



(12) 发明专利

(10) 授权公告号 CN 110796588 B

(45) 授权公告日 2024. 08. 23

(21) 申请号 201910451751.8

(22) 申请日 2019.05.28

(65) 同一申请的已公布的文献号
申请公布号 CN 110796588 A

(43) 申请公布日 2020.02.14

(30) 优先权数据
16/053,341 2018.08.02 US

(73) 专利权人 辉达公司
地址 美国加利福尼亚州

(72) 发明人 R·达什 G·帕尔默 广田源太郎
L·姗 J·肖凯特
E·克里加里夫 S·尼韦提
M·雷 S·加德雷 O·帕兰杰佩
杨磊 R·季米特洛夫

(74) 专利代理机构 北京市磐华律师事务所
11336

专利代理师 高伟 刘爱平

(51) Int.Cl.
G06T 1/20 (2006.01)
G06T 1/60 (2006.01)
G06F 9/48 (2006.01)
G06F 9/50 (2006.01)

(56) 对比文件
US 2002062435 A1,2002.05.23
US 2010076941 A1,2010.03.25
US 2017374147 A1,2017.12.28

审查员 肖美凤

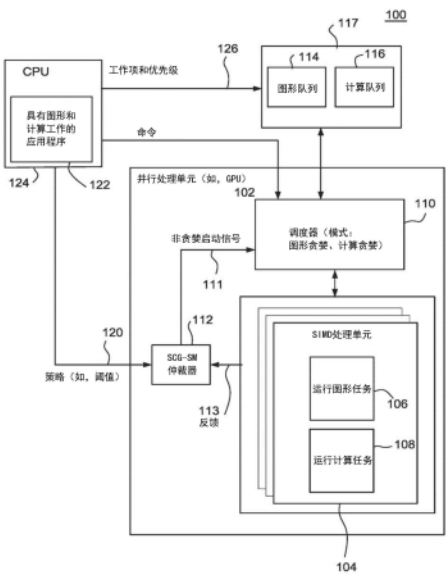
权利要求书3页 说明书30页 附图15页

(54) 发明名称

同时计算和图形调度

(57) 摘要

本发明提供了一种同时计算和图形调度。在一些示例中,描述了一种并行处理单元(例如,GPU),包括硬件调度器和硬件仲裁器,其启动用于在SIMD/SIMT处理单元上同时执行的图形工作和计算工作。并行处理单元的每个处理单元(例如,流式多处理器)在各个时间以图形贪婪模式或计算贪婪模式操作。硬件仲裁器响应于将至少一个所监控的性能或利用度量与用户配置的阈值进行比较的结果,可以选择性地:当处理单元在计算贪婪模式下操作时使处理单元运行来自计算队列的一个或多个计算工作项,以及当处理单元在图形贪婪模式下操作使处理单元运行来自图形队列的一个或多个图形工作项。还描述了相关联的方法和系统。



1. 一种处理系统,包括:

单指令多数数据 (SIMD) 处理器或单指令多线程 (SMT) 处理器,其执行并行指令流,并且被配置成在相应的时间以图形贪婪模式或者计算贪婪模式操作;以及

连接到所述处理器的硬件调度器,所述硬件调度器调度所述处理器以通过在以下两者之间选择来并行地同时执行至少一个图形线程束和至少一个计算线程束:(a) 在以将图形线程束从图形管线重复地调度到所述处理器的所述图形贪婪模式进行操作时将至少一个计算线程束调度到所述处理器,以及 (b) 在以将计算线程束从计算管线重复地调度到所述处理器的所述计算贪婪模式进行操作时将至少一个图形线程束调度到所述处理器;以及

硬件仲裁器,其被配置成响应于在所述调度期间检测到的与所述处理器相关的资源的利用不足,确定所述处理器的当前操作模式并向所述硬件调度器发出信号,以在所述当前操作模式为所述图形贪婪模式时执行调度至少一个计算线程束,或在所述当前操作模式为所述计算贪婪模式时执行调度至少一个图形线程束。

2. 一种并行处理单元,包括:

多个处理单元,每个处理单元配置成在相应的时间以图形贪婪模式或计算贪婪模式操作,以及同时运行来自图形队列的图形工作项和来自计算队列的计算工作项;

硬件调度器,其被配置成:当所述多个处理单元中的特定处理单元配置成在所述图形贪婪模式下操作时,连续地从所述图形队列中选择图形工作项以在所述特定处理单元上运行,和当所述特定处理单元配置成在所述计算贪婪模式下操作时,连续地从所述计算队列中选择计算工作项以在所述特定处理单元上运行;以及

硬件仲裁器,其被配置成响应于将至少一个所监视的性能或利用度量与用户配置的阈值进行比较的结果:当所述特定处理单元配置成在所述图形贪婪模式下操作时使所述特定处理单元运行来自所述计算队列的一个或更多个计算工作项,以及当所述特定处理单元配置成在所述计算贪婪模式下操作时使所述特定处理单元运行来自所述图形队列的一个或更多个图形工作项。

3. 如权利要求2所述的并行处理单元,其中所述多个处理单元中的每一个处理单元是单指令多数数据 (SIMD) 处理器或单指令多线程 (SMT) 处理器。

4. 如权利要求2所述的并行处理单元,其中所述硬件调度器还配置成:至少基于与所述图形工作项和所述计算工作项相关联的软件配置的优先级值来选择所述图形贪婪模式或所述计算贪婪模式。

5. 如权利要求4所述的并行处理单元,其中所述硬件仲裁器还配置成:基于软件配置的调度策略来选择使所述计算工作项或所述图形工作项运行。

6. 如权利要求5所述的并行处理单元,其中所述硬件仲裁器还配置成:进一步基于与所述图形工作项和所述计算工作项的处理和存储器资源的占用相对应的占用度量来选择使所述计算工作项或所述图形工作项运行。

7. 如权利要求6所述的并行处理单元,其中所述硬件仲裁器还配置成:基于所述特定处理单元中的度量来选择使所述计算工作项或所述图形工作项运行。

8. 如权利要求6所述的并行处理单元,其中硬件仲裁器还配置成:基于所述特定处理单元和其他处理单元中的度量来选择使所述计算工作项或所述图形工作项运行。

9. 如权利要求6所述的并行处理单元,其中所述硬件仲裁器还配置成:进一步基于与来

自所述处理单元的图形工作项的输出相关联的输出停止度量以及基于与对所述处理单元的所述图形工作项和所述计算工作项的输入相关联的输入饥饿度量,来选择使所述计算工作项或所述图形工作项运行。

10. 如权利要求9所述的并行处理单元,其中所述硬件仲裁器还配置成:进一步基于所述占用度量的时间平均值、与所述图形工作项相关联的所述输出停止度量、以及与所述图形工作项和所述计算工作项的输入相关联的所述输入饥饿度量,来选择使所述计算工作项或所述图形工作项运行。

11. 如权利要求10所述的并行处理单元,其中所述占用度量包括寄存器文件的占用度量、线程束资源的占用度量、共享存储器的占用度量以及ISBE存储器的占用度量中的一个或多个,其中与所述图形工作项相关联的所述输入饥饿度量包括顶点关联队列和像素关联队列中的至少一个,其中与所述计算工作项相关联的所述输入饥饿度量包括与所述计算队列相关联的饥饿度量,以及其中与所述图形工作项相关联的所述输出停止度量包括用于所述顶点关联队列和所述像素关联队列中的至少一个的输出停止度量。

12. 如权利要求9所述的并行处理单元,其中所述输出停止度量包括来自处理图形工作项的一个或多个固定功能单元的背压的影响。

13. 如权利要求4所述的并行处理单元,其中所述硬件调度器或所述硬件仲裁器还配置成:基于在软件指定的策略中指定的相应涓流参数来确定要从所述图形队列或所述计算队列中选择的工作项的数量。

14. 如权利要求2所述的并行处理单元,其中所述硬件调度器进一步配置为:

响应于确定启动一组图形工作项到所述特定处理单元,启动已经分配给所述硬件调度器的一个或多个图形工作项;以及

响应于确定启动一组计算工作项:

保留与所述特定处理单元相关联的资源,以用于特定数量的计算工作项;以及
从所述计算队列请求所述特定数量的计算工作项。

15. 如权利要求14所述的并行处理单元,其中所述硬件调度器还配置成:响应于确定启动所述一组计算工作项:

启动响应于所述特定处理单元上的请求而接收的计算工作项,或

响应于接收对所述请求的否定确认而不启动计算工作项。

16. 如权利要求15所述的并行处理单元,其中所述硬件调度器还配置成:在启动所述图形工作项或启动所述计算工作项的情况下,将优先级信息传递到所述特定处理单元。

17. 如权利要求2所述的并行处理单元,其中所述硬件调度器还配置成:自动改变活跃的调度策略,以及其中所述改变包括改变所述用户配置的阈值。

18. 如权利要求17所述的并行处理单元,其中所述自动改变适应于工作负载。

19. 如权利要求17所述的并行处理单元,其中所述自动改变包括:在多个预先配置的调度配置文件之间异步切换以改变所述活跃的调度策略,每个预先配置的调度配置文件包括各自不同的调度策略集。

20. 如权利要求2所述的并行处理单元,其中所述硬件调度器和/或所述硬件仲裁器接收来自所述多个处理单元的度量。

21. 一种用于在具有多个处理单元的并行处理器上同时执行图形工作项和计算工作项

的方法,包括:

接收来自图形管线的图形工作项和来自计算管线的计算工作项;以及

调度所述图形工作项的第一组和所述计算工作项的第二组,以在所述多个处理单元的所选单指令多数据(SIMD)或单指令多线程(SIMT)处理单元上同时执行,

其中所述调度还包括:进一步基于与来自所述处理单元的图形工作项的输出相关联的输出停止度量以及基于与对所述处理单元的所述图形工作项和所述计算工作项的输入相关联的输入饥饿度量,来选择所述第一组和所述第二组。

22. 如权利要求21所述的方法,其中所述调度包括:至少基于与所述图形工作项和所述计算工作项的相应组相关联的软件配置的优先级值来选择所述第一组和所述第二组。

23. 如权利要求22所述的方法,其中所述调度包括:进一步基于软件配置的调度策略来选择所述第一组和所述第二组。

24. 如权利要求22所述的方法,其中所述调度还包括:进一步基于与所述图形工作项和所述计算工作项的处理和存储器资源的占用相对应的占用度量来选择所述第一组和所述第二组。

25. 如权利要求21所述的方法,其中所述选择还基于与所述图形工作项和所述计算工作项的处理和存储器资源的占用相对应的占用度量的时间平均值、与所述图形工作项相关联的所述输出停止度量以及与所述图形工作项和所述计算工作项的输入相关联的所述输入饥饿度量。

26. 一种处理系统,包括:

CPU,配置成执行应用程序;

存储器,配置成具有图形队列和计算队列,其中,来自所述应用程序的图形工作项排入所述图形队列,来自所述应用程序的计算工作项排入所述计算队列;和

图形处理单元(GPU),包括:

多个处理单元,每个处理单元配置成在相应的时间以图形贪婪模式或计算贪婪模式操作,并且同时运行来自所述图形队列的图形工作项和来自所述计算队列的计算工作项;

硬件调度器,配置成:当所述多个处理单元中的特定处理单元配置成在所述图形贪婪模式下操作时,连续地从所述图形队列中选择图形工作项以在所述特定处理单元上运行,以及当所述特定处理单元配置成在所述计算贪婪模式下操作时,连续地从所述计算队列中选择计算工作项以在所述特定处理单元上运行;和

硬件仲裁器,配置成响应于将至少一个所监控的性能或利用度量与用户配置的阈值进行比较的结果,选择性地:当所述特定处理单元配置成在所述图形贪婪模式下操作时使所述特定处理单元运行来自所述计算队列的一个或更多个计算工作项,和当所述特定处理单元配置成在所述计算贪婪模式下操作时使所述特定处理单元运行来自所述图形队列的一个或更多个图形工作项。

同时计算和图形调度

[0001] 相关申请的交叉参考

[0002] 本申请涉及于2013年12月20日提交的、申请号为14/137,818、题为“用于同时执行计算和图形工作负载的系统、方法和计算机程序产品 (System, method, and computer program product for simultaneous execution of compute and graphics workloads)”的美国专利申请。

[0003] 优先权申明

[0004] 没有。

技术领域

[0005] 本公开涉及在计算机处理器上调度任务,更具体地涉及在并行处理单元(例如,图形处理单元(GPU))上调度图形任务和计算任务,以及更具体地涉及在并行处理单元(例如, GPU)的一组处理单元中的相同处理单元上调度图形任务和计算任务用于同时执行。

背景技术

[0006] 提高处理系统的性能的一种方法是使用可以并行执行许多指令流的并行处理核。在过去的几年中,中央处理单元(CPU)和图形处理单元(GPU)从这种增加的并行性中受益。例如,超标量体系结构的处理器可以将多个指令分派给不同的执行单元,从而提高平均执行速度。类似地,许多现代GPU具有大规模并行处理架构,这意味着它们包含许多并行处理器,这些处理器可以在图形图像的不同部分上并行地操作。

[0007] 过去,GPU功能仅限于一组明确定义的图形操作。当GPU变得可编程时,其发生了变化。例如,在2001年,NVidia发布了其GEForce3NV20GPU,提供可编程顶点和像素着色器。之后,Nvidia开发了**CUDA®**并行计算平台和编程模型用于图形处理单元(GPU)上的通用计算。借助**CUDA®**,除了图形任务之外,开发人员还可以通过使用GPU处理功能来执行计算任务,从而显著加速计算应用程序。

[0008] GPU通常专门用于通过并行处理与特定场景相关的大量线程来处理大数据块。例如,可以将GPU最初分配给图形场景,其中所有GPU线程配置成并行处理图形数据。当GPU完成渲染图形时,CPU可以将GPU切换到不同的场景,并重新使用所有GPU资源来计算(图形除外)数据处理。

[0009] 如本文所使用的,GPU图形场景包括与在GPU上执行指令以处理图形数据(例如渲染3D模型数据以生成2D图像数据,处理纹理,生成软阴影等)有关的状态。GPU计算场景包括涉及在GPU上执行指令以执行通用并行计算(例如在动画或大数据集分析中使用的物理计算)的状态。许多传统GPU可以配置成处理图形场景或计算场景,但不能同时处理两者。操作系统可以在执行期间根据需要动态地将GPU从一个场景切换到另一个场景,以便处理图形任务或计算任务。

[0010] 在许多应用程序中需要动态地改变图形任务和计算任务之间的至少一些处理能力的分配而不需要进行场景切换,并且可以潜在地提高应用程序的处理速度以及提高资源

利用率。例如,在处理图形的过程中,GPU可以在继续执行其他图形处理线程或线程束(warp)的同时完成一些图形处理线程或线程束。在GPU可以执行计算线程或线程束之前需要将场景从图形场景切换到计算场景,期间需要GPU在将任何处理资源分配给计算功能之前完成所有图形处理,从而增加延迟并降低处理器利用率。

[0011] 一些其他GPU将一些处理单元分配用于图形处理,而将其他处理单元分配用于计算处理,并且允许在不切换场景的情况下改变图形和计算之间的处理单元的分配比例。例如参见USP 9,626,216,其配置GPU以根据通用处理场景执行第一工作负载和第二工作负载,并且从执行第一工作负载转换到执行第二工作负载而不执行场景切换。这些GPU允许同时地一些处理单元上运行图形任务,而在其他处理单元运行计算任务。然而,在将该处理单元从图形重新分配给计算之前,至少一些诸如过去的商业实现可能需要从GPU处理单元中耗尽正在进行的工作,这可能导致低效率和延迟。此外,在处理单元级别分配资源的需求通常导致进一步低效率。

[0012] 因此,需要具有调度技术的系统,其可以通过允许在图形功能和计算功能之间更灵活地分配处理器来改善响应时间和/或处理资源利用,但是不会过度损害GPU快速执行图形处理的能力。

发明内容

[0013] 示例性实施例纠正了用于在图形任务和计算任务之间共享GPU的处理资源的上述技术的一些缺陷。

[0014] 本文公开了用于在并行处理单元中调度流式多处理器以同时并行执行至少一个图形线程束和至少一个计算线程束的方法、计算机可读介质和系统。

[0015] 示例实施例提供了一种图形处理单元,包括执行并行指令流的流式多处理器和连接到流式处理器的调度器。调度器调度流式多处理器以同时并行执行至少一个图形线程束和至少一个计算线程束。

[0016] 另一示例性实施例提供了一种并行处理单元,包括多个处理单元、硬件调度器和硬件仲裁器。每个处理单元配置成在各个时间以图形贪婪模式或计算贪婪模式操作,并且同时运行来自图形队列的图形工作项和来自计算队列的计算工作项。硬件调度器配置成,当特定处理单元配置成在图形贪婪模式下操作时连续地从图形队列中选择图形工作项以在特定处理单元上运行,以及当特定处理单元配置成在计算贪婪模式下操作时连续地从计算队列中选择计算工作项以在特定处理单元上运行。硬件仲裁器配置成,响应于将至少一个被监视的性能或利用度量与用户配置的阈值进行比较的结果,选择性地:当特定处理单元配置成在图形贪婪模式下操作时使特定处理单元运行来自计算队列的一个或更多个计算工作项,和当特定处理单元配置成在计算贪婪模式下操作时使特定处理单元运行来自图形队列的一个或更多个图形工作项。

[0017] 另一示例性实施例提供了一种用于在具有多个处理单元的并行处理器上同时执行图形工作项和计算工作项的方法。该方法包括:接收来自图形管线的图形工作项和来自计算管线的计算工作项,以及调度第一组所述图形工作项和第二组所述计算工作项以同时在所述多个处理单元的所选择的单指令多数据(SIMD)或单指令多线程(SIMT)处理单元上执行。

[0018] 另一示例实施例提供了一种系统,包括:CPU,配置成执行应用程序;存储器,配置成具有图形队列和计算队列;以及图形处理单元。图形处理单元包括多个处理单元,硬件调度器和硬件仲裁器。每个处理单元配置成在各个时间以图形贪婪模式或计算贪婪模式操作,并且同时运行来自图形队列的图形工作项和来自计算队列的计算工作项。硬件调度器配置成当特定处理单元配置成在图形贪婪模式下操作时连续地从图形队列中选择图形工作项以在特定处理单元上运行,以及当特定处理单元配置成在计算贪婪模式下操作时连续地从计算队列中选择计算工作项以在特定处理单元上运行。硬件仲裁器配置成响应于将至少一个所监视的性能或利用度量与用户配置的阈值进行比较的结果,选择性地:当特定处理单元配置成在图形贪婪模式下操作时使特定处理单元运行来自计算队列的一个或更多个计算工作项,或当特定处理单元配置成在计算贪婪模式下操作时使特定处理单元运行来自图形队列的一个或更多个图形工作项。

附图说明

[0019] 图1示出了根据一些示例性实施例的一种系统,其中可以在诸如GPU的并行处理单元的多个单指令多数据(SIMD)或单指令多线程(SIMT)的处理单元之一中同时处理图形工作项和计算工作项。

[0020] 图2A示出了根据一些示例性实施例的调度过程的流程图。

[0021] 图2B示出了根据传统技术的在处理单元上的工作的示例性线程束级视图。

[0022] 图2C示出了根据一些示例性实施例的在GPU的处理单元(例如,单个流式多处理器)上的工作的示例性线程束级视图。

[0023] 图3示出了根据一个实施例的并行处理单元。

[0024] 图4A示出了根据一个实施例的图3的并行处理单元内的通用处理集群。

[0025] 图4B示出了根据一个实施例的图3的并行处理单元的存储器分区单元。

[0026] 图5A示出了根据一个实施例的图4A的流式多处理器。

[0027] 图5B是根据一个实施例的使用图3的并行处理单元(PPU)实现的处理系统的概念图。

[0028] 图5C示出了可以实现各种先前实施例的各种架构和/或功能的示例性系统。

[0029] 图6是根据一个实施例的由图3的PPU实现的图形处理管道的概念图。

[0030] 图7示意性地示出了根据一些示例性实施例的关于图1和图2描述的调度器和仲裁器的进一步细节。

[0031] 图8示出了一种过程,应用程序可以通过这一过程配置调度策略以便控制GPU(或其他类型的PPU)如何调度其图形工作项及其计算工作项的一些方面。

[0032] 图9示出了根据一些示例性实施例的可以由调度器(诸如调度器)执行的过程的流程图。

[0033] 图10示出了根据一些示例性实施例的可以由仲裁器(诸如仲裁器)执行的过程的流程图。

[0034] 图11示出了一种过程的流程图,这一过程可以由诸如CPU的主处理器中的应用程序或其他程序执行,以便配置一个或更多个调度配置文件和用于在调度配置文件之间切换的定时器。

[0035] 图12示出了一种过程的流程图,这一过程可以在GPU中执行或在一些实施例中在M管线控制器(MPC)中执行,以在一系列配置的调度器配置文件中切换调度器配置文件。

[0036] 图13示出了根据一些示例实施例的空闲检测过程1300的流程图。

具体实施方式

[0037] 示例性实施例消除了上述用于在图形任务和计算任务之间共享GPU的处理资源的技术的一些缺陷。示例性实施例例如提供了一种硬件调度器,其可以有效地调度并行处理单元(例如,GPU)上的图形工作和计算工作的独立序列用于同时执行。具体地,该硬件调度器配置成通过基于指定软件的调度控制和来自执行单元对它们有多繁忙的本地化反馈以线程组粒度(例如,线程束粒度)进行调度决策来最佳地使用计算资源(例如,称为SM的流式多处理器)。一些示例性实施例使得能够通过启动一个或更多个图形线程束和一个或更多个计算线程束以在同一SM上同时执行来在图形管线和计算管线之间共享GPU处理单元(诸如SM),从而在图形功能和计算功能之间共享SM而无需场景切换或完成任一类型的功能。

[0038] 根据实施例的GPU配置成在单个SM上同时运行图形着色器(例如,顶点、曲面细分和几何(VTG)着色器、像素着色器等)和计算着色器。计算着色器是可编程着色器阶段,其扩展传统图形应用程序编程接口(API)(例如Microsoft Direct3D11)超出图形编程。这意味着来自所有工作类型的线程束可以在同一SM上同时运行,线程束可以竞争SM资源(例如寄存器文件空间、内部缓冲区条目(ISBE)空间、共享内存空间、SM ALU资源、L1缓存带宽、纵横开关带宽、帧缓冲带宽等)。

[0039] 实施例提供GPU以在线程束粒度下调度计算工作和图形工作,同时还提供软件以向与工作相一致的硬件调度器提供调度提示,以在不同工作桶之间微调性能,使得能够设置关于每个计算内核启动时的计算优先级相对于图形优先级,提供来自GPU上执行单元的反馈,以便在调度器尝试做出调度决策时保持通知调度器,并使能平衡跨执行单元的图形工作以减少由于图形管线的有序性质而产生的利用率不足等。示例性实施例使用硬件实现的同时计算和图形仲裁器(SCG-SM仲裁器)来读取SM占用率和利用率统计或其他度量,以确定用于图形与计算的SM的时间/空间分配百分比。

[0040] 以前的GPU,例如申请号为14/137,818的美国申请中描述的GPU,使硬件中的调度器能够在运行时调整SM的图形组与计算组的比率。但是,需要软件参考SM组的适当组合做出决定,并随着工作执行通过计算方法在帧期间对其进行调整。但是,许多实际场景中的图形工作负载和计算工作负载是异步的,并且在随机时间到达且随机对齐,因此在运行时通过计算方法调整调度对于软件来说是不实际的。示例实施例还可以改进特征中的先前技术,例如,硬编码的计算贪婪策略不够灵活以动态地对不同工作负载作出反应,不能将任何资源分配给计算或图形(例如,仅在z期间渲染或2D绘制),具有与SM的耗尽相关的高延迟,以及就整个SM等而言的具有高分配粒度。

[0041] 由于各个处理单元(诸如SM)在计算任务和图形任务之间的动态分配的底层支持,实施例的优点包括提升执行速度,改善资源利用等。在实施例的许多优点中,改善对于异步计算支持也与传统图形API相关,例如DirectX12(DX12)API,其高度鼓励在游戏中创建异步计算技术。同时计算和图形是DX12API的关键特征。在示例性实施例中提供用于在相同SM上同时运行图形和计算的能力,除了其他有利之外,还避免了在先前方法中发现的延迟时间

问题等,其中例如需要疏通协议在切换SM以运行计算线程束之前,强制所有图形线程束完成操作。

[0042] 用于在一个SM上同时执行计算和图形的系统和方法

[0043] 图1示出了根据一些示例性实施例的系统100,其中图形工作项和计算工作项可以在并行处理单元102 (如GPU) 的多个单指令多数据 (SIMD) 或单指令多线程 (SMT) 处理单元之一中同时处理。以下结合图3描述示例并行处理单元。在系统100中,示出SIMD处理单元104,其具有同时执行的第一组一个或更多个图形任务106和第二组一个或更多个计算任务108。这与先前的系统形成对比,例如,在于2013年12月20日提交的申请号为14/137,818中的共同拥有的共同未决的美国申请中描述的系统,尽管GPU可以在单独的处理单元上同时处理图形任务和计算任务,但是这种同时执行不在GPU的相同SIMD处理单元中。本公开中使用的“处理单元”指的是SIMD或SMT处理单元,例如但不限于诸如下面关于图4A描述的流式多处理器 (SM)。如图3和图4A所示,示例PPU 300在其每个处理核350中包括多个SM。

[0044] 调度第一组图形任务106和第二组计算任务108以由硬件调度器110在处理单元104上同时执行,硬件调度器110调度来自位于存储其117中的图形工作项的图形队列114的任务和计算工作项的计算队列116的任务。图形队列114中的图形工作项可以通过一个或更多个图形管线排列,并且计算队列116中的计算工作项可以通过一个或更多个计算管线排列。“图形管线”(图6中示出的示例)是处理与图像的阴影像素相关的图形数据的一组逻辑阶段,其中每个阶段能够以管线方式单独调度。“计算管线”是用于处理与图像中的阴影像素不直接相关的数据的一组逻辑阶段。示例计算管线可以包括但不限于与生成动画模型、分析来自科学或金融领域的大量数据集等相关联的物理计算。

[0045] 硬件调度器110配置成以“图形贪婪模式”或以“计算贪婪模式”操作。在“图形贪婪模式”中它从图形队列114重复提取工作项并发射到处理单元104,在“计算贪婪模式”中它从计算队列116重复提取工作项并发射到处理单元104。

[0046] 硬件仲裁器112操作以监控与处理单元104相关联的执行和存储资源的性能和/或利用率,并且在还考虑用户或软件定义的用于策略和/或任务优先级的配置参数的同时,确定是否应当将当前用于发射的非贪婪类型的任务插入到所重复选择的贪婪类型的任务之间以在处理单元104中执行。更具体地,硬件仲裁器112连续地(或在可配置阶段周期性地)监视113预定的一组处理或存储资源的运行时间统计数据,以识别当前资源使用中的“漏洞(hole)” (例如,过剩的未使用容量),然后可以通过启动非贪婪类型的一些任务来用尽(例如,占用)。当硬件调度器110以特定贪婪模式操作时,也可以将这些漏洞视为启动非贪婪类型的一个或更多个任务的“机会”。硬件仲裁器112可以向硬件调度器110提供信号111,通知它关于所检测到的任何漏洞。

[0047] 在示例实施例中,每个处理单元包括硬件调度器110和硬件仲裁器112。例如,在图4A中,每个SM 440可以具有相关联的硬件调度器110和相关联的硬件仲裁器112,布置在其数据处理集群(DPC)内。

[0048] 可以根据由在CPU 124上执行的应用程序122和/或驱动软件配置的策略120,而由硬件仲裁器112执行监控。策略120可以设置硬件仲裁器的类型(例如,SCG-SM仲裁器的类型),用于确定当前资源利用率中的漏洞的阈值,在检测到漏洞时启动非贪婪任务的速率等。应用程序122可以是计算机游戏(例如,也称为视频游戏),或者具有将工作项排入图形

队列114的图形要求和将工作项排入计算队列116的计算要求两者的其他应用程序。应用程序可以将其工作项126与相关联的任务优先级一起发送到存储器117和将相应的指令128发送到并行处理单元 (PPU) 102。

[0049] 图2A示出了根据一些示例性实施例的调度过程200的流程图。调度过程200可以例如在PPU 102中由包括硬件调度器110和硬件仲裁器112的硬件单元执行,以动态地调度图形任务和计算任务以在同一SIMD处理单元104上同时执行。

[0050] 在进入过程200之后,在操作202中,硬件调度器根据当前优先级模式从图形队列或计算队列重复启动工作项,以在诸如处理单元104 (例如,SM) 的处理单元上执行。例如,当处于图形贪婪模式时,可以重复启动来自图形队列的任务,并且当处于计算贪婪模式时,可以重复启动来自计算队列的任务。可以以各种方式设置优先级模式 (例如,图形贪婪或计算贪婪)。在一些实施例中,优先级模式基于图形队列和计算队列中的各个项的优先级中的较高者。

[0051] 在操作204中,作为监视与处理单元相关联的资源的结果,可以检测在当前利用的一个或更多个资源中的“漏洞”。可以通过将当前利用率 (由在时间间隔上平均的统计数据表示) 与调度策略中指定的一个或更多个阈值进行比较来检测漏洞。

[0052] 在操作206中,响应于检测到漏洞,调度器针对当前的非贪婪优先级从队列中启动一个或更多个工作项。例如,在图形贪婪模式下进行操作时,检测到漏洞可能会提示调度器启动某些计算工作项,或者可选地,在计算贪婪模式下操作时检测到的漏洞可能会提示调度器启动某些图形工作项。可以根据检测到的漏洞所指示的资源可用性和/或当前活跃的调度策略中指定的流动率来确定所启动的非贪婪优先级工作项。

[0053] 因此,在操作206中,处理单元可以具有在其上同时执行的图形工作项和计算工作项,同时两者竞争公共资源。

[0054] 尽管在处理单元和某些硬件组件的上下文中描述了方法200,但是方法200也可以由定制电路执行、或者由定制电路和程序的组合执行。例如,方法200可以由GPU (图形处理单元) 或能够执行操作202-206的任何一个或更多个处理器来执行。此外,本领域普通技术人员将理解,执行方法200的任何系统都在本发明实施例的范围和精神内。

[0055] 示例性线程束工作负载分配

[0056] 图2B示出了包括在传统GPU中的一个处理单元上的图形和计算的示例性工作负载。如图所示,在任何给定时间,只有单一类型的线程束可以占据处理单元。如图所示,处理单元执行序列,包括图形线程束的全部序列,接着是计算线程束的全部序列,并且再次图形线程束的全部序列。然后在只调度一部分计算任务的情况下继续序列,其中未使用一些处理单元的容量。

[0057] 图2C提供了根据某些示例性实施例的随时间分配给处理单元的示例性线程束。例如,处理单元可能能够具有多个 (例如,32个) 并发线程束。图2C示出了处理单元最初以图形贪婪模式操作,然后简单地处于计算贪婪模式,并再次处于图形贪婪模式。它还示出了在图形贪婪部分期间,在某些时间,例如,当检测到漏洞 (未充分利用) 时,在处理单元上调度了少量计算线程束。图中没有填充图案的区域可以指示在很短的时间之后检测到工作负载中的漏洞 (例如,由于时间上平均的统计数据)。超过一半的工作负载是计算线程束的区域可能代表计算贪婪阶段,具有大量图形线程束。

[0058] 图2B和2C的比较示出了对处理单元容量的更好利用。图2C中表示处理单元是共享的,并且还示出了在特定贪婪模式是活跃的同时,可以启动至少一些非贪婪类型的线程束。

[0059] 可以对示例性实施例的至少一些方面进行类比说明。警察在交叉路口处手动引导交通,其中两个进站道路中的汽车正在竞争以进入单个出站道路,可以类似于硬件调度器调度在特定处理单元上来自于两个工作队列上的工作。第一个进站车道拥有所有重要的车辆,警察允许他们尽快进入出站道路。在第一个进站道路的多个车道被填满的特别繁忙时期,警察可能有一个助手,该助手对交通水平具有更有利的观察点。与示例性实施例中的硬件仲裁器一样,该助手可能偶尔在观察到交通中的间歇时向警察发出信号,警察迅速允许来自第二进站道路的一个或其他少量汽车进入出站道路。因此,以类似于在同时发生的图形线程束和计算线程束之间共享以图形贪婪模式操作的特定处理单元的方式,在来自第一进站道路和第二进站道路的汽车之间共享出站道路。

[0060] 现在将根据用户的期望,阐述关于可以或可以不实现前述框架的各种可选架构和特征的更多说明性信息。应该强烈注意的是,出于说明性目的阐述了以下信息,并且不应将其解释为以任何方式进行限制。接下来的特征中可以选择性的并入或不排除所描述的其他特征。

[0061] 并行处理架构

[0062] 图3示出了根据一个实施例的并行处理单元 (PPU) 300。在一个实施例中,PPU 300是在一个或更多个集成电路设备上实现的多线程处理器。PPU 300是一种延迟隐藏架构,旨在并行处理多个线程。线程(即,执行的线程)是配置成由PPU 300执行的一组指令的实例。在一个实施例中,PPU 300是配置成实现图形渲染管线的图形处理单元 (GPU),用于处理三维 (3D) 图形数据以生成二维 (2D) 图像数据在诸如液晶显示 (LCD) 设备的显示设备上显示。在其他实施例中,PPU 300可以用于执行通用计算。尽管出于说明性目的在此提供了一个示例性并行处理器,但应该强烈注意,这样的处理器仅出于说明性目的而提出,并且可以采用任何处理器来补充和/或替代相同的处理器。

[0063] 一个或更多个PPU 300可以配置成加速数千个高性能计算 (HPC)、数据中心和机器学习应用程序。PPU 300可以配置成加速许多深度学习系统和应用程序,包括自动驾驶平台,深度学习,高精度语音、图像和文本识别系统,智能视频分析,分子模拟,药物发现,疾病诊断,天气预报,大数据分析,天文学,分子动力学模拟,金融建模,机器人技术,工厂自动化,实时语言翻译,在线搜索优化和个性化用户推荐等。

[0064] 如图3所示,PPU 300包括输入/输出 (I/O) 单元305,前端单元315,调度器单元320,工作分配单元325,集线器330,交叉开关 (Xbar) 370,一个或更多个通用处理集群 (GPC) 350,以及一个或更多个分区单元380。PPU 300可以经由一个或更多个高速NVLink 310互连连接到主处理器或其他PPU 300。PPU 300可以经由互连302连接到主处理器或其他外围设备。PPU 300还可以连接到包括多个存储器设备304的本地存储器。在一个实施例中,本地存储器可以包括多个动态随机存取存储器 (DRAM) 设备。DRAM设备可以配置成高带宽存储器 (HBM) 子系统,具有堆叠在每个设备内的多个DRAM管芯。

[0065] NVLink 310互连使系统能够扩展并包括与一个或更多个CPU结合的一个或更多个PPU 300,支持PPU 300和CPU之间的高速缓存一致性,以及CPU主控。数据和/或命令可以由NVLink 310通过集线器330发送到PPU 300的其他单元,或者可以从PPU 300的其他单元通

过集线器330传输,其他单元例如一个或多个复制引擎、视频编码器、视频解码器、电源管理单元等(未明确地显示)。结合图5B更详细地描述NVLink 310。

[0066] I/O单元305配置成通过互连302从主处理器(未示出)发送和接收通信(即,命令,数据等)。I/O单元305可以通过互连302或通过一个或多个中间设备(例如存储器桥)与主处理器直接通信。在一个实施例中,I/O单元305可以经由互连302与一个或多个其他处理器(诸如PPU 300中的一个或多个)通信。在一个实施例中,I/O单元305实现外围组件互连高速(PCIe)接口,以用于通过PCIe总线进行通信,以及互连302是PCIe总线。在可选实施例中,I/O单元305可以实现用于与外部设备通信的其他类型的众所周知的接口。

[0067] I/O单元305对经由互连302接收的数据包进行解码。在一个实施例中,数据包表示配置成使PPU 300执行各种操作的命令。如命令指定的那样,I/O单元305将解码的命令发送到PPU 300的各种其他单元。例如,可以将一些命令发送到前端单元315。可以将其他命令发送到集线器330或PPU 300的其他单元,诸如一个或多个复制引擎、视频编码器、视频解码器、电源管理单元等(未明确显示)。换句话说,I/O单元305配置成在PPU 300的各种逻辑单元之间路由通信。

[0068] 在一个实施例中,由主机处理器执行的程序对缓冲器中的命令流进行编码,该缓冲器向PPU 300提供工作负载以进行处理。工作负载可以包括由这些指令处理的若干指令和数据。缓冲区是存储器中可由主处理器和PPU 300访问(即,读/写)的区域。例如,I/O单元305可以配置成访问系统存储器中的缓冲区,系统存储器经由通过互连302传输的存储器请求连接到互连302。在一个实施例中,主处理器将命令流写入缓冲器,然后将指向命令流的开始的指针发送到PPU 300。前端单元315接收指向一个或多个命令流的指针。前端单元315管理一个或多个流,从流中读取命令并将命令转发到PPU 300的各个单元。

[0069] 前端单元315耦合到调度器单元320,调度器单元320配置各种GPC 350以处理由一个或多个流定义的任务。调度器单元320配置成跟踪与由调度器单元320管理的各种任务有关的状态信息。状态可以指示将任务分配给哪个GPC 350,任务是活跃的还是非活跃的,与任务相关联的优先级等等。调度器单元320管理一个或多个GPC 350上的多个任务的执行。

[0070] 调度器单元320耦合到工作分配单元325,工作分配单元325配置成分派任务以在GPC 350上执行。工作分配单元325可以跟踪从调度器单元320接收的多个调度任务。在一个实施例中,工作分配单元325管理每个GPC 350的待处理任务池和活跃任务池。待处理任务池可以包括多个时隙(例如,32个时隙),其包含将要分配以由特定GPC 350处理的任務。活跃任务池可以包括多个时隙(例如,4个时隙),用于由GPC 350正在活跃处理的任務。当GPC 350完成执行任务时,从GPC 350的活跃任务池中逐出该任务,并且选择来自待处理任务池的其他任务之一并对其进行调度以在GPC 350上执行。如果活跃任务在GPC 350上空闲,例如在等待有待解决的数据依赖性,则可以从GPC 350中逐出活跃任务并返回到待处理任务池,同时选择待处理任务池中的其他任务并调度其在GPC 350上执行。

[0071] 工作分配单元325经由XBar 370与一个或多个GPC 350通信。XBar 370是将PPU 300的许多单元耦合到PPU 300的其他单元的互连网络。例如,XBar 370可以配置成将工作分配单元325耦合到特定GPC 350。虽然未明确示出,但PPU 300的一个或多个其他单元也可以经由集线器330连接到XBar 370。

[0072] 任务由调度器单元320管理,并由工作分配单元325分发到GPC 350。GPC 350配置成处理任务并生成结果。结果可以由GPC 350内的其他任务消耗,经由XBar 370路由到不同的GPC 350,或者存储在存储器304中。该结果可以经由分区单元380写入存储器304,分区单元380实现用于向/从存储器304读取和写入数据的存储器接口。该结果可以通过NVLink 310传输到另一个PPU 300或CPU。在一个实施例中,PPU 300包括U个分区单元380,其与耦合到PPU 300的分离和不同的存储器设备304的数量相等。下面将结合图4B更详细地描述分区单元380。

[0073] 在一个实施例中,主处理器执行驱动程序内核,该驱动程序内核实现应用程序编程接口(API),该API允许在主处理器上执行的一个或更多个应用程序以调度在PPU 300上执行的操作。在一个实施例中,多个计算应用程序同时由PPU 300执行并且PPU 300为多个计算应用程序提供隔离、服务质量(QoS)和独立地址空间。应用程序可以生成指令(即,API调用),其使得驱动程序内核生成一个或更多个任务以供PPU 300执行。驱动程序内核将任务输出到由PPU 300处理的一个或更多个流。每个任务可以包括一组或更多组相关线程,在此称为线程束。在一个实施例中,线程束包括可以并行执行的32个相关线程。协作线程可以指代多个线程,包括执行任务的指令并且可以通过共享存储器交换数据。结合图5A更详细地描述线程和协作线程。

[0074] 图4A示出了根据一个实施例的图3的PPU 300的GPC 350。如图4A所示,每个GPC 350包括用于处理任务的多个硬件单元。在一个实施例中,每个GPC 350包括管线管理器410,前-后操作单元(PROP)415,光栅引擎425,工作分配交叉开关(WDX)480,存储器管理单元(MMU)490,以及一个或更多个数据处理集群(DPC)420。应当理解,图4A的GPC 350可以包括代替图4A中所示的单元或者除了图4A中所示的单元之外的其他硬件单元。

[0075] 在一个实施例中,GPC 350的操作由管线管理器410控制。管线管理器410管理一个或多个DPC 420的配置,用于处理分配给GPC 350的任务。在一个实施例中,管线管理器410可以配置一个或更多个DPC 420中的至少一个,以实现图形渲染管线的至少一部分。例如,DPC 420可以配置成在可编程流式多处理器(SM)440上执行顶点着色器程序。管线管理器410还可以配置成将从工作分配单元325接收的数据包路由到GPC 350内的适当逻辑单元。例如,可以将一些数据包路由到PROP 415和/或光栅引擎425中的固定功能硬件单元,而可以将其他数据包路由到DPC 420以供原始引擎435或SM 440处理。在一个实施例中,管线管理器410可以配置一个或更多个DPC 420中的至少一个以实现神经网络模型和/或计算管线。

[0076] PROP单元415配置成将由光栅引擎425和DPC 420生成的数据路由到光栅操作(ROP)单元,结合图4B更详细地描述。PROP单元415还可以配置成执行颜色混合的优化,组织像素数据,执行地址转换等。

[0077] 光栅引擎425包括多个固定功能硬件单元,其配置成执行各种光栅操作。在一个实施例中,光栅引擎425包括设置引擎、粗光栅引擎、剔除引擎、剪切引擎、精细光栅引擎和图块合并引擎。设置引擎接收变换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。将平面方程发送到粗光栅引擎以生成图元的覆盖信息(例如,图块(tile)的x,y覆盖掩模)。将粗光栅引擎的输出发送到剔除引擎,其中将与未通过z测试的图元相关联的片段剔除,并且将未剔除的片段发送到剪切引擎,其中位于视锥体外的片段被剪切。可以将经过剪

裁和剔除的那些片段传递到精细光栅引擎,以根据由设置引擎生成的平面方程生成像素片段的属性。光栅引擎425的输出包括,例如,将要由DPC 420内实现的片段着色器来处理的片段。

[0078] 包括在GPC 350中的每个DPC 420包括M管道控制器(MPC)430、基元引擎435和一个或多个SM 440。MPC 430控制DPC 420的操作,将从管线管理器410接收的数据包路由到DPC 420中适合的单元。例如,可以将与顶点相关联的数据包路由到图元引擎435,图元引擎435配置成从存储器304获取与该顶点相关联的顶点属性。相比之下,可以将与着色器程序相关联的数据包发送到SM 440。

[0079] SM 440包括可编程流式处理器,其配置成处理由多个线程表示的任务。每个SM 440是多线程的并且配置成同时执行来自特定线程组的多个线程(例如,32个线程)。在一个实施例中,SM 440实现SIMD(单指令、多数据)架构,其中线程组(即,线程束)中的每个线程配置成基于同一组指令处理不同的数据集。线程组中的所有线程都执行相同的指令。在另一实施例中,SM 440实现SIMT(单指令,多线程)架构,其中线程组中的每个线程配置成基于同一组指令处理不同的数据集,但是其中在线程组中的各个线程允许线程在执行期间发散。在一个实施例中,为每个线程束维护程序计数器,调用栈和执行状态,当线程束内的线程发散时,使线程束和线程束内的串行执行之间的并发成为可能。在另一实施例中,为每个单独的线程维护程序计数器、调用栈和执行状态,从而在线程束内和线程束之间的所有线程之间实现相等的并发性。当为每个单独的线程维护执行状态时,可以将执行相同指令的线程收敛并且并行执行以获得最大效率。下面结合图5A更详细地描述SM440。

[0080] MMU 490提供GPC 350和分区单元380之间的接口。MMU 490可以提供虚拟地址到物理地址的转换、存储器保护和存储器请求的仲裁。在一个实施例中,MMU 490提供一个或多个转换后备缓冲器(TLB),用于执行存储器304中的虚拟地址到物理地址的转换。

[0081] 图4B示出了根据一个实施例的图3的PPU 300的存储器分区单元380。如图4B所示,存储器分区单元380包括光栅操作(ROP)单元450,二级(L2)高速缓存460和存储器接口470。存储器接口470耦合到存储器304。存储器接口470可以实现用于高速数据传输的32、64、128、1024位数据总线等。在一个实施例中,PPU 300包含U个存储器接口470,每对分区单元380一个存储器接口470,其中每对分区单元380连接到相应的存储器设备304。例如,PPU 300可以连接到Y个存储器设备304,例如高带宽存储器堆栈或图形双数据速率、版本5、同步动态随机存取存储器或其他类型的持久存储器。

[0082] 在一个实施例中,存储器接口470实现HBM2存储器接口并且Y等于一半U。在一个实施例中,HBM2存储器堆栈与PPU 300位于相同的物理封装上,与传统GDDR5SDRAM系统相比节省了大量功率和面积。在一个实施例中,每个HBM2堆栈包括四个存储器管芯并且Y等于4, HBM2堆栈包括每个管芯的两个128位通道,总共8个通道和1024位的数据总线宽度。

[0083] 在一个实施例中,存储器304支持单错校正双错检测(Single-Error Correcting Double-Error Detecting, SECDED) 纠错码(ECC)以保护数据。ECC为对数据损坏敏感的计算应用程序提供更高的可靠性。在PPU 300处理非常大的数据集和/或长时间运行应用程序的大规模集群计算环境中,可靠性尤为重要。

[0084] 在一个实施例中,PPU 300实现多级存储器层次结构。在一个实施例中,存储器分区单元380支持统一存储器以为CPU和PPU 300存储器提供单个统一虚拟地址空间,从而实

现虚拟存储器系统之间的数据共享。在一个实施例中,跟踪PPU 300访问位于其他处理器上的存储器的频率,以确保将存储器页面移动到更频繁地访问页面的PPU 300的物理存储器。在一个实施例中,NVLink 310支持地址转换服务,允许PPU 300直接访问CPU的页表并通过PPU 300提供对CPU存储器的完全访问。

[0085] 在一个实施例中,复制引擎在多个PPU 300之间或在PPU 300和CPU之间传输数据。复制引擎可以为未映射到页表的地址生成页面错误。然后,存储器分区单元380可以处理页面错误,将地址映射到页表中,之后复制引擎可以执行传输。在传统系统中,将存储器固定(即,不可分页)以用于多个处理器之间的多个复制引擎操作,从而大大减少了可用存储器。随着硬件页面错误,可以将地址传递给复制引擎,而不必担心内存页是否驻留,并且复制过程是透明的。

[0086] 来自存储器304或其他系统存储器的数据可以由存储器分区单元380获取并存储在L2高速缓存460中,L2高速缓存460位于芯片上并且在各种GPC 350之间共享。如图所示,每个存储器分区单元380包括与相应的存储器设备304相关联的L2高速缓存460的一部分。然后,可以在GPC 350内的各种单元中实现较低级高速缓存。例如,SM 440中的每一个可以实现一级(L1)高速缓存。L1高速缓存是专用于特定SM 440的私有存储器。可以获取来自L2高速缓存460的数据并将其存储在每个L1高速缓存中,以便在SM 440的功能单元中进行处理。将L2高速缓存460耦合到存储器接口470和XBar 370。

[0087] ROP单元450执行与像素颜色有关的图形光栅操作,例如颜色压缩、像素混合等。ROP单元450还结合光栅引擎425实现深度测试,接收与来自光栅引擎425的剔除引擎的像素片段相关联的样本位置的深度。针对深度缓冲器中的对应深度为与片段关联的样本位置测试深度。如果片段通过样本位置的深度测试,则ROP单元450更新深度缓冲器并将深度测试的结果发送到光栅引擎425。应当理解,分区单元380的数量可以不同于GPC 350的数量,因此,每个ROP单元450可以耦合到GPC 350中的每个。ROP单元450跟踪从不同GPC 350接收的数据包,并确定将由ROP单元450生成的结果通过Xbar 370路由到哪个GPC 350。虽然ROP单元450包括在图4B中的存储器分区单元380内,但是在其他实施例中,ROP单元450可以位于存储器分区单元380之外。例如,ROP单元450可以驻留在GPC 350或其他单元中。

[0088] 图5A示出了根据一个实施例的图4A的流式多处理器440。如图5A所示,SM 440包括指令高速缓存505、一个或更多个调度器单元510、寄存器文件520、一个或更多个处理核心550、一个或更多个特殊功能单元(SFU) 552、一个或更多个负载/存储单元(LSU) 554、互连网络580、共享存储器/L1高速缓存570。

[0089] 如上所述,工作分配单元325分发任务用于在PPU 300的GPC 350上执行。将任务分配给GPC 350内的特定DPC 420,并且如果任务与着色器程序相关联,则可以将任务分配给SM 440。调度器单元510从工作分配单元325接收任务并管理指令调度用于将一个或更多个线程块分配给SM 440。调度器单元510调度用于执行的线程块作为并行线程的线程束,其中给每个线程块至少分配一个线程束。在一个实施例中,每个线程束执行32个线程。调度器单元510可以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期间将指令从多个不同的协作组分派给各个功能单元(即,核心550,SFU 552和LSU 554)。

[0090] 协作组是一种用于组织通信线程组的编程模型,它允许开发人员表达线程正在进行通信的粒度,从而实现更丰富、更有效的并行分解。协作启动API支持线程块之间的同步,

以执行并行算法。传统编程模型提供单个简单构造用于同步协作线程：跨线程块的所有线程的屏障（即，`syncthreads()` 函数）。但是，程序员通常希望以小于线程块粒度的方式定义线程组，并在所定义的组内进行同步，以便以集体组范围的功能接口的形式实现更高的性能、设计灵活性和软件重用。

[0091] 协作组使程序员能够以子块（即，与单个线程一样小）和多块粒度明确地定义线程组，并且执行集合操作，例如在协作组中的线程上进行同步。编程模型支持跨软件边界的清晰组合，因此库和实用函数可以在其本地上下文中安全地同步，而无需对收敛进行假设。协作组图元支持新的协作并行模式，包括生产者-消费者并行性，机会并行性以及跨整个线程块网格的全局同步。

[0092] 分派单元515配置成将指令发送到一个或多个功能单元。在该实施例中，调度器单元510包括两个分派单元515，其使得能够在每个时钟周期期间分派来自相同线程束的两个不同指令。在备选实施例中，每个调度器单元510可以包括单个分派单元515或附加分派单元515。

[0093] 每个SM 440包括寄存器文件520，其为SM 440的功能单元提供一组寄存器。在一个实施例中，寄存器文件520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520的一个专用部分。在另一个实施例中，寄存器文件520在由SM 440执行的不同线程束之间划分。寄存器文件520为连接到功能单元的数据路径的操作对象提供临时存储。

[0094] 每个SM 440包括L个处理核心550。在一个实施例中，SM 440包括大量（例如，128等）不同的处理核心550。每个核心550可以包括全管线，单精度，双精度，和/或包括浮点算术逻辑单元和整数算术逻辑单元的混合精度处理单元。在一个实施例中，浮点算术逻辑单元实现用于浮点算术的IEEE 754-2008标准。在一个实施例中，核心550包括64个单精度（32位）浮点核，64个整数核，32个双精度（64位）浮点核和8个张量核。

[0095] 张量核配置成执行矩阵运算，并且在一个实施例中，核心550中包括一个或多个张量核。具体地，张量核配置成执行深度学习矩阵算术，例如用于神经网络训练和推测的卷积运算。在一个实施例中，每个张量核在 4×4 矩阵上操作并执行矩阵乘法和累加操作 $D = A \times B + C$ ，其中A、B、C和D是 4×4 矩阵。

[0096] 在一个实施例中，矩阵乘法输入A和B是16位浮点矩阵，而累加矩阵C和D可以是16位浮点矩阵或32位浮点矩阵。张量核在具有32位浮点累加的16位浮点输入数据上运算。16位浮点乘法需要64次运算，并产生一个全精度乘积，然后使用32位浮点加法与其他中间乘积累加以用于 $4 \times 4 \times 4$ 矩阵乘法。在实践中，张量核用于执行更大的二维或更高维矩阵运算，这些运算是从这些较小的元素构建的。诸如CUDA 9C++API之类的API公开了专门的矩阵加载，矩阵乘法和累加以及矩阵存储运算，以有效地使用来自CUDA-C++程序的张量核。在CUDA级别，线程束级接口假设 16×16 大小的矩阵跨越线程束的所有32个线程。

[0097] 每个SM 440还包括执行特殊功能（例如，属性评估、倒数平方根等）的M个SFU 552。在一个实施例中，SFU 552可以包括树遍历单元，其配置成遍历分层树数据结构。在一个实施例中，SFU 552可以包括配置成执行纹理映射过滤操作的纹理单元。在一个实施例中，纹理单元配置成从存储器304加载纹理图（例如，2D纹理像素阵列）并对纹理图进行采样以产生采样纹理值以供在SM 440执行的着色器程序中使用。在一个实施例中，纹理图存储在共享存储器/L1高速缓存470中。纹理单元实现纹理操作，例如使用纹理映射匹配（mip-map）的

过滤操作(即,不同细节级别的纹理图)。在一个实施例中,每个SM 340包括两个纹理单元。

[0098] 每个SM 440还包括N个LSU 554,其实现共享存储器/L1高速缓存570和寄存器文件520之间的加载和存储操作。每个SM 440包括互连网络580,其将每个功能单元连接到寄存器文件520和将LSU 554连接到寄存器文件520、共享存储器/L1高速缓存570。在一个实施例中,互连网络580是交叉开关,其可以配置成将任何功能单元连接到寄存器文件520中的任何寄存器,并将LSU 554连接到寄存器文件和共享存储器/L1高速缓存570中的存储器位置。

[0099] 共享存储器/L1高速缓存570是片上存储器阵列,其允许SM 440与基元引擎435之间和SM 440中的线程之间的数据存储和通信。在一个实施例中,共享存储器/L1高速缓存570包括128KB的存储容量并且位于从SM 440到分区单元380的路径中。共享存储器/L1高速缓存570可用于高速缓存读和写。共享存储器/L1高速缓存570、L2高速缓存460和存储器304中的一个或多个是后备存储。

[0100] 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,以为两种类型的存储器访问提供了最佳的整体性能。容量可用作不使用共享存储器的程序的缓存。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。共享存储器/L1高速缓存570内的集成使得共享存储器/L1高速缓存570能够用作用于流数据的高吞吐量管道,同时提供对频繁重用数据的高带宽和低延迟访问。

[0101] 当配置为通用并行计算时,与图形处理相比,可以使用更简单的配置。具体地,绕过图3中所示的固定功能图形处理单元,从而创建更简单的编程模型。在通用并行计算配置中,工作分配单元325对线程块进行分配,并将其直接分配给DPC 420。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 440执行程序并执行计算,使用共享存储器/L1高速缓存570在线程之间通信,以及使用LSU 554通过共享存储器/L1高速缓存570和存储器分区单元380读写全局存储器。当配置用来进行通用并行计算时,SM 440还可以写命令,调度器单元320可以使用该命令以在DPC 420上启动新工作。

[0102] PPU 300可以包括在台式计算机,膝上型计算机,平板计算机,服务器,超级计算机,智能电话(例如,无线、手持设备),个人数字助理(PDA),数码相机,车辆,头戴式显示器,手持电子设备等中。在一个实施例中,PPU 300体现在单个半导体衬底上。在另一个实施例中,PPU 300与一个或更多个其他设备(例如附加PPU 300、存储器304、精简指令集计算机(RISC)CPU、存储器管理单元(MMU)、数模转换器(DAC)等)一起包括在片上系统(SoC)中。

[0103] 在一个实施例中,PPU 300可以包括在包括一个或更多个存储器设备304的图形卡上。图形卡可以配置成与台式计算机的主板上的PCIe插槽进行接口。在又一个实施例中,PPU 300可以是包括在主板的芯片组中的集成图形处理单元(iGPU)或并行处理器。

[0104] 示例性计算系统

[0105] 具有多个GPU和CPU的系统用于各种行业,因为开发人员在人工智能计算等应用程序中公开并利用更多并行性。数据中心,研究机构和超级计算机中部署了具有成千上万个计算节点的高性能GPU加速系统,以解决日益严峻的问题。随着高性能系统内处理设备数量的增加,需要扩展通信和数据传输机制以支持这一增加。

[0106] 图5B是根据实施例的使用图3的PPU 300实现的处理系统500的概念图。示例性系

统500可以配置成实现图2A中所示的方法200。处理系统500包括CPU 530、开关555、以及多个PPU 300和每个PPU各自的存储器304。NVLink 310在每个PPU 300之间提供高速通信链路。尽管在图5B中示出了特定数量的NVLink 310和互连302连接,但是到每个PPU 300和CPU 530的连接的数量是可以变化的。开关555在互连302和CPU 530之间接口。PPU 300、存储器304和NVLink 310可以位于单个半导体平台上以形成并行处理模块525。在一个实施例中,开关555支持两个或更多个各种不同连接和/或链接之间的接口协议。

[0107] 在另一个实施例(未示出)中,NVLink 310在每个PPU 300和CPU 530之间提供一个或更多个高速通信链路,并且开关555在互连302和每个PPU 300之间接口。PPU 300、存储器304和互连302可以位于单个半导体平台上以形成并行处理模块525。在又一个实施例(未示出)中,互连302在每个PPU 300和CPU 530之间提供一个或更多个通信链路。开关555使用NVLink310在每个PPU 300之间接口,以在PPU 300之间提供一个或更多个高速通信链路。在另一个实施例(未示出)中,NVLink 310通过开关555提供PPU 300和CPU 530之间的一个或更多个高速通信链路。在另一个实施例(未示出)中,互连302直接在每个PPU 300之间提供一个或更多个通信链路。一个或更多个NVLink 310高速通信链路可以使用与NVLink 310相同的协议实现为物理NVLink互连或片上互连或管芯上互连。

[0108] 在本说明书的上下文中,单个半导体平台可以指在管芯或芯片上制造的唯一的基于单一半导体的集成电路。应当注意,术语单个半导体平台还可以指具有增加的连接性的多芯片模块,其模拟片上操作并且相对于利用传统总线实现方式进行实质性改进。当然,根据用户的期望,各种电路或设备也可以单独地或以半导体平台的各种组合设置。或者,并行处理模块525可以实现为电路板基板,并且每个PPU 300和/或存储器304可以是封装设备。在一个实施例中,CPU 530、开关555和并行处理模块525位于单个半导体平台上。

[0109] 在一个实施例中,每个NVLink 310的信令速率是20到25千兆比特/秒,并且每个PPU 300包括六个NVLink 310接口(如图5B所示,每个PPU 300包括五个NVLink 310接口)。每个NVLink 310在每个方向上提供25千兆字节/秒的数据传输速率,其中六个链路提供300千兆字节/秒。当CPU 530还包括一个或更多个NVLink 310接口时,NVLink 310可以专门用于如图5B所示的PPU到PPU通信,或者PPU到PPU和PPU到CPU的某种组合。

[0110] 在一个实施例中,NVLink 310允许从CPU 530到每个PPU 300的存储器304的直接加载/存储/原子访问。在一个实施例中,NVLink 310支持一致性操作,允许将从存储器304读取的数据存储在CPU 530的高速缓存层次中,减少了CPU 530的高速缓存访问时延。在一个实施例中,NVLink 310包括支持用于地址转换服务(ATS),允许PPU 300直接访问CPU 530内的页表。一个或更多个NVLink 310还可以配置成在低功率模式下操作。

[0111] 图5C示出了示例性系统565,其中可以实现各种先前实施例的各种架构和/或功能。示例性系统565可以配置成实现图2中所示的方法200。

[0112] 如图所示,提供了系统565,其包括连接到通信总线575的至少一个中央处理单元530。通信总线575可以使用任何合适的协议来实现,例如PCI(外围组件互连),串行总线(PCI-Express),AGP(加速图形端口),超传输(HyperTransport)或任何其他总线或点对点通信协议。系统565还包括主存储器540。控制逻辑(软件)和数据存储在主存储器540中,主存储器540可以采用随机存取存储器(RAM)的形式。

[0113] 系统565还包括输入设备560、并行处理系统525和显示设备545(即传统的CRT(阴

极射线管)、LCD(液晶显示器)、LED(发光二极管)、等离子显示器等)。可以从例如键盘、鼠标、触摸板、麦克风等的输入设备560接收用户输入。上述模块和/或设备中的每一个甚至可以位于单个半导体平台上以形成系统565。或者,各个模块也可以根据用户的期望单独地或以半导体平台的各种组合来定位。

[0114] 此外,系统565可以通过网络接口535耦合到网络(例如,电信网络、局域网(LAN)、无线网络、诸如因特网的广域网(WAN)、对等网络、有线网络等)以用于通信。

[0115] 系统565还可以包括辅助存储(未示出)。辅助存储包括例如硬盘驱动器和/或可移除存储驱动,表现为软盘驱动、磁带驱动、光盘驱动、数字通用盘(DVD)驱动、记录设备、通用串行总线(USB)闪存。可移除存储驱动以众所周知的方式读写可移除存储单元。

[0116] 计算机程序或计算机控制逻辑算法可以存储在主存储器540和/或辅助存储中。在执行这样的计算机程序时使系统565能够执行各种功能。主存储器540、存储和/或任何其他存储是计算机可读介质的可能示例。

[0117] 可以在通用计算机系统、电路板系统、专用于娱乐目的的游戏控制台系统、特定应用系统和/或任何其他期望系统的环境中实现各种先前附图的架构和/或功能。例如,系统565可以采用的形式包括:台式计算机、膝上型计算机、平板计算机、服务器、超级计算机、智能电话(例如,无线、手持设备)、个人数字助理(PDA)、数码相机、车辆、头戴式显示器、手持电子设备、移动电话设备、电视、工作站、游戏控制台、嵌入式系统和/或任何其他类型的逻辑。

[0118] 虽然上面已经描述了各种实施例,但是应该理解,它们仅以示例的方式呈现,而不是限制。因此,优选实施例的宽度和范围不应受任何上述示例性实施例的限制,而应仅根据所附权利要求及其等同物来限定。

[0119] 图形处理管线

[0120] 在一个实施例中,PPU 300包括图形处理单元(GPU)。PPU 300配置成接收指定用于处理图形数据的着色器程序的命令。图形数据可以被定义为一组图元,例如点、线、三角形、四边形、三角形条带等。通常,图元包括为图元指定多个顶点的数据(例如,在模型空间坐标系中)以及与图元的每个顶点相关联的属性。PPU 300可以配置成处理图形图元以生成帧缓冲器(即,用于显示器的每个像素的像素数据)。

[0121] 应用程序将场景的模型数据(即,顶点和属性的集合)写入诸如系统存储器或存储器304的存储器。模型数据定义在显示器上可见的每个对象。然后,应用程序对驱动器内核进行API调用,以请求渲染和显示模型数据。驱动器内核读取模型数据并将命令写入一个或更多个流以执行处理模型数据的操作。命令可以引用要在PPU 300的SM 440上实现的不同着色器程序,包括顶点着色器、外壳着色器、域着色器、几何着色器和像素着色器中的一个或更多个。例如,SM 440中的一个或更多个可以配置成执行顶点着色器程序,该顶点着色器程序处理由模型数据定义的多个顶点。在一个实施例中,不同的SM 440可以配置成同时执行不同的着色器程序。例如,SM 440的第一子集可以配置成执行顶点着色器程序,而SM 440的第二子集可以配置成执行像素着色器程序。SM440的第一子集处理顶点数据以产生经处理的顶点数据并将经处理的顶点数据写入L2高速缓存460和/或存储器304。在将经处理的顶点数据光栅化之后(即,从三维数据变换为屏幕空间中二维数据)以产生片段数据,SM 440的第二子集执行像素着色器以产生经处理的片段数据,然后将其与其他经处理的片段

数据混合并写入存储器304中的帧缓冲器。顶点着色器程序和像素着色器程序可以同时执行,以管线方式处理来自同一场景的不同数据,直到将场景的所有模型数据都渲染到帧缓冲器。然后,将帧缓冲器的内容发送到显示控制器以在显示设备上显示。

[0122] 图6是根据一个实施例的由图3的PPU 300实现的图形处理管线600的概念图。图形处理管线600是实现为从3D几何数据生成2D计算机生成的图像的处理步骤的抽象流程图。众所周知,管线架构可以通过将操作分成多个阶段来更有效地执行长时延操作,其中将每个阶段的输出耦合到紧接的下一个阶段的输入。因此,图形处理管线600接收从图形处理管线600的一个阶段传输到下一个阶段的输入数据601以生成输出数据602。在一个实施例中,图形处理管线600可以表示由OpenGL[®] API定义的图形处理管线。作为选择,图形处理管线600可以在先前附图和/或任何后续附图的功能和架构的上下文中实现。

[0123] 如图6所示,图形处理管线600包括管线架构,该管线架构包括多个阶段。这些阶段包括但不限于数据集合(assembly)阶段610,顶点着色阶段620,图元集合阶段630,几何着色阶段640,视口缩放、剔除和剪辑(VSCC)阶段650,光栅化阶段660,片段着色阶段670和光栅操作阶段680。在一个实施例中,输入数据601包括由阶段处理的命令和几何图元(例如,点、线、三角形、四边形、三角形条或扇形等),该命令配置处理单元以实现图形处理管线600的阶段。输出数据602可以包括位于存储器中的将其复制到帧缓冲器中的像素数据(即,颜色数据)或其他类型的表面数据结构。

[0124] 数据集合阶段610接收输入数据601,输入数据601为高阶(high-order)表面、图元等指定顶点数据。数据集合阶段610在临时存储或队列中收集顶点数据,例如通过从主处理器接收包括指向存储器中的缓冲器的指针的命令和从缓冲器读取顶点数据。然后将顶点数据发送到顶点着色阶段620以进行处理。

[0125] 顶点着色阶段620通过对每个顶点执行一组操作(即,顶点着色器或程序)来处理顶点数据。例如,可以指定顶点为与一个或多个顶点属性(例如,颜色、纹理坐标、表面法线等)相关联的4坐标向量(即, $\langle x, y, z, w \rangle$)。顶点着色阶段620可以操纵各个顶点属性,例如位置、颜色、纹理坐标等。换句话说,顶点着色阶段620对顶点坐标或与顶点相关联的其他顶点属性执行操作。这些操作通常包括照明操作(即,修改顶点的颜色属性)和变换操作(即,修改顶点的坐标空间)。例如,可以使用对象坐标空间中的坐标来指定顶点,通过将坐标乘以将坐标从对象坐标空间转换为世界空间或标准化设备坐标(NDC)空间的矩阵来对顶点进行变换。顶点着色阶段620生成变换的顶点数据,将该变换的顶点数据发送到图元集合阶段630。

[0126] 图元集合阶段630收集由顶点着色阶段620输出的顶点,并将顶点分组为几何图元,以供几何着色阶段640处理。例如,图元集合阶段630可配置成将每三个连续顶点分组为几何图元(即,三角形),用于传输到几何着色阶段640。在一些实施例中,可以将特定顶点可以重新用于后续的几何图元(例如,三角形条带中的两个连续三角形可以共享两个顶点)。图元集合阶段630将几何图元(即,关联顶点的集合)发送到几何着色阶段640。

[0127] 几何着色阶段640通过对几何图元执行一组操作(即,几何着色器或程序)来处理几何图元。曲面细分操作可以从每个几何图元中生成一个或多个几何图元。换句话说,几何着色阶段640可以将每个几何图元细分为两个或多个几何图元的更精细网格,以供图形处理管线600的其余部分处理。几何着色阶段640将几何图元发送到视口SCC阶段650。

[0128] 在一个实施例中,图形处理管线600可以在流式多处理器内操作,以及顶点着色阶段620,图元集合阶段630,几何着色阶段640,片段着色阶段670和/或与其相关联的硬件/软件可以顺序地执行处理操作。一旦顺序处理操作完成,在一个实施例中,视口SCC阶段650可以利用该数据。在一个实施例中,可以将由图形处理管线600中的一个或多个阶段处理的图元数据写入高速缓存(例如,L1高速缓存、顶点高速缓存等)。在这种情况下,在一个实施例中,视口SCC阶段650可以访问高速缓存中的数据。在一个实施例中,可以将视口SCC阶段650和光栅化阶段660实现为固定功能电路。

[0129] 视口SCC阶段650执行几何图元的视口缩放、剔除和剪切。将渲染的每个表面与抽象摄像机位置相关联。摄像机位置表示观看者观看场景的位置,并定义包围场景的对象的视椎体(viewing frustum)。视椎体可包括观察平面、后平面和四个剪切平面。可以将完全在视椎体之外的任何几何图元剔除(即,丢弃),因为这些几何图元将不会对最终渲染场景做出贡献。可以将部分位于视椎体内部且部分位于视椎体外部的任何几何图元剪切(即,变换为封闭在视椎体内的新几何图元)。此外,几何图元可以各自基于视椎体的深度来缩放。然后,将所有可见的几何图元发送到光栅化阶段660。

[0130] 光栅化阶段660将3D几何图元转换为2D片段(例如,能够用于显示等)。光栅化阶段660可以配置成利用几何图元的顶点来设置一组平面方程,从中可以插入各种属性。光栅化阶段660还可以计算用于多个像素的覆盖掩模,其指示用于像素的一个或多个样本位置是否拦截几何图元。在一个实施例中,还可以执行z测试以确定该几何图元是否被已经光栅化的其他几何图元遮挡。光栅化阶段660生成片段数据(即,与每个所覆盖的像素的特定样本位置相关联的内插顶点属性),其被发送到片段着色阶段670。

[0131] 片段着色阶段670通过对每个片段执行一组操作(即,片段着色器或程序)来处理片段数据。片段着色阶段670可以诸如通过使用片段的内插纹理坐标执行照明操作或对纹理图进行采样,生成片段的像素数据(即,颜色值)。片段着色阶段670生成像素数据,其被发送到光栅操作阶段680。

[0132] 光栅操作阶段680可以对像素数据执行各种操作,诸如执行阿尔法测试、模板测试、以及将像素数据与对应于与像素相关联的其他片段的其他像素数据混合。当光栅操作阶段680已经完成处理像素数据(即,输出数据602)时,可以将像素数据写入渲染目标,例如帧缓冲器、颜色缓冲器等。

[0133] 应当理解,除了上述一个或多个阶段之外或代替上述一个或多个阶段,可以在图形处理管线600中包括一个或多个额外阶段。抽象图形处理管线的各种实现可以实现不同的阶段。此外,在一些实施例中,可以从图形处理管线中排除上述一个或多个阶段(例如,几何着色阶段640)。预期其他类型的图形处理管线也在本公开的范围。此外,图形处理管线600的任何阶段可以由诸如PPU 300的图形处理器内的一个或多个专用硬件单元实现。图形处理管线600的其他阶段可以由诸如PPU 300的SM 440的可编程硬件单元实现。

[0134] 图形处理管线600可以通过由主处理器(例如CPU)执行的应用程序来实现。在一个实施例中,设备驱动器可以实现应用程序编程接口(API),其定义可以由应用程序使用的各种功能,以便生成用于显示的图形数据。设备驱动器是包括控制PPU 300的操作的多个指令的软件程序。API为程序员提供抽象,该抽象允许程序员利用专用的图形硬件(例如PPU

300) 来生成图形数据而不需要程序员利用PPU 300的特定指令集。应用程序可以包括路由到PPU 300的设备驱动器的API调用。设备驱动器解释API调用并执行各种操作以响应API调用。在某些情况下,设备驱动器可以通过在CPU上执行指令来执行操作。在其他情况下,设备驱动器可以至少部分地通过利用CPU和PPU 300之间的输入/输出接口在PPU 300上启动操作来执行操作。在一个实施例中,设备驱动器配置成利用PPU 300的硬件实现图形处理管线600。

[0135] 可以在PPU 300内执行各种程序,以便实现图形处理管线600的各个阶段。例如,设备驱动器可以在PPU 300上启动内核以在一个SM 440(或者多个SM 440)上执行顶点着色阶段620。设备驱动器(或由PPU 300执行的初始内核)还可以启动PPU 300上的其他内核以执行图形处理管线600的其他阶段,例如几何着色阶段640和片段着色阶段670。此外,图形处理管线600的一些阶段可以在固定单元硬件上实现,例如在PPU 300内实现的光栅化器或数据汇编器。可以理解,来自一个内核的结果在由SM 440上的后续内核处理之前可以由一个或更多个中间固定硬件单元来处理。

[0136] 机器学习

[0137] 在PPU 300等处理器上开发的深度神经网络(DNN)已经被用于各种用例,从自动驾驶汽车到更快的药物开发速度,从在线图像数据库中的自动图像标注(captioning)到视频聊天应用程序中的智能实时语言翻译。深度学习是模拟人类大脑的神经学习过程的技术,不断学习,不断变得更聪明,并随着时间的推移更快地提供更准确结果。儿童最初由成人教导正确识别和分类各种形状,最终能够在没有任何指导的情况下识别形状。类似地,深度学习或神经学习系统需要在对象识别和分类方面进行训练,以便在识别基本对象、被遮挡对象等时更智能和更有效,同时还为对象分配上下文。

[0138] 在最简单的水平上,人类大脑中的神经元会查看接收到的各种输入,将重要性级别分配给这些输入中的每一个,并将输出传递给其他神经元以进行操作。人工神经元或感知器是神经网络的最基本模型。在一个示例中,感知器可以接收代表对象的各种特征的一个或更多个输入,感知器被训练以进行识别和分类,并且基于该特征在定义对象的形状时的重要性,为这些特征中的每一个分配特定权重。

[0139] 深度神经网络(DNN)模型包括多个连接的感知器(例如,节点)的多个层,其可以用大量输入数据进行训练以快速地以高精度快速解决复杂问题。在一个示例中,DNN模型的第一层将汽车的输入图像分解成各个部分并查找诸如线和角度的基本图案。第二层组装线以寻找更高级别的图案,例如轮子、挡风玻璃和镜子。下一层标识车辆的类型,最后几层为输入图像生成标签,识别特定汽车品牌的模型。

[0140] 一旦对DNN进行训练,就可以部署DNN并用于在称为推理的过程中识别和分类对象或模式。推理(DNN从给定输入中提取有用信息的过程)的示例包括识别存入ATM机的支票上的手写数字,识别照片中朋友的图像,向超过五千万用户提供电影推荐,在无人驾驶汽车中识别和分类不同类型的汽车、行人和道路危险,或实时翻译人类语音。

[0141] 在训练期间,数据在前向传播阶段中流过DNN,直到产生指示对应于输入的标签的预测。如果神经网络没有正确标记输入,则分析正确标签和预测标签之间的误差,并在向后传播阶段为每个特征调整权重,直到DNN正确标记输入和训练数据集中的其他输入。训练复杂的神经网络需要大量的并行计算性能,包括PPU 300支持的浮点乘法和加法。推理的计算

密集程度低于训练,是一种对延迟敏感的过程,其中将已经训练的神经网络应用于以前没有见过的新输入来分类图像,翻译语音,以及通常推断新信息。

[0142] 神经网络严重依赖于矩阵数学运算,复杂的多层网络需要大量的浮点性能和带宽,以提高效率和速度。PPU 300具有数千个处理内核,针对矩阵数学运算进行了优化,可提供数十至数百TFLOPS的性能,是一款能够提供基于深度神经网络的人工智能和机器学习应用所需性能的计算平台。

[0143] 在具有非平衡仲裁器的SM上的同时计算和图形处理

[0144] 图7示意性地示出了根据一些示例实施例的关于图1和图2描述的硬件调度器和硬件仲裁器的进一步细节。更具体地,图7示出了PPU(例如,PPU 102)的MPC 702以及在MPC 702中实现的调度器和仲裁器。在图7示出的实施例中,硬件调度器708和硬件仲裁器710位于与SM 704相关联的MPC 702中。在一些实施例中,调度器708和仲裁器710中的一个或两个实现在MPC 702外部,但是在与SM 704相同的DPC内。仲裁器710与SM 704的物理接近度促进对SM有效地进行非贪婪工作所必须的快速响应。物理接近度还提高了关于SM利用的反馈的准确性。

[0145] 根据一些实施例,调度器708和仲裁器710可以分别对应于调度器110和仲裁器112的实现。根据一些实施例,调度器708和仲裁器710以DPC级仲裁器模式操作,该模式是“非平衡的”,其中每个MPC基于来自其DPC的本地信息独立地做出决定(参考图4A描述了DPC和MPC)。

[0146] 调度器708操作以经由线程束启动块712重复调度线程组从而在SM 704上执行。由线程束启动块712提供给SM 704的线程束信息713可以包括SM可以在调度指令中使用的线程束优先级信息。线程组包括来自图形队列的图形工作项718或来自计算队列的计算工作项720,并且由调度器708根据当前优先级模式(例如,图形贪婪模式或计算贪婪模式)获得。每组线程可以是32个线程的组,如前所述,其可以被称为线程束。因为根据一些实施例,SM可以同时运行32个线程束(每个具有32个线程),所以调度器可以以不同的比例分配计算线程束和图形线程束,以在特定SM上同时执行。

[0147] 仲裁器710操作以鉴于一个或多个策略控制的阈值来监视预定的一组资源利用度量,并且通过提供信号716,使得当前在特定贪婪模式下操作的调度器708调度来自非贪婪工作队列一些工作项。

[0148] 反馈块714可以根据策略指定的一组阈值来实现对预定资源利用组的监视。

[0149] 在一些实施例中,调度器708获取图形工作项的方式可以与其接收计算工作项的方式不同。在图7的示意图中,调度器708可以直接(或通过前端单元(未示出))获得来自图形队列的图形工作项,但是需要请求计算工作分配单元706以获得计算工作项。计算工作分配单元706可以包括计算工作分配器(CWD) 722和计算调度单元(CSU) 724,其分别执行多个SM之间的计算工作项的分配和调度。

[0150] 图8示出了过程800,通过该过程,应用程序可以配置调度策略,以便控制GPU(或其他类型的PPU)如何调度其图形工作项及其计算工作项的一些方面。例如,过程800可以由诸如应用程序122的应用程序(例如,游戏、包括物理模拟的应用程序,使用诸如DX12的API的其他应用程序等)在CPU 124上执行来执行。

[0151] 在进入过程800之后,在操作802中,应用程序可以设置调度策略。调度策略可以包

括:用于从系统中可用的多个仲裁器中选择一个仲裁器的配置,用于针对每个图形贪婪模式和计算贪婪模式的操作检测未使用资源容量的“漏洞(hole)”的一个或多个性能度量,和/或资源利用/占用阈值,可以引入非贪婪工作项的涓流(trickle)率等。

[0152] 可以配置为策略内容的参数可以包括用于图形和用于计算的单独参数集。根据每种资源的特征和工作类型配置参数。例如,虽然将图形工作是否输出停止(stall)纳入性能和/或资源利用率度量可能是有帮助的,但是,至少在一些实施例中,将计算输出到共享存储器,对监控用于计算的输出停止可能没有额外的益处。与输出到共享存储器的计算相反,SM可以使用一个或多个固定功能加速单元进行图形工作。由于来自这些固定功能单元的背压在图形的输出停止度量中导致在SM处停滞,使得硬件调度器能够在已经备份与SM关联的固定功能单元的情况下调度计算而无需调度更多图形。

[0153] 设置为阈值的图形相关策略参数可以包括但不限于:输入饥饿阈值、输出停止阈值、图形线程束占用阈值、图形寄存器文件占用阈值、以及图形ISBE占用阈值。输入饥饿阈值所指定的阈值是,当低于该阈值时仲裁器可以认为特定输入管线正在使SM饥饿(可以分别为每个阿尔法管线,贝塔管线和像素管线(例如,分别为INPUT_STARVED_ALPHA, INPUT_STARVED_BETA, INPUT_STARVED_PIX)单独指定该参数;每个度量可以计算为二进制输出的积分,该二进制输出检查特定类型工作在平均窗口上的输入队列中是否可用)。输出停止阈值所指定的阈值是,当超过该阈值时因为输出不能像SM正在产生的那样快地消耗,仲裁器可以认为特定输出目的地停止SM(可以分别为输出到ISBE存储器的每个阿尔法管线、贝塔管线,以及输出到PROP/GPM的像素管线(例如,分别为PE_ALPHA_OUT_STALL, PE_BETA_OUT_STALL, GPM_PIX_OUT_STALL)单独指定该参数;该参数可以被计算作为等待用于复制到所分配的ISBE的数量的ISBE数量与仍然被活跃线程束使用的ISBE数量之间的比率,或者作为已经被计算但是等待排放到仍由像素线程束使用的像素寄存器的像素寄存器之间的比率)。图形线程束占用阈值表示线程束资源的图形贪婪阈值(例如,GFX_WARP_OCCUPANCY),当高于该阈值时仲裁器可以决定启动计算。图形寄存器文件占用阈值表示寄存器文件资源的图形贪婪阈值(例如,GFX_RF_OCCUPANCY),当高于该阈值时仲裁器可以决定启动计算。图形ISBE占用阈值表示ISBE的图形贪婪阈值(例如,GFX_ISBE_OCCUPANCY),当高于该阈值时仲裁器可以决定启动计算。可以将每个占用计算为在可配置平均窗口上的用于DPC的瞬时计算线程束占用的积分。

[0154] 与图形相关的策略参数还包括指定图形工作的优先级别的图形优先级(这是关于每个计算工作项数据结构内的驱动器所设置的计算优先级),以及在计算贪婪模式下由仲裁器授予图形时在MPC中保留的子项的数量或其他指定工作量。默认值被设置为较小的值,例如1,以便在计算贪婪模式下让图形涓流进入(trickle in)。

[0155] 设置为阈值的与计算相关的策略参数可以包括但不限于:输入饥饿计算阈值、计算线程束占用阈值、计算寄存器文件占用阈值、以及计算共享存储器占用阈值。输入饥饿计算阈值(例如,INPUT_STARVED_COMP),在这一阈值下仲裁器可以认为计算正在使SM饥饿(可以计算为一个二进制输出的积分,在不存在平均窗口上的否定确认时检查任务同步是否处于待处理状态)。计算线程束占用阈值(例如,COMP_WARP_OCCUPANCY)指示计算贪婪占用阈值,当高于该阈值时仲裁器可以决定启动图形。计算寄存器文件占用阈值(例如,COMP_RF_OCCUPANCY)指示用于寄存器文件资源的计算贪婪占用阈值,当高于该阈值时仲裁器可以决

定启动图形。计算共享存储器占用阈值(例如,COMP_SHM_OCCUPANCY)指示用于共享存储器资源的计算贪婪占用阈值,当高于该阈值时仲裁器可以决定启动图形。

[0156] 与计算相关的策略参数还包括计算优先级,其表示要为计算工作设置的优先级(例如,这是由每个计算工作项数据结构内的驱动器设置的优先级(例如,有时称为“QMD”))和计算MPC资源保留参数,其表示当处于图形贪婪模式时由仲裁器授予计算时在MPC中保留的CTA的数量(“协作线程阵列”(也是上面提到的协作组)或其他数量的计算工作)。将默认值设置为较小的值,例如1,以便在图形贪婪模式下让计算涓流进入(trickle in)。

[0157] 在一些实施例中,策略参数还可以包括平均窗口的大小,该平均窗口定义窗口持续时间,在该窗口持续时间上由MPC对SCG-SM仲裁器输入统计进行平均。MPC可以为提供给SCG-SM仲裁器的统计数据实现滑动窗口平均值。在一些实施例中,该窗口可以实现为样本的总数而不是时间窗口。策略参数还可以包括样本持续时间,其定义用于窗口的积分的采样率。

[0158] 在操作804中,应用程序可以将一个或更多个图形工作项和/或一个或更多个计算工作项的流中的指令和工作项发送到GPU。工作项可以适当地排列在由GPU访问的图形队列和计算队列中。可以将与工作项相关联的指令发送到GPU。

[0159] 图形工作项和计算工作项的传输可以在CPU上的整个应用程序的执行中连续,并且可以使图形在系统的显示器上渲染。

[0160] 图9示出了根据一些示例实施例的可由硬件调度器(例如,调度器708)执行的过程900的流程图。

[0161] 在进入过程900之后,在操作902-904中,调度器确定GPU的当前优先级模式。与图形队列中的下一个工作项和计算队列中的下一个项相关联的优先级中的较高者可以用于确定当前优先级模式。例如,如果图形队列中的下一个工作项的优先级高于计算工作队列中的下一个项的优先级,则将当前优先级模式设置为图形贪婪模式。或者,如果计算队列中的下一项具有比图形队列中的下一项更高的优先级,则将GPU设置为计算贪婪模式。

[0162] 对于图形工作项和计算工作项,可以由应用程序以相同的比例设置工作项的优先级,以便可以将它们相互比较。在一些示例实施例中,每个图形工作项和每个计算工作项与标度1-63中的优先级值相关联。根据一些实施例,图形工作项的优先级可以与指令流一致设置,并且可以将计算工作项的优先级设置在工作项本身中。

[0163] 在一些实施例中,图形优先级可以由应用程序或驱动器在内联图形命令中指定。可以在像素管线的顶部对图形命令设置优先级进行采样,使得它严格保持所有先前像素工作的顺序(但不严格按照后续顶点工作的顺序)。可以根据CWD正分发的当前网格的计算工作项中的字段来确定计算优先级。

[0164] 在操作902中,调度器检测每个图形队列和计算队列中的下一个项目的优先级,并且在操作904中,确定哪个优先级更高。

[0165] 如果确定图形优先级更高或相等,则在操作906中,将调度器设置为在图形贪婪模式下操作。在图形贪婪模式下操作涉及从图形队列重复启动图形工作项,同时主要考虑图形统计信息以决定何时允许调度更多的计算工作。

[0166] 当处于图形贪婪模式时,在操作908中,调度器重复地从图形队列中选择下一个项目并启动它以在相关联的SM上执行。重复操作908中,直到图形队列为空,调度器转换出图

形贪婪模式,或者接收信号以启动一个或多个非贪婪工作项。如上所述,这种使一些非贪婪工作项(在这种情况下是计算工作项)“trickle”的信号可以由诸如仲裁器710的硬件仲裁器生成。在操作910中,确定信号(例如,计算启动命令)是否已从仲裁器接收以使非贪婪(在这种情况下,计算工作项)涓流进入。如果是,则过程900进行到操作914以启动一个或多个计算工作项。在图形贪婪模式下当涓流进入计算工作时启动的工作项数可以是基于策略参数的(例如,COMP_MPC_RESOURCE_RESERVE参数指定的CTA数量,通常设置为1,以便MPC仅启动1个CTA用于来自SCG-SM仲裁器的每个计算启动命令)。

[0167] 如果否,则在操作911中确定调度器是否仍处于图形贪婪模式。如果调度器仍处于图形贪婪模式,则过程900进行到操作908以启动更多图形工作项。如果调度器不再处于图形贪婪模式,则例如,过程900可以进行到操作914以继续以计算贪婪模式进行处理。

[0168] 如果在操作904中,确定计算优先级大于图形优先级,则在操作912中,将调度器设置为在计算贪婪模式下操作。在计算贪婪模式下操作涉及从计算队列重复启动计算工作项,同时主要考虑计算统计信息以决定何时允许调度更多图形工作。

[0169] 从图形贪婪模式变为计算贪婪模式可能涉及某些步骤。如果由于CWD的任务同步导致高任务优先级导致更改,则MPC可以刷新已分配了寄存器文件空间、线程束和ISBE存储器的VTG工作,刷新已分配了资源的像素工作,然后切换到计算贪婪模式。请注意,可能不需要刷新整个图形输入管线。或者,如果改变是由图形管线中图形优先级降低引起的,则将贪婪模式从图形切换到计算之前,MPC可以刷新优先级设置之前的所有图形输入。

[0170] 在一些实施例中,计算工作项可以从计算队列获得并以与图形工作项相同的方式启动。然而,在图7所示的实施例中,调度器以与图形工作项不同的方式调度计算工作项。在所实施例中,在确定启动计算工作项之后,在操作914中,调度器为要启动的计算工作项预留资源,并且在操作916中,调度器请求计算任务。

[0171] 在某些示例实施例中,调度器(或MPC中的其他组件)从CWD请求计算工作项。调度器可以随时请求特定数量的CTA(例如,在任务同步时,当SCG-SM仲裁器改变决定时,或者当CTA完成时)。CWD将启动这些CTA或通过CTA启动否认(CTA launch Nack)回应MPC。如果它有更多要发送的内容,CWD将光栅化CTA。如果在ComputeInGraphics模式下(例如,“ComputeInGraphics”指的是在图形贪婪模式下请求涓流进入计算),CWD将在其无法启动MPC请求的CTA时发送CTA启动否认。一旦MPC请求CTA,就会保留SM中的资源,并且可以保证CWD启动的任何CTA都会快速加载到SM上。在ComputeInGraphics模式下,MPC可以基于SCG-SM仲裁器异步请求CTA。

[0172] CWD到MPC协议在概念上使MPC成为主设备,并让MPC从CWD请求CTA。在ComputeInGraphics模式下,协议允许SCG-SM仲裁器(定位于MPC或位于GPM中)在其无论何时决定启动CTA时不需要MPC内的本地CTA队列。使用ComputeInGraphics模式下的协议,CWD可以根据来自MPC的CTA请求在SM之间进行负载平衡,这已经考虑了在这些TPC上运行的图形所占用的资源。

[0173] 使用CWD到MPC协议,MPC可以通过三种方式从CWD请求CTA启动:在任务同步时使用CTA报告数据包,在先前的CTA在SM上完成时使用CTA完整数据包,或者例如当SCG仲裁器告诉MPC将更多SM资源用于计算而不是图形时使用新的CTA请求数据包。如果SCG-SM仲裁器请求MPC将更少的SM资源用于计算而不是图形,则MPC可以等待那些旧请求返回CTA启动或被

CWD否认,或者可以在现有CTA完成时请求0个额外CTA启动。这会慢慢将资源转移回图形。

[0174] 当MPC请求来自CWD的CTA启动时,MPC必须为计算保留所需的SM资源,直到这些启动实际发生或由CWD否认。这些资源未分配给同时发生的图形线程束启动,并保留用于计算。新的任务同步可能导致所有先前预留隐式否认。

[0175] SCG-SM仲裁器可以决定何时可以启动计算,并且可以通知MPC计算它需要从CWD请求的CTA的数量。MPC可以向CWD发起CTA请求,其中请求的CTA数量取决于计算应使用的SM资源的什么部分。MPC可以为这些CTA预先分配SM中的空间,并且可以仅在从例如CWD接收否认时释放资源。因此,当来自CWD的CTA到达时,它们可以配置成总是赢得MPC选择器内的选择以用于共享存储器分配和寄存器文件/线程束分配。MPC内部可能没有CTA排队或停止。

[0176] 在操作918中,确定是否接收或拒绝计算任务。如果拒绝对计算任务的请求(例如,如上所述,接收到否认),则在操作920中,释放预留的资源。

[0177] 资源预留可以考虑图形队列中的不同图形元素类型。例如,可能将用于VTG线程束的ISBE和用于计算线程束的共享存储器分配到SM中的相同暂存存储器,因此将争夺空间,但像素侧的TRAM将具有开创的空间。因此,至少在一些实施例,可以统一MPC中的ISBE和共享存储器资源跟踪以使用相同的资源计数器。对于寄存器文件和线程束分配,资源跟踪计数器可以统一用于计算和图形。

[0178] 如果对计算任务的请求导致(例如,从CWD)分配一个或多个任务,则在操作922中,启动任务。

[0179] 在资源释放步骤920或启动步骤922之后,在操作924中,确定仲裁器是否已经发信号通知非贪婪项(在这种情况下,图形工作项)涓流进入以启动。如果是,则过程900进行到操作908以启动来自图形队列的一个或多个图形工作项。

[0180] 如果否,则在操作925中确定调度器是否仍处于计算贪婪模式。如果调度器仍处于计算贪婪模式,则过程900进行到操作914以启动更多计算工作项。如果调度器不再处于计算贪婪模式,则例如,过程900可以进行到操作908以继续以图形贪婪模式进行处理。在计算贪婪模式下当涓流进入图形工作时所启动的图形工作项的数量可以是由基于策略的参数所控制的(例如,GFX_MPC_RESOURCE_RESERVE参数指定的子项或批次的数量,通常设置为1使得MPC将仅为来自SCG-SM仲裁器的每个图形启动命令启动1个子项(或其他图形工作单元))。

[0181] 图形与计算的优先级也会影响SM中的指令调度。MPC可以在线程束启动期间将2位优先级的线程束设置到SM上:根据从MPC传送的优先级,在图形贪婪模式中,SM可以以下优先级调度:VTG>PIX>COMP(例如,顶点图形>像素图形>计算);以及在计算贪婪模式下,SM可以以下列优先级调度:COMP>VTG>PIX。

[0182] SCG-SM仲裁器在计算/图形之间进行仲裁,并且可以向MPC内的选择器提供输入,选择器在工作类型之间进行选择(例如,简单地图形和计算,或顶点图形、像素图形和计算)。MPC可以提供一些反馈以使仲裁器能够做出正确的决定。一旦SCG仲裁器决定启动计算,MPC就会向CWD生成CTA请求,并为SM上的那些CTA分配空间。可以进行CTA的预仲裁以防止在MPC中排队CTA并保证一旦CTA到达,总是选中它们并且在没有停顿的情况下将其发送到SM。尽管如何对事物进行仲裁,但在ComputeInGraphics模式下,图形和计算可能基本上会竞争以下SM资源:TRAM可能专用于像素工作;VTG和计算可能竞争ISBE/共享存储器的共

享池;以及VTG、PIX和计算将竞争本地寄存器文件(LRF)和线程束槽(slot)。在ComputeInGraphics模式下,MPC还可以考虑在SM上运行的图形用于CTA槽计算,然后向CWD发送CTA请求,并且它可以根据SCG-SM仲裁器决策异步地请求CTA。可以允许来自VTG、像素和计算的线程束在SM上同时执行。

[0183] 图10示出了根据一些示例实施例的可由诸如仲裁器710的硬件仲裁器执行的过程1000的流程图。

[0184] 在进入过程1000之后,在操作1002中,仲裁器(或其他硬件单元)监视与SM相关联的资源的预定的一组性能和/或利用度量。

[0185] 根据一些实施例,根据DPC对线程束ID、寄存器文件空间和ISBE存储器等的使用,MPC收集关于图形资源占用的统计数据。它还可以收集与图形输入是否在饥饿和图形输出是否停止有关的接口统计数据。这些接口统计数据 and 占用统计数据可能不是即时的,而是在一些可配置的时间段上积分的。

[0186] MPC可以配置成基于DPC对线程束ID、寄存器文件空间和共享存储器空间等的使用来收集关于计算资源占用的统计数据。它还可以收集与计算输入是否正在饥饿有关的接口统计数据。这些接口统计数据也可能不是即时的,而是(类似于图形统计的情况)在一些可配置的时间段上积分的。

[0187] 根据所监控的统计数据的决策取决于活动优先级模式。因此,在操作1004中,确定调度器和仲裁器是处于图形贪婪模式还是处于计算贪婪模式。

[0188] MPC将这些采样统计数据与软件设置的相应策略控制进行比较。根据一些实施例,存在与每个采样统计数据相对应的软件阈值。

[0189] 根据一个实施例,MPC可以为SCG-SM仲裁器逻辑提供以下可用的设置和统计数据: PRIORITY_TYPE, INPUT_STARVED_ALPHA, INPUT_STARVED_BETA, INPUT_STARVED_PIX, INPUT_STARVED_COMP, PE_ALPHA_OUT_STALL, PE_BETA_OUT_STALL, GPM_PIX_OUT_STALL, COMP_WARP_OCCUPANCY, COMP_SHM_OCCUPANCY, COMP_RF_OCCUPANCY, GFX_WARP_OCCUPANCY, GFX_ISBE_OCCUPANCY和GFX_RF_OCCUPANCY。

[0190] SCG-SM仲裁器逻辑可以计算下列每个DPC: `isGraphicsGreedy = (PRIORITY_TYPE == GFX); isAlphaInputEmpty = (INPUT_STARVING_ALPHA < INPUT_STARVING_ALPHA_THRESHOLD); isBetaInputEmpty = (INPUT_STARVING_BETA < INPUT_STARVING_BETA_THRESHOLD); isPixInputEmpty = (INPUT_STARVING_PIX < INPUT_STARVING_PIX_THRESHOLD); isComputeInputEmpty = (INPUT_STARVING_COMP < INPUT_STARVING_COMP_THRESHOLD); isAlphaOutputStalled = (PE_ALPHA_OUT_STALL > PE_ALPHA_OUT_STALL_THRESHOLD); isBetaOutputStalled = (PE_BETA_OUT_STALL > PE_BETA_OUT_STALL_THRESHOLD); isPixOutputStalled = (GPM_PIX_OUT_STALL > GPM_PIX_OUT_STALL_THRESHOLD); isGraphicsEmptyOrStalled = (isAlphaInputEmpty || isAlphaOutputStalled) && (isBetaInputEmpty || isBetaOutputStalled) && (isPixInputEmpty || isPixOutputStalled); isGraphicsGreedyEnough = ((GFX_WARP_OCCUPANCY > GFX_WARP_OCCUPANCY_THRESHOLD) && (GFX_RF_OCCUPANCY > GFX_RF_OCCUPANCY_THRESHOLD) && (GFX_ISBE_OCCUPANCY > GFX_ISBE_OCCUPANCY_THRESHOLD)); isComputeGreedyEnough = ((COMP_WARP_OCCUPANCY > COMP_WARP_OCCUPANCY_THRESHOLD) &&`

(COMP_RF_OCCUPANCY>COMP_RF_OCCUPANCY_THRESHOLD)&&(COMP_SHM_OCCUPANCY>GFX_SHM_OCCUPANCY_THRESHOLD))。

[0191] 如果处于图形贪婪模式,则在操作1006中,主要根据与图形相关的统计数据来评估所监视的统计数据,以确定是否可以在当前资源利用中找到漏洞。在操作1010中,使用当前统计数据确定是否已识别出漏洞。例如,通过上面的示例性统计数据,“!isGraphicsGreedyEnough||isGraphicsEmptyOrStalled”为TRUE或者“!isGraphicsGreedyEnough&&isGraphicsEmptyOrStalled”为TRUE的评估,可以代表在图形贪婪模式下确定的漏洞。

[0192] 如果存在“漏洞”,则在操作1012中,可以确定用于请求计算工作项的参数。例如,可以从策略配置中确定计算工作项(例如,COMP_MPC_RESOURCE_RESERVE)的涓流率。

[0193] 根据优先级是将MPC置于图形贪婪模式还是计算贪婪模式,并根据图形/计算统计数据如何比较相应的阈值,仲裁器选择接下来应该启动图形还是启动计算。

[0194] 例如,在上述计算之后,SCG-SM仲裁器逻辑可以决定以下列方式启动图形或计算:如果(isGraphicsGreedy):如果(isGraphicsEmptyOrStalled&&!isComputeInputEmpty)则启动计算;如果(!isGraphicsGreedyEnough&&!isComputeInputEmpty)则启动计算。否则,调度器将继续启动图形。

[0195] 图形工作可以立即启动,因为它通常在MPC输入队列中等待处理。然而,在一些实施例中,计算工作不能立即启动,它必须首先与CWD协商。因此,当仲裁器授权计算时,MPC检查COMP_MPC_RESOURCE_RESERVE(NCTA)中指定的CTA数量是否匹配此TPC上的剩余空间。它将这个NCTA计数暴露给CWD,并为NCTA保留资源(由于我们希望在图形运行时CTA涓流进入空白空间,因此通常将NCTA的值设置为1以用于图形贪婪)。CWD必须通过启动多达N个CTA或通过在所有CTA已启动到其他TPCS后解除剩余预留(或由于某种原因从CWD解除调度任务)来响应。

[0196] 在操作1014中,仲裁器用信号(例如,启动命令)通知调度器以启动一个或更多个计算工作项。该信号可以指示要启动多少这样的计算工作项的数量或其他指示。

[0197] 如果在操作1004中,确定活跃优先级模式是计算贪婪模式,则在操作1008中,主要根据与计算相关的统计数据来评估所监控的统计数据,以确定是否可以在当前资源利用率中找到“漏洞”。

[0198] 在操作1016,确定是否已使用当前统计数据识别出“漏洞”。

[0199] 如果存在“漏洞”,则在操作1018中,可以确定用于请求图形工作项的参数。例如,可以从策略配置中确定图形工作项(例如,GFX_MPC_RESOURCE_RESERVE)的涓流率。

[0200] 例如,如果(isComputeGreedy):如果(isComputeInputEmpty&&!isGraphicsEmptyOrStalled)则启动图形;而如果(isComputeGreedyEnough&&!isGraphicsEmptyOrStalled)则启动图形。

[0201] 在操作1010或1016处确定为“否”之后,在操作1014或操作1020之后,过程1000可以终止。

[0202] 图11和12描述了与一些示例性实施例相关联的过程,在这些实施例中可以配置和使用多个调度配置文件。

[0203] 图11示出了过程1100的流程图,可以由诸如CPU的主处理器处的应用程序或其他

程序执行,以便配置一个或更多个调度配置文件和用于在调度配置文件之间切换的定时器。

[0204] 在操作1102中,程序配置一个或更多个调度器配置文件和相关联的策略。每个配置文件可以包括策略的不同组合和/或策略中的某些阈值的不同值。策略可以包括,例如,用于图形和用于计算的资源利用率的阈值,用于图形和用于计算的涓流率等。

[0205] 在操作1104中,程序可以配置定时器,其可用于在所配置的调度配置文件之间切换。

[0206] 图12示出了过程1200的流程图,可以在GPU中或者在一些实施例中在诸如MPC 702的MPC中执行,以在一系列所配置的调度器配置文件中切换调度器配置文件。

[0207] 软件(例如,在主处理器/CPU上执行的应用程序和/或驱动器)可以具有在整个帧中静态地设置大部分仲裁器阈值的选项,或者可以通过直接将阈值更改方法插入图形工作流程或计算工作流程来动态地调整它们与工作负载同步。

[0208] 此外,过程1200提供使用微码以异步方式动态改变仲裁器设置的能力。软件可以设置几组仲裁参数。每组参数称为仲裁器“配置文件”。每个配置文件包含如上定义的整套的图形调度参数和计算调度参数。它还可能包含图形优先级。可以从计算工作项数据结构确定计算优先级。每个配置文件还具有时间片值(例如,在Sysclk循环中)和有效位,其可以指示每个配置文件是否可用。在一些实施例中,微码可以支持多达8个配置文件。

[0209] 在操作1202中,配置文件计时器到期。当配置文件计时器到期时,在一些实施例中,可以将非阻塞中断插入到可以在包含仲裁器配置文件的存储器中获取数据结构的配置文件切换块(例如,在前端单元中)。

[0210] 在操作1204中,选择下一个配置文件。配置文件切换块可以找到下一个有效配置文件并更新(全部或仅一些)MPC中的调度参数。在操作1206中,根据所选择的下一个配置文件来配置策略。

[0211] 通过这种方式,仲裁器可以在没有任何软件干预的情况下即时重新配置。一个用例可能是在两个配置文件之间切换(toggle)。第一个配置文件可以是运行一定数量(例如,X个)周期的图形贪婪配置文件,第二个配置文件可以是运行另一个预定数量(例如,Y个)周期的计算贪婪配置文件。如果 $X=4*Y$,那么无论存在多少图形工作,这都将为计算提供20%的保证服务工作。当然,该用例仅仅是一个示例,并且可以设想在实施例中利用配置文件切换能力的许多其他方式。

[0212] 图13示出了根据一些示例实施例的空闲检测过程1300的流程图。当检测到当前活跃优先模式空闲时,可以操作过程1300以切换调度器的活跃优先模式。

[0213] 在进入过程1300之后,在操作1302中,检测到活跃的贪婪模式已变为空闲。例如,在图形贪婪模式下,调度器可以检测到图形队列(例如,阿尔法队列、贝塔队列或像素队列中的任一个)为空。

[0214] 在操作1304中,可选地,设置滞后计时器,并且将过程1300延迟一些可配置的时间间隔。如果设置了一个滞后计时器,在滞后计时器到期之后,则过程1300进行到操作1306。

[0215] 在操作1306中,确定活跃优先级模式是否仍然是空闲的,并且如果是,则在操作1308处,改变活跃优先级模式。例如,如果当前活跃优先级模式是图形贪婪模式并且确定为空闲,则将活跃优先级模式切换为计算贪婪模式。

[0216] 在一些示例实施例中,操作1302-1306可以包括监视每个用于计算队列和图形队列的位(bit)的MPC,其中每个位具有滞后,使得它们将随时间缓慢地指示为空闲而不是瞬时。此外,为了切换活跃优先级模式,除了确定活跃优先级模式是空闲之外,还必须确定当前非贪婪队列不为空。

[0217] 如果从图形贪婪切换到计算贪婪是由空闲检测逻辑引起的,那么根据定义,图形管线是空的。在这种情况下,MPC可以立即切换到计算贪婪,但可能需要启动一个新的CTA请求,其中包含所有可用的CWD空闲槽,以便它可以接收CTA启动。如果从计算贪婪切换到图形贪婪是由空闲检测逻辑引起的,那么根据定义,计算管线是空的。在这种情况下,MPC可以立即切换到图形贪婪并开始处理图形工作。

[0218] 其他SCG-SM仲裁器类型

[0219] 在一些实施例中,不同类型的SCG-SM仲裁器可以用作替代硬件仲裁器,每个仲裁器具有其自身的优点和缺点:位于MPC中的循环仲裁器,位于MPC中的DPC级SCG-SM仲裁器(上面描述为“非平衡”仲裁器),位于GPM中的平衡GPC级SCG-SM仲裁器从而使其具有对多个TPC的控制,以及GPM中的动态分区SCG-SM仲裁器也可以使其具有对多个TPC的控制。

[0220] 使用循环仲裁器,仲裁器决定以循环方式针对诸如阿尔法、贝塔和像素队列之类的图形类型启动计算。当轮到计算时,循环仲裁器可以启动一个(或其他预定数量的)CTA。当轮到图形时,取决于图形仲裁逻辑在阿尔法、贝塔和像素工作中做出什么决定,循环仲裁器可以启动1批,或者任务,或者子项。如上面关于非平衡仲裁器所描述的,计算和图形启动可以是不同的。

[0221] 例如,假设在各个队列中有无限的计算和图形工作流,SM可以容纳多达20个工作单元,每个图形请求需要2个单元,并且每个计算请求需要5个单元,循环仲裁器授权的工作序列可以是:G2、CS、G2、CS、G2,不再满足计算,因此为G2、G2。当线程束完成时,将授权更多工作。

[0222] GPM内部平衡的GPC级SCG-SM仲裁器试图考虑整个GPC中SM的工作负载,并平衡工作。更具体地说,该仲裁器位于GPM的中央,并考虑许多SM来决定何时启动非贪婪工作。

[0223] 此仲裁器聚集来自许多SM的反馈/统计数据,而不是单独查看它们,以通过与策略指定的阈值进行比较来确定资源利用率的漏洞。一旦仲裁器决定运行非贪婪工作,该决定将广播给GPC中的所有SM。这种工作分配方式可能会导致SM之间的贪婪工作更加平衡。

[0224] 该平衡仲裁器的策略控制可以包括上面关于非平衡仲裁器描述的策略控制的一些或全部。该仲裁器的策略控制还可以包括SCG_GRANULARITY_SEL,其编程为选择仲裁器动态分配的粒度(例如,SCG_GRANULARITY_SEL=SUB_SM意味着SCG仲裁器将在计算和图形之间以子SM粒度动态分配/解除分配);BALANCE_TPC_IN_GPC,其以布尔值设置,‘True’通知仲裁器平衡GPC中的所有DPC(‘False’表示让DPC独立);GFX_TPC_OCCUPANCY_MIN,表示用于图形的保证的最小DPC分配;以及COMP_TPC_OCCUPANCY_MIN,表示用于计算的保证的最小分配DPC。

[0225] GPM内部的动态分区SCG-SM仲裁器是另一个仲裁器,它尝试在GPC上跨多个SM平衡工作。与平衡仲裁器类型一样,此仲裁器也位于中心位置,并在多个SM之间聚集输入。此仲裁器以SM粒度动态仲裁资源,而不是以线程束粒度在计算和图形之间进行协商。当在图形和计算之间切换SM时,该仲裁器不会导致耗尽(drain)互锁导致利用不足。相反,当SM正在

从另一种类型开始工作时,可以同时发生一种类型的工作的耗尽。该仲裁器可以提供一种方法,基于工作负载的要求(通过动态硬件仲裁)将SM从一种类型无缝地转换到另一种类型,而无需花费周期进行显式消耗。

[0226] 该平衡仲裁器的策略控制可以包括上面关于非平衡仲裁器描述的一些或所有策略控制。该仲裁器的策略控制还可以包括:SCG_GRANULARITY_SEL,将其编程为选择仲裁器动态分配的粒度(例如,SCG_GRANULARITY_SEL=TPC意味着SCG仲裁器将在使用子SM分配之前首先在计算和图形之间动态分配/解除分配DPC);BALANCE_TPC_IN_GPC,其始终设置为“True”(例如,如果“False”则为维护(Assert));TPC_MASK_MIXED,其是每个GPC的DPC的固定掩码,它将同时运行图形和计算;MIN_TPC_MASK_GFX,其是对图形做出保证的每个GPC中的最小DPC数量(不包括混合DPC);MIN_TPC_MASK_COMP,其是对计算做出保证的每个GPC中最小DPC数量;和CURR_GFX_EN[numTPCperGPC],其是仲裁器按照MPC粒度发送的输出请求,并通知MPC何时切换到图形以及何时切换到计算。

[0227] 仲裁器内部状态可以包括CURR_GFX_TPC_MASK,其是跟踪SCG-SM仲裁器将GPC中的哪些DPC分配给图形的状态;以及CURR_COMP_TPC_MASK,其是跟踪SCG仲裁器将GPC中的哪些DPC分配计算的状态。

[0228] 示例性用例

[0229] 可以根据工作负载要求以各种方式控制上述调度技术和配置文件时间分片技术。可以考虑控制调度和时间切片,例如,与3D/计算工作同步或与3D/计算工作异步。示例实施例为应用程序提供了对PPU中的处理进行这种控制的能力。

[0230] 在一些方面,可以与3D/计算工作同步地改变调度配置文件。例如,软件可以以与管线中的工作同步的方式,经由图形工作项而插入新的调度参数。管线中的工作项可能会覆盖MPC中的当前配置文件。与3D/计算工作同步的控制可能涉及仅改变图形优先级和计算优先级,或者可能涉及所有(或更大的一组)配置文件参数。

[0231] 在与3D/计算工作同步地改变调度配置文件的一些示例中,可以对MPC中的所有调度参数进行静态设置,并且可以仅改变图形和计算优先级。如上所述,计算优先级可以在构建时附加到每个计算工作项数据结构,以及图形优先级可以由3D管线中的命令提供。更改优先级会影响调度器是以图形贪婪模式还是计算贪婪模式运行。例如,驱动器可以将图形优先级稳定保持在16并将低优先级计算工作项标记为8,将高优先级计算工作项标记为24。当仅存在图形和低优先级计算工作时,仲裁器将以图形贪婪模式运行,当高优先级计算工作项到达时,它将强制CWD停止分配低优先级网格并切换到高优先级网格。MPC将看到增加的计算优先级并将仲裁器切换到计算贪婪模式,直到启动所有高优先级计算工作。

[0232] 软件可以将同步计算工作标记为比异步计算工作更高的优先级,从而使CWD倾向于启动同步计算工作,这通常在更多3D或2D工作可以启动之前完成,以允许更多的同时计算和图形重叠。

[0233] 注意,图形管线可以包括三个连续阶段:阿尔法,贝塔和像素,并且在一些实施例中,在像素着色器阶段之前对图形优先级的改变进行解码。例如,在以下命令序列中:图形优先级=16;绘制A;图形优先级=32;绘制B,保证绘制A的像素工作以优先级16执行,并且保证绘制B的像素工作以优先级32执行。对相关VTG工作的影响不太清楚。保证绘制A的VTG工作以优先级16执行,但绘制B的VTG工作可以在16或32执行,具体取决于A的像素工作延迟

提高图形优先级命令的时间。然而,将图形优先级与像素着色器阶段联系可能是有益的,因为大多数游戏通常是以像素着色器为主。

[0234] 计算优先级通常在示例实施例中由CWD光栅化的一个计算工作项提供。由于计算管线只有一个阶段,即计算着色器阶段,因此不存在像图形那样的管线问题。MPC中的SCG-SM仲裁器可以将来自CWD的当前启动的计算工作项的计算优先级与像素管线中的图形优先级进行比较,即MPC中的SCG-SM仲裁器完全不知道多个计算工作项优先级。

[0235] 在与3D/计算工作同步改变调度配置文件的一些示例中,软件不仅可以改变优先级,还可以改变图形和计算调度参数。在示例实施例中,这可以与3D工作项流同步地完成,从而保证像素工作受到影响。软件可以提高某个绘制的优先级以确保它在图形贪婪模式下执行,或者它可以增加某个绘制的图形贪婪度(例如,使其非常贪婪)。

[0236] 相反,计算工作项流和计算工作项启动不一定在计算管线中同步。对计算调度参数的更改可能不会被尚未启动的先前计算工作项阻止,而是可能立即流向MPC。因此,如果软件想要保证第一计算工作项看到第一组参数而第二计算工作项看到第二组参数,则可能需要触发同步。

[0237] 软件使用SCG-SM仲裁器的另一种方法是通过多个配置文件和微码。以下是可以保存在存储器中的上下文缓冲区中的调度参数。软件可以创建预定数量(例如,8)的配置文件,并且每个配置文件可以包括有效位和时间片值。软件可能不通过图形命令发送任何调度参数,而是可以通过微码写入寄存器来提供它们。然而,软件可以通过命令提供图形优先级,或者这些也可以来自配置文件和微码写入。软件可以将N个有效配置文件编程到存储器中,并将每个配置文件与时间片值相关联。可以在激活每个配置文件时设置配置文件计时器,并且当计时器到期时,可以将硬件移动到下一个有效配置文件并重新启动计时器。通过硬件/微码写入MPC,可以更新计算和图形的调度参数,并且可选地,还可以更新图形优先级。这些写入可以同时广播到所有MPC,因为可以对它们全部进行相同编程。仍然可以从计算工作项数据结构确定计算优先级。通过这种方式,微码可用于在各种配置文件之间异步移动调度器。这可以用于允许一定量的保证服务用于计算,或者一定量的保证服务用于图形,或者同时计算和图形的任何其他期望组合。

[0238] 类似地,在一些实施例中,不是使用固定时间片在N个配置文件之间旋转,而是微码本身可以选择最佳配置文件。这里间隔计时器可以设置为某个固定值(例如,假设100K周期 $\approx 100\mu\text{s}$)。每当微码看到中断时,它可以读取芯片周围的寄存器以确定当前流型(regime)是帧缓冲器限制,L2限制,着色器数学限制,图形大量但计算少量等等。微码可以使用此信息来在N个预先计算的配置文件之间进行选择,这些配置文件针对各种应用程序流型(regime)进行调整。

[0239] 示例实施例提供对GPU和基于GPU的处理的许多改进。某些实施例的硬件调度器中的可编程性控制使得能够灵活地响应不同的工作混合以及由新游戏和其他应用程序施加的其他要求。硬件调度器的这种可编程性还允许计算和图形工作的不同混合,使得游戏开发者等能够探索新的应用程序和编程技术。

[0240] 在一些示例实施例中,计算工作负载的一部分,例如某些帧使用计算技术在帧结束时对图形数据进行后处理,可以在不延长帧处理时间的情况下以最高优先级通过执行图形来处理图形数据,并且在找到资源可用性的任何地方进行计算。

[0241] 示例实施例还可以加速由PhysX等实现的模拟等,PhysX是用于向游戏添加物理模拟(基于粒子的流体,类似生命的爆炸等)的库。当在GPU上运行时,PhysX通常形成计算工作的第二线程,独立于典型游戏中已存在的计算技术。PhysX计算工作也可以充分地涓流进入在图形贪婪模式下正处理的图形工作项之间。

[0242] 某些示例实施例可以允许CUDA工作负载有效地结合传统图形,这在某些深度学习或在游戏内等CUDA加速的其他应用程序中可能是有用的。硬件调度器还可以通过允许计算来启用光线跟踪,以使光线追踪优先于系统中运行的典型图形工作。由于硬件调度器的可编程性,某些示例实施例还可以通过降低必须专用于物理计算的帧时间的百分比,来为基于GPU的物理应用程序提供改进的可用性。某些示例实施例可以实现改进的虚拟现实应用程序。例如,将计算用于根据头部位置的最新采样修改最终帧图像,并且异步计算工作希望以最高优先级执行,从而其可以保证在下一个显示事件之前完成。

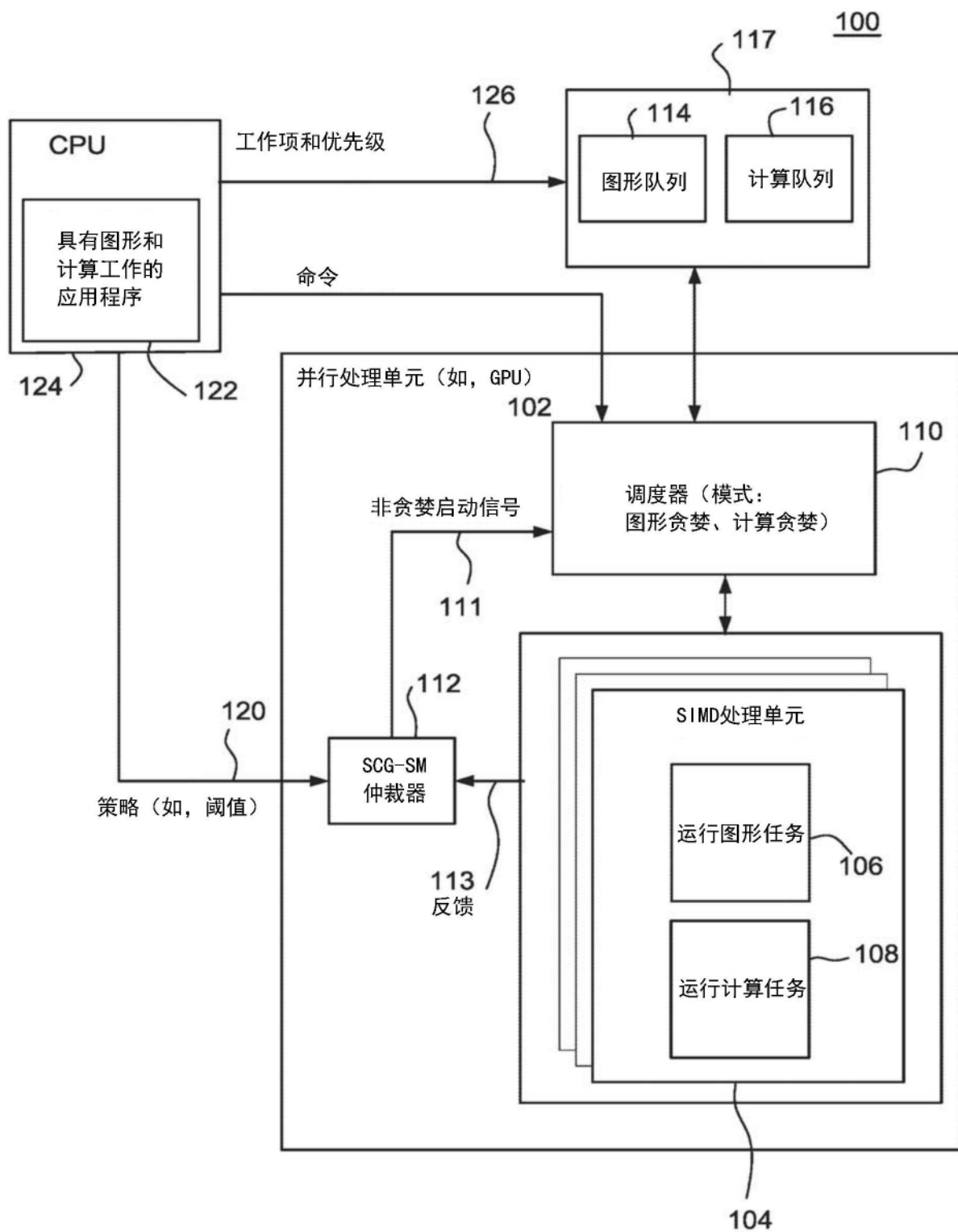


图1

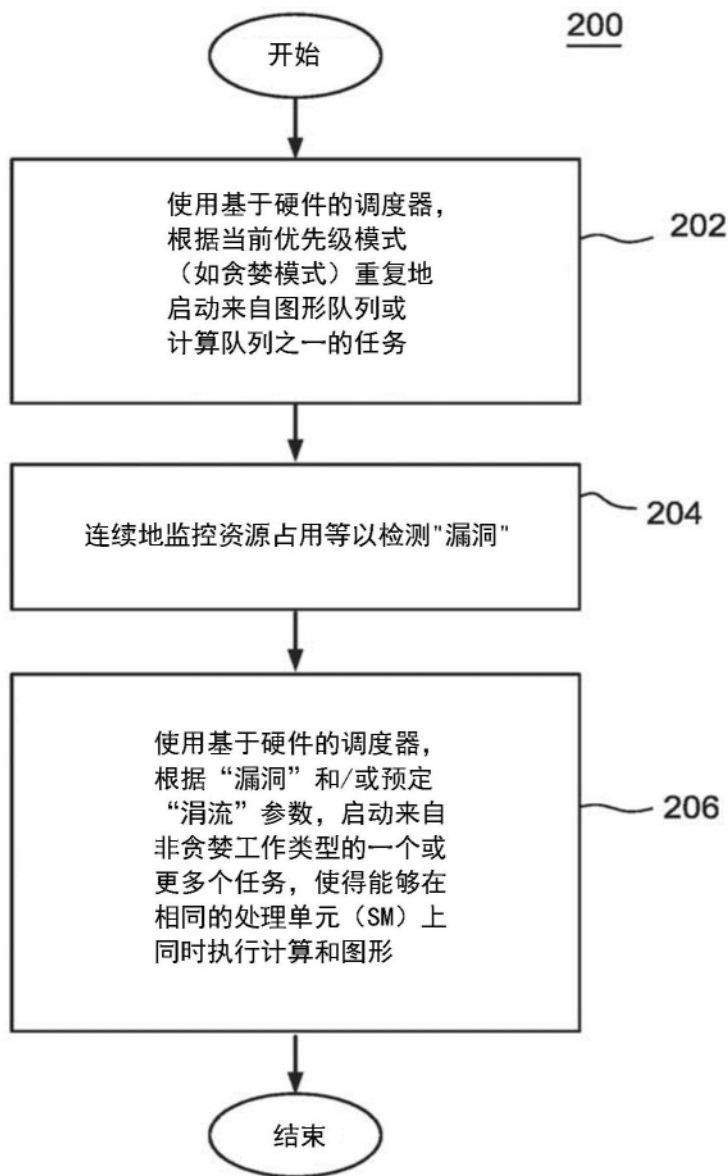


图2A

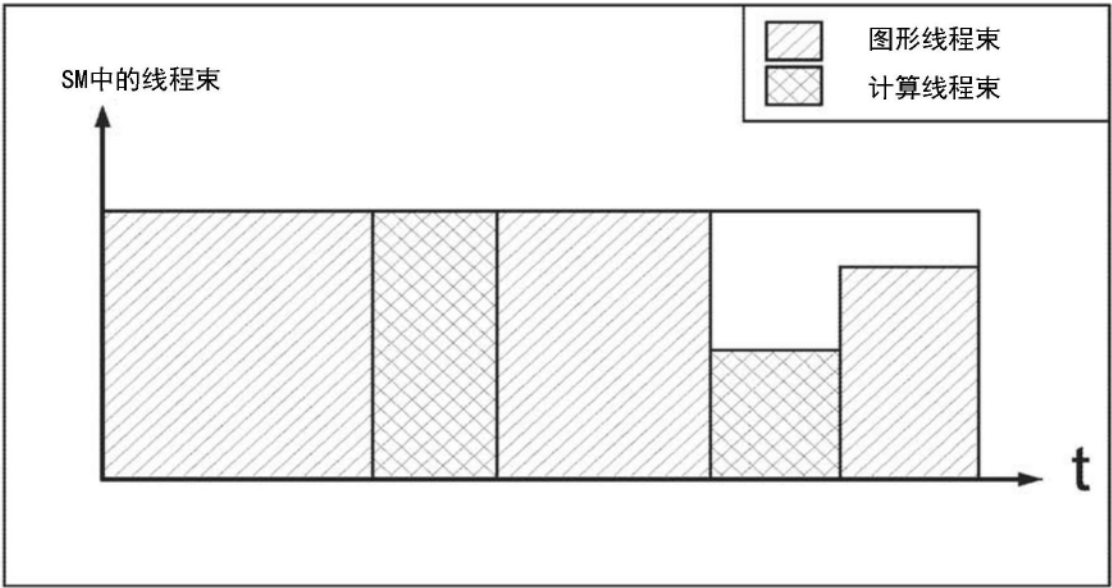


图2B

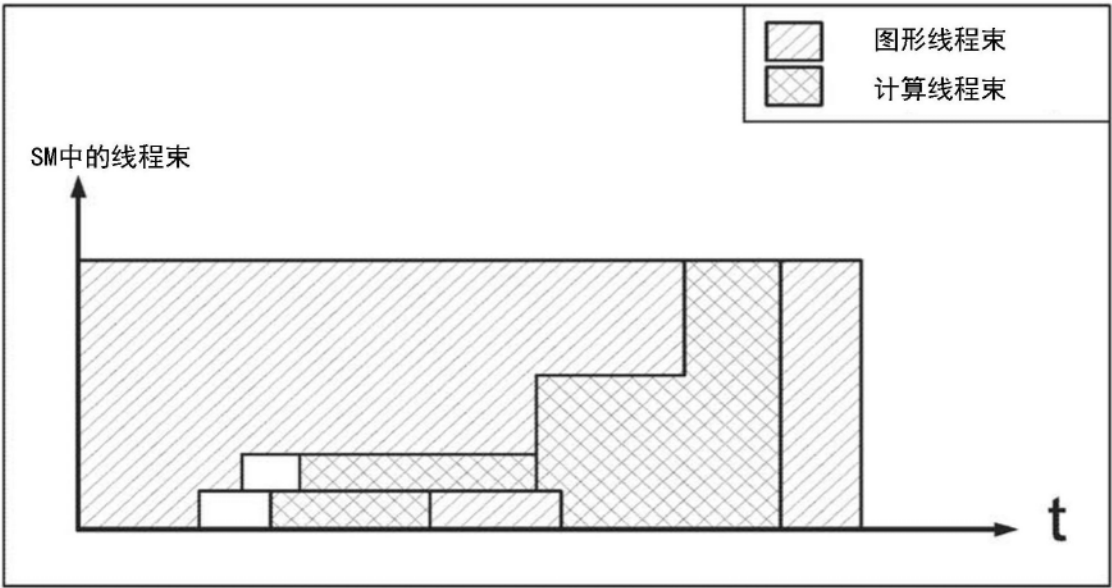


图2C

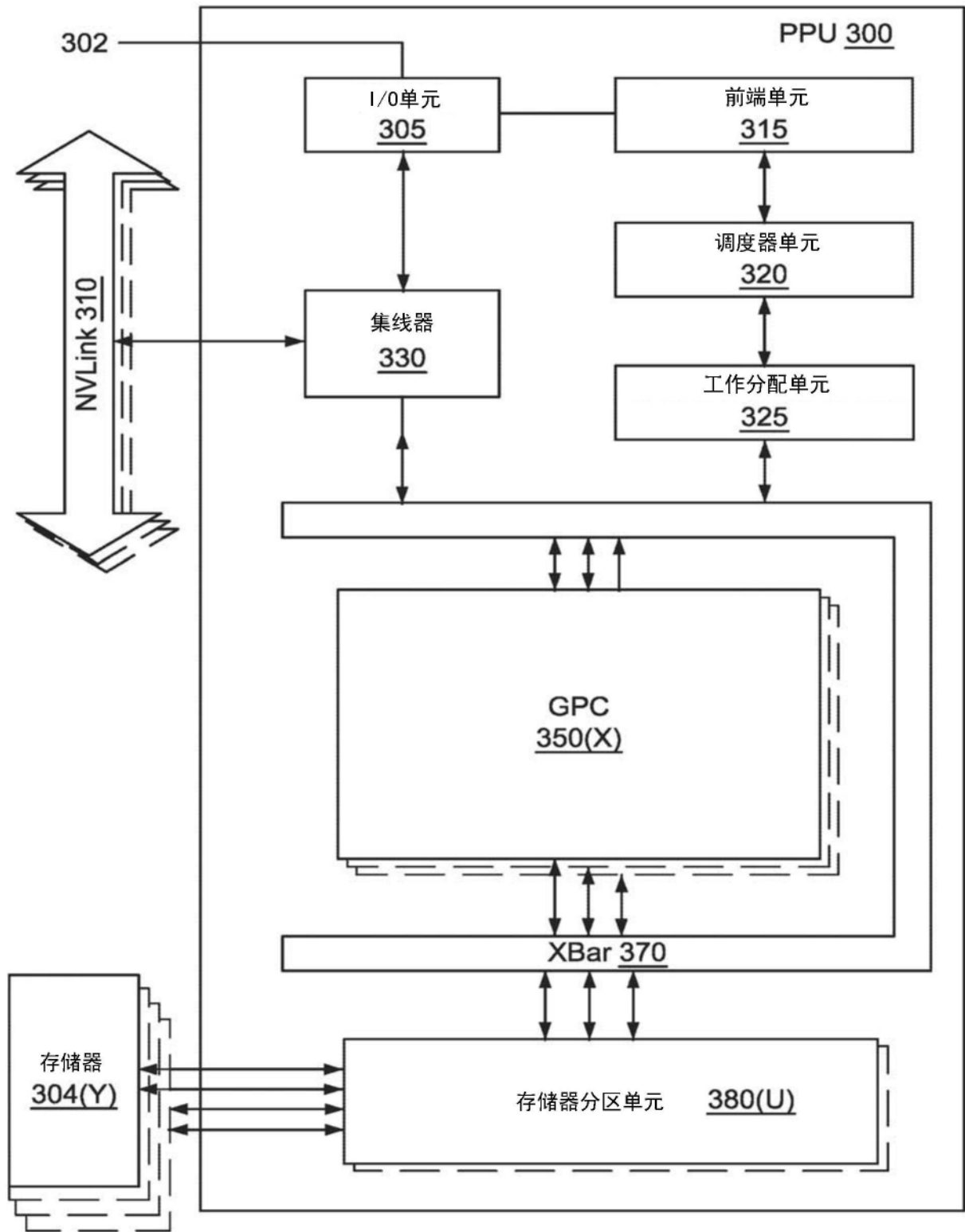


图3

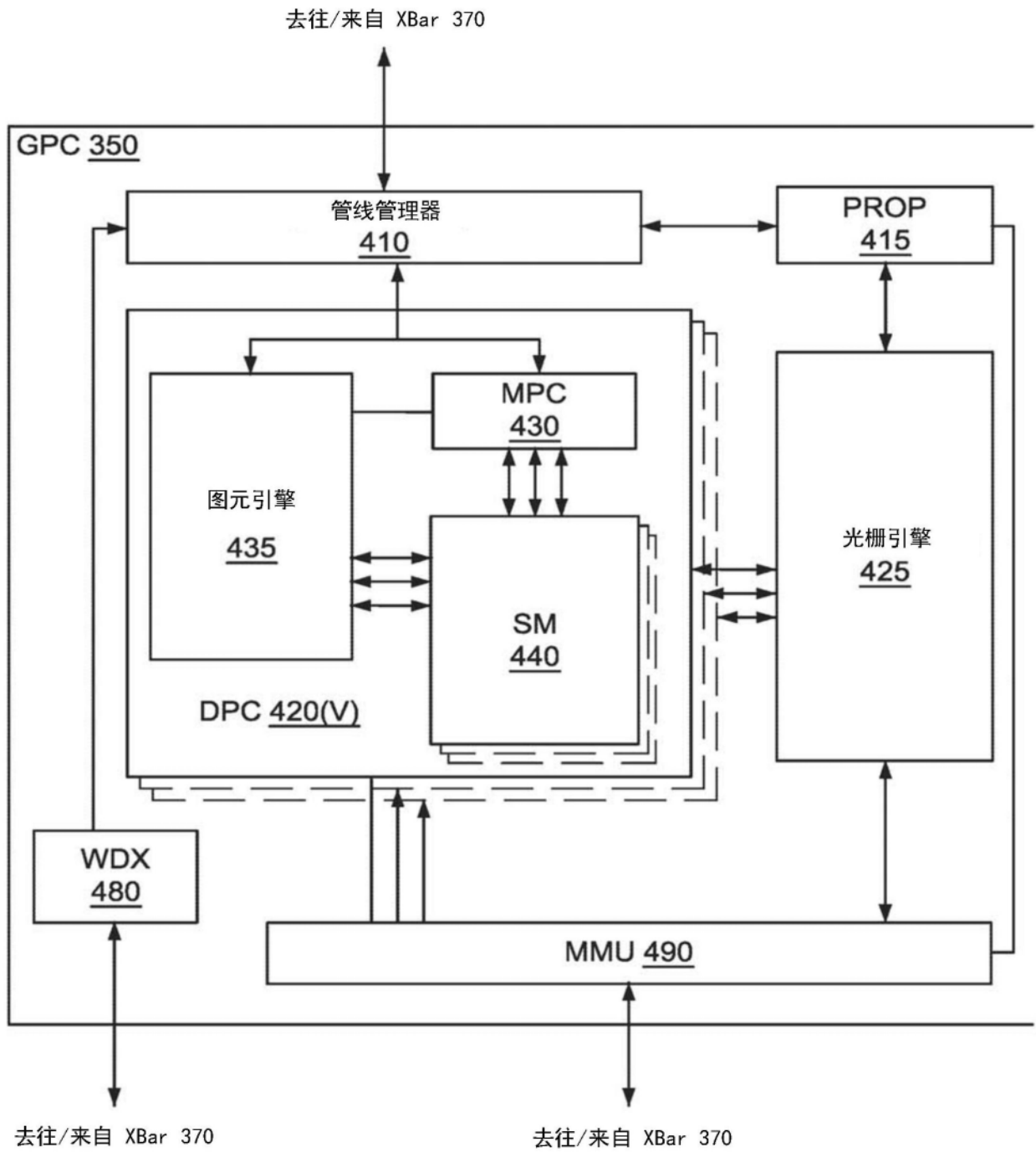


图4A

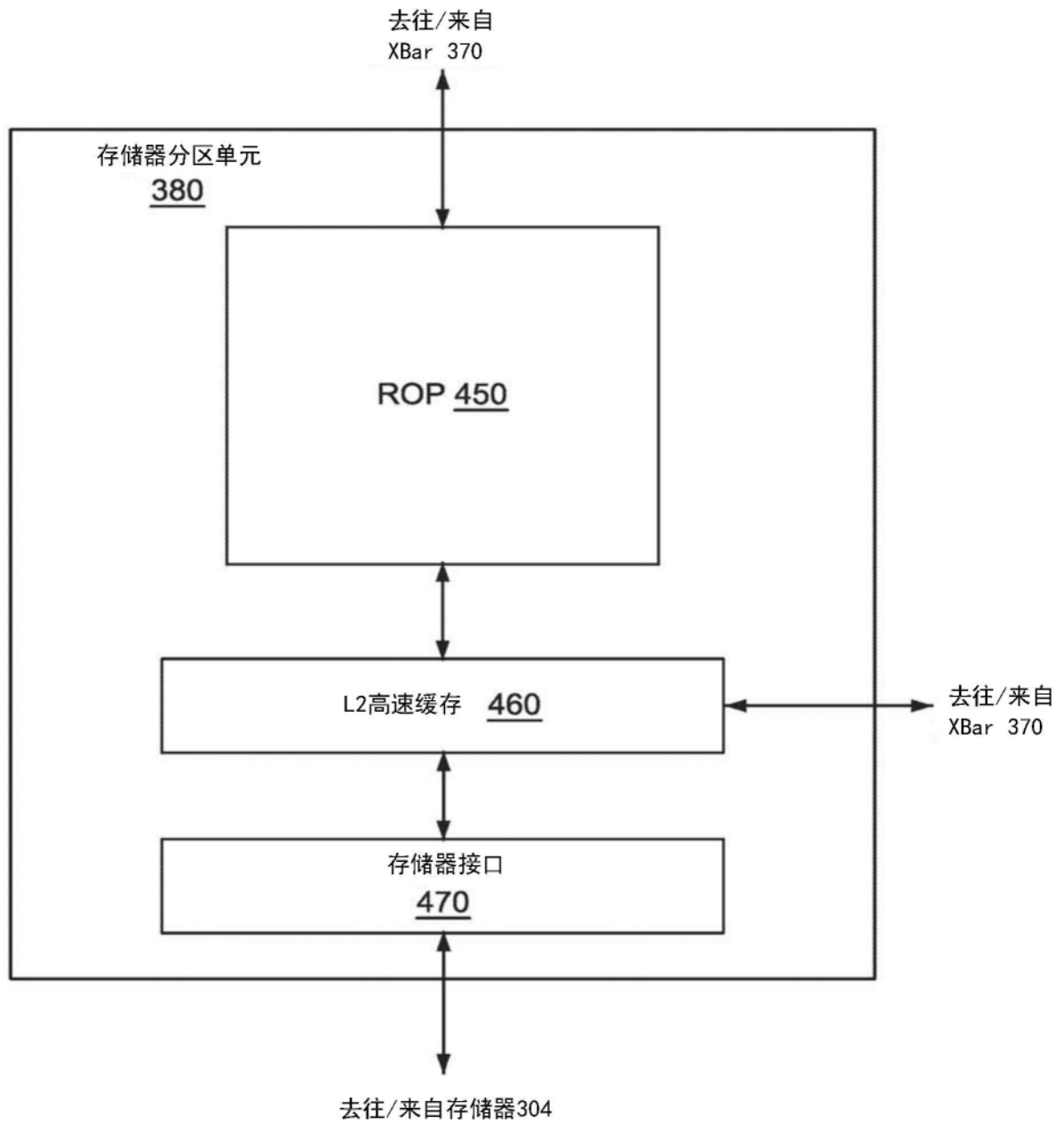


图4B

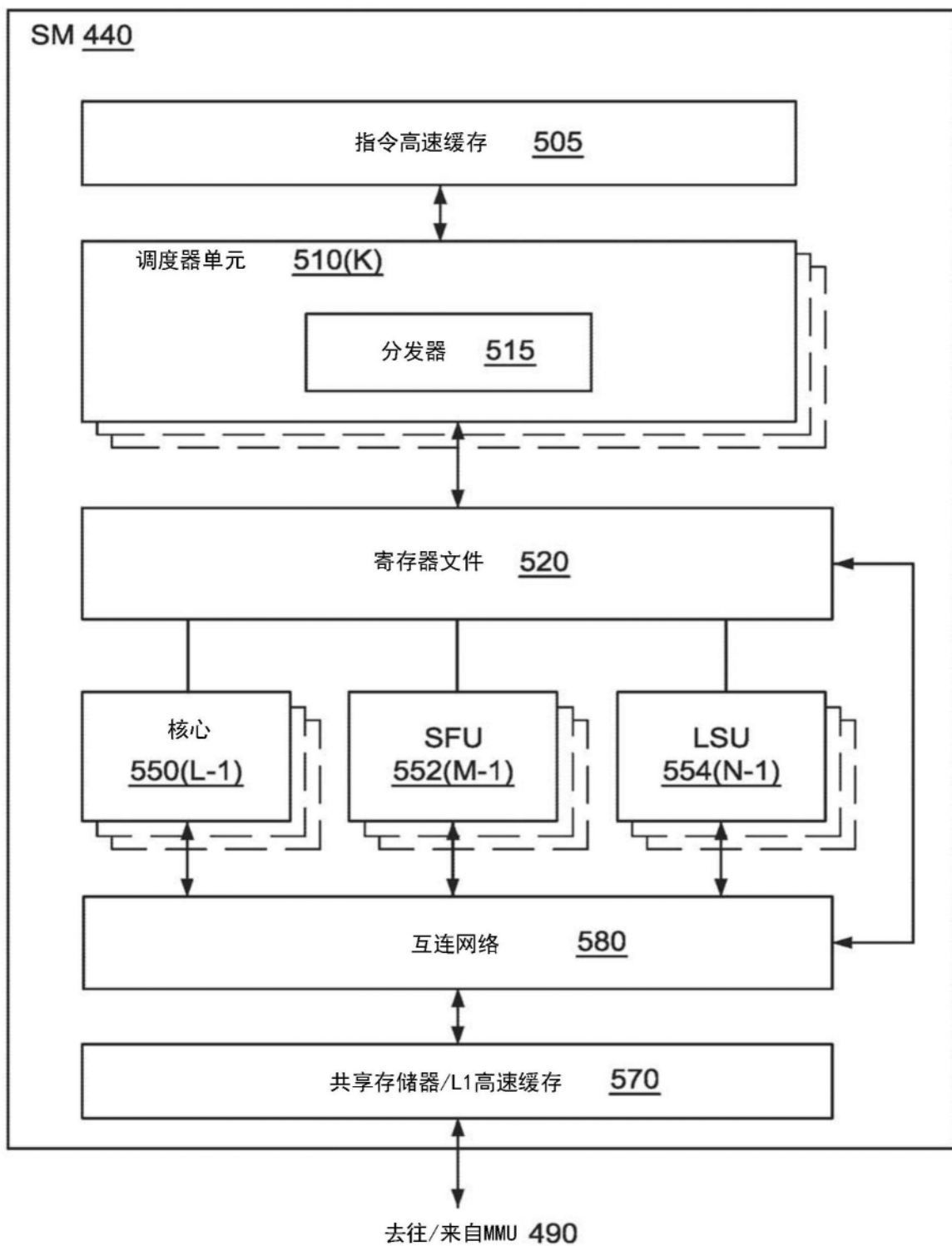


图5A

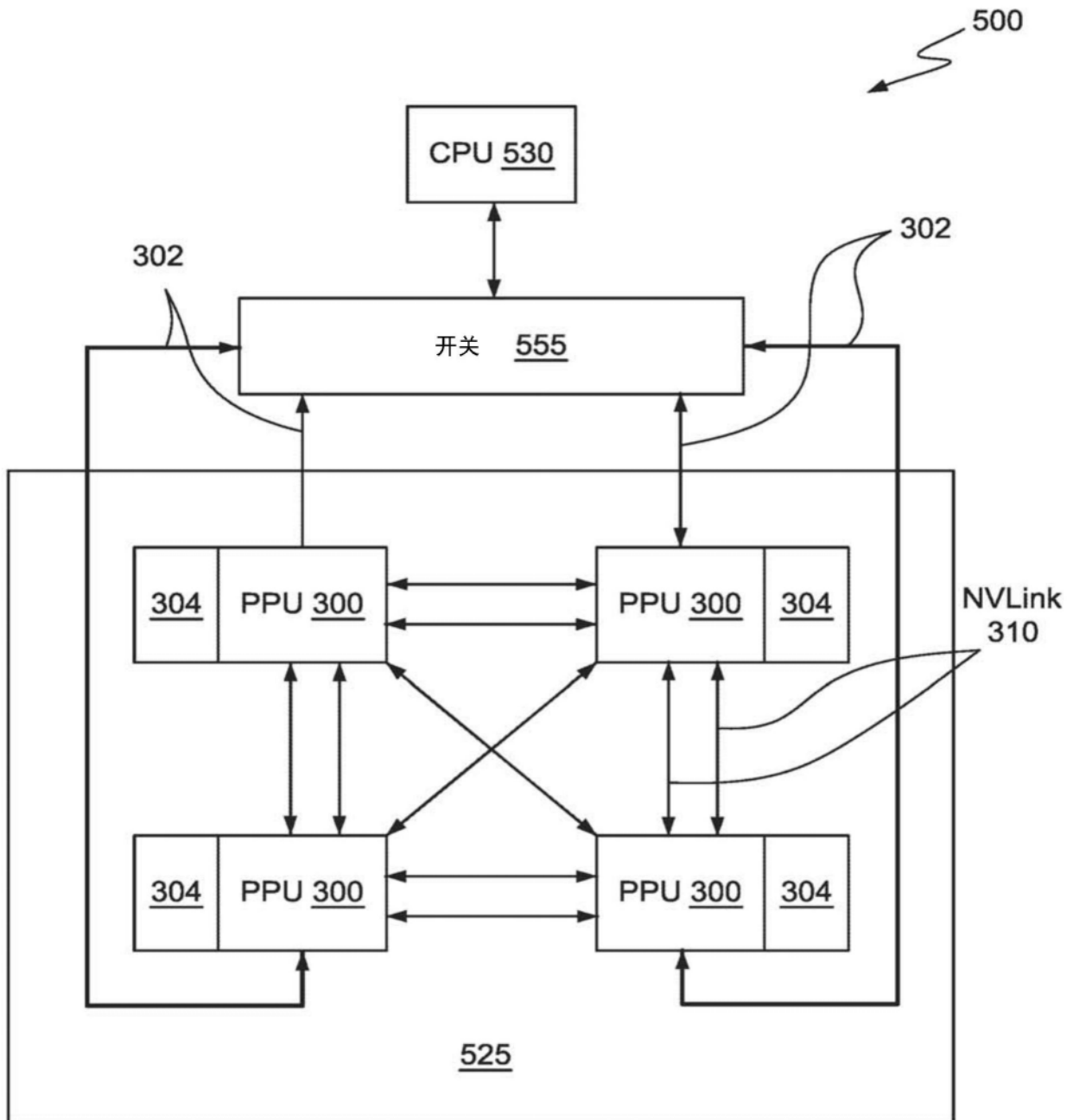


图5B

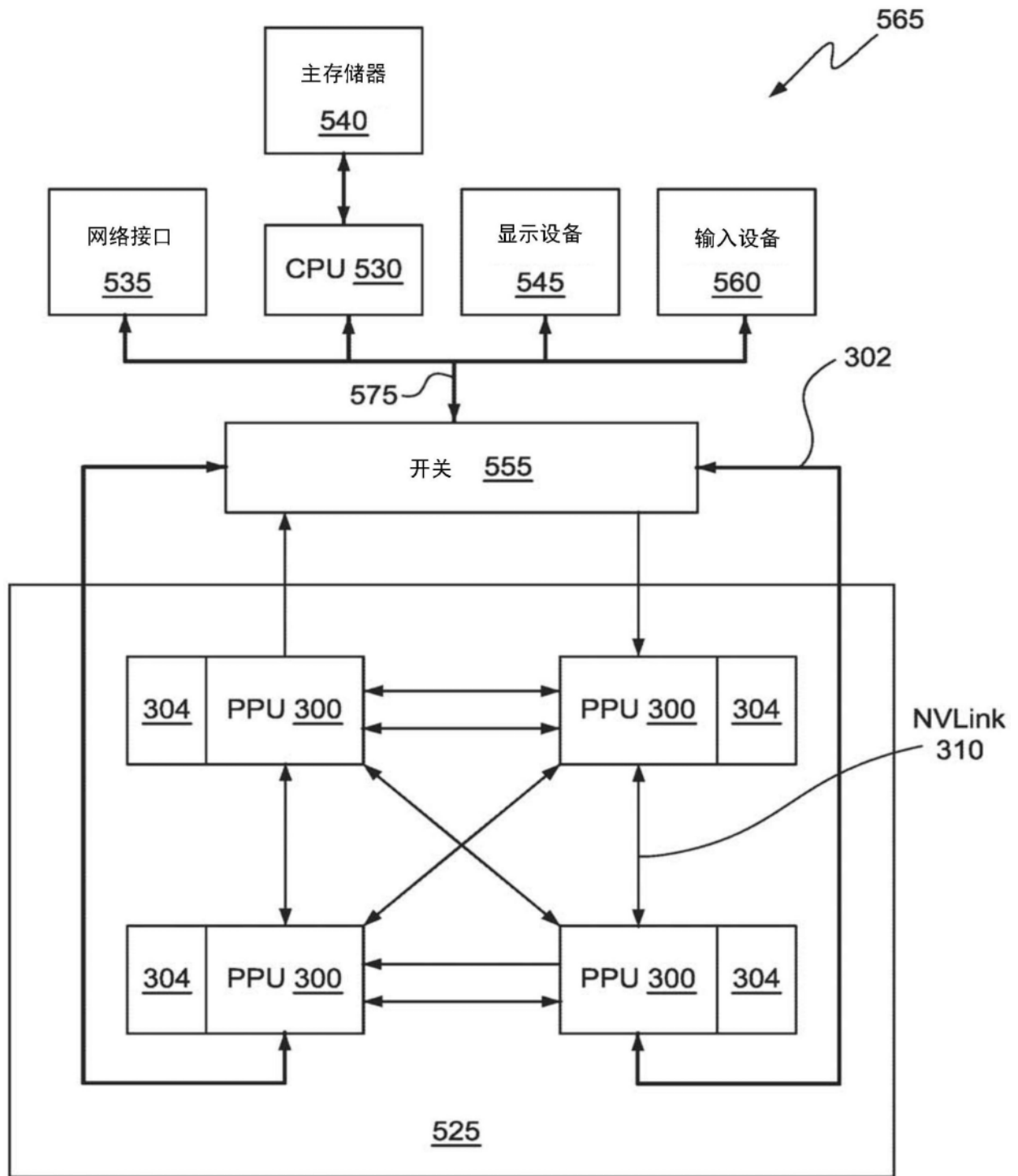


图5C



图6

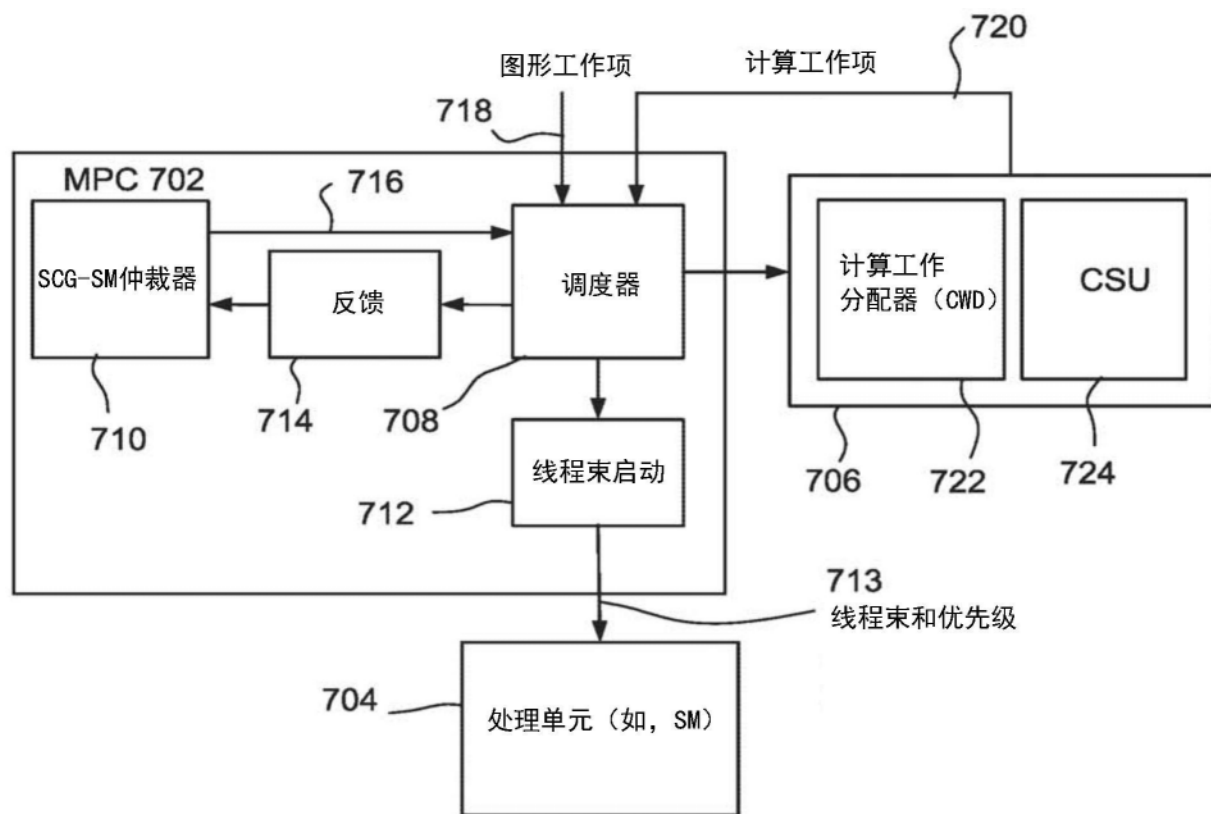


图7

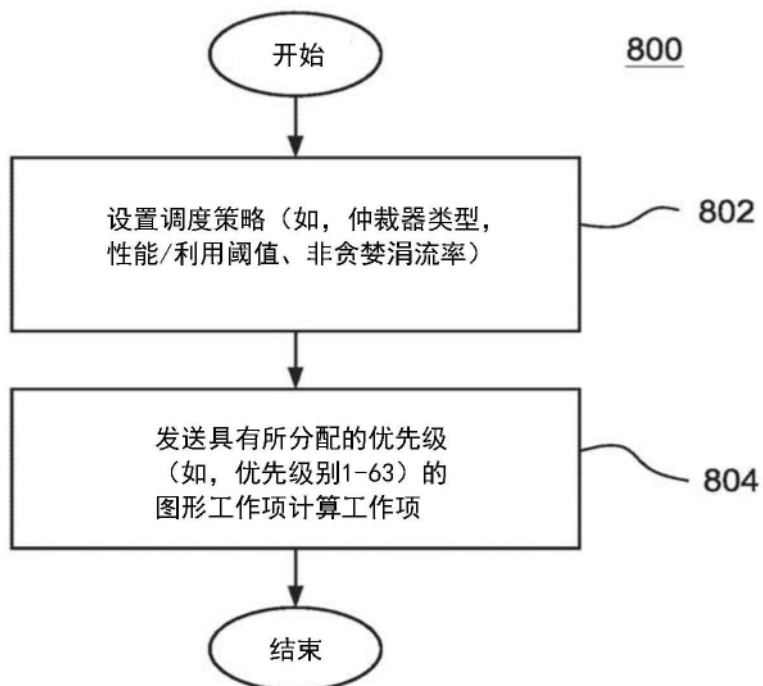


图8

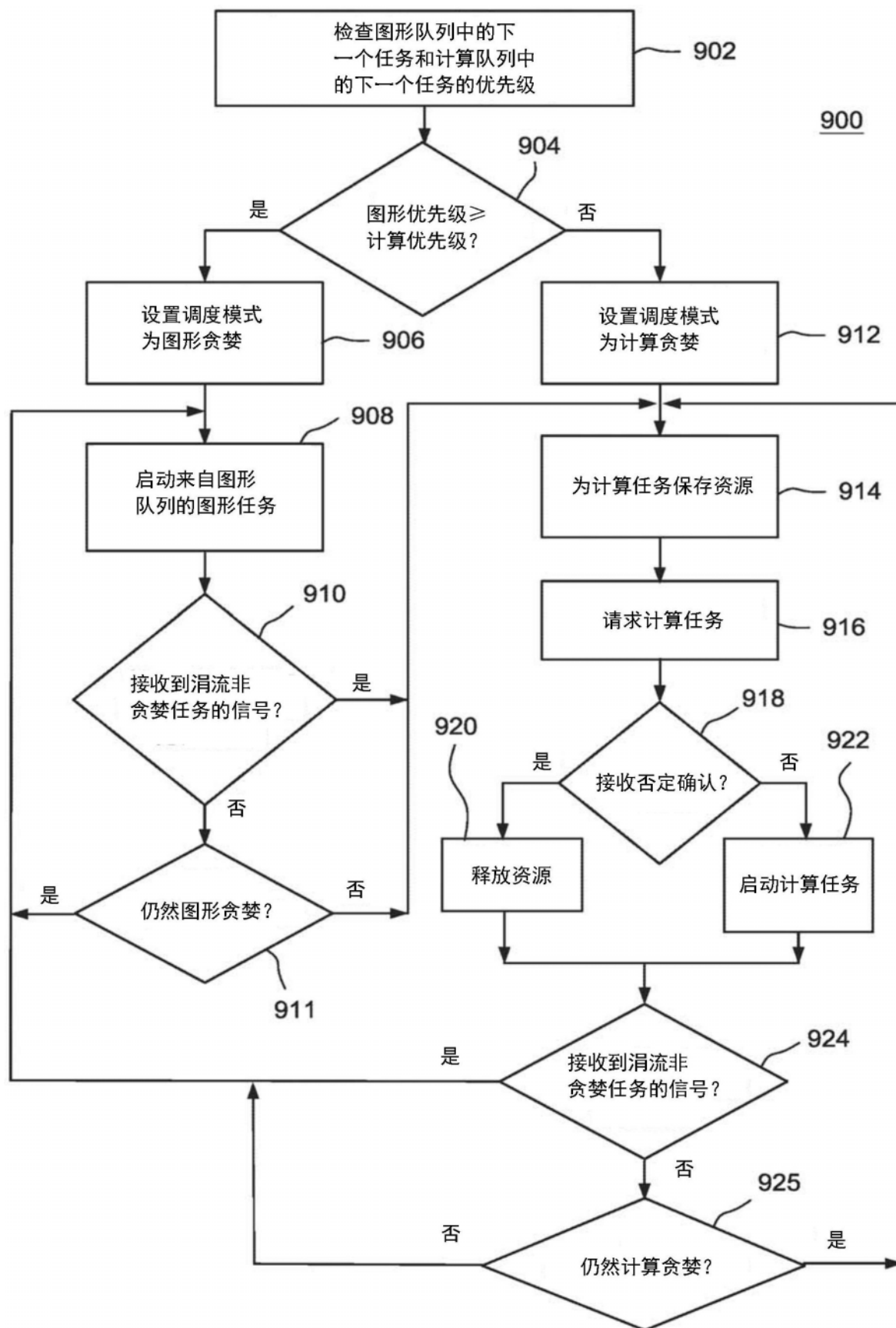


图9

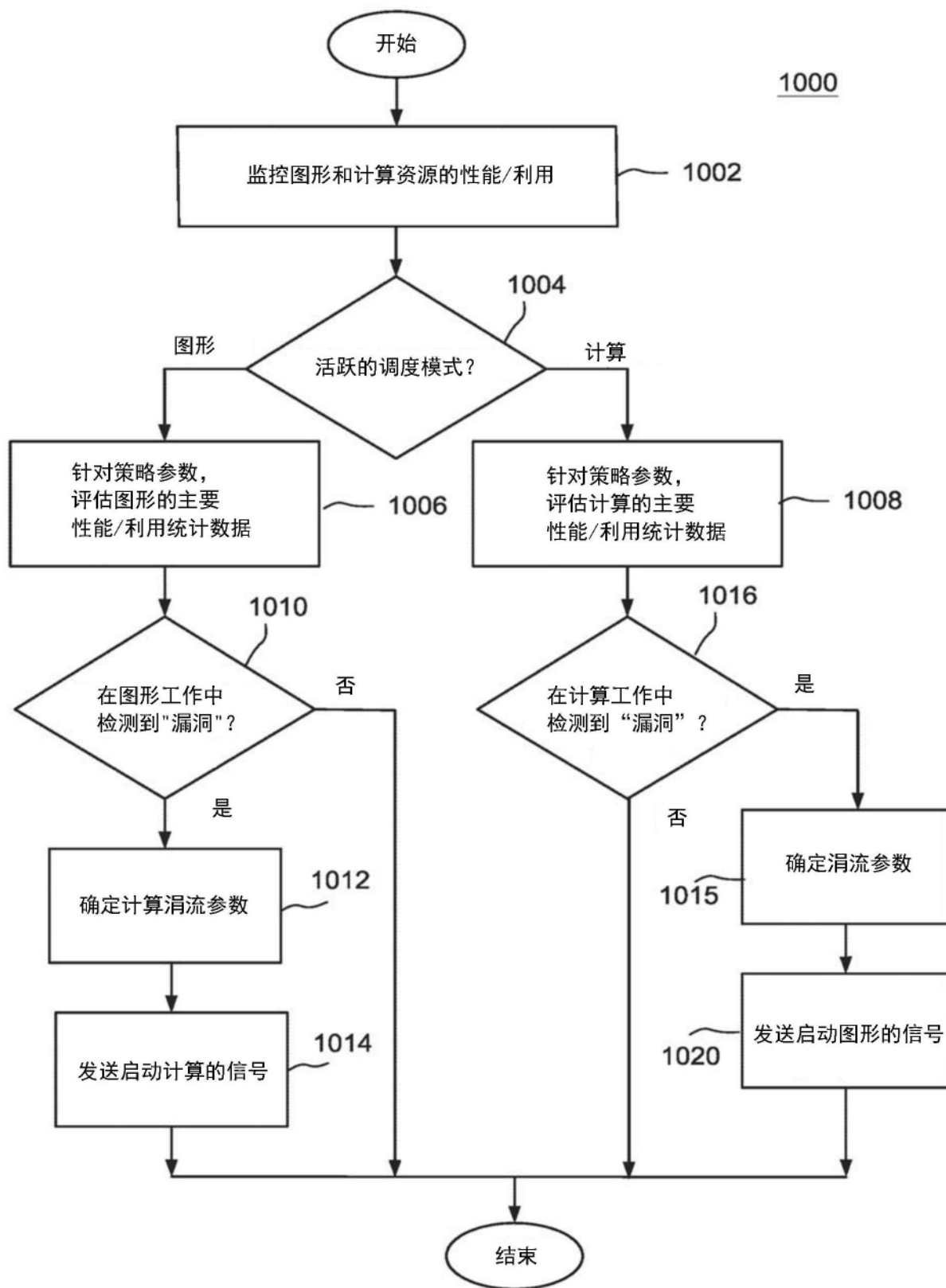


图10

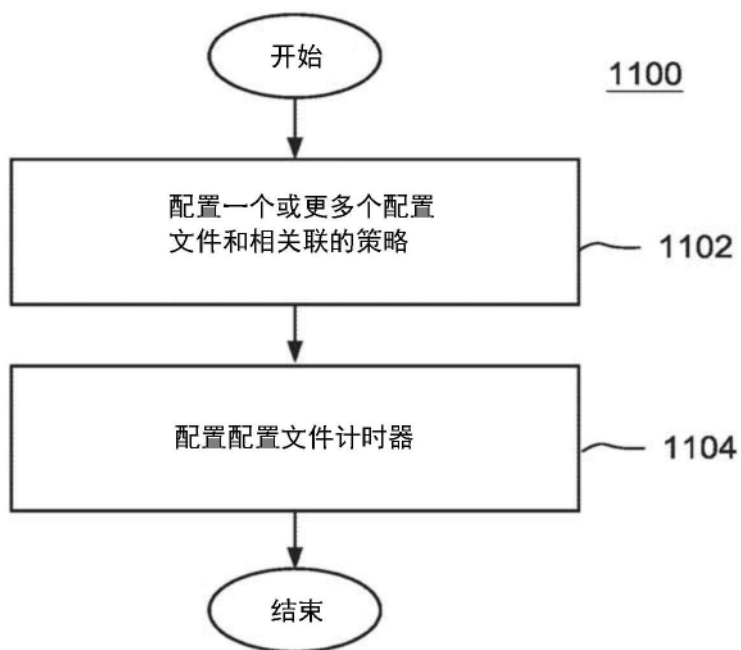


图11

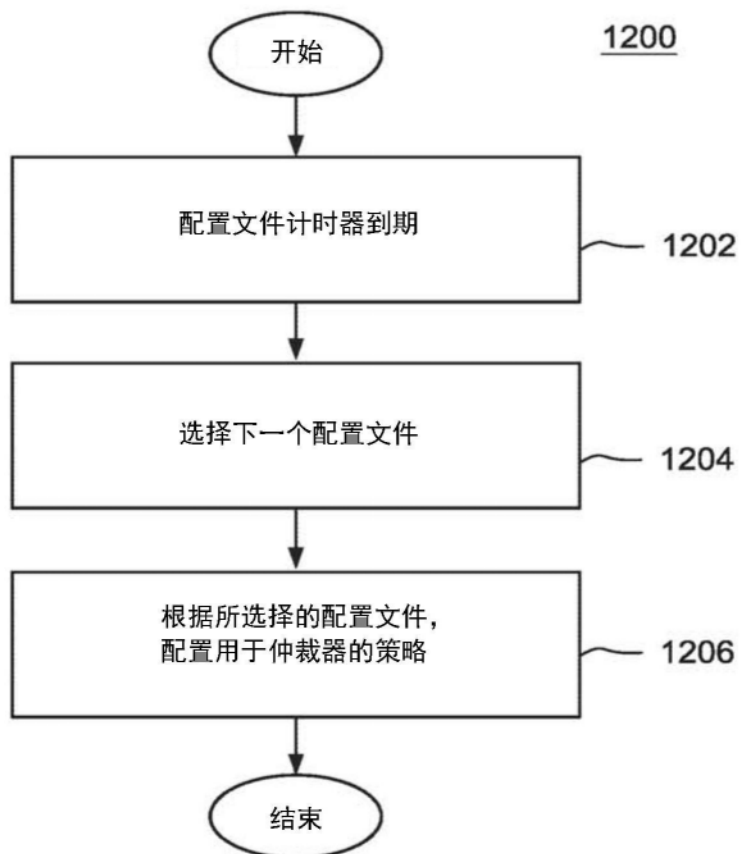


图12

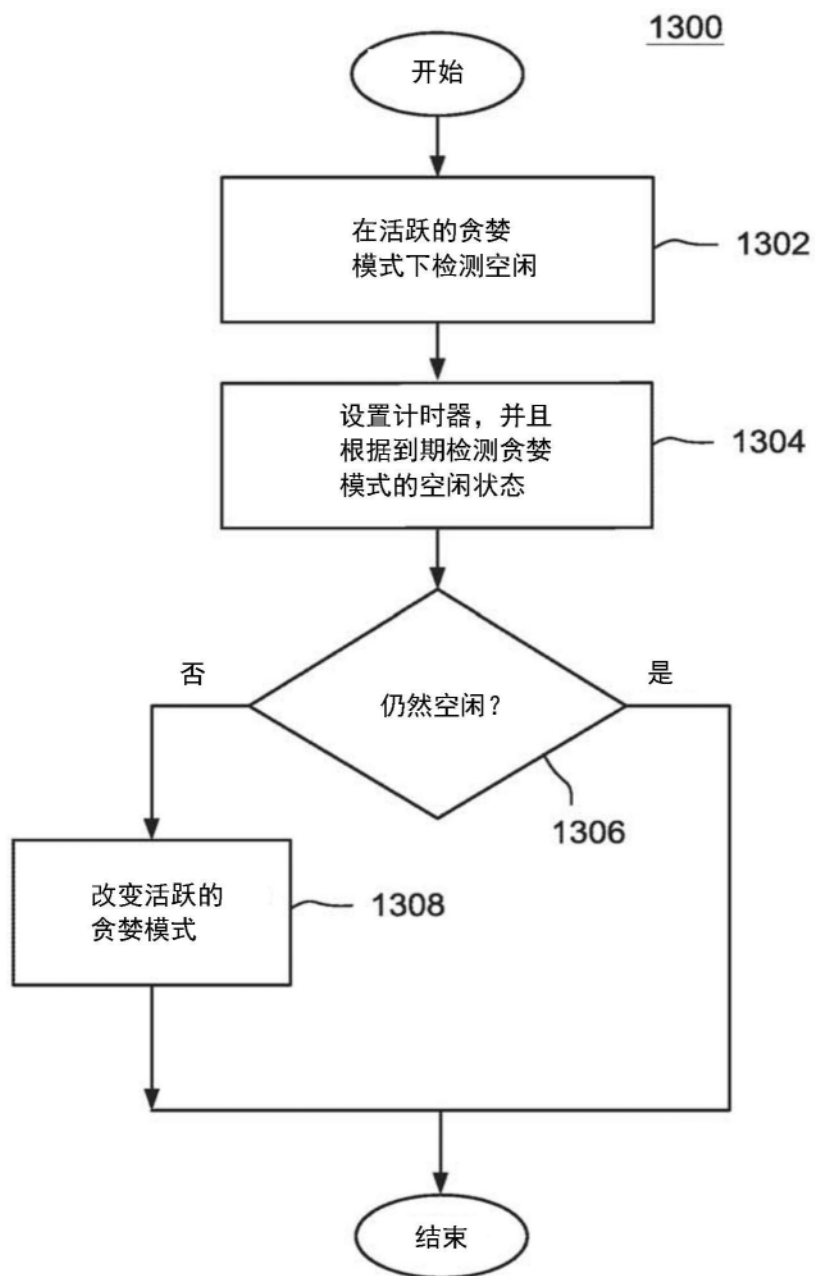


图13