



(12) 发明专利

(10) 授权公告号 CN 101840355 B

(45) 授权公告日 2013. 02. 13

(21) 申请号 201010157518. 8

(51) Int. Cl.

(22) 申请日 2004. 02. 13

G06F 9/48 (2006. 01)

(30) 优先权数据

G06F 9/50 (2006. 01)

60/448, 399 2003. 02. 18 US

(56) 对比文件

60/448, 400 2003. 02. 18 US

CN 1195812 A, 1998. 10. 14, 全文.

60/448, 402 2003. 02. 18 US

WO 0077626 A1, 2000. 12. 21, 全文.

60/474, 513 2003. 05. 29 US

US 6065071 A, 2000. 05. 16, 全文.

10/763, 778 2004. 01. 22 US

EP 0981085 A2, 2000. 02. 23, 全文.

(62) 分案原申请数据

审查员 张涛

200410039767. 1 2004. 02. 13

(73) 专利权人 微软公司

地址 美国华盛顿州雷德蒙德微软道 1 号

(72) 发明人 A · B · 高萨里亚 S · 普罗诺弗斯特

(74) 专利代理机构 上海专利商标事务所有限公司 31100

代理人 陈斌

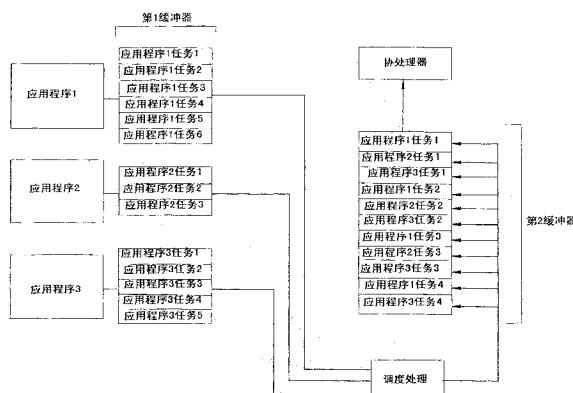
权利要求书 1 页 说明书 33 页 附图 30 页

(54) 发明名称

用于增强协处理器性能的系统和方法

(57) 摘要

用于将协处理器的“资源缺乏”最小化以及用于有效地调度协处理器中的处理的技术，从而获得更高的效率和能力。提供了一种运行列表，允许协处理器从一个任务向下一个转换，而不用等待 CPU 干预。一种称作“表面故障”的方法允许协处理器在一个大的任务开始的地方发生故障，而不是在该任务的中间的某个地方发生故障。可以将 DMA 控制指令即“电子篱笆”、“陷阱”以及“能够 / 不能进行设备环境转换”插入到处理流中，从而使协处理器执行增强协处理器的效率和能力的任务。这些指令还能用于创建高级同步目标程序。最后，描述了一种“触发”技术，可以把用于显示的基本索引从一个位置转换到另一个，由此改变整个显示表面。



1. 一种用于调度任务以用于协处理器中处理的方法,包括:

产生一个运行列表,所述运行列表包括一个由协处理器处理的任务的列表,其中,所述运行列表是由中央处理单元 CPU 产生的;

把所述运行列表传送给一调度程序,其中,所述调度程序准备所述运行列表上的所述任务用于由协处理器处理;以及

由所述协处理器按照所述运行列表所指示的顺序处理所述任务,其中,如果在处理了所述任务中的一个任务时引发一切换事件,则所述协处理器就立即切换到所述运行列表上的下一个任务。

2. 如权利要求 1 所述的方法,其特征在于,所述协处理器是图形处理单元 (GPU)。

3. 如权利要求 1 所述的方法,其特征在于,切换事件包括以下至少之一:完成处理之前提交的任务、在处理一任务的过程中的一个页面故障、在处理一个任务的过程中的一个常规的保护故障、由 CPU 提出的切换到一个新的运行列表的请求。

4. 如权利要求 1 所述的方法,其特征在于,还包括当所述协处理器从所述运行列表中的一个任务切换到所述运行列表中的下一个任务时,向所述 CPU 发出中断信号,其中,当所述 CPU 接收到所述中断信号时,所述 CPU 就为所述协处理器创建一个新的运行列表。

5. 如权利要求 1 所述的方法,其特征在于,还包括由所述调度程序产生第二运行列表,借此,所述调度程序就能够发起要由所述协处理器处理的任务的顺序的改变。

6. 如权利要求 5 所述的方法,其特征在于,还包括禁止第一运行列表中的第一任务出现在所述第二运行列表中。

7. 如权利要求 5 所述的方法,其特征在于,还包括禁止第一运行列表中的第二任务出现在所述第二运行列表中,除非所述第二任务是所述第二运行列表中的第一任务。

8. 如权利要求 1 所述的方法,其特征在于,还包括将关于协处理器从任务向任务切换的历史的信息保存在由所述调度程序可读的一指定的系统存储器位置中。

9. 如权利要求 8 所述的方法,其特征在于,由所述调度程序可读的所述系统存储器位置属于仅可用于单协处理器的历史缓冲器。

10. 如权利要求 8 所述的方法,其特征在于,所述历史缓冲器包括有效存储器以保存存储所述运行列表所需的至少两倍的信息量。

11. 如权利要求 8 所述的方法,其特征在于,还包括指定协处理器写指针,所述协处理器写指针指示所述历史缓冲器中的所述协处理器能够写新的信息的位置。

12. 一种用于调度任务以用于协处理器中处理的计算设备,包括:

产生一个运行列表的装置,所述运行列表包括一个由协处理器处理的任务的列表,其中,所述运行列表是由中央处理单元 CPU 产生的;

把所述运行列表传送给一调度程序的装置,其中,所述调度程序准备所述运行列表上的所述任务用于由协处理器处理;以及

由所述协处理器按照所述运行列表所指示的顺序处理所述任务的装置,其中,如果在处理了所述任务中的一个任务时引发一切换事件,则所述协处理器就立即切换到所述运行列表上的下一个任务。

## 用于增强协处理器性能的系统和方法

[0001] 本申请是申请日为 2004 年 2 月 13 日、申请号为 200410039767.1、发明名称为“用于增强协处理器性能的系统和方法”的中国专利申请的分案申请。

### 技术领域

[0002] 本发明涉及计算机处理器，并且尤其涉及用于调度协处理器的处理的硬件和软件。

### 背景技术

[0003] 现在，许多计算机系统都包括协处理器，例如，图形处理单元 (GPU)。在某些情况下，一个协处理器可以与中央处理单元 (CPU) 一起驻留 (reside) 在系统的主板上，例如微处理器，并且，在其它系统中一个协处理器可以驻留在一个单独的图形卡上。在实现其处理任务的过程中，协处理器经常要访问辅助存储器，例如视频存储器。当前的协处理器经常进行优化来实现三维图形计算，从而支持游戏和计算机辅助设计 (CAD) 之类的应用程序。尽管当运行单一的图形强化应用程序时充分地执行了当前的计算机系统和协处理器，但当运行多个图形强化应用程序时就可能遇到问题。

[0004] 这个问题的一个原因就是典型的协处理器不能有效地调度其工作量 (workload)。典型地，当前的协处理器执行协作的多任务，这是一种多任务，其中，一个当前控制该协处理器的应用程序必须放弃对其它应用程序的控制。如果该应用程序未能放弃控制，则它就可能有效地“挂起”协处理器。尽管在运行单个的用图表表示的增强程序时这并没有什么重大关系，然而当多应用程序都试图使用一个协处理器时，挂起协处理器的问题就会变得更加严重。

[0005] 尽管分配操作间处理的问题已经在 CPU 的设备环境中进行了编址，其中必然需要改进的多操作系统调度，然而，协处理器中的调度尚不能有效地编址。这是因为在现代系统中，协处理器通常被看作是一种资源来使得大量的计算和耗时的操作脱离 CPU，给 CPU 提供更多的处理时间来实现其它功能。这种大量的计算操作往往是图形操作，众所周知，这些图形操作要求具有有效的处理能力。

[0006] 如果为一个协处理器调度任务就引发一个问题，这个问题就是协处理器的“资源缺乏 (starvation)”的可能性。当协处理器不忙时就产生资源缺乏，并且因此计算机系统的处理资源不能被有效利用。为了这个以及其它原因，就需要有用于通过将协处理器资源缺乏最小化来增强协处理器的性能的又允许用于增强其它调度效率的系统和方法。

### 发明内容

[0007] 这个发明提供了多种技术，这些技术能够串联地或单个地使用，用于将协处理器的“资源缺乏”最小化，并且用于有效地调度协处理器中的处理以获得更高的效率和能力。在这方面，中央处理单元 (“CPU”) 提供了一种运行列表，用来允许协处理器在页面故障或者任务完成之类的转换事件发生的时候立即从一个任务向另一个转换，而不用等待 CPU 干预。除了该运行列表之外，一种称作“表面故障”的方法允许协处理器在一个大的任务开始

的地方发生故障,而不是在已经完成有效的处理资源之后在该表面的中间的某个地方发生故障,该大的任务例如再现一个表面。进一步,可以提供 DMA 控制指令即“电子篱笆”、“陷阱”以及“能够 / 不能进行设备环境转换”,该指令可以被插入到处理流中,从而使得协处理器执行增强协处理器的效率和能力的任务。如下面将要更详细地描述的那样,这些指令还能用于创建高级的同步目标程序。最后,描述了一种“触发”技术,其可以把用于显示的基本索引从一个位置转换到另一个位置,由此改变整个显示表面。除了对本发明的这些和其它方面的进一步说明之外,下面还提供了这些技术能够妥善使用的情况。

## 附图说明

- [0008] 本发明或申请文件包含至少一个用色彩来完成的图。本专利或专利申请的公开的副本与彩图一起都将通过贵局按照请求书和所需费用的支付金额来提供。
- [0009] 图 1 是现有技术的调度用于协处理器的处理的方法的总体示意图。
- [0010] 图 2 是按照本发明的协处理器调度改进的示例性示意图。
- [0011] 图 3 是涉及提供了图 2 中总体说明的调度改进的计算部件的更详细的示意图。
- [0012] 图 4(A) 和图 4(B) 是伪码算法,表明了多种非限定性可行的方式来将图 3 的步骤组合为一个函数序列。
- [0013] 图 5 说明了按照本发明的一个调度程序如何能够利用所给的信息来指定用于存储器资源的时间界限,该存储器资源用在直接存储器存取 (DMA) 缓冲器中。
- [0014] 图 6 是说明按照本发明的准备作业线程与辅助存储器管理器之间的动态的算法。
- [0015] 图 7 是按照本发明的页入缓冲器的准备的示例性示意图,该页入缓冲器示出了一个作业线程,该作业线程为该页入缓冲器作准备并且处理 CPU 对页入缓冲器的预处理。
- [0016] 图 8 是表示按照本发明的一个事件链的算法,该事件链可能发生在一个包括对页入缓冲器中的一个电子篱笆的处理的作业线程。
- [0017] 图 9 说明了一种采用内核模式的辅助存储器管理器“VidMm”,能够为协处理器的设备环境提供一个虚拟地址空间,还能管理不同的协处理器的设备环境中的物理存储器,以便它们能够进行合理的存储器共享。
- [0018] 图 10 说明了一种按照本发明的基本调度模型。
- [0019] 图 11 说明了一种按照本发明的改进的调度模型。
- [0020] 图 12(A) 和图 12(B) 都提供了一种能够实现该改进的调度模型的动作序列的示例性表示。
- [0021] 图 13 说明了本发明结合可变长度平面头部表 (flat pate table) 的使用。
- [0022] 图 14 说明了本发明结合多级页表的使用。
- [0023] 图 15 是由该调度程序与一个改进的调度模型一起支持的示例性处理的示意图,改进的调度模型支持表面能级故障。
- [0024] 图 16 是多设备环境的示意图,每一个都有其自己的 DMA 环,结合本发明能够在表面能级故障与本发明一起实现时同时进行处理。
- [0025] 图 17(A)、图 17(B) 和图 17(C) 都提供了一种伪码算法,描述了结合图 16 的部件的本发明的操作,包括多种可以证明有用附加特征。
- [0026] 图 18 是一个按照本发明大体上表示运行列表的使用的框图。

[0027] 图 19 说明了能够为了结合本发明的使用将设备环境转换历史写到由该调度程序可读的特定的系统存储器位置上的硬件的操作。

[0028] 图 20 说明了一种硬件方法,用来通过将有特权的命令直接插入到协处理器设备环境环中来支持有特权的 DMA 通道。

[0029] 图 21 说明了一种方法,用于支持该协处理器中限定的对有特权的 DMA 缓冲器,其中,将该间接命令的一个比特插入到一个环形缓冲器中。

[0030] 图 22 提供了一种查询协处理器关于当前显示表面的方法。

[0031] 图 23 是结合本发明的一个优选的方法,用于当把立即触发与本发明一起使用时查询触发。

[0032] 图 24 是用于同步访问资源来确保当提供两个或两个以上的处理器时都能够使用的示例性技术。

[0033] 图 25 说明了事件历史缓冲器的多种实施例。

[0034] 图 26 说明了支持各协处理器设备环境虚拟地址空间的优选方法,该虚拟地址空间使用一个 PCI 缝隙,该 PCI 缝隙能够重定向到辅助存储器中的任何地方。

## 具体实施方式

[0035] 本发明所实现的多种改进都能够通过图 1 和图 2 的比较从总体上进行说明。图 1 表示一种典型的现有的对协处理器的任务调度的方法。提供了一个缓冲器,该缓冲器能够由不同的应用程序来访问,例如应用程序 1、应用程序 2 以及应用程序 3。该应用程序能够将用于协处理器的任务加载到缓冲器中,并且能够在完成之前提交的任务之后用该协处理器来处理那些任务。如所示,这种方法只能打开一种可能的协处理器的“挂起”。图 1 中,App. 1 是挂起该协处理器。App. 1 要求该协处理器对七个任务起作用,而组合的另外两个应用程序要求仅对三个任务起作用。在类似于多应用程序需要该协处理器的情况下,一种系统就可提供改进的功能性,该系统例如是图 2 所提供的那样。

[0036] 图 2 说明了一种按照本发明的系统和方法,借此,每个应用程序,例如应用程序 1、应用程序 2 以及应用程序 3,都可以保持其自己的缓冲器,就是图 2 的“第一缓冲器”。把这些缓冲器(后面将称作“命令缓冲器”)都提交给一种调度处理,该调度处理能够在把多个任务传送给协处理器的时候作出判定。如图 2 所示,在这种情况下,该调度处理将任务插入到“第二缓冲器”中。为简单起见,图 2 中的“第二缓冲器”已经表示为一个单独的缓冲器。然而,实际上,可能要有多个缓冲器来执行图 2 中的“第二缓冲器”的功能。图 2 中的第二缓冲器已经为了传送给 协处理器而将这些任务进行了划分以便应用程序 1 可以不再挂起协处理器资源。该调度处理允许应用程序 1 在协处理器上完成第一任务,接着是应用程序 2,然后是应用程序 3,并且然后又是应用程序 1,以此类推。

[0037] 尽管图 2 中总体说明的系统和方法的实现比图 2 所示范的更为复杂,但是此处所公开的改进通常都是直接朝着支持如图 2 中所示的基本原理的方向的。现在,接下来的是本发明的实施例的更详细的说明,提供以下术语定义以易于参考:

[0038] 命令缓冲器 - 一种用用户模式驱动器创建的缓冲器。这种缓冲器可以是有规则地可分页的存储器,该存储器置于该再现(rendering)应用程序的设备环境中。

[0039] DMA 缓冲器 - “直接存储器存取”缓冲器。一种用内核模式驱动器创建的缓冲器。

这一缓冲器可以是基于命令缓冲器的内容的。通常，它从一个内核可分页存储器中进行分配并且仅仅对该内核是可见的。在这方面，在协处理器可以从其中读取之前，这些页可以锁定并通过一个缝隙来进行映射。

[0040] 页入缓冲器 - 一种用内核模式驱动器创建的缓冲器。这种缓冲器能够用于页入、收回以及移动一个特定的 DMA 缓冲器所需的存储器资源。页入缓冲器可以构造为在它们的 DMA 缓冲器的对应物 (counterpart) 之前立即运行。

[0041] 环形缓冲器 - 这是一个特定的协处理器设备环境缓冲器。向 DMA 缓冲器的指向可以插在这个缓冲器中。在这方面，一个协处理器能够从这样的环形缓冲器中取出命令来执行。一个环形缓冲器通常包含方向指令，该方向指令指示该协处理器开始从 DMA 缓冲器中读取命令，并且然后一旦该 DMA 缓冲器完成处理就返回到该环形缓冲器。

[0042] 辅助存储器 - 通常是专供协处理器使用并且无需作为物理系统存储器的部分的存储器。例如，它可以是驻留在图形卡上的本地视频存储器。它还可以是其它协处理器可读的存储器，例如通过系统存储器缝隙来映射的存储器。典型地，这种存储器不是集成或 UMA 图形设备。这种存储器不是通过基于缝隙的类似 GART 的页表来进行访问的。

[0043] 系统存储器缝隙 - 这是物理系统存储器的子设备。通过基于缝隙的类似 GART 的页表，它对于协处理器来说是可见的。CPU 能够不依赖于系统存储器缝隙而对物理系统存储器进行访问。当这样的存储器通过缝隙来进行存取时，大体上类似的一些例子为加速图形接口 (“AGP”)、外设部件互连 (“PCI”) 快速 存储器或者标准化存储器结构 (“UMA”) 存储器。

[0044] 在图 3 中可以找到本发明的不同实施例的更详细的视图。图 3 提供了不同的软件和硬件目标程序的示意图，这些目标程序可以组合起来提供图 2 中总体示出的功能。图 3 给出了一系列连续的步骤，这将在后面进行描述。为了清楚地解释和能够实现本发明的目的，顺序地给出了这些步骤，并且不应当把这些步骤当作是给出了用于实践本发明所要求的序列。其次序可以按照本领域所公知或者将来开发实践的需要来改变。下面的讨论将以图 3 中的系统和方法的概述开始，并对图 3 中的一些方面作详细的讨论。

[0045] 图 3 中，步骤 1 表示一个应用程序调用一个应用程序接口 (“API”)。一个应用程序可能是组成用户软件的任意一组文件。典型地，API 不仅是应用程序所使用的一种语言和信息格式从而与操作系统内核进行通信，还涉及这样一种格式，该格式用于与其它控制程序进行通信，这些控制程序例如是数据库管理系统 (DBMS) 或通信协议。结合本发明所使用的一种示例性 API 是由 MICROSOFT ® 开发的直接 3D 运行时间 API。

[0046] 步骤 2 表示一种从 API 向用户模式驱动器的调用。通常，该用户模式驱动器是一个程序例程 (或软件)，它能够将一个软件系统 (常常是操作系统) 链接到一个外部设备子程序，它也可以是软件或者硬件。此处，用户模式驱动器从 API 处接收调用，该 API 可能包含响应于来自步骤 1 的原始调用的 API 参数。步骤 3 表示命令缓冲器中如用户模式驱动器所产生的再现命令的累加。缓冲器是被用作中间储存库的存储器区域。当等着在两个位置之间的传送时，数据可以暂存在一个缓冲器中，这两个位置例如是数据区和用于处理的处理器或者协处理器。如用户模式驱动器所产生的，能够选择特定的命令缓冲器内容从而有利于转换成硬件指定的 DMA 缓冲器，如后面的进一步描述。而且，在规定一个命令缓冲器的过程中，它可能对于省略对存储器资源的直接存储器索引是有用的，该存储器资源例如是“纹理”或“顶点缓冲器”。相反，独立的硬件商 (“IHV”) 可以将命令缓冲器规定为任意地包含

处理,以便一个内核接口可以在都创建了这些存储器资源的同时将存储器索引资料提供给一个命令缓冲器。

[0047] 步骤 4 表示该命令缓冲器的刷新。“刷新”简单地讲是指将所运算的再现命令进行清空。如图所示,为了按照图 3 中所表明的将它们传送给协处理器内核,该再现命令可以发送回 API。刷新可以由任何原因而产生,包括但并不限于因为该 命令缓冲器已满而为了获得再现命令要求更多的空间所以进行刷新,以及该命令缓冲器中有要求立即处理的高优先权再现命令的存在。

[0048] 步骤 5 表示由 API 为该协处理器内核刷新所运算的命令缓冲器。众所周知,一个内核是操作系统的根本部分,是任意地管理存储器、文件以及外设的部分,而且还可以装入应用程序以及对系统资源进行定位。所期待的是,协处理器内核能够是任何类型的内核,包括初级系统内核或者一个分开的协处理器专用内核或者,例如, MICROSOFT ® DirectX Kernel (“DXG”) 之类的特定类型的内核。

[0049] 步骤 6 表示协处理器内核将该命令缓冲器给内核模式驱动器的提交。该协处理器内核能够将命令缓冲器指向内核模式驱动器。除了该内核模式驱动器能够按照内核模式操作之外,如其名称所暗示的,该内核模式驱动器通常可以是如以上参照用户模式驱动器所描述的驱动器。在这方面,内核模式驱动器就能够担负将命令缓冲器转化为 DMA 缓冲器的功能。IHV 可考虑提供该期望装置以便确保适当的有效性并且考虑将命令缓冲器复制到内核模式定位 DMA 缓冲器中。DMA 缓冲器可以是专用硬件,因为它们是最终指定给协处理器的命令的集合,并且因此应当适当地与该协处理器和支持硬件连结。

[0050] 注意,穿过图 3 有一条水平线,将用户模式和内核模式分开了。按照该线条所暗示的,本发明能够在传统的计算机存储器分配地址的线路图上进行操作,这是为了系统的安全性而执行的。另一方面,该内核模式是无特权的并且能被应用程序访问。尽管在理论上分配 DMA 缓冲器的内核模式能够映射到任何存储器空间中,但应当谨记,映射到该应用程序的专用处理空间中可能招致安全性的风险。这是因为一个应用程序专用处理空间中的由线程所涉及的任何虚拟地址的容量都能够修改;换言之,该 DMA 缓冲器的容量应当在确认的时间和硬件处理的时间之间进行修改。

[0051] 如步骤 7 所指出的,该内核模式驱动器还可创建一个存储器资源列表以供 DMA 缓冲器使用。这个可按照命令缓冲器的确定的部分来实现。这个列表可以包含,例如,一个用于列表上的不同存储器的内核编号 (handle) 以及一个缓冲器位置,其中在该位置处访问存储器资源。这个列表还可包括一个用于所列出的存储器资源的预计的设备环境状态。因为它们可能已经由于提交给协处理器的最后一个 DMA 缓冲器的缘故而改变了位置,所以这就允许存储器资源作为该列表的一部分而在 DMA 缓冲器的开始处改编程序,其中这些存储器资源是任何 当前硬件状态 (例如“当前再现目标”、“当前 Z- 缓冲器”等) 的一部分。

[0052] 步骤 8 表示遵循任何存储器资源列表来将一个 DMA 缓冲器发送给一个协处理器内核。接着,该协处理器内核可以将 DMA 缓冲器提交给一个协处理器调度程序,如步骤 9 中所示,并且按照步骤 10 中所示返回到用户模式。

[0053] 通常,一个协处理器调度程序负责用于该协处理器的任务流程的调度 (如不同的 DMA 缓冲器和发送给该协处理器的其它任务中所举出的)。该协处理器调度程序的功能性可能是非常广的,并且这样的说明包含许多潜在的功能,其中该协处理器调度程序可以完成这

些功能。该协处理器调度程序可以称作协处理器调度程序或者简称为调度程序。在不同的实施例中,如图3所示,该调度程序可以完成在把DMA缓冲器提交给协处理器之前的一种或多种功能。步骤11a动态地说明该调度程序的一个功能是提交DMA缓冲器来为处理作准备。

[0054] 步骤11b表示DMA缓冲器的选择,该调度程序确定是添加到准备好的DMA缓冲器的列表上还是接着运行。在这方面,该调度程序能够将该DMA缓冲器传递给一个预备线程。一个预备线程,此处使用该术语,通常提供一种功能,该功能是确保现有适当的存储器资源用于处理该DMA缓冲器。首先,该预备线程可以请求一个辅助存储器管理器过程(未示出)确定一个足够的位置,该位置中页入当前不在辅助存储器中(这是步骤12)的所有需要的存储器目标程序(图形原理中为“表面”)。注意,该术语“辅助存储器”是指为协处理器的使用而分配的存储器;就GPU协处理器而言,辅助存储器往往称作“视频存储器”。

[0055] 可能并不是所有的DMA缓冲器所需的存储器资源都将立刻适于有效的辅助存储器。该辅助存储器管理器可能无法在这点上为了不同的原因而带来辅助存储器中所有的表面。如果这是应当发生的,则可能做出某种进一步的处理来腾出辅助存储器中的更多的空位,或者,可替换地,或者组合腾出更多的空位,该DMA缓冲器可能拆分成多个段。在这种情况下,该预备线程可能使用一个驱动器预定拆分点来拆分该缓冲器并且力图为这个较小的DMA缓冲器所需的存储器资源的子设备进行定位。

[0056] 一旦有效的辅助存储器为DMA缓冲器进行了定位,那么该预备线程就能够请求一个内核模式驱动器,如步骤13所示。这可以是与步骤6、7和8的步骤有关的所提及的内核模式驱动器,或者它可能是一个单独的内核模式驱动器,如本领域技术人员所理解的那样。

[0057] 步骤14说明了该内核模式驱动器能够创建一个用于该DMA缓冲器等待处理的页入缓冲器。该内核模式驱动器可以基于来自该预备线程的处理命令创建这种页入缓冲器。如上所规定的,页入缓冲器是一种为了页入存储器资源的缓冲器。“页入”是指利用映射硬件改变存储器的一块(一页)的物理地址。一般来讲,页入缓冲器是一种包含协处理器指令来将存储器资源移动到它们所分配的位置上的DMA缓冲器。该页入缓冲器用于将DMA缓冲器所需的任何存储器资源带到一个正确的存储器位置,从该位置上,那些资源就能够在需要的时候被协处理器所访问。如果适当地产生一个页入缓冲器,那么就知道了用于一个特定的协处理器任务(即DMA缓冲器)的任何必要的存储器资源的位置。

[0058] 步骤15表示向一个预备线程通知已经产生了一个页入缓冲器。步骤16向该调度程序提示一个信号来表示页入缓冲器准备好了。在这点上,该调度程序可以假设下一个DMA缓冲器已经准备好处理了,或者在将其发送给协处理器处理之前,它可以继续在一个DMA缓冲器上实施进一步的预备操作。例如,因为存储器位置可能由于初始的DMA缓冲器的创建的缘故而改变了,所以在这点上,该调度程序可以再次调用该内核模式驱动器,从而允许它在DMA缓冲器中提取存储器资源的实际位置。最后,该调度程序可以把页入缓冲器和DMA缓冲器两个都提交给协处理器(和任何其它附加硬件)以处理该程序。

[0059] 如上所述的步骤1至16可通过硬件、软件及其组合来实现。在这方面,图4(A)和4(B)一般地以伪码算法的形式说明了图3的这些步骤。图4(A)和4(B)并不是该潜在的伪码算法步骤的穷举列表,这些步骤可以联系本发明得以实现,并且不应当解释为图4(A)和4(B)中的各个步骤都是实现本发明所必要的。相反,图4(A)和4(B)都是出于教导本发明的目的的启示性列表。

[0060] 与图 3 有关的上述讨论是本发明的多种实施例的一个说明。然而,如上所述与本发明的实现有关,已经公开了多种改进。这个说明的其余部分是为了能够做出不同的改进而且克服了在实现本发明的过程中可能出现的困难。

#### [0061] 调度补偿

[0062] 之前所规定的某些或所有的操作(见上述步骤 1-16)可能在 DMA 缓冲器提交给硬件之前发生。然而,这些操作中的一些可能很难执行,直到将 DMA 缓冲器提交给该硬件。例如,存储器资源的位置可能很难确定,直到将 DMA 缓冲器提供给该协处理器的那个时刻。这是因为辅助存储器资源应当随着每个 DMA 缓冲器进行移动,因为它是在该协处理器上运行的。

[0063] 步骤 1-16 所包含的一些操作,如上,可能很耗时,并且由此就不能在中断时间的时候执行,例如是在该调度程序挑选了接下来运行哪个任务之后。类似地,正好因为它们很耗时,所以就有利于该协处理器忙于做其它事情的同时在该中央处理单元(“CPU”)上执行这些操作。这就将协处理器的资源缺乏最小化了。协处理器资源缺乏仅指时间消耗,其中该协处理器不执行处理功能。为了解决这个问题,可能有利于结合该调度程序来利用一种“作业线程”。作业线程可以实现帮助处理一些费时的安装工作的功能。将一个作业线程添加到图 4(B) 的伪码算法中来用于其与本发明的其它处理有关的操作并作为其操作的例子。

[0064] 为补充这种调度补偿,注意,在图 3 的系统中的任何给定的时间内,可能运行 DMA 缓冲器(即当前由协处理器来处理 DMA 缓冲器),正在预备的 DMA 缓冲器以及准备好预备的 DMA 缓冲器的列表。在向该调度程序的提交上,新的 DMA 缓冲器可能插入到准备序列中并且根据其优先权进行适当排序。然而,在向该调度程序的提交上,如果新的 DMA 缓冲器不能抢先选作用于该协处理器的下一个任务的 DMA 缓冲器,本发明的多种实施例就可以增加功能性。其原因是预备一个 DMA 缓冲器在辅助存储器之中和之外可能包括页入存储器资源。因此,为处理所选择的下一个 DMA 缓冲器的抢先可导致改变为辅助存储器管理器的不变状态。如果正在预备的任务抢先了,就可能导致取消改变,该取消改变是由于新选出的 DMA 缓冲器的预备而得到的辅助存储器管理器的不变状态。通过在 DMA 缓冲器任务上的操作而在半路上对辅助存储器取消改变不可能是微不足道的,并且可能会导致更频繁的协处理器资源缺乏。

#### [0065] 拆分 DMA 缓冲器

[0066] 当通过 API 把一个命令缓冲器提交给协处理器内核时,然后,该内核模式驱动器可能随着一个特定的 DMA 缓冲器和存储器资源列表的产生而发生改变,该存储器资源列表用于运行那个 DMA 缓冲器。尽管特定的 DMA 缓冲器格式可以由那些 IHV 来指定,但软件提供商可能发现它们自身所具有的指定用于该内核模式驱动器的资源列表的格式的任务。

[0067] 该存储器资源列表能够提供有关不同的存储器资源的时间界限信息,这些存储器资源供 DMA 缓冲器使用。反过来,该调度程序能够使用这些存储器资源列表,从而在协处理器上运行之前,并且如果有必要拆分 DMA 缓冲器的话,例如当 DMA 缓冲器一次使用太多资源的时候,DMA 缓冲器就可以页入任何所需的存储器资源。

[0068] 如果 DMA 缓冲器由一个调度程序来拆分,那么通过在存储器资源列表上提供时间界限信息,内核模式驱动器就可能对此有利。这可以通过允许该驱动器在 DMA 缓冲器中规定一个“偏移”来实现。存储器资源正在通过插入一个存储器资源标识符来进行编程的时候,可以设置一个偏移,该标识符规定在该偏移上的存储器资源的使用。因为存储器资源在 DMA 缓冲器中可能出现不止一次,所以相同的存储器资源可能在存储器资源列表中多次出

现。对 DMA 缓冲器中的存储器资源的每个索引都将添加一个对该资源列表的项。

[0069] 实质上,那个编号 / 偏移列表可能不足以提供有关存储器资源的调度程序有效信息,其中该存储器资源需要拆分 DMA 缓冲器。为了精确地知道在 DMA 缓冲器中需要一个特定的存储器资源,该调度程序还可要求有关一个存储器资源被另一个资源代替时的信息。例如,一个第一纹理可能被一个第二纹理所代替,其中,该第一纹理,纹理 A,包括在第一纹理阶段中的 DMA 缓冲器的起始处,而该第二纹理,纹理 B,在中间,并且然后返回到该 DMA 缓冲器的末尾处的纹理 A。该调度程序可使用这种附加信息来将 DMA 缓冲器拆分成块,这些块将使用较少的存储器资源。然而,在上述概况中,纹理 B 还可能在第一纹理阶段已经进行了编程,在这种情况下就可能像纹理 A 一样同时被使用,并且不应该拆分成单独的 DMA 缓冲器子设备。

[0070] 为了获得该“更好的晶粒”暂时信息,其中该暂时信息是以上述复杂的方式拆分 DMA 缓冲器所需的,一个调度程序可以利用有关全部 DMA 缓冲器的存储器资源的使用的信  
息。在一个实施例中,当内核模式驱动器为存储器资源列表中的每一项都提供了一个资源标识符时,就能够将其实现。资源标识符简单地讲就是一个表示特定的存储器资源怎样就是将要使用的整数值。例如,值 0 可说明一个存储器正用作一个提交目标,而值 1 就说明一个资源正用作 Z- 缓冲器。具有这些信息,该调度程序就能够确定纹理 B 是否代替了纹理 A(例如,两个纹理是否具有相同的资源标识符)或者是纹理 B 是否替换了纹理 A(例如,A 和 B 具有不同的资源标识符)。用于该资源标识符及其意义的实际值可由 IHV 来指定,或者置于软件结构中。对于确保用作资源标识符的值是基于 0 的,它可能是有用的,并且为该驱动器规定最大的资源标识符值,它将在驱动器初始化时使用。

[0071] 图 5 说明了一个调度程序如何使用所提供的信息来指定一个时间界限,该时间界限是用于正用在 DMA 缓冲器中的存储器资源的。更值得注意的是,通常,DMA 缓冲器应当以当前存储器资源(即,那些之前 DMA 缓冲器的结尾处的当前的资源)“安装”或者初始化过程为开始。这是因为存储器资源可能已经由于执行一个之前的 DMA 缓冲器的缘故而移动了,并且这就可能需要改编程序。存储器资源可能需要改编程序,直到该 DMA 缓冲器调度用于处理的时刻。

[0072] 该存储器列表,如图 5 所示,可包含任何数量的字段。下表提供了一个有用字段的非穷举列表:

[0073]

编号	存储器资源的编号
资源 ID	资源指示器任意指定如何使用资源
偏移	DMA 缓冲器中的偏移,其中存储器资源可以进行编程。该调度程序可以要求该驱动器运行该 DMA 缓冲器直到那个点,如果它需要的话,从而由于存储器的约束而拆分该缓冲器。因此,这个偏移能够为 DMA 缓冲器提供一个有效的拆分点。
段提示 (HINT)	指定一个段,该驱动器可能用于特定的应用程序来提供最佳的性能。这会由于该位置而替换当前驱动器的优先权。
库提示	指定所提示的段中的一个库,其中该内核模式驱动器能够页入一个分配地址。这会由于该位置而替换当前驱动器的优先权。
段 ID	指定一个持有存储器资源的段的段标识符。这会在页入的过程中填满。
物理地址	指定一个段中的存储器资源的物理地址。这会在页入的过程中填满。

[0074] 页入

[0075] 通常,DMA 缓冲器所访问的存储器资源都在 DMA 缓冲器提供给协处理器执行之前就带进了存储器之中。将所访问的存储器资源带进存储器中称为页入资源。页入可包括一个如上所述的预备作业线程与一个内核模式驱动器之类的驱动器之间的交互。参见图 6,对于伪码算法,说明了该预备作业线程与辅助存储器管理器之间的动态关系。

[0076] 典型地,该页入步骤发生在已经选择用于处理的 DMA 缓冲器并且已经产生了用于

特定的 DMA 缓冲器的资源列表的时候。进行页入从而确定如何将存储器资源提供给辅助存储器并且确定在辅助存储器的什么位置放置它们。

[0077] 该页入过程可能有一个辅助存储器管理器来处理。该辅助存储器管理器可使用一个提示 (hint)，该提示由该内核模式驱动器在特殊分配的创建上任意地提供。创建该提示来为存储器资源在存储器中找到适当的位置。

[0078] 已有多种有关页入存储器资源的问题。可能没有足够的辅助存储器来有效地获取所有的资源，在这种情况下，普遍都会收回存储器中的某些资源。即使在收回辅助存储器中的其它目标程序之后，也可能存在用于 DMA 缓冲器的存储器不够的情况。在那种情况下，该 DMA 缓冲器就会被拆分成多个较小的块，这些块都要求较少的存储器资源。

[0079] 在页入过程中，辅助存储器管理器会创建一个命令列表，这些命令可以用于将该存储器资源放在适当的位置上。那个命令列表会，例如，从以下的操作中创建：

[0080] 1) 收回：将一个特定的存储器资源移动到当前段的外面以及为了给另一个资源腾出空间而移动到系统存储器中；

[0081] 2) 页入：将一个特定的存储器资源从系统存储器带进辅助存储器中的合适的位置上；

[0082] 3) 重新定位：将一个特定的存储器资源从一个辅助存储器的位置上移动到另一个位置。

[0083] 可允许该辅助存储器管理器使用这些操作中的任何操作，从而解决存储器的放置问题。这种非穷举的命令列表可以由该辅助存储器管理器在一个页入操作的过程中产生并且之后可以由该调度程序使用来产生一个页入缓冲器。为了重新定位、收回或页入或者否则以任何方式移动或改变的任何存储器资源，该辅助存储器管理器会在命令列表中产生一个项。在这方面，本发明的多种的实施例在命令列表中提供了以下字段：

[0084]

编号	存储器资源的编号，用来进行重新定位。
字段 ID	段标识符，用于把该段标识到当前所在的存储器资源中
物理地址	该存储器资源的当前段中的当前的物理地址
新字段 ID	用于该段的段标识符，标识该资源可以移动到的位置。

[0085] [0085]

新物理地址	一个新的段中的新的物理地址，表示该资源可以移动到的位置。
-------	------------------------------

[0086] 页入缓冲器的产生

[0087] 使用如上所述的命令列表，调度程序就可以产生一个页入缓冲器来执行这些命令。如图 7 中所示实现了用于与本发明有关的页入缓冲器的多种实施例。

[0088] 如图 7 中所示，一些命令可能需要在运行之前进行预处理，而其它命令不需要预处理就可以处理。预处理可以以任何数量的方式来实现，这些方式中包括作业线程方式。注意，可能在预处理命令的过程中必须等待，直到已经处理了一部分页入缓冲器。在图 7 所示的模型中，一个作业线程为该页入缓冲器作准备并且为了该页入缓冲器运行 CPU 预处理。当页入缓冲器中的一个操作之前需要 CPU 预处理的时候，该作业线程将协处理器中的该页入缓冲器上的操作分块。然后，它就在再次启动该页入缓冲器之前提出一个 CPU 请求从而完成该操作。

[0089] 这样，对于命令列表中的每条命令，以下的动作都是适合的：

[0090] 在产生页入缓冲器的时候进行预处理；

[0091] 在页入缓冲器的同步的点上进行 CPU 处理；

[0092] 运行“Blit”命令来移动存储器资源；

[0093] 一旦完成页入缓冲器就进行后处理 CPU 的工作。

[0094] 参见以上可能的动作的列表，页入缓冲器本身可能包含命令，这些命令将要求协处理器在 CPU 执行某些工作的同时是阻塞的。这样一种产生中断并拖延了协处理器的命令在此称作“电子篱笆”。一个页入缓冲器中的任一命令都可以加在电子篱笆的前头或者后头。因为中断是不可取的，所以通过将后操作电子篱笆合计到一个缓冲器的结尾，就能够减少 CPU 可能中断协处理器的次数。在后操作电子篱笆（或“后电子篱笆”）需要在该缓冲器的结尾之前的情况将由该调度程序来检测，并且将该命令的前操作电子篱笆（或“前电子篱笆”）进行合并，其中该命令应当要求该后电子篱笆是已经执行了的。

[0095] 注意，为了保持辅助存储器之间的相关性，可能在该页入缓冲器的处理过程中不允许外部中断。这样，如果在一个页入缓冲器完全实现之前就终止了一个量程，那么该页入缓冲器就可以允许保持在该协处理器的控制中，直到完成。

[0096] 参见图 8，为伪码算法表示了一连串的事件，这些事件可能发生在包括页入缓冲器中的电子篱笆处理的作业线程中。与图 8 有关，以下表格提供了广义的命令列表和可能产生的任何终止电子篱笆，其中这些命令在命令列表中发生。下表仅用作是帮助性的例子，并不意味着是作为可能的命令或者动作的类型的穷举列表，其中这些动作可能是关于那些命令而产生的。

[0097]

从辅助存储器向另一个辅助存储器位置移动	预处理： 无。 在该页入缓冲器中： 如果用硬件来进行该转移 该驱动器就能把一个 blit 添加到该页入缓冲器中。 如果用软件来进行该转移 刷新当前页入缓冲器。一旦刷新了，就在 CPU 上继续进行转移。 在该页入缓冲器的终止电子篱笆中： 无。
从辅助存储器向缝隙移动	预处理： 加入该处理，拥有要移动的辅助存储器资源； 主存储器探测和锁定该系统存储器缓冲器，并且为锁存的页获得一个 MDL； 如果主存储器探测和锁定失败， 处理软件中的该 blit； 从该处理中脱离出来； 如果已经分配的缝隙地址当前并不忙并且在当前命令之前该命令列表中没有命令，变换那个缝隙的范围。 用所产生的 MDL 对该缝隙进行编程。 注意该缝隙是编程了的。 在该页入缓冲器中： 如果该缝隙没有在该预处理阶段进行编程。 刷新当前页入缓冲器。 刷新之后，将 MDL 编程为一个缝隙。

[0098] [0098]

从缝隙中移动	隙。继续处理该页入缓冲器。 如果用硬件来进行该转移 该驱动器将添加一 blit 到该页入缓冲器中。 如果用软件来进行该转移 刷新当前页入缓冲器。 刷新之后，利用 CPU 转移该存储器。继续处理该页入缓冲器。 在该页入缓冲器的终止电子篱笆中： 无。
从缝隙中移动	预处理： 加入该处理，拥有要移动的辅助存储器资源； 主存储器探测和锁定该系统存储器缓冲器，并且为锁存的页获得一个 MDL； 如果主存储器探测和锁定失败， 处理软件中的该 blit； 从该处理中脱离出来； 如果已经分配的缝隙地址当前并不忙并且在当前命令之前该命令列表中没有命令，变换那个缝隙的范围。 用所产生的 MDL 对该缝隙进行编程。 注意该缝隙是编程了的。 在该页入缓冲器中： 如果该缝隙没有在该预处理阶段进行编程。 刷新当前页入缓冲器。 刷新之后，将 MDL 编程为一个缝隙。继续处理该页入缓冲器。 如果用硬件来进行该转移 该驱动器将添加一 blit 到该页入缓冲器中。 如果用软件来进行该转移 刷新当前页入缓冲器。 刷新之后，利用 CPU 转移该存储器。

[0099] [0099]

从缝隙中收回	器。继续处理该页入缓冲器。 在该页入缓冲器的终止电子篱笆中： 如果该缝隙的范围已经不能由该缓冲器中的另一个操作收回。 不映射到该缝隙的范围； 从拥有该表面的处理上附加； 主存储器未锁存该系统存储器缓冲器； 从该处理器中脱离。
从缝隙中收回	象从视频到缝隙进行移动一样的处理。只是在该页入缓冲器的终止电子篱笆上，没有映射到该缝隙范围。
从缝隙中收回	预处理： 如果该缝隙范围不忙， 不映射到该缝隙的范围； 加到拥有该表面的处理上； 主存储器未锁存该系统存储器缓冲器； 从该处理器中脱离。 在该页入缓冲器中： 无。 在该页入缓冲器的终止电子篱笆中： 如果该缝隙范围已经由任何之前的操作映射了。 不映射到该缝隙的范围； 加到拥有该表面的处理上； 主存储器未锁存该系统存储器缓冲器； 从该处理器中脱离。

[0100] 注意，此处表示出的调度模式可能要求有效数量的非平凡 CPU 处理，从而保证协处理器是繁忙的。依照如今现有的协处理器硬件的性能，迫使进行，至少部分地，这个工作。进一步，图形硬件可设计成具有更强大的存储器虚拟化和协处理器调度。在这方面，本发明已经得到了许多改进并且这些改进还将连同本发明一起公开。对于每个硬件的性能，我们阐明了改进的动机以及对上述调度模型的影响。某些改进是给予特定的实现方法来示出的。注意，尽管不是所有的这些方法都必然支持任何将来的模型，但是此处以一种方式描述

了多种改进,这种方式是如果实行该特定方法并且当实行该特定方法的时候向实现方法提供一种适于改进的基本原理。

[0101] 可中断的硬件

[0102] 为了增加协处理器调度的可靠性,协处理器可能支持更好的间隔尺寸上的中断,该间隔尺寸比整个 DMA 缓冲器的间隔尺寸要好。例如,协处理器和支持硬件可以支持三角处理中的中断,而不是仅仅先于或者滞后于处理三角。

[0103] 在这样的可中断硬件的多种实施例中,可以提供一种优选的设计方法,用于通过对于辅助存储器的协处理器设备环境的自动保存和恢复大概地完成协处理器的虚拟化。每个协处理器设备环境都可能具有,示例的方式而非限定,一个专用地址空间、一个专用环形缓冲器以及一个存储器的专用片断,其中在专用环形缓冲器中累加 DMA 缓冲器,而当协处理器的设备环境没有运行时,在存储器的专用片断中保存该硬件的状态。为了以这种设置来支持设备环境转换,通过一个所映射的存储器寄存器,调度程序可能把一个所保存的设备环境的辅助存储器中的物理地址提供给协处理器。然后,该协处理器载入那个协处理器设备环境、验证所有的存储器资源都是有效的,并且然后运行该 DMA 缓冲器,该 DMA 缓冲器已经在环形缓冲器中进行了累加,按照它们所遇到的资源的需要发生故障。

[0104] 与上述有关,它还能够使得内核模式驱动器查询没有运行的协处理器设备环境的状态。这通过利用“运行列表”事件跟踪文件(如下所述)来检查所保存的设备环境而得以实现,或者由任何查询装置都能够实现。在这方面,该驱动器能够确定有用的信息,例如(1)协处理器离开一个特定的设备环境(例如清空、新运行列表、页入失败)进行最近转换的原因;(2)硬件使用的存储器资源的列表(如果支持表面能级故障);(3)发生故障的地址(如果支持页面能级故障);以及(4)协处理器时钟周期数,其中该时钟周期是特定的设备环境正在运行的周期。

[0105] 还有,该内核模式驱动器能够将新的 DMA 缓冲器插到设备环境的一个当前并未运行的环中。在所保存的设备环境中,它还能够修改该环的位置、该页表 或者那个设备环境中存储的任何物理存储器资料。可能要求这种修改,例如,随后的存储器中的那些资源的移动。

[0106] 通用协处理器设备环境虚拟地址空间

[0107] 上述基本调度模型的某些复杂性是由于协处理器设备环境可能共享公共的协处理器地址空间的原因。虚拟化这个地址空间可以供给一种磨光器(sleeker)系统。在虚拟化该地址空间的过程中,辅助存储器管理器可以来回移动存储器,甚至从整个辅助存储器中收回资源。这就意味着用于资源的实际的协处理器可见的地址可以在其时间界限内改变。这样,以用户模式创建的命令缓冲器就不能直接索引对其地址的分配,因为那些地址可能是未知的,直到该命令缓冲器调度用来运行。

[0108] 例如,上述基本调度模型的以下要素能够通过通用协处理器设备环境地址空间的用户来消除:

[0109] 1) 通过用实际的存储器位置代替处理来修补命令缓冲器

[0110] 2) 确认用于存储器访问的命令缓冲器

[0111] 3) 以内核模式构造存储器资源列表

[0112] 4) 创建单独的命令和 DMA 缓冲器

[0113] 5) 将用于所中断的 DMA 缓冲器的资源带回预先中断位置

[0114] 在提供通用协处理器设备环境虚拟地址空间的过程中,特定的协处理器设备环境中的地址分配可以获得在那个设备环境的地址空间中的它们自己的唯一地址。该地址在该分配的持续期限里将不进行修改。这样,命令缓冲器就会直接访问那些地址,而不要求修补。也不需要确认命令缓冲器以及将命令缓冲器复制到 DMA 缓冲器中。因为 DMA 缓冲器中的存储器资料可能在协处理器的虚拟地址空间中,并且那个地址空间实际上是专用于任何协处理器设备环境的,所以就不需要确认存储器资料的有效性,并且这样就不需要隐藏 DMA 缓冲器中的命令缓冲器的确认的内容,其中这些内容是应用程序所不可见的。没有被一个分配或一个收回分配占用的地址空间(编号或实际地址)可能被该硬件重定向到一个虚页或导致访问失败。因为设备环境没有访问不打算访问的存储器,所以这就会确保内核模式存储器的安全性。

[0115] 通用协处理器设备环境虚拟地址空间的一些优点如下:每个分配都会在分配时获得一个协处理器可见的地址(或编号)。那里没有命令缓冲器;DMA 缓冲器对于用户模式驱动器是直接可见的并且可以由用户模式驱动器来填充。DMA 缓冲器可以直接指向它使用的地址分配的地址(或编号)。用于页入的资源列表可以由用户模式驱动器来创建。

[0116] 已经回忆了按照图 3 所阐明的本发明的多种实施例的模型以及相应的说明。这个模型还能利用可中断硬件和 / 或通用协处理器设备环境虚拟地址空间来进行进一步的改进。在这方面,除了通过本发明的附加的进步来进一步改进之外,下面的部分描述了类似于图 3 中的那些部分的原理。

#### [0117] 表面分配地址和重新分配地址

[0118] 在该改进的模型中,辅助存储器管理器,例如采用内核模式的视频存储器管理器“VidMm”,能够提供一种用于协处理器设备环境的虚拟地址空间,并且能够管理该多种协处理器设备环境中的物理存储器,以便它们能够获得其合理的存储器共享。图 9 中描绘了有关该基本模型的分配方案的这种改进的多种实施例。图 9 利用术语说明了本发明的一个实施例,其中该术语是本领域技术人员所熟悉的,因为它是相应于本领域所公认的概念的。例如“VidMm”是一个视频存储器管理器,而“Thunk interface”是一个形实转换程序接口。然而,注意,尽管术语是用于更清楚地解释本发明的,但不应当将其看作是限定本发明的意图的表示。这样,“VidMm”就可能是用于任何辅助存储器的存储器管理器,而“Thunkinterface”就可能是任何适当的接口,等等。

[0119] 与图 9 有关,该改进模型允许 DMA 缓冲器直接映射到应用程序的一个地址空间中,这就使得它们任意地直接由用户模式驱动器来访问。利用需要访问的(所以不要求修补)每个存储器资源的不变的虚拟地址或编号,该用户模式驱动器将再现原语直接分组给 DMA 缓冲器。该用户模式驱动器还创建了一个该 DMA 缓冲器正在使用的存储器资源列表,以便在调度 DMA 缓冲器之前,该辅助存储器管理器能够将它们带进辅助存储器中。如果恶意的应用程序修改了该资源列表,正确的资源组就不能适当地页入了。注意,这并非必然要破坏该存储器保护模型,因为不是索引有效存储器的地址空间的范围会要求对虚拟存储页进行索引,或者导致硬件失败并且阻塞该特定协处理器设备环境的运行。在两者之一的情况下,破坏资源列表并不会导致协处理器设备资源能够去访问另一个设备环境的存储器。

[0120] 该改进模型中,用户模式驱动器把 DMA 缓冲器提交给了内核模式驱动器,该内核模式驱动器又把该 DMA 缓冲器提交给了调度程序。要求该存储器管理器页入资源列表中的资源之后,该调度程序就把 DMA 缓冲器照原来的样子发送给该硬件。

[0121] 在改进的模型中进行调度

[0122] 在改进的模型中调度是十分类似于在基本模型中调度的。还有一种作业线程在把 DMA 缓冲器提交给协处理器之前就为该 DMA 缓冲器做好了准备。然而，能够由该作业线程在该改进模型中实现的工作仅限于页入操作。

[0123] 参见图 10 和图 11，给出了以基本模型调度和以改进模式调度的实施例。为了更清楚，该改进模型具有两个调度选项。当调度而非命令失败时，就能够执行准备阶段。然而，当该改进模型使用命令失败时，就不必要有准备阶段。

[0124] 另外，图 12(A)、12(B) 提供了一种能够实现该改进的调度模型的伪代码的流程图。

[0125] 在改进模型中页入

[0126] 在改进模型中页入是不同于在基本模型中页入的。在改进模型中，已经知道了页入的一个分配的地址，并且该存储器管理器只是需要使其有效。为了使资源列表中的一个分配地址有效，该存储器管理器需要找出物理辅助存储器的范围，该范围是空闲的，并且要求该驱动器将页表或编号映射到那个范围。如果必要的话，可以要求物理存储器的范围是连续的一组页。

[0127] 如果没有足够的物理视频存储器使该分配地址有效，那么辅助存储器管理器，在此称作 VidMm，就为收回而标记某些当前有效的分配地址。当收回一个分配地址时，其内容传送给了系统存储器（假设它还没在系统存储器中），并且然后就使其虚拟地址或编号无效。

[0128] 虚拟地址空间

[0129] 本领域公知的或者将来要开发的用于提供虚拟地址空间的技术可以结合本发明一起使用。为了示例可以使用地址空间的方式，此处给出了两个使用公共虚拟地址空间技术的例子。应当理解，已经有多种用来为协处理器创建虚拟地址空间的方式，并且本领域技术人员能够从此处给出的例子中推断出来。在这方面，此处描述了利用一个可变长度平面页表和一个多级页表的虚拟地址空间。

[0130] 可变长度平面页表。图 13 中示出了结合可变长度平面页表的本发明的使用。在该方法中，通过使用平面页表，将协处理器的地址空间进行了虚拟化。该虚拟地址空间可划分为预定的存储器总数的页，例如 4KB。对于虚拟地址空间中的每一页，页表都提供了其中包含标识符，例如 64- 位的项，用于规定相关物理存储器的物理地址和位置（例如，加速图形接口 (AGP)、外设部件互连 (PCI) 或者视频）。在一个实施例中，为了允许该协处理器页表对系统存储器的页进行索引，协处理器支持的页不是任意的而是必须是 4KB。还有在这个实施例中，该协处理器页表必须能够从相同的地址空间中对本地视频存储器和系统存储器进行寻址。该协处理器可能要求属于单个表面的所有页都映射到一个单一类型的存储器。例如，该协处理器可能要求属于一个特定再现目标的所有页都映射到本地视频存储器中。然而，将表面映射到多种物理存储器类型 (AGP、本地视频等) 的页表项都可以共存于该页表中。

[0131] 对于 PCI 和 AGP 适配器，各页表项的示例性实施例可能包含 32 位，允许全部 4GB 的物理地址空间对于协处理器都是可见的。对于利用快速 PCI 类型的适配器的实施例，该协处理器可以支持一个 64- 位的寻址周期。每个页表项都可包含 40 位或以上位从而对存储器的每千兆字节的存储器进行寻址。实现 64 位系统的实施例使用主板上的 40 位以上的物理地址线，如果相应的视频适配器不能对整个地址空间寻址，则该实施例可能遭受性能损失。因此，建议支持完全的 64 位。

[0132] 该平面页表方法类似于虚拟化装置，该虚拟化装置是当前现有的 INTER ® 8086 (x86) 家庭用 CPU，除了没有页目录之外，只有一个庞大的页表。

[0133] 与有效分配地址不相关的虚拟地址可以重定向到一个虚页，从而防止恶意的 DMA 缓冲器强制该协处理器访问不应该访问的存储器。该硬件可以执行各页表项中的有效位，该有效位规定了该项是否有效。

[0134] 当相关的协处理器设备环境当前没有在该协处理器上运行时，该页表是可重定位的。当该设备环境没有运行时，VidMm 可能将该页表收回给系统存储器。当该设备环境再次准备运行时，就可以将该页表带回给视频存储器，但它是位于一个可能不同的位置上。该驱动器也许可以更新所保存的协处理器设备环境中的页表的位置。

[0135] 在这个实施例中，所有的存储器访问都可通过协处理器虚拟地址而发生。然而，这并不意味着本发明要求这样的访问。某些元件可能是以其它方式来访问的。而且如果以其他方式访问甚至可以提供增强的功能性。可以不考虑该虚拟地址方案的一些数据项是：

[0136] 1) 该页表本身可以通过一个物理地址来索引。

[0137] 2) 该阴极射线管 (CRT) 可以编程为用于相邻存储器范围的物理地址。

[0138] 3) 虚拟打印引擎 (VPE) 能够直接执行 DMA 到一个物理地址。

[0139] 4) 重复占位可能直接从一个物理地址中进行读取。

[0140] 5) 该协处理器设备环境可能通过物理地址来进行索引。

[0141] 6) 该初始环形缓冲器可能通过物理地址来进行索引。

[0142] 注意，在设备环境转换的过程中，协处理器能够通过还原的设备环境重新转换正在使用的虚拟地址。可以确信，存储器资源是置于适当的位置上的，而不是允许该协处理器做出可能是错误的假设，该错误的假设为那些地址在该设备环境转换之前就对相同的物理页进行索引。还要注意，结合本发明的多种实施例，将有利于允许单一页表中有多个项或者有利于通过多个页表去访问相同的物理页。

[0143] 在多个实施例中，协处理器可以实现一个界限寄存器，该界限寄存器给出了页表的当前大小。经过页表末端的任何存储器索引都会被协处理器看作是无效访问并且都是这样处理的。该页表可以用 2 的幂次来扩展并且在一个实施例中支持至少 2GB 的地址空间（页表中的 2MB）。

[0144] 如果与协处理器设备环境相关的虚拟地址空间变成片断，一个 API，例如 MICROSOFT ® Direct3D 运行时间，能够执行无用单元的收集从而减少地址空间和相应页表的大小。将进行高虚拟地址处的分配删除并且重新分配到较低的地址。

[0145] 结合本发明利用可变长度平面页表来实现虚拟地址空间的优缺点对于本领域技术人员来说是显而易见的。概括为，利用该平面页表的一个优点是只有一级向物理存储器的间接寻址。另一个优点是，页入能够用一组不相邻的页来解决。然而，也有缺点。例如，当协处理器运行时，整个页表通常都需要位于存储器中。还有，页表可能耗费大量的存储器。页表可能难以定位，因为通常它都要求存储器中的一组相邻的页。

[0146] 该多级页表。图 14 示出了结合多级页表的本发明的使用。一个多级页表通常可能是类似于可变长度平面页表的，然而，多级页表中，虚拟地址的索引部分分散到了多级表中。例如，不同的实施例可以利用一个 32- 位的地址空间。在这种情况下，可以要求该硬件具有两级间接寻址。间接寻址的第一级称作页面 目录而第二级称作页表。当协处理器正

在运行一个特定的设备环境时,只有用于那个设备环境的页面目录和资源列表中的分配听需的页表需要处于存储器中。

[0147] 可能期望提供一种结合本发明的多级页表的一个优点是页入能够用一组不相邻的页来解决。还有,分配地址可能把系统和本地视频存储器中的页混淆,并且只有正在使用的页面目录和页表需要出现在存储器中,并且,各页面目录和页表只要求一个页面(要求没有多级相邻的分配)。然而,尽管有这些优点,但还是存在缺点,就是对存储器的访问要求两次间接寻址。

#### [0148] 表面能级故障

[0149] 随着通用协处理器设备环境虚拟地址空间的增加,该改进调度模型就适当地良好工作,并且通常不需要很多的CPU系统开销,尤其是当存储器压力很小或没有的时候。大多时候,当DMA缓冲器可以提交给调度程序的时候,它索引的资源就已经出现在了存储器中,并且因此该DMA缓冲器不需要通过页入线程来进行任何页入。然而,在调度方面,该模型还能通过增加保持的时间的精确性来进一步改进。

[0150] 在实现本发明的过程中面临的一个问题是,它也许不能预先知道特定的DMA缓冲器要用多长时间运行。这可能导致一个调度程序为下一个DMA缓冲器准备一个可能不利的选择。如果没有其它设备环境是相同或高于当前设备环境的优先权的,或者那个优先级上的所有其它的设备环境都是空的,那么该调度程序就可以从当前设备环境中挑选下一个DMA缓冲器。否则,该调度程序就可以从下一个设备环境中挑选下一个DMA缓冲器,该下一个设备环境具有与当前设备环境相同或比它更高的优先权。然而,并不保证那个选择是正确的。当从下一个最高优先权的设备环境中选出一个DMA缓冲器时,该调度程序会假设用于当前设备环境的DMA缓冲器将运行比一个时间片更长的时间。如果不是这种情况,那么该调度程序也许就很快向着远离那个硬件设备环境的方向转换了。在当前的DMA缓冲器运行比一个时间片短的情况下,该调度程序应当已经从当前设备环境中选择了下一个DMA缓冲器(因为这会将协处理器的有效使用最大化)。

[0151] 当存储器的压力很小或没有的时候,通常,用于下一个DMA缓冲器的两个可能的候选缓冲器可能已经使其所有的资源都出现在了存储器中,所以很可能没有缓冲器要求页入。在那种情况下,当第一DMA缓冲器的时间片结束的时候,该调度程序就会认识到其错误、立即改变其思想并且把正确的DMA缓冲器给予协处理器。

[0152] 然而,在存储器的压力下,该模型可能变得较为不稳定。下一个DMA缓冲器的“调整大小”可能变成确认平滑操作中的一个有利步骤。在存储器的压力下,碰巧是之前所描述的那种情况,用于该下一个DMA缓冲器的两个可能的候选缓冲器中的一个就要求某种页入并且因此会发送给预备线程。在那种情况下,通常对于调度程序在最后一分钟内“改变其思想”并且交换这两个DMA缓冲器是不合理的。然而,注意,可能做出这样的改变,并且这样的实际操作并没有落在本发明的说明书之外。例如,在下一个DMA缓冲器的准备完成并且其它可能的DMA缓冲器候选项并不要求页入的情况下,就会交换一个DMA缓冲器。这可能包含通过辅助存储器管理器用于可共享分配的某种特定的支持程序。

[0153] 上述可能时间保持误差,自然而然地,不是很坏,并且能够在随后的时间片过程中大约错过的处理时间里通过给予设备环境而运行,其中该处理时间是该设备环境。还有,在大多数情况下,DMA缓冲器包含足够的命令来用于多个协处理器时间片的运行,所以各设

备环境都能获得其完整的时间片。然而，在存储器的压力下，可能强迫辅助存储器管理器将 DMA 缓冲器（如上所述）拆分成较小的缓冲器，从而减少各设备环境的工作组。DMA 缓冲器的这种拆分减少了 DMA 缓冲器的大小，并且相应地，增加了上述的量化问题。

[0154] 另一个问题可能在存储器的压力下引发，这个问题就是该模型可能会人为地制造额外的压力，因为可能有多于 DMA 缓冲器实际使用的存储器进行了页入。所有那些页入的额外的存储器都将可能在下一个时间片之前被收回，并且会需要再次页入。在页入动作已经很高的时候，这会导致页入动作的增加。在基本的和改进的模型中，该辅助存储器管理器能够通过选出期望的收回策略来查出增加页入的问题的地址。例如，在轻微的存储器压力下，各设备环境也许会在其工作组中具有一定数量的存储器。在从其它设备环境中收回存储器之前，该辅助存储器管理器也许会努力地首先从当前设备环境中收回存储器，并且将其 DMA 缓冲器拆分从而使其适于现有的工作组。一旦特定设备环境的 DMA 缓冲器拆分成了其最小的尺寸，该辅助存储器管理器就只能从另一个设备环境中收回存储器了。

[0155] 解决这些问题的一个优选方法是允许协处理器所需的存储器的请求故障。那样，我们就能够确认只有协处理器所需的存储器的子设备出现在了存储器中。

[0156] 出于改进模型的目的的该级故障是在表面间隔尺寸上。然而，应当理解，任何级的故障都可以期望结合本发明来使用。还有，注意，就页表硬件而言，该硬件只会考虑分配的第一页的状态，从而确定分配是否合理，因为该辅助存储器管理器会立刻把整个分配带到存储器中。

[0157] 在不同的实施例中，硬件可能在以下两种情况的一种发生的时候产生页面故障：

[0158] 1) 发生了设备环境的转换，转换到了索引无效环形缓冲器或 DMA 缓冲器的设备环境。

[0159] 2) 将汲取出一种原语，并且某些所需的存储器资源并未出现（例如顶点遮掩码、顶点缓冲器、纹理）。

[0160] 注意，在第二种情况下，可能要求硬件在收回每个三角之前重新取样其当前的存储器资源。对于辅助存储器管理器，可能会在任何时刻使虚拟地址或编号无效，包括当该存储器资源正在运行的时候。也期待该硬件可以允许所有当前存储器正在进行的查询。该辅助存储器管理器可以使用那个信息来确定一个特定的分配什么时候会给硬件使用。该辅助存储器管理器可能假设如果一个分配没有出现在当前协处理器正在使用的资源列表中，在具有其无效的虚拟地址或编号之后，那么收回那个分配就是安全的，因为该协处理器不能访问那个分配地址。企图这么做就会导致页面故障。

[0161] 通过下面对一个表面能级故障模型的更详细的解释，提供了结合本发明的使用表面能级故障的进一步的解释。下面的模型是某些实施例的一个例子，并且不能当作是对本发明结合此处所提供的调度模型的设备环境之外的其它应用程序的表面能级故障的原理的可能的使用的限定。

[0162] 第一，用于存储器资源的分配进程表可能与这个文件中的通用协处理器设备环境虚拟地址空间中所述的是相同的。详细情况参见那个部分。

[0163] 第二，用于 DMA 缓冲器和资源列表的收回命令方案与这个文件中的通用协处理器设备环境虚拟地址空间中所作的解释也是相同的。在这个模型中，即使该图形硬件支持表面能级故障，也需要有该资源列表。该辅助存储器管理器（此处称作“VidMm”）使用该资源列表，从而获得有关存储器地址分配的使用信息。那个使用信息允许 VidMm 在它需要腾出

存储器中的空间的时候确定用于收回的 候选缓冲器。

[0164] 随着表面能级故障的增加,关于资源列表没有安全性的关系,所以它能以用户模式来创建。如果恶意的应用程序将无效数据放在了资源列表中,那么会发生的最严重的故障就是要遭受该恶意应用程序的执行。VidMm 可能产生一关于收回条件的不合理的选择,其将导致用于那个应用程序的外部页入活动。

[0165] 具有表面的命令故障的调度模型可能在许多方面不同于没有使用表面能级故障的模型。通常,准备好的列表中的处理可以直接提交给协处理器,而不需要预备阶段。该调度程序会保存用于设备环境的一个专用列表和一个页入线程,该设备环境要求解决页面故障。已有专门的 VidMm 协处理器设备环境用于页入操作。最后,链接提交给一个设备环境的 DMA 缓冲器,从而形成一个单独的工作项目。

[0166] 在这个模型中,取消了预备阶段。该调度程序可能要求协处理器从一个设备环境直接转换到另一个设备环境,并且它可以假设所有的设备环境都准备好在任意时刻运行了。如果要转换到的设备环境没有将其所有的存储器资源给出到存储器中,该硬件就会出错,并且该设备环境就会添加到列表上(例如,参见图 15 的页入列表),以便该页入线程可以开始致力于解决该故障。

[0167] 图 15 示出了结合这个模型由调度程序维护的示例性处理的一个列表。参见图 15,当发生故障时,引发该故障的设备环境就会添加到该页入列表中。然后,一个页入线程就可以解决该故障。该页入线程可能选择最高优先级的发生故障的设备环境,以便首先解决该故障。可以使用一种定期优先权提升来确信低优先权设备环境将最终获得足够高的优先权,从而解决其故障。当页入作业线程正在解决故障时,该调度程序可以调度更多的设备环境,这些设备环境都准备好了在协处理器上运行。当协处理器正在工作时,该页入作业线程能够通过要求该驱动器从地址中映射或者非映射到分配地址来操作视频存储器。

[0168] 可能当前协处理器正在使用的分配地址都将被无效掉。协处理器试图访问这样一个分配地址的下一时刻,就可能出错。然而,因为该协处理器不可能在任意一个时间上立即出错(例如某些协处理器将只对三角之间的当前分配地址的状态重新取样),所以在它已经变得无效之后,可能协处理器就会需要使用用于某个时刻的分配地址。

[0169] 为了防止发生那种情况,即使其虚拟地址或编号已经被无效了,VidMm 也可以确信用于该分配地址的存储器将保持有效直到转换到下一设备环境。这是通过在 VidMm 专用的协处理器设备环境中由于页入而将存储器转移来实现的。因为在一个单独的设备环境中进行了存储器转移,所以我们可以相信在改变存储器的内容之前将存在一个存储器的转换。对于索引系统存储器的虚拟地址或编号,在收回的过程中是不存在存储器转移的。在那种情况下,通过使其受到约束,VidMm 可以确信该系统存储器保持有效,直到该协处理器设备环境转换到 VidMm 的专用设备环境。

[0170] 该 VidMm 专用的协处理器设备环境是一种常规的协处理器设备环境,VidMm 利用该常规的协处理器设备环境在系统存储器和视频存储器之间进行存储器转移。该 VidMm 设备环境是一种可变优先权设备环境,该设备环境采用页入列表中的最高优先权项的优先权。使得所有页入操作在单一的设备环境中连续执行就简化了用于 VidMm 的同步模型。

[0171] 这个模型中另一个值得关注的不同点是为一个特定的设备环境提交的所有 DMA 缓冲器都能够链接起来从而形成一个单任务的方式。在以前的模型中,形成工作项的各 DMA

缓冲器和各设备环境都会包含一个那些工作项的列表。该调度程序不是必然要调度那个设备环境；它会调度（并且开始准备用于）与一个设备环境相关的特定工作项。在那个工作有机会完成之前，该调度程序可能不得不选择下一个工作项。各工作项不得不在其提交之前做好准备，从而该调度程序不得不实现得知该下一工作项是什么、哪一个总是不可能的。

[0172] 具有了表面能级故障，DMA 缓冲器就不需要准备工作了。为此，该调度程序不必把设备环境看作工作项的收集。相反，该调度程序真正地调度这些设备环境，并且一旦设备环境获得了该协处理器的控制，它就会保持该协处理器的控制。可以允许某些事件暂停(halt) 该协处理器的设备环境控制，例如：

[0173] 1) 该协处理器完成了当前已经排列好的所有的命令

[0174] 2) 该协处理器产生了由一个无效存储器访问所引起的页面故障

[0175] 3) 该调度程序请求转换到一个不同的设备环境

[0176] 4) 在该 DMA 数据流中的无效命令之后，该协处理器产生了一个无效操作中断

[0177] 图 16 提供了一种说明按照上述方案的本发明的多种实施例的框图。参见图 16，在相同的硬件环境下，从第一设备环境的插入到第二设备环境的插入中，这两端表示级数。在左边一端上，该调度程序要求该内核驱动器把一个特定的 DMA 缓冲器插入到协处理器设备环境 #1 的环中。该环可以由该驱动器来修改，并且可以更新协处理器的尾部从而索引新的位置。协处理器设备环境 #1 中的 DMA 缓冲器的插入是在协处理器设备环境 #1 的专门锁定的保护下发生的。这样，其它线程就能把 DMA 缓冲器插入到其它线程设备环境的环中。

[0178] 在右边一端上，该调度程序要求该内核模式驱动器把一个特定的 DMA 缓冲器插入到协处理器设备环境 #2 的环上。然而，该环已经满了，因此，线程 B 将被阻塞，直到环中的某个空间被释放。应当注意这样一种情况，由于是在其自己的环中插入一个新的 DMA 缓冲器，等待的线程 B 不阻塞线程 A。

[0179] 这个模型中，各设备环境都具有其自己的 DMA 环，该环能够保持重定向到要运行的 DMA 缓冲器的部分。在提交时间里，该调度程序可以试着把提交的 DMA 缓冲器添加到那个设备环境的环上。如果该环已经满了，那么该调度程序就会等待直到该环中有足够的空间用于另一个提交。注意，这种等待将只是把进一步的提交阻塞到要提交的特定的设备环境中。它并不把提交阻塞给其它设备环境。换言之，多个线程都能并行地把工作项添加到其自己的设备环境中。

[0180] 因为可以把新的 DMA 缓冲器添加到一个运行设备环境的队列里，所以该协处理器可以在产生一个中断之前对该队列的尾部进行重新取样，从而报告设备环境是空的。当然，也可能在该协处理器取样其队列之后立即把 DMA 缓冲器添加到该队列上。然而，在产生中断之前就取样该队列的尾部会减少这种情况发生的可能性，并且增加调度的准确性。当该调度程序告知设备环境是空的时，它将排列该驱动器从而了解实际上是不是那种情况。对于该驱动器，为了确定当前它里面有没有排列好的尚未处理的命令，可能要访问所保存的协处理器设备环境。图 17 提供了一种描述这个模型的伪码算法。

[0181] 如后面将要详细描述的那样，当允许该内核模式驱动器创建包含优先命令的 DMA 缓冲器时，将引入与限定的对有特权的 DMA 缓冲器的概念，从而允许 DMA 缓冲器直接以用户模式来创建而不会威胁到系统的安全。

[0182] 用这个模型表示的多种实施例可以结合限定的对有特权的存储器的概念来使用，

该限定的对有特权的存储器将在这个文件的后面进行描述。从现在起,注意,在这个模型中可能引发的一个问题,因为在之前所表示的存储器虚拟化模型中,限定的 DMA 缓冲器与 (versus) 有特权的 DMA 缓冲器能够访问的存储器之间并没有区别;所有虚拟存储器都是可存取的。这就意味着某些存储器资源,象页表或环形缓冲器,可能通过该协处理器虚拟地址空间不是恰好 (appropriately) 可见的,因为那样就会允许一个恶意的应用程序写到该页表或者该环形缓冲器上。为此,该硬件会设计成支持用于多种类型的资源的物理地址和用于其它类型的资源的虚拟地址。

[0183] 一种解决该问题的不同的方法是添加优先存储器的概念。在多种实施例中,优先存储器只能从有特权的 DMA 缓冲器中进行访问,并且如果限定的 DMA 缓冲器试图访问有特权的存储器位置,那么该协处理器就会发生页面故障。另一方面,一个有特权的 DMA 缓冲器能够无区别地对有特权的存储器和没有特权的存储器两个都进行访问。为了支持有特权的存储器,该硬件必须具有一个装置,从而按照通用编号原理 (在基于编号虚拟化的情况下) 或者通用页面原理 (在基于页表虚拟化的情况下) 规定该存储器是否是有特权的。

[0184] 注意,为了支持有特权的存储器,具有一个页表的支持表面能级故障的协处理器不再会仅在存储器资源的基地址上出错。该协处理器必须考虑当前资源所覆盖的所有页表项,并且确信它们都具有正确的保护位组。只检查存储器资源的第一页可能会允许一个恶意的应用程序对限定的存储器基址之后的有特权的存储器进行存取,其中该限定的存储器基址在限定的 DMA 缓冲器中作了规定。

#### [0185] 运行列表

[0186] 之前所示出的命令故障模型可能大量使用了中断从而发出多个事件的信号。这些事件中的某些事件,象页面故障,在存储器压力下发生的频率会很高。在中断是瞬时中断的时间和由 CPU 给予协处理器一个新的任务的时间之间,该协处理器可能供应不足。为了隐藏中断等待时间并且保持协处理器忙着,我们引入了运行列表的概念。

[0187] 简单讲,运行列表就是一个能够由协处理器运行而不需要 CPU 干预的协处理器设备环境的列表。这些设备环境可以按照所给的顺序或者按照经检验更便于本发明的那些实际应用的其它任何顺序来运行。根据用于任何广泛变化的原因的运行列表,该协处理器能够从一个设备环境转换到下一个,其中这些原因能够结合本发明得以实现,例如:

[0188] 1) 当前设备环境是空的,即没有任何事情可做。

[0189] 2) 当前设备环境发生了页面故障。

[0190] 3) 当前设备环境发生了一般的保护故障 (如果协处理器支持的话)

[0191] 4) 要求协处理器转换到一个新的运行列表

[0192] 在多种实施例中,当协处理器从运行列表中的一项转换到下一项时,就中断 CPU 但并不阻塞,还会将设备环境转换到该列表中的下一项,并且开始运行这一项。该运行列表的头部可以是该调度程序试图先运行的设备环境,而该运行列表的其它元素可以在那里部分地保持中断等待时间过程时协处理器是忙着的。CPU 一旦接收了协处理器进行了转换而离开该列表的头部的中断信号,CPU 就会创建一个新的运行列表并且将其发送给该协处理器。

[0193] 当该协处理器进行转换而离开该列表的头部的时候,可以开始执行该运行列表中的下一个设备环境,而它产生的中断是其朝着 CPU 运行。CPU 将会产生的新的运行列表的头部可能与该协处理器要转换的设备环境是不同的。在那种情况下,该协处理器将需要再次

转换，并且不可能花时间对那个设备环境做许多有用的工作。

[0194] 然而，CPU 创建的新的运行列表的头部的设备环境可能与之前的运行列表的第二个元素的设备环境是相同的，因为设备环境的优先权不会因创建了最新的运行列表而改变。在那种情况下，该协处理器就已经提前开始处理该正确的设备环境了。

[0195] 图 18 中提供了一种表示该运行列表的概念的框图。当一个运行列表包含在本发明的多种实施例中时，该调度程序的运行设备环境可用当前的运行列表替换。引入一个第二运行列表，称作待处理的运行列表，以便简化运行列表转换的同步。该当前运行列表是一种该调度程序能够假设该硬件当前正在运行的设备环境的列表，而待处理的运行列表是一种调度程序想让该硬件从一个运行列表变换到另一个运行列表时使用的待处理的性运行列表。当调度程序想要变化到一个新的运行列表时，它就创建一种待处理的性运行列表，并且要求该协处理器向它进行转换。一旦该调度程序接收来自该协处理器（通过一中断）已经开始运行该新的运行列表的确认，该待处理的性运行列表就变成该新的当前运行列表，并且会清空该待处理的性运行列表。

[0196] 当该待处理的性运行列表是空的时，该硬件可以运行该当前运行列表中的一个设备环境或者它可以是空闲的。当该待处理的性运行列表不是空的时，该调度程序也许就不知道该硬件当前正在运行的是哪个运行列表，直到它从发生 转移的协处理器中接收到确认。

[0197] 某些事件可以要求该调度程序把该运行列表重新列为优先。例如，一个页面故障可能解决了使高优先权协处理器设备环境准备运行的问题。为了简化这种事件的同步，该调度程序可以遵循的一般规则是：只有当不存在一个之前的事件所提交的待处理的性运行列表的时候，才会提交一个新的运行列表（待处理的性运行列表）。试着把一个待处理的性列表用另一个替换可能很难同步，因为该列表已经给予了协处理器，因此在任何时候都可能发生该转移，并且只有在这个事实发生之后才会通知给调度程序。

[0198] 在后一种情况下，该运行列表的重新列为优先可能会委托给该设备环境转换处理机。在将来的某个时候，可能接着要求该处理机从该待处理的性列表向运行列表发送该转移信号，并且那时，该处理机可以产生一个新的运行列表，以便如果已经改变了该优先权，则把新的运行列表发送给该硬件。

[0199] 运行列表转换同步。在一个运行列表模型中，当该图形硬件转换设备环境时，它会产生一个中断。因为中断传送和处理不是瞬时的，所以可能在 CPU 实际上已经中断之前就会产生多个中断。如果不进行适当地同步，该调度程序就会被搅乱并且做出不正确的调度决定。

[0200] 该调度程序会把注意力放在辨别上的两个关键事件是：第一，当协处理器进行了转换而离开运行列表的头部的时候，以及第二，当协处理器变化到该待处理的性运行列表的时候。只采用来自各设备环境转换上的简单的中断的信息，在这些事件之间进行辨别 (differentiating) 可能很困难。为了进一步说明这一点，要考虑以下例子：该协处理器目前正在运行的运行列表 A，其中该运行列表 A 由设备环境 1-3-5-2 组成，并且该调度程序想要变为运行列表 B，其中该运行列表 B 由设备环境 4-1-3-2 组成。

[0201] 下面是两种可能发生的情形：

[0202] 情形 #1

[0203] 协处理器目前正在执行运行列表 A(1-3-5-2)。

- [0204] 提交看关设备环境 4 的命令,其中,该设备环境 4 是空闲的并且比设备环境 1 的优先权高。生成运行列表 B(4-1-3-2) 并且该调度程序把运行列表 B 提交给协处理器。
- [0205] 运行设备环境 #1 直到协处理器从运行列表 B 中转移到设备环境 #4。
- [0206] 协处理器产生一个中断,发出转移的信号。
- [0207] 协处理器从设备环境 #4 向 #1 转移,然后在中断 CPU 之前转移到 #3。
- [0208] 中断 CPU,并且调用该设备环境转换处理机。
- [0209] 一个驱动器对当前协处理器设备环境进行取样,该设备环境是 #3。
- [0210] 情形 #2
- [0211] 协处理器当前正在执行运行列表 A(1-3-5-2)。
- [0212] 提交有关设备环境 4 的命令,其中,该设备环境 4 是空闲的并且比设备环境 1 的优先权高。调度程序把运行列表 B 提交给协处理器。
- [0213] 当调度程序正忙于创建运行列表 B 的时候,协处理器转移到设备环境 #3。
- [0214] 协处理器产生一个中断,向设备环境 #3 发出转移的信号。
- [0215] 中断 CPU,并且调用该设备环境转换处理机。
- [0216] 一个驱动器对当前协处理器设备环境进行取样,该设备环境是 #3。
- [0217] 在两种情况下,在设备环境转换中断的时候当前运行的设备环境都是 #3。然而,注意,没有附加信息的情况下,调度程序是不能辨别两种情形的。在第一种情形中,协处理器进行转换而离开了运行列表 B 的头部,并且因此调度程序就需要生成运行列表 C,还要求该协处理器变换到运行列表 C。然而,在第二种情形中,该第二运行列表甚至还没有启动,并且因此调度程序只能等待。
- [0218] 上面的例子示出了单独的设备环境转换中断也许不足以完全地支持调度模型中的运行列表。需要有一些更多的信息来辨别这两种情形。接下来的情形详述了某些方式,这个问题可以沿着硬件支持的方向进行寻址,该硬件支持在这样的问题的寻址中是有用的。
- [0219] 双元素运行列表。这种同步方法要求协处理器支持某些附加特征。能够结合双元素运行列表的实现得以支持的这些特征如下:
- [0220] 1) 一个双元素的运行列表。
- [0221] 2) 能够在各设备环境转换时产生一个中断(包括从设备环境 X 到 X 的假的设备环境转换)。
- [0222] 3) 用于 VidMm 的一种方式,在任意时刻查询当前正在运行的协处理器设备环境。
- [0223] 4) 中断之前把输出的(outgoing)的协处理器设备环境保存在存储器中。
- [0224] 5) 以一种方式保存协处理器设备环境,该方式是设备环境对于 CPU 是可读的,从而允许调度程序在设备环境转换之后确定该原因。
- [0225] 注意,尽管硬件可以用于支持以上功能,但这种特定的硬件没必要允许调度程序辨别常规的设备环境转换和运行列表转换。相反,当构造一个运行列表时,通过总是遵守一套简单的规则,调度程序可以辨别那两种事件之间的区别。尽管该特定的规则可以针对本发明的不同实施例变化,但提供这种功能的示例性规则是:第一,当前运行列表的第一设备环境不会出现在新的待处理的性运行列表中,以及第二,如果当前运行列表的第二设备环境不是新的待处理的性运行列表的头部,它必然根本不在该新的待处理的性运行列表中。下面是一种假设的表格,当遵循这两条示例性规则时,该调度程序可以在从一个设备环境

向另一个设备环境转移的过程中做出这种假设。在以下表格中,运行列表 A 是由设备环境 1-2 组成的;第二运行列表 B 是由设备环境 2-3 组成的;以及第三运行列表 C 是由设备环境 3-4 组成的。

#### [0226]

从A向B转移	采用的装置/动作
当终端CPU被中断时的当前设备环境#	采用的装置/动作
1	小故障,忽略该中断 这种小故障是由之前的运行列表从(X,1)到(1*,Y)的转换而引起的,我们错误地中断了按照运行列表转换的从X到1的转移。该真实的转移是X到1,然后1到1*。该当前中断是用于1到1*的转移的,并且可能被忽略(要求协处理器产生这种中断以便能够由调度程序检测到从1-X到1-Y的转移)。
2	发生运行列表转换。 这并不总是真的,并且可能引起之前的小故障。如果当前的转移实际上是1*-2*,那么CPU就会再次因2*-2**或2*-3***而中断。该待处理的运行列表(B)变成当前运行列表,并且该待处理的运行列表被清空。该调度程序需要处理设备环境转换而离开设备环境1*(不包括:页面故障)。
3	发生运行列表转换,并且已经完成了第二列表的头部。 运行列表B已经结束。该待处理的运行列表(B)变成了当前的运行列表。由调度程序创建一个新的待处理的运行列表并且发送给该协处理器。该调度程序需要处理设备环境转换而离开设备环境2***(不包括:页面故障)。

#### [0227]

从A向C转移	采用的装置/动作
当终端CPU被中断时的 当前设备环境#	采用的装置/动作

#### [0228]

1	小故障,忽略该中断 这种小故障是由之前的运行列表从(X,1)到(1*,Y)的转换而引起的,我们错误地中断了按照运行列表转换的从X到1的转移。该真实的转移是X到1,然后1到1*。该当前中断是用于1到1*的转移的,并且可能被忽略(要求协处理器产生这种中断以便能够由调度程序检测到从1-X到1-Y的转移)。
2	设备环境在当前运行列表中进行转换。协处理器转换到设备环境2*。该调度程序需要处理设备环境转换而离开设备环境1*(不包括:页面故障),否则就对运行列表什么都做不了。
3	发生运行列表转换。该待处理的运行列表(C)变成当前运行列表,并且该待处理的运行列表被清空。该调度程序需要处理设备环境转换而离开设备环境1*(不包括:页面故障)。不知道设备环境2*是否曾经执行过,它将被重新调度。
4	发生运行列表转换,并且已经完成了第二列表的头部。运行列表C已经结束。该待处理的运行列表(C)变成了当前的运行列表(硬件彻底空闲了)。由调度程序创建一个新的待处理的运行列表并且发送给该协处理器。不知道设备环境2*是否曾经执行过,它将被重新调度。

[0229] 实现运行列表的这种方法可能是最简单的,并且不必要求有效的附加硬件支持。然而,注意,以上表格中的这些列表在尺寸上进行了限定(超过两个的尺寸的扩展是不切实际的),并且某些信息,不是关键性的,可能在设备环境转换的过程中丢失。例如,该调度程序不会总是知道从 A 向 C 的转移中设备环境 #2 是否曾经运行过。其可能已经被执行,导致一个页面故障,但是也将其中断隐藏,由其他环境转换。在那种情况下,该调度程序就不可能知道它曾经发生过故障,更不会重新调度它。

[0230] 调度事件的协处理器轨迹。当硬件事件调度的某些历史信息提供给该调度程序的

时候,该运行列表可以很容易地扩展到尺寸N。采用单一中断的一个问题是多个中断可能会挤在一起,并且它也许不可能精确地确定发生了什么引起了一中断。结合本发明的方法,依照硬件特征,这是可以寻址的。通过实现硬件,可以把一个设备环境转换历史写到调度程序可读的特定的系统存储器位置上。为了解释本发明的这个方面,考虑以下情形:

- [0231] 1) 该调度程序调用运行列表 A(1-2-3-4-5)。
- [0232] 2) 针对设备环境 #1, 时间量到期, 并且该调度程序发送新的运行列表 B(2-3-4-5-1)。
- [0233] 3) 当在 CPU 上处理该量到期时, 协处理器完成设备环境 #1, 因为它变成了空的, 并且因此转移到设备环境 #2。由于这一事件, 该协处理器产生一设备环境转换。
- [0234] 4) 协处理器从 CPU 中接收有关新的运行列表的通知, 并且因此向它转移。由于这个事件, 协处理器产生一个设备环境转换中断。
- [0235] 5) 当处理新的运行列表的设备环境 #2 中的再现命令时, 协处理器遇到一种页面故障, 并且因此转换到设备环境 #3。由于这个事件, 该协处理器产生了一个设备环境转换中断。
- [0236] 6) 设备环境立刻碰到页面故障, 并且因此协处理器转换到设备环境 #4。由于这个事件, 该协处理器产生了一个设备环境转换中断。
- [0237] 7) 最后, 为了设备环境转换而中断了 CPU。实际上, 四个设备环境转换都是因为引发了初始中断而发生的。

[0238] 图 19 说明了以上情形中的硬件历史装置的操作。为了支持这种历史装置, 该硬件可以构造为能够执行下面的任务。以示例而非限定的方式提供这些任务:

[0239] 1) 为该历史缓冲器指定一个基地址。可能存在一个单独的历史缓冲器通用协处理器。在优选实施例中, 它可能是 PCI 或 AGP 存储器之一中的系统存储器位置。这可以由该操作系统列在一 4KB 范围上。对于 PCI 快速存储器, 更可取 的是, 对这个缓冲器的访问可以用一个探听周期来实现, 以便该系统存储器缓冲器能够高速地缓存从而进行更有效地 CPU 读取。

[0240] 2) 指定该历史缓冲器的大小。该历史缓冲器可以至少是运行列表的大小的两倍。这是为了确保该缓冲器中有足够的空间, 以便处理最坏的情况的情形, 该情形是当前运行列表和待处理的性运行列表两个都在中断发生之前就已经完成了。

[0241] 3) 指定协处理器写指针, 该指针可以是一个把要写的最终事件立即传给历史缓冲器的地址。VidMm 也许可以在任何时候查询这个指针, 包括在协处理器正在运行的时候。在更新该指针以便确信该调度程序已经获取了相关数据之前, 该历史缓冲器中的数据可以适当地刷新存储器。

[0242] 多种实施例都可以构造该历史缓冲器, 以便它对于 DMA 缓冲器是不可见的, 这些实施例都是以用户模式来创建的。如果历史缓冲器对于限定的 DMA 缓冲器是可见的, 那么恶意的应用程序就会写到历史缓冲器中、破坏该调度程序并且可能导致系统崩溃或者更坏的情况。为此, 这些实施例中的历史缓冲器可能由硬件通过一个物理地址或者通过虚拟地址来访问, 该虚拟地址只对有特权的 DMA 缓冲器才是可见的。在这些实施例中, 要求协处理器在历史缓冲器的尾部进行偏移而不需要 CPU 的干预。

[0243] 注意, 根据马上要描述的实施例的运行列表解决不了用于协处理器的所有需求, 从而由于相同的原因在相同的设备环境上会发生多次故障。对此, 一个原因是该调度程序

通常会创建一个新的运行列表而协处理器则忙于执行当前的运行列表。因为调度程序也许需要包括新的运行列表中的已经出现在之前的运行列表中的设备环境，所以在其提交正构造的运行列表的时候和该运行列表被提交给该协处理器的时候中间，改变重复的设备环境的状态成为可能。

[0244] 限定的对有特权的 DMA

[0245] 随着改进的调度模型中存储器保护的引入，发送给协处理器的 DMA 缓冲器可以主要是由该运行应用程序的处理中的用户模式驱动器来创建。这些 DMA 缓冲器可以映射到该应用程序的处理中，该用户模式驱动器能够直接写至其上，并且该内核驱动器不能使它们等效。DMA 缓冲器可以由该应用程序乱写附带地访问它们的虚拟地址，或者是一个恶意的应用程序故意乱写的。为了允许该驱动器模型保证安全，即不允许应用程序访问它不该访问的资源，以用户模式创建的 DMA 缓冲器会限定它们允许做什么。特别的，创建的 DMA 缓冲器可能以下面示例性方式限定功能性：

[0246] 1) 它们可能只包含对虚拟地址的索引，根本没有对物理地址的索引（包括电子篱笆）。

[0247] 2) 它们可能不允许包含会影响当前显示（例如 CRT、任意访问控制（DAC）、技术文件管理系统（TDMS）、电视输出口（TV-OUT）、互联网 2（I2C））总线的指令。

[0248] 3) 它们可能不包含通常会影响该适配器（例如锁相环路（PLL））的指令。

[0249] 4) 它们可能限定动力管理和 / 或配置空间。

[0250] 5) 它们可能不允许包含会妨碍设备环境转换的指令。

[0251] 正确的一组寄存器将可能从硬件向硬件变化，其中这组寄存器能够以用户模式创建的 DMA 缓冲器中编程。然而，不管硬件如何，寄存器都会遵循一种常用规则，也就是这样的一个 DMA 缓冲器应当只允许利用虚拟地址访问资源和电子篱笆的操作。为了提高增强的安全性，可能要求这样的 DMA 缓冲器不允许应用程序使用该应用程序不该访问的存储器，否则可能以某种灾难性的并且不可还原性的方式影响该硬件。

[0252] 为了防止以用户模式创建的 DMA 缓冲器访问某种功能，可以在协处理器中执行多种方法。这些方法可以根据功能性的自然性和该功能是否需要在应用程序的协处理器设备环境流中进行查询而变化。某些有特权的操作通常需要在协处理器设备环境流中进行查询，该协处理器设备环境流包含以用户模式（例如应用程序再现）创建的 DMA 缓冲器和以内核模式（例如查询的触发）创建的有特权的 DMA 缓冲器。

[0253] 不需要查询的功能性。大多有特权的功能性都不需要在应用程序的协处理器设备环境流中进行查询。例如下面的功能性就不需要查询：

[0254] 1) 对 CRT 计时进行编程。

[0255] 2) 更新用于该 DAC 的查询表（注意，不是绝对要求对 DAC LUT 进行编程从而作为有特权的功能性，因为任何应用程序都能再现为它想要的任何方式的主要的屏幕，并且对查询表（LUT）重新编程将会不允许应用程序给予用户访问，从而告知它还是不能访问）。

[0256] 3) 对显示输出（TDMS, TV-OUT, ...）进行编程

[0257] 4) 与子设备 / 子监视器（I2C, ...）进行通信

[0258] 5) 对时钟（PLL）进行编程

[0259] 6) 改变协处理器的功率状态

[0260] 7) 构造该协处理器（构造空间，基本输入输出系统 bios, ...）

[0261] 这种功能性通常需要遵循一种系统事件,该系统事件是完全独立于应用程序再现流的。(例如引导程序、方法变换、pnp 检测、动力管理。)照这样,这种功能性就不需要在特定的应用程序协处理器设备环境中进行查询。当发生该特定的系统事件而没有来自用户模式驱动器的任何干扰的时候,这种功能性可供内核模式驱动器本身使用。

[0262] 对于这种功能性,仅通过存储器映射输入输出 (MMIO), IHV 就能够决定使得所有底层寄存器都是可读的。因为寄存器通常只是映射到内核空间中,所以对于应用程序或者用户模式驱动器,也许不能访问它们,并且因此,该功能性得以有效地保护。

[0263] 另一种方法可能是实现通用协处理器设备环境特权级。采用这种方法,某些设备环境就会限定它们可以做什么而其它的就不能做。在那种情形下,以用户模式创建的应用程序设备环境就可以排列成一个限定的设备环境。另一方面,该内核模式驱动器可能使用有特权的设备环境从而提交该有特权的功能性。

[0264] 需要查询的功能性。因为限定了能够插入到以用户模式创建的 DMA 缓冲器中的命令,所以能够实现该改进模型,从而要求协处理器支持限定的 DMA 缓冲器(这是遵守之前的条件的 DMA 缓冲器)和有特权的 DMA 缓冲器。为了允许沿着协处理器设备环境的再现流对有特权的功能性进行查询,就需要有特权的 DMA 缓冲器。

[0265] 有特权的 DMA 缓冲器可包含没有特权的 DMA 缓冲器中找到的任何指令。本发明的多种优选实施例都可以实现有特权的 DMA 缓冲器,至少对下面的内容是允许的(后面部分中将作进一步详细的解释):

[0266] 1) 有特权的电子篱笆的插入

[0267] 2) 触发指令的插入

[0268] 3) “非设备环境转换”区域的插入

[0269] 还有,有特权的 DMA 缓冲器能够对 IHV 想要的任何硬件寄存器进行编程,并且如果需要的话对虚拟和物理存储器两个都能进行访问。有特权的 DMA 缓冲器也许不是以用户模式构造或可见的。只有委托的内核部件能够访问和创建有特权的 DMA 缓冲器。

[0270] 下面的部分表示了实现有特权的 DMA 缓冲器的三种可行的方式,并且想阐明有特权的 DMA 缓冲器的实现的概念而不是限定本发明可以实现的不同方式:

[0271] 1. 仅以内核模式创建 DMA 缓冲器

[0272] 支持不要求任何特定的硬件支持的有特权的 DMA 缓冲器的一种方式是要求发送给该硬件的实际的 DMA 缓冲器以内核模式来创建。在那种情形下,该用户模式驱动器会创建一个命令缓冲器并且将它提交给内核模式驱动器,其中该命令缓冲器非常类似一种 DMA 缓冲器。该内核模式驱动器会确认这个命令缓冲器并且将这个命令缓冲器复制到只以内核模式可见的 DMA 缓冲器中。在确认的过程中,该内核模式驱动器会检验不存在有特权的指令。这就类似于该确认,它是基本模型要求的但不要求对存储器访问的确认,因为存储器是虚拟化的。

[0273] 2. 把有特权的命令直接插到该环中

[0274] 或许,最简单的支持有特权的 DMA 通道的硬件方法是直接把有特权的命令插入到协处理器设备环境的环中。该环本身总是一个有特权的通道,仅对内核模式是可见的。这在图 20 的框图中进行了描述。

[0275] 3. 间接指定特权

[0276] 图 21 中说明了一种支持协处理器中的限定的对有特权的 DMA 缓冲器的不同的方法。参照那个图,注意,该起始和结束地址都以 DWORD 来定位。可以重复利用该地址中不用的位来规定标记。该起始地址的第一位可以规定重新定位的 DMA 缓冲器是一个有特权的 DMA 缓冲器。为了增强安全性,有特权的 DMA 缓冲器可以访问辅助存储器中的物理地址。限定的 DMA 缓冲器可以访问协处理器设备环境虚拟地址空间中的虚拟地址。

[0277] 在这种方法中,间接命令中的一位可以插入到该环形缓冲器中。该位指示正在执行的 DMA 缓冲器是不是一个有特权的 DMA 缓冲器。这就意味着该环形缓冲器本身可以由协处理器利用一个物理地址来访问,并且在协处理器虚拟地址空间中可以是不可见的。允许该主要的环形缓冲器在协处理器虚拟地址空间中是可见的就会允许一个恶意的应用程序写到主要的环形缓冲器上,并且允许它在有特权的级别上运行命令,这就相当于是在大多数计算环境中的安全性破坏。在这方面,有特权的 DMA 缓冲器会通过一个物理地址而不是象限定的 DMA 缓冲器那样的虚拟地址来访问。

[0278] DMA 控制指令

[0279] 为了该调度程序和辅助存储器管理器跟踪任何协处理器设备环境的级数并且控制那个设备环境的 DMA 流中的指令的流程,该协处理器可以构造为支持其 DMA 流中的以下示例性指令:

[0280] 1) 电子篱笆 (限定的和有特权的两者)

[0281] 2) 陷阱

[0282] 3) 能够 / 不能进行设备环境转换

[0283] 电子篱笆。一个电子篱笆可能是包含一块数据 (例如 64 位块的数据) 和一个能够插入到 DMA 缓冲器中的地址两部分的指令。当协处理器从该流中读取指令时,它会致使协处理器把有关该电子篱笆的数据块写到指定的地址上。协处理器可能把电子篱笆的数据写到存储器中之前,它必须确信来自原语的位与电子篱笆指令之前的元素已经收回并且已经适当地写入了存储器中。注意,这并不意味着协处理器需要阻塞整个流水线。当协处理器正在等待收回电子篱笆之前该指令的最后元素的时候,可以执行遵循该电子篱笆指令的原语。

[0284] 当符合以上描述的任何电子篱笆可以结合本发明来使用的时候,特别的,两种类型的电子篱笆在此将作进一步的描述:常规电子篱笆和有特权的电子篱笆。

[0285] 常规电子篱笆是能够插入到以用户模式驱动器建立的 DMA 缓冲器中的电子篱笆。因为 DMA 缓冲器的内容来自用户模式,所以是不可信的。因此,这种 DMA 缓冲器中的电子篱笆就能够访问那个协处理器设备环境的地址空间中的虚拟地址而不是物理地址。不用说,是通过与由该协处理器访问的任意其他虚拟地址相同的存储器确认装置来限制对这样的虚拟地址的访问。

[0286] 有特权的电子篱笆是能够只是插入到 (并且只是可见的) 以内核模式创建的 DMA 缓冲器中。这样的电子篱笆能够访问存储器中的物理地址从而增强系统的安全性。如果该电子篱笆目标地址在协处理器设备环境的地址空间中是可见的,那么恶意的应用程序就会在那个存储器的地址上进行图形操作,这样就不顾内核模式代码期望接收什么样的内容。解决潜在的安全问题的另一种方法是在 PTE 中具有一个有特权的位来表示虚拟地址能否从没有特权的 DMA 缓冲器中 进行存取。然而,以上的第一种方法看作是对早先的硬件产生的简化。

[0287] 注意,有特权的 DMA 缓冲器可包含常规的和有特权的电子篱笆两者。然而,当有特权的 DMA 缓冲器包含一个常规的电子篱笆时,该内核部件就知道所产生的 DMA 缓冲器可能从来都是不可见的,其中该 DMA 缓冲器是该电子篱笆插入的 DMA 缓冲器。

[0288] 为了将刷新所需的内部缓冲器的数量减到最小, IHV 可以决定支持额外类型的电子篱笆。以下类型的电子篱笆是为了此目的可以支持的电子篱笆的例子(注意,对于任何类型,都应该是支持有特权的和没有特权的两者的) :

[0289] 1) 写电子篱笆

[0290] 一个写电子篱笆可以是之前所述的类型的电子篱笆,并且只是所需类型的电子篱笆。一个写电子篱笆保证所有的存储器在处理电子篱笆的指令之前都已经写过了,这些存储器是全局可见的(即它们已经刷到高速缓存器的外面,并且已经从存储器控制器处接收到了通知)。

[0291] 2) 读电子篱笆

[0292] 读电子篱笆是一种类似于写电子篱笆的更不重要的类型的电子篱笆。读电子篱笆保证所有的存储器在完成该电子篱笆之前都为再现操作而进行了读取,但某些读取可能还没有完成。如果支持读电子篱笆,那么调度程序就会用它们控制非再现目标位置的使用寿命。

[0293] 3) 管道顶部电子篱笆

[0294] 管道顶部电子篱笆是一种非常不重要的电子篱笆。对管道顶部电子篱笆的支持是可选的。管道顶部电子篱笆只保证 DMA 缓冲器中的电子篱笆指令之前的最后类型由协处理器来读取(但也不是必然要处理)。在已经处理了那个电子篱笆之后(因为那个 DMA 缓冲器的内容可能不再有效了),电子篱笆该协处理器也许不能重复读取位于管道顶部电子篱笆之前的 DMA 缓冲器的任何部分。如果支持,那么这种类型的电子篱笆就供该调度程序使用,从而控制 DMA 缓冲器的使用寿命。

[0295] 陷阱。一个陷阱可以在本发明的不同实施例中实现。陷阱可以是插入到 DMA 缓冲器中的一个指令,当协处理器处理它的时候,能够产生 CPU 的中断。在协处理器能够中断 CPU 之前,它是可以确信来自该陷阱指令之前的原语的所有元素都已经收回并且适当地写入了存储器中(一个可以包括来自电子篱笆指令的存储器写入操作)。注意,这并不意味着协处理器需要阻塞整个流水线。在协处理器正等待要收回陷阱之前的指令的最后元素的时候,可以执行遵循该陷阱指令的原语。

[0296] 该陷阱指令不必是有特权的指令,并且能够插入到任何 DMA 缓冲器中,包括那些由用户模式驱动器直接创建的 DMA 缓冲器。

[0297] 能够 / 不能进行设备环境转换。对于支持下三角形中断的硬件,能够提供一指令来使其能够或者使其不能进行设备环境转换。当不能进行设备环境转换时,协处理器通常不会转换而离开当前的协处理器设备环境。尽管如果 CPU 提供了新的运行列表就可能要求协处理器更新其当前运行列表信息,但是协处理器可以推迟设备环境转换到那个新的运行列表,直到重新能够进行设备环境转换。该 OS 可以缺陷下面的规则在不能进行设备环境转换时仍然是真的:

[0298] 1) 将只处理有特权的 DMA 缓冲器。

[0299] 2) 将没有设备环境转换指令出现在该 DMA 流中。

[0300] 3) 该 DMA 缓冲器将不运行指令。

[0301] 4) 没有页面故障发生 (如果支持页面级故障的话)。

[0302] 在许多计算机系统中,没有能力和有能力进行设备环境转换是可以指出现在有特权的 DMA 缓冲器中的有特权的指令。用于这些指令的使用情形是允许调度程序调用出现在屏幕 (即显示 blit) 上的操作而不是其要中断的可能性。在这样的操作中进行中断会导致在一个重要的时间周期中产生在该屏幕上可见的非自然信号。

[0303] 注意,如果协处理器在 DMA 缓冲器中遭遇不可预见的误差,它就会进行设备环境转换而离开这个 DMA 缓冲器,尽管不能进行设备环境转换。因为只有以内核模式创建的 DMA 缓冲器可以包含不可中断的部分,所以不可预见的误差就可能是驱动器程序错误或者硬件程序错误的结果。如果协处理器不在那些情形下进行设备环境转换,那么显示监视器就会进行挂起并且为了恢复系统而复位该协处理器。

[0304] 可选的控制指令。尽管调度程序能够采用上述简单的控制指令创建高级的同步原语,但该结果可以变得更加有效。在许多计算机系统中,协处理器设备环境都是在它能够拥有同步目标程序的所有权之前由 CPU 进行中断。如果正拥有同步目标程序并且以高频释放,那么这就变成有问题的了。为了具有更有效的同步原语,调度程序可以从协处理器中接收特定的指令。尤其是,协处理器可以构造为在适当的时候发出一个“等待”指令和一个“信号”指令。

[0305] 把一个等待指令插入到 DMA 缓冲器中,从而通知协处理器它可以检查特定计数器的值。如果该计数器是非零的,则协处理器就能够减少该计数器并且继续运行当前的协处理器设备环境。如果该计数器是零,那么该协处理器就能够在该等待指令之前复位当前协处理器设备环境的指令指针并且转换到该运行列表的下一个设备环境。当一个协处理器设备环境需要在一个等待指令上阻塞并且之后再重新调度时,协处理器能够重新执行等待指令,因为可能还是不能满足该等待条件。

[0306] 该等待指令需要只具有一个参数:虚拟地址,规定要比较 / 减少的存储器位置。该计数器可能至少是 32 位,并且可能是任何有效的虚拟地址。在一个优选实施例中,该等待指令可以是不可中断的;也就是说,如果把一个新的运行列表给了协处理器,它就能够在该等待指令之前或者运行它之后转换到该新的运行列表。等待指令可以插入到限定的和有特权的两种 DMA 缓冲器中。

[0307] 一个信号指令可以插入到 DMA 流中,从而通知协处理器它能够更新计数器的值。然后,该协处理器可以把计数器的值加 1。该协处理器在加法过程中忽略了可能发生的溢出。可替换地,该协处理器能够因该流中有错误而报告溢出,从而帮助跟踪软件调试故障。

[0308] 该信号指令只需要具有一个参数:应当被更新的计数器的虚拟地址。该计数器大小可以做得与该等待指令的计数器大小相匹配,并且在一个优选实施例中,至少是 32 位。信号指令可以插入到限定的和有特权的两种 DMA 缓冲器中。

[0309] 触发

[0310] 为了允许所有的屏幕应用程序无缝地运行而不用在流水线中冒泡,该协处理器可能提供一种指令排队查询一个触发 (即,显示的基址的改变)。该显示表面通常是从物理存储器中连续地进行定位并且由 CRTC 使用物理地址而不是虚拟地址来进行访问。因此,该触发指令能够用于把 CRTC 编程为一个要显示的新的物理地址。因为这是一个物理地址而非虚拟地址,所以拙劣的应用程序就可能偷偷地把 CRTC 编程为显示属于另一个应用程

序或用户的辅助存储器的一部分（其中包含机密）。为此，一旦该目标是确定的，通过确信它是仅由内核模式驱动器插入到 DMA 流中的有特权的指令，该触发指令就能够实现保护大多数计算机系统的安全性。

[0311] 在结合触发功能来使用的本发明的多种优选实施例中，能够支持至少两种类型的触发：一种立即触发和一种与显示刷新同步的触发。当协处理器处理一个立即触发时，它能够立即更新显示的基地址，尽管这样做可能会引起可视图像撕裂。当协处理器处理一个同步触发时，它能够锁存一个新的基地址，但会使其更新延迟直到下一个纵向的同步周期。如果一个以上的同步触发由协处理器在纵向同步周期之间进行处理，那么该协处理器就只会锁存最后的一个并且忽略之前的。

[0312] 当处理一个同步触发时，可以构造多种实施例以便协处理器可以不用阻塞该图形流水线。该 OS 将确信并没有对该环形缓冲器中的任何再现命令进行排队，该环缓冲器就可能会接近当前可见的表面。在此要注意，按照下面将要进一步解释的“最佳触发”的情况下，也可以构造其它实施例而不需要这些要求。

[0313] 为了确定哪个表面是当前可见的，该驱动器也许首先得确定什么时候一个特定的排列触发已经发生并且把该事件通知调度程序，即在显示基地址已经改变了以后通知该调度程序。对于一个立即触发，当发生触发的时候进行确定是很容易的，因为从 DMA 流中读取该触发指令可以看作是与更新显示表面相同的事件。一个电子篱笆和一个中断都可以插入到 DMA 流中的触发指令的后面，从而通知该调度程序已经读取了一个特定的触发。

[0314] 在该同步触发的情况下，确定哪个表面是当前可见是更困难的。协处理器将首先从 DMA 流中读取该触发指令，但之后会在下一个 vsync 中断处更新该显示表面。为了取消在那个过程中阻塞协处理器的需要，可以在该显示表面改变变得有效的时候提供一种装置来通知调度程序。

[0315] 现有多种方式用来设计用于这种通知的装置以供结合本发明时使用。图 22 说明了一种可能比较简单的方法。图 22 提供了一种对协处理器查询有关当前显示表面的方法。在所示实施例中，这种功能可以按照 MMIO 寄存器所提供的那样来进行构想。图 22 的系统设计为当该寄存器读取真实的显示表面而不是最后的“锁定的显示表面”时将变得更加可靠。随着协处理器处理另一个排列的触发，对最后的锁存显示表面进行排列会导致一种竞态条件，这会导致屏幕上的图像撕裂。一个触发指令可以利用一种适当的技术来产生。用于与本发明相兼容的唯一的常用要求是所实现的方法应当确信并不去确认一个触发直到它是有效的。

[0316] 对触发进行排列。为了提供最高的性能，可以修改改进的调度模型从而在拥有监视器的应用程序的再现流中对触发操作进行排列。当进行 n 缓冲时，该调度程序可以允许 DMA 缓冲器重要排列的高达 n-1 个触发器，并且在第 n 个触发器将要插入的时候可以进行锁定。

[0317] 这就意味着在双缓冲过程中，该调度程序可以允许该应用程序对一个触发进行排列，并且在协处理器完成了对当前帧的再现并且处理 / 通知给了那个触发的时候，让它继续准备 DMA 缓冲器进行后面的帧。它还意味着随着 DMA 缓冲器准备后面的帧而完成该应用程序并且提交了第二个触发的时候，就可能将它锁定，直到把第一个触发通知给协处理器。

[0318] 当该调度程序使用立即触发的时候，对触发进行排列的机械结构按照以上所述进行工作。然而，当使用同步触发时，调度程序还会特别注意晚于触发 n-1 的排列的 DMA 缓冲

器。实际上，晚于那个触发的 DMA 缓冲器通常会再现为当前可见的表面。在大多系统中，这些 DMA 缓冲器并不进行处理直到当前排列的触发的数量下降到  $n-2$  以下，这是非常可取的。

[0319] 解决这个问题最简单的方法就是只允许  $n-2$  个触发进行排列而不是  $n-1$  个。然而，这种方法也就意味着在该双缓冲的情况下我们不能排列任意的触发，所以我们可能需要在完成每一帧之后锁定该应用程序，直到处理该相应的触发为止。

[0320] 图 23 中说明了在这种设置中的优选方法。如图所示，是允许  $n-1$  个触发的排列的。为了防止在触发  $n-1$  排列之后的 DMA 缓冲器运行，该调度程序可能累加用于那个协处理器设备环境的虚拟环形缓冲器中的那些 DMA 缓冲器。该调度程序可以等待，直到当前排列的触发的数量回落到  $n-2$ ，以便将它们提交给那个协处理器设备环境的正确的环。

[0321] 当一次运行多个应用程序时，该协处理器也许不得不如图 23 中所示的那样阻塞。尽管协处理器通常会阻塞处理来自一个特定的协处理器的 DMA 缓冲器，但该调度程序可以调度其它的协处理器设备环境来运行，有效地保持与该协处理器一样忙。然而，当运行一个单独的应用程序时，例如，当玩一个全屏游戏时，在那些时间间隔里该协处理器可能会阻塞。接下来的部分描述了一种装置，该装置是，如果支持的话，调度程序用来减少阻塞时间。

[0322] 最佳触发。试着优化全屏应用程序，就可能将协处理器用于阻塞的时间最小化。参见图 23，观察到该协处理器会阻塞至少是出于两种原因：第一，因为完成了这帧但系统正等待一个 vsync 来进行触发，以及第二，因为完成了该触发，但系统正等待一个中断来告知 CPU。

[0323] 为了减少出于第一种原因的阻塞，可能要把更多的缓冲器添加到触发链上。从双缓冲器到三缓冲器的转变，将极大地减少这种阻塞。这样做并不总是在驱动器的控制下的，并且可能会导致不合理的存储器消耗。

[0324] 为了减少出于第二种原因的阻塞，可能要添加一个协处理器装置从而完全地消除这种阻塞的需要。该协处理器能够提供一种等待触发指令，该指令会阻塞该协处理器直到已经处理了之前排列的触发。当支持这种指令时，该调度程序可以用它来为全屏应用程序排列触发，并且 CPU 不必在每个触发之后重新启动 DMA 流。

### [0325] 高级同步目标程序

[0326] 利用之前规定的控制指令，该调度程序就能创建高级的同步目标程序，例如关键部分和互斥。一旦满足等待条件，通过保护 DMA 缓冲器的一部分不去运行，该调度程序就能够实现这样的同步原语，直到它被 CPU 明确地重新调度。等待一个目标程序就可以由该调度程序来实现来作为一个电子篱笆。逻辑上遵循该电子篱笆的 DMA 缓冲器可以由调度程序进行排列，但并不提交到协处理器设备环境的环中，直到满足该等待条件。一旦在等待一个目标程序，那么就可以由该调度程序把一个协处理器设备环境移动到那个特定目标程序的等待列表上，直到它发出信号。目标程序可以通过插入协处理器设备环境 DMA 流中的一个后面是中断命令的电子篱笆来发出信号。当接收这样一个中断时，调度程序可以识别哪个目标程序正在发出信号，并且然后确定是否由任何等待协处理器设备环境应当放回准备好的排列中。当把一个协处理器设备环境放回该准备好的排列中时，该调度程序就插入从该环中阻止下来的该 DMA 缓冲器。

[0327] 例如，设想本发明的一个实施例，其中一个应用程序具有一个生产者和消费者之间共享的表面，并且该应用程序需要同步访问这些资源，以便在再现的时候，消费者总是可以使用有效的内容。图 24 说明了一种同步这种情形的可能的方法。

[0328] 转到图 24, 在调度程序的一端, 可以, 例如, 通过下面的内核形实转换程序, 实现该同步, 这可以通过其任意组合或与其它动作的组合来实现:

[0329] 1) CreateSynchronizationObject 创建同步目标程序: 为同步目标程序创建一个内核轨迹结构。把一个编号返回给用户模式以便可以在其后的等待 / 释放 / 删除调用中使用。

[0330] 2) DeleteSynchronizationObject 删除同步目标程序: 销毁之前创建的目标程序。

[0331] 3) WaitOnSingleObject/WaitOnMultipleObject 等待单个目标程序 / 等待多个目标程序: 把一个等待同步事件插入到当前协处理器设备环境的 DMA 流中。采用对正在等待的目标程序的索引, 把该事件插入到调度程序事件历史。

[0332] 4) ReleaseObject/SignalObject 释放目标程序 / 向目标程序发信号: 把一个信号同步事件插入到当前协处理器设备环境 (电子篱笆 / 中断) 的 DMA 流中。采用对正在释放或发出信号的目标程序的索引, 把该事件插入到调度程序事件历史。

[0333] 把图 24 的说明应用到一个互斥上, 一旦协处理器处理该 DMA 流中的一个同步事件, 该调度程序就能执行下面的动作, 这还可以通过其任意组合或与其它动作的组合来实现:

[0334] 1) 有关等待 (on a wait): 检查该互斥的状态。如果当前不是采用互斥, 则采用该互斥并且把协处理器线程放回到该调度程序的准备好的列表中。如果已经采用了该互斥, 则把该协处理器线程放到用于该互斥的准备好的序列中。

[0335] 2) 有关信号 (on a signal): 检查某些其它的协处理器线程是否正在等待该互斥。如果某些其它线程正在等, 则把正在等的第一线程放到该列表中, 并且把它放回到调度程序的准备好的列表中。如果没有线程正在等, 则把该互斥放回到非采用状态。

[0336] 利用这种装置, 就能创建该调度程序。例如, 设想能够由该调度程序创建的下面类型的同步原语:

[0337] 互斥: 一次只有一个协处理器线程能够访问共享资源。

[0338] 信号: 相同的时间里, 特定数量的协处理器线程都能够访问共享资源。

[0339] 通知事件: 一定数量的协处理器线程可能等待来自另一个协处理器线程的信号。

[0340] 在某些情形下, 一个应用程序可以构造为当协处理器完成对再现指令的处理时请求通知。为了支持这个, 调度程序可以允许驱动器请求它正在提交的用于 DMA 的通知。然后, 一旦协处理器完成提交 DMA 缓冲器, 该驱动器可以在提请的时间里指定一个能够发出信号的 CPU 同步事件。该调度程序可以把所给的 DMA 缓冲器插入到所给的协处理器设备环境的环中, 并且然后把一个用户模式协处理器事件通知添加到该环 (中断后面的一个电子篱笆) 上。当协处理器事件已经由协处理器进行了处理时, 该调度程序就能发出有关的 CPU 同步事件的信号。

[0341] 调度程序事件历史缓冲器

[0342] 该调度程序可以使用用于多种目的的上述同步装置。因为中断并不阻塞该协处理器, 所以 CPU 只需要了解通知的一部分, 并且因此某些通知就能够压缩在一起。为了适当地响应 DMA 缓冲器中的每个通知, 调度程序会保存事件的历史, 这些事件是连同处理那些事件所需的任何参数一起插入的。

[0343] 简单来讲,该事件历史缓冲器可以是事件信息结构的一个通用协处理器设备环境数组,该事件信息结构跟踪要求调度程序处理的并且已经插入到那个设备环境的 DMA 流中的各个事件。注意,调度程序电子篱笆是调度程序用来与一个事件同步的电子篱笆。每个协处理器设备环境都可能有一种电子篱笆,并且为了保证安全性,该电子篱笆可以做成只允许通过一个有特权的指令来更新。在任何情况下,这样一个事件都可以按照一个中断指令后面的电子篱笆指令插入到 DMA 流中。

[0344] 在各电子篱笆中断上,调度程序可以首先确定当前的电子篱笆,然后通过该事件历史缓冲器去确定发生了哪个事件。这种确定可以基于相关的电子篱笆来作出。调度程序可以着手处理该电子篱笆中断。图 25 说明了该事件历史缓冲器的多种实施例。

[0345] 任何数量的事件都可以得以支持。下面的表个描述了一些当前支持的事件,但并不意味着是对可能支持的事件的数量或类型的限定。

[0346]	事件类型	说明以及参数
[0347]	DMA 缓冲器的结束	将这个事件插入到 DMA 缓冲器的尾部。当调度程序处理这个事件时,相关的 DMA 缓冲器就放回到 DMA 缓冲器池中以用于那个处理。
[0348]		参数 :对那个需要被释放到该池中的 DMA 缓冲器的处理。
[0349]		
[0350]	等待同步	当协处理器线程需要检查一个事件的状态并且可能要等它的时候,就插入这个事件。当调度程序处理这个事件时,它检查是否已经满足了等待条件,并且如果是,则重新调度刚刚停止的该协处理器线程。如果不满足该等待条件,则把该协处理器线程放在等待状态并且添加到该同步目标程序的等待对列上。
[0351]		参数 :等待的目标程序的处理。
[0352]	目标程序	当协处理器线程需要发出一个通知目标程序或者释放一个同步目标程序的信号时插入这个事件。当调度程序处理这个事件时,它改变该目标程序的状态,并且可能唤醒正在等待该事件的某些协处理器线程。
[0353]		
[0354]		
[0355]		
[0356]		
[0357]		
[0358]	信号同步	参数 :释放的目标程序的处理。
[0359]	目标程序	当用户模式驱动器请求再现完成的通知时插入这个事件。当调度程序处理这个事件时,它向有关事件发出信号。
[0360]		
[0361]		
[0362]		
[0363]	用户模式	
[0364]	事件通知	
[0365]		参数 :发出信号的事件。
[0366]	可编程 PCI 缝隙	
[0367]	当今的协处理器公开了非常接近于 PCI 规格所允许的限制的 PCI 缝隙。将来产生的协处理器将通过缝隙在主板上具有比已有的更多的辅助存储器。因此,在将来,我们不能假设所有的辅助存储器在相同的时间上通过 PCI 缝隙都是可见的。	
[0368]	现有多种方法来使得这种限定得以解决。一种用于能够支持通用协处理器设备环境虚拟地址空间的改进的调度模型的优选方法是在 4KB 的间隔尺寸上使用能够重定向到辅助存储器的任何位置的 PCI 缝隙。这在图 26 中进行了描述。	
[0369]	如图 26 中所述,该 PCI 缝隙页表可以是独立于协处理器页表的。当协处理器从设	

备环境向设备环境进行自身转换的时候,可能有多个CPU过程在运行和访问该PCI缝隙的部分。用于该PCI缝隙的页表在所有的协处理器设备环境中是共享资源,并且能够从辅助存储器中进行分配。该驱动器可以提供一种映射/非映射DDI,从而允许该辅助存储器管理器VidMm管理正在运行的应用程序中的PCI缝隙地址空间。协处理器可以利用一个物理地址来找到用于该PCI缝隙的页表。

[0370] 注意,该PCI缝隙可以构造为只把该地址空间重定向到本地的辅助存储器。它并不需要把该地址重定向到系统存储器,因为VidMm总是直接映射到系统存储器,而不通过那个缝隙。

[0371] 页面能级故障

[0372] 当之前所述的表面能级故障通常在大多数情况下都可以很好的运行时,就存在它可以改进的情形。例如,利用表面能级故障,某些利用大量数据的应用程序也许就不能一次获得存储器中的整个数据组,并且因此,就不可以很好地运行。解决这个的一种方法是在改进的模型中由一个页面能级故障装置来实现。

[0373] 采用页面能级故障,该模块的工作是类似于之前的部分中所描述的那样的。主要的不同点是在页面故障报告的方式和VidMm处理的方式上。尽管表面能级故障可能要求协处理器指定它做进一步处理所需的(为了消除无限循环,该无限循环中页入一个资源就意味着要收回另一个所需的资源)资源的整个列表,页面能级故障并不需要协处理器公开虚拟地址的列表。对于页面能级故障,协处理器只需要报告发生故障的虚拟地址。VidMm能够找出这个地址是哪个位置的一部分,并且决定是否只有这个特定的页面需要驻留或者是否要求某种预取。当单独的元素要求多个页面时,可能会对那个单独的元素产生多个故障。还可能是,当带进另一个页面时,那个元素所需的页面会被收回。然而,只要正在工作的那组应用程序充分地大于一个元素所需的最大数量的页面,通过页面故障循环的可能性就很小。

[0374] 最后,应当可以理解,在此所述的多种技术都可以用硬件或软件或,适当地,两者的结合来实现。因此,本发明的方法和装置或者其某些方面或部分都可以采用体现在有形介质上的程序代码(即指令)的形式,其中,当把程序代码装入一个计算机之类的机器中并且由该机器来运行时,该机器就变成一个用于实现本发明的装置。在可编程计算机上运行程序代码的情况下,该计算设备通常包括一个处理器、一个处理器可读的存储介质(包括易失性和非易失性存储器和/或存储元件)、至少一个输入设备以及至少一个输出设备。可以实现或利用本发明的用户接口技术的一个或多个程序,例如通过处理API的数据的使用、可重复使用控制等,都能用面向高级程序或目标程序的编程语言来实现从而与计算机系统进行通信。然而,如果需要的话,这些程序都可能是用汇编语言或机器语言来实现的。在任何情况下,这些语言都可以是一种编译或解释语言,并且是与硬件实现相结合的。

[0375] 尽管示例性实施例涉及把本发明用于单机计算机系统的设备环境中,但本发明并不限于此,而是还可以在任何计算机环境中得以实现,例如网络或者分布式计算机环境。再进一步,本发明可以在多个处理芯片或设备中或者在多个处理芯片或设备上面实现,并且存储可以类似于多个设备上的作用。这样的设备可能包括个人计算机、网络服务器、手提式设备、超级计算机或者集成到其它汽车或飞机之类的系统中的计算机。因此,本发明不应该限于任何单一的实施例,而相反的应该按照附加的权利要求解释其宽度和范围。

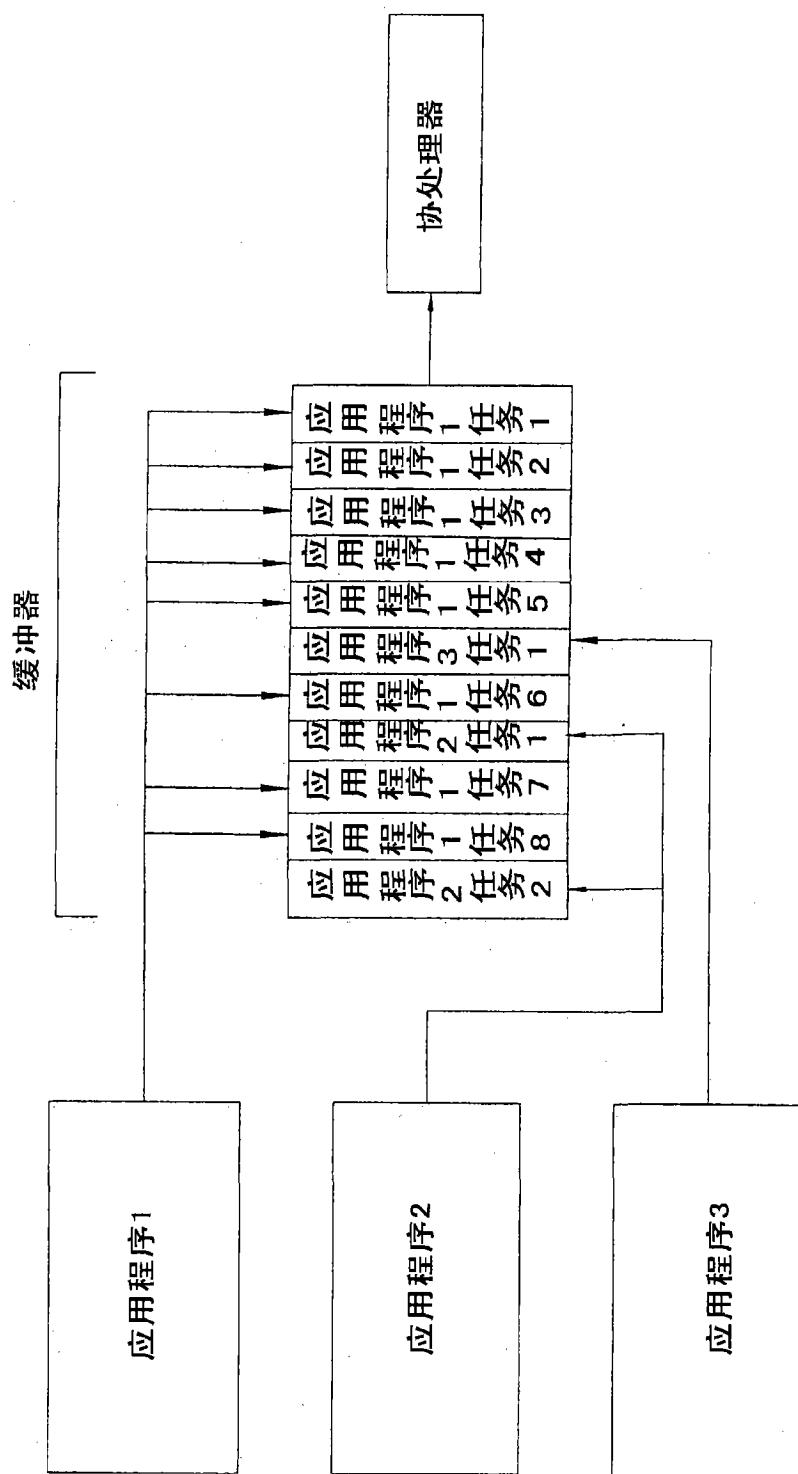


图 1

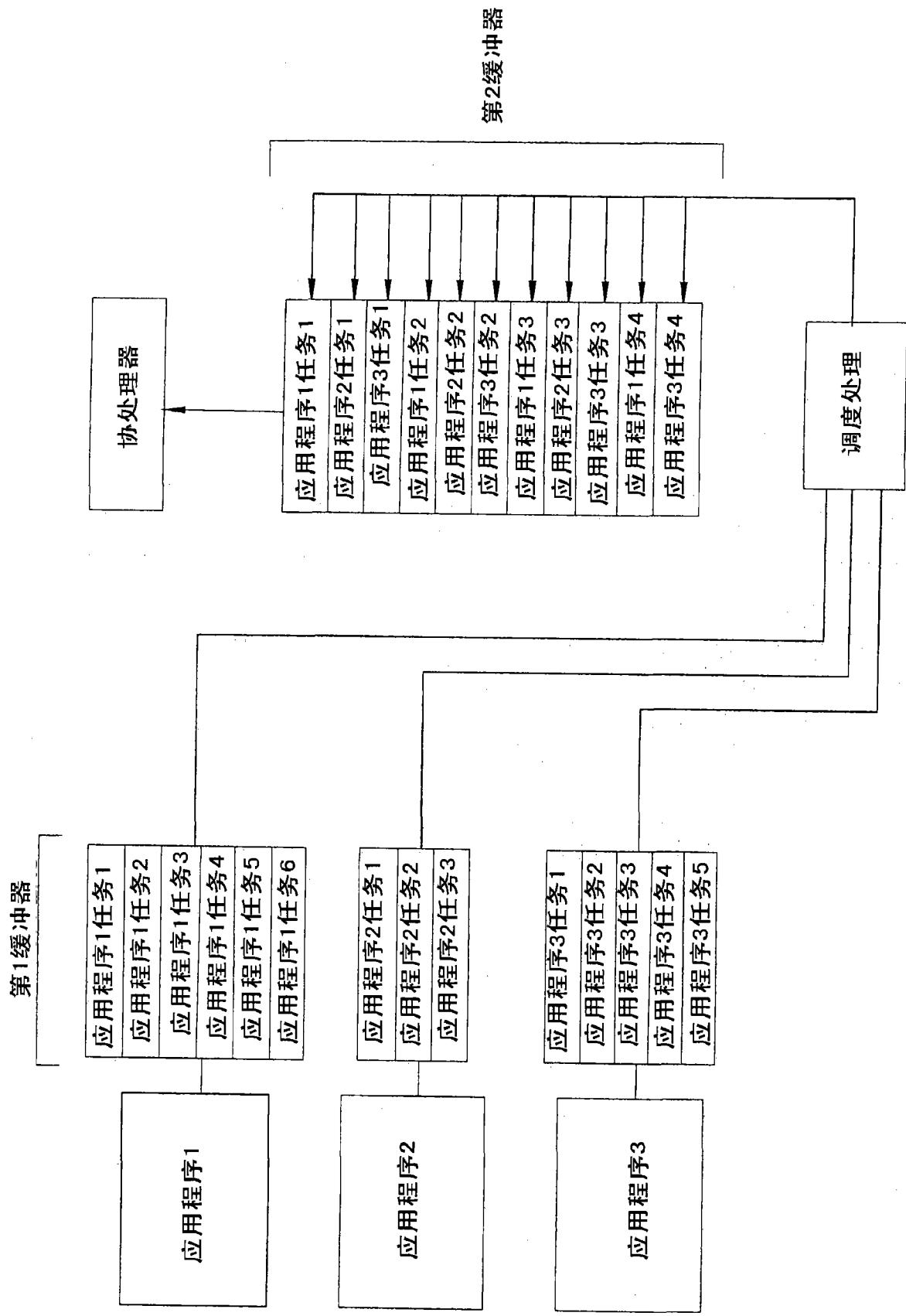


图 2

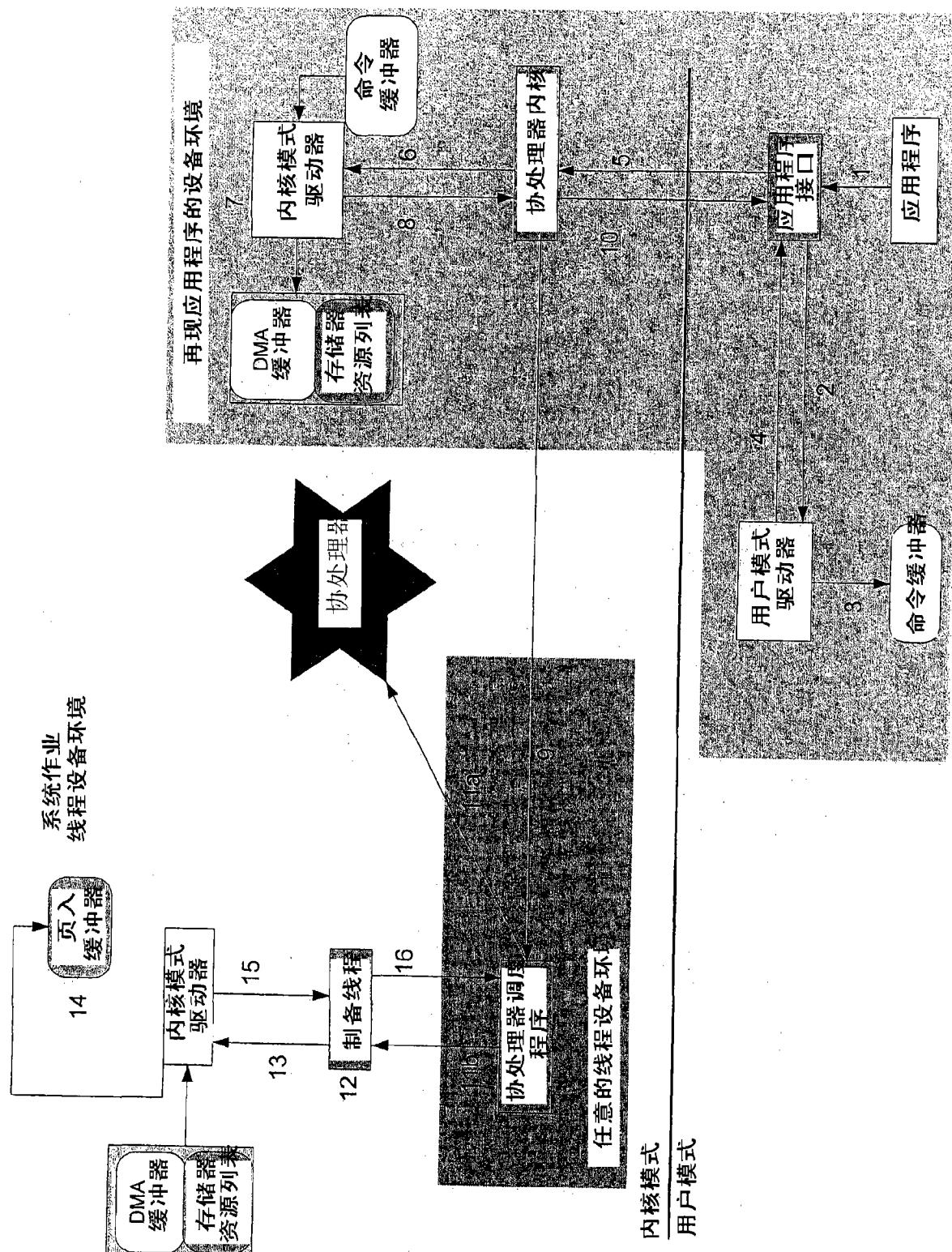
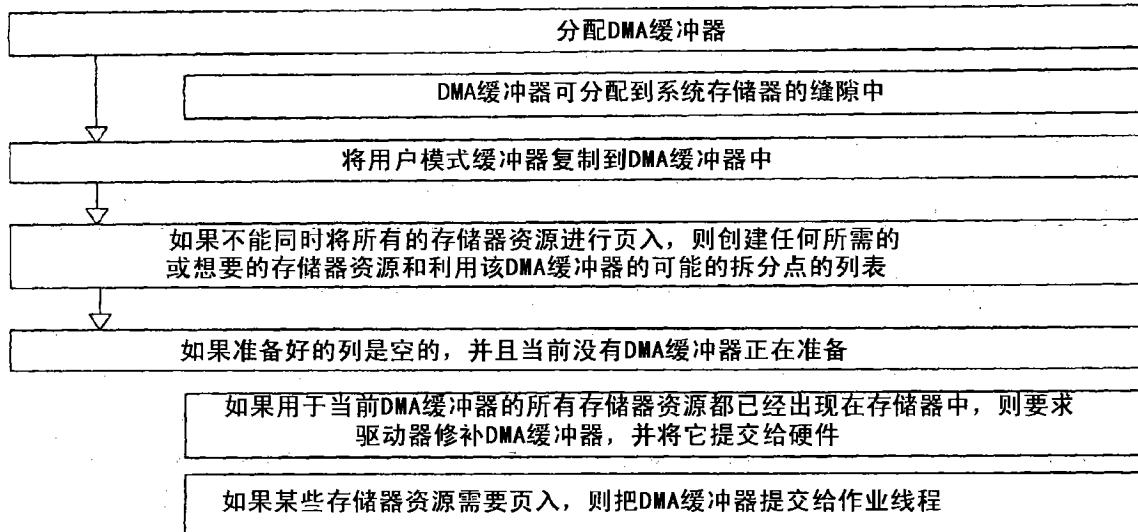


图 3

## 示例的算法

过程A: 提交(irql消极地, 再现线程设备环境)



过程B: 量程届满(irql设备, 任何线程设备环境)

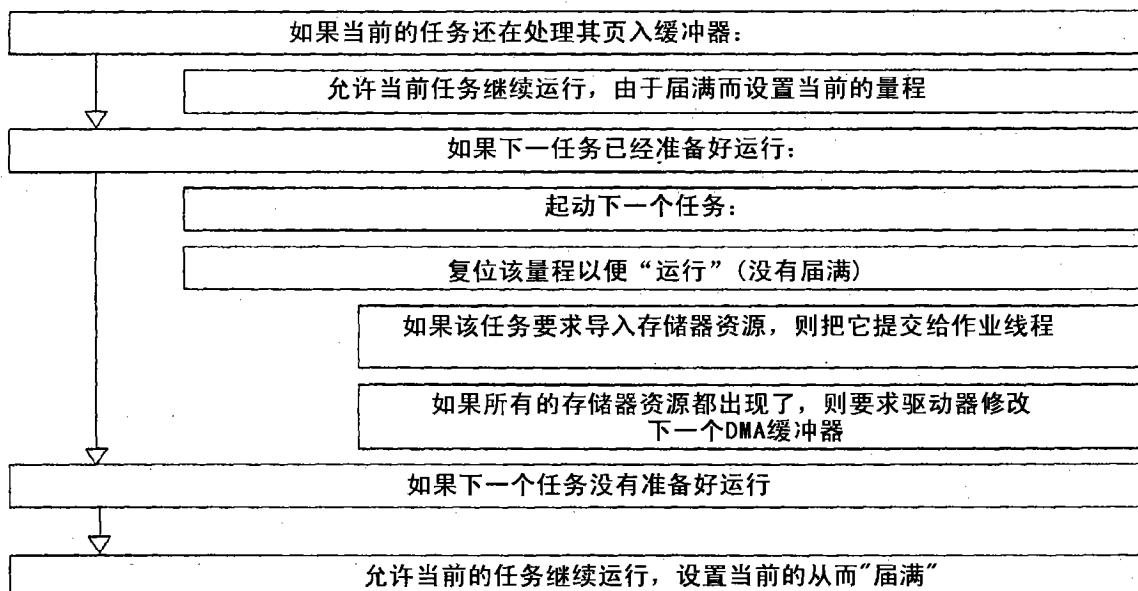


图 4(A)

## 示例的算法

过程C:任务完成(irq1设备, 任何线程设备环境)

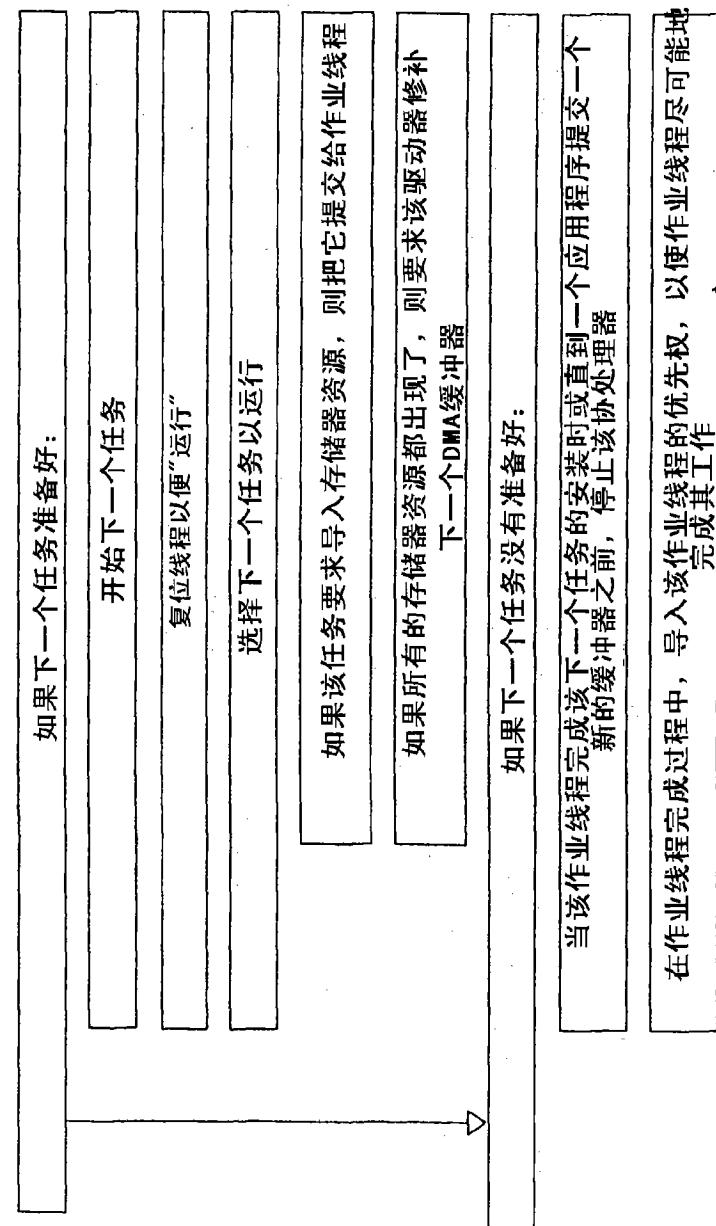
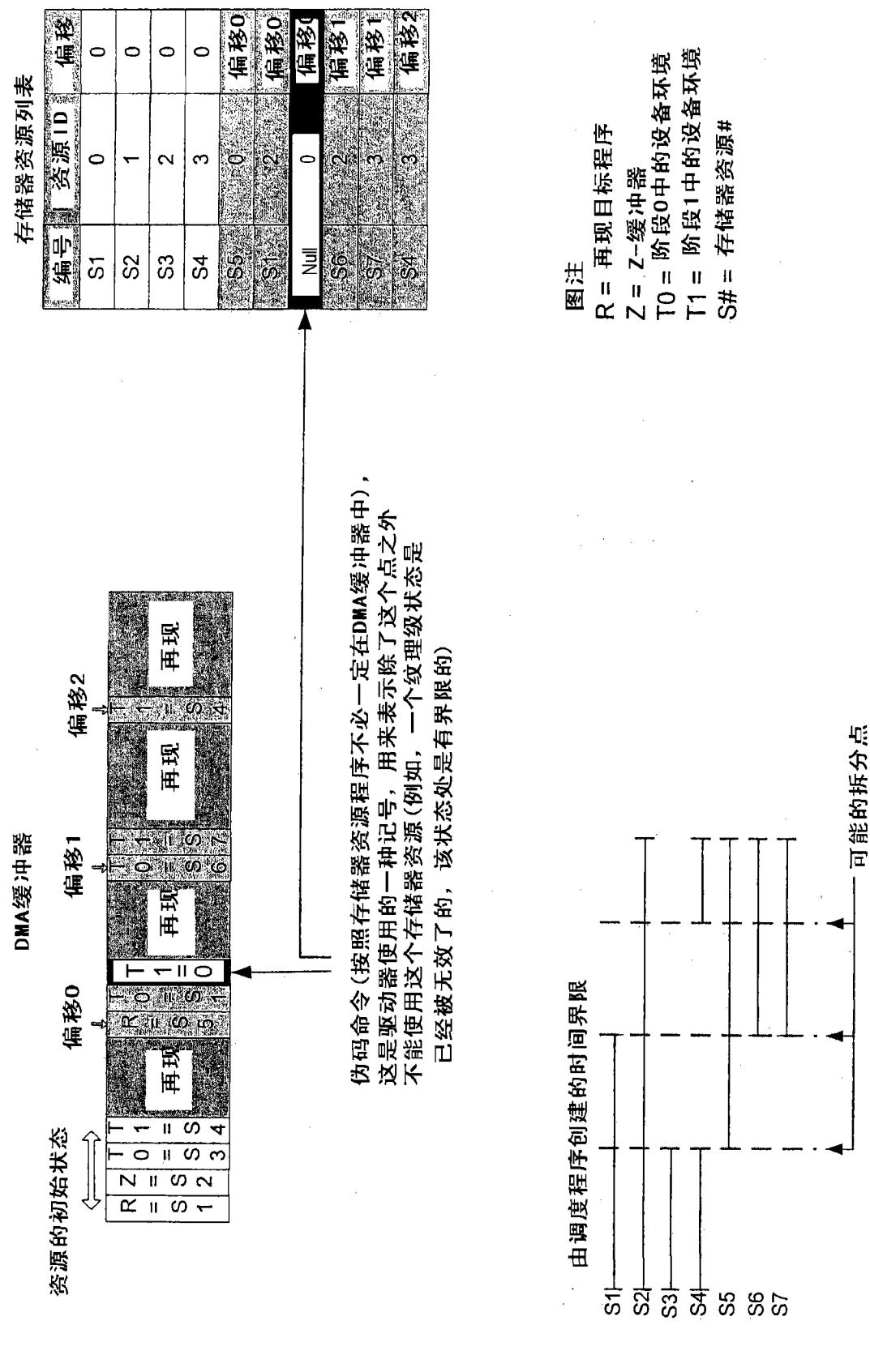


图 4(B)



图注 R = 再现目标程序  
 Z = Z-缓冲器  
 $T_0$  = 阶段0中的设备  
 $T_1$  = 阶段1中的设备  
 S# = 存储器资源#  
 S# = 存储器资源#

伪码命令(按照存储器资源程序不必一定在DMA缓冲器中),这是驱动器使用的一种记号,用来表示除了这个点之外不能使用这个存储器资源(例如,一个纹理级状态是已经被无效了的,该状态处是有界限的)

由调度程序创建的时间界限

S1  
 S2  
 S3  
 S4  
 S5  
 S6  
 S7

## 示例的算法

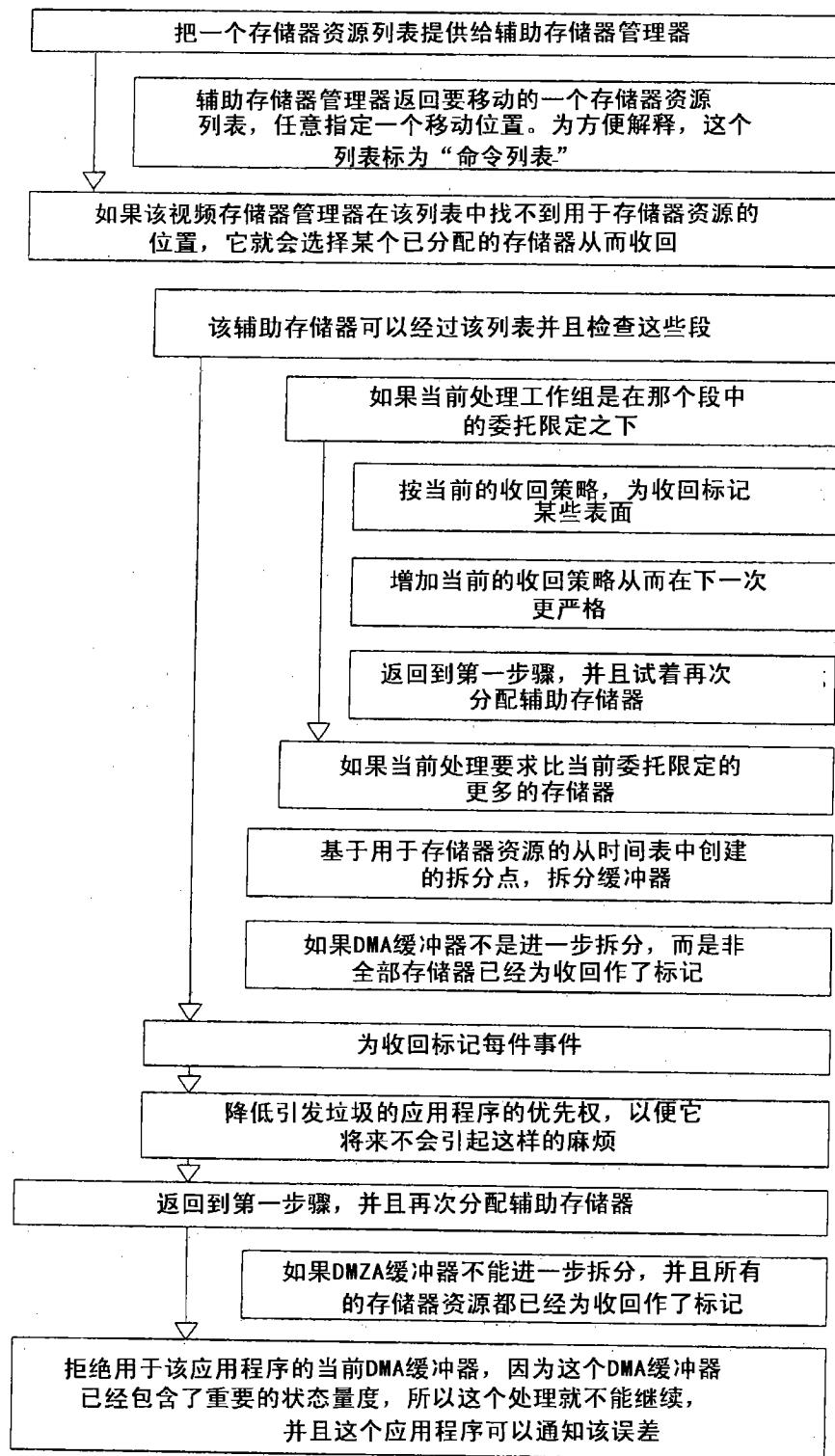


图 6

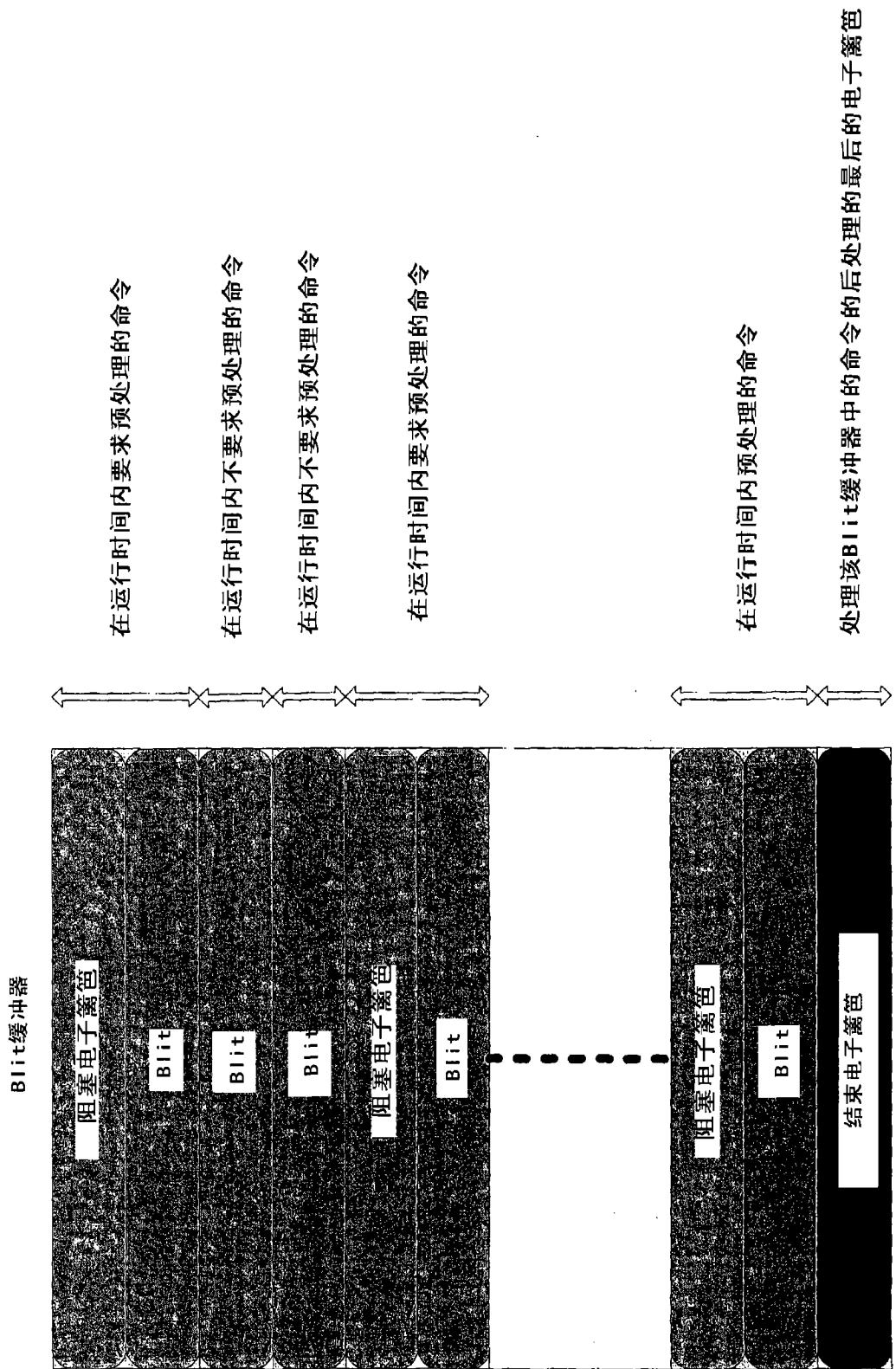


图 7

## 示例性算法

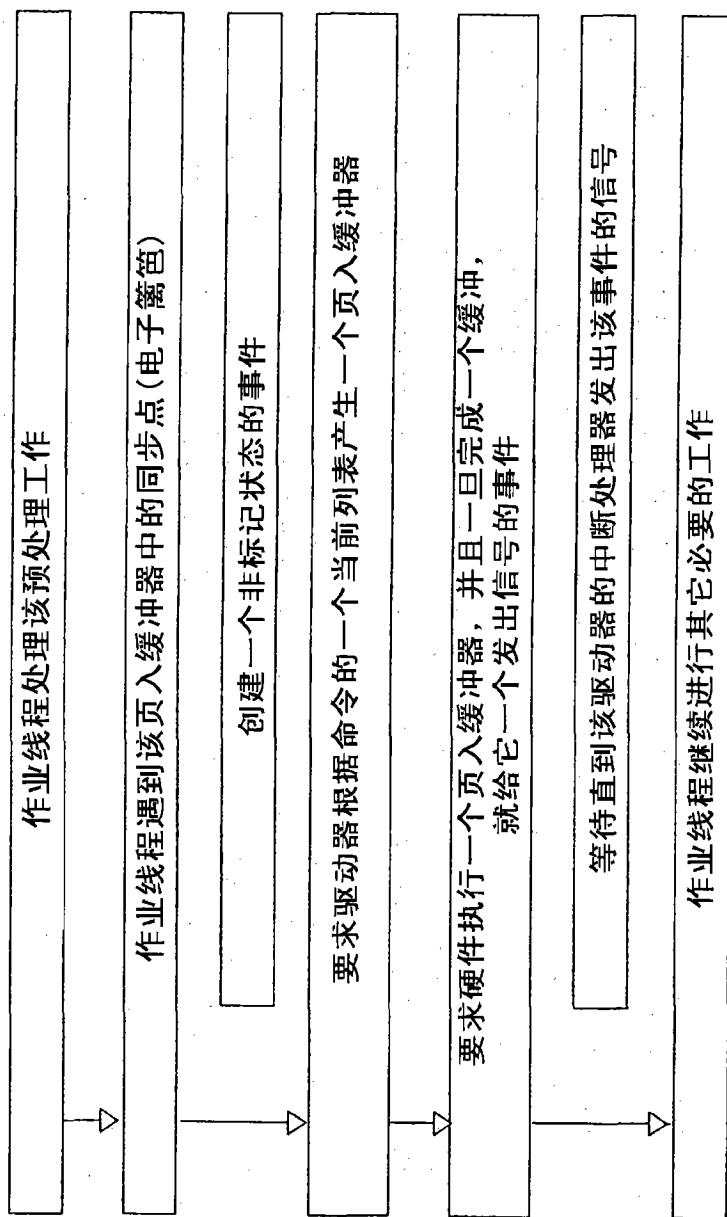


图 8

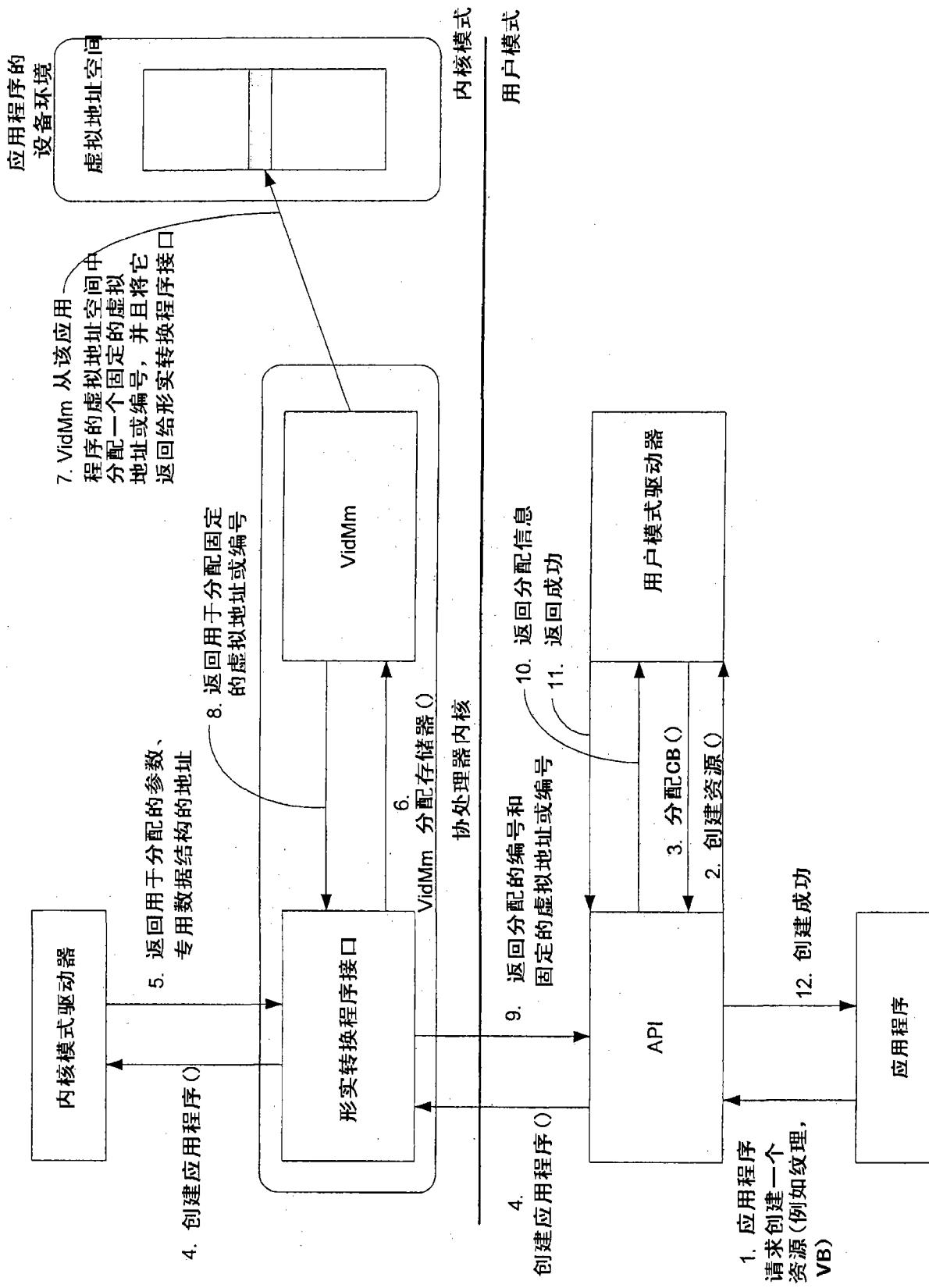


图 9

准备队列  
流水线中的任务

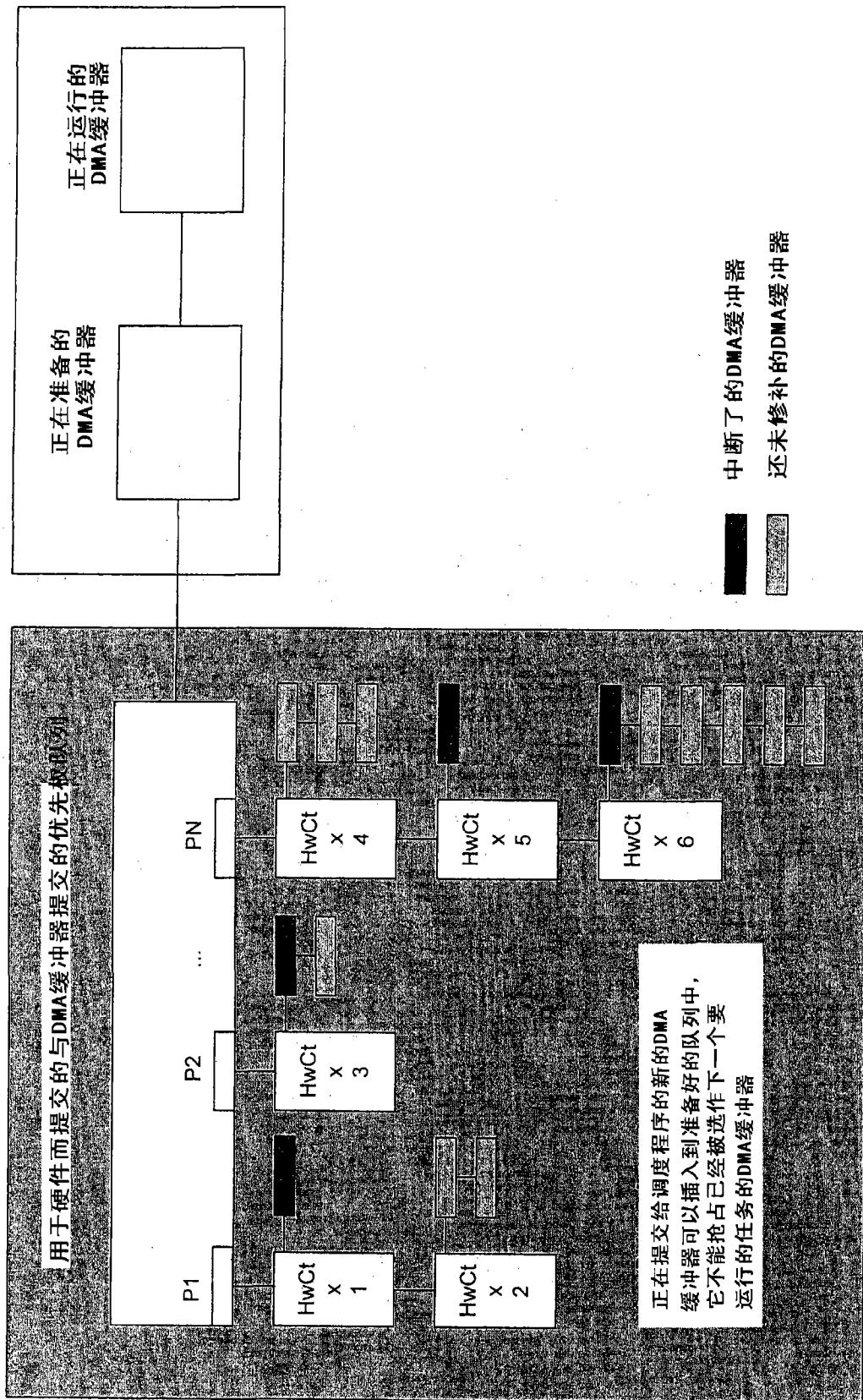


图 10

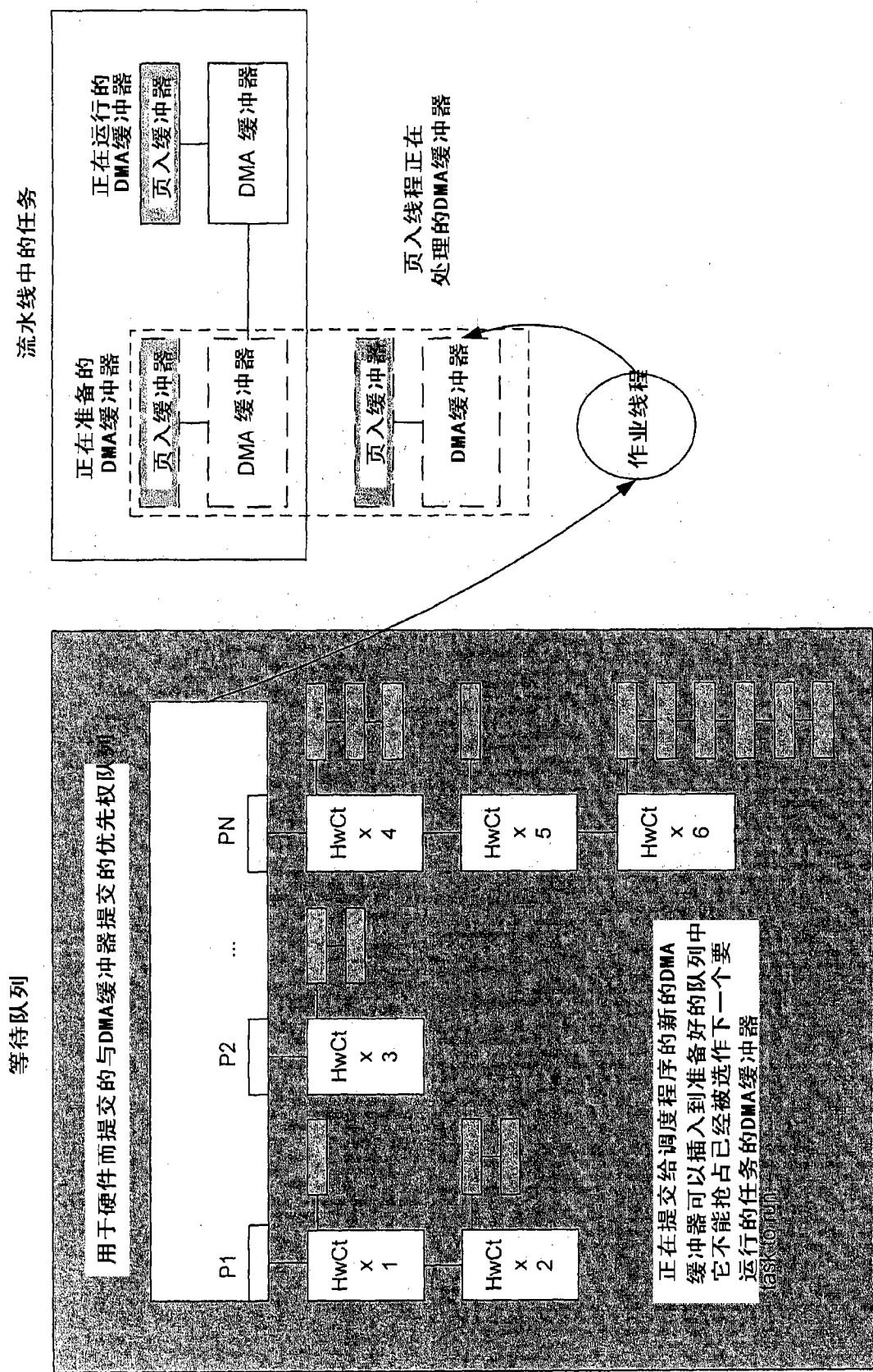


图 11

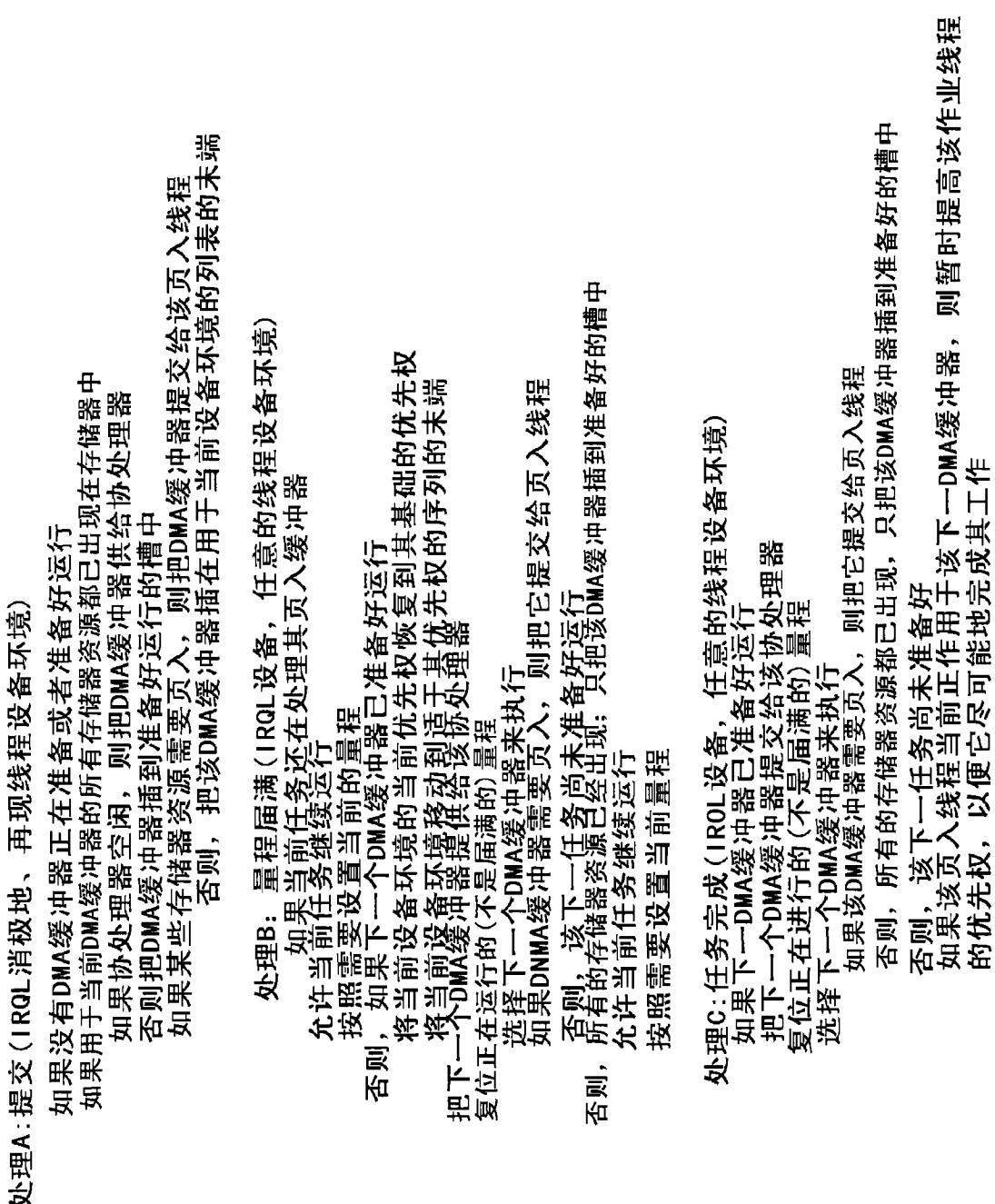


图 12 (A)

图 12(B)

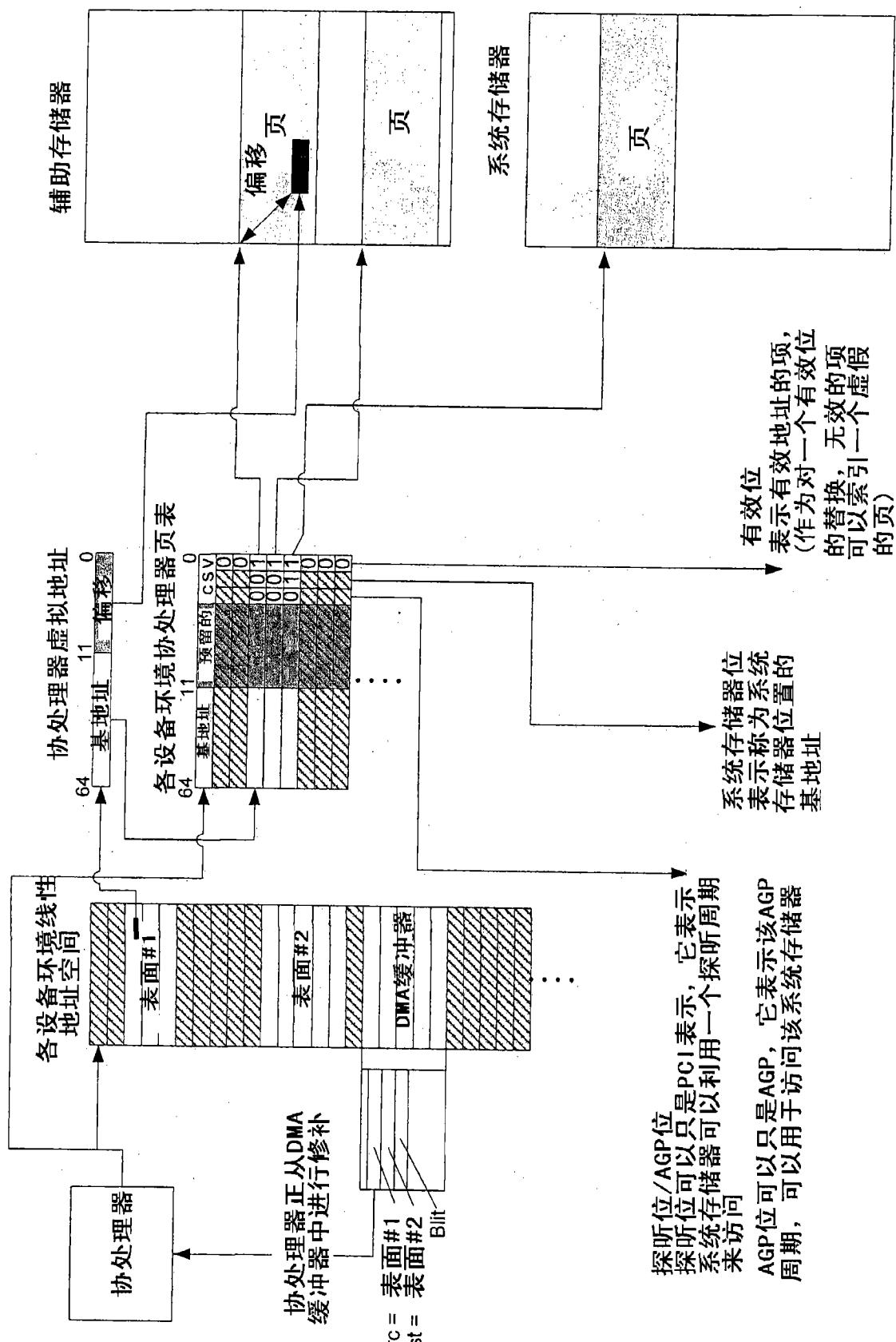


图 13

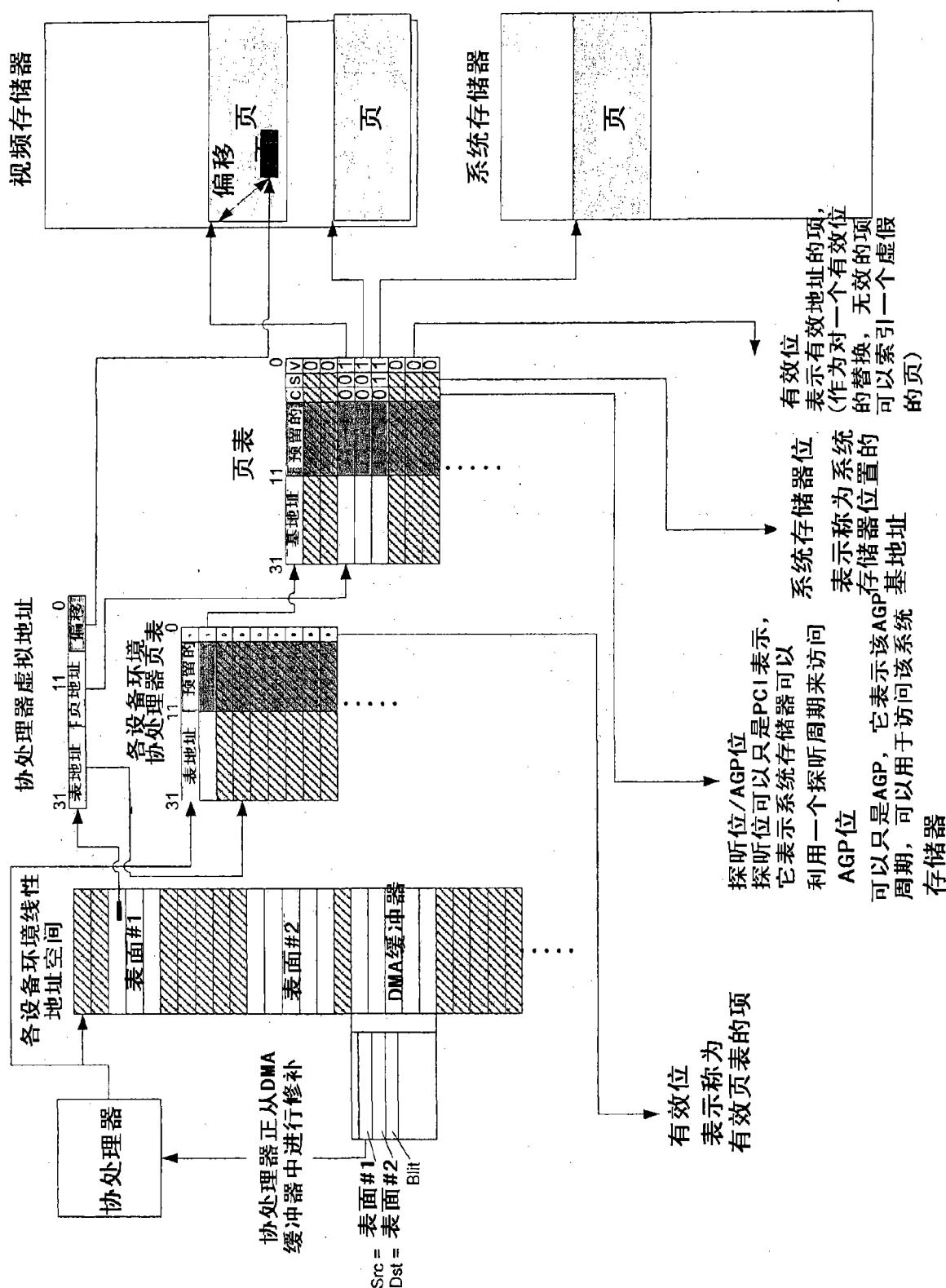


图 14

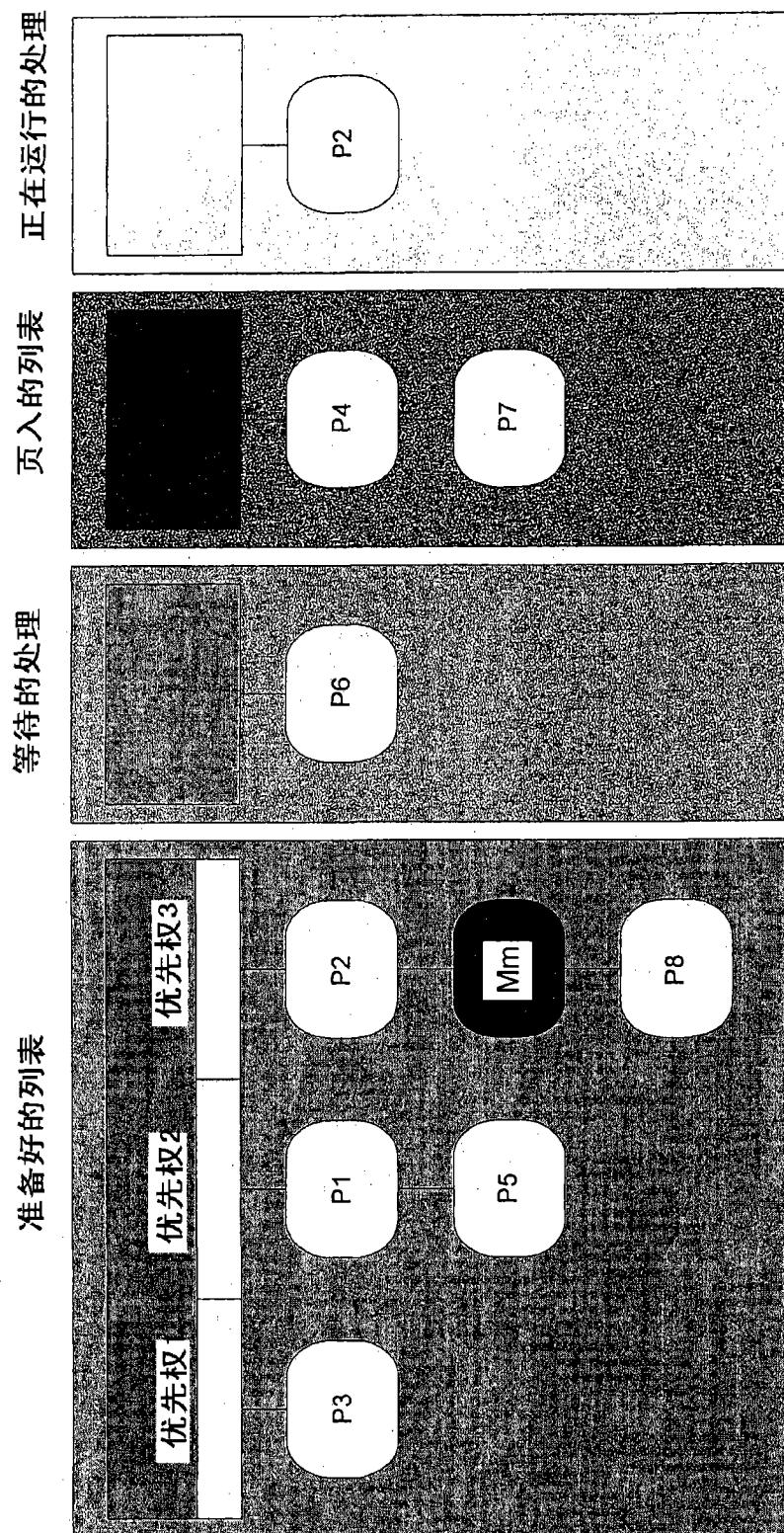


图 15

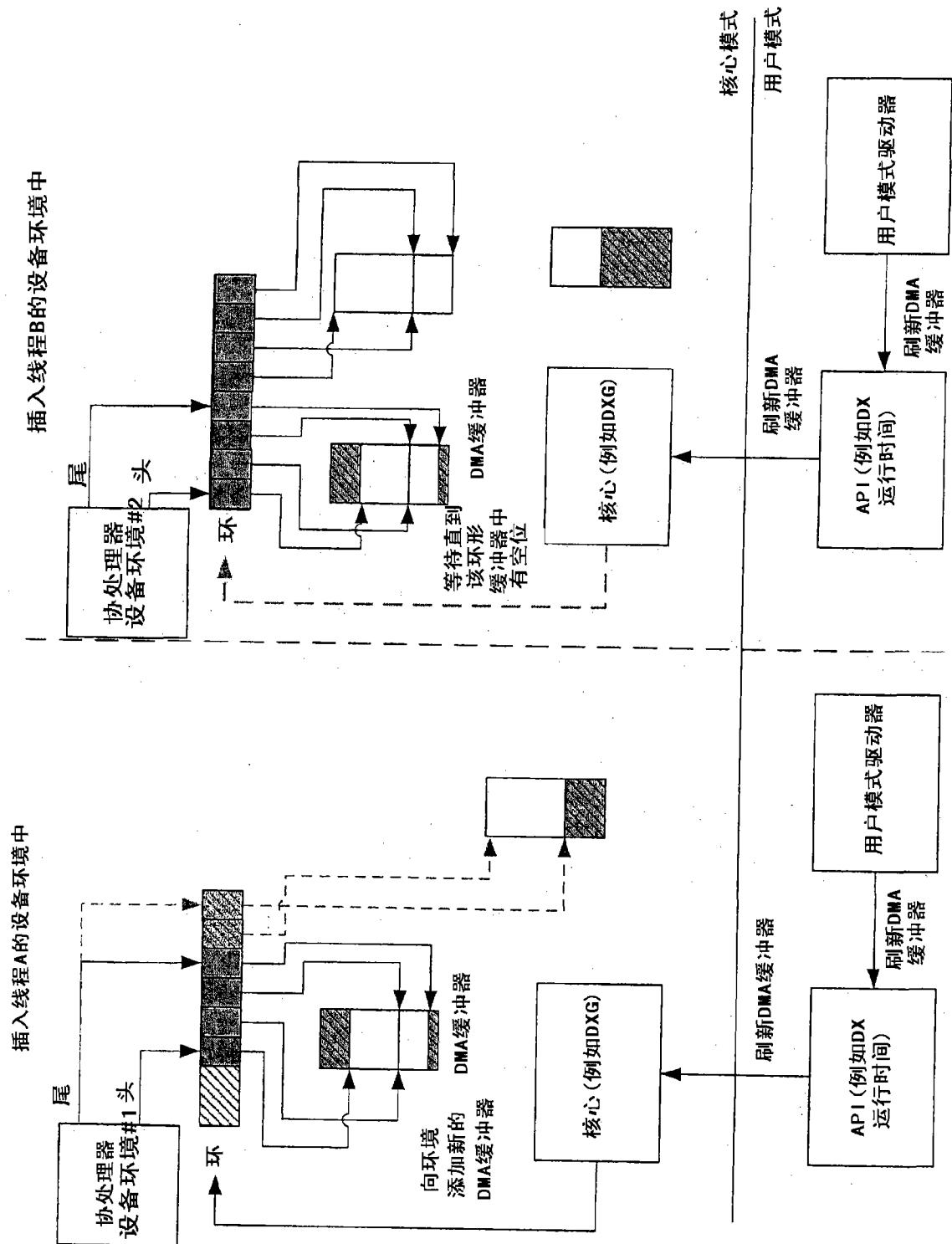


图 16

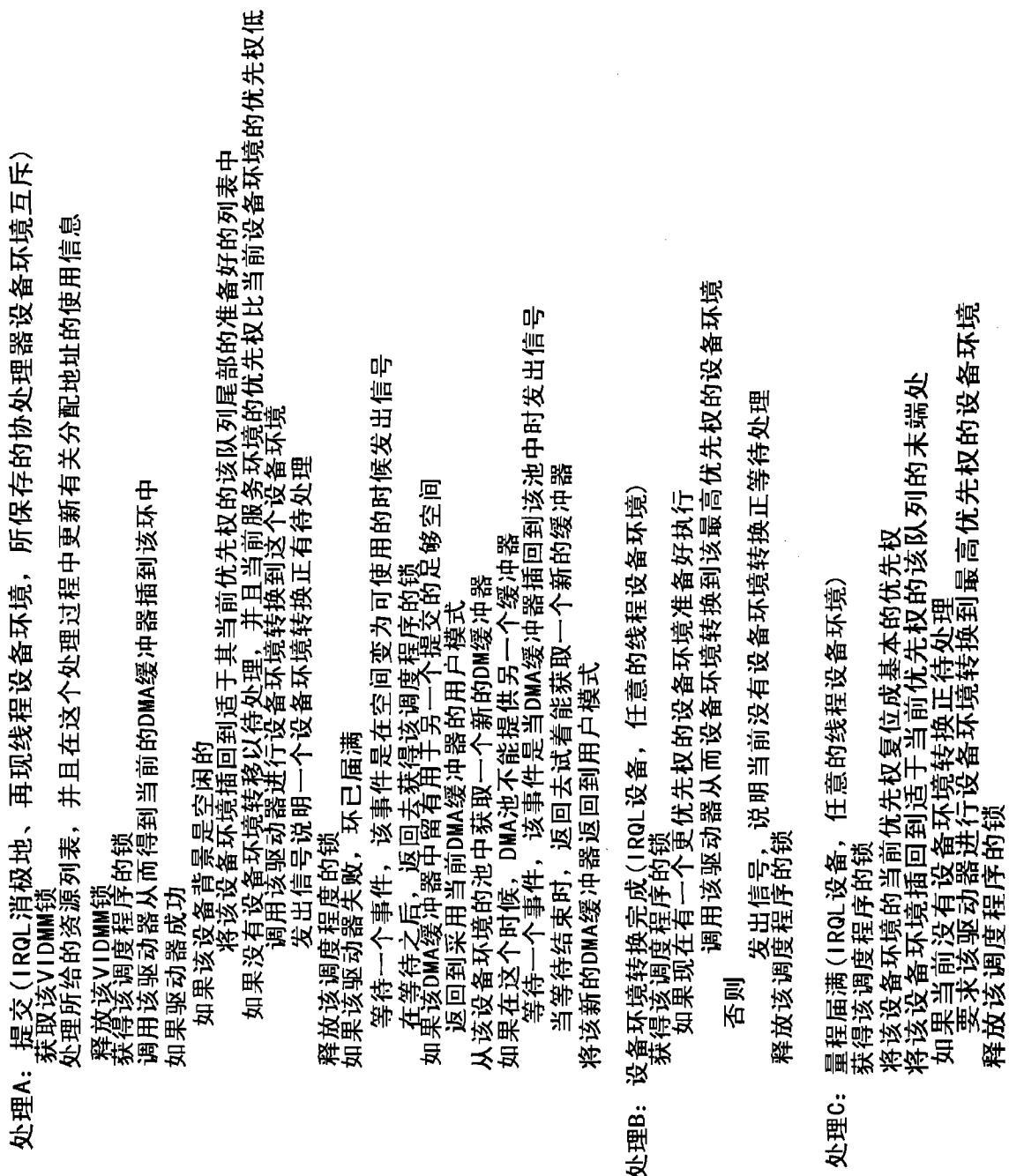


图 17 (A)

处理D: 任务完成 (IRQL 设备, 任意的线程设备环境)  
获得该调度程序的锁  
询问该驱动器该设备环境是否正是空的

如果该设备确实是空的  
将该设备环境的当前优先权复位到其基础的优先权  
将设备环境插入到该空闲列表中

如果该设备环境不是真的空的  
如果有设备环境不是待处理的  
要求该驱动器进行设备环境转换到最高优先权的设备环境  
释放该调度程序的锁

处理E: 页面故障 (IRQL 设备, 任意的线程设备环境)

获得该调度程序的锁  
从该准备好的列表中移除该设备环境  
将该页中的设备环境插入到该列表中作为一个自动操作

如果该页中的线程是空闲的  
如果排列DDC未发出信号唤醒该作业线程

如果当前没有设备环境是待处理的  
要求该驱动器进行一设备环境转换到最高优先权的设备环境  
释放该调度程序的锁

处理F: 解决故障 (IRQL, 任意的线程设备环境)

获得该调度程序的锁  
从该页表中移除该设备环境  
将该设备环境放到适于其当前优先权的准备好的列表中  
如果当前没有设备环境转换是待处理的，并且该当前的设备环境比当前运行的设备环境的优先权要高  
要求该驱动器执行设备环境转换到最高优先权的设备环境  
释放该调度程序的锁

- 处理G: 在页面作业线程中  
    经过该页入队列中的设备环境列表获得最高优先权的设备环境  
    要求驱动器有关在该设备环境上作进一步处理所需的资源列表  
    获得VIDMM锁  
    找出作进一步处理所需的各个分配地址的位置  
    使用于分配的虚拟地址或编号无效, 得以收回  
    要求该驱动器用必要的存储器转换命令来填充DMA缓冲器, 从而将所需的分配地址带到所选择的点上
- 释放该VIDMM的锁  
    提交该VIDMM设备环境作为一种常规的设备环境  
    如果设备环境的列表是空的, 则等待直到添加了一个项  
    放回到该循环到开始处
- 处理H: 循环计时器(消极程度, 系统线程设备环境)  
    获得该调度程序的锁  
    增加每个设备环境的当前优先权  
    释放该调度程序的锁

图 17 (C)

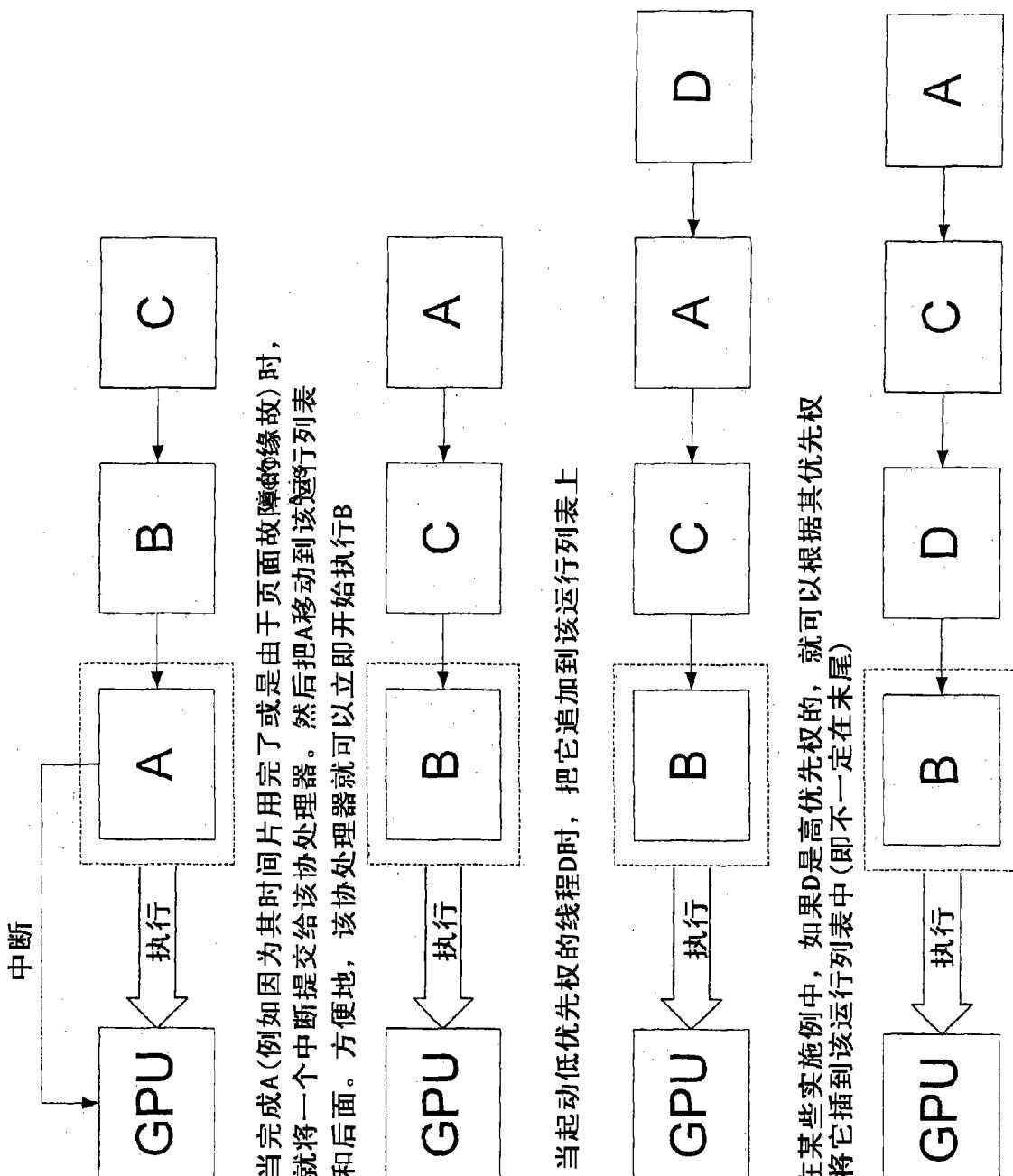


图 18

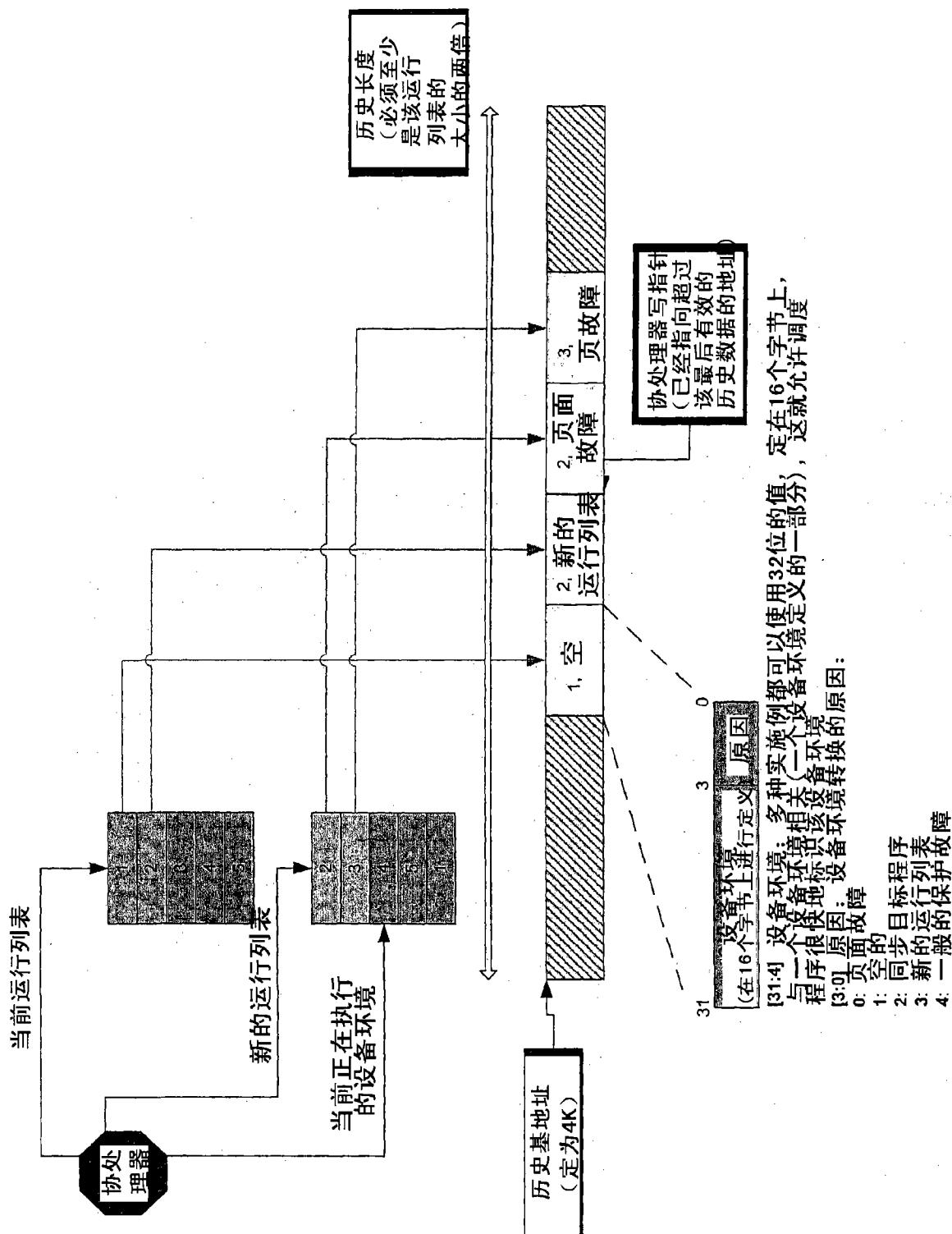


图 19

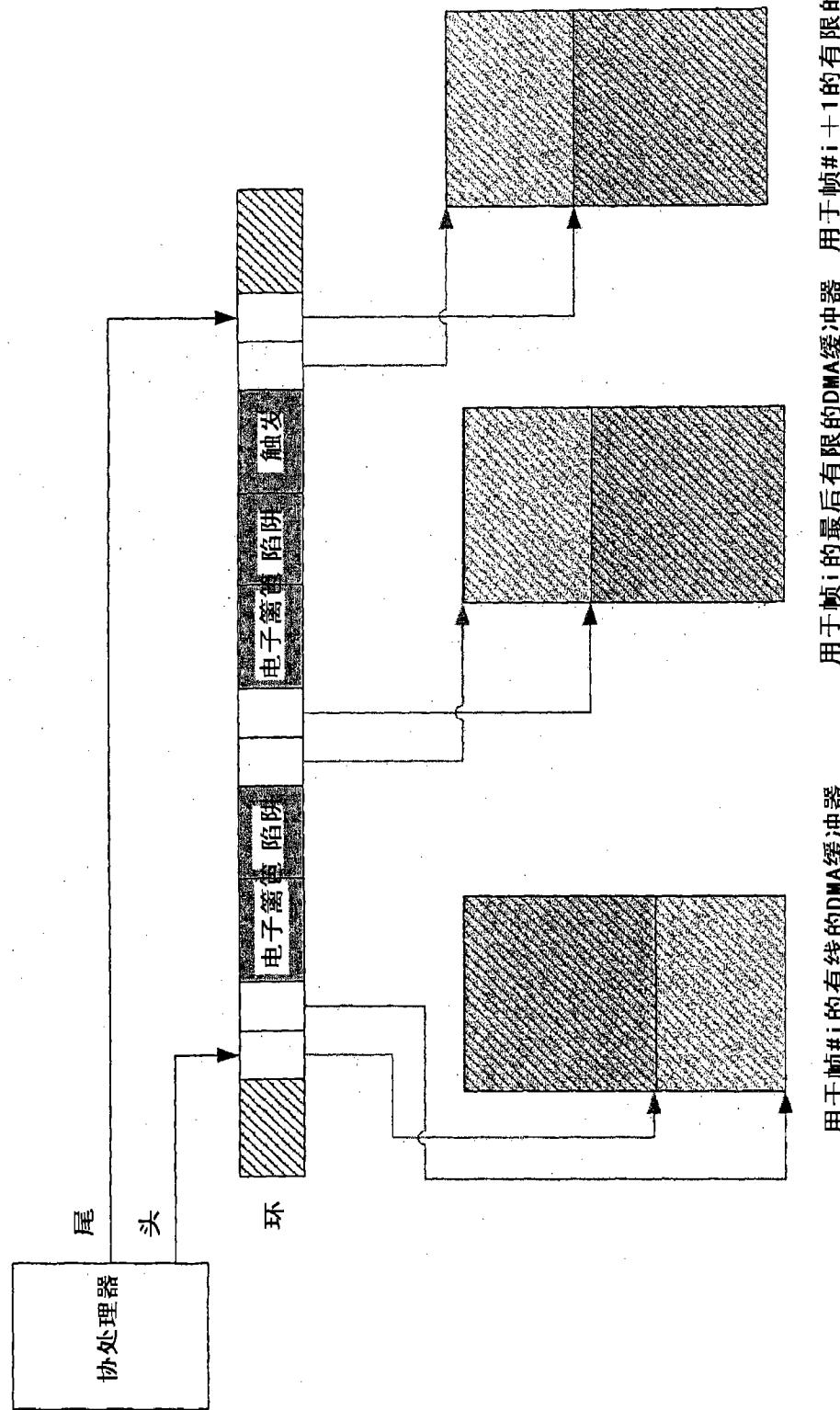
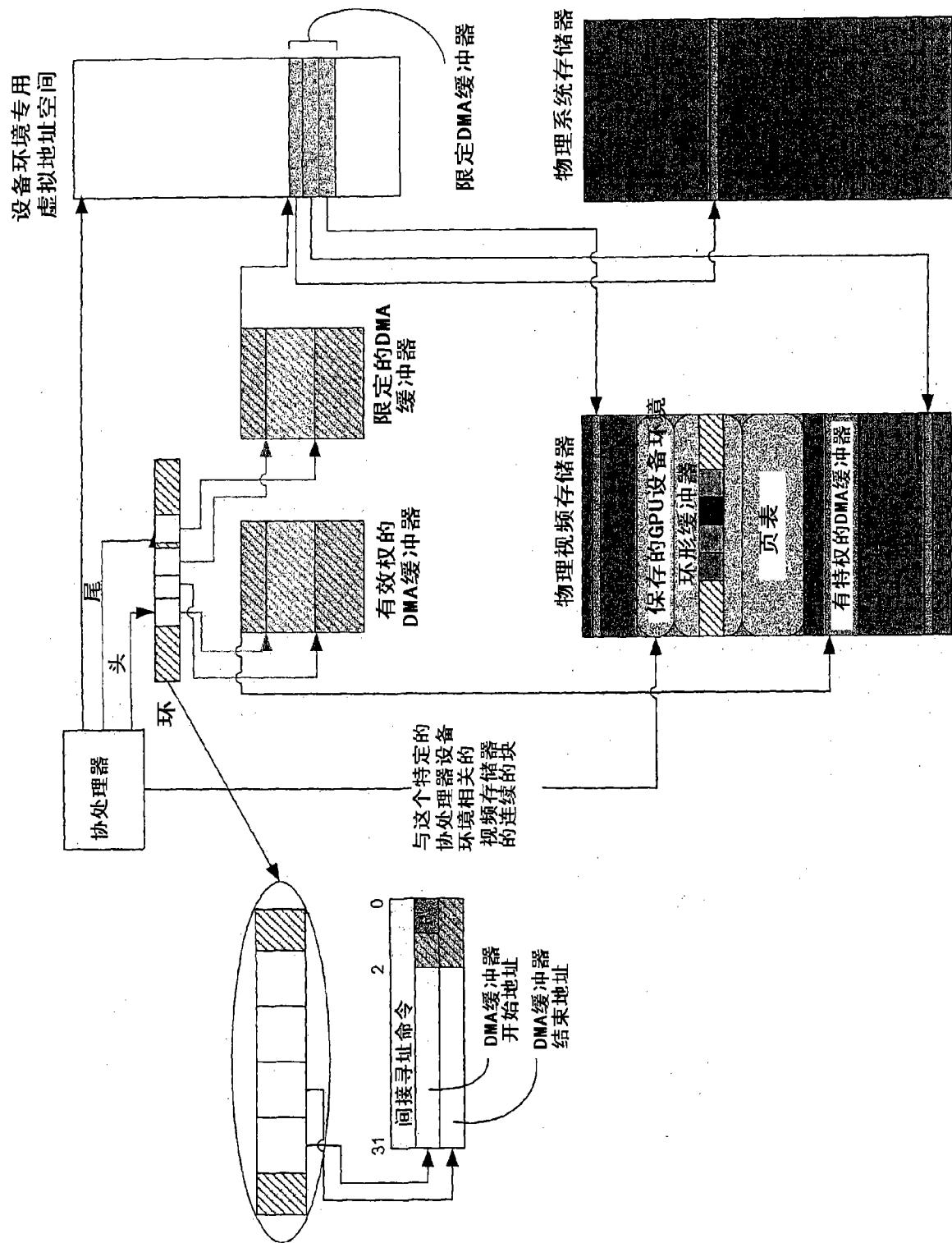


图 20



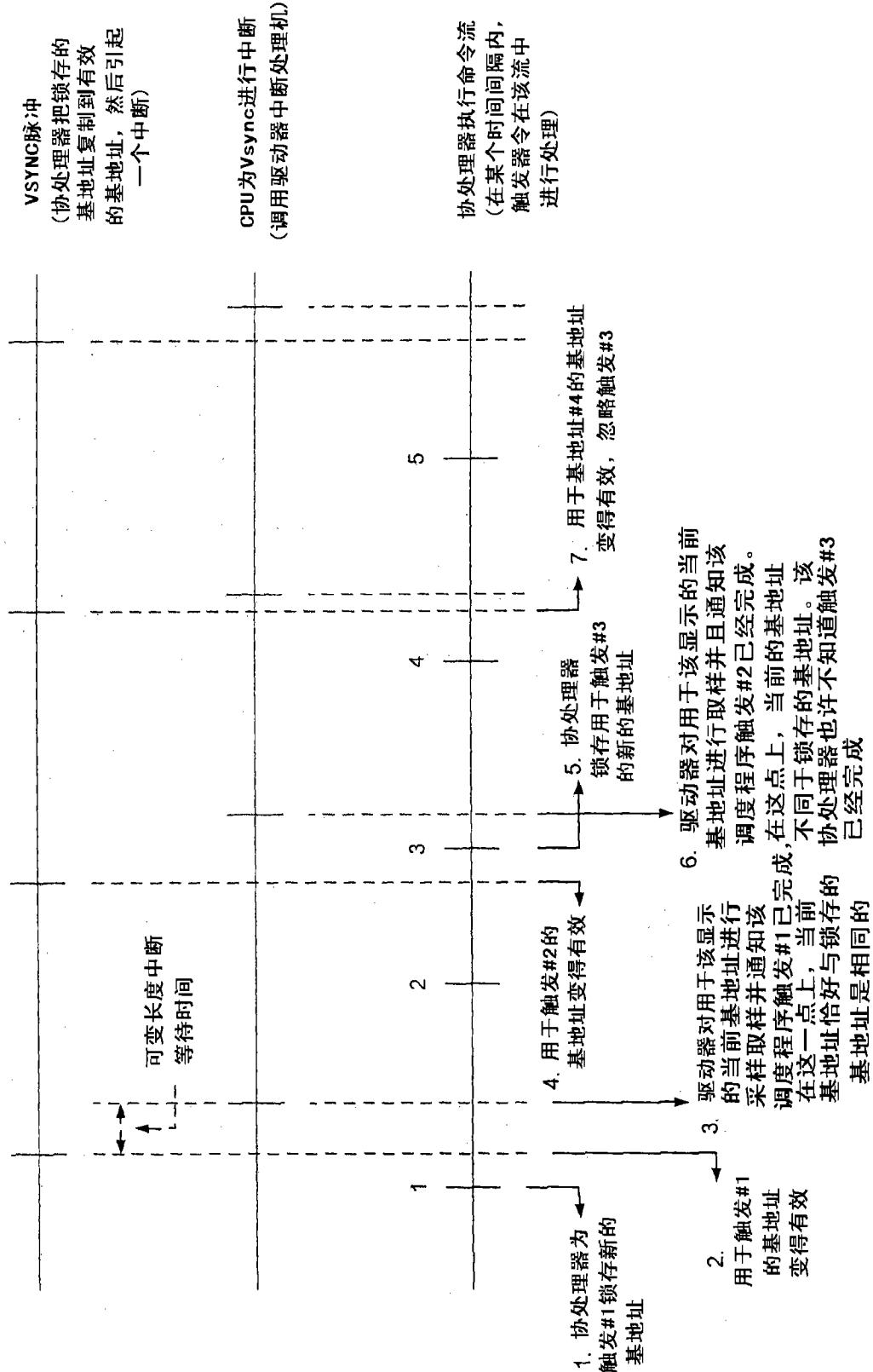


图 22

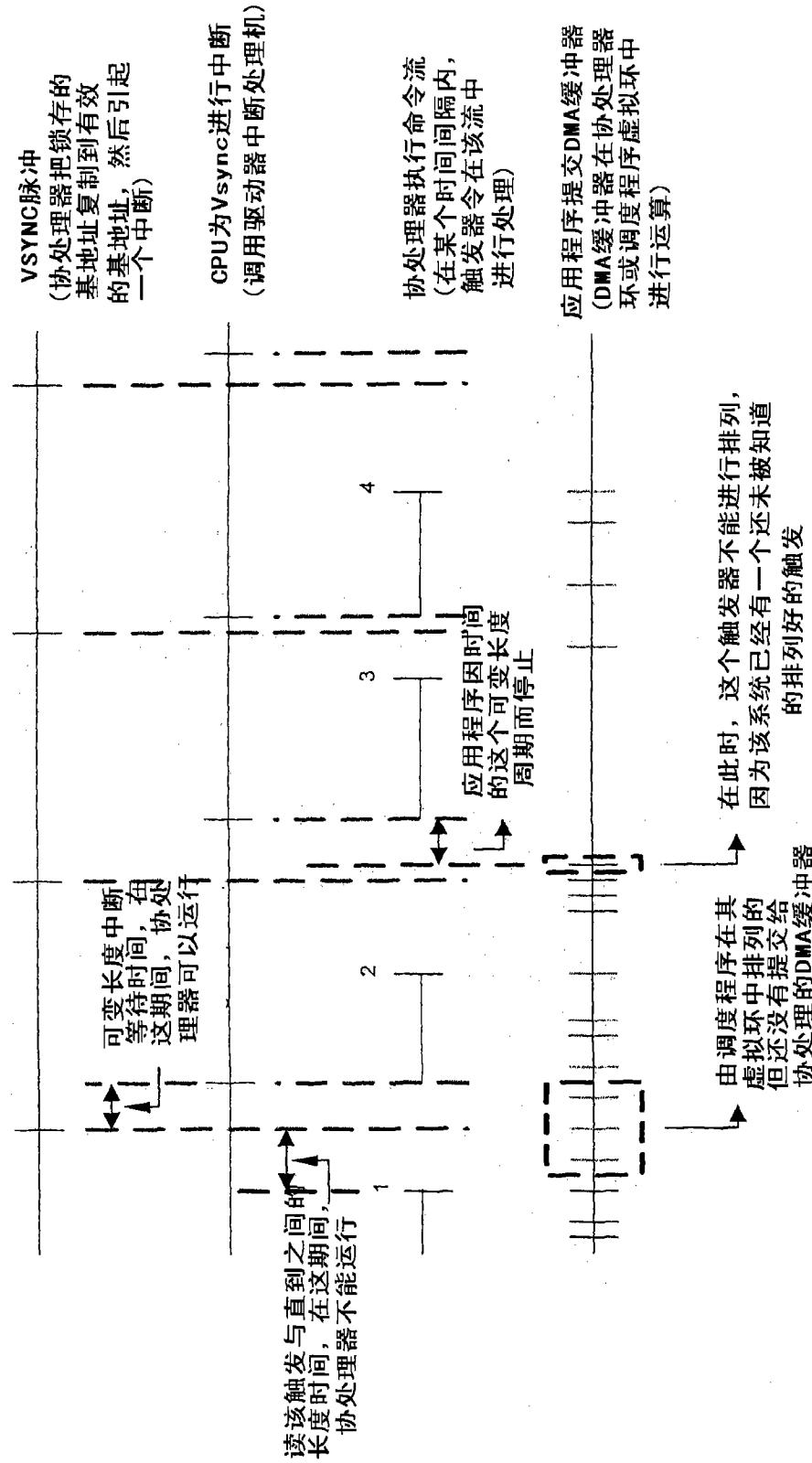


图 23

## 协处理器线程A

伪代码：

```

// 等待直到具有对共享表面的独占访问
// DxAcquireMutex(gSharedMutex);

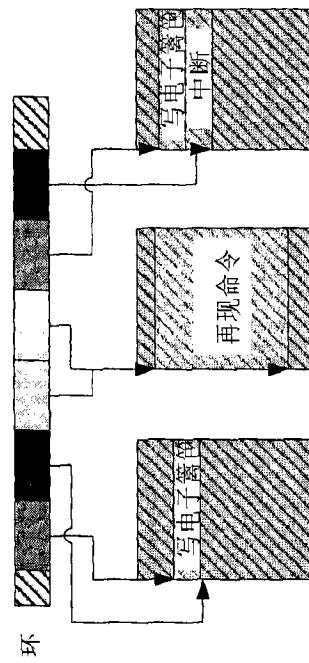
// 设置共享表面为描绘目标
// DxSetRenderTarget(gSharedSurface);

// 在共享表面内描绘所需的内容
// DxDrawSomething();

// 完成描绘，释放互斥
// DxReleaseMutex(gSharedMutex);

```

## 协处理器流：



有特权的DMA缓冲器 限定的DMA缓冲区 有特权的DMA缓冲器

## 协处理器线程B

伪代码：

```

// 等待直到具有对共享表面的独占访问
// DxAcquireMutex(gSharedMutex);

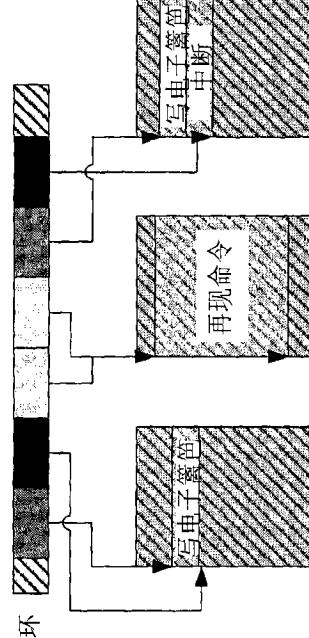
// 设置共享表面为纹理
// DxSetTexture(gSharedSurface);

// 在共享表面内描绘所需的内容
// DxDrawSomething();

// 完成描绘，释放互斥
// DxReleaseMutex(gSharedMutex);

```

## 协处理器流：



有特权的DMA缓冲器 限定的DMA缓冲区 有特权的DMA缓冲器 有特权的DMA缓冲器

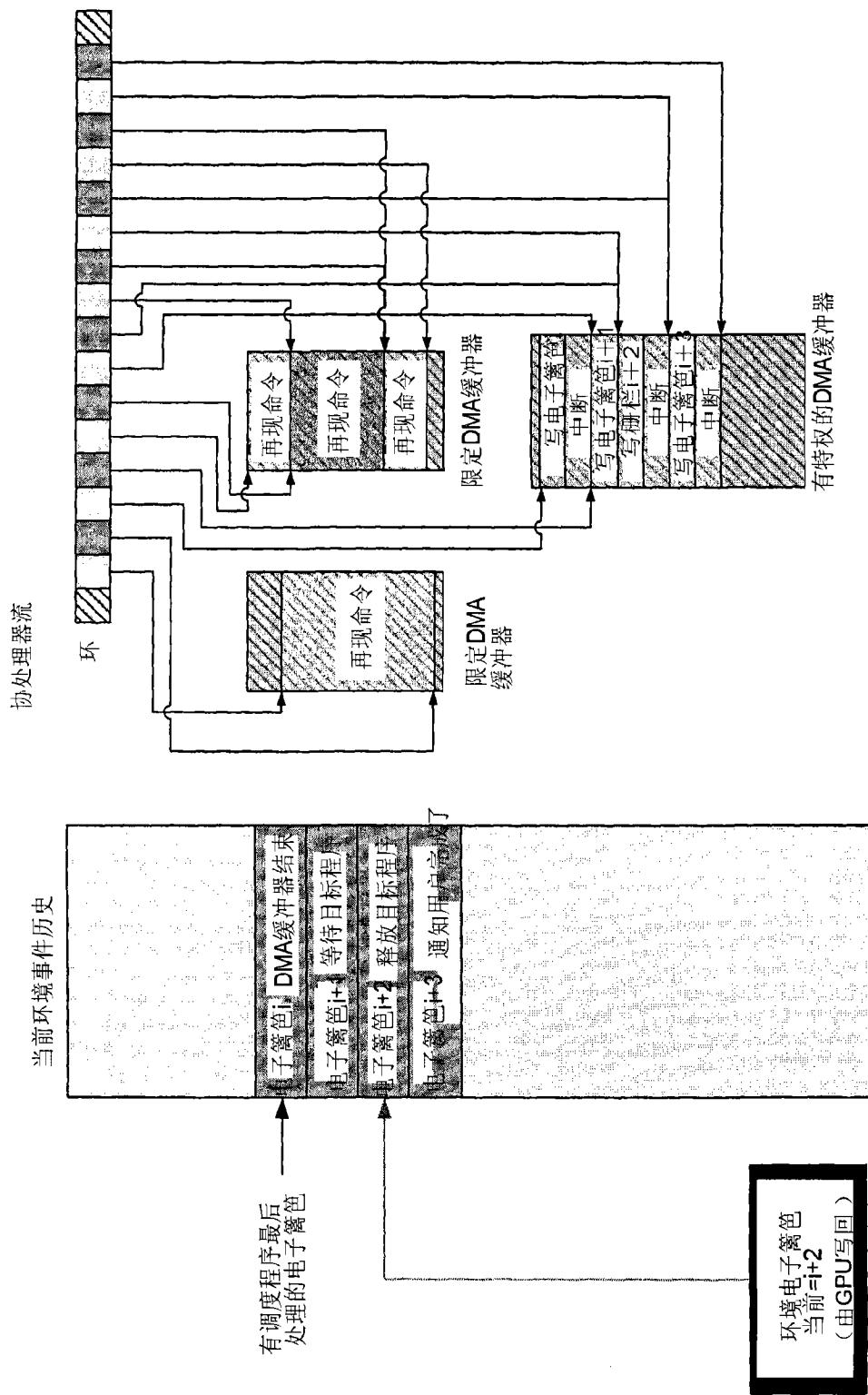


图 25

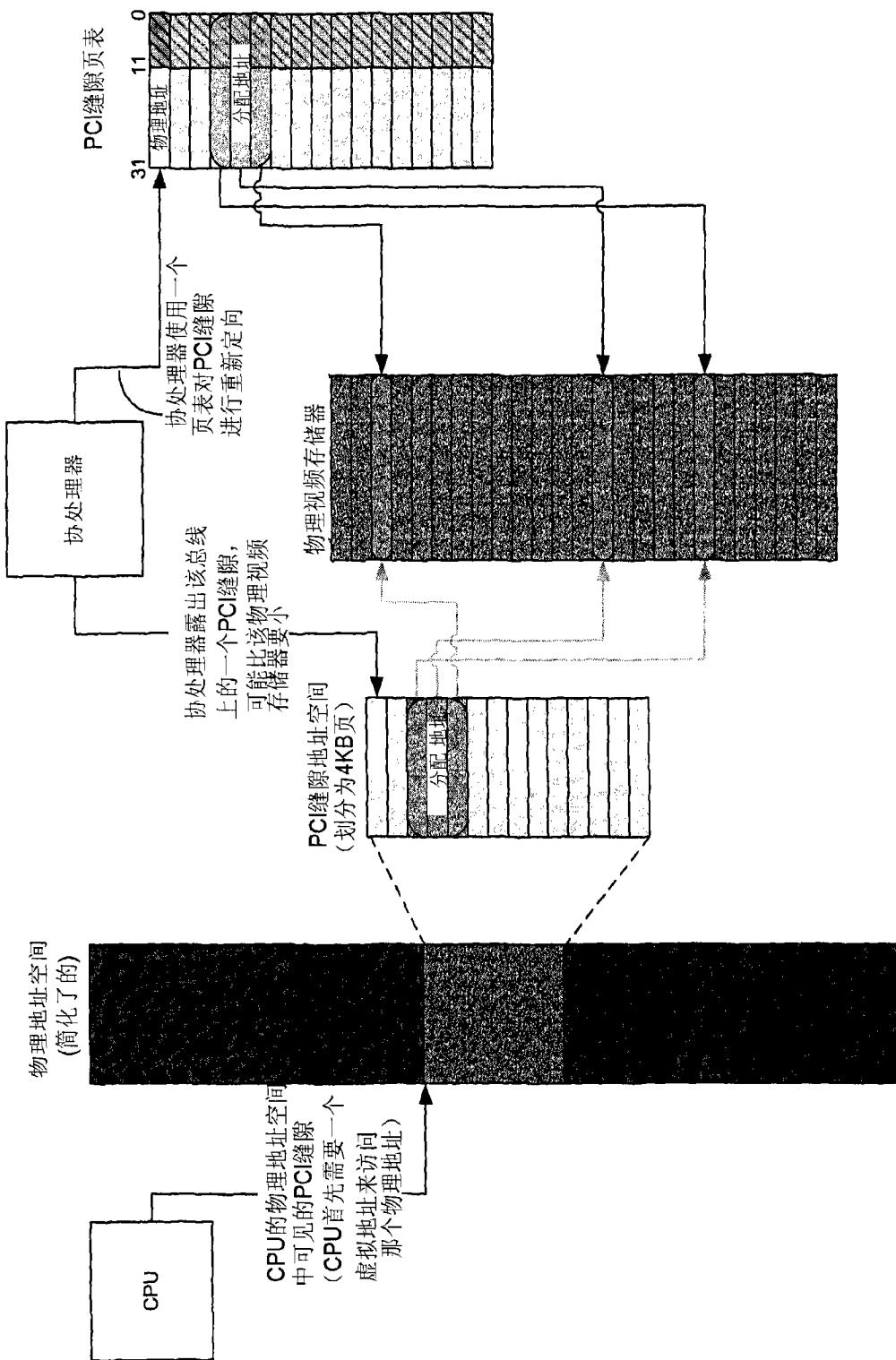


图 26