(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 104620223 B
(45)授权公告日 2018.11.16

(21)申请号 201380043004.1

(22)申请日 2013.06.10

(65)同一申请的已公布的文献号
申请公布号 CN 104620223 A

(43)申请公布日 2015.05.13

(30)优先权数据
61/660,526 2012.06.15 US

(85)PCT国际申请进入国家阶段日
2015.02.12

(86)PCT国际申请的申请数据
PCT/US2013/045020 2013.06.10

(87)PCT国际申请的公布数据
WO2013/188311 EN 2013.12.19

(73)专利权人 英特尔公司
地址 美国加利福尼亚州

(72)发明人 M·阿布达拉

(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 黄嵩泉

(51)Int.Cl.
G06F 9/46(2006.01)
G06F 9/38(2006.01)

(56)对比文件
US 2010/0281220 A1,2010.11.04,
US 2002/0199063 A1,2002.12.26,
CN 1728087 A,2006.02.01,
CN 1241752 A,2000.01.19,
US 2005/0071573 A1,2005.03.31,
CN 1383511 A,2002.12.04,

审查员 刘可欣

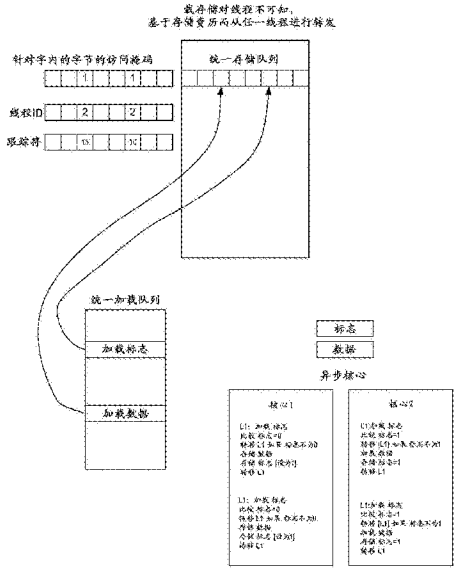
权利要求书2页 说明书20页 附图24页

(54)发明名称

对于基于存储资历实现从不同线程进行转发的线程不可知的加载存储缓冲器

(57)摘要

在处理器中,一种用于在使用共享存储器资源的存储器一致性模型中的乱序加载的线程不可知统一存储队列和统一加载队列方法。该方法包括:实现能够由多个异步核心访问的存储器资源,其中所述多个核心共享统一存储队列和统一加载队列;以及实现访问掩码,该访问掩码通过跟踪高速缓存的哪些字经由加载被访问来运行,其中高速缓存线包括存储器资源,其中加载在访问高速缓存线的字时设置访问掩码内的掩码位,并且其中掩码位阻止从多个核心的来自其他加载的访问。该方法还包括:在执行从多个核心到高速缓存线的后续存储时检查访问掩码,其中来自不同线程的存储能够在仍保持按序存储器一致性语义的同时向不同线程的加载进行转发;以及当到高速缓存线的部分的后续存储在访问掩码中见到来自加载的在先标记时引发缺失预测,其中后续存储将通过使用跟踪符寄存器和线程ID寄存器来用信号发送对应于该加载的加载队列条目。



CN 104620223 B

1.在处理器中，一种用于在使用共享存储器资源的存储器一致性模型中的乱序加载的线程不可知统一存储队列和统一加载队列方法，包括：

定义能够由多个异步核心访问的存储器资源，其中所述多个异步核心共享统一存储队列和统一加载队列；

实现访问掩码，所述访问掩码用于跟踪高速缓存线的哪些字经由加载被访问，其中所述访问掩码包括一组位且所述一组位中的每个位对应于所述高速缓存线中的一单独字，其中高速缓存线包括所述存储器资源且所述高速缓存线包括多个字，其中所述加载在访问所述高速缓存线的字时设置所述访问掩码内的掩码位，并且其中所述掩码位阻止从所述多个异步核心的来自其他加载的访问，其中所述掩码位在所述一组位中并且对应于所述加载所访问的字；

在执行从所述多个异步核心到所述高速缓存线的后续存储时检查所述访问掩码，其中来自不同线程的存储能够在仍保持按序存储器一致性语义的同时向不同线程的加载进行转发；以及

当到所述高速缓存线的与所述加载对应的字的后续存储在所述访问掩码中检测到来自加载的在先掩码位时引发缺失预测，其中所述后续存储通过使用跟踪符寄存器和线程ID寄存器来用信号发送对应于所述加载的加载队列条目。

2.根据权利要求1所述的方法，其中所述统一存储队列和所述统一加载队列对于能够访问它的所述多个线程中的任何线程而言是不可知的。

3.根据权利要求1所述的方法，其中所述统一存储队列和所述统一加载队列对于访问它的所述多个异步核心中的任何核心而言是不可知的。

4.根据权利要求1所述的方法，其中线程用于基于存储资历而从彼此进行转发。

5.根据权利要求4所述的方法，其中对于给定线程，如果不存在来自所述给定线程内的转发，则加载用于接收来自不同线程内的资深存储的转发。

6.根据权利要求4所述的方法，其中当同一线程中的给定存储之前的所有加载和存储都已经被执行时，所述给定存储是资深的。

7.根据权利要求1所述的方法，其中所述统一加载队列和所述统一存储队列包括单个存储器数据结构。

8.根据权利要求1所述的方法，其中所述存储器资源能够由多个线程访问。

9.根据权利要求1所述的方法，其中一旦加载正在从高速缓存线的部分进行读取，则所述加载设置与所述高速缓存线的所述部分相对应的相应访问掩码位。

10.根据权利要求9所述的方法，其中当所述加载退出时，所述相应访问掩码位被清除。

11.根据权利要求1所述的方法，还包括实现加载队列条目引用寄存器以跟踪加载队列条目引用，其中当存储将数据保存至所述高速缓存线的与所述加载队列条目引用寄存器中的匹配相对应的部分时，引发对应的加载队列条目进行缺失预测。

12.根据权利要求1所述的方法，其中所述后续存储通过使用所述跟踪符寄存器用信号发送对应于所述加载的加载队列条目，并且引发所述加载连同从属于所述加载的指令一起进行缺失预测。

13.根据权利要求1所述的方法，其中所述共享存储器资源包括标志资源和数据资源。

14.一种微处理器，包括：

多个核心和加载存储缓冲器，其中所述加载存储缓冲器通过以下方式实现用于在使用共享存储器资源的存储器一致性模型中的乱序加载的方法：

定义能够由多个异步核心访问的存储器资源，其中所述多个异步核心共享统一存储队列和统一加载队列；

实现访问掩码，所述访问掩码用于跟踪高速缓存线的哪些字经由加载被访问，其中所述访问掩码包括一组位且所述一组位中的每个位对应于所述高速缓存线中的一单独字，其中高速缓存线包括所述存储器资源且所述高速缓存线包括多个字，其中所述加载在访问所述高速缓存线的字时设置所述访问掩码内的掩码位，并且其中所述掩码位阻止从所述多个异步核心的来自其他加载的访问，其中所述掩码位在所述一组位中并且对应于所述加载所访问的字；

在执行从所述多个异步核心到所述高速缓存线的后续存储时检查所述访问掩码，其中来自不同线程的存储能够在仍保持按序存储器一致性语义的同时向不同线程的加载进行转发；以及

当到所述高速缓存线的与所述加载对应的字的后续存储在所述访问掩码中检测到来自所述加载的在先掩码位时引发缺失预测，其中所述后续存储通过使用跟踪符寄存器和线程ID寄存器来用信号发送对应于所述加载的加载队列条目。

15.根据权利要求14所述的微处理器，其中所述统一存储队列和所述统一加载队列对于访问它的所述多个线程中的任何线程而言是不可知的。

16.根据权利要求14所述的微处理器，其中所述统一存储队列和所述统一加载队列对于访问它的所述多个异步核心中的任何核心而言是不可知的。

17.根据权利要求14所述的微处理器，其中线程用于基于存储资历而从彼此进行转发。

18.根据权利要求17所述的微处理器，其中对于给定线程，如果不存在来自所述给定线程内的转发，则加载用于接收来自不同线程内的资深存储的转发。

19.根据权利要求17所述的微处理器，其中当同一线程中的给定存储之前的所有加载和存储都已经被执行时，所述给定存储是资深的。

20.根据权利要求14所述的微处理器，其中所述存储器资源能够由多个线程访问。

对于基于存储资历实现从不同线程进行转发的线程不可知的加载存储缓冲器

[0001] 本申请要求2012年6月15日由Mohammad A.Abdallah提交的题为“A LOAD STORE BUFFER AGNOSTIC TO THREADS IMPLEMENTING FORWARDING FROM DIFFERENT THREADS BASED ON STORE SENIORITY”的共同待决共同转让的美国临时专利申请序号61/660,526的权益，并且该申请被整体地并入到本文中。

技术领域

[0002] 本发明总体涉及数字计算机系统，更具体地涉及用于选择指令(包括指令序列)的系统和方法。

背景技术

[0003] 要求处理器来处理相关或完全独立的多个任务。此类处理器的内部状态通常由在程序执行的每个特定时刻可能保持不同的值的寄存器组成。在程序执行的每个时刻，将内部状态图像称为处理器的架构状态。

[0004] 当架构代码执行切换成运行另一功能(例如，另一线程、进程或程序)时，然后必须保存机器/处理器的状态，使得新功能可以利用内部寄存器以构建其新状态。一旦新功能已终止，则可以丢弃其状态，并且先前情境的状态将被恢复且该执行又继续。此类切换过程称为情境切换，并且通常包括10个或数百个循环，尤其是在采用大量寄存器(例如，64、128、256)和/或乱序执行的现代架构的情况下。

[0005] 在线程感知硬件架构中，硬件支持用于有限数目的硬件支持线程的多个情境状态是正常的。在这种情况下，硬件复制用于每个所支持线程的所有架构状态元素。这消除了在执行新线程时的情境切换的需要。然而，这仍具有多个缺点，即复制用于用硬件中所支持的每个附加线程的所有架构状态元素(即，寄存器)的面积、功率和复杂性。另外，如果软件线程的数目超过明确支持的硬件线程的数目，则仍必须执行情境切换。

[0006] 这变得常见，因为在要求大量线程的细粒度基础上需要并行性。具有复制情境状态硬件存储的硬件线程感知架构无助于非线程软件代码，并且仅仅减少了用于被线程化的软件的情境切换的次数。然而，那些线程通常是针对粗粒并行性而构造的，并且导致用于发起和同步、离开细粒并行性的沉重软件开销，诸如函数调用和回路并行执行，而没有高效的线程化发起/自动生成。此类所述开销伴随着针对非明确/容易并行化/线程化软件代码而使用现有技术编译程序或用户并行化技术进行此类代码的自动并行化的困难。

发明内容

[0007] 在一个实施例中，将本发明实现为一种用于在使用共享存储器资源的存储器一致性模型中的乱序加载的线程不可知统一存储队列和统一加载队列方法。该方法包括：实现能够由多个异步核心访问的存储器资源，其中所述多个核心共享统一存储队列和统一加载队列；以及实现访问掩码，该访问掩码通过跟踪高速缓存的哪些字经由加载被访问来运行，

其中高速缓存线包括存储器资源，其中加载在访问高速缓存线的字时设置访问掩码内的掩码位，并且其中掩码位阻止从多个核心的来自其他加载的访问。该方法还包括：在执行从多个核心到高速缓存线的后续存储时检查访问掩码，其中来自不同线程的存储能够在仍保持按序存储器一致性语义的同时向不同线程的加载进行转发；以及当到高速缓存线的部分的后续存储在访问掩码中见到来自加载的在先标记时导致缺失预测，其中后续存储将通过使用跟踪符寄存器和线程ID寄存器来用信号发送对应于该加载的加载队列条目。

[0008] 前述是概要且因此必要地包含细节的简化、一般化以及省略；因此，本领域的技术人员将认识到的是该概要仅仅是说明性的且并不意图以任何方式是限制性的。单独地由权利要求定义的本发明的其他方面、发明特征以及优点在下面阐述的非限制性详细描述中将变得显而易见。

附图说明

[0009] 在附图中以示例的方式而不是以限制的方式举例说明本发明，并且在附图中相同的附图标记指示类似元件。

[0010] 图1示出了根据本发明的一个实施例的加载队列和存储队列。

[0011] 图2示出了根据本发明的一个实施例的加载和存储指令分离的第一图。

[0012] 图3示出了根据本发明的一个实施例的加载和存储指令分离的第二图。

[0013] 图4示出了根据本发明的一个实施例的过程的步骤的流程图，其中图示出用于实现从由加载存储重新排序和优化所引发的推测性转发缺失预测/错误中恢复的规则。

[0014] 图5示出了图示出根据本发明的一个实施例的用处理器的加载队列和存储队列资源来实现过程300的规则的方式的图。

[0015] 图6示出了图示出根据本发明的一个实施例的用处理器的加载队列和存储队列资源来实现过程300的规则的方式的另一图。

[0016] 图7示出了图示出根据本发明的一个实施例的用处理器的加载队列和存储队列资源来实现过程300的规则的方式的另一图。

[0017] 图8示出了根据本发明的一个实施例的其中在加载之后分派存储的分派功能的概述的过程的流程图。

[0018] 图9示出了根据本发明的一个实施例的其中在存储之后分派存储的分派功能的概述的过程的流程图。

[0019] 图10示出了根据本发明的一个实施例的统一加载队列的图。

[0020] 图11示出了示出根据本发明的一个实施例的滑动加载分派窗口的统一加载队列。

[0021] 图12示出了根据本发明的一个实施例的分布式加载队列。

[0022] 图13示出了根据本发明的一个实施例的具有按序连续性的分布式加载队列。

[0023] 图14示出了根据本发明的一个实施例的用于多核处理器的分段存储器子系统的图。

[0024] 图15示出了如何通过本发明的实施例来处理加载和存储的图。

[0025] 图16示出了根据本发明的一个实施例的存储过滤算法的图。

[0026] 图17示出了根据本发明的一个实施例的在组成按序从存储器进行读取的加载的存储器一致性模型中具有乱序加载的信号量实现方式。

[0027] 图18示出了根据本发明的一个实施例的到通过使用基于锁的模型和基于事务的模型两者而组成按序的用于存储器的加载读取的存储器一致性模型中的乱序加载。

[0028] 图19示出了根据本发明的一个实施例的多核分段存储器子系统的多个核心。

[0029] 图20示出了根据本发明的一个实施例的访问统一存储队列的异步核心的图，其中存储可以基于存储资历而从任一线程获得。

[0030] 图21示出了描绘根据本发明的一个实施例的其中存储相比于其他线程中的对应存储而言具有资历的功能的图。

[0031] 图22示出了根据本发明的一个实施例的非消歧乱序加载存储队列退出实现方式。

[0032] 图23示出了根据本发明的一个实施例的非消歧乱序加载存储队列重新排序实现方式的重新排序实现方式。

[0033] 图24示出了根据本发明的一个实施例的指令序列(例如，踪迹)重新排序的推测性执行实现方式。

[0034] 图25示出了根据本发明的一个实施例的示例性微处理器管线的图。

具体实施方式

[0035] 虽然已结合一个实施例描述了本发明，但本发明并不意图局限于本文中所阐述的特定形式。相反地，其意图涵盖可以适当地包括在由所附权利要求定义的本发明的范围内的此类替换、修改以及等同物。

[0036] 在以下详细描述中，已阐述了许多特定细节，诸如特定方法顺序、结构、元件以及连接。然而应理解的是不需要利用这些及其他特定细节来实施本发明的实施例。在其他情况下，已省略众所周知的结构、元件或连接或未特别详细地描述以避免不必要地使本描述含糊。

[0037] 在本说明书内对“一个实施例”或“实施例”的引用意图指示在本发明的至少一个实施例中包括结合该实施例所述的特定特征、结构或特性。短语“在一个实施例中”在本说明书内的各种位置上的出现并不一定全部参考同一实施例，也不是其他实施例的互斥的单独或替换实施例。此外，描述了某些实施例可展示出而其他的没有的各种特征。同样地，描述了对于某些实施例而言可能是要求但其他实施例并非如此的各种要求。

[0038] 随后的详细描述的某些部分是在对计算机存储器内的数据位的操作的程序、步骤、逻辑块、处理及其他符号表示方面提出的。这些算法描述和表示是数据处理领域的技术人员用来最有效地向本领域的其他人传达其工作的实质的手段。程序、计算机执行步骤、逻辑块、过程等在这里且一般地被设想为是导致期望结果的步骤或指令的自相一致序列。该步骤是要求物理量的物理操作的那些。通常但不一定，这些量采取计算机可读存储介质的电或磁信号的形式，并且能够在计算机系统中被存储、传输、组合、比较以及操纵。主要由于一般使用的原因，已证明将这些信号称为位、值、元素、符号、字符、项、数字等有时是方便的。

[0039] 然而，应记住的是所有这些和类似术语将与适当的物理量相关联且仅仅是应用于这些量的方便标记。如从以下讨论显而易见的那样，除非另外具体地说，应认识到的是遍及本发明，利用诸如“处理”或“访问”或“写”或“存储”或“复制”等等术语的讨论指的是计算机系统或类似电子计算设备的动作或过程，其对在计算机系统的寄存器和存储器及其他计算

机可读介质内表示为物理(例如,电子)量的数据进行操纵并变换成在计算机系统存储器或寄存器或其他此类信息存储、传输或显示设备内同样地表示为物理量的其他数据。

[0040] 本发明的实施例实现了乱序指令调度过程,其中允许输入指令序列内的指令在处理器资源可用于执行它们时就立即乱序发出。本发明的实施例能够确保外部代理见到指令按序执行(例如,存储器一致性规则/模型)。

[0041] 确保指令对外部代理可见地按序执行,从而确保无错误程序执行。本发明的实施例确保处理器的存储器分级结构(例如,L1高速缓存、L2高速缓存、系统存储器等)见到指令的一致按序执行。

[0042] 图1示出了根据本发明的一个实施例的加载队列和存储队列。图1还示出了输入指令序列。如上所述,处理器的存储器分级结构(例如,L1高速缓存、L2高速缓存、系统存储器等)见到指令的一致性按序执行。其后常常称为加载/存储队列的加载队列和存储队列可以用来保持按序执行的语义。

[0043] 另外,加载和存储的乱序执行引发推测性执行。当执行推测性执行时,机器需要识别推测性错误。在图1的实施例中,加载/存储队列提供了一种用于实现从由加载存储重新排序和优化引发的推测性转发或缺失预测/错误中恢复的系统。加载/存储队列包括允许从作为转发、分支和错误的结果而由加载存储重新排序/优化引发的推测性错误中恢复的硬件支持。为了允许机器从推测性错误中恢复,将推测性执行的结果保持在加载队列和存储队列中。加载队列和存储队列保持推测性执行的结果直至可以修正错误且可以使存储结果退出(retire)至存储器为止。加载队列和存储队列的推测性执行内容对于外部代理而言是不可见的。相对于可见性,存储需要按序退出至存储器。

[0044] 图2示出了根据本发明的一个实施例的加载和存储指令分离的第一图。本发明的一个特征是这样的事实,即加载被分离成两个宏指令,第一个进行地址计算并取读到临时位置(加载存储队列),并且第二个是存储器地址内容(数据)到寄存器或ALU目的地的加载。应注意的是虽然在将加载和存储指令分离成两个相应宏指令并将其重新排序的情境内描述了本发明的实施例,但可以通过在微代码情境内将加载和存储指令分离成两个相应微指令并将其重新排序来实现该方法和系统。

[0045] 该功能对于存储而言是相同的。存储也被分离成两个宏指令。第一指令是存储地址和取读,第二指令是该地址处的数据的存储。存储和两个指令的分离遵循与下面针对加载所述的相同规则。

[0046] 加载到两个指令的分离允许运行时间优化器在给定指令序列内早得多地调度地址计算和取读指令。这允许通过将数据预取读到与高速缓存分级结构分离的临时缓冲器中而更容易地从存储器缺失恢复。使用临时缓冲器以便保证关于LA/SA和LD/SD之间的一一对应的经预取读的数据的可用性。如果存在与处于加载地址与加载数据之间的窗口中的在先存储的别名使用(例如,如果从先前的存储检测到转发情况)或者如果地址计算存在任何错误问题(例如,页面错误),则对应的加载数据指令可以重新发出。另外,将加载分离成两个指令还可以包括将信息复制到两个指令中。此类信息可以是地址信息、源信息、其他附加标识符等。此复制允许在不存在LA/SA的情况下进行两个指令的LD/SD的独立分派。

[0047] 加载地址和取读指令可以在不等待加载数据返回的情况下从实际机器退出窗口退出,由此甚至在高速缓存缺失该地址(例如,在段落开头处提到的加载地址)的情况下允

许机器向前进行。例如，在高速缓存缺失该地址(例如，地址X)时，机器可能停顿达几百个循环等待从存储器分级结构取读数据。通过在不等待加载数据返回的情况下使加载地址和取读指令从实际机器退出窗口退出，机器仍可以向前进行。

[0048] 应注意的是指令的分离使得本发明的实施例的关键优点能够将LA/SA指令重新排序得早于且更加远离LD/SD指令序列以使得能够实现加载和存储的较早分派和执行。

[0049] 图3示出了根据本发明的一个实施例的加载和存储指令分离的第二图。图2实施例示出了如何使用复制特征以便规定加载指令的分离。在本实施例中，将加载复制成两个宏指令，第一个是地址计算和取读到临时位置(加载存储队列)中，并且第二个是存储器地址内容(数据)到寄存器或ALU目的地中的加载。应注意的是虽然在将加载和存储指令复制成两个相应宏指令并将其重新排序的情境下描述本发明的实施例，但可以通过在微代码情境下将加载和存储指令复制成两个相应微指令并将其重新排序来实现该方法和系统。

[0050] 现在描述根据本发明的一个实施例的指令分离功能的更详细描述。在一个实施例中，指令集不具有到LA、SA、LD或SD的直接模拟指令。在此类实施例中，用指令前缀LAF、SAF、LASAF和伴随后缀指令的组合来实现这些概念。并且粗略地确实映射到LA上的指令集具有LAD且SA具有SAD，并且可以实现组合LADSAD。还可以将这些概念实现为微代码内的微指令。

[0051] a)可以将在这里被定义为LAF—前缀+后缀指令的描述为‘LD’。

[0052] b)可以将在这里被定义为SAF—前缀+后缀指令的描述为‘SD’。

[0053] c)可以将在这里被定义为LAD指令的描述为‘LA’。

[0054] d)可以将在这里被定义为SAD指令的描述为‘SA’。

[0055] e)此外，我们具有LASAF指令和LADSAD指令。这些指令包括可以用来实现信号量(锁定原子)操作的组合LAF/SAF—前缀+后缀指令。还可能定义组合LAD—SAD指令以再次地预取读存储器操作数，结果产生硬件的复杂化。

[0056] LAD代表‘LA-去融合’

[0057] LAD指令发起到执行管线中的数据预取读。其与正常预取读的不同之处在于其直接地加载到执行管线中，可承担比一级高速缓存更低的执行时延。在一个实施例中，通过将固定储存装置用于可以使用在LA—LD对之间的ID链路(例如QID数)来标记的LA—LD对来实现此功能。LAD指令计算有效存储器地址(例如，来自潜在复杂规范)，指定操作数大小(字节、半字、字、双字或更大)；发起存储器引用；通过TLB和高速缓存分级结构。记录例外(翻页缺失、特权、保护)以在LAF+后缀执行时报告，或者替换实现方式可以使Q条目取消/无效，迫使LAF+后缀指令重新执行，并且其将出现例外。

[0058] LAD指令具有一般格式和操作数：

[0059] LAD(os)QID,M[EA]

[0060] • EA—是有效地址规范，其可以是基础寄存器、索引寄存器、移位因数和/或索引偏移的组合。例如M[B,RI,sf,offset]

[0061] • os—是要读取的字节数的指示

[0062] • QID—将被用于存储器引用操作的加载存储器QID。其还用来链接LAD的操作和后续LAF前缀的指令。QID在1至N范围内，N是实现特定值。期望值是31、63、127。QID＝0被预留给不具有前LAD的LAF指令的特殊情况。QID＝0始终立即被LAF使用，因为其不可用于LAD。

[0063] LAF代表‘LA-融合’。

[0064] LAF是指令前缀，意味着其必须与后缀指令直接耦合(或融合)。后缀指令可以是独立的。后缀指令可以是具有至少一个源寄存器的任何指令。作为前缀的LAF必须被耦合。LAF前缀改变后缀指令的性质。其寄存器操作数中的一个或多个被前缀重新定义为存储器队列标识符(QID)。此外，被关联为源自于寄存器的数据现在源自于存储器队列。

[0065] LAF-前缀+后缀指令可具有也可不具有前LAD指令。如果QID＝＝0，则LAF没有前LAD。如果QID!＝0，则LAF具有或曾具有前LAD。当意图创建加载指令到LA和LD的分离时，则LAF将具有QID!＝0且将用在LAF前面的同一QID将LAD实例化(例如，这基本上创建LA/LD对)。

[0066] 当LAF/后缀执行且具有QID＝＝0时，使用存储器队列的0条目来完成‘LA’操作、存储器读、将数据升级到存储器队列，并且然后通过将数据加载到后缀指令源来完成，将所应用的操作与潜在其他源组合，并将结果写入一个或多个后缀指令目的地寄存器。另一方面，如果QID!＝0，则针对匹配QID查阅存储器队列(查找)，如果存在的话，则从存储器队列读取数据，并且应用后缀指令的操作，并将结果写入目的地寄存器。如果匹配QID是有效但不完整的，则数据停顿直至数据可用为止。如果QID是无效的，则LAF具有足够的信息(地址和数据操作操作数大小)以重新开始该操作。匹配的QID可能由于多种原因而不存在，其中的某些是：

[0067] a)前LAD从未执行、坏编码或其他原因。

[0068] b)LAF与前LAD之间的执行流程被例外或中断所打断

[0069] c)居间的存储操作与LAD的地址别名使用且使其数据完整性无效。

[0070] 在这些情况中的任何一个中，LAF前缀+后缀具有足以重复LAD(LA)操作的信息。这种能力使得我们的LAD指令成为暗示。LAD不必成功地执行或者在这方面甚至超过作为用于正确代码的NOP以使用它之外而实现。

[0071] 具有后缀指令的LAF指令的一般格式和操作数是：

[0072] LAF M[ea]

[0073] SUFFIX(os)Rt,QID,….

[0074] LAF指令借用是操作数大小、QID，并且来自后缀指令的编码。如果后缀是SIMD，则其还从后缀借用操作的SIMD宽度。始终在后缀指令的源寄存器规范字段中的一个中对QID进行编码。在SMI的特定实现方式中，这始终是位23:18，但情况不必如此。

[0075] SAD代表‘SA-去融合’

[0076] SAD是到仅用于存储的LAD的并行指令。其也预取读数据，将数据送到高速缓存以用于修改。此外，其创建存储器存储队列条目。主要SAD具有2个主要用途：

[0077] a)作为预取读，读取以用于数据的修改

[0078] b)保持正确的存储器排序并暴露并处理在存储(写)之前促进加载(读)之后的潜在读后写危险

[0079] SAD是暗示指令。SAD指令计算有效存储器地址(潜在地来自复杂规范)，指定凑作数大小(字节、半字、字、双字等)；发起存储器引用；通过TLB、高速缓存/存储器分级结构。在SAF+后缀执行时记录例外(翻页缺失、特权、保护)以重新执行且其出现例外。

[0080] SAD指令具有一般格式和操作数：

[0081] SAD(os)M[ea],QID

[0082] • Ea—是有效地址规范，其可以是基础寄存器、索引寄存器、移位因数和/或索引偏移的组合。例如M[B,RI,sf,offset]

[0083] • Os—是要写入Ea的字节数的指示

[0084] • QID—将被用于存储器引用操作的存储存储器QID。其还用来链接SAD的操作和后续SAF前缀指令。QID在1至N范围内，N是实现特定值。期望值是31、63、127。QID＝0被预留给不具有前SAD的SAF指令的特殊情况。此QID始终立即被SAF使用。

[0085] SAF代表‘SA-融合’

[0086] SAF是LAF前缀的并行前缀，仅用于存储。作为前缀，其必须与后缀指令直接地耦合(或融合)。后缀指令可以是独立的。后缀指令可以是具有至少一个目标寄存器的任何指令。作为前缀的SAF耦必须被合。SAF改变后缀指令的性质：目的地寄存器操作数中的一个或多个，其正常地是到存储器存储队列标识符(QID)中的寄存器选择索引，并且操作从以寄存器为目标到以存储器(更确切地存储器队列条目)为目标。同样地，其将寄存器操作变成存储存储器操作。

[0087] SAF指令可具有也可不具有前SAD。如果QID＝＝0，则SAF没有前SAD。如果QID!＝0，则SAF具有或曾具有前SAD。当SAF/后缀执行时，如果QID＝＝0，则使用存储器队列的0条目来完成‘SA’操作，存储器写，将数据升级到存储器队列，并且然后通过存储由后缀指令源供应的数据来完成。另一方面，如果QID!＝0，则针对匹配QID查阅存储器队列(查找)，如果存在的话，数据将在应用后缀指令时被写入存储器队列条目。如果匹配QID是有效但不完整的，则数据停顿直至数据可用为止。如果QID是无效的，则SAF具有足够的信息(地址和数据操作操作数大小)以又继续该操作并完成存储器写操作。。匹配QID可能由于多种原因而不存在，其中的某些是：

[0088] a)前SAD从未执行、坏编码或其他原因。

[0089] b)SAF与前SAD之间的执行流程被例外或中断所打断

[0090] 在这些情况中的任何一个中，SAF前缀+后缀具有足以重复SAD(SA)操作的信息。这种能力使得我们的SAD指令称为暗示。SAD不必成功地执行或者在这方面甚至超过作为用于正确代码的NOP以使用它之外而实现。

[0091] LASAF是指令前缀

[0092] 作为前缀的LASAF修改与源和目的地具有相同寄存器的指令。LASAF将此类指令变成原子存储器引用读/写一次操作。使用来自加载存储器队列的一个和来自存储存储器队列的一个。不存在前LAD或SAD指令。

[0093] LASAF M[ea3]

[0094] ADD QID1,QID2,R1

[0095] LASAF创建加载和存储存储器队列两者中的QID条目。并且其将使用QID2来读取存储器[ea3]，添加R1并将结果存储在存储存储器QID1中，实行M[ea3]的原子读修改写。

[0096] SMI实现(如果我们将实现它)将要求QID1＝＝QID2＝＝0。但是我们不想只局限于该实现方式。

[0097] 我们可以具有LASAD指令，认为是这样，但我们必须始终向存储器队列中进行嗅探以完成它。使命中的嗅探上的两个条目都无效。并且在LASAF上重新执行加载/存储。

[0098] 示例性使用：

[0099] A.远远在数据的使用之前在促进加载之后保存寄存器资源。

[0100] 假设原始代码为

[0101] LDR R1,M[ea1]

[0102] ADD32 Rt,R1,R2

[0103] 为了隐藏存储器访问时延,我们希望在执行流程中在R1数据(ADD)的使用之上尽快地促进LDR。

[0104] LDR R1,M[ea1]

[0105] ...许多指令

[0106] ADD32 Rt,R1,R2

[0107] 这样做的一个负面效应是其保持R1寄存器‘忙碌’等待数据,并且其不能被用于其他目的。存储器队列扩展资源池以保持数据。因此我们将LDR转换成LAD和后续LAD:

[0108] LAD QID,M[ea1]

[0109] ...许多指令

[0110] LAF M[ea1]

[0111] ADD32 Rt,QID,R2

[0112] 由于加载队列条目QID被使用,R1被自由地用于其他目的。

[0113] 或者加载具有Rt-QID的差的Rt,或者如果QID不存在,则从M[ea1]重新加载数据,从其减去R2,并将结果放置在Rt中。

[0114] 应足以的是用上述实现方式,不需要存储器地址计算在2个加载之间匹配。如果两个地址计算不同且QID仍是有效的,则很有可能存在不好的程序员。OS(在我们的示例32中)也不必在LAD与LAF之间匹配。比所需更多的数据可被LAD读取,在这种情况下,将使用数据读取的最低有效字节。或者LAF+后缀可能需要比LAD读更多的数据,在这种情况下将使用由LAD读取的最低有效字节,后面是0直至后缀操作足够为止。此外,地址计算操作数不必在LAD与LAF之间匹配,虽然为了有良好的编码,其应获得相同的结果产生的有效地址。

[0115] B.通过使正常ALU寄存器—寄存器操作变成ALU寄存器存储器操作,节省执行循环。

[0116] 在这里,我们正在使用特殊QID=0(%0)仅仅是为了将正常寄存器—寄存器ADD指令变成存储器引用。由于LAF是前缀且必须直接耦合到ADD,所以在中间不存在指令。QID=0始终可供使用,因为其立即被使用。

[0117] LAF M[ea1]

[0118] ADD32 Rt,%q0,R2

[0119] LAF有效地将以上指令变成

[0120] ADD32 Rt,M[ea1],R2

[0121] 我们还可以使用SAF来将ALU寄存器—寄存器操作变体成将ALU操作的结果存储到存储器中的操作。

[0122] SAF M[ea1]

[0123] ADD %q0,R2,R3

[0124] 将把将R2和R3相加的结果在地址ea1处存储到存储器中。

[0125] C.促进存储之上加载时,保持顺序语义

[0126] 另一问题是我们想要促进存储(STR)之上加载(LDR)，其可以也可不与加载的地址别名使用。别名:ea1的某些或所有数据地址与ea2相同.

[0127] STR M[ea2]

[0128] LDR R1,M[ea1]

[0129] ADD32 Rt,R1,R2

[0130] 变成

[0131] LDR R1,M[ea2]

[0132] 0至许多指令

[0133] STR M[ea2],R3

[0134] 0至许多指令

[0135] ADD32 Rt,R1,R2

[0136] 为了安全地完成此操作(生成正确代码)，我们需要工具以完成此操作。在这里，我们使用LAD和SAD指令两者及其相应的LAF和SAF前缀+后缀指令，并且能够保持执行顺序且生成正确的代码。

[0137] LOOP:

[0138] SAD M[ea2],R3

[0139] 0至许多指令

[0140] a)

[0141] LAD R1,M[ea1]

[0142] 0至许多指令

[0143] b)

[0144] SAF M[ea1],R3

[0145] 0至许多指令

[0146] c)

[0147] saf-后缀

[0148] LAF M[ea1]

[0149] d)

[0150] BRN回路

[0151] 在以上代码中，必须促进LAD和SAD两者且其保持相同的排序。可能发生什么?在每个点a、b、c、d处指示替换。

[0152] a1)中断，使SAD无效，后续SAF将不许重新执行a2，LAD与SAD别名使用，使SAD无效，或者将相反地不会被插入存储器队列中

[0153] b1)中断，使SAD和LAD无效

[0154] b2)SAF与LAD别名使用，并且使LAD无效

[0155] b3)SAF使用仍有效的SAF或者重新执行，

[0156] c1)中断，使LAD无效，

[0157] c2)如果仍有效，则LAF使用LAD的数据，否则重新执行。

[0158] c3)循环，利用硬件，用IP标记和执行序列ID的组合，并且适当的管理QID、LAD/SAD/LAF/SAF。

[0159] 在LA/SA和LD/SD的上述描述中，使用LA和SA相对程序顺序位置来强制执行用于转发目的的顺序。在另一实施例中，可以使用LD/SD相对程序顺序位置来强制执行用于转发目的的顺序(例如，如下所述)。

[0160] 图4示出了根据本发明的一个实施例的其中图示出用于实现从由加载存储重新排序和优化引发的推测性转发缺失预测/错误中恢复的规则的过程400的步骤的流程图。

[0161] 如步骤401所示，本发明的实施例的目的是在该存储与该加载之间有地址匹配时找到向加载进行转发的存储。在步骤402处，最近的早先存储(例如，按机器顺序)向加载进行转发。

[0162] 在步骤403中，当按机器顺序来分配LD/SD时，针对LA/SA更新实际年龄。为LA/SA实际年龄指派(assign)以与LD/SD年龄相同的值。LD/SD保持实际年龄并执行原始程序顺序语义。

[0163] 步骤404—407示出了用于在支持推测性执行的同时保持程序连续语义的规则。步骤404—407被示为被相互水平地布置以指示同时地实现这些规则功能的机制。如步骤404中所示，如果存储具有实际年龄但加载还没有获得实际年龄，则存储早于加载。如步骤405中所示，如果加载具有实际年龄但存储还没有获得实际年龄，则加载早于存储。如在步骤406中所示，如果加载或存储都没有获得实际年龄，则将使用虚拟标识符(VID)来找出哪个较早(例如，在某些实施例中，与加载/存储指令相关联的QID表示VID)。如步骤407中所示，如果加载和存储两者都已经获得实际年龄，则使用实际年龄来找出哪个较早。

[0164] 应注意的是用来确定加载与存储之间的相对年龄的图4实施例所述的算法还可以用来确定多个存储之间的相对年龄。这在如下面在图4和后续图中所述的更新存储年龄戳时有用。

[0165] 图5示出了图示出根据本发明的一个实施例的用处理器的加载队列和存储队列资源来实现过程400的规则的方式的图。图5实施例示出了其中指令回路已解开成两个相同指令序列401—402的示例。应注意的是SA和LA可以自由地重新排序，然而，SD和LD必须保持其相对程序顺序。较早的存储可以向较晚的加载进行转发。较早意味着较小的VID(例如，保持在虚拟ID表中)或较小的年龄。如果SA具有VID但没有年龄，则SA比具有年龄的加载晚。LA/SA的实际年龄在LD/SD分配时被更新，并被指派LD/SD的相同年龄。如果存储或加载具有实际年龄，则其与实际年龄相比较，否则使用VID年龄。

[0166] 应注意的是VID表通过经由存储对应于每个VID唯一标识符的LA/SA对应的机器ID和机器资源来跟踪LA/SA与LD/SD指令之间的关联而运行。还应注意的是术语“VID”与术语“QID”是同义词，如在图4和图5的讨论中所描述的。

[0167] 现在描述图4实施例的操作示例。最初，考虑其中分配指针410最初为3的情况。V3LA已在加载队列条目#4中分派和分配。V1SA和V2SA两者都已分派。它们与V3LA相比较，并且由于V2SA小于V3LA且比V1SA更接近于V3LA，所以其潜在地向V3LA进行转发，并且因此其更新用于V3LA加载队列条目的存储初始年龄。

[0168] 分配指针现在移动至6。V2SA(#5)的存储实际年龄现在更新V3LA加载队列条目(因为V2SA是已被加戳成向此加载进行转发的记录的存储)。V4SA现在分派并与加载初始年龄相比较，并且由于V4大于V3LA，所以其并不进行转发。分配指针现在移动至11。在分配V3LD时，其用V3LD(#7)的实际年龄来更新加载队列条目#4。现在分派V1SA#11。由于V3LA#1现在

具有实际年龄但V1SA#11没有，则加载早于存储，并且因此转发是不可能的。

[0169] 预测表用于检测其中默认假设不正确的情况。该默认假设是没有存储向加载进行转发。一旦针对加载存储对检测到转发，则记录加载存储对的程序计数器，使得加载将始终等待该存储地址被分配和被计算的地址以找出该加载地址是否与该存储地址匹配且因此需要从其进行转发。

[0170] 应注意的是在一个实施例中，本文所述的特征促进给定指令序列中的在分支之前或在分支范围内的LA/SA的重新排序，其中，允许LD/SD在不存在LA/SA的情况下进行分派。如果由于分支而跳过LA和SA，或者由于已引发存储器例外情况而将其忽视，则LD和SD仍可以正确地运行，因为其包括分派两次所需的信息：第一次作为LA/SA，并且第二次作为LD/SD。在这种情况下，LD/SD的第一分派是执行地址计算(例如，加载地址)。随后，同一LD/SD可以再次地分派以满足加载或存储(例如，加载数据)的消耗部分。可以将此机制称为加载和存储指令的“双重分派”。

[0171] 应注意的是在一个实施例中，LD/SD的双重分派在对应的经去融合的LA/SA不存在时发生(例如，如经融合的LD/SD的情况一样)，或者如果由于分支而跳过LA/SA，或者其由于已引发存储器例外情况而被忽视等。

[0172] 上述双重分派功能确保LD/SD不依赖于被丢失、忽视和/或跳过LA/SA而正确地执行。由上述特征提供的益处是由加载/存储指定的数据的预取读可以通过甚至在存在分支、潜在错误、例外情况等情况下较早地调度LA/SA而按照程序顺序较早地开始(例如，减少时延)。

[0173] 图6示出了图示出根据本发明的另一个实施例的用处理器的加载队列和存储队列资源来实现过程400的规则的方式的另一图。在图6实施例中，考虑其中分配指针最初为3的情况。V3LA已在加载队列条目#4中分派和分配。分配指针现在移动至6。V1和V2(#4,#5)的存储实际年龄现在用机器ID 2和3来更新对应的SA。V4SA现在分派并与加载初始年龄相比较，并且由于V4SA大于V3LA，所以其并不转发。分配指针现在移动至11。在分配V3LD时，其用V3LD(#7)的实际年龄来更新加载队列条目#4。现在分派ID 10的V1LA。

[0174] 现在分派机器ID 2的V1SA和机器ID 3的V2SA两者。其与ID10的V1LA相比较，并且由于ID 10的V1LA不具有机器年龄(其对应的LD尚未分配)，所以虽然机器ID 2的V1SA和机器ID 3的V2SA具有实际年龄，然后已知V1和V2存储比V1更早/更旧。然后，这两个存储(V2)中最新的一个可以向ID 10的V1进行转发。现在分派SA(V2)#11。由于V1LA和V2SA不具有实际年龄，但其VID被用于比较，并且未检测到转发。分配指针现在移动至16。现在分派ID 16的V4SA，并且其与ID 10的V1LA相比较，并且由于V1LA具有实际年龄但V4SA不具有，所以V4SA比V1LA晚。因此，不可能从此存储向此较早的加载进行转发。

[0175] 图7示出了图示出根据本发明的另一个个实施例的用处理器的加载队列和存储队列资源来实现过程400的规则的方式的另一图。在图7实施例中，考虑其中分配指针最初为3的情况。在存储队列条目#4和#5中已分派并分配V1SA和V2SA。分配指针现在移动至6并分派V4SA。V1SA和V2SA两者都获得其4和5的实际年龄。

[0176] 分配指针现在移动至11。V3LA获得7的实际年龄。分派V1SA#10V2SA#11。分派V3LA且其将其地址与存储队列条目相比较，并且找到跨V1SA、V2SA和V4SA和V2SA#11的匹配。由于V3LA的实际年龄为7，其将其实际年龄与最接近它的存储年龄相比较，其为5岁，属于

V2SA，并且因此该加载将从第一存储进行转发，并在加载队列中被这样标记。

[0177] 图8示出了根据本发明的一个实施例的其中在加载之后分派存储的分派功能的概述的过程800的流程图。

[0178] 过程800在步骤801中开始，其中将存储指令分离成SA和SD。如先前所述，SA指令随SD指令保持语义以在在分离的SA与刚刚分配的SD之间不存在VID表中的匹配的情况下允许双重分派。在步骤802中，将SA重新排序成较早的机器可见程序顺序，并且使用VID表来跟踪SA以留存原始SD程序顺序。在步骤803中，在分派SA时，针对加载队列中的所有加载进行检查针对SA的地址匹配。

[0179] 在步骤804中，在有地址匹配时，通过使用加载和SA的VID号或者使用加载和存储的实际年龄而比较匹配加载的程序顺序与SA的程序顺序。这是在图3的讨论中较早图示出的过程。如果存储具有实际年龄但加载没有，则存储早于加载。如果加载具有实际年龄但存储没有，则加载早于存储。如果加载或存储任一具有实际年龄，则可以使用虚拟标识符(VID)来找出哪个较早。如果加载和存储两者都具有实际年龄，则使用实际年龄来找出哪个较早。如上所述，VID号允许跟踪原始程序顺序及重新排序的SA和LA。VID表中的条目允许对应的SD和LD在其被分配时变得与指派给SA和LA的机器资源相关联。

[0180] 在步骤805中，针对按程序顺序较晚的加载，存储将进行检查以查看该加载是否已由其他存储向其进行转发。在步骤806中，如果是这样，则存储检查先前向此加载进行转发的存储的戳以查看该存储是否按照程序顺序比其本身早。在步骤807中，如果是这样，则存储检查先前向此加载进行转发的存储的戳以查看该存储是否按照程序顺序比其本身早。在步骤808中，如果不是，则存储不向此加载进行转发。

[0181] 图9示出了根据本发明的一个实施例的其中在存储之后加载被分派的分派功能的概述的过程900的流程图。

[0182] 在步骤901中，以上述方式将加载指令分离成LA和LD。在步骤902中，将LA重新排序成较早的机器可见程序顺序并使用VID表来跟踪，如上所述。作为903的替代，针对存储队列中的所有存储而检查LA针对加载的地址匹配。

[0183] 在904中，在有地址匹配时，通过使用加载和SA的VID号或者使用加载和存储的实际年龄而比较匹配加载的程序顺序与存储的程序顺序。这是在图3的讨论中较早图示出的过程。如果存储具有实际年龄但加载没有，则存储早于加载。如果加载具有实际年龄但存储没有，则加载早于存储。如果加载或存储任一具有实际年龄，则可以使用虚拟标识符(VID)来找出哪个较早。如果加载和存储两者都具有实际年龄，则使用实际年龄来找出哪个较早。如上所述，VID号允许跟踪原始程序顺序及重新排序的SA和LA。随后，在步骤905中，使加载消耗来自按程序顺序最接近于其自己的程序顺序的存储的数据。

[0184] 图10示出了根据本发明的一个实施例的统一加载队列的图。虚拟加载/存储队列的目的是允许处理器在机器中分配比使用其加载/存储队列的实际物理大小可以容纳的更多的加载/存储。作为回报，这允许处理器超过其处理器的加载/存储队列的物理大小限制分配除加载/存储之外的其他指令。即使某些加载/存储在加载/存储队列中仍不具有空间，这些其他指令仍可以被分派和执行。

[0185] 随着加载从加载队列退出，加载分派窗口移动至序列中的后续指令，并且将包括针对与已经从加载队列退出的加载数目等同的分派要考虑的更多已分配加载。在此图中，

加载分派窗口将从左向右移动。

[0186] 在一个实施例中，加载分派窗口将始终包括与加载队列中的条目的数目相等的加载数目。在任何时间不能在加载分派窗口之外分派加载。除加载之外的调度器窗口中的其他指令(例如，Sub、Add等)可以分派。加载分派窗口内的所有加载可以每当其准备就绪时分派。

[0187] 图11示出了根据本发明的实施例的示出滑动加载分派窗口的统一加载队列。图11示出了与图10相比较的后续时刻。随着加载退出加载队列，加载分派窗口移动至序列中的后续指令，并且将包括针对与已经从加载队列退出的加载数目等同的分派要考虑的更多已分配加载。加载分派窗口将始终包括与加载队列中的条目数目相等的加载数目。在任何时间不能在加载分派窗口之外分派加载。除加载之外的调度器窗口中的其他指令(例如，Sub、Add等)可以分派。加载分派窗口内的所有加载可以每当其准备就绪时分派。因此，用此方案获得的一个益处是如果超过加载或存储队列容量，到调度器的分配不停止，替代地，尽管已超过加载或存储队列容量，我们继续在调度器内部分配指令，包括加载和存储，加载和存储动态窗口将确保将不会分派在加载或存储队列的容量之外的加载或存储。

[0188] 图12示出了根据本发明的一个实施例的分布式加载队列。图12实施例的目的是实现分布式加载队列和分布式存储队列，其保持单个程序/线程连续语义，但仍允许跨多个核心/存储器片段进行加载和存储的乱序分派。

[0189] 图12图示出了避免死锁的加载队列扩展解决方案。创建加载/存储队列的扩展并从引发死锁的加载/存储点(从该点向前)按程序顺序向该扩展队列分配死锁加载/存储直至加载/存储队列具有可用的自由的条目。在图12情形中，LD 3加载取决于SD，SD又取决于由于load_Q B已满而不能被分派的LD 2(具有映射到load_Q B的地址)。在此死锁情形中，在检测到死锁时，允许LD 1和LD 2按序一个接一个地向预留部分B中分派和退出。用于分布式加载/存储队列的保守策略是为每个加载/存储预留每个加载/存储分布式队列中的条目。在此图中，每个分配的加载需要预留load_Q A中的条目和load_Q B中的另一条目。

[0190] 应注意的是在分布式加载/存储队列中，存在关于已分配加载/存储的问题，因为其地址在分配时是未知的。因此，在乱序分派时仅知道给定加载或存储将占用哪个分布式队列。

[0191] 本发明的实施例可以采用用于分布式加载/存储队列的三个不同解决方案来避免关于乱序分派的死锁。

[0192] 1.在死锁(不具有分派给加载/存储缓冲器的空间)的最早加载/存储处引发缺失预测和清刷并开始按序分派加载/存储达一段时间，或者通过其中每个加载/存储在所有分布式队列中分配空间的保守分配。一旦(在分派时)知道该加载/存储的地址，因此知道将接收该加载/存储的特定加载队列，其可以将其他队列中的预留空间解除分配。

[0193] 2.创建加载/存储队列的扩展并用来从引发死锁的加载/存储点按程序顺序向该扩展队列分配死锁加载/存储(图9)。

[0194] 3.动态分派窗口大小确定，其中，在连续性窗口之外的未分派加载的和应小于或等于该特定加载队列中的自由非预留空间的数目(例如，图11)。

[0195] 图13示出了根据本发明的一个实施例的具有按序连续性窗口的分布式加载队列。确定动态分派窗口大小确定，使得在连续性窗口之外的未分派加载的和应小于或等于该特

定加载队列中的自由非预留空间的数目。每个加载队列将使用如这里所示的其相应分派窗口来跟踪其条目。在任何时间用于每个加载队列的动态窗口大小＝该队列的实际物理大小加虚拟条目(在这种情况下6+4＝10)，因此在这种情况下，窗口大小应仅覆盖10个负载。请注意，不对用于其他队列的负载进行计数(例如，LD 4)。

[0196] 预留的预订比是3。该预订比是竞争每个预留空间的按序加载的数目。在本示例中，只有前两个按序未分派加载(从左向右扫描按序连续性窗口)可以向预留分配进行分派(假设队列的2个条目被分配给预留)。因此，虚拟条目的数目＝(预订比-1)*预留条目的数目＝(3-1)*2＝4。

[0197] 关于按序连续性窗口大小确定，未被分派给加载队列中的条目(被捕捉空间)的在任何时间的加载数目(从最旧的至最新的进行计数)加到预留空间的分派加载的数目必须小于或等于(预订比*预留条目的数目)。在这种情况下，加载的数目必须小于或等于3。该预订比是设计可配置性能度量，其确定预留空间的已接受(占用VS预订)比是什么。这在最早的未分派加载不能找到要分派到预留条目外面的队列空间的情况中实行。在这种情况下，从最早(最旧)加载开始的那些加载将竞争预留空间，预订比确定多少加载将等待占用每个预留条目，预留条目始终被首先分配给最旧的未分派加载，并且一旦该加载退出，则下一个最旧加载可以占用条目(预订比确定从所分派的最旧的一个开始一个接一个地占用预留条目的那些加载的数目)。

[0198] 应注意的是在一个实施例中，来自每个队列的按序连续性窗口的加载可以在该队列的未预留部分中未留下空间(按序从最旧加载开始)时向该队列的预留空间进行分派。还应注意的是在一个实施例中，在任一队列的按序连续性窗口外面和在该队列的动态分派窗口内的加载不能向该队列的预留部分进行分派。

[0199] 还应注意的是只要在队列的未预留部分中存在空间，该队列的整个动态分派窗口内的任何加载就可以向任何分布式队列的未预留部分的任何条目进行乱序分派。每个循环调整任一队列的按序连续性窗口和动态分派窗口两者的大小以反映每个加载分派或退出之后的在上文提供的等式中所述的其大小限制。

[0200] 图14示出了根据本发明的一个实施例的用于多核处理器的分段存储器子系统的图。图13示出了线程之间和/或一般地加载和存储之间的同步方案的全面方案和实现。该方案描述跨加载/存储架构和/或跨存储器引用和/或线程的存储器访问的存储器引用的同步和消歧的优选方法。在图15中，示出了寄存器堆(地址和或数据寄存器)的多个分段以及执行单元、地址计算单元和1层高速缓存和/或加载存储缓冲器的片段和/或2层高速缓存和地址寄存器互连1200和地址计算单元互连1201。那些分段元素可以通过将其集中式资源分段并分布到多个引擎中而在一个核心/处理器内构造，或者其可由采取多核心/多处理器配置的不同核心/处理器的元件构成。那些片段1211中的一个在图中被示为片段1号；可以将片段按比例缩放至大的数目(一般地至N个片段，如图中所示)。

[0201] 这种机制还充当用于那些引擎/核心/处理器之间的存储器架构的相干性方案。此方案通过来自一个片段/核心/处理器中的地址计算单元中的一个的地址请求开始。例如，假设由片段1(例如，1211)请求地址。其可以使用属于其自己的片段的地址寄存器和或使用地址互连总线1200根据跨其他片段的寄存器来获得并计算其地址。在计算该地址之后，其创建用来访问高速缓存和存储器的32位地址或64位地址的引用地址。此地址通常被分段成

标签字段及设定和线路字段。此特定片段/引擎/核心将把地址存储到其加载存储缓冲器和/或L1和/或L2地址阵列1202中，同时其将通过使用压缩技术来创建标签的压缩版本(具有比地址的原始标签字段更小的位数)。

[0202] 此外，不同的片段/引擎/核心/处理器将使用设定字段或设定字段的子集作为索引来识别地址被保持在哪个片段/核心/处理器中。由地址设定字段位进行的此片段编索引确保不包括特定片段/核心/引擎中的地址的所有权，即使对应于该地址的存储器数据可以存在于另一或多个其他片段/引擎/核心/处理器中。即使在每个片段中将地址CAM/标签阵列1202/1026示为与数据阵列1207耦合，其也可能仅在放置和布局的物理接近区域中被耦合，或者甚至由于两者都属于特定引擎/核心/处理器但在保持在地址阵列的地址与一个片段内部的数据阵列中的数据之间不存在关系的事实。

[0203] 图15示出了如何由本发明的实施例来处理加载和存储的图。如图15中所描绘的，每个片段与其加载存储缓冲器和存储退出缓冲器相关联。针对任何给定片段，指定与该片段或另一片段相关联的地址范围的加载和存储被发送到该片段的加载存储缓冲器以用于处理。应注意的是其可随着核心乱序执行指令而乱序到达。在每个核心内，核心不仅可访问其自己的寄存器堆，而且可访问其他核心的寄存器堆中的每一个。

[0204] 本发明的实施例实现分布式加载存储排序系统。该系统跨多个片段分布。在片段内，由该片段来执行本地数据相关性检查。这是因为该片段仅在该特定片段的存储退出缓冲器内加载和存储。这限制了必须注意其他片段以保持数据相干性的需要。以这种方式，本地地执行片段内的数据相关性。

[0205] 关于数据一致性，存储分派门根据严格的按程序顺序存储器一致性规则来强制执行存储退出。存储乱序到达加载存储缓冲器处。加载也乱序到达加载存储缓冲器处。同时，乱序加载和存储被转发到存储退出缓冲器以用于处理。应注意的是虽然存储在给定片段内按序退出，但随着其转到存储分派门，其可以乱序从多个片段出来。存储分派门强制执行这样的策略，即确保即使存储可乱序跨存储退出缓冲器而驻留，并且即使缓冲器可相对于其他缓冲器的存储乱序将存储转发到存储分派门，分派门也确保其被严格地按序转发到片段存储器。这是因为存储分派门具有存储退出的全局视图，并且仅允许存储跨所有片段、例如全局地按序离开至存储器的全局可见侧。以这种方式，存储分派门充当全局观察者以确保存储最终跨所有片段按序返回到存储器。

[0206] 图16示出了根据本发明的一个实施例的存储过滤算法的图。图16实施例的目的是过滤存储以防止所有存储必须针对加载队列中的所有条目进行检查。

[0207] 存储针对地址匹配而嗅探高速缓存以保持相干性。如果线程/核心X加载从高速缓存线进行读取，则其标记其从那里加载数据的高速缓存线的部分。在另一线程/核心Y存储嗅探高速缓存时，如果任何此类存储与该高速缓存线部分重叠，则针对线程/核心X的该加载引发缺失预测。

[0208] 用于过滤这些嗅探的一个解决方案是跟踪加载队列条目的引用。在这种情况下，存储不需要嗅探加载队列。如果存储具有与访问掩码的匹配，则从引用跟踪符获得的加载队列条目将引发该加载条目进行缺失预测。

[0209] 在另一解决方案(其中不存在引用跟踪符)中，如果存储与访问掩码具有匹配，则该存储地址将嗅探加载队列条目并将引发匹配的加载条目进行缺失预测。

[0210] 用两个解决方案，一旦加载从高速缓存线进行读取，其设定相应的访问掩码位。当该加载退出时，其将该位重置。

[0211] 图17示出了根据本发明的一个实施例的具有组成从存储器按序进行读取的存储器一致性模型中的乱序加载的信号量实现方式。本文所使用的术语信号量指的是为多个线程/核心提供对公共资源的访问控制的数据构造。

[0212] 在图17的实施例中，使用访问掩码来控制由多个线程/核心对存储器资源的访问。访问掩码通过跟踪高速缓存线的哪些字具有待决加载而运行。乱序加载在访问高速缓存线的字时设定掩码位，并在该加载退出时清除掩码位。如果来自另一线程/核心的存储在设定掩码位的同时向该字进行写入，则其将用信号通知对应于该加载的加载队列条目(例如，经由跟踪符)以被缺失预测/清刷或用其从属指令重试。访问掩码还跟踪线程/核心。

[0213] 以这种方式，访问掩码确保正确地实现存储器一致性规则。存储器一致性规则规定对于此信号量而言存储按序更新存储器且加载按序从存储器读取以跨两个核心/线程工作。因此，由核心1和核心2执行的代码将被正确地执行，其中，其两者都访问存储器位置“标志”和“数据”。

[0214] 图18示出了根据本发明的一个实施例的通过使用基于锁的模型和基于事务的模型两者而组成针对存储器进行按序读取的加载的到存储器一致性模型中的乱序加载。

[0215] 如上所述，存储器一致性规则规定存储按序更新存储器且加载按序引用存储器以便两个核心/线程适当地进行通信。在图18中的右侧底部，示出了两个核心，核心1和核心2。使用两个存储器资源，标志和数据，实现通信并正确地在核心1与核心2之间共享数据。例如，当核心1想要将数据传递至核心2时，如核心1内的代码所指示的，其将存储数据且然后设定标志。如核心2内的代码所指示的，核心2将加载标志并检查标志是否等于1。如果标志不等于1，则核心2将向后转移并保持检查标志直至其确实等于1。在该时间点处，其将加载数据。

[0216] 用其中加载和存储乱序执行的乱序架构，可以使用基于锁的存储器一致性模型来确保两个实体(例如，核心1和核心2)保持按序存储器一致性语义。这通过使用访问掩码、线程ID寄存器以及跟踪符寄存器而示出。通过在代码的关键段内设置任何加载的对应访问掩码位来设定该锁。如果发生从另一线程/核心到该高速缓存线字的任何访问，则锁将阻止该访问。在一个实施例中，这可以通过将访问示为缺失来实现。当该锁被清除时，允许对该字的访问。

[0217] 替换地，可以使用基于事务的方法来保持按序存储器一致性语义。用基于事务的方法，通过设置事务内的任何加载的对应访问掩码位来设定原子数。如果在设定掩码位的同时发生从另一线程/核心或并行事务到该高速缓存线的任何访问，则其将用信号通知对应于该加载的加载队列条目(例如，经由跟踪符)以被缺失预测/清刷或用其从属指令重试。访问掩码还跟踪线程/核心。当该事务结束时，掩码位将被清除。使用线程ID寄存器来跟踪哪个线程正在访问统一存储队列条目的哪个字。

[0218] 图19示出了根据本发明的一个实施例的多核分段存储器子系统的多个核心。本实施例示出了将如何防止来自多核分段存储器子系统内的加载访问被标记为进行中的事务的一部分(例如，看类似于锁定情况)的字。

[0219] 应注意的是如果此多核分段子系统是较大集群的一部分，则存在具有共享存储器

子系统的外部处理器/核心/集群。在这种情况下，属于其他外部处理器/核心/集群的加载将继续进行且将不会被防止从任何存储器位置进行加载，无论该存储器位置是否是事务访问的一部分。然而，所有加载将对访问掩码进行标记以通知作为事务的一部分的未来存储。

[0220] 来自其他处理器的嗅探存储将其地址与掩码相比较。如果存储见到其正在尝试存储的地址在来自另一线程加载(作为事务的一部分的加载)的访问掩码中被标记，则存储将引发该加载被缺失预测。否则，将在该加载退出时清除该标记(例如，由此完成事务)。

[0221] 图20示出了根据本发明的一个实施例的访问统一存储队列的异步核心的图，其中存储可以基于存储资历而将数据转发到任一线程中的加载。

[0222] 如上所述，存储器一致性规则规定存储按序更新存储器且加载按序从存储器进行读取，使得核心/线程适当地进行通信。在图20中的右侧底部，示出了两个核心，核心1和核心2。两个核心是异步的，并执行在每个核心内所指示的代码以访问标志和数据存储器资源。

[0223] 在图20的实施例中，统一存储队列对于可以访问它的多个线程中的任何线程而言是不可知的。在本实现方式中，来自不同线程的存储可以向不同线程的加载进行转发，而同时通过遵循一组算法规则而仍保持按序存储器一致性语义。线程可以基于存储资历而从彼此进行转发。

[0224] 当在同一线程中的在一存储的前面的所有加载和存储都已被执行时，该存储是资深的。从另一线程接收到转发的线程不能独立地退出加载/存储。线程必须在它们从其接收到转发的其他线程已进行缺失预测的情况下有条件地进行缺失预测。特定加载可以从同一线程转发存储或从不同的线程资深存储进行转发，如果在同一线程内不存在向其进行转发的存储的话。

[0225] 用图20方法，通过设置对统一存储队列条目中的字内的字节的任何访问的对应访问掩码位来设定原子数。如果在设定掩码位的同时发生从另一线程/核心或并行事务到该存储队列条目的任何访问，则其将用信号通知对应于该加载的加载队列条目(例如，经由跟踪符)以被缺失预测/清刷或用其从属指令重试。访问掩码还跟踪线程/核心。当该事务结束时，掩码位将被清除。

[0226] 图21示出了根据本发明的一个实施例的描绘其中存储具有资历的功能的图。如图21中所描绘的，特定加载将从同一线程转发存储进行转发。如果不存在来自线程内的传送，则其可以从不同的线程资深存储进行转发。此原理在其中多个核心/线程正在访问共享存储器的情况下起作用。在这种情况下，存储可以基于存储资历从任一线程向来自任一线程的加载进行转发，然而，只有当不存在从线程内向特定加载的转发时。当在同一线程中的在一存储前面的所有加载和存储都已被执行时，该存储是资深的。

[0227] 另外，应注意的是线程不能独立地退出加载/存储。线程在它从其接收到转发的另一线程进行存储缺失预测或清刷时必须进行加载缺失预测。

[0228] 图21直观地描述了两个异步核心/线程(例如，核心/线程1和核心/线程2)之间的执行的示例性流。线2101—2105示出了存储基于其资历而向不同的加载进行转发的方式。为了帮助举例说明资历如何从存储向存储进展，紧挨着每个指令示出了数字以示出随着其从0进展至14的执行的不同阶段。特别地，根据上述规则，应注意线2103所指示的存储向同一线程内的加载进行转发的方式。因此，如上所述，从其自己的线程内进行转发的加载不能

从任何相邻线程进行转发。这用跨转发线的黑色十字形示出。

[0229] 图22示出了非推测性的根据本发明的一个实施例的未消歧乱序加载存储队列退出实现方式(例如,提供低功率、低裸片面积以及较低的定时紧迫性)。

[0230] 加载退出/重新排序缓冲器(SRB)可以以两个实现方式、退出实现方式和重新排序实现方式进行操作。

[0231] 在退出实现方式中,在存储退出时按照原始程序顺序从存储队列将存储加载到SRB中,使得按照原始程序顺序较早的存储在SRB的顶部处。后续加载然后寻找地址匹配(例如,使用地址CAM),并从SRB/存储高速缓存中的匹配条目进行转发。在其中存在两个或更多地址匹配的情况下,优先级编码器可以通过扫描第一个来定位正确的转发条目。这节省了到存储器的转移并允许机器向前继续进行。如果分派了加载且向其进行转发的存储已退出到SRB/存储高速缓存,则该加载从SRB/存储高速缓存进行转发并将配对关系记录在预测表中。为了检测其中在向加载进行转发的存储退出到SRB/存储高速缓存之前分派加载的情况,加载必须创建地址掩码,其在那里标记其自己的地址。这可以以不同的方式来实现(例如,图17的实施例)。

[0232] 如上文所讨论的,图17描述访问掩码,其通过跟踪高速缓存线的哪些字具有待决加载而运行。乱序加载在访问高速缓存线的字时设定掩码,并在该加载退出时清除掩码位。如果来自同一线程/核心的存储在其退出时检测到其在掩码位被设置的同时向该字进行写入,则其将用信号通知对应于该加载的加载队列条目(经由跟踪符)以被缺失预测/清刷或用其从属指令重试。访问掩码还跟踪线程/核心。

[0233] 图22是非消歧加载存储队列,因为其不包括用以将乱序加载和存储消歧的对应硬件的事实。加载和存储根据机器资源允许而乱序分派。传统上,在加载队列和存储队列两者中使用地址匹配和对应的消歧硬件以确保正确的存储队列条目被转发到请求加载队列条目,如上所述(例如,图5和图6)。加载队列和存储队列的内容对外面的核心/线程不可见。

[0234] 在图22中,分派的加载和存储地址并未相对于存储队列或加载队列中的条目而被消歧。加载/存储队里现在是具有减小的裸片面积、功率消耗以及定时要求的改进的缓冲器实现方式。SRB将执行消歧功能。在SRB中检测到第一匹配时,使用那些匹配来填充存储至加载转发预测表中的条目以随着指令序列的执行向前进行而强制执行转发。

[0235] 在加载被分派时,其检查预测表以查看其是否与对应的存储配对。如果加载被配对且该特定存储已被分派,则加载将从在预测表中记录的该存储队列条目号进行转发。如果该存储尚未被分派,则加载将其加载队列条目号寄存在预测表中并将在加载队列中标记其本身以等待存储数据被转发。当稍后分派存储时,其检查预测表以获得加载队列条目号并向该加载进行转发。

[0236] 一旦针对加载存储对检测到转发,则记录加载存储对的PC和地址,使得地址匹配被验证。如果地址匹配,则加载将不会分派直至存储数据被分派,并且该加载将被标记为从其进行转发。使用预测阈值来设置加载存储对之间的转发关系中的置信度水平。

[0237] 图23示出了根据本发明的一个实施例的未消歧乱序加载存储队列重新排序实现方式的重新排序实现方式。图23还提供非推测性的低功率、低裸片面积以及较低的定时紧迫性。

[0238] 加载退出/重新排序缓冲器(SRB)可以以两个实现方式、退出实现方式和重新排序

实现方式进行操作。

[0239] 在图23的重新排序实现方式中，从存储队列将存储地址乱序加载到SRB中(例如，根据资源允许)。在分配每个存储时，其接收序列号。SRB然后通过根据它们的序列号将存储重新排序而运行，使得它们按照原始程序顺序而驻留于SRB中。按照程序顺序较早的存储在SRB的顶部处。后续加载然后寻找地址匹配和分配年龄(在加载和存储的分配时间给定的程序顺序序列号)。随着加载被分派，其查看SRB，如果其见到尚未分派(尚无地址计算)的较早存储(与其自己的序列号相比)，则可以实现两个解决方案中的一个。

[0240] 1.加载不分派，其在其将其本身分派之前等到所有较早存储都已分派。

[0241] 2.加载分派并在高速缓存的访问掩码中标记其地址(如图17中所示)。后续存储检查访问掩码并遵循相同的方法，如图17中所描述的。

[0242] 应注意的是优先级编码器如上所述地运行以定位正确的转发条目。

[0243] 图24示出了根据本发明的一个实施例的指令序列(例如，踪迹)重新排序的推测性执行实现方式。在推测性模式下，存储在存储退出时按照原始程序顺序从存储队列移动至SRB中，使得按照原始程序顺序较早的存储在SRB的顶部处。后续加载然后寻找地址匹配(例如，使用地址CAM)，并从SRB/存储高速缓存中的匹配条目进行转发。在其中存在两个或更多地址匹配的情况下，优先级编码器可以通过扫描第一个来定位正确的转发条目。这允许机器向前继续进行。如果分派了加载(其第一次检查SRB))且向其进行转发的存储退出到SRB/存储高速缓存，则该加载从SRB/存储高速缓存进行转发并将其配对关系记录在预测表中。为了检测其中在向加载进行转发的存储退出到SRB/存储高速缓存之前分派加载的情况，退出时的加载将再一次检查存储队列。如果加载发现转发存储匹配，则其将用信号通知对应于该加载的加载队列条目以使其被缺失预测/清刷或用其从属指令进行重试。转发预测器将从此缺失转发进行学习。

[0244] 应注意的是加载将能够针对先前的存储在SRB中检查匹配地址，因为并非SRB中的所有存储都将被提交给外部高速缓存/存储高速缓存架构上可见状态(将SRB存储装置留给可见存储器)直至包括所述加载的踪迹中的所有指令都已达到踪迹提交(commit)状态(例如，全部变成非推测性的且踪迹整体上准备好提交)。

[0245] 存储退出/重新排序缓冲器在功能上启用推测性执行。可以将推测性执行的结果保存在存储退出/重新排序缓冲器中直至已知推测性结果。推测性结果并不是在架构上可见的。一旦提交推测性状态，则可以将存储提交到存储高速缓存。在提交该状态之前，需要重新尝试的任何例外或加载和存储将用信号通知将防止状态提交的例外或缺失预测。存储与对应的加载之间的转发缺失预测可以是固定的(例如，通过引发在缺失＝转发加载点处清刷机器的缺失预测等)。

[0246] 在2012年1月27日由Mohammad Abdallah提交的代理人档案号SMII-033、“HARDWARE ACCELERATION COMPONENTS FOR TRANSLATING GUEST INSTRUCTIONS TO NATIVE INSTRUCTIONS”的美国专利申请13/360,024中可以找到SRB功能的附加描述。

[0247] 图25示出了根据本发明的一个实施例的示例性微处理器管线2500的图。微处理器管线2500包括取读模块2501，其实现用于识别并提取包括执行的指令的过程的功能，如上所述。在图25的实施例中，取读模块后面是解码模块2502、分配模块2503、分派模块2504、执行模块2505和退出模块2506。应注意的是微处理器管线2500仅仅是实现上述本发明的实施

例的功能的管线的一个示例。本领域的技术人员将认识到可以实现包括上述解码模块的功能的其他微处理器管线。

[0248] 出于说明的目的，前述描述参考并不意图为穷举的或限制本发明的特定实施例。根据以上讲授内容，可以有许多修改和变更。选择和描述实施例是为了最好地解释本发明的原理及其实际应用，从而使得本领域的技术人员在有和适合于其特定使用的各种修改的情况下最好地利用本发明及其各种实施例。

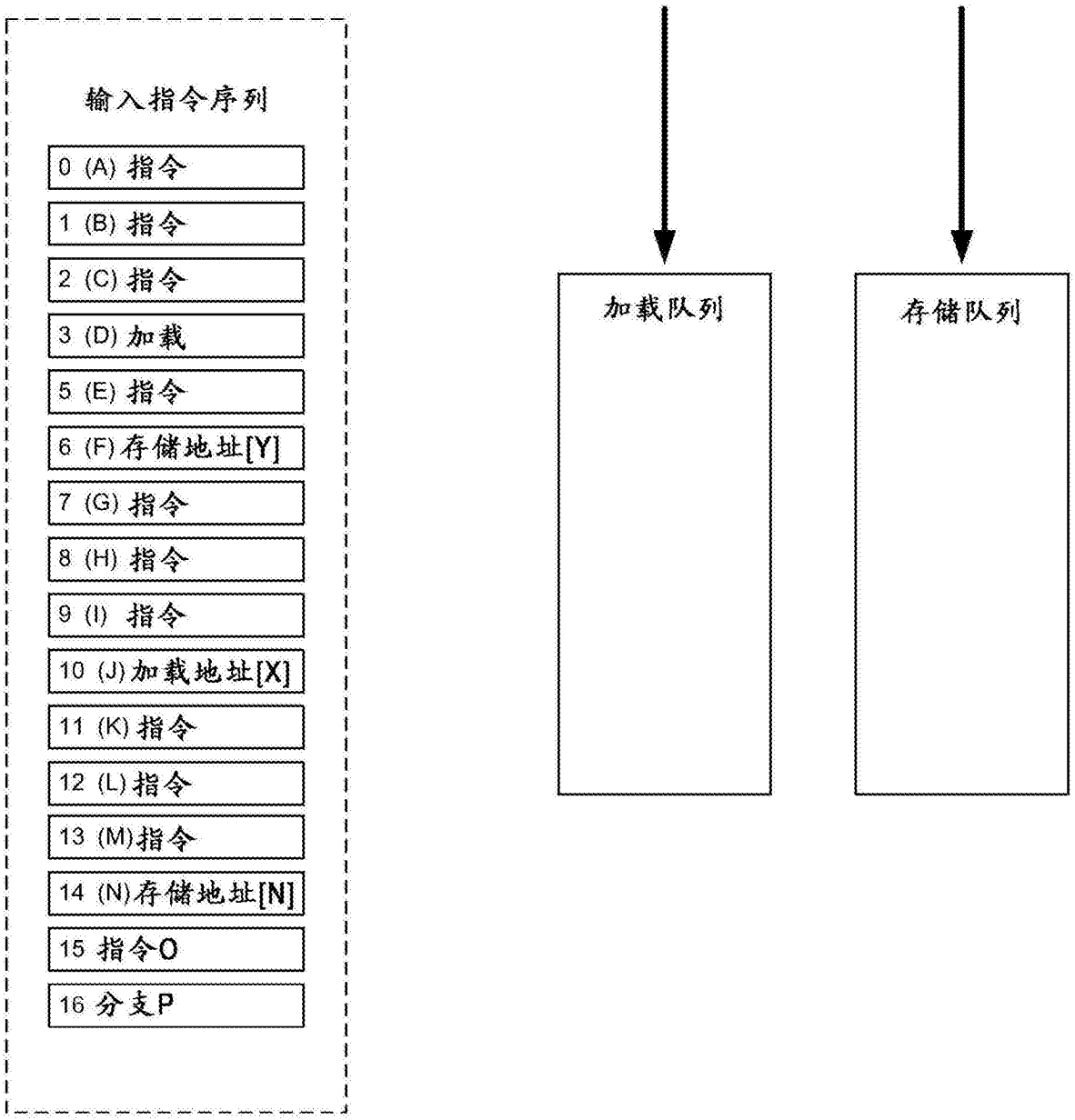

输入指令序列
0 (A) 指令
1 (B) 指令
2 (C) 指令
3 (D) 加载
5 (E) 指令
6 (F) 存储地址[Y]
7 (G) 指令
8 (H) 指令
9 (I) 指令
10 (J) 加载地址[X]
11 (K) 指令
12 (L) 指令
13 (M) 指令
14 (N) 存储地址[N]
15 指令O
16 分支P
加载队列
存储队列

图1

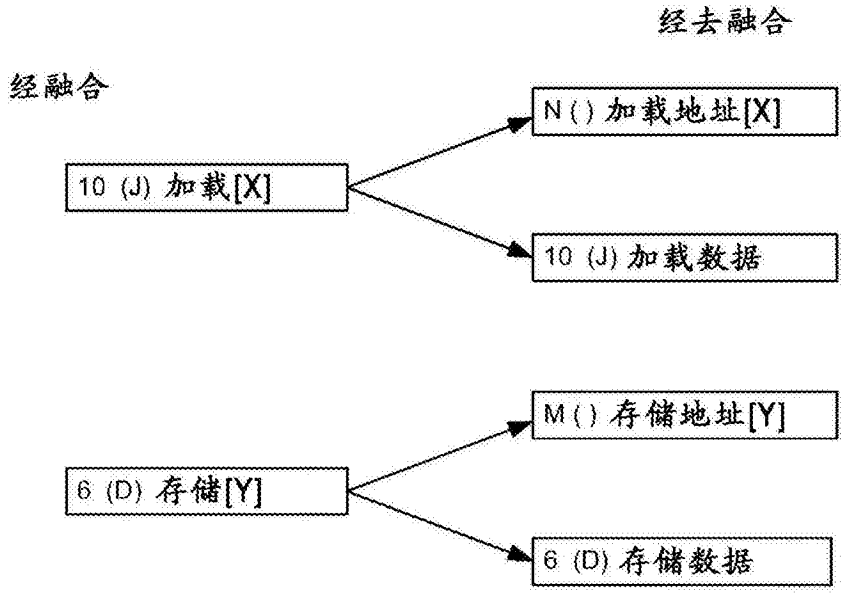

经去融合
经融合
N ( ) 加载地址[X]
10 (J) 加载[X]
10 (J) 加载数据
M ( ) 存储地址[Y]
6 (D) 存储[Y]
6 (D) 存储数据

图2

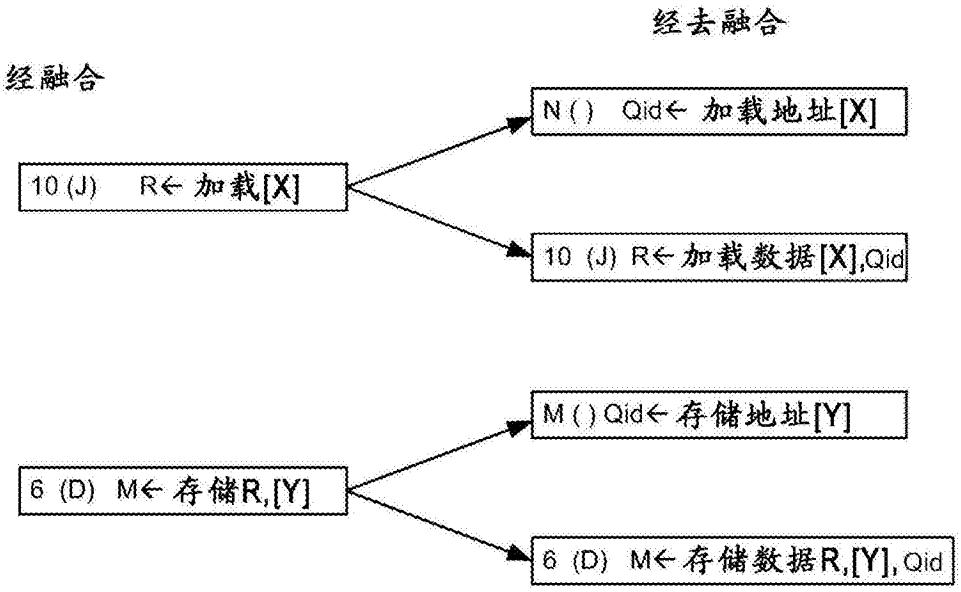

经去融合
经融合
N ( ) Qid← 加载地址[X]
10 (J) R← 加载[X]
10 (J) R← 加载数据[X],Qid
M ( ) Qid← 存储地址[Y]
6 (D) M← 存储R,[Y]
6 (D) M← 存储数据R,[Y],Qid

图3

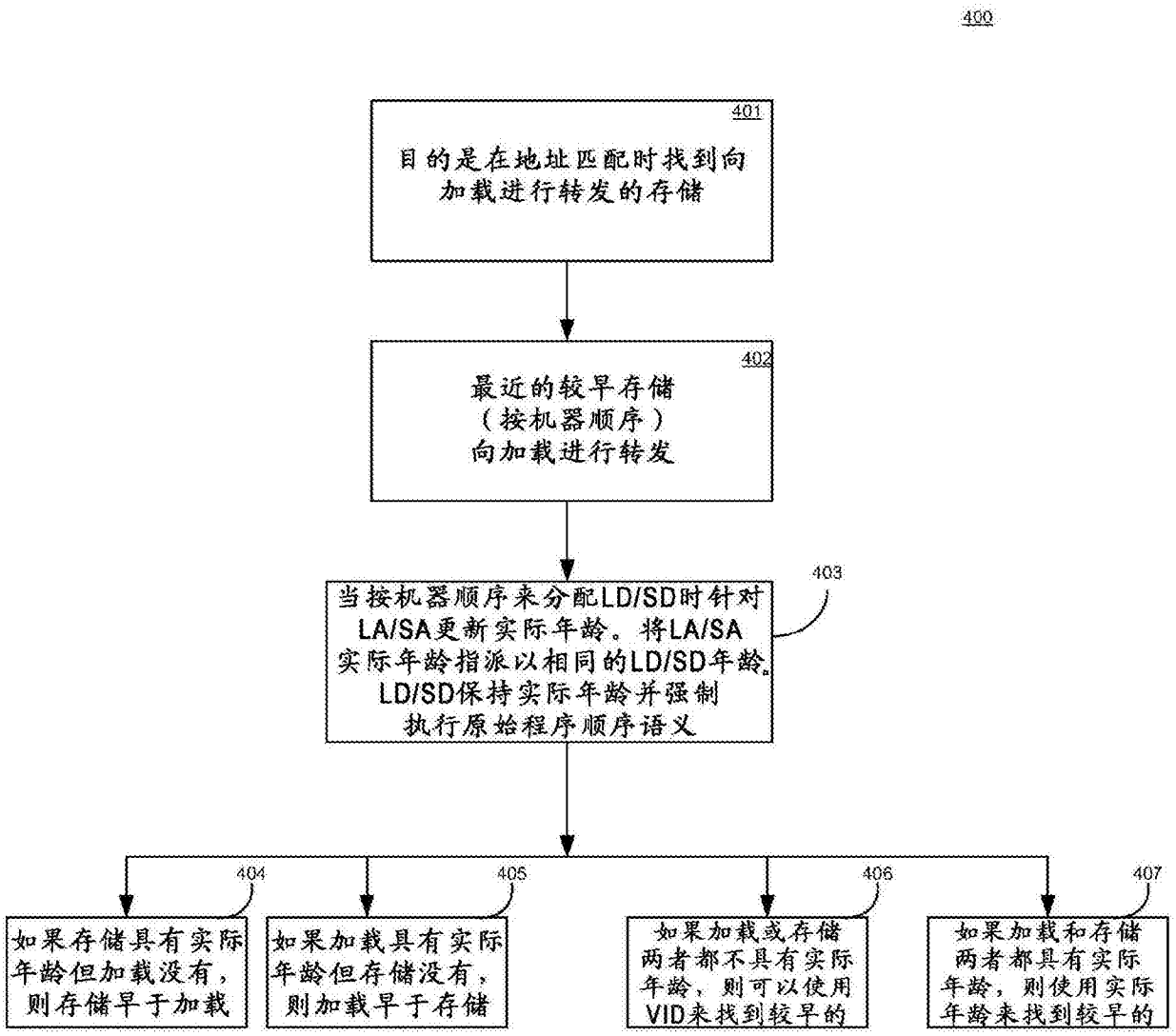

400
401
目的是在地址匹配时找到向加载进行转发的存储
402
最近的较早存储（按机器顺序）向加载进行转发
403
当按机器顺序来分配LD/SD时针对LA/SA更新实际年龄。将LA/SA实际年龄指派以相同的LD/SD年龄。LD/SD保持实际年龄并强制执行原始程序顺序语义
404
如果存储具有实际年龄但加载没有，则存储早于加载
405
如果加载具有实际年龄但存储没有，则加载早于存储
406
如果加载或存储两者都不具有实际年龄，则可以使用VID来找到较早的
407
如果加载和存储两者都具有实际年龄，则使用实际年龄来找到较早的

图4

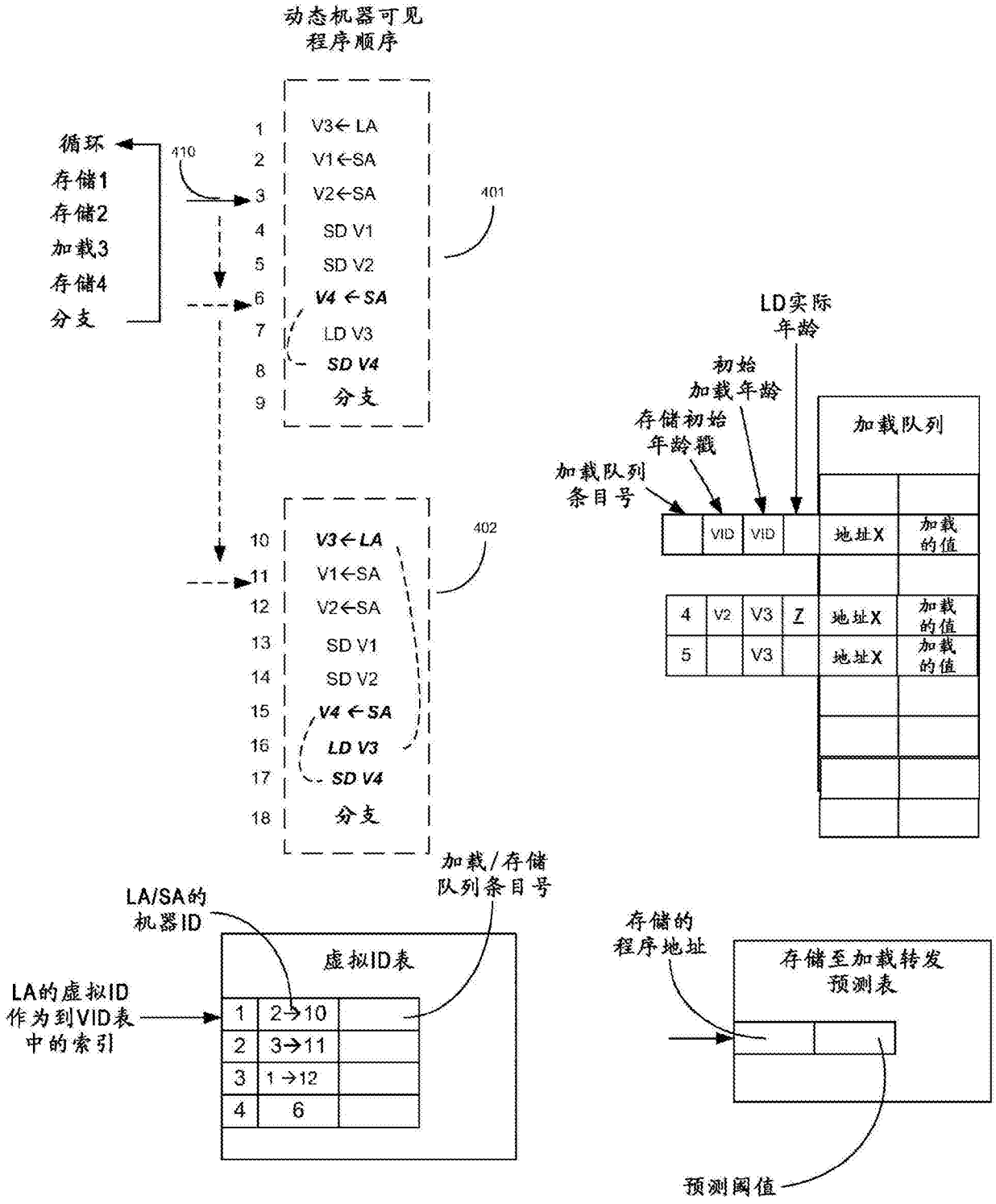

动态机器可见
程序顺序
循环
存储1
存储2
加载3
存储4
分支
410
401
1 V3←LA
2 V1←SA
3 V2←SA
4 SD V1
5 SD V2
6 V4←SA
7 LD V3
8 SD V4
9 分支
402
10 V3←LA
11 V1←SA
12 V2←SA
13 SD V1
14 SD V2
15 V4←SA
16 LD V3
17 SD V4
18 分支
LD实际
年龄
初始
加载年龄
存储初始
年龄戳
加载队列
条目号
加载队列
VID VID 地址X 加载的值
4 V2 V3 Z 地址X 加载的值
5 V3 地址X 加载的值
加载/存储
队列条目号
LA/SA的
机器ID
虚拟ID表
LA的虚拟ID
作为到VID表
中的索引
1 2→10
2 3→11
3 1→12
4 6
存储的
程序地址
存储至加载转发
预测表
预测阈值

图5

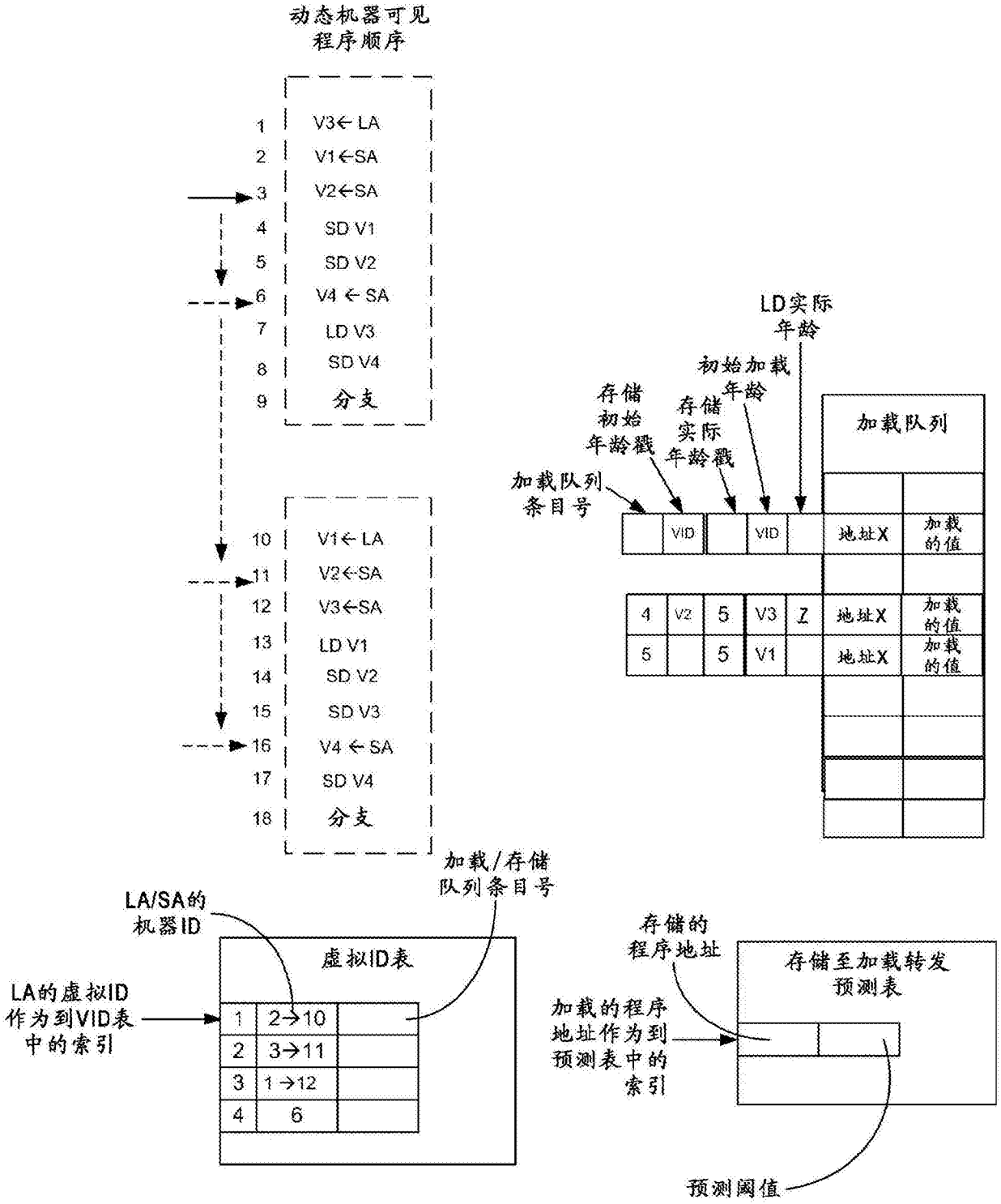

动态机器可见
程序顺序
1 V3← LA
2 V1←SA
3 V2←SA
4 SD V1
5 SD V2
6 V4 ← SA
7 LD V3
8 SD V4
9 分支
10 V1← LA
11 V2←SA
12 V3←SA
13 LD V1
14 SD V2
15 SD V3
16 V4 ← SA
17 SD V4
18 分支
LD实际
年龄
初始加载
年龄
存储
初始
年龄戳
存储
实际
年龄戳
加载队列
加载队列
条目号
VID
VID
地址X
加载
的值
4 V2 5 V3 7 地址X 加载的值
5 5 V1 地址X 加载的值
加载/存储
队列条目号
LA/SA的
机器ID
虚拟ID表
LA的虚拟ID
作为到VID表
中的索引
1 2→10
2 3→11
3 1→12
4 6
存储的
程序地址
存储至加载转发
预测表
加载的程序
地址作为到
预测表中的
索引
预测阈值

图6

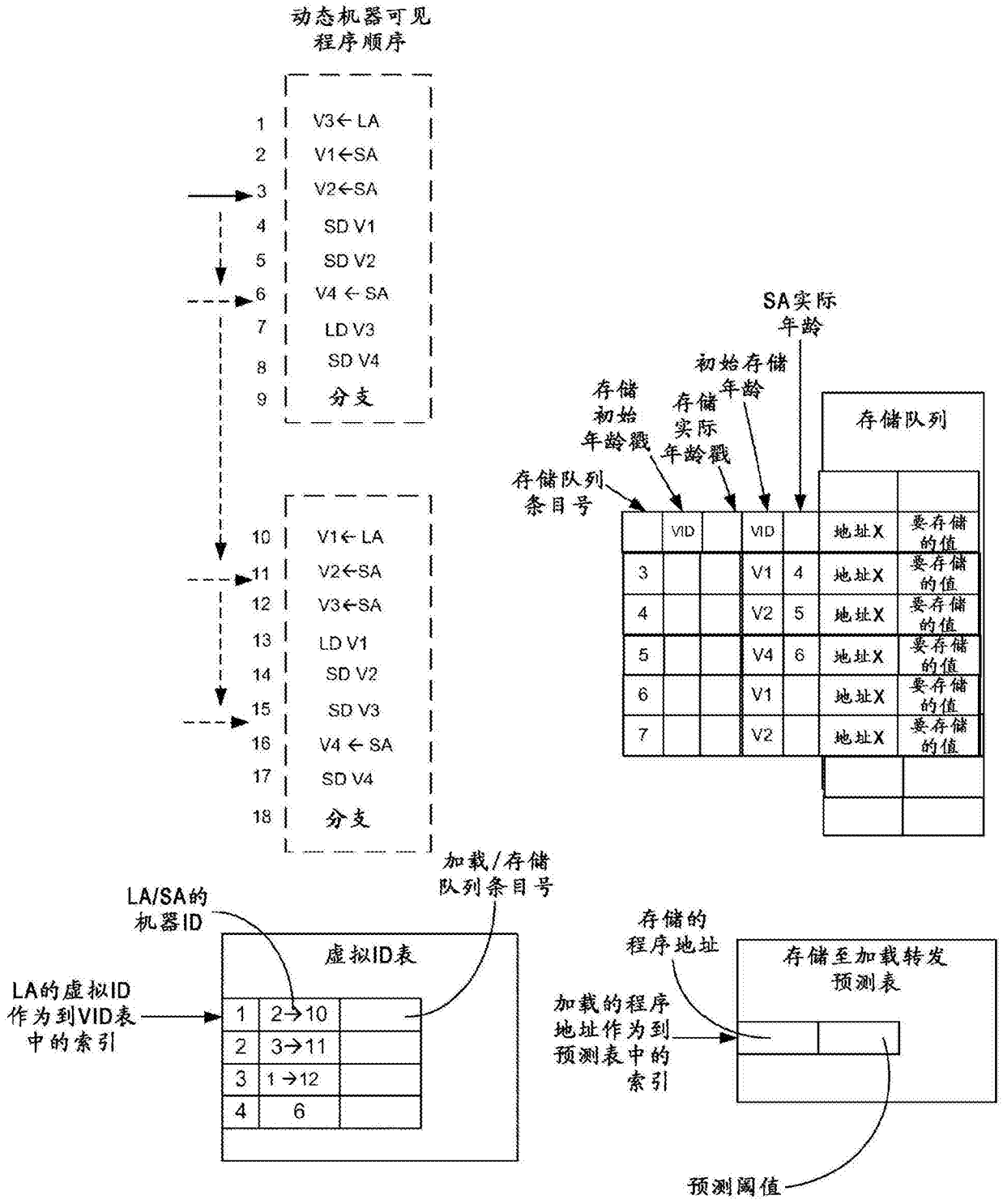

动态机器可见
程序顺序
1 V3← LA
2 V1←SA
3 V2←SA
4 SD V1
5 SD V2
6 V4 ← SA
7 LD V3
8 SD V4
9 分支
10 V1← LA
11 V2←SA
12 V3←SA
13 LD V1
14 SD V2
15 SD V3
16 V4 ← SA
17 SD V4
18 分支
SA实际
年龄
初始存储
年龄
存储
初始
年龄戳
存储
实际
年龄戳
存储队列
存储队列
条目号
VID
VID
地址X
要存储
的值
3
V1
4
4
V2
5
5
V4
6
6
V1
7
V2
加载/存储
队列条目号
LA/SA的
机器ID
虚拟ID表
LA的虚拟ID
作为到VID表
中的索引
1 2→10
2 3→11
3 1→12
4 6
存储的
程序地址
存储至加载转发
预测表
加载的程序
地址作为到
预测表中的
索引
预测阈值

图7

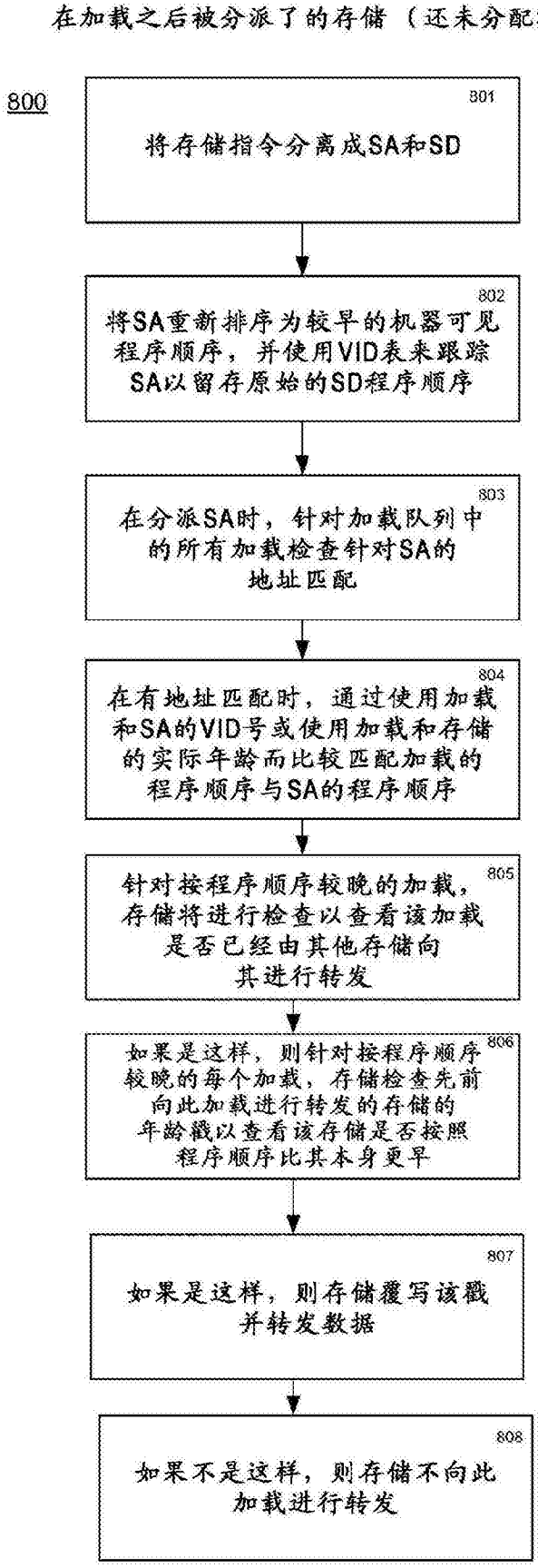

在加载之后被分派了的存储（还未分配SD）
800
801
将存储指令分离成SA和SD
802
将SA重新排序为较早的机器可见程序顺序，并使用VID表来跟踪SA以留存原始的SD程序顺序
803
在分派SA时，针对加载队列中的所有加载检查针对SA的地址匹配
804
在有地址匹配时，通过使用加载和SA的VID号或使用加载和存储的实际年龄而比较匹配加载的程序顺序与SA的程序顺序
805
针对按程序顺序较晚的加载，存储将进行检查以查看该加载是否已经由其他存储向其进行转发
806
如果是这样，则针对按程序顺序较晚的每个加载，存储检查先前向此加载进行转发的存储的年龄戳以查看该存储是否按照程序顺序比其本身更早
807
如果是这样，则存储覆写该戳并转发数据
808
如果不是这样，则存储不向此加载进行转发

图8

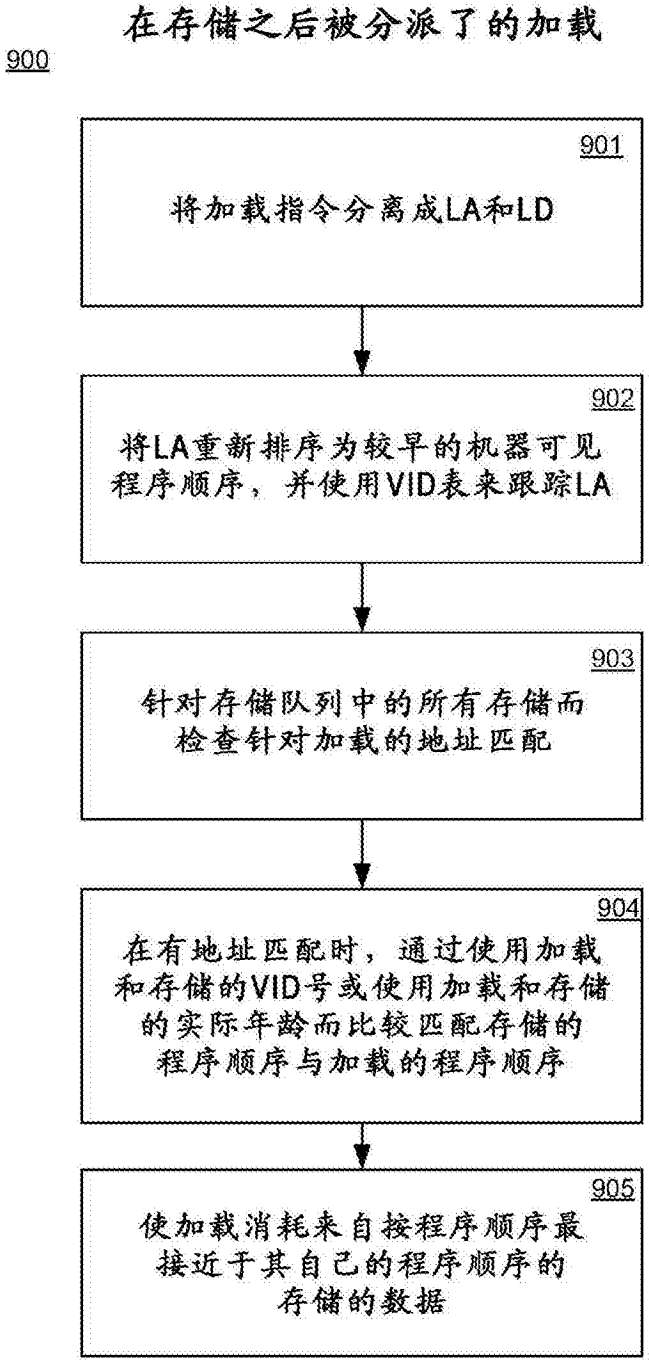

在存储之后被分派了的加载
900
901
将加载指令分离成LA和LD
902
将LA重新排序为较早的机器可见程序顺序，并使用VID表来跟踪LA
903
针对存储队列中的所有存储而检查针对加载的地址匹配
904
在有地址匹配时，通过使用加载和存储的VID号或使用加载和存储的实际年龄而比较匹配存储的程序顺序与加载的程序顺序
905
使加载消耗来自按程序顺序最接近于其自己的程序顺序的存储的数据

图9

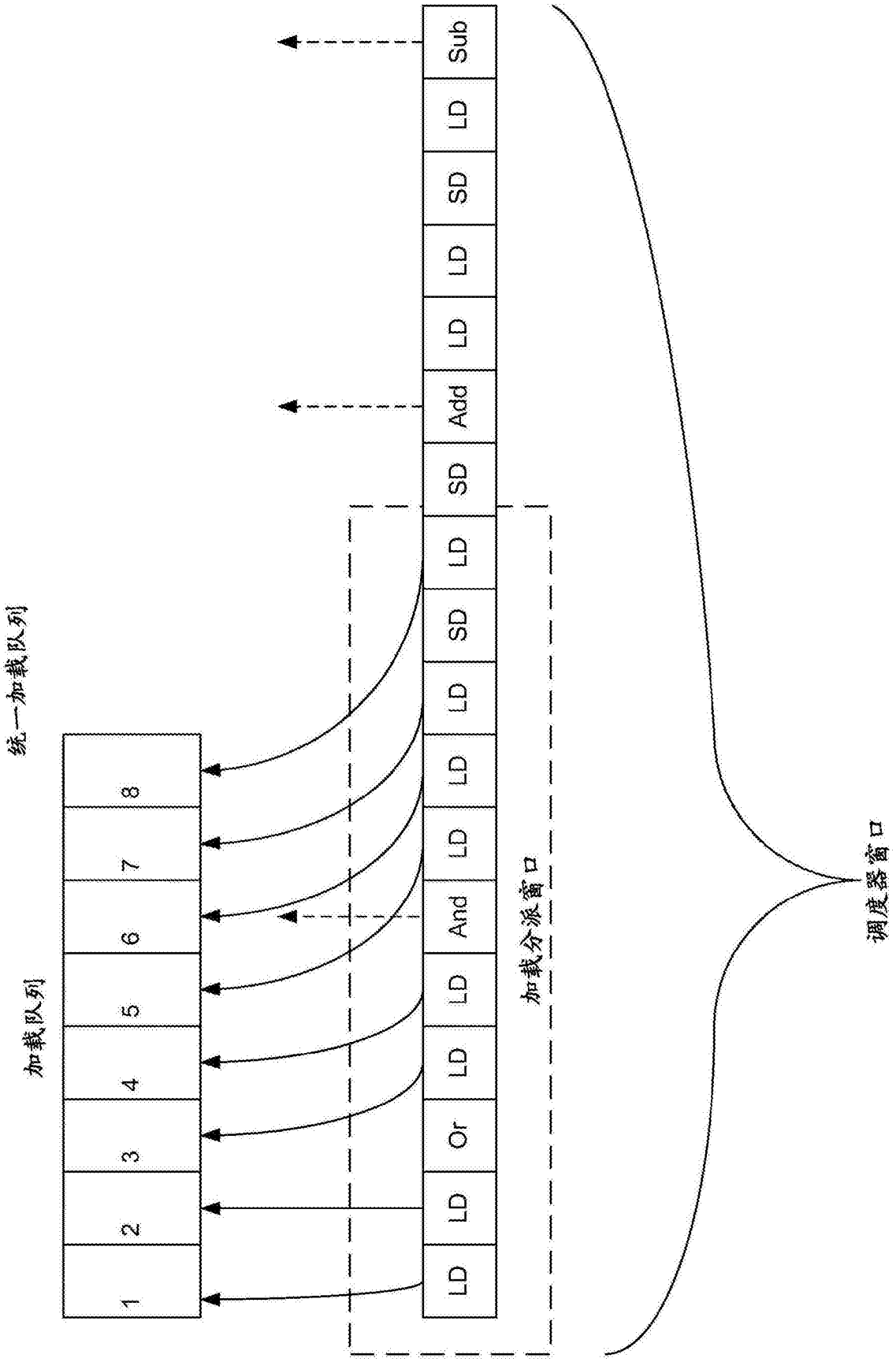

统一加载队列
加载队列
1
2
3
4
5
6
7
8
加载分派窗口
LD
LD
Or
LD
And
LD
LD
LD
SD
LD
Add
SD
LD
LD
SD
LD
Sub
调度器窗口

图10

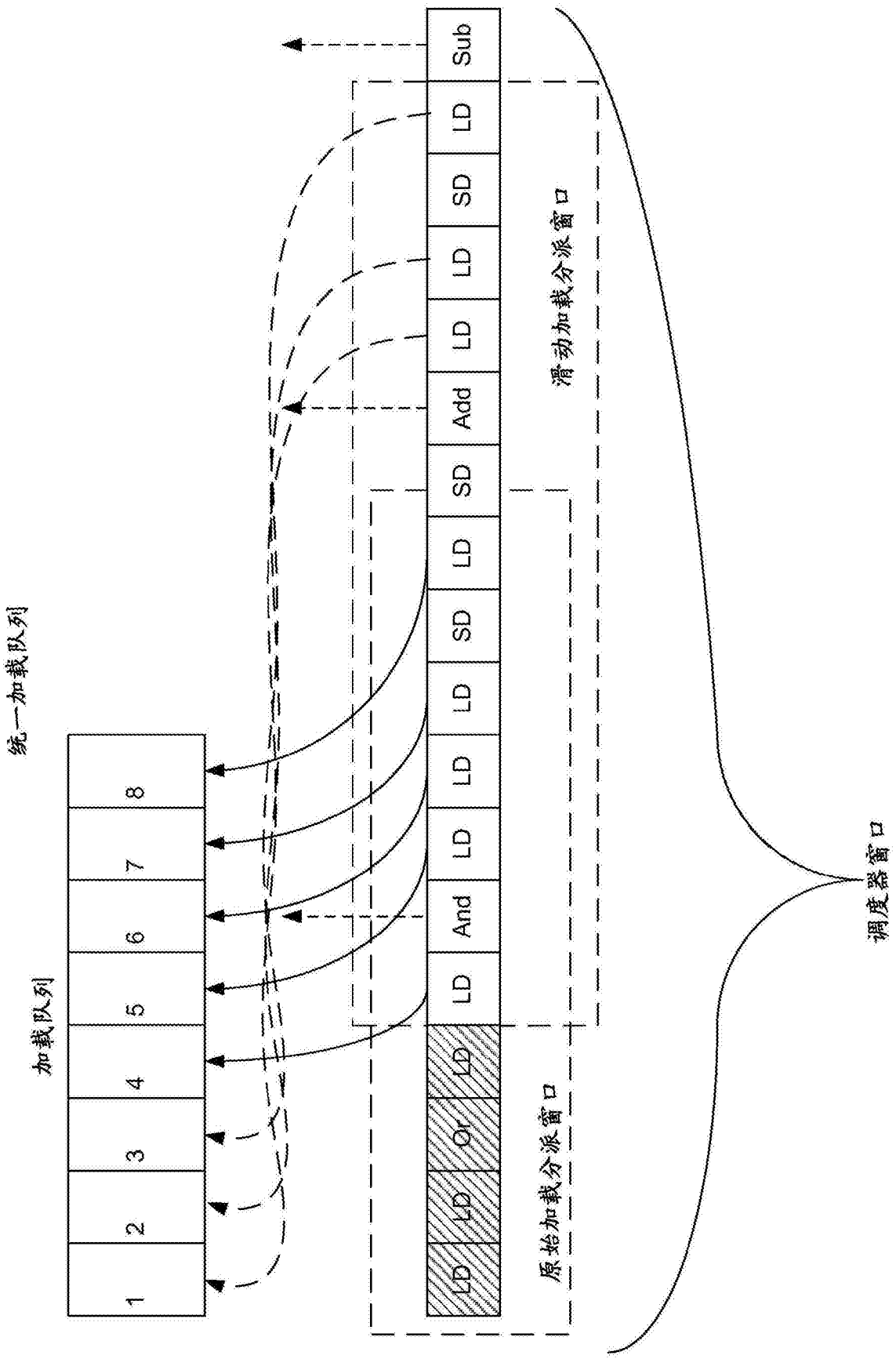

统一加载队列
加载队列
1
2
3
4
5
6
7
8
LD
And
LD
LD
SD
LD
SD
LD
Add
LD
SD
LD
Sub
原始加载分派窗口
滑动加载分派窗口
调度器窗口

图11

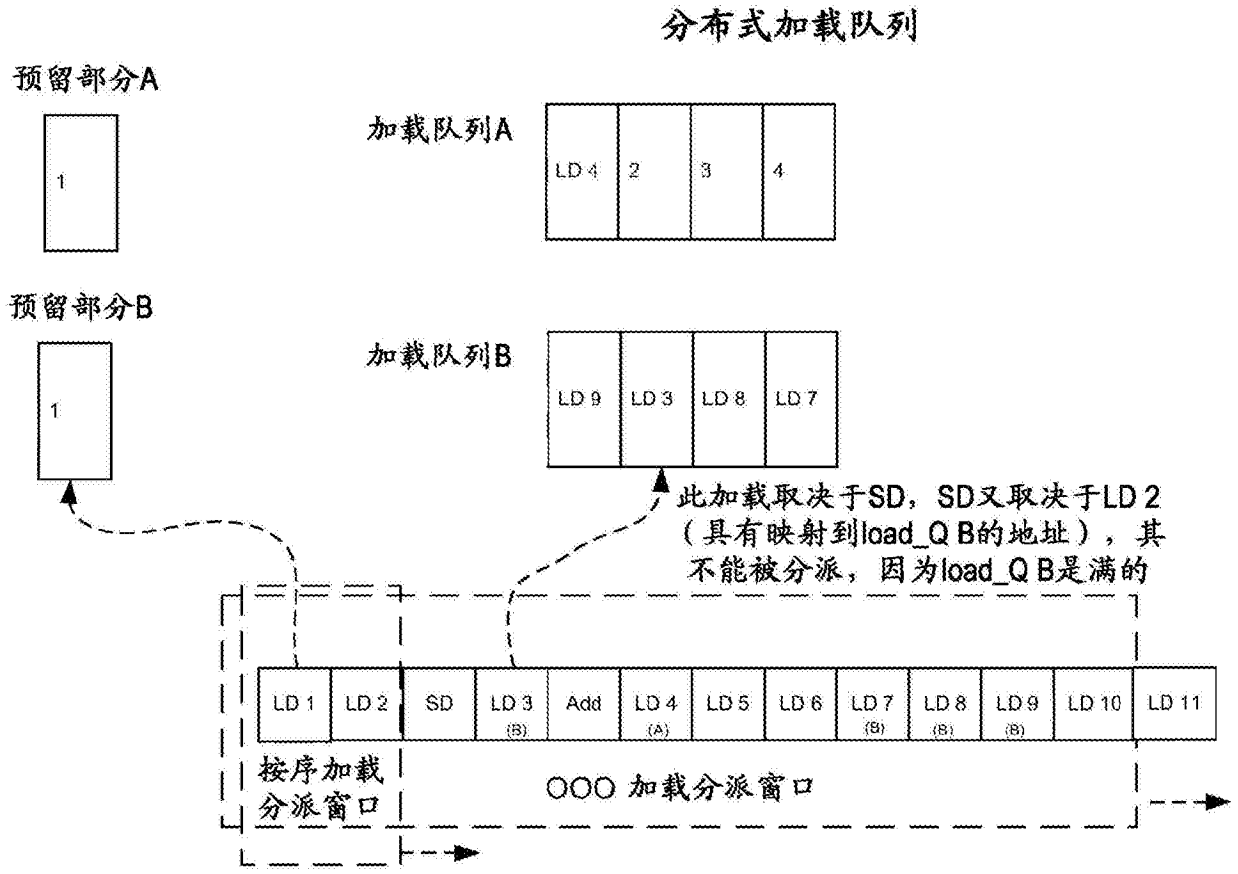

分布式加载队列
预留部分A
1
加载队列A
LD 4
2
3
4
预留部分B
1
加载队列B
LD 9
LD 3
LD 8
LD 7
此加载取决于SD，SD又取决于LD 2
（具有映射到load_Q B的地址），其
不能被分派，因为load_Q B是满的
LD 1
LD 2
SD
LD 3
(B)
Add
LD 4
(A)
LD 5
LD 6
LD 7
(B)
LD 8
(B)
LD 9
(B)
LD 10
LD 11
按序加载
分派窗口
OOO 加载分派窗口

图12

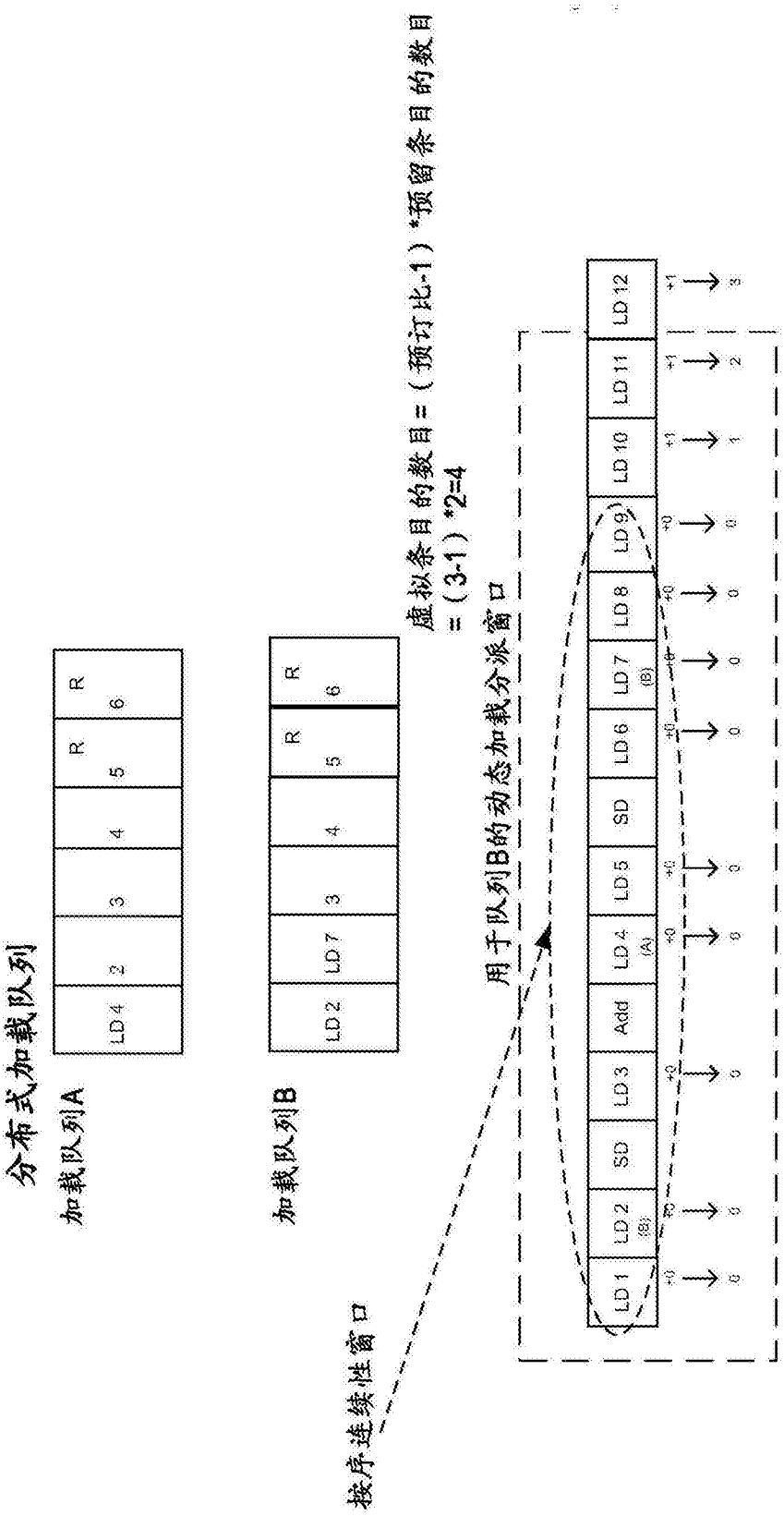

分布式加载队列
加载队列A
LD 4
2
3
4
R
5
R
6
加载队列B
LD 2
LD 7
3
4
R
5
R
6
虚拟条目的数目=（预订比-1）*预留条目的数目
=（3-1）*2=4
用于队列B的动态加载分派窗口
按序连续性窗口
LD 1
LD 2 (B)
SD
LD 3
Add
LD 4 (A)
LD 5
SD
LD 6
LD 7 (B)
LD 8
LD 9
LD 10
LD 11
LD 12

图13

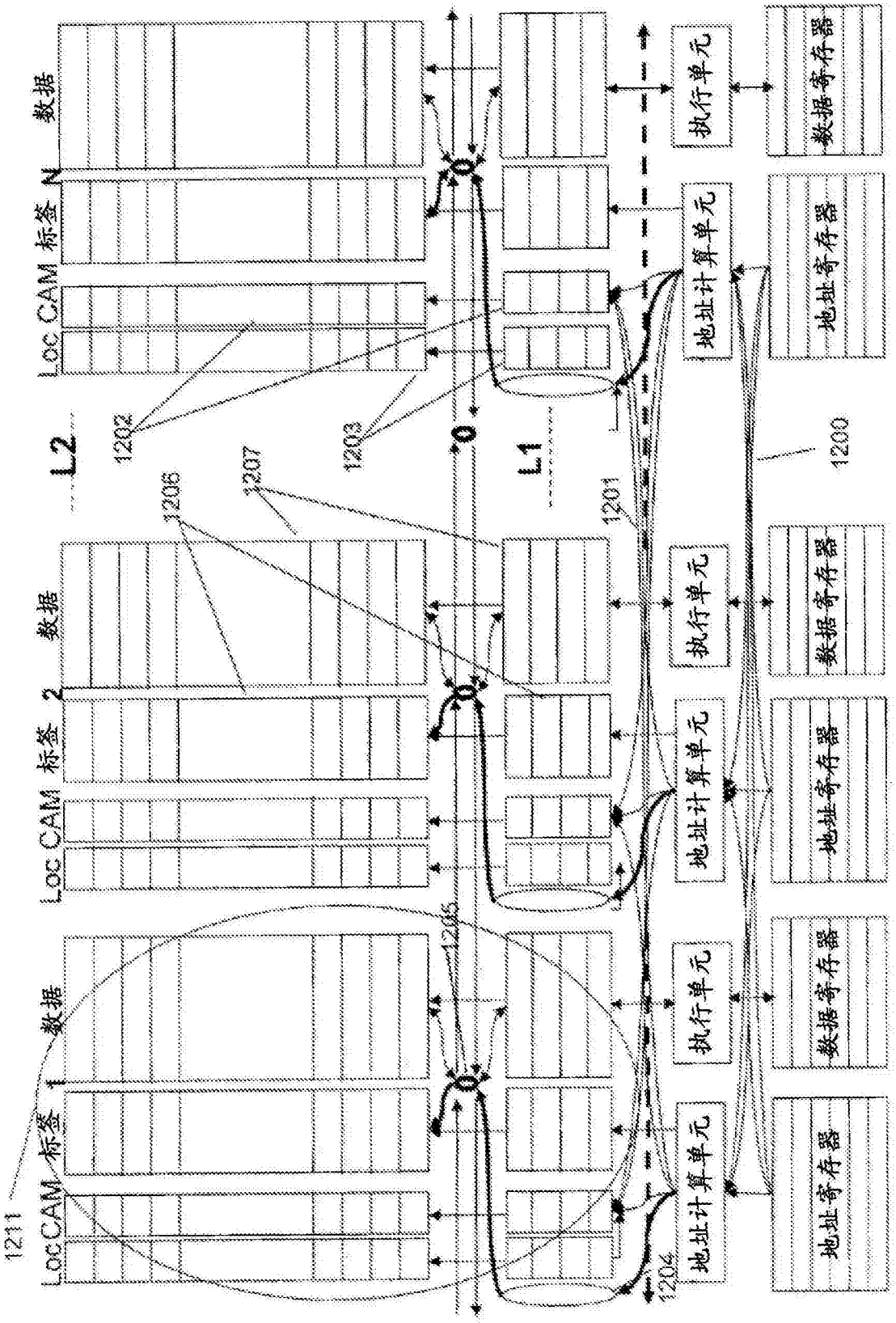

L2
L1
Loc CAM
标签
数据
地址计算单元
执行单元
地址寄存器
数据寄存器
1200
1201
1202
1203
1204
1205
1206
1207
1211

图14

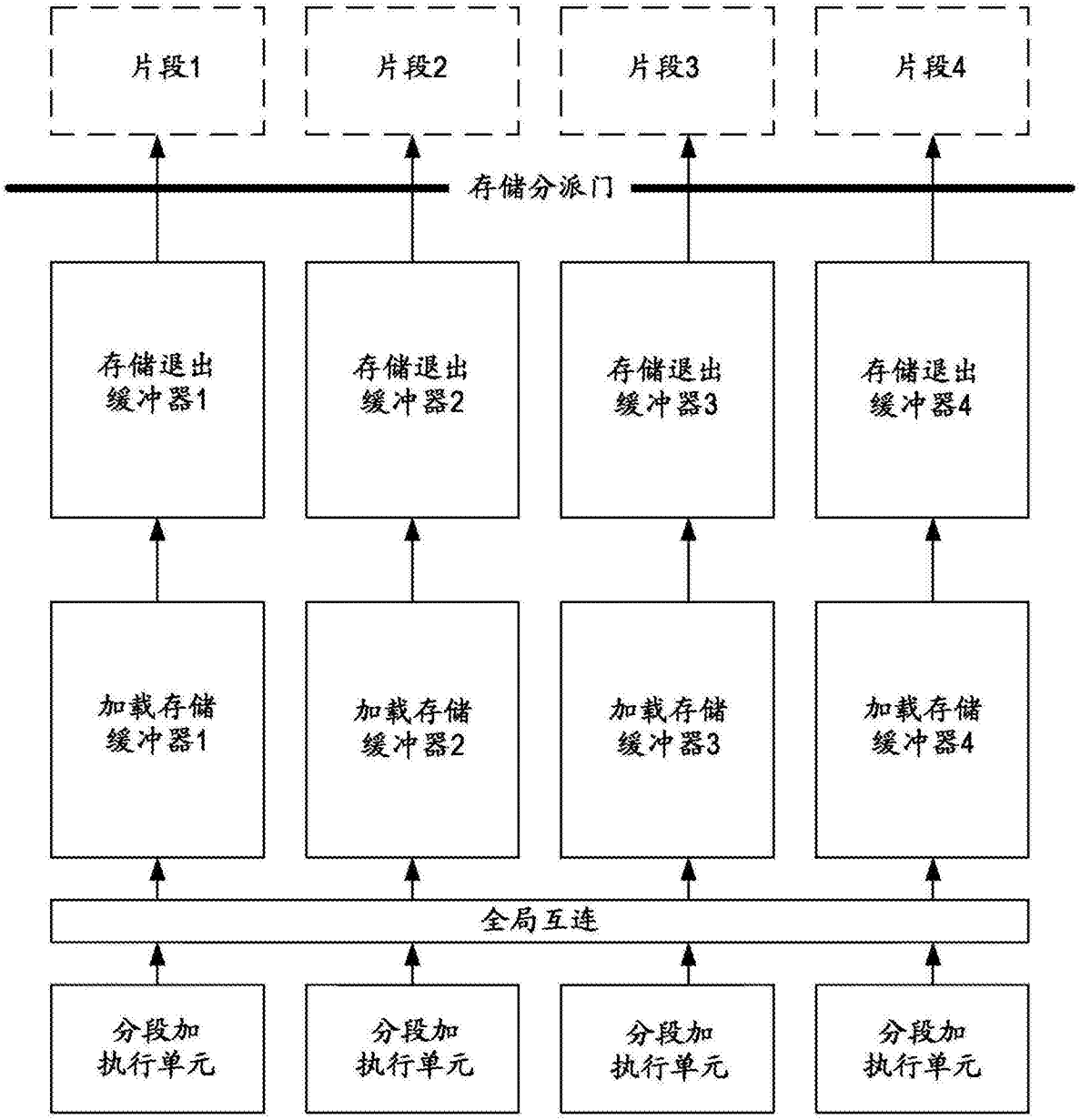

片段1
片段2
片段3
片段4
存储分派门
存储退出缓冲器1
存储退出缓冲器2
存储退出缓冲器3
存储退出缓冲器4
加载存储缓冲器1
加载存储缓冲器2
加载存储缓冲器3
加载存储缓冲器4
全局互连
分段加执行单元
分段加执行单元
分段加执行单元
分段加执行单元

图15

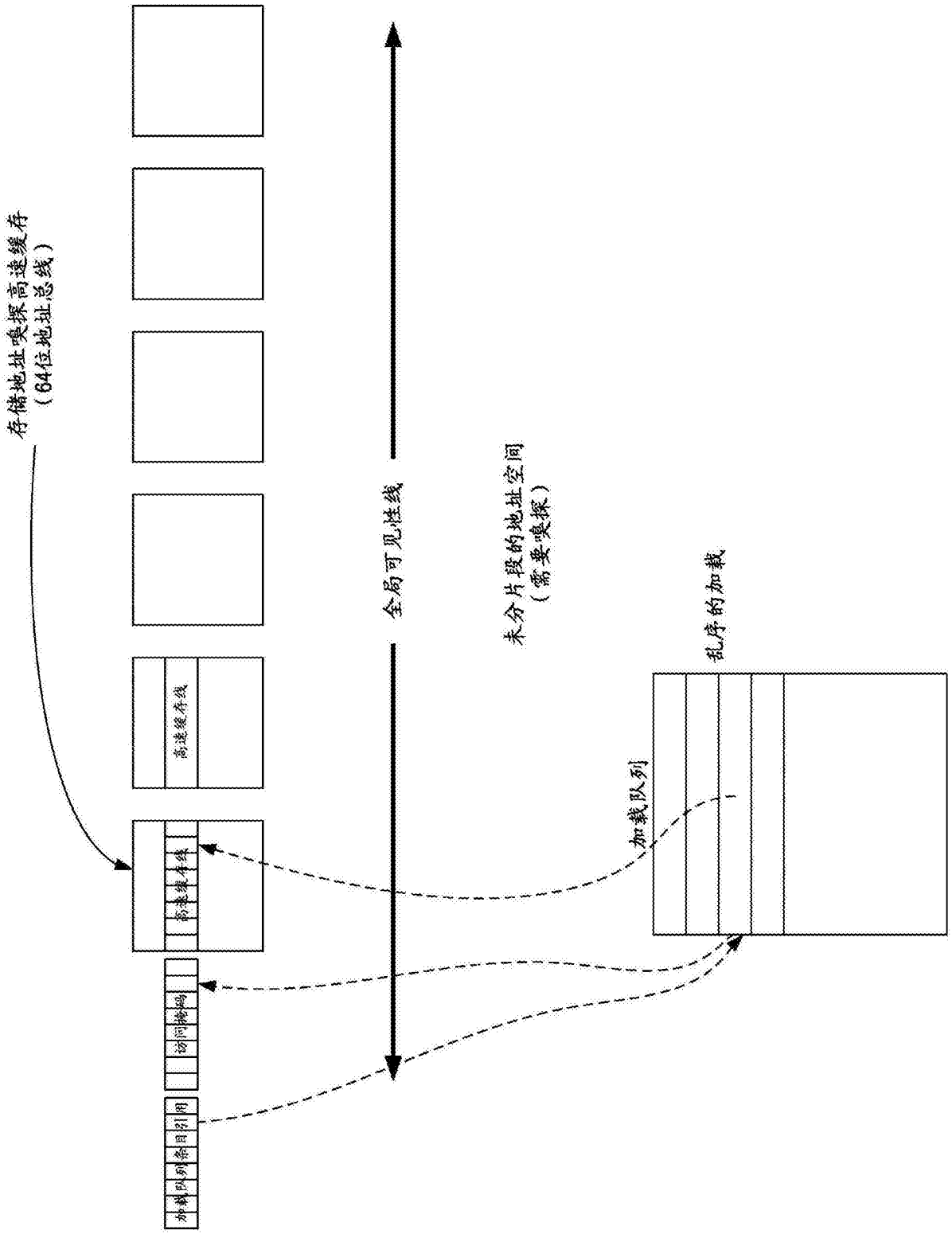

存储地址嗅探高速缓存
（64位地址总线）
高速缓存线
高速缓存线
访问掩码
加载队列条目引用
全局可见性线
未分片段的地址空间
（需要嗅探）
乱序的加载
加载队列

图16

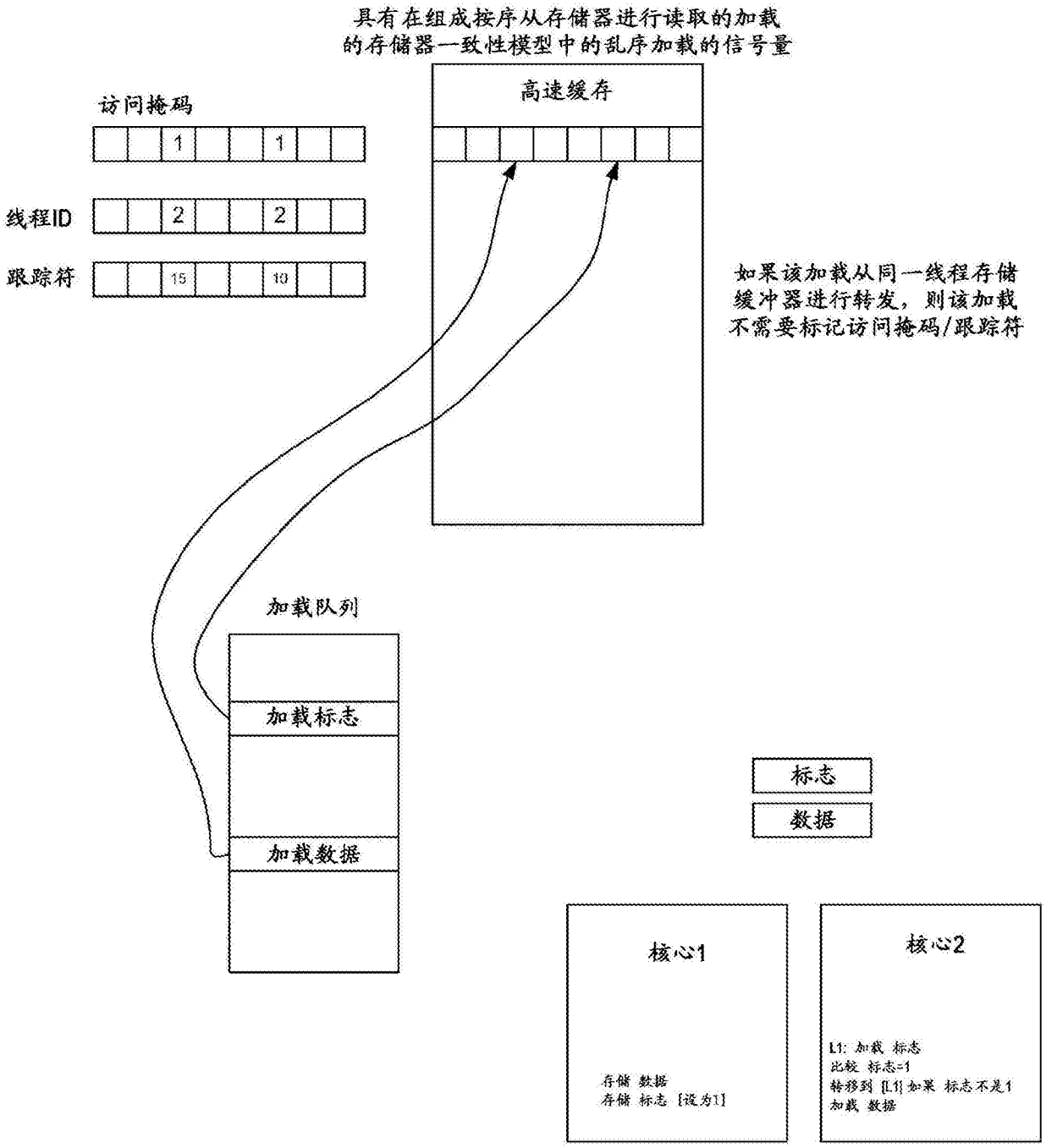

具有在组成按序从存储器进行读取的加载
的存储器一致性模型中的乱序加载的信号量
访问掩码
线程ID
跟踪符
高速缓存
如果该加载从同一线程存储
缓冲器进行转发，则该加载
不需要标记访问掩码/跟踪符
加载队列
加载标志
加载数据
标志
数据
核心1
存储 数据
存储 标志 [设为1]
核心2
L1: 加载 标志
比较 标志=1
转移到 [L1] 如果 标志不是1
加载 数据

图17

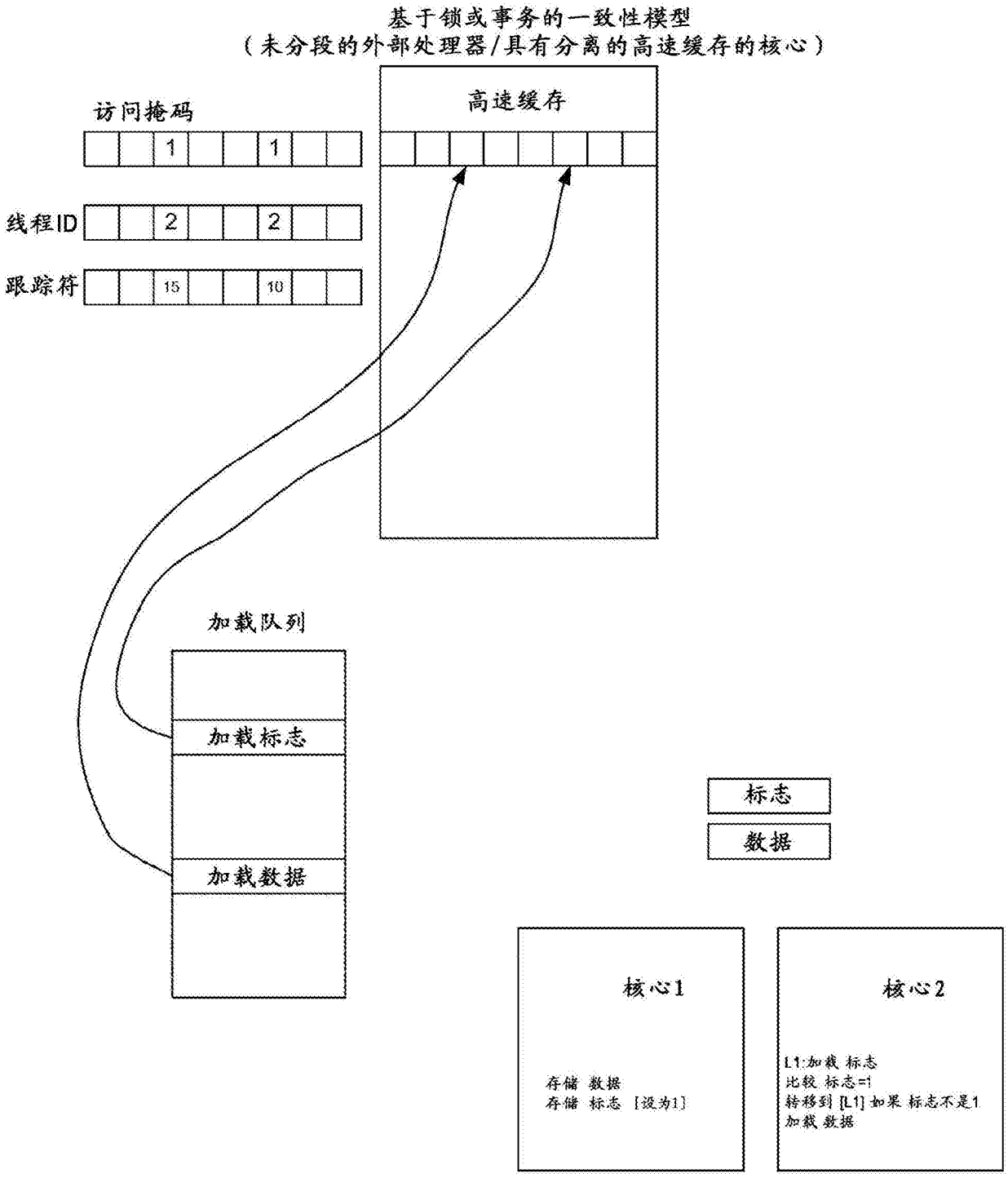

基于锁或事务的一致性模型
（未分段的外部处理器/具有分离的高速缓存的核心）
访问掩码
1
1
线程ID
2
2
跟踪符
15
10
高速缓存
加载队列
加载标志
加载数据
标志
数据
核心1
存储 数据
存储 标志 [设为1]
核心2
L1:加载 标志
比较 标志=1
转移到 [L1] 如果 标志不是1
加载 数据

图18

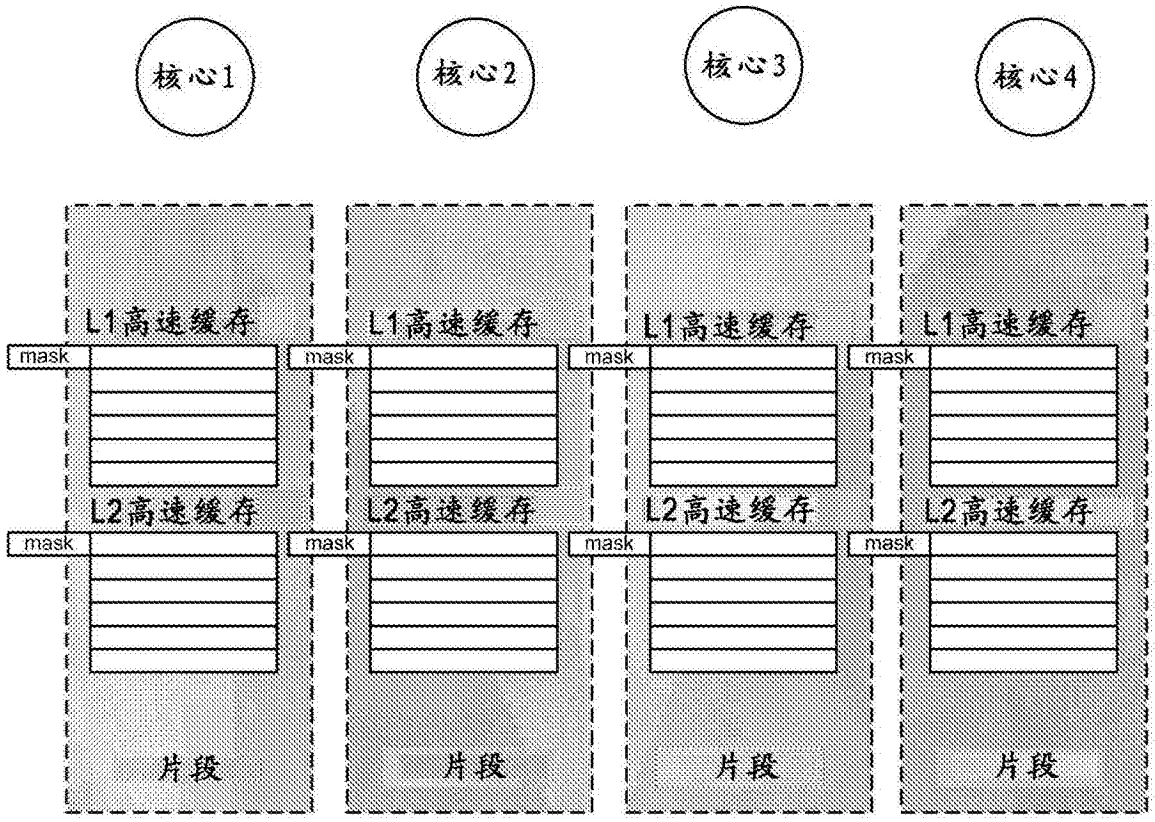


图19

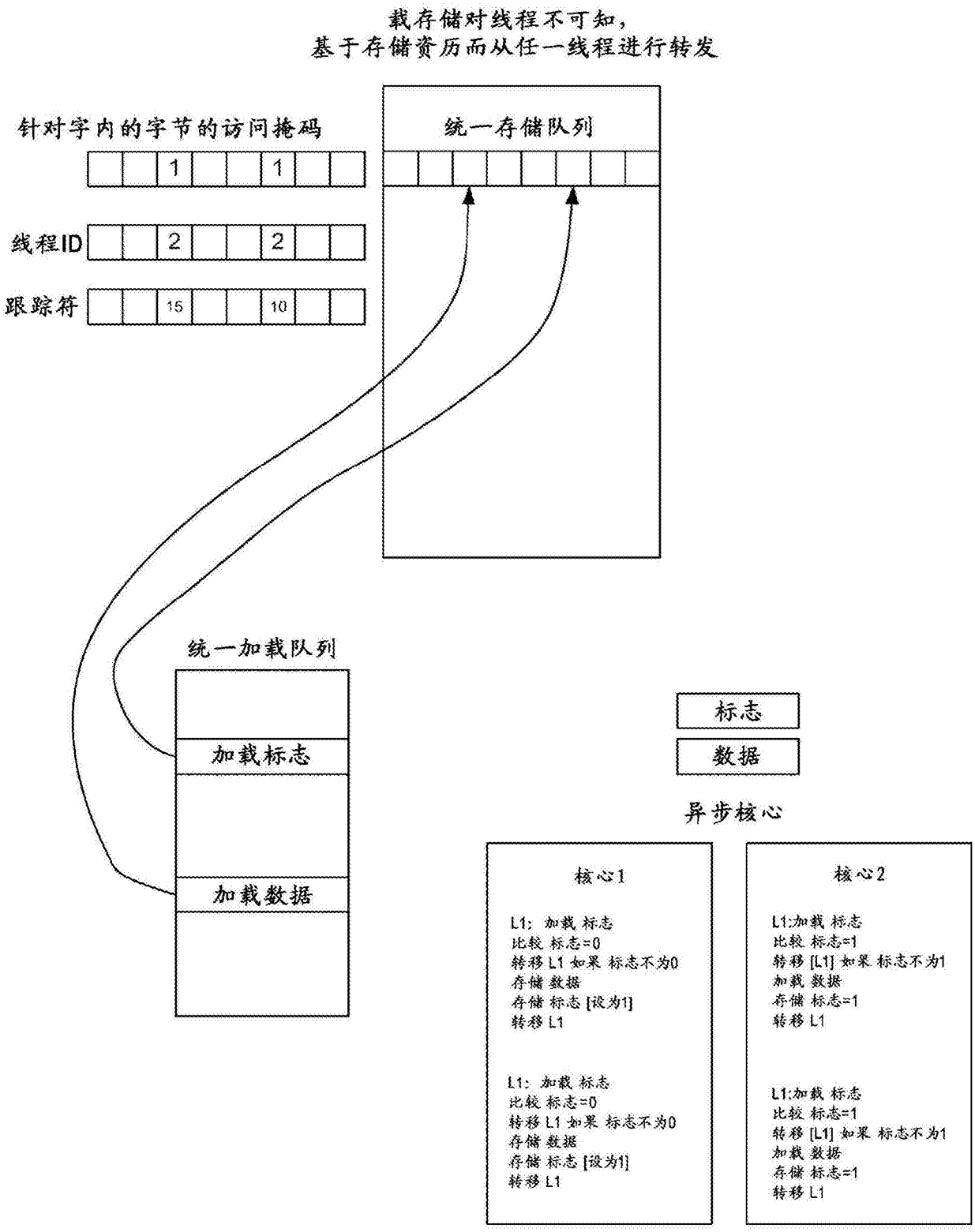

载存储对线程不可知，
基于存储资历而从任一线程进行转发
针对字内的字节的访问掩码
统一存储队列
1
1
线程ID
2
2
跟踪符
15
10
统一加载队列
加载标志
加载数据
标志
数据
异步核心
核心1
L1: 加载 标志
比较 标志=0
转移 L1 如果 标志不为0
存储 数据
存储 标志 [设为1]
转移 L1
L1: 加载 标志
比较 标志=0
转移 L1 如果 标志不为0
存储 数据
存储 标志 [设为1]
转移 L1
核心2
L1:加载 标志
比较 标志=1
转移 [L1] 如果 标志不为1
加载 数据
存储 标志=1
转移 L1
L1:加载 标志
比较 标志=1
转移 [L1] 如果 标志不为1
加载 数据
存储 标志=1
转移 L1

图20

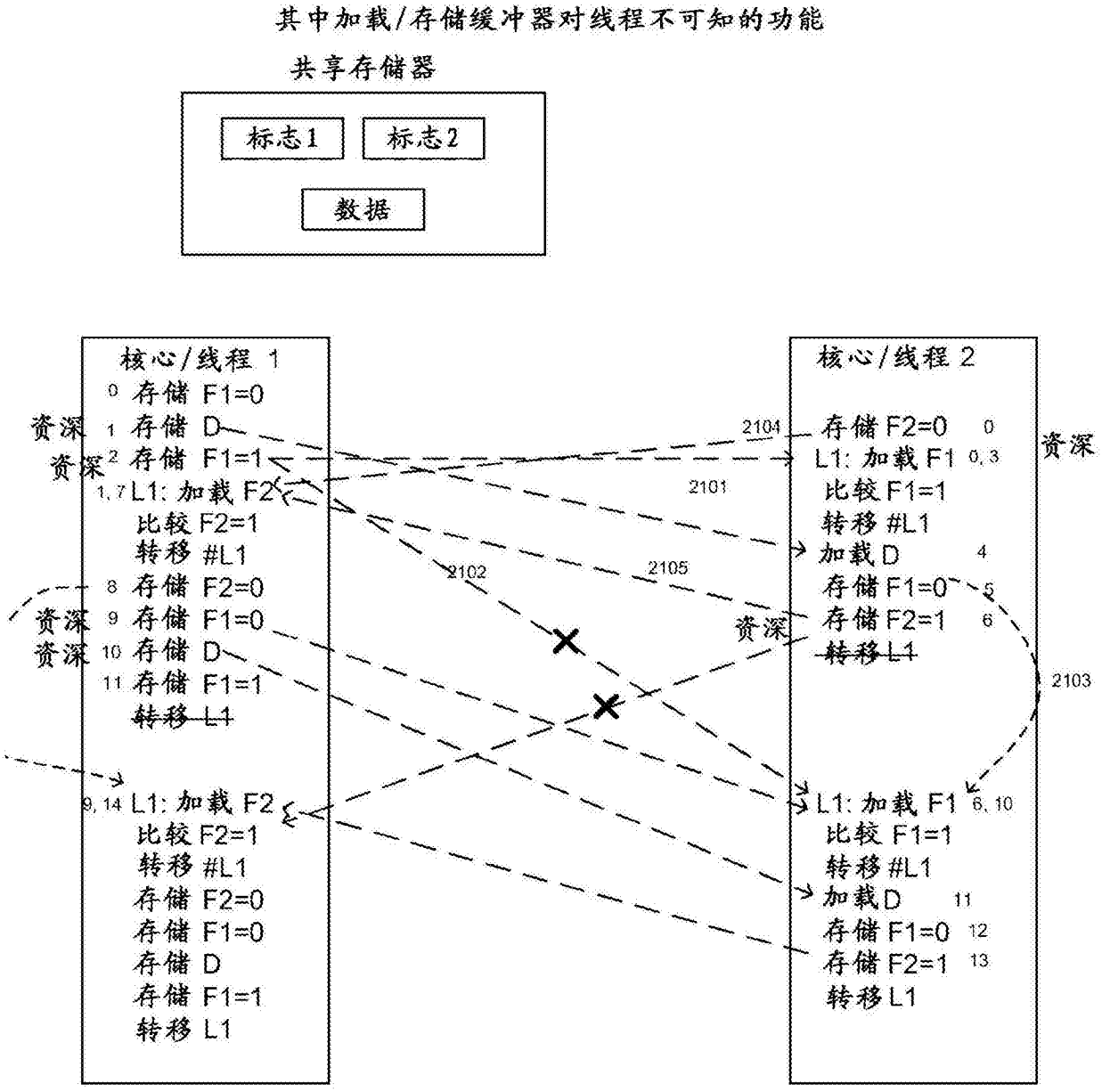

其中加载/存储缓冲器对线程不可知的功能
共享存储器
标志1
标志2
数据
核心/线程 1
0 存储 F1=0
资深 1 存储 D
资深 2 存储 F1=1
1, 7 L1: 加载 F2
比较 F2=1
转移 #L1
8 存储 F2=0
资深 9 存储 F1=0
资深 10 存储 D
11 存储 F1=1
转移 L1
9, 14 L1: 加载 F2
比较 F2=1
转移 #L1
存储 F2=0
存储 F1=0
存储 D
存储 F1=1
转移 L1
核心/线程 2
2104
存储 F2=0 0
L1: 加载 F1 0, 3
比较 F1=1
转移 #L1
加载 D 4
存储 F1=0 5
存储 F2=1 6
转移 L1
资深
资深
2101
2102
2105
2103
L1: 加载 F1 6, 10
比较 F1=1
转移 #L1
加载 D 11
存储 F1=0 12
存储 F2=1 13
转移 L1

图21

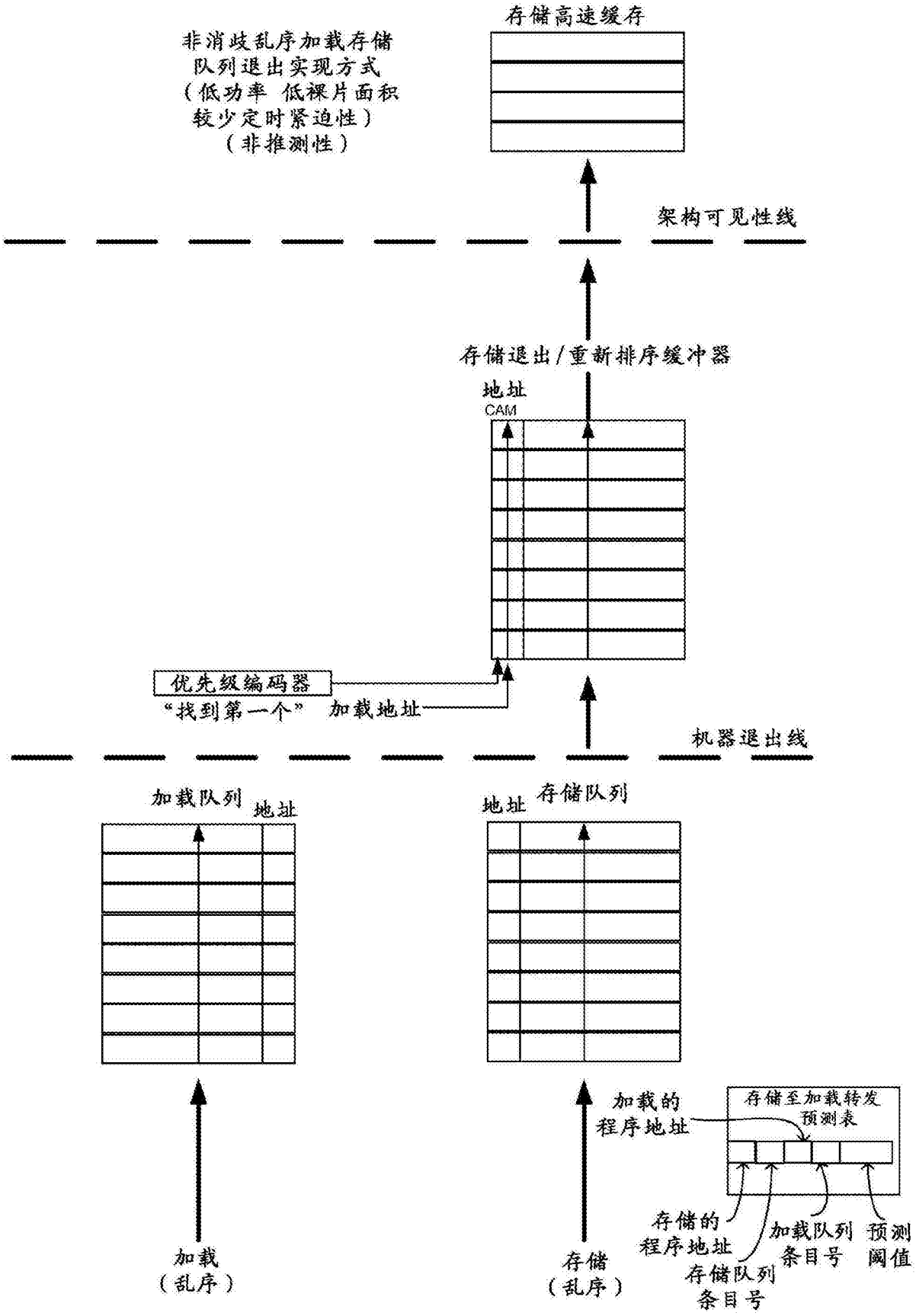

非消歧乱序加载存储
队列退出实现方式
（低功率 低裸片面积
较少定时紧迫性）
（非推测性）
存储高速缓存
架构可见性线
存储退出/重新排序缓冲器
地址
CAM
优先级编码器
“找到第一个”
加载地址
机器退出线
加载队列
地址
地址
存储队列
加载的
程序地址
存储至加载转发
预测表
存储的
程序地址
加载队列
条目号
预测
阈值
存储队列
条目号
加载
（乱序）
存储
（乱序）

图22

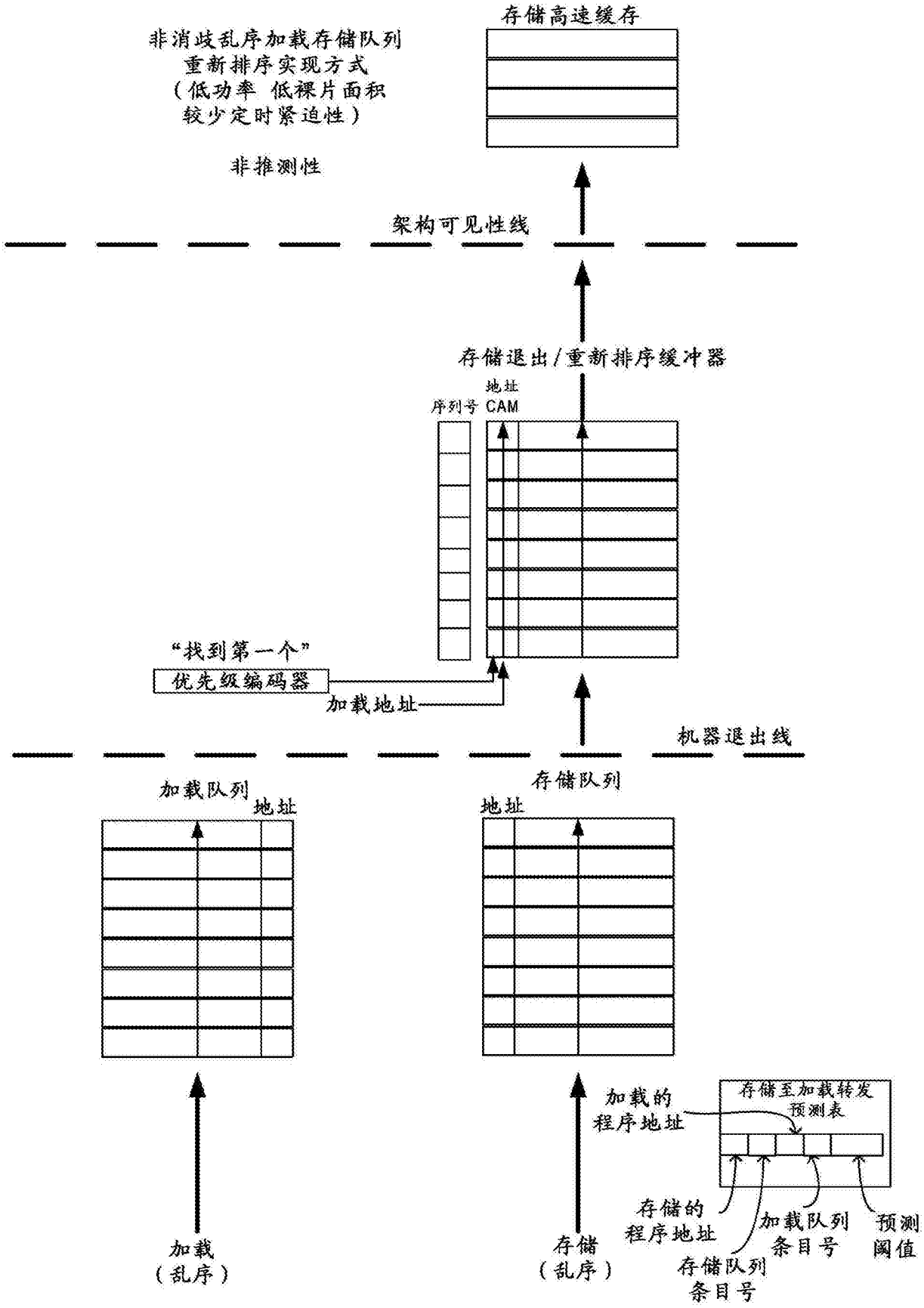

非消歧乱序加载存储队列
重新排序实现方式
（低功率 低裸片面积
较少定时紧迫性）
非推测性
存储高速缓存
架构可见性线
存储退出/重新排序缓冲器
地址
序列号 CAM
“找到第一个”
优先级编码器
加载地址
机器退出线
加载队列
地址
存储队列
地址
加载的
程序地址
存储至加载转发
预测表
存储的
程序地址
加载队列
条目号
预测
阈值
存储队列
条目号
加载
（乱序）
存储
（乱序）

图23

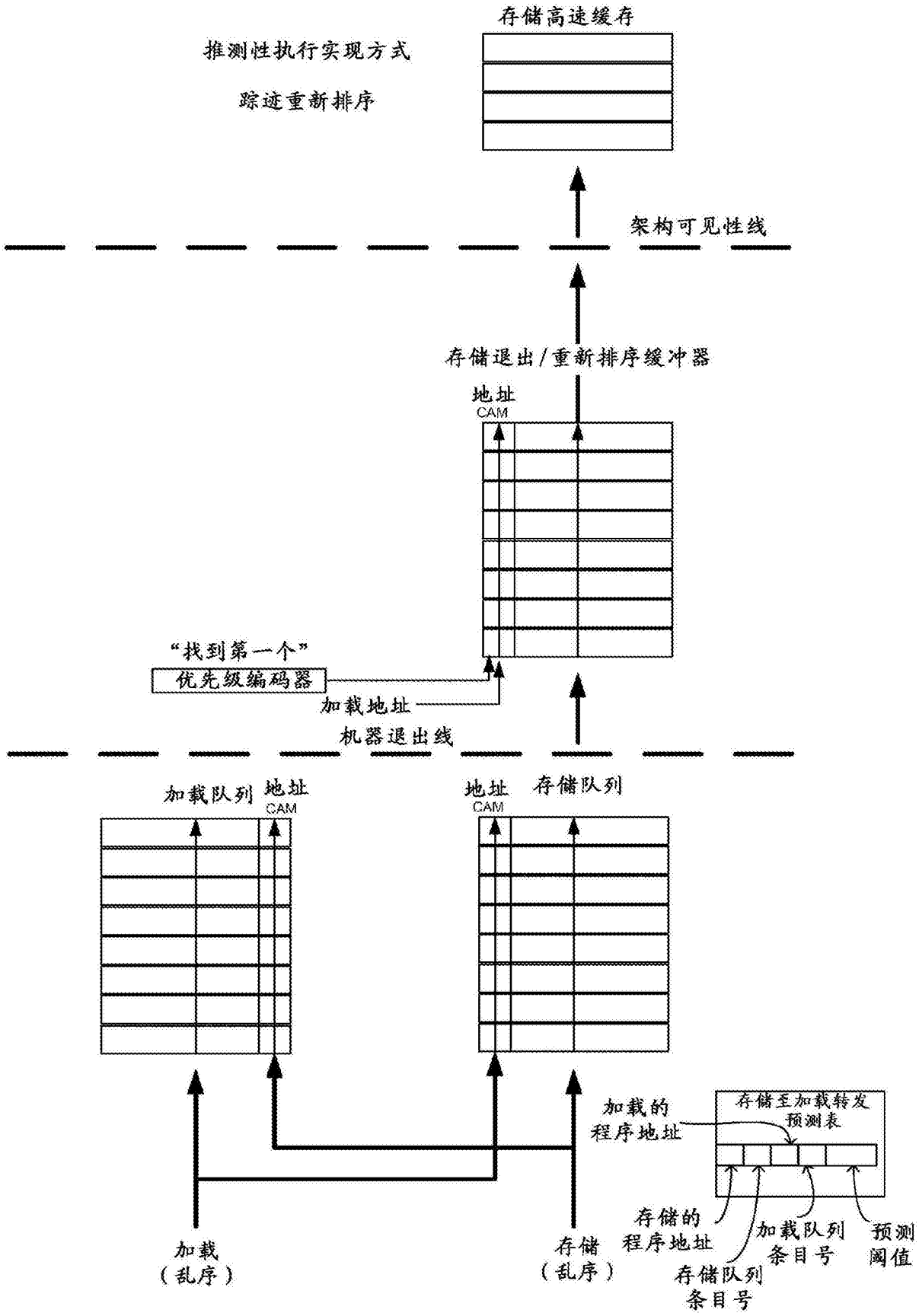

存储高速缓存
推测性执行实现方式
踪迹重新排序
架构可见性线
存储退出/重新排序缓冲器
地址
CAM
“找到第一个”
优先级编码器
加载地址
机器退出线
加载队列
地址
CAM
地址
CAM
存储队列
加载的
程序地址
存储至加载转发
预测表
存储的
程序地址
加载队列
条目号
预测
阈值
存储队列
条目号
加载
（乱序）
存储
（乱序）

图24

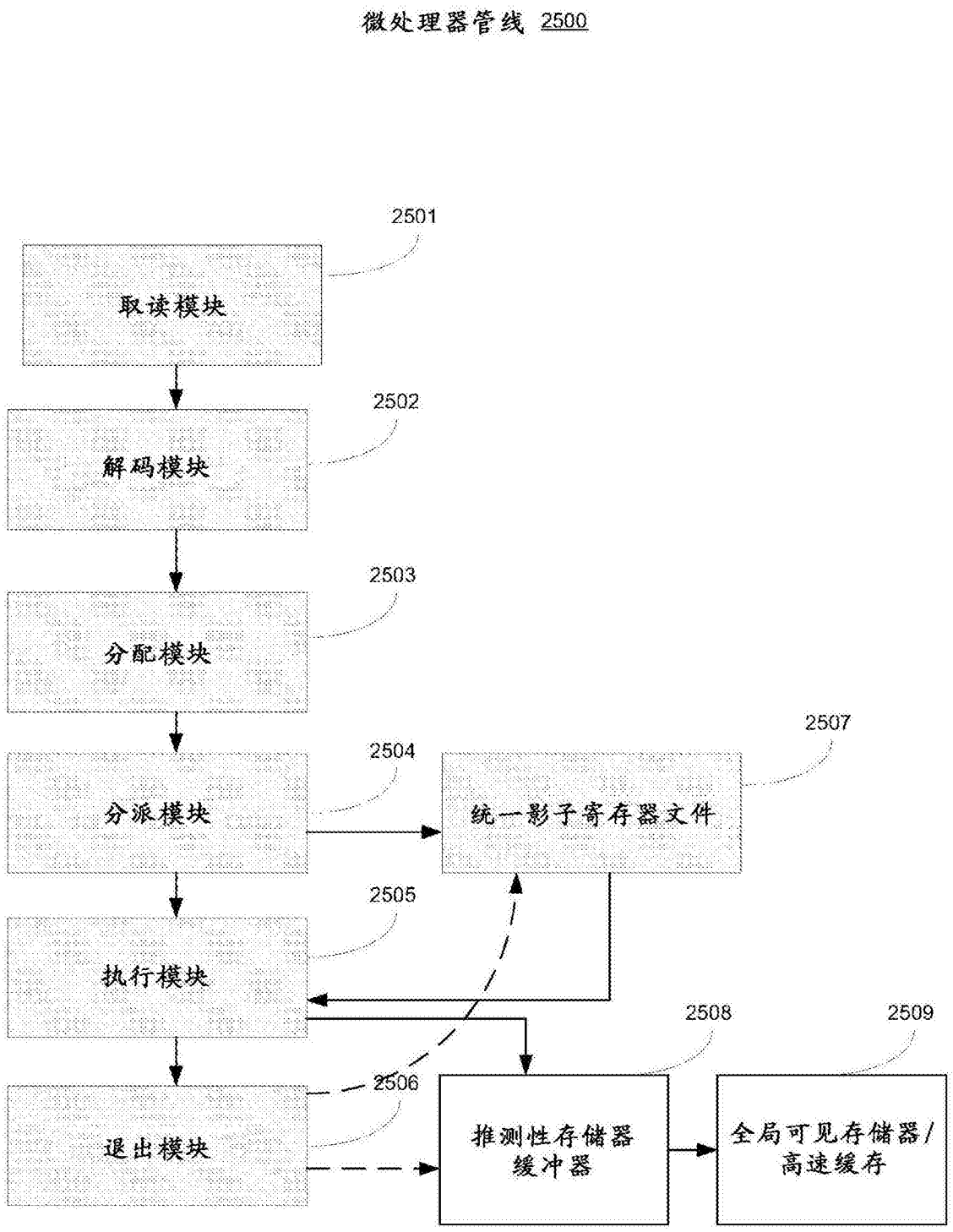

微处理器管线 2500
2501
取读模块
2502
解码模块
2503
分配模块
2504
分派模块
2507
统一影子寄存器文件
2505
执行模块
2508
2509
2506
退出模块
推测性存储器
缓冲器
全局可见存储器/
高速缓存

图25