[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 12/12 (2006.01)

[12] 发明专利申请公开说明书

[21] 申请号 200480021817.1

[43] 公开日 2006 年 9 月 6 日

[11] 公开号 CN 1829979A

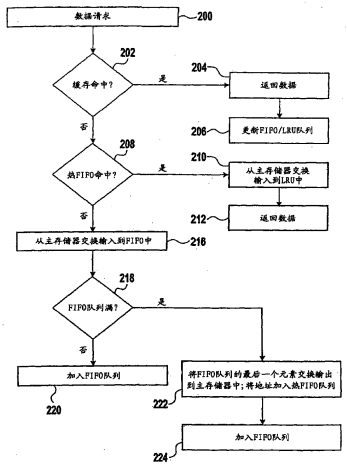
[22] 申请日 2004.7.5
[21] 申请号 200480021817.1
[30] 优先权
[32] 2003. 8. 5 [33] EP [31] 03017835.4
[86] 国际申请 PCT/EP2004/007315 2004.7.5
[87] 国际公布 WO2005/015408 英 2005.2.17
[85] 进入国家阶段日期 2006.1.26
[71] 申请人 SAP 股份公司
地址 德国瓦尔多夫
[72] 发明人 伊凡·施雷特
[74] 专利代理机构 北京市柳沈律师事务所
代理人 邵亚丽 李晓舒
权利要求书 2 页 说明书 8 页 附图 2 页

[54] 发明名称

数据缓存方法

[57] 摘要

本发明涉及既使用 FIFO 缓存替换算法也使用 LRU 缓存替换算法的数据缓存方法。 此外，将辅助 FIFO 队列用于缓存管理。 结果，缓存命中率增加了，这对整体系统性能有正面影响。



1. 一种数据缓存方法，包含步骤:

a） 建立 FIFO 队列和 LRU 队列，

b） 为已经被交换输出到外部存储器的缓存线的地址建立辅助 FIFO 队列，

c） 在对于数据请求存在缓存错失的情况下：确定所述辅助 FIFO 队列中是否存在命中，并且，如果如此，则将数据交换输入到所述 LRU 队列中，否则将数据交换输入到所述 FIFO 队列中。

2. 如权利要求 1 所述的方法，其中，所述 FIFO 队列具有第一预定义最大大小，并且所述 LRU 队列具有第二预定义最大大小，其中所述第一预定义最大大小在所述第二预定义最大大小以下。

3. 如权利要求 1 或 2 所述的方法，其中，在所述辅助 FIFO 队列中存在错失的情况下，执行下列步骤:

- 从外部存储器交换输入数据，

- 将所述交换输入的数据加入所述 FIFO 队列，

- 如果所述 FIFO 队列已经达到预定义的最大大小：将元素从所述 FIFO 队列交换输出到所述辅助 FIFO 队列中。

4. 如权利要求 1、2 或 3 所述的方法，其中，所述第一预定义最大大小在所述缓存存储器的大小的 5％和 25％之间，最好为大约 10%。

5. 如权利要求 1 到 4 中的任何一个所述的方法，其中，所述 LRU 队列的所述第二预定义最大大小在所述缓存存储器大小的 75％和 95％之间，最好是 90％。

6. 一种缓存存储器，包含:

- 用于建立 FIFO 队列和 LRU 队列的装置（110、114），

- 用于为已经被交换输出的缓存线的地址建立辅助 FIFO 队列的装置（112、114），

- 用于确定所述辅助 FIFO 队列中是否存在命中，并且在命中的情况下将对应数据从外部存储器交换输入到所述 LRU 队列中的装置（114）。

7. 如权利要求 6 所述的缓存存储器，其中，所述 FIFO 队列具有第一预定义最大大小，并且所述 LRU 队列具有第二预定义最大大小，其中所述第一

预定义最大大小在所述第二预定义最大大小以下。

8. 如权利要求 6 或 7 所述的缓存存储器，还包含装置（114），用于执行下列步骤:

- 在所述辅助 FIFO 队列中存在错失的情况下，从外部存储器（108）交换输入数据，

- 将所述交换输入的数据加入所述 FIFO 队列，

- 如果所述 FIFO 队列已经达到预定义的最大大小：将元素从所述 FIFO 队列交换输出到外部存储器，并将其地址放入辅助 FIFO 队列。

9. 如权利要求 6、7 或 8 所述的缓存存储器，其中，所述第一预定义最大大小在所述缓存存储器的大小的 5％和 25％之间，最好为大约 10%。

10. 如权利要求 6 到 9 中的任何一个所述的缓存存储器，其中，所述 LRU 队列的所述第二预定义最大大小在所述缓存存储器大小的 75％和 95％之间，最好是 90％。

11. 一种数据处理系统，包含:

- 至少一个处理器（102），用于运行应用程序（104），

- 外部存储器（108），

- 耦合在所述处理器和所述外部存储器之间的缓存模块（106），所述缓存存储器具有:

a） 用于建立 FIFO 队列和 LRU 队列的装置（110、114），

b） 用于为已经被交换输出到外部存储器的缓存线地址建立辅助 FIFO 队列的装置（112、114），

c） 用于确定所述辅助 FIFO 队列中是否存在命中，并且在命中的情况下将对应数据从所述外部存储器交换输入到所述 LRU 队列中的装置（114）。

12. 如权利要求 11 所述的数据处理系统，所述外部存储器是主存储器或基于磁盘的储存设备。

13. 如权利要求 11 或 12 所述的数据处理系统，所述缓存线是来自数据库的同等大小的数据页面或数据块。

数据缓存方法

技术领域

本发明涉及数据处理领域，并特别涉及数据缓存。

背景技术

在计算技术中，缓存存储器用来储存主存储器的可能很快就会被用到一部分存储器内容。如这里所使用的，术语“缓存（cache）”也将用来指缓存存储器（cache memory）。缓存通常比主存储器更小且更快，并被用来掩蔽(mask)从主存储器取回存储器操作数时所涉及的延迟（latency）。在现代计算机系统中，缓存存取时间通常比主存储器存取时间快大约500％到3000％。

缓存的单元（entry）在技术上称为缓存线（cache line），通常缓存线将储存小的连续范围主存储器内容，例如32或64个字节。虽然缓存存储器不限于CPU，但是缓存存储器的基本应用却是储存一个或更多个中央处理单元（CPU）所需的存储器操作数。另一种常见的缓存使用是在数据库系统中。数据库系统缓存在计算机存储器（计算机存储器起到来自磁盘的数据页面的缓存的作用）中缓存来自基于磁盘的储存装置（类似于主存储器）的同等大小的数据页面或数据块（类似于缓存线）。

注意，提供多级缓存在技术上是公知的。例如，可以在CPU所处同一集成电路上给CPU提供第一级（L1）缓存，并在CPU所处同一模块中提供较大且较慢的第二级（L2）缓存。在接下来的讨论中，将假设从主存储器中把存储器操作数加载到缓存中。但是，熟练技术人员将认识到，如果在更高级缓存中存在操作数，也可以从更高级缓存加载这样的操作数。

由于缓存存储器通常比它们耦合到的主存储器小，所以需要用于确定要把主存储器的哪些内容储存在缓存中的策略。这种策略通常包括两个组成部分：缓存结构和缓存替换算法。替换算法确定当缓存（或如下面所描述的关联缓存集）变满时应该替换哪一个缓存线。

最简单的缓存结构之一是直接映射（direct-mapped）缓存结构。在直接映射缓存中，主存储器地址的一部分被用作索引，并且，主存储器地址的剩

余部分（不包括代表缓存线内的字节的主存储器地址的任何位）被用作标签。用于索引的位数与缓存的大小对应。例如，具有 64 个缓存线的直接映射缓存将具有包含 6 位的索引。当发生读操作并且存储器操作数不在缓存中时（即标签不匹配），存储器操作数被从主存储器取出并储存在对应于所述索引的缓存线中，并且，标签被储存在和缓存线相关联的标签字段中。

假设下一次发生读操作时存储器操作数仍在缓存中（即标签匹配），则将从缓存中取回(retrieve)该存储器操作数。顺便说明，在技术上使用术语“缓存命中”(cache hit)来指所需存储器操作数已经在缓存中的存储器访问，而在技术上使用术语“缓存错失”(cache miss)来指存储器操作数不在缓存中并且必须从主存储器或更高级缓存加载的存储器访问。

与直接映射缓存一起使用的替换算法是简单的。对于主存储器中的任何给定字节，只有一个其中可以储存该字节的缓存线。因此，如果缓存线正在使用，则缓存线的旧内容被新内容简单地覆盖。在缓存线已经被从存储器加载之后更改缓存线内容的操作在技术上称为“弄脏”(dirtying)该缓存线。在新内容可以被储存在“弄脏的”(dirty)缓存线中之前，该“弄脏的”缓存线必须被写回主存储器。如果缓存线中的旧内容与主存储器中的内容相同，则可以覆盖旧内容而不必将旧内容回写到主存储器。

与直接映射缓存存储器相关联的一个问题是：两个经常使用的存储器操作数可能需要被储存在同一缓存线中。由于这两个存储器操作数将竞争同一缓存线，所以当这两个操作数持续地彼此替换时，将丧失缓存所提供的很多优点。

另一种缓存结构是关联缓存(associative cache)结构。完全关联缓存(fullassociative cache)就是具有缓存线池(cache line pool)，并且存储器操作数可以储存在任何缓存线中。当存储器操作数储存在关联缓存中时，该存储器操作数的地址（不包括代表储存在该缓存线中的字节的任何位）被储存在与缓存线相关联的标签字段中。无论何时只要发生存储器操作，则搜索与每一个缓存线相关联的标签字段，以察看该存储器操作数是否储存在缓存中。关联缓存的一个缺点是所有缓存线的所有标签字段都必须被搜索，并且，随着缓存线的数量增加，搜索所有的标签字段（和/或搜索逻辑的复杂性）所需的时间也增加。

集关联缓存(set associative)缓存结构是直接映射存储器结构和关联存储

器结构的混合。在集关联缓存中，存储器地址的索引部分标识了缓存线的子集。像上面一样，标签字段与每一个缓存线相关联。但是，只有被索引标识的缓存线子集的标签需要被关联搜索。例如，考虑具有 256 个单元的缓存，所述 256 个单元被组织成 64 个子集，每一个子集具有四个缓存线。这样的存储器将具有包含 6 位的索引。

当发生存储器操作时，索引标识 64 个子集其中的一个，并且，和该子集中的四个缓存线相关联的标签字段被搜索，以察看存储器操作数是否在缓存中。集关联缓存结构允许缓存具有很多个缓存线，同时限制了必须搜索的标签字段的数量。此外，存储器操作数无需像在直接映射缓存中那样竞争同一缓存线。

如这里所使用的，术语“关联集”将用来指纯关联缓存的所有缓存线，并且指集关联缓存的一个集。当关联集已满并且新的缓存线必须被储存在该关联集中时，需要一种算法来确定哪一个缓存线可以被替换。在技术上公知几种这样的算法。“随机”算法简单地随机选取缓存线。虽然该实现方案简单，但是随机算法提供了相对较差的结果，因为在为替换所选择的缓存线内容和很快就需要该所选择的内容的可能性之间不存在对应。

一种更好的算法是先进先出（FIFO）算法。这种算法将关联集作为循环队列对待，其中，已经在关联集中最久的缓存线内容被替换。这种算法提供了比随机算法更好的结果，因为该算法观察了缓存错失，以便在为替换所选择的缓存线和将很快需要该缓存线的可能性之间建立对应。

当 CPU 所需的所有存储器内容被加载到缓存中，并且其他的缓存错失不引起所需的存储器内容被替换时，该算法工作良好。但是，该算法未认识到如果缓存线被 CPU 重复地访问，则它不应该被替换。所考虑的唯一因素是存储器内容已经在缓存中的时间长度。该算法比随机算法实施起来略为复杂。通常，使用与关联集相关联的单个计数器来提供指示按顺序哪一个缓存线是下一个要替换的索引，并且每次有缓存错失并从主存储器加载操作数时，递增计数器。

最好的现有技术缓存替换算法之一是最近最少使用（least recently used，LRU）算法。顾名思义，这种算法丢弃最近最少使用的缓存线内容。这种算法往往非常有效，因为该算法既观察缓存命中，也观察缓存错失，以便在为替换所选择的缓存线和很快就将需要该缓存线的可能性之间建立对应。但是，

该算法实施起来相对复杂，因为计数器值通常与每一个缓存线相关联。

为了说明 LRU 算法如何工作，考虑具有 8 个缓存线的完全关联集(full associative set)。3 位的 LRU 计数器值与每一个缓存线相关联，并且每一个计数器值是唯一的，“000”的计数器值代表最近最少使用的缓存线，而“111”的计数器值代表最近使用最多的缓存线。当发生缓存错失时，存储器操作数被加载到具有“000”的计数器值的缓存线中，该缓存线的计数器值被设置为“111”，并且所有其他的计数器值被递减。

当发生缓存命中时，具有大于包含所需的存储器操作数的缓存线的计数器值的计数器值的所有缓存线的计数器值被递减，并且，包含所需操作数的缓存线的计数器值被设置为“111”。很清楚，实施 LRU 算法的逻辑比实施 FIFO 算法所需的逻辑更为复杂。在现有技术上公知有接近 LRU 算法但是实施起来不那么复杂的其它算法。对于 CPU 访问模式，LRU 算法（以及在较轻程度上 FIFO 算法）工作良好，因为由于循环和数据操纵，CPU 往往使用同样的数据和代码数次。

但是，当访问大块的数据（所述数据不能装入缓存）仅一次时，LRU 算法退化。在这种情况下，整个数据缓存将被来自将不再需要的所述大块的数据覆盖。例如，当针对大数组计算聚合函数（aggregate function）时，可能相当经常地发生这种情况。在这种情况下，系统的性能降低。

No. 6,490, 654 号美国专利示出了一种缓存存储器，所述缓存存储器包括多个被关联访问的缓存线，与每一个缓存线相关联的计数单元储存定义替换级别(class)的计数值。当缓存线被访问时，通常给所述计数单元加载计数值，所述计数值指示将很快需要所述缓存线的内容的可能性。

换句话说，给可能很快就需要的数据分配了较高的替换级别，而给不确定的并且较不可能很快就需要的数据分配较低的替换级别。当缓存存储器变满时，替换算法为替换选择具有最低的替换级别的那些缓存线。因此，为替换所选择的缓存线包含缓存中最不确定的、最不可能很快就需要的数据。

No. 6,601, 143 号美国专利示出了一种用于确定有效的缓存线替换算法的自适应缓存管理方法，所述缓存线替换算法用于选择要从缓存中去除(evict)哪些对象（或线）。对象被根据针对每一个对象动态地确定的权重区分优先级。在一个时间间隔期间，观察缓存存储器的命中率，同时将控制参数设置为某个值。

调整所述控制参数，并且在连续的时间间隔期间观察命中率。然后，将控制参数调整一个增量，所述增量具有根据所述命中率是提高了还是降低了所确定的大小和方向。根据所观察的命中率，可以持续地并自动地调整控制参数，并且，所述算法可以包括与其他的对象属性相关联的额外控制参数，以类似的方式来调整所述额外的控制参数。

发明内容

本发明提供了一种数据缓存方法，所述方法既使用 FIFO 缓存替换算法，也使用 LRU 缓存替换算法。在缓存存储器中，缓存线的子集建立 FIFO 队列，并且，所述缓存线的分离子集建立 LRU 队列。此外，为已经被从 FIFO 队列交换输出(swapped-out)的缓存线地址建立辅助 FIFO 队列。

在对于给定的数据请求存在缓存错失的情况下，确定在辅助 FIFO 队列中是否存在命中。如果情况如此，则对应的数据被从主存储器交换输入(swapped-in)并加入 LRU 队列。

根据本发明的优选实施例，FIFO 队列和 LRU 队列都具有预定义的最大大小。例如，FIFO 队列的最大大小在缓存大小的 5％到 25％之间，最好是 10％。LRU 队列的最大大小在缓存大小的 75％到 95％之间，最好是 90％。

根据本发明的又一个优选实施例，需要从缓存存储器交换输出的 FIFO 队列元素的地址被交换输入到辅助 FIFO 队列中。以这种方法可以显著地提高缓存命中率，因为仅被访问一次的元素仅仅通过相对较小的辅助 FIFO 队列就能“就位”(make it)，并且不引起 LRU 队列中经常使用的元素的替换，这对数据处理系统的整体性能有正面影响。

根据本发明的又一个优选实施例，某些数据被分类为 FIFO 类型数据，而其它数据被分类为 LRU 数据。当这种预先分类的数据被交换输入到缓存存储器中时，数据被直接加入其相应的队列，即，被分类为 FIFO 类型的数据被加入 FIFO 队列，被分类为 LRU 类型的数据被加入 LRU 队列。

换句话说，当预先分类的 FIFO 类型数据被交换输出到存储器中且其地址被输入辅助 FIFO 队列，并且稍后它被再次交换输入到缓存存储器中时，它仍被加入 FIFO 队列而非 LRU 队列。

要注意，本发明不限于缓存主存储器的行。另一种应用是缓存数据库系统。在这种情况下，来自基于磁盘的储存装置（类似于主存储器）的同等大

小的数据页面或数据块（类似于缓存线）被缓存在计算机的存储器中，计算机的存储器起到来自磁盘的数据页面的缓存的作用。

附图说明

下面将通过参考附图来更为详细地描述本发明的优选实施例，其中：

图 1 是数据处理系统的框图，

图 2 示出了执行数据缓存的流程图。

具体实施方式

图 1 示出了数据处理系统 100，数据处理系统 100 具有至少一个处理器 102，用于运行应用程序 104。数据处理系统 100 还具有缓存模块 106，用于将处理器 102 耦合到主存储器 108。

缓存模块 106 具有用于储存多个缓存线的缓存存储器 110。储存在缓存存储器 110 中的每一个缓存线均具有相关联的标签："地址"、"FIFO"、"LRU" 以及 "指针"。

标签 "地址" 指示储存在对应的缓存线中的数据的地址或者页号。标签 "FIFO" 指示缓存线是否属于 FIFO 队列。标签 "LRU" 指示对应的缓存线是否属于 LRU 队列。

"指针" 指向相应队列中的下一个元素和前一个元素。如果缓存线属于 FIFO 队列，则 "指针" 具有一个指向 FIFO 队列的下一个缓存线的指针和一个指向 FIFO 队列的前一个缓存线的指针；同样地，如果缓存线属于 "LRU 队列"，"指针" 具有指向 LRU 队列的下一个和前一个元素的两个指针。以这种方法，为 FIFO 和 LRU 队列创建了双向链表。双向链表具有在 LRU 情况下快速地解除链接的优点。

缓存模块 106 还具有辅助存储器 112，用于储存缓存线地址的辅助 FIFO 队列。辅助 FIFO 队列的每一项均以和储存在缓存存储器 110 中的缓存线相同的方式包含地址和/或页号信息。在下面，辅助 FIFO 队列 112 将被称为热 FIFO 队列（hot FIFO queue）。要注意，存储器 112 只储存热 FIFO 队列元素的地址，而不储存缓存线自身，缓存线自身储存在主存储器 108 中。

缓存存储器 110 和辅助存储器 112 可以由同一物理存储器器件实现或由分离的存储器部件实现。例如，辅助存储器 112 可以被实现为专用 FIFO 储存

部件。

缓存模块 106 还具有用于控制数据缓存的逻辑电路 114。逻辑电路 114 可以是硬连接的逻辑，或者，它可以运行用于控制数据缓存的程序，或者，它可以是两者的组合。

在工作中，数据从主存储器 108 交换输入到缓存存储器 110 中。储存在缓存存储器 110 中的缓存线的第一子集建立 FIFO 队列，而储存在缓存存储器 110 中的缓存线的分离第二子集建立 LRU 队列。

在工作中可能发生下列情况。缓存模块 106 从应用程序 104 接收到数据请求 116。作为响应，逻辑电路 114 检查缓存存储器 110 中是否存在缓存命中。如果情况并非如此，则检查缓存存储器 112，即热 FIFO，以查找命中。如果缓存存储器 112 中存在命中，则对应的缓存线被从主存储器 108 交换输入到缓存存储器 110，并被加入 LRU 队列。当缓存线从主存储器 108 交换输入到缓存存储器 110 中时，对应的标签也被储存，即“地址”标签、“LRU”标签以及从被交换输入的缓存线到 LRU 队列的下一个和前一个元素的“指针”。

如果在辅助存储器 112 中未找到被请求的数据地址，则数据请求 116 被转发到主存储器 108，并且数据 118 被从主存储器 108 交换输入。按定义，数据 118 被作为 FIFO 队列的新元素储存在缓存存储器 110 中。如果 FIFO 队列满了，则这要求将 FIFO 队列的的最后一个元素交换输出到主存储器 108 中，并且将其地址输入辅助 FIFO 队列 112。

FIFO 队列的最大大小最好被设置成缓存存储器 110 的储存容量的 5％到 25％之间的水平。

图 2 示出了对应的流程图。在步骤 200，缓存模块接收到数据请求。在步骤 202，确定对于该数据请求是否存在缓存命中。如果如此，则在步骤 204 中，对应的数据被返回到请求者，即请求应用程序。在步骤 206 中，相应队列被更新：如果发生缓存命中的被请求缓存线是 FIFO 队列的元素，则不必做任何事情；如果发生缓存命中的缓存线是 LRU 队列的元素，则在步骤 206 中，根据 LRU 算法对 LRU 队列进行更新。

如果在步骤 202 中确定发生了缓存错失，则控制去往步骤 208。在步骤 208 中，确定被请求的数据地址是否储存在热 FIFO 中（比较图 2 的缓存存储器 112）。如果存在热 FIFO 命中，则在步骤 210 中，被请求的数据被从主存储器交换输入到缓存，并被输入 LRU 队列；接着，在步骤 212 中，数据被返

回到请求者。

如果在步骤 210 中 LRU 队列满了，则 LRU 队列的最后一个元素需要被交换输出到主存储器，以便为新的被请求数据释放空间。

如果在步骤 208 中存在热 FIFO 错失，则控制去往步骤 216，以便从主存储器交换输入被请求数据。在步骤 218 中，确定缓存存储器的 FIFO 队列是否已经达到其最大大小。如果情况并非如此，则在步骤 220 中，在步骤 216 中交换输入的数据被根据 FIFO 算法加入缓存存储器的 FIFO 队列。

如果情况相反，则控制去往步骤 222。在步骤 222，FIFO 队列的第一个元素被交换输出到主存储器中，并且其地址被输入热 FIFO。在热 FIFO 队列是满的的情况下，这可能导致丢弃热 FIFO 的第一个元素。接下来在步骤 224 中，在步骤 216 中交换输入的数据被根据 FIFO 算法加入缓存存储器的 FIFO 队列。

作为图 2 方法的结果，经常使用的数据可能属于 LRU 队列，而较少使用的数据则可能属于 FIFO 队列。结果，总的命中率增加了，这对数据处理系统的性能有正面影响。

附图标记列表

100 数据处理系统

102 处理器

104 应用程序

106 缓存模块

108 主存储器

110 缓存存储器

112 缓存存储器

114 逻辑电路

116 数据请求

118 数据

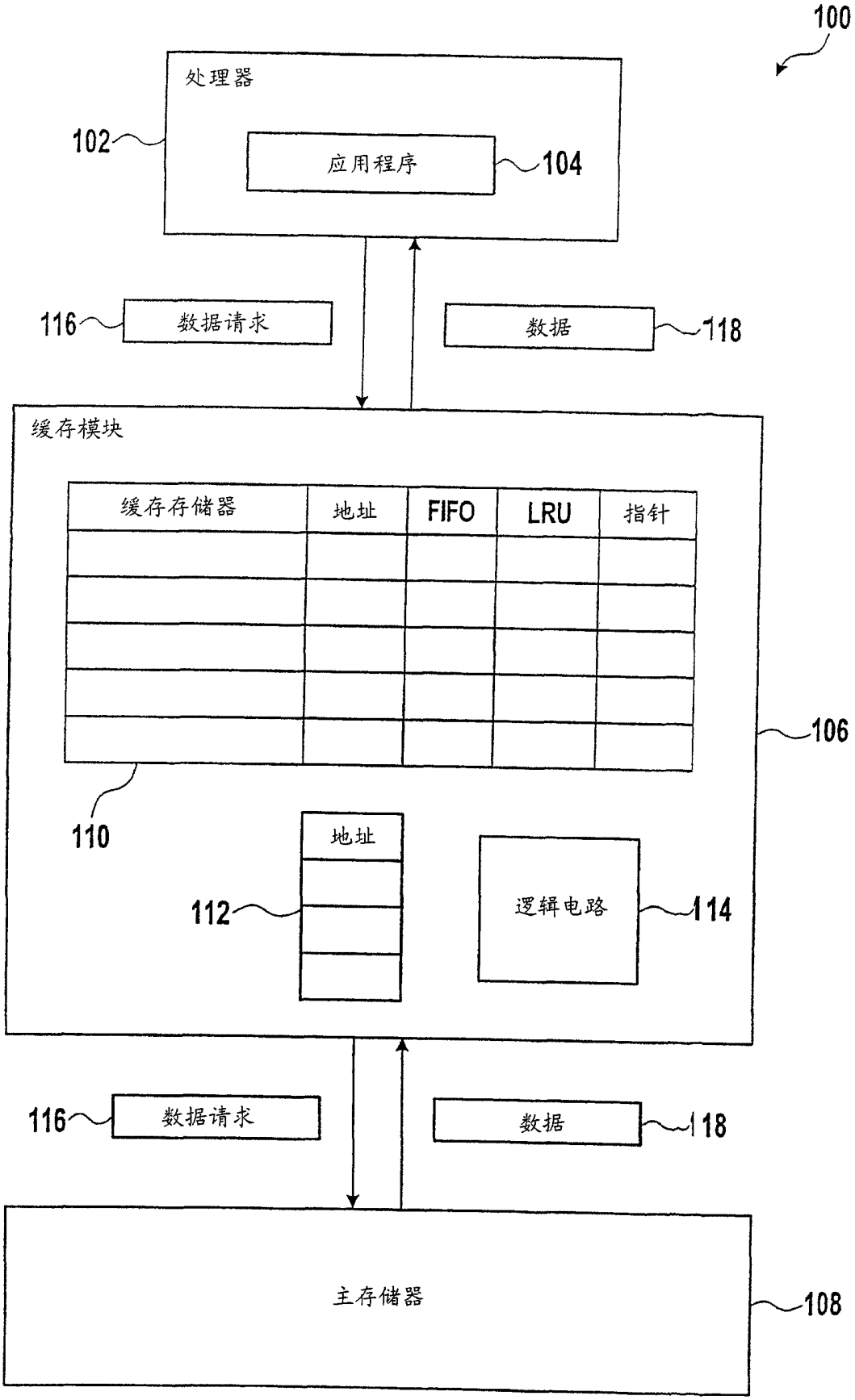

100
处理器
102
应用程序
104
116
数据请求
数据
118
缓存模块
缓存存储器
地址
FIFO
LRU
指针
106
110
地址
112
逻辑电路
114
116
数据请求
数据
18
主存储器
108

图 1

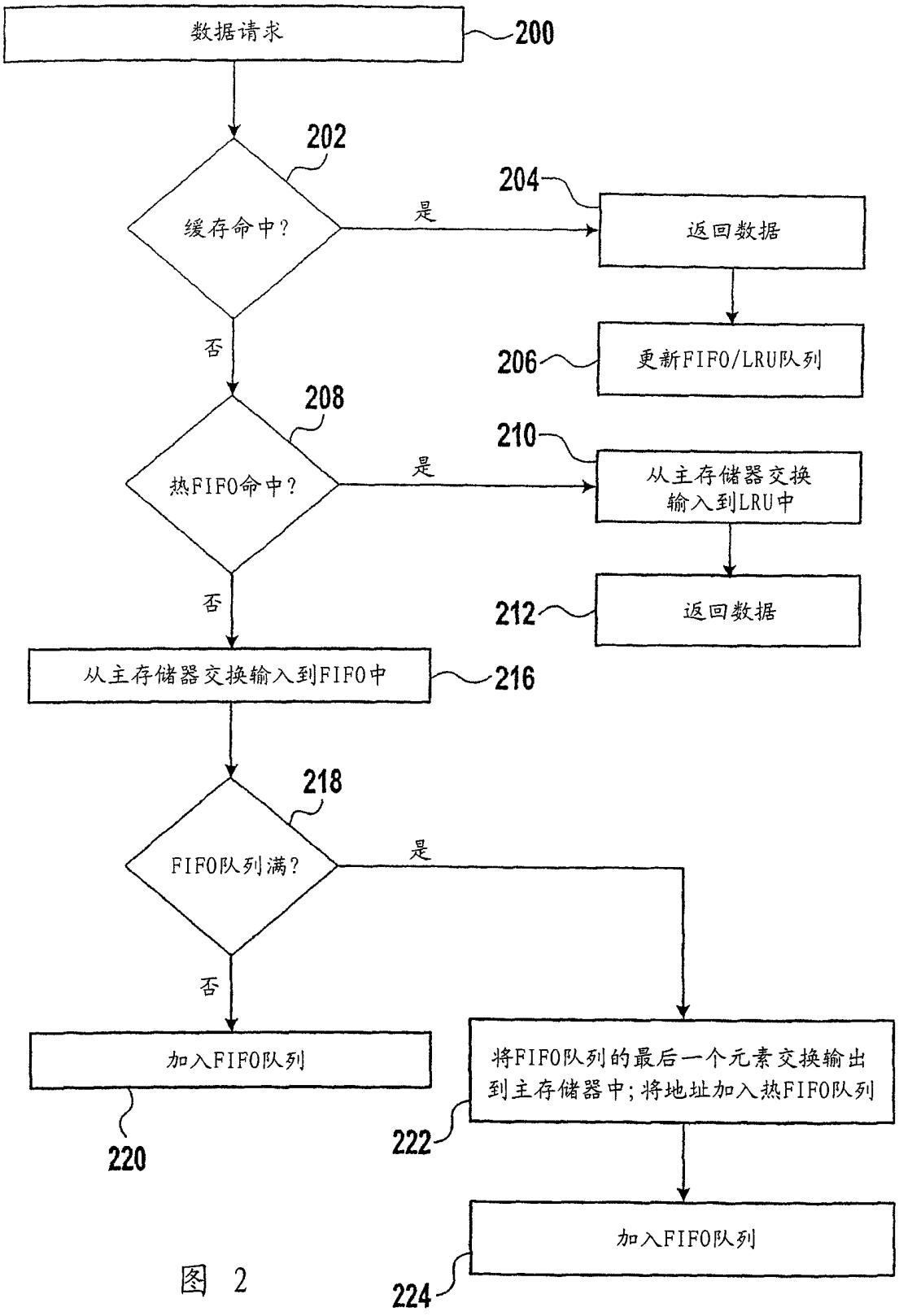

数据请求
200
202
缓存命中?
是
204
返回数据
否
206
更新FIFO/LRU队列
208
热FIFO命中?
是
210
从主存储器交换输入到LRU中
否
212
返回数据
从主存储器交换输入到FIFO中
216
218
FIFO队列满?
是
否
加入FIFO队列
220
将FIFO队列的最后一个元素交换输出到主存储器中;将地址加入热FIFO队列
222
加入FIFO队列
224

图 2