



(12) 发明专利

(10) 授权公告号 CN 113395216 B

(45) 授权公告日 2024. 04. 09

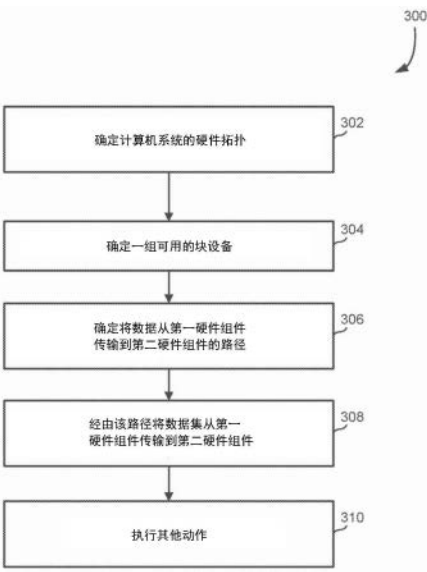
|                                                                                      |                               |
|--------------------------------------------------------------------------------------|-------------------------------|
| (21) 申请号 202110261657.3                                                              | CN 110858840 A,2020.03.03     |
| (22) 申请日 2021.03.10                                                                  | JP 3248659 B2,2002.01.21      |
| (65) 同一申请的已公布的文献号<br>申请公布号 CN 113395216 A                                            | US 10325343 B1,2019.06.18     |
| (43) 申请公布日 2021.09.14                                                                | US 2011264833 A1,2011.10.27   |
| (30) 优先权数据<br>16/816,122 2020.03.11 US                                               | US 2014359563 A1,2014.12.04   |
| (73) 专利权人 辉达公司<br>地址 美国加利福尼亚州                                                        | DE 102015009430 A1,2016.02.25 |
| (72) 发明人 K·K·莫杜库里 C·J·纽伯恩<br>S·森 A·凯拉什 S·乔希                                          | CN 107004013 A,2017.08.01     |
| (74) 专利代理机构 北京市磐华律师事务所<br>11336<br>专利代理师 高伟                                          | CN 109657794 A,2019.04.19     |
| (51) Int.Cl.<br>H04L 69/16 (2022.01)<br>G06F 13/40 (2006.01)<br>H04L 45/00 (2022.01) | CN 109033001 A,2018.12.18     |
| (56) 对比文件<br>CN 106776455 A,2017.05.31<br>CN 110337828 A,2019.10.15                  | US 2015100536 A1,2015.04.09   |
|                                                                                      | CN 109240832 A,2019.01.18     |
|                                                                                      | US 2011179390 A1,2011.07.21   |
|                                                                                      | CN 105404889 A,2016.03.16     |
|                                                                                      | CN 105721357 A,2016.06.29     |
|                                                                                      | US 2009248941 A1,2009.10.01   |
|                                                                                      | US 7561163 B1,2009.07.14      |
|                                                                                      | US 2020065283 A1,2020.02.27   |
|                                                                                      | US 2019238412 A1,2019.08.01   |
|                                                                                      | CN 101763221 A,2010.06.30     |
|                                                                                      | CN 107346351 A,2017.11.14     |
|                                                                                      | CN 104023062 A,2014.09.03     |
|                                                                                      | CN 105190530 A,2015.12.23 (续) |
|                                                                                      | 审查员 庞素琴                       |
|                                                                                      | 权利要求书2页 说明书69页 附图44页          |

(54) 发明名称

在硬件设备之间传输数据的技术

(57) 摘要

本发明公开了在硬件设备之间传输数据的技术,具体公开了在硬件设备之间路由数据传输的装置、系统和技术。在至少一个实施例中,至少部分地基于可用于传输数据的不同路径的一个或更多个特性,确定将数据从计算机系统的第一硬件组件传输到计算机系统的第二硬件组件的路径。



CN 113395216 B

[接上页]

**(56) 对比文件**

EP 2802107 A1, 2014.11.12

US 9436441 B1, 2016.09.06

付正文;杨冲;黄先锋;张帆;金艳.一种最小生成树的自动纹理映射方法.测绘科学.2016, (第07期),全文.

Ukem, E.O.;Dajab, D.D..Enhancing

Hardware Maintenance Through the Use of a Network Management System.《Computing and Information Systems》.2009,

米建伟.电力电缆故障诊断中信号检测与增强技术研究.《中国博士学位论文全文数据库(工程科技II辑)》.2019,

1. 一种处理器, 包括:

一个或多个电路, 用于至少部分地基于设备层次结构树和能够用于传输数据的不同路径的一个或多个特性, 确定将所述数据从计算机系统的第一硬件组件传输到所述计算机系统的第二硬件组件的路径。

2. 根据权利要求1所述的处理器, 其中所述第一硬件组件和所述第二硬件组件中的一个或多个是图形处理单元 (GPU)。

3. 根据权利要求1所述的处理器, 其中所述设备层次结构树为包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示。

4. 根据权利要求3所述的处理器, 其中所述一个或多个电路还用于至少部分地基于外围组件互连快速 (PCIe) 总线设备功能 (BDF) 信息来生成所述设备层次结构树。

5. 根据权利要求3所述的处理器, 其中所述第一硬件组件由第一操作系统 (OS) 实例控制, 所述第二硬件组件由第二操作系统实例控制, 并且所述第一硬件组件和所述第二硬件组件连接在节点间路径上, 所述节点间路径包括存储器结构或网络结构中的至少一个。

6. 根据权利要求1所述的处理器, 其中所述一个或多个电路还用于确定可用块设备的集合, 并至少部分地基于所述可用块设备的集合来确定所述路径。

7. 根据权利要求1所述的处理器, 其中所述路径包括由中间设备管理的缓冲器。

8. 根据权利要求7所述的处理器, 其中所述一个或多个电路还用于确定所述缓冲器允许直接存储器访问 (DMA) 并且具有预定量的可用空间。

9. 根据权利要求1所述的处理器, 其中所述一个或多个电路还用于确定可用块设备的集合, 并至少部分地基于可用链路的集合来确定所述路径。

10. 根据权利要求1所述的处理器, 其中所述一个或多个电路还用于确定对应于多个动态组件条件的多个值, 并至少部分地基于所述多个值来确定所述路径。

11. 一种非暂时性机器可读介质, 其上存储有一组指令, 所述指令如果由一个或多个处理器执行, 则使所述一个或多个处理器至少:

至少部分地基于能够设备层次结构树和用于传输数据的不同路径的一个或多个特性, 确定将所述数据从计算机系统的第一硬件组件传输到所述计算机系统的第二硬件组件的路径, 其中所述第一硬件组件和所述第二硬件组件中的一个或多个是图形处理单元 (GPU)。

12. 根据权利要求11所述的非暂时性机器可读介质, 其中所述设备层次结构树为包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示。

13. 根据权利要求12所述的非暂时性机器可读介质, 其中所述指令如果由所述一个或多个处理器执行, 进一步使所述一个或多个处理器至少部分地基于外围组件互连快速 (PCIe) 总线设备功能 (BDF) 信息来生成所述设备层次结构树。

14. 根据权利要求11所述的非暂时性机器可读介质, 其中所述指令如果由所述一个或多个处理器执行, 进一步使所述一个或多个处理器确定可用块设备的集合, 并至少部分地基于所述可用块设备的集合来确定所述路径。

15. 根据权利要求11所述的非暂时性机器可读介质, 其中所述路径包括由中间设备管理的缓冲器。

16. 根据权利要求15所述的非暂时性机器可读介质, 其中所述指令如果由所述一个或

更多个处理器执行,进一步使所述一个或更多个处理器确定所述缓冲器允许直接存储器访问(DMA)并且具有预定量的可用空间。

17.根据权利要求11所述的非暂时性机器可读介质,其中所述指令如果由所述一个或更多个处理器执行,进一步使所述一个或更多个处理器确定对应于多个动态组件条件的多个值,并至少部分地基于所述多个值来确定所述路径。

18.一种用于在硬件组件之间传输数据的方法,包括:

至少部分地基于设备层次结构树和能够用于传输数据的不同路径的一个或更多个特性,确定将所述数据从计算机系统的第一硬件组件传输到所述计算机系统的第二硬件组件的路径。

19.根据权利要求18所述的方法,还包括经由所述路径将数据集从所述第一硬件组件传输到所述第二硬件组件。

20.根据权利要求18所述的方法,其中所述第一硬件组件和所述第二硬件组件中的一个或更多个是图形处理单元(GPU)。

21.根据权利要求18所述的方法,其中所述设备层次结构树为包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示。

22.根据权利要求18所述的方法,还包括:确定可用块设备的集合,其中至少部分地基于所述可用块设备的集合来确定所述路径。

23.根据权利要求18所述的方法,其中所述路径包括由中间设备管理的缓冲器。

24.根据权利要求23所述的方法,还包括:确定所述缓冲器允许直接存储器访问(DMA)并且具有预定量的可用空间。

25.根据权利要求18所述的方法,还包括:确定对应于多个动态组件条件的多个值,其中至少部分地基于所述多个值来确定所述路径。

26.一种用于在硬件组件之间传输数据的系统,包括:

一个或更多个处理器,用于至少部分地基于设备层次结构树和能够用于传输数据的不同路径的一个或更多个特性,确定将所述数据从计算机系统的第一硬件组件传输到所述计算机系统的第二硬件组件的路径;以及

一个或更多个存储器,用于存储表示所述路径的数据集,其中所述第一硬件组件和所述第二硬件组件中的一个或更多个是图形处理单元(GPU)。

27.根据权利要求26所述的系统,其中所述设备层次结构树为包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示。

28.根据权利要求27所述的系统,其中所述一个或更多个处理器还用于至少部分地基于外围组件互连快速(PCIe)总线设备功能(BDF)信息来生成所述设备层次结构树。

29.根据权利要求26所述的系统,其中所述一个或更多个处理器还用于确定可用块设备的集合,并至少部分地基于所述可用块设备的集合来确定所述路径。

30.根据权利要求26所述的系统,其中所述路径包括由中间设备管理的缓冲器。

31.根据权利要求30所述的系统,其中所述一个或更多个处理器还用于确定所述缓冲器允许直接存储器访问(DMA)并且具有预定量的可用空间。

32.根据权利要求26所述的系统,其中所述一个或更多个处理器还用于确定对应于多个动态组件条件的多个值,并至少部分地基于所述多个值来确定所述路径。

## 在硬件设备之间传输数据的技术

### 技术领域

[0001] 至少一个实施例涉及用于执行为并行计算平台和应用程序接口编写的一个或多个程序的处理资源。例如,至少一个实施例涉及根据本文描述的各种新颖技术的用于在硬件设备之间路由和暂存(stage)数据传输的处理器或计算系统。

### 背景技术

[0002] 在硬件设备之间执行数据传输会占用大量存储器、时间或计算资源。可以改善用于在硬件设备之间执行数据传输的存储器、时间或计算资源的量。

### 附图说明

- [0003] 图1A是示出根据至少一个实施例的计算机系统的框图;
- [0004] 图1B是示出根据至少一个实施例的计算机系统的框图;
- [0005] 图2是示出根据至少一个实施例的计算机系统的框图;
- [0006] 图3示出了根据至少一个实施例的确定传输数据的路径的技术的流程图;
- [0007] 图4示出了根据至少一个实施例的创建设备层次结构树和硬件设备的排序列表的技术的流程图;
- [0008] 图5示出了根据至少一个实施例的构建一个或多个块设备层次结构组的技术的流程图;
- [0009] 图6示出了根据至少一个实施例的目标设备缓冲器管理的技术的流程图;
- [0010] 图7示出了根据至少一个实施例的确定数据传输路径的技术的流程图;
- [0011] 图8示出了根据至少一个实施例的示例性数据中心;
- [0012] 图9示出了根据至少一个实施例的处理系统;
- [0013] 图10示出了根据至少一个实施例的计算机系统;
- [0014] 图11示出了根据至少一个实施例的系统;
- [0015] 图12示出了根据至少一个实施例的示例性集成电路;
- [0016] 图13示出了根据至少一个实施例的计算系统;
- [0017] 图14示出了根据至少一个实施例的APU;
- [0018] 图15示出了根据至少一个实施例的CPU;
- [0019] 图16示出了根据至少一个实施例的示例性加速器集成切片;
- [0020] 图17A和图17B示出了根据至少一个实施例的示例性图形处理器;
- [0021] 图18A示出了根据至少一个实施例的图形核心;
- [0022] 图18B示出了根据至少一个实施例的GPGPU;
- [0023] 图19A示出了根据至少一个实施例的并行处理器;
- [0024] 图19B示出了根据至少一个实施例的处理集群;
- [0025] 图19C示出了根据至少一个实施例的图形多处理器;
- [0026] 图20示出了根据至少一个实施例的图形处理器;

- [0027] 图21示出了根据至少一个实施例的处理器；
- [0028] 图22示出了根据至少一个实施例的处理器；
- [0029] 图23示出了根据至少一个实施例的图形处理器核心；
- [0030] 图24示出了根据至少一个实施例的PPU；
- [0031] 图25示出了根据至少一个实施例的GPC；
- [0032] 图26示出了根据至少一个实施例的流式多处理器；
- [0033] 图27示出了根据至少一个实施例的编程平台的软件栈；
- [0034] 图28示出了根据至少一个实施例的图27的软件栈的CUDA实现；
- [0035] 图29示出了根据至少一个实施例的图27的软件栈的ROCm实现；
- [0036] 图30示出了根据至少一个实施例的图27的软件栈的OpenCL实现；
- [0037] 图31示出了根据至少一个实施例的由编程平台支持的软件；
- [0038] 图32示出了根据至少一个实施例的在图27-30的编程平台上执行的编译代码；
- [0039] 图33更详细地示出了根据至少一个实施例的在图27-30的编程平台上执行的编译代码；
- [0040] 图34示出了根据至少一个实施例的在编译源代码之前转换源代码；
- [0041] 图35A示出了根据至少一个实施例的被配置为使用不同类型的处理单元来编译和执行CUDA源代码的系统；
- [0042] 图35B示出了根据至少一个实施例的被配置为使用CPU和启用CUDA的GPU来编译和执行图35A的CUDA源代码的系统；
- [0043] 图35C示出了根据至少一个实施例的被配置为使用CPU和未启用CUDA的GPU来编译和执行图35A的CUDA源代码的系统；
- [0044] 图36示出了根据至少一个实施例的由图35C的CUDA到HIP转换工具转换的示例性内核；
- [0045] 图37更详细地示出了根据至少一个实施例的图35C的未启用CUDA的GPU；以及
- [0046] 图38示出了根据至少一个实施例的示例性CUDA网格的线程如何被映射到图37的不同计算单元。

## 具体实施方式

[0047] 在以下描述中,阐述了许多具体细节以提供对至少一个实施例的更透彻的理解。然而,对于本领域的技术人员将显而易见的是,可以在没有这些具体细节中的一个或更多的情况下实践本发明构思。

[0048] 图1A是示出根据至少一个实施例的包括数据传输路径确定能力的计算机系统100的框图。在至少一个实施例中,计算机系统100包括在第一CPU插槽(socket)中的第一中央处理单元(CPU)102,其被指定为CPU插槽0,以及在第二CPU插槽中的第二CPU 104,被指定为CPU插槽1。应当理解,出于说明的目的呈现了计算机系统100的特定配置和组件,并且任何合适的计算机系统配置和/或硬件组件可以实现各种实施例的动态数据路由和/或数据传输路径确定能力。在至少一个实施例中,第一CPU102与第一外围组件互连(PCI)快速(PCIe)交换机106通信地耦合,并且第二CPU 104与第二PCIe交换机108通信地耦合。在至少一个实施例中,经由第一PCIe交换机106与第一CPU 102耦合的附加硬件组件被认为在第一域110

中,并且经由第二PCIe交换机108与第二CPU 104耦合的附加硬件组件被认为在第二域112中。在至少一个实施例中,示出分隔线114以更清楚地指示属于第一域110和第二域112的硬件组件。在至少一个实施例中,代替PCIe交换机和链路或除PCIe交换机和链路之外,还使用另一种类型的互连(例如,PCI)或任何其他合适的互连。

[0049] 在至少一个实施例中,第一PCIe交换机106向其他组件提供第一PCIex16链接116,第二PCIe交换机108向其他组件提供第二PCIex16链接118。在至少一个实施例中,第一网络接口卡(NIC)120与第一PCIe交换机106耦合,并且第二NIC 122与第二PCIe交换机108耦合。在至少一个实施例中,第一NIC 120和第二NIC 122与网络耦合,为清楚起见未示出。在至少一个实施例中,第三PCIe交换机124与第一PCIe交换机106耦合,并且第四PCIe交换机126与第二PCIe交换机108耦合。在至少一个实施例中,以GPU 0表示的第一图形处理单元(GPU)128与第三PCIe交换机124耦合,并且以GPU 1表示的第二GPU 130与第四PCIe交换机126耦合。在至少一个实施例中,第一GPU 128具有第一GPU存储器132,并且第二GPU 130具有第二GPU存储器134。在至少一个实施例中,第一GPU存储器132的一部分被指定为第一反弹缓冲器池136,第二GPU存储器134的一部分被指定为第二反弹缓冲器池138。在至少一个实施例中,第一反弹缓冲器池136包括一个或更多个反弹缓冲器的第一集合,第二反弹缓冲器池138包括一个或更多个反弹缓冲器的第二集合。

[0050] 在至少一个实施例中,第一存储设备140与第三PCIe交换机124耦合,并且第二存储设备142与第四PCIe交换机126耦合。在至少一个实施例中,第一存储设备140具有存储器144中的第一数据集,第二存储设备142具有存储器146中的第二数据集。在至少一个实施例中,计算机系统100包括其他组件,为清楚起见未示出,例如与第一CPU 102耦合的第一系统存储器,与第二CPU 104耦合的第二系统存储器104,或与第一CPU102和第二CPU 104两者耦合的共享系统存储器。在至少一个实施例中,计算机系统100包括在第一CPU 102和第二CPU 104之间的链路148,例如快速路径互连(QPI)链路,超路径互连(UPI)链路或任何其他合适的数据传输链接。在至少一个实施例中,计算机系统100包括在第一GPU 128和第二GPU 130之间的链路150,诸如NVLink GPU互连或任何其他合适的GPU-GPU数据传输互连。在至少一个实施例中,经由PCIe交换机的数据传输被认为是在第一存储器结构上,并且通过GPU-GPU链路(诸如链路150)的数据传输被认为是在第二存储器结构上。

[0051] 在至少一个实施例中,计算机系统100的数据传输路径确定能力可以用于确定GPU(例如,第一GPU 128)和存储设备(例如,第二存储设备142)之间的数据的数据传输路径。在至少一个实施例中,路由数据传输包括确定数据传输路径并使得数据集在确定的数据传输路径上传输。在至少一个实施例中,数据传输路径包括反弹缓冲器。在至少一个实施例中,在将分段数据集从反弹缓冲器传送到目标存储器(例如,用于读取操作)或存储设备(例如,用于写操作)之前,将数据集传送到反弹缓冲器被认为是对数据集分段。

[0052] 在至少一个实施例中,计算机系统100的数据传输路径确定能力包括数据传输计划确定能力。在至少一个实施例中,路由数据传输包括确定一组阶段。在至少一个实施例中,阶段是一系列端点设备、中点设备和链路,这些数据可以通过由计算代理发起的单个命令来传输。在至少一个实施例中,确定数据传输计划包括确定要在中点和/或端点的子集中使用的缓冲器的集合。在至少一个实施例中,确定数据传输计划包括确定用于在每个阶段发送命令以实现传输的计算代理的集合。在至少一个实施例中,确定数据传输计划包括确

定哪个计算代理协调由计算代理的集合对每阶段命令的发起。在至少一个实施例中,协调至少部分地基于确保在下一阶段的传输开始之前一个阶段中的传输已经完成。

[0053] 在至少一个实施例中,计算机系统100的数据传输路径确定能力至少部分地基于可用于传输数据的不同路径的一个或多个特性确定将数据从计算机系统100的第一硬件组件传输到计算机系统100的第二硬件组件的路径。在至少一个实施例中,数据传输路径确定被称为动态IO路由。在至少一个实施例中,第一硬件组件和第二硬件组件中的一个或多个是GPU。在至少一个实施例中,应用程序(例如,在第一CPU 102,第二CPU 104,第一GPU 128和/或第二GPU 130上运行)经由API(例如,在第一CPU 102,第二CPU 104,第一GPU 128和/或第二GPU 130上运行)发送IO函数调用(例如,指定读取或写入操作),以及API执行动作以动态路由IO函数调用请求的数据传输操作。应该理解,当一个或多个API被称为执行关于实施例或技术的方面的动作时,运行API的计算机系统的一个或多个硬件组件(例如,CPU,GPU和/或其他硬件组件)执行技术的动作或方面。在至少一个实施例中,使用一个以上的API来执行动态数据传输路由(例如,用于不同函数调用的不同API和/或调用另一个API的第一API),而不是一个API来执行动态数据传输路由。在至少一个实施例中,由API执行以动态路由数据传输操作的动作对于应用程序是未知的。

[0054] 在至少一个实施例中,运行API的计算机系统至少部分地基于包括第一硬件组件和第二硬件组件的硬件拓扑的表示来确定数据传输路径。在至少一个实施例中,硬件拓扑的表示是设备层次结构树,并且计算机系统100的一个或多个组件(例如,运行API的第一CPU 102,第二CPU 104,第一GPU 128和/或第二GPU 130)至少部分地基于PCIe总线设备功能(BDF)信息生成设备层次结构树。在至少一个实施例中,计算机系统100的一个或多个组件确定可用块设备的集合,并且至少部分地基于所确定的可用块设备的集合来确定数据传输路径。在至少一个实施例中,数据传输路径包括由中间设备(例如,不是数据传输的源或目的地的GPU)管理的缓冲器(例如,反弹缓冲器)。在至少一个实施例中,计算机系统100的一个或多个组件确定缓冲器是否允许直接存储器访问(DMA),并且具有预定量的可用空间,并且至少部分地基于DMA能力和/或可用空间确定数据传输路径。在至少一个实施例中,计算机系统100的一个或多个组件确定与多个动态组件条件相对应的多个值,并且至少部分地基于所确定的多个值中的一个或多个来确定数据传输路径。在至少一个实施例中,API至少部分地基于将至少一个函数调用(例如,至少一个输入/输出控制(IOCTL)调用)发送给内核驱动器,经由所确定的数据传输路径来传输数据。

[0055] 在至少一个实施例中,数据传输是动态路由的。在至少一个实施例中,当不能直接将目标存储器用于执行IO操作时,经由反弹缓冲器(例如,在GPU中)路由数据传输。在至少一个实施例中,将存储器预先保留(例如,由API)在应用程序可以访问的一个或多个GPU中,以用作反弹缓冲器或高速缓存。在至少一个实施例中,在一种或更多种情况下,数据传输使用基于GPU的反弹缓冲器,例如:目标存储器指针不是4字节对齐的GPU地址;文件偏移量不与扇区大小对齐;IO事务的大小不是扇区大小的倍数(例如512字节或4KB);对等NVMe设备不在同一PCIe根层次结构中;目标存储器是预定类型的托管存储器(例如cudaMalloc托管存储器);基于暴露的GPU的地址寄存器(BAR)存储器小于预定阈值大小;和/或IO被高速缓存在本地高速存储器中。在至少一个实施例中,API为反弹缓冲器分配预定量(例如64KB)的预保留存储器,以支持缓存的IO情况。在至少一个实施例中,API响应于函数调用

(例如, `cuFileDriverOpen()`) 为反弹缓冲器分配存储器。在至少一个实施例中, API分配附加存储器作为数据传输操作(例如, 诸如 `cuFileRead` 或 `cuFileWrite` 的读/写操作)的一部分。在至少一个实施例中, API至少部分地基于由应用程序设置的一个或多个属性(例如, 通过使用 `props.max_device_cache_pages`) 来分配附加存储器作为数据传输操作的一部分。在至少一个实施例中, API响应于函数调用(例如, `cuFileDriverClose()`) 释放缓冲器。在至少一个实施例中, 至少一个处理器隐式地移动数据。在至少一个实施例中, 至少一个处理器在将文件映射到存储器之后(例如, 使用Linux `mmap`命令), 隐式地移动数据。在至少一个实施例中, 在为页面故障提供服务时, 刺激隐式传输。在至少一个实施例中, 在为页面故障提供服务时, 页面故障处理程序使用类似于通过基于 `cuFile` 的API调用发起的传输的实现方式来发起数据传输。在至少一个实施例中, 提供了一种并行计算平台和应用程序编程接口(例如, `OpenCL`和/或计算统一设备架构(CUDA))。在至少一个实施例中, 操作系统或独立驱动器中的文件系统、块系统、对象系统或键-值存储系统使用与采用基于 `cuFile` 的API调用发起的传输类似的实现方式执行预读操作以完成预取。在至少一个实施例中, 至少一个处理器至少部分地基于显式预取(例如, CUDA `cudaMemPrefetch`) 来移动数据。

[0056] 在至少一个实施例中, API基于PCIe层次结构执行动态IO路由。在至少一个实施例中, API执行一项或更多项检查, 以解决在整个PCIe根层次结构中某些硬件配置中对等(P2P)传输在PCIe对等设备上执行不佳的情况。在至少一个实施例中, 用户空间库中的读写应用程序编程接口(API)检查层次结构差异(例如, 硬件组件之间的距离), 并尝试通过将IO数据弹回中间GPU存储器(如果可用)来尝试执行IO操作, 并使用P2P操作(例如 `cudaMemcpyPeerToPeer`) 将数据从中间GPU存储器移动到目标GPU存储器。在至少一个实施例中, 动态IO路由技术包括发现关于GPU和所支持的块存储设备的PCIe层次结构。在至少一个实施例中, API的动态IO路由技术将IO传输到相同PCIe根层次结构下的GPU中的预先保留的缓冲器空间, 并通过GPU-GPU链路(例如, `NVLink`) 将数据复制到目标GPU存储器。在至少一个实施例中, 动态IO路由技术包括在不同OS实例的管理下使用GPU中的空间。在至少一个实施例中, 动态IO路由技术对关于是否使用包括诸如 `NVLink` 之类的存储器结构或诸如 `InfiniBand` 之类的网络结构的路径进行权衡。

[0057] 在至少一个实施例中, API至少部分地基于未对齐的GPU存储器指针和IO偏移中的一个或多个来执行动态IO路由。在至少一个实施例中, 如果GPU地址未对齐(例如, 不是4字节的倍数)或IO偏移量不是扇区大小的倍数, 则读和/或写需要对一个设备扇区大小(例如512或4K字节)执行未对齐的读取, 然后执行对齐的读/写, 并可能执行另一个未对齐的读/写。在至少一个实施例中, 一个或多个API通过使用对齐的IO到预先保留的GPU存储器来处理未对其, 然后将数据移动到目标位置。

[0058] 在至少一个实施例中, API至少部分地基于确定使用了预定类型的托管存储器(例如, `cudaMallocManaged`存储器)来执行动态IO路由。在至少一个实施例中, API执行动态IO路由以解决P2P远程DMA(RDMA)不适用于预定类型的托管存储器的情况。在至少一个实施例中, 由API的动态IO路由通过使用GPU反弹缓冲器来支持托管存储器, 并且执行反弹缓冲器与目标存储器之间的数据(例如, `cudaMemcpy`)的传输。在至少一个实施例中, API使用驱动程序设置过程来发现GPU和NIC。在至少一个实施例中, API使用缓冲器注册来确定缓冲器的类型和缓冲器基地址寄存器1(BAR1)的限制。在至少一个实施例中, API使用文件导入来发

现文件系统 (FS) 和存在的块设备。在至少一个实施例中, API至少部分地基于选择靠近与块设备和NIC相关联的PCIe交换机的反弹缓冲器来确定IO路径。在至少一个实施例中, API预计算和/或累积至少一个校验和或其他元数据信息以用于缓冲器传输。在至少一个实施例中, API与数据传输并行地将校验和和/或其他元数据信息传递回驱动器。在至少一个实施例中, API执行从本地存储的高速缓存拉取。在至少一个实施例中, 当识别的文件系统或存储不支持GPU直接DMA时, API通过系统存储器路由数据传输路径。在至少一个实施例中, 如果系统软件、内核驱动程序和/或用户驱动程序中的一个或多个不存在或不支持特定路径, 则API退回到使用CPU反弹缓冲器。在至少一个实施例中, 如果未安装启用动态数据传输路由的驱动程序 (例如, GPU直接存储驱动程序) 和/或厂商堆栈不支持动态数据传输路由, 则API退回到使用CPU反弹缓冲器。在至少一个实施例中, 如果直接路径 (例如, 由Linux 0\_DIRECT标志选择的路径) 对于系统软件中的特定情况不可用 (例如, 必须更新元数据, 必须计算校验和), 则API退回到使用CPU反弹缓冲器。

[0059] 在至少一个实施例中, API从应用程序接收函数调用, 以从存储器146中的第二数据集向第一GPU存储器132中的目标存储器152传输数据集 (例如, 读取调用)。在至少一个实施例中, API将先前已经为计算机系统100和所标识的可用块设备生成了至少一个PCIe设备层次结构树。在至少一个实施例中, API还确定一个或多个设备和/或路径特性, 例如用于数据传输的一个或多个延迟度量。在至少一个实施例中, API为来自应用程序的函数调用中指定的数据集确定数据传输路径。在至少一个实施例中, API确定数据传输路径是经由第二PCIe交换机126从第二存储设备142到反弹缓冲器138, 然后是经由链路150从反弹缓冲器138传输到目标存储器152。在至少一个实施例中, API支持确定数据传输路径, 该数据传输路径包括任意数量的路径段和任意数量的暂存缓冲器 (staging buffer), 以解决广泛变化的计算机系统硬件设备架构。在至少一个实施例中, API至少部分地基于一个或多个生成的PCIe设备层次结构树、可用的块设备、设备特性和/或路径特性来确定数据传输路径。在至少一个实施例中, PCIe设备层次结构树是至少部分地基于基础硬件拓扑生成的数据结构。

[0060] 图1B是根据至少一个实施例的计算机系统160的框图。在至少一个实施例中, 计算机系统160以与参照图1A所描述的计算机系统100相似但不一定相同的方式构造。在至少一个实施例中, 图1B的至少一些组件类似于参照图1A描述的组件, 并且包括为清楚起见在图1B未示出的方面, 例如GPU存储器、反弹缓冲器池、目标存储器以及存储器中的数据集。

[0061] 在至少一个实施例中, 计算机系统160包括在第一CPU插槽中的第一CPU 162, 其被指定为CPU插槽0; 在第二CPU插槽中的第二CPU 164, 被指定为CPU插槽1。在一个实施例中, 计算机系统160包括通过网络170进行数据通信的第一NIC 166和第二NIC168。在至少一个实施例中, 网络170是有线网络 (例如, 以太网或InfiniBand网络), 无线网络或其任何合适的组合。在至少一个实施例中, 第一CPU 162运行第一OS实例, 第二CPU 164运行不同于第一OS实例的第二OS实例。在至少一个实施例中, 计算机系统160的CPU插槽0和计算机系统160的CPU插槽1未通过对应于图1A的链路148的CPU-CPU链路 (例如, QPI或UPI) 连接。在至少一个实施例中, 计算机系统160包括被指定为GPU 0的第一GPU172 (例如, 对应于第一GPU 128) 和被指定为GPU 1的第二GPU 174 (例如, 对应于第二GPU 130)。在至少一个实施例中, 计算机系统160包括在第一GPU 172和第二GPU 174之间的GPU-GPU链路176 (例如, NVLinkGPU互连或任何其他合适的GPU-GPU数据传输互连)。在至少一个实施例中, GPU-GPU链路176是存

存储器结构(例如,Nvidia的NVLink,英特尔计算表达链路(CXL),AMD的Infinity)而不是网络结构,并且允许第一OS实例中的计算元件直接执行加载、存储和由另一个OS实例控制的存储器中的原子操作。

[0062] 在至少一个实施例中,计算机系统160的第一节点178是指通过不包括网络170或GPU-GPU链路176的路径与第一CPU 162进行数据通信的计算机系统160的组件。在至少一个实施例中,计算机系统160的第二节点180是指通过不包括网络170或GPU-GPU链路176的路径与第二CPU164进行数据通信的计算机系统160的组件。在至少一个实施例中,第一节点178包括第三GPU 184,其被指定为GPU 2,第二节点180包括第四GPU 186,其被指定为GPU 3。在至少一个实施例中,第三GPU 184包括第三GPU存储器188,第四GPU包括第四GPU存储器190。在至少一个实施例中,第三GPU存储器188的至少一部分被指定为第三反弹缓冲器池,为了清楚起见未示出,和/或第四GPU存储器190的至少一部分被指定为第四反弹缓冲器池,为了清楚起见未示出。在至少一个实施例中,第一OS实例管理第一节点178的组件,并且与第一OS实例不同的第二OS实例管理第二节点180的组件。

[0063] 在至少一个实施例中,动态IO路由技术包括在不同的OS实例的管理下使用GPU中的空间。在至少一个实施例中,动态IO路由技术对关于是否使用包括诸如NVLink之类的存储器架构(例如,GPU-GPU链路176)或诸如InfiniBand之类的网络架构(例如网络170)的路径进行权衡。在至少一个实施例中,动态IO路由技术确定路径,该路径包括由第一OS实例管理的第一硬件组件(例如,第一节点178的硬件组件)和由第二OS实例管理的第二硬件组件(例如,第二节点180的硬件组件)之间的节点间路径。在至少一个实施例中,节点间路径包括存储器结构上的链路或网络结构上的链路中的至少一个。在至少一个实施例中,网络结构上的节点间路径还包括中间设备(例如,NIC)。在至少一个实施例中,用于数据传输的路径的源和/或目的地在没有直接连接到GPU-GPU节点间链路的GPU的GPU存储器中(例如,第三GPU 184中的第三GPU存储器188)。在至少一个实施例中,用于数据传输的路径包括具有与另一GPU的节点间GPU-GPU连接的中间GPU(例如,用于从第三GPU 184进行数据传输的路径包括第一GPU 172作为中间GPU,以当动态IO路由技术确定路径包括GPU-GPU链路176时,经由GPU-GPU链路176将数据传输到第二节点180中的硬件组件)。在至少一个实施例中,用于数据传输的路径包括中间GPU中的反弹缓冲器。在至少一个实施例中,用于数据传输的路径至少部分地基于计算机系统160的硬件拓扑的至少一种表示。在至少一个实施例中,用于数据传输的路径至少部分地基于代表第一节点178的硬件拓扑的第一设备层次结构树,以及代表第二节点180的硬件拓扑的第二设备层次结构树。在至少一个实施例中,数据传输的路径至少部分地基于组合的硬件表示(例如,表示第一节点178和第二节点180两者的硬件拓扑的设备层次结构树)。在至少一个实施例中,用于数据传输的路径包括至少一个附加节点。在至少一个实施例中,用于数据传输的路径遍历由至少一个附加OS控制的至少一个硬件组件。

[0064] 图2是示出根据至少一个实施例的包括数据传输路径确定能力的计算机系统200的框图。在至少一个实施例中,计算机系统200包括第一CPU202和第二CPU 204。应当理解,计算机系统200的特定配置和组件是出于说明的目的而呈现的,并且任何合适的计算机系统配置和/或硬件组件可以实现各种实施例的动态数据路由和/或数据传输路径确定能力。在至少一个实施例中,第一CPU 202通过链路203(诸如超路径互连(UPI)链路或任何其他合

适的CPU-CPU数据通信互连)与第二CPU 204通信地耦合。在至少一个实施例中,代替或除了链路203之外,还存在不止一个链路。在至少一个实施例中,第一CPU 202与第一PCIe交换机206和第二PCIe交换机208通信地耦合,以及第二CPU 204与第三PCIe交换机210和第四PCIe交换机212通信地耦合。在至少一个实施例中,经由第一PCIe交换机206或第二PCIe交换机208与第一CPU 202耦合的附加硬件组件被认为在第一域214中,并且通过第三PCIe交换机210或第四PCIe交换机212与第二CPU 204耦合的附加硬件组件被认为在第二域216中。根据至少一个实施例,示出了分隔线218以更清楚地指示属于第一域214和第二域216的硬件组件。

[0065] 在至少一个实施例中,计算机系统200包括与第一CPU 202通信地耦合的第一组系统存储器220,以及与第二CPU 204通信地耦合的第二组系统存储器222。在至少一个实施例中,第一组系统存储器220和/或第二组系统存储器222可以是动态随机存取存储器(DRAM),或任何其他合适类型的系统存储器。在至少一个实施例中,第一组GPU 224经由第一PCIe交换机206和第二PCIe交换机208与第一CPU 202耦合。在至少一个实施例中,第二组GPU 226经由第三PCIe交换机210和第四PCIe交换机212与第二CPU 204耦合。

[0066] 在至少一个实施例中,第一组GPU 224包括第一组八个GPU,示出为GPU 228,GPU 230,GPU 232,GPU 234,GPU 236,GPU 238,GPU 240和GPU 242。在至少一个实施例中,第二组GPU 226包括第二组八个GPU,示出为GPU 244,GPU 246,GPU 248,GPU 250,GPU 252,GPU 254,GPU 256和GPU 258。在至少一个实施例中,第一组GPU 224包括将第一组GPU 224的GPU耦合到第一PCIe交换机206和第二PCIe交换机208的附加PCIe交换机,示出为第五PCIe交换机260,第六PCIe交换机262,第七PCIe交换机264和第八PCIe交换机266。在至少一个实施例中,第二组GPU 226包括将第二组GPU 226的GPU耦合到第三PCIe交换机210和第四PCIe交换机212的附加PCIe交换机,示出为第九PCIe交换机268,第十PCIe交换机270,第十一PCIe交换机272和第十二PCIe交换机274。

[0067] 在至少一个实施例中,第一组GPU 224包括第一组数据通信互连,示出为互连276,互连278,互连280和互连282。在至少一个实施例中,第二组GPU 226包括第二组数据通信互连,示出为互连284,互连286,互连288和互连290。在至少一个实施例中,第一组和/或第二组数据通信互连是高速数据通信互连,例如InfiniBand互连,或任何其他合适类型的数据通信互连。

[0068] 在至少一个实施例中,计算机系统200包括第一组GPU 224和/或第二组GPU 226的一个或更多个GPU之间的通信路径292,为清楚起见未示出连接和组件。在一个实施例中,通信路径292不经过包括第一CPU 202或第二CPU 204的路径。在至少一个实施例中,通信路径292包括一个或更多个交换机。在至少一个实施例中,通信路径292包括一组交换机,所述一组交换机使得能够在第一组GPU 224中的任何GPU与第一组GPU 224中的任何其他GPU之间,在第一组GPU 224中的任何GPU与第二组GPU 226中的任何其他GPU之间,以及第二组GPU 226中的任何GPU与第二组GPU 226中的任何其他GPU之间进行数据通信。在至少一个实施例中,通信路径292的一组交换机支持从一个GPU到另一GPU的直接GPU到GPU数据通信,而无需通过附加中间GPU或穿过通过CPU的通信链路。在至少一个实施例中,通信路径292不包括第一组数据通信互连或第二组数据通信互连。在至少一个实施例中,通信路径292包括第一组数据通信互连和/或第二组数据通信互连中的一个或更多个互连。

[0069] 在至少一个实施例中,第一存储控制器291通过接口293与第一PCIe交换机206和第二PCIe交换机208通信地耦合。在至少一个实施例中,第二存储控制器294通过接口295与第三PCIe交换机210和第四PCIe交换机212通信地耦合。在至少一个实施例中,接口293是第一网络接口卡(NIC),而接口295是第二NIC。在至少一个实施例中,计算机系统200包括一个或更多个附加NIC,为清楚起见未示出。在至少一个实施例中,第一存储控制器291控制第一组存储设备296的存储操作,而第二存储控制器294控制第二组存储设备297的存储操作。在至少一个实施例中,替代非易失性存储器快速(NVME)存储设备(诸如第一组存储设备296和第二组存储设备297),或者除此以外,还使用其他一些合适类型的存储设备。在至少一个实施例中,第一存储器298与第一存储控制器291耦合并且第二存储器299与第二存储控制器294耦合。在至少一个实施例中,代替或除了DRAM(诸如第一存储器298和第二存储器299)之外,还使用某种其他类型的存储器(例如,SRAM)。

[0070] 在至少一个实施例中,计算机系统200的数据传输路径确定能力至少部分地基于可用于传输数据的不同路径的一个或更多个特性来确定从计算机系统200的第一硬件组件向计算机系统200的第二硬件组件传输数据的路径。在至少一个实施例中,数据传输路径确定被称为动态IO路由。在至少一个实施例中,第一硬件组件和第二硬件组件中的一个或更多个是GPU。在至少一个实施例中,应用程序(例如,在第一CPU 202,第二CPU 204和/或计算机系统200的一个或更多个GPU上运行)经由API(例如在第一CPU 102,第二CPU 104和/或计算机系统200的一个或更多个GPU上运行)发送IO函数调用(例如,指定读取或写入操作),并且API执行动作以动态地路由IO函数调用请求的数据传输操作。在至少一个实施例中,不止一个API用于执行动态数据传输路由(例如,用于不同函数调用的不同API,和/或调用另一个API的第一API)。在至少一个实施例中,API执行动作以响应于除接收IO函数调用以外的其他一些触发动作而动态地路由数据传输操作。在至少一个实施例中,由API执行以动态地路由数据传输操作的动作对于应用程序是未知的。在至少一个实施例中,动态数据传输路由通过增加数据传输性能(例如,通过减少数据传输时间)提供了好处。在至少一个实施例中,将IO函数调用发送给API的应用程序在用户级别的CPU上运行。

[0071] 在至少一个实施例中,API执行从第一硬件组件端点(例如,GPU,NIC,CPU,NVMe,串行连接的SCSI(SAS),FPGA,ASIC,智能存储,PCIe组件或任何其他合适的计算外围设备)到第二硬件组件端点(例如,GPU,NIC,CPU,NVMe,SAS,FPGA,ASIC,智能存储,PCIe组件或任何其他合适的计算外围设备)的动态数据传输路由。在至少一个实施例中,不需要跨平台改变的通用API抽象被应用程序调用。在至少一个实施例中,API包括用于诸如数据的读取或写入,缓冲器注册和注销,和/或初始化和完成操作的API。在至少一个实施例中,降低了平台特定的非均匀存储器访问(NUMA)优化的重要性。在至少一个实施例中,API在优化的实现的顶部上运行。在至少一个实施例中,API可以将路径段的任何组合拼凑在一起以形成完整的路径。在至少一个实施例中,API可以添加附加的缓冲,以及在缓冲之上或之中的操作,诸如对齐,暂存,划分和/或专门化。在至少一个实施例中,API可以涉及代理的任何组合以读取/写入,管理存储器,管理通信和/或执行附加的转换。

[0072] 在至少一个实施例中,API以设备拓扑(例如,PCIe层次结构树)和成本度量(例如,延迟度量)的表示形式发现或读取。在至少一个实施例中,设备拓扑和/或成本度量的表示被存储在CPU存储器中的用户驱动器中。在至少一个实施例中,API选择路径,该路径可以包

括多个段而不是单个传输。在至少一个实施例中,API选择将建立传输的代理(例如,在GPU中)。在至少一个实施例中,API选择将发起和/或管理转移的代理。在至少一个实施例中,API选择一种或更多种同步技术。在至少一个实施例中,API确定缓冲器的放置和/或大小。在至少一个实施例中,API选择任何特殊操作(例如,压缩,解压缩,校验和,签名和/或其他安全措施)。

[0073] 在至少一个实施例中,API执行一个或更多个配置操作。在至少一个实施例中,API和/或由API选择的代理管理存储器(例如,分配,设置存储器类型,在完成时解除分配)。在至少一个实施例中,API和/或由API选择的代理管理同步(例如,分配,准备,在完成时解除分配)。在至少一个实施例中,API设置用于代理的命令。在至少一个实施例中,API执行命令。在至少一个实施例中,API存储对应于与数据传输路径有关的至少一个特性的至少一个值(例如,位置,NUMA亲和力,存储器管理策略)。在至少一个实施例中,API临时存储至少一个值(例如,仅在数据传输期间)。在至少一个实施例中,API持久地存储至少一个值(例如,在数据传输之后仍然,尽管不一定是永久的)。在至少一个实施例中,API确定在传输数据时存储至少一个值。

[0074] 在至少一个实施例中,API接收开放驱动程序调用(例如,cuFileDriverOpen)并设置资源(例如,指定缓冲器,确定设备拓扑),以至少部分地基于接收到的开放驱动程序函数调用来实现动态数据传输路由。在至少一个实施例中,将cuFileDriverOpen指定为:CUfileerror\_tcuFileDriverOpen();不带参数。在至少一个实施例中,cuFileDriverOpen在成功打开时返回CU\_FILE\_SUCCESS,如果无法打开驱动程序则返回CU\_FILE\_DRIVER\_NOT\_INITIALIZED,如果无法打开则返回CU\_FILE\_PERMISSION\_DENIED,如果已经打开了驱动程序则返回CU\_FILE\_ALREADY\_OPEN,并且如果cuFile库和内核驱动程序不匹配则返回CU\_FILE\_DRIVER\_VERSION\_MISMATCH。在至少一个实施例中,基于cuFileDriverOpen调用,API打开与内核驱动程序的会话以从用户空间向内核空间进行通信。

[0075] 在至少一个实施例中,API至少部分地基于接收到的读取函数调用来接收读取函数调用(例如,cuFileRead)并执行动态数据传输路由。在至少一个实施例中,cuFileRead被指定为:ssize\_t cuFileRead(CUFileHandle fh,void\*devPtr,size\_t size,off\_t offset);其中fh是文件的文件描述符,devPtr是要读取的设备指针的起始地址,size是要读取的字节大小,offset是要从中读取的文件中的偏移量。在至少一个实施例中,基于cuFileRead调用,API使用动态数据传输路由将指定字节从文件描述符读入设备存储器中。在至少一个实施例中,与cuMemCpy的非异步版本不同,cuFileRead不具有对空流中的其他工作进行排序的语义。在至少一个实施例中,基于cuFileRead调用,API返回成功读取的字节大小,错误返回-1,在这种情况下,错误号(errno)设置为指示文件系统错误,而所有其他情况返回负整数值CUfileOpError枚举值。在至少一个实施例中,基于cuFileRead调用,API以指定的偏移量和大小字节从指定的文件句柄读取数据到GPU存储器中。在至少一个实施例中,API使用动态数据传输路由将数据读入GPU存储器。在至少一个实施例中,用于cuFileRead的API对于未对齐的偏移量和任何数据大小均可正常工作。在至少一个实施例中,cuFileRead是同步调用,并且将阻塞直到IO完成。

[0076] 在至少一个实施例中,API接收写入函数调用(例如,cuFileWrite),并且至少部分地基于接收到的写入函数调用来执行动态数据传输路由。在至少一个实施例中,

cuFileWrite被指定为:ssize\_t cuFileWrite(CUFileHandle fh,void\*devPtr,size\_t size,off\_t offset);其中fh是文件的文件描述符,devPtr是要读取的设备指针的起始地址,size是要写入的字节大小,offset是要写入的文件中的偏移量。在至少一个实施例中,基于cuFileWrite调用,API使用动态数据传输路由将来自设备存储器的指定字节写入文件描述符。在至少一个实施例中,与cuMemCpy的非异步版本不同,cuFileRead不具有对空流中的其他工作进行排序的语义。在至少一个实施例中,基于cuFileWrite调用,API返回成功写入的字节大小,错误时返回-1,在这种情况下,将errno设置为指示文件系统错误,而所有其他错误将返回负整数CUfileOpError枚举值。在至少一个实施例中,基于cuFileWrite调用,API使用动态数据传输路由以指定的偏移量和大小字节将来自指定文件句柄的数据写入GPU存储器。在至少一个实施例中,用于cuFileWrite的API对于未对齐的偏移量和数据大小可以正常工作。在至少一个实施例中,cuFileWrite是同步调用,并且将阻塞直到IO完成。

[0077] 在至少一个实施例中,API响应于接收到某种其他类型的函数调用,诸如数据流函数调用或数据批量传输函数调用,来执行动态数据传输路由。在至少一个实施例中,API响应于数据批量传输函数调用来确定多个数据传输路径。

[0078] 在至少一个实施例中,API(例如,在第一CPU 202或第二CPU 204上运行)从应用程序接收函数调用,以从第一GPU 228(例如,从源存储器)传输数据集(例如,写调用)到第二组存储设备297。在至少一个实施例中,API将事先为计算机系统200和已标识的可用块设备生成至少一个PCIe设备层次结构树。在至少一个实施例中,API还确定一个或多个设备和/或路径特性,例如用于数据传输的一个或多个延迟度量。在至少一个实施例中,API为来自应用程序的函数调用中指定的数据集确定数据传输路径。在至少一个实施例中,API确定数据传输路径是从GPU 228到GPU 258中的反弹缓冲器(例如,经由通信路径292),随后是经由第十二PCIe交换机274,第四PCIe交换机212,接口295和第二存储控制器294从GPU 258中的反弹缓冲器传输到第二存储设备297。在至少一个实施例中,API至少部分地基于一个或多个生成的PCIe设备层次结构树,可用的块设备、设备特性和/或路径特性来确定数据传输路径。

[0079] 在至少一个实施例中,API至少部分地基于以下项经由通信路径292确定数据传输路径:确定链路203的拥塞级别高于预定阈值,确定第一域214中的一个或多个PCIe链路的拥塞级别高于预定阈值,确定通信路径292的带宽高于穿过链路203的路径的带宽,确定通信路径292的拥塞级别低于预定阈值,和/或确定通信路径292避免了一种或更多种结构模式成本或功耗级别。在至少一个实施例中,API至少部分地基于带宽解耦来确定反弹缓冲器,带宽解耦允许通信路径292比替代路径更快和/或在静态缓冲器上的可能竞争。在至少一个实施例中,API使用在用户分配的固定缓冲器中的传递。在至少一个实施例中,API指示复制引擎(例如,在GPU258中)以避免反弹缓冲器和/或到达其他存储器。在至少一个实施例中,API跨OS实例确定在一个或多个PCIe链路上的数据传输路径,PCIe结构模式(例如,跨PCIe树),UPI,NVLink,CNVLink,GNVLink和/或NVLink。

[0080] 在至少一个实施例中,API执行一种或更多种优化。在至少一个实施例中,API至少部分地基于与延迟和/或带宽有关的优化目标来执行优化。在至少一个实施例中,API引入用于暂存和/或分区的缓存。在至少一个实施例中,API对齐传输。在至少一个实施例中,API执行一种或更多种特殊操作,例如解压缩,压缩,解密或加密。在至少一个实施例中,API引

入了用于管理状态、管理同步或移动数据的附加代理(例如,CPU,SmartNIC)。在至少一个实施例中,API同时使用一个或更多个路径。在至少一个实施例中,由API执行的一个或更多个动作在策略控制下。在至少一个实施例中,策略是不透明的。

[0081] 在至少一个实施例中,一个以上的API至少部分地基于要传输的数据类型或是否要通过网络进行传输中的一种或更多种来确定数据传输路径。在至少一个实施例中,第一API用于确定计算机系统(例如,cuFile)内的通用数据的数据传输路径。在至少一个实施例中,第二API用于确定对象(例如,cuObject)的数据传输路径。在至少一个实施例中,第三API用于确定通过网络(例如,cuRDMA)的数据传输路径。在至少一个实施例中,第二和第三API中的一个或更多个是第一API的变体。在至少一个实施例中,API支持多GPU和多节点通信基元(例如,利用NVSHMEM)。在至少一个实施例中,API支持并行计算平台和应用程序编程接口(例如,OpenCL和/或计算统一设备架构(CUDA))。在至少一个实施例中,API支持CUDA感知统一通信X(UCX)和消息传递接口(MPI)。在至少一个实施例中,API支持同步、异步、无批和批传输。

[0082] 在至少一个实施例中,一个或更多个代理能够接收读/写请求(例如,在用户或内核级别的CPU上,在用户级别的GPU上,在NIC上)。在至少一个实施例中,API使用的存储器,包括配置的存储(例如,PCIe树)可以在CPU,GPU,NIC或任何其他合适的硬件设备中。在至少一个实施例中,CPU(例如,用户或内核),GPU或NIC发起访问。在至少一个实施例中,对存储器或存储的访问可以在代理之间变化,使得沿着各个段的能力允许将在单次传输中不可用的组合。

[0083] 在至少一个实施例中,数据传输路径包括一种或更多种类型的存储,例如文件系统,块系统,对象系统,键-值存储或任何其他合适类型的存储。在至少一个实施例中,数据传输路径涉及一种或更多种类型的介质,例如NVMe,SAS,分布式存储器或任何其他合适类型的介质。在至少一个实施例中,数据传输路径包括在一个或更多个硬件设备(例如,CPU,GPU或任何其他合适的硬件设备)上的高速缓存。在至少一个实施例中,数据传输路径包括一个或更多个处理器读取和写入。在至少一个实施例中,数据传输路径利用一个或更多个DMA引擎。

[0084] 在至少一个实施例中,API或任何其他合适的机制(例如,在CPU,GPU或任何其他合适的硬件设备上运行的数据传输引擎)至少部分地基于预定成本函数来确定数据传输路径。在至少一个实施例中,成本函数针对带宽和延迟中的一个或更多个进行优化(例如,关于瞬时或峰值性能)。在至少一个实施例中,成本函数包括带宽和延迟中的每一个的通用表达式。在至少一个实施例中,成本函数基于连接的表征将对一个或更多个通用表达式的输入用于带宽和/或延迟。在至少一个实施例中,连接的表征是静态的(例如,从文件中读取),半动态的(例如,基于启动时的表征)和/或基于当前使用情况动态地建模。在至少一个实施例中,数据传输路径的反弹缓冲器在GPU中。在至少一个实施例中,将反弹缓冲器一般化为任何其他存储器结构(例如,与GPU,CPU,NIC或任何其他合适的硬件组件相关联的DRAM或SRAM)。在至少一个实施例中,API至少部分地基于基础硬件设备拓扑(例如,至少一个设备层次结构树)的表示,一个或更多个硬件能力,一个或更多个目标,一个或更多硬件设备可用性和/或任何其他合适的特性来确定数据传输路径。

[0085] 图3示出了根据至少一个实施例的确定通过其将数据从计算机系统的第一硬件组

件(例如,计算机系统100、160或200)传输到计算机系统的第二硬件组件的路径的技术300的流程图。在至少一个实施例中,技术300由至少一个电路,至少一个系统,至少一个处理器,至少一个图形处理单元,至少一个并行处理器和/或至少一些在本文描述和/或示出的其他处理器或其组件来执行。在至少一个实施例中,API执行技术300的一个或更多个方面(例如,如参照图1A、图1B和/或图2所描述的)。在至少一个实施例中,第一硬件组件和第二硬件组件中的一个或更多个是GPU。在至少一个实施例中,在技术300的一个或更多个方面中使用的DMA引擎是非GPU设备。在至少一个实施例中,在技术300的一个或更多个方面中使用的DMA引擎是GPU(例如,作为数据传输路径的端点的GPU)。在至少一个实施例中,第一硬件组件和第二硬件组件都不是GPU。

[0086] 在至少一个实施例中,技术300在框302处包括确定计算机系统(例如计算机系统100,计算机系统160,计算机系统200或任何其他合适的计算机系统)的硬件拓扑。在至少一个实施例中,确定硬件拓扑对应于创建关于图4的技术400描述的设备层次结构树。在至少一个实施例中,在框304处,技术300包括确定一组可用块设备。在至少一个实施例中,确定可用块设备的集合对应于如关于图5的技术500所描述的建立一个或更多个块设备层次结构组。

[0087] 在至少一个实施例中,在框306处,技术300包括确定将数据从第一硬件组件传输到第二硬件组件的路径。在至少一个实施例中,在框306处确定传输数据的路径包括参照图7的技术700和/或图6的技术600描述的一个或更多个方面。在至少一个实施例中,在框308处,技术300包括经由确定的路径将数据集从第一硬件组件传输到第二硬件组件。在至少一个实施例中,在框310处,技术300包括执行附加动作。在至少一个实施例中,其他动作包括确定一个或更多个路径特性(例如,延迟度量)以及至少部分地基于所确定的路径特性来更新值。

[0088] 图4示出了根据至少一个实施例的创建设备层次结构树和硬件设备的排名列表的技术400的流程图。在至少一个实施例中,技术400包括建立GPU和NIC设备PCIe组层次结构。在至少一个实施例中,技术400由至少一个电路、至少一个系统、至少一个处理器、至少一个图形处理单元,至少一个并行处理器和/或在本文描述和/或示出的其至少一些其他处理器或组件来执行。在至少一个实施例中,技术400包括在框402处确定已经执行了触发动作。在至少一个实施例中,触发动作包括接收函数调用,例如指定为cuFileDriverOpen的函数调用。

[0089] 在至少一个实施例中,在框404处,技术400包括创建设备层次结构树。在至少一个实施例中,设备层次结构树是PCIe层次结构树。在至少一个实施例中,使用PCIe总线设备功能(BDF)信息创建层次结构树。在至少一个实施例中,根复合体(诸如CPU插槽)是层次结构树的父节点。在至少一个实施例中,PCIe交换机是设备层次结构树的中间子级。在至少一个实施例中,NIC和GPU设备是设备层次结构树的叶节点。

[0090] 在至少一个实施例中,在框406处,技术400包括创建一个或更多个映射。在至少一个实施例中,创建一个或更多个映射包括通过id读取可用于应用程序的所有GPU,以生成被指定为GPUidList的GPU列表。在至少一个实施例中,创建一个或更多个映射包括通过id读取可用于应用程序的所有IB设备,以生成指定为NICidList的列表。在至少一个实施例中,创建一个或更多个映射包括创建一个或更多个哈希映射。在至少一个实施例中,创建一个

或更多个哈希映射包括创建用于GPU的HashMapGPUGroup,其中HashMap用于存储键和值对。在至少一个实施例中,创建一个或更多个哈希映射包括创建用于NIC的HashMap NICGroup。在至少一个实施例中,创建一个或更多个哈希映射包括为用于GPUid→GPUGroup的GPUidMap创建HashMap。在至少一个实施例中,创建一个或更多个哈希映射包括为用于NICid→NICGroup的NICidMap创建HashMap。在至少一个实施例中,创建一个或更多个哈希映射包括为用于GPUid→NICidList的GPUToNICidMap创建HashMap。在至少一个实施例中,创建一个或更多个哈希映射包括为GPUid→GPUGroup创建HashMap。

[0091] 在至少一个实施例中,在框408处,技术400包括至少部分基于设备层次结构树来识别硬件设备。在至少一个实施例中,识别硬件设备包括走遍(walk)或以其他方式遍历PCIe树以找到PCIe树中存在的任何GPU和/或NIC设备。在至少一个实施例中,将GPU组标识符(例如GPU\_GROUP\_ID)设置为用于找到的GPU的PCIe根交换机的BDF。在至少一个实施例中,将NIC组标识符(例如NIC\_GROUP\_ID)设置为用于找到的NIC的PCIe根交换机的BDF。在至少一个实施例中,在框410处,技术400包括将识别出的硬件设备添加到一个或更多个映射。在至少一个实施例中,将标识出的硬件设备添加到一个或更多个映射包括:将与GPU相关联的标识符(例如GPUid)添加到GPU组哈希映射(例如GPUGroup HashMap),其中GPU组标识符(例如GPU\_GROUP\_ID)作为键(key)。在至少一个实施例中,将识别出的硬件设备添加到一个或更多个映射包括:将与NIC相关联的标识符(例如NICid)添加到例如用于NIC设备的NIC组哈希映射中(诸如NICGroup HashMap),其中NIC组标识符(例如NIC\_GROUP\_ID)作为键。

[0092] 在至少一个实施例中,在框412处,技术400包括将一个或更多个组级别映射添加到一个或更多个映射。在至少一个实施例中,将一个或更多个组级别映射添加到一个或更多个映射包括:将GPU组(例如,GPUGroup)的标识符添加到哈希映射(例如,GPUidMap),其中GPU的标识符(例如,GPUid)作为键。在至少一个实施例中,将一个或更多个组级别映射添加到一个或更多个映射包括:将NIC组(例如,NICGroup)的标识符添加到哈希映射(例如,NICidMap),其中NIC的标识符(例如,NICid)作为键。

[0093] 在至少一个实施例中,在判定框414处,技术400包括确定是否已经遍历整个设备层次结构树。在至少一个实施例中,确定是否已经遍历整个设备层次结构树包括:确定是否已经在设备层次结构树中找到了GPU列表(例如,GPUidList)中的每个条目和NIC列表(例如,NICidList)中的每个条目。在至少一个实施例中,如果在判定框414处确定尚未遍历整个设备层次结构树,则技术400返回到框408,其包括基于设备层次结构树来识别附加硬件设备。

[0094] 在至少一个实施例中,如果在判定框414处确定已经遍历整个设备层次结构树,则技术400进行到框416,在框416处,技术400包括创建硬件设备的排名列表。在至少一个实施例中,创建硬件设备的排名列表包括创建NIC的排名列表。在至少一个实施例中,创建NIC的排名列表包括计算与PCIe树中的每个GPU id的距离。在至少一个实施例中,距离是从与GPU标识符相关联的GPU到NIC的跳数。在至少一个实施例中,将排名列表与标识符(例如,NICidList)相关联地存储。在至少一个实施例中,在框416处,技术400还包括以GPU标识符(例如,GPUid)作为键将排名列表(例如,NICidList)添加到哈希映射(例如,GPUToNICMap)。在至少一个实施例中,在框418处,技术400包括执行附加动作。

[0095] 图5示出了根据至少一个实施例的构建一个或更多个块设备层次结构组的技术

500的流程图。在至少一个实施例中,技术500包括至少部分地基于文件描述信息(例如,指定为Fd的文件描述符)来构建一个或多个块设备PCIe层次结构组。在至少一个实施例中,技术500由至少一个电路,至少一个系统、至少一个处理器、至少一个图形处理单元、至少一个并行处理器和/或在本文描述和/或示出的至少一些其他处理器或其组件来执行。在至少一个实施例中,技术500包括在框502处确定已经执行了触发动作。在至少一个实施例中,触发动作包括接收函数调用,例如指定为cuFileImportExternalFile(FH,descr)的函数调用。在至少一个实施例中,触发动作是与操作系统(OS)无关的(例如,支持Linux和Windows OS)。在至少一个实施例中,触发动作至少部分地基于描述信息(例如,descr)和文件句柄(例如,FH)中的一个或多个。

[0096] 在至少一个实施例中,在框504处,技术500包括识别文件系统、块设备和文件描述符信息。在至少一个实施例中,识别文件描述符信息包括从描述信息(例如descr)获得文件描述符(例如Fd)。在至少一个实施例中,识别文件系统信息包括从函数调用(例如stat())获得文件系统类型(例如FS)。在至少一个实施例中,识别块设备信息包括从函数调用(例如stat())获得块设备(例如bd)。在至少一个实施例中,识别块设备信息包括确定是否存在RAID分区,并且如果存在,则找到基础子块设备(例如,bd)。在至少一个实施例中,在框504处,技术500还包括创建数据结构(例如,fdinfo结构)以存储所识别出的文件描述符(例如,Fd)信息。

[0097] 在至少一个实施例中,在判定框506,技术500包括确定块设备(例如,bd)是否在文件系统组(例如,FSGroup)中。在至少一个实施例中,如果在判定框506处确定块设备不在文件系统组中,则在框508处,技术500包括确定用于块设备的组标识符。在至少一个实施例中,确定用于块设备的组标识符包括:通过扫描用于块设备的PCIe BDF信息并确定用于块设备的根PCIe交换机来获得PCIe路径层次结构。在至少一个实施例中,将用于块设备的组标识符(例如,GROUP\_ID)设置为用于块设备的PCIe根交换机的BDF。

[0098] 在至少一个实施例中,在框510处,技术500包括将组标识符和块设备添加到一个或多个映射。在至少一个实施例中,将组标识符和块设备添加到一个或多个映射包括:创建文件系统组哈希映射(例如,HashMapFSGroup),如果尚不存在的话。在至少一个实施例中,将组标识符和块设备添加到一个或多个映射包括:将块设备标识符(例如,blockdev(bd))添加到用于块设备的哈希映射(例如,HashMap)中,其中块设备标识符(例如,blockdev(bd))作为键,并且组标识符(例如,GROUP\_ID)作为值。在至少一个实施例中,将组标识符和块设备添加到一个或多个映射包括:更新被创建以存储文件描述符信息(例如,fdinfo)的数据结构的元素。在至少一个实施例中,数据结构(例如,fdinfo)包括文件系统组(例如,FSGroup)和子组(例如,SubGroup(bd))数据元素。在至少一个实施例中,更新元素包括:更新将被设置为组标识符(例如,GROUP\_ID)的文件系统组和子组数据元素(如果尚未设置),例如通过设置fdinfo→FSGroup=GROUP\_ID,以及通过设置fdinfo→SubGroup(bd)=GROUP\_ID。在至少一个实施例中,技术500还包括:在框510处,将跨域指示符设置为“真”或“假”。在至少一个实施例中,如果文件描述符数据结构的子组元素与文件描述符数据结构的文件系统元素不同(例如,如果fdinfo→SubGroup(bd)与fdinfo→FSGroup不同),则将跨域指示符设置为“真”。在至少一个实施例中,跨域指示符作为元素存储在文件描述符数据结构中(例如,fdinfo→Cross\_domain=TRUE)。在至少一个实施例中,如果子组元素和文

件系统元素被设置为相同的值,则跨域指示符被设置为“假”。在至少一个实施例中,如果在判定框506处确定块设备在文件系统组中,则技术跳过框508并前进至框510。

[0099] 在至少一个实施例中,在判定框512处,技术500包括确定是否已处理所有块设备。在至少一个实施例中,在判定框512,如果确定尚未处理所有块设备,技术500返回到判定框506。在至少一个实施例中,如果在判定框512,确定已处理所有块设备,则技术500进行到框514,其包括创建键文件句柄并使用键文件句柄更新映射。在至少一个实施例中,至少部分地基于文件描述符来创建键文件句柄(例如,从(fd)创建键FH)。在至少一个实施例中,用键文件句柄更新映射包括:将文件描述符数据结构添加到具有键文件句柄的哈希映射,并返回文件句柄作为指针(例如,通过使用键FH将fdinfo添加到FdHashMap并返回FH作为指针)。在至少一个实施例中,在框516处,技术500包括执行其他动作。

[0100] 图6示出了根据至少一个实施例的目标设备缓冲器管理的技术600的流程图。在至少一个实施例中,技术600由至少一个电路、至少一个系统、至少一个处理器、至少一个图形处理单元、至少一个并行处理器和/或在本文描述和/或示出的至少一些其他处理器或其组件来执行。在至少一个实施例中,技术600包括:在框602处,至少部分地基于缓冲器标识符确定已经执行了触发动作。在至少一个实施例中,触发动作包括接收函数调用,例如被指定为cuFileBufRegister(buf,size)的函数调用,其中例如buf是缓冲器标识符。在至少一个实施例中,缓冲器标识符指代目标设备缓冲器(例如,在目标存储器152中)。

[0101] 在至少一个实施例中,在框604,技术600包括识别与缓冲器标识符(例如,buf)有关的缓冲器信息,并创建缓冲器跟踪条目。在至少一个实施例中,识别缓冲器信息包括获得与缓冲器相关联的一个或多个属性(例如,通过经由诸如OpenCL或CUDA API之类的API发送命令)。在至少一个实施例中,识别缓冲器信息包括获得用于缓冲器的GPU标识符(例如,GPUid)。在至少一个实施例中,识别缓冲器信息包括获得缓冲器的缓冲器类型(例如,buf\_type)。在至少一个实施例中,创建缓冲器跟踪条目包括在数据结构中创建用于跟踪缓冲器的条目(例如,创建用于跟踪buf的nvinfo条目)。在至少一个实施例中,技术600还包括:在框604处,在数据结构中分配GPU组(例如,使用GPUidMap中的GPUid从查找中分配nvinfo→GPU\_Group)。

[0102] 在至少一个实施例中,在判定框606处,技术600包括确定缓冲器是否允许直接存储器访问。在至少一个实施例中,如果在判定框606处,确定缓冲器不允许直接存储器访问,技术600包括:在框608处,更新缓冲器跟踪条目。在至少一个实施例中,更新缓冲器跟踪条目包括:将指示符设置为指示不能使用缓冲器的值(例如,通过设置nvinfo→GPU\_Group=-1)。在至少一个实施例中,如果在判定框606处,确定缓冲器允许直接存储器访问,技术600包括:在判定框610,确定是否有足够的空间可用。在至少一个实施例中,确定是否有足够的空间可用包括:确定是否有足够的基地址寄存器(BAR)1(BAR1)空间可用。在至少一个实施例中,如果在判定框610处,确定没有足够的空间可用,则技术600前进到在框608处更新缓冲器跟踪条目。

[0103] 在至少一个实施例中,如果在判定框610处确定有足够的空间可用,则技术600包括在框612处映射缓冲器。在至少一个实施例中,映射缓冲器包括映射与缓冲器标识符(例如buf)相关联的缓冲器。在至少一个实施例中,映射缓冲器包括通过向GPU驱动器发送命令(例如,通过向nvidia-fs驱动器发送MAP命令)来映射GPU缓冲器。在至少一个实施例中,在

判定框614处,技术600包括确定缓冲器映射是否成功。在至少一个实施例中,如果在判定框614处确定映射不成功,则技术600前进到在框608处更新缓冲器跟踪条目。在至少一个实施例中,如果在判定框614处确定映射成功,则技术600包括在框616处将缓冲器跟踪条目添加到映射,其中以缓冲器标识符(例如,buf)作为键。在至少一个实施例中,将缓冲器跟踪条目添加到映射包括:将条目添加哈希映射(例如,通过将nvinfo添加到HashMap NvHashMap中,其中以buf作为键)。在至少一个实施例中,在框618处,技术600包括执行其他动作。

[0104] 图7示出了根据至少一个实施例的确定数据传输路径的技术700的流程图。在至少一个实施例中,技术700由至少一个电路、至少一个系统、至少一个处理器、至少一个图形处理单元、至少一个并行处理器和/或在本文描述和/或示出的至少一些其他处理器或其组件来执行。在至少一个实施例中,技术700包括:在框702处,至少部分地基于与数据集相关联的信息确定已经执行了触发动作。在至少一个实施例中,与数据集相关联的信息包括文件句柄、缓冲器标识符、数据大小和/或偏移指示符中的至少一个。在至少一个实施例中,触发动作包括接收函数调用,例如指定为cuFileRead(fh,buf,size,offset)的函数调用,其中例如fh是文件句柄,而buf是缓冲器标识符。在至少一个实施例中,触发动作不是显式的API函数调用。在至少一个实施例中,触发动作是提示(例如,cudaMemPrefetchAsync或cudaMemAdvise)。在至少一个实施例中,技术700的一个或更多个方面经由故障隐式地进行。在至少一个实施例中,在文件被存储器映射之后,数据通过故障隐式地移动。在至少一个实施例中,第一提示设置策略(例如,cudaMemAdvise)和第二提示(例如,cudaMemPrefetchAsync)触发显式移动。在至少一个实施例中,第一提示设置策略,第二提示在经由访问请求故障之前触发显式移动。

[0105] 在至少一个实施例中,在框704处,技术700包括至少部分地基于与数据集相关联的信息来获得附加信息。在至少一个实施例中,获得附加信息包括执行一个或更多个查找操作,例如一个或更多个哈希查找操作(例如,Fdhash(Fh)中的Hash Lookup fdinfo和来自Nvhash(buf)的HashLookup nvinfo)。在至少一个实施例中,技术700还包括:在框704处,至少部分地基于获得的附加信息来设置一个或更多个值。在至少一个实施例中,设置一个或更多个值包括设置文件系统组、GPU组和GPU标识符中的一个或更多个(例如,通过设置:FSGroup=fdinfo→pci\_group;GPUGroup=nvinfo→pci\_group;以及GPUid=nvinfo→GPUid)。

[0106] 在至少一个实施例中,在判定框706处,技术700包括确定GPU组(例如GPUGroup)是否与文件系统组(例如FSGroup)相同。在至少一个实施例中,如果在判定框706处确定GPU组和文件系统组不同,则技术700包括:在判定框708处,确定数据集的传输是否涉及跨域路径(例如,基于fdinfo→is\_cross\_domain)。

[0107] 在至少一个实施例中,如果在判定框708处确定传输涉及跨域路径,则技术700包括:在框710处,选择GPU子组。在至少一个实施例中,在框710处选择GPU子组包括:至少部分地基于文件描述符信息来选择GPU子组(例如,在fdinfoinfo→Subgroup中选择GPU子组)。在至少一个实施例中,在框712处,技术700包括选择GPU。在至少一个实施例中,通过从具有可用的反弹缓冲器的一组GPU中选择GPU来执行选择GPU。在至少一个实施例中,根据一个或更多个预定标准,通过选择具有比其他可用GPU更好的一个或更多个关联特性的GPU,来执行选择GPU(例如,选择在FSGroup中具有可用反弹缓冲器的GPU,该GPU与其他可用GPU相比,

具有倒数第二低的I/O平均延迟和排名。

[0108] 在至少一个实施例中,在框714处,技术700包括选择缓冲器。在至少一个实施例中,选择缓冲器包括:选择最接近GPUid中请求的大小的第一自由反弹缓冲器。在至少一个实施例中,选择缓冲器包括选择大于或等于GPUid中所请求的大小的第一自由反弹缓冲器。在至少一个实施例中,选择缓冲器包括在GPU存储器中选择反弹缓冲器。在至少一个实施例中,选择缓冲器包括为兼容模式在同一NUMA节点中的系统存储器中选择反弹缓冲器。在至少一个实施例中,在判定框716处,技术700包括确定是否已经处理了所有GPU子组。如果在判定框716处确定尚未处理所有GPU子组,则技术700返回到框710以选择附加GPU子组。

[0109] 在至少一个实施例中,如果在判定框708处确定传输不涉及跨域路径,则技术700在框712处包括选择GPU。在至少一个实施例中,如果在判定框708处确定传输不涉及跨域路径,则技术700通过框712和框714的动作进行一次,并且确定所有GPU子组已经在判定框716处处理。

[0110] 在至少一个实施例中,在框718处,技术700包括执行数据传输。在至少一个实施例中,执行数据传输包括:经由在框714选择的反弹缓冲器来执行数据传输。在至少一个实施例中,执行数据传输包括一个或更多个P2P DMA传输。在至少一个实施例中,执行数据传输包括一个或更多个复制操作。在至少一个实施例中,对于读取操作(例如,读取函数调用作为框702处的触发动作),执行数据传输包括执行P2P DMA直接从硬件设备端点(例如,存储或NIC)传输到反弹缓冲器,然后是从反弹缓冲器到目标存储器的设备到设备复制操作(例如,在GPU或其他硬件设备端点中)。在至少一个实施例中,对于写操作(例如,在框702处的写入函数调用作为触发动作),执行数据传输包括:执行从源存储器(例如,在GPU或其他硬件设备端点中)到反弹缓冲器的设备到设备复制操作,然后是从反弹缓冲器到硬件设备端点(例如,存储或NIC)的P2P DMA传输。在至少一个实施例中,在不可能进行P2P DMA传输的某些情况下(例如,由于一个或更多个硬件设备不支持这种能力),执行数据传输包括:执行POSIX读取操作或POSIX写入操作,代替P2P DMA传输操作,这取决于框702处处的触发动作的类型。在至少一个实施例中,在框718处,技术700包括:至少部分地基于与由GPU标识符标识的GPU相关联的BDF信息,来选择用于数据传输路径的NIC(例如,将BDF从GPUid设置为nvidia-fs驱动程序,以帮助从NICGroup选择NIC)。

[0111] 在至少一个实施例中,在框722处,技术700包括更新信息。在至少一个实施例中,更新信息包括:将反弹缓冲器返回到反弹缓冲器池(例如,将bbuf返回到用于GPUid的池)。在至少一个实施例中,更新信息包括:更新一个或更多个性能度量(例如,诸如倒数第二平均I/O延迟的延迟度量)。

[0112] 在至少一个实施例中,如果在判定框706处确定GPU组和文件系统组相同,则技术700包括:在判定框724处,确定存储器是否对齐。在至少一个实施例中,确定存储器是否对齐包括:确定偏移量是否对齐到预定大小(例如4096字节)。在至少一个实施例中,确定存储器是否对齐包括:确定目标存储器缓冲器是否对齐。在至少一个实施例中,技术700还包括:在判定框706处,确定目标存储器缓冲器是否为预定存储器类型。在至少一个实施例中,确定目标存储器缓冲器是否为预定存储器类型包括:确定目标存储器缓冲器是否支持DMA。

[0113] 在至少一个实施例中,如果在判定框724处确定存储器未对齐,则技术700前进到在框714处选择缓冲器。在至少一个实施例中,如果在判定框724处确定存储器是对齐的,技

术700包括:在步骤726处,执行数据传输。在至少一个实施例中,在框726处执行数据传输包括:在不使用反弹缓冲器的情况下执行数据的DMA传输(例如,通过直接从诸如存储或NIC之类的硬件端点执行到目标存储器缓冲器的DMA传输以进行读取操作)。在至少一个实施例中,在框726处执行数据传输包括:选择NIC并且经由NIC执行数据传输(例如,通过将BDF从GPUid设置为nvidia-fs驱动器,以帮助从NICGroup和NICidList选择NIC)。在至少一个实施例中,在框728处,技术700包括执行附加动作,诸如监视下一触发动作和/或确定一个或更多个硬件设备或路径特性。

#### [0114] 数据中心

[0115] 图8示出了根据至少一个实施例的示例数据中心800。在至少一个实施例中,数据中心800包括但不限于数据中心基础设施层810、框架层820、软件层830和应用层840。

[0116] 在至少一个实施例中,如图8所示,数据中心基础设施层810可以包括资源协调器812、分组的计算资源814和节点计算资源(“节点C.R.”)816(1)-816(N),其中“N”代表任何完整的正整数。在至少一个实施例中,节点C.R.816(1)-816(N)可以包括但不限于任意数量的中央处理单元(“CPU”)或其他处理器(包括加速器、现场可编程门数组(“FPGA”)、图形处理器等),存储器设备(例如动态只读存储器),存储设备(例如固态硬盘或磁盘驱动器),网络输入/输出(“NW I/O”)设备,网络交换机,虚拟机(“VM”),电源模块和冷却模块等。在至少一个实施例中,节点C.R.816(1)-816(N)中的一个或更多个节点C.R.可以是具有一个或更多个上述计算资源的服务器。

[0117] 在至少一个实施例中,分组的计算资源814可以包括容纳在一个或更多个机架内的节点C.R.的单独分组(未示出),或者容纳在各个地理位置的数据中心内的许多机架(也未示出)。分组的计算资源814内的节点C.R.的单独分组可以包括可以被配置或分配为支持一个或更多个工作负载的分组的计算、网络、存储器或存储资源。在至少一个实施例中,可以将包括CPU或处理器的几个节点C.R.分组在一个或更多个机架内,以提供计算资源来支持一个或更多个工作负载。在至少一个实施例中,一个或更多个机架还可以包括任意数量的电源模块、冷却模块和网络交换机,以任意组合。

[0118] 在至少一个实施例中,资源协调器812可以配置或以其他方式控制一个或更多个节点C.R.816(1)-816(N)和/或分组的计算资源814。在至少一个实施例中,资源协调器812可以包括用于数据中心800的软件设计基础结构(“SDI”)管理实体。在至少一个实施例中,资源协调器812可以包括硬件、软件或其某种组合。

[0119] 在至少一个实施例中,如图8所示,框架层820包括但不限于作业调度器832、配置管理器834、资源管理器836和分布式文件系统838。在至少一个实施例中,框架层820可以包括支持软件层830的软件852和/或应用程序层840的一个或更多个应用程序842的框架。在至少一个实施例中,软件852或应用程序842可以分别包括基于Web的服务软件或应用程序,例如由Amazon Web Services,Google Cloud和Microsoft Azure提供的服务或应用程序。在至少一个实施例中,框架层820可以是但不限于一种免费和开放源软件网络应用框架,例如可以利用分布式文件系统838来进行大范围数据处理(例如“大数据”)的Apache Spark™(以下称为“Spark”)。在至少一个实施例中,作业调度器832可以包括Spark驱动器,以促进对数据中心800的各个层所支持的工作负载进行调度。在至少一个实施例中,配置管理器834可以能够配置不同的层,例如软件层830,和用于支持大规模数据处理的框架层820,其

包括Spark和分布式文件系统838。在至少一个实施例中,资源管理器836能够管理映射到或分配用于支持分布式文件系统838和作业调度器832的集群或分组计算资源。在至少一个实施例中,集群或分组计算资源可以包括数据中心基础设施层810上的分组的计算资源814。在至少一个实施例中,资源管理器836可以与资源协调器812协调以管理这些映射的或分配的计算资源。

[0120] 在至少一个实施例中,包括在软件层830中的软件852可以包括由节点C.R.816(1)-816(N)的至少一部分,分组计算资源814和/或框架层820的分布式文件系统838使用的软件。一种或更多种类型的软件可以包括但不限于Internet网页搜索软件、电子邮件病毒扫描软件、数据库软件和流视频内容软件。

[0121] 在至少一个实施例中,应用层840中包括的一个或更多个应用程序842可以包括由节点C.R.816(1)-816(N)的至少一部分、分组的计算资源814和/或框架层820的分布式文件系统838使用的一种或更多种类型的应用程序。一种或更多种类型的应用程序可以包括但不限于CUDA应用程序。

[0122] 在至少一个实施例中,配置管理器834、资源管理器836和资源协调器812中的任何一个可以基于以任何技术上可行的方式获取的任意数量和类型的数据来实现任意数量和类型的自我修改动作。在至少一个实施例中,自我修改动作可以减轻数据中心800的数据中心操作员做出可能不好的配置决定并且可以避免数据中心的未充分利用和/或执行差的部分。

[0123] 基于计算机的系统

[0124] 以下各图提出但不限于可用于实现至少一个实施例的示例性的基于计算机的系统。在至少一个实施例中,以下附图的一个或更多个基于计算机的系统可以实现参照图1A描述的实施例、参照图1B描述的实施例、参照图2描述的实施例、参照图3描述的技术300、参照图4描述的技术400、参考图5描述的技术500、参照图6描述的技术600和/或参照图7描述的技术700的一个或更多个方面。

[0125] 图9示出了根据至少一个实施例的处理系统900。在至少一个实施例中,系统900包括一个或更多个处理器902和一个或更多个图形处理器908,并且可以是单处理器台式机系统、多处理器工作站系统或具有大量处理器902或处理器核心907的服务器系统。在至少一个实施例中,处理系统900是结合在片上系统(SoC)集成电路内的处理平台,以用于移动、手持或嵌入式设备。

[0126] 在至少一个实施例中,处理系统900可以包括或结合在基于服务器的游戏平台中,包括游戏和媒体控制台的游戏控制台、移动游戏控制台、手持游戏控制台或在线游戏控制台。在至少一个实施例中,处理系统900是移动电话、智能电话、平板计算设备或移动互联网设备。在至少一个实施例中,处理系统900还可包括与可穿戴设备耦合或集成在可穿戴设备中,例如智能手表可穿戴设备、智能眼镜设备、增强现实设备或虚拟现实设备。在至少一个实施例中,处理系统900是电视或机顶盒设备,其具有一个或更多个处理器902以及由一个或更多个图形处理器908生成的图形界面。

[0127] 在至少一个实施例中,一个或更多个处理器902每个包括一个或更多个处理器核心907,以处理指令,该指令在被执行时执行针对系统和用户软件的操作。在至少一个实施例中,一个或更多个处理器核心907中的每一个被配置为处理特定指令集909。在至少一个

实施例中,指令集909可以促进复杂指令集计算(CISC)、精简指令集计算(RISC),或通过超长指令字(VLIW)进行计算。在至少一个实施例中,多个处理器核心907可以各自处理不同的指令集909,该指令集909可以包括有助于仿真其他指令集的指令。在至少一个实施例中,处理器核心907还可以包括其他处理设备,例如数字信号处理器(DSP)。

[0128] 在至少一个实施例中,处理器902包括高速缓存存储器(cache)904。在至少一个实施例中,处理器902可以具有单个内部高速缓存或多个级别的内部高速缓存。在至少一个实施例中,高速缓存存储器在处理器902的各个组件之间共享。在至少一个实施例中,处理器902还使用外部高速缓存(例如,三级(L3)高速缓存或最后一级高速缓存(LLC))(未示出),其可以使用已知的高速缓存一致性技术在处理器核心907之间共享该逻辑。在至少一个实施例中,处理器902中另外包括寄存器文件906,处理器902可以包括用于存储不同类型的数据的不同类型的寄存器(例如,整数寄存器、浮点寄存器、状态寄存器和指令指针寄存器)。在至少一个实施例中,寄存器文件906可以包括通用寄存器或其他寄存器。

[0129] 在至少一个实施例中,一个或更多个处理器902与一个或更多个接口总线910耦合,以在处理器902与系统900中的其他组件之间传输通信信号,例如地址、数据或控制信号。在至少一个实施例中,接口总线910在一个实施例中可以是处理器总线,例如直接媒体接口(DMI)总线的版本。在至少一个实施例中,接口总线910不限于DMI总线,并且可以包括一个或更多个外围组件互连总线(例如,PCI,PCI Express)、存储器总线或其他类型的接口总线。在至少一个实施例中,处理器902包括集成存储器控制器916和平台控制器集线器930。在至少一个实施例中,存储器控制器916促进存储设备与处理系统900的其他组件之间的通信,而平台控制器集线器(PCH)930通过本地I/O总线提供到输入/输出(I/O)设备的连接。

[0130] 在至少一个实施例中,存储设备920可以是动态随机存取存储器(DRAM)设备、静态随机存取存储器(SRAM)设备、闪存设备、相变存储设备或具有适当的性能以用作处理器存储器。在至少一个实施例中,存储设备920可以用作处理系统900的系统存储器,以存储数据922和指令921,以在一个或更多个处理器902执行应用或过程时使用。在至少一个实施例中,存储器控制器916还与可选的外部图形处理器912耦合,其可以与处理器902中的一个或更多个图形处理器908通信以执行图和媒体操作。在至少一个实施例中,显示设备911可以连接至处理器902。在至少一个实施例中,显示设备911可以包括内部显示设备中的一个或更多个,例如在移动电子设备或便携式计算机设备或通过显示器接口(例如显示端口(DisplayPort)等)连接的外部显示设备。在至少一个实施例中,显示设备911可以包括头戴式显示器(HMD),诸如用于虚拟现实(VR)应用或增强现实(AR)应用中的立体显示设备。

[0131] 在至少一个实施例中,平台控制器集线器930使外围设备能够通过高速I/O总线连接到存储设备920和处理器902。在至少一个实施例中,I/O外围设备包括但不限于音频控制器946、网络控制器934、固件接口928、无线收发器926、触摸传感器925、数据存储设备924(例如,硬盘驱动器、闪存等)。在至少一个实施例中,数据存储设备924可以经由存储器接口(例如,SATA)或经由外围总线来连接,诸如外围组件互连总线(例如,PCI、PCIe)。在至少一个实施例中,触摸传感器925可以包括触摸屏传感器、压力传感器或指纹传感器。在至少一个实施例中,无线收发器926可以是Wi-Fi收发器、蓝牙收发器或移动网络收发器,诸如3G、4G或长期演进(LTE)收发器。在至少一个实施例中,固件接口928使能与系统固件的通信,并

且可以是例如统一的可扩展固件接口 (UEFI)。在至少一个实施例中,网络控制器934可以启用到有线网络的网络连接。在至少一个实施例中,高性能网络控制器(未示出)与接口总线910耦合。在至少一个实施例中,音频控制器946是多通道高清晰度音频控制器。在至少一个实施例中,处理系统900包括可选的传统(legacy) I/O控制器940,用于将遗留(例如,个人系统2 (PS/2)) 设备耦合到处理系统900。在至少一个实施例中,平台控制器集线器930还可以连接到一个或更多个通用串行总线 (USB) 控制器942,该控制器连接输入设备,诸如键盘和鼠标943组合、相机944或其他USB输入设备。

[0132] 在至少一个实施例中,存储器控制器916和平台控制器集线器930的实例可以集成到离散的外部图形处理器中,例如外部图形处理器912。在至少一个实施例中,平台控制器集线器930和/或存储控制器916可以在一个或更多个处理器902的外部。例如,在至少一个实施例中,处理系统900可以包括外部存储控制器916和平台控制器集线器930,其可以配置成在与处理器902通信的系统芯片组中的存储器控制器集线器和外围控制器集线器。

[0133] 图10示出了根据至少一个实施例的计算机系统1000。在至少一个实施例中,计算机系统1000可以是具有互连的设备和组件, SOC, 或某种组合的系统。在至少一个实施例中,计算机系统1000由处理器1002形成,该处理器1002可以包括用于执行指令的执行单元。在至少一个实施例中,计算机系统1000可以包括但不限于组件,例如处理器1002,其采用包括逻辑的执行单元以执行用于过程数据的算法。在至少一个实施例中,计算机系统1000可以包括处理器,例如可从加利福尼亚圣塔克拉拉的英特尔公司 (Intel Corporation of Santa Clara, California) 获得的PENTIUM®处理器家族、Xeon™、Itanium®、XScale™和/或StrongARM™, Intel® Core™或 Intel® Nervana™微处理器,尽管也可以使用其他系统(包括具有其他微处理器的PC、工程工作站、机顶盒等)。在至少一个实施例中,计算机系统1000可以执行可从华盛顿州雷蒙德的微软公司 (Microsoft Corporation of Redmond, Wash.) 获得的WINDOWS操作系统版本,尽管其他操作系统(例如UNIX和Linux)、嵌入式软件和/或图形用户界面也可以使用。

[0134] 在至少一个实施例中,计算机系统1000可以用在其他设备中,例如手持设备和嵌入式应用。手持设备的一些示例包括蜂窝电话、互联网协议 (Internet Protocol) 设备、数码相机、个人数字助理 (“PDA”) 和手持PC。在至少一个实施例中,嵌入式应用可以包括微控制器、数字信号处理器 (“DSP”)、SoC、网络计算机 (“NetPC”)、机顶盒、网络集线器、广域网 (“WAN”) 交换机,或根据至少一个实施例可以执行一个或更多个指令的任何其他系统。

[0135] 在至少一个实施例中,计算机系统1000可包括但不限于处理器1002,该处理器1002可包括但不限于一个或更多个执行单元1008,其可以配置为执行计算统一设备架构 (“CUDA”) (CUDA®由加利福尼亚州圣克拉拉的NVIDIA Corporation开发) 程序。在至少一个实施例中,CUDA程序是用CUDA编程语言编写的软件应用程序的至少一部分。在至少一个实施例中,计算机系统1000是单处理器台式机或服务器系统。在至少一个实施例中,计算机系统1000可以是多处理器系统。在至少一个实施例中,处理器1002可以包括但不限于CISC微处理器、RISC微处理器、VLIW微处理器、实现指令集组合的处理器,或任何其他处理器设备,例如数字信号处理器。在至少一个实施例中,处理器1002可以耦合到处理器总线1010,该处理器总线1010可以在处理器1002与计算机系统1000中的其他组件之间传输数据

信号。

[0136] 在至少一个实施例中,处理器1002可以包括但不限于1级(“L1”)内部高速缓存存储器(“cache”)1004。在至少一个实施例中,处理器1002可以具有单个内部高速缓存或多级内部缓存。在至少一个实施例中,高速缓存存储器可以驻留在处理器1002的外部。在至少一个实施例中,处理器1002可以包括内部和外部高速缓存的组合。在至少一个实施例中,寄存器文件1006可以在各种寄存器中存储不同类型的数据,包括但不限于整数寄存器、浮点寄存器、状态寄存器和指令指针寄存器。

[0137] 在至少一个实施例中,包括但不限于执行整数和浮点运算的逻辑的执行单元1008,其也位于处理器1002中。处理器1002还可以包括微码(“ucode”)只读存储器(“ROM”),用于存储某些宏指令的微代码。在至少一个实施例中,执行单元1008可以包括用于处理封装指令集1009的逻辑。在至少一个实施例中,通过将封装指令集1009包括在通用处理器1002的指令集中,以及要执行指令的相关电路,可以使用通用处理器1002中的封装数据来执行许多多媒体应用程序使用的操作。在至少一个实施例中,可以通过使用处理器的数据总线的全宽度来在封装的数据上执行操作来加速和更有效地执行许多多媒体应用程序,这可能不需要在处理器的数据总线上传输较小的数据单元来一次对一个数据元素执行一个或更多个操作。

[0138] 在至少一个实施例中,执行单元1008也可以用在微控制器、嵌入式处理器、图形设备、DSP和其他类型的逻辑电路中。在至少一个实施例中,计算机系统1000可以包括但不限于存储器1020。在至少一个实施例中,存储器1020可以被实现为DRAM设备、SRAM设备、闪存设备或其他存储设备。存储器1020可以存储由处理器1002可以执行的由数据信号表示的指令1019和/或数据1021。

[0139] 在至少一个实施例中,系统逻辑芯片可以耦合到处理器总线1010和存储器1020。在至少一个实施例中,系统逻辑芯片可以包括但不限于存储器控制器集线器(“MCH”)1016,并且处理器1002可以经由处理器总线1010与MCH 1016通信。在至少一个实施例中,MCH 1016可以提供到存储器1020的高带宽存储器路径1018以用于指令和数据存储以及用于图形命令、数据和纹理的存储。在至少一个实施例中,MCH 1016可以在处理器1002、存储器1020和计算机系统1000中的其他组件之间启动数据信号,并且在处理器总线1010、存储器1020和系统I/O 1022之间桥接数据信号。在至少一个实施例中,系统逻辑芯片可以提供用于耦合到图形控制器的图形端口。在至少一个实施例中,MCH 1016可以通过高带宽存储器路径1018耦合到存储器1020,并且图形/视频卡1012可以通过加速图形端口(Accelerated Graphics Port) (“AGP”)互连1014耦合到MCH 1016。

[0140] 在至少一个实施例中,计算机系统1000可以使用系统I/O 1022作为专有集线器接口总线来将MCH 1016耦合到I/O控制器集线器(“ICH”)1030。在至少一个实施例中,ICH 1030可以通过本地I/O总线提供与某些I/O设备的直接连接。在至少一个实施例中,本地I/O总线可以包括但不限于用于将外围设备连接到存储器1020、芯片组和处理器1002的高速I/O总线。示例可以包括但不限于音频控制器1029、固件集线器(“Flash BIOS”)1028、无线收发器1026、数据存储1024、包含用户输入1025的传统I/O控制器1023和键盘接口、串行扩展端口1027(例如USB)和网络控制器1034。数据存储1024可以包括硬盘驱动器、软盘驱动器、CD-ROM设备、闪存设备或其他大容量存储设备。

[0141] 在至少一个实施例中,图10示出了包括互连的硬件设备或“芯片”的系统。在至少一个实施例中,图10可以示出示例性SoC。在至少一个实施例中,图10中示出的设备可以与专有互连、标准化互连(例如,PCIe)或其某种组合互连。在至少一个实施例中,系统1000的一个或更多个组件使用计算快速链路(CXL)互连来互连。

[0142] 图11示出了根据至少一个实施例的系统1100。在至少一个实施例中,系统1100是利用处理器1110的电子设备。在至少一个实施例中,系统1100可以是,例如但不限于,笔记本电脑、塔式服务器、机架服务器、刀片服务器、膝上型计算机、台式机、平板电脑、移动设备、电话、嵌入式计算机或任何其他合适的电子设备。

[0143] 在至少一个实施例中,系统1100可以包括但不限于通信地耦合到任何合适数量或种类的组件、外围设备、模块或设备的处理器1110。在至少一个实施例中,处理器1110使用总线或接口耦合,诸如I<sup>2</sup>C总线、系统管理总线(“SMBus”)、低引脚数(LPC)总线、串行外围接口(“SPI”)、高清音频(“HDA”)总线、串行高级技术附件(“SATA”)总线、USB(1、2、3版)或通用异步接收器/发送器(“UART”)总线。在至少一个实施例中,图11示出了系统,该系统包括互连的硬件设备或“芯片”。在至少一个实施例中,图11可以示出示例性SoC。在至少一个实施例中,图11中所示的设备可以与专有互连线、标准化互连(例如,PCIe)或其某种组合互连。在至少一个实施例中,图11的一个或更多个组件使用计算快速链路(CXL)互连线来互连。

[0144] 在至少一个实施例中,图11可以包括显示器1124、触摸屏1125、触摸板1130、近场通信单元(“NFC”)1145、传感器集线器1140、热传感器1146、快速芯片组(“EC”)1135、可信平台模块(“TPM”)1138、BIOS/固件/闪存(“BIOS,FW Flash”)1122、DSP1160、固态硬盘(“SSD”)或硬盘驱动器(“HDD”)1120、无线局域网单元(“WLAN”)1150、蓝牙单元1152、无线广域网单元(“WWAN”)1156、全球定位系统(GPS)1155、相机(“USB 3.0相机”)1154(例如USB 3.0相机)或以例如LPDDR3标准实现的低功耗双倍数据速率(“LPDDR”)存储器单元(“LPDDR3”)1115。这些组件可以各自以任何合适的方式实现。

[0145] 在至少一个实施例中,其他组件可以通过以上讨论的组件通信地耦合到处理器1110。在至少一个实施例中,加速度计1141、环境光传感器(“ALS”)1142、罗盘1143和陀螺仪1144可以通信地耦合到传感器集线器1140。在至少一个实施例中,热传感器1139、风扇1137、键盘1146和触摸板1130可以通信地耦合到EC1135。在至少一个实施例中,扬声器1163、耳机1164和麦克风(“mic”)1165可以通信地耦合到音频单元(“音频编解码器和D类放大器”)1164,其又可以通信地耦合到DSP1160。在至少一个实施例中,音频单元1164可以包括例如但不限于音频编码器/解码器(“编解码器”)和D类放大器。在至少一个实施例中,SIM卡(“SIM”)1157可以通信地耦合到WWAN单元1156。在至少一个实施例中,组件(诸如WLAN单元1150和蓝牙单元1152以及WWAN单元1156)可以被实现为下一代形式因素(NGFF)。

[0146] 图12示出了根据至少一个实施例的示例性集成电路1200。在至少一个实施例中,示例性集成电路1200是SoC,其可使用一个或更多个IP核心制造。在至少一个实施例中,集成电路1200包括一个或更多个应用处理器1205(例如,CPU)、至少一个图形处理器1210,并且可以另外包括图像处理器1215和/或视频处理器1220,其中任意一个可能是模块化IP核心。在至少一个实施例中,集成电路1200包括外围或总线逻辑,其包括USB控制器1225、UART控制器1230、SPI/SDIO控制器1235和I<sup>2</sup>S/I<sup>2</sup>C控制器1240。在至少一个实施例中,集成电路1200可以包括显示设备1245耦合到高清多媒体接口(HDMI)控制器1250和移动工业处理器

接口 (MIPI) 显示接口1255中的一个或多个。在至少一个实施例中,存储可以由闪存子系统1260提供,包括闪存和闪存控制器。在至少一个实施例中,可以经由存储器控制器1265提供存储器接口以用于访问SDRAM或SRAM存储器设备。在至少一个实施例中,一些集成电路还包括嵌入式安全引擎1270。

[0147] 图13示出了根据至少一个实施例的计算系统1300。在至少一个实施例中,计算系统1300包括处理子系统1301,其具有经由可以包括存储器集线器1305的互连路径通信的一个或多个处理器1302和系统存储器1304。在至少一个实施例中,存储器集线器1305可以是芯片组组件内的单独组件,也可以集成在一个或多个处理器1302内。在至少一个实施例中,存储器集线器1305通过通信链路1306与I/O子系统1311耦合。在至少一个实施例中,I/O子系统1311包括I/O集线器1307,其可以使计算系统1300能够接收来自一个或多个输入设备1308的输入。在至少一个实施例中,I/O集线器1307可以使能显示控制器,其包括在一个或多个处理器1302中,用于向一个或多个显示设备1310A提供输出。在至少一个实施例中,与I/O集线器1307耦合的一个或多个显示设备1310A可以包括本地、内部或嵌入式显示设备。

[0148] 在至少一个实施例中,处理子系统1301包括经由总线或其他通信链路1313耦合到存储器集线器1305的一个或多个并行处理器1312。在至少一个实施例中,通信链路1313可以是许多基于标准的通信链路技术或协议中的一种,例如但不限于PCIe,或者可以是针对供应商的通信接口或通信结构。在至少一个实施例中,一个或多个并行处理器1312形成计算集中的并行或向量处理系统,该系统可以包括大量的处理核心和/或处理集群,例如多集成核心(MIC)处理器。在至少一个实施例中,一个或多个并行处理器1312形成可以将像素输出到经由I/O集线器1307耦合的一个或多个显示设备1310A之一的图形处理子系统。在至少一个实施例中,一个或多个并行处理器1312还可以包括显示控制器和显示接口(未示出),以使得能够直接连接到一个或多个显示设备1310B。

[0149] 在至少一个实施例中,系统存储单元1314可以连接到I/O集线器1307,以提供用于计算系统1300的存储机制。在至少一个实施例中,I/O交换机1316可以用于提供接口机制,以实现I/O集线器1307与其他组件之间的连接,例如可以集成到平台中的网络适配器1318和/或无线网络适配器1319,以及可以通过一个或多个附加设备1320添加的各种其他设备。在至少一个实施例中,网络适配器1318可以是有以太网适配器或另一有线网络适配器。在至少一个实施例中,无线网络适配器1319可以包括Wi-Fi、蓝牙、NFC的一个或多个或其他包括一个或多个无线电的网络设备。

[0150] 在至少一个实施例中,计算系统1300可以包括未明确示出的其他组件,包括USB或其他端口连接、光存储驱动器、视频捕获设备等,也可以连接到I/O集线器1307。在至少一个实施例中,对图13中的各个组件进行互连的通信路径可以使用任何合适的协议来实现,诸如基于PCI(外围组件互连)的协议(例如,PCIe),或其他总线或点对点通信接口和/或协议,诸如存储器结构(例如,NVLink高速互连,英特尔计算快速链接(CXL),AMD无限)或互连协议。

[0151] 在至少一个实施例中,一个或多个并行处理器1312包括针对图形和视频处理而优化的电路(包括例如视频输出电路),并构成图形处理单元(GPU)。在至少一个实施例中,一个或多个并行处理器1312包括针对通用处理而优化的电路。在至少一个实施例中,计

算系统1300的组件可以与单个集成电路上的一個或更多个其他系统元件集成。例如,在至少一个实施例中,一个或更多个并行处理器1312、存储器集线器1305、处理器1302和I/O集线器1307可以被集成到片上系统(SoC)集成电路中。在至少一个实施例中,计算系统1300的组件可以被集成到单个封装中以形成系统级封装(SIP)配置。在至少一个实施例中,计算系统1300的组件的至少一部分可以被集成到多芯片模块(MCM)中,该多芯片模块可以与其他多芯片模块互连到模块化计算系统中。在至少一个实施例中,从计算系统1300中省略了I/O子系统1311和显示设备1310B。

#### [0152] 处理系统

[0153] 以下各图阐述了但不限于可用于实现至少一个实施例的示例性处理系统。在至少一个实施例中,以下附图的一个或更多个处理系统可以实现参照图1A描述的实施例,参照图1B描述的实施例、参照图2描述的实施例、参照图3描述的技术300、参照图4描述的技术400、参照图5描述的技术500、参照图6描述的技术600和/或参照图7描述的技术700的一个或更多个方面。

[0154] 图14示出了根据至少一个实施例的加速处理单元(“APU”)1400。在至少一个实施例中,APU 1400由加利福尼亚州圣克拉拉市的AMD公司开发。在至少一个实施例中,APU 1400可以被配置为执行应用程序,诸如CUDA程序。在至少一个实施例中,APU 1400包括但不限于核心复合体1410、图形复合体1440、结构1460、I/O接口1470、存储器控制器1480、显示控制器1492和多媒体引擎1494。在至少一个实施例中,APU 1400可以包括但不限于任意数量的核心复合体1410、任意数量的图形复合体1450、任意数量的显示控制器1492和任意数量的多媒体引擎1494的任何组合。为了说明的目的,在本文中用附图标记表示相似对象的多个实例,其中附图标记标识该对象,并且括号中的数字标识所需要的实例。

[0155] 在至少一个实施例中,核心复合体1410是CPU,图形复合体1440是GPU,并且APU 1400是将不限于1410和1440集成到单个芯片上的处理单元。在至少一个实施例中,一些任务可以被分配给核心复合体1410,而其他任务可以被分配给图形复合体1440。在至少一个实施例中,核心复合体1410被配置为执行与APU 1400相关联的主控制软件,例如操作系统。在至少一个实施例中,核心复合体1410是APU 1400的主处理器,其控制和协调其他处理器的操作。在至少一个实施例中,核心复合体1410发出控制图形复合体1440的操作的命令。在至少一个实施例中,核心复合体1410可以被配置为执行从CUDA源代码派生的主机可执行代码,并且图形复合体1440可以被配置为执行从CUDA源代码派生的设备可执行代码。

[0156] 在至少一个实施例中,核心复合体1410包括但不限于核心1420(1)-1420(4)和L3高速缓存1430。在至少一个实施例中,核心复合体1410可以包括但不限于任意数量的核心1420以及任意数量和类型的高速缓存的任何组合。在至少一个实施例中,核心1420被配置为执行特定指令集架构(“ISA”)的指令。在至少一个实施例中,每个核心1420是CPU核心。

[0157] 在至少一个实施例中,每个核心1420包括但不限于获取/解码单元1422,整数执行引擎1424,浮点执行引擎1426和L2高速缓存1428。在至少一个实施例中,获取/解码单元1422获取指令,对这些指令进行解码,生成微操作,并将单独的微指令分派给整数执行引擎1424和浮点执行引擎1426。在至少一个实施例中,获取/解码单元1422可以同时分派一个微指令到整数执行引擎1424和另一微指令到浮点执行引擎1426。在至少一个实施例中,整数执行引擎1424执行不限于整数和存储器操作。在至少一个实施例中,浮点引擎1426执行不

限于浮点和向量运算。在至少一个实施例中,获取-解码单元1422将微指令分派给单个执行引擎,该执行引擎代替整数执行引擎1424和浮点执行引擎1426两者。

[0158] 在至少一个实施例中,每个核心1420(i)可以访问包括在核心1420(i)中的L2高速缓存1428(i),其中i是表示核心1420的特定实例的整数。在至少一个实施例中,包括在核心复合体1410(j)中的每个核心1420经由包括在核心复合体1410(j)中的L3高速缓存1430(j)连接到包括在核心复合体1410(j)中的其他核心1420,其中j是表示核心复合体1410的特定实例的整数。在至少一个实施例中,包括在核心复合体1410(j)中的核心1420可以访问包括在核心复合体1410(j)中的所有L3高速缓存1430(j),其中j是表示核心复合体1410的特定实例的整数。在至少一个实施例中,L3高速缓存1430可以包括但不限于任意数量的切片(slice)。

[0159] 在至少一个实施例中,图形复合体1440可以被配置为以高度并行的方式执行计算操作。在至少一个实施例中,图形复合体1440被配置为执行图形管线操作,诸如绘制命令、像素操作、几何计算以及与将图像渲染至显示器相关联的其他操作。在至少一个实施例中,图形复合体1440被配置为执行与图形无关的操作。在至少一个实施例中,图形复合体1440被配置为执行与图形有关的操作和与图形无关的操作。

[0160] 在至少一个实施例中,图形复合体1440包括但不限于任意数量的计算单元1450和L2高速缓存1442。在至少一个实施例中,计算单元1450共享L2高速缓存1442。在至少一个实施例中,L2高速缓存1442被分区。在至少一个实施例中,图形复合体1440包括但不限于任意数量的计算单元1450以及任意数量(包括零)和类型的高速缓存。在至少一个实施例中,图形复合体1440包括但不限于任意数量的专用图形硬件。

[0161] 在至少一个实施例中,每个计算单元1450包括但不限于任意数量的SIMD单元1452和共享存储器1454。在至少一个实施例中,每个SIMD单元1452实现SIMD架构并且被配置为并行执行操作。在至少一个实施例中,每个计算单元1450可以执行任意数量的线程块,但是每个线程块在单个计算单元1450上执行。在至少一个实施例中,线程块包括但不限于任意数量的执行线程。在至少一个实施例中,工作组是线程块。在至少一个实施例中,每个SIMD单元1452执行不同的线程束(warp)。在至少一个实施例中,线程束是一组线程(例如16个线程),其中线程束中的每个线程属于单个线程块,并且被配置为基于单个指令集来处理不同的数据集。在至少一个实施例中,可以使用预测(predication)来禁用线程束中的一个或多个线程。在至少一个实施例中,通道是线程。在至少一个实施例中,工作项是线程。在至少一个实施例中,波前是线程束。在至少一个实施例中,线程块中的不同波前可一起同步并经由共享存储器1454进行通信。

[0162] 在至少一个实施例中,结构1460是系统互连,其促进跨核心复合体1410、图形复合体1440、I/O接口1470、存储器控制器1480、显示控制器1492和多媒体引擎1494的数据和控制传输。在至少一个实施例中,除了结构1460之外或代替结构1460,APU 1400还可以包括但不限于任意数量和类型的系统互连,该结构1460促进跨可以在APU 1400内部或外部的任意数量和类型的直接或间接链接的组件的数据和控制传输。在至少一个实施例中,I/O接口1470表示任意数量和类型的I/O接口(例如,PCI,PCI-Extended(“PCI-X”),PCIe,千兆以太网(“GbE”),USB等)。在至少一个实施例中,各种类型的外围设备耦合到I/O接口1470。在至少一个实施例中,耦合到I/O接口1470的外围设备可以包括但不限于键盘,鼠标,打印机,扫

描仪,操纵杆或其他类型的游戏控制器、媒体记录设备、外部存储设备、网络接口卡等。

[0163] 在至少一个实施例中,显示控制器AMD92在一个或多个显示设备(例如液晶显示器(LCD)设备)上显示图像。在至少一个实施例中,多媒体引擎240包括但不限于任意数量和类型的与多媒体相关的电路,例如视频解码器、视频编码器、图像信号处理器等。在至少一个实施例中,存储器控制器1480促进APU 1400与统一系统存储器1490之间的数据传输。在至少一个实施例中,核心复合体1410和图形复合体1440共享统一系统存储器1490。

[0164] 在至少一个实施例中,APU 1400实现存储器子系统,其包括但不限于任意数量和类型的存储器控制器1480和可以专用于一个组件或在多个组件之间共享的存储器设备(例如,共享存储器1454)。在至少一个实施例中,APU 1400实现高速缓存子系统,其包括但不限于一个或多个高速缓存存储器(例如,L2高速缓存1528,L3高速缓存1430和L2高速缓存1442),每个高速缓存存储器可以是组件私有的或在任意数量的组件(例如,核心1420,核心复合体1410,SIMD单元1452,计算单元1450和图形复合体1440)之间共享。

[0165] 图15示出了根据至少一个实施例的CPU 1500。在至少一个实施例中,CPU 1500由加利福尼亚州圣克拉拉市的AMD公司开发。在至少一个实施例中,CPU 1500可以被配置为执行应用程序。在至少一个实施例中,CPU1500被配置为执行主控制软件,例如操作系统。在至少一个实施例中,CPU1500发出控制外部GPU(未示出)的操作的命令。在至少一个实施例中,CPU 1500可以被配置为执行从CUDA源代码派生的主机可执行代码,并且外部GPU可以被配置为执行从这种CUDA源代码派生的设备可执行代码。在至少一个实施例中,CPU 1500包括但不限于任意数量的核心复合体1510,结构1560,I/O接口1570和存储器控制器1580。

[0166] 在至少一个实施例中,核心复合体1510包括但不限于核心1520(1)-1520(4)和L3高速缓存1530。在至少一个实施例中,核心复合体1510可以包括但不限于任意数量的核心1520以及任意数量和类型的高速缓存的任何组合。在至少一个实施例中,核心1520被配置为执行特定ISA的指令。在至少一个实施例中,每个核心1520是CPU核心。

[0167] 在至少一个实施例中,每个核心1520包括但不限于获取/解码单元1522,整数执行引擎1524,浮点执行引擎1526和L2高速缓存1528。在至少一个实施例中,获取/解码单元1522获取指令,对这些指令进行解码,生成微操作,并将单独的微指令分派给整数执行引擎1524和浮点执行引擎1526。在至少一个实施例中,获取/解码单元1522可以同时分派一个微指令至整数执行引擎1524和另一微指令至浮点执行引擎1526。在至少一个实施例中,整数执行引擎1524执行不限于整数和存储器操作。在至少一个实施例中,浮点引擎1526执行不限于浮点和向量运算。在至少一个实施例中,获取-解码单元1522将微指令分派给单个执行引擎,该引擎代替整数执行引擎1524和浮点执行引擎1526两者。

[0168] 在至少一个实施例中,每个核心1520(i)可以访问包括在核心1520(i)中的L2高速缓存1528(i),其中i是表示核心1520的特定实例的整数。在至少一个实施例中,包括在核心复合体1510(j)中的每个核心1520经由包括在核心复合体1510(j)中的L3高速缓存1530(j)连接到核心复合体1510(j)中的其他核心1520,其中j是表示核心复合体1510的特定实例的整数。在至少一个实施例中,包括在核心复合体1510(j)中的核心1520可以访问包括在核心复合体1510(j)中的所有L3高速缓存1530(j),其中j是表示核心复合体1510的特定实例的整数。在至少一个实施例中,L3高速缓存1530可以包括但不限于任意数量的切片。

[0169] 在至少一个实施例中,结构1560是系统互连,其促进跨核心复合体1510(1)-1510

(N) (其中N是大于零的整数)、I/O接口1570和存储器控制器1580的数据和控制传输。在至少一个实施例中,除了结构1560之外或代替结构1560,CPU 1500还可以包括但不限于任意数量和类型的系统互连,该结构1560促进跨可以在CPU 1500内部或外部的任意数量和类型的直接或间接链接的组件的数据和控制传输。在至少一个实施例中,I/O接口1570表示任意数量和类型的I/O接口(例如PCI,PCI-X,PCIe,GBE,USB等)。在至少一个实施例中,各种类型的外围设备耦合到I/O接口1570。在至少一个实施例中,耦合到I/O接口1570的外围设备可以包括但不限于显示器,键盘,鼠标,打印机,扫描仪,操纵杆或其他类型的游戏控制器、媒体记录设备、外部存储设备、网络接口卡等。

[0170] 在至少一个实施例中,存储器控制器1580促进CPU 1500与系统存储器1590之间的数据传输。在至少一个实施例中,核心复合体1510和图形复合体1540共享系统存储器1590。在至少一个实施例中,CPU 1500实现存储器子系统,其包括但不限于任意数量和类型的存储器控制器1580和可以专用于一个组件或在多个组件之间共享的存储器设备。在至少一个实施例中,CPU 1500实现了高速缓存子系统,其包括但不限于一个或更多个高速缓存存储器(例如,L2高速缓存1528和L3高速缓存1530),每个高速缓存存储器可以是组件私有的或在任意数量的组件(例如,核心1520和核心复合体1510)之间共享。

[0171] 图16示出了根据至少一个实施例的示例性加速器集成切片1690。如本文所使用的,“切片”包括加速器集成电路的处理资源的指定部分。在至少一个实施例中,加速器集成电路代表多个图形加速模块种的多个图形处理引擎提供高速缓存管理、存储器访问、环境管理和中断管理服务。图形处理引擎可以各自包括单独的GPU。可选地,图形处理引擎可包括GPU内的不同类型的图形处理引擎,例如图形执行单元、媒体处理引擎(例如,视频编码器/解码器)、采样器和blit引擎。在至少一个实施例中,图形加速模块可以是具有多个图形处理引擎的GPU。在至少一个实施例中,图形处理引擎可以是集成在通用封装、线卡或芯片上的各个GPU。

[0172] 系统存储器1614内的应用程序有效地址空间1682存储进程元素1683。在一个实施例中,响应于来自处理器1607上执行的应用程序1680的GPU调用1681而存储进程元素1683。进程元素1683包含对应应用程序1680的处理状态。包含在进程元素1683中的工作描述符(WD) 1684可以是应用程序请求的单个作业或可能包含指向作业队列的指针。在至少一个实施例中,WD 1684是指向应用程序有效地址空间1682中的作业请求队列的指针。

[0173] 图形加速模块1646和/或各个图形处理引擎可以由系统中的全部或部分进程共享。在至少一个实施例中,可以包括用于建立处理状态并将WD1684发送到图形加速模块1646以在虚拟化环境中开始作业的基础设施。

[0174] 在至少一个实施例中,专用进程编程模型是针对实现的。在该模型中,单个进程拥有图形加速模块1646或个体图形处理引擎。由于图形加速模块1646由单个进程拥有,因此管理程序为拥有的分区初始化加速器集成电路,并且当分配图形加速模块1646时操作系统对加速器集成电路进行初始化以用于拥有的分区。

[0175] 在操作中,加速器集成切片1690中的WD获取单元1691获取下一个WD 1684,其中包括要由图形加速模块1646的一个或更多个图形处理引擎完成的工作的指示。来自WD 1684的数据可以存储在寄存器1645被存储器管理单元(MMU) 1639、中断管理电路1647和/或环境管理电路1648使用,如图所示。例如,MMU 1639的一个实施例包括用于访问OS虚拟地址空间

1685内的段/页表1686的段/页面漫游电路。中断管理电路1647可以处理从图形加速模块1646接收到的中断事件 (INT) 1692。当执行图操作时,由图形处理引擎产生的有效地址1693由MMU 1639转换为实际地址。

[0176] 在一个实施例中,为每个图形处理引擎和/或图形加速模块1646复制相同的寄存器组1645,并且可以由系统管理程序或操作系统来初始化。这些复制的寄存器中的每一个都可以包含在加速器集成切片1690中。表1中显示了可由管理程序初始化的示例性寄存器。

[0177] 表1-管理程序初始化的寄存器

|        |   |                         |
|--------|---|-------------------------|
| [0178] | 1 | 切片控制寄存器                 |
|        | 2 | 实地址 (RA) 计划的处理区域指针      |
|        | 3 | 授权掩码覆盖寄存器               |
|        | 4 | 中断向量表输入偏移               |
|        | 5 | 中断向量表入口限制               |
|        | 6 | 状态寄存器                   |
|        | 7 | 逻辑分区ID                  |
|        | 8 | 实地址 (RA) 管理程序加速器利用率记录指针 |
|        | 9 | 存储描述寄存器                 |

[0179] 表2中示出了可以由操作系统初始化的示例性寄存器。

[0180] 表2-操作系统初始化寄存器

|        |   |                      |
|--------|---|----------------------|
| [0181] | 1 | 进程和线程识别              |
|        | 2 | 有效地址 (EA) 环境保存/还原指针  |
|        | 3 | 虚拟地址 (VA) 加速器利用率记录指针 |
|        | 4 | 虚拟地址 (VA) 存储分段表指针    |
|        | 5 | 权威面具                 |
| [0182] | 6 | 工作描述符                |

[0183] 在一个实施例中,每个WD 1684特定于特定的图形加速模块1646和/或特定图形处理引擎。它包含图形处理引擎进行工作或工作所需的所有信息,或者它可以是指向存储器位置的指针,其中应用程序建立了要完成的工作的命令队列。

[0184] 图17A和图17B示出了根据本文至少一个实施例的示例性图形处理器。在至少一个实施例中,任何示例性图形处理器可以使用一个或更多个IP核心来制造。除了图示之外,在至少一个实施例中可以包括其他逻辑和电路,包括附加的图形处理器/核心、外围接口控制器或通用处理器核心。在至少一个实施例中,示例性图形处理器用于SoC内。

[0185] 图17A示出了根据至少一个实施例的SoC集成电路的示例性图形处理器1710,其可

以使用一个或更多个IP核心来制造。图17B示出了根据至少一个实施例的SoC集成电路的附加示例性图形处理器1740,其可以使用一个或更多个IP核心来制造。在至少一个实施例中,图17A的图形处理器1710是低功耗图形处理器核心。在至少一个实施例中,图17B的图形处理器1740是更高性能的图形处理器核心。在至少一个实施例中,每个图形处理器1710、1740可以是图12的图形处理器1210的变体。

[0186] 在至少一个实施例中,图形处理器1710包括顶点处理器1705和一个或更多个片段处理器1715A-1715N(例如1715A、1715B、1715C、1715D至1715N-1和1715N)。在至少一个实施例中,图形处理器1710可以经由单独的逻辑来执行不同的着色器程序,使得顶点处理器1705被优化以执行针对顶点着色器程序的操作,而一个或更多个片段处理器1715A-1715N执行片段(例如,像素)着色操作于片段或像素或着色器程序。在至少一个实施例中,顶点处理器1705执行3D图形管线的顶点处理阶段并生成图元和顶点数据。在至少一个实施例中,片段处理器1715A-1715N使用由顶点处理器1705生成的图元和顶点数据来生成在显示设备上显示的帧缓冲器。在至少一个实施例中,片段处理器1715A-1715N被优化以执行如在OpenGL API中所提供的片段着色器程序,其可以用于执行与在Direct 3D API中所提供的像素着色器程序类似的操作。

[0187] 在至少一个实施例中,图形处理器1710附加地包括一个或更多个MMU 1720A-1720B、高速缓存1725A-1725B和电路互连1730A-1730B。在至少一个实施例中,一个或更多个MMU 1720A-1720B提供用于图形处理器1710的虚拟到物理地址的映射,包括用于顶点处理器1705和/或片段处理器1715A-1715N,其可以引用存储在存储器中的顶点或图像/纹理数据,除了存储在一个或更多个高速缓存1725A-1725B中的顶点或图像/纹理数据之外。在至少一个实施例中,一个或更多个MMU 1720A-1720B可以与系统内的其他MMU同步,包括与图12的一个或更多个应用处理器1205、图像处理器1215和/或视频处理器1220相关联的一个或更多个MMU,使得每个处理器1205-1220可以参与共享或统一的虚拟存储器系统。在至少一个实施例中,一个或更多个电路互连1730A-1730B使图形处理器1710能够经由SoC的内部总线或经由直接连接与SoC内的其他IP核心相连接。

[0188] 在至少一个实施例中,图形处理器1740包括图17A的图形处理器1710的一个或更多个MMU 1720A-1720B、高速缓存1725A-1725B和电路互连1730A-1730B。在至少一个实施例中,图形处理器1740包括一个或更多个着色器核心1755A-1755N(例如,1755A、1755B、1755C、1755D、1755E、1755F、至1755N-1和1755N),其提供了统一的着色器核心架构,其中单个核心或类型或核心可以执行所有类型的可编程着色器代码,包括用于实现顶点着色器、片段着色器和/或计算着色器的着色器程序代码。在至少一个实施例中,多个着色器核心可以变化。在至少一个实施例中,图形处理器1740包括核心间任务管理器1745,其充当线程分派器以将执行线程分派给一个或更多个着色器核心1755A-1755N和分块单元1758,以加速基于瓦片渲染的分块操作,其中在图像空间中细分了场景的渲染操作,例如,以利用场景内的局部空间一致性或优化内部缓存的使用。

[0189] 图18A示出了根据至少一个实施例的图形核心1800。在至少一个实施例中,图形核心1800可以包括在图12的图形处理器1210内。在至少一个实施例中,图形核心1800可以是图17B中统一的着色器核心1755A-1755N。在至少一个实施例中,图形核心1800包括共享指令高速缓存1802、纹理单元1818和高速缓存/共享存储器1820,它们是图形核心1800内的执

行资源所共有的。在至少一个实施例中,图形核心1800可以包括多个切片(slice)1801A-1801N或每个核心的分区,图形处理器可以包括图形核心1800的多个实例。切片1801A-1801N可以包括支持逻辑,该支持逻辑包括本地指令高速缓存1804A-1804N、线程调度器1806A-1806N、线程分派器1808A-1808N和一组寄存器1810A-1810N。在至少一个实施例中,切片1801A-1801N可以包括一组附加功能单元(AFU)1812A-1812N、浮点单元(FPU)1814A-1814N、整数算术逻辑单元(ALU)1816A-1816N、地址计算单元(ACU)1813A-1813N、双精度浮点单元(DPFPFU)1815A-1815N和矩阵处理单元(MPU)1817A-1817N。

[0190] 在一个实施例中,FPU 1814A-1814N可以执行单精度(32位)和半精度(16位)浮点运算,而DPFPFU 1815A-1815N可以执行双精度(64位)浮点运算点操作。在至少一个实施例中,ALU 1816A-1816N可以以8位、16位和32位精度执行可变精度整数运算,并且可以被配置用于混合精度运算。在至少一个实施例中,MPU 1817A-1817N还可被配置用于混合精度矩阵运算,包括半精度浮点运算和8位整数运算。在至少一个实施例中,MPU 1817A-1817N可以执行各种矩阵操作以加速CUDA程序,包括使得能够支持加速的通用矩阵到矩阵乘法(GEMM)。在至少一个实施例中,AFU 1812A-1812N可以执行浮点数或整数单元不支持的附加逻辑运算,包括三角运算(例如,Sine、Cosine等)。

[0191] 图18B示出了在至少一个实施例中的通用图形处理单元(GPGPU)1830。在至少一个实施例中,GPGPU 1830是高度并行的并且适合于部署在多芯片模块上。在至少一个实施例中,GPGPU1830可以被配置为使得高度并行的计算操作能够由GPU数组来执行。在至少一个实施例中,GPGPU1830可以直接链路到GPGPU 1830的其他实例,以创建多组GPU群以提高用于CUDA程序的执行时间。在至少一个实施例中,GPGPU 1830包括主机接口1832以实现与主机处理器的连接。在至少一个实施例中,主机接口1832是PCIe接口。在至少一个实施例中,主机接口1832可以是厂商专用的通信接口或通信结构。在至少一个实施例中,GPGPU 1830从主机处理器接收命令,并使用全局调度器1834将与那些命令相关联的执行线程分派给一组计算集群1836A-1836H。在至少一个实施例中,计算集群1836A-1836H共享高速缓存存储器1838。在至少一个实施例中,高速缓存存储器1838可以用作计算集群1836A-1836H内的高速缓存存储器的高级高速缓存。

[0192] 在至少一个实施例中,GPGPU 1830包括经由一组存储器控制器1842A-1842B与计算集群1836A-1836H耦合的存储器1844A-1844B。在至少一个实施例中,存储器1844A-1844B可以包括各种类型的存储器设备,包括动态随机存取存储器(DRAM)或图形随机存取存储器,例如同步图形随机存取存储器(SGRAM),包括图形双倍数据速率(GDDR)存储器。

[0193] 在至少一个实施例中,计算集群1836A-1836H各自包括一组图形核心,诸如图18A的图形核心1800,其可以包括多种类型的整数和浮点逻辑单元,可以以各种精度执行计算操作,包括适合与CUDA程序相关的计算。例如,在至少一个实施例中,每个计算集群1836A-1836H中的浮点单元的至少一个子集可以配置为执行16位或32位浮点运算,而不同的浮点单元的子集可以配置为执行64位浮点运算。

[0194] 在至少一个实施例中,GPGPU 1830的多个实例可以被配置为操作为计算集群。计算集群1836A-1836H可以实现用于同步和数据交换的任何技术上可行的通信技术。在至少一个实施例中,GPGPU 1830的多个实例通过主机接口1832进行通信。在至少一个实施例中,GPGPU 1830包括I/O集线器1839,其将GPGPU 1830与GPU链路1840耦合,使得能够直接连接

至GPGPU1830的其他的实例。在至少一个实施例中,GPU链路1840耦合到专用GPU到GPU桥接器,其使得能够在GPGPU 1830的多个实例之间进行通信和同步。在至少一个实施例中,GPU链路1840与高速互连耦合,以向其他GPGPU或并行处理器发送和接收数据。在至少一个实施例中,GPGPU1830的多个实例位于单独的数据处理系统中,并经由可经由主机接口1832访问的网络设备进行通信。在至少一个实施例中,GPU链路1840可被配置为能够连接到主机处理器,附加或替代主机接口1832。在至少一个实施例中,GPGPU 1830可以配置为执行CUDA程序。

[0195] 图19A示出了根据至少一个实施例的并行处理器1900。在至少一个实施例中,并行处理器1900的各种组件可以使用一个或更多个集成电路设备来实现,例如可编程处理器、专用集成电路(ASIC)或FPGA。

[0196] 在至少一个实施例中,并行处理器1900包括并行处理单元1902。在至少一个实施例中,并行处理单元1902包括I/O单元1904,其使得能够与其他设备进行通信,包括并行处理单元1902的其他实例。在至少一个实施例中,I/O单元1904可以直接连接到其他设备。在至少一个实施例中,I/O单元1904通过使用集线器或交换机接口(例如,存储器集线器1305)与其他设备连接。在至少一个实施例中,存储器集线器1305与I/O单元1904之间的连接形成通信链路。在至少一个实施例中,I/O单元1904与主机接口1906和存储器交叉交换机1916连接,其中主机接口1906接收用于执行处理操作的命令,而存储器交叉交换机1916接收用于执行存储器操作的命令。

[0197] 在至少一个实施例中,当主机接口1906经由I/O单元1904接收命令缓冲器时,主机接口1906可以引导工作操作以执行那些命令到前端1908。在至少一个实施例中,前端1908与调度器1910耦合,调度器1910配置成将命令或其他工作项分配给处理数组1912。在至少一个实施例中,调度器1910确保在将任务分配给处理数组1912中的处理数组1912之前,处理数组1912被正确地配置并且处于有效状态。在至少一个实施例中,调度器1910通过在微控制器上执行的固件逻辑来实现。在至少一个实施例中,微控制器实现的调度器1910可配置成以粗粒度和细粒度执行复杂的调度和工作分配操作,从而实现对在处理数组1912上执行的线程的快速抢占和环境切换。在至少一个实施例中,主机软件可以证明用于通过多个图形处理门铃之一在处理数组1912上进行调度的工作负载。在至少一个实施例中,工作负载然后可以由包括调度器1910的微控制器内的调度器1910逻辑在处理数组1912上自动分配。

[0198] 在至少一个实施例中,处理数组1912可以包括多达“N”个处理集群(例如,集群1914A、集群1914B到集群1914N)。在至少一个实施例中,处理数组1912的每个集群1914A-1914N可以执行大量并发线程。在至少一个实施例中,调度器1910可以使用各种调度和/或工作分配算法将工作分配给处理数组1912的集群1914A-1914N,其可以根据每种程序或计算类型产生的工作负载而变化。在至少一个实施例中,调度可以由调度器1910动态地处理,或者可以在配置为由处理数组1912执行的程序逻辑的编译期间部分地由编译器逻辑来辅助。在至少一个实施例中,可将处理数组1912的不同的集群1914A-1914N分配用于处理不同类型的程序或用于执行不同类型的计算。

[0199] 在至少一个实施例中,处理数组1912可以配置成执行各种类型的并行处理操作。在至少一个实施例中,处理数组1912配置成执行通用并行计算操作。例如,在至少一个实施

例中,处理数组1912可以包括执行处理任务的逻辑,该处理任务包括对视频和/或音频数据的过滤,执行建模操作,包括物理操作以及执行数据转换。

[0200] 在至少一个实施例中,处理数组1912配置成执行并行图形处理操作。在至少一个实施例中,处理数组1912可以包括附加逻辑以支持这种图形处理操作的执行,包括但不限于执行纹理操作的纹理采样逻辑,以及镶嵌逻辑和其他顶点处理逻辑。在至少一个实施例中,处理数组1912可以配置成执行与图形处理有关的着色器程序,例如但不限于顶点着色器、曲面细分着色器、几何着色器和像素着色器。在至少一个实施例中,并行处理单元1902可以经由I/O单元1904从系统存储器传送数据以进行处理。在至少一个实施例中,在处理期间,可以在处理期间将传送的数据存储到片上存储器(例如,并行处理器存储器1922),然后将其写回到系统存储器。

[0201] 在至少一个实施例中,当并行处理单元1902用于执行图处理时,调度器1910可以配置成将处理工作负载划分为近似相等大小的任务,以更好地将图形处理操作分配给处理数组1912的多个集群1914A-1914N。在至少一个实施例中,处理数组1912的部分可以配置成执行不同类型的处理。例如,在至少一个实施例中,第一部分可以配置成执行顶点着色和拓扑生成,第二部分可以配置成执行镶嵌和几何着色,并且第三部分可以配置成执行像素着色或其他屏幕空间操作,以生成用于显示的渲染图像。在至少一个实施例中,可以将由集群1914A-1914N中的一个或多个产生的中间数据存储于缓冲器中,以允许在集群1914A-1914N之间传输中间数据以进行进一步处理。

[0202] 在至少一个实施例中,处理数组1912可以经由调度器1910接收要执行的处理任务,该调度器1910从前端1908接收定义处理任务的命令。在至少一个实施例中,处理任务可以包括要被处理的数据的索引,例如可以包括表面(补丁)数据、原始数据、顶点数据和/或像素数据,以及状态参数和定义如何处理数据的命令(例如,要执行什么程序)。在至少一个实施例中,调度器1910可以配置成获取与任务相对应的索引,或者可以从前端1908接收索引。在至少一个实施例中,前端1908可以配置成确保在启动由传入命令缓冲器(例如,批缓冲器(batch-buffer)、推送缓冲器等)指定的工作负载之前,处理数组1912配置成有效状态。

[0203] 在至少一个实施例中,并行处理单元1902的一个或多个实例中的每一个可以与并行处理器存储器1922耦合。在至少一个实施例中,可以经由存储器交叉交换机1916访问并行处理器存储器1922,所述存储器交叉交换机1916可以接收来自处理数组1912以及I/O单元1904的存储器请求。在至少一个实施例中,存储器交叉交换机1916可以经由存储器接口1918访问并行处理器存储器1922。在至少一个实施例中,存储器接口1918可以包括多个分区单元(例如,分区单元1920A、分区单元1920B到分区单元1920N),其可各自耦合至并行处理器存储器1922的一部分(例如,存储器单元)。在至少一个实施例中,多个分区单元1920A-1920N为配置为等于存储器单元的数量,使得第一分区单元1920A具有对应的第一存储器单元1924A,第二分区单元1920B具有对应的存储器单元1924B,第N分区单元1920N具有对应的第N存储器单元1924N。在至少一个实施例中,分区单元1920A-1920N的数量可以不等于存储器设备的数量。

[0204] 在至少一个实施例中,存储器单元1924A-1924N可以包括各种类型的存储器设备,包括动态随机存取存储器(DRAM)或图形随机存取存储器,例如同步图形随机存取存储器

(SGRAM),包括图形双倍数据速率(GDDR)存储器。在至少一个实施例中,存储器单元1924A-1924N还可包括3D堆叠存储器,包括但不限于高带宽存储器(HBM)。在至少一个实施例中,可以跨存储器单元1924A-1924N来存储诸如帧缓冲器或纹理映射的渲染目标,从而允许分区单元1920A-1920N并行地写入每个渲染目标的部分,以有效地使用并行处理器存储器1922的可用带宽。在至少一个实施例中,可以排除并行处理器存储器1922的本地实例,以有利于利用系统存储器与本地高速缓存存储器结合的统一存储器设计。

[0205] 在至少一个实施例中,处理数组1912的集群1914A-1914N中的任何一个都可以处理将被写入并行处理器存储器1922内的任何存储器单元1924A-1924N中的数据。在至少一个实施例中,存储器交叉交换机1916可以配置为将每个集群1914A-1914N的输出传输到任何分区单元1920A-1920N或另一个集群1914A-1914N,集群1914A-1914N可以对输出执行其他处理操作。在至少一个实施例中,每个集群1914A-1914N可以通过存储器交叉交换机1916与存储器接口1918通信,以从各种外部存储设备读取或写入各种外部存储设备。在至少一个实施例中,存储器交叉交换机1916具有到存储器接口1918的连接以及与I/O单元1904通信,以及到并行处理器存储器1922的本地实例的连接,从而使不同处理集群1914A-1914N内的处理单元与系统存储器或不是并行处理单元1902本地的其他存储器进行通信。在至少一个实施例中,存储器交叉交换机1916可以使用虚拟通道来分离集群1914A-1914N和分区单元1920A-1920N之间的业务流。

[0206] 在至少一个实施例中,可以在单个插入卡上提供并行处理单元1902的多个实例,或者可以将多个插入卡互连。在至少一个实施例中,并行处理单元1902的不同实例可以配置成相互操作,即使不同实例具有不同数量的处理核心,不同数量的本地并行处理器存储器和/或其他配置差异。例如,在至少一个实施例中,并行处理单元1902的一些实例可以包括相对于其他实例而言更高精度的浮点单元。在至少一个实施例中,结合并行处理单元1902或并行处理器1900的一个或更多个实例的系统可以以各种配置和形式因素来实现,包括但不限于台式机、膝上型计算机或手持式个人计算机、服务器、工作站、游戏机和/或嵌入式系统。

[0207] 图19B示出了根据至少一个实施例的处理集群1994。在至少一个实施例中,处理集群1994被包括在并行处理单元内。在至少一个实施例中,处理集群1994是图19A的处理集群1914A-1914N之一的实例。在至少一个实施例中,处理集群1994可以配置成并行执行许多线程,其中术语“线程”是指在特定的一组输入数据上执行的特定程序的实例。在至少一个实施例中,单指令多数据(SIMD)指令发布技术用于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。在至少一个实施例中,使用单指令多线程(SIMT)技术来支持并行执行大量一般同步的线程,这使用了公共指令单元,该公共指令单元配置成向每个处理集群1994内的一组处理引擎发出指令。

[0208] 在至少一个实施例中,可以通过将处理任务分配给SIMT并行处理器的管线管理器1932来控制处理集群1994的操作。在至少一个实施例中,管线管理器1932从图19A的调度器1910接收指令,通过图形多处理器1934和/或纹理单元1936管理这些指令的执行。在至少一个实施例中,图形多处理器1934是SIMT并行处理器的示例性实例。然而,在至少一个实施例中,处理集群1994内可以包括不同架构的各种类型的SIMT并行处理器。在至少一个实施例中,在处理集群1994内可以包括图形多处理器1934的一个或更多个实例。在至少一个实施

例中,图形多处理器1934可以处理数据,并且数据交叉交换机1940可以用于将处理后的数据分发到多个可能的目的(包括其他着色器单元)地之一。在至少一个实施例中,管线管理器1932可以通过指定要经由数据交叉交换机1940分配的处理后的数据的目的地来促进处理后的数据的分配。

[0209] 在至少一个实施例中,处理集群1994内的每个图形多处理器1934可以包括相同的一组功能执行逻辑(例如,算术逻辑单元、加载存储单元(LSU)等)。在至少一个实施例中,可以以管线方式配置功能执行逻辑,其中可以在先前的指令完成之前发出新的指令。在至少一个实施例中,功能执行逻辑支持多种运算,包括整数和浮点算术、比较操作、布尔运算、移位和各种代数函数的计算。在至少一个实施例中,可以利用相同的功能单元硬件来执行不同的操作,并且可以存在功能单元的任何组合。

[0210] 在至少一个实施例中,传送到处理集群1994的指令构成线程。在至少一个实施例中,跨一组并行处理引擎执行的一组线程是线程组。在至少一个实施例中,线程组在不同的输入数据上执行程序。在至少一个实施例中,线程组内的每个线程可被分配给图形多处理器1934内的不同处理引擎。在至少一个实施例中,线程组可包括比图形多处理器1934内的多个处理引擎更少的线程。在至少一个实施例中,当线程组包括的线程数少于处理引擎的数量时,一个或更多个处理引擎在正在处理该线程组的循环期间可能是空闲的。在至少一个实施例中,线程组还可以包括比图形多处理器1934内的多个处理引擎更多的线程。在至少一个实施例中,当线程组包括比图形多处理器1934内的处理引擎的数量更多的线程时,可以在连续的时钟周期内执行处理。在至少一个实施例中,可以在图形多处理器1934上同时执行多个线程组。

[0211] 在至少一个实施例中,图形多处理器1934包括内部高速缓存存储器,以执行加载和存储操作。在至少一个实施例中,图形多处理器1934可以放弃内部高速缓存并使用处理集群1994内的高速缓存存储器(例如,L1高速缓存1948)。在至少一个实施例中,每个图形多处理器1934还可以访问分区单元(例如,图19A的分区单元1920A-1920N)内的L2高速缓存,这些分区单元在所有处理集群1994之间共享并且可以用于在线程之间传输数据。在至少一个实施例中,图形多处理器1934还可以访问片外全局存储器,其可以包括本地并行处理器存储器和/或系统存储器中的一个或更多个。在至少一个实施例中,并行处理单元1902外部的任何存储器都可以用作全局存储器。在至少一个实施例中,处理集群1994包括图形多处理器1934的多个实例,它们可以共享可以存储在L1高速缓存1948中的公共指令和数据。

[0212] 在至少一个实施例中,每个处理集群1994可以包括配置成将虚拟地址映射为物理地址的MMU1945。在至少一个实施例中,MMU 1945的一个或更多个实例可以驻留在图19A的存储器接口1918内。在至少一个实施例中,MMU 1945包括一组页表条目(PTE),其用于将虚拟地址映射到瓦片(谈论有关瓦片的更多信息)的物理地址以及可选地映射到高速缓存行索引。在至少一个实施例中,MMU 1945可以包括地址转换后备缓冲器(TLB)或可以驻留在图形多处理器1934或L1高速缓存1948或处理集群1994内的高速缓存。在至少一个实施例中,处理物理地址以分配表面数据访问局部性,以便在分区单元之间进行有效的请求交织。在至少一个实施例中,高速缓存行索引可以用于确定对高速缓存行的请求是命中还是未命中。

[0213] 在至少一个实施例中,可以配置处理集群1994,使得每个图形多处理器1934耦合

到纹理单元1936,以执行纹理映射操作,例如,可以涉及确定纹理样本位置、读取纹理数据以及过滤纹理数据。在至少一个实施例中,根据需从内部纹理L1高速缓存(未示出)或从图形多处理器1934内的L1高速缓存中读取纹理数据,并从L2高速缓存、本地并行处理器存储器或系统存储器中获取纹理数据。在至少一个实施例中,每个图形多处理器1934将处理后的任务输出到数据交叉交换机1940,以将处理后的任务提供给另一处理集群1994以进行进一步处理或将处理后的任务存储在L2高速缓存、本地并行处理器存储器、或经由存储器交叉交换机1916的系统存储器中。在至少一个实施例中,光栅前操作单元(preROP)1942配置成从图形多处理器1934接收数据,将数据引导至ROP单元,该ROP单元可以与本文所述的分区单元(例如,图19A的分区单元1920A-1920N)一起定位。在至少一个实施例中,PreROP1942单元可以执行用于颜色混合的优化、组织像素颜色数据以及执行地址转换。

[0214] 图19C示出了根据至少一个实施例的图形多处理器1996。在至少一个实施例中,图形多处理器1996是图19B的图形多处理器1934。在至少一个实施例中,图形多处理器1996与处理集群1994的管线管理器1932耦合。在至少一个实施例中,图形多处理器1996具有执行管线,该执行管线包括但不限于指令高速缓存1952、指令单元1954、地址映射单元1956、寄存器文件1958、一个或更多个GPGPU核心1962和一个或更多个LSU1966。GPGPU核心1962和LSU 1966与高速缓存存储器1972和共享存储器1970通过存储器和高速缓存互连1968耦合。

[0215] 在至少一个实施例中,指令高速缓存1952从管线管理器1932接收要执行的指令流。在至少一个实施例中,将指令高速缓存在指令高速缓存1952中并将其分派以供指令单元1954执行。在一个实施例中,指令单元1954可以分派指令作为线程组(例如,线程束),将线程组的每个线程分配给GPGPU核心1962内的不同执行单元。在至少一个实施例中,指令可以通过在统一地址空间内指定地址来访问任何本地、共享或全局地址空间。在至少一个实施例中,地址映射单元1956可以用于将统一地址空间中的地址转换成可以由LSU 1966访问的不同的存储器地址。

[0216] 在至少一个实施例中,寄存器文件1958为图形多处理器1996的功能单元提供了一组寄存器。在至少一个实施例中,寄存器文件1958为连接到图形多处理器1996的功能单元(例如,GPGPU核心1962、LSU 1966)的数据路径的操作数提供了临时存储。在至少一个实施例中,在每个功能单元之间划分寄存器文件1958,使得为每个功能单元分配寄存器文件1958的专用部分。在至少一个实施例中,寄存器文件1958在图形多处理器1996正在执行的不同线程组之间划分。

[0217] 在至少一个实施例中,GPGPU核心1962可以各自包括用于执行图多处理器1996的指令的FPU和/或ALU。GPGPU核心1962在架构上可以相似或架构可能有所不同。在至少一个实施例中,GPGPU核心1962的第一部分包括单精度FPU和整数ALU,而GPGPU核心1962的第二部分包括双精度FPU。在至少一个实施例中,FPU可以实现用于浮点算法的IEEE754-2008标准或启用可变精度浮点算法。在至少一个实施例中,图形多处理器1996可以另外包括一个或更多个固定功能或特殊功能单元,以执行特定功能,诸如复制矩形或像素混合操作。在至少一个实施例中,GPGPU核心1962中的一个或更多个也可以包括固定或特殊功能逻辑。

[0218] 在至少一个实施例中,GPGPU核心1962包括能够对多组数据执行单个指令的SIMD逻辑。在至少一个实施例中,GPGPU核心1962可以物理地执行SIMD4、SIMD8和SIMD9指令,并且在逻辑上执行SIMD1、SIMD2和SIMD32指令。在至少一个实施例中,用于GPGPU核心的SIMD

指令可以在编译时由着色器编译器生成,或者在执行针对单程序多数据 (SPMD) 或SIMT架构编写和编译的程序时自动生成。在至少一个实施例中,可以通过单个SIMD指令来执行为SIMT执行模型配置的程序的多个线程。例如,在至少一个实施例中,可以通过单个SIMD8逻辑单元并行执行执行相同或相似操作的八个SIMT线程。

[0219] 在至少一个实施例中,存储器和高速缓存互连1968是将图形多处理器1996的每个功能单元连接到寄存器文件1958和共享存储器1970的互连网络。在至少一个实施例中,存储器和高速缓存互连1968是交叉交换机互连,其允许LSU 1966在共享存储器1970和寄存器文件1958之间实现加载和存储操作。在至少一个实施例中,寄存器文件1958可以以与GPGPU核心1962相同的频率操作,从而在GPGPU核心1962和寄存器文件1958之间进行数据传输的延迟非常低。在至少一个实施例中,共享存储器1970可以用于启用在图形多处理器1996内的功能单元上执行的线程之间的通信。在至少一个实施例中,高速缓存存储器1972可以用作例如数据高速缓存,以高速缓存在功能单元和纹理单元1936之间通信的纹理数据。在至少一个实施例中,共享存储器1970也可以用作程序管理的高速缓存。在至少一个实施例中,除了存储在高速缓存存储器1972中的自动高速缓存的数据之外,在GPGPU核心1962上执行的线程还可以以编程方式将数据存储在共享存储器中。

[0220] 在至少一个实施例中,如本文所述的并行处理器或GPGPU通信地耦合到主机/处理器核心,以加速图形操作、机器学习操作、图案分析操作以及各种通用GPU (GPGPU) 功能。在至少一个实施例中,GPU可以通过总线或其他互连(例如,诸如PCIe或NVLink的高速互连)通信地耦合到主机处理器/核心。在至少一个实施例中,GPU可以与核心集成在相同的封装或芯片上,并通过内部处理器总线/互连(即,封装或芯片的内部)通信地耦合到核心。在至少一个实施例中,不管GPU连接的方式如何,处理器核心可以以WD包含的命令/指令序列的形式向GPU分配工作。在至少一个实施例中,GPU然后使用专用电路/逻辑来有效地处理这些命令/指令。

[0221] 图20示出了根据至少一个实施例的图形处理器2000。在至少一个实施例中,图形处理器2000包括环形互连2002、管线前端2004、媒体引擎2037和图形核心2080A-2080N。在至少一个实施例中,环形互连2002将图形处理器2000耦合到其他处理单元,包括其他图形处理器或一个或更多个通用处理器核心。在至少一个实施例中,图形处理器2000是集成在多核心处理系统内的许多处理器之一。

[0222] 在至少一个实施例中,图形处理器2000经由环形互连2002接收多批命令。在至少一个实施例中,输入命令由管线前端2004中的命令流转化器2003解释。在至少一个实施例中,图形处理器2000包括可缩放执行逻辑,以经由图形核心2080A-2080N执行3D几何处理和媒体处理。在至少一个实施例中,对于3D几何处理命令,命令流转化器2003将命令提供给几何管线2036。在至少一个实施例中,对于至少一些媒体处理命令,命令流转化器2003将命令提供给视频前端2034,其与媒体引擎2037耦合。在至少一个实施例中,媒体引擎2037包括用于视频和图像后处理的视频质量引擎 (VQE) 2030,以及用于提供硬件加速媒体数据编码和解码的多格式编码/解码 (MFX) 引擎2033。在至少一个实施例中,几何管线2036和媒体引擎2037各自生成用于由至少一个图形核心2080A提供的线程执行资源的执行线程。

[0223] 在至少一个实施例中,图形处理器2000包括以模块化图形核心2080A-2080N(有时称为核心切片)为特征的可缩放线程执行资源,每个模块核心具有多个子核心2050A-

2050N、2060A-2060N(有时称为核心子切片)。在至少一个实施例中,图形处理器2000可以具有任意数量的图形核心2080A至2080N。在至少一个实施例中,图形处理器2000包括具有至少第一子核心2050A和第二子核心2060A的图形核心2080A。在至少一个实施例中,图形处理器2000是具有单个子核心(例如子核心2050A)的低功率处理器。在至少一个实施例中,图形处理器2000包括多个图形核心2080A-2080N,每个图形核心包括一组第一子核心2050A-2050N和一组第二子核心2060A-2060N。在至少一个实施例中,第一子核心2050A-2050N中的每个子核心至少包括第一组执行单元(EU) 2052A-2052N和媒体/纹理采样器2054A-2054N。在至少一个实施例中,第二子核心2060A-2060N中的每个子核心至少包括第二组执行单元2062A-2062N和采样器2064A-2064N。在至少一个实施例中,每个子核心2050A-2050N、2060A-2060N共享一组共享资源2070A-2070N。在至少一个实施例中,共享资源2070包括共享高速缓冲存储器和像素操作逻辑。

[0224] 图21示出了根据至少一个实施例的用于处理器2100。在至少一个实施例中,处理器2100可以包括但不限于执行指令的逻辑电路。在至少一个实施例中,处理器2100可以执行指令,包括x86指令、ARM指令、用于ASIC的专用指令等。在至少一个实施例中,处理器2110可以包括用于存储封装数据的寄存器,例如作为加利福尼亚州圣克拉拉市英特尔公司采用MMX技术启用的微处理器中的64位宽MMXTM寄存器。在至少一个实施例中,整数和浮点数形式可用的MMX寄存器可以与封装的数据元素一起运行,所述封装的数据元素伴随SIMD和流式SIMD扩展(“SSE”)指令。在至少一个实施例中,与SSE2、SSE3、SSE4、AVX或更高版本(一般称为“SSEx”)技术有关的128位宽XMM寄存器可以保存此类封装数据操作数。在至少一个实施例中,处理器2110可以执行指令以加速CUAD程序。

[0225] 在至少一个实施例中,处理器2100包括有序前端(“前端”)2101,以提取要执行的指令并准备稍后在处理器管线中使用的指令。在至少一个实施例中,前端2101可以包括几个单元。在至少一个实施例中,指令预取器2126从存储器中获取指令并将指令提供给指令解码器2128,指令解码器2128又对指令进行解码或解释。例如,在至少一个实施例中,指令解码器2128将接收到的指令解码用于执行的所谓的“微指令”或“微操作”(也称为“微操作”或“微指令”)的一个或更多个操作。在至少一个实施例中,指令解码器2128将指令解析为操作码以及相应的数据和控制字段,其可以由微架构用来使用以执行操作。在至少一个实施例中,跟踪高速缓存2130可以将解码的微指令组装成微指令队列2134中的程序排序的序列或追踪以供执行。在至少一个实施例中,当追踪高速缓存2130遇到复杂指令时,微码ROM 2132提供完成操作所需的微指令。

[0226] 在至少一个实施例中,可以将一些指令转换成单个微操作,而另一些指令则需要几个微操作来完成全部操作。在至少一个实施例中,如果需要多于四个的微指令来完成一条指令,则指令解码器2128可以访问微码ROM 2132以执行指令。在至少一个实施例中,可以将指令解码为少量的微指令以在指令解码器2128处进行处理。在至少一个实施例中,如果需要多个微指令完成操作,则可以将指令存储在微码ROM 2132中。在至少一个实施例中,追踪高速缓存器2130参考入口点可编程逻辑数组(“PLA”)以确定正确的微指令指针,用于根据至少一个实施例从微码ROM 2132读取微码序列以完成一个或更多个指令。在至少一个实施例中,在微码ROM2132完成对指令的微操作排序之后,机器的前端2101可以恢复从追踪高速缓存2130获取微操作。

[0227] 在至少一个实施例中,乱序执行引擎(“乱序引擎”)2103可以准备用于执行的指令。在至少一个实施例中,乱序执行逻辑具有多个缓冲器,以使指令流平滑并重新排序,以在指令沿管线下降并被调度执行时优化性能。乱序执行引擎2103包括但不限于分配器/寄存器重命名器2140、存储器微指令队列2142、整数/浮点微指令队列2144、存储器调度器2146、快速调度器2102、慢速/通用浮点调度器(“慢速/通用FP调度器”)2104和简单浮点调度器(“简单FP调度器”)2106。在至少一个实施例中,快速调度器2102、慢速/通用浮点调度器2104和简单浮点调度器2106也统称为“微指令调度器2102、2104、2106”。分配器/寄存器重命名器2140分配每个微指令按顺序执行所需要的机器缓冲器和资源。在至少一个实施例中,分配器/寄存器重命名器2140将逻辑寄存器重命名为寄存器文件中的条目。在至少一个实施例中,分配器/寄存器重命名器2140还为两个微指令队列之一中的每个微指令分配条目,存储器微指令队列2142用于存储器操作和整数/浮点微指令队列2144用于非存储器操作,在存储器调度器2146和微指令调度器2102、2104、2106的前面。在至少一个实施例中,微指令调度器2102、2104、2106基于它们的从属输入寄存器操作数源的就绪性和需要完成的执行资源微指令的可用性来确定何时准备好执行微指令。在至少一个实施例中,至少一个实施例的快速调度器2102可以在主时钟周期的每个一半上调度,而慢速/通用浮点调度器2104和简单浮点调度器2106可以在每个主处理器时钟周期调度一次。在至少一个实施例中,微指令调度器2102、2104、2106对调度端口进行仲裁,以调度用于执行的微指令。

[0228] 在至少一个实施例中,执行块b11包括但不限于整数寄存器文件/支路网络2108、浮点寄存器文件/支路网络(“FP寄存器文件/支路网络”)2110、地址生成单元(“AGU”)2112和2114、快速算术逻辑单元(“快速ALU”)2116和2118、慢速ALU2120、浮点ALU(“FP”)2122和浮点移动单元(“FP移动”)2124。在至少一个实施例中,整数寄存器文件/旁路网络2108和浮点寄存器文件/旁路网络2110在本文中也称为“寄存器文件2108、2110”。在至少一个实施例中,AGU 2112和2114、快速ALU 2116和2118、慢速ALU 2120、浮点ALU 2122和浮点移动单元2124在本文中也称为“执行单元2112、2114、2116、2118、2120、2122和2124”。在至少一个实施例中,执行框可以包括但不限于任意数量(包括零)和类型的寄存器文件、支路网络、地址生成单元和执行单元(以任何组合)。

[0229] 在至少一个实施例中,寄存器文件2108、2110可以布置在微指令调度器2102、2104、2106与执行单元2112、2114、2116、2118、2120、2122和2124之间。在至少一个实施例中,整数寄存器文件/旁路网络2108执行整数运算。在至少一个实施例中,浮点寄存器文件/旁路网络2110执行浮点操作。在至少一个实施例中,寄存器文件2108、2110中的每一个可以包括但不限于支路网络,该支路网络可以绕过或转发尚未写入寄存器文件中的刚刚完成的结果到新的从属对象。在至少一个实施例中,寄存器文件2108、2110可以彼此通信数据。在至少一个实施例中,整数寄存器文件/支路网络2108可以包括但不限于两个单独的寄存器文件、一个寄存器文件用于低阶32位数据,第二寄存器文件用于高阶32位数据。在至少一个实施例中,浮点寄存器文件/支路网络2110可以包括但不限于128位宽的条目,因为浮点指令通常具有宽度为64至128位的操作数。

[0230] 在至少一个实施例中,执行单元2112、2114、2116、2118、2120、2122、2124可以执行指令。在至少一个实施例中,寄存器文件2108、2110存储微指令需要执行的整数和浮点数据操作数值。在至少一个实施例中,处理器2100可以包括但不限于任意数量的执行单元2112、

2114、2116、2118、2120、2122、2124及其组合。在至少一个实施例中,浮点ALU 2122和浮点移动单元2124,可以执行浮点、MMX、SIMD、AVX和SSE或其他操作,包括专门的机器学习指令。在至少一个实施例中,浮点ALU 2122可以包括但不限于64位乘64位浮点除法器,以执行除法、平方根和余数微操作。在至少一个实施例中,可以用浮点硬件来处理涉及浮点值的指令。在至少一个实施例中,可以将ALU操作传递给快速ALU 2116、2118。在至少一个实施例中,快速ALUS 2116、2118可以以半个时钟周期的有效延迟执行快速操作。在至少一个实施例中,大多数复杂的整数运算进入慢速ALU 2120,因为慢速ALU 2120可以包括但不限于用于长延迟类型操作的整数执行硬件,例如乘法器、移位、标志逻辑和分支处理。在至少一个实施例中,存储器加载/存储操作可以由AGUS 2112、2114执行。在至少一个实施例中,快速ALU 2116、快速ALU 2118和慢速ALU 2120可以对64位数据操作数执行整数运算。在至少一个实施例中,可以实现快速ALU2116、快速ALU 2118和慢速ALU 2120以支持包括16、32、128、256等的各种数据位大小。在至少一个实施例中,浮点ALU 2122和浮点移动单元2124可以实现为支持具有各种宽度的位的一定范围的操作数。在至少一个实施例中,浮点ALU 2122和浮点移动单元2124可以结合SIMD和多媒体指令对128位宽封装数据操作数进行操作。

[0231] 在至少一个实施例中,微指令调度器2102、2104、2106在父加载完成执行之前调度从属操作。在至少一个实施例中,由于可以在处理器2100中推测性地调度和执行微指令,处理器2100还可以包括用于处理存储器未命中的逻辑。在至少一个实施例中,如果数据高速缓存中的数据加载未命中,则可能存在在管线中正在运行的从属操作,其使调度器暂时没有正确的数据。在至少一个实施例中,一种重放机制追踪并重新执行使用不正确数据的指令。在至少一个实施例中,可能需要重放从属操作并且可以允许完成独立操作。在至少一个实施例中,处理器的至少一个实施例的调度器和重放机制也可以设计为捕获用于文本串比较操作的指令序列。

[0232] 在至少一个实施例中,术语“寄存器”可以指代可以用作识别操作数的指令的一部分的机载处理器存储位置。在至少一个实施例中,寄存器可以是那些可以从处理器外部使用的寄存器(从程序员的角度来看)。在至少一个实施例中,寄存器可能不限于特定类型的电路。相反,在至少一个实施例中,寄存器可以存储数据、提供数据并执行本文描述的功能。在至少一个实施例中,本文描述的寄存器可以通过处理器内的电路使用多种不同技术来实现,例如专用物理寄存器、使用寄存器重命名动态分配的物理寄存器、专用和动态分配的物理寄存器的组合等。在至少一个实施例中,整数寄存器存储32位整数数据。至少一个实施例的寄存器文件还包含八个用于封装数据的多媒体SIMD寄存器。

[0233] 图22示出了根据至少一个实施例的处理器2200。在至少一个实施例中,处理器2200包括但不限于一个或更多个处理器核心(核心) 2202A-2202N、集成存储器控制器2214和集成图形处理器2208。在至少一个实施例中,处理器2200可以包括直至并包括由虚线框表示的附加处理器核心2202N的附加核心。在至少一个实施例中,每个处理器核心2202A-2202N包括一个或更多个内部高速缓存单元2204A-2204N。在至少一个实施例中,每个处理器核心还可以访问一个或更多个共享高速缓存的单元2206。

[0234] 在至少一个实施例中,内部高速缓存单元2204A-2204N和共享高速缓存单元2206表示处理器2200内的高速缓存存储器层次结构。在至少一个实施例中,高速缓存存储器单元2204A-2204N可以包括每个处理器核心内的至少一级指令和数据以及共享中级缓存中的

一级或更多级缓存,例如L2、L3、4级(L4)或其他级别的缓存,其中在外部存储器之前将最高级别的缓存归类为LLC。在至少一个实施例中,高速缓存一致性逻辑维持各种高速缓存单元2206和2204A-2204N之间的一致性。

[0235] 在至少一个实施例中,处理器2200还可包括一组一个或多个总线控制器单元2216和系统代理核心2210。在至少一个实施例中,一个或多个总线控制器单元2216管理一组外围总线,例如一个或多个PCI或PCI Express总线。在至少一个实施例中,系统代理核心2210为各种处理器组件提供管理功能。在至少一个实施例中,系统代理核心2210包括一个或多个集成存储器控制器2214,以管理对各种外部存储器设备(未示出)的访问。

[0236] 在至少一个实施例中,一个或多个处理器核心2202A-2202N包括对多线程同时进行的支持。在至少一个实施例中,系统代理核心2210包括用于在多线程处理期间协调和操作处理器核心2202A-2202N的组件。在至少一个实施例中,系统代理核心2210可以另外包括电源控制单元(PCU),该电源控制单元包括逻辑和组件以调节处理器核心2202A-2202N和图形处理器2208的一个或多个电源状态。

[0237] 在至少一个实施例中,处理器2200另外包括图形处理器2208以执行图处理操作。在至少一个实施例中,图形处理器2208与共享高速缓存单元2206和包括一个或多个集成存储器控制器2214的系统代理核心2210耦合。在至少一个实施例中,系统代理核心2210还包括用于驱动图形处理器输出到一个或多个耦合的显示器的显示器控制器2211。在至少一个实施例中,显示器控制器2211也可以是经由至少一个互连与图形处理器2208耦合的独立模块,或者可以集成在图形处理器2208内。

[0238] 在至少一个实施例中,基于环的互连单元2212用于耦合处理器2200的内部组件。在至少一个实施例中,可以使用替代性互连单元,例如点对点互连、交换互连或其他技术。在至少一个实施例中,图形处理器2208经由I/O链路2213与环形互连2212耦合。

[0239] 在至少一个实施例中,I/O链路2213代表多种I/O互连中的至少一种,包括促进各种处理器组件与高性能嵌入式存储器模块2218(例如eDRAM模块)之间的通信的封装I/O互连。在至少一个实施例中,处理器核心2202A-2202N和图形处理器2208中的每一个使用嵌入式存储器模块2218作为共享的LLC。

[0240] 在至少一个实施例中,处理器核心2202A-2202N是执行公共指令集架构的同质核心。在至少一个实施例中,处理器核心2202A-2202N在ISA方面是异构的,其中一个或多个处理器核心2202A-2202N执行公共指令集,而一个或多个其他处理器核心2202A-2202N执行公共指令集或不同指令集的子集。在至少一个实施例中,就微架构而言,处理器核心2202A-2202N是异构的,其中具有相对较高功耗的一个或多个核心与具有较低功耗的一个或多个功率核心耦合。在至少一个实施例中,处理器2200可以实现在一个或多个芯片上或被实现为SoC集成电路。

[0241] 图23示出了根据所描述的至少一个实施例的图形处理器核心2300。在至少一个实施例中,图形处理器核心2300被包括在图形核心数组内。在至少一个实施例中,图形处理器核心2300(有时称为核心切片)可以是模块化图形处理器内的一个或多个图形核心。在至少一个实施例中,图形处理器核心2300是一个图形核心切片的示例,并且本文所述的图形处理器可以基于目标功率和性能包络线包括多个图形核心切片。在至少一个实施例中,每个图形核心2300可以包括与多个子核心2301A-2301F耦合的固定功能块2330,也称为子切

片,其包括通用和固定功能逻辑的模块块。

[0242] 在至少一个实施例中,固定功能块2330包括几何/固定功能管线2336,例如,在较低性能和/或较低功率的图形处理器实施方式中,该几何/固定功能管线2336可以由图形处理器2300中的所有子核心共享。在至少一个实施例中,几何/固定功能管线2336包括3D固定功能管线、视频前端单元,线程产生器和线程分派器以及管理统一返回缓冲器的统一返回缓冲器管理器。

[0243] 在至少一个实施例中,固定功能块2330还包括图形SoC接口2337、图形微控制器2338和媒体管线2339。图形SoC接口2337提供了图形核心2300以及SoC集成电路系统中的其他处理器核心之间的接口。在至少一个实施例中,图形微控制器2338是可编程子处理器,其可配置为管理图形处理器2300的各种功能,包括线程分派、调度和抢占。在至少一个实施例中,媒体管线2339包括有助于对包括图像和视频数据的多媒体数据进行解码、编码、预处理和/或后处理的逻辑。在至少一个实施例中,媒体管线2339经由对子核心2301-2301F内的计算或采样逻辑的请求来实现媒体操作。

[0244] 在至少一个实施例中,SoC接口2337使图形核心2300能够与通用应用处理器核心(例如,CPU)和/或SoC内的其他组件通信,包括存储器层次结构元素,诸如共享的LLC存储器、系统RAM和/或嵌入式片上或封装DRAM。在至少一个实施例中,SoC接口2337还可以使得能够与SoC内的固定功能设备(例如,相机成像管线)进行通信,并且使得能够使用和/或实现可以在图形核心2300和SoC内部的CPU之间共享的全局存储器原子。在至少一个实施例中,SoC接口2337还可以实现用于图形核心2300的电源管理控制,并且启用图形核心2300的时钟域与SoC内的其他时钟域之间的接口。在至少一个实施例中,SoC接口2337使得能够从命令流转化器和全局线程分派器接收命令缓冲器,其配置为向图形处理器内的一个或更多个图形核心中的每一个提供命令和指令。在至少一个实施例中,当要执行媒体操作时,可以将命令和指令分派给媒体管线2339,或者当要执行图处理操作时,可以将其分配给几何形状和固定功能管线(例如,几何形状和固定功能管线2336、几何形状和固定功能管线2314)。

[0245] 在至少一个实施例中,图形微控制器2338可以配置为对图形核心2300执行各种调度和管理任务。在至少一个实施例中,图形微控制器2338可以在子核心2301A-2301F中的执行单元(EU)数组2302A-2302F、2304A-2304F内的各种图形并行引擎上执行图和/或计算工作负载调度。在至少一个实施例中,在包括图形核心2300的SoC的CPU核心上执行的主机软件可以提交多个图形处理器门铃之一的工作负载,其调用适当的图形引擎上的调度操作。在至少一个实施例中,调度操作包括确定接下来要运行哪个工作负载、将工作负载提交给命令流转化器、抢先在引擎上运行的现有工作负载、监控工作负载的进度以及在工作负载完成时通知主机软件。在至少一个实施例中,图形微控制器2338还可以促进图形核心2300的低功率或空闲状态,从而为图形核心2300提供在图形核心2300内独立于操作系统和/或系统上的图形驱动器软件的跨低功率状态转换的保存和恢复寄存器的能力。

[0246] 在至少一个实施例中,图形核心2300可以具有比所示的子核心2301A-2301F更多或更少的子核心,达N个模块化子核心。对于每组N个子核心,在至少一个实施例中,图形核心2300还可以包括共享功能逻辑2310、共享和/或高速缓存存储器2312、几何/固定功能管线2314以及附加的固定功能逻辑2316以加速各种图形和计算处理操作。在至少一个实施例中,共享功能逻辑2310可以包括可由图形核心2300内的每个N个子核心共享的逻辑单元(例

如,采样器、数学和/或线程间通信逻辑)。共享和/或高速缓存存储器2312可以是图形核心2300内的N个子核心2301A-2301F的LLC,并且还可以用作可由多个子核心访问的共享存储器。在至少一个实施例中,可以包括几何/固定功能管线2314来代替固定功能块2330内的几何/固定功能管线2336,并且可以包括相同或相似的逻辑单元。

[0247] 在至少一个实施例中,图形核心2300包括附加的固定功能逻辑2316,其可以包括供图形核心2300使用的各种固定功能加速逻辑。在至少一个实施例中,附加的固定功能逻辑2316包括用于仅位置着色中使用的附加的几何管线。在仅位置着色中,存在至少两个几何管线,而在几何/固定功能管线2316、2336内的完整几何管线和剔除管线中,其是可以包括在附加的固定功能逻辑2316中的附加几何管线。在至少一个实施例中,剔除管线是完整几何管线的修整版。在至少一个实施例中,完整管线和剔除管线可以执行应用程序的不同实例,每个实例具有单独的环境。在至少一个实施例中,仅位置着色可以隐藏被丢弃的三角形的长剔除运行,从而在某些情况下可以更早地完成着色。例如,在至少一个实施例中,附加固定功能逻辑2316中的剔除管线逻辑可以与主应用程序并行执行位置着色器,并且通常比完整管线更快地生成关键结果,因为剔除管线获取并遮蔽顶点的位置属性,无需执行光栅化和将像素渲染到帧缓冲器。在至少一个实施例中,剔除管线可以使用生成的临界结果来计算所有三角形的可见性信息,而与这些三角形是否被剔除无关。在至少一个实施例中,完整管线(在这种情况下可以称为重播管线)可以消耗可见性信息来跳过剔除的三角形以仅遮盖最终传递到光栅化阶段的可见三角形。

[0248] 在至少一个实施例中,附加的固定功能逻辑2316还可包括通用目标处理加速逻辑,例如固定功能矩阵乘法逻辑,用于实现减速CUAD程序。

[0249] 在至少一个实施例中,在每个图形子核心2301A-2301F内包括一组执行资源,其可用于响应于图形管线、媒体管线或着色器程序的请求来执行图、媒体和计算操作。在至少一个实施例中,图形子核心2301A-2301F包括多个EU数组2302A-2302F、2304A-2304F,线程分派和线程间通信(TD/IC)逻辑2303A-2303F,3D(例如,纹理)采样器2305A-2305F,媒体采样器2306A-2306F,着色器处理器2307A-2307F和共享本地存储器(SLM)2308A-2308F。EU数组2302A-2302F、2304A-2304F每个都包含多个执行单元,这些执行单元是GUGPU,能够为图形、媒体或计算操作提供服务,执行浮点和整数/定点逻辑运算,包括图形、媒体或计算着色器程序。在至少一个实施例中,TD/IC逻辑2303A-2303F为子核心内的执行单元执行本地线程分派和线程控制操作,并促进在子核心的执行单元上执行的线程之间的通信。在至少一个实施例中,3D采样器2305A-2305F可以将与纹理或其他3D图形相关的数据读取到存储器中。在至少一个实施例中,3D采样器可以基于与给定纹理相关联的配置的采样状态和纹理格式来不同地读取纹理数据。在至少一个实施例中,媒体采样器2306A-2306F可以基于与媒体数据相关联的类型和格式来执行类似的读取操作。在至少一个实施例中,每个图形子核心2301A-2301F可以可替代地包括统一的3D和媒体采样器。在至少一个实施例中,在每个子核心2301A-2301F内的执行单元上执行的线程可以利用每个子核心内的共享本地存储器2308A-2308F,以使在线程组内执行的线程能够使用片上存储器的公共池来执行。

[0250] 图24示出了根据至少一个实施例的并行处理单元(“PPU”)2400。在至少一个实施例中,PPU 2400配置有机器可读代码,该机器可读代码如果由PPU 2400执行,则使得PPU 2400执行贯穿本文描述的一些或全部过程和技术。在至少一个实施例中,PPU 2400是在一

个或更多个集成电路设备上实现的多线程处理器,并且利用多线程作为被设计为处理在多个线程上并行执行的计算机可读指令(也称为机器可读指令或简单的指令)的延迟隐藏技术。在至少一个实施例中,线程是指执行线程,并且是被配置为由PPU 2400执行的一组指令的实例。在至少一个实施例中,PPU 2400是图形处理单元(“GPU”),图形处理单元配置为实现用于处理三维(“3D”)图形数据的图形渲染管线,以便生成用于在显示设备(诸如LCD设备)上显示的二维(“2D”)图像数据。在至少一个实施例中,PPU 2400用于执行计算,诸如线性代数运算和机器学习运算。图24仅出于说明性目的示出了示例并行处理器,并且应被解释为在至少一个实施例中实现的处理器架构的非限制性示例。

[0251] 在至少一个实施例中,一个或更多个PPU 2400配置成加速高性能计算(“HPC”)、数据中心和机器学习应用程序。在至少一个实施例中,一个或更多个PPU 2400配置成加速CUDA程序。在至少一个实施例中,PPU2400包括但不限于I/O单元2406、前端单元2410、调度器单元2412、工作分配单元2414、集线器2416、交叉交换机(“Xbar”)2420、一个或更多个通用处理集群(“GPC”)2418和一个或更多个分区单元(“存储器分区单元”)2422。在至少一个实施例中,PPU 2400通过一个或更多个高速GPU互连(“GPU互连”)2408连接到主机处理器或其他PPU 2400。在至少一个实施例中,PPU 2400通过互连2402连接到主机处理器或其他外围设备。在一实施例中,PPU 2400连接到包括一个或更多个存储器设备(“存储器”)2404的本地存储器。在至少一个实施例中,存储器设备2404包括但不限于一个或更多个动态随机存取存储器(“DRAM”)设备。在至少一个实施例中,一个或更多个DRAM设备配置和/或可配置为高带宽存储器(“HBM”)子系统,并且在每个设备内堆叠有多个DRAM管芯。

[0252] 在至少一个实施例中,高速GPU互连2408可以指代系统使用其来进行缩放的基于线的多通道通信链路,并包括与一个或更多个CPU结合的一个或更多个PPU 2400(“CPU”),支持PPU 2400和CPU之间的高速缓存一致性以及CPU主控。在至少一个实施例中,高速GPU互连2408通过集线器2416将数据和/或命令传输到PPU 2400的其他单元,例如一个或更多个复制引擎、视频编码器、视频解码器、电源管理单元和/或在图24中可能未明确示出的其他组件。

[0253] 在至少一个实施例中,I/O单元2406配置为通过系统总线2402从主机处理器(图24中未示出)发送和接收通信(例如,命令、数据)。在至少一个实施例中,I/O单元2406直接通过系统总线2402或通过一个或更多个中间设备(例如存储器桥)与主机处理器通信。在至少一个实施例中,I/O单元2406可以经由系统总线2402与一个或更多个其他处理器(例如一个或更多个PPU 2400)通信。在至少一个实施例中,I/O单元2406实现PCIe接口,用于通过PCIe总线进行通信。在至少一个实施例中,I/O单元2406实现用于与外部设备通信的接口。

[0254] 在至少一个实施例中,I/O单元2406对经由系统总线2402接收的分组进行解码。在至少一个实施例中,至少一些分组表示被配置为使PPU2400执行各种操作的命令。在至少一个实施例中,I/O单元2406如命令所指定的那样将解码的命令发送到PPU 2400的各种其他单元。在至少一个实施例中,命令被发送到前端单元2410和/或被发送到集线器2416或PPU2400的其他单元,例如一个或更多个复制引擎、视频编码器、视频解码器、电源管理单元等(图24中未明确示出)。在至少一个实施例中,I/O单元2406配置为在PPU 2400的各种逻辑单元之间路由通信。

[0255] 在至少一个实施例中,由主机处理器执行的程序在缓冲器中对命令流进行编码,

该缓冲器将工作负载提供给PPU 2400以进行处理。在至少一个实施例中,工作负载包括指令和要由那些指令处理的数据。在至少一个实施例中,缓冲器是可由主机处理器和PPU 2400两者访问(例如,读/写)的存储器中的区域—主机接口单元可以配置为访问经由I/O单元2406通过系统总线2402传输的存储器请求连接到系统总线2402的系统存储器中的缓冲器。在至少一个实施例中,主机处理器将命令流写入缓冲器,然后将指示命令流开始的指针发送给PPU 2400,使得前端单元2410接收指向一个或更多个命令流指针并管理一个或更多个命令流,从命令流中读取命令并将命令转发到PPU 2400的各个单元。

[0256] 在至少一个实施例中,前端单元2410耦合到调度器单元2412,该调度器单元2412配置各种GPC 2418以处理由一个或更多个命令流定义的任务。在至少一个实施例中,调度器单元2412配置为跟踪与调度器单元2412管理的各种任务有关的状态信息,其中状态信息可以指示任务被分配给哪个GPC 2418,任务是活跃的还是非活跃的,与任务相关联的优先级等等。在至少一个实施例中,调度器单元2412管理在一个或更多个GPC 2418上执行的多个任务。

[0257] 在至少一个实施例中,调度器单元2412耦合到工作分配单元2414,该工作分配单元2414配置为分派任务以在GPC 2418上执行。在至少一个实施例中,工作分配单元2414跟踪从调度器单元2412接收到的多个调度任务并且工作分配单元2414管理每个GPC 2418的待处理任务池和活跃任务池。在至少一个实施例中,待处理任务池包括多个时隙(例如32个时隙),这些时隙包含分配给要由特定的GPC 2418处理的任务;活跃任务池可包括用于由GPC 2418主动处理的任务的多个时隙(例如4个时隙),以使随着GPC 2418中的一个完成任务的执行,该任务将从GPC 2418的活动任务池中逐出,并且从待处理任务池中选择其他任务之一,并安排其在GPC2418上执行。在至少一个实施例中,如果活跃任务在GPC 2418上处于空闲状态,例如在等待数据依赖性解决时,则活跃任务从GPC 2418中驱逐并返回到待处理任务池,同时选择了待处理任务池中的另一个任务并调度在GPC 2418上执行。

[0258] 在至少一个实施例中,工作分配单元2414经由XBar2420与一个或更多个GPC 2418通信。在至少一个实施例中,XBar2420是互连网络,其将PPU 2400的许多单元耦合到PPU 2400的其他单元,并且可以配置为将工作分配单元2414耦合到特定的GPC2418。在至少一个实施例中,一个或更多个PPU 2400的其他单元也可以通过集线器2416连接到XBar2420。

[0259] 在至少一个实施例中,任务由调度器单元2412管理,并由工作分配单元2414分配给GPC 2418之一。GPC 2418配置为处理任务并产生结果。在至少一个实施例中,结果可以由GPC 2418中的其他任务消耗,通过XBar2420路由到不同的GPC 2418或存储在存储器2404中。在至少一个实施例中,结果可以通过分区单元2422写到存储器2404中,其实现了用于向存储器2404写入数据或从存储器2404读取数据的存储器接口。在至少一个实施例中,结果可以经由高速GPU互连2408传输到另一PPU 2400或CPU。在至少一个实施例中,PPU 2400包括但不限于U个分区单元2422,其等于耦合到PPU 2400的分离且不同的存储器设备2404的数量。

[0260] 在至少一个实施例中,主机处理器执行驱动器核心,该驱动器核心实现应用程序编程接口(API),该应用程序编程接口使在主机处理器上执行的一个或更多个应用程序能够调度操作以在PPU 2400上执行。在一个实施例中,多个计算应用由PPU 2400同时执行,并且PPU 2400为多个计算应用程序提供隔离、服务质量(“QoS”)和独立的地址空间。在至少一

个实施例中,应用程序生成指令(例如,以API调用的形式),该指令使驱动器核心生成一个或多个任务以供PPU 2400执行,并且驱动器核心将任务输出至由PPU 2400处理的一个或多个流。在至少一个实施例中,每个任务包括一个或多个相关线程组,其可以被称为线程束(warp)。在至少一个实施例中,线程束包括可以并行执行的多个相关线程(例如32个线程)。在至少一个实施例中,协作线程可以指代多个线程,包括用于执行任务并且通过共享存储器交换数据的指令。

[0261] 图25示出了根据至少一个实施例的GPC2500。在至少一个实施例中,GPC 2500是图24的GPC 2418。在至少一个实施例中,每个GPC 2500包括但不限于用于处理任务的多个硬件单元,并且每个GPC 2500包括但不限于管线管理器2502、预光栅操作单元(“PROP”)2504、光栅引擎2508、工作分配交叉交换机(“WDX”)2516、存储器管理单元(“MMU”)2518、一个或多个数据处理集群(“DPC”)2506,以及部件的任何合适组合。

[0262] 在至少一个实施例中,GPC 2500的操作由管线管理器2502控制。在至少一个实施例中,管线管理器2502管理一个或多个DPC 2506的配置,以处理分配给GPC 2500的任务。在至少一个实施例中,管线管理器2502配置一个或多个DPC 2506中的至少一个以实现图形渲染管线的至少一部分。在至少一个实施例中,DPC 2506配置为在可编程流式多处理器(“SM”)2514上执行顶点着色器程序。在至少一个实施例中,管线管理器2502配置为将从工作分配单元接收的数据包路由到GPC 2500内的适当逻辑单元,以及在至少一个实施例中,可以将一些数据包路由到PROP 2504和/或光栅引擎2508中的固定功能硬件单元,而可以将其他数据包路由到DPC 2506以由原始引擎2512或SM2514进行处理。在至少一个实施例中,管线管理器2502配置DPC 2506中的至少一个以实现神经网络模型和/或计算管线。在至少一个实施例中,管线管理器2502配置DPC 2506中的至少一个以执行CUDA程序的至少一部分。

[0263] 在至少一个实施例中,PROP单元2504配置为将由光栅引擎2508和DPC 2506生成的数据路由到分区单元中的光栅操作(“ROP”)单元,例如上面结合图24更详细描述存储器分区单元2422等。在至少一个实施例中,PROP单元2504配置为执行用于颜色混合的优化、组织像素数据、执行地址转换等等。在至少一个实施例中,光栅引擎2508包括但不限于配置为执行各种光栅操作的多个固定功能硬件单元,并且在至少一个实施例中,光栅引擎2508包括但不限于设置引擎、粗光栅引擎、剔除引擎、裁剪引擎、精细光栅引擎、瓦片聚合引擎及其任意合适的组合。在至少一个实施例中,设置引擎接收变换后的顶点并生成与由顶点定义的几何图元相关联的平面方程;平面方程式被传送到粗光栅引擎以生成基本图元的覆盖信息(例如,瓦片的x、y覆盖范围掩码);粗光栅引擎的输出将传输到剔除引擎,在剔除引擎中与z测试失败的图元相关联的片段将被剔除,并传输到剪切引擎,在剪切引擎中剪切位于视锥范围之外的片段。在至少一个实施例中,将经过裁剪和剔除的片段传递给精细光栅引擎,以基于设置引擎生成的平面方程式生成像素片段的属性。在至少一个实施例中,光栅引擎2508的输出包括将由任何适当的实体(例如,由在DPC 2506内实现的片段着色器)处理的片段。

[0264] 在至少一个实施例中,包括在GPC 2500中的每个DPC 2506包括但不限于M管线控制器(“MPC”)2510;图元引擎2512;一个或多个SM2514;及其任何合适的组合。在至少一个实施例中,MPC 2510控制DPC2506的操作,将从管线管理器2502接收的分组路由到DPC 2506

中的适当单元。在至少一个实施例中,将与顶点相关联的分组路由到图元引擎2512,图元引擎2512配置为从存储器中获取与顶点关联的顶点属性;相反,可以将与着色器程序相关联的数据包发送到SM 2514。

[0265] 在至少一个实施例中,SM 2514包括但不限于可编程流式处理器,其配置为处理由多个线程表示的任务。在至少一个实施例中,SM 2514是多线程的并且配置为同时执行来自特定线程组的多个线程(例如32个线程),并且实现单指令、多数据(“SIMD”)架构,其中将一组线程(例如,线程束)中的每个线程配置为基于相同的指令集来处理不同的数据集。在至少一个实施例中,线程组中的所有线程执行相同的指令。在至少一个实施例中,SM 2514实施单指令、多线程(“SIMT”)架构,其中一组线程中的每个线程配置为基于相同的指令集来处理不同的数据集,但是其中线程组中的各个线程允许在执行期间发散。在至少一个实施例中,为每个线程束维护程序计数器、调用栈和执行状态,从而当线程束中的线程发散时,实现线程束和线程束内的串行执行之间的并发性。在另一个实施例中,为每个单独的线程维护程序计数器、调用栈和执行状态,从而使得在线程束内和线程束之间的所有线程之间具有相等的并发性。在至少一个实施例中,为每个单独的线程维持执行状态,并且可以收敛并并行地执行执行相同指令的线程以提高效率。下面结合图26更详细地描述SM 2514的至少一个实施例。

[0266] 在至少一个实施例中,MMU 2518在GPC 2500和存储器分区单元(例如,图24的分区单元2422)之间提供接口,并且MMU 2518提供虚拟地址到物理地址的转换、存储器保护以及存储器请求的仲裁。在至少一个实施例中,MMU 2518提供一个或多个转换后备缓冲器(“TLB”),用于执行虚拟地址到存储器中的物理地址的转换。

[0267] 图26示出了根据至少一个实施例的流式多处理器(“SM”)2600。在至少一个实施例中,SM 2600是图25的SM 2514。在至少一个实施例中,SM 2600包括但不限于指令高速缓存2602;一个或多个调度器单元2604;寄存器文件2608;一个或多个处理核心(“核心”)2610;一个或多个特殊功能单元(“SFU”)2612;一个或多个加载/存储单元(“LSU”)2614;互连网络2616;共享存储器/一级(“L1”)高速缓存2618;及其任何合适的组合。在至少一个实施例中,工作分配单元调度任务以在并行处理单元(“PPU”)的通用处理集群(“GPC”)上执行,并且每个任务被分配给GPC内部的特定数据处理集群(“DPC”),并且如果任务与着色器程序相关联,则将任务分配给SM 2600之一。在至少一个实施例中,调度器单元2604从工作分配单元接收任务并管理分配给SM 2600的一个或多个线程块的指令调度。在至少一个实施例中,调度器单元2604调度线程块以作为并行线程的线程束来执行,其中每个线程块被分配至少一个线程束。在至少一个实施例中,每个线程束执行线程。在至少一个实施例中,调度器单元2604管理多个不同的线程块,将线程束分配给不同的线程块,然后在每个时钟周期内将来自多个不同的协作组的指令分派给各种功能单元(例如,处理核心2610、SFU 2612和LSU 2614)。

[0268] 在至少一个实施例中,“合作组”可以指用于组织通信线程组的编程模型,其允许开发人员表达线程正在通信的粒度,从而能够表达更丰富、更有效的并行分解。在至少一个实施例中,协作启动API支持线程块之间的同步以执行并行算法。在至少一个实施例中,常规编程模型的API提供了用于同步协作线程的单一、简单的构造:跨线程块的所有线程的屏障(例如,syncthreads()函数)。但是,在至少一个实施例中,程序员可以在小于线程块粒

度的情形下来定义线程组,并在所定义的组内进行同步,以实现更高的性能、设计灵活性以及以集合组范围功能接口的形式实现软件重用。在至少一个实施例中,协作组使程序员能够以子块和多块粒度明确定义线程组,并执行集合操作,例如对协作组中的线程进行同步。在至少一个实施例中,子块粒度与单个线程一样小。在至少一个实施例中,编程模型支持跨软件边界的干净组合,从而库和实用程序功能可以在其本地环境中安全地同步,而不必进行关于收敛的假设。在至少一个实施例中,协作组图元使协作并行的新图案成为可能,包括但不限于生产者-消费者并行,机会主义并行以及整个线程块网格上的全局同步。

[0269] 在至少一个实施例中,分派单元2606配置为将指令发送到功能单元中的一个或多个,并且调度器单元2604包括但不限于两个分派单元2606,该两个分派单元2606使得来自相同线程束的两个不同指令能够在每个时钟周期被分派。在至少一个实施例中,每个调度器单元2604包括单个分派单元2606或附加分派单元2606。

[0270] 在至少一个实施例中,每个SM 2600在至少一个实施例中包括但不限于寄存器文件2608,该寄存器文件2608为SM 2600的功能单元提供了一组寄存器。在至少一个实施例中,寄存器文件2608在每个功能单元之间划分,从而为每个功能单元分配寄存器文件2608的专用部分。在至少一个实施例中,寄存器文件2608在由SM 2600执行的不同线程束之间划分,并且寄存器文件2608为连接到功能单元的数据路径的操作数提供临时存储。在至少一个实施例中,每个SM 2600包括但不限于多个L个处理核心2610。在至少一个实施例中,SM 2600包括但不限于大量(例如128个或更多)不同的处理核心2610。在至少一个实施例中,每个处理核心2610在至少一个实施例中包括但不限于全管线、单精度、双精度和/或混合精度处理单元,其包括但不限于浮点算术逻辑单元和整数算术逻辑单元。在至少一个实施例中,浮点算术逻辑单元实现用于浮点算术的IEEE 754-2008标准。在至少一个实施例中,处理核心2610包括但不限于64个单精度(32位)浮点核心、64个整数核心、32个双精度(64位)浮点核心和8个张量核心。

[0271] 在至少一个实施例中,张量核心配置为执行矩阵运算。在至少一个实施例中,一个或多个张量核心包括在处理核心2610中。在至少一个实施例中,张量核心配置为执行深度学习矩阵算术,例如用于神经网络训练和推理的卷积运算。在至少一个实施例中,每个张量核心在 $4 \times 4$ 矩阵上操作并且执行矩阵乘法和累加运算 $D = A \times B + C$ ,其中A、B、C和D是 $4 \times 4$ 矩阵。

[0272] 在至少一个实施例中,矩阵乘法输入A和B是16位浮点矩阵,并且累加矩阵C和D是16位浮点或32位浮点矩阵。在至少一个实施例中,张量核心对16位浮点输入数据进行32位浮点累加运算。在至少一个实施例中,16位浮点乘法使用64个运算,并得到全精度乘积,然后使用32位浮点加法与其他中间乘积累加起来,以进行 $4 \times 4 \times 4$ 矩阵乘法。在至少一个实施例中,张量核心用于执行由这些较小的元件构成的更大的二维或更高维度的矩阵运算。在至少一个实施例中,API(诸如CUDA-C++API)公开专门的矩阵加载、矩阵乘法和累加以及矩阵存储操作,以有效地使用来自CUDA-C++程序的张量核心。在至少一个实施例中,在CUDA级别,线程束级别接口假定跨越所有32个线程束线程的 $16 \times 16$ 大小的矩阵。

[0273] 在至少一个实施例中,每个SM 2600包括但不限于执行特殊功能(例如,属性评估、倒数平方根等)的M个SFU 2612。在至少一个实施例中,SFU 2612包括但不限于配置为遍历分层树数据结构的树遍历单元。在至少一个实施例中,SFU 2612包括但不限于配置为执行

纹理映射过滤操作的纹理单元。在至少一个实施例中,纹理单元配置为从存储器中加载纹理映射(例如,纹理像素的2D数组)和采样纹理映射,以产生采样的纹理值以供由SM 2600执行的着色器程序使用。在至少一个实施例中,将纹理映射存储在共享存储器/L1高速缓存2618中。在至少一个实施例中,纹理单元使用mip映射(mip-maps)(例如,细节级别不同的纹理映射)来实现纹理操作(诸如过滤操作)。在至少一个实施例中,每个SM 2600包括但不限于两个纹理单元。

[0274] 在至少一个实施例中,每个SM 2600包括但不限于实现共享存储器/L1高速缓存2618与寄存器文件2608之间的加载和存储操作的N个LSU2614。在至少一个实施例中,每个SM 2600包括但不限于互连网络2616,互连网络2616将每个功能单元连接到寄存器文件2608,并且LSU 2614连接到寄存器文件2608和共享存储器/L1高速缓存2618。在至少一个实施例中,互连网络2616是交叉交换机,其可以配置为将任何功能单元连接到寄存器文件2608中的任何寄存器,并且将LSU 2614连接到寄存器文件2608和共享存储器/L1高速缓存2618中的存储器位置。

[0275] 在至少一个实施例中,共享存储器/L1高速缓存2618是片上存储器的数组,其在至少一个实施例中允许SM 2600与图元引擎之间以及SM2600中的线程之间的数据存储和通信。在至少一个实施例中,共享存储器/L1高速缓存2618包括但不限于128KB的存储容量,并且位于从SM 2600到分区单元的路径中。在至少一个实施例中,共享存储器/L1高速缓存2618在至少一个实施例中用于高速缓存读取和写入。在至少一个实施例中,共享存储器/L1高速缓存2618、L2高速缓存和存储器中的一个或多个是后备存储。

[0276] 在至少一个实施例中,将数据高速缓存和共享存储器功能组合到单个存储器块中,为两种类型的存储器访问提供了改进的性能。在至少一个实施例中,容量由不使用共享存储器的程序使用或将其用作高速缓存,例如如果共享存储器配置为使用一半容量,则纹理和加载/存储操作可以使用剩余容量。根据至少一个实施例,在共享存储器/L1高速缓存2618内的集成使共享存储器/L1高速缓存2618能够用作用于流传输数据的高吞吐量管线,同时提供对频繁重用的数据的高带宽和低延迟访问。在至少一个实施例中,当配置用于通用并行计算时,与图形处理相比,可以使用更简单的配置。在至少一个实施例中,绕过固定功能GPU,从而创建了更加简单的编程模型。在至少一个实施例中,在通用并行计算配置中,工作分配单元直接将线程的块分配和分布给DPC。在至少一个实施例中,块中的线程执行相同的程序,在计算中使用唯一的线程ID以确保每个线程生成唯一的结果,使用SM 2600执行程序并执行计算,使用共享存储器/L1高速缓存2618在线程之间进行通信,以及使用LSU 2614通过共享存储器/L1高速缓存2618和存储器分区单元来读写全局存储器。在至少一个实施例中,当被配置用于通用并行计算时,SM 2600向调度器单元2604写入可以用来在DPC上启动新工作的命令。

[0277] 在至少一个实施例中,PPU被包括在台式计算机、膝上型计算机、平板电脑、服务器、超级计算机、智能电话(例如,无线、手持设备)、PDA、数码相机、车辆、头戴式显示器、手持式电子设备等中或与之耦合。在至少一个实施例中,PPU被实现在单个半导体衬底上。在至少一个实施例中,PPU与一个或多个其他设备(例如附加的PPU、存储器、RISCCPU,MMU、数模转换器(“DAC”)等)一起被包括在片上系统(“SoC”)中。

[0278] 在至少一个实施例中,PPU可以被包括在包括一个或多个存储设备的图形卡上。

图形卡可以配置为与台式计算机主板上的PCIe插槽相连接。在至少一个实施例中,PPU可以是包括在主板的芯片组中的集成GPU(“iGPU”)。

[0279] 用于通用计算的软件构造

[0280] 以下各图阐述但不限于用于实现至少一个实施例的示例性软件构造。在至少一个实施例中,以下附图的一个或多个基于计算机的系统可以实现参照图1A描述的实施例,参照图1B描述的实施例,参照图2描述的实施例,参照图3描述的技术300,参照图4描述的技术400,参考图5描述的技术500,参照图6描述的技术600和/或参照图7描述的技术700的一个或多个方面。

[0281] 图27示出了根据至少一个实施例的编程平台的软件栈。在至少一个实施例中,编程平台是用于利用计算系统上的硬件来加速计算任务的平台。在至少一个实施例中,软件开发人员可以通过库、编译器指令和/或对编程语言的扩展来访问编程平台。在至少一个实施例中,编程平台可以是但不限于CUDA,Radeon开放计算平台(“ROCm”),OpenCL(由Khronosgroup开发的OpenCL™),SYCL或Intel One API。

[0282] 在至少一个实施例中,编程平台的软件栈2700为应用程序2701提供执行环境。在至少一个实施例中,应用程序2701可以包括能够在软件栈2700上启动的任何计算机软件。在至少一个实施例中,应用程序2701可以包括但不限于人工智能(“AI”)/机器学习(“ML”)应用程序,高性能计算(“HPC”)应用程序,虚拟桌面基础架构(“VDI”)或数据中心工作负载。

[0283] 在至少一个实施例中,应用程序2701和软件栈2700在硬件2707上运行。在至少一个实施例中,硬件2707可以包括一个或多个GPU,CPU,FPGA,AI引擎和/或支持编程平台的其他类型的计算设备。在至少一个实施例中,例如采用CUDA,软件栈2700可以是厂商专用的,并且仅与来自特定厂商的设备兼容。在至少一个实施例中,例如在采用OpenCL中,软件栈2700可以与来自不同供应商的设备一起使用。在至少一个实施例中,硬件2707包括连接到一个或多个设备的主机,该设备可经由应用程序编程接口(API)调用被访问以执行计算任务。在至少一个实施例中,与硬件2707内的主机相比,其可以包括但不限于CPU(但还可以包括计算设备)及其存储器,硬件2707内的设备可以包括但不限于GPU,FPGA,AI引擎或其他计算设备(但还可以包括CPU)及其存储器。

[0284] 在至少一个实施例中,编程平台的软件栈2700包括但不限于多个库2703,运行时(runtime)2705和设备内核驱动器2706。在至少一个实施例中,库2703中的每个库可以包括可以由计算机程序使用并在软件开发期间利用的数据和编程代码。在至少一个实施例中,库2703可以包括但不限于预写的代码和子例程,类,值,类型规范,配置数据,文档,帮助数据和/或消息模板。在至少一个实施例中,库2703包括被优化用于在一种或更多种类型的设备上执行的功能。在至少一个实施例中,库2703可以包括但不限于用于在设备上执行数学、深度学习和/或其他类型的运算的功能。在至少一个实施例中,库2803与对应的API 2802相关联,API 2802可包括一个或多个API,其暴露在库2803中实现的功能。

[0285] 在至少一个实施例中,将应用程序2701编写为源代码,该源代码被编译成可执行代码,如下面结合图32-34更详细讨论的。在至少一个实施例中,应用程序2701的可执行代码可以至少部分地在由软件栈2700提供的执行环境上运行。在至少一个实施例中,在应用程序2701的执行期间,可以得到需要在设备(与主机相比)上运行的代码。在这种情况下,在至少一个实施例中,可以调用运行时2705以在设备上加载和启动必需的代码。在至少一个

实施例中,运行时2705可以包括能够支持应用程序2701的执行的任何技术上可行的运行时系统。

[0286] 在至少一个实施例中,运行时2705被实现为与对应的API (其被示为API2704) 相关的一个或更多个运行时库。在至少一个实施例中,一个或更多个这样的运行时库可以包括但不限于用于存储器管理,执行控制,设备管理,错误处理和/或同步等的功能。在至少一个实施例中,存储器管理功能可以包括但不限于用于分配、解除分配和复制设备存储器以及在主机存储器和设备存储器之间传输数据的功能。在至少一个实施例中,执行控制功能可以包括但不限于在设备上启动函数(当函数是可从主机调用的全局函数时,有时称为“内核”)的功能,和用于在运行时库为要在设备上执行的给定功能维护的缓冲器中设置属性值的功能。

[0287] 在至少一个实施例中,可以任何技术上可行的方式来实现运行时库和相应的API 2704。在至少一个实施例中,一个(或任意数量的)API可以公开用于设备的细粒度控制的低级功能集,而另一(或任意数量的)API可以公开这样的较高级的功能集。在至少一个实施例中,可以在低级API之上构建高级运行时API。在至少一个实施例中,一个或更多个运行时API可以是在与语言无关的运行时API之上分层的特定于语言的API。

[0288] 在至少一个实施例中,设备内核驱动器2706被配置为促进与底层设备的通信。在至少一个实施例中,设备内核驱动器2706可以提供诸如API2704之类的API和/或其他软件所依赖的低级功能。在至少一个实施例中,设备内核驱动器2706可以被配置为在运行时将中间表示(“IR”)代码编译成二进制代码。在至少一个实施例中,对于CUDA,设备内核驱动器2706可以在运行时将非硬件专用的并行线程执行(“PTX”)IR代码编译为用于特定目标设备的二进制代码(高速缓存已编译的二进制代码),其有时也称为“最终”代码。在至少一个实施例中,这样做可以允许最终代码在目标设备上运行,而当源代码最初被编译为PTX代码时,该目标设备可能不存在。备选地,在至少一个实施例中,设备源代码可以离线地编译成二进制代码,而不需要设备内核驱动器2706在运行时编译IR代码。

[0289] 图28示出了根据至少一个实施例的图27的软件栈2700的CUDA实现。在至少一个实施例中,可在其上启动应用程序2801的CUDA软件栈2800包括CUDA库2803,CUDA运行时2805,CUDA驱动器2807和设备内核驱动器2808。在至少一个实施例中,CUDA软件栈2800在硬件2809上执行,该硬件2809可以包括支持CUDA的GPU,其由加利福尼亚州圣克拉拉市的NVIDIA公司开发。

[0290] 在至少一个实施例中,应用程序2801、CUDA运行时2805和设备内核驱动器2808可以分别执行与应用程序2701、运行时2705和设备内核驱动器2706类似的功能,以上结合图27对其进行了描述。在至少一个实施例中,CUDA驱动器2807包括实现CUDA驱动器API 2806的库(libcuda.so)。在至少一个实施例中,类似于由CUDA运行时库(cudart)实现的CUDA运行时API 2804,CUDA驱动器API 2806可以公开但不限于用于存储器管理、执行控制、设备管理、错误处理、同步和/或图形互操作性等的功能。在至少一个实施例中,CUDA驱动器API 2806与CUDA运行时API 2804的不同之处在于,CUDA运行时API 2804通过提供隐式初始化、上下文(类似于进程)管理和模块(类似于动态加载的库)管理来简化设备代码管理。与高级CUDA运行时API 2804相反,在至少一个实施例中,CUDA驱动器API 2806是提供对设备的更细粒度控制的低级API,特别是关于上下文和模块加载。在至少一个实施例中,CUDA驱动器

API 2806可以公开没有由CUDA运行时API 2804公开的用于上下文管理的功能。在至少一个实施例中,CUDA驱动器API 2806也与语言无关,并且除了支持CUDA运行时API 2804之外,还支持例如OpenCL。此外,在至少一个实施例中,包括CUDA运行时2805在内的开发库可被视为与驱动器组件分离,包括用户模式的CUDA驱动器2807和内核模式的设备驱动器2808(有时也称为“显示”驱动器)。

[0291] 在至少一个实施例中,CUDA库2803可以包括但不限于数学库,深度学习库,并行算法库和/或信号/图像/视频处理库,并行计算应用程序(例如应用程序2801)可以利用这些库。在至少一个实施例中,CUDA库2803可包括数学库,例如cuBLAS库,其是用于执行线性代数运算的基本线性代数子程序(“BLAS”)的实现;用于计算快速傅立叶变换(“FFT”)的cuFFT库,以及用于生成随机数的cuRAND库等。在至少一个实施例中,CUDA库2803可以包括深度学习库,诸如用于深度神经网络的基元的cuDNN库和用于高性能深度学习推理的TensorRT平台等等。

[0292] 图29示出了根据至少一个实施例的图27的软件栈2700的ROCm实现。在至少一个实施例中,可在其上启动应用程序2901的ROCm软件栈2900包括语言运行时2903,系统运行时2905,thunk 2907,ROCm内核驱动器2908和设备内核驱动器2909。在至少一个实施例中,ROCm软件栈2900在硬件2910上执行,硬件2909可以包括支持ROCm的GPU,其由加利福尼亚州圣克拉拉市的AMD公司开发。

[0293] 在至少一个实施例中,应用程序2901可以执行与以上结合图27讨论的应用程序2701类似的功能。另外,在至少一个实施例中,语言运行时2903和系统运行时2905可以执行与以上结合图27讨论的运行2705类似的功能。在至少一个实施例中,语言运行时2903和系统运行时2905的不同之处在于,系统运行时2905是实现ROCr系统运行时API 2904并利用异构系统架构(“HSA”)运行时API的语言无关运行时。在至少一个实施例中,HSA运行时API是一种瘦用户模式API,它公开接口以供访问和与AMDGPU交互,包括用于存储器管理,通过架构分派内核的执行控制,错误处理,系统和代理信息以及运行时初始化和关闭等的功能。在至少一个实施例中,与系统运行时2905相比,语言运行时2903是ROCr系统运行时API 2904之上分层的特定于语言的运行时API 2902的实现。在至少一个实施例中,语言运行时API可以包括但不限于可移植异构计算接口(“HIP”)语言运行时API,异构计算编译器(“HCC”)语言运行时API或OpenCL API等等。特别是,HIP语言是C++编程语言的扩展,具有CUDA机制的功能相似版本,并且在至少一个实施例中,HIP语言运行时API包括与以上结合图28讨论的CUDA运行时API 2804相似的功能,例如用于存储器管理,执行控制,设备管理,错误处理和同步等的功能。

[0294] 在至少一个实施例中,thunk (ROCr) 2907是可用于与底层ROCm驱动器2908交互的接口。在至少一个实施例中,ROCm驱动器2908是ROCr驱动器,其是AMDGPU驱动器和HSA内核驱动器(amdkfd)的组合。在至少一个实施例中,AMDGPU驱动器是由AMD开发的用于GPU的设备内核驱动器,其执行与以上结合图27讨论的设备内核驱动器2706类似的功能。在至少一个实施例中,HSA内核驱动器是允许不同类型的处理器经由硬件特征更有效地共享系统资源的驱动器。

[0295] 在至少一个实施例中,各种库(未示出)可以被包括在语言运行时2903上方的ROCm软件栈2900中,并且提供与以上结合图28讨论的CUDA库2803相似的功能。在至少一个实施

例中,各种库可以包括但不限于数学、深度学习和/或其他库,例如实现与CUDA cuBLAS类似的功能的hipBLAS库,类似于CUDA cuFFT用于计算FFT的rocFFT库等。

[0296] 图30示出了根据至少一个实施例的图27的软件栈2700的OpenCL实现。在至少一个实施例中,可以在其上启动应用程序3001的OpenCL软件栈3000包括OpenCL框架3005,OpenCL运行时3006和驱动器3007。在至少一个实施例中,OpenCL软件栈3000在不是特定于供应商的硬件2809上执行。在至少一个实施例中,由于由不同厂商开发的设备支持OpenCL,因此可能需要特定的OpenCL驱动器才能与来自此类厂商的硬件进行互操作。

[0297] 在至少一个实施例中,应用程序3001,OpenCL运行时3006,设备内核驱动器3007和硬件3008可以分别执行与上面结合图27讨论的应用程序2701、运行时2705、设备内核驱动器2706和硬件2707类似的功能。在至少一个实施例中,应用程序3001还包括具有将在设备上执行的代码的OpenCL内核3002。

[0298] 在至少一个实施例中,OpenCL定义了一种“平台”,其允许主机控制连接到该主机的设备。在至少一个实施例中,OpenCL框架提供平台层API和运行时API,示出为平台API 3003和运行时API3005。在至少一个实施例中,运行时API 3005使用上下文来管理设备上内核的执行。在至少一个实施例中,每个标识的设备可以与各自的上下文相关联,运行时API3005可以使用该上下文来管理该设备的命令队列、程序对象和内核对象、共享存储器对象等。在至少一个实施例中,平台API 3003公开了允许设备上下文用于选择和初始化设备,经由命令队列将工作提交给设备,以及使得能够进行来自和去往设备的数据传输等的功能。另外,在至少一个实施例中,OpenCL框架提供各种内置功能(未示出),包括数学功能、关系功能和图像处理功能等。

[0299] 在至少一个实施例中,编译器3004也被包括在OpenCL框架3005中。在至少一个实施例中,源代码可以在执行应用程序之前被离线编译或者在执行应用程序期间被在线编译。与CUDA和ROCm相反,至少一个实施例中的OpenCL应用程序可以由编译器3004在线编译,编译器3004被包括以代表可以用于将源代码和/或IR代码(例如标准可移植中间表示(“SPIR-V”)代码)编译为二进制代码的任意数量的编译器。可替代地,在至少一个实施例中,可以在执行这样的应用程序之前离线编译OpenCL应用程序。

[0300] 图31示出了根据至少一个实施例的由编程平台支持的软件。在至少一个实施例中,编程平台3104被配置为支持应用程序3100可以依赖的各种编程模型3103,中间件和/或库3102以及框架3101。在至少一个实施例中,应用程序3100可以是使用例如深度学习框架(例如,MXNet,PyTorch或TensorFlow)实现的AI/ML应用,其可以依赖于诸如cuDNN,NVIDIACollective Communications Library(“NCCL”)和/或NVIDIA开发人员数据加载库(“DALI”)CUDA库之类的库,以在底层硬件上提供加速的计算。

[0301] 在至少一个实施例中,编程平台3104可以是以上分别结合图28、图29和图30描述的CUDA、ROCm或OpenCL平台之一。在至少一个实施例中,编程平台3104支持多个编程模型3103,其是底层计算系统的抽象,其允许算法和数据结构的表达。在至少一个实施例中,编程模型3103可以暴露底层硬件的特征以便改善性能。在至少一个实施例中,编程模型3103可以包括但不限于CUDA,HIP,OpenCL,C++加速大规模并行性(“C++AMP”),开放多处理(“OpenMP”),开放加速器(“OpenACC”)和/或Vulcan计算(Vulcan Compute)。

[0302] 在至少一个实施例中,库和/或中间件3102提供编程模型3104的抽象的实现。在至

少一个实施例中,这样的库包括可由计算机程序使用并在软件开发期间利用的数据和编程代码。在至少一个实施例中,除了可以从编程平台3104获得的那些之外,这样的中间件还包括向应用程序提供服务的软件。在至少一个实施例中,库和/或中间件3102可以包括但不限于cuBLAS、cuFFT、cuRAND和其他CUDA库,或rocBLAS、rocFFT、rocRAND和其他ROCm库。另外,在至少一个实施例中,库和/或中间件3102可以包括NCCL和ROCm通信集合库(“RCCL”)库,其提供用于GPU的通信例程,用于深度学习加速的MIOpen库和/或用于线性代数、矩阵和向量运算、几何变换、数值求解器以及相关算法的本征库。

[0303] 在至少一个实施例中,应用程序框架3101依赖于库和/或中间件3102。在至少一个实施例中,每个应用程序框架3101是用于实现应用软件的标准结构的软件框架。回到上面讨论的AI/ML示例,在至少一个实施例中,可以使用框架(诸如Caffe,Caffe2,TensorFlow,Keras,PyTorch或MxNet深度学习框架)来实现AI/ML应用。

[0304] 图32示出了根据至少一个实施例的编译代码以在图27-30的编程平台之一上执行。在至少一个实施例中,编译器3201接收源代码3200,其包括主机代码以及设备代码两者。在至少一个实施例中,编译器3201被配置为将源代码3200转换为用于在主机上执行的主机可执行代码3202以及用于在设备上执行的设备可执行代码3203。在至少一个实施例中,源代码3200可以在执行应用程序之前离线编译,或者在执行应用程序期间在线编译。

[0305] 在至少一个实施例中,源代码3200可以包括编译器3201支持的任何编程语言的代码,例如C++、C、Fortran等。在至少一个实施例中,源代码3200可以包括在单源(single-source)文件中,其具有主机代码和设备代码的混合,并在其中指示了设备代码的位置。在至少一个实施例中,单源文件可以是包括CUDA代码的.cu文件或包括HIP代码的.hip.cpp文件。备选地,在至少一个实施例中,源代码3200可以包括多个源代码文件,而不是单源文件,在该单源文件中主机代码和设备代码是分开的。

[0306] 在至少一个实施例中,编译器3201被配置为将源代码3200编译成用于在主机上执行的主机可执行代码3202和用于在设备上执行的设备可执行代码3203。在至少一个实施例中,编译器3201执行操作,包括将源代码3200解析为抽象系统树(AST),执行优化以及生成可执行代码。在源代码3200包括单源文件的至少一个实施例中,编译器3201可以将设备代码与主机代码在这种单源文件中分开,将设备代码和主机代码分别编译成设备可执行代码3203和主机可执行代码3202,以及将设备可执行代码3203和主机可执行代码3202在单个文件中链接到一起,如下面关于图33更详细讨论的。

[0307] 在至少一个实施例中,主机可执行代码3202和设备可执行代码3203可以是任何合适的格式,例如二进制代码和/或IR代码。在CUDA的情况下,在至少一个实施例中,主机可执行代码3202可以包括本地对象代码,而设备可执行代码3203可以包括PTX中间表示的代码。在至少一个实施例中,在ROCm的情况下,主机可执行代码3202和设备可执行代码3203都可以包括目标二进制代码。

[0308] 图33是根据至少一个实施例的编译代码以在图27-30的编程平台之一上执行的更详细图示。在至少一个实施例中,编译器3301被配置为接收源代码3300,编译源代码3300,并输出可执行文件3308。在至少一个实施例中,源代码3300是单源文件,例如.cu文件,.hip.cpp文件或其他格式的文件,其包括主机代码和设备代码两者。在至少一个实施例中,编译器3301可以是但不限于用于在.cu文件中编译CUDA代码的NVIDIA CUDA编译器

(“NVCC”),或用于在.hip.cpp文件中编译HIP代码的HCC编译器。

[0309] 在至少一个实施例中,编译器3301包括编译器前端3302,主机编译器3305,设备编译器3306和链接器3309。在至少一个实施例中,编译器前端3302被配置为在源代码3300中将设备代码3304与主机代码3303分开。在至少一个实施例中,设备代码3304由设备编译器3306编译成设备可执行代码3308,如所描述的,其可以包括二进制代码或IR代码。在至少一个实施例中,主机代码3303由主机编译器3305单独地编译成主机可执行代码3307。在至少一个实施例中,对于NVCC,主机编译器3305可以是但不限于输出本机目标代码的通用C/C++编译器,而设备编译器3306可以是但不限于基于低级虚拟机(“LLVM”)的编译器,其将LLVM编译器基础架构分叉,并输出PTX代码或二进制代码。在至少一个实施例中,对于HCC,主机编译器3305和设备编译器3306两者可以是但不限于输出目标二进制代码的基于LLVM的编译器。

[0310] 在至少一个实施例中,在将源代码3300编译成主机可执行代码3307和设备可执行代码3308之后,链接器3309将主机和设备可执行代码3307和3308在可执行文件3310中链接到一起。在至少一个实施例中,主机和PTX的本机目标代码或设备的二进制代码可以在可执行和可链接格式(“ELF”)文件中链接在一起,该文件是用于存储目标代码的容器格式。

[0311] 图34示出了根据至少一个实施例的在编译源代码之前转换源代码。在至少一个实施例中,源代码3400通过转换工具3401传递,转换工具3401将源代码3400转换成转换后的源代码3402。在至少一个实施例中,编译器3403用于将转换后的源代码3402编译成主机可执行代码3404和设备可执行代码3405,其过程类似于由编译器3201将源代码3200编译成主机可执行代码3202和设备可执行代码3203的过程,如以上结合图32所讨论的。

[0312] 在至少一个实施例中,由转换工具3401执行的转换被用于移植(port)源代码3400,以在与最初打算在其上运行的不同的环境中执行。在至少一个实施例中,转换工具3401可以包括但不限于HIP转换器,其用于将用于CUDA平台的CUDA代码“移植(hipify)”为可以在ROCm平台上编译和执行的HIP代码。在至少一个实施例中,源代码3400的转换可以包括:解析源代码3400,并将对由一个编程模型(例如,CUDA)提供的API的调用转换为对由另一编程模型(例如,例如,HIP)提供的API的相应调用,如下面结合图35A-36更详细地讨论的。返回到移植CUDA代码的示例,在至少一个实施例中,对CUDA运行时API、CUDA驱动器API和/或CUDA库的调用可以被转换为对应的HIP API调用。在至少一个实施例中,由转换工具3401执行的自动转换有时可能是不完整的,需要额外的人工来完全移植源代码3400。

[0313] 配置GPU用于通用计算

[0314] 以下各图阐述但不限于根据至少一个实施例的用于编译和执行计算源代码的示例性架构。在至少一个实施例中,以下附图的一个或更多个基于计算机的系统可以实现参照图1A描述的实施例,参照图1B描述的实施例,参照图2描述的实施例,参照图3描述的技术300,参照图4描述的技术400,参照图5描述的技术500,参照图6描述的技术600和/或参照图7描述的技术700的一个或更多个方面。

[0315] 图35A示出了根据至少一个实施例的被配置为使用不同类型的处理单元来编译和执行CUDA源代码3510的系统3500。在至少一个实施例中,系统3500包括但不限于CUDA源代码3510,CUDA编译器3550,主机可执行代码3570(1),主机可执行代码3570(2),CUDA设备可执行代码3584,CPU 3590,启用CUDA的GPU 3594,GPU 3592,CUDA到HIP转换工具3520,HIP源

代码3530,HIP编译器驱动器3540,HCC 3560和HCC设备可执行代码3582。

[0316] 在至少一个实施例中,CUDA源代码3510是CUDA编程语言的人类可读代码的集合。在至少一个实施例中,CUDA代码是CUDA编程语言的人类可读代码。在至少一个实施例中,CUDA编程语言是C++编程语言的扩展,其包括但不限于定义设备代码以及区分设备代码和主机代码的机制。在至少一个实施例中,设备代码是在编译之后可在设备上并行执行的源代码。在至少一个实施例中,设备可以是针对并行指令处理而优化的处理器,例如启用CUDA的GPU 3590、GPU 3592或另一GPGPU等。在至少一个实施例中,主机代码是在编译后可以在主机上执行的源代码。在至少一个实施例中,主机是针对顺序指令处理而优化的处理器,例如CPU 3590。

[0317] 在至少一个实施例中,CUDA源代码3510包括但不限于,任意数量(包括零)的全局函数3512,任意数量(包括零)的设备函数3514,任意数量(包括零)的主机函数3516,以及任意数量(包括零)的主机/设备函数3518。在至少一个实施例中,全局函数3512,设备函数3514,主机函数3516和主机/设备函数3518在CUDA源代码3510中可以混合。在至少一个实施例中,每个全局函数3512可在设备上执行并且可从主机调用。因此,在至少一个实施例中,全局函数3512中的一个或更多个可以充当设备的入口点。在至少一个实施例中,每个全局函数3512是内核。在至少一个实施例中以及在一种称为动态并行性的技术中,一个或更多个全局函数3512定义了一内核,该内核可以在设备上执行并且可以从这样的设备调用。在至少一个实施例中,内核在执行期间由设备上的N个不同线程并行执行N次(其中N为任何正整数)。

[0318] 在至少一个实施例中,每个设备函数3514在设备上执行并且只能从这样的设备调用。在至少一个实施例中,每个主机函数3516在主机上执行并且只能从这样的主机调用。在至少一个实施例中,每个主机/设备函数3516既定义了主机上可执行并且只能从这样的主机调用的函数的主机版本,也定义了设备上可执行并且只能从这样的设备调用的函数的设备版本。

[0319] 在至少一个实施例中,CUDA源代码3510还可包括但不限于对通过CUDA运行时API 3502定义的任意数量的函数的任意数量的调用。在至少一个实施例中,CUDA运行时API 3502可以包括但不限于在主机上执行的任意数量的函数,用于分配和解除分配设备存储器,在主机存储器和设备存储器之间传输数据,管理具有多个设备的系统等。在至少一个实施例中,CUDA源代码3510还可以包括对在任意数量的其他CUDA API中指定的任意数量的函数的任意数量的调用。在至少一个实施例中,CUDA API可以是设计为由CUDA代码使用的任何API。在至少一个实施例中,CUDA API包括但不限于CUDA运行时API 3502,CUDA驱动器API,用于任意数量的CUDA库的API等。在至少一个实施例中并且相对于CUDA运行时API 3502,CUDA驱动器API是较低级别的API,但可以提供对设备的更细粒度的控制。在至少一个实施例中,CUDA库的示例包括但不限于cuBLAS,cuFFT,cuRAND,cuDNN等。

[0320] 在至少一个实施例中,CUDA编译器3550编译输入的CUDA代码(例如,CUDA源代码3510)以生成主机可执行代码3570(1)和CUDA设备可执行代码3584。在至少一个实施例中,CUDA编译器3550是NVCC。在至少一个实施例中,主机可执行代码3570(1)是在CPU 3590上可执行的输入源代码中包括的主机代码的编译版本。在至少一个实施例中,CPU3590可以是针对顺序指令处理而优化的任何处理器。

[0321] 在至少一个实施例中,CUDA设备可执行代码3584是在启用CUDA的GPU 3594上可执行的输入源代码中包括的设备代码的编译版本。在至少一个实施例中,CUDA设备可执行代码3584包括但不限于二进制代码。在至少一个实施例中,CUDA设备可执行代码3584包括但不限于IR代码,例如PTX代码,该IR代码在运行时被设备驱动器进一步编译为用于特定目标设备(例如,启用CUDA的GPU 3594)的二进制代码。在至少一个实施例中,启用CUDA的GPU 3594可以是针对并行指令处理而优化并且支持CUDA的任何处理器。在至少一个实施例中,启用CUDA的GPU 3594由加利福尼亚州圣克拉拉市的NVIDIA公司开发。

[0322] 在至少一个实施例中,CUDA到HIP转换工具3520被配置为将CUDA源代码3510转换成功能上相似的HIP源代码3530。在至少一个实施例中,HIP源代码3530是HIP编程语言的人类可读代码的集合。在至少一个实施例中,HIP代码是HIP编程语言的人类可读代码。在至少一个实施例中,HIP编程语言是C++编程语言的扩展,其包括但不限于CUDA机制的功能上相似版本,用于定义设备代码并区分设备代码和主机代码。在至少一个实施例中,HIP编程语言可以包括CUDA编程语言的功能的子集。在至少一个实施例中,例如,HIP编程语言包括但不限于定义全局函数3512的机制,但是这样的HIP编程语言可能缺乏对动态并行性的支持,因此,在HIP代码中定义的全局函数3512仅可从主机调用。

[0323] 在至少一个实施例中,HIP源代码3530包括但不限于任意数量(包括零)的全局函数3512,任意数量(包括零)的设备函数3514,任意数量(包括零)的主机函数3516以及任意数量(包括零)的主机/设备函数3518。在至少一个实施例中,HIP源代码3530还可以包括对在HIP运行时API3532中指定的任意数量的函数的任意数量的调用。在一个实施例中,HIP运行时API 3532包括但不限于CUDA运行时API 3502中包括的函数的子集的功能上相似版本。在至少一个实施例中,HIP源代码3530还可以包括对在任意数量的其他HIP API中指定的任意数量的函数的任意数量的调用。在至少一个实施例中,HIP API可以是被设计为供HIP代码和/或ROCm使用的任何API。在至少一个实施例中,HIP API包括但不限于HIP运行时API 3532,HIP驱动器API,用于任意数量的HIP库的API,用于任意数量的ROCm库的API等。

[0324] 在至少一个实施例中,CUDA到HIP转换工具3520将CUDA代码中的每个内核调用从CUDA语法转换为HIP语法,并将CUDA代码中的任意数量的其他CUDA调用转换为任意数量的其他功能上相似的HIP调用。在至少一个实施例中,CUDA调用是对在CUDA API中指定的函数的调用,并且HIP调用是对在HIP API中指定的函数的调用。在至少一个实施例中,CUDA到HIP转换工具3520将对在CUDA运行时API 3502中指定的函数的任意数量的调用转换为对在HIP运行时API 3532中指定的函数的任意数量的调用。

[0325] 在至少一个实施例中,CUDA到HIP转换工具3520是被称为hipify-perl的工具,其执行基于文本的转换过程。在至少一个实施例中,CUDA到HIP转换工具3520是被称为hipify-clang的工具,相对于hipify-perl,其执行更复杂且更鲁棒的转换过程,该过程涉及使用clang(编译器前端)解析CUDA代码,然后转换得到的符号。在至少一个实施例中,除了由CUDA到HIP转换工具3520执行的那些修改之外,将CUDA代码正确地转换成HIP代码可能还需要修改(例如,手动编辑)。

[0326] 在至少一个实施例中,HIP编译器驱动器3540是确定目标设备3546,然后配置与目标设备3546兼容的编译器以编译HIP源代码3530的前端。在至少一个实施例中,目标设备3546是针对并行指令处理而优化的处理器。在至少一个实施例中,HIP编译器驱动器3540可

以以任何技术上可行的方式确定目标设备3546。

[0327] 在至少一个实施例中,如果目标设备3546与CUDA兼容(例如,启用CUDA的GPU 3594),则HIP编译器驱动器3540生成HIP/NVCC编译命令3542。在至少一个实施例中并且结合图35B更详细地描述的,HIP/NVCC编译命令3542配置CUDA编译器3550以使用但不限于HIP到CUDA转换头和CUDA运行时库来编译HIP源代码3530。在至少一个实施例中并且响应于HIP/NVCC编译命令3542,CUDA编译器3550生成主机可执行代码3570(1)和CUDA设备可执行代码3584。

[0328] 在至少一个实施例中,如果目标设备3546与CUDA不兼容,则HIP编译器驱动器3540生成HIP/HCC编译命令3544。在至少一个实施例中并且如结合图35C更详细地描述的,HIP/HCC编译命令3544配置HCC 3560以使用HCC头和HIP/HCC运行时库编译HIP源代码3530。在至少一个实施例中并且响应于HIP/HCC编译命令3544,HCC 3560生成主机可执行代码3570(2)和HCC设备可执行代码3582。在至少一个实施例中,HCC设备可执行代码3582是HIP源代码3530中包含的可在GPU 3592上执行的设备代码的编译版本。在至少一个实施例中,GPU 3592可以是针对并行指令处理而优化的、与CUDA不兼容且与HCC兼容的任何处理器。在至少一个实施例中,GPU 3592由加利福尼亚州圣克拉拉市的AMD公司开发。在至少一个实施例中,GPU 3592是不启用CUDA的GPU 3592。

[0329] 仅出于说明性目的,在图35A中描绘了在至少一个实施例中可以实现为编译CUDA源代码3510以在CPU 3590和不同设备上执行的三个不同流程。在至少一个实施例中,直接CUDA流程编译CUDA源代码3510以在CPU 3590和启用CUDA的GPU 3594上执行,而无需将CUDA源代码3510转换为HIP源代码3530。在至少一个实施例中,间接CUDA流程将CUDA源代码3510转换为HIP源代码3530,然后编译HIP源代码3530以在CPU 3590和启用CUDA的GPU 3594上执行。在至少一个实施例中,CUDA/HCC流程将CUDA源代码3510转换为HIP源代码3530,然后编译HIP源代码3530以在CPU 3590和GPU 3592上执行。

[0330] 可以通过虚线和一系列气泡注释A1-A3描绘可以在至少一个实施例中实现的直接CUDA流程。在至少一个实施例中,并且如气泡注释A1所示,CUDA编译器3550接收CUDA源代码3510和配置CUDA编译器3550以编译CUDA源代码3510的CUDA编译命令3548。在至少一个实施例中,直接CUDA流程中使用的CUDA源代码3510是用CUDA编程语言编写的,该CUDA编程语言基于除C++之外的其他编程语言(例如C, Fortran, Python, Java等)。在至少一个实施例中,并且响应于CUDA编译命令3548,CUDA编译器3550生成主机可执行代码3570(1)和CUDA设备可执行代码3584(用气泡注释A2表示)。在至少一个实施例中并且如用气泡注释A3所示,主机可执行代码3570(1)和CUDA设备可执行代码3584可以分别在CPU 3590和启用CUDA的GPU 3594上执行。在至少一个实施例中,CUDA设备可执行代码3584包括但不限于二进制代码。在至少一个实施例中,CUDA设备可执行代码3584包括但不限于PTX代码,并且在运行时被进一步编译成用于特定目标设备的二进制代码。

[0331] 可以通过虚线和一系列气泡注释B1-B6来描述可以在至少一个实施例中实现的间接CUDA流程。在至少一个实施例中并且如气泡注释B1所示,CUDA到HIP转换工具3520接收CUDA源代码3510。在至少一个实施例中并且如气泡注释B2所示,CUDA到HIP转换工具3520将CUDA源代码3510转换为HIP源代码3530。在至少一个实施例中并且如气泡注释B3所示,HIP编译器驱动器3540接收HIP源代码3530,并确定目标设备3546是否启用了CUDA。

[0332] 在至少一个实施例中并且如气泡注释B4所示, HIP编译器驱动器3540生成HIP/NVCC编译命令3542, 并将HIP/NVCC编译命令3542和HIP源代码3530两者都发送到CUDA编译器3550。在至少一个实施例中并且如结合图35B更详细地描述的, HIP/NVCC编译命令3542配置CUDA编译器3550以使用但不限于HIP到CUDA转换头和CUDA运行时库来编译HIP源代码3530。在至少一个实施例中并且响应于HIP/NVCC编译命令3542, CUDA编译器3550生成主机可执行代码3570 (1) 和CUDA设备可执行代码3584 (用气泡注释B5表示)。在至少一个实施例中并且如气泡注释B6所示, 主机可执行代码3570 (1) 和CUDA设备可执行代码3584可以分别在CPU 3590和启用CUDA的GPU 3594上执行。在至少一个实施例中, CUDA设备可执行代码3584包括但不限于二进制代码。在至少一个实施例中, CUDA设备可执行代码3584包括但不限于PTX代码, 并且在运行时被进一步编译成用于特定目标设备的二进制代码。

[0333] 可以通过实线和一系列气泡注释C1-C6来描述可以在至少一个实施例中实现的CUDA/HCC流程。在至少一个实施例中并且如气泡注释C1所示, CUDA到HIP转换工具3520接收CUDA源代码3510。在至少一个实施例中并且如气泡注释C2所示, CUDA到HIP转换工具3520将CUDA源代码3510转换为HIP源代码3530。在至少一个实施例中并且如气泡注释C3所示, HIP编译器驱动器3540接收HIP源代码3530, 并确定目标设备3546未启用CUDA。

[0334] 在至少一个实施例中, HIP编译器驱动器3540生成HIP/HCC编译命令3544, 并且将HIP/HCC编译命令3544和HIP源代码3530两者发送到HCC 3560 (用气泡注释C4表示)。在至少一个实施例中并且如结合图35C更详细地描述的, HIP/HCC编译命令3544配置HCC 3560以使用但不限于HCC头和HIP/HCC运行时库编译HIP源代码3530。在至少一个实施例中并且响应于HIP/HCC编译命令3544, HCC 3560生成主机可执行代码3570 (2) 和HCC设备可执行代码3582 (用气泡注释C5表示)。在至少一个实施例中并且如气泡注释C6所示, 主机可执行代码3570 (2) 和HCC设备可执行代码3582可以分别在CPU 3590和GPU 3592上执行。

[0335] 在至少一个实施例中, 在将CUDA源代码3510转换为HIP源代码3530之后, HIP编译器驱动器3540可随后用于生成用于启用CUDA的GPU3594或GPU 3592的可执行代码, 而无需将CUDA重新执行为HIP转换工具3520。在至少一个实施例中, CUDA到HIP转换工具3520将CUDA源代码3510转换为HIP源代码3530, 然后将其存储在存储器中。在至少一个实施例中, HIP编译器驱动器3540然后配置HCC 3560以基于HIP源代码3530生成主机可执行代码3570 (2) 和HCC设备可执行代码3582。在至少一个实施例中, HIP编译器驱动器3540随后配置CUDA编译器3550以基于存储的HIP源代码3530生成主机可执行代码3570 (1) 和CUDA设备可执行代码3584。

[0336] 图35B示出了根据至少一个实施例的被配置为使用CPU 3590和启用CUDA的GPU 3594来编译和执行图35A的CUDA源代码3510的系统3504。在至少一个实施例中, 系统3504包括但不限于CUDA源代码3510, CUDA到HIP转换工具3520, HIP源代码3530, HIP编译器驱动器3540, CUDA编译器3550, 主机可执行代码3570 (1), CUDA设备可执行代码3584, CPU 3590和启用CUDA的GPU 3594。

[0337] 在至少一个实施例中并且如本文先前结合图35A所描述的, CUDA源代码3510包括但不限于任意数量 (包括零) 的全局函数3512, 任意数量 (包括零) 的设备函数3514, 任意数量 (包括零) 的主机函数3516以及任意数量 (包括零) 的主机/设备函数3518。在至少一个实施例中, CUDA源代码3510还包括但不限于对在任意数量的CUDA API中指定的任意数量的函

数的任意数量的调用。

[0338] 在至少一个实施例中,CUDA到HIP转换工具3520将CUDA源代码3510转换成HIP源代码3530。在至少一个实施例中,CUDA到HIP转换工具3520将CUDA源代码3510中的每个内核调用从CUDA语法转换为HIP语法,并将CUDA源代码3510中任意数量的其他CUDA调用转换为任意数量的其他功能上相似的HIP调用。

[0339] 在至少一个实施例中,HIP编译器驱动器3540确定目标设备3546是启用CUDA的,并且生成HIP/NVCC编译命令3542。在至少一个实施例中,然后HIP编译器驱动器3540经由HIP/NVCC编译命令3542配置CUDA编译器3550以编译HIP源代码3530。在至少一个实施例中,作为配置CUDA编译器3550的一部分,HIP编译器驱动器3540提供对HIP到CUDA转换头3552的访问。在至少一个实施例中,HIP到CUDA转换头3552将任意数量的HIP API中指定的任意数量的机制(例如,函数)转换为任意数量的CUDA API中指定的任意数量的机制。在至少一个实施例中,CUDA编译器3550将HIP到CUDA转换头3552与对应于CUDA运行时API 3502的CUDA运行时库3554结合使用,以生成主机可执行代码3570(1)和CUDA设备可执行代码3584。在至少一个实施例中,然后可以分别在CPU3590和启用CUDA的GPU 3594上执行主机可执行代码3570(1)和CUDA设备可执行代码3584。在至少一个实施例中,CUDA设备可执行代码3584包括但不限于二进制代码。在至少一个实施例中,CUDA设备可执行代码3584包括但不限于PTX代码,并且在运行时被进一步编译成用于特定目标设备的二进制代码。

[0340] 图35C示出了根据至少一个实施例的系统3506,该系统3506被配置为使用CPU 3590和未启用CUDA的GPU 3592来编译和执行图35A的CUDA源代码3510。在至少一个实施例中,系统3506包括但不限于CUDA源代码3510,CUDA到HIP转换工具3520,HIP源代码3530,HIP编译器驱动器3540,HCC 3560,主机可执行代码3570(2),HCC设备可执行代码3582,CPU 3590和GPU 3592。

[0341] 在至少一个实施例中,并且如本文先前结合图35A所描述的,CUDA源代码3510包括但不限于任意数量(包括零)的全局函数3512,任意数量(包括零)的设备函数3514,任意数量(包括零)的主机函数3516以及任意数量(包括零)的主机/设备函数3518。在至少一个实施例中,CUDA源代码3510还包括但不限于对在任意数量的CUDA API中指定的任意数量的函数的任意数量的调用。

[0342] 在至少一个实施例中,CUDA到HIP转换工具3520将CUDA源代码3510转换成HIP源代码3530。在至少一个实施例中,CUDA到HIP转换工具3520将CUDA源代码3510中的每个内核调用从CUDA语法转换为HIP语法,并将源代码3510中任意数量的其他CUDA调用转换为任意数量的其他功能上相似的HIP调用。

[0343] 在至少一个实施例中,HIP编译器驱动器3540随后确定目标设备3546不是启用CUDA的,并生成HIP/HCC编译命令3544。在至少一个实施例中,然后HIP编译器驱动器3540配置HCC 3560以执行HIP/HCC编译命令3544,从而编译HIP源代码3530。在至少一个实施例中,HIP/HCC编译命令3544将HCC 3560配置为使用但不限于HIP/HCC运行时库3558和HCC头3556来生成主机可执行代码3570(2)和HCC设备可执行代码3582。在至少一个实施例中,HIP/HCC运行时库3558对应于HIP运行时API3532。在至少一个实施例中,HCC头3556包括但不限于用于HIP和HCC的任意数量和类型的互操作性机制。在至少一个实施例中,主机可执行代码3570(2)和HCC设备可执行代码3582可以分别在CPU 3590和GPU 3592上执行。

[0344] 图36示出了根据至少一个实施例的由图35C的CUDA到HIP转换工具3520转换的示例性内核。在至少一个实施例中,CUDA源代码3510将给定内核被设计为解决的总体问题划分为可以使用线程块独立解决的相对粗糙的子问题。在至少一个实施例中,每个线程块包括但不限于任意数量的线程。在至少一个实施例中,每个子问题被划分为相对细小的部分(pieces),这些部分可以由线程块中的线程协作并行地解决。在至少一个实施例中,线程块内的线程可以通过共享存储器共享数据并通过同步执行以协调存储器访问来协作。

[0345] 在至少一个实施例中,CUDA源代码3510将与给定内核相关联的线程块组织成线程块的一维、二维或三维网格。在至少一个实施例中,每个线程块包括但不限于任意数量的线程,并且网格包括但不限于任意数量的线程块。

[0346] 在至少一个实施例中,内核是使用“\_\_global\_\_”声明说明符(specifier)定义的设备代码中的函数。在至少一个实施例中,使用CUDA内核启动语法3610来指定针对给定内核调用执行内核的网格的尺寸以及相关流。在至少一个实施例中,CUDA内核启动语法3610被指定为

[0347] “KernelName<<<GridSize,BlockSize,SharedMemorySize,Stream>>>

[0348] (KernelArguments);”。在至少一个实施例中,执行配置语法是“<<<...>>>”构造,其被插入在内核名称(“KernelName”)和内核参数的括号列表(“KernelArguments”)之间。在至少一个实施例中,CUDA内核启动语法3610包括但不限于CUDA启动函数语法而不是执行配置语法。

[0349] 在至少一个实施例中,“GridSize”是dim3类型的,并且指定网格的尺寸和大小。在至少一个实施例中,类型dim3是CUDA定义的结构,其包括但不限于无符号整数x,y和z。在至少一个实施例中,如果未指定z,则z默认为1。在至少一个实施例中,如果未指定y,则y默认为1。在至少一个实施例中,网格中的线程块的数量等于GridSize.x、GridSize.y和GridSize.z的乘积。在至少一个实施例中,“BlockSize”是dim3类型的,并且指定每个线程块的尺寸和大小。在至少一个实施例中,每线程块的线程数等于BlockSize.x、BlockSize.y和BlockSize.z的乘积。在至少一个实施例中,给定执行内核的每个线程唯一的线程ID,该线程ID可通过内置变量(例如“threadIdx”)在内核内访问。

[0350] 在至少一个实施例中,关于CUDA内核启动语法3510,“SharedMemorySize”是一可选参数,它指定共享存储器中除静态分配的存储器外,针对给定内核调用为每个线程块动态分配的字节数。在至少一个实施例中并且关于CUDA内核启动语法3610,SharedMemorySize默认为零。在至少一个实施例中并且关于CUDA内核启动语法3610,“流”是可选的参数,其指定相关联的流并且默认为零以指定默认流。在至少一个实施例中,流是按顺序执行的命令序列(其可能由不同的主机线程发出)。在至少一个实施例中,不同的流可以相对于彼此无序地或同时地执行命令。

[0351] 在至少一个实施例中,CUDA源代码3510包括但不限于用于示例性内核“MatAdd”的内核定义和主函数。在至少一个实施例中,主函数是在主机上执行的主机代码,并且包括但不限于使内核MatAdd在设备上执行的内核调用。在至少一个实施例中,如图所示,内核MatAdd将大小为NxN的两个矩阵A和B相加,其中N为正整数,并将结果存储在矩阵C中。在至少一个实施例中,主函数将threadsPerBlock变量定义为16x 16,numBlocks变量为N/16x N/16。在至少一个实施例中,然后主函数指定内核调用“MatAdd<<<numBlocks,

threadsPerBlock>>>(A,B,C);”。在至少一个实施例中,并且根据CUDA内核启动语法3610,使用尺寸为 $N/16 \times N/16$ 的线程块网格来执行内核MatAdd,其中每个线程块的尺寸为 $16 \times 16$ 。在至少一个实施例中,每个线程块包括256个线程,创建具有足够块的网格以使每个矩阵元素具有一个线程,并且该网格中的每个线程执行内核MatAdd以执行一个逐对的加法。

[0352] 在至少一个实施例中,在将CUDA源代码3510转换成HIP源代码3530的同时,CUDA到HIP转换工具3520将CUDA源代码3510中的每个内核调用从CUDA内核启动语法3610转换成HIP内核启动语法3620,并将源代码3510中的任意数量的其他CUDA调用转换为任意数量的其他功能上相似的HIP调用。在至少一个实施例中,HIP内核启动语法3620被指定为“hipLaunchKernelGGL(KernelName,GridSize,BlockSize,SharedMemorySize,Stream,KernelArguments);”。在至少一个实施例中,KernelName,GridSize,BlockSize,ShareMemorySize,Stream和KernelArguments中的每一个在HIP内核启动语法3620中具有与在CUDA内核启动语法3610中(本文先前描述)相同的含义。在至少一个实施例中,参数SharedMemorySize和Stream在HIP内核启动语法3620中是必需的,而在CUDA内核启动语法3610中是可选的。

[0353] 在至少一个实施例中,除了使内核MatAdd在设备上执行的内核调用之外,图36中描绘的HIP源代码3530的一部分与图36中描绘的CUDA源代码3510的一部分相同。在至少一个实施例中,在HIP源代码3530中定义内核MatAdd,具有与在CUDA源代码3510中定义内核MatAdd相同的“\_\_global\_\_”声明说明符。在至少一个实施例中,在HIP源代码3530中的内核调用是“hipLaunchKernelGGL(MatAdd,numBlocks,threadsPerBlock,0,0,A,B,C);”,而CUDA源代码3510中的相应内核调用是“MatAdd<<<numBlocks,threadsPerBlock>>>(A,B,C);”。

[0354] 图37更详细地示出了根据至少一个实施例的图35C的未启用CUDA的GPU 3592。在至少一个实施例中,GPU 3592由圣塔克拉拉市的AMD公司开发。在至少一个实施例中,GPU 3592可以被配置为以高度并行的方式执行计算操作。在至少一个实施例中,GPU 3592被配置为执行图形管线操作,诸如绘制命令、像素操作、几何计算以及与将图像渲染到显示器相关联的其他操作。在至少一个实施例中,GPU 3592被配置为执行与图形无关的操作。在至少一个实施例中,GPU 3592被配置为执行与图形有关的操作和与图形无关的操作两者。在至少一个实施例中,GPU 3592可以被配置为执行HIP源代码3530中包括的设备代码。

[0355] 在至少一个实施例中,GPU 3592包括但不限于任意数量的可编程处理单元3720,命令处理器3710,L2高速缓存3722,存储器控制器3770,DMA引擎3780(1),系统存储器控制器3782,DMA引擎3780(2)和GPU控制器3784。在至少一个实施例中,每个可编程处理单元3720包括但不限于工作负载管理器3730和任意数量的计算单元3740。在至少一个实施例中,命令处理器3710读取来自一个或更多个命令队列(未示出)的命令,并将命令分发给工作负载管理器3730。在至少一个实施例中,对于每个可编程处理单元3720,相关的工作负载管理器3730将工作分发给包括在可编程处理单元3720中的计算单元3740。在至少一个实施例中,每个计算单元3740可以执行任意数量的线程块,但是每个线程块在单个计算单元3740上执行。在至少一个实施例中,工作组是线程块。

[0356] 在至少一个实施例中,每个计算单元3740包括但不限于任意数量的SIMD单元3750和共享存储器3760。在至少一个实施例中,每个SIMD单元3750实现SIMD架构并且被配置为

并行执行操作。在至少一个实施例中,每个SIMD单元3750包括但不限于向量ALU 3752和向量寄存器文件3754。在至少一个实施例中,每个SIMD单元3750执行不同的线程束。在至少一个实施例中,线程束是一组线程(例如16个线程),其中线程束中的每个线程属于单个线程块,并且被配置为基于单个指令集来处理不同的数据集。在至少一个实施例中,可以使用预测来禁用线程束中的一个或更多个线程。在至少一个实施例中,通道是线程。在至少一个实施例中,工作项是线程。在至少一个实施例中,波前是线程束。在至少一个实施例中,线程块中的不同波前可一起同步并经由共享存储器3760进行通信。

[0357] 在至少一个实施例中,可编程处理单元3720被称为“着色引擎”。在至少一个实施例中,除了计算单元3740之外,每个可编程处理单元3720包括但不限于任意数量的专用图形硬件。在至少一个实施例中,每个可编程处理单元3720包括但不限于任意数量(包括零)的几何处理器,任意数量(包括零)的光栅化器,任意数量(包括零)的渲染后端,工作负载管理器3730和任意数量的计算单元3740。

[0358] 在至少一个实施例中,计算单元3740共享L2高速缓存3722。在至少一个实施例中,L2高速缓存3722被分区。在至少一个实施例中,GPU3592中的所有计算单元3740可访问GPU存储器3790。在至少一个实施例中,存储器控制器3770和系统存储器控制器3782促进GPU 3592与主机之间的数据传输,并且DMA引擎3780(1)使能GPU 3592与此主机之间的异步存储器传输。在至少一个实施例中,存储器控制器3770和GPU控制器3784促进GPU 3592与其他GPU 3592之间的数据传输,并且DMA引擎3780(2)使能GPU 3592与其他GPU 3592之间的异步存储器传输。

[0359] 在至少一个实施例中,GPU 3592包括但不限于任意数量和类型的系统互连,该系统互连促进在GPU 3592内部或外部的任意数量和类型的直接或间接链接的组件之间的数据和控制传输。在至少一个实施例中,GPU3592包括但不限于耦合到任意数量和类型的外围设备的任意数量和类型的I/O接口(例如,PCIe)。在至少一个实施例中,GPU 3592可以包括但不限于任意数量(包括零)的显示引擎和任意数量(包括零)的多媒体引擎。在至少一个实施例中,GPU 3592实现了存储器子系统,该存储器子系统包括但不限于任意数量和类型的存储器控制器(例如,存储器控制器3770和系统存储器控制器3782)以及专用于一个组件或在多个组件之间共享的存储器设备(例如,共享存储器3760)。在至少一个实施例中,GPU3592实现了高速缓存子系统,该高速缓存子系统包括但不限于一个或更多个高速缓存存储器(例如,L2高速缓存3722),每个高速缓存存储器可以是私有的或在任意数量的组件(例如,SIMD单元3750,计算单元3740和可编程处理单元3720)之间共享。

[0360] 图38示出了根据至少一个实施例的示例性CUDA网格3820的线程如何被映射到图37的不同计算单元3740。在至少一个实施例中,并且仅出于说明目的,网格3820具有BX乘以BY乘以1的GridSize和TX乘以TY乘以1的BlockSize。因此,在至少一个实施例中,网格3820包括但不限于(BX\*BY)线程块3830,每个线程块3830包括但不限于(TX\*TY)线程3840。线程3840在图38中被描绘为弯曲箭头。

[0361] 在至少一个实施例中,网格3820被映射到可编程处理单元3720(1),该可编程处理单元3720(1)包括但不限于计算单元3740(1)-3740(C)。在至少一个实施例中并且如图所示,将(BJ\*BY)线程块3830映射到计算单元3740(1),并且将其余线程块3830映射到计算单元3740(2)。在至少一个实施例中,每个线程块3830可以包括但不限于任意数量的线程束,

并且每个线程束被映射到图37的不同的SIMD单元3750。

[0362] 在至少一个实施例中,给定线程块3830中的线程束可以一起同步并通过关联的计算单元3740中包括的共享存储器3760进行通信。例如并且在至少一个实施例中,线程块3830 (BJ, 1) 中的线程束可以一起同步并通过共享存储器3760 (1) 进行通信。例如并且在至少一个实施例中,线程块3830 (BJ+1, 1) 中的线程束可以一起同步并通过共享存储器3760 (2) 进行通信。

[0363] 可以根据以下条款描述本公开的至少一个实施例:

[0364] 1. 一种处理器,包括:

[0365] 一个或更多个电路,用于至少部分地基于可用于传输数据的不同路径的一个或更多个特性,确定将所述数据从计算机系统的第一硬件组件传输到所述计算机系统的第二硬件组件的路径。

[0366] 2. 根据条款1所述的处理器,其中所述第一硬件组件和所述第二硬件组件中的一个或更多个是图形处理单元 (GPU)。

[0367] 3. 根据条款1-2中的任意一项所述的处理器,其中所述一个或更多个电路用于至少部分地基于包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示来确定所述路径。

[0368] 4. 根据条款1-3中的任意一项所述的处理器,其中所述硬件拓扑的表示是设备层次结构树,并且所述一个或更多个电路还用于至少部分地基于外围组件互连快速 (PCIe) 总线设备功能 (BDF) 信息来生成所述设备层次结构树。

[0369] 5. 根据条款1-4中的任意一项所述的处理器,其中所述第一硬件组件由第一操作系统 (OS) 实例控制,所述第二硬件组件由第二操作系统实例控制,并且所述第一硬件组件和所述第二硬件组件连接在节点间路径上,所述节点间路径包括存储器结构或网络结构中的至少一个。

[0370] 6. 根据条款1-5中的任意一项所述的处理器,其中所述一个或更多个电路还用于确定可用块设备的集合,并至少部分地基于所述可用块设备的集合来确定所述路径。

[0371] 7. 根据条款1-6中的任意一项所述的处理器,其中所述路径包括由中间设备管理的缓冲器。

[0372] 8. 根据条款1-7中的任意一项所述的处理器,其中所述一个或更多个电路还用于确定所述缓冲器允许直接存储器访问 (DMA) 并且具有预定量的可用空间。

[0373] 9. 根据条款1-8中的任意一项所述的处理器,其中所述一个或更多个电路还用于确定可用块设备的集合,并至少部分地基于可用链路的集合来确定所述路径。

[0374] 10. 根据条款1-9中的任意一项所述的处理器,其中所述一个或更多个电路还用于确定对应于多个动态组件条件的多个值,并至少部分地基于所述多个值来确定所述路径。

[0375] 11. 一种其上存储有一组指令的机器可读介质,所述一组指令如果由一个或更多个处理器执行,则使所述一个或更多个处理器至少:

[0376] 至少部分地基于可用于传输数据的不同路径的一个或更多个特性,确定将所述数据从计算机系统的第一硬件组件传输到所述计算机系统的第二硬件组件的路径。

[0377] 12. 根据条款11所述的机器可读介质,其中所述第一硬件组件和所述第二硬件组件中的一个或更多个是图形处理单元 (GPU)。

[0378] 13.根据条款11-12中的任意一项所述的机器可读介质,其中所述指令如果由所述一个或更多个处理器执行,进一步使所述一个或更多个处理器至少部分地基于包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示来确定所述路径。

[0379] 14.根据条款11-13中的任意一项所述的机器可读介质,其中所述硬件拓扑的表示是设备层次结构树,并且所述指令如果由所述一个或更多个处理器执行,进一步使所述一个或更多个处理器至少部分地基于外围组件互连快速 (PCIe) 总线设备功能 (BDF) 信息来生成所述设备层次结构树。

[0380] 15.根据条款11-14中的任意一项所述的机器可读介质,其中所述指令如果由所述一个或更多个处理器执行,进一步使所述一个或更多个处理器确定可用块设备的集合,并至少部分地基于所述可用块设备的集合来确定所述路径。

[0381] 16.根据条款11-15中的任意一项所述的机器可读介质,其中所述路径包括由中间设备管理的缓冲器。

[0382] 17.根据条款11-16中的任意一项所述的机器可读介质,其中所述指令如果由所述一个或更多个处理器执行,进一步使所述一个或更多个处理器确定所述缓冲器允许直接存储器访问 (DMA) 并且具有预定量的可用空间。

[0383] 18.根据条款11-17中的任意一项所述的机器可读介质,其中所述指令如果由所述一个或更多个处理器执行,进一步使所述一个或更多个处理器确定对应于多个动态组件条件的多个值,并至少部分地基于所述多个值来确定所述路径。

[0384] 19.一种方法,包括:

[0385] 至少部分地基于可用于传输数据的不同路径的一个或更多个特性,确定将所述数据从计算机系统的第一硬件组件传输到所述第二硬件组件的路径。

[0386] 20.根据条款19所述的方法,还包括经由所述路径将数据集从所述第一硬件组件传输到所述第二硬件组件。

[0387] 21.根据条款19-20中的任意一项所述的方法,其中所述第一硬件组件和所述第二硬件组件中的一个或更多个是图形处理单元 (GPU)。

[0388] 22.根据条款19-21中的任意一项所述的方法,其中确定所述路径包括至少部分地基于包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示来确定所述路径。

[0389] 23.根据条款19-22中的任意一项所述的方法,还包括:确定可用块设备的集合,其中至少部分地基于所述可用块设备的集合来确定所述路径。

[0390] 24.根据条款19-23中的任意一项所述的方法,其中所述路径包括由中间设备管理的缓冲器。

[0391] 25.根据条款19-24中的任意一项所述的方法,还包括:确定所述缓冲器允许直接存储器访问 (DMA) 并且具有预定量的可用空间。

[0392] 26.根据条款19-25中的任意一项所述的方法,还包括:确定对应于多个动态组件条件的多个值,其中至少部分地基于所述多个值来确定所述路径。

[0393] 27.一种系统,包括:

[0394] 一个或更多个处理器,用于至少部分地基于可用于传输数据的不同路径的一个或更多个特性,确定将所述数据从计算机系统的第一硬件组件传输到所述第二硬件组件的路径;以及

[0395] 一个或多个存储器,用于存储表示所述路径的数据集。

[0396] 28.根据条款27所述的系统,其中所述第一硬件组件和所述第二硬件组件中的一个或多个是图形处理单元(GPU)。

[0397] 29.根据条款27-28中的任意一项所述的系统,其中所述一个或多个处理器用于至少部分地基于包括所述第一硬件组件和所述第二硬件组件的硬件拓扑的表示来确定所述路径。

[0398] 30.根据条款27-29中的任意一项所述的系统,其中所述硬件拓扑的表示是设备层次结构树,并且所述一个或多个处理器还用于至少部分地基于外围组件互连快速(PCIe)总线设备功能(BDF)信息来生成所述设备层次结构树。

[0399] 31.根据条款27-30中的任意一项所述的系统,其中所述一个或多个处理器还用于确定可用块设备的集合,并至少部分地基于所述可用块设备的集合来确定所述路径。

[0400] 32.根据条款27-31中的任意一项所述的系统,其中所述路径包括由中间设备管理的缓冲器。

[0401] 33.根据条款27-32中的任意一项所述的系统,其中所述一个或多个处理器还用于确定所述缓冲器允许直接存储器访问(DMA)并且具有预定量的可用空间

[0402] 34.根据条款27-33中的任意一项所述的系统,其中所述一个或多个处理器还用于确定对应于多个动态组件条件的多个值,并至少部分地基于所述多个值来确定所述路径。

[0403] 其他变型在本公开的精神内。因此,尽管公开的技术易于进行各种修改和替代构造,但是某些示出的其实施例在附图中示出并且已经在上面进行了详细描述。然而,应理解,无意将公开内容限制为所公开的一种或更多种特定形式,而是相反,其意图是涵盖落入如所附权利要求书所定义的本公开内容的精神和范围内的所有修改、替代构造和等同物。

[0404] 除非另有说明或显然与上下文矛盾,否则在描述所公开的实施例的上下文中(特别是在所附权利要求的上下文中),术语“一”和“一个”和“该”以及类似指代的使用应被解释为涵盖单数和复数,而不是作为术语的定义。除非另有说明,否则术语“包括”、“具有”、“包含”和“含有”应被解释为开放式术语(意味着“包括但不限于”)。术语“连接”(在未经修改时指的是物理连接)应解释为部分或全部包含在内、附接到或连接在一起,即使有某些介入。除非本文另外指出,否则本文中对数值范围的引用仅旨在用作分别指代落入该范围内的每个单独值的简写方法,并且每个单独值都被并入说明书中,就如同其在本文中被单独叙述一样。除非另外指出或与上下文矛盾,否则术语“集”(例如“项目集”)或“子集”的使用应解释为包括一个或多个成员的非空集合。此外,除非另外指出或与上下文矛盾,否则术语相应集的“子集”不一定表示对应集的适当子集,而是子集和对应集可以相等。

[0405] 除非以其他方式明确指出或与上下文明显矛盾,否则诸如“A,B和C中的至少一个”或“A,B与C中的至少一个”形式的短语之类的连接语在上下文中理解为通常用来表示项目、条款等,其可以是A或B或C,也可以是A和B和C集的任何非空子集。例如,在具有三个成员的集的说明性示例中,连接短语“A,B和C中的至少一个”和“A,B与C中的至少一个”是指以下任意集:{A},{B},{C},{A,B},{A,C},{B,C},{A,B,C}。因此,这种连接语言通常不旨在暗示某些实施例要求存在A中的至少一个,B中的至少一个和C中的至少一个。另外,除非另有说明或与上下文矛盾,否则术语“多个”表示复数的状态(例如,“多个项目”表示多个项目)。多个

项目中项目的数量至少为两个,但如果明确指示或通过上下文指示,则可以更多。此外,除非另有说明或从上下文中可以清楚得知,否则短语“基于”是指“至少部分基于”而不是“仅基于”。

[0406] 除非本文另外指出或与上下文明显矛盾,否则本文描述的过程的操作可以任何合适的顺序执行。在至少一个实施例中,诸如本文所述的那些过程(或其变形和/或其组合)的过程在配置有可执行指令的一个或更多个计算机系统的控制下执行,并且被实现为代码(例如,可执行指令,一个或更多个计算机程序或一个或更多个应用程序),该代码通过硬件或其组合在一个或更多个处理器上共同执行。在至少一个实施例中,代码以例如计算机程序的形式存储在计算机可读存储介质上,该计算机程序包括可由一个或更多个处理器执行的多个指令。在至少一个实施例中,一种计算机可读存储介质是一种非暂时性计算机可读存储介质,其排除了暂时性信号(例如,传播的瞬态电或电磁传输),但包括了非暂时性数据存储电路(例如,缓冲器,高速缓存和队列)。在至少一个实施例中,代码(例如,可执行代码或源代码)被存储在其上存储有可执行指令(或其他存储器以存储可执行指令)的一组一个或更多个非暂时性计算机可读存储介质上,该可执行指令(或其他存储器以存储可执行指令)在以下情况下被存储。由计算机系统的一个或更多个处理器执行(例如,作为被执行的结果)而导致的计算机系统执行本文所述的操作。在至少一个实施例中,一组非暂时性计算机可读存储介质包括多个非暂时性计算机可读存储介质,并且多个非暂时性计算机可读存储介质中的一个或更多个单独的非暂时性存储介质缺少全部多个非暂时性计算机可读存储介质共同存储所有代码。在至少一个实施例中,执行可执行指令,以使得不同的指令由不同的处理器执行,例如,非暂时性计算机可读存储介质存储指令,并且主中央处理单元(“CPU”)执行一些指令,而图形处理单元(“GPU”)执行其他指令。在至少一个实施例中,计算机系统的不同组件具有单独的处理器,并且不同的处理器执行指令的不同子集。

[0407] 因此,在至少一个实施例中,计算机系统被配置为实现单独地或共同地执行本文所述的过程的操作的一种或更多种服务,并且这样的计算机系统被配置有能够实现操作的适用的硬件和/或软件。此外,实现本公开的至少一个实施例的计算机系统是单个设备,并且在另一实施例中,是一种分布式计算机系统,其包括以不同方式操作的多个设备,使得分布式计算机系统执行本文所述的操作,并且使得单个设备不执行所有操作。

[0408] 本文提供的任何和所有示例或示例性语言(例如,“诸如”)的使用仅旨在更好地阐明本公开的实施例,并且不对公开的范围构成限制,除非另有要求。说明书中的任何语言都不应被解释为表示任何未要求保护的要素对于实施公开都是必不可少的。

[0409] 本文引用的所有参考文献,包括出版物,专利申请和专利,均以引用的方式并入本文,如同每个参考文献被单独且具体地指示以引用的方式并入本文一样。

[0410] 在描述和权利要求中,可以使用术语“耦合”和“连接”以及它们的派生词。应当理解,这些术语可能不旨在作为彼此的同义词。相反,在特定示例中,“连接”或“耦合”可用于指示两个或更多个元件彼此直接或间接物理或电接触。“耦合”也可能意味着两个或多个元素彼此不直接接触,但仍彼此协作或交互。

[0411] 除非另有明确说明,否则可以理解,在整个说明书中,诸如“处理”,“计算”,“计算”,“确定”等,是指计算机或计算系统或类似的电子计算设备的动作和/或过程,将计算系统的寄存器和/或存储器中表示为物理量(例如电子)的数据处理和/或转换为类似表示为

计算系统的存储器,寄存器或其他此类信息存储,传输或显示设备中的物理量的其他数据。

[0412] 以类似的方式,术语“处理器”可以指处理来自寄存器和/或存储器的电子数据并将该电子数据转换成可以存储在寄存器和/或存储器中的其他电子数据的任何设备或存储器的一部分。作为非限制性示例,“处理器”可以是CPU或GPU。“计算平台”可以包括一个或多个处理器。如本文所使用的,“软件”过程可以包括例如随时间执行工作的软件和/或硬件实体,诸如任务,线程和智能代理。同样,每个过程可以指代多个过程,以连续地或间歇地顺序地或并行地执行指令。术语“系统”和“方法”在本文中可以互换使用,只要系统可以体现一种或更多种方法,并且方法可以被认为是系统。

[0413] 在本文件中,可以参考获得,获取,接收或将模拟或数字数据输入子系统,计算机系统或计算机实现的机器中。可以通过多种方式来完成获得,获取,接收或输入模拟和数字数据的过程,例如通过接收作为函数调用或对应用程序接口的调用的参数的数据。在一些实施方式中,可以通过经由串行或并行接口传输数据来完成获得,获取,接收或输入模拟或数字数据的过程。在另一实施方式中,可以通过经由计算机网络将数据从提供实体传输到获取实体来完成获得,获取,接收或输入模拟或数字数据的过程。也可以参考提供,输出,传输,发送或呈现模拟或数字数据。在各种示例中,提供,输出,传输,发送或呈现模拟或数字数据的过程可以通过将数据作为函数调用的输入或输出参数,应用程序编程接口或进程间通信机制的参数进行传输来实现。

[0414] 尽管上面的讨论阐述了所描述的技术的示例实现,但是其他架构可以用于实现所描述的功能,并且旨在落入本公开的范围。此外,尽管出于讨论的目的在上面定义了具体的职责分配,但是根据情况,可以以不同的方式分配和划分各种功能和职责。

[0415] 此外,尽管已经用特定于结构特征和/或方法动作的语言描述了主题,但是应当理解,所附权利要求书所要求保护的主体不必限于所描述的特定特征或动作。而是,公开了特定的特征和动作作为实现权利要求的示例性形式。

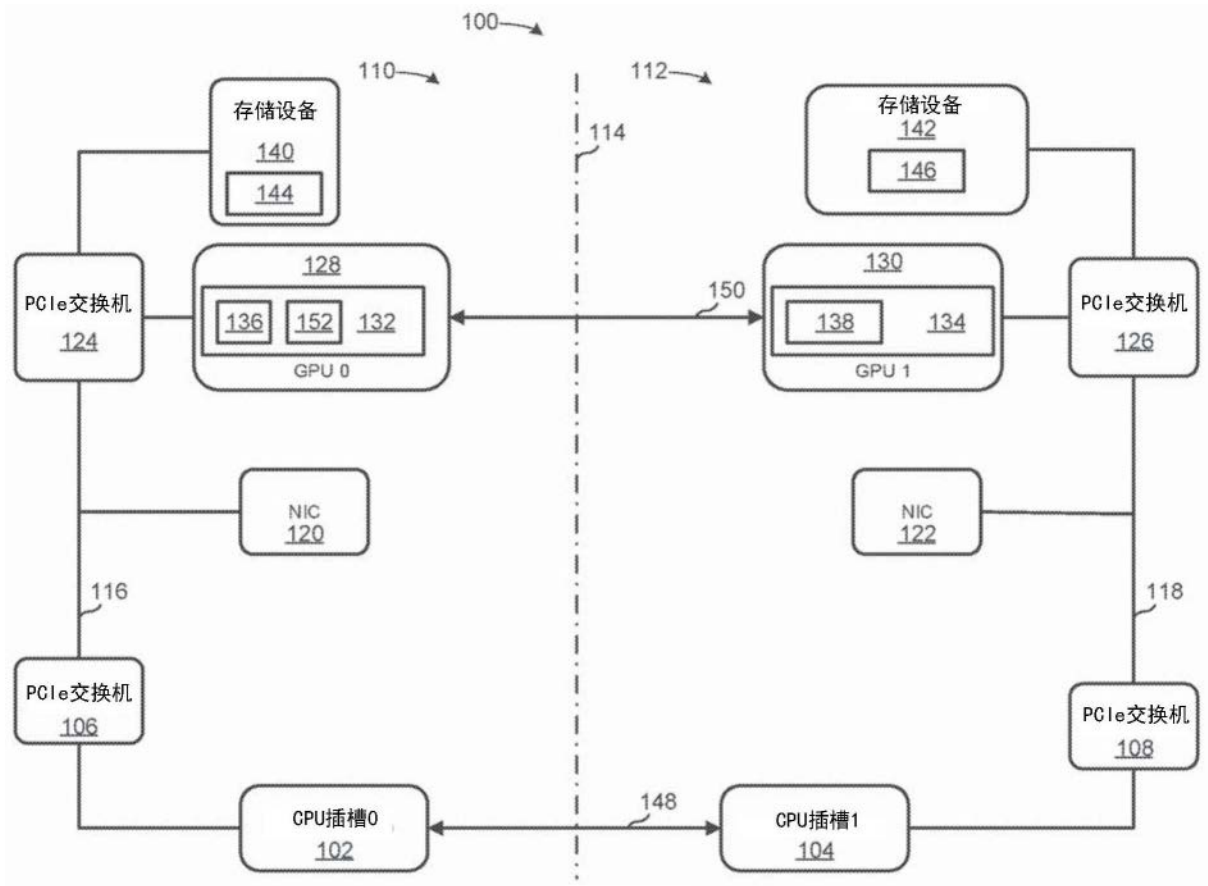


图1A

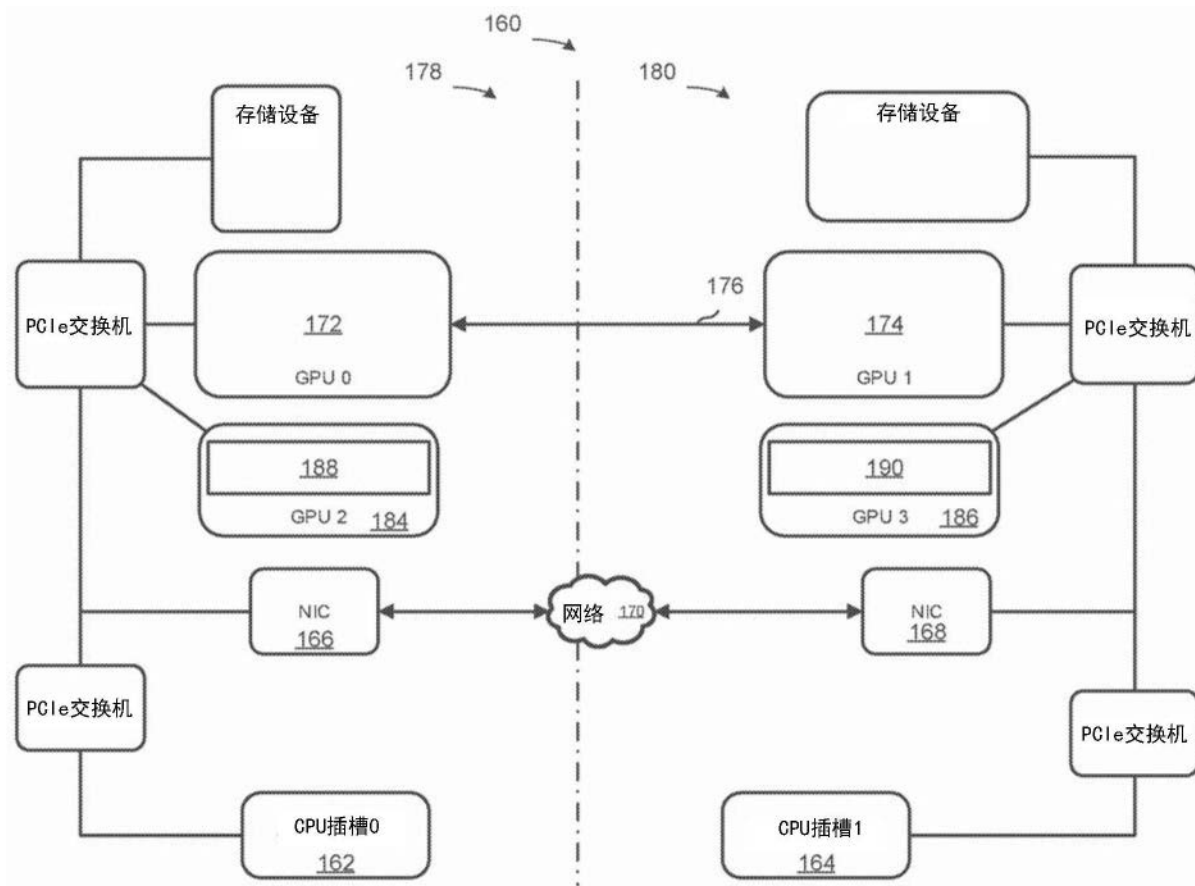


图1B

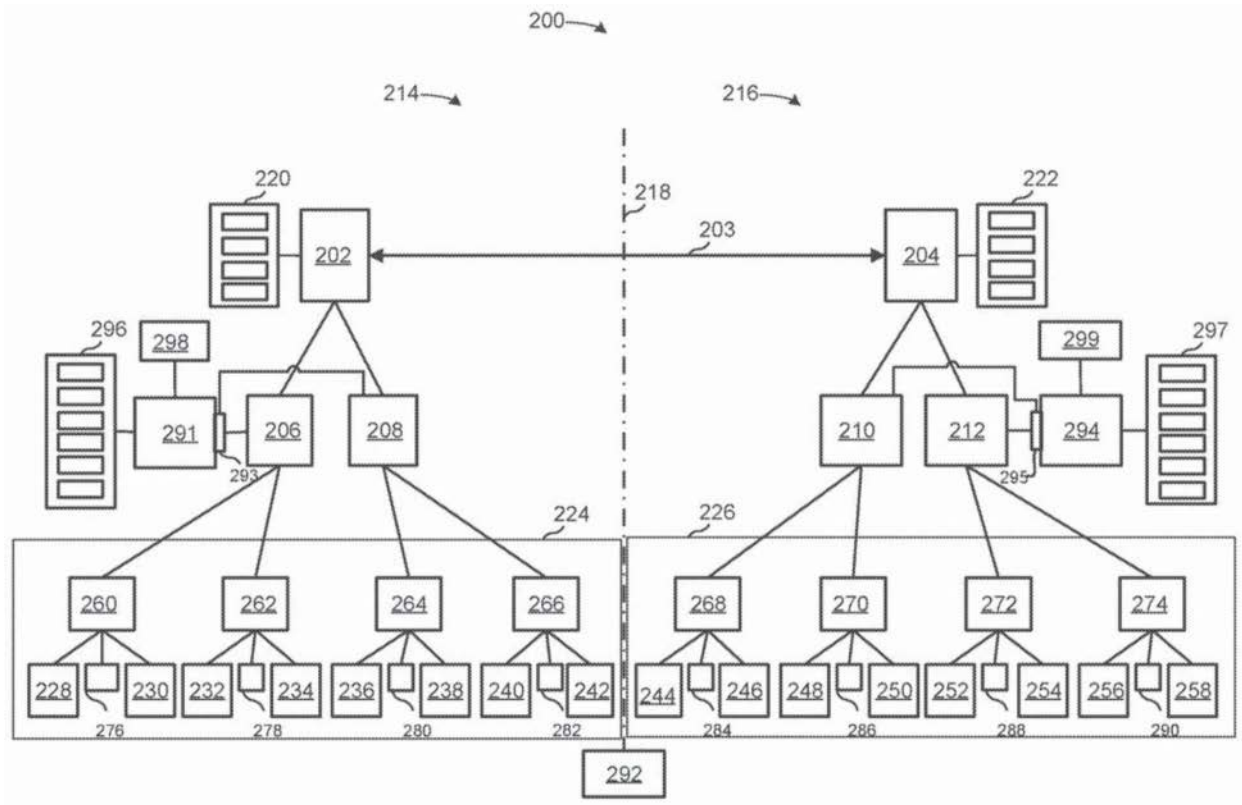


图2

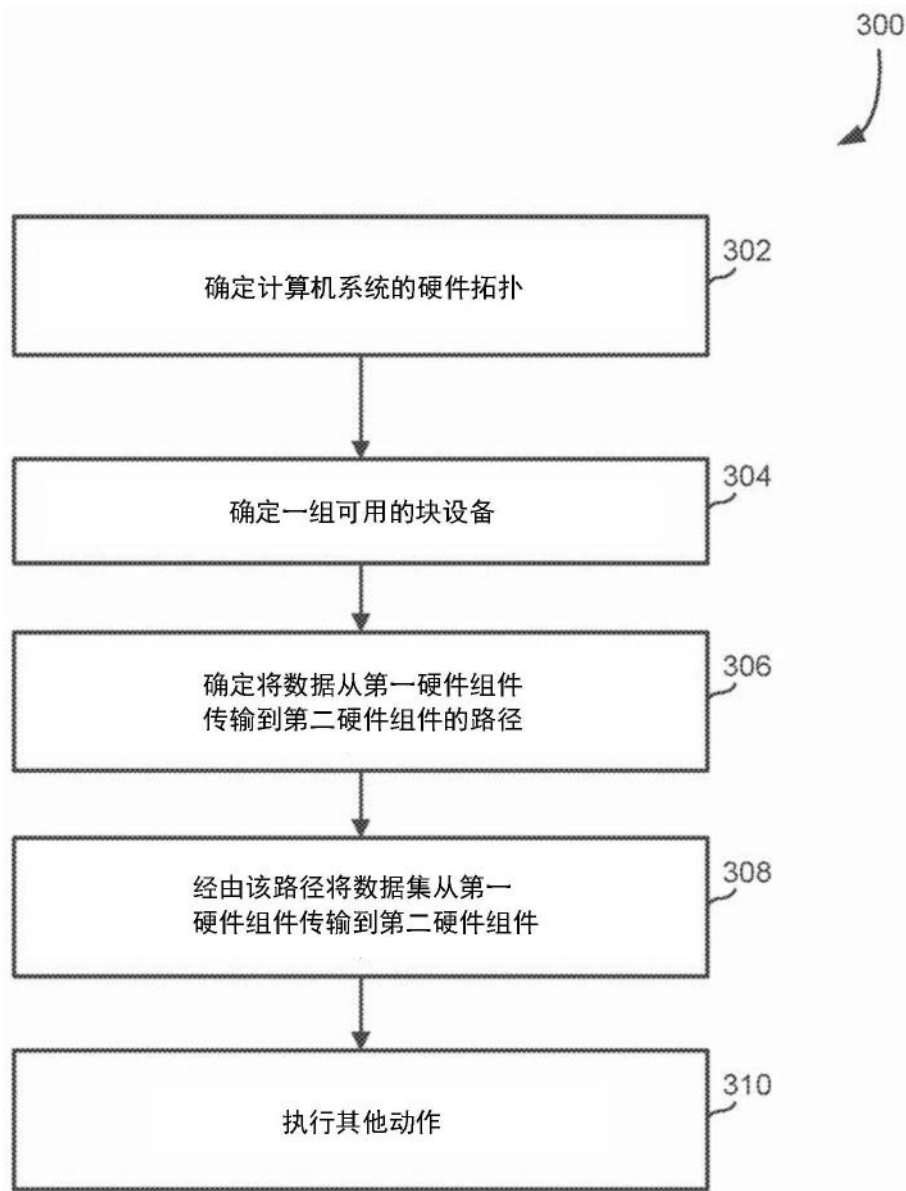


图3

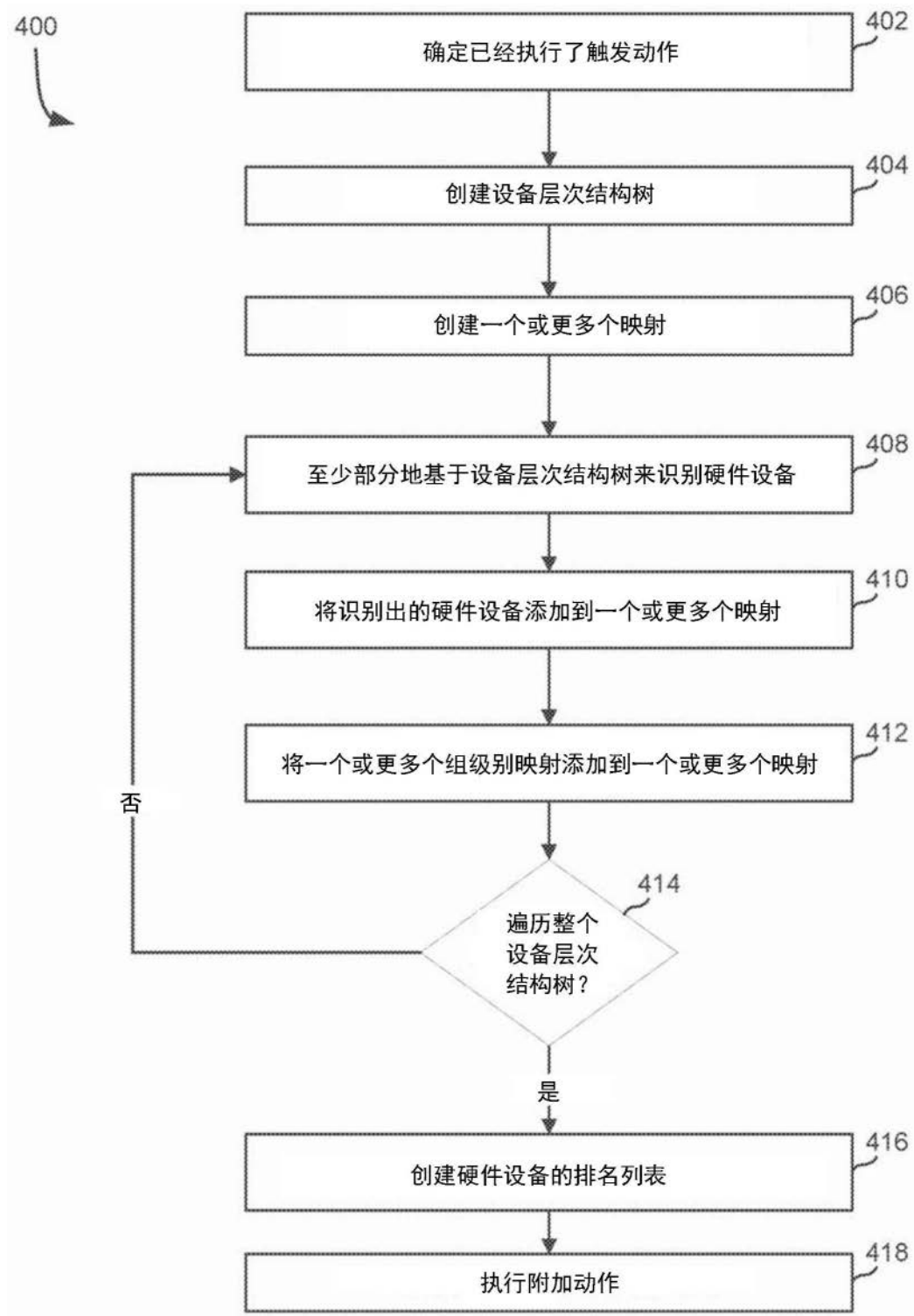


图4

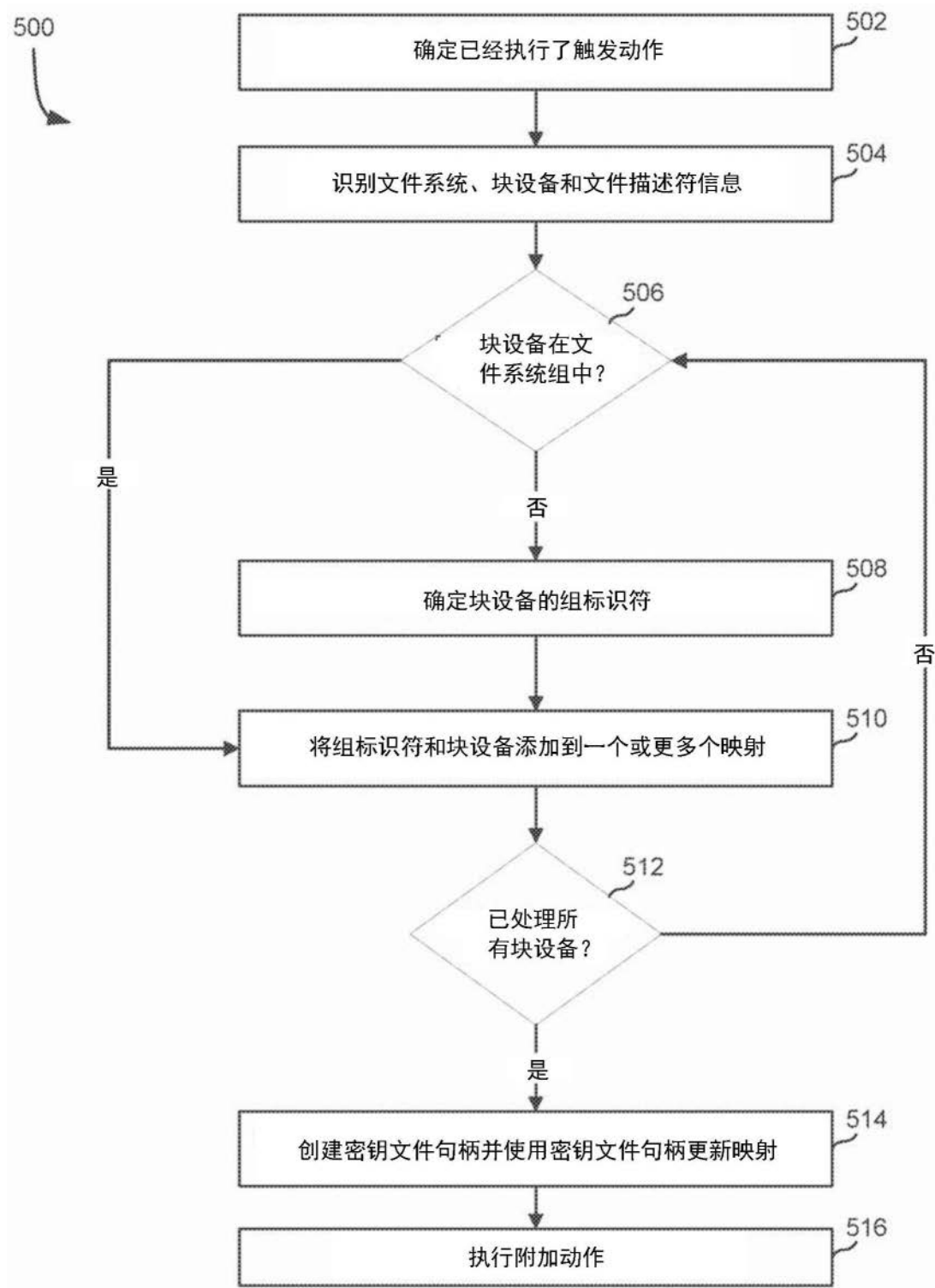


图5

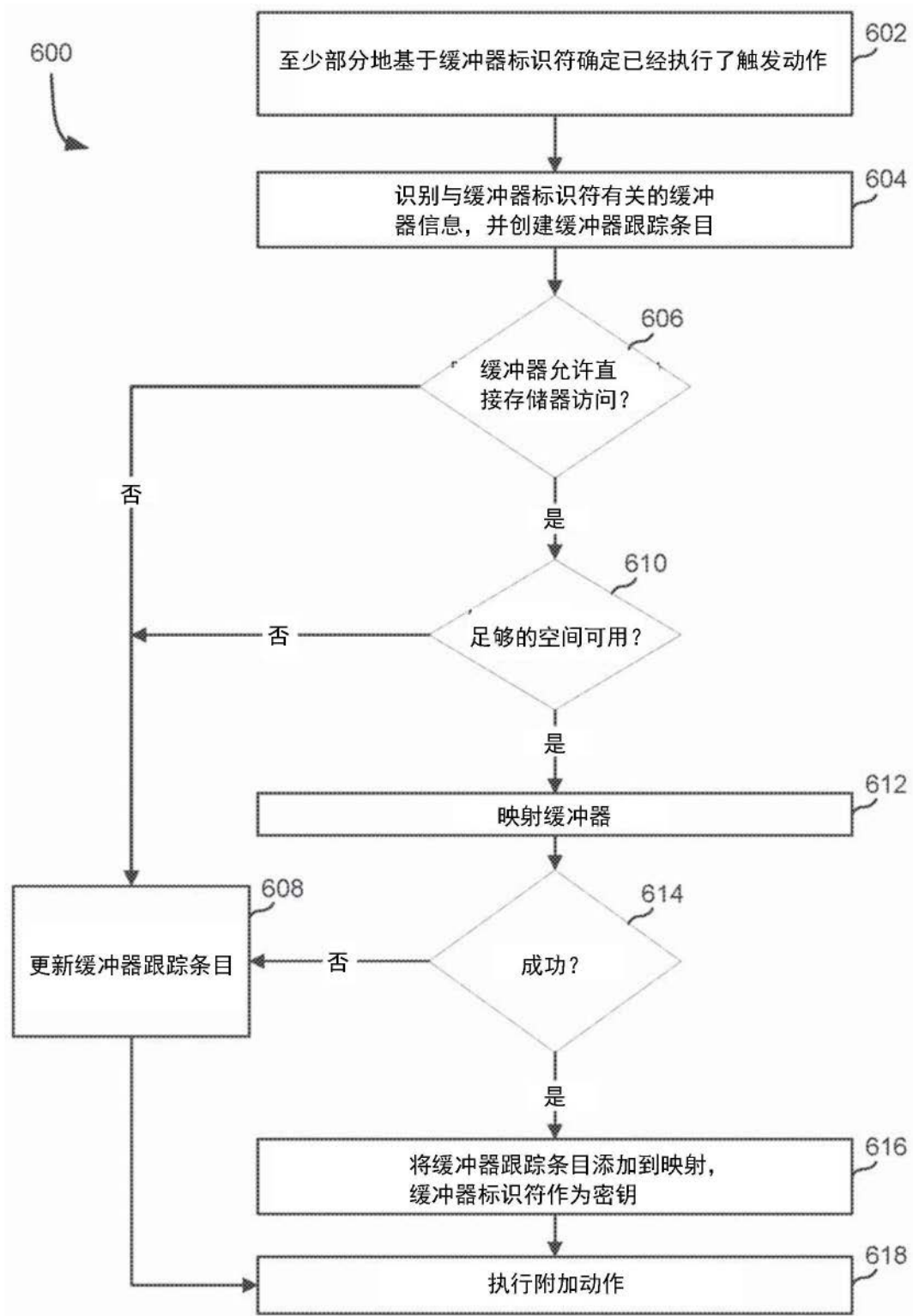


图6

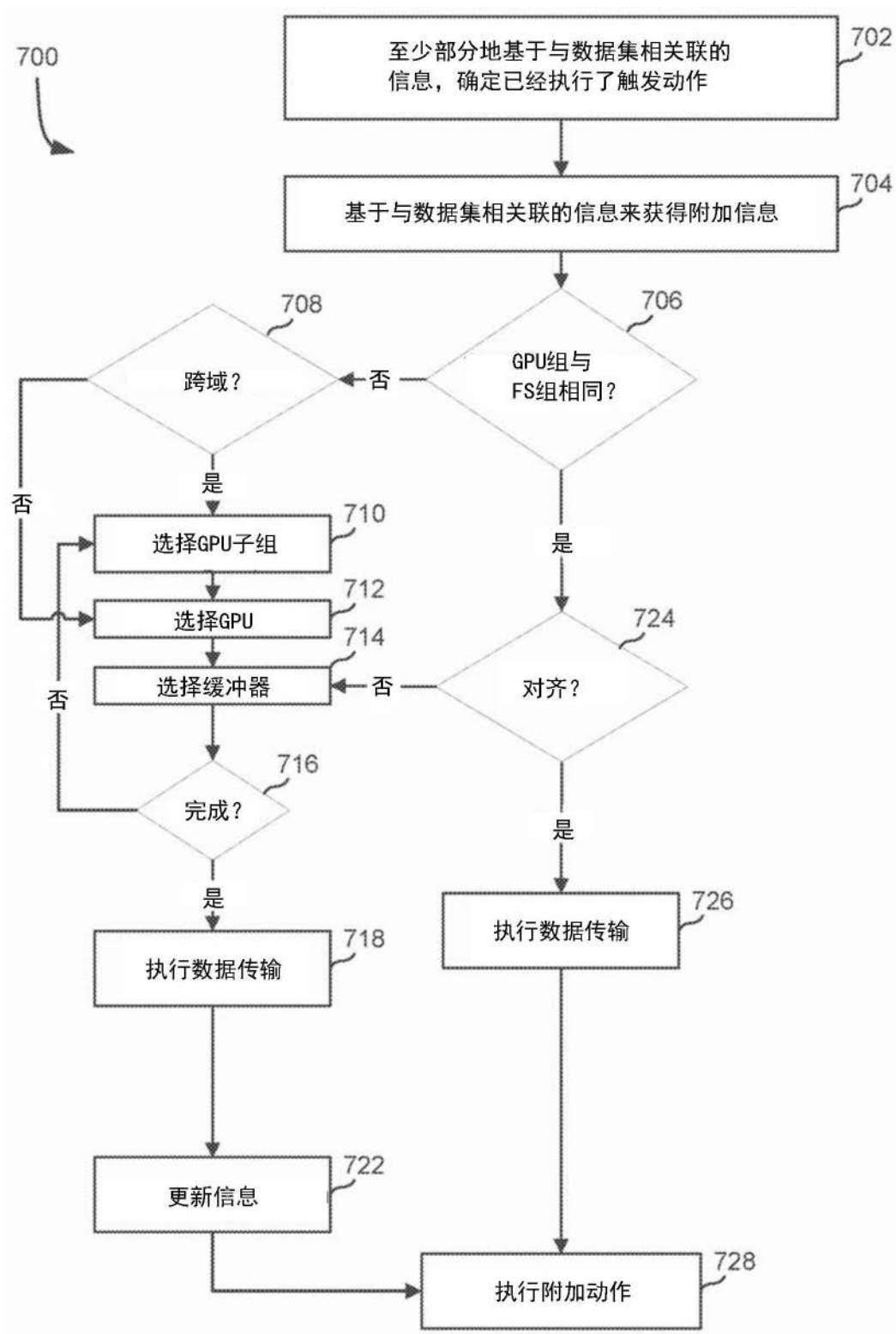


图7



图8

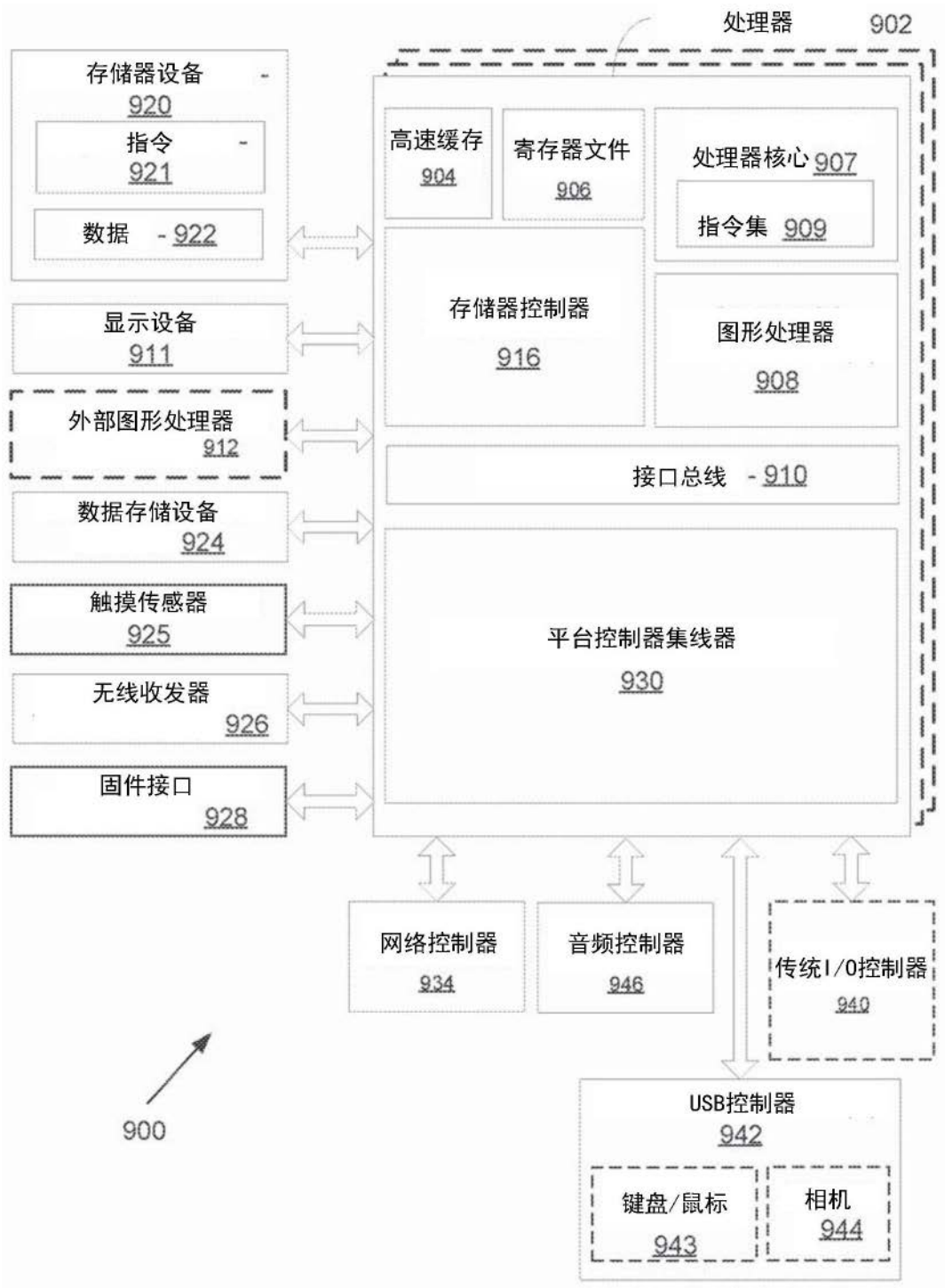


图9

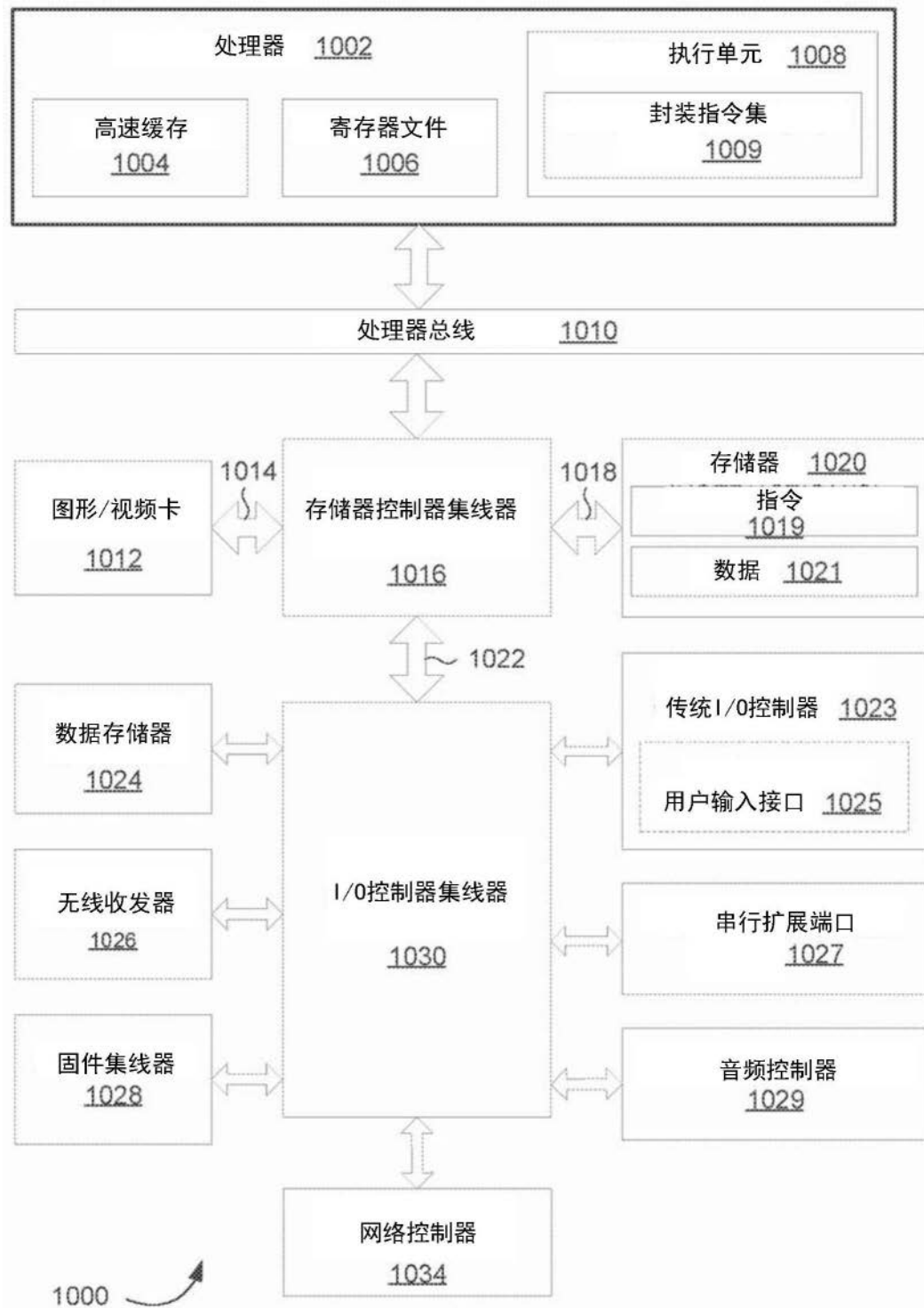


图10

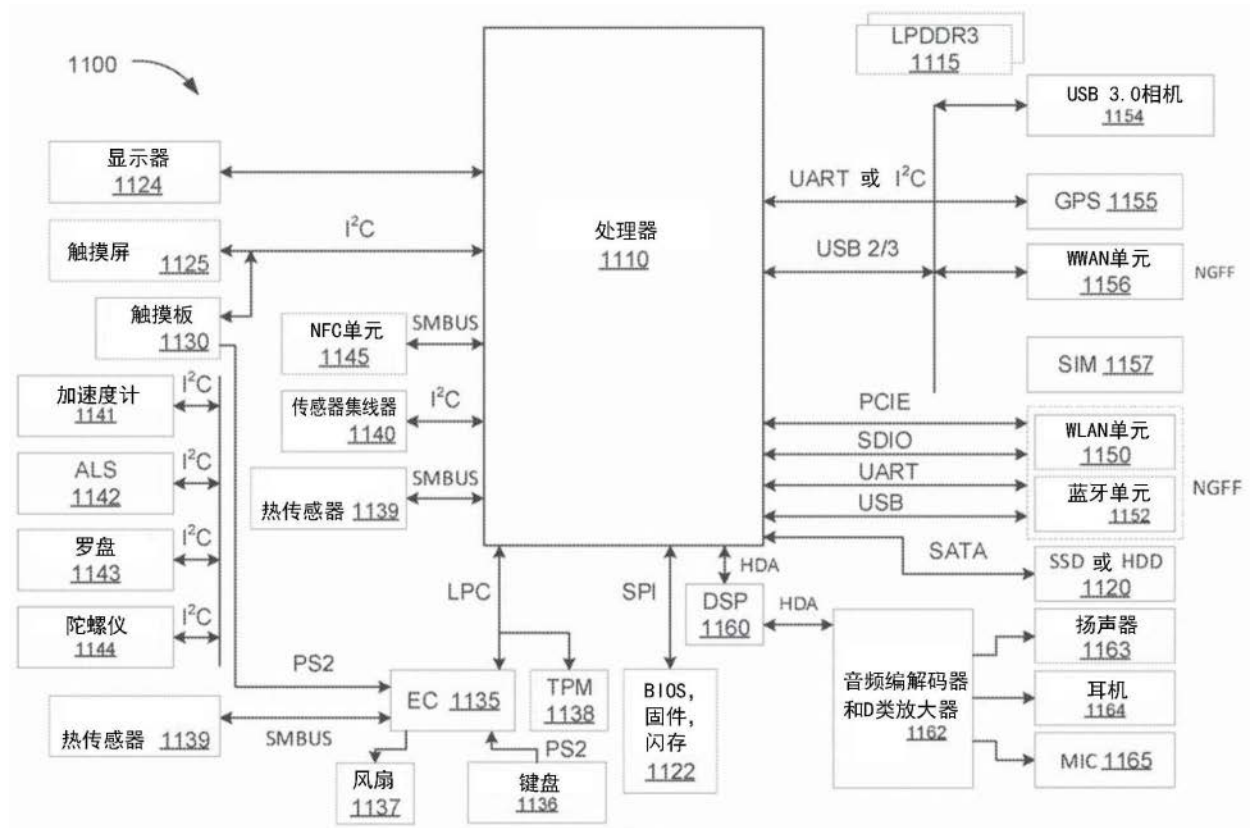


图11

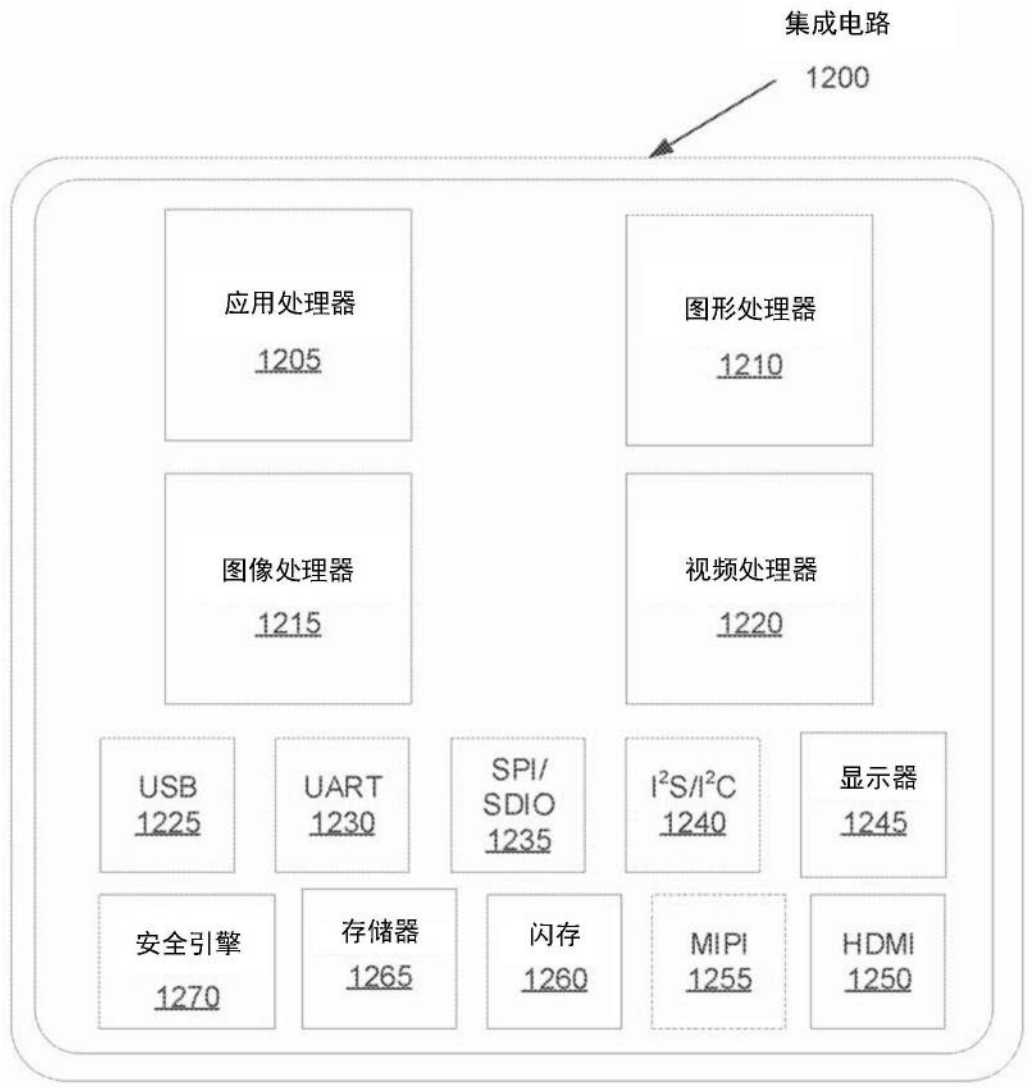


图12

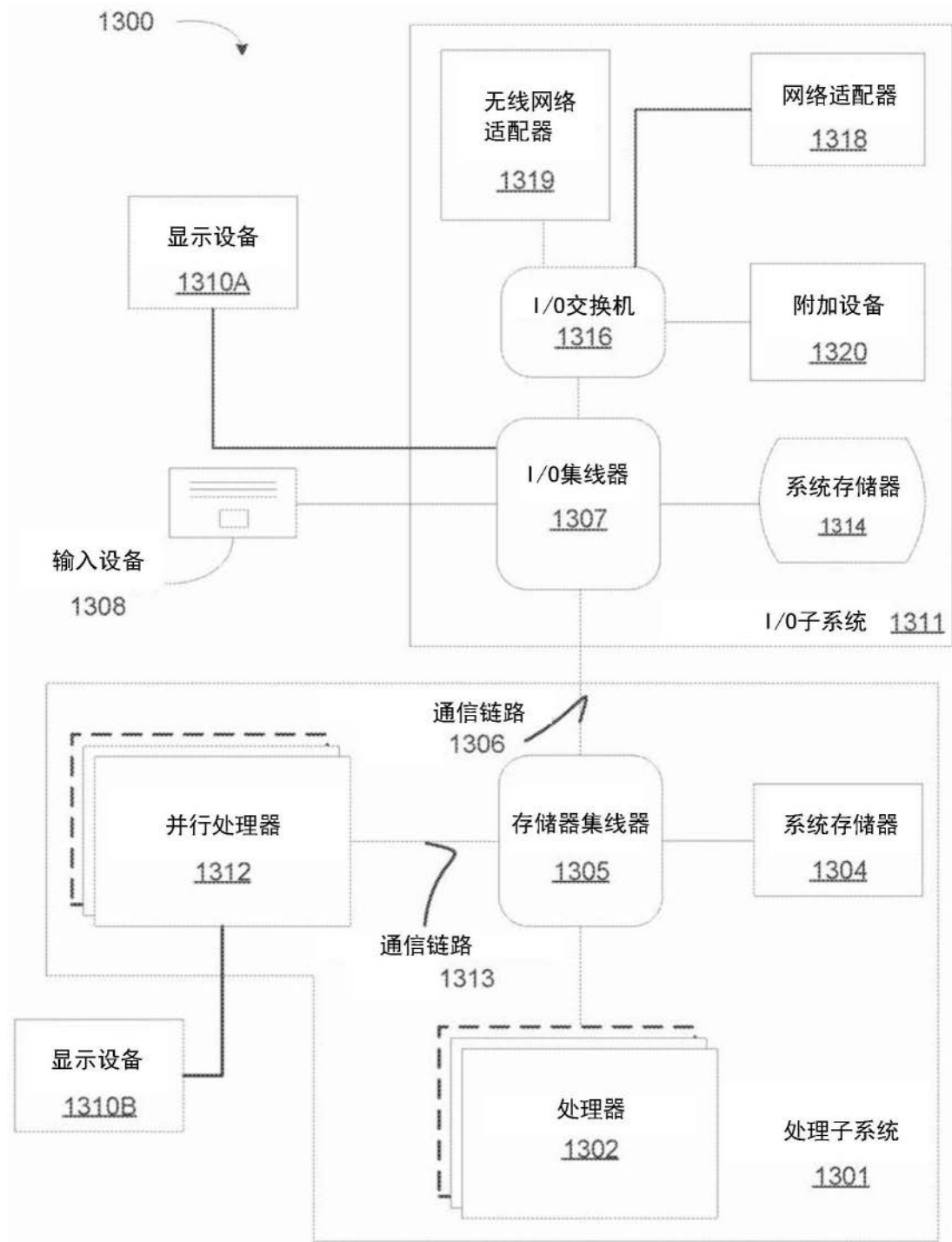


图13

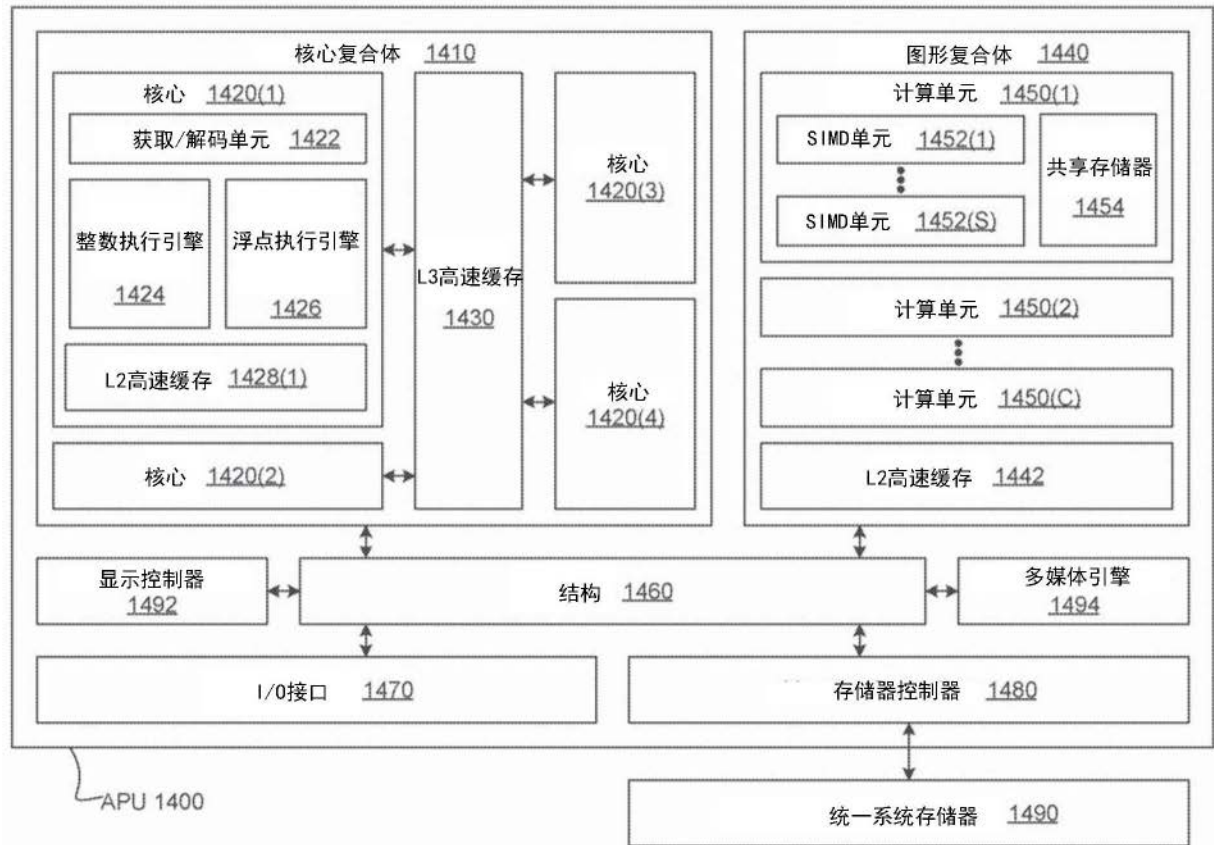


图14

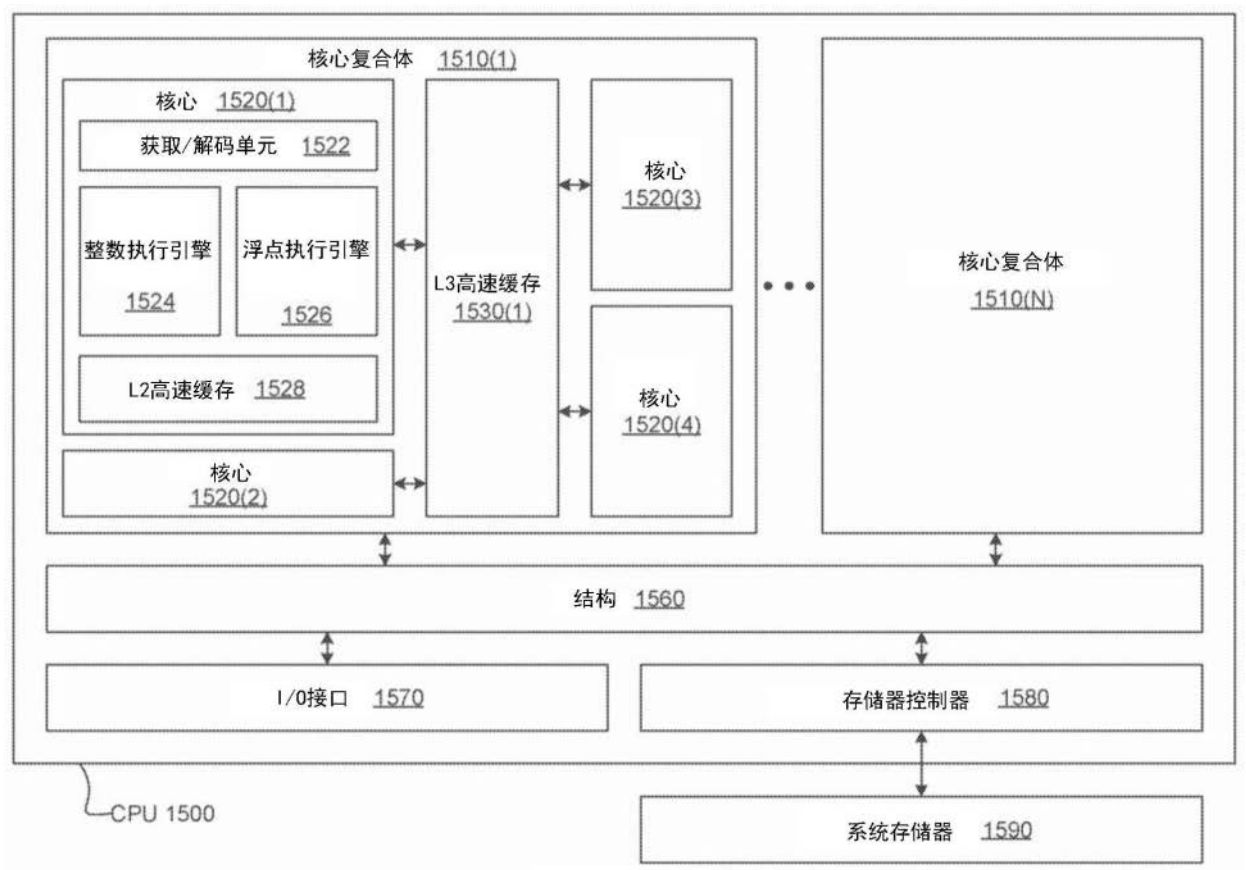


图15

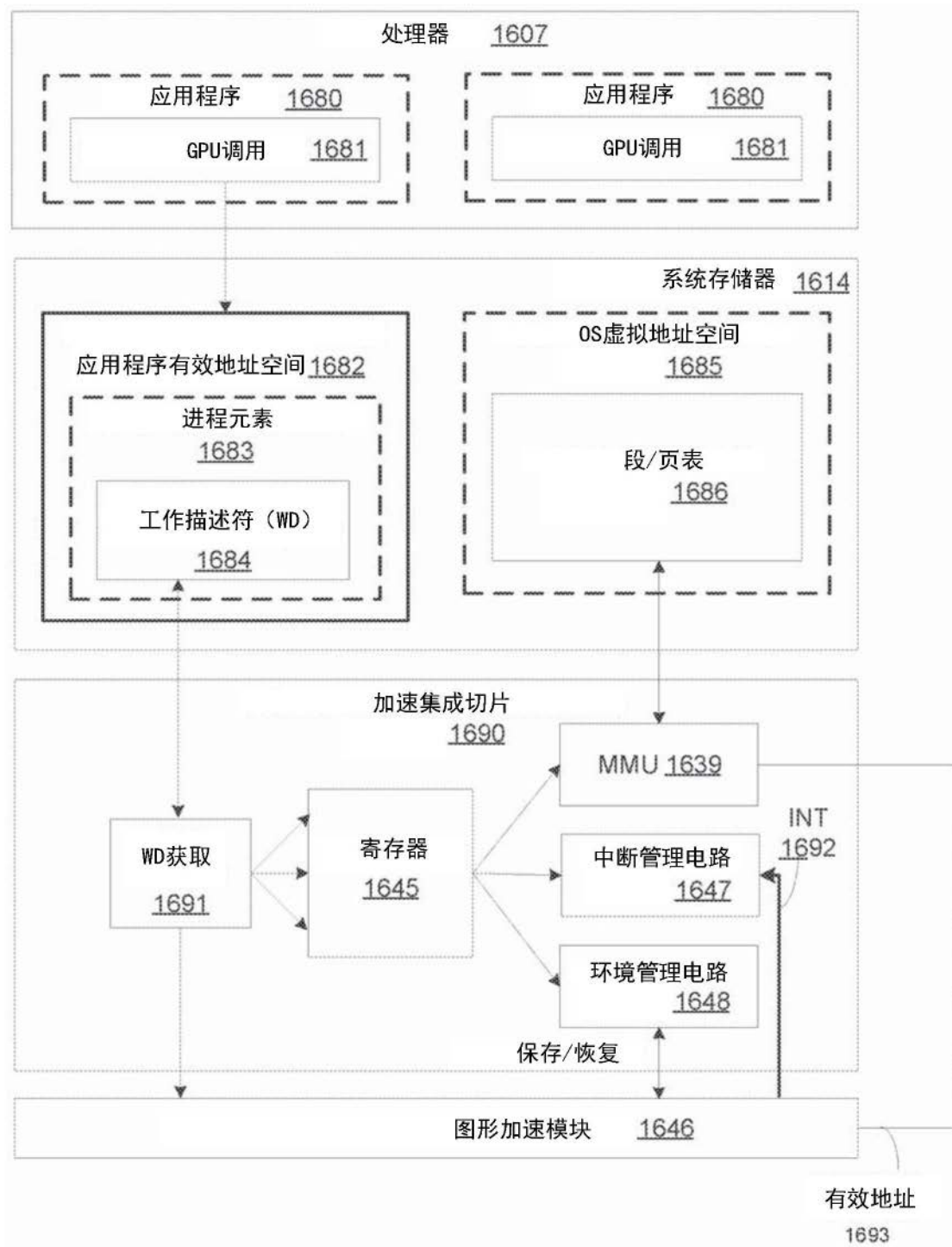


图16

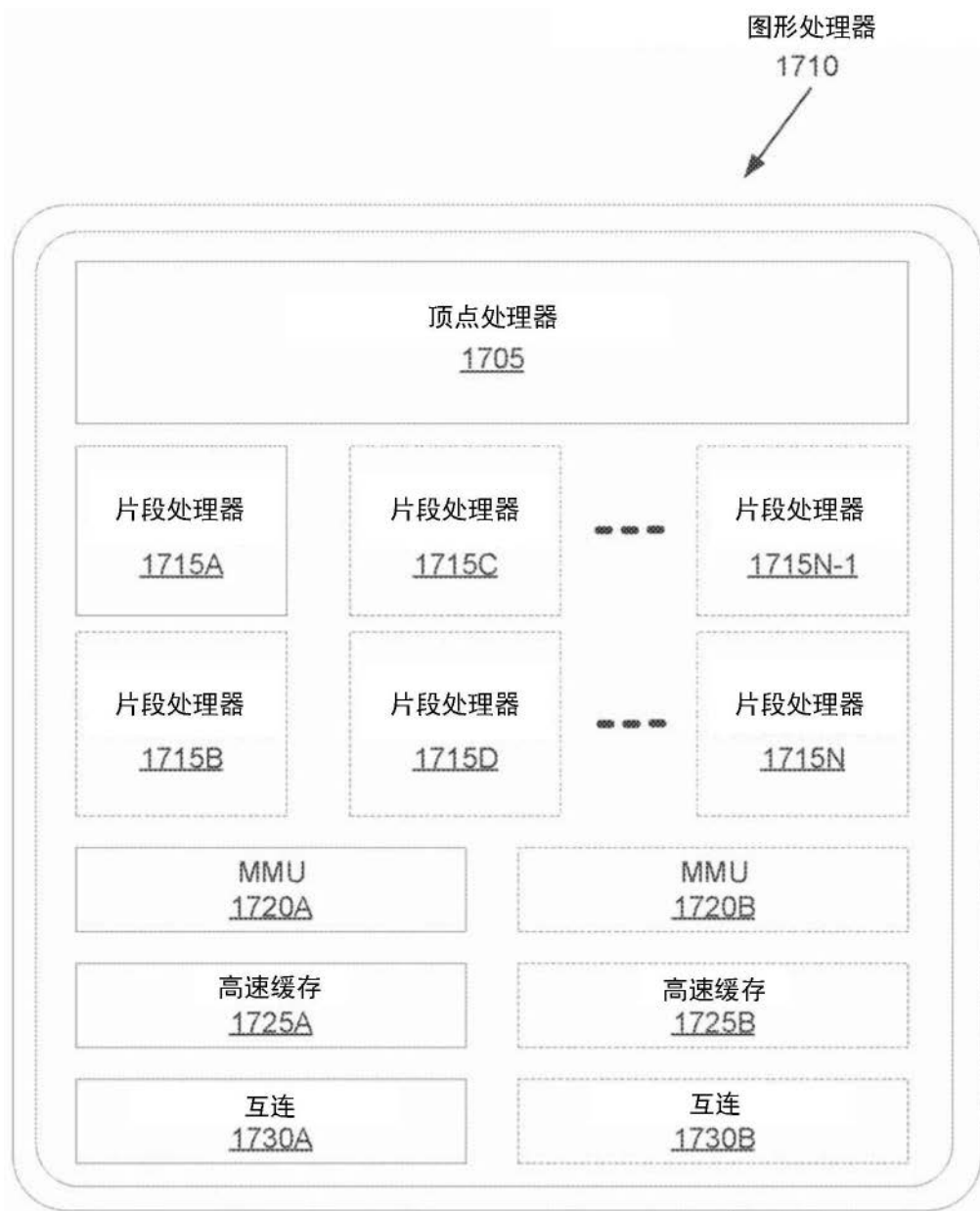


图17A



图17B

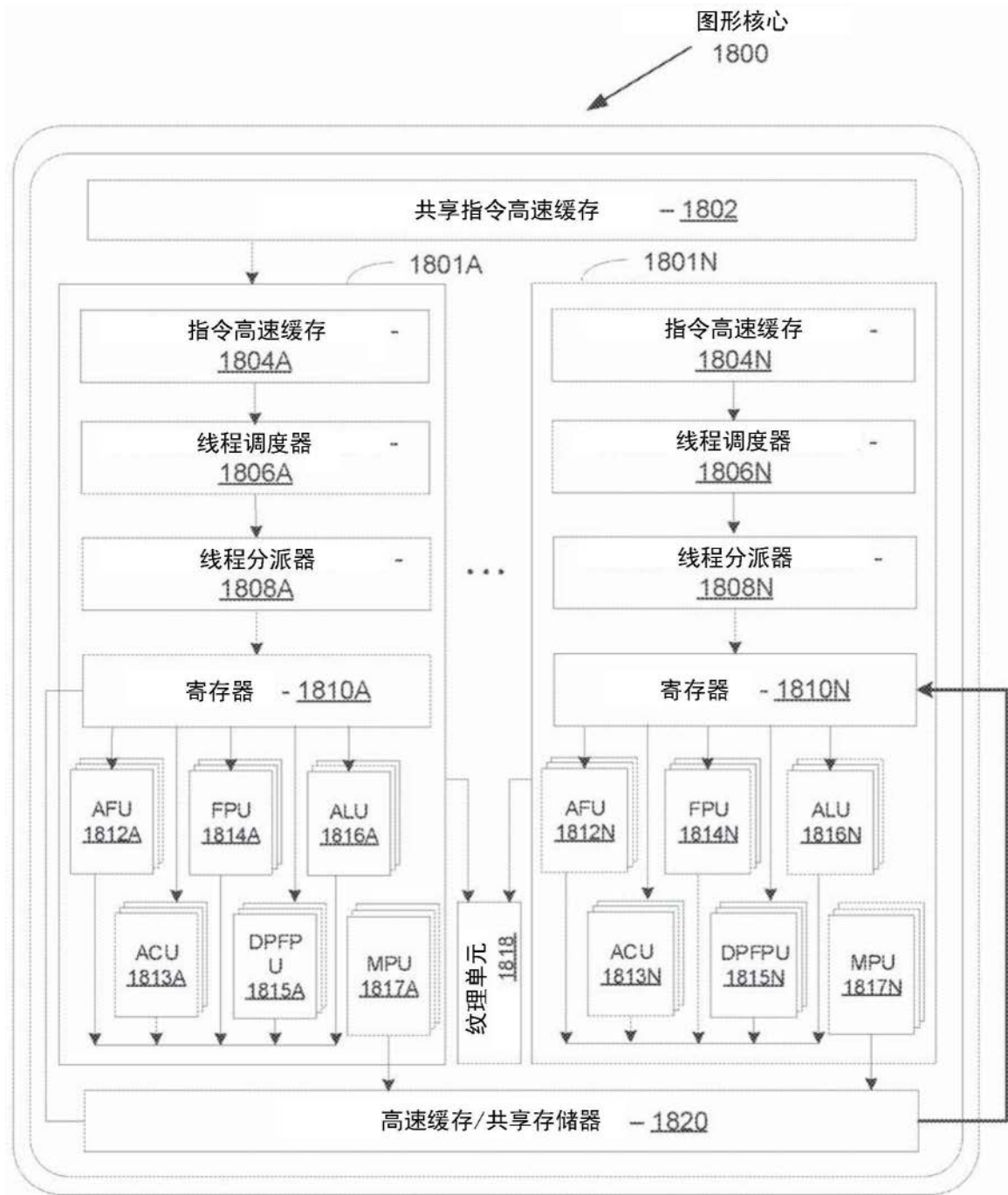


图18A



图18B

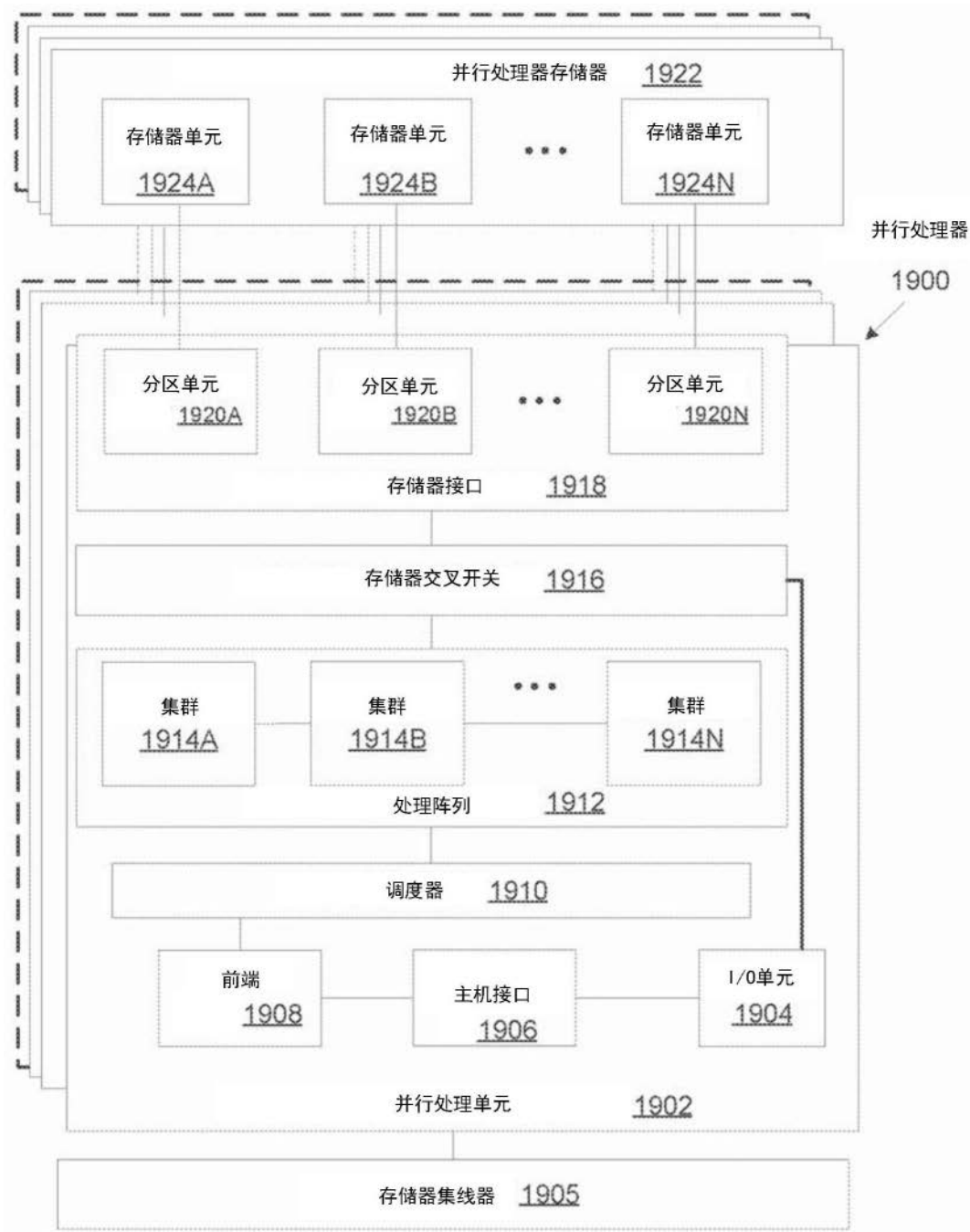


图19A

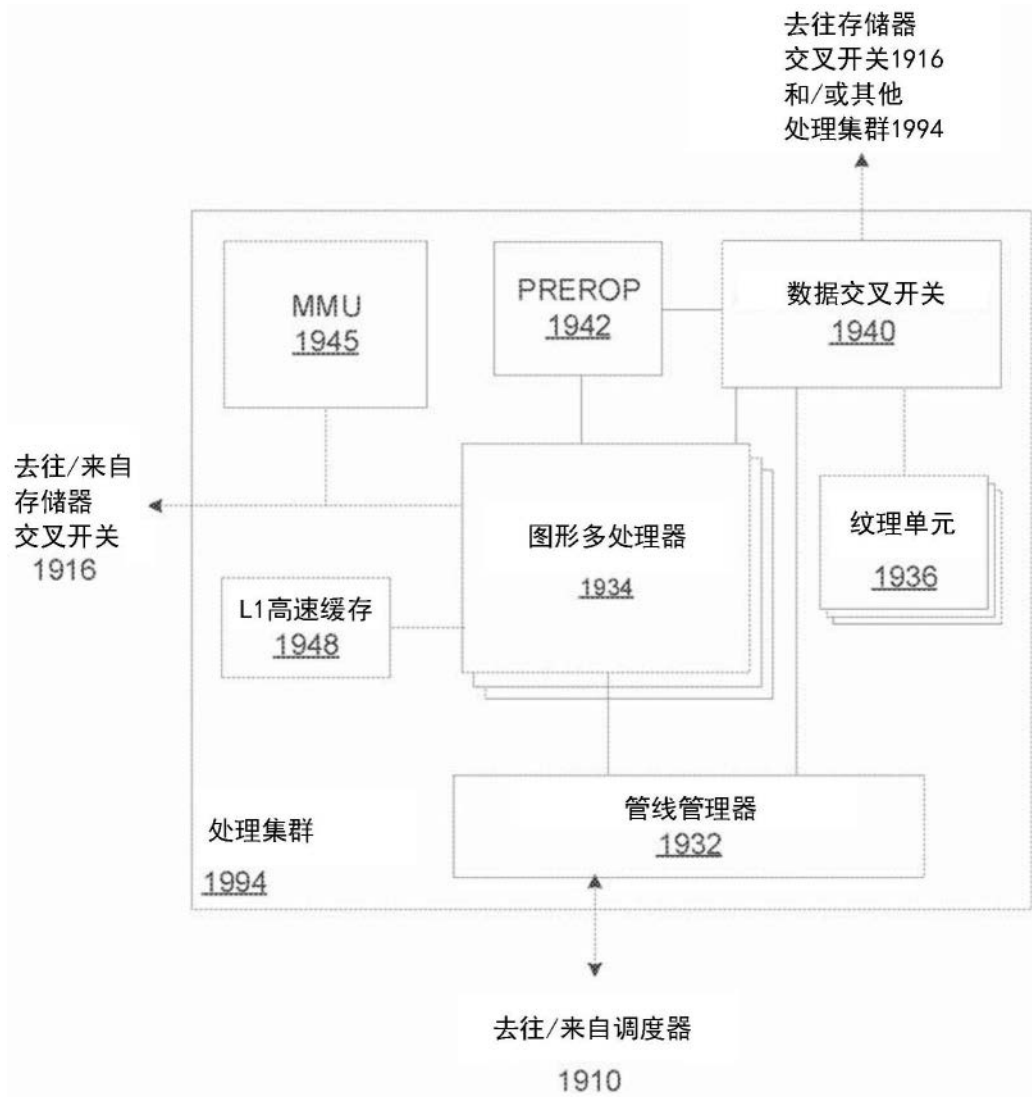


图19B

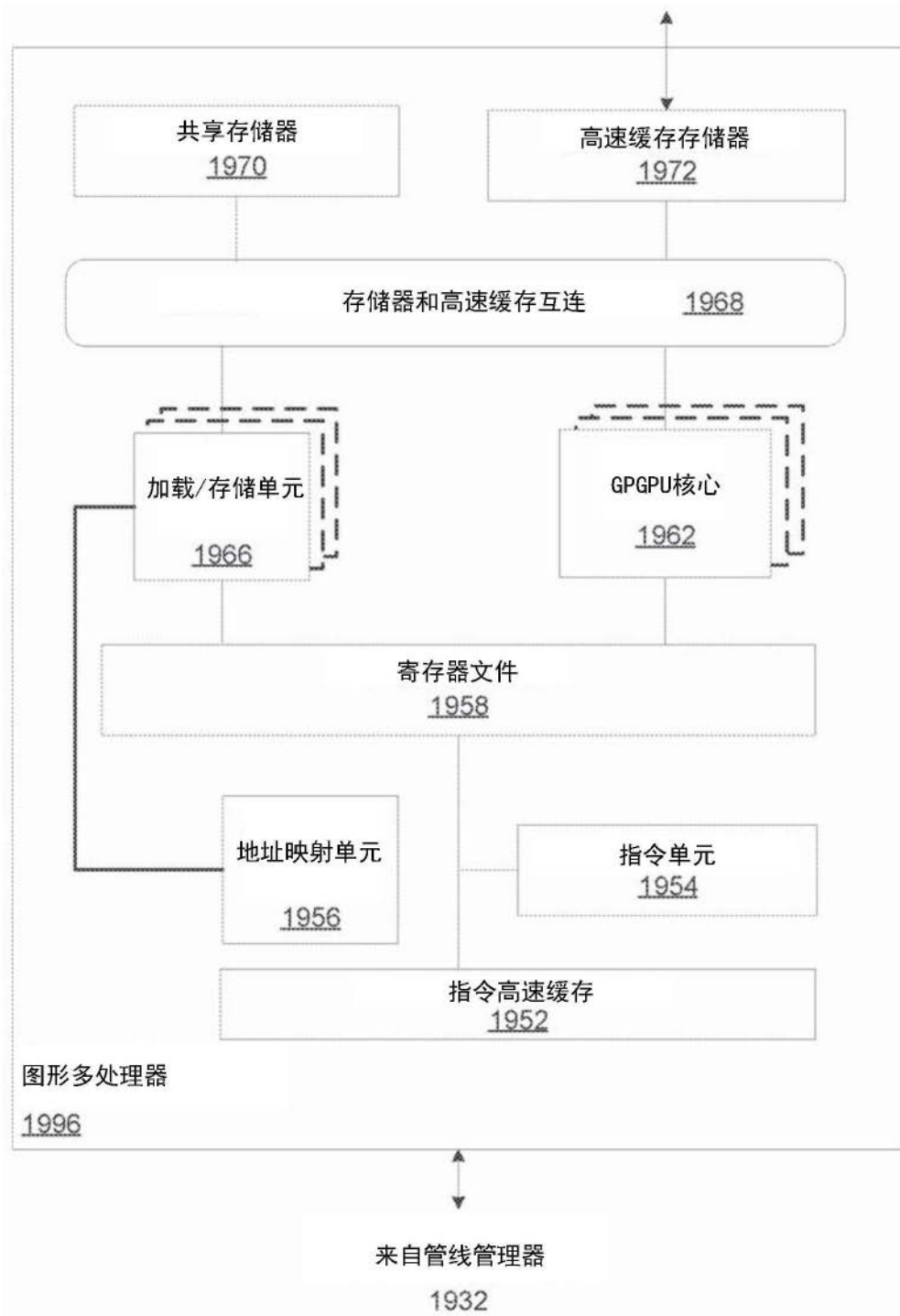


图19C

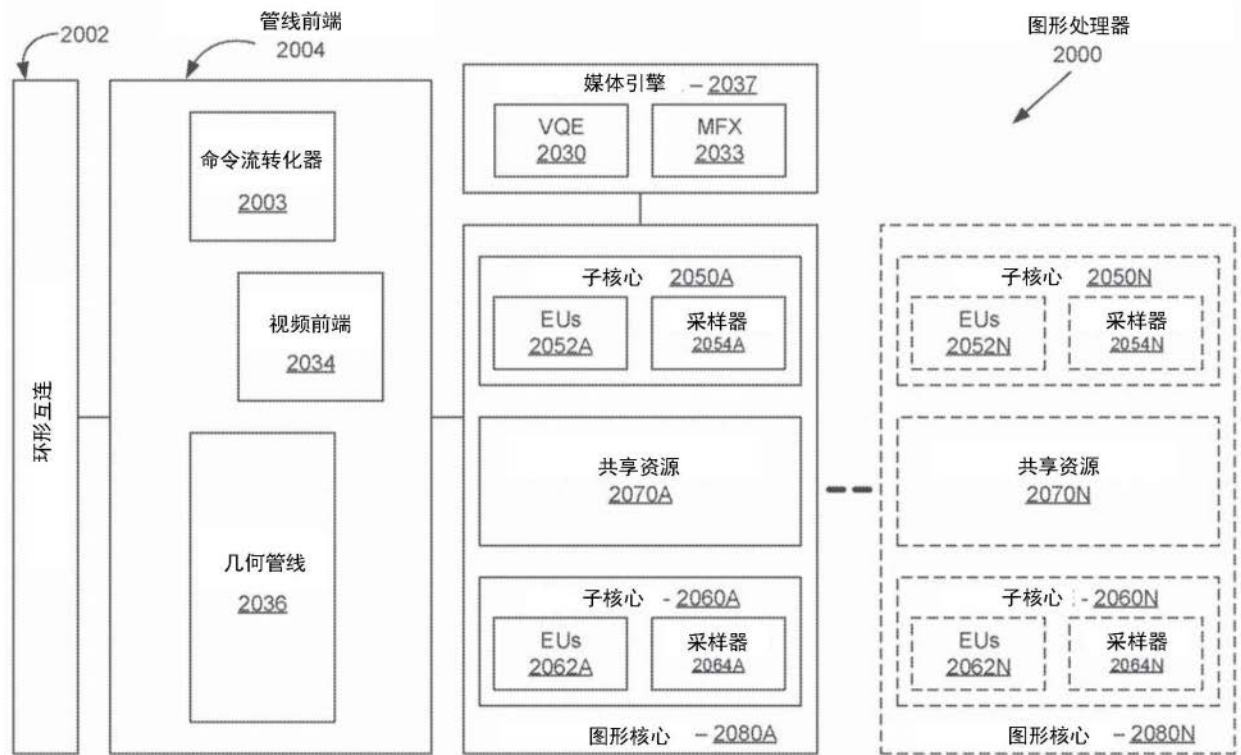


图20

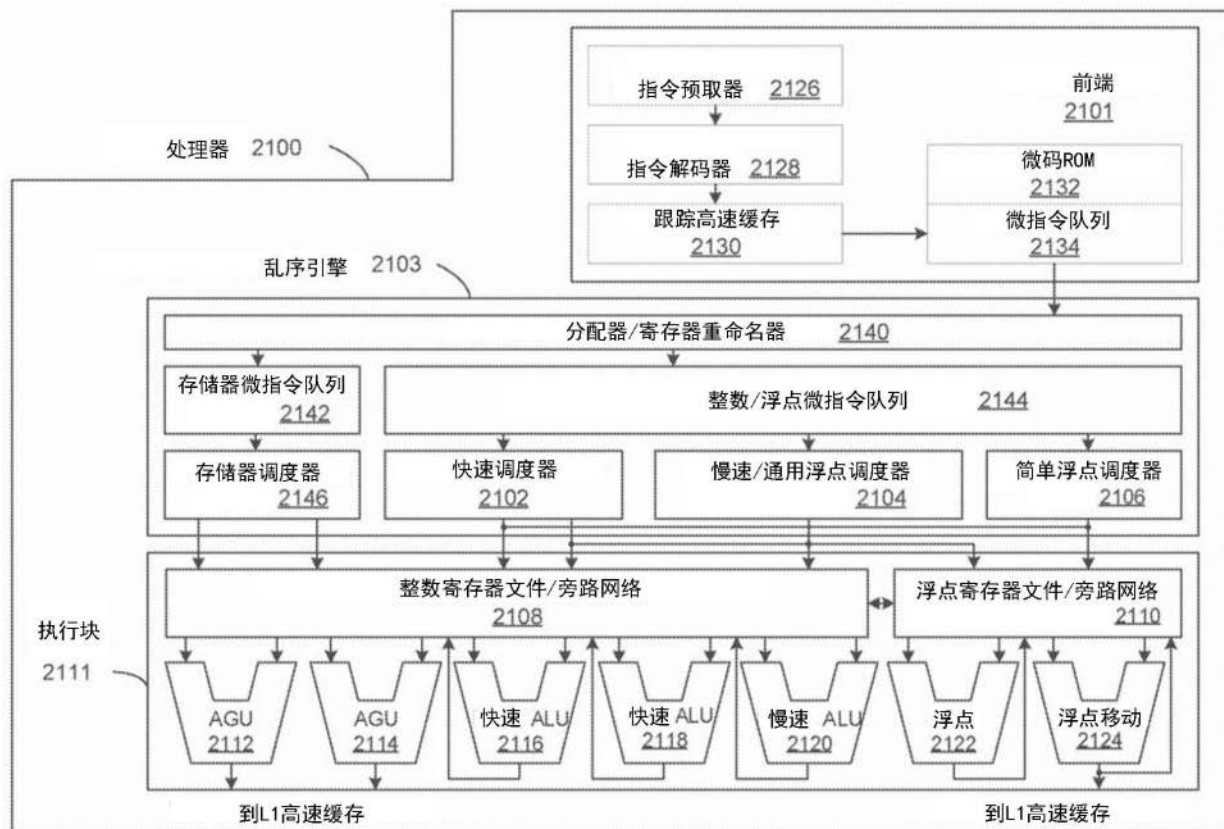


图21

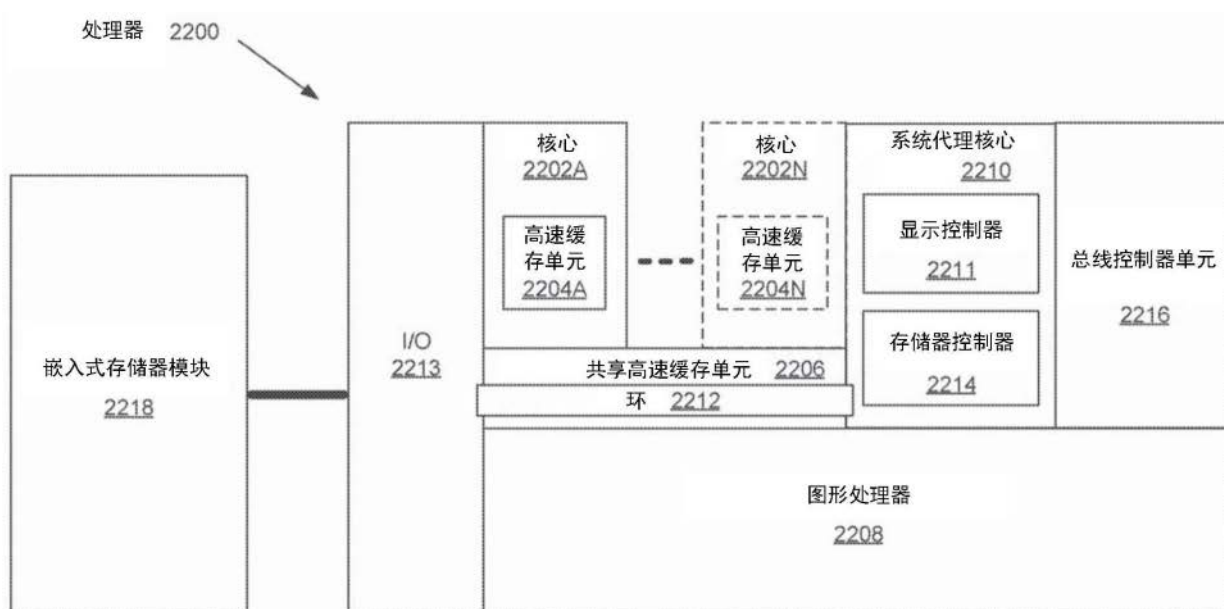


图22



图23

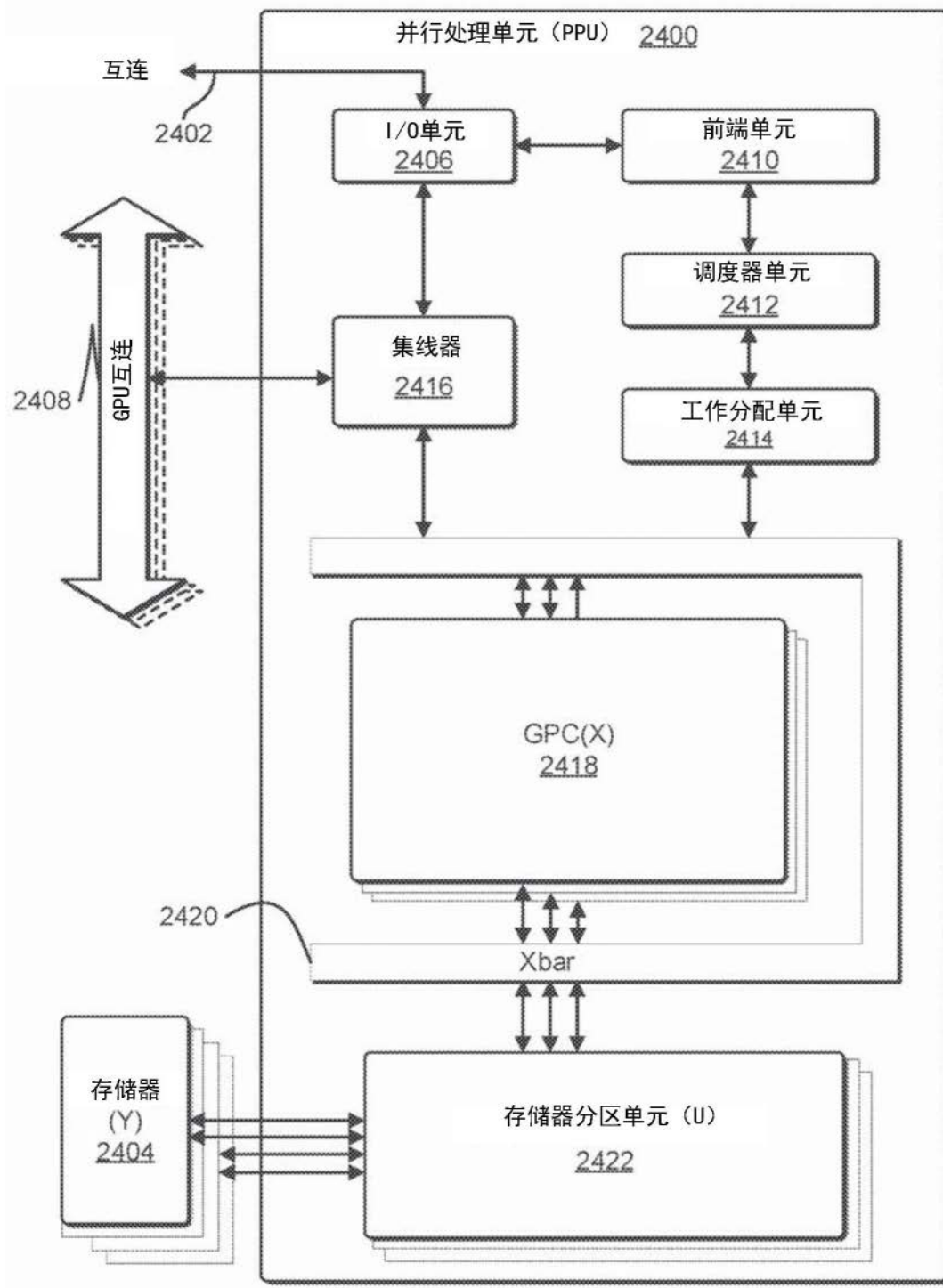


图24

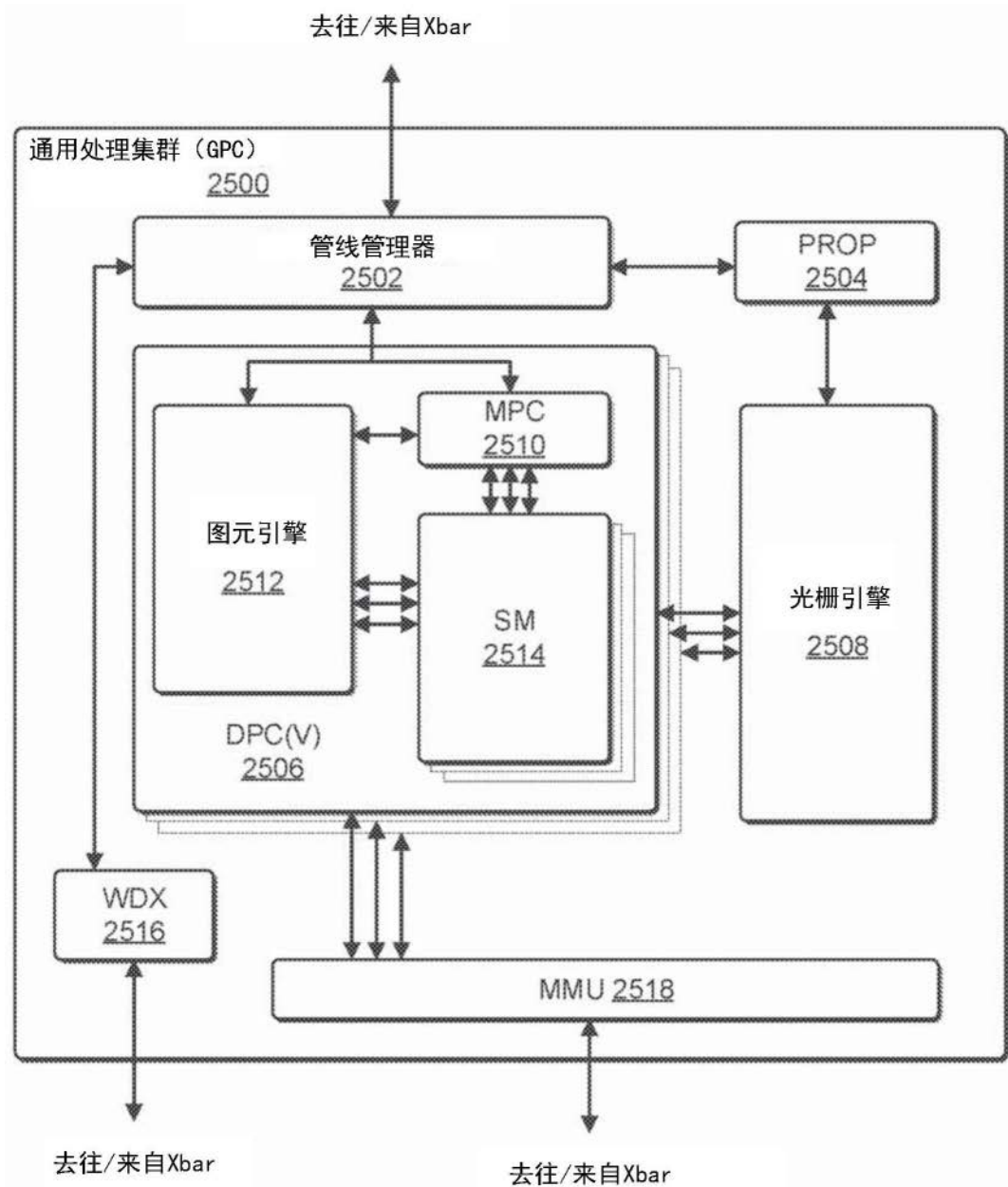


图25

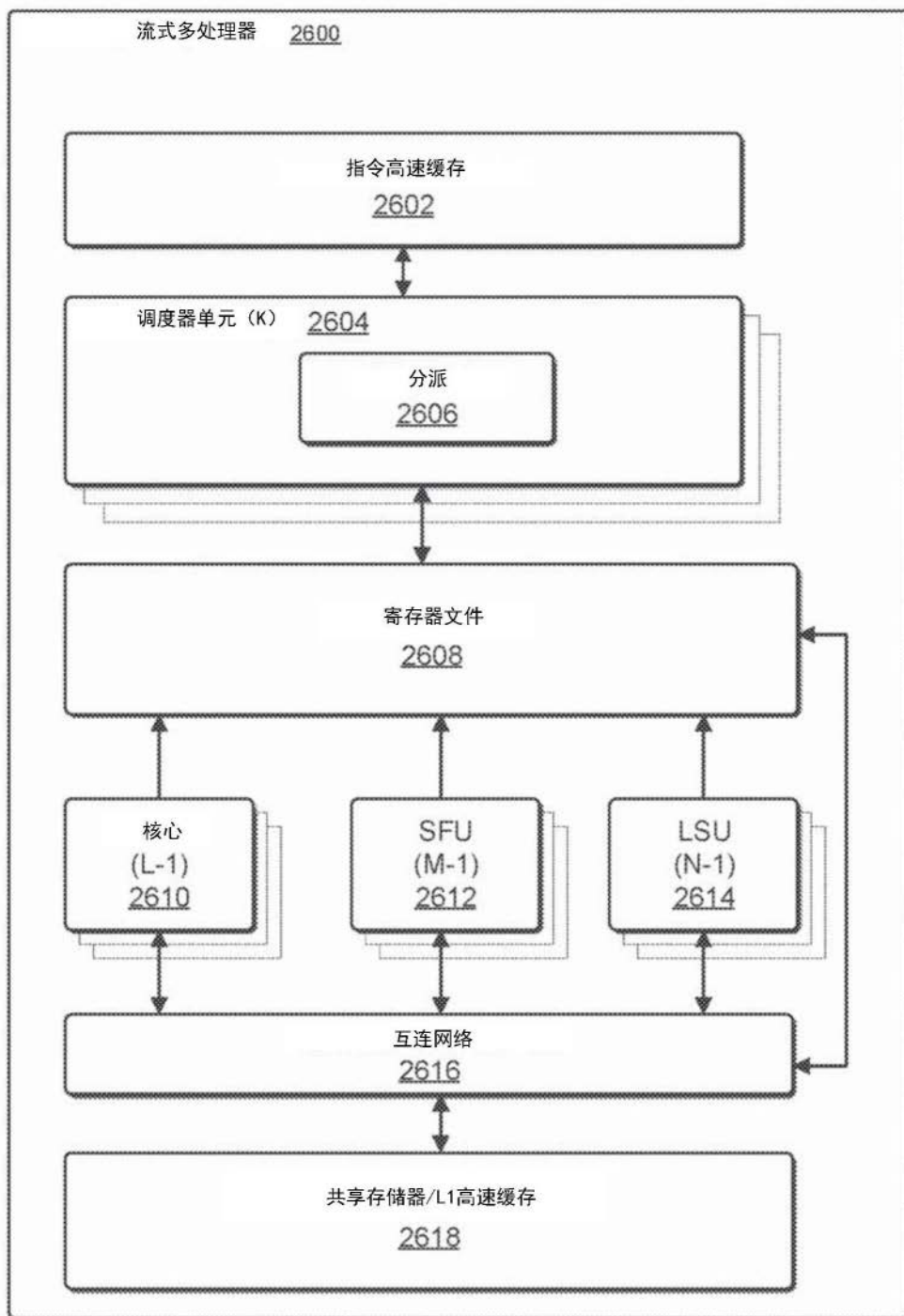


图26

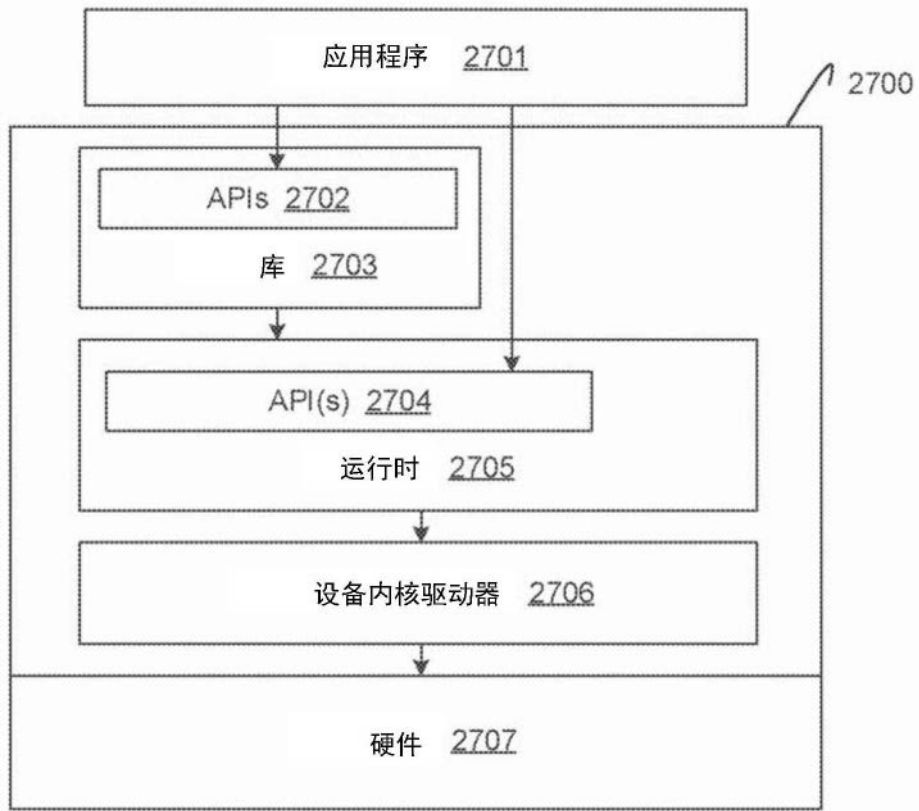


图27

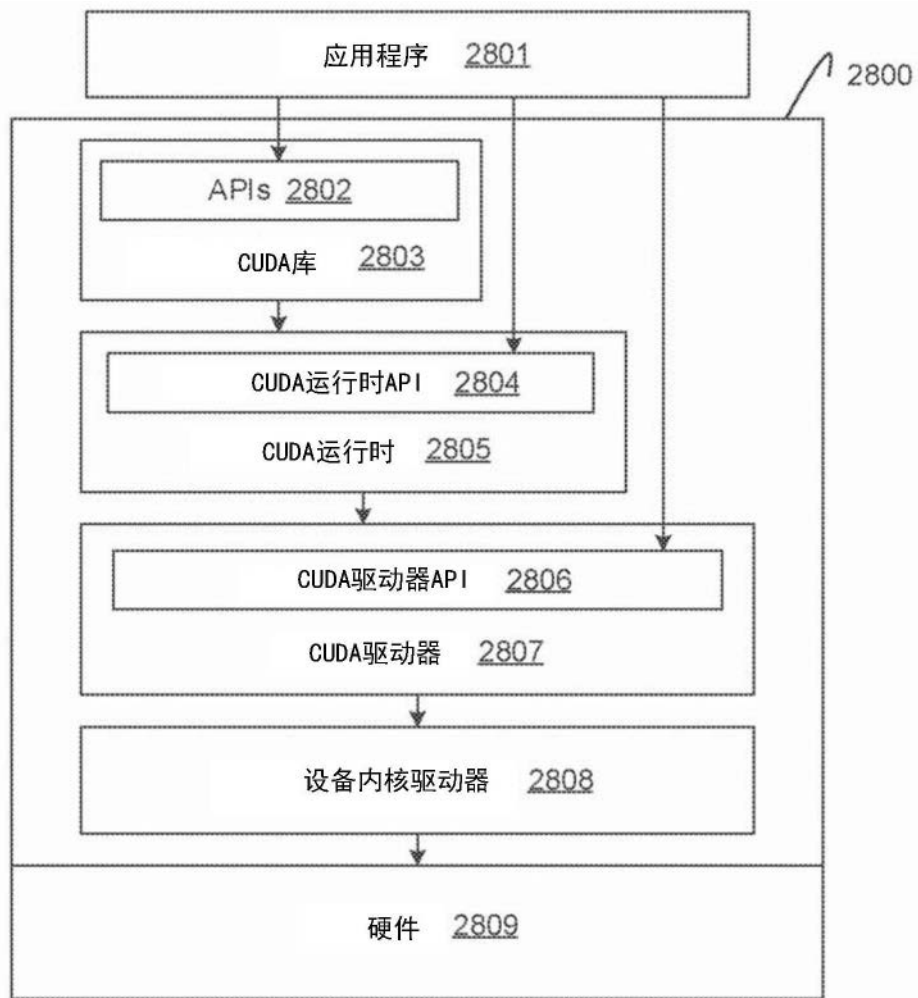


图28

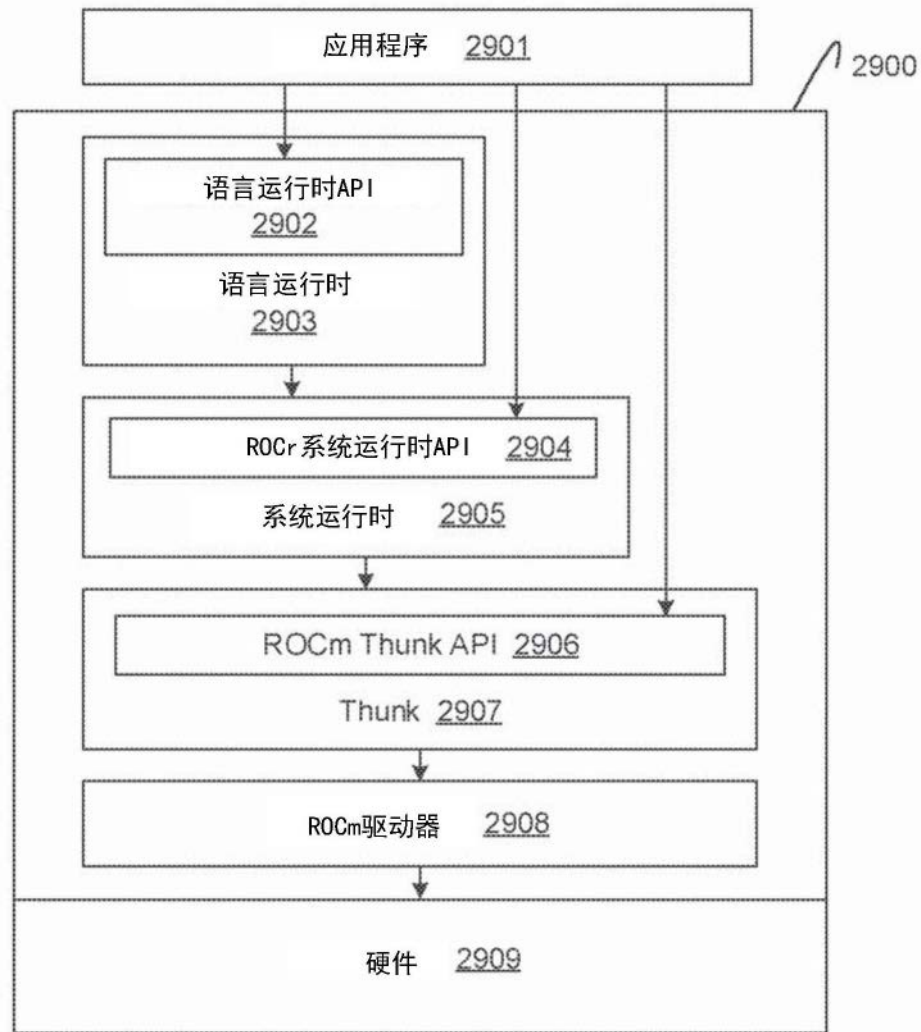


图29

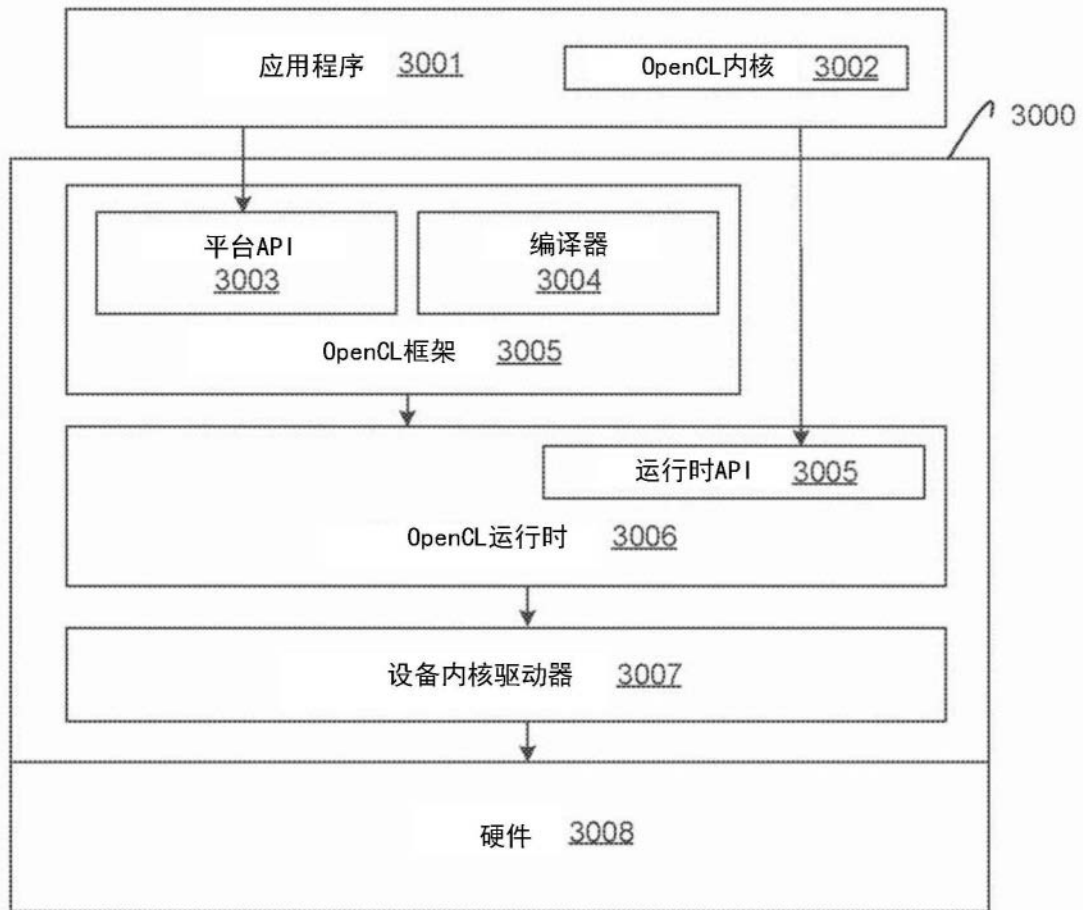


图30

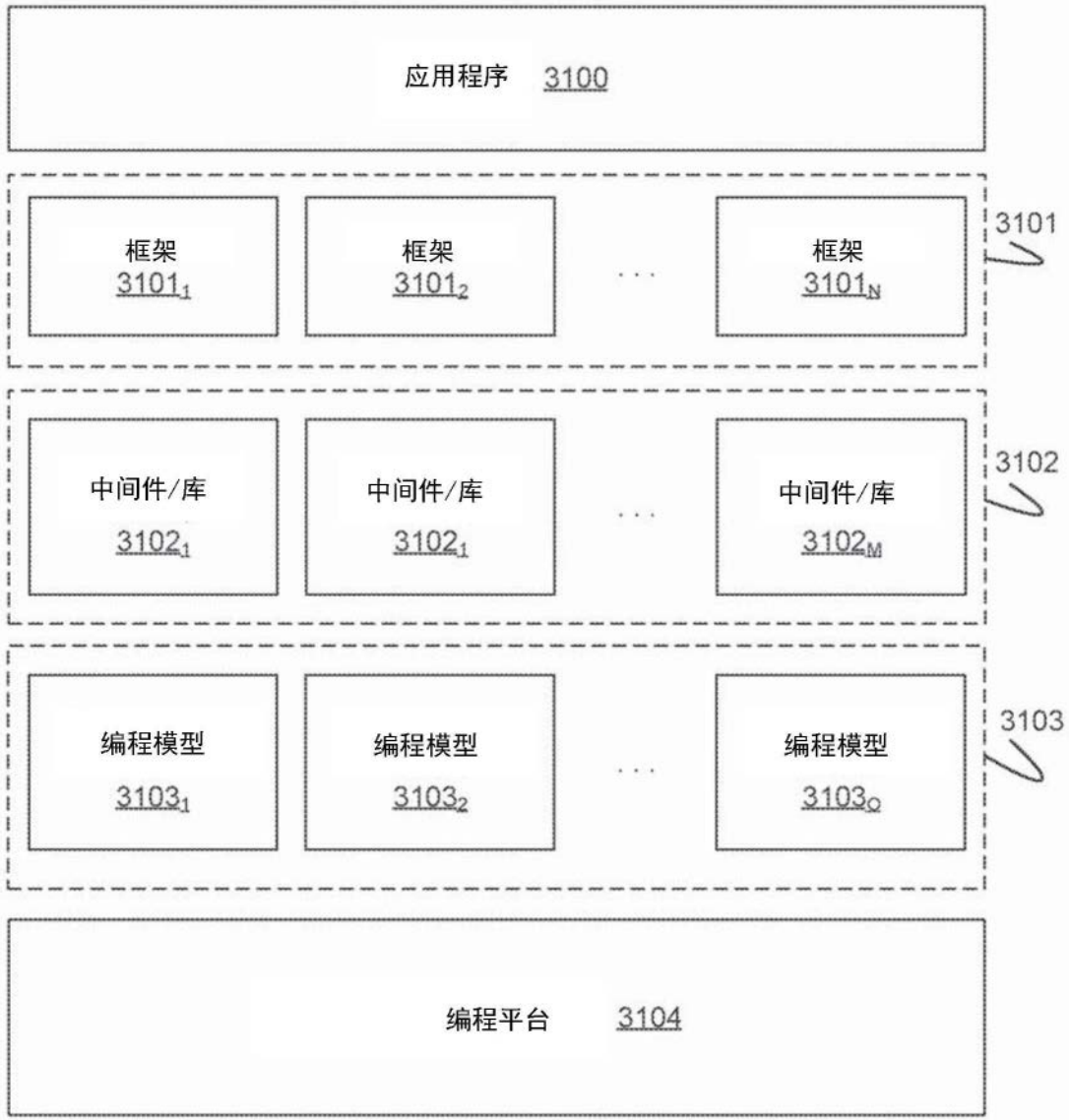


图31



图32

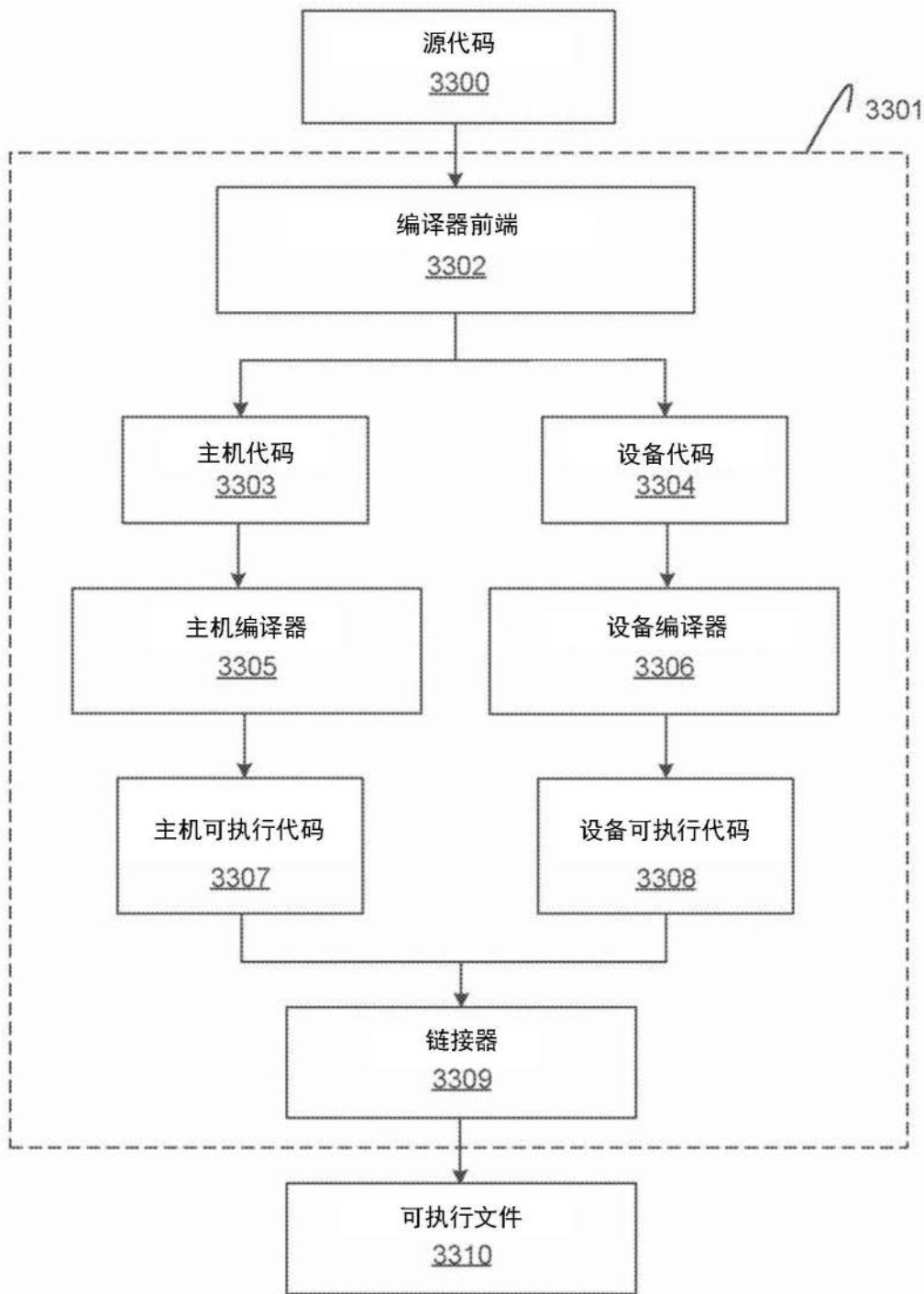


图33

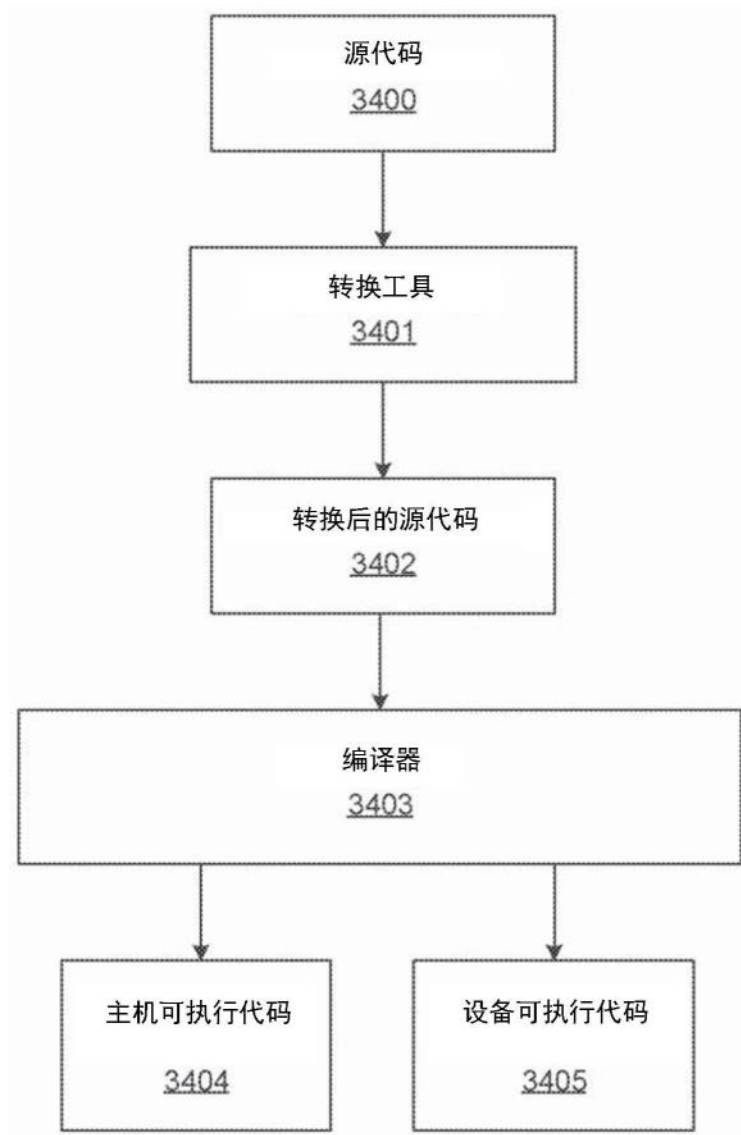


图34

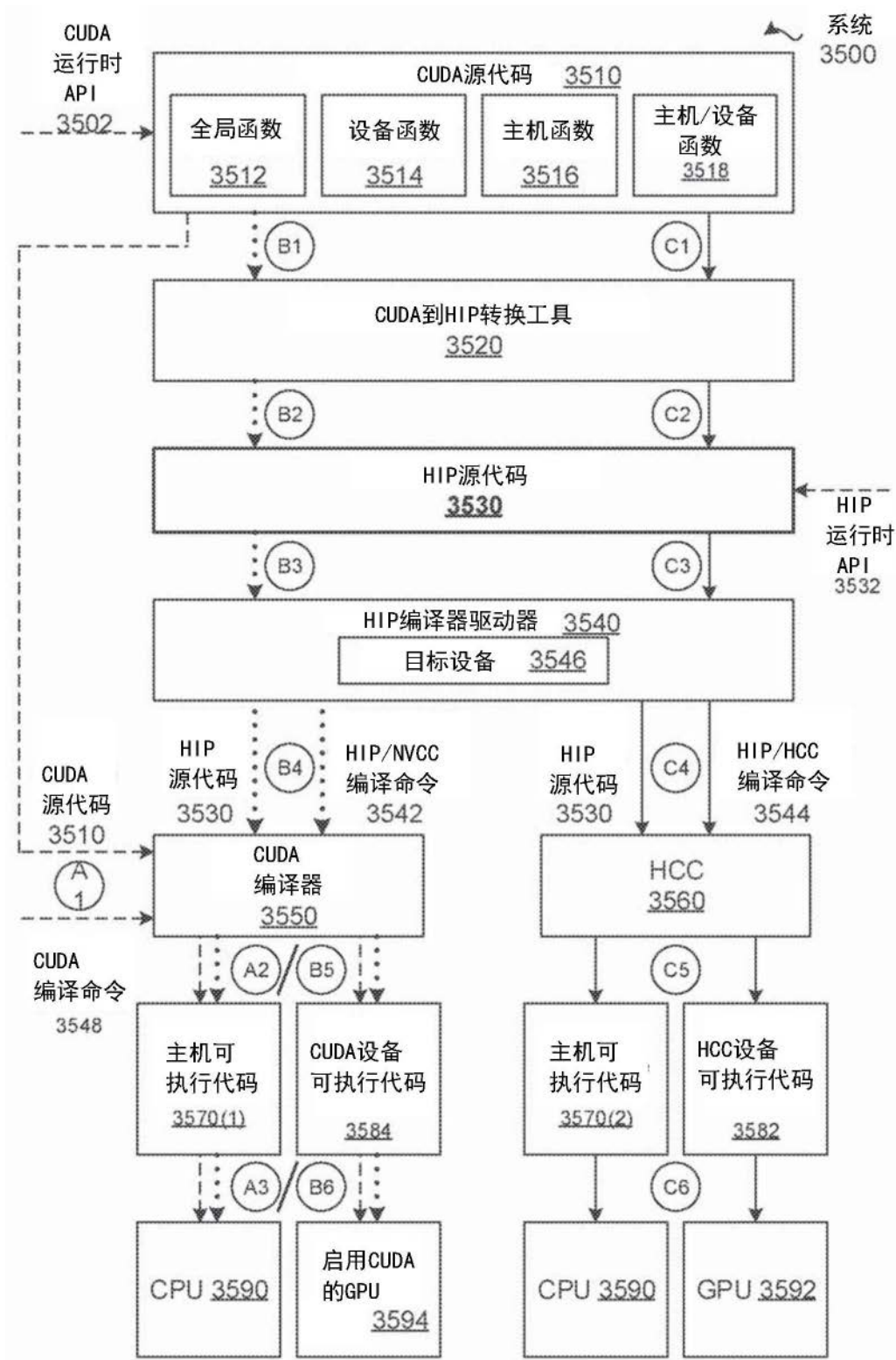


图35A

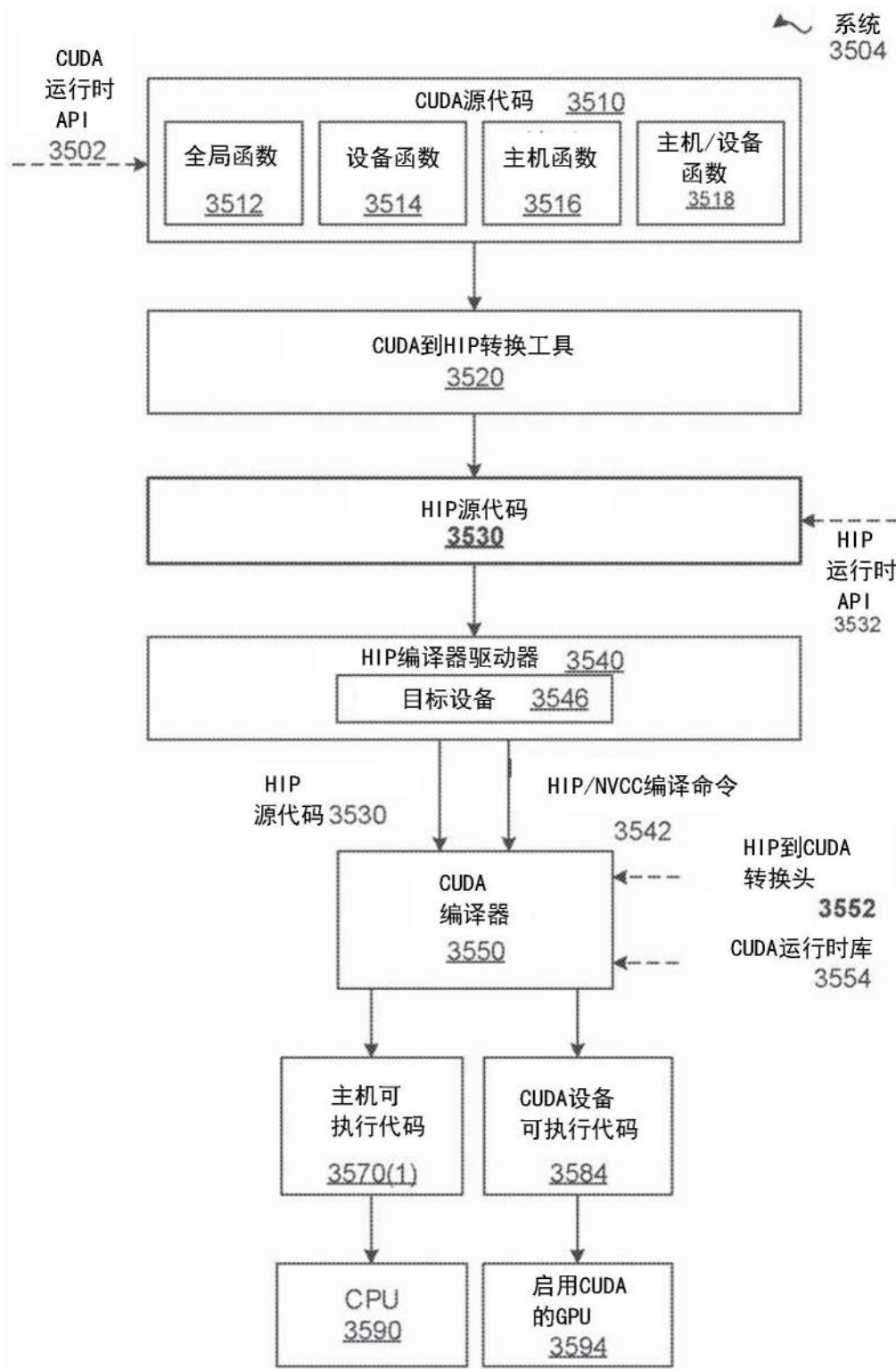


图35B

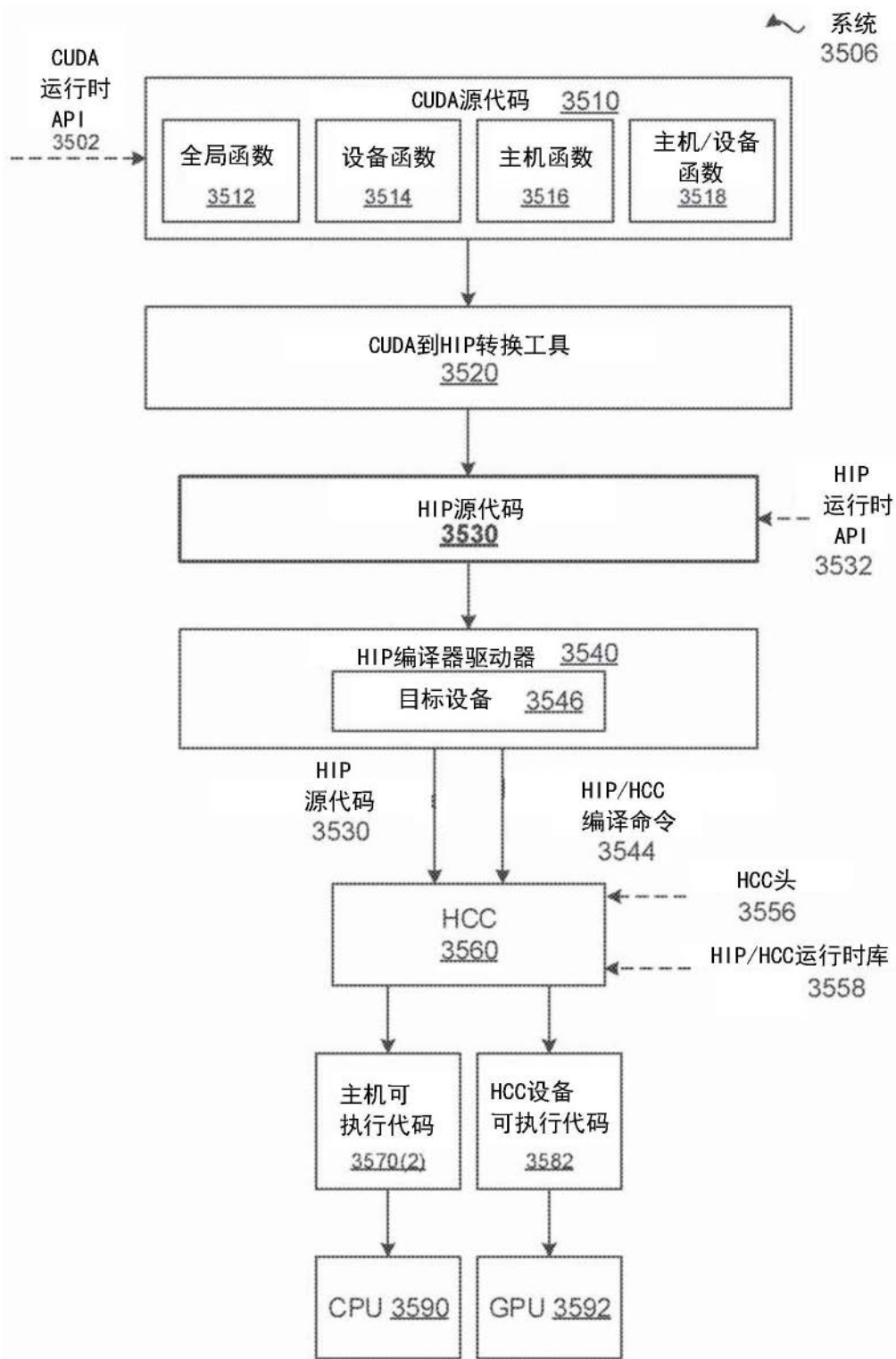


图35C

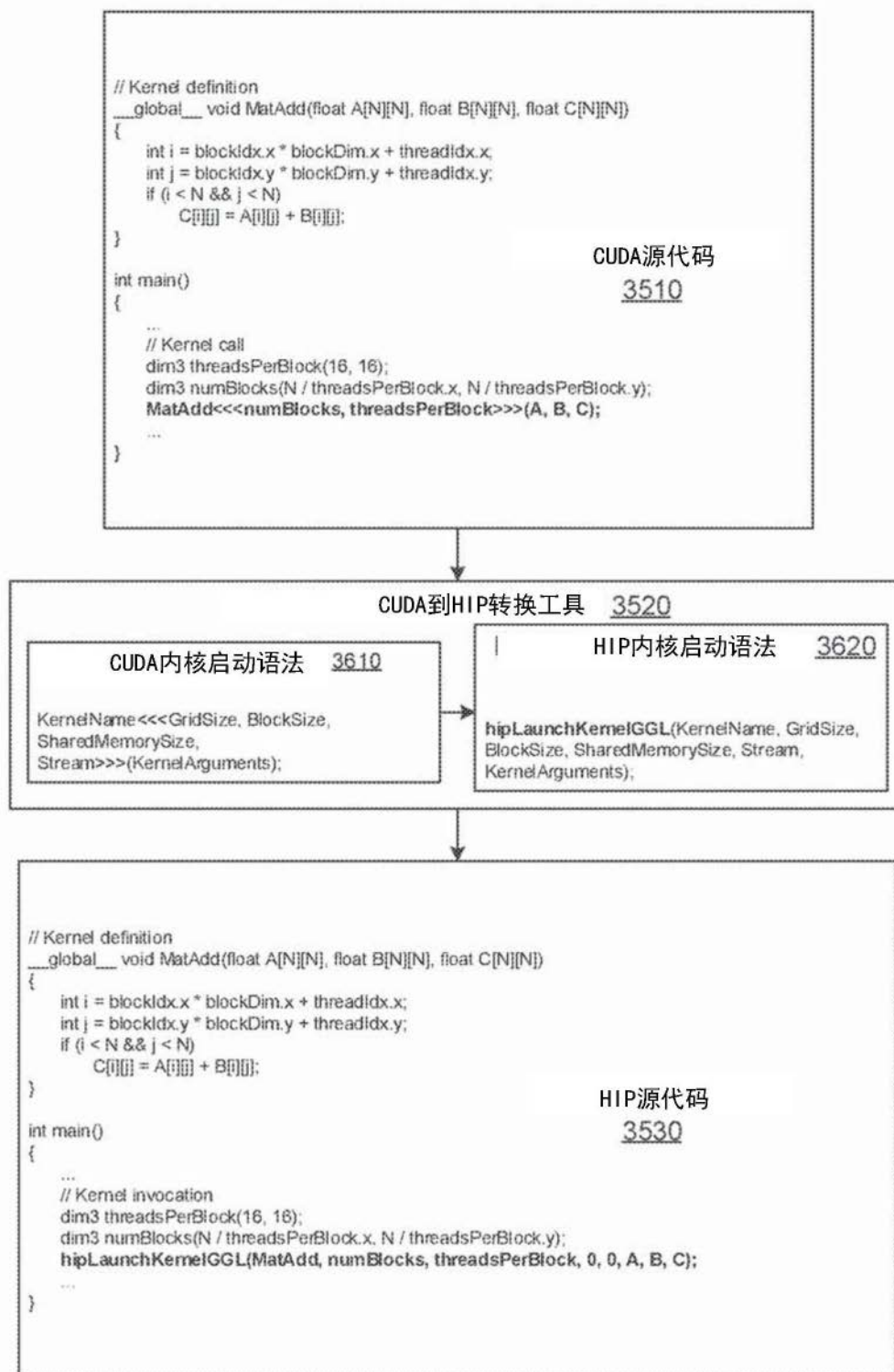


图36

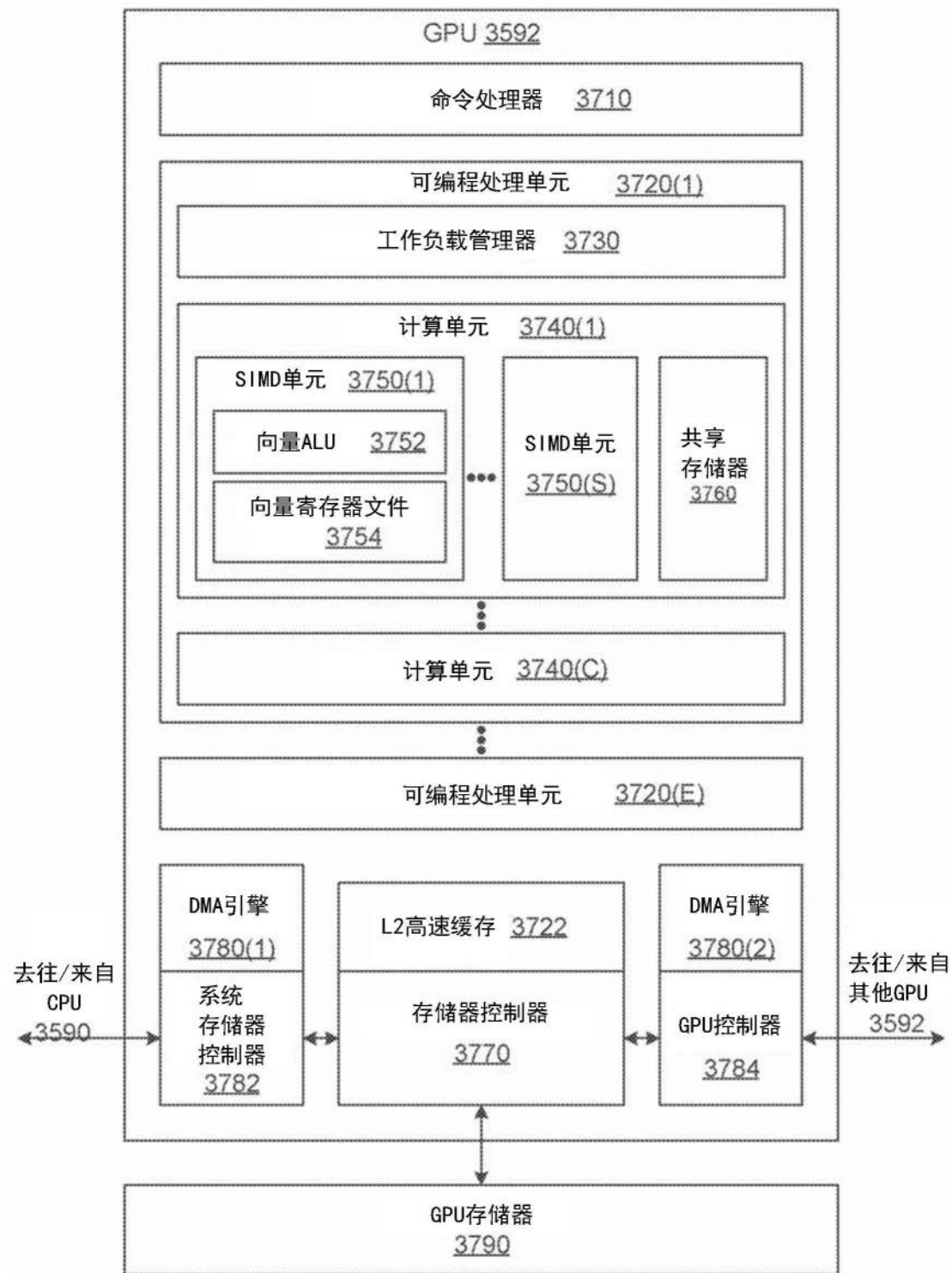


图37

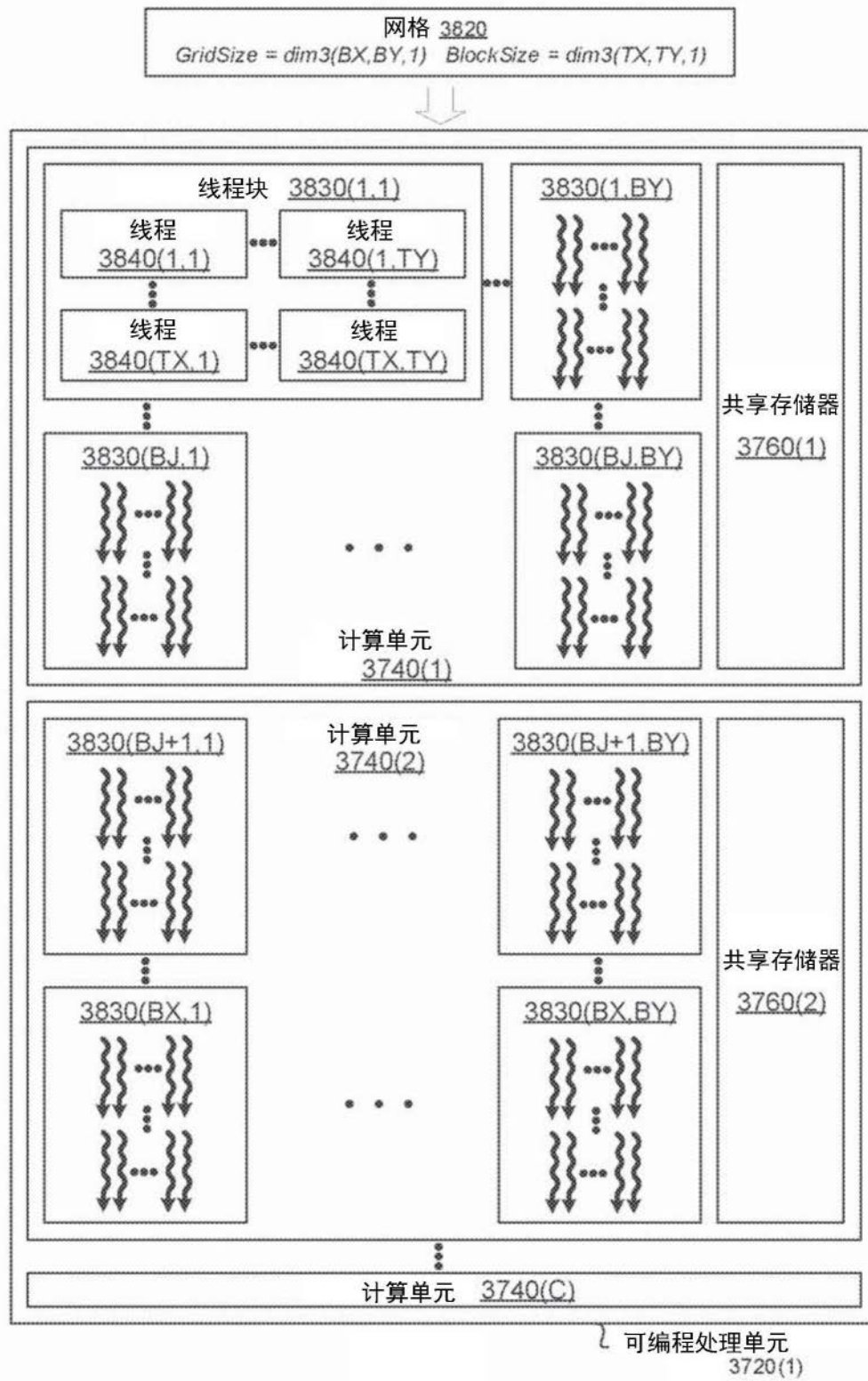


图38