(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 102648450 A
(43)申请公布日 2012.08.22

(21)申请号 201080053155.1
(22)申请日 2010.09.23
(30)优先权数据
61/245,174 2009.09.23 US
12/853,161 2010.08.09 US
12/868,596 2010.08.25 US
(85)PCT申请进入国家阶段日
2012.05.23
(86)PCT申请的申请数据
PCT/US2010/049961 2010.09.23
(87)PCT申请的公布数据
WO2011/038092 EN 2011.03.31
(71)申请人 辉达公司
地址 美国加利福尼亚州
(72)发明人 杰罗姆·F·小杜鲁克
杰西·大卫·豪
亨利·帕尔德·莫顿
帕特里克·R·布朗
(74)专利代理机构 北京市磐华律师事务所 11336
代理人 董巍 魏宁
(51)Int.Cl.
G06F 9/30(2006.01)

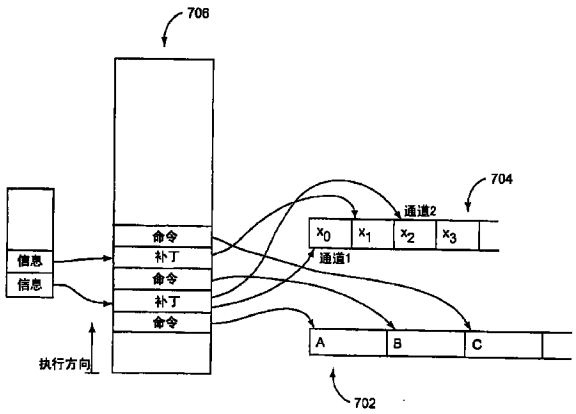
权利要求书 2 页 说明书 15 页 附图 10 页

(54)发明名称

用于并行命令列表生成的硬件

(57)摘要

本发明提供一种用于在多线程处理环境中提供跨命令列表的状态继承方法。所述方法包括：接收包括多个并行线程的应用程序；为多个并行线程的每个线程生成命令列表；使与多个并行线程的第一线程相关联的第一命令列表由处理单元执行；以及使与多个并行线程的第二线程相关联的第二命令列表由处理单元执行，其中第二命令列表从第一命令列表继承与处理单元相关联的状态。



CN 102648450 A

1. 一种用于为多线程处理环境提供初始默认状态的方法，所述方法包括：

接收包括多个并行线程的应用程序；

为所述多个并行线程的每个线程生成命令列表；

将释放方法插入到与所述多个并行线程的第一线程相关联的第一命令列表中以由处理单元执行，其中所述释放方法是将由所述处理单元所执行的命令；

使所述释放方法由所述处理单元执行，使得处理单元的状态的每个参数被重置；以及，

在执行所述释放方法之后，使包括在所述第一命令列表中的命令由所述处理单元执行。

2. 根据权利要求 1 所述的方法，其中所述处理单元包括图形处理单元。

3. 根据权利要求 1 所述的方法，其中所述处理单元包括中央处理单元的内核。

4. 根据权利要求 1 所述的方法，其中所述释放方法按顺序是在将由所述处理单元所执行的第一命令列表中的第一个命令。

5. 根据权利要求 1 所述的方法，其中存储器单元的一个或多个行被配置为存储所述处理单元的状态，并且进一步包括，将与所述一个或多个行的每一者相关联的一个或多个位设置为无效状态。

6. 根据权利要求 1 所述的方法，其中重置所述处理单元的所述状态包括，清空与所述处理单元的所述状态相关联的高速缓存存储器中的一个或多个高速缓存线。

7. 根据权利要求 1 所述的方法，其中重置所述处理单元的所述状态包括，清空与所述处理单元的所述状态相关联的存储体存储器的一个或多个存储体。

8. 一种用于在多线程处理环境中提供跨命令列表的状态继承的方法，所述方法包括：

接收包括多个并行线程的应用程序；

为所述多个并行线程的每个线程生成命令列表；

使与所述多个并行线程的第一线程相关联的第一命令列表由处理单元执行；以及，

使与所述多个并行线程的第二线程相关联的第二命令列表由所述处理单元执行，其中所述第二命令列表从所述第一命令列表继承与所述处理单元相关联的状态。

9. 根据权利要求 8 所述的方法，其中所述第一命令列表包括在命令缓冲区中的方法和补丁方法缓冲区中的方法之间交替变化的令牌，所述命令缓冲区中的方法于命令列表构建期间被写入，所述补丁方法缓冲区中的方法于提交命令列表用于执行期间被写入。

10. 根据权利要求 9 所述的方法，其中每个令牌包括指向存储器单元的指针，所述存储器单元被配置为存储由所述处理单元所要执行的命令。

11. 根据权利要求 9 所述的方法，其中索引大小是与所述处理单元相关联的状态的参数，并且将基于所述索引大小的绘图命令包括在所述命令缓冲区中，而无需相应的补丁方法。

12. 根据权利要求 9 所述的方法，其中基元拓扑是与所述处理单元相关联的状态的参数，并且将基于所述基元拓扑的绘图命令包括在所述命令缓冲区中，而无需相应的补丁方法。

13. 根据权利要求 9 所述的方法，其中在几何处理管线中的每个着色阶段都具有相关联的一组裁剪平面使能位，并且所述处理单元被配置为使用与最后一个被使能的着色阶段相关联的所述裁剪平面使能位。

14. 根据权利要求 8 所述的方法，其中所述处理单元包括图形处理单元。

15. 一种计算机系统，包括：

处理器，以及

存储指令的存储器，当所述指令由所述处理器执行时，使得所述处理器通过执行以下步骤，在多线程处理环境中提供跨命令列表的状态继承：

接收包括多个并行线程的应用程序；

为所述多个并行线程的每个线程生成命令列表；

使与所述多个并行线程的第一线程相关联的第一命令列表由处理单元执行；以及，

使与所述多个并行线程的第二线程相关联的第二命令列表由所述处理单元执行，其中所述第二命令列表从所述第一个命令列表继承与所述处理单元相关联的状态。

用于并行命令列表生成的硬件

[0001] 相关申请的交叉引用

[0002] 本申请要求于 2009 年 9 月 23 日提交的序列号为 61/245,174 的美国临时专利申请、2010 年 8 月 9 日提交的序列号为 12/853,161 的美国专利申请以及 2010 年 8 月 25 日提交的序列号为 12/853,161 的美国专利申请的优先权。

技术领域

[0003] 本发明涉及处理单元，并且，具体地说，涉及用于并行命令列表生成的硬件。

背景技术

[0004] Microsoft® Direct3D 11(DX11) 是支持曲面细分 (tessellation) 以及允许经改进多线程的 API(应用程序接口)，用于帮助开发者开发更好地利用多核处理器的应用程序。

[0005] 在 DX11 中，CPU(中央处理单元) 的每个内核均能并行地执行命令线程。每个内核或在相同内核上的不同线程均通过自己的用户模式驱动副本来生成独立的命令列表以提高软件应用程序的性能。命令列表是命令缓冲区 (buffer) 的 API 级的抽象，它是较低级别的概念。当驱动从应用程序接收到 API 命令时，该驱动建立命令缓存 ；命令列表由完整的命令缓冲区加上任意附加的定义实施的 (implementation-defined) 元 (meta) 信息来表现。命令列表或命令缓冲区的内容典型地由 GPU(图形处理单元) 执行。在这些 CPU 内核的其中之一上运行有单个线程，该线程以特定顺序提交用于执行的命令列表。这些命令列表的顺序，以及由此命令缓冲区的顺序，由所述应用程序确定。通过入栈缓冲区 (pushbuffer) 将命令缓冲区输入到内核中。所述命令缓冲区包括由内核执行的方法，特别是由 GPU 执行的方法。

[0006] 然而，DX11 不允许跨命令列表的处理器状态继承。相反，处理器状态在每个命令列表开始时被重置为所谓的“空白状态 (clean slate state)”。也就是指，每个用户模式驱动线程都在命令列表开始时在处理器中设置所有的状态参数。由于在执行应用程序时线程不能配合，所以不提供跨命令列表的状态继承具有极大的缺点。此外，使用几十或几百个命令来将处理器的状态重置为空白状态所增加的处理开销降低了系统的效率，因此，降低了整体性能。

[0007] 如上所述，本技术领域需要一种改进的技术解决上述当前方法的局限性。

发明内容

[0008] 本发明的一个实施例提供了一种用于在多线程处理环境中提供跨命令列表的状态继承方法。所述方法包括 ：接收包括多个并行线程的应用程序 ；为所述多个并行线程的每个线程生成命令列表 ；使与所述多个并行线程的第一线程相关联的第一命令列表由处理单元执行 ；以及使与所述多个并行线程的第二线程相关联的第二命令列表由所述处理单元执行，其中所述第二命令列表从所述第一命令列表继承与所述处理单元相关联的状态。

[0009] 本发明的另一个实施例提供一种用于为多线程处理环境提供初始默认状态的方法。所述方法包括：接收包括多个并行线程的应用程序；为所述多个并行线程的每个线程生成命令列表；将释放方法插入到与所述多个并行线程的第一线程相关联的第一命令列表中以由处理单元执行，其中所述释放方法是将由所述处理单元所执行的命令；使所述释放方法由所述处理单元执行，使得处理单元的状态的每个参数被重置；以及，在执行所述释放方法后，使被包括在所述第一命令列表中的命令由所述处理单元执行。

[0010] 本发明实施例所提供的一个优点在于，相对现有技术实现更高的处理效率。

附图说明

[0011] 因此，可以详细地理解本发明的上述特征，并且可以参考实施例对于上面所简要说明的本发明进行更具体的描述，其中一些实施例在附图中示出。然而，应当注意的是，附图仅示出了本发明的典型实施例，因此不应被认为是对本发明范围的限制。本发明可以适用于其他等效的实施例。

[0012] 图 1 为示出了被配置为实现本发明一个或多个方面的计算机系统的框图；

[0013] 图 2 为根据本发明的一个实施例的，用于图 1 的计算机系统的并行处理子系统的框图；

[0014] 图 3A 为根据本发明一个实施例的，图 2 的一个 PPU 内的 GPC 的框图；

[0015] 图 3B 为根据本发明一个实施例的，图 2 的一个 PPU 内的分区单元的框图；

[0016] 图 4 为根据本发明一个实施例的，图 2 的一个或多个 PPU 可经配置以实现的图形处理管线的示意图；

[0017] 图 5 为根据本发明一个实施例的，示出了使用并行命令列表的多线程处理的示意图。

[0018] 图 6 为根据本发明一个实施例的，示出了跨命令列表的状态继承的示意图。

[0019] 图 7 为根据本发明一个实施例的，示出了用于状态继承的命令列表的示意图。

[0020] 图 8 为根据本发明一个实施例的，用于具有跨命令列表状态继承的多线程处理的方法步骤的流程图；

[0021] 图 9 为根据本发明一个实施例的，用于生成命令列表的方法步骤的流程图；

[0022] 图 10 为根据本发明一个实施例的，用于使用 UnbindAll( ) 方法实现多线程处理的方法步骤的流程图。

具体实施方式

[0023] 在下面的描述中，将阐述大量的具体细节以提供对本发明更为彻底的理解。然而，对于本技术领域技术人员来讲将显而易见的是，本发明可以在缺少这些具体细节的一个或者多个的情况下得以实施。在其他实例中，没有描述公知的特征以避免对本发明造成混淆。

[0024] 系统概述

[0025] 图 1 为示出了被配置为实施本发明的一个或多个方面的计算机系统 100 的框图。计算机系统 100 包括中央处理单元 (CPU)102 和系统存储器 104，两者通过存储器桥 105 经由总线路径通信。DX 11 多线程特征主要针对多内核 CPU。因此，在一些实施例中，CPU 102 是多内核 CPU。如图 1 所示，存储器桥 105 可整合在 CPU 102 中。可选地，存储器桥 105 可

以是常见设备，例如北桥芯片，其通过总线连接到 CPU 102。存储器桥 105 可经由通信路径 106（例如超传输 (HyperTransport) 链路）连接到 I/O（输入 / 输出）桥 107。I/O 桥 107，其可以例如是南桥芯片，从一个或多个用户输入设备 108（诸如键盘、鼠标）接收用户输入并且经由路径 106 和存储器桥 105 将所述输入转发 (forward) 到 CPU 102。并行处理子系统 112 经由总线或其他通信路径 113（例如 PCI Express、加速图形端口或超传输链路）耦合到存储器桥 105；在一个实施例中，并行处理子系统 112 是将像素传输到显示设备 110（例如常见的基于 CRT 或 LCD 的显示器）的图形子系统。系统盘 114 也连接到 I/O 桥 107。开关 116 为 I/O 桥与诸如网络适配器 118 以及各种外插卡 (add-in card)120 和 121 的其他组件之间提供了连接。其他组件（未明确示出）也可以连接到 I/O 桥 107，包括 USB 或其他端口连接、CD 驱动器、DVD 驱动器、胶片记录设备等。图 1 中将各种组件互连的通信路径可以采用任何适合的协议来实现，诸如 PCI（外部组件互连）、PCI-Express、AGP（加速图形端口）、超传输或任何其他总线或者点对点通信协议。并且不同设备间的连接可采用本技术领域已知的不同协议。

[0026] 在一个实施例中，并行处理子系统 112 包含被优化用于图形和视频处理的电路，例如包括视频输出电路，并且构成图形处理单元 (GPU)。在另一个实施例中，并行处理子系统 112 包含被优化用于通用目的处理的电路，同时保留底层 (underlying) 的计算架构，本文将进行更为详细的描述。在另一个实施例中，可以将并行处理子系统 112 与一个或多个其他系统元件集成，诸如存储器桥 105、CPU 102、以及 I/O 桥 107，以形成片上系统 (SoC)。

[0027] 应该理解，本文所示系统是示例性的，可以对其进行变形和修改。可根据需要修改连接拓扑，包括桥的数量和布置。例如，在一些实施例中，系统存储器 104 直接连接到 CPU 102 而不是通过桥，其他设备经由存储器桥 105 以及 CPU 102 与系统存储器 104 通信。在其他替代拓扑中，并行处理子系统 112 连接到 I/O 桥 107 或直接连接到 CPU 102，而不是连接到存储器桥 105。在又一个实施例中，CPU 102 的一个或多个、I/O 桥 107、并行处理子系统 112 和存储器桥 105 可被集成到一个或多个芯片中。本文所示的特定组件是可选的；例如，可以支持任意数量的外插卡或外围装置。在一些实施例中，开关 116 被去掉，网络适配器 118 和外插卡 120、121 直接连接到 I/O 桥 107。

[0028] 图 2 示出根据本发明的一个实施例的并行处理子系统 112。如图所示，并行处理子系统 112 包括一个或多个并行处理单元 (PPU)202，每个并行处理单元 PPU 都耦合到本地并行处理 (PP) 存储器 204。通常，并行处理子系统包括 U 个 PPU，其中 U ≥ 1。（本文中，相似对象的多个实体用标识该对象的参考数字和根据需要结合标识该实体的带括号的数字加以表示。）PPU 202 和并行处理存储器 204 可使用一个或多个集成电路器件来实现，该集成电路器件诸如可编程处理器、专用集成电路 (ASIC) 或存储器器件，或以任何其他技术上可行的方式来实现。

[0029] 再参考图 1，在一些实施例中，并行处理子系统 112 的一些或所有 PPU202 是具有渲染管线的图形处理器，它可以被配置为执行与下述各项相关的各种任务：从 CPU102 和 / 或系统存储器 104 所提供的图形数据生成像素数据、与本地并行处理存储器 204（可被用作图形存储器，包括例如常用的帧缓冲器）交互以存储和更新像素数据、输送像素数据到显示设备 110 等等。在一些实施例中，并行处理子系统 112 可包括一个或多个作为图形处理器而操作的 PPU 202，以及一个或多个用作通用计算的其他 PPU 202。这些 PPU 可以是相同的

或不同的，并且每个 PPU 均可有自己的专用并行处理存储器设备或非专用并行处理存储器设备。一个或多个 PPU 202 可输出数据到显示设备 110，或每个 PPU 202 均可输出数据到一个或多个显示设备 110。

[0030] 在操作中，CPU 102 是计算机系统 100 的主处理器，控制和协调其他系统组件的操作。具体地，CPU 102 发出控制 PPU 202 的操作的命令。在一些实施例中，CPU 102 将针对每个 PPU 202 的命令流写入到命令缓冲区中（在图 1 或图 2 中未明确示出），所述命令缓冲区可位于系统存储器 104、并行处理存储器 204、或其他可由 CPU 102 和 PPU 202 访问的存储位置中。PPU 202 从命令缓冲区读取命令流，然后相对于 CPU 102 的操作异步地执行命令。CPU 102 也可创建可由 PPU 202 响应命令缓冲区中的命令而读取的数据缓冲区。每个命令和数据缓冲区均可由每个 PPU 202 所读取。

[0031] 现在返回参考图 2，每个 PPU 202 均包括 I/O（输入 / 输出）单元 205，该 I/O 单元 205 经由连接到存储器桥 105（或，在一个替代实施例中，直接连接到 CPU 102）的通信路径 113 与计算机系统 100 的其余部分通信。PPU 202 与计算机系统 100 的其余部分的连接也可以不同。在一些实施例中，并行处理子系统 112 可作为外插卡来实现，所述外插卡可被插入到计算机系统 100 的扩展插槽中。在其他实施例中，PPU 202 可以和诸如存储器桥 105 或 I/O 桥 107 的总线桥一起集成在单个芯片上。在又一些实施例中，PPU 202 的一些或所有元件可以和 CPU 102 一起集成在单个芯片上。

[0032] 在一个实施例中，通信路径 113 是 PCI-Express 链路，其中给每个 PPU202 分配专用的通道（lane），如本技术领域所知。也可使用其他的通信路径。I/O 单元 205 生成数据包（或其他信号）用于在通信路径 113 上传送，并且还从通信路径 113 接收所有传入的数据包（或其他信号），将传入的数据包引向 PPU 202 的适当组件。例如，可将与处理任务有关的命令引向主机接口 206，而可将与存储器操作有关的命令（例如，对并行处理存储器 204 的读取或写入）引向存储器交叉开关（crossbar）单元 210。主机接口 206 读取每个命令缓冲区，并且将该命令缓冲区指定的工作输出到前端 212。

[0033] 有利地，每个 PPU 202 都实现高度并行处理架构。如图中详细所示，PPU 202(0) 包括处理集群阵列 230，该阵列 230 包括 C 个通用处理集群 (GPC)208，其中 C ≥ 1。每个 GPC 208 均能并发执行大量的（例如，几百或几千）线程，其中每个线程均是程序的实例 (instance)。在各种应用中，可以分配不同的 GPC208 用于处理不同类型的程序或用于执行不同类型的计算。例如，在图形应用中，可分配第一组 GPC 208 来执行曲面细分操作并为曲面片 (patch) 生成基元 (primitive) 拓扑，并且可分配第二组 GPC 208 来执行曲面细分着色以评估用于基元拓扑的曲面片参数以及确定顶点位置和每顶点的其他属性。GPC 208 的分配可根据为每种类型的程序或计算所产生的工作量而不同。可替代地，GPC208 可被分配以使用分时 (time-slice) 机制来执行处理任务，从而在不同的处理任务间切换。

[0034] GPC 208 经由工作分布单元 200 接收将被执行的处理任务，所述工作分布单元 200 从前端单元 212 接收定义处理任务的命令。处理任务包括指向将被处理的数据的指针以及定义将如何处理数据（例如，将执行哪个程序）的命令和状态参数，所述指向将被处理的数据的指针例如表面（曲面片）数据、基元数据、顶点数据和 / 或像素数据。工作分布单元 200 可以被配置为获取与这些处理任务相对应的指针、可以从前端 212 接收指针或者可以直接从前端接收数据。在一些实施例中，索引指定了数据在阵列中的位置。前端 212 确保在命

令缓冲区所指定的处理启动前，GPC 208 被配置为有效状态。

[0035] 当 PPU 202 被用于图形处理时，例如，将每个曲面片的处理工作量分成大小近似相等的任务，以使曲面细分处理能够被分布到多个 GPC 208。工作分布单元 200 可被配置为以能够提供任务给多个 GPC 208 进行处理的频率输出任务。在本发明的一些实施例中，部分 GPC 208 可被配置为执行不同类型的处理。例如，第一部分可被配置为执行顶点着色和拓扑生成，第二部分可被配置为执行曲面细分和几何着色，并且第三部分可被配置为在屏幕空间中执行像素着色以产生渲染后的图像。分配部分 GPC 208 用于执行不同类型处理任务的能力有效地适应了由这些不同类型处理任务所产生的数据的扩展和伸缩。在下游 GPC 208 接收数据的速率滞后于上游 GPC208 产生数据的速率的情况下，由 GPC 208 产生的中间数据可以被缓冲，以允许该中间数据在 GPC 208 之间以最小的停顿 (stalling) 来传送。

[0036] 可将存储器接口 214 划分为 D 个存储器分区单元，其每一个单元均耦合到并行处理存储器 204 的一部分，其中 D ≥ 1。通常，并行处理存储器 204 的每一部分均包括一个或多个存储器设备（例如，DRAM 220）。本技术领域的技术人员应该理解，DRAM 220 可以由其他合适的存储设备代替，并且可以是一般常规设计。因此省略详细的描述。渲染目标，诸如帧缓冲区或纹理映射，可以被跨 DRAM 220 存储，允许分区单元 215 并行地写入每个渲染目标的一部分，从而有效地使用并行处理存储器 204 的可用带宽。

[0037] 任意一个 GPC 208 都可以处理将被写到并行处理存储器 204 内任意 DRAM 220 的数据。交叉开关单元 210 被配置为将每个 GPC 208 的输出路由到任意分区单元 215 的输入或路由到另一个 GPC 208 用于进一步处理。GPC 208 通过交叉开关单元 210 与存储器接口 214 通信，以对各种外部存储器设备进行读写。在一个实施例中，交叉开关单元 210 具有到存储器接口 214 的连接以和 I/O 单元 205 通信，以及具有到本地并行处理存储器 204 的连接，从而使得在不同 GPC 208 内的处理内核能够与系统存储器 104 或相对于 PPU 202 来讲非本地的其他存储器通信。交叉开关单元 210 可使用虚拟通道来分开 GPC 208 与分区单元 215 之间的业务流。

[0038] 再者，GPC 208 可被编程为执行与种类繁多的应用相关的处理任务，包括但不限于，线性和非线性数据转换、视频和 / 或音频数据的过滤、建模操作（例如，应用物理定律以确定对象的位置、速率和其他属性）、图像渲染操作（例如，曲面细分着色、顶点着色、几何着色和 / 或像素着色程序）等等。PPU 202 可将数据从系统存储器 104 和 / 或本地并行处理存储器 204 传输到内部（片上）存储器中，处理数据，并且将结果数据写回到系统存储器 104 和 / 或本地并行处理存储器 204，在这里这样的数据可以由其他系统组件访问，所述其他系统组件包括 CPU 102 或另一并行处理子系统 112。

[0039] PPU202 可拥有任意容量 (amount) 的本地并行处理存储器 204，包括没有本地存储器，并且 PPU202 可以以任意组合方式使用本地存储器和系统存储器。例如，在统一存储架构 (UMA) 实施例中，PPU 202 可以是图形处理器。在这样的实施例中，将不提供或几乎不提供专用的图形（并行处理）存储器，并且 PPU 202 会以排他的方式或几乎以排他的方式使用系统存储器。在 UMA 实施例中，PPU 202 可被集成到桥式芯片中或处理器芯片中，或作为分立芯片被提供，所述分立芯片具有经由桥式芯片或其他通信方式将 PPU 202 连接到系统存储器的高速链路（例如，PCI-Express）。

[0040] 如上所述，并行处理子系统 112 可以包括任意数量的 PPU 202。例如，在单个外插

卡上可以提供多个 PPU 202，或可以将多个外插卡连接到通信路径 113，或可以将一个或多个 PPU 202 集成到桥式芯片中。多 PPU 系统中的 PPU 202 可以彼此相同或不同。例如，不同的 PPU 202 可能具有不同数量的处理内核、不同容量的本地并行处理存储器等等。在出现多个 PPU202 的情况下，可并行地操作这些 PPU 从而以高于单个 PPU 202 可能达到的吞吐量来处理数据。包括一个或多个 PPU 202 的系统可以以各种配置和形式因素来加以实现，包括桌上型电脑、膝上型电脑或手持式个人计算机、服务器、工作站、游戏控制台、嵌入式系统等。

[0041] 处理集群阵列概述

[0042] 图 3A 为根据本发明一个实施例的，图 2 的一个 PPU 202 内的 GPC 208 的框图。每个 GPC 208 可被配置为以并行方式执行大量的线程，其中术语“线程”是指针对特定的一组输入数据执行的特定程序的实例。在一些实施例中，使用单指令多数据 (SIMD) 指令发送技术来支持大量线程的并行执行，而无需提供多个独立的指令单元。在其他实施例中，采用单指令多线程 (SIMT) 技术，使用被配置为发送指令到每个 GPC 208 内一组处理引擎的公共指令单元，来支持大量通常同步化的线程的并行执行。与其中所有处理引擎一般都执行相同指令的 SIMD 执行机制不同，SIMT 的执行通过给定的线程程序，允许不同线程更容易地跟踪 (follow) 分散的执行路径。本技术领域的技术人员应该理解，SIMD 处理机制代表 SIMT 处理机制的一个功能子集。

[0043] 在图形应用中，GPC 208 可被配置以实现用于执行屏幕空间图形处理功能的基元引擎，所述屏幕空间图形处理功能可包括但不限于，基元建立、光栅化以及 z 裁剪 (z culling)。该基元引擎从工作分布单元 200 接收处理任务，并且当该处理任务不需要基元引擎执行操作时，将该处理任务通过基元引擎传到管线管理器 305。经由管线管理器 305 来方便地控制 GPC 208 的操作，所述管线管理器 305 分布处理任务给流多处理器 (SPM)310。管线管理器 305 也可被配置为通过为由 SPM 310 输出的处理后的数据指定目的地来控制工作分布交叉开关 330。

[0044] 在一个实施例中，每个 GPC 208 均包括 M 个 SPM 310，其中 M ≥ 1，每个 SPM 310 均被配置为处理一个或多个线程组。传送到特定 GPC 208 的一系列指令构成线程，如本文前面所定义的，并且跨 SPM310 内并行处理引擎（未示出）所并发执行的一定数量的线程的集合在本文中被称作“线程组”。如本文所使用的，“线程组”是指针对不同输入数据并发执行相同程序的一组线程，该组的每个线程被分配给 SPM 310 内的不同处理引擎。线程组可以包括比 SPM 310 内处理引擎的数量更少的线程，在这种情况下，在正在处理该线程组的周期期间，一些处理引擎将处于空闲状态。线程组也可以包括比 SPM 310 内处理引擎的数量更多的线程，在这种情况下，将在多个时钟周期进行处理。因为每个 SPM 310 能并发地支持多达 G 个线程组，所以在任意给定的时间，GPC 208 中都可以执行多达 G×M 个线程组。

[0045] 此外，在 SPM 310 内，多个相关的线程组可同时处于激活状态（处于不同的执行阶段）。这个线程组的集合在本文中被称作“协作线程阵列”（“CTA”）。特定 CTA 的大小等于 m×k，其中 k 是线程组中并发执行的线程的数量，并且一般是 SPM 310 内并行处理引擎数量的整数倍，m 是在 SPM 310 内同时处于激活状态的线程组的数量。CTA 的大小通常由编辑人员以及 CTA 可用的硬件资源例如存储器或寄存器的容量来确定。

[0046] 每个线程可使用独享的本地地址空间，并且共享的每 CTA(per-CTA) 地址空间用

于在 CTA 内的线程之间传递数据。将存储在每线程地址空间和每 CTA 地址空间中的数据存储在 L1 高速缓存 (cache)320 中，并且可使用回收 (eviction) 策略以有助于在 L1 高速缓存 320 中保持该数据。每个 SPM 310 使用相应的 L1 高速缓存 320 中用于执行加载和存储操作的空间。每个 SPM 310 也可以访问分区单元 215 内的 L2 高速缓存，该 L2 高速缓存在所有 GPC 208 之间被共享并且可以用于在线程之间传输数据。最后，SPM310 也可以访问片外“全局”存储器，包括例如并行处理存储器 204 和 / 或系统存储器 104。L2 高速缓存可用于存储写入全局存储器或从全局存储器读出的数据。应该理解，PPU 202 外部的任何存储器都可以被用作全局存储器。

[0047] 另外，如本技术领域所知，每个 SPM 310 有利地包括可被管线化的相同的一组功能单元 (例如，算术逻辑单元)，从而允许在前一个指令完成之前发送新的指令。可以提供功能单元的任意组合。在一个实施例中，这些功能单元支持各种操作，包括整数和浮点算法 (例如，加法和乘法)、比较操作、布尔操作 (AND、OR、XOR)、移位 (bit-shifting) 以及各种代数函数计算 (例如，平面插值、三角、指数和对数函数等)；并且相同的功能单元硬件可均衡地用于 (be leveraged to) 执行不同操作。

[0048] 每个 GPC 208 可包括被配置为将虚拟地址映射到物理地址的存储器管理单元 (MMU)328。在其他实施例中，MMU328 可驻留在存储器接口 214 内。MMU 328 包括用于将虚拟地址映射到像素块 (tile) 的物理地址的一组页表条目 (PTE)，以及可选地包括高速缓存线索引 (cache line index)。处理所述物理地址以分散表面数据访问位置，从而允许在分区单元交错的高效请求。所述高速缓存线索引可被用于确定对高速缓存线的请求是否命中或者未命中。

[0049] 在图形应用中，GPC 208 可被配置为使得每个 SPM 310 耦合到纹理单元 315，用于执行纹理映射操作，例如确定纹理采样位置、读取纹理数据以及过滤纹理数据。纹理数据经由存储器接口 214 读取，以及当需要的时候从 L2 高速缓存、并行处理存储器 204 或系统存储器 104 获取。纹理单元 315 可被配置为在内部高速缓存中存储纹理数据。在一些实施例中，将纹理单元 315 耦合至 L1 高速缓存 320，并且将纹理数据存储在 L1 高速缓存 320 中。为了提供处理后的任务给另一个 GPC 208 用于进一步的处理，或者为了经由交叉开关单元 210 在 L2 高速缓存、并行处理存储器 204 或系统存储器 104 中存储该处理后的任务，每个 SPM 310 都将处理后的任务输出给工作分布交叉开关 330。preROP(pre-raster operations，预光栅操作) 单元被配置为从 SPM 310 接收数据，将数据引向分区单元 215 中的 ROP 单元并执行对色彩混合的优化、组织像素色彩数据以及执行地址转换。

[0050] 应该理解，本文所描述的内核架构是示例性的，可以对其进行各种变形和修改。GPC 208 内可以包括任意数量的处理引擎例如基元引擎、SPM310、纹理单元 315、或 preROP 325。进一步，虽然只示出了一个 GPC 208，但 PPU 202 可以包括任意数量的 GPC 208，这些 GPC 208 最好功能上彼此相似，从而执行行为不依赖于哪个 GPC 208 接收到特定处理任务。进一步地，有利地，每个 GPC 208 最好使用单独的且各异的处理引擎、L1 高速缓存 320 等，独立于其他 GPC 208 操作。

[0051] 图 3B 为根据本发明一个实施例的，图 2 的一个 PPU 202 内分区单元 215 的框图。如图所示，分区单元 215 包括 L2 高速缓存 350、帧缓冲区 (FB)355 以及光栅操作单元 (ROP)360。L2 高速缓存 350 是读 / 写高速缓存，其被配置为对从交叉开关单元 210 及 ROP

360 接收的操作执行加载和存储。读缺失和紧急写回请求由 L2 高速缓存 350 输出到 FB 355 用于处理。脏更新也被发送到 FB 355 用于伺机处理。FB 355 直接与 DRAM 220 交互，输出读和写请求，并接收从 DRAM 220 读取的数据。

[0052] 在图形应用中，ROP 360 是执行诸如模板、z 测试、混合等等光栅操作的处理单元，并输出像素数据作为处理后的图形数据用于在图形存储器中存储。在本发明的一些实施例中，将 ROP 360 包括在每个 GPC 208 中，而不是分区单元 215 中，并且通过交叉开关单元 210 传送像素读写请求而不是像素片段数据。

[0053] 处理后的图形数据可以在显示设备 110 上显示，或者被路由用于由 CPU 102 或由并行处理子系统 112 内的处理实体之一来进一步处理。为了分布光栅操作的处理，每个分区单元 215 均包括 ROP 360。在一些实施例中，ROP 360 可被配置为压缩写入存储器的 z 数据或色彩数据，以及解压缩从存储器中读出的 z 数据或色彩数据。

[0054] 本技术领域的技术人员应该理解，图 1、2、3A 和 3B 中描述的架构不以任何方式限制本发明的范围，以及在此教导的技术可以在任意合适配置的处理单元上实现，包括但不限于一个或多个 CPU、一个或多个多内核 CPU、一个或多个 PPU 202、一个或多个 GPC 208、一个或多个图形或特殊用途的处理单元等等，均不超出本发明的范围。

[0055] 图 4 为根据本发明一个实施例的，由图 2 中一个或多个 PPU 202 经配置以实现的图形处理管线 400 的示意图。例如，一个 SPM 310 可被配置为执行一个或多个顶点处理单元 415、几何 (geometry) 处理单元 425 以及片段处理单元 460 的功能。数据汇编器 410、基元汇编器 420、光栅化器 455 以及光栅操作单元 465 的功能也可以由 GPC 208 和相应的分区单元 215 中的其他处理引擎执行。可替代地，可采用用于一个或多个功能的专用处理单元来实现图形处理管线 400。

[0056] 数据汇编器 410 处理单元针对高序位的表面、基元等采集顶点数据，并输出包括顶点属性的顶点数据到顶点处理单元 415。顶点处理单元 415 是被配置为执行顶点着色程序的可编程执行单元，根据顶点着色程序的指定，光照 (lighting) 和转换顶点数据。例如，顶点处理单元 415 可被编程为将顶点数据从基于对象的坐标表示（对象空间）转换到诸如世界空间或标准化设备坐标 (NDC) 空间的替代基础坐标系统。顶点处理单元 415 可读取由数据汇编器 410 存储在 L1 高速缓存 320、并行处理存储器 204、系统存储器 104 中的数据，用于处理顶点数据。

[0057] 基元汇编器从顶点处理单元 415 接收顶点属性，按需读取存储的顶点属性，并且构建用于通过几何处理单元 425 处理的图形基元。图形基元包括三角形、线段、点等。几何处理单元 425 是被配置为执行几何着色程序的可编程执行单元，根据所述几何着色程序的指定，转换从基元汇编器 420 接收的图形基元。例如，几何处理单元 425 可经编程以细分图形基元为一个或多个新的图形基元，并计算被用于光栅化所述新的图形基元的参数，例如平面方程系数。

[0058] 在一些实施例中，几何处理单元 425 也可增加或删除几何流中的元素。几何处理单元 425 输出指定新的图形基元的参数和顶点到视窗 (viewport) 缩放、剔除 (cull) 及裁剪 (clip) 单元 450。几何处理单元 425 可读取存储在并行处理存储器 204 或系统存储器 104 中的数据，用于处理几何数据。视窗缩放、剔除及裁剪单元 450 执行裁剪、剔除以及视窗缩放，并输出处理后的图形基元到光栅化器 455。

[0059] 光栅化器 455 扫描转换新的图形基元，并输出片段和覆盖数据到片段处理单元 460。此外，光栅化器 455 可被配置为执行 z 剔除和其他基于 z 的优化。

[0060] 片段处理单元 460 是被配置为执行片段着色程序的可编程执行单元，根据片段着色程序的指定，转换从光栅化器 455 接收的片段。例如，片段处理单元 460 可经编程以执行诸如透视校正、纹理映射、着色、混合等操作，以产生输出到光栅操作单元 465 的着色片段。片段处理单元 460 可读取存储在并行处理存储器 204 或系统存储器 104 中的数据，用于处理片段数据。可以根据编程的采样率，来按像素、采样或其他粒度来对片段进行着色。

[0061] 光栅操作单元 465 是执行诸如模板 (stencil)、z 测试、混合等光栅操作的处理单元，并输出像素数据作为处理后的图形数据，用于存储在图形存储器中。处理后的图形数据可存储在例如并行处理存储器 204 和 / 或系统存储器 104 的图形存储器中，用于在显示设备 110 上显示或用于由 CPU 102 或并行处理子系统 112 进一步处理。在本发明的一些实施例中，光栅操作单元 465 被配置为压缩被写入存储器中的 z 数据或色彩数据，以及解压缩从存储器中读出的 z 数据或色彩数据。

[0062] 用于并行命令列表生成的硬件

[0063] 本发明的实施例涉及在多线程处理系统中将处理器状态从一个命令列表延递 (carry over) 到下一个命令列表。也就是说，诸如 GPU 或 CPU 的处理器的状态可跨多个命令列表累积。这一特征也被称作“跨命令列表的状态继承”。跨命令列表的状态继承给驱动器带来严重的问题，因为选择在命令列表中放入哪种方法依赖于在 GPU 中执行该方法时 GPU 的状态。GPU 状态其实是之前被执行的所有命令缓冲区中累积的状态表现。然而，GPU 状态可以在由不同驱动线程所生成的上一个命令缓存区中设置，但所述不同驱动线程并未完成对该上一个命令缓冲区的创建。本发明的实施例或者移除对未知继承关系的依赖，或者一旦继承状态已知则更新命令缓存区的状态依赖部分。

[0064] 在一个实施例中，将处理状态定义为与执行命令的处理单元相关联的参数集。在处理状态中所包括的参数的示例尤其包括 :顶点着色器、几何着色器、像素着色器等的选择，对绑定到像素着色器的一组不同纹理进行定义的一个或多个参数，对如何执行混合进行定义的参数，目标渲染表面的列表。这里所使用的“方法”是发送到处理硬件的命令，所述处理硬件设置一个或多个定义处理器状态的参数。在一个实施例中，设置处理器状态定义了不同的处理阶段如何执行随后的命令。

[0065] 本发明的实施例试图消除这样一种情况 :放入命令列表中的方法依赖于 GPU 所有命令列表的执行顺序中的当前状态。这一特征被称作“使驱动器无状态 (stateless)”，因为当生成命令列表时，驱动器不需要考虑当前 GPU 状态。

[0066] 本发明的实施例的另一个动机在于，减小 CPU 从应用程序向硬件提交渲染命令的开销，即，避免 CPU 成为瓶颈。这一开销的原因在于，CPU 花费时间检查当前状态以确定发送哪个方法。如果不需要检查当前状态来写入所述方法，需要的开销更少。

[0067] 在一个实施例中，每个处理设备有一个主线程和多个辅助线程 (worker thread)。例如，这种一主 /N 辅的布置可由应用程序确定。每个辅助线程都“拥有”一个与将被处理设备所执行的命令相关联的命令列表。在一些实施例中，处理设备包括 PPU 202 或 CPU 内核。辅助线程通过进行 API 调用（例如，状态变化、绘图命令等）将它们的命令列表并发地填充到驱动器中。当完成命令列表时，这些命令列表被传给主线程，该主线程对它们进行排

序并提交到驱动器 / 硬件。在一些实施例中，可多次提交命令列表。

[0068] 至少初看起来，实现这一模型要求驱动器是“无状态”的，意思是任意设备驱动器接口 (DDI) 入口点都可以被完全处理和转换为方法，而无需参考“当前”API 或处理器状态。在一个实施例中，每个 DDI 入口点可简单地被编码到命令令牌 (token) 和参数数据中，其可以被附加到与命令列表相关联的缓冲区。当调度命令列表以便执行时，这些令牌可被解释并转换为入栈缓冲区中的方法 / 命令。然而，这种方式在多线程命令列表生成中遇到瓶颈问题，这是由于达到这一结果所需的大多处理工作仍然在单线程中连续发生。

[0069] 在一个实施例中，每个命令列表包括令牌化的命令队列，以及一个或多个关联的 GPU 可读命令缓冲段。很多 DDI 入口点是无状态的，并且正好附加到命令缓冲区。命令令牌之一可能是“附加接下来的 N 个命令缓冲区字节”。而可能需要其他命令用于依赖状态 (state-dependent) 的处理。例如，当提交命令列表时，这一处理可发生在所述主线程，并且它的结果叠加到硬件可见的方法流中。

[0070] 在一个实施例中，每个命令列表都继承当前命令列表执行前所执行的命令列表留下的任意状态。这意味着当命令列表被生成时，可以不知道其初始状态，并且意味着每次命令列表被执行时这一状态甚至都可能不同，即，如果命令列表的排序改变。在这种情况下，驱动器在创建命令列表时，并不总是一直都知道当前 API 状态。

[0071] 在一个实施例中，在命令列表的资源参考 (reference) 与命令列表所使用的实际资源之间插入间接处理 (indirection)。当提交命令列表时，将该参考绑定到实际资源，并且可在提交之间改变。

[0072] 图 5 示出根据本发明一个实施例的，使用并行命令列表的多线程处理的示意图。如图所示，由应用程序开发人员所编写的软件应用程序可被分为多个线程 512-1、512-2、512-3。每个线程 512-1、512-2、512-3 分别与不同的驱动器 504-1、504-2、504-3 相关联。每个驱动器 504-1、504-2、504-3 分别与不同的命令列表 506-1、506-2、506-3 相关联。创建命令列表的线程执行于由应用程序和操作系统所确定的 CPU 内核上。一旦这些线程完成其命令列表的创建，则提交或调度这些命令列表用于由诸如 GPU 的处理单元 510 进行执行。在应用程序 502 经由信号 514 的控制下，经由软件多路复用器 508 执行命令列表提交步骤。

[0073] 图 6 示出根据本发明一个实施例的，跨命令列表的状态继承的示意图。如所描述的，实现跨命令列表的状态继承时，当在临近一个线程的执行结束时设置状态并且在临近另一个命令列表开始时执行依赖于该状态的命令时，可能会出现问题。依赖于所述状态的命令的示例是绘图命令。

[0074] 在图 6 所示的示例中，假设首先执行命令列表 506-1，接下来执行命令列表 506-2。如图 6 所示，在临近执行命令列表 506-1 的结束时由一个或多个命令 604 设置处理单元 510 的状态。当实现跨命令列表的状态继承时，处理单元 510 的状态被延递给命令列表 506-2 的执行，由路径 602 示出。绘图命令 606 可被包括在命令列表 506-2 中且临近命令列表 506-2 的开始处。因为分别与命令缓冲区 506-1、506-2 相关联的线程 512-1、512-2 可以是不相关的线程，当没有校正措施来保证执行所述画图命令 606 时正确地设置所述状态，则延递所述状态到线程 512-2 可能发生错误。如本文更详细的描述，本发明实施例支持有效地、无误地跨命令列表状态继承。

[0075] 图 7 示出根据本发明一个实施例的，用于状态继承的命令列表 706 的示意图。命

令列表 706 可包括一系列令牌。在一个实施例中，每个令牌都与指向命令缓冲区的指针相关联。

[0076] 在一些实施例中，所述命令列表 706 交替出现与应用程序命令相关联的令牌和与补丁方法相关联的令牌，如本文所描述。在图 7 所示的实施例中，命令列表 706 中的令牌包含指向缓冲区 702、704 的指针。每个缓冲区 702、704 存储将由诸如 PPU 202 或 CPU 内核这类执行单元所执行的命令，也称作“方法”。在一个实施例中，缓冲区 702 包括被包含在线程中将被执行的“常规”应用程序命令，并且缓冲区 704 包括被用于针对由处理单元所执行的后续命令将处理器状态设置为合适状态的“补丁”方法。

[0077] 在一些实施例中，驱动器被配置为设置包括于命令列表 706 中的指针，并且在缓冲区 702、704 中存储命令。驱动器顺序地遍历线程中的命令，并使用当前处理器状态来将所执行的命令存储到缓冲区 702 的块 A 中。一旦驱动器遇到依赖于不同处理器状态的命令，该驱动器停止在块 A 中存储命令。相反，驱动器在缓冲区 704 的块 x_0 中存储一个或多个补丁方法，其中存储在块 x_0 中的命令 / 方法被配置为将处理器状态修改为该线程中随后的命令所希望的形式。一旦将补丁方法存储在缓冲区 704 中，驱动器继续将包括在线程中的命令存储在缓冲区 702 中下一个可用块中，即块 B。重复这个过程，直到该线程中的所有命令都被存储在缓冲区 702 中，并且所需的补丁方法被存储在缓冲区 704 中。

[0078] 在执行时，处理单元遇到补丁命令块并生成补丁。然后将该补丁插入到命令队列中。当创建命令列表时，驱动器仅写入到缓冲区 702 和 706 中。在 706 中的“补丁”表项描述了随后的表项需要哪种状态信息。当提交用于执行的命令列表，尤其在主线程上提交时，补丁表项用于将补丁方法写入缓冲区 704。在命令队列中的插入是虚拟的 :命令队列仅是指向包括方法的缓冲区段的指针序列，因此它将指向段 {A，x0，B，x1，.....}。

[0079] 如图所示，命令列表 706 在指向将被执行的命令的指针与指向补丁方法的指针之间交替变化。存储在命令列表 706 中的指针或者指向缓冲区 702 中的线程命令块，或者指向缓冲区 704 中的补丁方法块。同样如图所示，缓冲区 704 和 / 或缓冲区 702 中的块在后续通过命令列表时被重复使用。例如，如图所示，在第一次通过时，命令列表 706 中的特定补丁方法指针可指向块 x_0，但在随后的通过中，同一个指针可指向块 x_2。使用图 7 所示的示例，将由处理单元所执行的块的序列可以是，例如 :

[0080] > A，x_0，B，x_1，C.....A，x_2，B.....

[0081] 在一些实施例中，当存在的补丁较少并且每个补丁尽可能小，即每个补丁的命令 / 方法较少时，将获得更高的效率。本发明的一些实施例包括一个或多个命令，如下所述，其经配置以更有效地执行上述的状态打补丁。相应地，本发明的实施例与下述内容相关联 :对处理单元的“状态”增加一个或多个附加参数，并且提供基于硬件的技术用于修改所述一个或多个附加参数。

[0082] 1. 索引缓冲区格式

[0083] 在一个实施例中，将索引缓冲区格式作为处理器状态的参数添加在硬件中。例如，当硬件绘制索引三角列表时，索引可以是 16 位或 32 位索引。在诸如 DX11 的传统方法中，较老的硬件要求将索引大小编码在绘图方法中，这是因为绘图命令依赖于所述索引大小。相应地，在 DX11 中，对于每个所遇到的绘图命令都需要补丁。

[0084] 相反，本发明的实施例包括作为处理器状态参数的索引缓冲区格式。因此，绘图命

令不需要与该索引命令一起包括索引大小。当执行绘图命令时，处理单元可简单地参考与索引缓冲区格式相关联的状态参数。为了修改与索引缓冲区格式相关联的状态参数，可实施具有 IndexSize 字段的单个 SetIndexBuffer( ) 方法。

[0085] 2. 基元拓扑

[0086] 在传统方法中，基元拓扑并未被作为处理器状态的一部分而包括在硬件中。因此，对于每个绘图命令来说，与该绘图命令相关联的基元拓扑（例如，三角形、三角形带、线等等）将需要被包括在该绘图命令中。根据本发明的实施例，基元拓扑被作为状态参数而添加，不需要将其作为绘图命令的一部分来包括。然而，当处理单元接收绘图命令时，该处理单元可能不知道基元拓扑参数的当前设置。因此，本发明的实施例实施单个方法 SetPrimitiveTopology( ) 以设置基元拓扑，而不需要驱动器将该基元拓扑作为绘图命令的一部分（或 Begin 方法的一部分）来加以包括。

[0087] 3. 用户裁剪平面使能

[0088] 处理顶点的某些可编程着色单元支持用户写入高达 N 个不同的裁剪距离输出。例如，N 可以等于 8。为了执行裁剪，着色单元可估计顶点相对于特定裁剪平面的位置。每个裁剪平面都将屏幕分离为应该绘制顶点的区域和应该切除且不绘制顶点的区域。如果顶点相对于裁剪平面具有正值，那么顶点在平面的“正确的”一侧，并且应该被绘制。如果顶点相对于裁剪平面具有负值，那么顶点在平面“错误的”一侧，并且不应被绘制。

[0089] 如一个实施例所述，几何处理管线中的一个或多个着色阶段可以写裁剪距离。将由最后被使能的着色阶段所写的裁剪距离用于裁剪；由之前阶段所写的裁剪距离被简单输出给它们的随后阶段。当实现跨命令列表的状态继承时，不同的线程可“跨接 (hook up)”或使用不同的着色器。相应地，本发明的实施例提供用于自动确定哪一个是最后使用的着色器的技术。基于与该着色器相关联的裁剪信息，硬件可确定已经写入哪个裁剪距离（即，候选用于被裁剪为该距离）。使用跨命令列表的状态继承，当驱动器正创建命令列表时，该驱动器不知道哪个阶段被使能。这样，驱动器不知道最后被使能的阶段是什么。因此，该驱动器不能告知硬件哪个阶段的裁剪距离用于剪切。

[0090] 此外，在一些实施例中，使能位可以与 N 个不同的裁剪距离输出中的每一个相关联，所述 N 个不同的裁剪距离输出与特定命令相关联。可将这一组 N 个使能位和用于配置着色器的、与最后一个着色器相关联的裁剪信息进行逻辑与。

[0091] 例如，可编程处理管线可包括处理点并且确定顶点位置的顶点着色器，以及操作全部基元的几何着色器。在第一配置中，可编程处理管线可被配置以使得在管线中几何着色器在顶点着色器之后被调用。相应地，由最后一个阶段即几何着色器来设置裁剪距离。在第二配置中，可编程处理管线可被配置为在管线中在顶点着色器之后不调用几何着色器（即，空的几何着色器）。在没有几何着色器的第二配置中，由顶点着色器设置裁剪距离。因此，本发明的实施例实施包括用于每个用户裁剪平面的单独使能位的单个方法 SetUserClipEnable()。如所述的，这一组 N 个使能位可以和与最后被使用的着色器相关联的裁剪信息进行逻辑与。

[0092] 4. 预测渲染重写 (override)

[0093] 有时驱动器需要入栈 / 出栈 (push/pop) 预测状态，用于诸如着色器 / 纹理数据头 (texheader)/ 采样器上传的“内部”位块，或者用于应该忽略预测的操作。例如，驱动器可

能需要进行内部绘图调用，以完成并非对应于来自应用程序绘图命令的某些动作。

[0094] 相应地，为了在内部操作之后恢复当前的预测状态，需要知道该状态。本发明的实施例将 SetRenderEnableOverride( ) 方法添加到 API 以重写当前预测状态，提供一级堆栈用于预测状态的入栈 / 出栈。

[0095] 图 8 为根据本发明一个实施例的，用于具有跨命令列表状态继承的多线程处理的方法步骤的流程图。本技术领域的技术人员应该理解，虽然与图 1-7 中的系统结合来描述方法 800，被配置为以任意顺序执行所述方法步骤的任意系统都包括在本发明实施例的范围内。

[0096] 如图所示，方法 800 从步骤 802 开始，其中由处理器执行的驱动器接收包括多个并行线程的应用程序。如图 5 中所示，应用程序可以由程序开发人员编写。在步骤 804 中，驱动器为每个线程生成一个命令列表。如上述图 7 中的描述，每个命令列表都在指向应用程序命令的缓冲区的指针和指向补丁方法的缓冲区的指针之间交替变化。下面在图 9 中详细描述步骤 804。

[0097] 在步骤 806，处理单元执行被包括在与第一线程相关联的第一命令列表中的命令。在一些实施例中，处理单元利用包括在处理管线中的一个或多个处理阶段来执行这些命令。例如，如图 7 所示，处理单元接收被包括在各缓冲区 702、704 中的命令。处理单元可首先执行来自缓冲区 702 的块 A 中的应用程序命令，然后执行来自缓冲区 704 的块 x_0 的补丁方法，然后执行来自缓冲区 702 的块 B 中的应用程序命令，等等。在该命令列表结束时，处理单元停止执行来自第一线程的命令列表的命令，并切换为执行来自第二线程的命令列表的命令。

[0098] 在步骤 808，当处理单元停止执行来自第一线程的命令列表的命令时，驱动器保持该处理器状态。如所述的，将处理器状态定义为与执行命令的处理单元相关联的参数集。被包括在处理器状态中的参数的示例尤其包括 :顶点着色器、几何着色器、像素着色器等的选择、一组绑定到像素着色器的不同纹理，对如何执行混合进行定义的参数，目标渲染表面列表。在步骤 810 中，处理单元执行被包括在与第二线程相关联的第二命令列表中的命令。步骤 810 与上述步骤 806 基本上类似。从而，处理器实现跨命令列表的状态继承。

[0099] 图 9 为根据本发明一个实施例的，用于生成命令列表的方法步骤的流程图。本技术领域的技术人员应该理解，虽然与图 1-7 中的系统结合来描述方法 900，被配置为以任意顺序执行所述方法步骤的任意系统都包括在本发明实施例的范围内。

[0100] 如图所示，方法 900 从步骤 902 开始，其中驱动器接收被包括在应用程序命令的线程中的命令。如图 5 中所述，应用程序可以由应用程序开发人员编写，并且可以包括多个并行线程。如图 8 中所述，在步骤 804，为每个并行线程生成命令列表。

[0101] 在步骤 904，驱动器确定仅使用处理器的已知状态是否能编码命令。问题在于驱动器是否知道有关执行时间 (execution-time) 处理器状态的足够信息来生成方法 (即，命令的硬件表示)。一些方法可以在不知道任何其他的处理器状态的情况下被写得。其他方法依赖于其他的处理器状态，但该处理器状态是在驱动器建立命令列表时被获知的。在命令列表构建期间，这两种情况之一可被立即写入到命令缓冲区中。如果方法的编码依赖于其他状态，并且当构建命令列表时该状态是未知的，那么此时该方法不能被写到命令缓冲区——必须被延迟到命令列表被执行并且执行时间状态已知。

[0102] 如果驱动器确定仅使用处理器已知的状态就可编码命令，那么方法 900 进入步骤 906。在步骤 906，驱动器将命令插入与应用程序命令相关联的第一命令缓冲区。如图 7 所示，第一命令缓冲区即缓冲区 702，可被分为应用程序命令块。随后将指向适当的应用程序命令块的指针增加到命令列表中。

[0103] 在步骤 908，驱动器确定线程中是否包括其他的命令。如果线程中包括其他的命令，那么方法 900 转回到步骤 902，如上所述。当生成命令列表时，方法 900 遍历被包括在线程中的每个应用程序命令。如果线程中不包括其他的命令，那么方法 900 结束。

[0104] 在步骤 904 中，如果驱动器确定仅使用处理器已知的状态不能编码命令，那么方法 900 进入步骤 910。在步骤 910，驱动器将有关需要什么补丁方法的信息存储到边带队列中。之后，处理该队列，并且当执行命令列表时写入补丁方法。例如，将索引大小作为非独立的状态参数避免了对于补丁的需求。当在绘图方法中编码索引大小时，则索引大小未知时所发送的任意绘图命令随后都将需要打补丁。其目的在于减小补丁的数量。

[0105] 总而言之，本发明实施例提供用于实现跨命令列表的状态继承的技术。每个命令列表在指向应用程序命令的指针和指向补丁方法的指针之间交替变化。在命令列表构建期间，在任何遇到依赖于未知的处理器状态的应用程序命令的时候，将补丁方法插入命令列表。

[0106] 有利地，相对于不提供跨命令列表的状态继承的现有技术，获得了更高的处理效率。因为每次执行不同的线程时，不需要重新设置处理器为“空白状态”，所以需要的处理开销更少。

[0107] UnbindAll（全部释放）方法

[0108] 如上所述，DX11 不允许跨命令列表的状态继承。相反，处理器状态在每个命令列表开始时被重置为所谓的“空白状态”。也就是指，每个用户模式驱动器线程在命令列表开始时在处理器中设置所有的状态参数。在 DX11 中，使用几十或几百的命令来将处理器状态重置为空白状态所增加的处理开销降低了系统的效率，因此，降低了整体性能。

[0109] 在一个实施例中，空白状态实质上是一组用于所有类方法状态的初始条件，其中没有资源被绑定，例如，没有纹理数据头、没有纹理采样器、没有常量缓冲区以及没有渲染目标。在 DX11 中，在每个命令列表开始时，驱动器将插入所有的状态设置方法以设置初始条件。在 DX11 中，采用 819 个单独的方法将所有资源逐位置 (slot-by-slot) 地释放：

[0110] (5 种着色类型)*((每种着色类型的 128 个纹理数据头绑定方法）

[0111] +（每种着色类型 16 个采样器绑定方法）

[0112] +（每种着色类型 18 个常量缓冲区绑定方法））

[0113] +(9 个目标“绑定”方法）

[0114] ＝ 819 个方法

[0115] 每次执行不同的命令列表时都执行 819 个方法会占用大量的处理资源。相应地，本发明实施例实施 UnbindAll（）方法，其使用一种方法来释放所有内容。实施这一方法提高了驱动器的性能，并且减少了 GPU 中方法所需的带宽。

[0116] 在一个实施例中，将诸如纹理数据头的每个状态参数存储在存储器单元的不同行中。为了实现 UnbindAll（）方法，有效位被附加到存储单元的每一行。为了释放所有的状态参数，将每个有效位设置为无效状态。

[0117] 在另一个实施例中，如果状态参数被存储在高速缓存存储器中，则可通过归零该高速缓存存储器中的一个或多个高速缓存线来实现 UnbindAll( ) 方法。而在另一个实施例中，如果状态参数被存储在存储体存储器 (banked memory) 中，通过立即清空一个或多个存储体来实现 UnbindAll( ) 方法。

[0118] 图 10 为根据本发明一个实施例的，使用 UnbindAll( ) 方法实现多线程处理的方法步骤的流程图。本技术领域的技术人员应该理解，虽然与图 1-7 中的系统结合来描述方法 1000，被配置为以任意顺序执行所述方法步骤的任意系统都包括在本发明实施例的范围内。

[0119] 如图所示，方法 1000 从步骤 1002 开始，其中驱动器接收包括多个并行线程的应用程序。在步骤 1004 中，驱动器为每个线程生成命令列表。在步骤 1006，处理器执行与第一命令列表相关联的命令，所述第一命令列表与第一线程相关联。步骤 1002、1004、1006 分别与如上所述的步骤 802、804、806 基本上相似。

[0120] 在步骤 1008，处理器执行被包括在与第二线程相关联的第二命令列表中的 UnbindAll( ) 方法。如上所述，UnbindAll( ) 方法采用一个方法来释放所有状态参数。在一个实施例中，UnbindAll( ) 方法可作为第一个方法来被插入每个命令列表中。在另一个实施例中，UnbindAll( ) 方法可作为最后一个方法来被插入每个命令列表中。在步骤 1008 中，处理器执行与第二命令列表相关联的其余命令。步骤 1010 可以与如上所述的步骤 810 基本上相似。

[0121] 本发明的一个实施例可以被实现为与计算机系统一同使用的程序产品。程序产品的程序对实施例的功能（包括在此描述的方法）进行定义，并且可被包含在各种各样的计算机可读存储介质内。说明性的计算机可读存储介质包括但不限于 :(i) 信息在其上永久保存的非可写存储介质（例如，计算机内的只读存储装置，诸如对 CD-ROM 驱动器可读的 CD-ROM 盘、闪存、ROM 芯片或者任意类型的固态非易失性半导体存储器）;以及 (ii) 其上存储有可改变的信息的可写存储介质（例如，软盘驱动器内的软盘，或硬盘驱动器，或任意类型的固态随机存取半导体存储器）。

[0122] 以上已经参考具体实施例对本发明进行了描述。然而，本技术领域的技术人员应该理解，可以进行各种修改和变化，而不脱离如所附权利要求所阐释的本发明的较宽精神和范围。相应地，前面的描述和附图应被视为是示例性的而非限制性的。

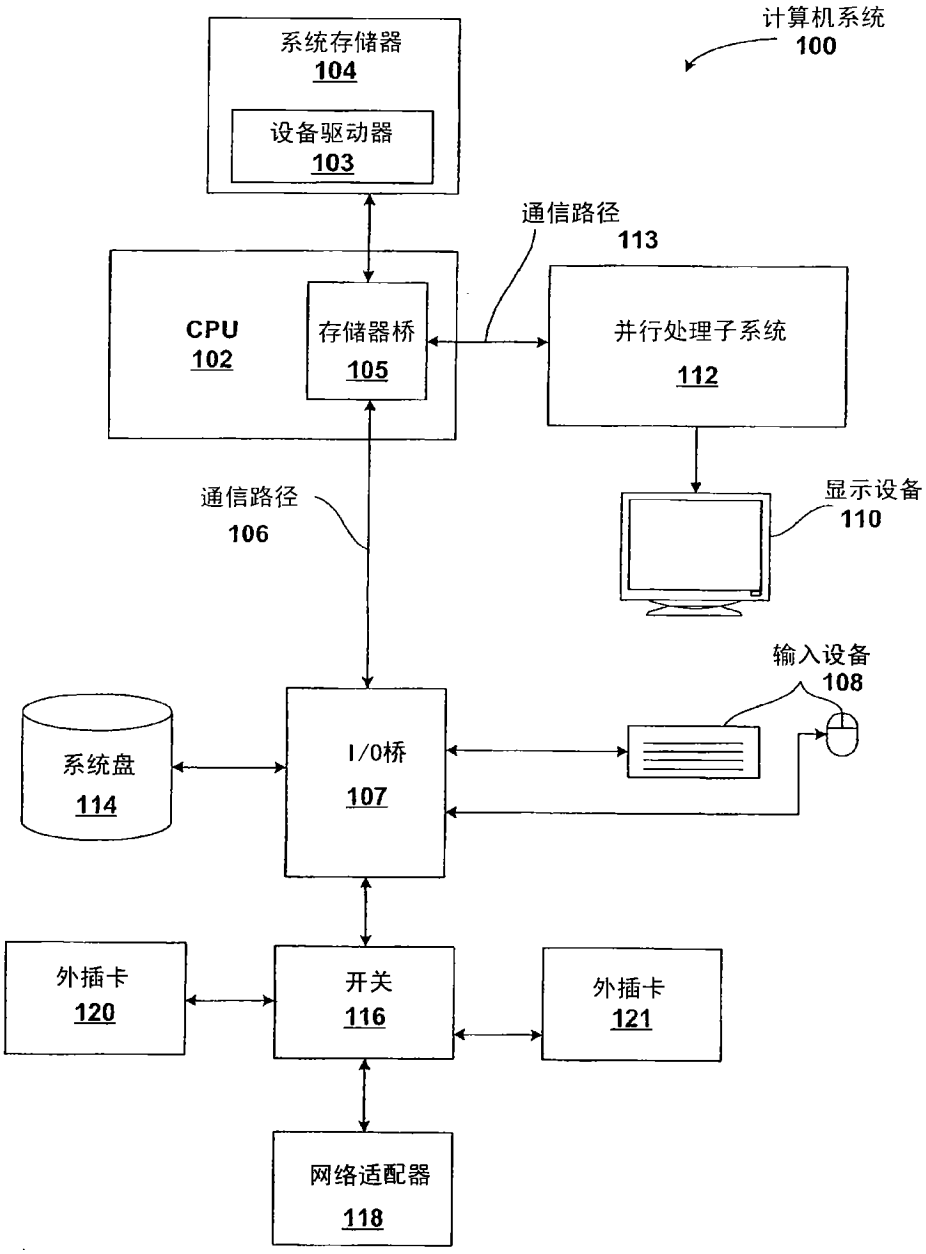

计算机系统
100
系统存储器
104
设备驱动器
103
通信路径
113
CPU
102
存储器桥
105
并行处理子系统
112
通信路径
106
显示设备
110
输入设备
108
系统盘
114
I/O桥
107
外插卡
120
开关
116
外插卡
121
网络适配器
118

图 1

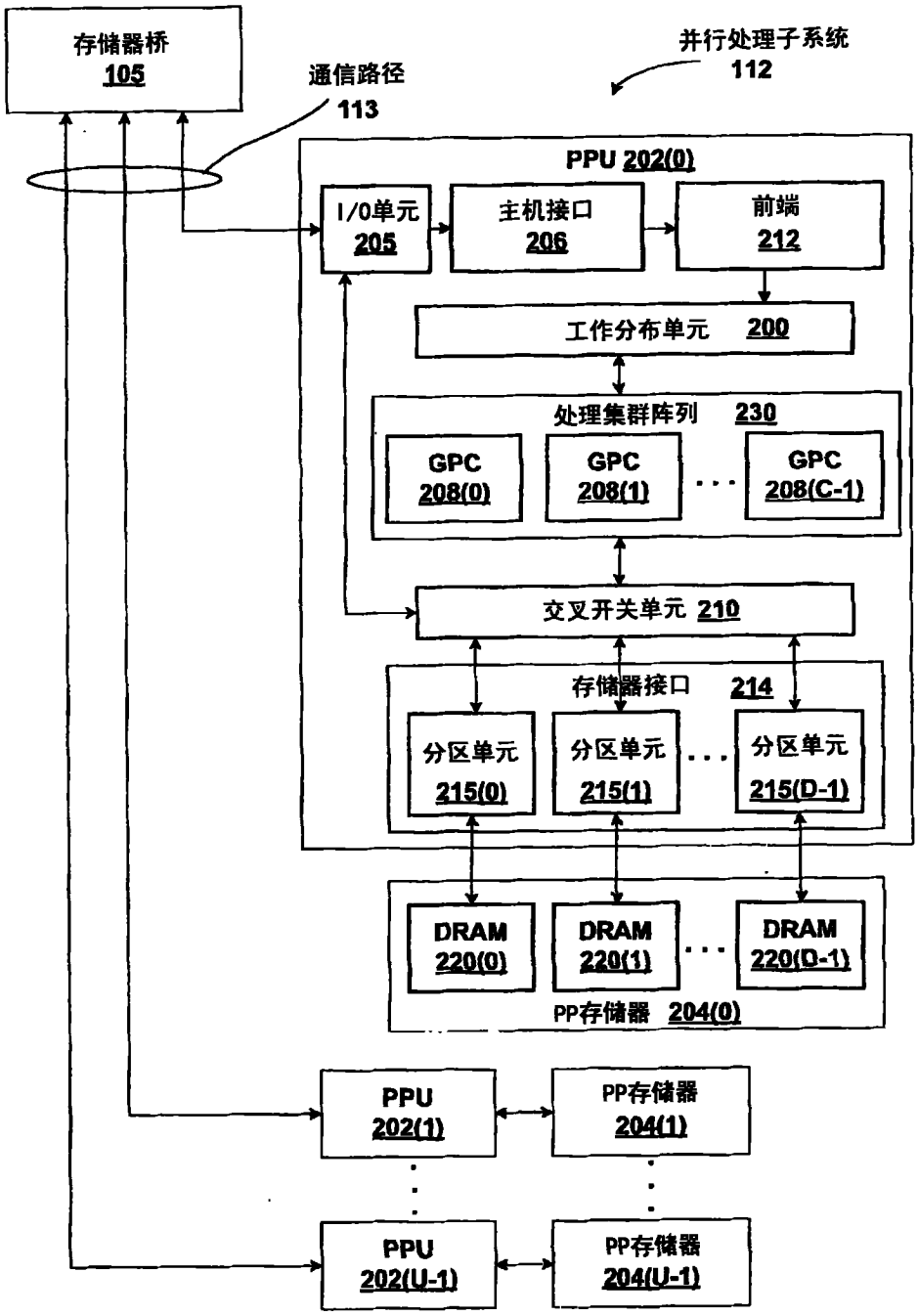

存储器桥
105
通信路径
113
并行处理子系统
112
PPU 202(0)
I/O单元
205
主机接口
206
前端
212
工作分布单元 200
处理集群阵列 230
GPC
208(0)
GPC
208(1)
GPC
208(C-1)
交叉开关单元 210
存储器接口 214
分区单元
215(0)
分区单元
215(1)
分区单元
215(D-1)
DRAM
220(0)
DRAM
220(1)
DRAM
220(D-1)
PP存储器 204(0)
PPU
202(1)
PP存储器
204(1)
PPU
202(U-1)
PP存储器
204(U-1)

图 2

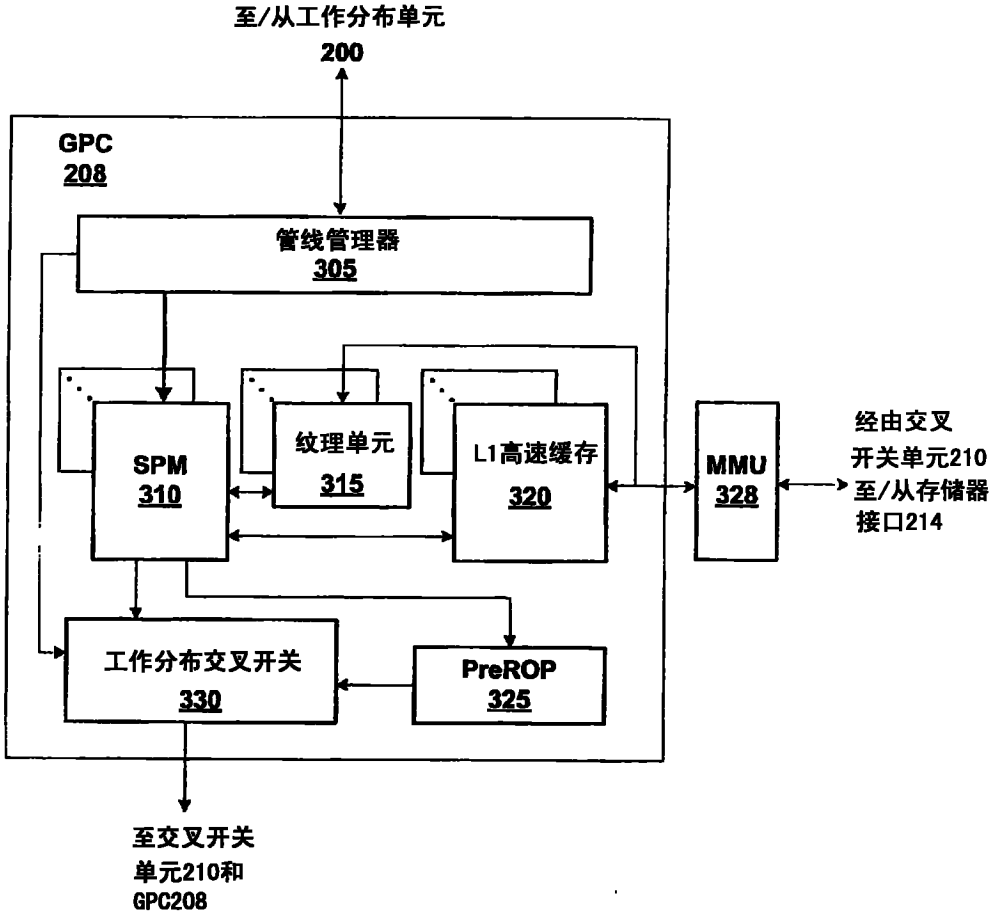

至/从工作分布单元
200
GPC
208
管线管理器
305
SPM
310
纹理单元
315
L1高速缓存
320
MMU
328
经由交叉
开关单元210
至/从存储器
接口214
工作分布交叉开关
330
PreROP
325
至交叉开关
单元210和
GPC208

图 3A

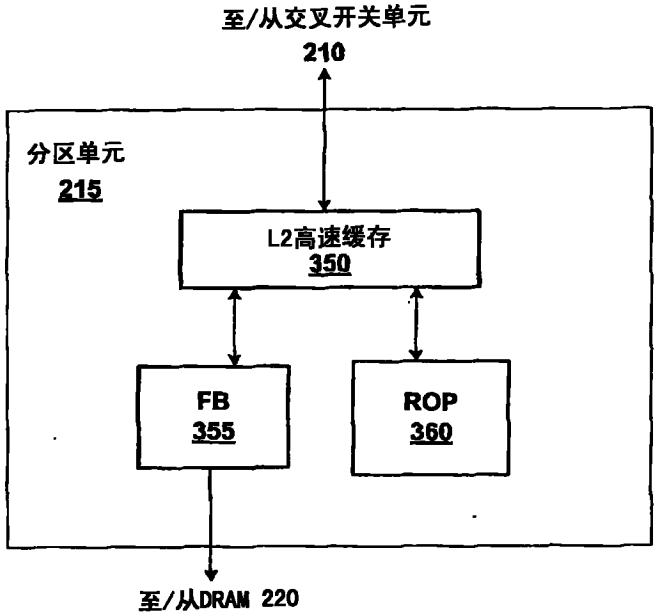

至/从交叉开关单元
210
分区单元
215
L2高速缓存
350
FB
355
ROP
360
至/从DRAM 220

图 3B

示意图

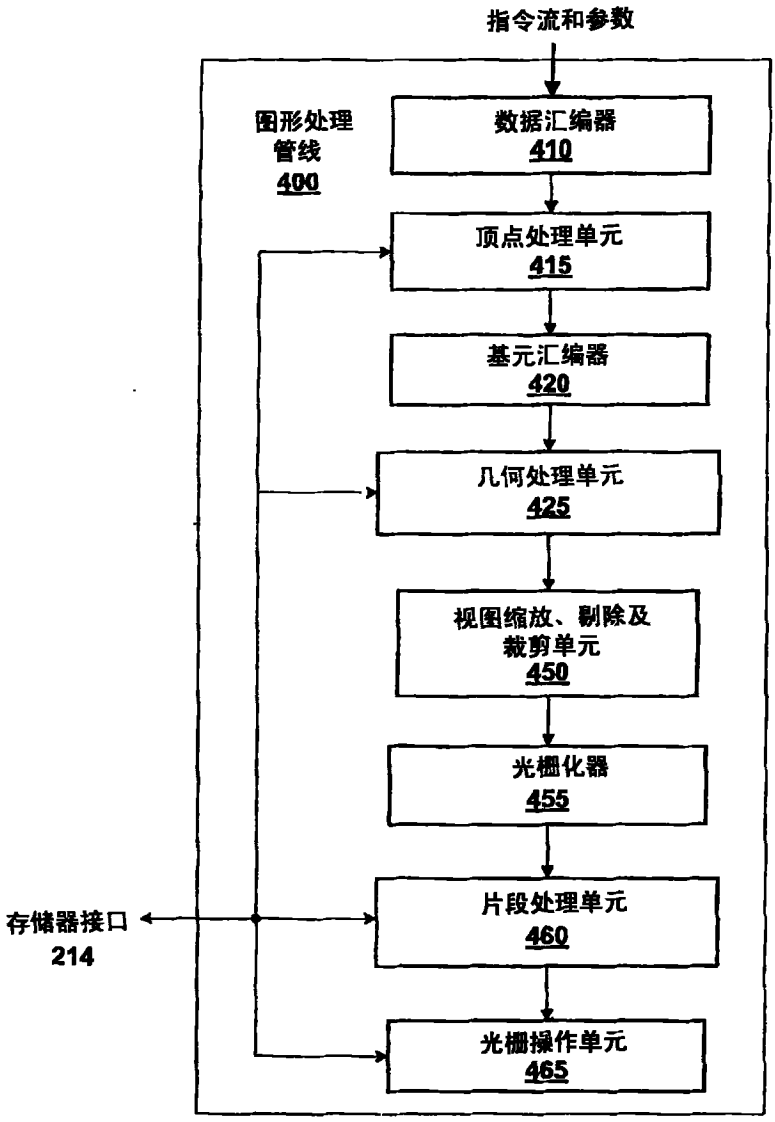

指令流和参数
图形处理管线
400
数据汇编器
410
顶点处理单元
415
基元汇编器
420
几何处理单元
425
视图缩放、剔除及裁剪单元
450
光栅化器
455
片段处理单元
460
光栅操作单元
465
存储器接口
214

图 4

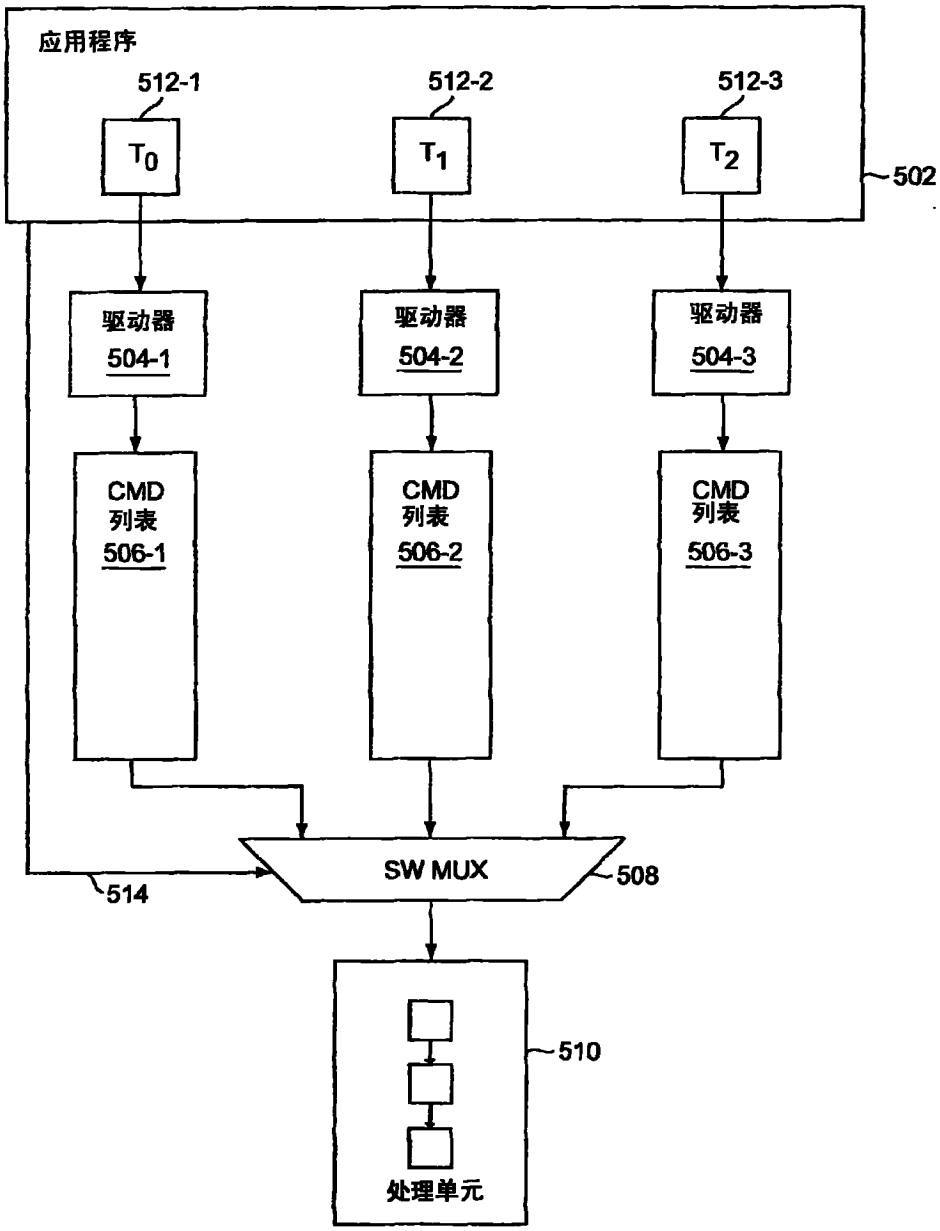

应用程序
512-1
512-2
512-3
T0
T1
T2
502
驱动器
504-1
驱动器
504-2
驱动器
504-3
CMD
列表
506-1
CMD
列表
506-2
CMD
列表
506-3
SW MUX
508
514
510
处理单元

图 5

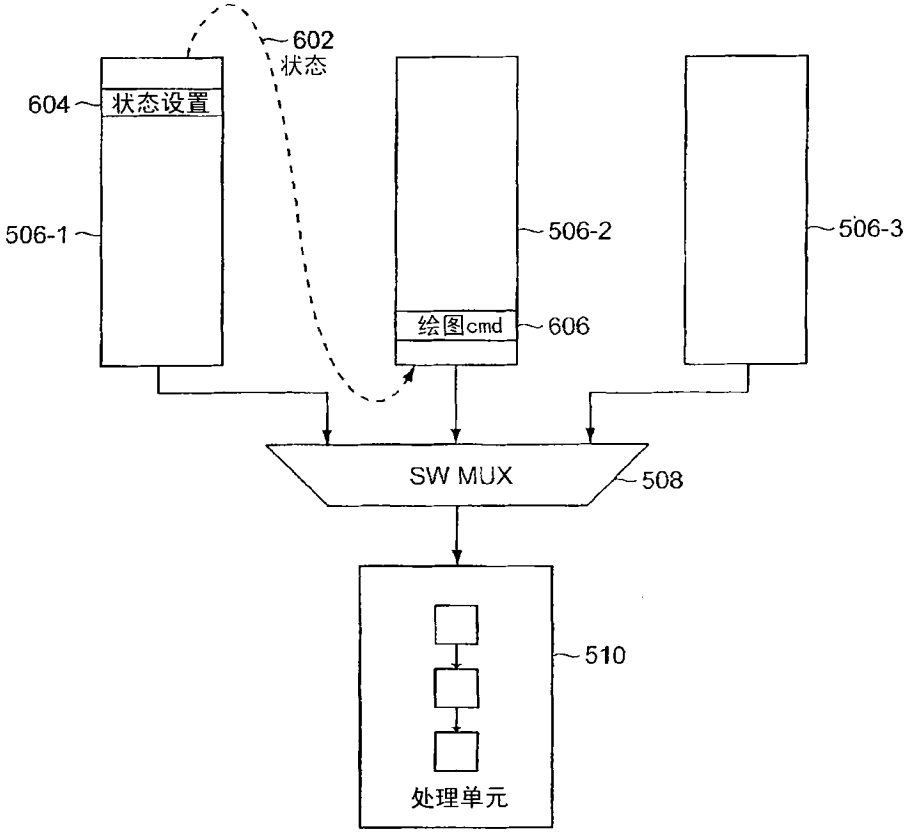


图 6

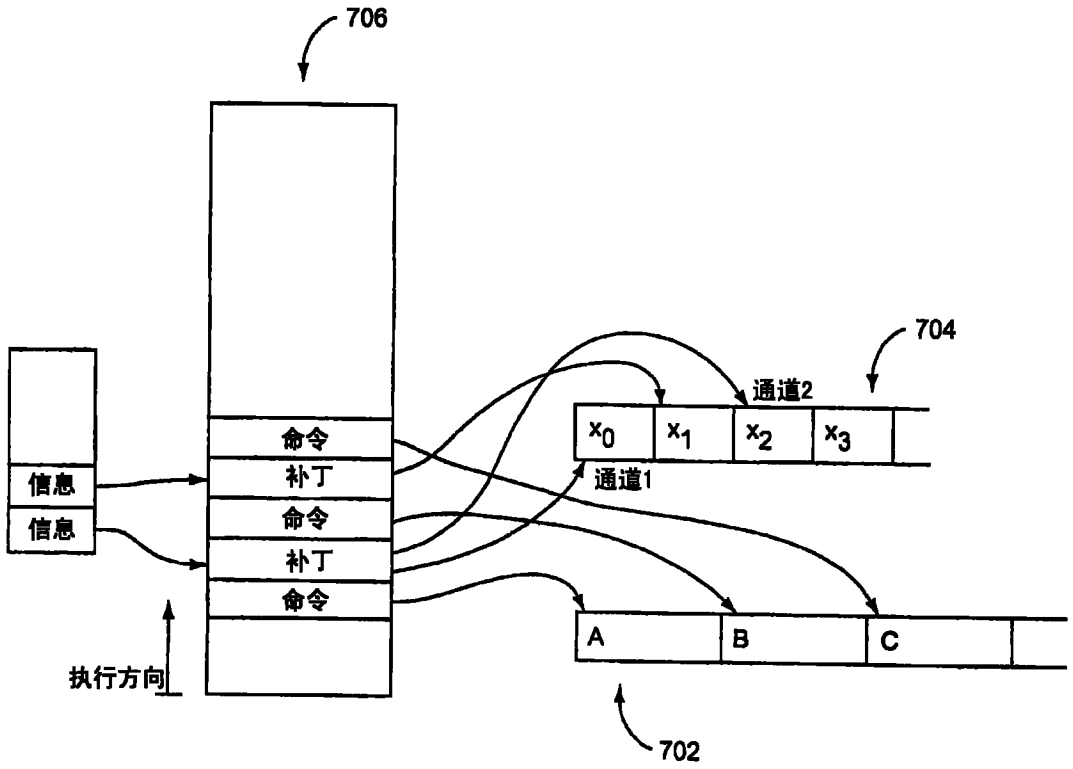


图 7

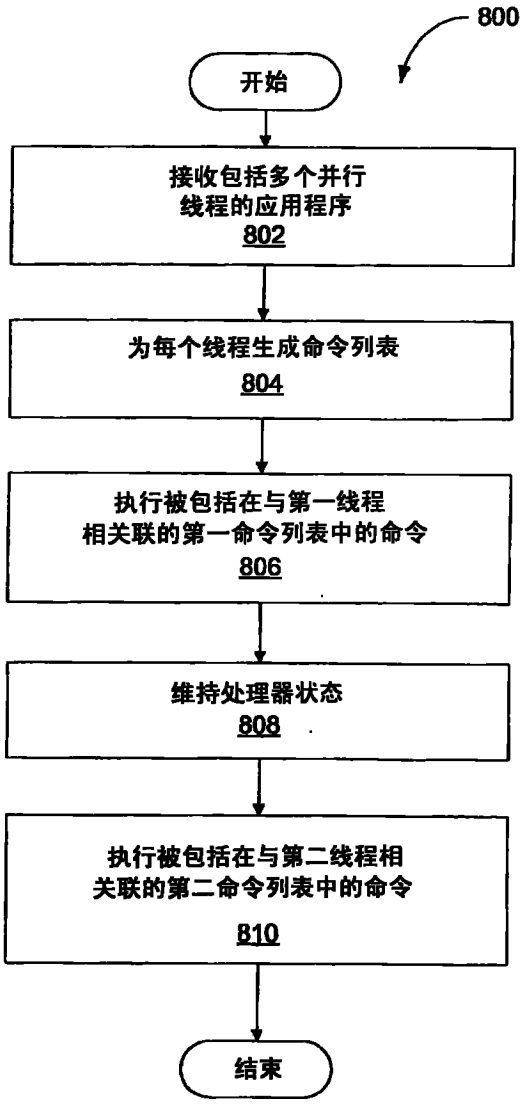

800
开始
接收包括多个并行线程的应用程序
802
为每个线程生成命令列表
804
执行被包括在与第一线程相关联的第一命令列表中的命令
806
维持处理器状态
808
执行被包括在与第二线程相关联的第二命令列表中的命令
810
结束

图 8

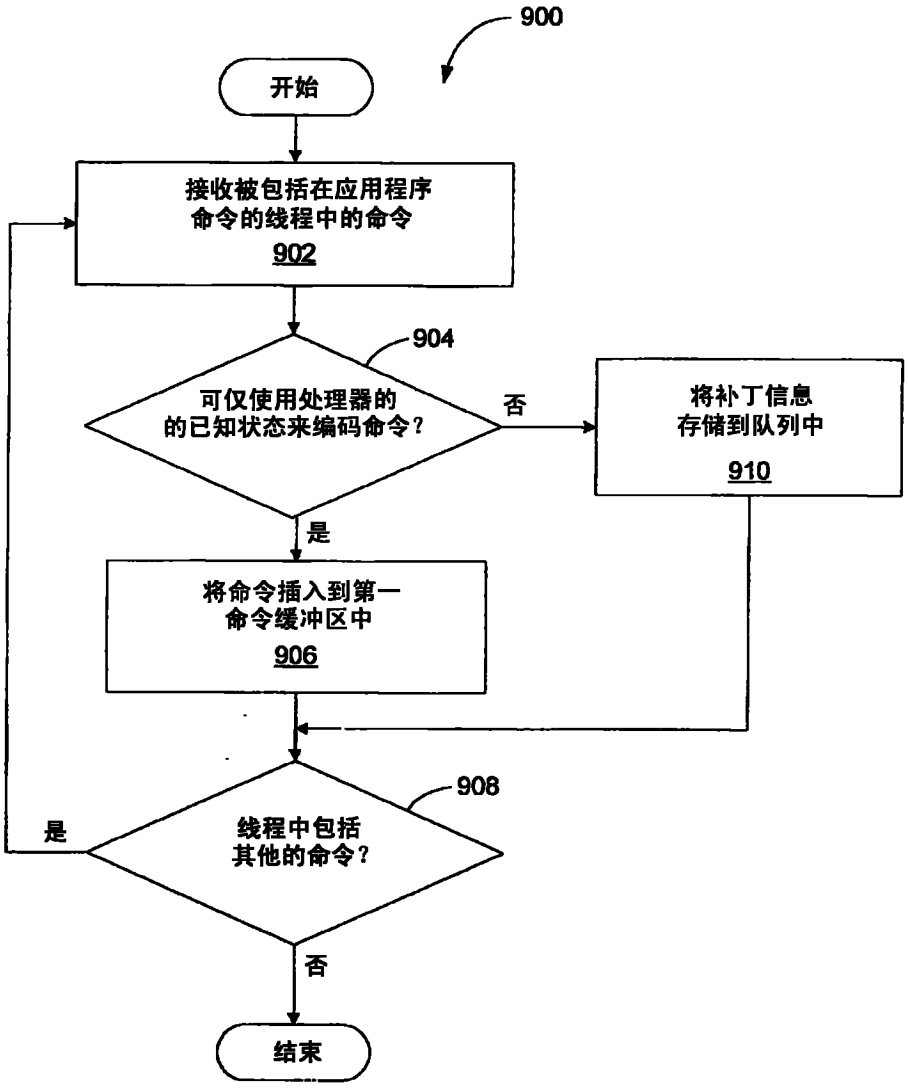

900
开始
接收被包括在应用程序命令的线程中的命令
902
904
可仅使用处理器的的已知状态来编码命令？
否
将补丁信息存储到队列中
910
是
将命令插入到第一命令缓冲区中
906
908
线程中包括其他的命令？
是
否
结束

图 9

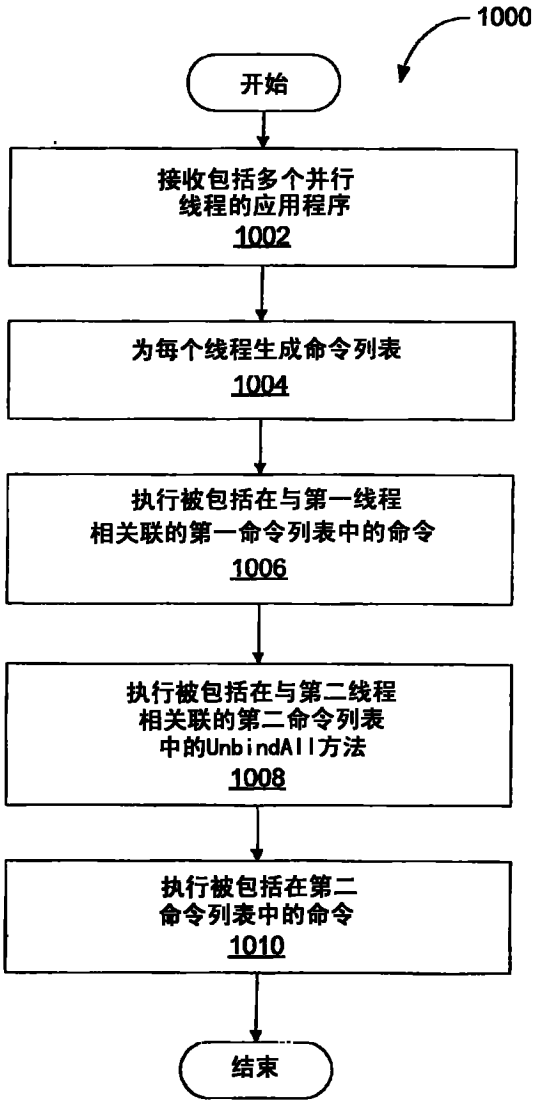

1000
开始
接收包括多个并行线程的应用程序
1002
为每个线程生成命令列表
1004
执行被包括在与第一线程相关联的第一命令列表中的命令
1006
执行被包括在与第二线程相关联的第二命令列表中的UnbindAll方法
1008
执行被包括在第二命令列表中的命令
1010
结束

图 10