(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 102713837 B
(45)授权公告日 2016.03.02

(21)申请号 201080053153.2

(22)申请日 2010.09.23

(30)优先权数据
61/245,222 2009.09.23 US
61/246,040 2009.09.25 US
12/888,409 2010.09.22 US

(85)PCT国际申请进入国家阶段日
2012.05.23

(86)PCT国际申请的申请数据
PCT/US2010/049978 2010.09.23

(87)PCT国际申请的公布数据
WO2011/038106 EN 2011.03.31

(73)专利权人 辉达公司
地址 美国加利福尼亚州

(72)发明人 约翰·R·尼科尔斯
布雷特·W·库恩
迈克尔·C·希巴依

(74)专利代理机构 北京市磐华律师事务所 11336
代理人 顾珊 魏宁

(51)Int.Cl.
G06F 9/30(2006.01)

(56)对比文件
US 2005/0055540 A1,2005.03.10,
US 2005/0138295 A1,2005.06.23,
CN 1991792 A,2007.07.04,
US 6877086 B1,2005.04.05,

审查员 邢白灵

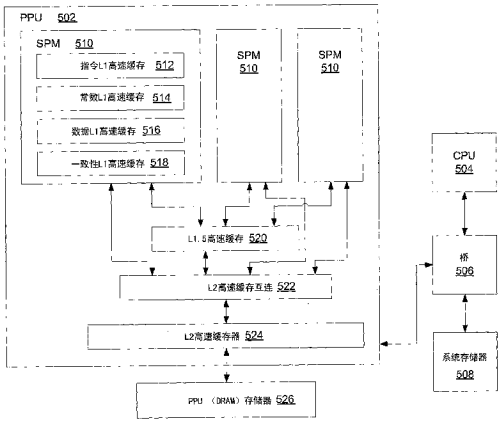
权利要求书1页 说明书19页 附图9页

(54)发明名称

用于管理并行高速缓存层级的指令

(57)摘要

一种用于管理处理单元中的并行高速缓存层级的方法。该方法包括从调度器单元接收指令，其中该指令包括加载指令或存储指令；确定该指令包括高速缓存操作修饰符，该修饰符识别用于在并行高速缓存层级的一个或多个级别上对与指令相关的数据进行高速缓存的策略；以及执行该指令，并且基于该高速缓存操作修饰符对与该指令相关联的数据进行高速缓存。



CN 102713837 B

1. 一种用于管理处理单元中的并行高速缓存层级的方法，该方法包括：

从调度器单元接收指令，其中所述指令包括加载指令或者存储指令，并且其中所述指令进一步地与识别存储器区域的地址相关联；

确定所述指令包括高速缓存操作修饰符，所述修饰符识别用于在所述并行高速缓存层级的一个或多个级别上对与所述指令相关联的数据进行高速缓存的策略，其中所述并行高速缓存层级包括 L1 高速缓存级别以及 L2 高速缓存级别；以及

执行所述指令，并且基于所述高速缓存操作修饰符对与所述指令相关联的所述数据进行高速缓存，其中如果所述地址位于本地存储器区域之中，则对所述数据在 L1 高速缓存级别和 L2 高速缓存级别上进行高速缓存，并且如果所述地址位于全局区域中，则对所述数据在所述 L2 高速缓存级别上进行高速缓存，而非在所述 L1 高速缓存级别上进行高速缓存。

2. 如权利要求 1 的方法，其中包括在所述处理单元中的每个处理器都包括位于所述 L1 高速缓存级别上的不同的 L1 高速缓存，所述 L2 高速缓存级别包括每个处理器经配置均可对其进行访问的至少一个 L2 高速缓存。

3. 如权利要求 1 的方法，其中实现所述高速缓存操作修饰符，以在加载指令之后，无效和丢弃被高速缓存在所述 L1 高速缓存中的所述数据。

4. 如权利要求 1 的方法，其中实现所述高速缓存操作修饰符，以使得利用回写策略或者透写策略，对与存储指令相关联的数据进行高速缓存。

5. 如权利要求 1 的方法，其中实现所述高速缓存操作修饰符，以使得利用易失性始终获取策略，对与加载指令相关联的数据进行高速缓存。

6. 一种用于管理并行高速缓存层级的系统，该系统包括：

处理器，其被配置为：

接收指令，其中所述指令包括加载指令或者存储指令，并且其中所述指令进一步地与识别存储器区域的地址相关联，

确定所述指令包括高速缓存操作修饰符，所述修饰符识别用于在所述并行高速缓存层级的一个或多个级别上对与所述指令相关联的数据进行高速缓存的策略，其中所述并行高速缓存层级包括 L1 高速缓存级别以及 L2 高速缓存级别以及

执行所述指令，并且基于所述高速缓存操作修饰符对与所述指令相关联的所述数据进行高速缓存，其中如果所述地址位于本地存储器区域之中，则对所述数据在 L1 高速缓存级别和 L2 高速缓存级别上进行高速缓存，并且如果所述地址位于全局区域中，则对所述数据在所述 L2 高速缓存级别上进行高速缓存，而非在所述 L1 高速缓存级别上进行高速缓存。

用于管理并行高速缓存层级的指令

[0001] 相关申请的交叉引用

[0002] 本申请要求于 2009 年 9 月 23 日申请的编号为 61/245,222 的美国临时专利申请，于 2009 年 9 月 25 日申请的编号为 61/246,040 的美国临时专利申请，以及于 2010 年 9 月 22 日申请的编号为 12/888,409 的美国专利申请的优先权。

技术领域

[0003] 本发明的实施例总体上涉及多线程处理，以及，更具体地，涉及一组能够使应用软件对并行线程处理器中的并行高速缓存层级 (cache hierarchy) 进行管理的指令。

背景技术

[0004] 传统的高速缓存策略技术试图通过确定加载和存储的操作模式，来努力预测应该对哪些数据进行高速缓存和 / 或清除。然而，在高度多线程并行处理器中，要对模式进行确定则是极端困难的。例如，可以同时执行超过 10000 个线程，而这使得模式检测是非常困难的。

[0005] 此外，高度多线程并行处理器，诸如图形处理单元 (GPU)，与诸如 CPU(中央处理器) 内核的串行处理器相比，其每一个线程仅具有相对较小的高速缓存容量。

[0006] 相应地，本技术领域需要的是 :能够有效地利用多线程并行处理器中有限的高速缓存容量的高速缓存管理技术。

发明内容

[0007] 本发明的实施例提供了使并行多线程应用软件能够协调并发的线程以有效地利用工作集 (working-set) 容量有限的高速缓存的指令。本发明的实施例提供了针对加载 / 存储存储器访问指令的显式高速缓存行为修饰符 (modifier)。该修饰符可以使程序员和 / 或编译器能够对以下行为指定高速缓存的优化 :针对特定的加载 / 存储存储器指令的易失和非高速缓存行为、工作集行为以及流行为。

[0008] 本发明的一个实施例提供了一种用于管理处理单元中的并行高速缓存层级的方法。该方法包括从调度单元接收指令，其中所述指令包括加载指令或存储指令 ;确定该指令包括高速缓存操作修饰符，所述修饰符识别用于在所述并行高速缓存层级的一个或多个级别上对与指令相关联的数据进行高速缓存的策略 ;以及执行所述指令，并且基于所述高速缓存操作修饰符对与所述指令相关联的所述数据进行高速缓存。

[0009] 有利地，本发明的实施例允许程序员和 / 或编译器来指定在哪个高速缓存级别上对数据进行高速缓存。这使得程序的执行以及数据的访问更加有效率。

附图说明

[0010] 为了详细地理解本发明的上述特征，对于以上简要说明的发明，可以参照实施例进行更为具体的描述，其中一些实施例示出于附图中。然而，应注意的是，附图中示出的只

是本发明的代表性实施例，因此不应被认为是对本发明的范围的限制，本发明可以适用于其他同等有效的实施例。

[0011] 图 1 为示出了被配置以实现本发明一个或多个方面的计算机系统的框图。

[0012] 图 2 为根据本发明的一个实施例，用于图 1 的计算机系统的并行处理子系统的框图。

[0013] 图 3A 为根据本发明的一个实施例，图 2 中一个 PPU 内的 GPC 的框图。

[0014] 图 3B 为根据本发明的一个实施例，图 2 中一个 PPU 内的分区单元的框图。

[0015] 图 3C 为根据本发明的一个实施例，图 3A 中 SPM 的一部分的框图。

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

[0017] 图 5 为根据本发明的一个实施例，示出了并行线程处理器中的并行高速缓存层级的示意图。

[0018] 图 6 为根据本发明一个实施例，用于处理加载存储器访问指令的方法步骤的流程图。

[0019] 图 7 为根据本发明的一个实施例，用于处理存储存储器访问指令的方法步骤的流程图。

具体实施方式

[0020] 在下面的说明中，将阐述大量的细节以提供对本发明更为彻底的理解。然而，显而易见地是，对于本领域技术人员来说缺少这些细节的一个或多个也可以实施本发明。在其他实例中，没有描述公知的特征，以免对本发明造成混淆。

[0021] 系统概述

[0022] 图 1 的框图示出被配置为实现本发明一个或多个方面的计算机系统 100。计算机系统 100 包括中央处理器 (CPU)102 以及经由互连路径进行通信的系统存储器 104，该互连路径可包括存储器桥 105。存储器桥 105，其可能是例如北桥芯片，其经由总线或者其它的通信路径 106（例如超传输链路）与 I/O（输入 / 输出）桥 107 相连。I/O 桥 107，其例如可能是南桥芯片，从一个或多个用户输入设备 108（例如，键盘，鼠标）接收用户输入，并且将该输入经由路径 106 和存储器桥 105 转发给 CPU 102。并行处理子系统 112 经由总线或者其它的通信路径 113（例如，PCI Express，加速图形端口，或者超传输链路）耦合至存储器桥 105；在一个实施例中，并行处理子系统 112 为图形子系统，其向显示设备 110（例如，传统的基于 CRT 或者 LCD 的监视器）传送像素。系统磁盘 114 也连接到 I/O 桥 107。开关 116 提供了在 I/O 桥 107 及其他诸如网络接口卡 118 的组件与不同的外插卡 (add-in card)120 和 121 之间的连接。其它的组件（未明确的示出），包括 USB 或者其它的端口连接、CD 驱动器、DVD 驱动器、胶片录制设备等等，也可以连接到 I/O 桥 107。可以使用任何适宜的协议，诸如 PCI（外围组件互连）、PCI-Express、AGP（加速图形端口）、超传输或者任何其它的总线或者点对点通信协议来实现图 1 中将各组件互连的通信路径，并且不同设备之间的连接也可使用现有技术中已知的不同的协议。

[0023] 在一个实施例中，并行处理子系统 112 结合了被优化用于图形和视频处理的电路，包括例如视频输出电路，并且构成了图形处理单元 (GPU)。在另一个实施例中，并行处

理子系统 112 结合了被优化用于通用处理的电路，同时保留了底层 (underlying) 的计算架构，在这里将对其进行更为详细地描述。在另一实施例中，可将并行处理子系统 112 与一个或多个其它的系统元件，诸如存储器桥 105、CPU 102 以及 I/O 桥 107 集成，来形成片上系统 (SoC)。

[0024] 可以理解到在这里所显示的系统是说明性的，并且可对其进行变化和修改。可根据需要修改连接拓扑结构，包括桥的数量和布置、CPU 102 的数量以及并行处理子系统 112 的数量。例如，在一些实施例中，系统存储器 104 直接地，而不是通过桥来连接到 CPU 102，并且其它的设备经由存储器桥 105 以及 CPU 102 与系统存储器 104 进行通信。在其它可供选择的拓扑结构中，并行处理子系统 112 连接到 I/O 桥 107，或者直接连接到 CPU102，而不是连接到存储器桥 105。但在其它的实施例中，可将 I/O 桥 107 以及存储器桥 105 集成到单一的芯片中。大量的实施例可以包括两个或更多的 CPU 102 以及两个或更多的并行处理系统 112。在这里所显示的具体的组件是可选择的；例如，可支持任何数量的外插卡或者外围设备。在一些实施例中，删除了开关 116，并且网络适配器 118 和外插卡 120、121 直接连接到 I/O 桥 107。

[0025] 图 2 示出了根据本发明一个实施例的并行处理子系统 112。如同所显示的，并行处理子系统 112 包括一个或多个并行处理单元 (PPU)202，其每一个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（此处，相似对象的多个实体使用标识该对象的参考数字以及根据需要结合标识该实体的带有括号的数字来表示）。可以使用一个或多个诸如可编程处理器、专用集成电路 (ASIC) 或者存储器器件的集成电路设备来实现，或者以任何其它技术上可行的方式来实现。

[0026] 再次参考图 1，在一些实施例中，并行处理子系统 112 中的一些或者所有的 PPU 202，都是具有渲染管线的图形处理器，可以被配置为执行与下述各项相关的各种任务：从图形数据中生成像素数据，所述图形数据是由 CPU 102 和 / 或系统存储器 104 经由存储器桥 105 以及总线 113 所提供的；与本地并行处理存储器 204（其能被作为包括了例如传统帧缓冲区的图形存储器来使用）交互来存储并更新像素数据；将像素数据输送到显示设备 110，等等。在一些实施例中，并行处理子系统 112 可以包括一个或多个作为图形处理器来操作的 PPU 202，以及一个或多个被用于通用计算的其他 PPU 202。PPU 可以是相同的或者是不同的，并且每个 PPU 可具有其自己专用的并行处理存储器装置或者非专用的并行处理存储器装置。一个或多个 PPU 202 可以向显示设备 110 输出数据，或者每一个 PPU 202 都可以向一个或多个显示设备 110 输出数据。

[0027] 操作时，CPU 102 是计算机系统 100 的主处理器，其控制并且协调其他系统组件的操作。具体地，CPU 102 发出控制 PPU 202 操作的命令。在一些实施例中，CPU 102 对于每一个 PPU 202 都将命令流写入到入栈缓冲区 (pushbuffer) 中（未明确地在图 1 或者图 2 中示出），该入栈缓冲区可位于系统存储器 104、并行处理存储器 204、或者可为 CPU 102 和 PPU 202 访问的另外的存储位置中。PPU 202 从入栈缓冲区中读取命令流，然后相对于 CPU 102 的操作异步地执行命令。

[0028] 现在返回参考图 2，每一个 PPU 202 都包括 I/O（输入 / 输出）单元 205，该单元经由通信路径 113 与计算机系统 100 其余的部分相互通信，该通信路径 113 连接到存储器桥 105（或者在另一个替代实施例中，直接连接到 CPU 102）。PPU 202 与计算机系统 100 其余

部分的连接也可以改变。在一些实施例中，可将并行处理子系统 112 实现为外插卡，该卡可以被插入到计算机系统 100 的扩展槽中。在其它的实施例中，可以将 PPU 202 与诸如存储器桥 105 或 I/O 桥 107 的总线桥一起集成到单一芯片上。但在其它的实施例中，可将 PPU 202 的某些或者所有的元件与 CPU 102 一起集成到单个芯片上。

[0029] 在一个实施例中，通信路径 113 是 PCI-EXPRESS 链路，其中对于每个 PPU 202 都分配了专用通道 (lane)，如同现有技术中所已知的。也可以使用其它的通信路径。I/O 单元 205 生成用于在通信路径 113 上传输的数据包（或者其它的信号），以及从通信路径 113 上接收所有输入进来的数据包（或者其它的信号），将输入的数据包引导到 PPU 202 的适当的组件。例如，可将与处理任务相关的命令引导到主机接口 206，而可将与存储器操作相关的命令（例如，对并行处理存储器 204 的读或者写）引导到存储器交叉开关单元 210。主机接口 206 读取每个入栈缓冲区并且将由入栈缓冲区指定的工作输出给前端 212。

[0030] 每一个 PPU 202 都有利地实现了高度的并行处理构架。如同所详细显示地，PPU 202(0) 包括了处理集群阵列 230，该处理集群阵列 230 包括 C 个通用处理集群 (GPC) 208，其中 C ≥ 1。每个 GPC 208 都能够并发地执行大量的（例如数百或者数千个）线程，其中每个线程都是程序的实例。在不同的应用程序中，不同的 GPC 208 可被分配用于处理不同类型的程序或者用于执行不同类型的计算。例如，在图形应用程序中，第一组 GPC 208 可被分配用于执行曲面细分 (tessellation) 操作以及用于对曲面片生成基元拓扑，并且第二组 GPC 208 可被分配用于执行曲面细分着色以对于基元拓扑的曲面片参数进行评价并确定顶点位置及每个顶点的其他属性。GPC208 的分配可依据每个类型的程序或计算产生的工作量而不同。

[0031] GPC 208 经由工作分布单元 200 接收所要执行的处理任务，工作分布单元 200 从前端单元 212 接收定义处理任务的命令。处理任务包括将要处理的数据索引，例如表面（曲面片）数据、基元数据、顶点数据和 / 或像素数据，以及定义了如何对该数据进行处理（例如，将要执行什么程序）的状态参数和命令。可将工作分布单元 200 配置为获取与任务相对应的索引，或者工作分布单元 200 可以从前端 212 接收索引。前端 212 确保，在由入栈缓冲区所指定的处理开始之前，GPC 208 被配置成为有效的状态。

[0032] 当 PPU 202 用于图形处理时，例如，将对于每个曲面片的处理工作量分成近似相等规模的任务，以使得可以将曲面细分处理分布给多个 GPC208。可将工作分布单元 200 配置为以能够为多个 GPC 208 提供任务进行处理的频率来生成任务。相反地，在传统的系统中，典型地是由单一的处理引擎来执行处理，而同时其他处理引擎则是保持空闲，在开始它们的处理任务之前等待单个的处理引擎完成其任务。在本发明的一些实施例中，GPC 208 的各部分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 (vertex shading) 和拓扑生成，第二部分可被配置为执行细分曲面和几何着色，第三部分可被配置为在屏幕空间中执行像素着色以生成渲染后的图像。由 GPC 208 生成的中间数据可被存储在缓冲区中，以允许在 GPC 208 之间对该中间数据进行传输，以用于进一步的处理。

[0033] 存储器接口 214 包括 D 个分区单元 215，其每一个都直接地耦合到并行处理存储器 204 的一部分，其中 D ≥ 1。如同所显示的，分区单元 215 的数量通常都等于 DRAM 220 的数量。在其它的实施例中，分区单元 215 的数量可以不等于存储设备的数量。本领域技术人员将会理解 :DRAM 220 可被替换为其它适宜的存储设备并且可以是通常的常规设计。因此

省略了详细的说明。可跨 DRAM 220 来存储渲染对象，例如帧缓冲器或者纹理映射，允许分区单元 215 并行地写入每个渲染对象的一部分，以有效地使用并行处理存储器 204 的可用带宽。

[0034] 任何一个 GPC 208 都可以对将要写入到并行处理存储器 204 中任何一个 DRAM 220 的数据进行处理。交叉开关单元 210 被配置为将每个 GPC208 的输出路由给任一分区单元 215 的输入或者路由至另一个 GPC 208，用于进一步的处理。GPC 208 与存储器接口 214 通过交叉开关单元 210 进行通信，以对不同的外部储存器设备进行读取或写入。在一个实施例中，交叉开关单元 210 具有到存储器接口 214 的连接以与 I/O 单元 205 进行通信，以及到本地并行处理存储器 204 的连接，从而使在不同的 GPC 208 之中的处理内核能够与系统存储器 104 或者其它的对于 PPU 202 来说非本地的存储器进行通信。在图 2 所显示的实施例中，交叉开关单元 210 直接与 I/O 单元 205 相连。交叉开关单元 210 可以使用虚拟信道，以分离 GPC 208 和分区单元 215 之间的业务流。

[0035] 在另一方面，GPC 208 可以被编程用来执行与各式各样的应用程序相关的处理任务，包括但不限于，线性和非线性数据变换、视频和 / 或音频数据的过滤、建模操作（例如，应用物理定律来确定对象位置、速率及其他属性）、图像渲染操作（例如，细分曲面着色、顶点着色、几何着色和 / 或像素着色程序）等等。PPU 202 可以将来自系统存储器 104 和 / 或本地并行处理存储器 204 的数据传送给内部的（片上）存储器，处理该数据，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，其中这样的数据可以由其它的系统组件访问，包括 CPU 102 或者另一个并行处理子系统 112。

[0036] PPU 202 可拥有任何容量 (amount) 的本地并行处理存储器 204，包括不设置本地存储器，并且可以任何组合形式来使用本地存储器和系统存储器。例如，在统一 (unified) 存储器架构 (UMA) 实施例中，PPU 202 可以是图形处理器。在这样的实施例中，将几乎没有或者没有提供专用的图形（并行处理）存储器，并且 PPU 202 将独占地或者是几乎独占地使用系统存储器。在 UMA 实施例中，可将 PPU 202 集成到桥式芯片或者处理器芯片中，或者作为具有高速链路（例如，PCI-EXPRESS）的分立芯片加以提供，该高速链路经由桥式芯片或者其它的通信方式将 PPU 202 连接到系统存储器。

[0037] 如上所述，并行处理子系统 112 中可包括许任意数量的 PPU 202。例如，多个 PPU 202 可以被设置在单个外插卡上，或者多个外插卡可以连接到通信路径 113，或者 PPU 202 中的一个或多个可以被集成到桥式芯片中。多 PPU 系统中的 PPU 202 可以是彼此相同或者不同的。例如，不同的 PPU202 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。在具有多个 PPU 202 时，可以并行地进行操作这些 PPU，以高于采用单个 PPU 202 可能达到的吞吐量来对数据进行处理。可以用各式各样的配置和形式因素来实现包含一个或多个 PPU 202 的系统，包括桌上型电脑、膝上型电脑、或者手持个人电脑、服务器、工作站、游戏控制台、嵌入式系统等等，。

[0038] 处理集群阵列 (Processing Cluster Array) 概述

[0039] 图 3A 是根据本发明一个实施例的、图 2 的一个 PPU 202 中 GPC 208 的框图。每个 GPC 208 可被配置为并行地执行大量的线程，其中术语“线程”指的是对一组特定的输入数据所执行的特定程序的实例。在一些实施例中，采用单指令多数据 (SIMD) 指令发送技术来支持大量线程的并行执行，而不必提供多个独立的指令单元。在其它的实施例中，采用单指

令多线程(SIMT)技术,使用被配置为发送指令到每一个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线程的并行执行。与所有的处理引擎一般都执行相同指令的SIMD执行方式不同,SIMT执行通过给定线程程序允许不同的线程更加容易地跟踪离散型的执行路径。本领域技术人员将会理解到SIMD处理机制相当于SIMT处理机制的功能子集。

[0040] 经由管线管理器305可方便地控制GPC 208的操作,该管线管理器305向流多处理器(SPM)310分布处理任务。管线管理器305也可被配置为,通过为SPM 310所输出的经处理数据指定目的地,来控制工作分布交叉开关330。

[0041] 在一个实施例中,每个GPC 208都包括M个SPM 310,其中M≥1,每个SPM 310都被配置为处理一个或多个线程组。而且,每个SPM 310有利地包括相同的一组可被管线化的功能执行单元(例如,算术逻辑单元,以及加载-存储单元,如在图3C中所示的Exec单元302和LSU 303),允许在前一个指令结束之前发送新的指令,如现有技术中所知的。可提供功能执行单元的任何组合。在一个实施例中,功能单元支持多种运算,包括整数和浮点算法(例如,加法和乘法)、比较运算、布尔运算(AND和、OR或、XOR异或)、位移以及不同代数函数的计算(例如,平面内插、三角、指数、以及对数函数,等等);并且相同功能单元硬件可均衡地用于(beleveraged to)执行不同运算。

[0042] 传送到特定GPC 208的指令序列形成线程,如之前本文所定义的,并且在此将跨SPM 310中并行处理引擎(没有示出)所并发执行的一定数量的线程的集合称为“卷绕包(warp)”或“线程组”。如同在此所使用的,“线程组”指的是针对不同的输入数据,并发执行相同程序的一组线程,其中该组中有一个线程被分配给SPM 310中的不同的处理引擎。线程组可以包括比SPM 310中的处理引擎数量更少的线程,而在这样情况下,在该线程组正在被执行的周期内,一些处理引擎将会处于空闲状态。线程组同样可以包括比SPM 310中的处理引擎数量更多的线程,而在这样情况下,处理将在连续的时钟脉冲周期上发生。因为每个SPM 310可以并发支持多达G个线程组,于是在任何给定的时间在GPC 208中可以允许执行多达G*M个线程组。

[0043] 此外,在SPM 310中可有多个相关的线程组同时处于激活状态(处于执行的不同阶段)。在此将该线程组的集合称为“协作线程阵列”(“CTA”)或者“线程阵列”。特定CTA的大小等于m*k,其中k是在线程组中并发执行线程的数量,并且k一般是SPM 310中并行处理引擎数量的整数倍数,并且m是SPM 310中同时处于激活状态的线程组的数量。CTA的大小通常是由程序员以及CTA可用的硬件资源的诸如存储器或者寄存器的容量来决定的。

[0044] 每个SPM 310都包括L1高速缓存(没有示出)或者使用SPM 310之外的相应的L1高速缓存中用于执行加载和存储操作的空间。而且每个SPM310都可以访问分区单元215内的L2高速缓存,该高速缓存由所有的GPC208所共享,并且可用于在线程之间传送数据。最后,SPM 310还可以访问片外“全局”存储器,其可包括例如并行处理存储器204和/或系统存储器104。将要理解到的是PPU 202之外的任何存储器都可作为全局存储器来使用。此外,可将L1.5高速缓存335包括在GPC 208之内,其被配置为据SPM 310的请求经由存储器接口214来接收并且保持从存储器中所获取的数据,其包括指令、一致性(uniform)数据以及常数数据,并且将所请求的数据提供给SPM 310。在GPC 208中具有多个SPM 310的实施例有利于共享被缓存在L1.5高速缓存335中的共同指令和数据。

[0045] 每个 GPC 208 都可包括存储器管理单元 (MMU)328，其被配置为将虚拟地址映射为物理地址。在其他的实施例中，MMU 328 可以位于存储器接口 214 内。MMU 328 包括一组页表项目 (PTE) 以及可选地包括高速缓存线索引 (cache line index)，该组 PTE 被用于将虚拟地址映射到像素块 (tile) 的物理地址。MMU 328 可包括地址转换后备缓冲区 (TLB) 或者高速缓存，其可以位于多处理器 SPM 310 或者 L1 高速缓存或者 GPC 208 之中。对物理地址进行处理，来分散表面数据访问位置，以允许在分区单元之间交错的高效请求。高速缓存线索引可用来确定对于高速缓存线的请求是否命中或失败。

[0046] 在图形以及计算应用程序中，可对 GPC 208 进行配置使得每个 SPM310 都耦合到纹理单元 315，以执行纹理映射操作，例如确定纹理采样位置、读取纹理数据以及过滤纹理数据。根据需要，从内部的纹理 L1 高速缓存（没有示出）中读取纹理数据，或者在一些实施例中从 SPM 310 中的 L1 高速缓存读取纹理数据，以及从 L2 高速缓存、并行处理存储器 204 或者系统存储器 104 中取得纹理数据。每个 SPM 310 都向工作分布交叉开关 330 输出处理后的任务，以便向另一个 GPC 208 提供该处理后的任务用于进一步的处理，或者经由交叉开关单元 210 将该处理后的任务存入 L2 高速缓存、并行处理存储器 204 或者系统存储器 104 中。PreROP(pre-raster operations，预光栅操作)325 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，并且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。

[0047] 应该予以理解的是，在此所描述的内核架构是说明性的而且是可变化以及修改的。可在 GPC 208 中包括任何数量的处理单元，例如 SPM 310 或者纹理单元 315、PreROP 325。此外，虽然仅显示了一个 GPC 208，但是 PPU 202 可包括任何数量的 GPC 208，其最好在功能上彼此相类似，以便执行行为将不会依赖于是哪个 GPC 208 接收了特定的处理任务。此外，每个 GPC 208 都最好使用单独且不同的处理单元、L1 高速缓存等，独立于其它的 GPC 208 进行操作。

[0048] 图 3B 是根据本发明的一个实施例的、在图 2 中的一个 PPU 202 之内的分区单元 215 的框图。如所显示的，分区单元 215 包括 L2 高速缓存 350、帧缓冲区 (FB)DRAM 接口 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 和 ROP 360 所接收的操作进行加载和存储。由 L2 高速缓存 350 将读取失败以及紧急回写请求输出给 FB DRAM 接口 355 以用于处理。还将脏的更新 (Dirtyupdate) 发送给 FB 355 用于伺机处理。FB 355 与 DRAM 220 直接对接，输出读和写请求并且接收从 DRAM 220 读取的数据。

[0049] 在图形应用程序中，ROP 360 是处理单元，其执行光栅操作并且输出作为处理后图形数据的像素数据用于在图形存储器中存储，其中光栅操作例如模板 (stencil)、z 测试、混合等等。在本发明的一些实施例中，ROP 360 被包括在每个 GPC 208 而非分区单元 215 中，并且将像素读和写请求而不是像素片段数据通过交叉开关单元 210 进行传输。

[0050] 可在显示设备 110 上显示处理后的图形数据，或者对该图形数据进行路由以用于由 CPU 102 或又并行处理子系统 112 中的一个处理实体来进行进一步的处理。每个分区单元 215 都包括 ROP 360 以便分布光栅操作的处理。在一些实施例中，可将 ROP 360 配置为对被写入存储器中 z 数据或者色彩数据进行压缩，以及对从存储器中读取出来的 z 数据或者色彩数据进行解压缩。

[0051] 本领域技术人员将会理解：在图 1、2、3A 和 3B 中描述的架构不以任何方式对本发明的范围进行限制，而且在此教导的技术可在任何被适当设置的处理单元上实现，该处理单元包括但并非限制，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、一个或多个图形或者特殊用途处理单元等等，均不脱离本发明的范围。

[0052] 在本发明的实施例中，使用 PPU 202 或者其它计算系统的处理器来采用线程阵列执行通用计算是可取的。在线程阵列中的每个线程都分配有唯一的线程标识符（“线程 ID”），线程在其执行期间可对该线程标识符进行访问。线程 ID 可被定义为一维的或多维的数值，控制线程处理行为各个方面。例如，线程 ID 可能被用来确定线程将对输入数据集的哪个部分进行处理，和 / 或确定线程将要生成或者写入输出数据集哪个部分。

[0053] 每一线程的指令序列都可以包括至少一个指令，该指令定义了代表性线程和线程阵列中一个或多个其它线程之间的协作行为。例如，每一线程的指令序列可能包括下列指令：在序列中的特定点处挂起代表性线程的操作的执行直到一个或多个其他线程到达该特定点时为止的指令，指示代表性线程将数据存储在一个或多个其他线程有权访问的共享存储器中的指令，，指示代表性线程自动读取和更新存储在共享存储器中的数据的指令，一个或多个其他线程基于其线程 ID 有权访问所述共享存储器，等等。CTA 程序还可以包括计算将从中读取数据的共享存储器中的地址的指令，其中地址为线程 ID 的函数。通过定义适宜的函数并提供同步技术，可以用可预测的方式，由 CTA 的一个线程将数据写入到共享存储器中给定的位置，以及由同一个 CTA 中不同的线程从该位置读取出数据。从而，线程之间任何期望模式的数据共享都可以得到支持，并且在 CTA 中任何的线程都可以与同一 CTA 中的任何其他线程共享数据。如果需要的话，可由 CTA 程序来决定在 CTA 的线程之中数据共享的程度；如此，可以理解到在特定的使用了 CTA 的应用程序中，CTA 的线程彼此之间可能或者可能不实际地共享数据，这取决于 CTA 程序，并且在此对术语“CTA”和“线程阵列”以相同的含义进行使用。

[0054] 图 3C 是根据本发明的一个实施例，图 3A 中 SPM 310 的框图。SPM 310 包括指令 L1 高速缓存 370，其被配置为经由 L1.5 高速缓存 335 从存储器接收指令和常数。warp 调度器和指令单元 312 从指令 L1 高速缓存器 370 接收指令和常数，并且根据该指令和常数对本地寄存器文件 304 和 SPM310 功能单元进行控制。SPM 310 功能单元包括 N 个 exec（执行或者处理）单元 302 和 P 个加载 - 存储单元 (LSU)303。

[0055] SPM 310 提供了具有不同访问级别 (level) 的片上（内部的）数据存储。专用寄存器（没有显示）对于 LSU 303 来说是可读取的而不是可写入的，并且其用于存储定义了每个 CTA 线程“位置”的参数。在一个实施例中，专用寄存器对于每个 CTA 线程（或者对于 SPM 310 中每个 exec 单元 302）都包括一个存储线程 ID 的寄存器；每个线程 ID 寄存器仅可由一个相应的 exec 单元 302 进行访问。专用寄存器还可包括辅助寄存器，其可由所有的 CTA 线程（或者由所有的 LSU 303）进行读取，该辅助寄存器存储了 CTA 标识符、CTA 维度 (dimension)、CTA 所属的栅格 (grid) 的维度以及 CTA 所属的栅格的标识符。在初始化期间响应于经由前端 212 从设备驱动器 103 接收的命令，来对专用寄存器进行写操作，并且该专用寄存器在 CTA 执行期间不会变化。

[0056] 参数存储器（未被显示）存储了运行时 (runtime) 参数（常数），可以由任何的 CTA 线程（或者任何的 LSU 303）对该参数进行读取而非写入。在一个实施例中，设备驱动

器 103 在引导 SPM 310 开始对使用这些参数的 CTA 的执行之前，将参数提供给参数存储器。在任何 CTA（或者 SPM 310 中的任何 exec 单元 302）中的任何 CTA 线程可以通过存储器接口 214 来访问全局存储器。可在 L1 高速缓存 320 中存储全局存储器的一部分。

[0057] 本地寄存器文件 304 被每个 CTA 线程用作临时空间（scratch space）；每个寄存器都被分配用于一个线程的排他性使用，并且任何本地寄存器文件 304 中的数据仅对于该寄存器被分配给的 CTA 线程可访问。可以将本地寄存器文件 304 实现为在物理上或者逻辑上被划分成 P 个通道的寄存器文件，每个通道都具有一定数量的条目（entry）（其中每个条目都可能存储，例如 32 比特的字符）。对 N 个 exec 单元 302 和 P 个加载 - 存储单元 LSU 303 中的每一个都分配一个通道，并且可以使用用于执行同一个程序的不同线程的数据来填充不同通道中的相应条目，以有助于 SIMD 的执行。可以将通道的不同部分分配给 G 个并发线程组中不同的线程组，以便本地寄存器文件 304 中给定的条目仅对特定的线程是可访问的。在一个实施例中，在本地寄存器文件 304 中的某些条目被保留用于存储线程标识符，其实现了一个专用寄存器。

[0058] 共享存储器 306 对于所有的 CTA 线程（在单一的 CTA 之内）都是可访问的；共享存储器 306 中的任何位置对于在同一个 CTA 中（或者对于 SPM 310 中的任何处理引擎）的任何 CTA 线程来说都是可访问的。共享存储器 306 可以被实现为具有互连的共享寄存器文件或者共享片上高速缓存存储器，该互连允许任何处理引擎从共享存储器中的任何位置读取或写入。在其它的实施例中，共享状态空间可以映射到片外（off-chip）存储器的每个 CTA 区域，并且可以在 L1 高速缓存 320 中进行高速缓存。可将参数存储器实现为：在实现了共享存储器 306 的同一个共享寄存器文件或者共享高速缓存存储器中的指定部分，或者 LSU 303 仅能进行只读访问的单独的共享寄存器文件或片上高速缓存存储器。在一个实施例中，也可以将实现了参数存储器的区域用于存储 CTA ID 和栅格 ID，以及 CTA 维度和栅格的维度，由此实现了部分的专用寄存器。SPM 310 中的每个 LSU 303 都耦合到统一地址映射单元 352，该统一地址映射单元将在统一的存储器空间中所指定的为加载和存储指令而提供的地址转换为每个不同存储器空间中的地址。从而，指令可用来访问任何由统一的存储器空间中的地址所指定的本地的、共享的、或者全局的存储器空间。

[0059] 每个 SPM 310 中的 L1 高速缓存 320 都可用于对私有的每个线程的本地数据以及每个应用程序的全局数据进行高速缓存。在一些实施例中，可在 L1 高速缓存 320 中对每个 CTA 的共享数据进行高速缓存。LSU 303 经由存储器和高速缓存的互连 380 耦合到一致性 L1 高速缓存 375、共享存储器 306 以及 L1 高速缓存 320。一致性 L1 高速缓存 375 被配置为经由 L1.5 高速缓存 335 从存储器接收只读数据和常数。

[0060] 图 4 是根据本发明的一个实施例的图形处理管线 400 的示意图，可由图 2 中的一个或多个 PPU 202 配置实现。例如，一个 SPM 310 可被配置为执行顶点处理单元 415 的功能、几何处理单元 425 以及片段处理单元 460 中的一个或多个的功能。还可以由 GPC 208 中的其它处理引擎以及相应的分区单元 215 来执行数据汇编器 410、基元汇编器 420、光栅化器 455 以及光栅操作单元 465 的功能。另一方面，可使用针对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。

[0061] 数据汇编器 410 处理单元为高阶表面、基元等等采集顶点数据，并且向顶点处理单元 415 输出包括了顶点属性的顶点数据。顶点处理单元 415 为可编程执行单元，其

被配置为执行顶点着色程序，从而根据顶点着色程序的指定来光照（lighting）和变换（transforming）顶点数据。例如，可对顶点处理单元 415 进行编程以将顶点数据从对基于对象的坐标表示（对象空间）变换到诸如世界空间或者规格化设备坐标（NDC）空间的替代基础坐标系统。顶点处理单元 415 可以通过数据汇编器 410 读取存储在 L1 高速缓存 320、并行处理存储器 204 或者系统存储器 104 中的数据来供顶点数据处理之用。

[0062] 基元汇编器 420 从顶点处理单元 415 接收顶点属性，根据需要读取所存储的顶点属性，并且构建图形基元用于由几何处理单元 425 进行处理。图形基元包括三角形、线段、点等等。几何处理单元 425 是可编程执行单元，其被配置为执行几何着色程序，依据几何着色程序的指定对从基元汇编器 420 接收的图形基元进行变换。例如，可对几何处理单元 425 进行编程，以将图形基元细分成为一个或多个新的图形基元和计算参数，例如平面方程系数，其用于对新的图形基元进行光栅化。

[0063] 在一些实施例中，几何处理单元 425 也可以添加或者删除几何流中的元素。几何处理单元 425 向视图缩放、剔除（cull）以及裁剪（clip）单元 450 输出指定新的图形基元的参数和顶点。几何处理单元 425 可以读取存储在并行处理存储器 204 或者系统存储器 104 中的数据用于在处理几何数据中使用。视图缩放、剔除以及裁剪单元 450 执行裁剪、剔除以及视图缩放，并且将处理后的图形基元输出到光栅化器 455。

[0064] 光栅化器 455 对新的图形基元进行扫描转换并且将片段和覆盖（coverage）数据输出给片段处理单元 460。此外，可将光栅化器 455 配置为执行 z 剔除及其他基于 z 的最佳化。

[0065] 片段处理单元 460 是可编程执行单元，其被配置为执行片段着色程序，依据片段着色程序的指定，来对从光栅化器 455 接收的片段进行变换。例如，可对片段处理单元 460 编程以执行例如透视校正、纹理映射、着色、混合等诸如此类的操作，以生成输出到光栅操作单元 465 的着色后的片段。片段处理单元 460 可以读取存储在并行处理存储器 204 或者系统存储器 104 中的数据用于在处理片段数据中使用。取决于所编程的采样率，可在像素、样本或者其它粒度上对片段进行着色。

[0066] 光栅操作单元 465 是处理单元，其执行模板（stencil）、z 测试、混合等诸如此类的光栅操作并且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的。可将处理后的图形数据存储在图形存储器，例如并行处理存储器 204 和 / 或系统存储器 104 中，以用于在显示设备 110 上进行显示，或者用于进一步的由 CPU 102 或并行处理子系统 112 进行处理。在本发明的一些实施例中，可将光栅操作单元 465 配置为对写入到存储器中的 z 数据或者色彩数据进行压缩，以及对从存储器中读取出来的 z 数据或者色彩数据进行解压缩。

[0067] 虽然结合图 1、2、3A、3B 和 3C 中的系统对方法步骤进行了描述，但是本领域技术人员将理解到：任何配置为以任意顺序执行该方法步骤的系统都在本发明的范围之内。

[0068] 用于管理并行高速缓存层级的指令

[0069] 图 5 是根据本发明一个实施例的示意图，示出了在并行线程处理器中的并行高速缓存层级。如所显示的，PPU 502 包括一个或多个 SPM 510。PPU 502 耦合到 PPU 存储器 526，该 PPU 存储器 526 可以包括 DRAM。PPU502 也耦合到桥 506。桥 506 耦合到 CPU 504 和系统存储器 508。在一种实现方式中，PPU 502 经由 PCI-Express 链路和桥 506 耦合到 CPU 504 和系统存储器 508。

[0070] 如图 5 所示，每个 SPM 510 均包括指令 L1 高速缓存 512、常数 L1 高速缓存 514、数据 L1 高速缓存 516 和 / 或一致性 L1 高速缓存 518。PPU 也包括耦合到每个 SPM 510 的 L1.5 高速缓存 520。L2 高速缓存互连耦合到每个 SPM 510、L1.5 高速缓存 520 和 L2 高速缓存 524 中。L2 高速缓存 524 耦合到 PPU 存储器 526。

[0071] 在一个实施例中，每个 PPU 502 都相当于图 2 中所显示的 PPU 202，并且每个 SPM 510 都相当于在图 3C 中所显示的 SPM 310。例如，如同在图 3C 中所显示的，指令 L1 高速缓存 512 相当于指令 L1 高速缓存 370，数据 L1 高速缓存 516 相当于 L1 高速缓存 320，并且一致性 L1 高速缓存 518 相当于一致性 L1 高速缓存 375。L1.5 高速缓存 520 可以相当于在图 3A 中所显示的 L1.5 高速缓存 335。L2 高速缓存 524 可以相当于如在图 3B 中所显示的 L2 高速缓存 350。

[0072] 在图 5 所举例说明的示意图中，仅仅显示了并行线程处理器架构中并行高速缓存层级的一种实现方式，具有被称为流多处理器 (SPM) 510 的数量可扩展的线程处理器。在一个实施例中，warp 调度器和指令单元 312 也被包括在 SPM 510 中，从并行线程向并行执行单元 302 和并行加载 - 存储单元 303 提供指令，如同在图 3C 中所描述的。

[0073] 在图 5 所示的实施例中，每个 SPM 510 都包括多个不同的 L1 高速缓存 :L1 指令高速缓存 512、L1 常数高速缓存 514、L1 数据高速缓存 516 以及一致性数据 L1 高速缓存 518。SPM 510 和 L1 高速缓存经由高速缓存互连网络 522 来共享统一的 L2 高速缓存 524。在一些实施例中，在 L1 高速缓存和 L2 高速缓存之间提供了辅助的高速缓存层，即，L1.5 高速缓存 520。

[0074] L2 高速缓存 524 访问 PPU DRAM 存储器 526，经由 PCIe 接口来访问系统存储器 508，以及可选地，经由 PCIe 接口访问辅助的对等设备存储器。对等 (peer) 设备存储器的实例是与附属于同一 PCIe 网络的另一个 PPU 的 DRAM 存储器。

[0075] SPM 加载 - 存储单元 (LSU) 303 (在图 3C 中所显示的) 执行存储器访问指令，包括如下所列的加载、存储和高速缓存控制指令 :

[0076] ld{.cop}.sz rd, [ra+offset] ;// 从存储器加载

[0077] ldu.sz rd, [ra+offset] ;// 经由一致性高速缓存加载

[0078] st{.cop}.sz [ra+offset], rb ;// 存储到存储器中

[0079] cctl.ca che.op {rd,}[ra+offset] ;// 高速缓存控制操作

[0080] 如同在此所使用的，术语“加载 (load)”描述了从存储器读取并返回值的指令，而术语“存储 (store)”描述了将值写到存储器中的指令。一些指令，例如原子和锁操作，会修改存储器并返回值，并且应该被看作兼备加载和存储的语义，因此，同时遵循加载和存储的规则。

[0081] 下面对加载指令和存储指令“高速缓存操作”(.cop) 加以描述。下面还对高速缓存控制指令 cctl 加以描述。

[0082] 用于加载和存储指令的高速缓存操作

[0083] 加载和存储指令在由地址操作数所指定的有效地址上对存储器进行读取或者写入。.sz 后缀指定了将要读取或者写入存储器中的字节大小，并且 SPM 指令集架构 (ISA) 可以支持 1、2、4-、8 和 16 个字节的大小用于加载 / 存储指令。有效存储器地址是寄存器 ra 加上立即数 offset (偏移) 的字节总和。

[0084] ld{.cop}.sz rd,[ra+offset];// 从存储器加载 rd

[0085] st{.cop}.sz [ra+offset],rb;// 将 rb 存储到存储器中

[0086] 在一些实施例中，SPM 510 使用 32 比特的地址，以及使用指定有后缀 .e 的 64 比特的扩充地址，实现了两个版本的存储器存取指令（即，指令 ld.e 和 st.e）。

[0087] 加载指令高速缓存操作 ld.cop

[0088] 加载指令具有可选的由 .cop 所指定的高速缓存操作，编译程序和 / 或程序员可以用来优化对全局存储器空间和对本地每个线程的专用存储器空间进行访问的高速缓存使用。依赖于由系统软件和 PPU 存储器管理单元

[0089] (MMU) 页表 (page table) 所提供的虚拟地址到物理地址的映射，全局的和本地的存储器访问可以映射到 PPU(DRAM) 存储器 526、系统存储器 508 和 PCIe 设备存储器。在一种实现方案中，对共享存储器 RAM 的访问忽略了高速缓存操作，但是对共享存储器空间进行高速缓存的实现方案则可以使用高速缓存操作。

[0090] 关于加载指令 ld{.cop} 和 ld.e{.cop} 的可选高速缓存操作是：

[0091] .ca 在所有级别 (level) 上进行高速缓存，有可能被再次访问（缺省值）

[0092] .cg 在全局级别上进行高速缓存（在 L2 和之下的级别，非 L1 上进行高速缓存）

[0093] .cs 高速缓存流，有可能被访问一次，绕过高速缓存或者提前回收 (evict)

[0094] .lu 最后使用：如果地址是每个线程的本地地址，并且高速缓存线被完全地覆盖（由 warp 的线程来访问在高速缓存线中的所有数据），则加载然后使该线无效并且取消任何待定的脏回写，否则加载并且将高速缓存线标记为回收优先。.cop 的编码与 .cs 的相同。

[0095] .cv 如果地址是在系统存储器中，则以易失 (volatile) 的方式进行高速缓存；认为高速缓存的系统存储器线失去时效时，再次获取。

[0096] 缺省的 ld 指令高速缓存操作是 ld.ca，其在所有级别 (L1 和 L2) 上用正常的回收策略来分配高速缓存线。在一个实施例中，当应用程序希望多次访问同一个高速缓存线，并且希望其访问命中在 L1 高速缓存的工作集中的时候，应用程序可以使用这个指令。

[0097] 全局数据在 L2 高速缓存级别中是连贯的 (coherent)，但是在一种实现方案中，在每个 SPM 中的多个 L1 高速缓存对于全局数据来说并非彼此连贯。如果一个线程经由一个 L1 高速缓存存储到全局存储器，并且在不同的 SPM 中的第二个线程使用 ld.ca 经由第二个 L1 高速缓存来加载那个地址，那么第二个线程可能得到的是失去时效的 L1 高速缓存数据，而不是由第一个线程所存储的数据。由此，驱动器将会使全局 L1 高速缓存线在并行线程的从属栅格之间无效。如同下面所要详细描述的，程序还可以使用高速缓存控制指令 cctl 来使 L1 高速缓存线无效。随后由第二个栅格程序正确地获取由第一个栅格程序所进行的存储，该第二个栅格程序发布被高速缓存在 L1 高速缓存中的缺省的 ld.ca 加载。这个指令支持可选的实现方案，其提供了在多个 L1 高速缓存中的高速缓存连贯性。可供选择地，程序可以利用如下所述的 ld.cg 加载高速缓存全局操作，来绕过 L1 高速缓存级别，以避免读取失去时效的 L1 数据。

[0098] 在一个实施例中，指令 ld.cg 仅用于全局的高速缓存加载，绕过 L1 高速缓存并且仅在全局的 (L2 高速缓存）级别上进行高速缓存。当应用程序希望读取一次地址时，并且在相对较小的 L1 高速缓存中减少工作集的扰动的时候，其可以使用这个指令。这个指令使不同 SPM 中的线程之间能够通信。

[0099] 该 ld.cs 加载高速缓存流操作，使用回收 - 优先策略在 L1 和 L2 中分配全局线，以限制由临时性流数据导致的高速缓存污染，该临时性流数据可被访问一次或两次。在另一个实施例中，流数据可以经由小的流高速缓存或者与每个高速缓存相邻的先入先出 (FIFO) 来绕过 L1 和 L2 高速缓存，以便流数据不会干扰 L1 或者 L2 的工作集。当将 ld.cs 用于本地窗口地址时，其执行如下所述的 ld.lu 操作。

[0100] 该 ld.lu 加载最后使用操作，当被应用于本地每线程私有的地址的时候，如果本地 L1 高速缓存线被完全覆盖（该高速缓存线中的所有数据被 warp 的线程读取），则该操作在加载之后将该线无效（即，如果自在先的存储起该线是脏的，则丢弃和取消该线的任何待定的脏的回写）。当要恢复溢出的寄存器以及弹出功能堆栈帧时，编译器和 / 或程序员可以使用 ld.lu 以避免对不会被再次使用的线进行不必要的回写。ld.lu 指令具有与 ld.cs 相同的高速缓存操作 .cop 编码，并且 ld.lu 指令执行在全局地址上的加载高速缓存流操作。

[0101] 该 ld.cv 加载高速缓存易失操作被应用于全局的系统存储器地址，其使匹配的 L2 线无效（即，丢弃）并且对于每个新的加载进行线的重新获取，以允许线程程序轮询由 CPU 写入的系统存储器的位置。

[0102] 如表 1 所示，应用于 PPU DRAM 地址的 ld.cv 与 ld.cs 相同，回收 - 优先。

[0103]

LD.cop [全局地址]				LD.cop [本地地址]		
.cop	L1	L2 DRAM	L2 SysMem	*.cop*	L1	L2
.ca*	回收-正常	回收-正常	回收-正常	.ca*	回收-正常	回收-正常
.cg	不-缓存 [1]	回收-正常	回收-正常	.cg	回收-优先	回收-正常
.cs	回收-优先	回收-优先	回收-优先	.lu	最后使用 [2]	回收-优先
.cv	不-缓存[1]	回收-优先	获取易失的[3]	.cv	回收-优先	回收-优先

[0104] 表 1

[0105] * 指示缺省值。

[0106] [1] 在 ld.cg 或者 ld.cv 之前，L1 使匹配的线无效。在这个实现方案中，L1 不是连贯的 - 其没有对全局的写进行监听，所以匹配的 L1 线可能会失去时效。在 ld.cg 或者 ld.cv 之后，在 L1 中没有留下记录。

[0107] [2] 只有在线被完全覆盖（其所有数据均由 warp 的线程读取）的时候，L1 将会返回本地的每线程的数据，然后使线无效并且取消待定的脏的回写取消；否则，其将返回该线并且以回收 - 优先的方式加以保留。

[0108] [3] 应用于系统存储器的加载易失高速缓存 ld.cv 使匹配的 L2 线无效并且对于每个新的加载进行线的重新获取，以允许线程程序轮询由 CPU 写入的 SysMem 位置。L2 可以将爆发的加载合并 (coalesce) 到同一个 SysMem 地址。应用于帧缓冲 DRAM 地址的 ld.cv 与

ld.cs 相同，回收 - 优先。存储指令高速缓存操作 st.cop

[0109] 与上述加载指令类似，存储指令具有可选的由 .cop 所指定的高速缓存操作，编译器和程序员可以用其来优化对全局存储器空间和本地每个线程的专用存储器空间进行访问的高速缓存使用。

[0110] 关于存储指令 st{.cop} 和 st.e.{.cop} 的可选高速缓存操作是：

[0111] .wb 回写全部连贯的级别（缺省值）

[0112] .cg 在全局级别进行高速缓存（在 L2 和以下的级别，而非 L1 上进行高速缓存）

[0113] .cg 高速缓存流，有可能被写入一次（绕过高速缓存或者提前回收

[0114] .wt 高速缓存透写 (write-through)（对于在系统存储器中的地址）

[0115] 在一个实施例中，当将共享存储器实现为 RAM 的时候，忽略对共享存储器的高速缓存操作。与那些用于全局存储器的高速缓存操作相比，对于本地存储器的高速缓存操作可以具有不同的含义。

[0116] 缺省的 st 通用存储高速缓存操作是存储回写 (write-back)st.wb，其利用正常的回收策略回写连贯高速缓存级别的高速缓存线。，利用回写将被存储到本地每线程的存储器中的数据在 L1 高速缓存和 L2 高速缓存中进行高速缓存。然而，在一个实施例中，由于对于全局数据来说多个 L1 高速缓存是不连贯的，因此不对 L1 中的全局存储数据进行高速缓存。全局存储绕过了 L1 高速缓存并且将任何匹配的 L1 高速缓存线丢弃，而不考虑 .cop 高速缓存操作。其它的实施例可提供全局连贯的 L1 高速缓存，并且 st.wb 可以从 L1 高速缓存中回写脏的全局储存数据。

[0117] 在表 2 所示的一个实施例中，如果一个线程绕过其 L1 高速缓存器存储到全局存储器，并且在不同的 SPM 中的第二个线程随后利用 ld.ca 经由不同的 L1 高速缓存来加载那个地址，那么第二个线程可能命中失去时效的 L1 高速缓存数据，而不是从 L2 或者存储器得到由第一个线程所存储的数据。相应地，驱动器必须使线程阵列的附属栅格之间的全局 L1 高速缓存线无效。然后由第一格栅程序进行的存储会在 L1 高速缓存中得到正确地忽略，并被发布了缺省 ld.ca 加载的第二栅格程序所获取。

[0118] 该高速缓存操作 st.cg 全局高速缓存，仅全局性地用于对全局储存数据进行高速缓存，绕过 L1 高速缓存，以及仅在 L2 高速缓存中进行高速缓存。在表 2 所显示的一种实现方案中，st.cg 高速缓存全局策略还可用于针对全局数据的 st.wb 指令，但是针对本地存储器的 st.cg 使用 L1 高速缓存，并且将本地 L1 线标记为回收 - 优先。

[0119] 该 st.cs 存储高速缓存流操作，利用回收 - 优先策略在 L2 高速缓存（和 L1 高速缓存，如果是本地的话）中分配高速缓存线，以限制由流输出数据导致的高速缓存污染；全局流数据绕过 L1。自程序发布一次流写入后，st.cs 的另一个实现方案将使流数据经由小的流高速缓存或者与每个高速缓存相邻接的先入先出 (FIFO) 来绕过 L1 和 L2 高速缓存，以使流数据不会对 L1 高速缓存或者 L2 高速缓存的工作集形成干扰。

[0120] 该 st.wt 存储透写操作，被应用于全局系统存储器地址，其将对 L2 高速缓存进行透写操作，以允许 CPU 程序利用 st.wt 来轮询由 PPU 所写入的系统存储器位置。在一个实现方案中，不在系统存储器中的地址使用正常的 L2 回写。

[0121] 存储指令高速缓存操作的一个实施例采用在表 2 中所示的高速缓存操作策略。

[0122]

ST.cop [全局地址]				ST.cop [本地地址]		
.cop	L1	L2 DRAM	L2 SysMem	.cop	L1	L2
.wb*	不-缓存[1]	回收-正常	回收-正常	.WB*	回收-正常	回收-正常
.cg	不-缓存[1]	回收-正常	回收-正常	.CG	回收-优先	回收-正常
.cs	不-缓存[1]	回收-优先	回收-优先	.CS	回收-优先	回收-优先
.wt	不-缓存[1]	回收-优先	透写 [2]	.WT	回收-优先	回收-优先

[0123] 表 2

[0124] * 缺省指示

[0125] [1] 在这个实施例中，全局数据存储绕过 L1。L1 没有对全局存储数据进行高速缓存；其是对本地的每线程数据进行高速缓存。因为不能弄脏全局 L1 线，所以在进行全局的 ST 之前，L1 丢弃了所匹配的全局线。L1 不是全局性连贯的 - 其没有对全局的存储进行监听，所以匹配的 L1 线可能会失去时效。在对全局进行 ST 之后在 L1 中没有留下记录。

[0126] [2] 被应用于全局系统存储器的存储透写 (st.wt)，透写 L2 高速缓存线到系统存储器，以利用 st.wt 允许 CPU 对由 GPU 写入的 SysMem 位置进行轮询。L2 不会将爆发的透写存储合并到同一个 SysMem 地址；其透写每一个到 SysMem。应用于 PPU 存储器 (204) 帧缓冲 DRAM 地址的 st.wt 与 st.cs 相同，流回收 - 优先回写。

[0127] 高速缓存控制指令 cctl.cache.op

[0128] 高速缓存控制指令 cctl.cache.cop 和 cctll.cop 对包括了特定统一的或者本地的每线程地址的高速缓存线进行控制或者查询。

[0129] CCTL{.E}{.cache}.cop [Ra+ImmS32] ;// 高速缓存控制，统一地址

[0130] CCTLL.cop{.S} [Ra+ImmS24] ;// 高速缓存控制，本地地址

[0131] cctl.cache 高速缓存层级的说明符如下：

[0132] .d 数据高速缓存层级 L1，L2（如果省略则为缺省值）

[0133] .u 一致性高速缓存层级 L1，L1.5，L2

[0134] .c 常数高速缓存层级 L1，L1.5，L2

[0135] .i 指令高速缓存层级 L1，L1.5，L2

[0136] .t 纹理高速缓存层级 L1，L2

[0137] cctl.cop 高速缓存操作的说明符如下：

[0138] .qry1 将 L1 的线状态（有效，脏的）写入 Rd

[0139] .pf1 将线预取到 L1 高速缓存中

[0140] .pf1.5 将线预取到 L1.5 高速缓存中

[0141] .pf2 将线预取到 L2 高速缓存中

[0142] .wb 回写脏的高速缓存线（刷新 flush 存储器）

[0143] .iv 使高速缓存线无效（如果是脏的，则首先进行回写）

[0144] .ivall 使所有的高速缓存线无效（如果是脏的，则进行回写）

[0145] .rs 对线进行复位，标记无效而不预先进行无效

[0146] 将字节地址计算为寄存器 Ra 加上带符号的 (signed) 立即数偏移 ImmS32（或者

ImmS24)的总和，于是其由零比特扩展为40比特。如果指定了.e扩充，则将统一字节地址计算为64比特值(R[a+1],R[a])加上带符号扩展后的(sign-extended)立即数偏移ImmS32的总和。在由cctl.cache所指定的高速缓存地址空间内解释有效地址。

[0147] 可以使用cctl对若干高速缓存地址空间进行控制或者查询：数据地址、一致性全局地址、常数地址、指令地址以及在一些实施例中的纹理地址。对于.d以及.u高速缓存器层级使用统一的通用线程字节地址。对于.c高速缓存器层级[bank][offset]使用常数库(bank)以及在库内的偏移量。对于.i高速缓存器层级使用指令字节地址。对于.t高速缓存器层级使用纹理或者全局地址。

[0148] cctl指令对于包括了已提供地址的高速缓存线进行控制或者查询。cctl.qry1写入目的寄存器rd；其它的cctl.cop操作没有对rd进行写入，并且必须忽略rd。将忽略的rd组合为rz(无效的目的地)。

[0149] 本地存储器cctll没有使用.cache说明符；它的地址在本地数据空间范围之内。cctll对在本地空间之内有效的每个线程的[Ra+ImmS24]的本地地址进行评价，并且在所选择的本地数据高速缓存线上执行操作.cop。在共享存储器可高速缓存的情况下，则保留cctls指令名称。一些实施例将cctlt名称用于纹理高速缓存控制。

[0150] cctl.d.ivall不会取得地址；其将L1数据高速缓存中的所有全局线无效。类似地，cctl.u.ivall不会取得地址；其将一致性L1高速缓存和L1.5高速缓存中的所有的全局线无效。在回写任何脏的线之后，本地存储器cctll.ivall将L1数据高速缓存中的所有本地线无效。

[0151] cctl{l}.qry1将经编址的高速缓存线状态以这一方式写入rd：

比特	说明

[0152]

0	L1线有效：	0=无效，1=有效
1	L1线脏：	0=无效或干净，1=脏
31:2	0	

[0153] cctl(l).qry1的替代实施例将查询结果写入一个或多个判定寄存器(predicate register)中。

[0154] 指令和常数高速缓存是只读的。对只读存储器的回写被忽略。预取操作悄然忽略无效地址或者具有MMU转化错误的地址。

[0155] cctl.d和cctl.u操作应用于统一的通用地址空间。如在ld中所描述的那样，对位于本地或者共享存储器窗口中的地址进行变换；任何与寻址有关的错误都不会导致被报告的错误。

[0156] 高速缓存控制预取指令不对错误进行报告。这允许程序使用可能是无效的地址来请求高速缓存线的预取，这种情况可发生在编译器在程序中提早移动预取以将其提早启动的时候。

[0157] 图6是根据本发明的一个实施例，用于处理加载存储器访问指令的方法步骤的流

程图。本领域技术人员将理解，尽管结合图 1-5 的系统对方法 600 进行描述，任何被配置为以任何顺序执行该方法步骤的系统，都在本发明实施例的范围之内。在图 6 中的流程图描述了根据对全局存储器地址执行“加载”存储器访问指令，L1 和 L2 高速缓存的高速缓存操作。

[0158] 如所显示，方法 600 开始于步骤 602，其中包含在 SPM 中的加载 - 存储单元 (LSU) 接收“加载”存储器访问指令。在步骤 604，LSU 确定存储器访问指令是否包括高速缓存操作 (.cop) 修饰符。如果 LSU 确定存储器存取指令并不包括 .cop 修饰符 (即，缺省的存储器访问指令)，则方法 600 进行到步骤 606。在步骤 606，LSU 利用正常的回收策略将与存储器访问指令有关的数据在 L1 和 L2 高速缓存中进行高速缓存。

[0159] 在步骤 604，如果 LSU 确定存储器访问指令确实包括 .cop 修饰符，那么方法 600 进行到步骤 608。在步骤 608，LSU 确定 .cop 是否等于“.ca”(即，全部进行高速缓存)。如果 LSU 确定 .cop 等于“.ca”，则方法 600 进行到上面描述的步骤 606。如果 LSU 确定 .cop 不等于“.ca”，则方法 600 进行到步骤 610。

[0160] 在步骤 610，LSU 确定 .cop 是否等于“.cg”(即，全局高速缓存)。如果 LSU 确定 .cop 等于“.cg”，则方法 600 进行到步骤 612。在步骤 612，LSU 利用正常的回收策略将与存储器访问指令有关的数据仅在 L2 高速缓存中进行高速缓存，而并不在 L1 高速缓存中进行高速缓存。如同上面描述的，在 ld.cg 或者 ld.cv 指令之前 L1 会将匹配的线无效。在一个实现方案中，L1 不是连贯的 - 其没有对全局的写进行监听，所以匹配的 L1 线可能会失去时效。在 ld.cg 或者 ld.cv 之后，在 L1 中不会留下记录。在步骤 610，如果 LSU 确定 .cop 不等于“.cg”，则方法 600 进行到步骤 611。

[0161] 在步骤 611，LSU 确定 .cop 是否等于“.lu”(即，“最后使用”)，以及地址是否是本地每线程的地址和线是否被完全地覆盖 (在 warp 中的线程加载了本地高速缓存线中的全部数据)。如果 LSU 确定 .cop 等于“.lu”并且该地址是被完全覆盖的本地地址，则方法 600 进行到步骤 613。在步骤 613，在加载了“最后使用”数据之后，LSU 将高速缓存线无效，并且取消任何待定的脏的高速缓存线的回写，然后方法 600 结束。如果 LSU 确定 .cop 不等于“.lu”和 / 或该地址不是被完全覆盖的本地地址，则方法 600 进行到步骤 614。

[0162] 在步骤 614，LSU 确定 .cop 是否等于“.cs”(即，高速缓存流)。如果 LSU 确定 .cop 等于“.cs”，则方法 600 进行到步骤 616。在步骤 616，LSU 利用“回收 - 优先”的回收策略将与存储器访问指令有关的数据在 L1 和 L2 高速缓存中进行高速缓存。在一个实现方案中，将仅对这个数据读取一次，使用一次，并且不会再次使用，并且可在流高速缓存或者 FIFO 中绕过 L1 和 L2 高速缓存。如果，在步骤 614，LSU 确定 .cop 不等于“.cs”，则方法 600 进行到步骤 618。

[0163] 在步骤 618，LSU 确定 .cop 是否等于“.cv”(即，易失高速缓存)。如果 LSU 确定 .cop 等于“.cv”，则方法 600 进行到步骤 620。在步骤 620，LSU 利用回收 - 优先的回收策略将与存储器访问指令有关的数据在 L2 高速缓存中进行高速缓存，而并不在 L1 高速缓存中进行高速缓存。如果该地址是在系统存储器中，则 L2 忽略任何先前被高速缓存的 L2 数据，并且总是从系统存储器中获取线，以实现“易失高速缓存”的策略。如果，在步骤 618，LSU 确定 .cop 不等于“.cv”，则方法 600 进行到步骤 622。在 622，LSU 确定 .cop 是无效的并且返回错误。

[0164] 图 7 是根据在表 2 中所显示的本发明的一个实施例，用于对存储存储器访问指令进行处理的方法步骤的流程图。本领域技术人员将理解，尽管结合图 1-5 的系统对方法 700 进行了描述，任何被配置为以任何顺序执行该方法步骤的系统，都在本发明实施例的范围之内。

[0165] 在图 7 中的流程图根据到全局存储器地址的“存储”存储器访问指令的执行，描述了 L1 和 L2 高速缓存的高速缓存操作。如同所显示的，方法 700 开始于步骤 702，其中包含在 SPM 中的加载 - 存储单元 (LSU) 接收“存储”存储器访问指令。在步骤 704，LSU 确定存储器访问指令是否包括高速缓存操作 (.cop) 修饰符。如果 LSU 确定存储器访问指令不包括 .cop 修饰符 (即，缺省的存储器访问指令)，则方法 700 进行到步骤 706。在步骤 706，实施用于存储的缺省以及回写策略，LSU 使用正常的回收策略，将在被连贯高速缓存的地址上的与存储器访问指令有关的数据，以回写策略在 L1 和 L2 高速缓存中进行高速缓存。在表 1 和表 2 所显示的实施例中，对于本地每线程地址的存储是连贯的并且因此在 L1 和 L2 中用回写策略对其进行高速缓存，但是对于全局地址的存储绕过了 L1 并且无效 L1 中匹配的高速缓存线，以及仅在 L2 中采用回写进行高速缓存。

[0166] 在步骤 704，如果 LSU 确定存储器访问指令确实包括 .cop 修饰符，那么方法 700 进行到步骤 708。在步骤 708，LSU 确定 .cop 是否等于“.wb”(即，回写)。如果 LSU 确定 .cop 等于“.wb”，则方法 700 进行到上面描述的步骤 706。如果 LSU 确定 .cop 不等于“.wb”，则方法 700 进行到步骤 710。

[0167] 在步骤 710，LSU 确定 .cop 是否等于“.cg”(即，全局高速缓存)。如果 LSU 确定 .cop 等于“.cg”，则方法 700 进行到步骤 712。在步骤 712，LSU 使用正常的回收策略，将与用于每线程本地地址的存储器访问指令相关的数据，在 L1 和 L2 高速缓存中进行高速缓存，并且将与用于全局地址的存储器访问指令相关的数据绕过 L1 且不在 L1 高速缓存中进行高速缓存。在步骤 710，如果 LSU 确定 .cop 不等于“.cg”，则方法 700 进行到步骤 714。

[0168] 在步骤 714，LSU 确定 .cop 是否等于“.cs”(即，高速缓存流)。如果 LSU 确定 .cop 等于“.cs”，则方法 700 进行到步骤 716。在步骤 716，LSU 使用“回收 - 优先”的回收策略，将与用于每线程本地地址的存储器访问指令相关的数据，在 L1 和 L2 高速缓存中进行高速缓存，并且将与用于全局地址的存储器访问指令相关的数据绕过 L1 且不在 L1 高速缓存中进行高速缓存。使用流高速缓存策略进行存储表示这一数据将仅被存储一次。在步骤 714，如果 LSU 确定 .cop 不等于“.cs”，则方法 700 进行到步骤 718。

[0169] 在步骤 718，LSU 确定 .cop 是否等于“.wt”(即，透写)。如果 LSU 确定 .cop 等于“.wt”，则方法 700 进行到步骤 720。在步骤 720，LSU 利用透写策略以及回收 - 优先的回收策略，将在全局系统存储器地址上的与存储器访问指令有关的数据在 L2 高速缓存中进行高速缓存，而不在 L1 高速缓存中进行高速缓存。在步骤 718，如果 LSU 确定 .cop 不等于“.wt”，则方法 700 进行到步骤 722。在步骤 722，LSU 确定 .cop 是无效的并且返回错误。

[0170] 总而言之，本发明的实施例提供了指令，该指令使并行的多线程应用软件能够协调并发线程，以有效地使用工作集容量有限的高速缓存。本发明的实施例提供了针对加载 / 存储存储器访问指令的显式高速缓存行为修饰符。该修饰符使得程序员和 / 或编译器能够为下列行为指定高速缓存优化 :针对特定加载 / 存储存储器指令的易失和非高速缓存行为、工作集行为和流行为。

[0171] 有利地，本发明的实施例允许程序员和 / 或编译器指定高速缓存策略以及指定在哪个高速缓存级别对数据进行高速缓存。这使得程序的执行以及数据的访问更加有效率。

[0172] 本发明的一个实施例可以实现为与计算机系统一起使用的程序产品。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，并且能够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设备，诸如可被 CD-ROM 驱动器读出的 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）。

[0173] 以上已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，在不脱离如所附权利要求所述的本发明的较宽精神和范围的情况下，可对此做出各种修改和变化。相应地，前面的描述和附图应被视为是示例性而非限制性的。

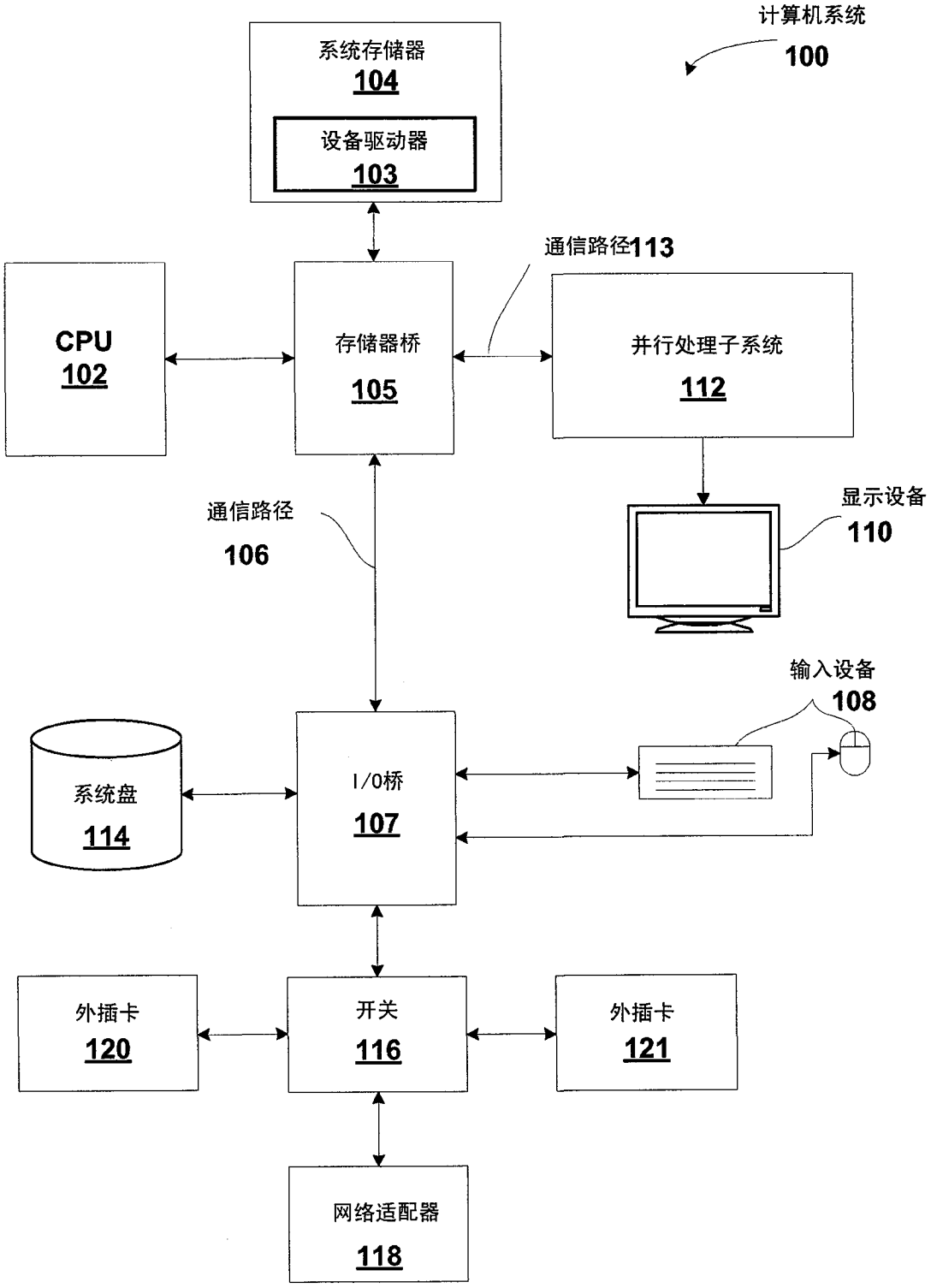

计算机系统
100
系统存储器
104
设备驱动器
103
通信路径113
CPU
102
存储器桥
105
并行处理子系统
112
显示设备
110
通信路径
106
输入设备
108
系统盘
114
I/O桥
107
外插卡
120
开关
116
外插卡
121
网络适配器
118

图 1

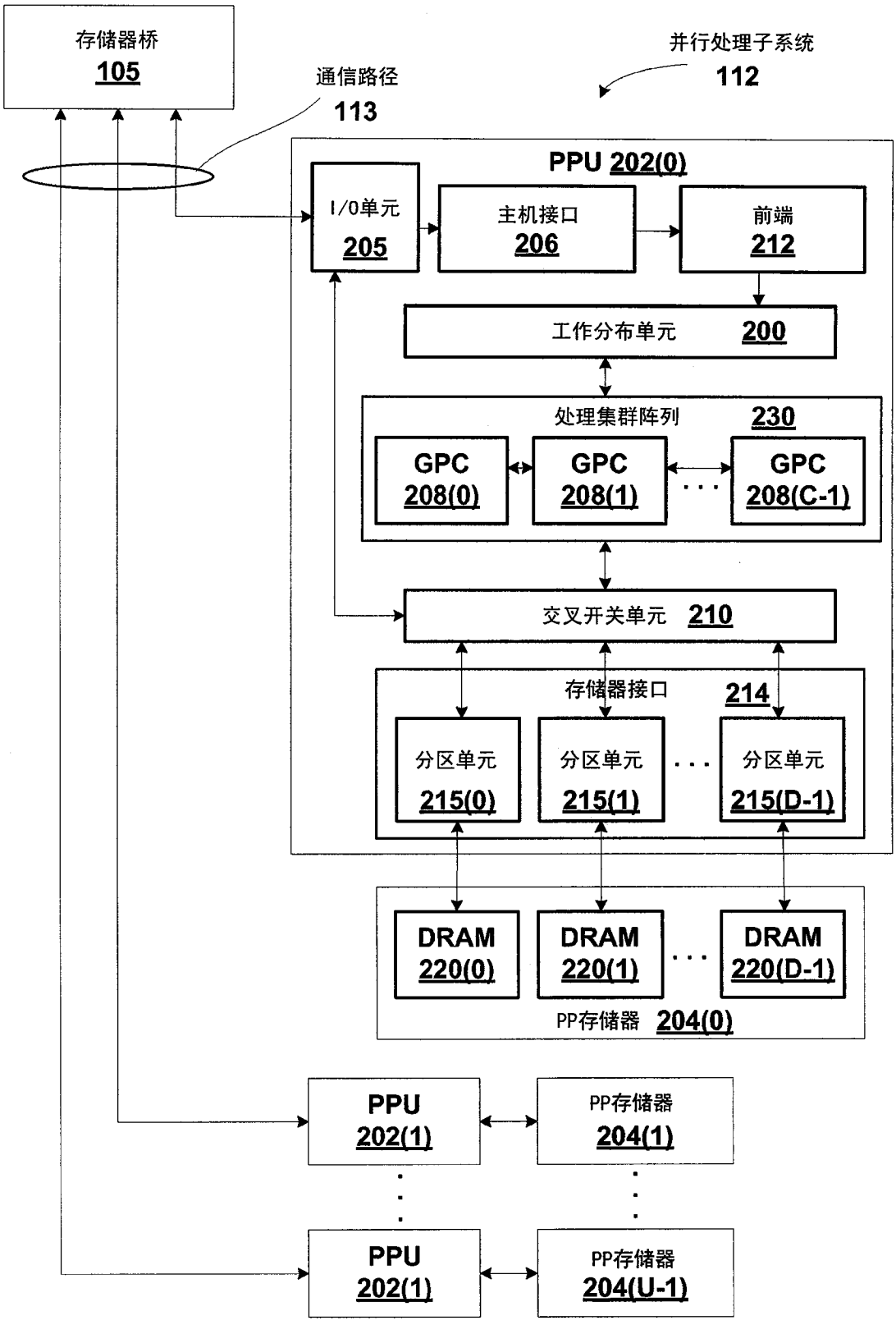

存储器桥
105
通信路径
113
并行处理子系统
112
PPU 202(0)
I/O单元
205
主机接口
206
前端
212
工作分布单元
200
处理集群阵列
230
GPC
208(0)
GPC
208(1)
GPC
208(C-1)
交叉开关单元
210
存储器接口
214
分区单元
215(0)
分区单元
215(1)
分区单元
215(D-1)
DRAM
220(0)
DRAM
220(1)
DRAM
220(D-1)
PP存储器
204(0)
PPU
202(1)
PP存储器
204(1)
PPU
202(1)
PP存储器
204(U-1)

图 2

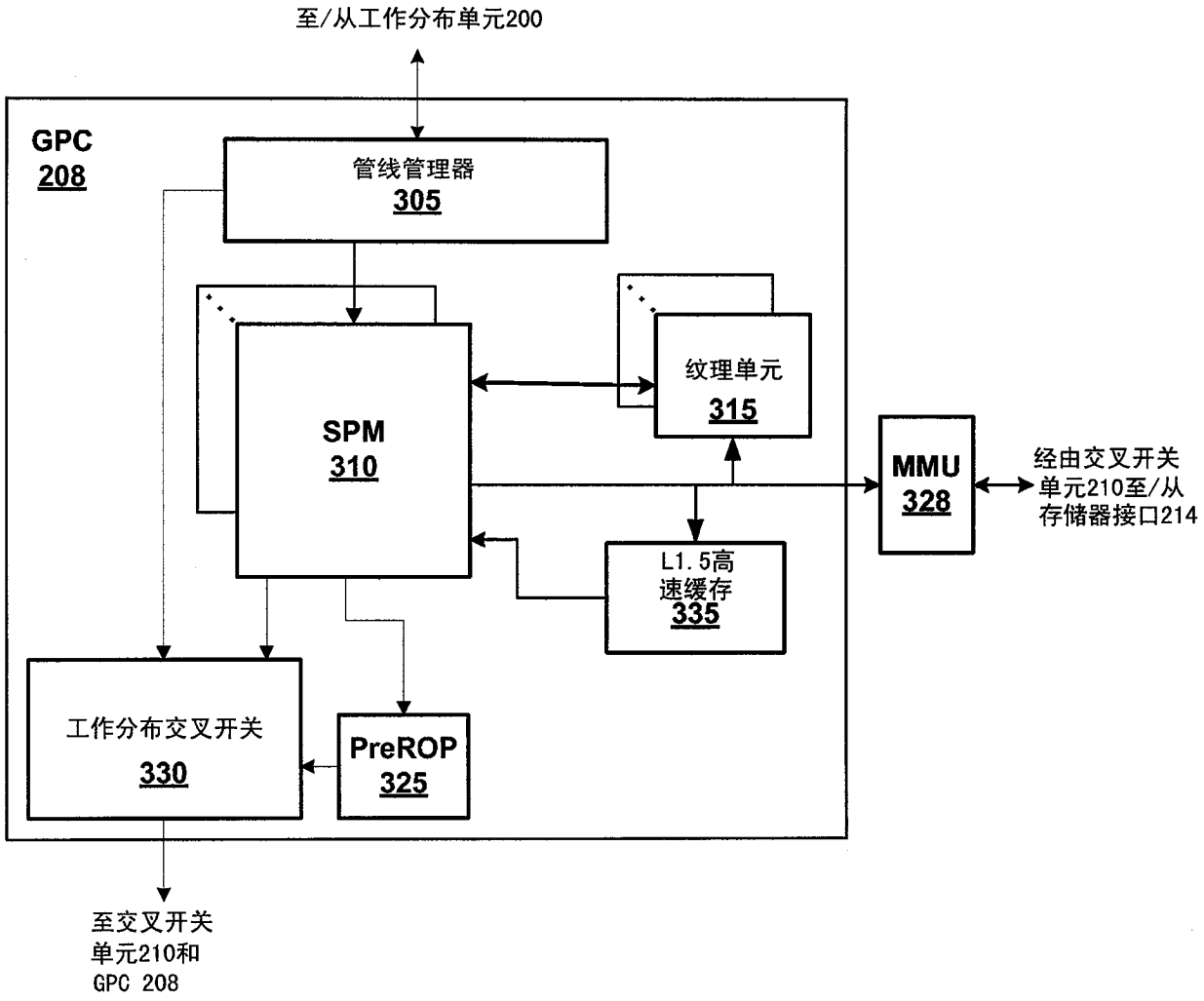

至/从工作分布单元200
GPC
208
管线管理器
305
SPM
310
纹理单元
315
MMU
328
经由交叉开关
单元210至/从
存储器接口214
L1.5高
速缓存
335
工作分布交叉开关
330
PreROP
325
至交叉开关
单元210和
GPC 208

图 3A

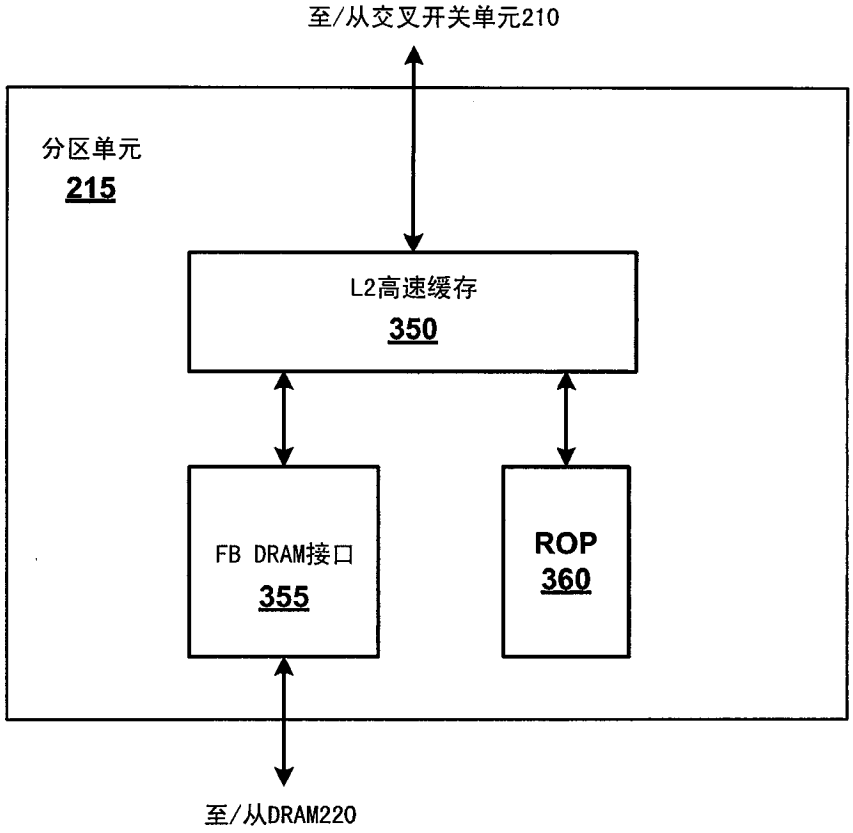

至/从交叉开关单元210
分区单元
215
L2高速缓存
350
FB DRAM接口
355
ROP
360
至/从DRAM220

图 3B

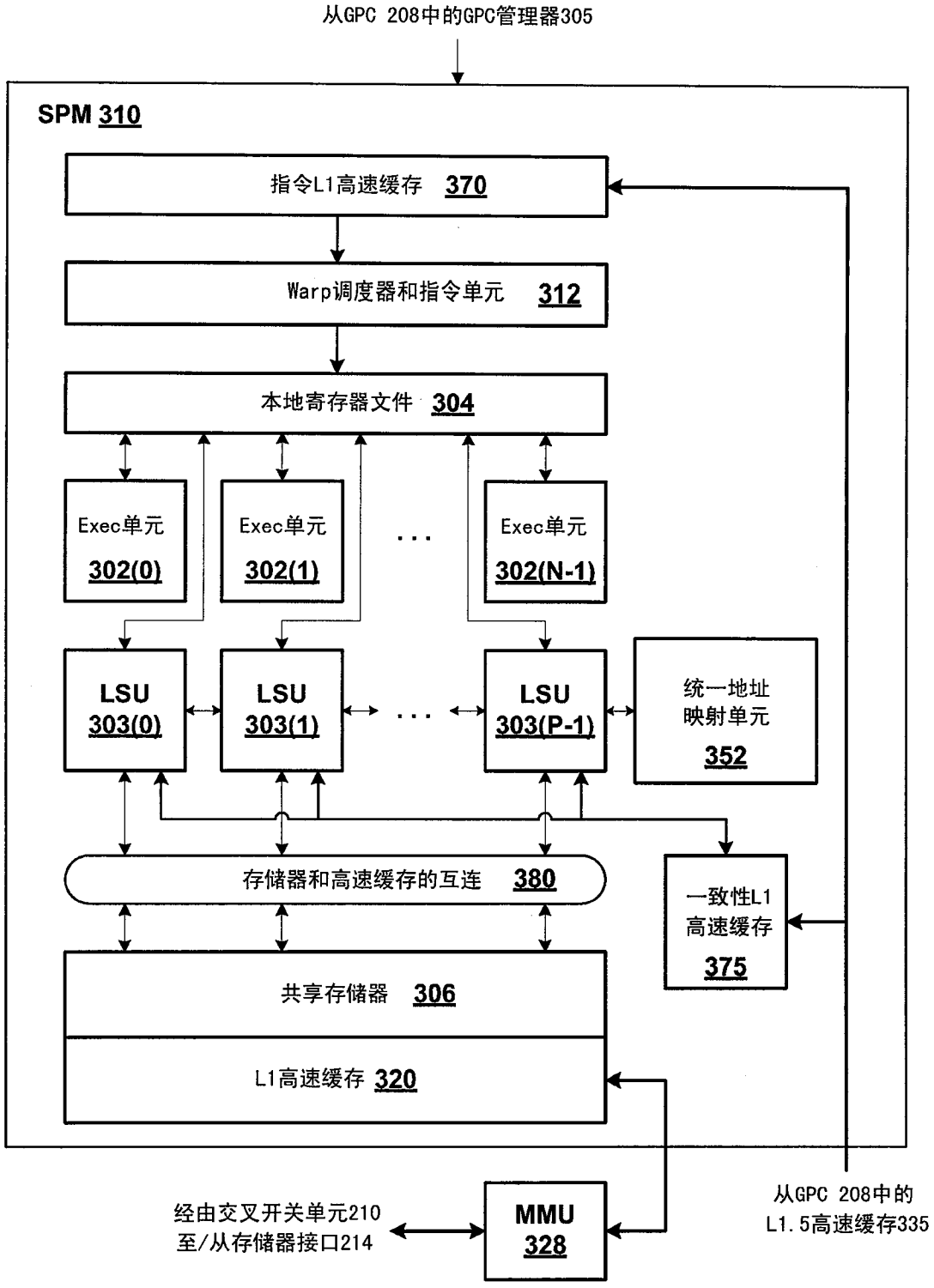

从GPC 208中的GPC管理器305
SPM 310
指令L1高速缓存 370
Warp调度器和指令单元 312
本地寄存器文件 304
Exec单元 302(0)
Exec单元 302(1)
Exec单元 302(N-1)
LSU 303(0)
LSU 303(1)
LSU 303(P-1)
统一地址映射单元 352
存储器和高速缓存的互连 380
一致性L1高速缓存 375
共享存储器 306
L1高速缓存 320
经由交叉开关单元210至/从存储器接口214
MMU 328
从GPC 208中的L1.5高速缓存335

图 3C

示意图

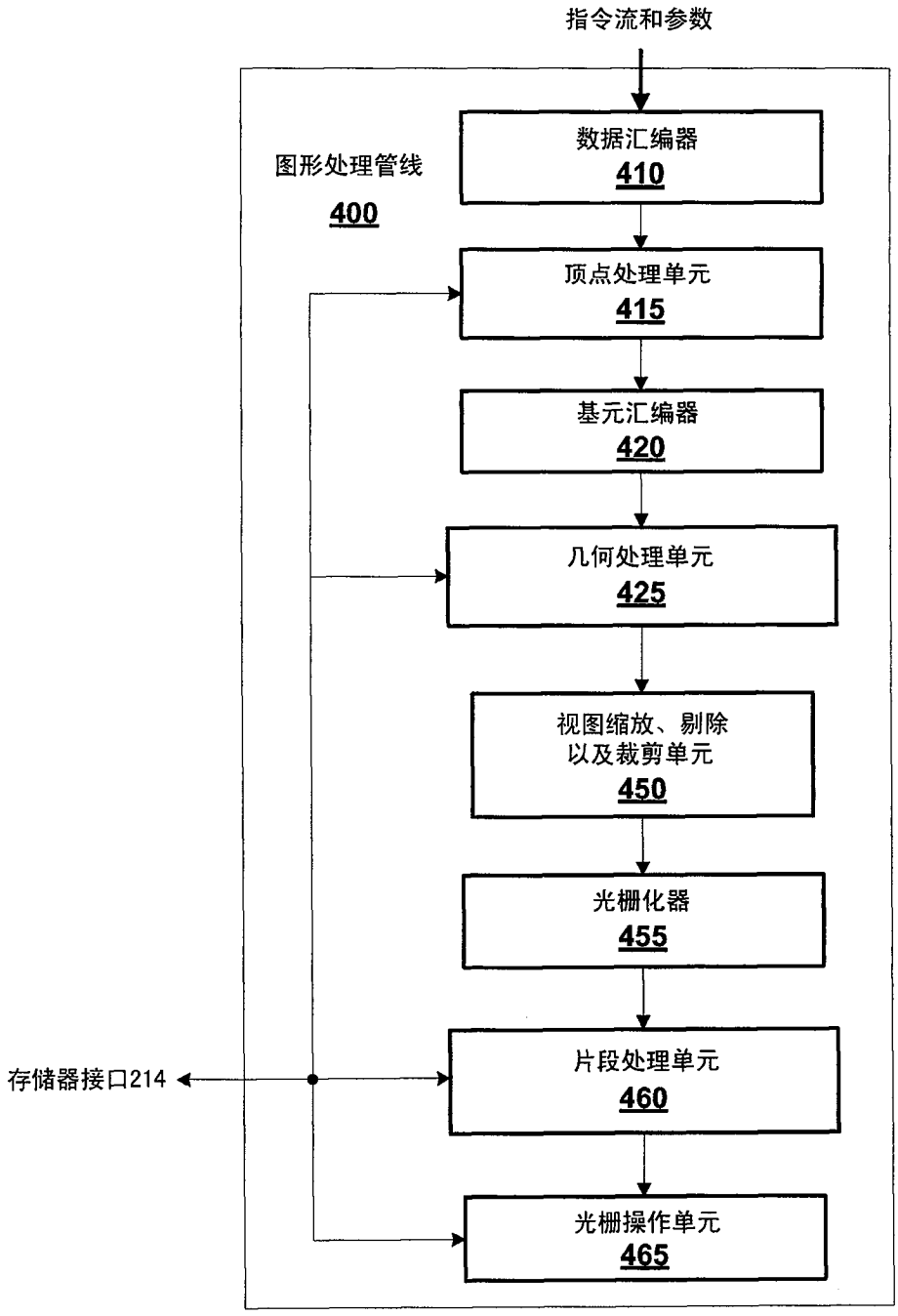

指令流和参数
图形处理管线
400
数据汇编器
410
顶点处理单元
415
基元汇编器
420
几何处理单元
425
视图缩放、剔除
以及裁剪单元
450
光栅化器
455
片段处理单元
460
光栅操作单元
465
存储器接口214

图 4

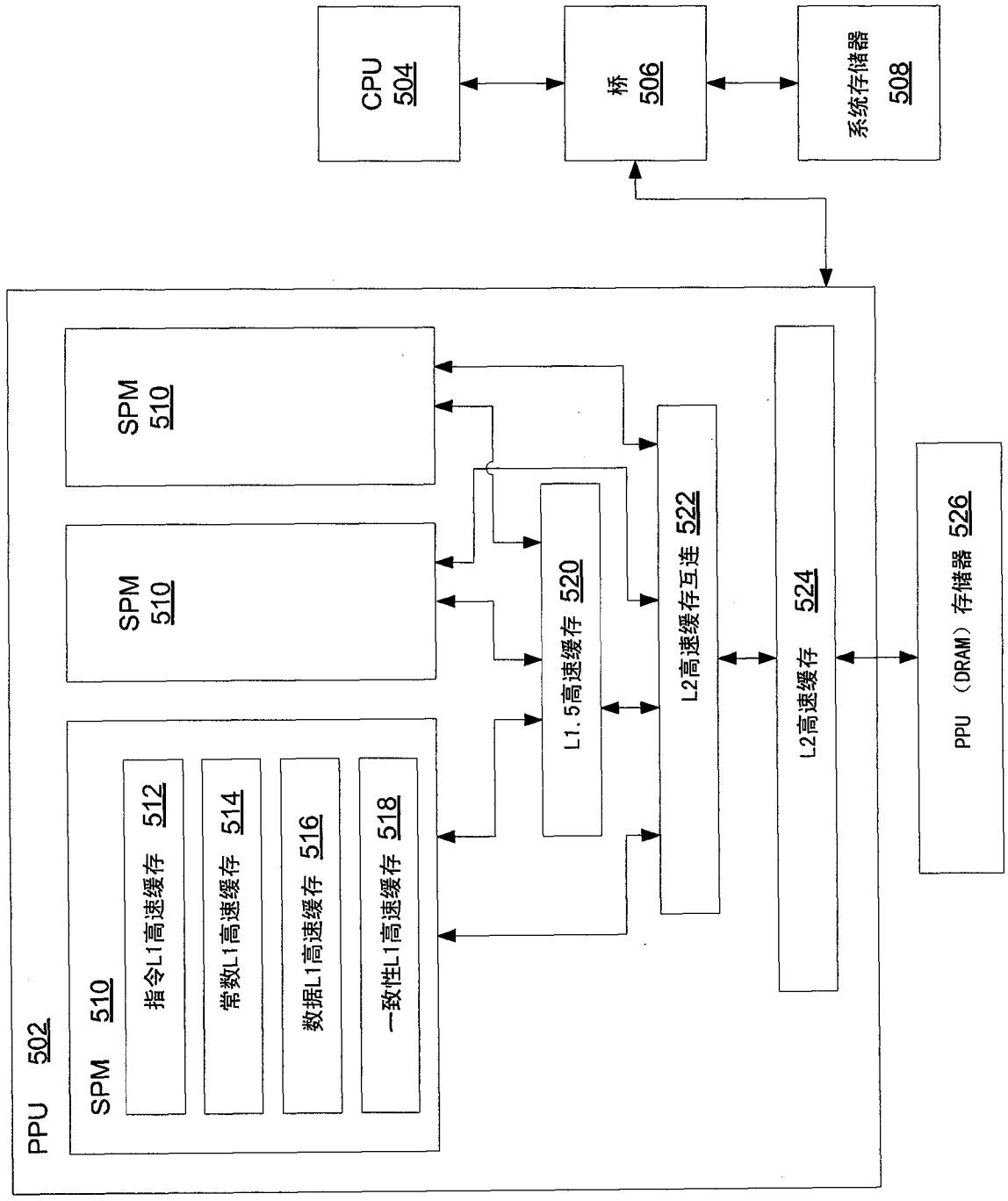

CPU 504
桥 506
系统存储器 508
PPU 502
SPM 510
SPM 510
SPM 510
指令L1高速缓存 512
常数L1高速缓存 514
数据L1高速缓存 516
一致性L1高速缓存 518
L1.5高速缓存 520
L2高速缓存互连 522
L2高速缓存 524
PPU （DRAM）存储器 526

图 5

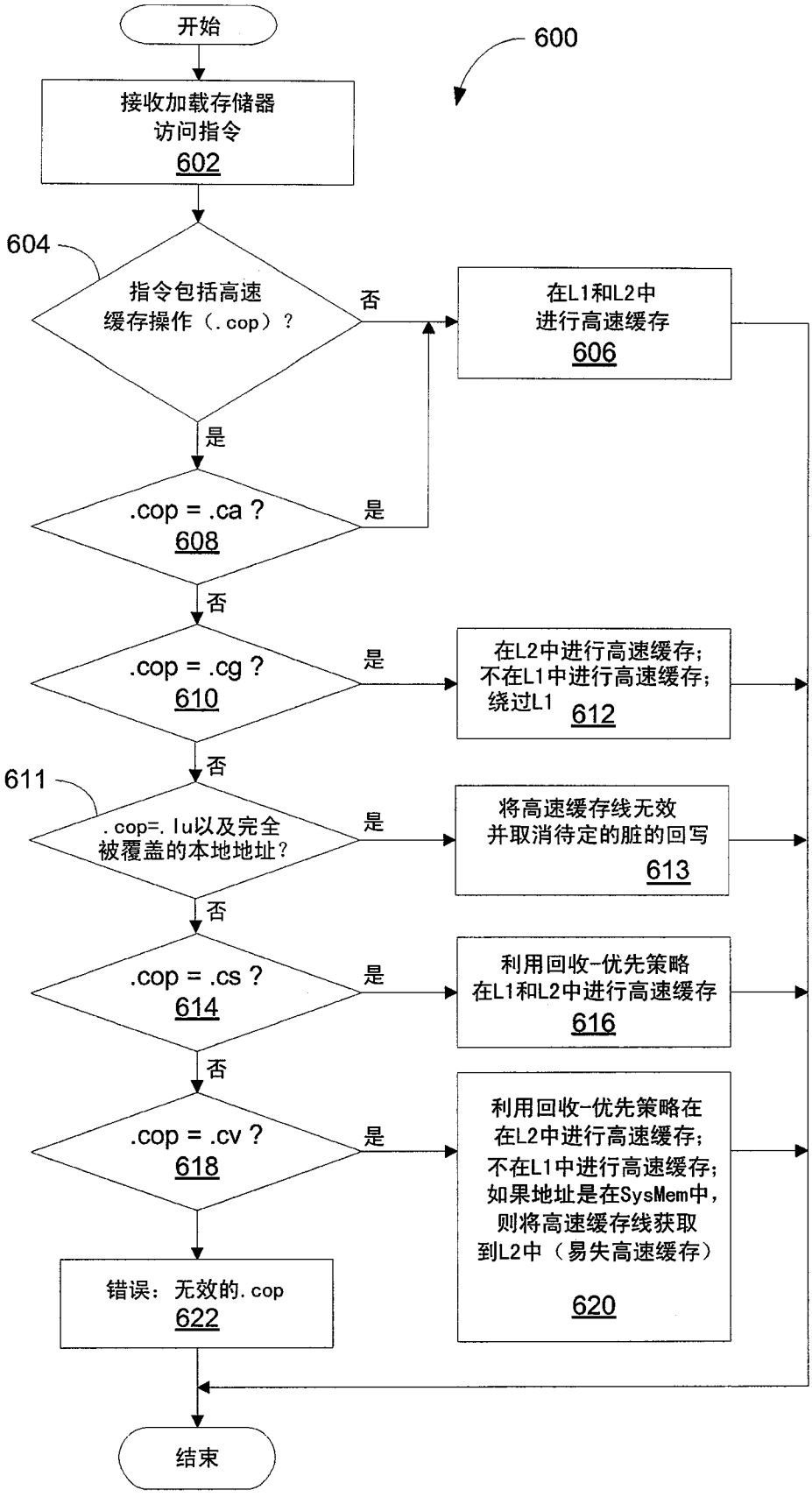

开始
600
接收加载存储器
访问指令
602
604
指令包括高速
缓存操作（.cop）？
否
在L1和L2中
进行高速缓存
606
是
.cop = .ca ?
608
是
否
.cop = .cg ?
610
是
在L2中进行高速缓存；
不在L1中进行高速缓存；
绕过L1
612
否
611
.cop=.lu以及完全
被覆盖的本地地址？
是
将高速缓存线无效
并取消待定的脏的回写
613
否
.cop = .cs ?
614
是
利用回收-优先策略
在L1和L2中进行高速缓存
616
否
.cop = .cv ?
618
是
利用回收-优先策略在
在L2中进行高速缓存；
不在L1中进行高速缓存；
如果地址是在SysMem中，
则将高速缓存线获取
到L2中（易失高速缓存）
620
错误：无效的.cop
622
结束

图 6

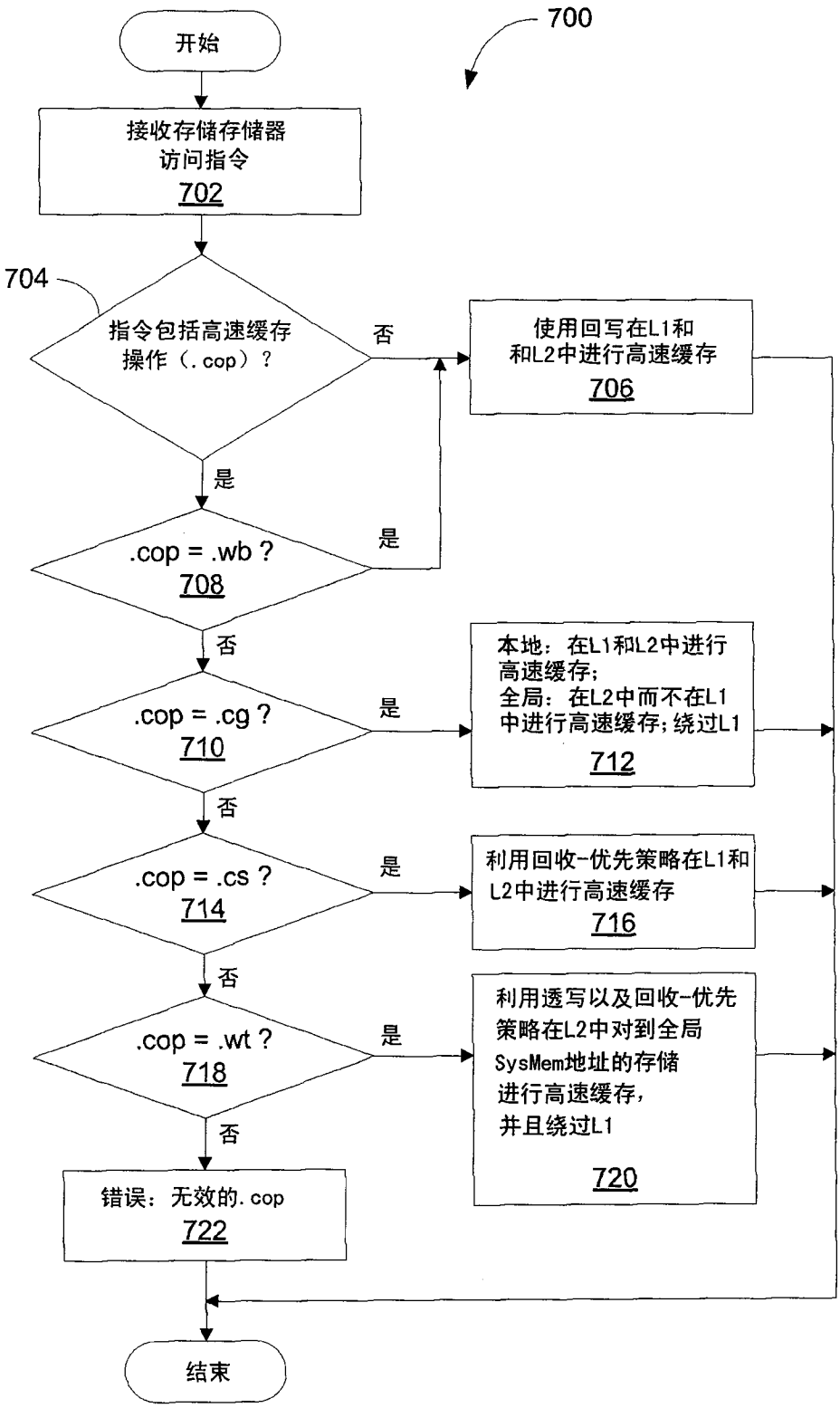

700
开始
接收存储存储器
访问指令
702
704
指令包括高速缓存
操作（.cop）？
否
使用回写在L1和
和L2中进行高速缓存
706
是
.cop = .wb ?
708
是
否
.cop = .cg ?
710
是
本地：在L1和L2中进行
高速缓存；
全局：在L2中而不在L1
中进行高速缓存；绕过L1
712
否
.cop = .cs ?
714
是
利用回收-优先策略在L1和
L2中进行高速缓存
716
否
.cop = .wt ?
718
是
利用透写以及回收-优先
策略在L2中对到全局
SysMem地址的存储
进行高速缓存，
并且绕过L1
720
否
错误：无效的.cop
722
结束

图 7