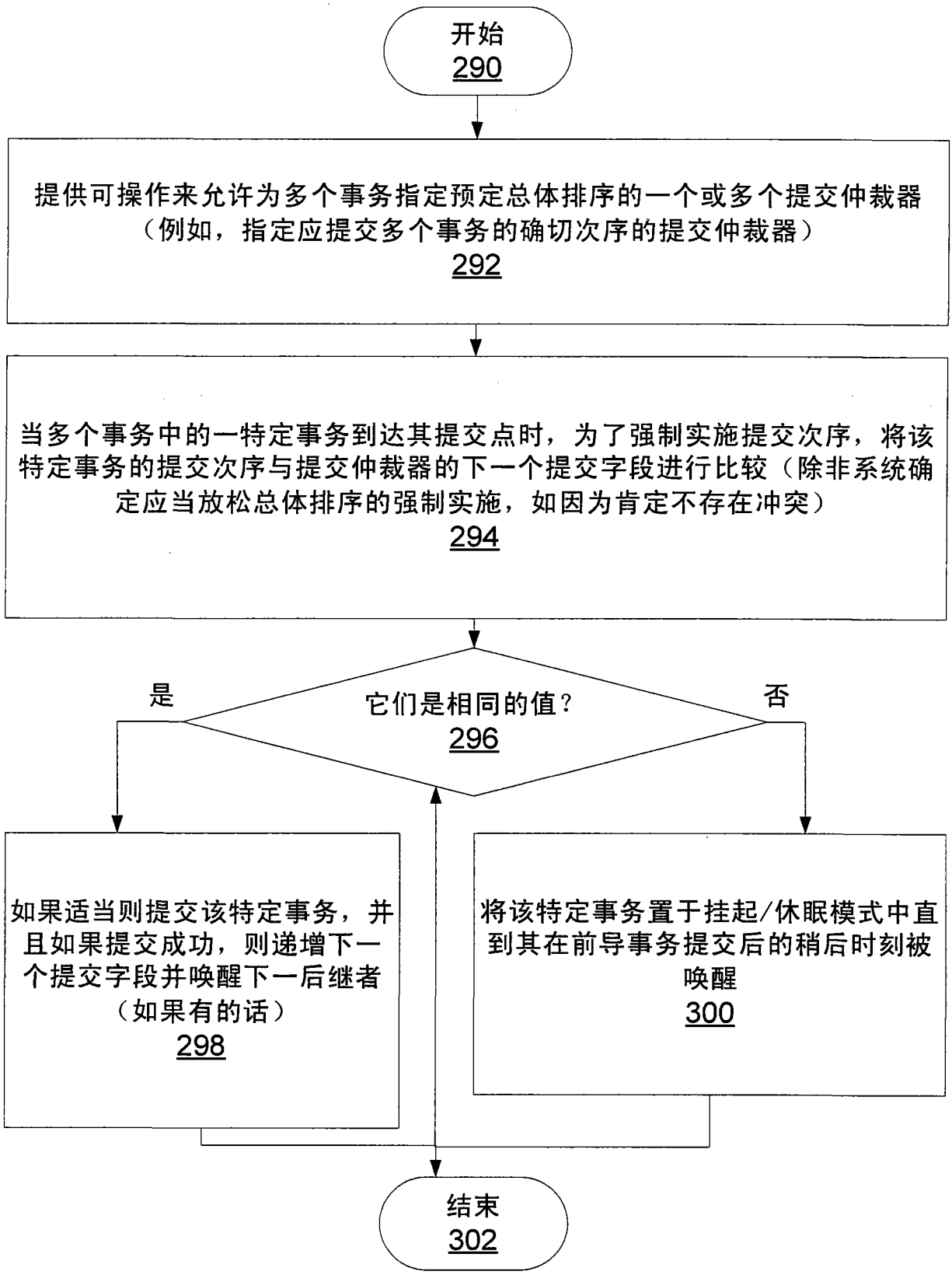

开始
290
提供可操作来允许为多个事务指定预定总体排序的一个或多个提交仲裁器（例如，指定应提交多个事务的确切次序的提交仲裁器）
292
当多个事务中的一特定事务到达其提交点时，为了强制实施提交次序，将该特定事务的提交次序与提交仲裁器的下一个提交字段进行比较（除非系统确定应当放松总体排序的强制实施，如因为肯定不存在冲突）
294
它们是相同的值？
296
是
否
如果适当则提交该特定事务，并且如果提交成功，则递增下一个提交字段并唤醒下一后继者（如果有的话）
298
将该特定事务置于挂起/休眠模式中直到其在前导事务提交后的稍后时刻被唤醒
300
结束
302

图 5

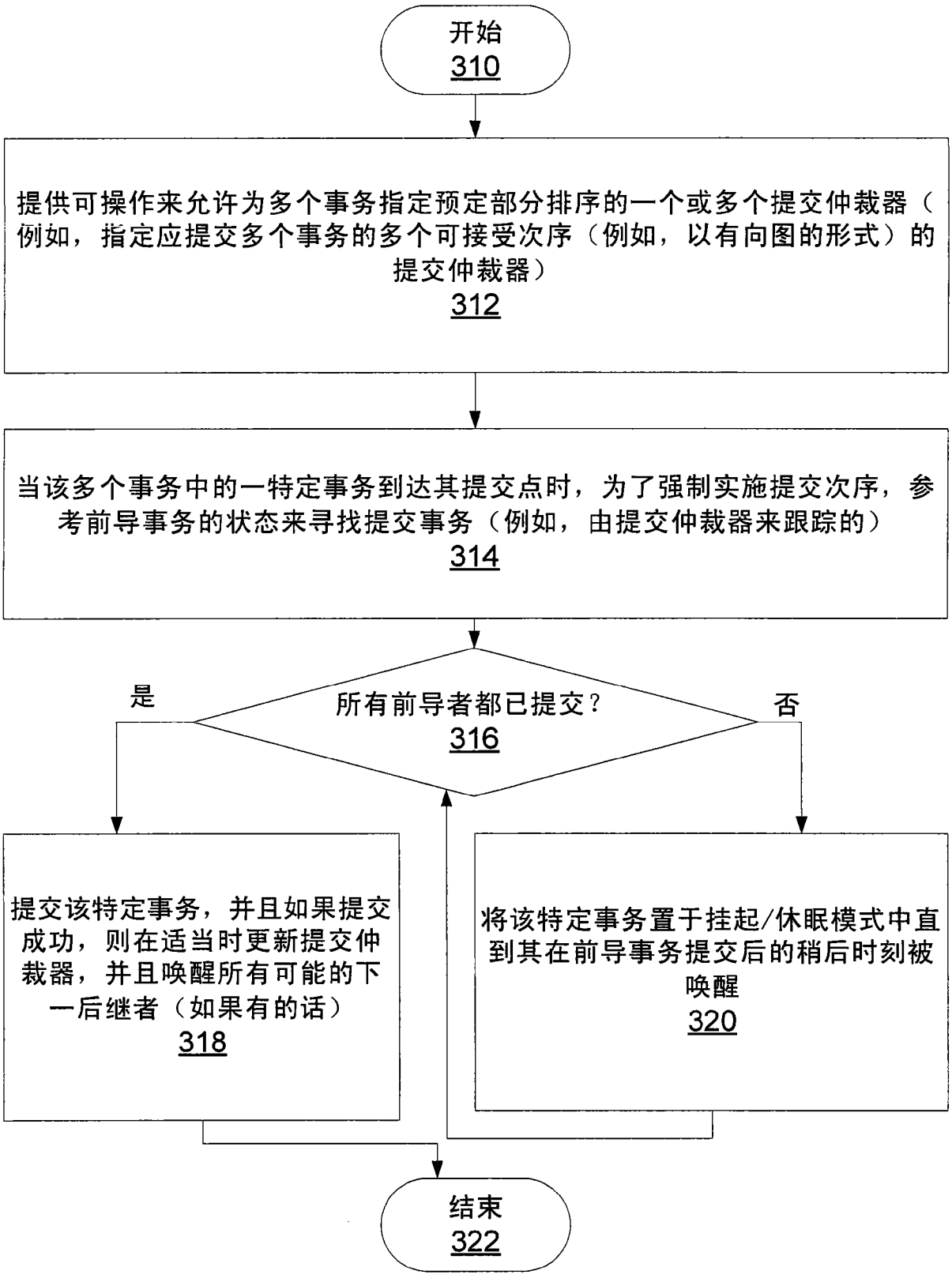

开始
310
提供可操作来允许为多个事务指定预定部分排序的一个或多个提交仲裁器（例如，指定应提交多个事务的多个可接受次序（例如，以有向图的形式）的提交仲裁器）
312
当该多个事务中的一特定事务到达其提交点时，为了强制实施提交次序，参考前导事务的状态来寻找提交事务（例如，由提交仲裁器来跟踪的）
314
所有前导者都已提交？
316
是
否
提交该特定事务，并且如果提交成功，则在适当时更新提交仲裁器，并且唤醒所有可能的下一后继者（如果有的话）
318
将该特定事务置于挂起/休眠模式中直到其在前导事务提交后的稍后时刻被唤醒
320
结束
322

图 6

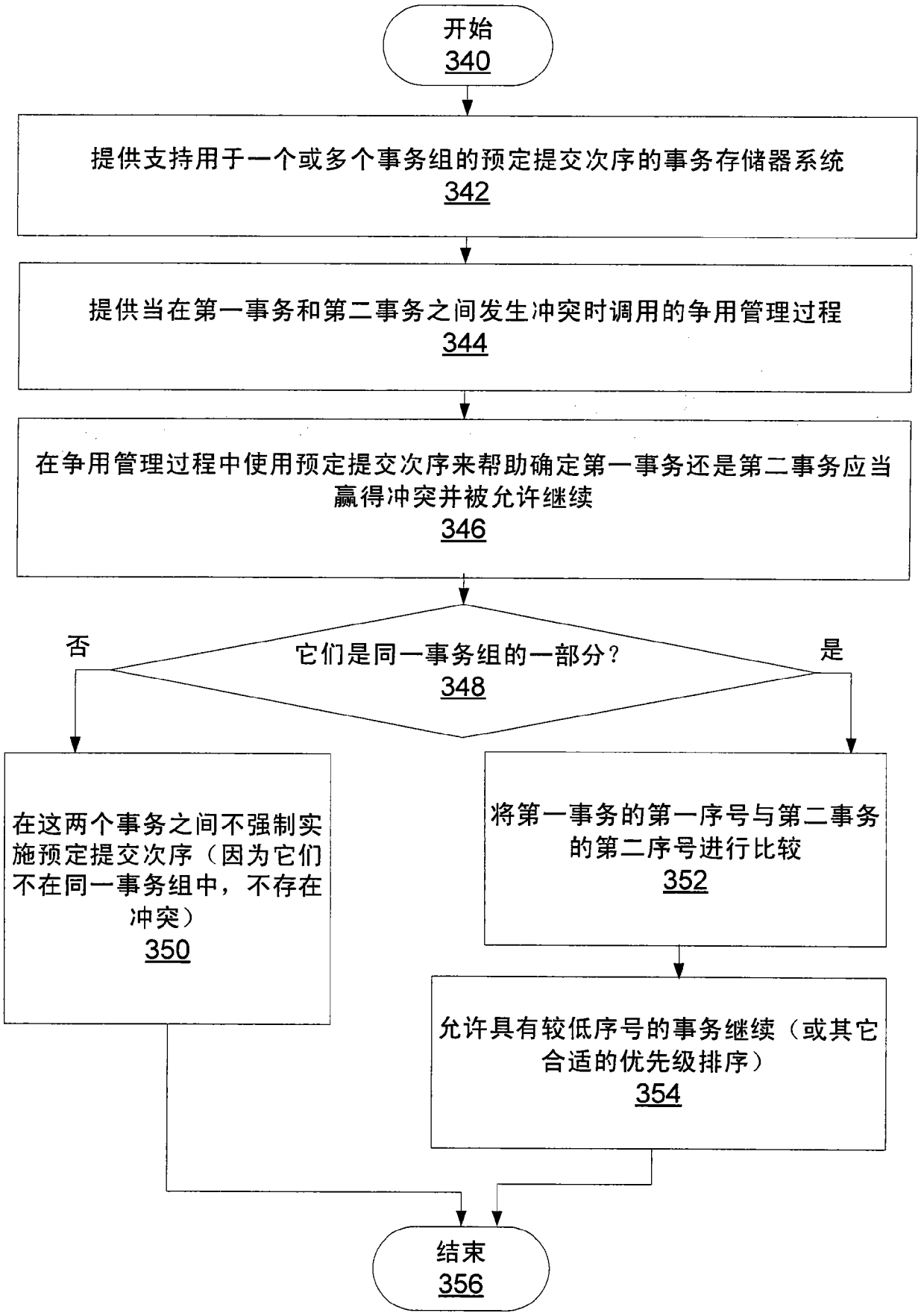

开始
340
提供支持用于一个或多个事务组的预定提交次序的事务存储器系统
342
提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程
344
在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续
346
它们是同一事务组的一部分？
348
否
是
在这两个事务之间不强制实施预定提交次序（因为它们不在同一事务组中，不存在冲突）
350
将第一事务的第一序号与第二事务的第二序号进行比较
352
允许具有较低序号的事务继续（或其它合适的优先级排序）
354
结束
356

图 7

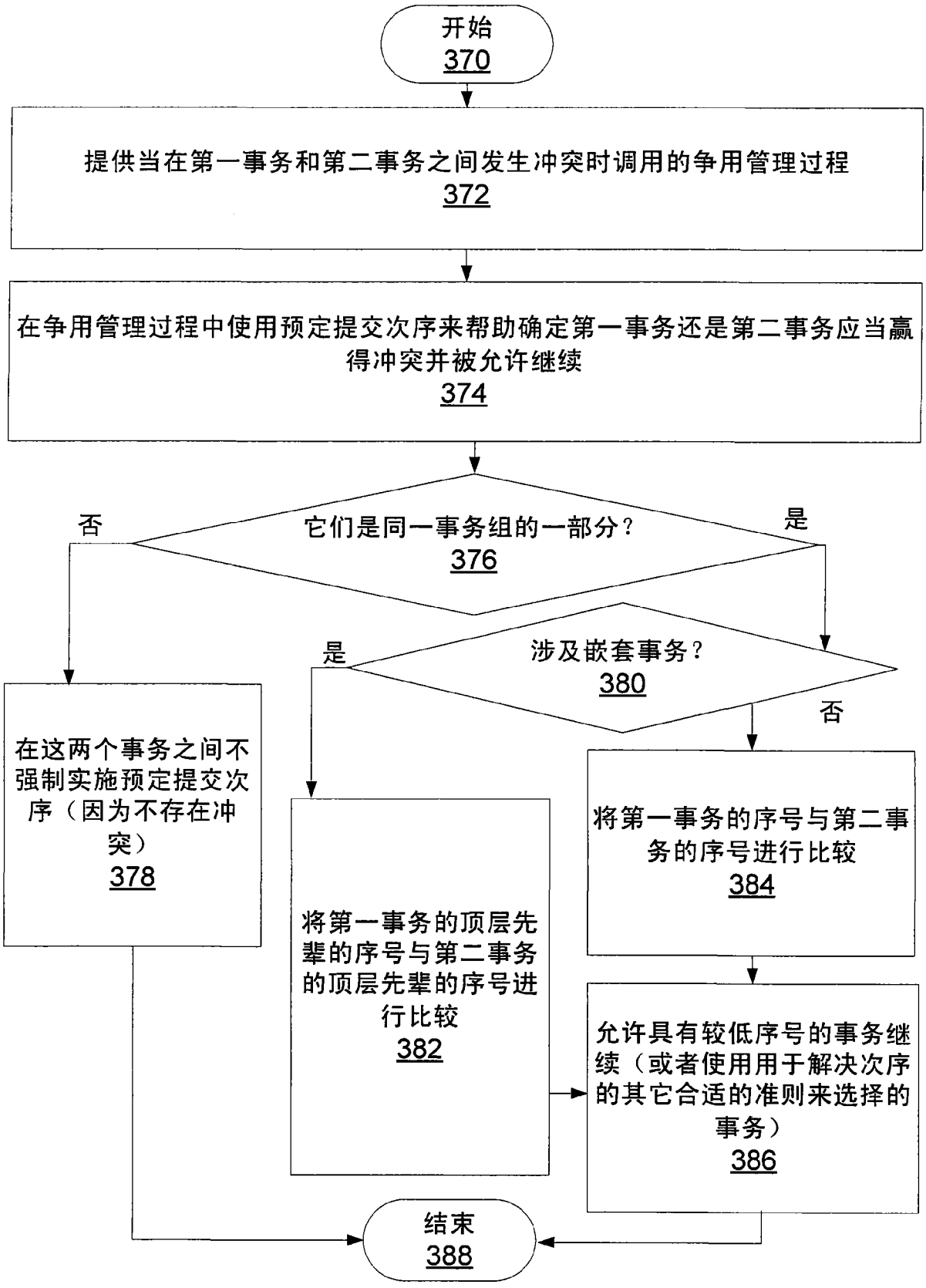

开始
370
提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程
372
在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续
374
它们是同一事务组的一部分？
376
否
是
涉及嵌套事务？
380
是
否
在这两个事务之间不强制实施预定提交次序（因为不存在冲突）
378
将第一事务的顶层先辈的序号与第二事务的顶层先辈的序号进行比较
382
将第一事务的序号与第二事务的序号进行比较
384
允许具有较低序号的事务继续（或者使用用于解决次序的其它合适的准则来选择的事务）
386
结束
388

图 8

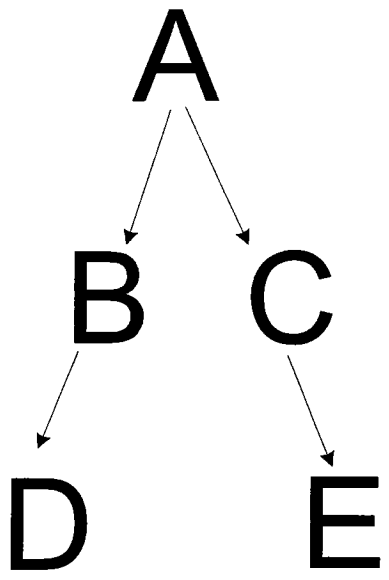


图 9

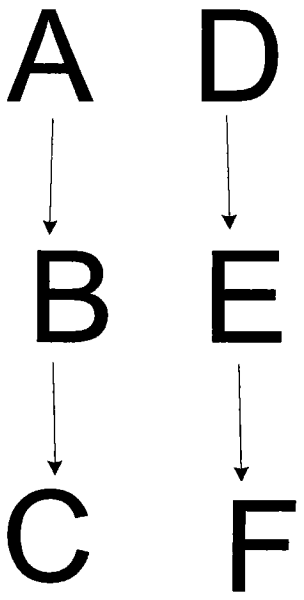


图 10

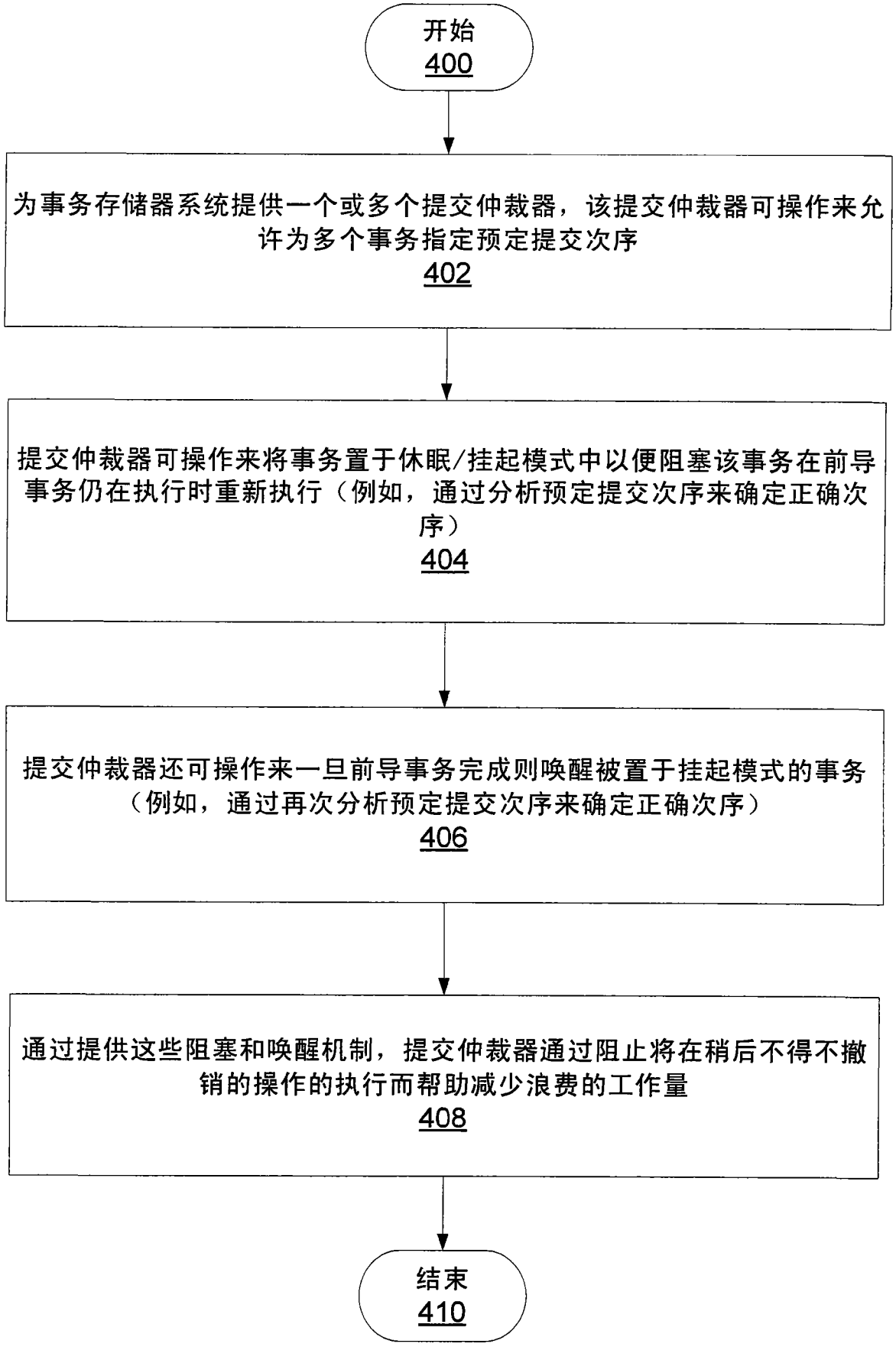

开始
400
为事务存储器系统提供一个或多个提交仲裁器，该提交仲裁器可操作来允许为多个事务指定预定提交次序
402
提交仲裁器可操作来将事务置于休眠/挂起模式中以便阻塞该事务在前导事务仍在执行时重新执行（例如，通过分析预定提交次序来确定正确次序）
404
提交仲裁器还可操作来一旦前导事务完成则唤醒被置于挂起模式的事务（例如，通过再次分析预定提交次序来确定正确次序）
406
通过提供这些阻塞和唤醒机制，提交仲裁器通过阻止将在稍后不得不撤销的操作的执行而帮助减少浪费的工作量
408
结束
410

图 11

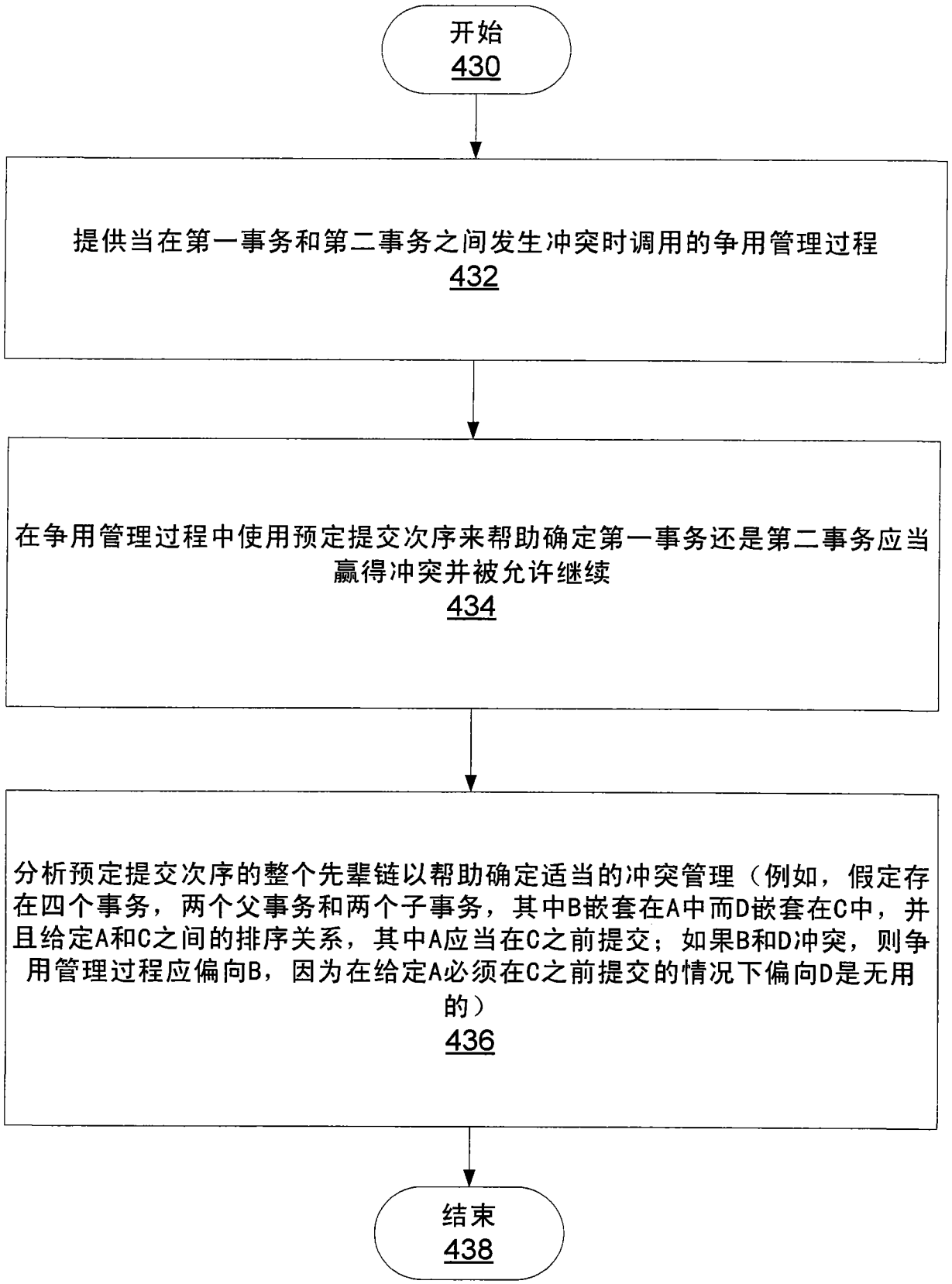

开始
430
提供当在第一事务和第二事务之间发生冲突时调用的争用管理过程
432
在争用管理过程中使用预定提交次序来帮助确定第一事务还是第二事务应当赢得冲突并被允许继续
434
分析预定提交次序的整个先辈链以帮助确定适当的冲突管理（例如，假定存在四个事务，两个父事务和两个子事务，其中B嵌套在A中而D嵌套在C中，并且给定A和C之间的排序关系，其中A应当在C之前提交；如果B和D冲突，则争用管理过程应偏向B，因为在给定A必须在C之前提交的情况下偏向D是无用的）
436
结束
438

图 12

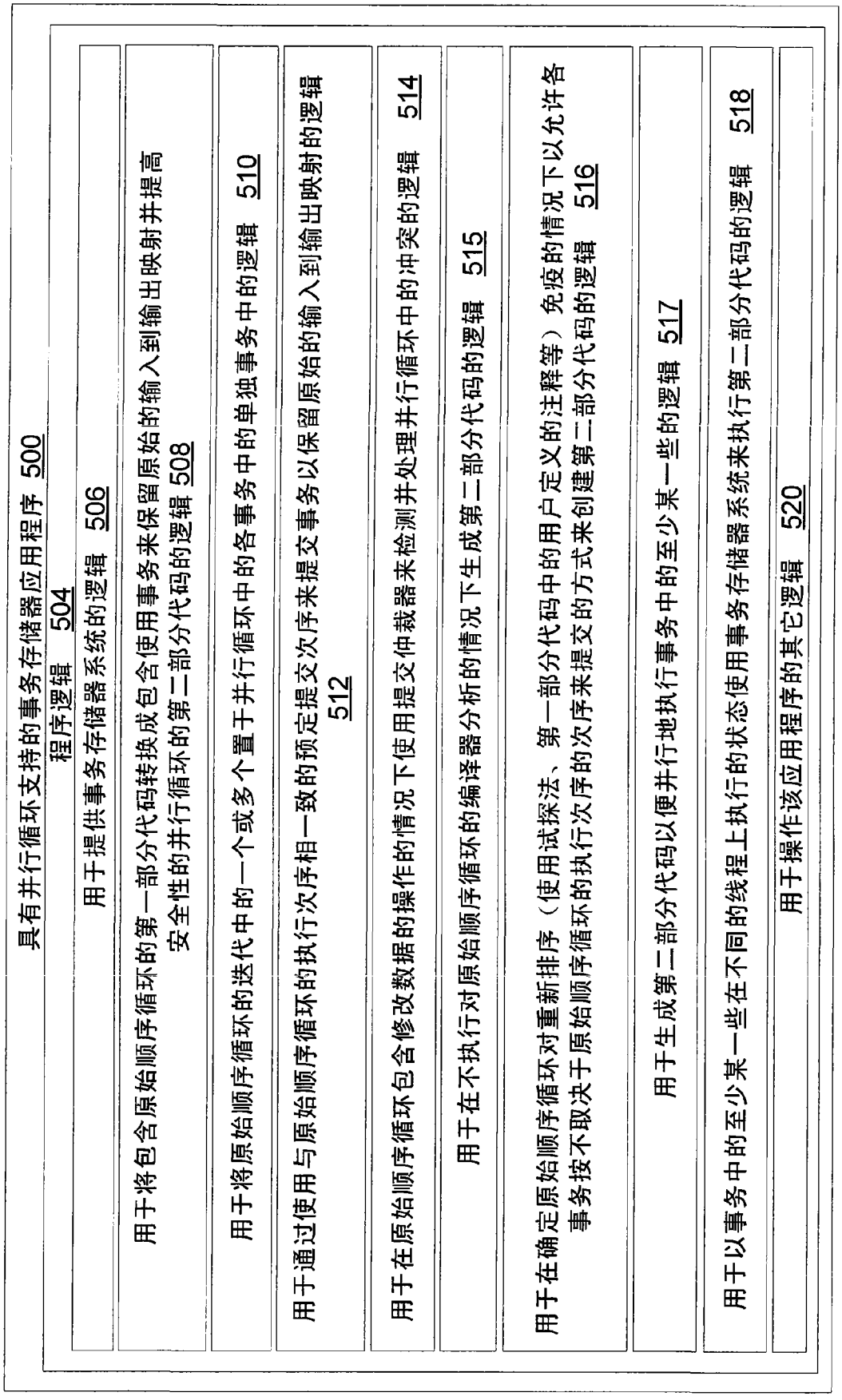

具有并行循环支持的事务存储器应用程序 500
程序逻辑 504
用于提供事务存储器系统的逻辑 506
用于将包含原始顺序循环的第一部分代码转换成包含使用事务来保留原始的输入到输出映射并提高安全性的并行循环的第二部分代码的逻辑 508
用于将原始顺序循环的迭代中的一个或多个置于并行循环中的各事务中的单独事务中的逻辑 510
用于通过使用与原始顺序循环的执行次序相一致的预定提交次序来提交事务以保留原始的输入到输出映射的逻辑 512
用于在原始顺序循环包含修改数据的操作的情况下使用提交仲裁器来检测并处理并行循环中的冲突的逻辑 514
用于在不执行对原始顺序循环的编译器分析的情况下生成第二部分代码的逻辑 515
用于在确定原始顺序循环对重新排序（使用试探法、第一部分代码中的用户定义的注释等）免疫的情况下以允许各事务按不取决于原始顺序循环的执行次序的次序来提交的方式来创建第二部分代码的逻辑 516
用于生成第二部分代码以便并行地执行事务中的至少某一些的逻辑 517
用于以事务中的至少某一些在不同的线程上执行的状态使用事务存储器系统来执行第二部分代码的逻辑 518
用于操作该应用程序的其它逻辑 520

图 13

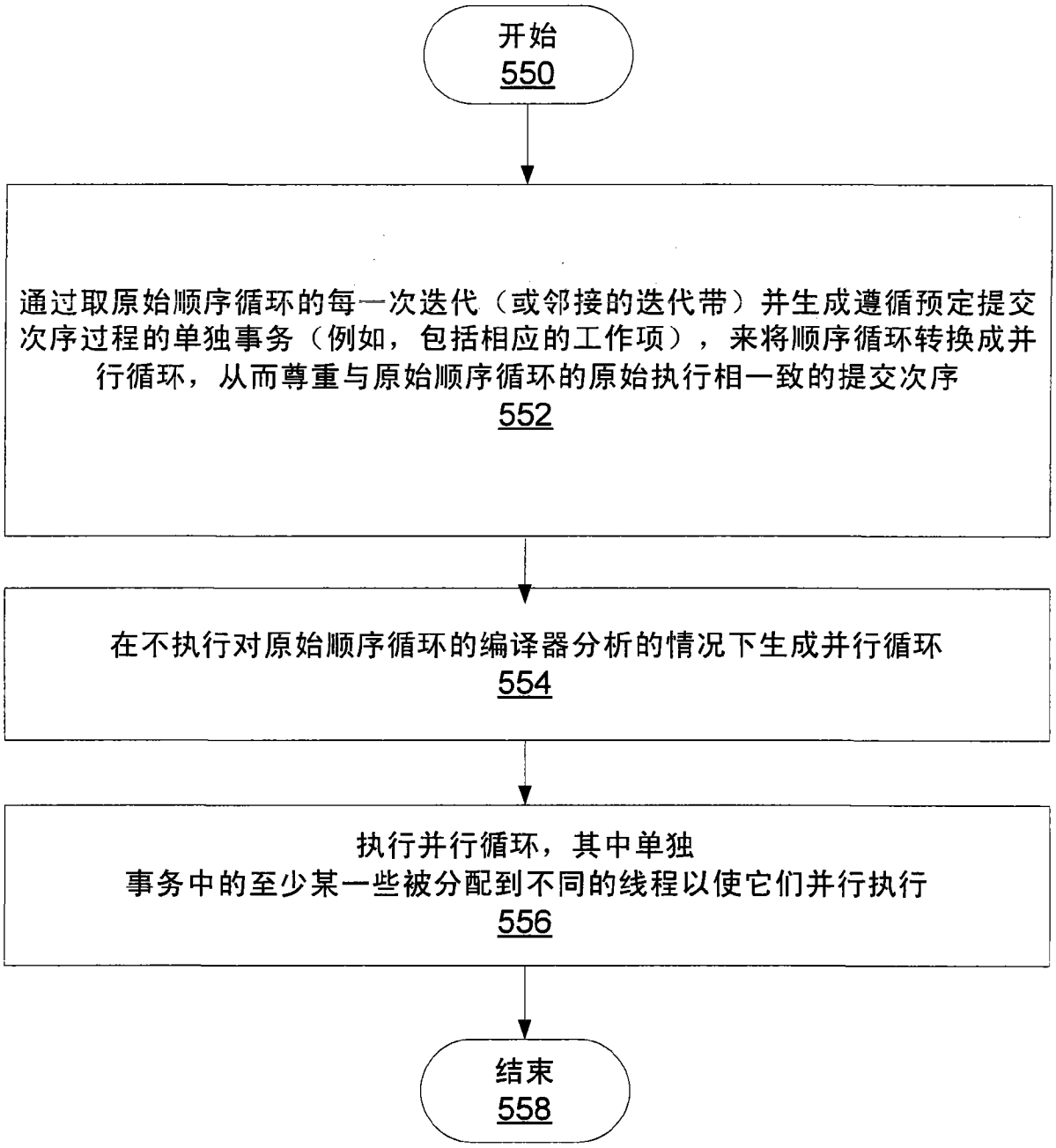

开始
550
通过取原始顺序循环的每一次迭代（或邻接的迭代带）并生成遵循预定提交次序过程的单独事务（例如，包括相应的工作项），来将顺序循环转换成并行循环，从而尊重与原始顺序循环的原始执行相一致的提交次序
552
在不执行对原始顺序循环的编译器分析的情况下生成并行循环
554
执行并行循环，其中单独
事务中的至少某一些被分配到不同的线程以使它们并行执行
556
结束
558

图 14

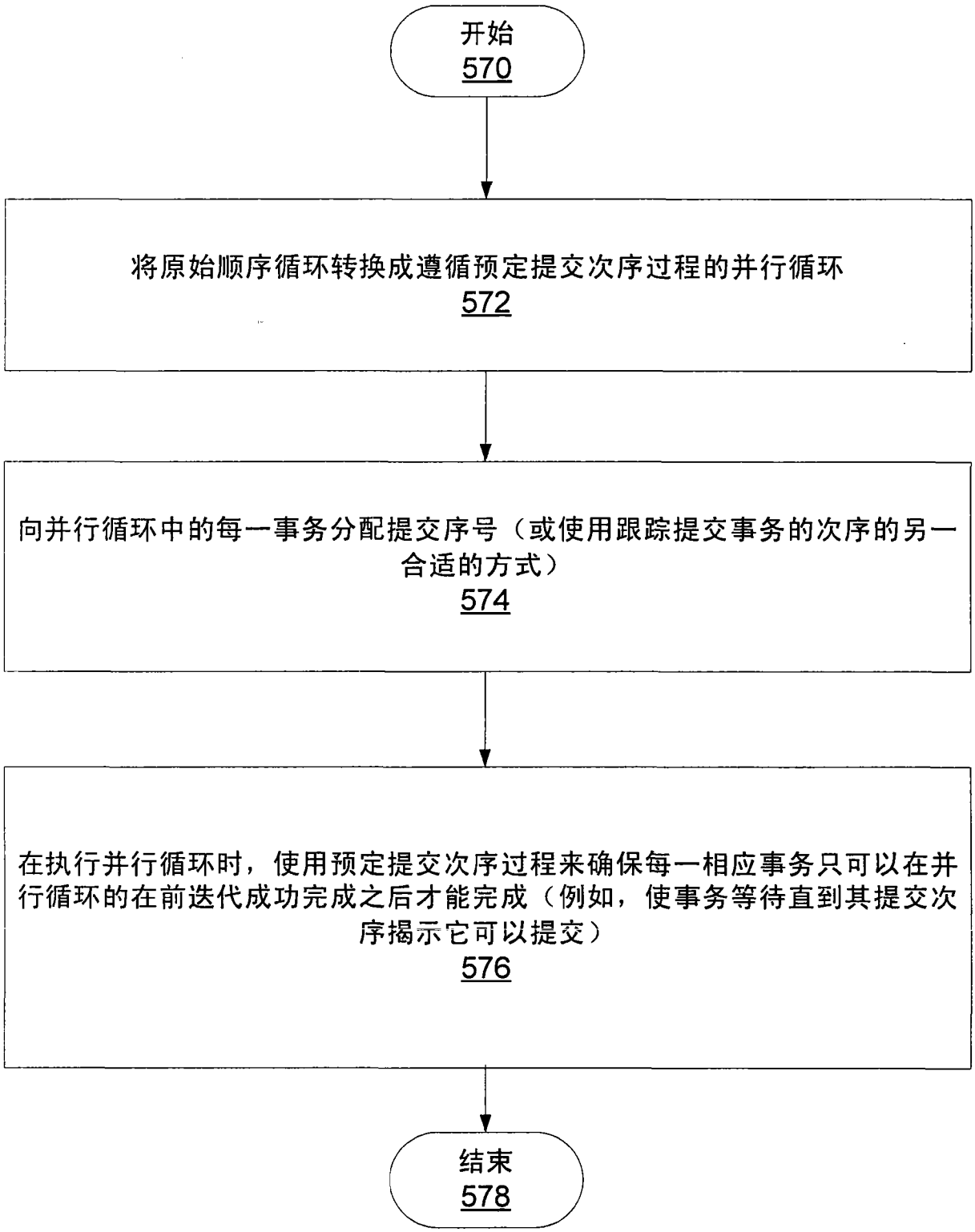

开始
570
将原始顺序循环转换成遵循预定提交次序过程的并行循环
572
向并行循环中的每一事务分配提交序号（或使用跟踪提交事务的次序的另一合适的方式）
574
在执行并行循环时，使用预定提交次序过程来确保每一相应事务只可以在并行循环的在前迭代成功完成之后才能完成（例如，使事务等待直到其提交次序揭示它可以提交）
576
结束
578

图 15

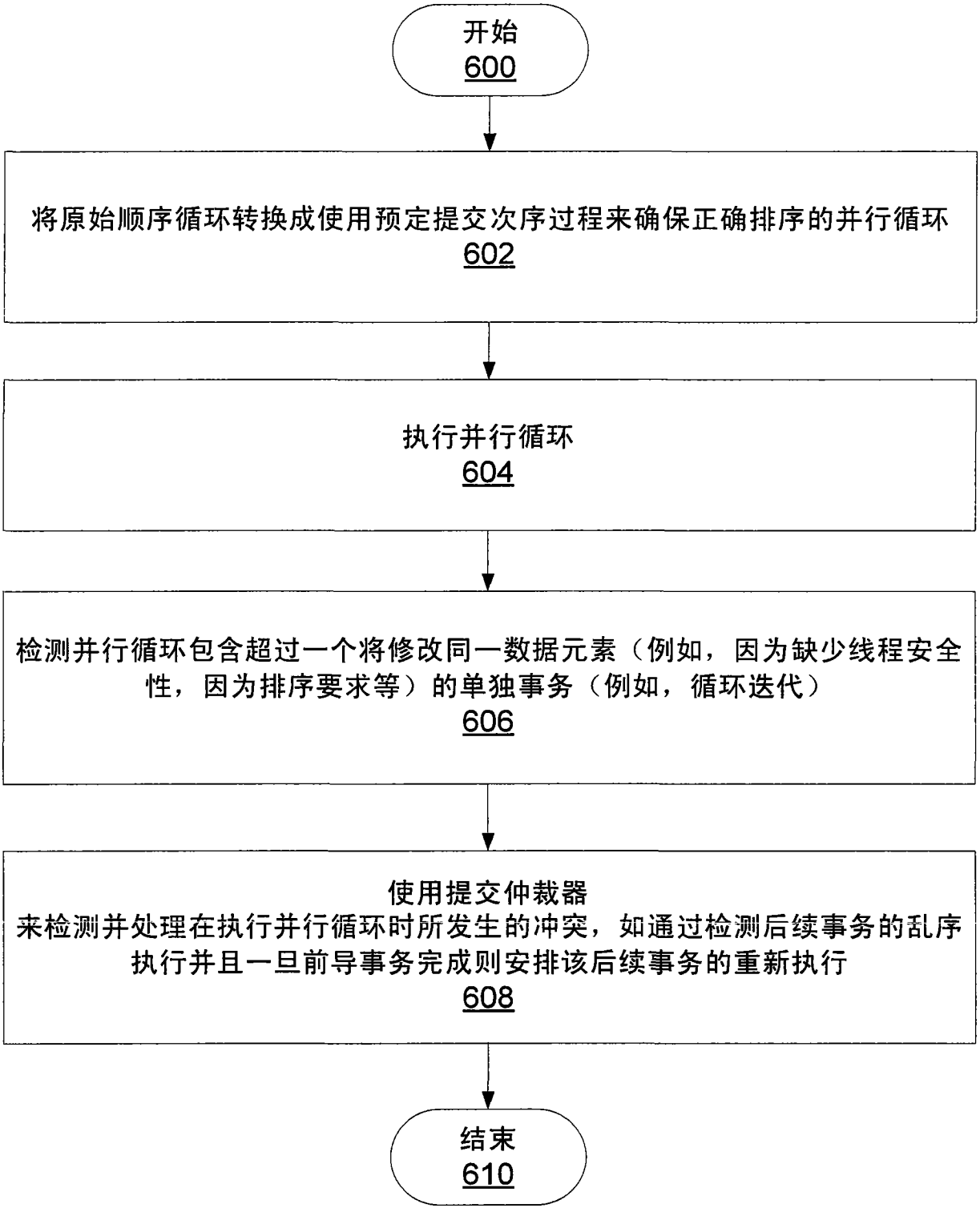

开始
600
将原始顺序循环转换成使用预定提交次序过程来确保正确排序的并行循环
602
执行并行循环
604
检测并行循环包含超过一个将修改同一数据元素（例如，因为缺少线程安全性，因为排序要求等）的单独事务（例如，循环迭代）
606
使用提交仲裁器
来检测并处理在执行并行循环时所发生的冲突，如通过检测后续事务的乱序执行并且一旦前导事务完成则安排该后续事务的重新执行
608
结束
610

图 16

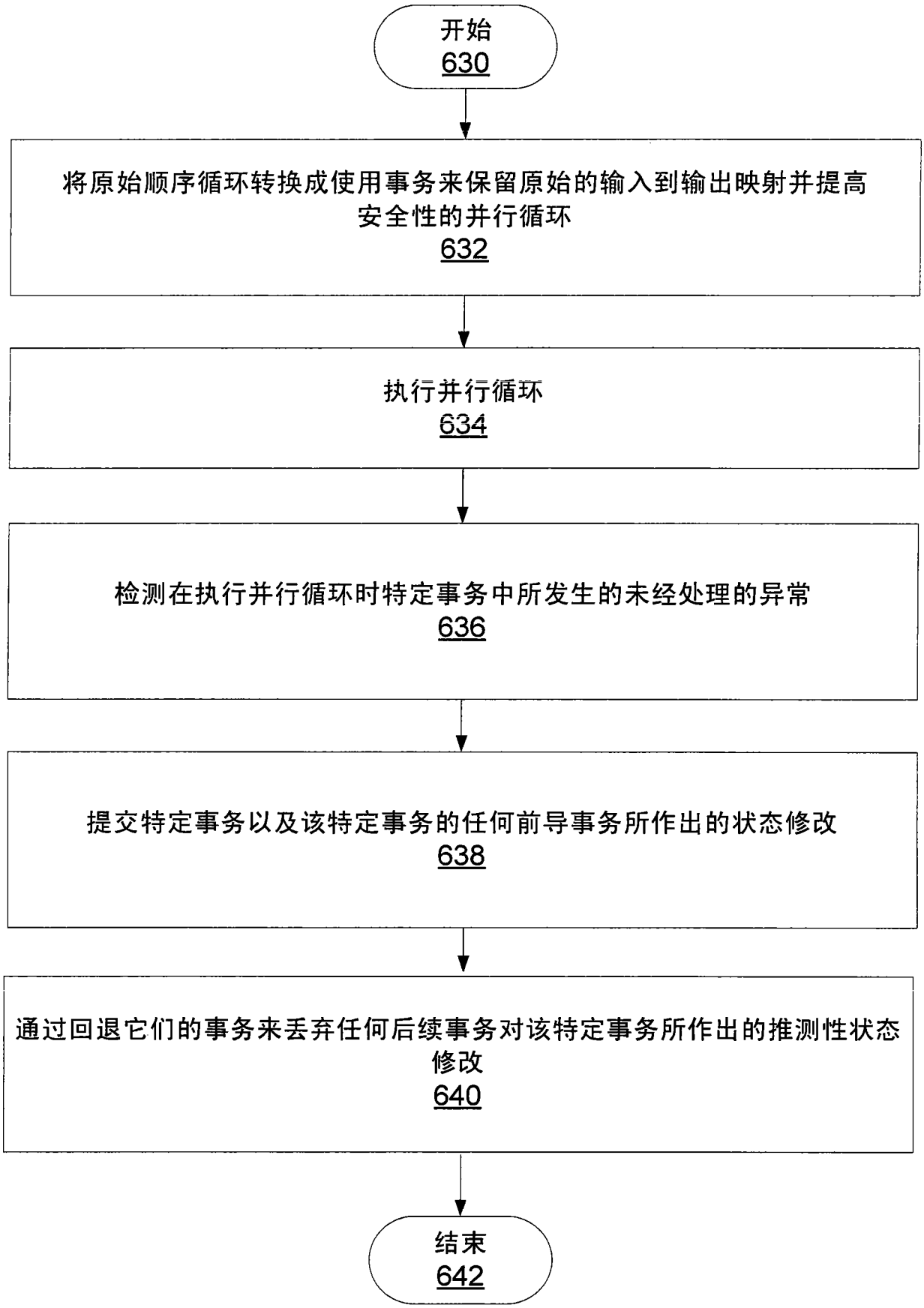

开始
630
将原始顺序循环转换成使用事务来保留原始的输入到输出映射并提高安全性的并行循环
632
执行并行循环
634
检测在执行并行循环时特定事务中所发生的未经处理的异常
636
提交特定事务以及该特定事务的任何前导事务所作出的状态修改
638
通过回退它们的事务来丢弃任何后续事务对该特定事务所作出的推测性状态修改
640
结束
642

图 17

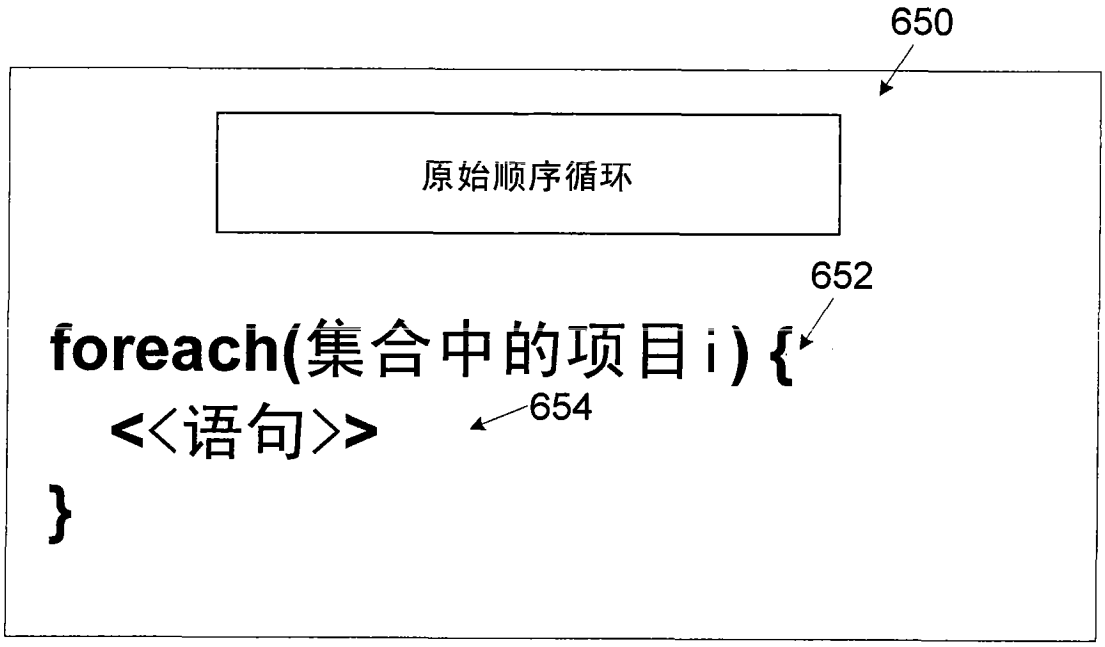


图 18A

660
并行循环
662
public class IterationWorkItem
{ Delegate loopBody;
Transaction transaction;
...
void Run() {
transaction.Execute(loopBody); } }
Transaction previousTransaction = null;
664
foreach(集合中的项目 i) {
Transaction currentTransaction = new Transaction(... ,
previousTransaction);
667
IterationWorkItem iwi = new IterationWorkItem(
delegate() { <语句>},
previousTransaction);
<将 i w i 入队以用于异步执行>
previousTransaction = currentTransaction;
<等待直到所有工作项完成>
}

图 18B

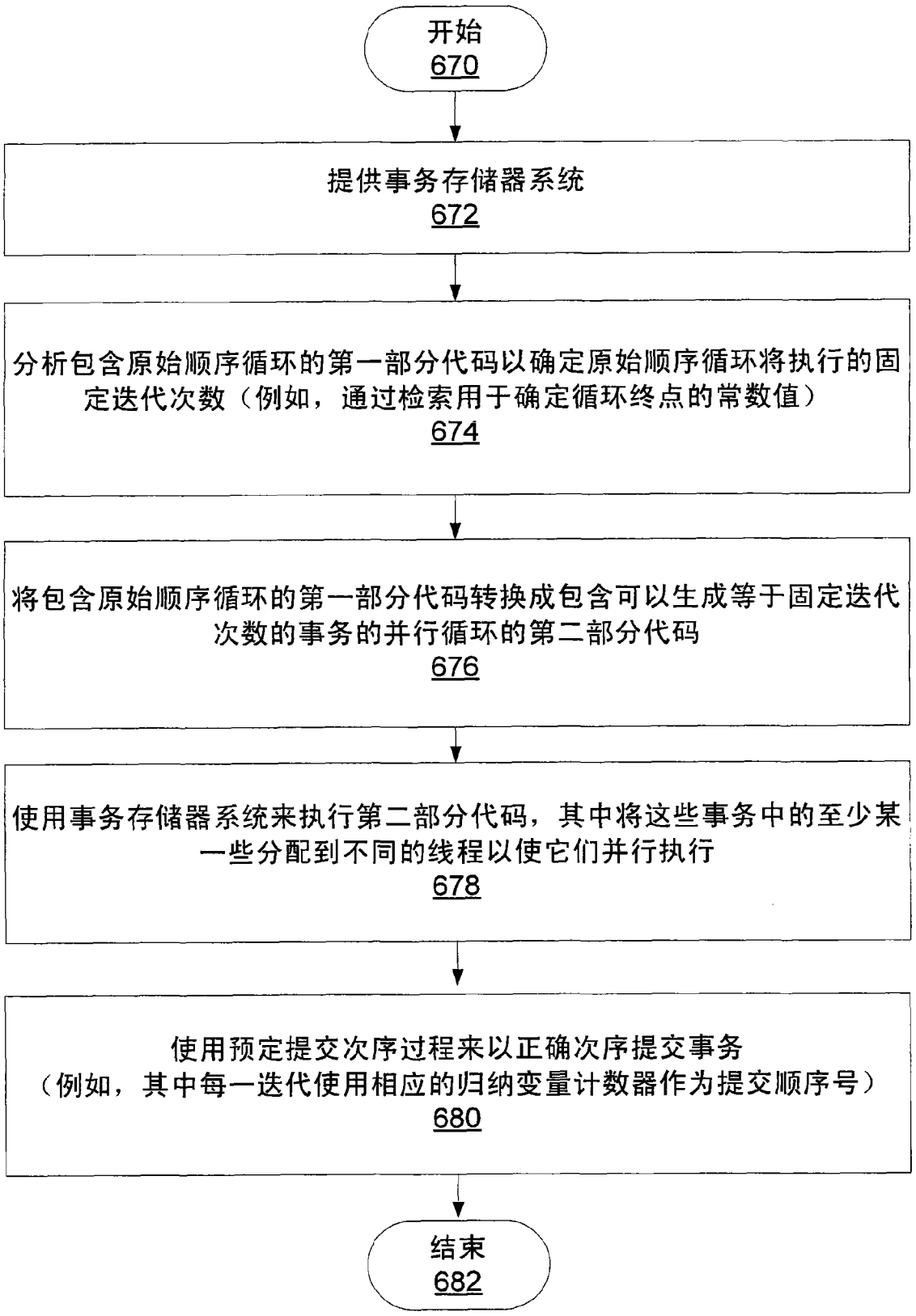

开始
670
提供事务存储器系统
672
分析包含原始顺序循环的第一部分代码以确定原始顺序循环将执行的固定迭代次数（例如，通过检索用于确定循环终点的常数值）
674
将包含原始顺序循环的第一部分代码转换成包含可以生成等于固定迭代次数的事务的并行循环的第二部分代码
676
使用事务存储器系统来执行第二部分代码，其中将这些事务中的至少某一些分配到不同的线程以使它们并行执行
678
使用预定提交次序过程来以正确次序提交事务
（例如，其中每一迭代使用相应的归纳变量计数器作为提交顺序号）
680
结束
682

图 19

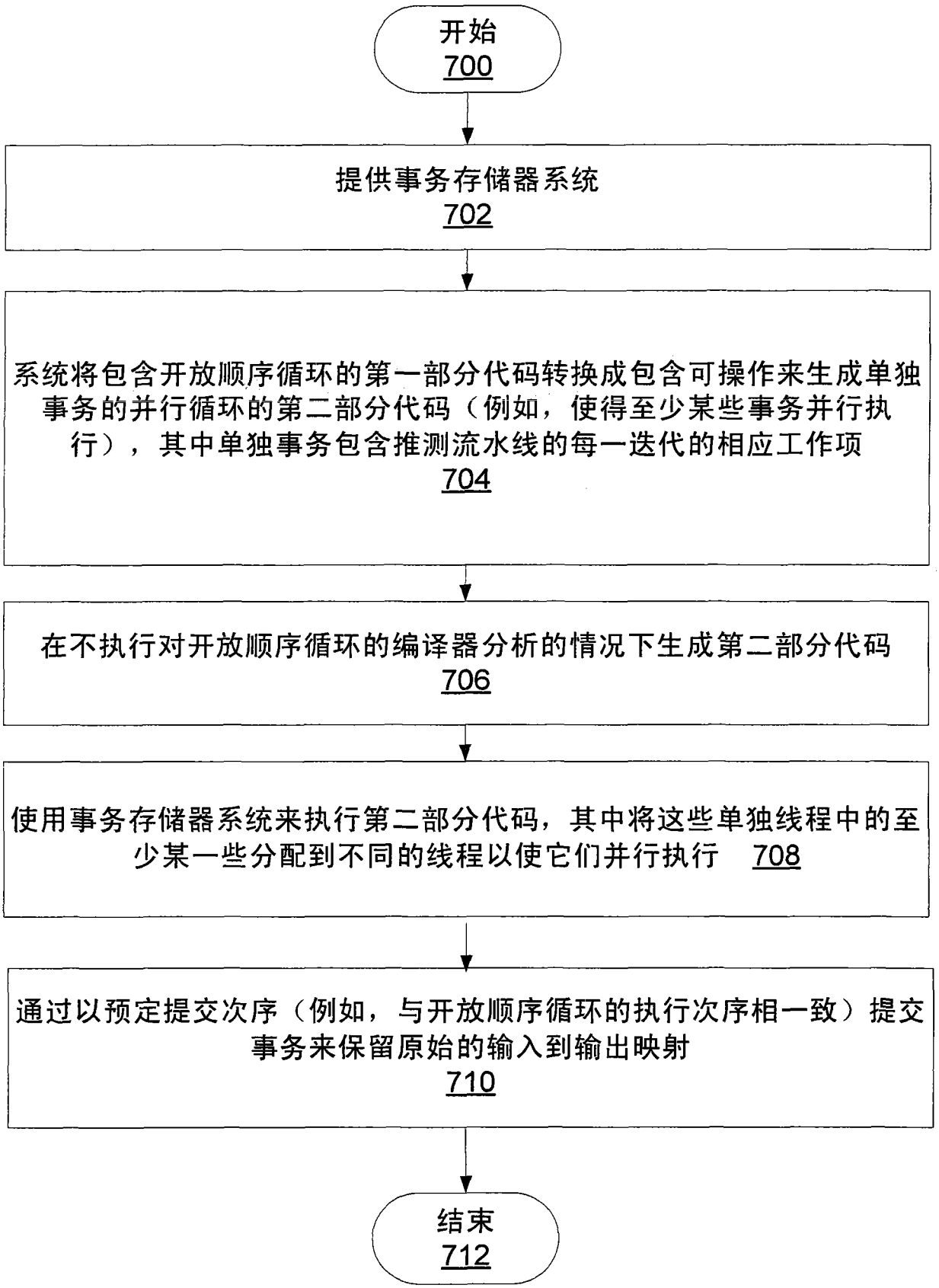

开始
700
提供事务存储器系统
702
系统将包含开放顺序循环的第一部分代码转换成包含可操作来生成单独事务的并行循环的第二部分代码（例如，使得至少某些事务并行执行），其中单独事务包含推测流水线的每一迭代的相应工作项
704
在不执行对开放顺序循环的编译器分析的情况下生成第二部分代码
706
使用事务存储器系统来执行第二部分代码，其中将这些单独线程中的至少某一些分配到不同的线程以使它们并行执行 708
通过以预定提交次序（例如，与开放顺序循环的执行次序相一致）提交事务来保留原始的输入到输出映射
710
结束
712

图 20

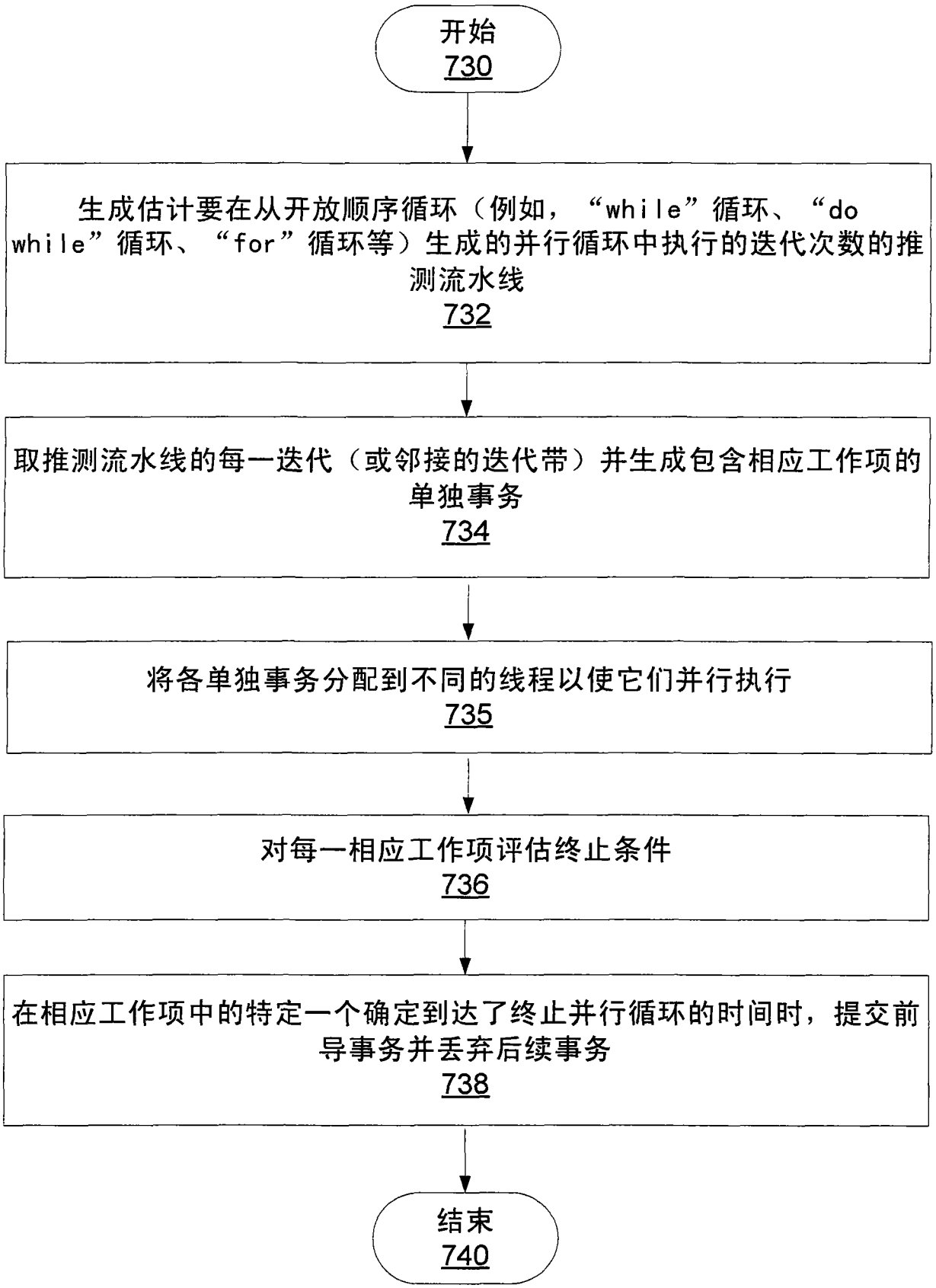

开始
730
生成估计要在从开放顺序循环（例如，“while”循环、“do while”循环、“for”循环等）生成的并行循环中执行的迭代次数的推测流水线
732
取推测流水线的每一迭代（或邻接的迭代带）并生成包含相应工作项的单独事务
734
将各单独事务分配到不同的线程以使它们并行执行
735
对每一相应工作项评估终止条件
736
在相应工作项中的特定一个确定到达了终止并行循环的时间时，提交前导事务并丢弃后续事务
738
结束
740

图 21

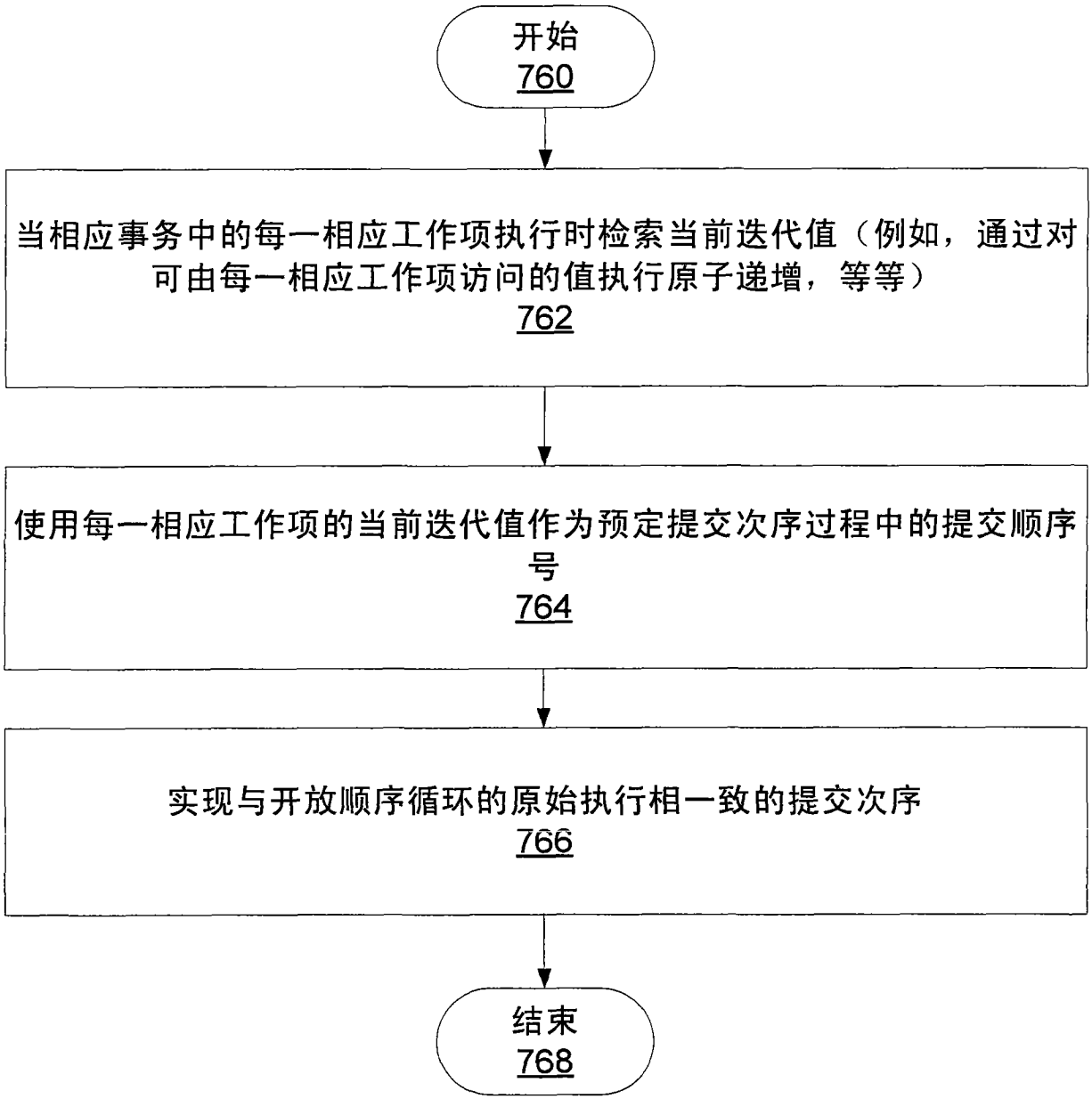

开始
760
当相应事务中的每一相应工作项执行时检索当前迭代值（例如，通过对可由每一相应工作项访问的值执行原子递增，等等）
762
使用每一相应工作项的当前迭代值作为预定提交次序过程中的提交顺序号
764
实现与开放顺序循环的原始执行相一致的提交次序
766
结束
768

图 22

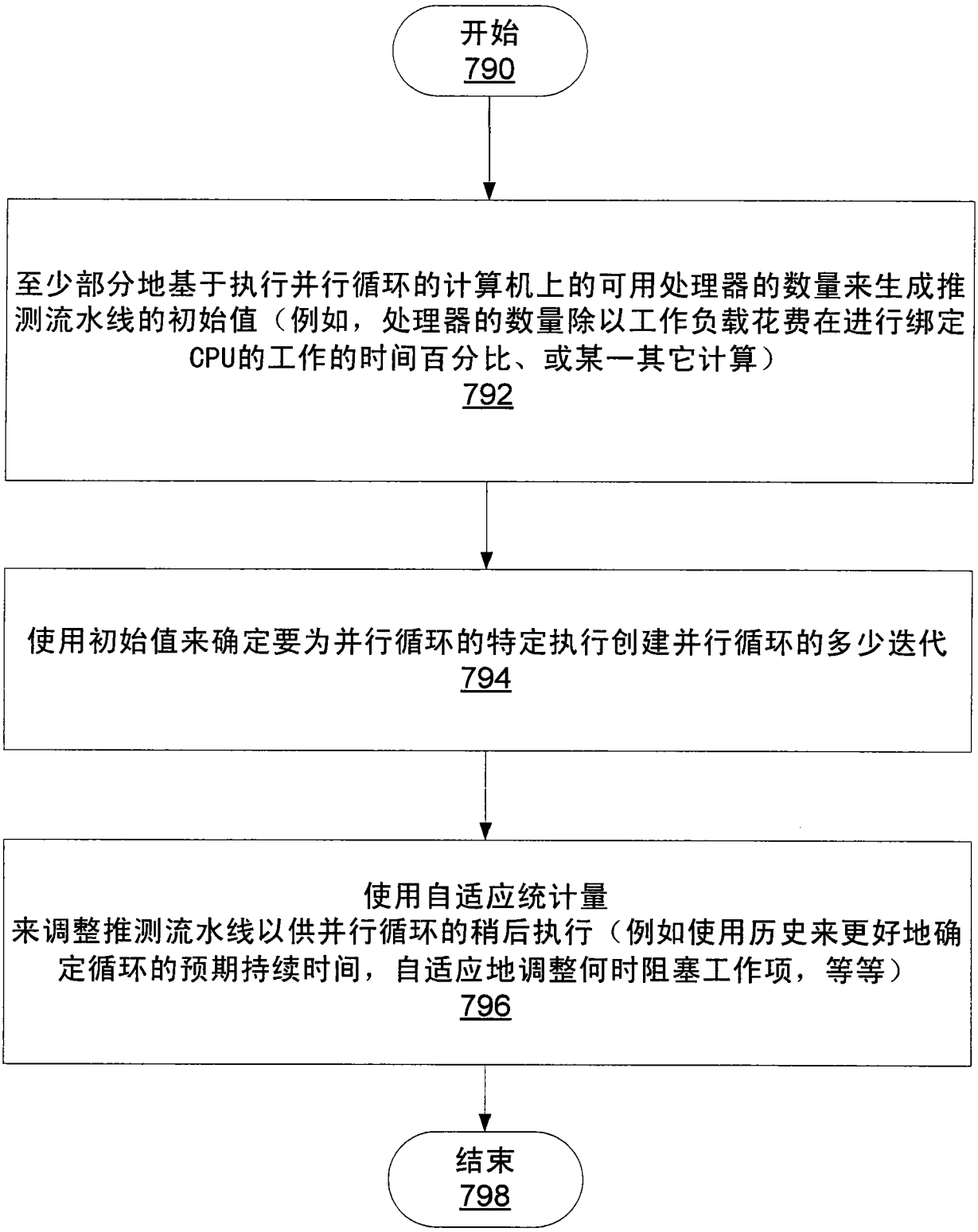

开始
790
至少部分地基于执行并行循环的计算机上的可用处理器的数量来生成推测流水线的初始值（例如，处理器的数量除以工作负载花费在进行绑定CPU的工作的时间百分比、或某一其它计算）
792
使用初始值来确定要为并行循环的特定执行创建并行循环的多少迭代
794
使用自适应统计量
来调整推测流水线以供并行循环的稍后执行（例如使用历史来更好地确定循环的预期持续时间，自适应地调整何时阻塞工作项，等等）
796
结束
798

图 23

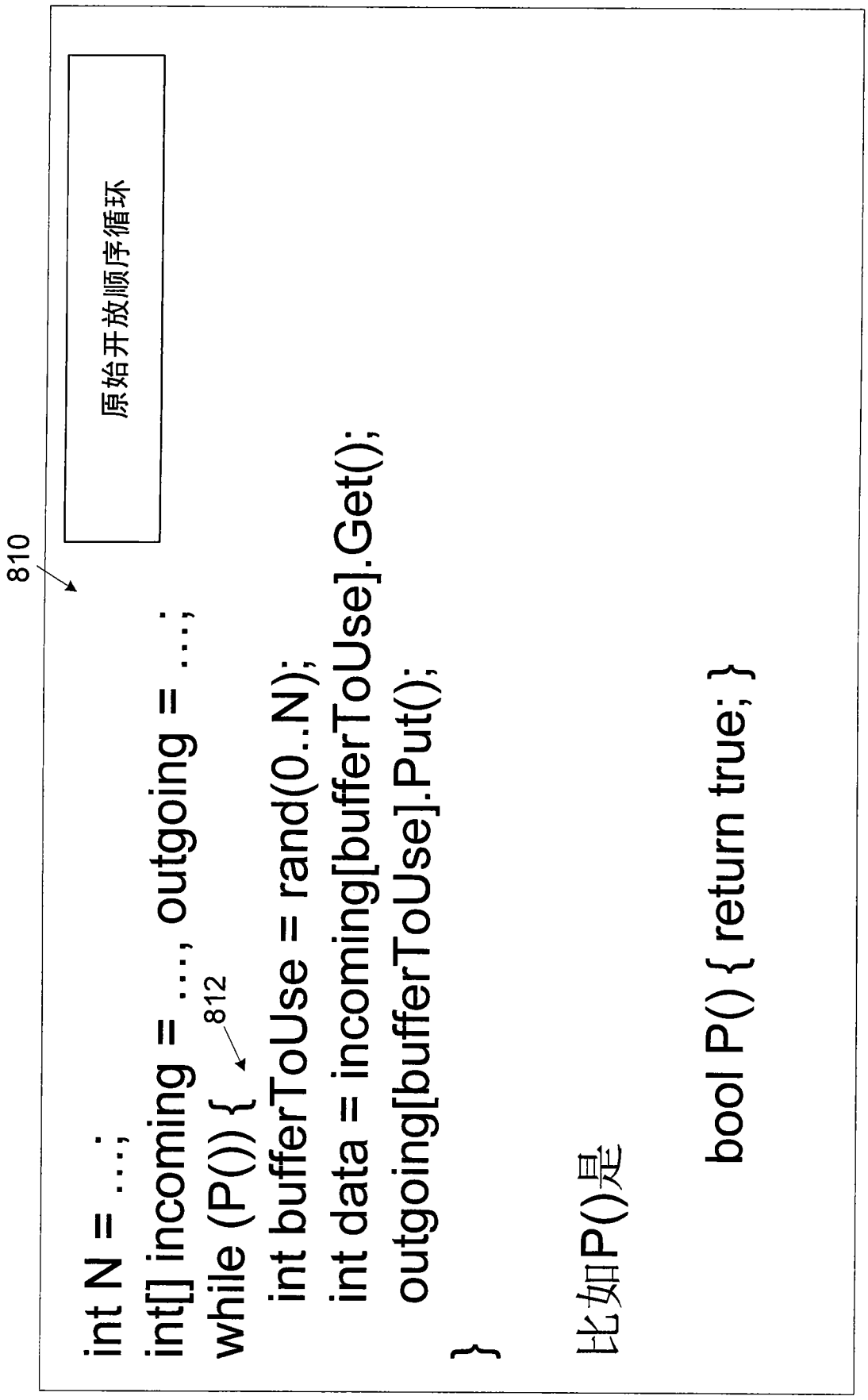

原始开放顺序循环
810
int N = ...;
int[] incoming = ..., outgoing = ...;
while (P()) {
812
int bufferToUse = rand(0..N);
int data = incoming[bufferToUse].Get();
outgoing[bufferToUse].Put();
}
比如P()是
bool P() { return true; }

图 24A

820

并行循环

```
// 在迭代之间共享
int currentIteration = 0;
int loopTerminated = 0;
TXCommitOrderContext ctx = ...; // 允许确立前导事务/后续事务关系
...
while (!loopTerminated) {                    822
    int myIteration = Interlocked.Increment(&currentIteration);
    TX tx(..., OrderedCommitContext, myIteration);
    if (IP()) {
        loopTerminated = 1;
        CommitAndAbortSuccessors();
        Break;
    }
    ... 如上所示地执行一个（事务化）循环迭代 ...
    ... 提交/重试/异常/其它 ...
}
```

图 24B