(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 101894044 B
(45)授权公告日 2012.05.09

(21)申请号 201010158612.5

(22)申请日 2004.02.18

(30)优先权数据
60/448,399 2003.02.18 US
60/448,402 2003.02.18 US
60/448,400 2003.02.18 US
60/474,513 2003.05.29 US
10/763,777 2004.01.22 US

(62)分案原申请数据
200410028357.7 2004.02.18

(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·戈沙利亚 S·布鲁诺弗斯特

(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

(51)Int.Cl.
G06F 9/48(2006.01)
G06F 9/50(2006.01)

(56)对比文件
US 5748468 A,1998.05.05, 全文.
WO 9739437 A1,1997.10.23, 全文.
EP 0592121 A1,1994.04.13, 全文.

审查员 安亚磊

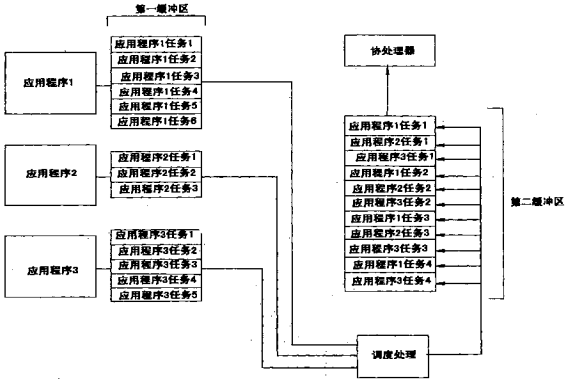
权利要求书 2 页 说明书 38 页 附图 29 页

(54)发明名称

用于调度协处理器的处理的方法和系统

(57)摘要

本发明提供了用于调度协处理器的处理的方法和系统。例如,本发明提供了一种用于向应用程序提供内存以支持协处理器处理任务的方法,包括:向一应用程序提供虚拟地址空间;在虚拟地址空间中存储与由协处理器处理的一个或多个任务有关的信息,其中所述一个或多个任务是由所述应用程序产生的;标识物理内存中对应于所述虚拟地址空间中的一虚拟地址的位置;当所述一个或多个任务被提交给所述协处理器以供处理时,访问所述物理内存中对应于所述虚拟地址空间中的所述虚拟地址的所述位置。



CN 101894044 B

1. 一种用于向应用程序提供内存以支持协处理器处理任务的方法，包括：

向一应用程序提供虚拟地址空间；

在虚拟地址空间中存储与由协处理器处理的一个或多个任务有关的信息，其中所述一个或多个任务是由所述应用程序产生的；

标识物理内存中对应于所述虚拟地址空间中的一虚拟地址的位置；

当所述一个或多个任务被提交给所述协处理器以供处理时，访问所述物理内存中对应于所述虚拟地址空间中的所述虚拟地址的所述位置。

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

3. 如权利要求 1 所述的方法，其特征在于，标识物理内存中的位置是由内存管理器完成的，所述内存管理器能将内存资源移动至物理内存中的另一个位置。

4. 如权利要求 1 所述的方法，其特征在于，所述与一个或多个任务有关的信息被分配给所述虚拟地址空间的分开的部分，每个部分包括环境信息，所述环境包括处理任务所需的内存资源的位置。

5. 如权利要求 1 所述的方法，其特征在于，还包括证实内存资源，其中所述证实包括发现空闲的物理内存的范围，并且请求驱动程序将内存资源句柄映射到该范围。

6. 如权利要求 1 所述的方法，其特征在于，所述虚拟地址空间是通过使用平面页表来虚拟化的，其中所述平面页表将协处理器可读的内存分成预定义内存数量的页面，其中在所述虚拟地址空间中进一步提供包含用于指定协处理器可读的内存地址的标识符的页表。

7. 如权利要求 1 所述的方法，其特征在于，所述虚拟地址空间是通过使用多级页表来虚拟化的，其中所述多级页表将协处理器可读的内存分成预定义内存数量的页面，其中在虚拟地址空间中进一步提供包含用于指定协处理器可读的内存地址的标识符的多个页表。

8. 如权利要求 1 所述的方法，其特征在于，所述物理内存的一部分被用来指示在物理内存中是否与需要处理的任务相关联的所有所需内存资源都是可供处理的。

9. 如权利要求 1 所述的方法，其特征在于，所述物理内存包括两部分，一个大的部分和一个小的部分，并且所述小的部分包括对所述大的部分中的内存单元的引用。

10. 如权利要求 9 所述的方法，其特征在于，所述小的部分引用所述大的部分中的四千字节的内存块。

11. 如权利要求 9 所述的方法，其特征在于，一辅助内存管理器将所述引用映射到所述大的部分中的内存单元。

12. 一种用于向应用程序提供内存以支持协处理器处理任务的计算设备，包括：

向一应用程序提供虚拟地址空间的装置；

在虚拟地址空间中存储与由协处理器处理的一个或多个任务有关的信息的装置，其中所述一个或多个任务是由所述应用程序产生的；

标识物理内存中对应于所述虚拟地址空间中的一虚拟地址的位置的装置；

当所述一个或多个任务被提交给所述协处理器以供处理时，访问所述物理内存中对应于所述虚拟地址空间中的所述虚拟地址的所述位置的装置。

13. 一种用于调度任务以供协处理器处理的方法，包括：

在应用程序接口 API 接收来自第一应用程序的调用，所述调用请求要求由协处理器处理的任务；

由所述 API 向用户工作方式驱动器发送所述任务用于存储在命令缓冲器内存组中，所述命令缓冲器是在所述第一应用程序的环境中分配的；

根据所述命令缓冲区的清除，由所述 API 从所述命令缓冲区接收所述任务；

由所述 API 将所述任务发送到协处理器内核用于处理；

由所述协处理器内核将所述任务交付给协处理器调度程序，其中协处理器调度程序功能包括确定处理所述任务和与一个或多个其它应用程序有关的其它任务的次序，所述次序指明在与所述任务有关的所述第一应用程序以及与所述其它任务有关的所述一个或多个其它应用程序中间的任何相对优先级，以及指明所述第一应用程序以及所述一个或多个其它应用程序被授权的相应处理时间量；

通过确保任何所需的内存资源在协处理器可存取的内存单元中是可用的，来预备任务以供处理，其中所述预备任务以协处理器调度程序所确定的所述次序发生；以及

向协处理器提交预备好的任务以供处理。

14. 如权利要求 13 所述的方法，其特征在于，所述 API 是 Direct3D 运行时 API。

15. 如权利要求 13 所述的方法，其特征在于，所述协处理器内核是 DirectX 内核。

16. 如权利要求 13 所述的方法，其特征在于，还包括由内核模式驱动器产生直接内存存取 DMA 缓冲区，其中要求由协处理器处理的一个或多个任务用于产生所述 DMA 缓冲区。

17. 如权利要求 16 所述的方法，其特征在于，还包括由内核模式驱动器产生一内存资源列表，其中协处理器需要由所述列表所表示的内存资源来处理由所述 DMA 缓冲区所表示的一个或多个任务。

18. 如权利要求 16 所述的方法，其特征在于，还包括构造分页缓冲区以将内存资源列表上的内存资源引入到协处理器可存取的内存单元内的正确的内存地址。

19. 一种用于调度任务以供协处理器处理的计算设备，包括：

在应用程序接口 API 接收来自第一应用程序的调用的装置，所述调用请求要求由协处理器处理的任务；

由所述 API 向用户工作方式驱动器发送所述任务用于存储在命令缓冲器内存组中的装置，所述命令缓冲器是在所述第一应用程序的环境中分配的；

根据所述命令缓冲区的清除，由所述 API 从所述命令缓冲区接收所述任务的装置；

由所述 API 将所述任务发送到协处理器内核用于处理的装置；

由所述协处理器内核将所述任务交付给协处理器调度程序的装置，其中协处理器调度程序功能包括确定处理所述任务和与一个或多个其它应用程序有关的其它任务的次序，所述次序指明在与所述任务有关的所述第一应用程序以及与所述其它任务有关的所述一个或多个其它应用程序中间的任何相对优先级，以及指明所述第一应用程序以及所述一个或多个其它应用程序被授权的相应处理时间量；

通过确保任何所需的内存资源在协处理器可存取的内存单元中是可用的，来预备任务以供处理的装置，其中所述预备任务以协处理器调度程序所确定的所述次序发生；以及

向协处理器提交预备好的任务以供处理的装置。

用于调度协处理器的处理的方法和系统

[0001] 本申请是申请日为 2004 年 2 月 18 日、申请号为 200410028357.7、发明名称为“图形处理单元的多线程内核”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，更具体而言涉及用于调度协处理器处理的硬件和软件。

背景技术

[0003] 目前，许多计算机系统都包括协处理器，比如图形处理单元 (GPU)。有时候，协处理器可以驻留在带有诸如微处理器之类的中央处理器 (CPU) 的系统的主板上，而在另外一些系统中，协处理器可以驻留在独立的图形卡上。协处理器在执行其处理任务进程中经常要访问辅助的内存，比如视频内存。目前的协处理器往往被优化成执行三维图形计算，以用来支持诸如游戏和计算机辅助设计 (CAD) 之类的应用程序。虽然当运行单个图形密集的应用程序时，目前的计算机系统和协处理器足以胜任执行工作，但是当运行多个图形密集的应用程序时它们可能会遇到问题。

[0004] 这其中的一个原因就是 :通常的协处理器不能有效地调度其工作量。目前的协处理器通常实现了协作式多任务处理，它是这样一种多任务处理，即其中当前控制协处理器的应用程序必须把控制让与其它的应用程序。如果应用程序没有让与控制权，那么它就能够有效地“独占 (hog)”协处理器。虽然当执行单个图形密集的程序时这还不存在显著的利害关系，但是当多个应用程序试图利用协处理器时，协处理器的独占问题可能会变得更加严重。

[0005] 尽管在 CPU 中已经虑及操作间的均分处理问题，其中多重操作的复杂调度已经变得必要，但是协处理器中的调度尚未得到有效地处理。这是因为 :在目前的系统中，协处理器通常被视作为一种用来把繁重的计算操作及耗时的操作转离 CPU 并为 CPU 提供更多用于其它功能的处理时间的资源。这种繁重的计算操作通常为图形操作，人们都知道，这种图形操作需要效果显著的处理能力。随着应用程序的复杂性增加，它们往往对协处理器需要更大的依赖性，以便处理稳健的计算活动。这种增加的依赖性继而又产生迄今为止无法预知的需要，以便克服准智能化均分协处理器资源中所涉及到的技术壁垒。对于这些及其它原因，就希望得到一种用于有效地调度协处理器工作及其它协处理器资源的使用的系统和方法。

发明内容

[0006] 为此，就希望得到一种用于有效地支持 GPU 中环境 (context) 切换的系统和方法。在一个实施例中，这种系统和方法使用了每协处理器的 (per-coprocessor) 环境地址空间。协处理器一般使用专用图形内存 (例如，图形卡上的内存)，或使用为 GPU 的使用而分配的部分计算机主系统内存，或使用上述两者的组合。在带有每协处理器的环境地址空间

的系统中，可以将 GPU 配置成依照其自己的内存观察 (view) 来提供每个应用程序或线程。举例来说，只要所述结构 (texture) 是特定应用程序所需要的，载入内存中的结构的地址就可以保持恒定。

[0007] 依照本发明各种不同的实施例，通过支持那些需要进行处理的各种任务之间的切换，协处理器能够首先处理较高优先级的任务，然后按比例分配多个任务之间的处理。当运行多个图形密集的应用程序时，协处理器可以转换环境以便服务于多个应用程序。本发明进一步考虑到任务列表（如队列）的维护，所述任务列表需要为每个应用程序进行处理。可以将这些任务提交给调度程序 (scheduler)，然后调度程序就能判定每个应用程序有权接收多少处理。结合此进程，系统就可以维护由内存管理器所管理的实际物理内存或虚拟内存当中的任务列表。此外，也提供了各种不同的技术，以用来判断特定任务是否已准备好进行处理，或者判断是否可能发生因内存资源失调而造成的故障。可以使用“运行列表”来确保协处理器不会在任务之间或中断之后耗费时间。本发明同时也提供了用于通过不允许应用程序修改部分内存来确保计算机系统安全的技术，其中所述部分内存可能是维护主系统操作正常运行的必要组成部分。本发明的这些及其它方面和优点将在下面作详细描述。

[0008] 本发明提供了一种用于向应用程序提供内存以支持协处理器处理任务的方法，包括：向一应用程序提供虚拟地址空间；在虚拟地址空间中存储与由协处理器处理的一个或多个任务有关的信息，其中所述一个或多个任务是由所述应用程序产生的；标识物理内存中对应于所述虚拟地址空间中的一虚拟地址的位置；当所述一个或多个任务被提交给所述协处理器以供处理时，访问所述物理内存中对应于所述虚拟地址空间中的所述虚拟地址的所述位置。

[0009] 本发明提供了一种用于向应用程序提供内存以支持协处理器处理任务的计算设备，包括：向一应用程序提供虚拟地址空间的装置；在虚拟地址空间中存储与由协处理器处理的一个或多个任务有关的信息的装置，其中所述一个或多个任务是由所述应用程序产生的；标识物理内存中对应于所述虚拟地址空间中的一虚拟地址的位置的装置；当所述一个或多个任务被提交给所述协处理器以供处理时，访问所述物理内存中对应于所述虚拟地址空间中的所述虚拟地址的所述位置的装置。

[0010] 本发明提供了一种用于调度任务以供协处理器处理的方法，包括：在应用程序接口 API 接收来自第一应用程序的调用，所述调用请求要求由协处理器处理的任务；由所述 API 向用户工作方式驱动器发送所述任务用于存储在命令缓冲器内存组中，所述命令缓冲器是在所述第一应用程序的环境中分配的；根据所述命令缓冲区的清除，由所述 API 从所述命令缓冲区接收所述任务；由所述 API 将所述任务发送到协处理器内核用于处理；由所述协处理器内核将所述任务交付给协处理器调度程序，其中协处理器调度程序功能包括确定处理所述任务和与一个或多个其它应用程序有关的其它任务的次序，所述次序指明在与所述任务有关的所述第一应用程序以及与所述其它任务有关的所述一个或多个其它应用程序中间的任何相对优先级，以及指明所述第一应用程序以及所述一个或多个其它应用程序被授权的相应处理时间量；通过确保任何所需的内存资源在协处理器可存取的内存单元中是可用的，来预备任务以供处理，其中所述预备任务以协处理器调度程序所确定的所述次序发生；以及向协处理器提交预备好的任务以供处理。

[0011] 本发明提供了一种用于调度任务以供协处理器处理的计算设备，包括：在应用程

序接口 API 接收来自第一应用程序的调用的装置，所述调用请求要求由协处理器处理的任务；由所述 API 向用户工作方式驱动器发送所述任务用于存储在命令缓冲器内存组中的装置，所述命令缓冲器是在所述第一应用程序的环境中分配的；根据所述命令缓冲区的清除，由所述 API 从所述命令缓冲区接收所述任务的装置；由所述 API 将所述任务发送到协处理器内核用于处理的装置；由所述协处理器内核将所述任务交付给协处理器调度程序的装置，其中协处理器调度程序功能包括确定处理所述任务和与一个或多个其它应用程序有关的其它任务的次序，所述次序指明在与所述任务有关的所述第一应用程序以及与所述其它任务有关的所述一个或多个其它应用程序中间的任何相对优先级，以及指明所述第一应用程序以及所述一个或多个其它应用程序被授权的相应处理时间量；通过确保任何所需的内存资源在协处理器可存取的内存单元中是可用的，来预备任务以供处理的装置，其中所述预备任务以协处理器调度程序所确定的所述次序发生；以及向协处理器提交预备好的任务以供处理的装置。

附图说明

[0012] 本专利或申请文件包含至少一个附图。

[0013] 图 1 是现有技术中协处理器的调度处理的概念例图。

[0014] 图 2 是依照本发明的协处理器调度改进的示范性例图。

[0015] 图 3 是在提供图 2 中从概念上举例说明的调度改进时所涉及的计算组件的更详细例图。

[0016] 图 4(A) 和图 4(B) 是伪代码算法，它们表示将图 3 的步骤组合成功能序列的各种非限定性的可能方式。

[0017] 图 5 举例说明了依照本发明、调度程序可如何使用已提供的信息，以定义用于直接内存存取 (DMA) 缓冲区中的内存资源的时间安排 (timeline)。

[0018] 图 6 是依照本发明的算法，其举例说明了准备工作器线程与辅助内存管理器之间的动态性。

[0019] 图 7 是依照本发明的分页缓冲区预备的示范性例图，它示出准备分页缓冲区并处理分页缓冲区的 CPU 预处理的工作器线程。

[0020] 图 8 是依照本发明、表示一连串在工作器线程中可能发生的事件的算法，所述工作器线程包括分页缓冲区中的栅栏 (fence) 处理。

[0021] 图 9 举例说明处于内核工作方式下的辅助内存管理器“VidMm”，所述内存管理器能为协处理器环境提供虚拟地址空间，并能管理各种不同协处理器环境中间的物理内存，以便它们能获得其应得的那份内存。

[0022] 图 10 举例说明了依照本发明的基本调度模型。

[0023] 图 11 举例说明了依照本发明的高级调度模型。

[0024] 图 12(A) 和图 12(B) 提供能实现高级调度模型的动作序列的示范性表示。

[0025] 图 13 举例说明了本发明结合可变长度平面页表的使用。

[0026] 图 14 举例说明了本发明结合多级页表的使用。

[0027] 图 15 是示范性进程的例图，所述进程是由与高级调度模型相结合的调度程序来维护的，所述高级调度模型支持表面级故障。

[0028] 图 16 是多重环境的例图，其中每一种环境都带有其自己的 DMA 环 (ring)，当结合本发明实现了表面级故障时能同时地处理所述多重环境。

[0029] 图 17(A)、图 17(B) 和图 17(C) 提供了一种伪代码算法，该算法描述了本发明结合图 16 的组件的操作，并且还包括可以证明有效的各种附加特征。

[0030] 图 18 是依照本发明、概念上表示运行列表使用的图表。

[0031] 图 19 举例说明了能够将环境转换历史写入到指定系统内存单元的硬件操作，所述指定系统内存单元可由结合本发明所使用的调度程序来读取。

[0032] 图 20 举例说明了通过将特权命令直接插入到协处理器环境环中来支持特权 DMA 通道的硬件方法。

[0033] 图 21 举例说明了用于支持协处理器中的受限相对于特权 DMA 缓冲区的方法，在所述协处理器中，将间接命令中的位 (bit) 插入到环形缓冲区中。

[0034] 图 22 提供了一种查询协处理器有关当前显示表面的方法。

[0035] 图 23 是当结合本发明使用即时交换 (flip) 时排队交换 (queuing flip) 的优选方法。

[0036] 图 24 是用于使对资源的访问同步化以确保两个或多个处理器在描绘 (rendering) 时能使用有效内容的示范性技术。

[0037] 图 25 举例说明了事件历史缓冲区的各种实施例。

[0038] 图 26 举例说明了支持使用 PCI 插孔的每协处理器的环境虚拟地址空间的优选方法，所述 PCI 插孔能在辅助内存中的任何地方被重定向。

具体实施方式

[0039] 许多这类的系统和方法在以下几篇美国临时专利申请中作了描述：由 Steve Pronovost 于 2003 年 2 月 8 日申请的“Video Memory Manager Rectangular Heap”、由 Steve Pronovost 于 2003 年 2 月 18 日申请的“视频内存 Manager Architecture”以及由 Steve Pronovost 于 2003 年 2 月 18 日申请的“GPU Scheduler Overview”，序号为 60/448,402。将这三篇临时专利申请全部引入于此，以供参考。

[0040] 通过图 1 和图 2 的比较，能够从概念上说明本发明所完成的部分改进。图 1 表示协处理器的任务调度的一般的现有技术方法。提供能够被各种应用程序访问的缓冲区，所述各种应用程序比如是应用程序 1、应用程序 2 和应用程序 3。所述应用程序能够将协处理器的任务载入到缓冲区中，并且协处理器可以在完成预先提交任务之后处理那些任务。正如所说明的那样，这个方法未解决潜在的协处理器的“独占 (hogging)”。在图 1 中，应用程序 1 正在独占协处理器。应用程序 1 已经请求协处理器对七个任务进行处理，与此同时其它两个已组合的应用程序已经请求仅仅处理三个任务。在像这种多个应用程序需要协处理器的情形下，诸如像图 2 所提供的这类系统可以提供改进功能。

[0041] 图 2 举例说明了依照本发明的一种系统和方法，借此每个应用程序，例如应用程序 1、应用程序 2 和应用程序 3 能够维持其自己的缓冲区，即图 2 的“第一缓冲区”。将这些缓冲区（下文将称为“命令缓冲区”）提交给调度进程，该调度进程能够确定何时把各种任务交付给协处理器。如图 2 中举例说明的那样，在这种情况下，调度进程已经把任务插入到“第二缓冲区”中。为简明起见，将图 2 的“第二缓冲区”举例说明为单个缓冲区。然而，实

际上，可能需要几个缓冲区来执行图 2 中的“第二缓冲区”的功能。图 2 的第二缓冲区已将任务分开以待交付给协处理器，以便应用程序 1 不再能独占协处理器资源。调度进程已经允许将应用程序 1 作为协处理器上的第一个任务，继而是应用程序 2、再继而是应用程序 3、然后再是应用程序 1 等等。

[0042] 尽管图 2 中从概念上举例说明的系统和方法的实施方案比图 2 示范的更加复杂，但是这里所公开的改进方案通常致力于支持如图 2 中所举例说明的基本概念。现在，转到本发明实施例的更详细说明，为便于参考而提供下列术语定义：

[0043] 命令缓冲区 - 一种由用户工作方式驱动器构造的缓冲区。这种缓冲区可以是在描绘 (rendering) 应用程序的情况下的规则的可分页内存。

[0044] DMA 缓冲区 - “直接内存存取”缓冲区。一种内核工作方式驱动器构造的缓冲区。这种缓冲区可以是基于命令缓冲区的内容的。一般说来，它是从内核可分页内存中分配的，并且仅对内核而言是可见的。在这点上，页面可在协处理器能从其中读取之前通过插孔来锁定并映射。

[0045] 分页缓冲区 - 一种由内核工作方式驱动器构造的缓冲区。这种缓冲区可用于页入、驱逐和移动特定 DMA 缓冲区所需的内存资源。分页缓冲区可以配置成在其 DMA 缓冲区对应方之前立即运行。

[0046] 环形缓冲区 - 这是一种协处理器环境特定的缓冲区。可以将到 DMA 缓冲区的方向插入到这种缓冲区中。在这点上，协处理器能够从这种环形缓冲区中取出命令来执行。环形缓冲区通常包含重定向指令，一旦 DMA 缓冲区已经完全被处理过，该重定向指令就指示协处理器开始从 DMA 缓冲区中读取命令，并继而返回到环形缓冲区。

[0047] 辅助内存 - 通常专由协处理器使用的内存，它不必是物理系统内存的一部分。举例来说，它可以是驻留在图形卡上的本地视频内存。它也可以是其它的协处理器可读的内存，比如通过系统内存插孔映射的内存。一般而言，这种内存不不存在于集成的或 UMA 图形设备中。这种内存不是经由基于 GART 状页表的插孔来进行访问的。

[0048] 系统内存插孔 - 这是物理系统内存的子设备 (subset)。它对于协处理器经由基于 GART 状页表的插孔而言是可见的。CPU 将能独立于系统内存插孔而访问物理系统内存。当经由插孔访问这种内存时，概念上相似的一些示例为：加速图形接口（“AGP”）内存、外设部件互连（“PCI”）专用内存、或者统一内存体系结构（“UMA”）内存。

[0049] 可以在图 3 中看到本发明各种实施例的更详细视图。图 3 提供了各种软件和硬件对象的示意图，可以把所述软件和硬件对象组合起来，以提供图 2 中从概念上示范的功能。图 3 显现了将在下面描述的一系列顺序步骤。为了清楚地解释和实现本发明起见，顺序地显现出这些步骤，但是不应该把它理解成显现实践本发明所需要的序列。依照所属技术领域中已知的或者未来将开发的实践方式，上述顺序是可以变化的。下列论述将从图 3 的系统和方法的概观开始，然后继续进行图 3 一些方面中的更详细论述。

[0050] 图 3 中的步骤 1 表示对应用程序接口（“API”）的应用程序调用。一个应用程序可以是供用户编制软件用的任何文件组。API 一般是供应用程序与操作系统内核进行通信而使用的语言和消息格式，而且还是指用来与其它控制程序进行通信的这类格式，所述其它控制程序比如是数据库管理系统 (DBMS) 或通信协议。结合本发明使用的一个示范性 API 是由微软公司(MICROSOFT®)开发的 Direct3D Runtime API。

[0051] 步骤 2 表示从 API 到用户工作方式驱动器的调用。通常，用户工作方式驱动器是能将软件系统（通常为操作系统）链接到外围子例程的程序例程（或硬件），外围子例程可以是软件或硬件。这里，用户工作方式驱动器接收来自于 API 的调用，所述 API 可以包含与来自于步骤 1 的原始调用相对应的 API 参数。步骤 3 表示命令缓冲区中描绘 (rendering) 命令的累积，所述描绘命令是由用户工作方式驱动器生成的。缓冲区是为用作中间储存库而保留的内存区域。当等待在两个位置之间进行传递数据时，数据可以被暂时保存在缓冲区中，所述位置比如是数据区域以及用来处理的处理器或协处理器。正如下面进一步描述的那样，可以选择由用户工作方式驱动器生成的命令缓冲区内容的具体细节，以便于转换成硬件特定的 DMA 缓冲区。而且，在定义命令缓冲区的进程中，忽略对内存资源的直接内存引用可能是有用的，所述内存资源比如“结构 (texture)”或“顶缓冲区 (vertex buffer)”。或者，独立硬件供应商（“IHV”）可以定义选择性包含句柄 (handle) 的命令缓冲区，以便当这种内存资源正被创建时，内核接口可以提供对命令缓冲区的内存引用。

[0052] 步骤 4 表示命令缓冲区的清除 (flushing)。“清除”简单来说是指让所累积的描绘命令变空。正如举例说明的那样，为了将描绘命令传送到图 3 中所示的协处理器内核，可以将这些描绘命令送回到API。清除可能因任何原因发生，包括但不限于因命令缓冲区已满和需要更多空间以供新来的描绘命令用而清除，以及因命令缓冲区中存在需要立即处理的高优先级描绘命令而清除。

[0053] 步骤 5 表示由 API 将已累积的命令缓冲区清除到协处理器内核。内核通常称作为操作系统的核心部分，选择性地管理内存、文件和外围设备的部分，并且也可以运行应用程序和分配系统资源。将要领会到的是：协处理器内核可以是任何类型的内核，包括主系统内核或独立地特定协处理器的内核，或例如是一种特定类型的内核，比如MICROSOFT®的 DirectX 内核（“DXG”）。

[0054] 步骤 6 表示命令缓冲区到内核工作方式驱动器的提交情况。协处理器内核能够把命令缓冲区指向内核工作方式驱动器。正如上面参照用户工作方式驱动器所描述的那样，内核工作方式驱动器通常可以是除能以内核工作方式操作的内核工作方式驱动器以外的驱动器，就如其名称暗指的那样。在这点上，内核工作方式驱动器能负责将命令变换成 DMA 缓冲区。IHV 可以考虑将用以确保命令缓冲区的适当证实 (validation) 及复制的适当机制提供至指定 DMA 缓冲区的内核工作方式中。DMA 缓冲区可以是指定硬件的，因为它们是最终指定给协处理器的命令的汇集，因此它们应该正确地与协处理器和支持硬件相对接。

[0055] 注意：横贯图 3 的水平线将图 3 分成用户工作方式和内核工作方式。正如该直线暗指的那样，本发明可以在计算机内存分配的常规布局内操作，所述常规布局是为系统安全而实施的。用户工作方式为无特权的内存，它可以被应用程序访问。另一方面，内核工作方式是有特权的，它不能被应用程序访问。尽管内核工作方式分配的 DMA 缓冲区理论上能被映射到任何存储空间中，但是应该紧记的是，映射到应用程序的专用进程空间中可能会导致安全性风险。这是因为在应用程序的专用进程空间中，由线程指示的任何虚拟地址的内容都能被修改；换言之，DMA 缓冲区的内容可以在它被证实的时刻与它被硬件处理的时刻之间被修改。

[0056] 正如步骤 7 表明的那样，内核工作方式驱动器也可以构造一个由 DMA 缓冲区使用的内存资源列表。可以把这当作为命令缓冲区证实的一部分来完成。这个列表例如可以

包括：列表上各种内存资源的内核句柄(handle)和内存资源被引用的缓冲区位置(buffer location)。此外，这个列表还可以包括所列内存资源的预期环境状态。这允许作为任何当前硬件状态一部分的内存资源（例如“当前的描绘目标”、“当前的 z 缓冲区”等）成为列表的一部分，所述列表将在 DMA 缓冲区起始处被重新编程，这是因为它们可能自上一个 DMA 缓冲区被提交给协处理器以来改变了位置。

[0057] 步骤 8 表示连同任何内存资源列表一起将 DMA 缓冲区发送到协处理器内核。所述协处理器内核继而可以向协处理器调度程序提交 DMA 缓冲区，正如步骤 9 所示的那样，并且返回到用户工作方式，正如步骤 10 中那样。

[0058] 协处理器调度程序通常负责为协处理器调度任务流（诸如被包含在各种 DMA 缓冲区中以及被送到协处理器的其它工作）。协处理器调度程序的功能可能是很广的，而本说明书包括许多协处理器调度程序可能执行的潜在功能。协处理器调度程序可以称为协处理器调度程序或者简称为调度程序。在各种不同的实施例中，如图 3 所示，调度程序可以在将 DMA 缓冲区提交到协处理器之前先执行一个或多个功能。步骤 11a 动态地说明了：调度程序的一个功能就是提交待处理的 DMA 缓冲区。

[0059] 步骤 11b 表示调度程序确定添加到预备的 DMA 缓冲区的列表或者下一次运行的 DMA 缓冲区选择。在这点上，调度程序能够将 DMA 缓冲区传递至预备线程。这里使用的术语‘预备线程’通常提供了确保现存有适当内存资源以处理 DMA 缓冲区的功能。首先，预备线程可能调用辅助内存管理进程（未示出），来确定用以对全部所需的内存对象进行分页的充足存储单元（在图形环境中，“表面”），其中所述内存对象当前不在辅助内存中（这是步骤 12）。注意，术语“辅助内存”是指为供协处理器使用而分配的内存；就 GPU 协处理器来说，将辅助内存通常称为“视频内存”。

[0060] 有可能并非 DMA 缓冲区所需的全部内存资源都将立刻适合可用的辅助内存。在这一点上来讲，由于种种原因，辅助内存管理器可能无法取来辅助内存中的所有表面。如果这种情况发生的话，那么可能会进行某种进一步的处理，以便在辅助内存中腾出更多空间，或者作为选择与腾出更多空间相结合进行，DMA 缓冲区可以被分成多个片段(fragment)。在这种情况下，预备线程可以利用驱动器预定义分割点来分割缓冲区，并且试图定位小型 DMA 缓冲区所需的内存资源的子设备(subset)。

[0061] 如步骤 13 举例说明的那样，一旦为 DMA 缓冲区定位了充足的辅助内存，预备线程就可以调用内核工作方式驱动器。正如本领域技术人员将会领会到的那样：这可以是结合步骤 6、7 和 8 所提及的内核工作方式驱动器，或者它可以是独立的内核工作方式驱动器。

[0062] 步骤 14 举例说明了：内核工作方式驱动能够为等待处理的 DMA 缓冲区构造分页缓冲区。内核工作方式驱动器可以根据来自于预备线程的处理命令来构造这种分页缓冲区。如上文所定义的，分页缓冲区是以分页内存资源为目的的缓冲区。“分页”是指利用映射硬件来改变内存块（页面）的物理地址。一般而言，分页缓冲区是包含协处理器指令的 DMA 缓冲区，所述协处理器指令用于将内存资源转移到它们指定的存储单元。分页缓冲区起到了将 DMA 缓冲区所需的任何内存资源引入到正确存储单元的作用，其中必要时协处理器可以从所述存储单元那里访问那些资源。如果恰当地产生了分页缓冲区，则特定协处理器任务（即，DMA 缓冲区）的任何必备内存资源的存储单元就都是已知的。

[0063] 步骤 15 表示通知预备线程已经产生分页缓冲区。步骤 16 表示分页缓冲区就绪

的调度程序的信号。在此，调度程序可以假定下一个 DMA 缓冲区就绪，以待进行处理，或者它可以在将其发送到协处理器以进行处理之前，先对 DMA 缓冲区继续进行进一步的预备操作。例如，由于存储单元也许自原始 DMA 缓冲区创建以来已经改变，因此调度程序此时可以再一次调用内核工作方式驱动器，从而允许它用内存资源的实际存储单元来修补 (patch) DMA 缓冲区。最后，调度程序可以将分页缓冲区（如果它存在）以及 DMA 缓冲区提交给协处理器（以及任何其它辅助硬件）以待处理。

[0064] 如上所述的步骤 1 至 16 能够通过硬件、软件以及上述两者的组合来实现。关于这一点，图 4(A) 和 4(B) 以伪算法的形式从总体上举例说明了图 3 的步骤。图 4(A) 和 4(B) 并不是潜在伪算法步骤的穷举列表，这些步骤可以结合本发明加以实施，而且不应将其解释为图 4(A) 和 4(B) 中的每一个步骤都是实施发明所必须的。而是为了讲授本发明，将图 4(A) 和 4(B) 作为一种示意性的列表。

[0065] 结合图 3 提供的以上论述是对本发明各种实施例的说明。然而，结合如上所述的本发明的实施方案，已经看出了多个进步之处。本说明的其余部分是为了实现各种改进并且克服本发明在实际中可能出现的困难。

[0066] 调度考虑

[0067] 预先定义的一些或全部操作（参看以上步骤 1-16）可能会在 DMA 缓冲区被提交给硬件之前发生。然而，这些操作中的一些操作可能难以执行，直至 DMA 缓冲区被提交给硬件为止。举例来说，内存资源的存储单元可能难以确定，直至 DMA 缓冲区被提交给协处理器以前的瞬间为止。这是因为辅助内存资源会因其在协处理器上运行而随着每个 DMA 缓冲区迁移。

[0068] 由上文的步骤 1-16 包含的操作中的一些操作可能是耗时的，而由此无法在中断时间完成，比如无法在调度程序挑选接下来运行哪个任务之后完成。同样，恰恰由于它们耗时，当协处理器忙于处理其它事情的时候在中央处理单元（“CPU”）上执行它们将是有益的。这是为了使协处理器资源缺乏最小化。协处理器资源缺乏仅指在协处理器没有执行处理功能时花费的时间。响应于这个问题，也许有益于结合所述调度程序使用“工作器线程”。工作器线程能够执行帮助处理一些耗时的设置工作的功能。作为它结合本发明其它进程的操作示例，将工作器线程添加到图 4(B) 的伪算法中。

[0069] 进一步针对此调度考虑，注意，在图 3 的系统中的任何指定时间，可能存在处于以下三种状态下的 DMA 缓冲区：正在运行的 DMA 缓冲区（即，当前正由协处理器进行处理的 DMA 缓冲区）、正在预备的 DMA 缓冲区、以及一列正准备将预备的 DMA 缓冲区。新 DMA 缓冲区一旦提交给调度程序，就能将它插入到就绪队列并且依它们的优先级而作适当地排序。然而，如果新 DMA 缓冲区在被提交给调度程序时无法抢占 (preempt) 已选作为协处理器的下一个任务的 DMA 缓冲区，那么本发明的各种实施例就都可以提高功能性。这个的理由就是：预备 DMA 缓冲区可能会牵扯到把内存资源页入和页出辅助内存。因此，已选定进行处理的下一个 DMA 缓冲区的抢占，可能会导致辅助内存管理器的持续状态的变化。如果正在预备的任务可能会被抢占，那么由于预备了重新选定的 DMA 缓冲区，因而它就会导致撤销对辅助内存管理器的持续状态所作的改变。通过 DMA 缓冲区上的操作中途撤销对辅助内存的改变可能不是微不足道的，而是可能会导致潜在地更频繁的协处理器资源缺乏。

[0070] 分割 DMA 缓冲区

[0071] 当 API 将命令缓冲区提交给协处理器内核时，内核工作方式驱动器继而可以负责生成硬件特定的 DMA 缓冲区，以及生成运行该 DMA 缓冲区所需的一列内存资源。虽然特定的 DMA 缓冲区格式可以由 IHV 来定义，但是软件供应商可能会发现他们本身就肩负着为内核工作方式驱动器定义资源列表格式的任务。

[0072] 内存资源列表能够提供关于不同内存资源的时间安排 (timeline) 信息，这些不同内存资源可由 DMA 缓冲区使用。继而，调度程序又能使用内存资源列表，以便当 DMA 缓冲区在协处理器上运行之前页入 (page in) 任何所需要的内存资源，并且如有必要则对 DMA 缓冲区进行分割，比如当 DMA 缓冲区同时使用太多资源时等等。

[0073] 如果调度程序将对 DMA 缓冲区进行分割，那么内核工作方式驱动器就可以通过在内存资源列表中提供时间安排信息来便于此分割操作。这可以通过允许所述驱动器规定 DMA 缓冲区内的“偏移量”来完成。当正通过插入内存资源标识符来对内存资源进行编排时，可以设定偏移量，所述内存资源标识符规定了在所述偏移量处对内存资源的使用。由于内存资源能够不止一次出现在 DMA 缓冲区中，因而相同的内存资源可能会在内存资源列表中出现多次。在 DMA 缓冲区中，每次对内存资源的引用都将把一个项目附加到资源列表中。

[0074] 实质上，所述那个句柄 / 偏移量 (handle/offset) 列表可能不足以给调度程序充分的信息，所述信息是关于其需要分割 DMA 缓冲区的内存资源的信息。为了精确地了解在 DMA 缓冲区中何时需要特定的内存资源，所述调度程序可能还需要有关何时内存资源被另一个资源替代的信息。举例来说，在第一结构 (texture) 阶段，DMA 缓冲区的起始处可以包含第一结构 (texture) 即结构 A，它被位于中间处的第二结构 (texture) 即结构 B 所替代，然后恢复成缓冲区末尾处的结构 A。调度程序可以使用此附加信息来将 DMA 缓冲区分割成块，这些块将使用较少的内存资源。然而，在上述情形中，也可能已经在第一结构阶段中对结构 B 进行了编排，而在这样情况下，可能在使用结构 A 的同时就已使用了所述结构 B，且不应该将它分成 DMA 缓冲区的独立的子设备。

[0075] 在上述改进方法中，为了实现分割 DMA 缓冲区所需的“细粒状 (finer grain)”临时信息，调度程序可以在整个 DMA 缓冲区中使用关于内存资源使用的信息。在一个实施例中，这可以当内核工作方式驱动器为内存资源列表中的每个项目提供资源标识符时加以实现。简单来说，资源标识符是一个整数值，它表示将要如何使用特定的内存资源。例如，值 0 可表示内存资源正在用作为描绘 (render) 目标，而值 1 可表示资源正在用作为 Z 缓冲区。利用这一信息，调度程序能够确定结构 B 是否正在替代结构 A（例如，如果两者都具有相同的资源标识符），或者确定在使用结构 A 的同时是否将要使用结构 B（例如，A 和 B 具有不同的资源标识符）。资源标识符的实际值及其平均值可以由 IHV 来定义，或者在软件体系结构中提供上述值。这能有助于确保用作资源标识符的值是基于零的，并且为所述驱动器规定其将在驱动器初始化时使用的最大追索 (recourse) 标识符值。

[0076] 图 5 举例说明了调度程序是如何使用所提供的信息来为内存资源定义时间安排的，所述内存资源被用于 DMA 缓冲区中。调度程序可以继续使用所述时间安排来定义缓冲区分割点。也许注意如下问题是很重要的，所述问题就是 :通常，DMA 缓冲区应该从当前内存资源（即，当前位于前一 DMA 缓冲区末尾处的那个内存）的“设置 (setup)”或识别 (identification) 进程开始。这个的理由就是 :内存资源也许已经因执行前一 DMA 缓冲区而迁移，且因此可能需要被重新编程。直到调度 DMA 缓冲区以进行处理之前，都可能需要对

内存资源进行重新编程。

[0077] 正如图 5 中举例说明的那样，内存资源列表可以包含许多个域。下列表提供了非排他性的有效域的列表：

[0078]

Handle	内存资源的句柄（Handle）
ResourceId	选择性地规定将如何使用资源的资源标识符。
Offset	当能对内存资源进行编排时的DMA缓冲区内的偏移量。调度程序能够要求所述驱动器运行DMA缓冲区，直到它因存储限制而需要分割缓冲区的那个时刻为止。因此，此偏移量能够为DMA缓冲区提供有效的分割点。
SegmentHint	规定驱动器意欲用作特定分配的片段，以提供最佳性能。这可以替代为分配而优选的当前驱动器。
BankHint	规定当内核工作方式驱动器能够对配置进行分页时提示片段（hinted segment）内的库（bank）。这可以替代为配置而优选的当前驱动器。
SegmentId	规定持有内存资源的片段的段标识符。这可以在分页期间加以填充。
PhysicalAddress	规定片段内的内存资源的物理地址。这将在分页期间加以填充。

[0079] 分页

[0080] 一般说来，在提交 DMA 缓冲区以待由协处理器执行之前，将 DMA 缓冲区所引用的内存资源纳入内存。将所引用的内存资源纳入到内存的进程，称作资源的分页。分页能够涉及到上述预备工作器线程与诸如内核工作方式驱动器这类驱动器之间的交互。现在参照图 6，该图 6 举例说明了预备工作器线程与辅助内存管理器之间动态性的伪算法。

[0081] 一般而言，当已经选定 DMA 缓冲区以进行处理、并且已经生成特定 DMA 缓冲区的资源列表时，将会出现分页步骤。实施分页是为了确定如何把内存资源获取到辅助内存中，以及确定在辅助内存中的哪些位置放置这些内存资源。

[0082] 分页进程可以由辅助内存管理器来处理。辅助内存管理器能够使用提示 (hint)，一旦创建了特定的配置，就选择性地由内核工作方式驱动器来提供所述提示。创建所述提示是为了在内存中为内存资源找到适当的位置。

[0083] 存在几个与分页内存资源相关的问题。可能没有充足的可用空闲辅助内存来纳入所有的资源，而在上述情况下，会驱逐当前位于内存中的一些资源。即使在驱逐辅助内存中的其它对象之后，对于所述 DMA 缓冲区而言，也可能没有充足的内存。在那种情况下，可以把 DMA 缓冲区分成多个小块，所述多个小块需要很少的内存资源。

[0084] 在分页期间，辅助内存管理器能够构造一列命令，这列命令可以用于将内存资源

置于适当的位置。举例来说，可以根据下列操作来构造此命令列表：

[0085] 1) 驱逐：将特定的内存资源从其当前片段中移出并移动至系统内存，以便为另一个资源腾出空间；

[0086] 2) 页入：将特定的内存资源从系统内存引入到辅助内存中的空闲单元。

[0087] 3) 再定位：将特定的内存资源从一个辅助内存单元移动至另一个辅助内存单元。

[0088] 可以允许辅助内存管理器使用这些操作中的任何一个，以便解决内存布局问题。这是非排他性的。命令列表可以由辅助内存管理器在分页操作期间生成，并且在之后由调度程序使用以生成分页缓冲区。辅助内存管理器能够为以任何方式被再定位、被驱逐或被页入或者反之被迁移或变更的任何内存资源生成命令列表中的项目。在这点上，本发明的各种实施例都可以提供命令列表中的下列域：

[0089]

句柄	要重定位的内存资源的句柄
SegmentId	片段的段标识符，当前内存资源被定位于所述片段中。
PhysAddress	内存资源的当前片段内的当前物理地址。
NewSegmentId	片段的段标识符，可以将资源移动至所述片段。
NewPhysAddress	新片段内的新物理地址，在其中可以移动资源。

[0090] 分页缓冲区生成

[0091] 利用上述命令列表，调度程序可以生成分页缓冲区，以执行所述命令。结合本发明使用的分页缓冲区的各种实施例能够按照图 7 中举例说明的那样加以实现。

[0092] 正如图 7 中举例说明的那样，一些命令可能在它们被执行以前需要进行预处理，而同时可以处理其它命令而无需进行预处理。预处理能以包含在工作器线程中的许多方式来完成。注意，在预处理命令的过程中，也许必须要等待，直到分页缓冲区的一部分已被处理完为止。在按图 7 举例说明的模型中，工作器线程预备分页缓冲区并为该分页缓冲区操控 CPU 预处理。当在分页缓冲区中的操作以前必须进行 CPU 预处理时，工作器线程块就会对协处理器中的分页缓冲区进行操作。继而，它在重新启动分页缓冲区以前，再一次提交 CPU 请求，以完成所述操作。

[0093] 因此，对于命令列表中的每个命令来说，下列动作可能是适当的：

[0094] 在分页缓冲区生成时进行预处理；

[0095] 在分页缓冲区中，在同步点处进行 CPU 处理；

[0096] “Blit”命令，以移动内存资源；

[0097] 一旦完成分页缓冲区，就对 CPU 工作进行后处理。

[0098] 参照上述可能动作的列表，分页缓冲区本身可以包含当 CPU 处理一些工作时请求协处理器停止的命令。这里，将这类产生中断并使协处理器停止的命令称为“块栅栏(fence)”。利用块栅栏，能够将分页缓冲区中的任何命令放在前面或放在后面。因为中断并不是所希望的，所以可以通过把后操作栅栏聚集到缓冲区的末尾，来减少 CPU 可能中断协处理器的次数。调度程序将会检测出在缓冲结束以前需要后操作栅栏（或“后栅栏”）的

情况，并且将其与命令的预操作栅栏（或“预栅栏”）合并在一起，所述命令将需要将被执行的后栅栏。

[0099] 注意，为了维护辅助内存的相于性，不允许在处理分页缓冲区的进程中发生外部中断也许是有利的。因此，如果在分页缓冲区被完全执行以前时间片（quantum）期满，那么就可以允许该分页缓冲区保持处于协处理器的控制下，直到完成它为止。

[0100] 现在参照图 8，该图 8 是表示一连串的事件的伪算法，这一连串事件可能在工作器线程中发生，所述工作器线程包括分页缓冲区中的栅栏处理。结合图 8，以下列表提供了一列归纳后的命令，这列命令可能在命令列表中发生，并且还提供了就预处理而言的那类命令的可能分支、分页缓冲区生成以及可能产生的任何终止栅栏。下列表仅仅是作为有用示例来提供的，而不意味着它是可能命令的类型的排他性列表或是结合那些可能发生的动作的排他性列表。

[0101]

从辅助内存移动到另一个辅助内存单元	**预处理：** 无。 **在分页缓冲区中：**

[0102]

	如果在硬件中将要传递 那么驱动器就能在分页缓冲区中添加blit。 如果将要在软件中传递 那么就清除当前的页面缓冲区。一旦它被清除，就在CPU上继续进行传递。 **在分页缓冲区的终止栅栏中：** 无。
从辅助内存移动到插孔	**预处理：** 附加于拥有正被迁移的辅助内存资源的进程。 MmProbeAndLock系统内存缓冲区，并且获得被锁页面的MDL； 如果MmProbeAndLock页面失败 那么就在软件中处理blit； 从进程上解除附加； 如果当前所分配的插孔位置不忙，而且在当前命令之前没有命令在命令列表中，那么就操纵那个插孔范围（range）。 利用我们产生的MDL来编排插孔。 注意，正在编排插孔。 **在分页缓冲区中：** 如果在预处理阶段中尚未编排插孔。 那么就清除当前的页面缓冲区。清除之后，程序MDL清除到插孔中。继续处理分页缓冲区。 如果在硬件中将要传递 那么驱动器将在分页缓冲区中添加blit。 如果在软件中将要传递； 那么就清除当前的页面缓冲区。在清除之后，利用CPU来传递内存。继续处理分页缓冲区。 **在分页缓冲区的终止栅栏中：** 无。
从插孔移动到辅	**预处理：**

[0103]

助内存	附加于拥有正在迁移的辅助内存资源的进程； MmProbeAndLock系统内存缓冲区并且获得被锁页面的MDL； 如果MmProbeAndLock页面失败 在软件中处理blit； 从进程上解除附加； 如果当前所分配的插孔位置不忙，而且在当前命令以前没有命令在命令列表中， 那么就操作那个插孔范围。 利用我们产生的MDL来编排插孔。 注意，正在编排插孔。 **在分页缓冲区中：** 如果在预处理阶段中尚未编排插孔。 那么就清除当前的页面缓冲区。在清除之后，将MDL编排到插孔中。继续处理分页缓冲区。 如果在硬件中将要传递 驱动器将在分页缓冲区中添加blit。 如果在软件中将要传递； 那么就清除当前的页面缓冲区。在清除之后，利用CPU来传递内存。继续处理分页缓冲区。 **在分页缓冲区的终止栅栏中：** 如果已经由缓冲区中的另一个操作回收（reclaim）了插孔范围。 解除对插孔范围的映射； 从拥有表面的进程上附加； MmUnlock系统内存缓冲区； 从进程上解除附加。
从辅助内存中驱逐	与从视频信号移动到插孔相同的进程。除了在分页缓冲区的终止栅栏上，插孔范围被解除映射。
从插孔中驱逐	**预处理：**

[0104]

	如果插孔范围不忙。 解除对插孔范围的映射； 附加于拥有表面的进程； MmUnlock系统内存缓冲区； 从进程中解除附加。 **在分页缓冲区中：** 无。 **在分页缓冲区的终止栅栏中：** 如果插孔范围尚未被任何前行操作解除映射。 解除对插孔范围的映射； 附加于拥有表面的进程； MmUnlock系统内存缓冲区； 从进程中解除附加。

[0105] 注意，这里给出的调度模型可能需要显著数量的非平凡的 CPU 处理，以保持协处理器忙。此工作至少在某种程度上是现今存在的协处理器硬件的能力所必须的。未来图形硬件可以设计成具有更强的内存虚拟化和协处理器调度。在这点上，已经实现了几项进步，并且也将结合本发明被公开。对于每个硬件能力，我们为所述改进解释一下促动因素以及对上述调度模型的影响。根据具体的实施方法，展现出了确定的改进。注意，虽然在任何未来模型中并不是必须支持所有这些方法，但是如果并且当特定方法被实施时，那么这里如此所述的各种改进就是为了提供将改进采用到实施方法上的基础。

[0106] 可中断硬件

[0107] 为了提高协处理器调度的可靠性，协处理器能够支持细小的粒状中断，而不是整个 DMA 缓冲区的中断。例如，协处理器和支持硬件可以支持三角处理内的中断，而不是仅仅在处理三角之前或处理三角之后的中断。

[0108] 在此类可中断硬件的各种实施例中，优选的设计方法可能就是通过自动地把协处理器环境保存和恢复到辅助内存，来提供协处理器的潜在完全的虚拟化。例如但不限于：每个协处理器环境都可以具有专用地址空间、在其中累积 DMA 缓冲区的专用环形缓冲区、以及当协处理器环境不运行时保存硬件状态的专用内存块。为了支持这些设置中的环境转换，调度程序能够通过存储映像注册来向协处理器提供已保存环境的辅助内存中的物理地址。协处理器于是将那个协处理器环境载入，证实所有内存资源都有效，然后执行已在环形缓冲区中累积的 DMA 缓冲区，而当它们遇到故障时载入需要的资源。

[0109] 结合上文，内核工作方式驱动器还能够查询未运行的协处理器环境的状态。这可以通过检查保存环境、通过使用“运行列表”事件追踪（如下所述）或通过任何查询手段来完成。在这点上所述驱动器能够确定有用的信息，比如 (1) 为何协处理器最近转离特定环境（例如，空、新运行列表、页面错误）的理由；(2) 硬件正在使用的内存资源列表（如果支持表面级故障的话）；(3) 故障地址（如果支持页面级故障的话）；以及 (4) 正已运行特定

环境的协处理器时钟循环次数。

[0110] 此外，内核工作方式驱动器能够把新 DMA 缓冲区插入到当前未运行着的环境的环当中。在已保存的环境中，它还能够修改所述环的位置，存储在该环境中的页表或任何其它的物理内存参考。举例来说，随着内存中那些资源的移动，可能需要此类修改。

[0111] 每协处理器的环境虚拟地址空间

[0112] 上述基本调度模型的某种复杂性是由于如下事实：协处理器环境可能正在共享公共的协处理器地址空间。虚拟化这一地址空间的进程中能够提供较光滑的系统。在虚拟化地址空间的过程中，辅助内存管理器能够来回移动辅助内存，甚至是彻底地从辅助内存当中驱逐资源。那意味着：对于资源而言，实际的协处理器可见的地址可能在其使用期 (lifetime) 期间发生改变。因此，在用户工作方式中构造的命令缓冲区不能直接通过其地址来引用分配 (allocation)，这是因为直到调度所述命令缓冲区以供执行以前，那个地址都有可能是未知的。

[0113] 例如，可以通过每协处理器的环境地址空间的用户来消除上述基本调度模型的下列要素 (element)：

[0114] 1) 通过用实际位置替代句柄 (handle) 来修补命令缓冲区

[0115] 2) 为内存存取证实命令缓冲区

[0116] 3) 在内核工作方式中构造内存资源列表

[0117] 4) 创建独立的命令以及 DMA 缓冲区

[0118] 5) 把被中断的 DMA 缓冲区的资源引回预中断位置

[0119] 在提供每协处理器的环境虚拟地址空间的过程中，特定协处理器环境内的配置能够获得它们自己在那个环境的地址空间内的唯一地址。在配置的使用期期间，不会要求改变地址。因此，命令缓冲区能够直接引用那个地址并且不需要修补。对具备已证实并且已复制到 DMA 存取缓冲区中的命令缓冲区的需要也将会消失。由于 DMA 缓冲区中的内存引用将会在协处理器的虚拟地址空间中发生，而且地址空间实际上将被专用于任何协处理器环境，因而将不必为有效性而证实内存引用，且因此也不必在 DMA 缓冲区中隐藏已证实的命令缓冲区内容，所述 DMA 缓冲区对于应用程序而言是不可见的。硬件可以将未由配置或已驱逐配置占用的地址空间（句柄或物理地址）重定向至假页面 (dummy page) 或者引起引用故障。这将能保护内核工作方式内存的安全性，这是因为所述这些环境将没有机会访问那些它们不应该访问的内存。

[0120] 每协处理器的环境虚拟地址空间的其中一些优点如下：在分配 (allocation) 时，每个分配都将获得协处理器可见的地址（或句柄）。将不存在命令缓冲区；对于用户工作方式驱动器而言，DMA 缓冲区将是直接可见的，并且可由用户工作方式驱动器来填充。DMA 缓冲区将直接引用其使用的分配的地址（或句柄）。用于分页的资源列表将由用户工作方式驱动器构造。

[0121] 回想一下如在图 3 中阐述的本发明各种实施例的模型以及相应的说明。可以利用可中断硬件和 / 或每协处理器的环境虚拟地址空间来进一步地改进这个模型。在这点上，除了本发明的附加进步所进一步改进的地方之外，以下部分描述了与图 3 相类似地基本原理。

[0122] 表面分配以及解除分配

[0123] 在高级模型中，辅助内存管理器、比如处于内核工作方式中的视频内存管理器“VidMm”能够为协处理器环境提供虚拟地址空间，并且能够管理各种协处理器环境中间的物理内存，以便它们能够获得应得的那份内存。图 9 中描绘出了在基本模型的分配方案上的这一改进的各种实施例。图 9 举例说明了使用本领域技术人员所熟知的术语的本发明实施例，因为它符合于本技术领域中所公认的概念。举例来说，“VidMm”为视频内存管理器，而“Thunk 接口”是形实转换程序接口。然而，请注意，尽管这个术语是用来更清楚解释本发明的，但是不应该将它理解成是意图限制本发明的体现。因此，“VidMm”可以是任何辅助内存的内存管理器，而“Thunk 接口”可以是任何适当的接口等等。

[0124] 结合图 9，所述高级模型允许 DMA 缓冲区被直接映射到应用程序的地址空间中，这选择性地使它们可直接由用户工作方式驱动器访问。用户工作方式驱动器利用它需要访问的每个内存资源的永久性虚拟地址或句柄 (handle)，将描绘描绘原语 (rendering primitive) 直接批处理到 DMA 缓冲区（所以不需要进行修补）。用户工作方式驱动器还构造一列 DMA 缓冲区正在使用的内存资源，以便辅助内存管理器能够在 DMA 缓冲区被调度之前将那些内存资源引入到辅助内存中。如果恶意应用程序修改了资源列表，那么将不能恰当地把正确的资源组页入。注意，这未必会中断存储保护模型，这是因为可能需要未访问有效内存的地址空间范围访问假存储页面或者令硬件产生故障并停止执行特定协处理器环境。不论发生那一种情况，破坏的资源列表都不必产生能够访问另一个环境的内存的协处理器环境。

[0125] 在高级模型中，用户工作方式驱动器将 DMA 缓冲区提交给内核工作方式驱动器，该内核工作方式驱动器又将 DMA 缓冲区提交给调度程序。在请求内存管理器将资源页入至资源列表中后，调度程序按现状将 DMA 缓冲区发送到硬件。

[0126] 高级模型中的调度

[0127] 高级模型中的调度非常类似于基本模型中的调度。在 DMA 缓冲区被提交给协处理器以前，仍然存在有预备 DMA 缓冲区的工作器线程。然而，在高级模型中，附有工作器线程的工作仅需限制于分页操作。

[0128] 现在参照图 10 和图 11，该图 10 和图 11 是基本模型中的调度以及高级模型中的调度的实施例。正如将变得显而易见的是，高级模型具有两个调度选择。当在没有需求故障的情况下进行调度时，能够实现预备阶段。然而，当高级模型使用需求故障时，就不必实现预备阶段了。

[0129] 此外，图 12(A)、12(B) 并 12(C) 提供了能实现高级调度模型的示范性伪代码的流程图。

[0130] 页入高级模型

[0131] 页入高级模型不同于页入基本模型。在高级模型中，早已知道正在分页的分配地址，而内存管理器仅仅需要使其有效。为了让资源列表中的分配有效，内存管理器需要发现大范围的空闲的物理辅助内存，并且请求所述驱动器将页表或句柄映射到那个范围中。如有必要，物理内存的范围可能就需要是一组连续页面。

[0132] 如果没有足够的物理视频内存可以用来使分配有效，那么这里称为 VidMm 的辅助内存管理器就会使某个当前的有效分配被驱逐出去。当分配被驱逐时，其内容被转入系统内存（假定它尚不在系统内存中），然后其使虚拟地址或句柄无效。

[0133] 虚拟地址空间

[0134] 结合本发明可以使用本技术领域中的或者未来将开发的任何用于提供虚拟地址空间的已知技术。为了示范说明在其中能使用此类地址空间的方法，在这里提供了使用公共虚拟地址空间技术的两个示例。应当理解的是，存在多种方法来为协处理器创建虚拟地址空间，而且本领域的技术人员将能从这里所提供的示例中推导出来。在这点上，在这里描述了使用可变长度平面页表和多级页表的虚拟地址空间。

[0135] 可变长度平面页表。在图 13 中举例说明了本发明结合可变长度平面页表的使用。在此方法中，通过使用平面页表来将协处理器的地址空间虚拟化。虚拟地址空间能够被分成预定内存数量的页面，比如 4KB。对于虚拟地址空间中的每一个页面而言，提供包含标识符的页表，比如 64 位项目，以用来规定关联的物理内存的物理地址及位置（例如，加速图形接口 (AGP)、外设部件互连 (PCI) 或视频）。在一个实施例中，协处理器所支持的页面大小不是任意的，而必须为 4KB，这是为了允许协处理器页表能够访问系统内存页面。此外，在这个实施例中，协处理器页表必须既能够寻址来自于同一地址空间的本地视频内存和系统内存。协处理器可以要求所有属于单个表面的页面都被映射到单一类型的内存中。例如，协处理器能够要求所有属于特定描绘描绘目标的页面被映射到本地视频内存中。然而，将表面映射到多种物理内存类型 (AGP、本地视频等等）的页表项目能够共存于所述页表当中。

[0136] 对于 PCI 和 AGP 适配器而言，每个页表项目的示范性实施例都可以包含 32 位，其允许完全 4GB 的物理地址空间都能由协处理器可见。对于使用 PCI 专用类型的适配器的实施例而言，协处理器可以支持 64 位寻址周期。每个页表项目都可以包含 40 位或更多位，以寻址内存的每个千兆字节。如果相应的视频适配器不能寻址整个地址空间的话，那么实现与主板毗连的 64 位系统的实施例就可能会经受性能损失 (performance penalty)，所述 64 位系统使用了多于 40 位的物理地址。因此，建议它支持全 64 位。

[0137] 除了没有页面目录之外，平面页表方法类似于INTEL® 8086(x86) 族 CPU 上现用的虚拟化机制，仅有巨大的页表。

[0138] 与有效分配无关联的虚拟地址能够被重定向到假页面，以防止让恶意 DMA 缓冲区强制协处理器存取它不该存取的内存。所述硬件能够实现每一个页表项目中的有效位，该有效位规定了所述项目是否有效。

[0139] 当关联的协处理器环境当前未在协处理器上运行时，页表可以是可重定位的。当所述环境未运行时，VidMm 能够将页表驱逐到系统内存。当所述环境即将再次运行时，可将页表引回到视频内存，但却是在可能不同的位置。所述驱动器也许能更新保存协处理器环境中页表的位置。

[0140] 在这个实施例中，所有的内存存取都可能经由协处理器虚拟地址发生。然而，它不应该暗指本发明要求这类存取。可以以其它方法来存取某些要素，如果以其它方法来存取的话，甚至可以提供增强的功能。可能从虚拟地址方案当中被舍掉的项目的一些示例是：

[0141] 1) 经由物理地址可以访问页表本身。

[0142] 2) 阴极射线管 (CRT) 可以被编排成连续内存范围的物理地址。

[0143] 3) 虚拟打印引擎 (VPE) 能够直接对物理地址执行 DMA。

[0144] 4) 重复占位程序段 (Overlay) 能够从物理地址上直接读取。

[0145] 5) 能够经由物理地址来引用协处理器环境。

[0146] 6) 能够经由物理地址来引用主环形缓冲区。

[0147] 注意，在环境转换期间，协处理器能够对正被恢复的环境正使用的虚拟地址进行重译。这将确保内存资源位于适当位置，而不是允许协处理器产生潜在地错误假设，即当在环境转换之前时那些地址正在访问相同的物理页面。还应注意的是，结合本发明的各种实施例，将会有益于允许单个页表中或是跨过多个页表的多个项目访问同一个物理页面。

[0148] 在各种实施例中，协处理器可以实现给定页表当前尺寸的界限寄存器。经过页表末尾存取的任何内存都能被视为协处理器的无效存取并按此予以对待。页表可以按 2 的幂加以扩大，并且在一个实施例中，能够支持至少 2GB 的地址空间 (2MB 的页表)。

[0149] 如果与协处理器环境相关联的虚拟地址空间变得支离破碎，那么 API，例如 MICROSOFT® Direct3D Runtime，就能执行垃圾回收，以减少地址空间以及相关页表的尺寸。在高位虚拟地址的分配将被删除并再分配到较低位的地址。

[0150] 结合本发明使用可变长度平面页表来实现虚拟地址空间的优点和缺点，应当对本领域的技术人员来说是显而易见的。综上所述，使用平面页表的一个优点就是 :仅仅存在对物理内存的一级间接寻址。另一个优点是 :可以利用一组不连续的页面来解决分页。然而，也存在缺点。例如，当协处理器正在运行时，整个页表通常都将需要存在于内存中。此外，页表还会耗费大量的内存。可能难以确定页表，因为它通常需要内存中的一组连续页面。

[0151] 多级页表。在图 14 中举例说明了本发明结合多级页表的使用。多级页表通常可类似于可变长度平面页表，然而在多级页表中，虚拟地址的变址部分跨过多个表而分裂。例如，各种实施例都可以使用 32 位地址空间。在这种情况下，可以要求硬件具有二级间接寻址。间接寻址的第一级称为页面目录，而第二级称为页表。当协处理器正在运行特定环境时，仅仅是在资源列表中分配所需的环境和页表的页面目录必须存在于内存中。

[0152] 将会认识到的是，提供结合本发明的多级页表的一个优点就是 :能够利用一组不连续的页面来解决分页。同样，分配能够将来自于系统视频内存和本地视频内存的页面混合，只有使用中的页面目录和页表需要存在于内存中，并且页面目录和页表中的每一个都仅仅需要一个页面 (不需要多个页面邻接分配)。然而，尽管存在这些优点，但是仍存在存取内存需要两次间接寻址的缺点。

[0153] 表面级故障

[0154] 在具有额外的每协处理器的环境虚拟地址空间的情况下，高级调度模型就能相当好地进行工作，并且通常不需要更多的 CPU 开销，特别是当有一点或无内存压力时。时常，当 DMA 缓冲区能够被提交给调度程序时，其访问的资源早已存在于内存中，并且由此 DMA 缓冲区不需要由分页线程进行任何分页。然而，就调度而言，能够进一步地通过提高计时 (time keeping) 的准确度来改进该模型。

[0155] 在实现本发明的过程中所遇到的一个问题就是 :它也许不能预先知道特定 DMA 缓冲区将花费多久时间来执行。这可能会导致调度程序为下一个 DMA 缓冲区预备时的潜在低劣选择。如果没有别的环境处于与当前环境相同的优先级或较高优先级，或者如果处于那个优先级的所有其它环境都为空，那么调度程序就可以从当前环境中挑选下一个 DMA 缓冲区。否则，调度程序就可以从具有与当前环境相同优先级的或较高优先级的下一个环境中挑选下一个 DMA 缓冲区。然而，那个选择不能保证是准确的。当从下一个较高优先级环境中选择 DMA 缓冲区时，调度程序可以假定当前环境的 DMA 缓冲区将运行比一个时间片更长

的时间。如果不是这种情况的话，调度程序就可能会过快地转离那个硬件环境。在当前 DMA 缓冲区运行时间少于一个时间片的情况下，调度程序应该已经从当前环境中选择了下一个 DMA 缓冲区（因为这将已把协处理器的有效利用最大化了）。

[0156] 当有一点或无内存压力时，一般说来，对下一个 DMA 缓冲区的两种可能候选早就可以让它们的资源存在于内存中了，所以缓冲区很可能也不需要分页。在那种情况下，当第一 DMA 缓冲区的时间片结束时，调度程序能够意识到它的错误，立即改变它的想法，并将正确的 DMA 缓冲区给予协处理器。

[0157] 然而，在内存压力下，该模型可能变得不太稳定。在确保平稳运转的进程中，下一个 DMA 缓冲区的“尺寸调节 (sizing)”可能变成了一个优选的步骤。在内存压力下，对于需要某种分页且由此而被发送给预备线程的下一个 DMA 缓冲区两种可能候选的而言，在先前描述情形下的几率是两种可能候选的其中一种。在那种情况下，对于调度程序而言，在最后一刻“改变其想法”并且交换两个 DMA 缓冲区的作法通常是不可取的。然而，请注意，可以作出这类改变，并且这种实施不超出本发明的说明书范围。例如，在完成预备下一个 DMA 缓冲区并且其它可能的 DMA 缓冲区候选不需要分页的情况下，就可以交换 DMA 缓冲区。这可能暗指可由辅助内存管理器共享的分配的某种特定支持，但是它是明显可能的。

[0158] 自然而然的是，上述潜在的计时误差不是非常坏的，并且可能通过给定环境来绕过它在连续时间片期间遗漏的处理时间。同样，在大多数情况下，DMA 缓冲区包含足够的命令以供多个协处理器时间片运行用，所以每个环境都可以获得其完全的时间片。然而，在内存压力下，辅助内存管理器可能要被迫将 DMA 缓冲区（如上所述）分割成更小的缓冲区，以便于减少每个环境的工作区。这种 DMA 缓冲区的分割减少了 DMA 缓冲区的尺寸，并且相应地增加了上述的量化问题。

[0159] 在内存压力下，可能出现的另一个问题是：所述模型可以人工地引起额外压力，这是因为潜在地存在比实际由 DMA 缓冲区使用的更多的获得页入的内存。所有被页入的额外内存都将可能在下一个时间片到来以前遭到驱逐，并且将需要再一次被页入。当分页活动早已很高时，这可能会导致增多的分页活动。在基本模型和高级模型中，辅助内存管理器可以通过选择适当的驱逐策略来面对增多分页的问题。例如，在轻的内存压力下，每个环境在其工作区中很可能都具有适当数量的内存。在从其它的环境中驱逐内存以前，辅助内存管理器可以试图从第一当前环境中驱逐内存，并分割其 DMA 缓冲区，以便使其在可用工作区中大小适宜。一旦特定环境的 DMA 缓冲区正被分割成其最小尺寸，就让辅助内存管理器不作选择而是从另一个环境中驱逐内存。

[0160] 解决这个问题的一个优选的方法就是，允许协处理器所需的内存的需求故障。那样，我们就能够确保只有协处理器所需的内存子设备存在于内存中。

[0161] 为高级模型而建议的故障级是以表面粒度为准的。然而，应当理解的是，任何故障级都可适合结合本发明的使用。同样，还要注意到：由于辅助内存管理器会在内存中同时产生整体分配，因而就页表硬件来说，硬件只得查看分配的第一页面的状态，以便判断分配是否有效。

[0162] 在各种实施例中，当以下几种情况出现时，硬件可能会产生页面错误：

[0163] 1) 到访问无效环形缓冲区或 DMA 缓冲区的环境转换发生。

[0164] 2) 即将形成的原语以及一些所需的内存资源不存在（例如，顶 shader 代码、顶缓

冲区、结构 (texture))。

[0165] 注意，在第二种情况下，可能需要硬件在描绘描绘每个三角关系以前重采样其当前的内存资源。辅助内存管理器将可以随时地使虚拟地址或句柄无效，包括当协处理器正在运行的时候。同时，也期望硬件可以允许查询所有它当前正在使用的内存资源。辅助内存管理器可以使用那个信息来判断何时特定分配可以被硬件使用。辅助内存管理器可以假定：如果分配在使其虚拟地址或句柄无效之后、未出现在当前正由协处理器使用的资源的列表中，那么它必定驱逐那个分配，这是因为协处理器无法访问那个分配。试图这样做会引起页面错误。

[0166] 通过以下对表面级故障模型的更详述解释，进一步解释了对结合本发明的表面级故障的使用。下列模型只是一些实施例的示例，而不应该把它视作是对本发明的可能使用的限制或是对结合这里所提供的调度模型环境以外的其它应用程序的表面级故障概念的使用的限制。

[0167] 首先，内存资源的分配方案可以与在本篇的每协处理器的环境虚拟地址空间部分中描述的分配方案相同。可以参看那个部分的详细描述。

[0168] 其次，DMA 缓冲区和资源列表的描绘描绘命令方案也与在本篇的每协处理器的环境虚拟地址空间部分中解释的描绘描绘命令方案相同。在这个模型中，即使图形硬件支持表面级故障，但也仍需要资源列表。辅助内存管理器（在此是“VidMm”）利用资源列表来获得关于内存分配的使用信息。当需要在内存中腾出空间时，此使用信息允许 VidMm 为驱逐而确定候选者。

[0169] 在具有额外的表面级故障的情况下，不存在有关资源列表的安全性关系，所以就能够以用户工作方式来构造它。如果恶意应用程序将无效数据放入到资源列表中，那么可能发生的最坏情况就是将遭受恶意应用程序的运行。VidMm 可能会为驱逐而提出关于候选者的不合逻辑的选择，这将会为那个应用程序带来额外的分页活动。

[0170] 带有表面的需求故障的调度模型可能在许多方面不同于那些未使用表面级故障的模型。一般说来，可以将就绪表中的进程直接提交给协处理器，而不需要预备阶段。调度程序可以为那些要求此页面错误被解决的环境维护专用列表以及分页线程。存在有被用来分页操作的特定 VidMm 的协处理器环境。最后，使提交给环境的 DMA 缓冲区连接起来，以形成单个工作项目。

[0171] 在这个模型中，可以消除预备阶段。调度程序可以请求协处理器直接从一个环境切换到另一个环境，并且假定所有环境都已就绪，以待随时地执行。如果正在转入的环境没有让其所有的内存资源都存在于内存中，那么硬件就可能发生故障，并且该环境将被加到列表（比如，页入列表，参看图 15）中，以便分页线程能够开始解决所述故障。

[0172] 在图 15 中举例说明了结合这个模型的由调度程序维护的一列示范性进程。现在参照图 15，当故障发生时，就能够将引起该故障的环境添加到页入列表中。然后，分页线程解决该故障。分页线程能够选择出现故障的最高优先级的环境，以便首先加以解决。可以使用循环优先级推进来确保低优先环境将最终能获得足够高的优先级，以便让它们的故障得以解决。当页入工作器线程正在解决这些故障时，调度程序可以调度更多待在协处理器上执行的环境。当协处理器正在工作时，页入工作器线程能够通过调用驱动器从地址处映射或解除映射分配，来操纵视频内存。

[0173] 可以让当前协处理器正在使用的分配变为无效。下次协处理器试图访问这种分配时，它将会发生故障。然而，由于协处理器不能在任意时间立即发生故障（例如，一些协处理器将仅对三角关系间的当前分配的状态进行重采样），因而存在这样的可能，即协处理器将在它已经被无效掉之后需要一段时间来使用分配。

[0174] 为了防止这种情况发生，VidMm 可以确保用于分配的内存保持有效，直到下一个环境转换为止，即使它的虚拟地址或句柄已经被无效了。这可以通过因在专用 VidMm 协处理器环境中作了分页而进行清除来实现。由于清除是在独立的环境中完成的，因而我们能够肯定的是：将在内存内容被改变之前存在环境转换。对于访问系统内存的虚拟地址或句柄而言，在驱逐期间没有清除。在那种情况下，VidMm 可以通过保持它受约束 (pinned down) 而确保系统内存保持有效，直到协处理器环境转换到 VidMm 的专用环境。

[0175] 专用 VidMm 的协处理器环境是常规的协处理器环境，它被 VidMm 使用以完成系统内存与视频内存之间的清除。VidMm 环境是可变优先级的环境，它在页入列表中持有最高优先级项目的优先级。在单一环境中拥有串行化后的所有分页操作，简化了 VidMm 的同步模型 (synchronization model)。

[0176] 在这个模型中，另一个引起关注的差异就是：所有为特定环境提交的 DMA 缓冲区都能够被连接起来以形式单个任务的方式。在上述模型中，形成工作项目的每个 DMA 缓冲区以及每个环境都将维护那些工作项目的列表。调度程序将不必调度该环境；它将为与环境相关联的特定工作项目进行调度（并启动预备）。在那个工作项目有机会完成之前，该调度程序将已经选择了下一个工作项目。每个工作项目都不得不在它被提交之前加以预备，这样调度程序就必须预先知道下一个工作项目应该是什么，这并不总是可能的。

[0177] 利用表面级故障，DMA 缓冲区不需要预备。由于这个原因，调度程序就不必把环境看作为许多工作项目。反之，调度程序真实地调度环境，并且一旦环境获得协处理器的控制，它就能够保持协处理器的控制。可以允许一些事件来停止处理器的环境控制，例如：

[0178] 1) 协处理器结束当前已排队的所有命令

[0179] 2) 协处理器产生由无效的内存存取所引起的页面错误

[0180] 3) 调度程序请求转换到不同的环境

[0181] 4) 协处理器按照 DMA 流中的无效命令而产生无效操作中断。

[0182] 图 16 提供了举例说明依照上述的本发明的各种实施例的图表。现在参照图 16，在同一个硬件环境中，两侧均表示从第一环境的介入到第二环境的介入的连续进阶 (progression)。在左手一侧，调度程序请求内核驱动程序将特定 DMA 缓冲区插入到协处理器环境 #1 的环中。驱动器修改所述环，并且更新协处理器，以引用新的位置。在协处理器环境 #1 中 DMA 缓冲区的插入发生在特定协处理器环境 #1 的加锁的保护下。由此，其它的线程都能够将 DMA 缓冲区插入到其它协处理器环境的环中。

[0183] 在右手一侧，调度程序请求内核工作方式驱动器将特定 DMA 缓冲区插入到协处理器环境 #2 的环中。然而，所述环已经满了，由此线程 B 将被加锁，直到在此环中空闲出一些空间。注意这样的事实，即：正在等待的线程 B 没有阻止线程 A 在其拥有的环中插入新 DMA 缓冲区。

[0184] 在这个模型中，每个环境都有其自己的 DMA 环，所述 DMA 环可以包含对部分 DMA 缓冲区的重定向，以待执行。在提交时，调度程序可以试图将已提交的 DMA 缓冲区添加到那个

环境的环中。如果所述环已经满了，那么调度程序就可以等待，直到在此环中有足够的空间以供另一次提交。注意，此等待将仅仅锁定对正提交给的特定环境的进一步提交。它不会锁定对其它环境的提交。换言之，多个线程都能并行地将工作项目添加到它们自己的环境中。

[0185] 由于新 DMA 缓冲区能够被添加到正运行着的环境的队列中，因而协处理器可以在产生中断之前对队列尾进行重采样，所述中断用于报告一个环境为空。当然，能够紧接着协处理器对其队列采样之后就将 DMA 缓冲区添加到队列。然而，恰好在产生中断之前采样所述队列尾，减少了这种情况发生的可能性，同时增加了调度的准确性。当通知调度程序环境为空时，它将查询驱动器，以便看看是否真是这种情况。对于驱动器来说，为了判断当前在其中是否存在尚未被处理的队列命令而存取已保存的协处理器环境，应当是可能的。图 17 提供了描述这种模型的伪代码算法。

[0186] 正如将在后面详述的那样，介绍受限 - 特权 DMA 缓冲区的概念，以便当允许内核工作方式驱动器构造包含特权命令的 DMA 缓冲区的同时，能允许以用户工作方式直接构造 DMA 缓冲区，而无需兼顾系统安全性。

[0187] 由这种模型来表示的各种实施例可以结合受限 - 特权内存的概念来使用，所述受限 - 特权内存将在本篇后面的部分中作出描述。眼下要注意的是 :在这种模型中可能会出现问题，这是因为在先前所展示的内存虚拟化模型中，未能在由受限 DMA 缓冲区 - 特权 DMA 缓冲区存取的内存之间作出区分 ;所有的虚拟内存都是可存取的。这意味着，一些内存资源，比如像页表或环形缓冲区，也许经由协处理器虚拟地址空间而被不适当地显现出来了，因此这会允许恶意应用程序改写页表或环形缓冲区。由于这个原因，可以将硬件设计成能支持一些类资源的物理地址以及其它类资源的虚拟寻址。

[0188] 解决这个问题的一个不同方法就是加入特权内存的概念。在各种实施例中，只能从特权 DMA 缓冲区来访问特权内存，并且如果受限 DMA 缓冲区试图访问特权内存单元时，协处理器会引起页面错误。从另一方面来说，特权 DMA 缓冲区能够不加区别地访问特权内存以及无特权内存。为了支持特权内存，硬件必须具有这样一种机制，即 :根据每句柄 (per-handle) 来规定（在基于句柄的虚拟化的情况下）或者根据每页面来规定（在基于页表的虚拟化的情况下）所述内存是否是有特权的。

[0189] 注意，为了支持特权内存，支持带有页表的表面级故障的协处理器不再能仅仅根据内存资源的基址来发生故障。协处理器必须查看由当前资源覆盖的所有页表项目，并且确定所有这些页表项目都具有正确的保护位设置。仅仅检查内存资源的第一个页面，将可能允许恶意应用程序按照受限内存基址来访问特权内存，所述基址是在受限 DMA 缓冲区中规定的。

[0190] 运行列表

[0191] 先前所展示的需求故障模型可能难以使用中断来发信号通知多次事件。这些事件中的一些，例如像页面错误，能够在内存压力下高频发生。在命中中断时与 CPU 给协处理器新任务时之间，协处理器可能是处于资源缺乏的。为了隐藏中断等待时间并保持协处理器忙，我们介绍一下运行列表的概念。

[0192] 运行列表只不过是一列能够由协处理器运行而无需 CPU 介入的协处理器环境。环境可以按照指定的次序运行，或者按照能为实施本发明的那些人证明其便利性的任何其它

命令运行。协处理器能够因多种原因中的任何一种原因而从一个环境转换到运行列表上的下一个环境，所述环境是能够结合本发明实施的环境，例如：

[0193] 1）当前环境为空，也就是没有任何事需要作

[0194] 2）当前环境产生页面故障

[0195] 3）当前环境产生一般性保护故障（如果由协处理器支持的话））

[0196] 4）协处理器被要求转换到新运行列表

[0197] 在各种实施例中，当协处理器从运行列表中的一个项目转换为下一个项目时，它中断 CPU 但不停止，并且可以将环境转换到列表中的下一个项目，且开始执行它。运行列表的首部可以是调度程序能够试图首先运行的环境，并且运行列表的其它元素都可以分几部分位于此处，以便在中断等待时间期间保持协处理器忙。一旦 CPU 接收到了协处理器已转离列表首部的中断信号，CPU 就可以构造新运行列表并将它发送到协处理器。

[0198] 当协处理器转离列表首部时，它可以开始执行运行列表中的下一个环境，与此同时中断它使用 CPU 的路径。CPU 将产生的新运行列表的首部可以不同于协处理器恰好转入的环境。在那种情况下，协处理器将需要再次进行转换，并且也许没有时间在完成更多有关那个环境的工作。

[0199] 然而，由 CPU 构造的新运行列表的首部环境可以是与上述运行列表的第二元素相同的环境，这是因为自从上次运行列表被构造以来环境优先级不会被改变。在那种情况下，协处理器将已提早开始处理正确的环境。

[0200] 在图 18 中提供了表示运行列表概念的图。当运行列表被归入本发明的各种实施例中时，调度程序的运行环境可以由当前的运行列表所替代。将称为待定运行列表的第二运行列表引入，以便简化运行列表转换的同步。当前的运行列表是调度程序能够假定当前正在执行硬件的环境的列表，而待定运行列表是当调度程序想要将硬件从一个运行列表改变为另一个时所使用的过渡运行列表。当调度程序想要转换为新运行列表时，它构造待定运行列表并且请求协处理器转换为它。一旦调度程序从协处理器接收（通过中断）到了证实，所述协处理器就开始执行新运行列表，待定运行列表变成新的当前运行列表，而且待定运行列表可能为空。

[0201] 当待定运行列表为空时，硬件可能正在执行当前运行列表中的环境或者也许处于空闲。当待定运行列表不为空时，调度程序可能不知道硬件当前正在执行哪个运行列表，直到它从发生变化的协处理器那里接收到证实为止。

[0202] 特定的事件可能需要调度程序对运行列表重定优先级。例如，页面错误也许已通过使高优先级协处理器环境待执行而被解决。为了简化这类事件的同步，调度程序可能遵从的一般规则就是：只有当已经没有由上述事件提交的待定运行列表时，它才将提交新运行列表（待定运行列表）。试图用另一个待定列表来替换一个待定列表可能难以同步，这是由于所述列表已经被交付给协处理器，由此而随时可能发生变化，并且只是在上述事实情况发生以后才会通知调度程序。

[0203] 在后者的情况下，运行列表的重定优先级可以委托给环境转换处理机来完成。在未来的一些时刻上，可接着调用所述处理机来通知从待定列表到运行列表的变化，并且如果优先级已经改变的话，则在那时处理机就能够产生新运行列表以发送到硬件。

[0204] 运行列表转换同步。在一个运行列表模型中，图形硬件当它转换环境时能产生中

断。由于中断交付及处理不是瞬时的，因而可能会在 CPU 实际获得中断以前产生多次中断。如果没有恰当地实现同步，那么调度程序可能会受到干扰，继而就会作出不正确的调度决定。

[0205] 调度程序可能需要区别的两个关键事件就是：第一，当协处理器从运行列表的首部转离时，以及第二，当协处理器转变为待定运行列表时。仅利用在每个环境转换时的来自于简单中断的信息来把那些事件区分开，可能是很困难的。为了进一步举例说明这点，考虑下列示例：当前协处理器正在运行运行列表 A，该运行列表 A 由环境 1-3-5-2 组成，并且调度程序要转变为运行列表 B，该运行列表 B 由环境 4-1-3-2 组成。下列两种情况可能会发生：

[0206] 情况 #1

[0207] 当前协处理器正在执行运行列表 A(1-3-5-2)。

[0208] 考虑环境 4 来提交命令，该环境 4 是空闲的并且具有比环境 1 更高的优先级。产生运行列表 B(4-1-3-2) 并且调度程序将运行列表 B 提交给协处理器。

[0209] 环境 #1 运行，直到协处理器变化为运行列表 B 中的环境 #4 为止。

[0210] 协处理器产生通知所述变化的中断。

[0211] 协处理器从环境 #4 变化到环境 #1，继而在 CPU 被中断以前变化为环境 #3。

[0212] CPU 被中断，并且调用环境转换处理机。

[0213] 驱动器对当前的协处理器环境进行采样，它为环境 #3。

[0214] 情况 #2

[0215] 当前协处理器正在执行运行列表 A(1-3-5-2)。

[0216] 考虑环境 4 来提交命令，该环境 4 是空闲的并且具有比环境 1 更高的优先级。调度程序将运行列表 B 提交给协处理器。

[0217] 当调度程序正在忙于构造运行列表 B 的时候，协处理器变化到环境 #3。

[0218] 协处理器产生通知已变化到环境 #3 的中断。

[0219] CPU 被中断，并且调用环境转换处理机。

[0220] 驱动器对当前的协处理器环境进行采样，它是环境 #3。

[0221] 在上述两种情况下，在环境转换中断时，当前正在运行的环境是环境 #3。然而，注意，没有附加信息的话，调度程序就无法在上述两种情况之间进行区别。在第一种情况下，协处理器从运行列表 B 的首部转离，并且由此调度程序需要产生运行列表 C 并且请求协处理器改变成它。然而，在第二种情况下，甚至第二运行列表还仍未开始，并且由此调度程序将会等待。

[0222] 以上示例表明：单独的环境转换中断可能不足以恰当地支持调度模型中的运行列表。需要更多信息来区分开上述情况。下一部分详细描述一下：可以连同硬件支持一起来解决此问题的一些方式，所述硬件支持是在解决此类问题的进程中可能有用的支持。

[0223] 二元素运行列表。这种同步方法需要协处理器支持一些附加特征。在这些特征当中，结合二元素运行列表的实施方案能够支持的特征如下：

[0224] 1) 二元素的运行列表。

[0225] 2) 具有在每次环境转换时产生中断的能力（包括从环境 X 到 X 的伪造环境转换）。

[0226] 3) 使 VidMm 随时查询当前正在运行的协处理器环境的方法。

[0227] 4) 在中断以前，将待发 (outgoing) 协处理器环境保存到内存。

[0228] 5) 以环境由 CPU 可读的方式来保存协处理器环境，以便允许调度程序确定环境转换幕后的原因。

[0229] 注意，尽管硬件可以用来支持上述功能，但是这种专用设备不是为允许调度程序区分正规环境转换与运行列表所必需的。而是，调度程序可以通过当构造运行列表时总是注意区分那两个事件来完成。尽管对于本发明的各种实施例而言，具体规则可能会变，但是提供这些功能的示范性规则就是：第一，无法出现在新的待定运行列表中的当前运行列表的第一环境，以及第二，如果当前运行列表的第二环境不是新的待定运行列表的首部，那么它就根本不准处于新的待定运行列表中。以下是假设的表，该表是当这两个示范性规则如下时、调度程序在从一个环境变化到另一个环境期间可能会产生的表。在以下的表中，运行列表 A 由环境 1-2 组成；第二运行列表 B 由环境 2-3 组成；而第三运行列表 C 由环境 3-4 组成。

[0230]

从A到B变化	
当CPU被中断时的当前环境#	采取的手段/动作
1	假信号，忽略此中断。 这种假信号是由上述运行列表（X,1）到（1,Y）的切换而引起的，其中当运行列表转换时我们不正确地将X翻译为 1 次变化。真实变化是X到 1，继而是 1 到 1。当前的中断是对于 1 到 1 变化的，而且能够被忽略（需要协处理器产生这种中断，以便能由检测出从 1 - X到 1 - Y的变化）。
2	运行列表发生转换。 这并不总是真的，并可能会导致上述假信号。如果当前的变化真的是 1 - 2，那么CPU就将被再次为 2 - 2 或者 2 - 3 的变化而中断。待定运行列表（B）变成当前运行列表，而待定运行列表是空的。调度程序需要处理转离环境 1（例如，页面错误）的环境转换。
3	运行列表发生转换，并且第二列表的首部已经完成。 运行列表B结束。待定运行列表（B）变为当前运行列表。调度程序构造新的待定运行列表并将其送到协处理器。调度程序需要处理远离环境 1 和 2（例如，页面错误）的环境转换。

[0231]

从A到C的变化	
当CPU被中断时的当前环境#	采用的手段/动作
1	假信号，忽略此中断。 这种假信号是由上述运行列表（X,1）到（1,Y）的切换而引起的，其中当运行列表转换时我们不正确地将
	X翻译为 1 次变化。真实变化是X到 1，继而是 1 到 1。当前的中断对于 1 到 1 变化的，而且能够被忽略（需要协处理器产生这种中断，以便能由检测出从 1 - X到 1 - Y的变化）。
2	当前运行列表中的环境转换。 协处理器转换到环境 2。调度程序需要处理远离环境 1（例如，页面错误）的环境转换，但是关于运行列表无事可作。
3	运行列表发生转换。 待定运行列表（C）变成当前运行列表，而待定运行列表是空的。调度程序需要处理转离环境 1（例如，页面错误）的环境转换。环境 2 是否曾经被执行过是未知的，它将被重新调度。
4	运行列表发生转换，且第二列表的首部已经完成。 运行列表C结束。待定运行列表（C）变为当前运行列表（尽管硬件是空闲的）。调度程序构造新的待定运行列表并将其送到协处理器。环境 2 是否曾经被执行过是未知的，它将被重新调度。

[0232] 实现运行列表的这个方法或许是最简单的，且不必需要效果重要的附加硬件支持。然而，注意，上述表中的运行列表在尺寸方面受限制（扩展超过二倍尺寸的时候可能会变得不实际），并且在环境转换期间可能会丢失一些非关键的信息。例如，从 A 到 C 的变化过程中，调度程序也许不总是知道环境 #2 是否曾被执行过。它可能已被执行过了，引起了页面错误，但是使其中断被另一个环境转换所隐瞒了。在那种情况下，调度程序不会知道曾经产生过故障并且将重新调度它。

[0233] 调度事件的协处理器追踪。当硬件向调度程序提供一些调度事件的历史信息时，运行列表可能易于被扩大 N 倍的尺寸。随着简单中断而来的一个问题就是：多次中断可能

被挤在一起，并且它也许不能精确地确定发生了什么而引起中断。可以结合本发明的方法，利用硬件特征来解决这个问题。可通过实施能将环境转换历史写入到指定的系统内存单元的硬件来完成，所述系统内存单元是调度程序可读的。为了解释本发明的这个方面，考虑下列情况：

[0234] 1) 调度程序调度运行列表 A(1-2-3-4-5)。

[0235] 2) 环境 #1 的时间片期满，并且调度程序发送新运行列表 B(2-3-4-5-1)。

[0236] 3) 当处理 CPU 上的时间片满期时，协处理器结束了环境 #1，因为它转换成空的，而因此转换成环境 #2。协处理器产生对于这个事件的环境转换中断。

[0237] 4) 协处理器从 CPU 那里接收关于新运行列表的通知，且因此而转换它。协处理器产生对于这个事件的环境转换中断。

[0238] 5) 当处理新运行列表的环境 #2 中的描绘命令时，协处理器遇到页面错误，且因此而转换成环境 #3。协处理器产生对于这个事件的环境转换中断。

[0239] 6) 环境 #3 立刻命中页面错误，且因此协处理器转换为环境 #4。协处理器产生对于这个事件的环境转换中断。

[0240] 7) 最后 CPU 因环境转换而被中断。实际上已经由于产生原始中断而发生了四次环境转换。

[0241] 图 19 举例说明了上述情况下的硬件历史机制的操作。为了支持这种历史机制，该硬件可以被配置成能够执行下列任务。提供这些任务是作为举例而并非是限制：

[0242] 1) 规定历史缓冲区的基址。每个协处理器中可以存在单个历史缓冲区。在优选实施例中，这可以是 PCI 或者 AGP 内存中的系统内存单元。这可以由操作系统调整在 4KB 范围上。对于 PCI 专用的系统来说，优选地，访问这个缓冲区可以利用窥探 (snoop) 周期来实现，以便系统内存缓冲区能被高速缓存，以供更高效的 CPU 读取。

[0243] 2) 规定历史缓冲区的尺寸。历史缓冲区至少可以是运行列表尺寸的两倍。这是为了确保在缓冲区中有足够的空间来处理最坏的情况，所述最坏情况的就是：当前运行列表和待定运行列表在中断发生以前完成。

[0244] 3) 规定协处理器写入点，该点可以是经过最后一次事件时紧接于写入历史缓冲区的地址。VidMm 也许能时时查询这个指针，包括当协处理器正在运行时。在更新指针以前，历史缓冲区中的数据可以被恰当地清除到内存中，以便确保调度程序总是获得连贯的数据。

[0245] 各种实施例可能配置历史缓冲区，所以它对于 DMA 缓冲区而言是不可见的，所述 DMA 缓冲区是以用户工作方式来构造的。如果历史缓冲区对于受限 DMA 缓冲区而言是可见的，那么恶意应用程序就可能会改写历史缓冲区，打断调度程序并且可能会引起系统崩溃或更恶劣的情况。由于这个原因，这些实施例中的历史缓冲区可以由硬件通过仅在特权 DMA 缓冲区中可见的物理地址或者通过虚拟地址来访问。在这些实施例中，可能需要协处理器来环绕式处理 (wrap around) 历史缓冲区的末尾，而无需 CPU 介入。

[0246] 注意，依照所述实施例的运行列表没有立刻消除所有协处理器能够多次在同一环境上因相同的原因而发生故障的需要。这其中的一个原因就是：调度程序通常是当协处理器忙着执行当前运行列表的同时构造新运行列表。由于调度程序可能需要包括已经存在于上述运行列表中的新运行列表中的一些环境，因此存在这样的可能，就是：正在重复的环境的状态、在它被放入正在构造的运行列表的时间与运行列表被提交给协处理器的时间之间

改变。

[0247] 受限 - 特权 DMA

[0248] 依照高级调度模型中内存保护的介绍，发送给协处理器的 DMA 缓冲区可以主要由运行应用程序进程内部的用户工作方式驱动器来构造。可以在应用程序的进程中映射那些 DMA 缓冲区，用户工作方式驱动器能够直接写入它们，而内核驱动程序无法证实它们。可以在应用程序偶然访问它们的虚拟地址或恶意应用程序故意对其访问时滥写 DMA 缓冲区。为了允许驱动器模型保持安全性，也就是说，不容许应用程序有机会访问它不应访问的资源，在能允许以用户工作方式构造的 DMA 缓冲区做什么的方面，对该 DMA 缓冲区进行限制。具体来讲，内置的 DMA 缓冲区可以具有在以下示范性方式中限制的功能 ：

[0249] 1) 它们可以仅仅包含对虚拟地址的访问，而根本不具有对物理地址的访问 (包括栅栏)。

[0250] 2) 不能允许它们包含将影响当前显示 (例如，CRT、空闲存取控制 (DAC)、技术文件管理系统 (TDMS)、电视输出端口 (TV-OUT)、Internet2(I2C) 总线) 的指令。

[0251] 3) 一般说来，它们不包含将影响适配器 (例如，锁相环 (PLL)) 的指令。

[0252] 4) 可以限制它们的功率管理和 / 或配置空间。

[0253] 5) 不能允许它们包含将防碍环境转换的指令。

[0254] 可以在以用户工作方式构造的 DMA 缓冲区中编排的实际的寄存器组，将可能会从硬件到硬件变化。然而，不管硬件怎样，寄存器都遵循一般的规则，也就是说，这种 DMA 缓冲区应当仅允许使用对资源和栅栏的虚拟地址访问的描绘操作。为了提供增强的安全性，能够要求这种 DMA 缓冲区不许应用程序使用该应用程序不应该访问的内存，或是不允许在某种潜伏地大变动和不可回收的方式中可能会影响硬件的应用程序对其访问。

[0255] 为了防止以用户工作方式构造的 DMA 缓冲区访问确定的功能，可以在协处理器中实施多种手段。这些手段可以依功能的性质以及功能是否需要在应用程序的协处理器环境流中排队而变。确定的特权操作通常需要在协处理器环境流中排队，所述环境流既包含以用户工作方式构造的 DMA 缓冲区 (例如，描绘的应用程序) 又包含以内核工作方式构造的特权 DMA 缓冲区 (例如，队列交换)。

[0256] 不需要排队的函数。大多数特权函数都不需要在应用程序协处理器环境流中排队。诸如下列这样的函数是不需要排队的 ：

[0257] 1) 程序 CRT 计时 (timing)。

[0258] 2) 更新 DAC 的查找表 (注意 ：由于如果主屏幕无论如何想要的话那么任何应用程序都可以给予主屏幕，因而并非绝对地要求编排 DAC LUT 是特权函数，并且重新编排查找表 (LUT) 不会允许应用对它早已不能防问的信息进行用户访问)。

[0259] 3) 对显示器输出编程 (TDMS、TV-OUT)

[0260] 4) 与子女设备 / 监视器相通信 (I2C、)

[0261] 5) 程序时钟 (PLL)

[0262] 6) 改变协处理器的功率状态

[0263] 7) 配置协处理器 (配置空间、基本输入输出系统、)

[0264] 通常要求这种函数遵循系统事件，该系统事件是完全独立于应用程序描绘流的 (例如，引导、分辨率改变、pnp 检测、功率管理)。照此，不需要这种函数在特定应用程序的

协处理器环境中排队。当在无任何来自于用户工作方式驱动器的介入的情况下专用系统事件发生时，这种函数可以由内核工作方式驱动器本身使用。

[0265] 对于这种函数而言，IHV 可以决定仅通过存储映像输入 / 输出 (MMIO) 来让所有下层寄存器成为可访问的。由于寄存器通常仅仅被映射到内核空间中，因而它可能不能让应用程序或用户工作方式驱动器访问它们，因此有效地保护了所述函数。

[0266] 另一个手段将会是实施每协处理器的环境特权级。利用这个手段，一些环境将在当别人不能做时它们能做什么的方面受限制。在那种情况下，以用户工作方式构造的应用程序的 DMA 缓冲区将排队到受限环境中。在另一方面，内核工作方式驱动器将使用特权环境以提交特权函数。

[0267] 需要排队的函数。由于能够插入以用户工作方式构造的 DMA 缓冲区中的命令是受限制的，因而可以实施高级模型以要求协处理器支持受限 DMA 缓冲区（也就是考虑先前状态的 DMA 缓冲区）和特权 DMA 缓冲区。需要特权 DMA 缓冲区，是为了允许特权函数沿着协处理器环境的描绘流而排队。

[0268] 特权 DMA 缓冲区可以包含在无特权 DMA 缓冲区中发现的任何指令。本发明的各种优选实施例都可以实施特权 DMA 缓冲区，该 DMA 缓冲区至少允许下列情况（在后面的部分中将用进一步详述来解释）：

[0269] 1) 特权栅栏的插入

[0270] 2) 交换指令的插入

[0271] 3)“非环境转换”区域的插入

[0272] 此外，特权 DMA 缓冲区能够对任何 IHV 需要的硬件寄存器进行编排，并且如有需要则能够访问虚拟内存和物理内存两者。特权 DMA 缓冲区可能不能被构造或者在用户工作方式中不可见。只有信任的内核组件能够访问并构造特权 DMA 缓冲区。

[0273] 下列部分表示了实施特权 DMA 缓冲区的三种可能方式，并且意图是阐明特权 DMA 缓冲区实施方案的概念，而不是为了限制可以实践本发明的各种方式：

[0274] 1. 仅以内核工作方式的 DMA 缓冲区构造

[0275] 支持特权 DMA 缓冲区的一个方法就是：不需要将要求实际的 DMA 缓冲区发送给以内核工作方式构造的硬件的任何专用硬件支持。在那种情况下，用户工作方式驱动器将构造十分类似于 DMA 缓冲区的命令缓冲区，并且将它提交给内核工作方式驱动器。内核工作方式驱动器将证实并将这种命令缓冲区复制到 DMA 缓冲区中，也就是只有在内核工作方式中可见。在证实期间，内核工作方式驱动器将证实不存在无特许的指令。这类似于基本模型所需要的证实，但是不需要内存访问的证实，这是因为内存是虚拟化的。

[0276] 2. 直接将特权命令插入到环中

[0277] 大概支持特权 DMA 通道的最容易的硬件手段就是：直接将特权命令插入到协处理器环境环中。环本身已经是仅可从内核工作方式访问的特权通道了。这在图 20 的图中描绘出了。

[0278] 3. 通过间接寻址来规定特权

[0279] 在图 21 中举例说明了支持协处理器中的受限 - 特权 DMA 缓冲区的不同手段。参照该图，注意，起始地址和结束地址都能按 DWORD 设置。地址未使用位可以被再使用，以规定标志位。起始地址的第一位能够规定正被重定向为特权 DMA 缓冲区的 DMA 缓冲区。为提

高安全性，特权 DMA 缓冲区能够访问辅助内存中的物理地址。受限 DMA 缓冲区能够访问协处理器环境虚拟地址空间中的虚拟地址。

[0280] 在这个手段中，间接寻址命令中的位可以被插入到环形缓冲区中。该位表示正在被执行的 DMA 缓冲区是否是特权 DMA 缓冲区。这个暗指：协处理器可以利用物理地址来访问环形缓冲区本身，并且在协处理器虚拟地址空间中所述环形缓冲区本身可能是不可见的。允许主环形缓冲区在协处理器虚拟地址空间中可见，将会允许恶意应用程序改写该主环形缓冲区，并且允许它以特权级运行命令，这将意味着在多数计算环境中的安全性破坏。在这点上，像受限 DMA 缓冲区一样，能够通过物理地址而非虚拟地址来访问特权 DMA 缓冲区。

[0281] DMA 控制指令

[0282] 对于用以追踪任何协处理器环境的连续进阶并且控制那个环境的 DMA 流当中的指令流量的调度程序和辅助内存管理器而言，协处理器可以被配置成支持其 DMA 流中的下列示范性指令：

[0283] 1）栅栏（受限的和特权的两者）

[0284] 2）陷阱

[0285] 3）启动 / 无效环境转换

[0286] 栅栏。栅栏能够是这样的指令，它既包含一段数据（例如，64 位数据段）又包含能够插入在 DMA 流中的地址。当协处理器从所述流中读取该指令时，它将使得协处理器在特定地址处对与栅栏相关联的数据段进行写。在协处理器能够在内存中写入栅栏数据以前，它必须确保来自于栅栏指令前的原语的像素已经退休并且已经被完完全全地写入内存了。注意，这并不意味着协处理器需要停止整个流水线。当协处理器正在等待将要退休的栅栏前的指令的最后一个像素时，能够执行跟随栅栏指令之后的所述原语。

[0287] 尽管可以结合本发明使用适合上述说明的任何栅栏，但是将在这里具体描述两种类型的栅栏：正规栅栏和特权栅栏。

[0288] 正规栅栏是能够被插入在 DMA 缓冲区中的栅栏，所述 DMA 缓冲区是由用户工作方式驱动器创建的。由于 DMA 缓冲区的内容出自于用户工作方式，因而它不被信任。由此，这种 DMA 缓冲区内的栅栏能够访问那个协处理器环境的地址空间中的虚拟地址而不是物理地址。不言而喻，就如协处理器所访问的任何其它虚拟地址一样，对这种虚拟地址的访问受到相同的内存证实机制的约束。

[0289] 特权栅栏是只能被插入在 DMA 缓冲区中的栅栏，所述 DMA 缓冲区是以内核工作方式创建的（并且仅仅在内核工作方式中可见）。这种栅栏能够访问内存中的物理地址，以提高系统的安全性。如果栅栏目标地址在协处理器环境的地址空间中是可见的，那么恶意应用程序就会在那个内存单元上进行图形操作，由此而覆盖掉内核工作方式代码正在期待接收的那些东西的内容。对于潜在安全性问题的另一个解决方案将是：在 PTE 中具有特权位，该特权位表示虚拟地址是否能够被来自于非特权 DMA 缓冲区访问。然而，上面的第一个手段被视为对于早期硬件生成的简单手段。

[0290] 注意，特权 DMA 缓冲区能够包含正规栅栏和特权栅栏两者。然而，当特权 DMA 缓冲区包含正规栅栏时，产生 DMA 缓冲区的内核组件得知它插入的栅栏可能从不可见。

[0291] IHV 可以决定支持额外类型的栅栏，以为了最小化需要被清除的内部缓冲区的数目。下列类型的栅栏是可以为此目的而支持的示范性栅栏（注意：特权的和无特权的）：

[0292] 1. 写栅栏

[0293] 写栅栏可以是先前所述那种类型的栅栏，而且是唯一所需的栅栏类型。写栅栏确保在栅栏指令被处理前的所有内存写入均是全局可见的（即，已将它们清除出了高速缓存，并且已从内存控制器那里接收到了确认）。

[0294] 2. 读栅栏

[0295] 读栅栏是类似于写栅栏的更轻的一种栅栏类型。读栅栏确保在栅栏结束以前用于描绘操作的所有内存读取，但是一些写入仍然可以是尚未完成的。如果支持读栅栏，那么调度程序就将利用它们来控制非描绘目标分配的使用期。

[0296] 3. 管顶端栅栏

[0297] 管顶端栅栏是非常轻量级的栅栏。管顶端栅栏的支持是可任选的。管顶端栅栏仅仅确保协处理器读取防 DMA 缓冲区中的御指令前的末尾字节（然而仍不是必须要处理的）。在栅栏被处理之后，协处理器可以不必再读取管顶端栅栏前的任何部分的 DMA 缓冲区（由于 DMA 缓冲区的内容可能不再有效）。如果支持的话，则这类栅栏将由调度程序使用，以控制 DMA 缓冲区的使用期。

[0298] 陷阱。可以在本发明的各种实施例中实施陷阱。陷阱可以是插入在 DMA 缓冲区中的指令，该指令能够当其由协处理器处理时产生 CPU 中断。在协处理器能够中断 CPU 以前，最好确保来自于陷阱指令前的原语的所有像素都已经退休并且都已被适当地写入到内存（可以包括来自于栅栏指令的内存写入的操作）。注意，这不意味着协处理器需要停止整个流水线。当协处理器在等待将要退休的陷阱之前的指令的末尾像素时，能够执行跟随陷阱指令的原语。

[0299] 陷阱指令不必是特许指令并且能够被插入在任何 DMA 缓冲区中，包括那些直接由用户工作方式驱动器构造的 DMA 缓冲区。

[0300] 启动 / 禁用环境转换。对于支持次三角关系中断的硬件来说，能够提供指令来启动和禁用环境转换。尽管环境转换被禁用，但是协处理器通常不应该转离当前的协处理器环境。尽管如果 CPU 提供新运行列表则可以要求协处理器更新其当前的运行列表信息，但是协处理器可以推迟对于那个新运行列表的环境转换，直到环境转换被重新启动为止。OS 能够确保下列规则保持树当环境转换时被禁用：

[0301] 1) 仅仅特权 DMA 缓冲区将被处理。

[0302] 2) 非环境转换指令将存在于 DMA 流中。

[0303] 3)DMA 流将不在指令外运行。

[0304] 4) 页面错误将不会发生（如果支持页面级故障的话）。

[0305] 在许多计算机系统中，禁用和启动环境转换都是可以仅仅存在于特权 DMA 缓冲区中的特许指令。对于这些指令的使用情况是为了允许调度程序在其不可能被中断的情况下调度将出现在屏幕上的操作（即，显现 blit）。在这样一个操作中被中断将会导致在显著的一段时间中在屏幕上可以见到伪像。

[0306] 注意，如果协处理器遇到 DMA 缓冲区中无法预知的误差，那么即使环境转换被禁用，它也能够进行环境转换以转离这个 DMA 缓冲区。由于仅仅以内核工作方式构造的 DMA 缓冲区可以包含不可中断的部分，因而无法预知的误差将成为驱动器错误 (bug) 或硬件错误的结果。如果在那些情况下协处理器不进行环境转换以转离出去，那么显示器看门狗

(watchdog) 就将捕获到所述挂起 (hang) 并重置协处理器，以便恢复系统。

[0307] 任选的控制指令。尽管调度程序能够利用上述的简单控制指令来构造高级同步原语，但是可以让结果变得更为有效。在许多计算机系统中，协处理器环境在其能取得同步对象的所有权以前被 CPU 中断。如果取得了同步对象并且以高频加以释放的话，那么这就会成为问题。为了具有更多有效的同步原语，调度程序能够从协处理器那里接收专用指令。具体来讲，协处理器可以被配置成能在适当时间发送“等待”指令和“发信号”指令。

[0308] 等待指令被插入在 DMA 流中，以通知协处理器它检查指定计数器的值。如果计数器是非零的，那么协处理器就能够递减计数器并继续执行当前的协处理器环境。如果计数器是零，那么协处理器就能够在等待指令以前重置当前协处理器环境的指令指针，并且转入运行列表中的下一个环境。当协处理器环境需要停止在等待指令上并且在之后被重新调度时，由于仍不满足等待条件，因此协处理器可以重新执行该等待指令。

[0309] 等待指令仅仅需要具有一个参数，即规定将被比较 / 缩减的内存单元的虚拟地址。计数器可以是至少 32 位并且可以是任何有效的虚拟地址。在优选实施例中，等待指令可以是不可中断的；也就是说，如果将新运行列表给予协处理器，那么它就能够在等待指令以前或者在它完成之后而转入新运行列表。等待指令能够被插入在受限 DMA 缓冲区和特权 DMA 缓冲区两者中。

[0310] 发信号指令可以被插入在 DMA 流中，以通知协处理器它可以更新计数器的值。接着，协处理器可以将计数器的值加 1。协处理器可以忽略在加法期间的潜在溢出。作为选择，协处理器能够将溢出作为流中的误差予以报告，以便帮助追踪软件错误。

[0311] 发信号指令仅仅需要具有一个参数，即应该更新的计数器的虚拟地址。可以把计数器尺寸作成匹配等待指令计数器的尺寸，并且在优选实施例中，该尺寸至少是 32 位。发信号指令能够被插入在受限 DMA 缓冲区和特权 DMA 缓冲区两者中。

[0312] 交换

[0313] 为了允许全屏幕应用程序无缝地运行而没有流水线中的气泡，协处理器可以提供用于交换排队的指令（即，显示的基址的改变）。通常连续地分配来自于物理内存的显示表面，并且 CRTC 利用物理地址而非虚拟地址来访问所述显示表面。因此，交换指令可用于将 CRTC 编程为将要显示的新物理地址。由于这是物理地址而非虚拟地址，因而骗子应用程序能够潜在地将 CRTC 编程为显示属于另一个应用程序或用户的部分辅助内存（这部分内存可能包含有机密）。为此，可以实施交换指令，从而通过确保它仅仅是由内核工作方式驱动器插入在 DMA 流中的特许指令，并且一旦目的地已被证实，就能保护多数计算机系统的安全性。

[0314] 在结合交换功能的本发明的各种优选实施例中，至少可以支持两种类型的交换：立即交换和显示刷新的同步交换。当协处理器处理立即交换时，它可以立刻更新显示的基址，即使这样做将造成可见的图像撕裂 (tearing)。当协处理器处理同步交换时，它可以锁住新的基址，但是推迟其更新直到下一个垂直同步周期到来为止。如果协处理器在垂直同步周期之间正在处理多于一个的同步交换，那么协处理器仅仅能够锁住最近的一个并且忽略前面的一个。

[0315] 当处理同步交换时，可能配置各种实施例，以便协处理器可以不必停止图形流水线。OS 将确保不会在环形缓冲区中排列任何描绘命令，所述环形缓冲区将形成当前可见的

表面。注意，在这里，正如在将于下面进一步解释的“优化交换”情形那样，也可以在无这些要求的情况下配置其它实施例。

[0316] 为确定哪个表面是当前可见的，驱动器可能首先能确定特定的排队交换何时已经出现并且通知调度程序这一事件，也就是在显示基址被改变之后通知调度程序。对于立即交换而言，确定何时发生交换是容易的，这是因为从 DMA 流读取交换指令可以被视为是与更新显示表面相同的事件。栅栏和中断都能够被插入在跟随交换指令后的 DMA 流中，以便通知调度程序特定交换被读取。

[0317] 在同步交换的情形中，确定哪个表面是当前可见的，更加困难。协处理器将首先从 DMA 流中读取交换指令，但是将在下一个 vsync 中断时迟一些更新显示表面。为了消除在此期间停止协处理器的需要，可以提供一种用以通知调度程序显示表面改变何时生效的机制。

[0318] 对于这种供结合本发明之用的通知而言，存在了许多方法来设计这种机制。在图 22 中举例说明了一个可能的简单手段。图 22 提供了一种查询与当前显示表面有关的协处理器的方法。在举例说明的实施例中，这些功能可以被认为是由 MMIO 寄存器提供的。图 22 的系统是这样一种设计，即当寄存器读取实际显示表面而不是读取最新的“已锁定的显示表面”时，这种设计将产生更大的可靠性。在协处理器处理另一个排队的交换的同时，查询最新的已锁定显示表面能够产生竞态条件 (race condition)，这可能会导致屏幕上的图像撕裂。可以利用任何适当的技术来产生交换指令。对于本发明，唯一的兼容性一般要求就是 :实施解决方案应该确保交换不会被确认，直到它有效为止。

[0319] 队列交换。为了提供最大的性能，可以修改高级调度模型以便在应用程序的描绘流中排队交换操作，所述应用程序拥有监视器。当进行 n 型缓冲时，调度程序可以允许达 n-1 次之多的交换，所述交换将要在 DMA 流中被排队，并且当第 n 次交换即将被插入时，该调度程序还可以将其阻塞。

[0320] 这就意味着 :在双重缓冲中，调度程序可以允许应用程序对一次交换排队，并且让它继续预备 DMA 缓冲区以待接下来的帧，而此时协处理器结束描绘当前帧并且处理 / 确认此次交换。它还意味着 :如果随着为下列帧而预备 DMA 缓冲区并提交第二次交换、直到应用程序结束为止时，那么它就可以被阻塞，直到第一次交换得到协处理器确认为止。

[0321] 当调度程序使用立即交换时，排队交换机构按上述那样工作。然而，当使用同步交换时，调度程序还可以专门地照顾经过 n-1 次交换而被排队的 DMA 缓冲区。实际上，经过那次交换的 DMA 缓冲区通常将被描绘给当前可见的表面。建议在多数系统中，不处理这些 DMA 缓冲区，直到当前已排队的交换的次数一直变回到 n-2 或 n-2 以下。

[0322] 解决这个问题的最简单手段就是 :仅允许 n-2 次交换排队，而不是 n-1 次。然而，这种解决方案将还意味着 :在双重缓冲的情况下，我们无法将任何交换排队，所以我们将需要在每一帧完成之后阻塞所述应用程序，直到相应的交换被处理为止。

[0323] 在图 23 中，举例说明了在此设置中的优选手段。作为举例说明，允许 n-1 次的交换排队。为了防止在交换 n-1 次之后执行对 DMA 缓冲区的排队，调度程序可以在那个协处理器的虚拟环形缓冲区中累积那些 DMA 缓冲区。调度程序可以等待直到当前队列交换的次数一直变回到 n-2，以便将那些队列交换提交给实际的协处理器环境的环。

[0324] 正如在图 23 中举例说明的那样，当多个应用程序同时运行时，协处理器可以不必

停止。尽管协处理器通常将停止处理来自于特定协处理器环境的 DMA 缓冲区，但是调度程序可以调度其它的协处理器环境以便运行，由此有效地保持了协处理器忙。然而，当单个应用程序正在运行时，例如当玩全屏幕游戏时，协处理器可以在那些时间间隔期间停止。下一个部分描述了这样一种机制，即如果能得到支持的话，所述机制将由调度程序来使用，以便减少停止时间。

[0325] 优化交换。试图为全屏幕应用程序而优化，我们想要减少协处理器从停止到最小值所占的时间。参看图 23，注意到，协处理器能够因至少两个原因而停止：第一、因为帧结束，但系统正在等待 vsync 以便交换；第二、因为交换结束，但系统正在等待中断以便通知 CPU。

[0326] 为了因第一个原因而减少停止，可以将更多的缓冲区添加到交换链中。例如从双倍缓冲区开始直到三倍缓冲区，由此将大大地减少这种停止。然而，这样做并不总是在驱动器的控制下，并且可能产生过度的内存耗费。

[0327] 为了因第二个原因而减少停止，可以添加协处理器机制以便完全地消除掉这一停止的需要。协处理器能够提供在交换指令上的等待，所述交换指令将停止协处理器，直到先前队列交换已经被处理为止。当支持此类指令时，调度程序能够为全屏幕应用程序而使用它，以便对交换进行排队，并且 CPU 不必在每一次交换之后重新启动 DMA 流。

[0328] 高级同步对象

[0329] 通过使用先前定义的控制指令，调度程序能够构造诸如临界区以及人工干预之类的高级同步对象。一旦符合等待状态时，调度程序就能够通过防止执行部分 DMA 缓冲区直到它由 CPU 显式地重新调度，来实施这种同步原语。可以由调度程序来实施等待对象，就像栅栏一样。逻辑上遵循栅栏的 DMA 缓冲区可以由调度程序进行排队，但不将其提交到协处理器环境的环中，直到符合等待条件为止。一旦它正在等待对象，协处理器环境继而就可以由调度程序移动到特定对象上的等待列表中，直到发信号通知它为止。可以通过在协处理器环境 DMA 流中插入栅栏并继之以中断指令，来发信号通知对象。当接收这种中断时，调度程序可以识别正在发信号通知哪个对象，继而可以确定是否有任何等待协处理器环境应该在就绪队列中向后移。当在就绪队列中将协处理器环境向后移时，调度程序将从环中阻止的 DMA 缓冲区插入进来。

[0330] 例如，考虑本发明的实施例，在其中应用程序具有在生产者和消费者之间共享的表面，并且所述应用程序需要同步访问资源，以便当描绘时、消费者总是能使用有效的内容。在图 24 中举例说明了同步化此情形的一种可能的方法。

[0331] 转向图 24，在调度程序端上，例如可以通过下列内核形实转换程序 (kernel thunk) 来实施同步，所述内核形实转换程序可以在任何组合中或者结合其它动作来实施：

[0332] 1)CreateSynchronizationObject：为同步对象创建内核轨道结构。将对象的句柄返回到可被用于后续等待 / 释放 / 删除调用的用户工作方式。

[0333] 2)DeleteSynchronizationObject：销毁先前创建的对象。

[0334] 3)WaitOnSingleObject/WaitOnMultipleObject：将等待同步事件 (wait-on-synchronization event) 插入到当前协处理器环境的 DMA 流中。连同正对在等待的对象的访问一起，将所述事件插入到调度程序事件历史中。

[0335] 4)ReleaseObject/SignalObject：将发信号同步事件插入 (signal

synchronization event）到当前协处理器环境（栅栏 / 中断）的 DMA 流中。连同对正在释放或者发信号的对象的访问一起，将所述事件插入到调度程序事件历史中。

[0336] 将图 24 的例图应用到人工干预，一旦协处理器处理 DMA 流中的同步事件，调度程序就能够执行下列动作，这些动作也可以在任何组合中或者结合其它的动作来实施：

[0337] 1）等待（On a wait）：检查人工干预的状态。如果当前未采取人工干预，则采取人工干预并将协处理器线程放回到调度程序的就绪队列中。如果已经采取了人工干预，则将协处理器线程放在等待队列中以供人工干预。

[0338] 2）发信号（On a signal）：检测其它协处理器线程是否正在等待人工干预。如果其它线程正在等待，则取得在列表中正在等待的线程，并将它放回到调度程序的就绪列表中。如果没有线程正在等待，则将人工干预置回未采取状态。

[0339] 利用这个机制，调度程序就能够构造。例如，考虑下列类型的能由调度程序构造的同步原语：

[0340] 人工干预（Mutex）：同时只有一个协处理器线程有权存取共享资源。

[0341] 打信号（Semaphore）：指定数目的协处理器线程有权同时存取共享资源。

[0342] 通知事件（Notification event）：许多协处理器线程能够等待来自于另一个协处理器线程的信号。

[0343] 在一些情况下，应用程序可以被配置成当协处理器已经结束处理描绘指令时请求通知。为了支持这个，调度程序可能允许驱动器为 DMA 缓冲区请求它正在进行提交的通知。所述驱动器继而可以在提交时间规定 CPU 同步事件，一旦协处理器已经完成已提交的 DMA 缓冲区，就可以发信号通知所述提交时间。所述调度程序能够将指定的 DMA 缓冲区插入在指定协处理器环境的环中，并且接着将用户工作方式协处理器事件通知添加到该环（继之以中断的栅栏）中。当协处理器事件得到协处理器的处理时，调度程序能够发信号通知相关的 CPU 同步事件。

[0344] 调度程序事件历史缓冲区

[0345] 调度程序可以为多个目的而使用上述的同步机制。由于中断不会停止协处理器，因而 CPU 仅仅需要参看通知的子设备，且因此可以将一些通知挤压在一起。为了适当地响应 DMA 缓冲区中的每个通知，调度程序可以维护事件的历史，所述事件是连同任何处理那些事件所需的参数一起被插入的事件。

[0346] 事件历史缓冲区可能只不过是事件信息结构的每协处理器的环境阵列，所述事件信息结构追踪每个需要调度程序处理以及被插入那个环境的 DMA 流中的事件。注意，调度程序栅栏是由调度程序使用的栅栏，以便同步化事件。每协处理器环境都可能有一个栅栏，以保持安全性，可将所述栅栏作成仅仅允许通过特许指令来进行更新。在任何情况下，这类事件都可以作为继之以中断指令的栅栏指令而被插入到 DMA 流中。

[0347] 一旦每个栅栏中断，调度程序就可以首先确定当前的栅栏，然后仔细检查事件历史缓冲区来确定事件已经发生。这个确定可以根据相关联的栅栏来完成。调度程序可以继续处理栅栏中断。图 25 举例说明了事件历史缓冲区的各种实施例。

[0348] 可以支持许多事件。以下列表描述了一些当前支持的事件，但是这不意指限制可能支持的事件的个数和类型。

[0349]

事件的类型	说明和参数
DMA缓冲区的结束	这个事件被插入在DMA缓冲区的末尾处。 当这个事件由调度程序处理时，相关联的DMA缓冲区被放回到DMA缓冲区池中以供进行那个处理。 **参数：**需要被释放到池中的DMA缓冲区的句柄。
等待同步对象	当协处理器线程需要检查事件状态并且可能等待它时，
	插入这个事件。 当调度程序处理这个事件时，它检查是否已经符合等待条件，并且如果是，则重新调度恰好被停止的协处理器线程。如果不符合等待条件，则协处理器线程就被置于等待状态，并且被加到同步对象的等待队列中。 **参数：**正在等待的对象的句柄。
发信号通知同步对象	当协处理器线程需要发信号通知对象或者释放同步对象时，插入这个事件。当调度程序处理这个事件时，它改变所述对象的状态，并且可能唤醒一些正在等待的协处理器线程或事件。 **参数：**正在释放的对象的句柄。
用户工作方式事件通知	当用户工作方式驱动器请求描绘结束通知时，插入这个事件。当调度程序处理这个事件时，它发信号通知所述相关的事件。 **参数：**发信号通知的事件。

[0350] 可编程 PCI 插孔

[0351] 现今的协处理器正在显露为非常接近于 PCI 规格所允许的限制的 PCI 插孔。未来一代协处理器将具有比通过插孔置露的辅助内存更多的插件级辅助内存。因此，在将来我们无法假定所有的辅助内存都同时将通过 PCI 插孔可见。

[0352] 这个限制可以实行多种方式。支持每协处理器的环境虚拟地址空间的高级调度模型的优选方法是：使用无论在辅助内存中的任何地方都能够被重定向的 PCI 插孔，所述辅助内存处于 4KB 的粒度。这在图 26 中作了描绘。

[0353] 正如在图 26 中所描绘的那样，PCI 插孔页表能够独立于协处理器页表。当协处理器本身正在从环境到环境转换时，存在多个正在运行并访问 PCI 插孔的一部分的 CPU 进程。PCI 插孔的页表是所有协处理器环境当中的共享资源，并且将根据辅助内存来对其进行分配。驱动器能够提供一个映射 / 解除映射的 DDI，以便允许辅助内存管理器 VidMm 管理正运行的应用程序当中的 PCI 插孔地址空间。可以利用物理地址来让协处理器访问 PCI 插孔的页表。

[0354] 注意，PCI 插孔可以被配置成仅仅将地址空间重定向到本地辅助内存。它不需要将所述地址空间重定向到系统内存，因为 VidMm 将总是直接映射系统内存，而不通过那个插孔。

[0355] 页面级故障

[0356] 尽管先前描述的表面级故障通常能在大多数情况下较好地工作，但是也存在一些可能需要对其进行改进的情况。例如，利用表面级故障，某些使用非常大量数据集的应用程序不能同时获得内存中的全部数据集，因此也许不能很好地工作。可以在高级模型中实施的对此问题的解决方案就是 :页面级故障机制。

[0357] 利用页面级故障，所述模型类似于在前面部分中所述的模型那样进行工作。主要差异就在于页面错误被汇报且由 VidMm 句柄的方法不同。尽管表面级故障可能需要协处理器规定它需要取得进一步进展的整个资源列表（为了消除死循环，在其中分页一个资源意味着驱逐所需的另一个），页面级故障不需要协处理器展示一列虚拟地址。对于页面级故障而言，协处理器仅仅需要报告那个故障的虚拟地址。VidMm 能够找出这个地址属于哪个分配，并且判断是仅仅这个特定页面需要被驻留还是需要一些预取操作。当单个像素需要多个页面时，可能会因那个单象素而产生多个故障。当另一个页面正被引入时，那个像素需要的页面也有可能会遭到驱逐。然而，只要应用程序的所述工作组足够地大于像素可能需要的页面的最大数目，通过页面错误的循环的概率就是非常小的。

[0358] 最后，应当理解的是，在这里描述的各种技术都可以结合硬件或软件来实施，或者在适当的情况下，可以结合上述两者加以实施。因此，本发明的方法和设备或者其中的某些方面或部分，都可以采取嵌入在有形介质中的程序代码（即，指令）的形式，所述有形介质比如是软盘、CD-ROM、硬盘、或任何其它机器可读的存储介质，其中当程序代码被载入并由诸如计算机之类的机器执行时，所述机器就成为用于实施本发明的设备。在程序代码在可编程计算机上执行情况下，所述计算设备通常包括 :处理器、由处理器可读的存储介质（包括易失性和非易失性内存和 / 或存储元件）、至少一个输入装置以及至少一个输出装置。为了与计算机系统相通信，可以实施或者使用本发明的用户接口技术的一个或多个程序，例如通过使用数据处理 API、可再次使用的控制器等等，都是优选以高级程序或者面向对象编程语言来实施的。然而，如有需要，则所述程序能够以汇编语言或者机器语言中来实施。在任何情况下，所述语言都可以是编译语言或者解释语言，并且同硬件实现相结合。

[0359] 尽管示范性实施例是涉及在独立的计算机系统环境中使用本发明的，但是本发明不限于此，而是可以结合诸如网络或分布式计算环境之类的任何计算环境加以实施。更进一步来讲，本发明可以在多个处理芯片或设备中或者跨过多个处理芯片或设备加以实施，并且跨过多个设备也可以同样地实施存储。此类设备可以包括个人电脑、网络服务器、手持设备、超大规模计算机或者集成于其它诸如汽车和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个实施例，而是应该解释成落入依照所附权利要求的广度和范围中。

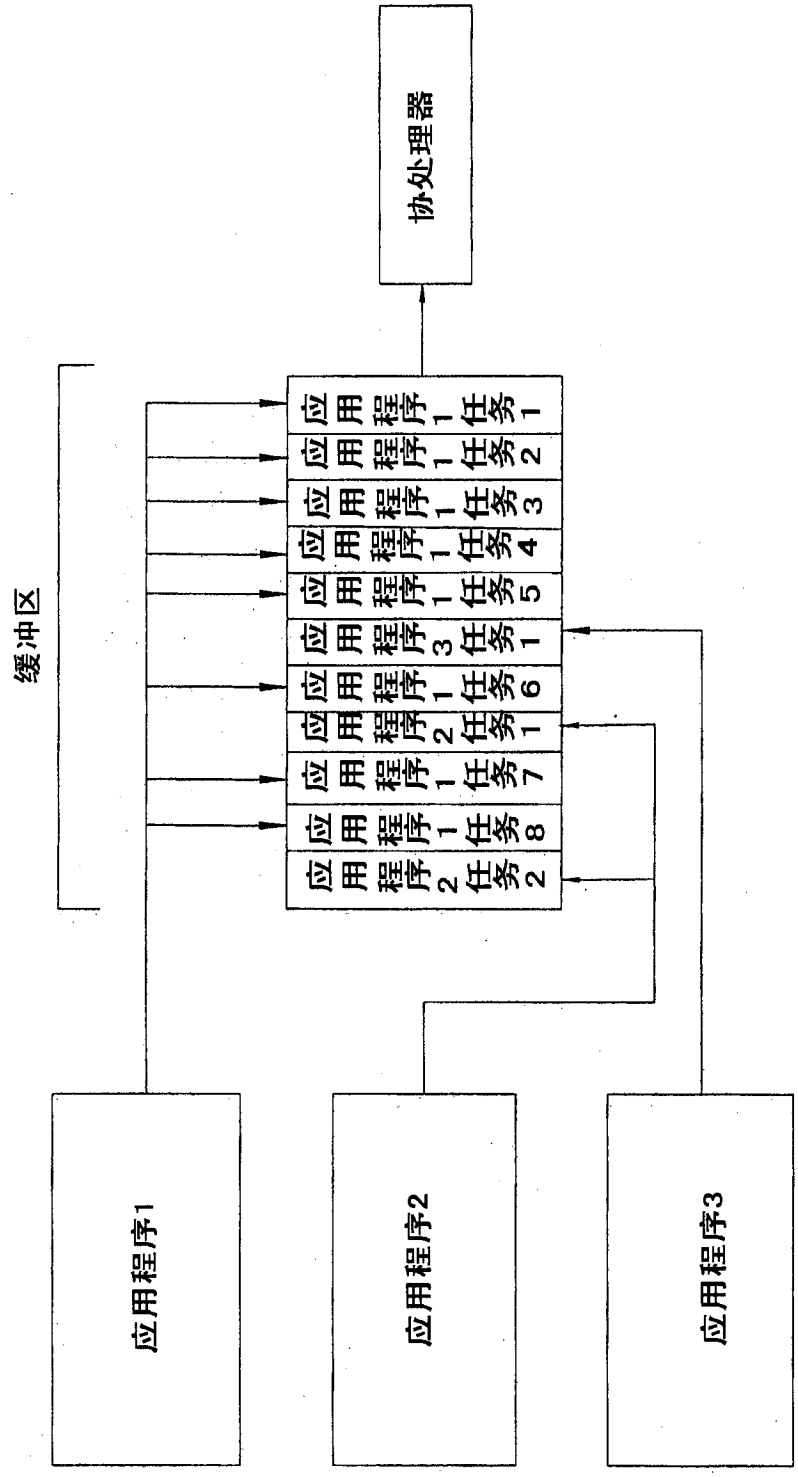

协处理器
缓冲区
应用程序1_任务1
应用程序1_任务2
应用程序1_任务3
应用程序1_任务4
应用程序1_任务5
应用程序3_任务1
应用程序1_任务6
应用程序2_任务1
应用程序1_任务7
应用程序1_任务8
应用程序2_任务2
应用程序1
应用程序2
应用程序3

图 1

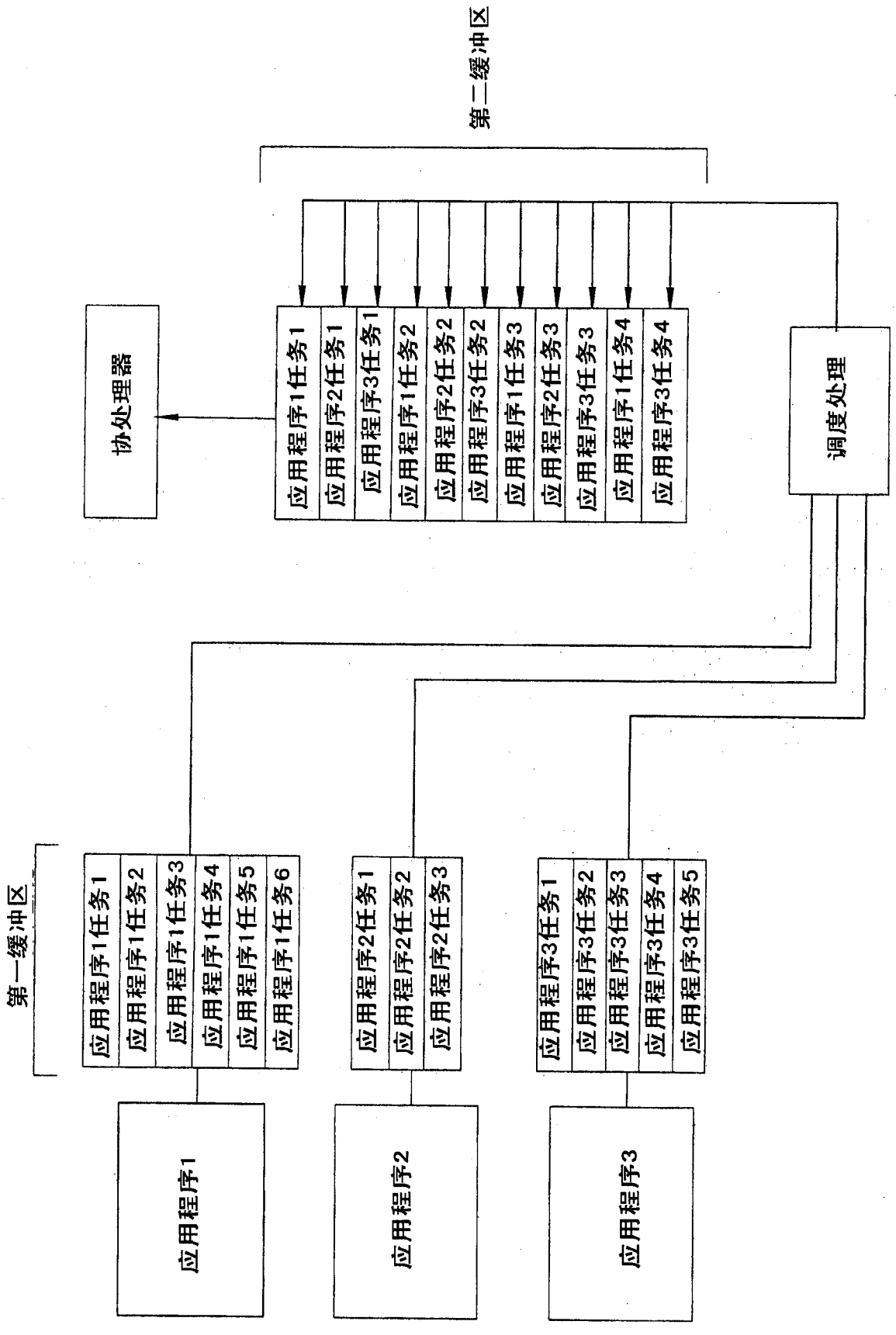

第二缓冲区
协处理器
应用程序1任务1
应用程序2任务1
应用程序3任务1
应用程序1任务2
应用程序2任务2
应用程序3任务2
应用程序1任务3
应用程序2任务3
应用程序3任务3
应用程序1任务4
应用程序3任务4
调度处理
第一缓冲区
应用程序1任务1
应用程序1任务2
应用程序1任务3
应用程序1任务4
应用程序1任务5
应用程序1任务6
应用程序2任务1
应用程序2任务2
应用程序2任务3
应用程序3任务1
应用程序3任务2
应用程序3任务3
应用程序3任务4
应用程序3任务5
应用程序1
应用程序2
应用程序3

图 2

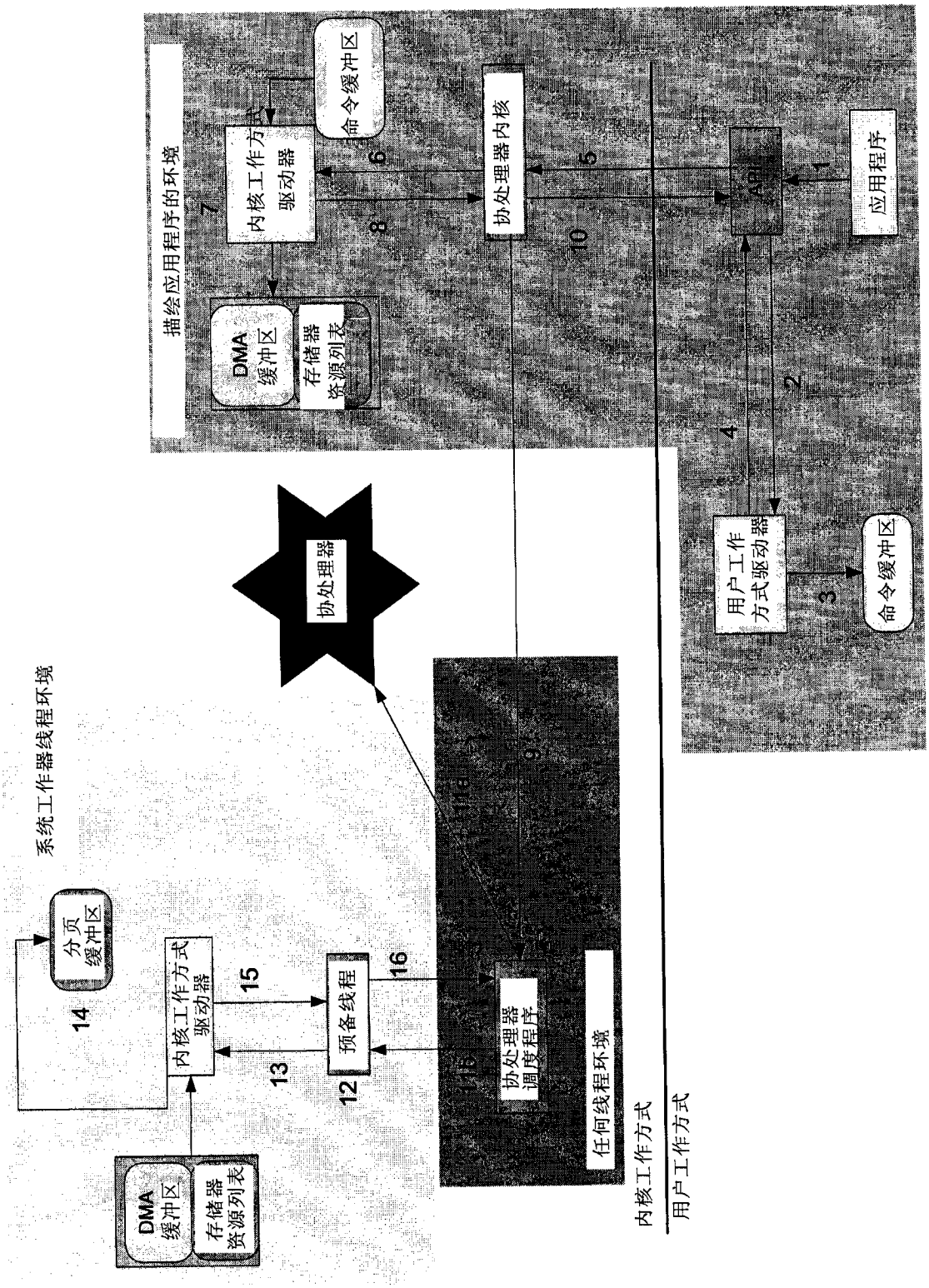
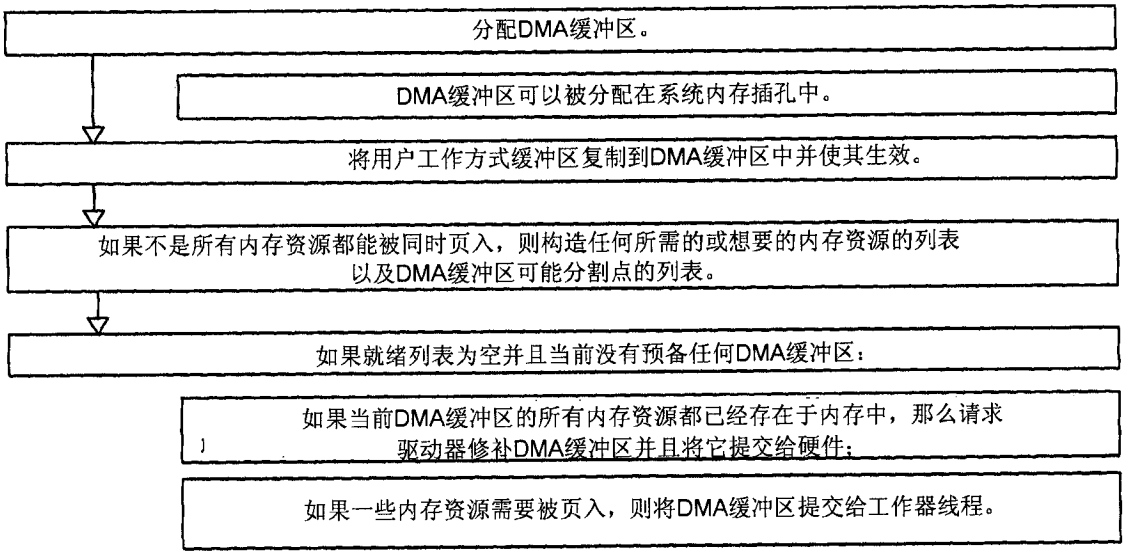

描绘应用程序的环境
应用程序
API
用户工作方式驱动器
命令缓冲区
协处理器内核
内核工作方式驱动器
DMA缓冲区
存储器资源列表
协处理器
系统工作器线程环境
分页缓冲区
预备线程
协处理器调度程序
任何线程环境
内核工作方式
用户工作方式
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

图 3

示范性算法

进程A：提交（中断请求无源，描绘线程环境）



进程B：量子期满（中断请求设备，任何线程环境）

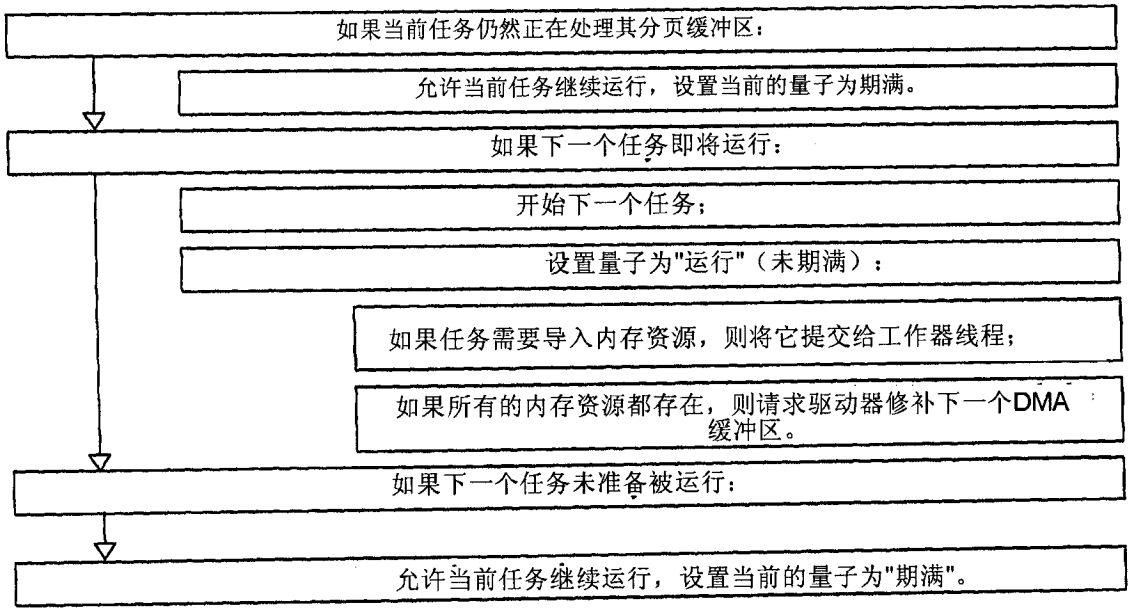


图 4(A)

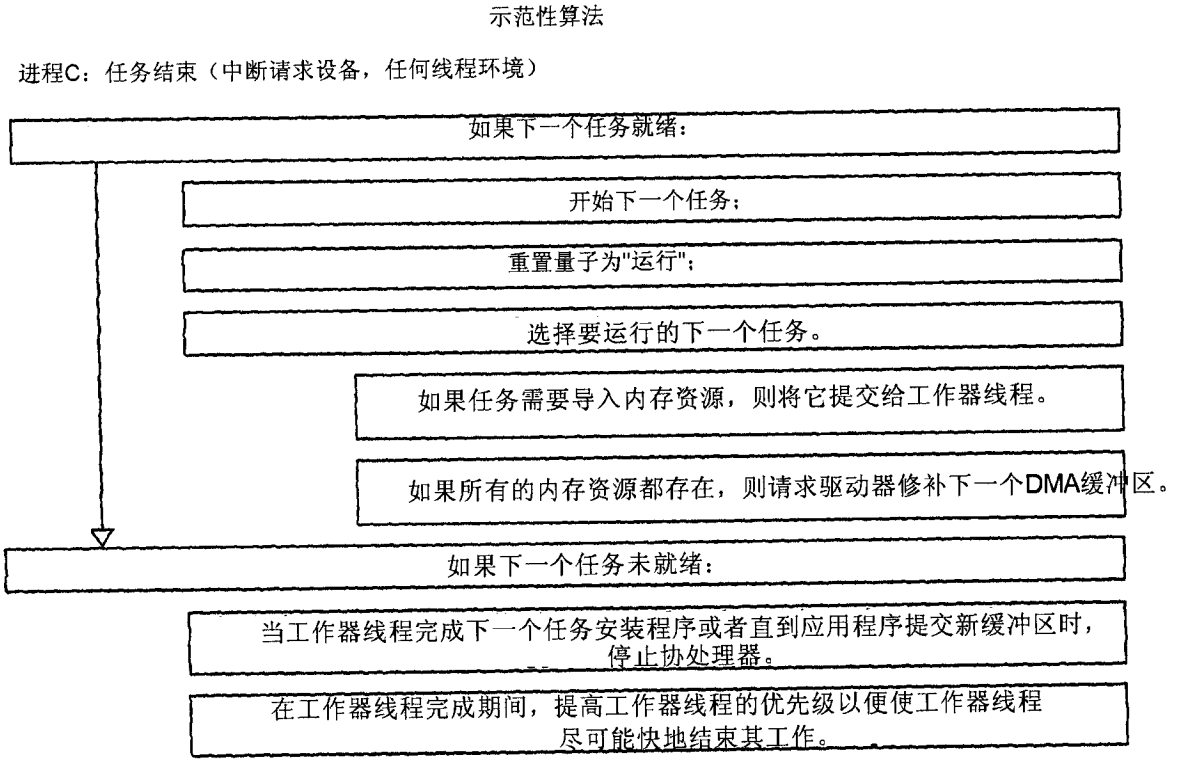

示范性算法
进程C：任务结束（中断请求设备，任何线程环境）
如果下一个任务就绪：
开始下一个任务；
重置量子为"运行"；
选择要运行的下一个任务。
如果任务需要导入内存资源，则将它提交给工作器线程。
如果所有的内存资源都存在，则请求驱动器修补下一个DMA缓冲区。
如果下一个任务未就绪：
当工作器线程完成下一个任务安装程序或者直到应用程序提交新缓冲区时，停止协处理器。
在工作器线程完成期间，提高工作器线程的优先级以便使工作器线程尽可能快地结束其工作。

图 4(B)

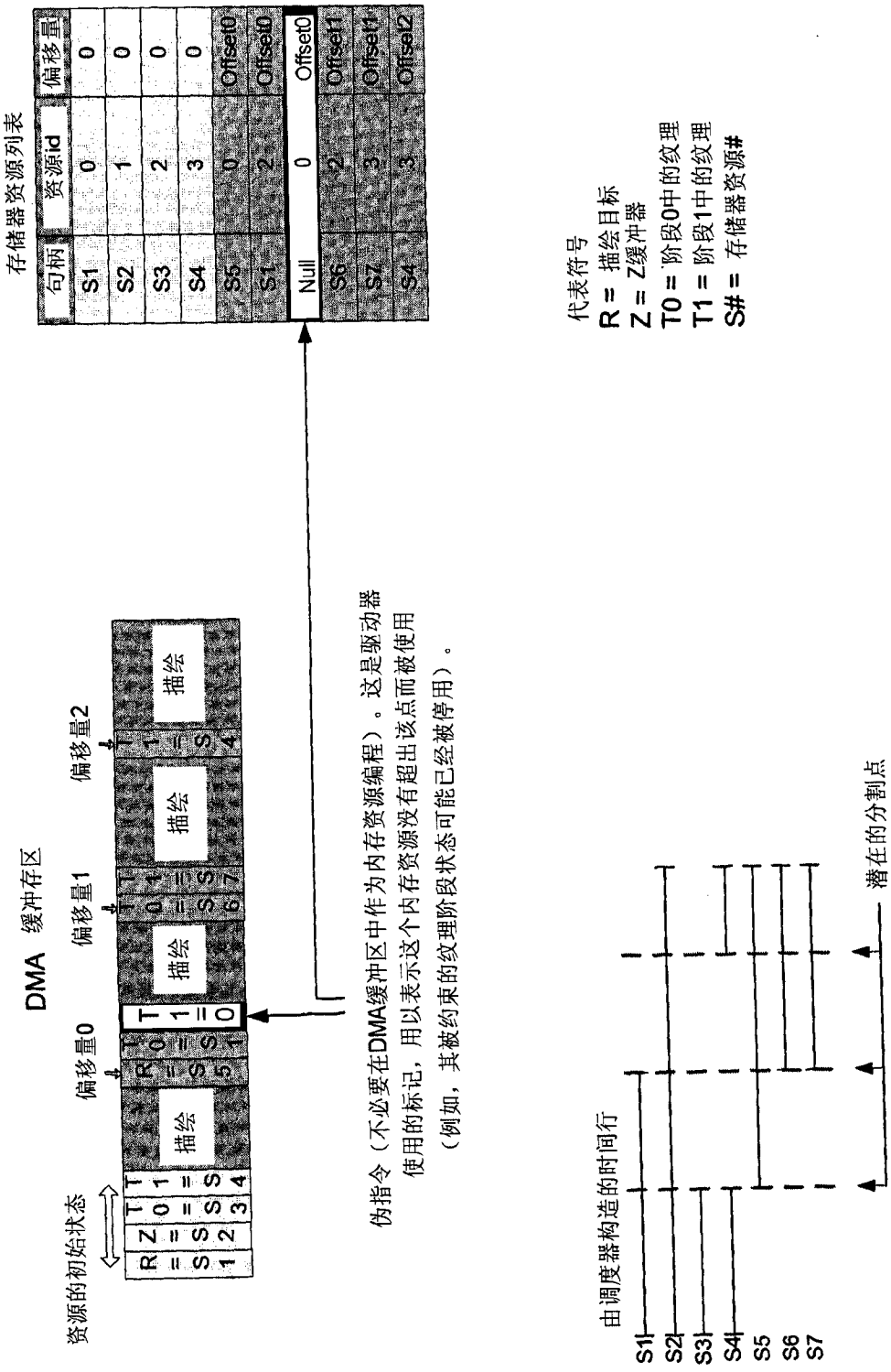

存储器资源列表
句柄
资源id
偏移量
S1 0 0
S2 1 0
S3 2 0
S4 3 0
S5 0 Offset0
S1 2 Offset0
Null 0 Offset0
S6 2 Offset1
S7 3 Offset1
S4 3 Offset2
代表符号
R = 描绘目标
Z = Z缓冲器
T0 = 阶段0中的纹理
T1 = 阶段1中的纹理
S# = 存储器资源#
DMA 缓冲存区
资源的初始状态
偏移量0
偏移量1
偏移量2
描绘
伪指令（不必要在DMA缓冲区中作为内存资源编程）。这是驱动器使用的标记，用以表示这个内存资源没有超出该点而被使用（例如，其被约束的纹理阶段状态可能已经被停用）。
由调度器构造的时间行
S1
S2
S3
S4
S5
S6
S7
潜在的分割点

图 5

示范性算法

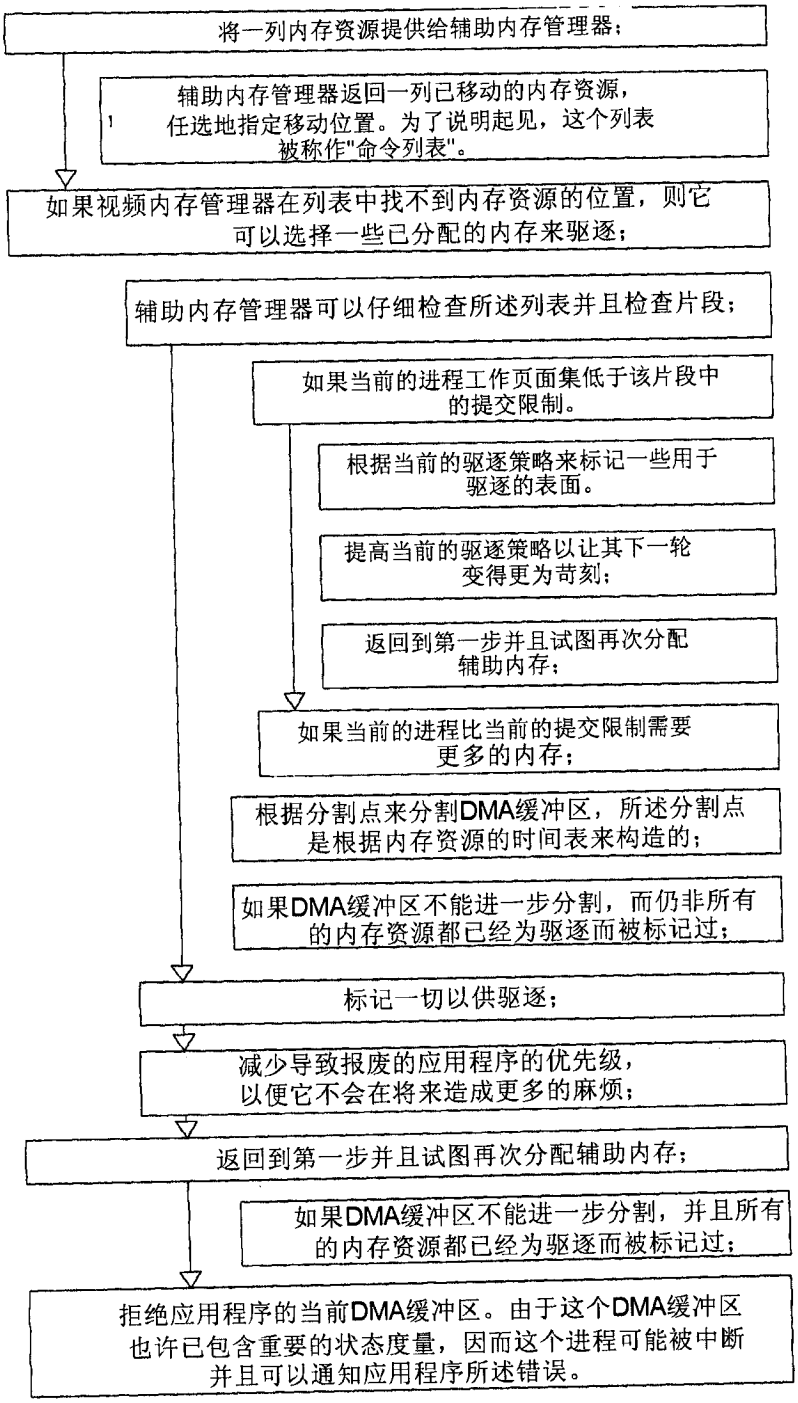

将一列内存资源提供给辅助内存管理器；
1 辅助内存管理器返回一列已移动的内存资源，任选地指定移动位置。为了说明起见，这个列表被称作"命令列表"。
如果视频内存管理器在列表中找不到内存资源的位置，则它可以选择一些已分配的内存来驱逐；
辅助内存管理器可以仔细检查所述列表并且检查片段；
如果当前的进程工作页面集低于该片段中的提交限制。
根据当前的驱逐策略来标记一些用于驱逐的表面。
提高当前的驱逐策略以让其下一轮变得更为苛刻；
返回到第一步并且试图再次分配辅助内存；
如果当前的进程比当前的提交限制需要更多的内存；
根据分割点来分割DMA缓冲区，所述分割点是根据内存资源的时间表来构造的；
如果DMA缓冲区不能进一步分割，而仍非所有的内存资源都已经为驱逐而被标记过；
标记一切以供驱逐；
减少导致报废的应用程序的优先级，以便它不会在将来造成更多的麻烦；
返回到第一步并且试图再次分配辅助内存；
如果DMA缓冲区不能进一步分割，并且所有的内存资源都已经为驱逐而被标记过；
拒绝应用程序的当前DMA缓冲区。由于这个DMA缓冲区也许已包含重要的状态度量，因而这个进程可能被中断并且可以通知应用程序所述错误。

图 6

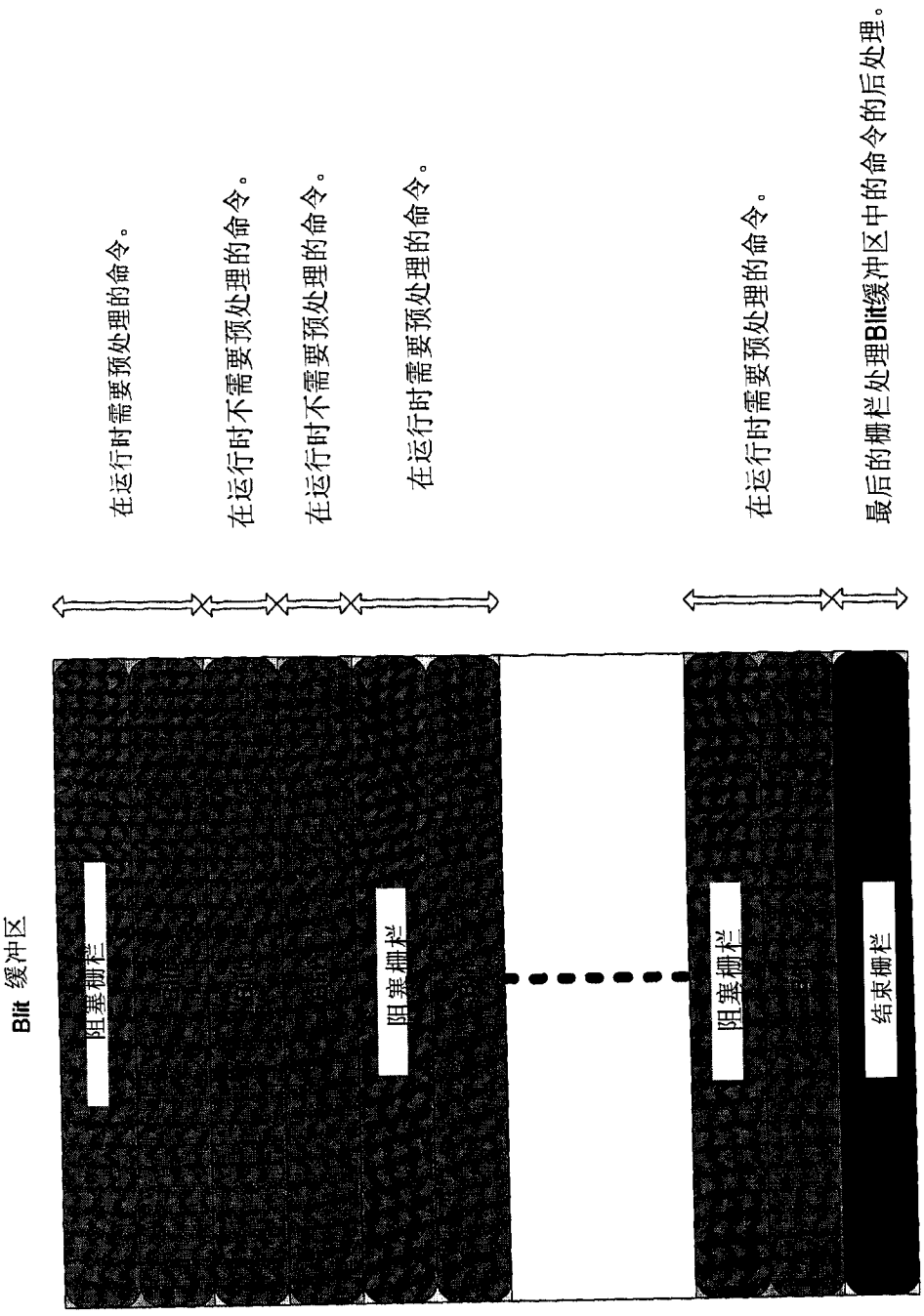

Blit 缓冲区
阻塞栅栏
阻塞栅栏
阻塞栅栏
结束栅栏
在运行时需要预处理的命令。
在运行时不需要预处理的命令。
在运行时不需要预处理的命令。
在运行时需要预处理的命令。
在运行时需要预处理的命令。
最后的栅栏处理Blit缓冲区中的命令的后处理。

图 7

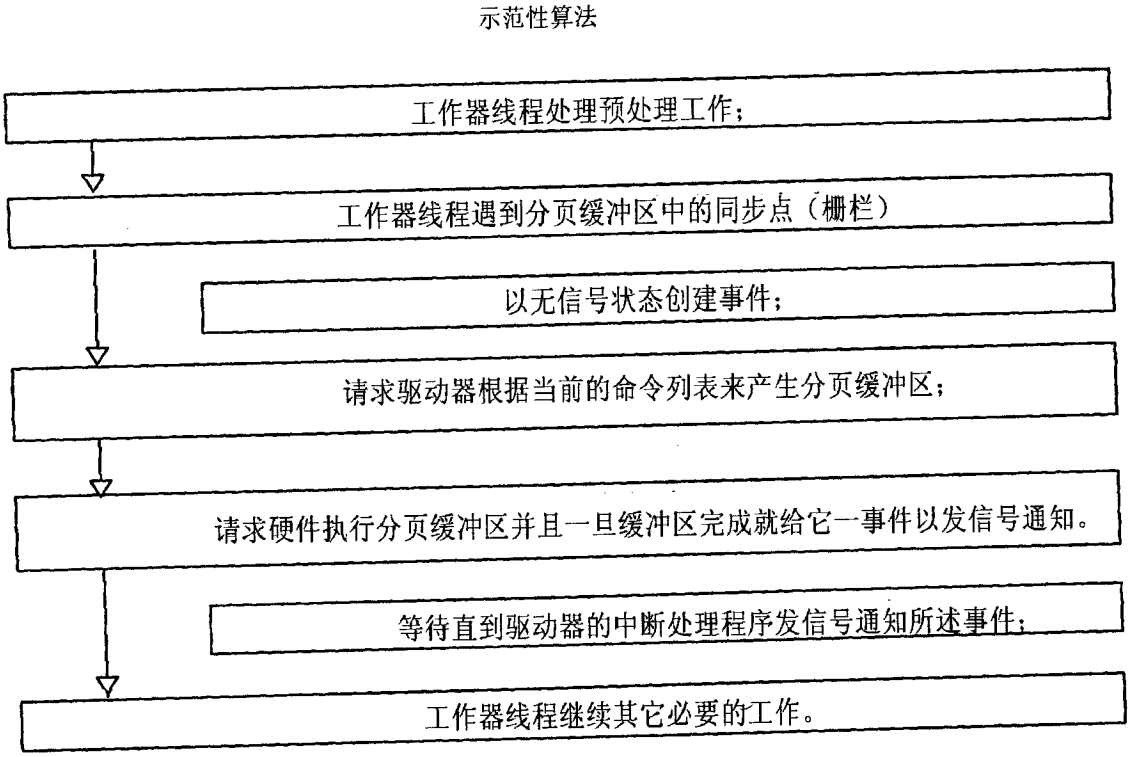

示范性算法
工作器线程处理预处理工作；
工作器线程遇到分页缓冲区中的同步点（栅栏）
以无信号状态创建事件；
请求驱动器根据当前的命令列表来产生分页缓冲区；
请求硬件执行分页缓冲区并且一旦缓冲区完成就给它一事件以发信号通知。
等待直到驱动器的中断处理程序发信号通知所述事件；
工作器线程继续其它必要的工作。

图 8

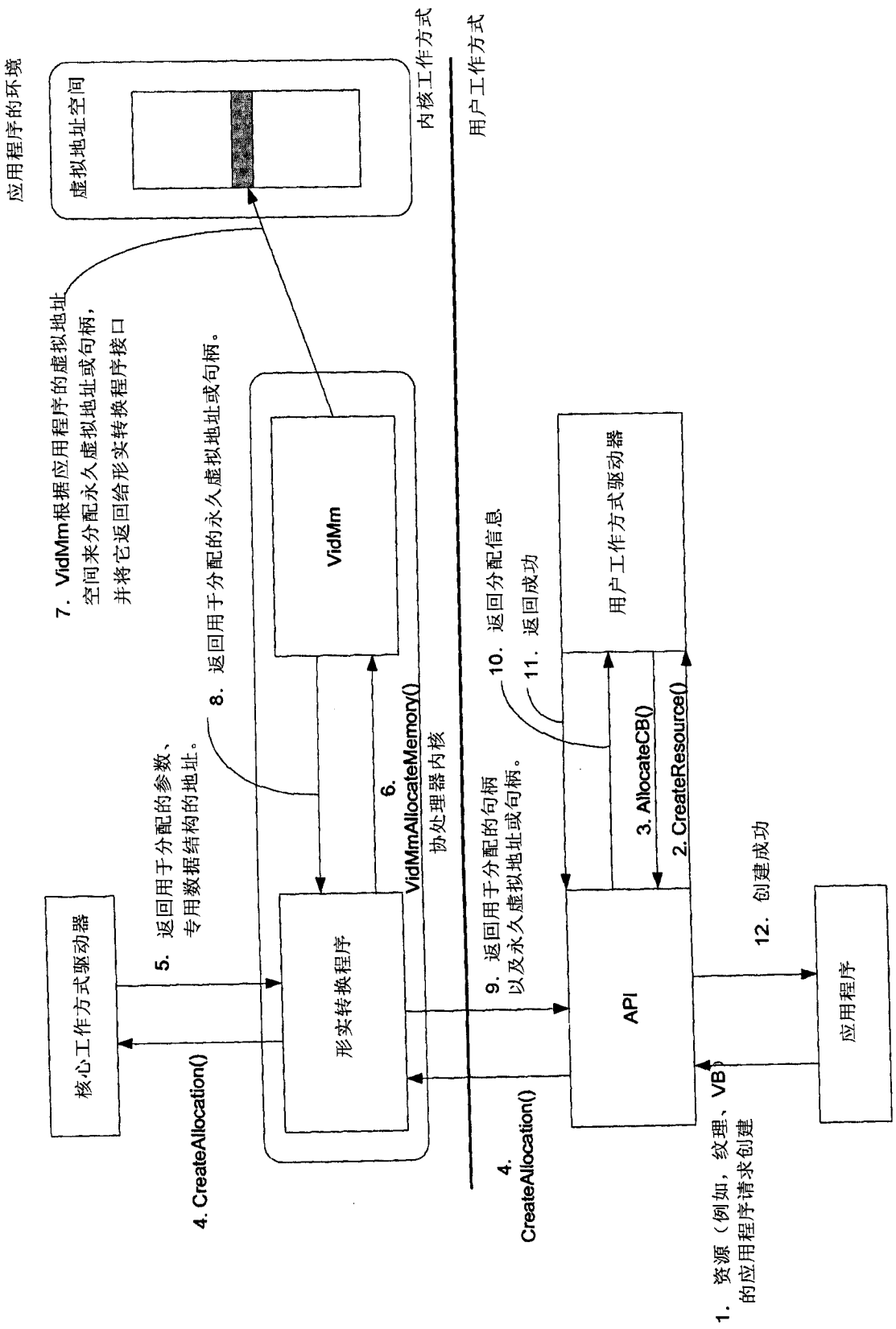

应用程序的环境
虚拟地址空间
内核工作方式
用户工作方式
7. VidMm根据应用程序的虚拟地址
空间来分配永久虚拟地址或句柄，
并将它返回给形实转换程序接口
8. 返回用于分配的永久虚拟地址或句柄。
VidMm
6.
VidMmAllocateMemory()
协处理器内核
5. 返回用于分配的参数、
专用数据结构的地址。
4. CreateAllocation()
核心工作方式驱动器
形实转换程序
9. 返回用于分配的句柄
以及永久虚拟地址或句柄。
10. 返回分配信息
11. 返回成功
用户工作方式驱动器
3. AllocateCB()
2. CreateResource()
4.
CreateAllocation()
API
12. 创建成功
1. 资源（例如，纹理、VB）
的应用程序请求创建
应用程序

图 9

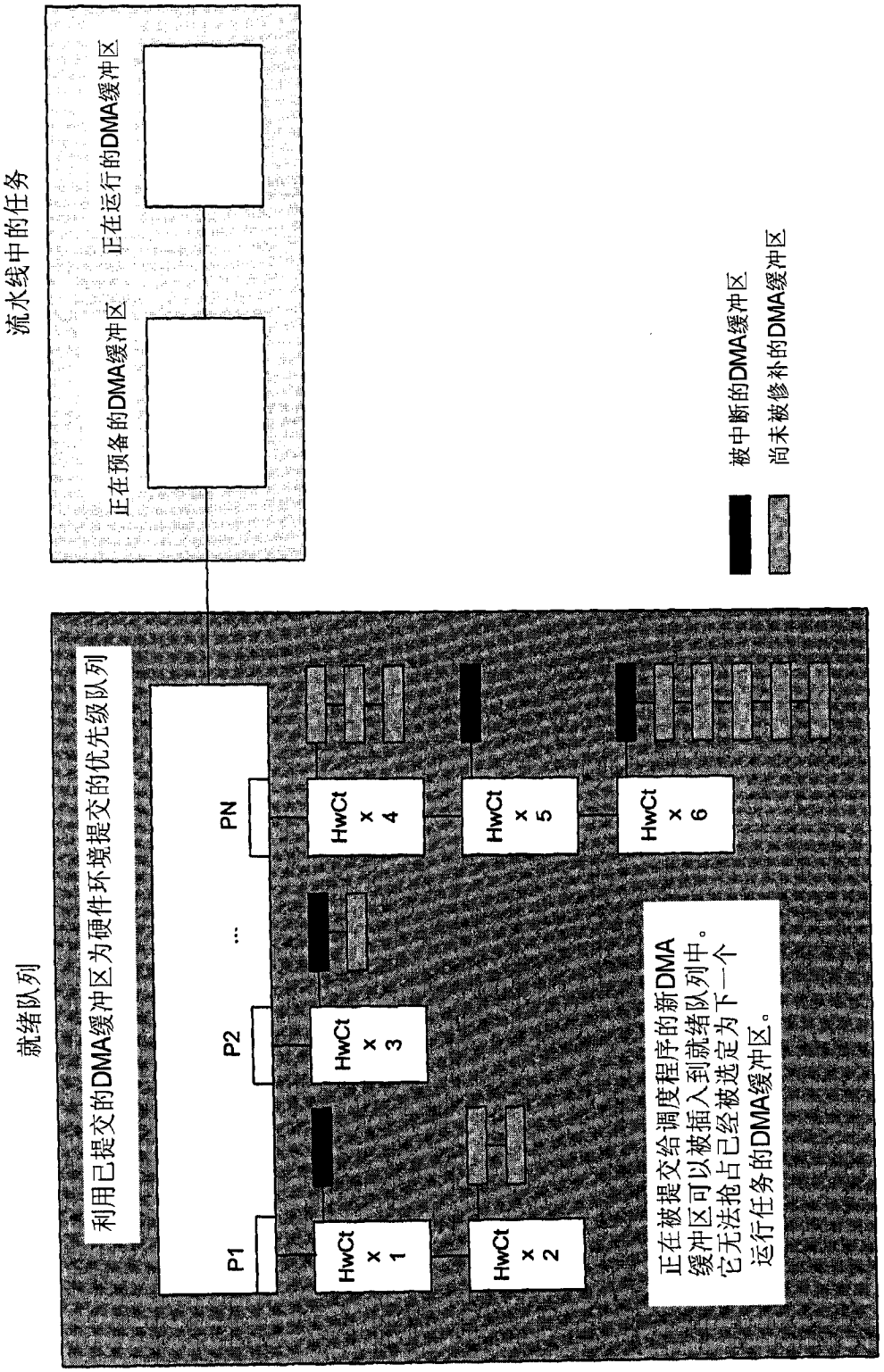

流水线中的任务
正在预备的DMA缓冲区
正在运行的DMA缓冲区
被中断的DMA缓冲区
尚未被修补的DMA缓冲区
就绪队列
利用已提交的DMA缓冲区为硬件环境提交的优先级队列
P1
P2
...
PN
HwCt x 1
HwCt x 2
HwCt x 3
HwCt x 4
HwCt x 5
HwCt x 6
正在被提交给调度程序的新DMA缓冲区可以被插入到就绪队列中。它无法抢占已经被选定为下一个运行任务的DMA缓冲区。

图 10

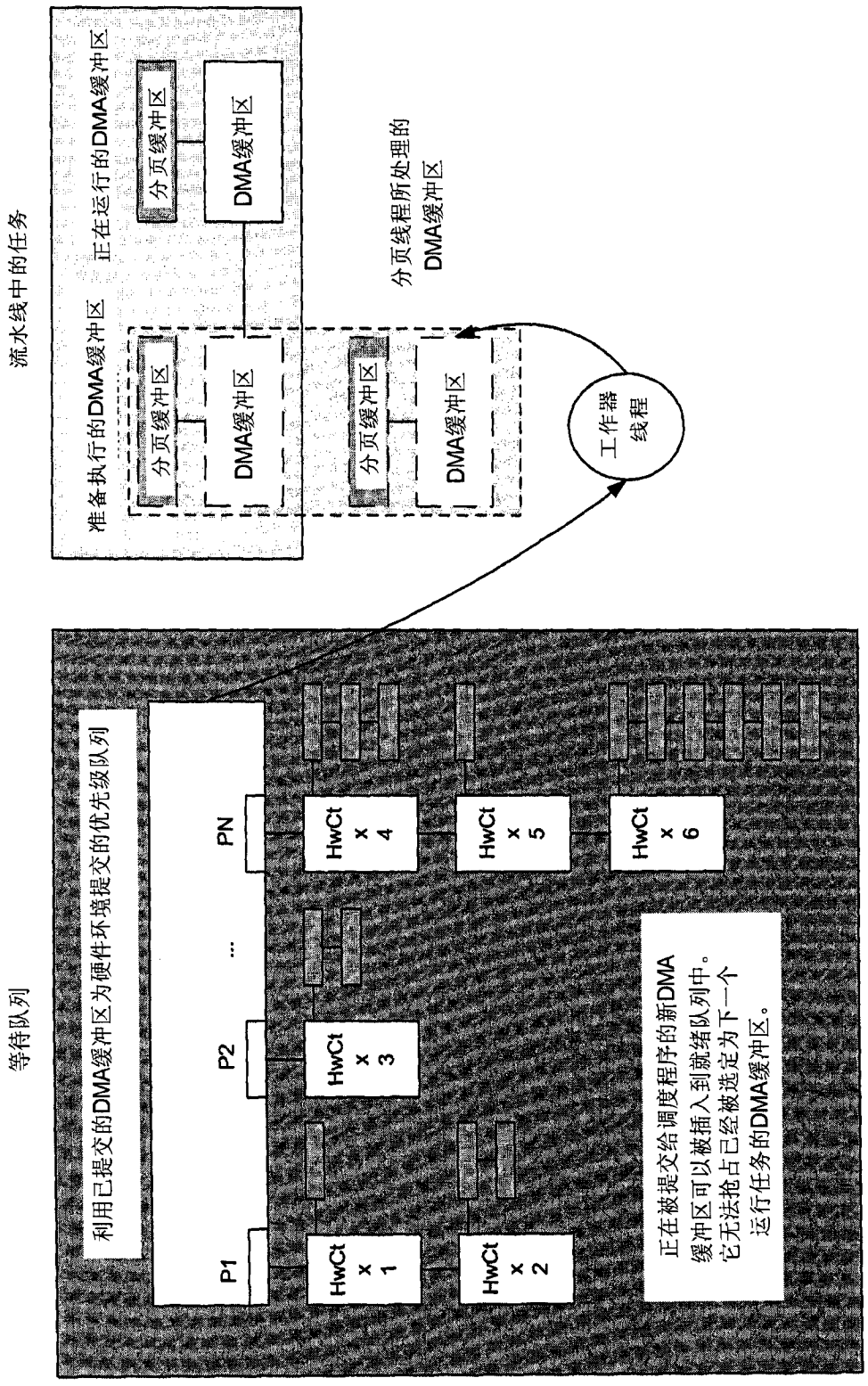

流水线中的任务
准备执行的DMA缓冲区
正在运行的DMA缓冲区
分页缓冲区
DMA缓冲区
分页缓冲区
DMA缓冲区
分页线程所处理的
DMA缓冲区
分页缓冲区
DMA缓冲区
工作器
线程
等待队列
利用已提交的DMA缓冲区为硬件环境提交的优先级队列
P1
P2
...
PN
HwCt x 1
HwCt x 2
HwCt x 3
HwCt x 4
HwCt x 5
HwCt x 6
正在被提交给调度程序的新DMA
缓冲区可以被插入到就绪队列中。
它无法抢占已经被选定为下一个
运行任务的DMA缓冲区。

图 11

```
进程A：提交（中断请求无源，描绘线程环境）
    如果没有DMA缓冲区被准备或准备执行。
    如果当前DMA缓冲区的所有内存资源都已经存在于内存中，
    如果协处理器空闲，就把DMA缓冲区给予协处理器。
    否则，就把DMA缓冲区插入到准备执行的时隙中。
    如果一些内存资源需要被页入，则将DMA缓冲区提交给分页线程。
        否则，则在当前环境列表的结尾处插入DMA缓冲区。

            进程B：量子期满（中断请求设备，任何线程环境）
                如果当前任务仍然正在处理其分页缓冲区，
                    则允许当前任务继续运行。
                    设置当前的量子为期满。
                否则，如果下一个DMA缓冲区准备好被运行，
                    则把当前环境的当前优先级设置为其基本优先级。
                    把当前环境移到其优先级的队列结尾处。
                设置量子为运行（未期满）。
            选择下一个DMA缓冲区以执行。
                如果DMA缓冲区需要分页，则将它提交给分页线程。
                否则，所有的内存资源都已存在；仅仅把DMA缓冲区插入就绪时隙。

                允许当前任务继续运行。
                设置当前的量子为"期满"。
            进程C：任务结束（中断请求设备，任何线程环境）
                如果下一个DMA缓冲区准备被运行，
                就把下一个DMA缓冲区提交给协处理器。
                重置量子为"运行"(未期满);
                选择要执行的下一个任务。
                    如果DMA缓冲区需要分页，则将它提交给分页线程。
                    否则，所有的内存资源都已存在；仅仅把DMA缓冲区插入就绪时隙。
                    否则，下一个任务未就绪；
                    如果分页线程当前继续对下一个DMA缓冲区起作用，就暂时提高工作器线程的优先级以便使它
                    尽可能快地结束其工作。
```

图 12(A)

```
进程D：分页线程(中断请求无源，系统线程)
    设置当前的驱逐策略为第一策略。
    请求内存管理器页入资源列表。
    如果所有资源都被成功地页入，
        就把分页缓冲区和DMA缓冲区移到准备执行的时隙。
        如果当前DMA缓冲区的量子期满，
        则把下一个DMA缓冲区提交给协处理器。
        选择要执行的下一个DMA缓冲区。
          如果DMA缓冲区需要分页，则将它提交给分页线程。
              否则，所有内存资源都已存在，仅仅把DMA缓冲区插入就绪时隙
      否则如果由于分页缓冲区满而使内存管理器失败，
                  等待直到当前DMA缓冲区的量子结尾或结束。
                  把分页缓冲区提交给协处理器。
                  等待直到分页缓冲区完成。
                  返回请求内存管理器页入资源列表的其余内容。
                  否则如果由于没有足够的可用资源而使内存管理器失败，
      如果已通过最后的驱逐策略，
      则撤销资源移动，或者运行分页缓冲区。
          拒绝DMA缓冲区。
          完成。
          否则如果当前的驱逐策略高于应用程序冲突。
          如果DMA缓冲区尚未分割。
          则以与当前页入的资源最接近的点分割DMA缓冲区。
          把分页缓冲区和分割的DMA缓冲区移到准备执行的时隙。
          把其余的DMA缓冲区移回到环境就绪队列的头部。

      如果当前DMA缓冲区的量子期满，
      则把下一个DMA缓冲区提交给协处理器。
      重置量子为运行(未期满)。
          选择要执行的下一个DMA缓冲区。
          如果DMA缓冲区需要分页，则将它提交给分页线程。
                否则，所有内存资源都已存在；仅仅把DMA缓冲区插入就绪时隙

          如果VidMm返回错误，表示用当前策略不能标记任何内存，
                则提高驱逐策略。
                返回到驱逐策略校验的起始处。
                      否则，标记一些内存。
                      返回试图页入资源。
```

图 12(B)

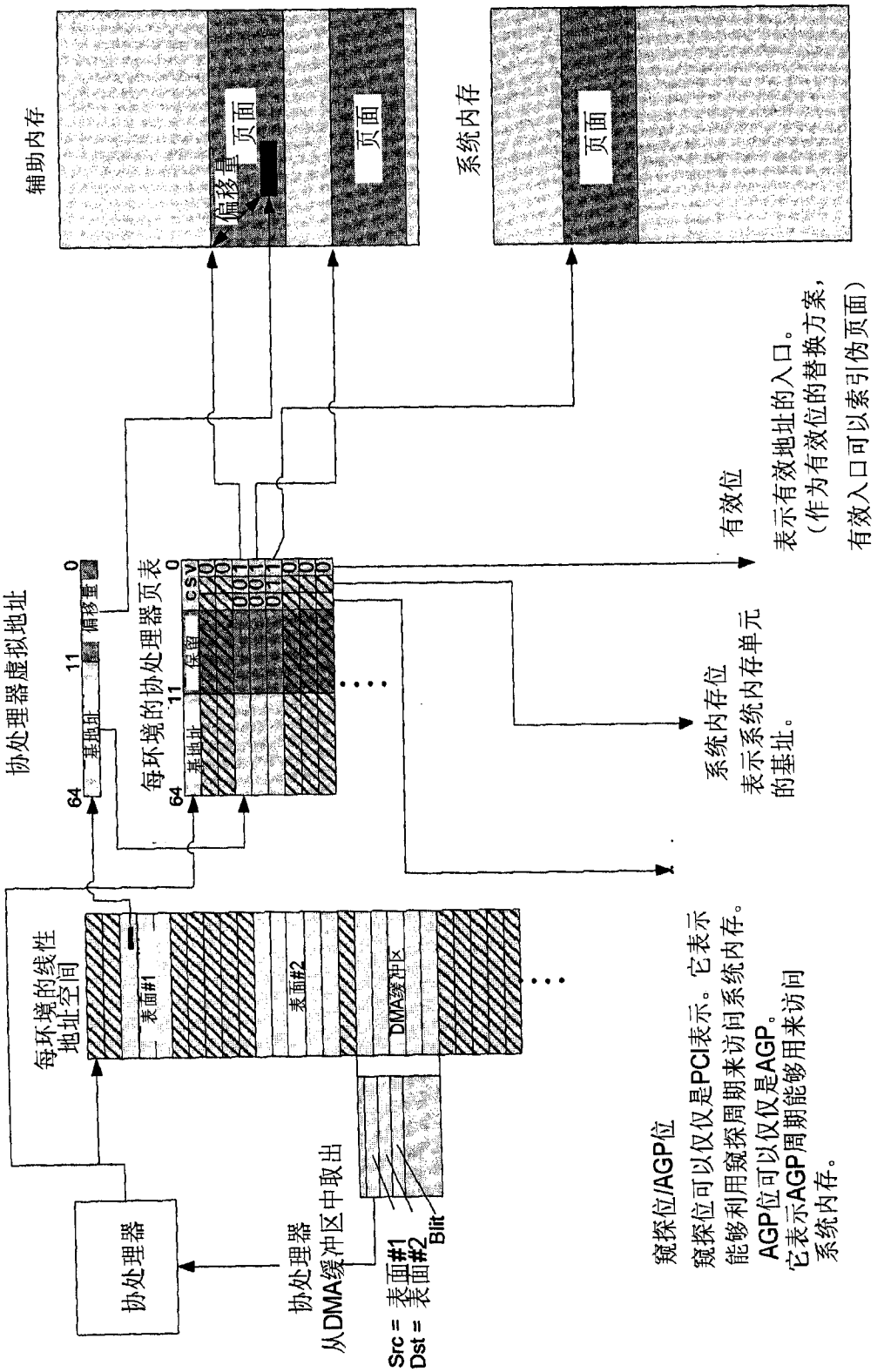

辅助内存
偏移量
页面
页面
系统内存
页面
协处理器虚拟地址
64
基地址
11
偏移量
0
每环境的协处理器页表
0
64
基地址
11
保留
csv
有效位
表示有效地址的入口。
（作为有效位的替换方案，
有效入口可以索引伪页面）
系统内存位
表示系统内存单元
的基址。
每环境的线性
地址空间
表面#1
表面#2
DMA缓冲区
协处理器
协处理器
从DMA缓冲区中取出
Src = 表面#1
Dst = 表面#2
Blit
窥探位/AGP位
窥探位可以仅仅是PCI表示。它表示
能够利用窥探周期来访问系统内存。
AGP位可以仅仅是AGP。
它表示AGP周期能够用来访问
系统内存。

图 13

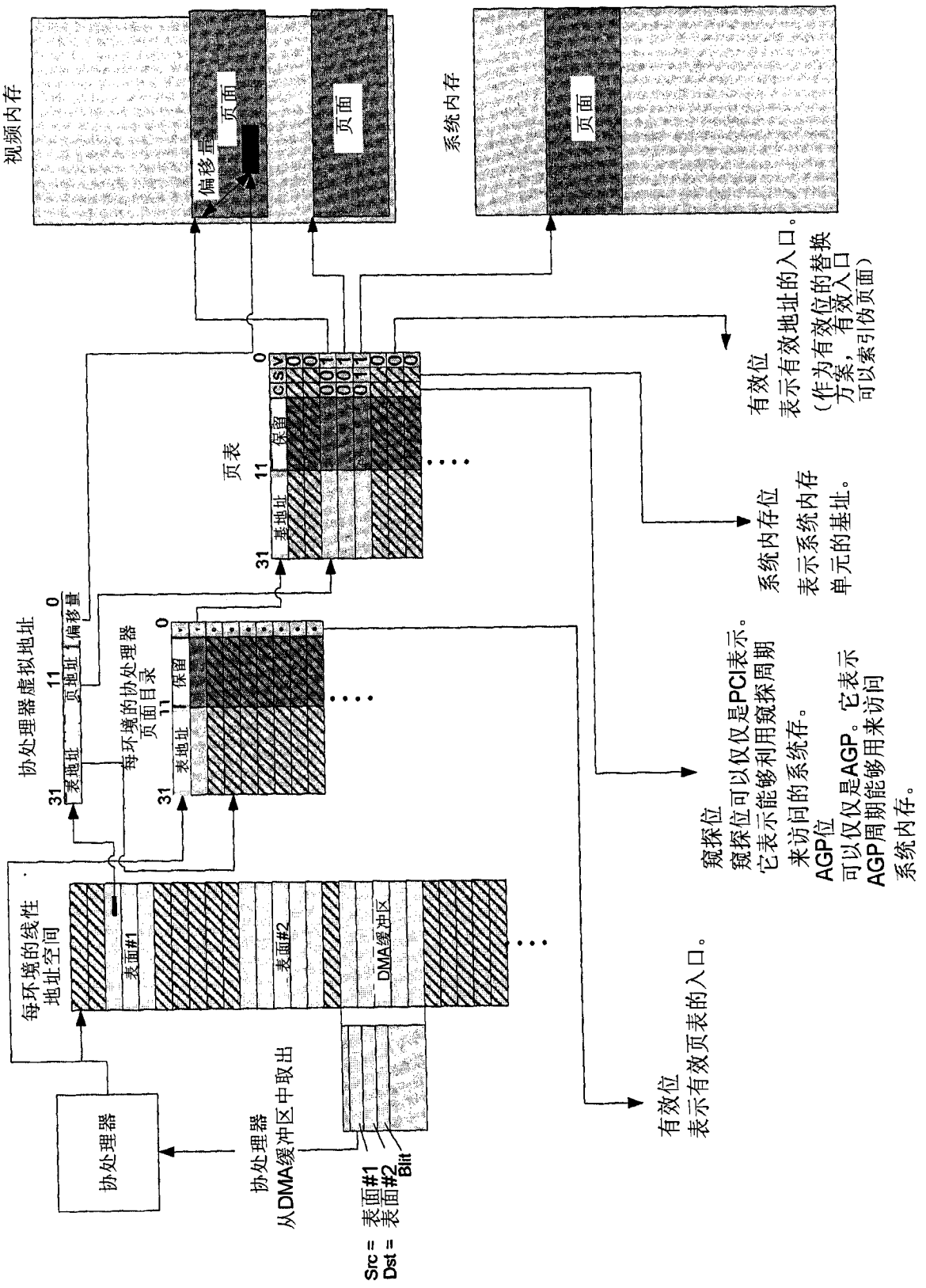

视频内存
偏移量
页面
页面
系统内存
页面
页表
保留
基地址
C
S
V
31
11
0
协处理器虚拟地址
页地址
偏移量
表地址
每环境的协处理器
页面目录
表地址
保留
每环境的线性
地址空间
表面#1
表面#2
DMA缓冲区
协处理器
协处理器
从DMA缓冲区中取出
Src = 表面#1
Dst = 表面#2
Blit
有效位
表示有效页表的入口。
窥探位
窥探位可以仅仅是PCI表示。
它表示能够利用窥探周期
来访问的系统存。
AGP位
可以仅仅是AGP。它表示
AGP周期能够用来访问
系统内存。
系统内存位
表示系统内存
单元的基址。
有效位
表示有效地址的入口。
(作为有效位的替换
方案，有效入口
可以索引伪页面)

图 14

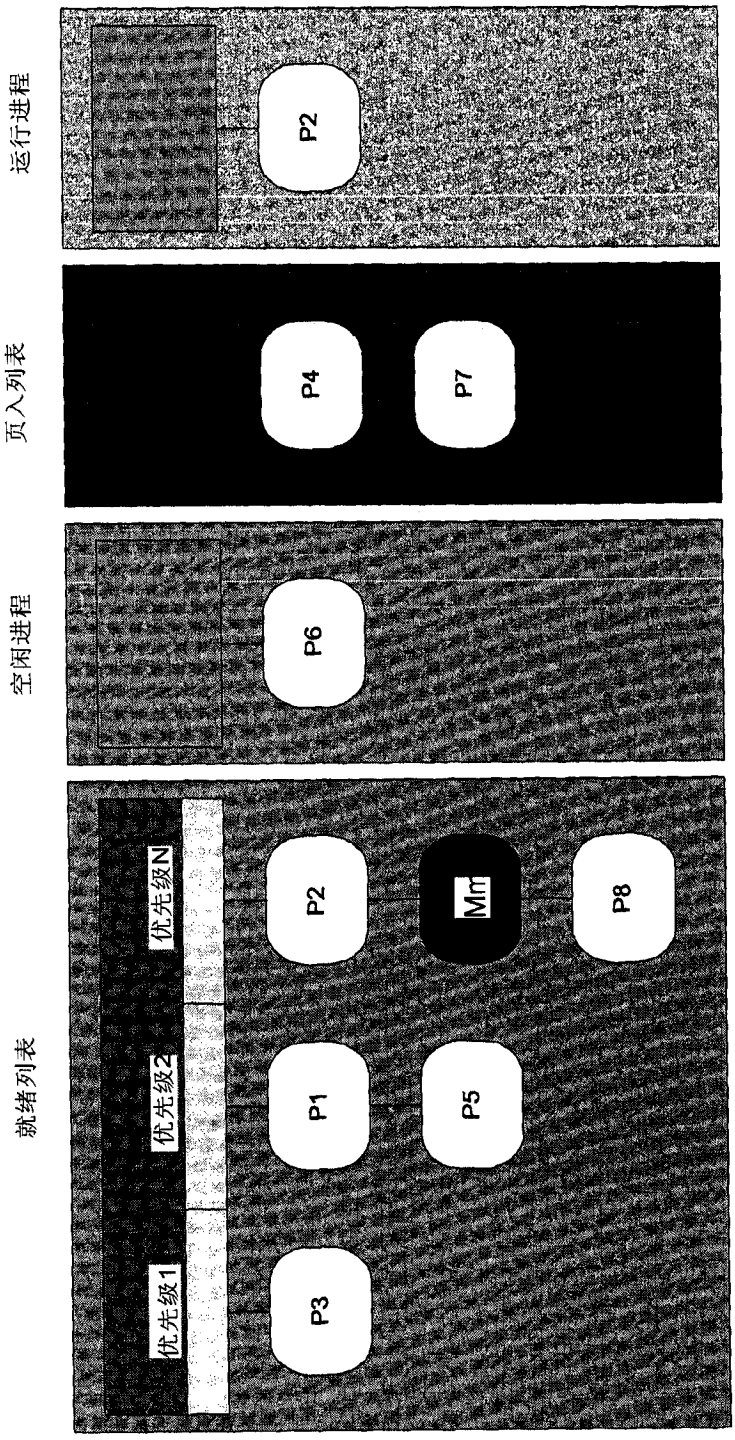


图 15

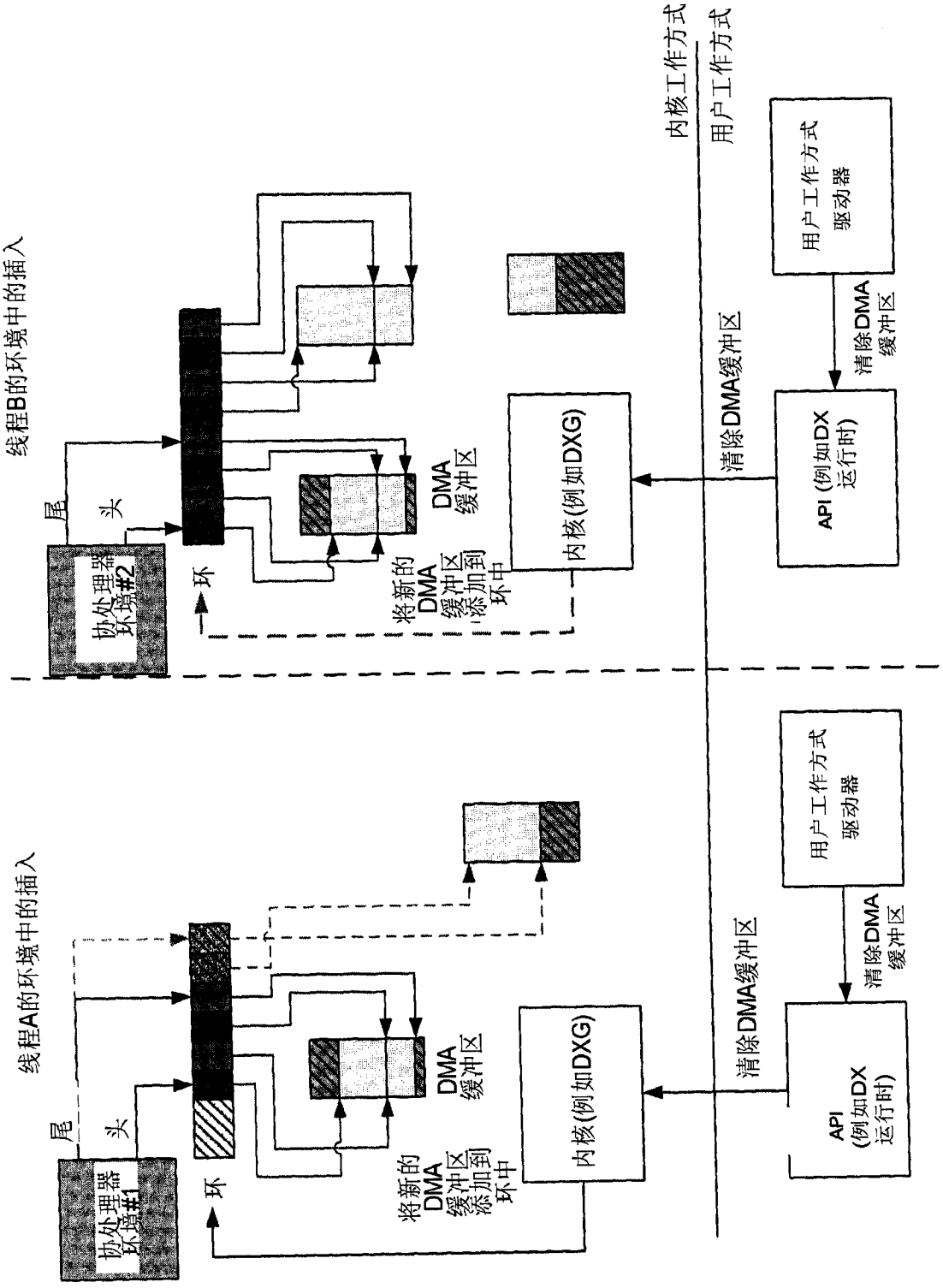

线程B的环境中的插入
线程A的环境中的插入
内核工作方式
用户工作方式
协处理器环境#2
协处理器环境#1
尾
头
环
将新的DMA缓冲区添加到环中
DMA缓冲区
内核(例如DXG)
清除DMA缓冲区
API (例如DX运行时)
用户工作方式驱动器
清除DMA缓冲区

图 16

```
进程A：提交（中断请求无源，描绘线程环境，保持协处理器环境互斥）
    获得VIDMM_lock
    处理给定的一列资源，并且更新与该进程中的分配有关的使用信息。
    释放VIDMM锁

    调用驱动器把当前的DMA缓冲区插入环中。
    如果驱动器成功，
        如果环境空闲。
            则在其当前优先级队列的尾部把环境插回就绪列表。
        如果没有未决的环境转换且当前环境的优先级低于当前运行的环境
            则调用驱动器把环境转换到该环境
            通知存在未决的环境转换
    释放调度程序锁定。
    如果驱动器失败，环为满。
        等待当空间可用时会通知的事件。
        在等待后，返回获取调度程序锁定。
    如果在DMA缓冲区中有足够的剩余空间用于另一次提交
        则用当前DMA缓冲区返回用户工作方式。
    从环境库中获取新的DMA缓冲区
    如果DMA库此时不给出另一缓冲区
        则等待当DMA缓冲区被插回到库中时会通知的事件
        当等待结束后，返回试图取得新的DMA缓冲区。
    把新的DMA缓冲区返回到用户工作方式

进程B：环境转换完成（中断请求设备，任何线程环境）
    获取调度程序锁定。
        如果较高优先级的 环境现在准备执行。
            则调用驱动器把环境转换到最高优先级的环境。
        否则
            通知目前没有未决的环境转换。

    释放调度程序锁定

进程C：量子期满（中断请求设备，任何线程环境）
    获取调度程序锁定。
    将环境的当前优先级重置为其基本优先级。
    把环境插回到其当前优先级的队列结尾处
    如果目前没有未决的环境转换。
        则请求驱动器把环境转换到最高优先级的环境。
    释放调度程序锁定
```

图 17(A)

进程D：任务结束(中断请求设备，任何线程环境)
获取调度程序锁定
询问驱动器该环境是否真的为空。
如果环境真的为空。
则把环境的当前优先级重置为其基本优先级。
把环境插入空闲列表。
如果环境不是真的为空
如果当前没有未决的环境转换。
则请求驱动器把环境转换到最高优先级的环境
释放调度程序锁定。

进程E：页面错误(中断请求设备，任何线程环境)
获取调度程序锁定
从就绪列表中删除环境
根据基本运算把环境插入页面列表
如果页入线程当前正在睡眠
则排队DPC来通知唤醒工作器线程。
如果当前没有未决的环境转换。
则请求驱动器把环境转换到最高优先级的环境
释放调度程序锁定。

进程F：解决错误(中断请求设备，任何线程环境)
获取调度程序锁定
从页面列表中删除环境
把环境插回其当前优先级的就绪列表
如果当前没有未决的环境转换，且当前环境的优先级高于当前运行的环境
则请求驱动器把环境转换到最高优先级的环境
释放调度程序锁定。

图 17(B)

进程G：页入工作器线程
仔细检查页入队列中的环境列表。选出最高优先级的环境。
向驱动器请求对环境取得进展所需的资源列表。
获取VIDMM锁定
为进展所需的每一次分配找到一个位置
使被驱逐的分配的虚拟地址或句柄无效
请求驱动器用内存转移命令填充DMA缓冲区，所述内存转移命令对于把所需的分配带到它们所选的地点是必要的。
释放VIDMM锁定
提交VidMm环境作为常规的协处理器环境
如果环境列表为空，则睡眠直到添加了一项位置。
返回到循环的开始处。

进程H：周期定时器(无源级，系统线程环境)
获取调度程序锁定
提高每个环境的当前优先级
释放调度程序锁定

图 17(C)

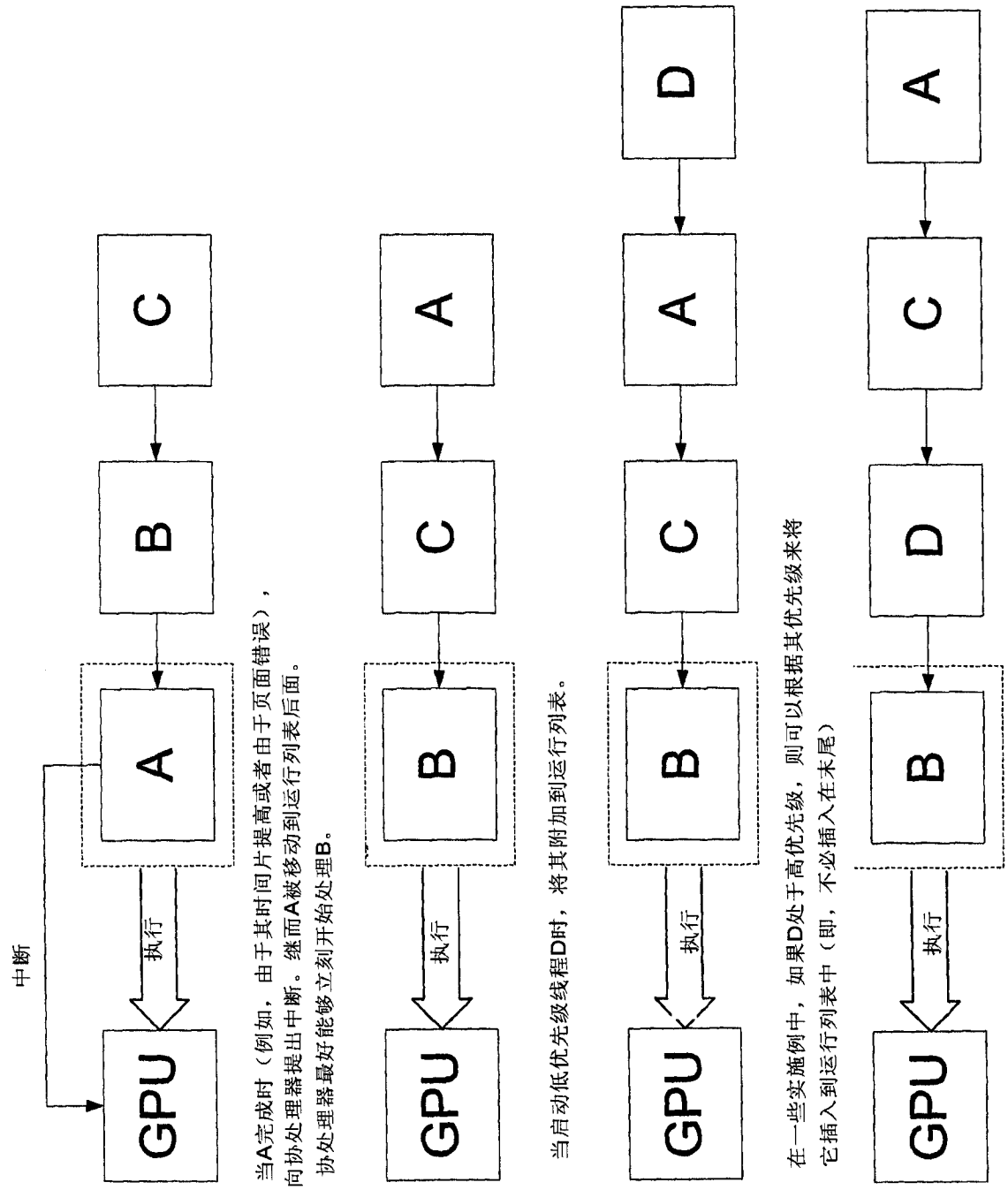

中断
GPU
执行
A
B
C
当A完成时（例如，由于其时间片提高或者由于页面错误），向协处理器提出中断。继而A被移动到运行列表后面。协处理器最好能够立刻开始处理B。
GPU
执行
B
C
A
当启动低优先级线程D时，将其附加到运行列表。
GPU
执行
B
C
A
D
在一些实施例中，如果D处于高优先级，则可以根据其优先级来将它插入到运行列表中（即，不必插入在末尾）
GPU
执行
B
D
C
A

图 18

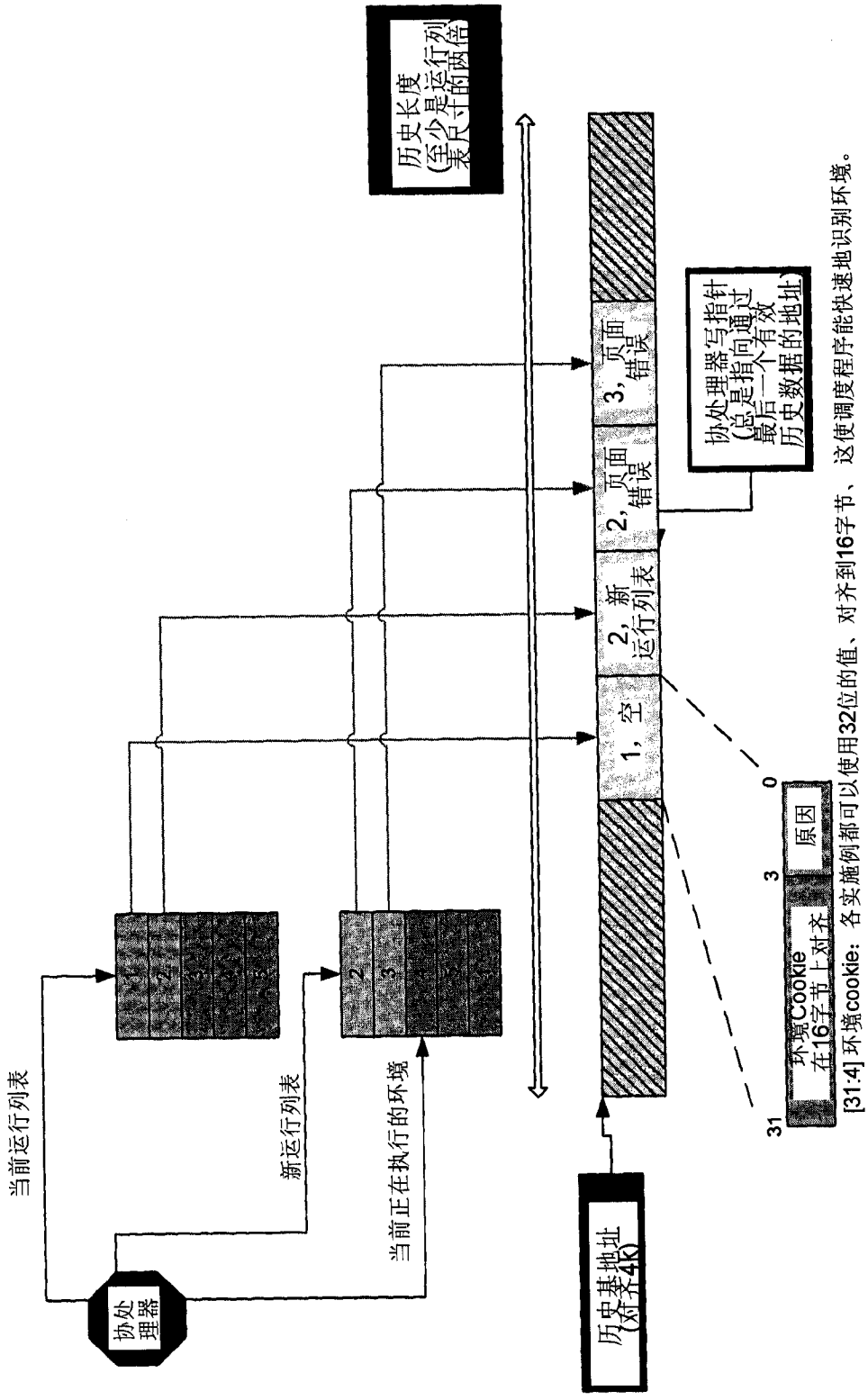

历史长度
(至少是运行列表尺寸的两倍)
协处理器写指针
(总是指向通过最后一个有效历史数据的地址)
当前运行列表
新运行列表
当前正在执行的环境
协处理器
历史基地址
(对齐4K)
1, 空
2, 新运行列表
2, 页面错误
3, 页面错误
环境Cookie
在16字节上对齐
原因
31
3
0
[31:4] 环境cookie：各实施例都可以使用32位的值、对齐到16字节、这使调度程序能快速地识别环境。
与环境相关联的值（环境定义的一部分）。
[3:0] 原因：环境转换的原因：
0: 页面错误
1: 空
2: 同步对象
3: 新运行列表
4: 一般保护错误。

图 19

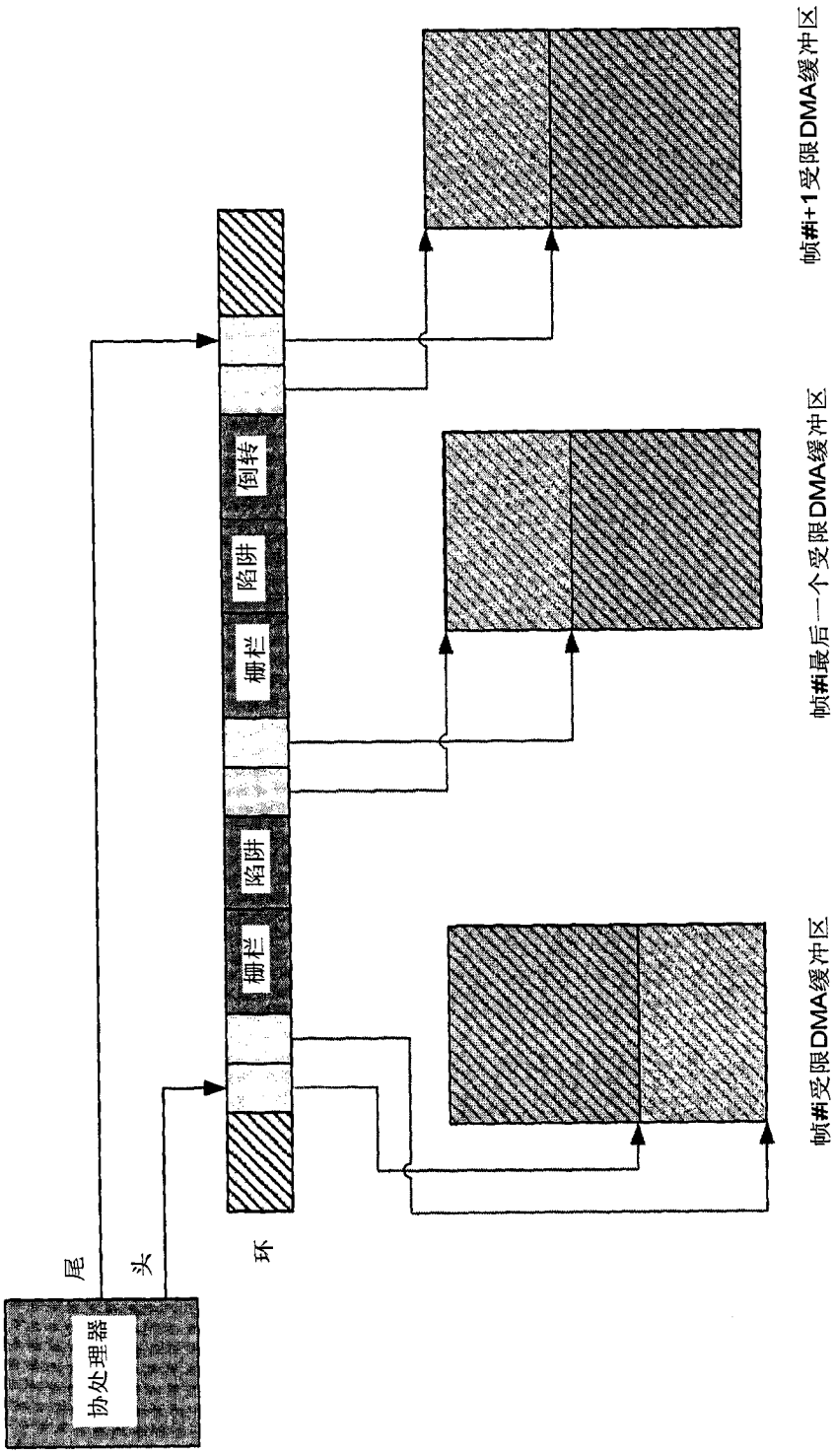


图 20

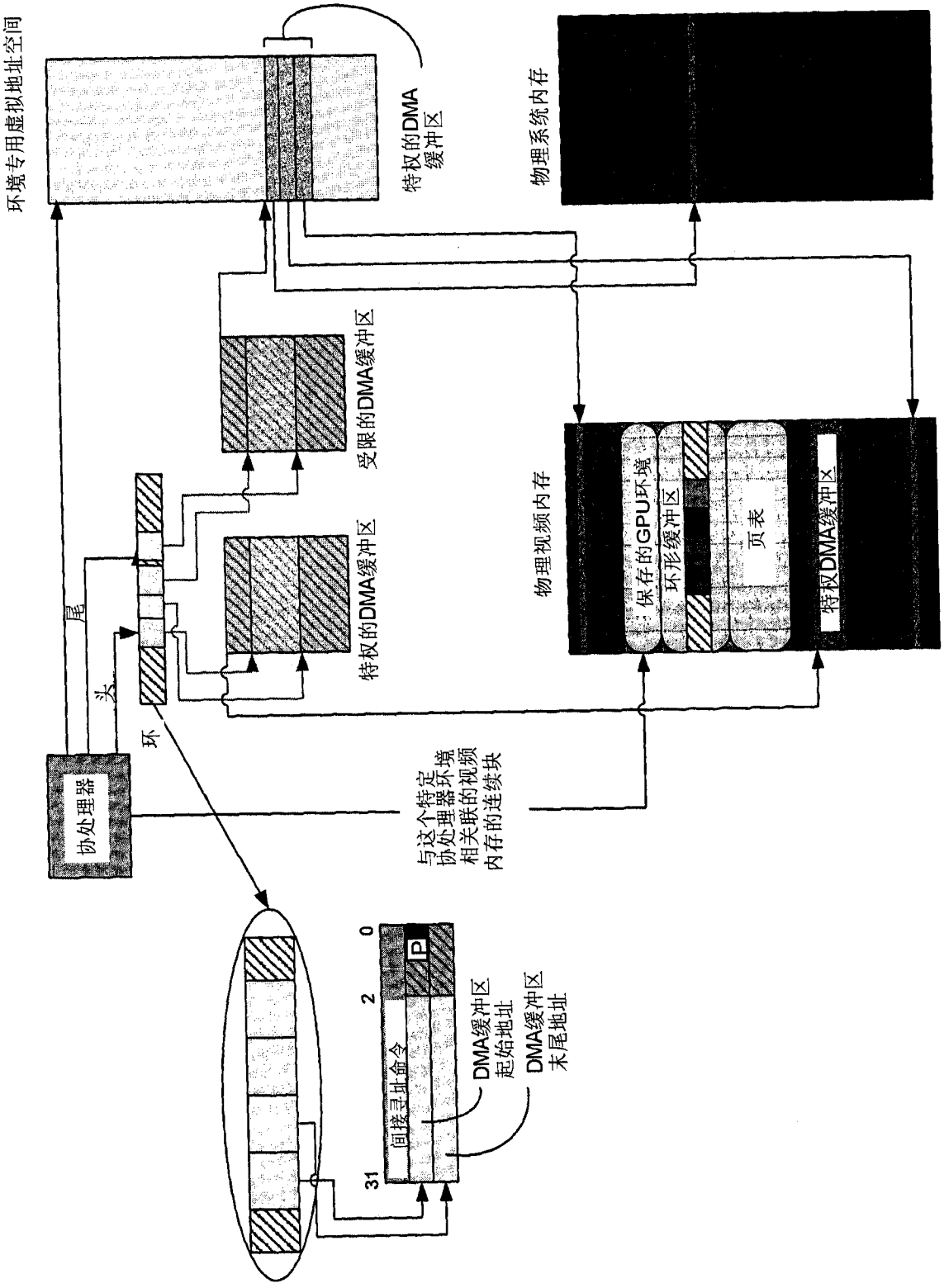

环境专用虚拟地址空间
特权的DMA缓冲区
物理系统内存
受限的DMA缓冲区
特权的DMA缓冲区
物理视频内存
保存的GPU环境
环形缓冲区
页表
特权DMA缓冲区
尾
头
环
协处理器
与这个特定协处理器环境相关联的视频内存的连续块
0
2
P
31
间接寻址命令
DMA缓冲区起始地址
DMA缓冲区末尾地址

图 21

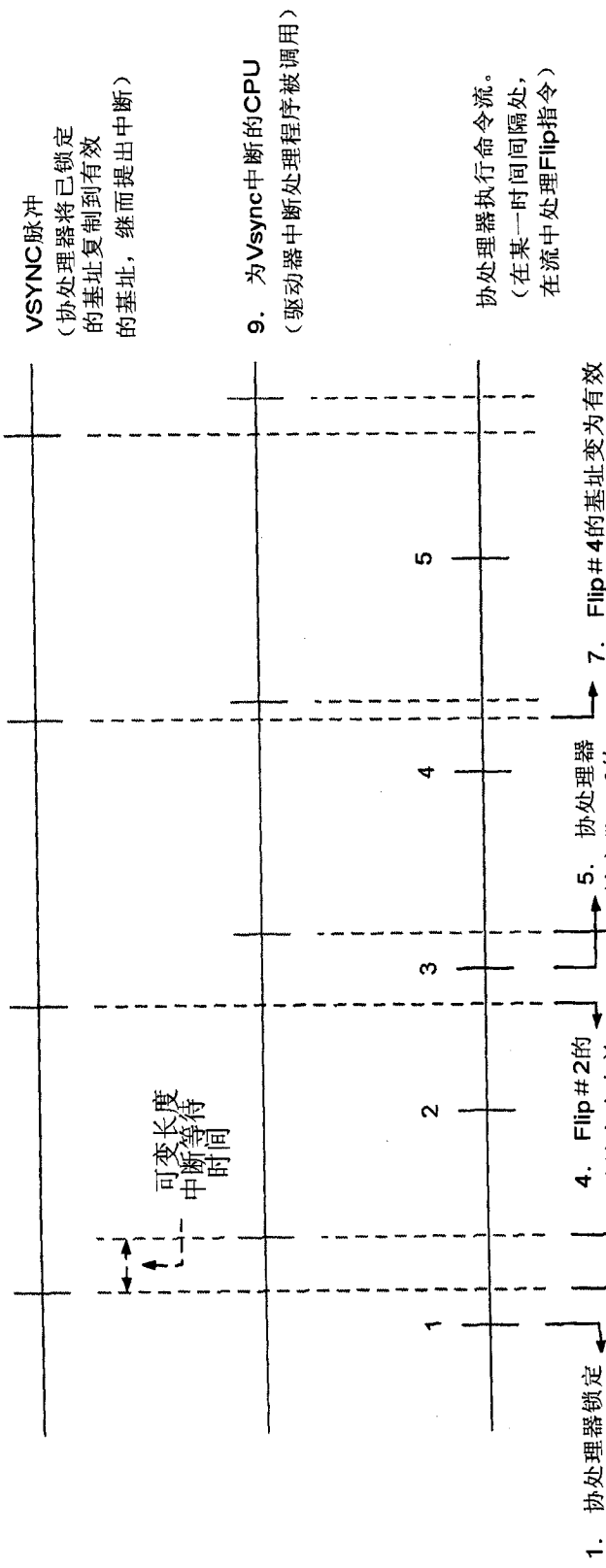

VSYNC脉冲
（协处理器将已锁定
的基址复制到有效
的基址，继而提出中断）
9. 为Vsync中断的CPU
（驱动器中断处理程序被调用）
协处理器执行命令流。
（在某一时间间隔处，
在流中处理Flip指令）
可变长度
中断等待
时间
1
2
3
4
5
1. 协处理器锁定
flip＃1的新基址。
2. Flip＃1的基址
变为有效。
3. 驱动器采样显示器
的当前基址并通知
调度程序完成了flip＃1。
4. Flip＃2的
基址变为有效。
5. 协处理器
锁定flip＃3的
新基址
6. 驱动器采样显示器的
当前基址并通知调度
程序完成了flip＃2。
在这一点，当前基址
不同于已锁定的基址。
协处理器可能不确认
完成了flip＃3。
7. Flip＃4的基址变为有效
Flip＃3被忽略。

图 22

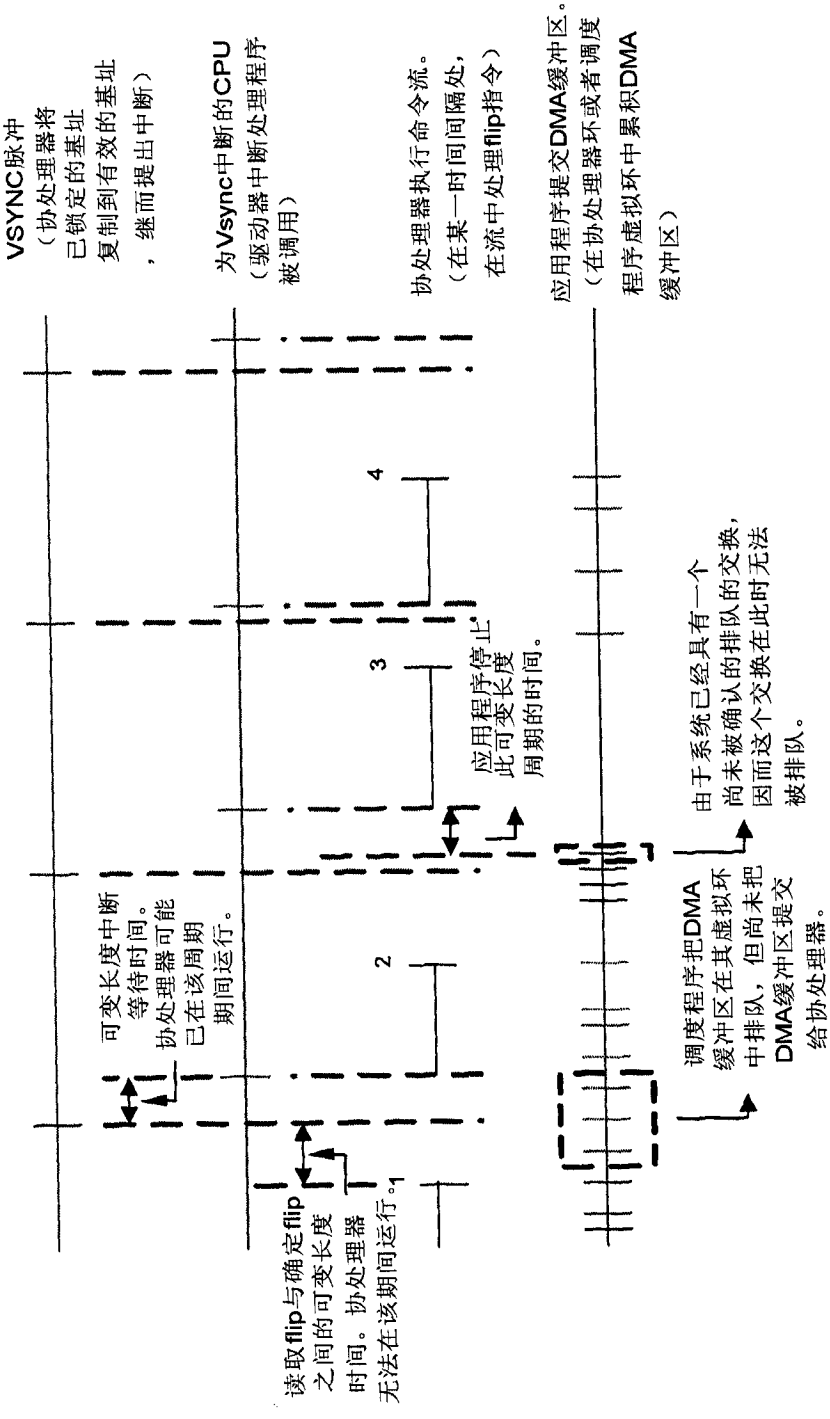

VSYNC脉冲
（协处理器将
已锁定的基址
复制到有效的基址
，继而提出中断）
为Vsync中断的CPU
（驱动器中断处理程序
被调用）
协处理器执行命令流。
（在某一时间间隔处，
在流中处理flip指令）
应用程序提交DMA缓冲区。
（在协处理器环或者调度
程序虚拟环中累积DMA
缓冲区）
可变长度中断
等待时间。
协处理器可能
已在该周期
期间运行。
2
3
4
应用程序停止
此可变长度
周期的时间。
读取flip与确定flip
之间的可变长度
时间。协处理器
无法在该期间运行。1
调度程序把DMA
缓冲区在其虚拟环
中排队，但尚未把
DMA缓冲区提交
给协处理器。
由于系统已经具有一个
尚未被确认的排队的交换，
因而这个交换在此时无法
被排队。

图 23

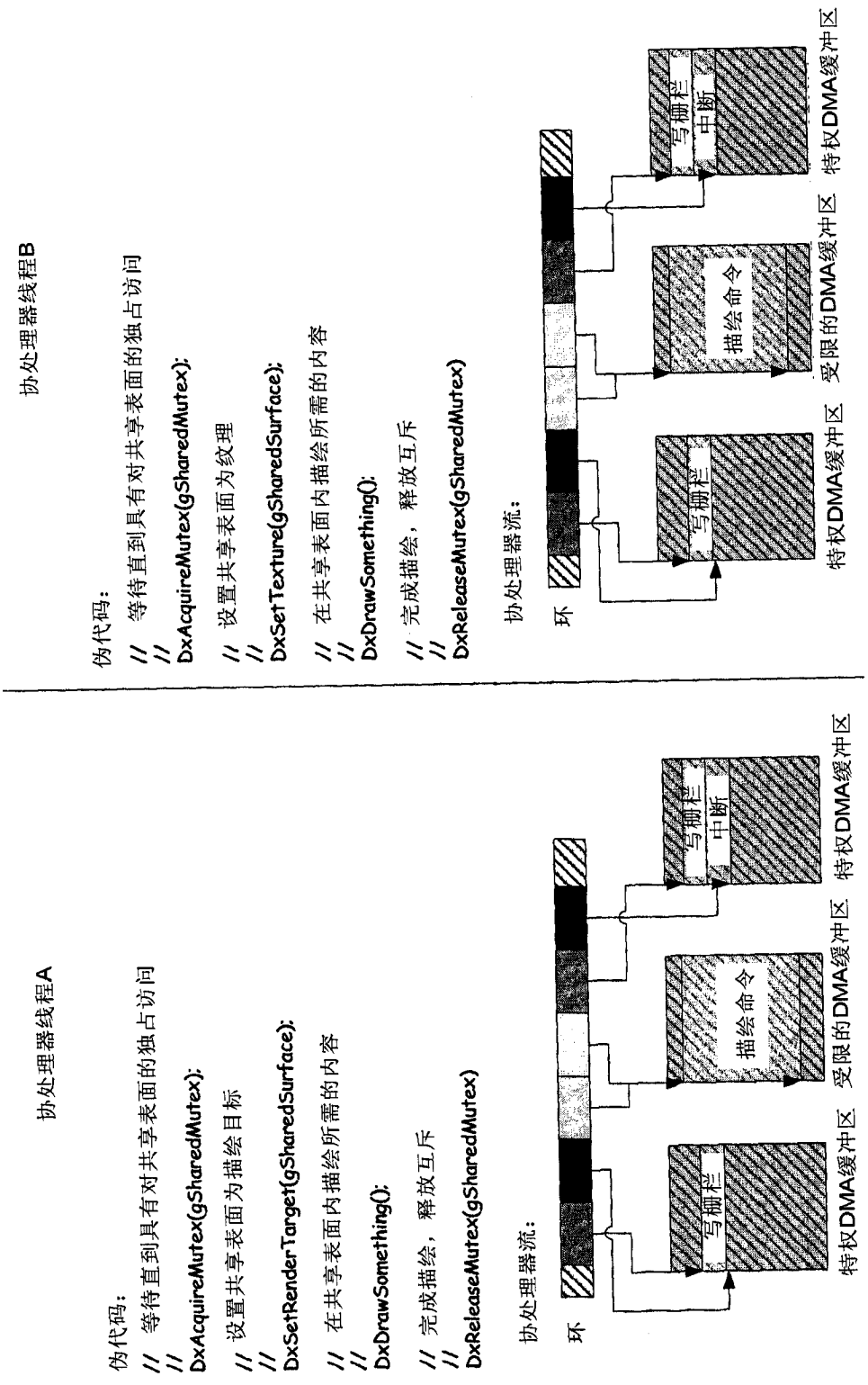

协处理器线程A
伪代码：
// 等待直到具有对共享表面的独占访问
//
DxAcquireMutex(gSharedMutex);
// 设置共享表面为描绘目标
//
DxSetRenderTarget(gSharedSurface);
// 在共享表面内描绘所需的内容
//
DxDrawSomething();
// 完成描绘，释放互斥
//
DxReleaseMutex(gSharedMutex)
协处理器流：
环
写栅栏
描绘命令
写栅栏
中断
特权DMA缓冲区
受限的DMA缓冲区
特权DMA缓冲区
协处理器线程B
伪代码：
// 等待直到具有对共享表面的独占访问
//
DxAcquireMutex(gSharedMutex);
// 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：
环
写栅栏
描绘命令
写栅栏
中断
特权DMA缓冲区
受限的DMA缓冲区
特权DMA缓冲区

图 24

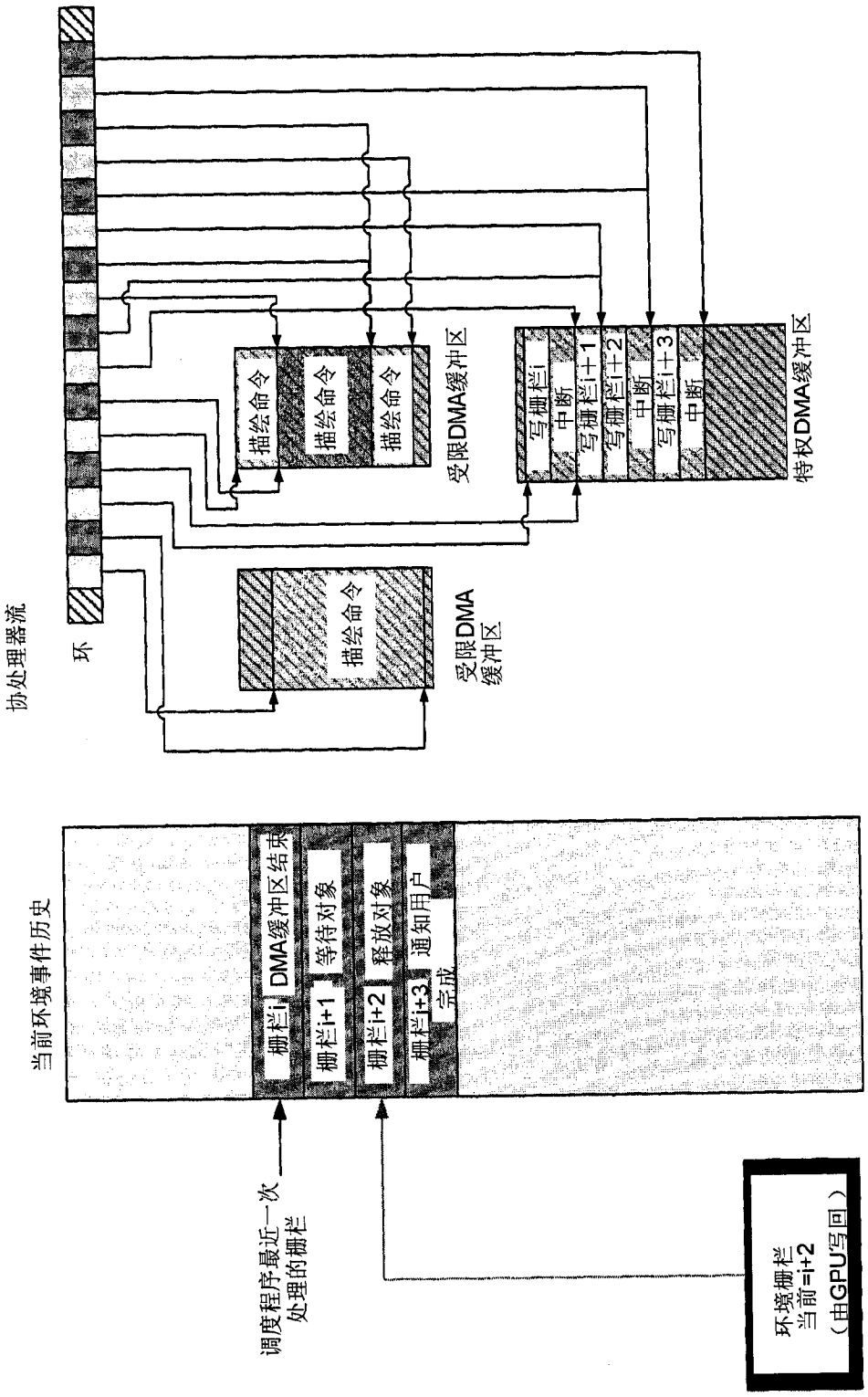

协处理器流
环
描绘命令
描绘命令
描绘命令
受限DMA缓冲区
写栅栏i
中断
写栅栏i+1
写栅栏i+2
中断
写栅栏i+3
中断
特权DMA缓冲区
描绘命令
受限DMA
缓冲区
当前环境事件历史
栅栏i DMA缓冲区结束
栅栏i+1 等待对象
栅栏i+2 释放对象
栅栏i+3 通知用户
完成
调度程序最近一次
处理的栅栏
环境栅栏
当前=i+2
（由GPU写回）

图 25

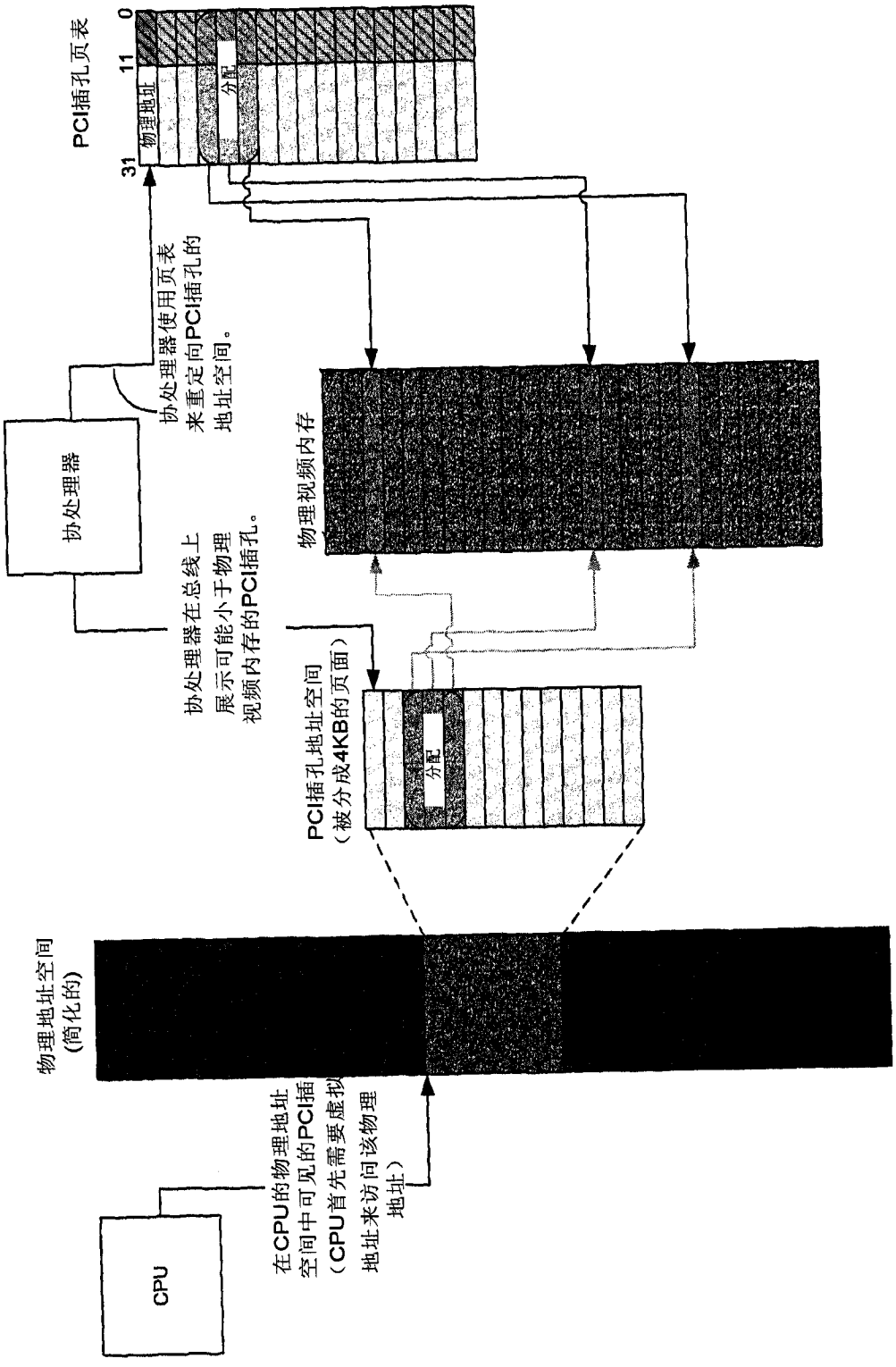

PCI插孔页表
31
11
0
物理地址
分配
协处理器使用页表
来重定向PCI插孔的
地址空间。
协处理器
协处理器在总线上
展示可能小于物理
视频内存的PCI插孔。
物理视频内存
PCI插孔地址空间
（被分成4KB的页面）
分配
物理地址空间
(简化的)
在CPU的物理地址
空间中可见的PCI插
（CPU首先需要虚拟
地址来访问该物理
地址）
CPU

图 26