



(12) 发明专利

(10) 授权公告号 CN 112445609 B

(45) 授权公告日 2025. 03. 28

(21) 申请号 202010196535.6

(22) 申请日 2020.03.19

(65) 同一申请的已公布的文献号
申请公布号 CN 112445609 A

(43) 申请公布日 2021.03.05

(30) 优先权数据
16/562,359 2019.09.05 US

(73) 专利权人 辉达公司
地址 美国加利福尼亚州

(72) 发明人 杰尔姆·F·小杜鲁克
G·S·帕尔默 J·S·R·埃万斯
S·辛格 S·H·邓肯
W·A·甘地 L·V·姗 E·罗克
苏斐琦 J·L·德明
A·梅内塞斯 P·维迪雅

P·乔吉尼帕里 T·J·珀塞尔
M·曼达尔

(74) 专利代理机构 北京市磐华律师事务所
11336

专利代理师 刘娟

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

(56) 对比文件
US 2013290978 A1, 2013.10.31
US 2014380028 A1, 2014.12.25
US 2018308198 A1, 2018.10.25

审查员 罗容

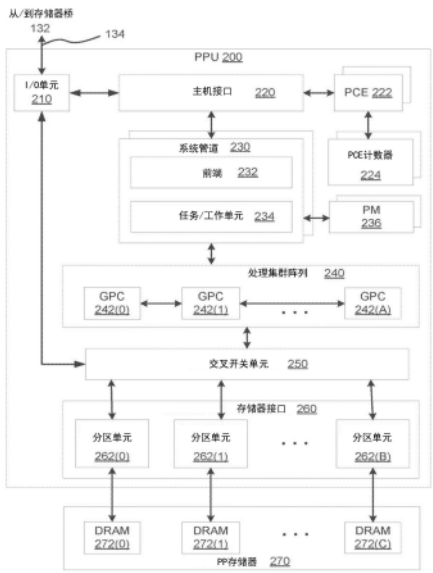
权利要求书3页 说明书52页 附图43页

(54) 发明名称

用于配置处理器以作为多个独立处理器功能的方法及系统

(57) 摘要

并行处理单元 (PPU) 可以被分为多个分区。每个分区被配置为与整个 PPU 操作类似的操作。给定分区包括与整个 PPU 关联的计算和存储资源的子集。在 CPU 上执行的软件将管理员用户的 PPU 分区。访客用户被分配给一个分区,并且所述访客用户可以与分配给任何其他分区的任何其他访客用户隔离地在该分区内执行处理任务。因为 PPU 可以被分为隔离的分区,所以多个 CPU 进程可以有效地利用 PPU 资源。



1. 一种计算机实现的方法,包括:

对处理器中包括的硬件资源集合进行分区,以生成多个逻辑分区,其中,每个逻辑分区使用所述处理器总容量的一部分来执行所述处理器的功能,每个逻辑分区被分配独占使用所述硬件资源集合的子集,每个逻辑分区与其他逻辑分区在功能上隔离地执行,并且其中,分配给第一逻辑分区的硬件资源的第一子集包括至少多个处理核心;

基于与所述第一逻辑分区相关联的目标配置来配置所述第一逻辑分区中的多个引擎,其中,所述第一逻辑分区中的多个引擎共享所述硬件资源的第一子集;

检测到包含在所述硬件资源的第一子集中的分配给所述第一逻辑分区的硬件资源是非功能性的;以及

确定一个或多个处理任务能通过所述第一逻辑分区、不使用分配的所述硬件资源而被执行。

2. 根据权利要求1所述的计算机实现的方法,其中对所述硬件资源集合进行分区还包括:对所述硬件资源集合进行分区,以生成包括硬件资源的第二子集的第二逻辑分区,其中所述硬件资源的第一子集包括比所述硬件资源的第二子集更多的硬件资源。

3. 根据权利要求1所述的计算机实现的方法,其中所述多个引擎中包括的每个引擎执行与不同的处理上下文相关联的一组处理任务。

4. 根据权利要求1所述的计算机实现的方法,还包括:

使所述多个引擎中包括的第一引擎在给定的时间间隔期间执行与处理上下文相关联的一组处理任务;以及

使所述多个引擎中包括的第二引擎在所述给定的时间间隔期间执行一个或多个上下文切换操作。

5. 根据权利要求1所述的计算机实现的方法,还包括:

使所述多个引擎中包括的第一引擎在给定的时间间隔期间执行与第一处理上下文相关联的第一组处理任务;以及

响应于在所述多个引擎中包括的第二引擎执行与第二处理上下文相关联的第二组处理任务时发生的故障,使所述第二引擎在所述给定的时间间隔期间复位。

6. 根据权利要求1所述的计算机实现的方法,还包括:

使所述多个引擎中包括的第一引擎在第一时间间隔期间执行与第一处理子上下文相关联的第一组处理任务;以及

使所述第一引擎在所述第一时间间隔期间执行与第二处理子上下文相关联的第二组处理任务,

其中所述第一处理子上下文和所述第二处理子上下文均从第一处理上下文导出,并且其中根据所述第一处理上下文配置所述第一引擎。

7. 根据权利要求1所述的计算机实现的方法,还包括:

分析所述硬件资源集合以识别第一硬件资源的非功能实例;

确定所述硬件资源的子集包括所述第一硬件资源的功能实例;以及

电隔离所述第一硬件资源的所述非功能实例。

8. 根据权利要求1所述的计算机实现的方法,还包括:

进一步对所述硬件资源集合进行分区以生成第二逻辑分区;

根据所述第一逻辑分区配置所述第二逻辑分区；

识别所述第一逻辑分区内包括的第一引擎,所述第一引擎执行与第一处理上下文相关联的第一处理任务;以及

将所述第一处理上下文迁移到所述第二逻辑分区内包括的第二引擎。

9.根据权利要求1所述的计算机实现的方法,还包括:

确定所述第一逻辑分区包括第一硬件资源的非功能实例;

确定第二逻辑分区包括所述第一硬件资源的功能实例;

停用所述第一硬件资源的所述功能实例;

根据所述第一逻辑分区配置所述第二逻辑分区;以及

将与所述第一逻辑分区相关联的第一处理上下文迁移到所述第二逻辑分区。

10.根据权利要求1所述的计算机实现的方法,还包括:

接收与第一计算环境相关联的第一输入,其中基于第一组许可在所述第一计算环境内的所述处理器上执行第一组操作;以及

接收与第二计算环境相关联的第二输入,其中基于第二组许可在所述第二计算环境内的所述处理器上执行第二组操作,并且所述第一组许可包括比所述第二组许可更多数量的许可,

其中基于所述第一输入对所述硬件资源集合进行分区,以及

其中基于所述第二输入配置所述多个引擎中包括的第一引擎。

11.一种非暂时性计算机可读介质,其存储程序指令,所述程序指令由处理器执行时使所述处理器执行以下步骤:

对处理器中包括的硬件资源集合进行分区,以生成多个逻辑分区,其中,每个逻辑分区使用所述处理器总容量的一部分来执行所述处理器的功能,每个逻辑分区被分配独占使用所述硬件资源集合的子集,每个逻辑分区与其他逻辑分区在功能上隔离地执行,并且其中,分配给第一逻辑分区的硬件资源的第一子集包括至少多个处理核心;

基于与所述第一逻辑分区相关联的目标配置来配置所述第一逻辑分区中的多个引擎,其中,所述第一逻辑分区中的多个引擎共享所述硬件资源的第一子集的一部分;

检测到包含在所述硬件资源的第一子集中的分配给所述第一逻辑分区的硬件资源是非功能性的;以及

确定一个或更多个处理任务能通过所述第一逻辑分区、不使用分配的所述硬件资源而被执行。

12.根据权利要求11所述的非暂时性计算机可读介质,其中所述多个引擎中包括的每个引擎执行与不同的处理上下文相关联的一组处理任务。

13.根据权利要求11所述的非暂时性计算机可读介质,还包括以下步骤:

使所述多个引擎中包括的第一引擎在给定的时间间隔期间执行与处理上下文相关联的一组处理任务;以及

使所述多个引擎中包括的第二引擎在所述给定的时间间隔期间执行一个或更多个上下文切换操作。

14.根据权利要求11所述的非暂时性计算机可读介质,还包括以下步骤:

使所述多个引擎中包括的第一引擎在给定的时间间隔期间执行与第一处理上下文相

关联的第一组处理任务;以及

响应于在所述多个引擎中包括的第二引擎执行与第二处理上下文相关联的第二组处理任务时发生的故障,使所述第二引擎在所述给定的时间间隔期间复位。

15. 根据权利要求11所述的非暂时性计算机可读介质,还包括以下步骤:

使所述多个引擎中包括的第一引擎在第一时间间隔期间执行与第一处理子上下文相关联的第一组处理任务;以及

使所述第一引擎在所述第一时间间隔期间执行与第二处理子上下文相关联的第二组处理任务,

其中所述第一处理子上下文和所述第二处理子上下文均从第一处理上下文导出,并且其中根据所述第一处理上下文配置所述第一引擎。

16. 根据权利要求11所述的非暂时性计算机可读介质,还包括以下步骤:

确定所述第一逻辑分区包括第一硬件资源的非功能实例;

确定第二逻辑分区包括所述第一硬件资源的功能实例;

停用所述第一硬件资源的所述功能实例;

根据所述第一逻辑分区配置所述第二逻辑分区;以及

将与所述第一逻辑分区相关联的第一处理上下文迁移到所述第二逻辑分区。

17. 根据权利要求11所述的非暂时性计算机可读介质,还包括以下步骤:

接收与主机计算环境相关联的分区输入;以及

接收与访客计算环境相关联的配置输入,

其中基于所述分区输入对所述硬件资源集合进行分区,以及

其中基于所述配置输入配置所述多个引擎中包括的第一引擎。

18. 根据权利要求11所述的非暂时性计算机可读介质,其中所述多个引擎被配置为实现多个图形处理管线。

19. 根据权利要求11所述的非暂时性计算机可读介质,其中对所述硬件资源集合进行分区的步骤包括:基于一组位激活与所述处理器相关联的逻辑边界集合,其中所述一组位中包括的每个位对应于所述逻辑边界集合中包括的不同逻辑边界。

20. 一种系统,包括:

存储器,其存储软件应用程序;以及

处理器,其在执行所述软件应用程序时,被配置为执行以下步骤:

对处理器中包括的硬件资源集合进行分区,以生成多个逻辑分区,其中,每个逻辑分区使用所述处理器总容量的一部分来执行所述处理器的功能,每个逻辑分区被分配独占使用所述硬件资源集合的子集,每个逻辑分区与其他逻辑分区在功能上隔离地执行,并且其中,分配给第一逻辑分区的硬件资源的第一子集包括至少多个处理核心;

基于与所述第一逻辑分区相关联的目标配置来配置所述第一逻辑分区中的多个引擎,其中,所述第一逻辑分区中的多个引擎共享所述硬件资源的第一子集的一部分;

检测到包含在所述硬件资源的第一子集中的分配给所述第一逻辑分区的硬件资源是非功能性的;以及

确定一个或更多个处理任务能通过所述第一逻辑分区、不使用分配的所述硬件资源而被执行。

用于配置处理器以作为多个独立处理器功能的方法及系统

技术领域

[0001] 各种实施例通常涉及并行处理架构,更具体地,涉及用于配置处理器以作为多个独立处理器的功能的技术。

背景技术

[0002] 传统的中央处理单元 (CPU) 通常包括可以执行相对少的数量的CPU进程的相对少的数量的处理核心。相反,传统的图形处理单元 (GPU) 通常包括可以彼此并行执行数百个线程的数百个处理核心。因此,考虑到在使用传统GPU时可以部署大量处理的资源,传统GPU通常可以比传统CPU更快、更有效地执行某些处理任务。

[0003] 在一些实施方式中,在CPU上执行的CPU进程可以将给定的处理任务卸载到GPU,以便使该处理任务更快地执行。这样做,CPU进程在GPU上生成处理上下文(context),该处理上下文指定了将要实现以执行处理任务的各种GPU资源的目标状态。那些GPU资源可以包括处理、图形和存储器资源等。然后,CPU进程根据处理上下文在GPU上启动线程组,并且该线程组利用各种GPU资源来执行处理任务。在许多这些类型的实现中,GPU一次仅根据一个处理环境进行配置。但是,在某些情况下,CPU需要在相同的时间间隔内将一个以上的CPU进程卸载到GPU。在这种情况下,CPU可以在不同的时间点动态更改在GPU上实现的处理上下文,以便在一定时间间隔期间为这些CPU进程提供串行服务。但是,这种方法的一个缺点是,某些CPU进程卸载的处理任务无法充分利用GPU的资源。因此,当在GPU上串行执行与这些CPU进程相关联的一个或更多个处理任务时,某些GPU资源可能会被闲置,这会降低总体GPU性能和利用率。

[0004] 一种在GPU上同时执行多个CPU进程的方法是在给定的“父(parent)”处理上下文中生成多个不同的处理子上下文,并将每个不同的处理子上下文分配给不同的CPU进程。然后,多个CPU进程可以同时也在GPU上启动不同的线程组,其中每个线程组利用根据特定处理子上下文配置的特定GPU资源。通过这种方法,可以更有效地利用GPU,因为一个以上的CPU进程可以在同一时间点将处理任务卸载到GPU,从而有可能避免某些GPU资源被闲置的情况。

[0005] 上述方法的一个问题是与不同的处理子上下文相关联的CPU进程可能会不公平地消耗GPU资源,这些GPU资源应在不同的处理子上下文中更均匀地分配或分布。例如,第一CPU进程可以在执行大量读取请求并消耗大量可用GPU存储器带宽的第一处理子上下文中启动第一线程组。第二CPU进程随后可以在也执行大量的读取请求的第二处理子上下文中启动第二线程组。但是,由于第一线程组已经消耗了许多可用GPU存储器带宽,第二线程集可能会遇到高延迟,这可能会导致第二CPU进程停滞。

[0006] 上述方法的另一个问题是,因为处理子上下文共享父上下文,所以当与一个处理子上下文关联的线程执行时,发生的任何错误都可能干扰与共享同一父上下文的另一个处理子上下文相关联的其他线程的执行。例如,第一CPU进程可以启动与第一处理子上下文相关联的第一线程组以执行第一处理任务。第二CPU进程可以启动与第二处理子上下文相关

联的第二线程组,并且第二线程组随后可能会出现故障并失败。为了从故障中恢复,GPU将必须重置父上下文,这将自动重置第一处理子上下文和第二处理子上下文。在这种情况下,即使故障是由第二线程组而不是第一线程组引起的,也会中断第一线程组的执行。

[0007] 如前所述,在本领域中需要用于配置GPU来执行与多个上下文相关联的处理任务的更有效的技术。

发明内容

[0008] 各个实施例包括一种计算机实现的方法,该方法包括:对包括在处理器中的硬件资源集进行分区,以生成包括硬件资源的第一子集的第一逻辑分区;以及在该第一逻辑分区内生成多个引擎,其中包括在所述多个引擎中的每个引擎被分配了硬件资源的第一子集的不同部分,并与包括在所述多个引擎的所有其他引擎在功能上隔离执行。

[0009] 相对于现有技术,所公开的技术的一个技术优势在于,利用所公开的技术,并行处理单元(PPU)(例如GPU)可以同时支持多个上下文并且在功能上彼此隔离。因此,多个CPU进程可以通过同时执行多个不同上下文来有效地利用PPU资源,而上下文之间不会相互干扰。

附图说明

[0010] 为了可以详细地理解各个实施例的上述特征的方式,可以通过参考各个实施例(其中一些在附图中示出)来对以上简要概述的发明构思进行更具体的描述。然而,应注意,附图仅示出了本发明构思的典型实施例,因此不应视为以任何方式限制范围,并且存在其他等效的实施例。

[0011] 图1是被配置为实现各种实施例的一个或更多个方面的计算机系统的框图;

[0012] 图2是根据各种实施例的包括在图1的并行处理子系统内的并行处理单元(PPU)的框图;

[0013] 图3是根据各种实施例的包括在图2的并行处理单元中的通用处理集群的框图;

[0014] 图4是根据各种实施例的包括在图2的PPU中的分区单元的框图;

[0015] 图5是根据各种实施例的包括在图2的PPU中的各种PPU资源的框图;

[0016] 图6是根据各种实施例的图1的管理程序如何将PPU资源逻辑分组为PPU分区集的示例;

[0017] 图7示出了根据各种实施例的图1的管理程序如何配置PPU分区集以实现一个或更多个同时多上下文(SMC)引擎;

[0018] 图8A是根据各种实施例的图7的DRAM的更详细的图示;

[0019] 图8B示出了根据各种实施例的如何寻址图8B的各个DRAM部分;

[0020] 图9是根据各种实施例的图1的管理程序如何分区和配置PPU的数据流程图;

[0021] 图10是根据各种实施例的用于代表一个或更多个用户对PPU进行分区和配置的方法步骤的流程图;

[0022] 图11示出了根据各种实施例的根据图1的管理程序可以配置一个或更多个PPU分区的分区配置表;

[0023] 图12示出了根据各种实施例的图1的管理程序如何分区PPU以生成一个或更多个PPU分区;

- [0024] 图13示出了根据各种实施例的图1的管理程序在分区期间如何分配各种PPU资源；
- [0025] 图14A示出了根据各种实施例的运行多个VM的多个访客OS如何在一个或多个PPU分区内同时启动多个处理上下文；
- [0026] 图14B示出了根据各种实施例的主机OS如何在一个或多个PPU分区内同时启动多个处理环境；
- [0027] 图15示出了根据各种实施例的图1的管理程序如何将虚拟地址空间标识符分配给不同的SMC引擎；
- [0028] 图16示出了根据各种实施例的在减轻故障时存储器管理单元如何转换本地虚拟地址空间标识符；
- [0029] 图17示出了根据各种实施例的当在不同PPU上的SMC引擎之间迁移处理上下文时图1的管理程序如何实现软发言权；
- [0030] 图18是根据各种实施例的用于在PPU内配置计算资源以同时支持与多个处理上下文相关联的操作的方法步骤的流程图；
- [0031] 图19示出了根据各种实施例的边界选项集,根据边界选项集图1的管理程序可以生成一个或多个PPU存储器分区；
- [0032] 图20示出了根据各种实施例的图1的管理程序如何分区PPU存储器以生成一个或多个PPU存储器分区的示例；
- [0033] 图21示出了根据各种实施例的图16的存储器管理单元如何提供对不同的PPU存储器分区的访问；
- [0034] 图22示出了根据各种实施例的图16的存储器管理单元如何执行各种地址转换；
- [0035] 图23示出了根据各种实施例的图16的存储器管理单元如何同时提供与多个处理上下文相关联的支持操作；
- [0036] 图24是根据各种实施例的用于在PPU内配置存储器资源以同时支持与多个处理上下文相关联的操作的方法步骤的流程图；
- [0037] 图25是根据各种实施例的示出了与图2的PPU相关联的VM级时间切片的时间线集；
- [0038] 图26是根据各种其他实施例的示出了与图2的PPU相关联的VM级时间切片另一时间线集；
- [0039] 图27是根据各种实施例的示出了与图2的PPU相关联的SMC级时间切片的时间线；
- [0040] 图28示出了根据各种实施例的VM如何从一个PPU迁移到另一PPU；
- [0041] 图29是根据各种实施例的示出与图2的PPU相关联的精细VM迁移的时间线集；
- [0042] 图30A-30B示出了根据各种实施例的用于对图2的PPU中的VM进行时间切片的方法步骤的流程图；
- [0043] 图31是根据各种实施例的存储器映射,其示出了BAR0地址空间如何映射到图2的PPU内的特权寄存器空间；
- [0044] 图32是根据各种实施例的用于在图2的PPU中寻址特权寄存器地址空间的方法步骤的流程图；
- [0045] 图33是根据各种实施例的用于图2的PPU的性能监视系统的框图；
- [0046] 图34A-34B示出了根据各种实施例的图33的性能多路复用器单元的各种配置；
- [0047] 图35是根据各种实施例的用于图2的PPU的性能监视器聚合系统的框图；

[0048] 图36示出了根据各种实施例的与图35的性能监视器聚合系统相关联的触发分组的格式;

[0049] 图37是根据各种实施例的用于监视图2的PPU的性能的方法步骤的流程图;

[0050] 图38是根据各种实施例的用于图2的PPU的功率和时钟频率管理系统的框图;以及

[0051] 图39是根据各个实施例的用于管理图2的PPU 200的功耗的方法步骤的流程图。

具体实施方式

[0052] 在以下描述中,阐述了许多具体细节以提供对各种实施例的更透彻的理解。然而,对于本领域技术人员将显而易见的是,可以在没有或更多个这些具体细节的情况下实践本发明构思。

[0053] 如上所述,传统GPU通常可以比传统CPU更快地执行某些处理任务。在一些配置中,在CPU上执行的CPU进程可以将给定的处理任务卸载到GPU,以便更快地执行该处理任务。这样,CPU进程会在GPU上生成处理上下文,该上下文指定各种GPU资源的目标状态,并且然后在GPU上启动线程组以执行处理任务。

[0054] 在某些情况下,可能需要多个CPU进程在相同的时间间隔期间将处理任务卸载到GPU。但是,GPU一次只能根据一个处理上下文进行配置。在这种情况下,CPU可以在不同的时间点动态更改GPU的处理上下文,以便跨时间间隔连续服务多个CPU进程。但是,某些CPU进程在执行处理任务时可能无法充分利用GPU资源,有时使各种GPU资源处于空闲状态。为了解决此问题,CPU可以在“父”处理上下文中生成多个处理子上下文,并将这些处理子上下文分配给不同的CPU进程。然后,这些CPU进程可以同时也在GPU上启动不同的线程组,并且每个线程组可以利用根据特定处理子上下文配置的特定GPU资源。可以实施此方法以更有效地利用GPU资源。但是,这种方法有几个缺点。

[0055] 首先,与不同的处理子上下文相关联的CPU进程会不公平地消耗应该在不同的处理子上下文之间公平共享的GPU资源,从而导致一个CPU进程可能使另一个CPU进程的进度停滞的情况。其次,由于处理子上下文共享父处理上下文,因此在执行与一个处理子上下文相关联的线程的过程中发生的任何错误都可能破坏与同一父处理上下文中包括的其他处理子上下文相关联的线程的执行。在某些情况下,一个处理子上下文中发生的故障可能导致同一父处理上下文中的所有其他处理子上下文被重置并重新启动。

[0056] 通常,与处理子上下文相关联的上述缺点限制了传统GPU可以支持多租户的程度。如本文所指,“多租户”是指GPU配置,其中多个用户或“租户”同时或在重叠的时间间隔期间使用GPU资源执行处理操作。通常,传统GPU通过允许不同的租户在给定的父处理上下文中使用不同的处理子上下文执行不同的处理任务来提供对多租户的支持。但是,处理子上下文不是隔离的计算环境,因为由于上述各种原因,在不同的处理子上下文中执行的处理任务可能会相互干扰。因此,占用给定GPU的任何给定租户都会对GPU提供给其他租户的服务质量产生负面影响。这些因素可能会降低基于云的GPU部署的吸引力,在这种部署中多个用户可能同时访问同一GPU。

[0057] 为了解决这些问题,各种实施例包括可以被分为分区的并行处理单元(PPU)。每个分区被配置为同时执行与多个处理环境相关联的处理任务。给定的分区包括一个或更多个GPU资源的逻辑分组或“切片”。每个切片提供足够的计算、图形和存储器资源,以模拟整个

PPU的操作。在CPU上执行的管理程序(hypervisor)代表管理员用户执行各种技术来对PPU进行分区。访客用户被分配给一个分区,并且然后可以在该分区内执行与分配给任何其他分区的任何其他访客用户隔离的处理任务。

[0058] 相对于现有技术,所公开的技术的一个技术优势在于,利用所公开的技术,PPU可以同时并且在功能上彼此隔离地支持多个处理上下文。因此,多个CPU进程可以经由多个不同的处理上下文有效地利用PPU资源,并且不会彼此干扰。所公开的技术的另一技术优势在于,因为可以使用所公开的技术将PPU分区为隔离的计算环境,所以相对于依赖于处理子上下文来提供多租户功能的现有技术方法,PPU可以支持更健壮的多租户形式。因此,当实现所公开的技术时,PPU变得更适合于基于云的部署,在这种部署中可以向不同的和潜在竞争的实体提供对同一PPU内不同分区的访问。这些技术优势代表相对于现有技术方法的一种或更多种技术进步。

[0059] 系统总览

[0060] 图1是被配置为实现本发明的一个或更多个方面的计算机系统的框图。如图所示,计算机系统100包括经由存储器桥132耦合在一起的中央处理单元(CPU)110、系统存储器120和并行处理子系统130。并行处理子系统130经由通信路径134耦合至存储器桥132。一个或更多个显示设备136可以耦合到并行处理子系统130。计算机系统100还包括系统盘140、一个或更多个附加卡150和网络适配器160。系统盘140耦合到I/O桥142。I/O桥142经由通信路径138耦合到存储器桥132,并且还耦合到输入设备146。一个或更多个附加卡150和网络适配器160经由交换机148耦合在一起,交换机148又耦合到I/O桥142。

[0061] 存储器桥132是硬件单元,其促进CPU 110、系统存储器120和并行处理子系统130以及计算机系统100的其他组件之间的通信。例如,存储器桥132可以是北桥芯片。通信路径134是高速和/或高带宽数据连接,其促进跨一个或更多个独立通道的并行处理子系统130和存储器桥132之间的低延迟通信。例如,通信路径134可以是外围组件互连快速(PCIe)链路、加速图形端口(AGP)、超传输或任何其他技术上可行的通信总线类型。

[0062] I/O桥142是硬件单元,其促进利用系统盘140、输入设备146、一个或更多个附加卡150、网络适配器160以及计算机系统100的各种其他组件执行的输入和/或输出操作。例如,I/O桥142可以是南桥芯片。通信路径138是促进存储器桥132和I/O桥142之间的低延迟通信的高速和/或高带宽数据连接。例如,通信路径138可以是PCIe链路、AGP、超传输或任何其他技术上可行的通信总线类型。利用所示的配置,耦合到存储器桥132或I/O桥142的任何组件都可以与耦合到存储器桥132或I/O桥142的任何其他组件通信。

[0063] CPU 110是被配置为协调计算机系统100的整体操作的处理器。这样,CPU 110执行指令以向包括在计算机系统100中的各种其他组件发出命令。CPU110也被配置执行指令以便处理由计算机系统100中包括的任何其他组件(包括系统存储器120和系统盘140)生成和/或存储的数据。系统存储器120和系统盘140是包括被配置为存储数据和软件应用程序的计算机可读介质的存储器设备。系统存储器120包括设备驱动器122和管理程序124,其操作在下面更详细地描述。并行处理子系统130包括一个或更多个并行处理单元(PPU),其被配置为高度并行处理架构同时执行多个操作。每个PPU包括以并行方式执行通用计算操作的一个或更多个计算引擎和/或以并行方式执行面向图形的操作的一个或更多个图形引擎。给定的PPU可以被配置为生成像素,以经由显示设备136显示。示例性的PPU在下面结合

图2-图4更详细地描述。

[0064] 设备驱动器122是软件应用程序,当由CPU 110执行时,其操作为CPU 110与并行处理子系统130之间的接口进行操作。特别地,设备驱动器122允许CPU 110将各种处理操作卸载到并行处理子系统130中用于高度并行执行,包括通用计算操作以及图形处理操作。管理程序124是一种软件应用程序,当由CPU 110执行时,其对并行处理子系统130中包括的各种计算、图形和存储器资源进行分区,以便为单独的用户提供对这些资源的独立使用,这将在下面结合图5-图10更详细地描述。

[0065] 在各种实施例中,计算机系统100的部分或全部组件可以在潜在地跨广阔的地理区域分布的基于云的环境中实现。例如,计算机系统100的各种组件可以跨地理上不同的数据中心部署。在这样的实施例中,计算机系统100的各种组件可以通过一个或更多个网络(包括任何数量的本地内联网和/或因特网)彼此通信。在各种其他实施例中,计算机系统100的某些组件可以经由一个或更多个虚拟化设备来实现。例如,CPU 110可以被实现为硬件CPU的虚拟实例。在一些实施例中,部分或全部并行处理子系统130可以与计算机系统100的一个或更多个其他组件集成以便形成单个芯片,例如片上系统(SoC)。

[0066] 本领域技术人员将理解,计算机系统100的架构足够灵活以跨广阔的潜在场景和用例实现。例如,计算机系统100可以在云计算中心中实现,以向一个或更多个用户公开通用计算能力和/或通用图形处理能力。可选地,计算机系统100可以被部署在汽车实现中以便执行与车辆导航相关联的数据处理操作。本领域技术人员将进一步理解,在不脱离本实施例的整体范围和精神的情况下,可以以任何技术上可行的方式来修改计算机系统100的各种组件以及这些组件之间的连接拓扑。

[0067] 图2是根据各种实施例的包括在图1的并行处理子系统PPU中的框图。如图所示,PPU 200包括I/O单元210、主机接口220、系统(sys)管道230、处理集群阵列240、交叉开关单元250和存储器接口260。PPU200耦合到PPU存储器270。可以通过任何技术上可行的硬件类型和/或任何技术上可行的硬件和软件组合来实现所示的每个组件。

[0068] I/O单元210经由通信路径134和存储器桥132耦合至图1的CPU110。I/O单元210还耦合至主机接口220和交叉开关单元250。主机接口220耦合至一个或更多个物理复制引擎(PCE)222,该物理复制引擎(PCE)222又耦合到一个或更多个PCE计数器224。主机接口220也耦合到系统管道230。给定的系统管道230包括前端232、任务/工作单元234以及性能监视器(PM)236,并且耦合到处理集群阵列240。处理集群阵列240包括通用处理集群(GPC)242(0)至242(A),其中A为正整数。处理集群阵列240耦合到交叉开关单元250。交叉开关单元250耦合到存储器接口260。存储器接口260包括分区单元262(0)至262(B),其中B是正整数值。每个分区单元262可以分别连接到交叉开关单元250。PPU存储器270包括动态随机存取存储器(DRAM)272(0)至272(C),其中C是正整数值。为了促进在多个处理上下文上同时操作,PPU 200内的各个单元被复制如下:(a) 主机接口220包括PBDMA 520(0)至520(7);(b) 系统管道230包括系统管道230(0)至230(7),使得任务/工作单元234对应于SKED 500(0)至SKED 500(7);并且任务/工作单元234对应于CWD 560(0)至560(7)。

[0069] 在操作中,I/O单元210从CPU 110获得各种类型的命令数据,并将该命令数据分发给PPU 200的相关组件以供执行。特别地,I/O单元210从CPU 110获得与处理任务相关联的命令数据,并将该命令数据路由至主机接口220。I/O单元210还从CPU 110获得与存储器访

问操作相关联的命令数据,并将该命令数据路由至交叉开关单元250。与处理任务有关的命令数据通常包括一个或更多个指向任务元数据(TMD)的指针,该指针存储在PPU存储器270或计算机系统100内其他位置的命令队列中。给定的TMD是编码处理任务,其描述要处理的数据的索引、要对该数据执行的操作、与那些操作关联的状态参数、执行优先级以及其他面向处理任务的信息。

[0070] 主机接口220从I/O单元210接收与处理任务有关的命令数据,然后经由一个或更多个命令流将该命令数据分发到系统管道230。在一些配置中,主机接口220为每个不同的系统管道230生成不同的命令流,其中给定命令流包括指向与对应系统管道230有关的TMD的指针。

[0071] 给定的系统管道230用接收到的命令数据执行各种预处理操作,以促进在处理集群阵列240内的GPC 242上执行相应的处理任务。在接收到与一个或更多个处理任务相关联的命令数据时,给定的系统管道230中的前端232获取相关联的处理任务并将这些处理任务中继到任务/工作单元234。任务/工作单元234将一个或更多个GPC 242配置为适合于执行处理任务的操作状态,并且然后将处理任务传输给那些GPC 242以执行。每个系统管道230可将复制任务卸载到一个或更多个执行专用复制操作的PCE 222。PCE计数器224跟踪PCE 222的使用以便平衡不同系统管道230之间的复制操作工作负载。PM236监视相应系统管道230的整体性能和/或资源消耗,并且可以限制由该系统管道230执行的各种操作以在所有系统管道230上保持平衡的资源消耗。

[0072] 每个GPC 242包括多个并行处理核心,这些并行处理核心能够同时执行大量线程,并且与其他GPC 242具有任何程度的独立性和/或隔离性。例如,给定的GPC 242可以执行与任何其他GPC242结合或隔离的数百个或数千个并发线程。在GPC 242上执行的并发线程组可以执行同一程序的单独实例或不同程序的单独实例。在某些配置中,GPC 242在所有系统管道230中共享,而在其他配置中,不同组的GPC 242被分配为与特定的系统管道230一起操作。每个GPC 242从一个或更多个系统管道230接收处理任务,并且作为响应,启动一个或多个线程组以执行那些处理任务并生成输出数据。在完成给定的处理任务后,给定的GPC 242将输出数据传输到另一个GPC 242进行进一步处理,或者传输到交叉开关单元250进行适当的路由。下面结合图3更详细地描述示例性GPC。

[0073] 交叉开关单元250是在I/O单元210、处理集群阵列240和存储器接口260之间路由各种类型的数据的切换机制。如上所述,I/O单元210发送与存储器访问操作有关的命令数据到交叉开关单元250。作为响应,交叉开关单元250将相关联的存储器访问操作提交给存储器接口260以进行处理。在某些情况下,交叉开关单元250还将从存储器接口260返回的读取数据路由回到请求读取数据的组件。如上所述,交叉开关单元250还从GPC 242接收输出数据,然后可以将该输出数据路由到I/O单元210以传输到CPU 110,或者将该数据路由到存储器接口260以存储和/或处理。交叉开关单元250通常被配置为在GPC 242之间并将数据从任何GPC 242路由到任何分区单元262。在各个实施例,交叉开关单元250可以实现虚拟通道以分离GPC 242和分区单元262之间的业务流。在各种实施例中,交叉开关单元250可以允许一组GPC 242与一组分区单元262之间的非共享路径。

[0074] 存储器接口260实现分区单元262,以提供对PPU存储器270中的DRAM 272的高带宽存储器访问。每个分区单元262可以与不同的DRAM 272彼此并行执行存储器访问操作,从而

有效地利用PPU存储器270的可用存储带宽。给定的分区单元262还经由一个或更多个内部高速缓存提供高速缓存支持。下面结合图4更详细地描述示例性分区单元262。

[0075] 通常,PPU存储器270,并且特别是DRAM 272,可以被配置为存储与通用计算应用程序和/或图形处理应用程序相关联的任何技术上可行的数据。例如,DRAM 272可以在通用计算应用中存储与神经网络相关联的大型数据值矩阵,或者可选地,在图形处理应用中存储包括多个渲染目标的一个或更多个帧缓冲器。在各种实施例中,可以经由任何技术上可行的存储器设备来实现DRAM 272。

[0076] 上面阐述的架构允许PPU 200以快速的方式并且相对于CPU 110的操作异步地执行各种各样的处理操作。特别地,PPU 200的并行架构允许并行执行大量的操作,并且彼此之间以及与CPU 110上执行的操作具有任何程度的独立性,从而加快了这些操作的整体性能。

[0077] 在一个实施例中,PPU 200可以被配置为执行通用计算操作,以便加快涉及大数据集的计算。此类数据集可能涉及金融时间序列、动态模拟数据、实时传感器读数、神经网络权重矩阵和/或张量以及机器学习参数等。在另一实施例中,PPU 200可以被配置为用作图形处理单元(GPU),其实现一个或更多个图形渲染管线以基于由CPU 110生成的图形命令来生成像素数据。PPU200然后可以经由显示设备136输出像素数据作为一个或更多个帧。PPU存储器170可以被配置为用作图形存储器,其以如上所述的方式存储一个或更多个帧缓冲器和/或一个或更多个渲染目标。在又一个实施例中,PPU 200可以被配置为同时执行通用计算操作和图形处理操作。在这样的配置中,一个或更多个系统管道230可以被配置为经由一个或更多个GPC 242来实现通用计算操作,并且一个或更多个其他系统管道230可以被配置为经由一个或更多个GPC 242来实现一个或更多个图形处理管线。

[0078] 对于任何上述结构,设备驱动器122和管理程序124,进行互操作以便将包含在PPU 200的各种计算、图形和存储器资源细分成单独的“PPU分区”。或者,可以有多个设备驱动器122,每个与“PPU分区”相关联。优选地,设备驱动器在CPU 110中的一组核上执行。给定的PPU分区整体上以与PPU 200基本相似的方式操作。特别地,每个PPU分区可以被配置为与其他PPU分区相对隔离地执行通用计算操作、图形处理操作或两种类型的操作。此外,给定的PPU分区可以被配置为在分配至给定的PPU分区的计算、图形和存储器资源上同时执行一个或更多个虚拟机(VM)时,同时实现多个处理上下文。下面结合图5-图8更详细地描述PPU资源到PPU分区的逻辑分组。下面结合图9-图10更详细地描述用于分区和配置PPU资源的技术。

[0079] 图3是根据本发明的各种实施例的包括在图2的PPU中的GPC的框图。如图所示,GPC 242耦合到存储器管理单元(MMU) 300,并且包括管线管理器310、工作分配交叉开关320、一个或更多个纹理处理集群(的TPC) 330、一个或更多个纹理单元340、L1.5级(L1.5)高速缓存350、PM 360和光栅前运算处理器(preROP) 370。管线管理器310耦合到工作分配交叉开关320和TPC330。每个TPC 330包括一个或更多个流式多处理器(SM) 332并耦合到纹理单元340、MMU 300、L1.5高速缓存350、PM 360和preROP370。纹理单元340和L1.5高速缓存350也耦合到MMU 300并彼此耦合。PreROP 370耦合到工作分配交叉开关320。可以通过任何技术上可行的硬件类型和/或任何技术上可行的硬件和软件组合来实现所示的每个组件。

[0080] GPC 242配置有高度并行架构,该架构支持并行执行大量线程。如本文所提到的,

“线程”是在特定的一组输入数据上执行以执行各种类型的操作(包括通用计算操作和图形处理操作)的特定程序的实例。在一实施例中, GPC 242可实现单指令多数据(SIMD)技术以支持并行执行大量线程,而不必依赖于多个独立指令单元。

[0081] 在另一个实施例中, GPC 242可实现单指令多线程(SIMT)技术,以经由向一个更多个处理引擎发出指令的公共指令单元来支持大量通用同步的线程的并行执行。本领域的技术人员将理解, SIMT执行允许不同的线程更容易通过给定的程序遵循分散执行路径,这与其中的所有线程通过给定的程序通常遵循非发散执行路径的SIMD执行不同。本领域技术人员将认识到, SIMD技术代表SIMT技术的功能子集。

[0082] GPC 242可以经由TPC 330中包括的SM 332执行大量的并行线程。每个SM 332包括一组功能单元(未示出), 包括一个或更多个执行单元和/或一个或更多个负载存储单元,其被配置为执行与接收到的处理任务相关联的指令。给定的功能单元可以以管线方式执行指令,这意味着可以在先前指令的执行完成之前向功能单元发出指令。在各个实施例中, SM 332内的功能单元可以被配置为执行各种不同的操作, 该操作包括整数和浮点算术(例如, 加法和乘法等)、比较操作、布尔运算(例如, AND、OR和XOR等)、位移和各种代数函数(例如, 平面插值和三角函数、指数函数和对数函数等)的计算。每个功能单元可以将中间数据存储在驻留于SM 332中的1级(L1)高速缓存中。

[0083] 通过上述功能单元, SM 332被配置为处理在不同输入数据上同时执行相同程序的一个或更多个“线程组”(也称为“线程束(warp)”)。线程组中的每个线程通常都经由不同的功能单元执行, 尽管在某些情况下并非所有功能单元都执行线程。例如, 如果线程组中包括的线程数量少于功能单元的数量, 则未使用的功能单元可能在线程组的处理期间保持空闲。在其他情况下, 线程组中的多个线程在不同时间经由同一功能单元执行。例如, 如果线程组中包括的线程数量大于功能单元的数量, 则一个或更多个功能单元可以在连续的时钟周期内执行不同的线程。

[0084] 在一个实施例中, 一组相关线程组可以在SM 332内的不同执行阶段中同时活动。一组相关线程组在本文中被称作“合作线程阵列”(CTA)或“线程阵列”。相同CTA中的线程或不同CTA中的线程通常可以通过一个或更多个L1高速缓存(包括那些SM 332、L1.5高速缓存350、在SM 332之间共享的一个或更多个L2高速缓存), 或通过驻留在计算机系统100中包括的任何存储器设备上的任何共享存储器、全局存储器或其他类型的存储器相互共享中间数据和/或输出数据。在一个实施例中, L1.5高速缓存350可以被配置为高速缓存将由线程在SM 332上执行的指令。

[0085] 给定的线程组或CTA中的每个线程通常都分配在执行期间可以访问的唯一的线程标识符(线程ID)。分配给给定的线程的线程ID可以定义为一维或多维数值。给定的线程的执行和处理行为可能因线程ID而异。例如, 线程可以基于线程ID确定要处理输入数据集的哪一部分和/或要写入输出数据集的哪一部分。

[0086] 在一个实施例中, 每线程指令序列可以包括至少一个指令, 该指令定义给定线程与一个或更多个其他线程之间的协作行为。例如, 每线程指令序列可以包括一条指令, 该指令在被执行时将给定线程挂起在特定的执行状态, 直到其他部分或全部线程达到相应的执行状态为止。在另一示例中, 每线程指令序列可以包括一条指令, 该指令在被执行时使给定线程将数据存储在其他部分或全部线程可以访问的共享存储器中。在又一个示例中, 每线

程指令序列可以包括一条指令,该指令在被执行时使给定线程自动读取和更新存储在其他部分或全部线程可以访问的共享存储器中的数据,具体取决于那些线程的线程ID。在又一个示例中,每线程指令序列可以包括一条指令,该指令在被执行时使给定线程基于对应的线程ID来计算共享存储器中的地址,以便从该共享存储器中读取数据。利用上述同步技术,第一线程可以将数据写入共享存储器中的给定位置,并且第二线程可以以可预测的方式从共享存储器中读取该数据。因此,可以将线程配置为在给定线程组内或给定CTA内或跨不同线程组或不同CTA的线程内实现多种数据共享模式。在各种实施例中,以计算统一设备架构(CUDA)编程语言编写的软件应用程序描述了在GPC 242上执行的线程的行为和操作,包括任何上述行为和操作。

[0087] 在操作中,管线管理器310通常协调GPC 242中处理任务的并行执行。管线管理器310从任务/工作单元234接收处理任务,并将那些处理任务分配给TPC 330以通过SM 332执行。给定的处理任务通常与可以在一个或更多个TPC 330中的一个或更多个SM 332上执行的一个或更多个CTA相关联。在一个实施例中,给定任务/工作单元234可以通过启动针对一个或更多个特定TPC 330的一个或更多个CTA将一个或更多个处理任务分配给GPC 242。管线管理器310可以从任务/工作单元234接收已启动的CTA,并将CTA传输到相关的TPC 330,以通过TPC 330中包含的一个或更多个SM 332执行。在执行给定的处理任务期间或之后,每个SM 332都会生成输出数据,并根据当前配置和/或当前处理任务的性质将输出数据传输到各个位置。

[0088] 在与通用计算或图形处理有关的配置中,SM 332可以将输出数据传输到工作分配交叉开关320,并且工作分配交叉开关320然后将输出数据路由到一个或更多个GPC 242以附加处理或将输出数据路由到交叉开关单元250以进一步路由。交叉开关单元250可以将输出数据路由到给定分区单元262中包括的L2高速缓存、PPU存储器270或系统存储器120以及其他目的地。管线管理器310通常基于与输出数据相关联的处理任务来协调由工作分配交叉开关320执行的输出数据的路由。

[0089] 在特定于图形处理的配置中,SM 332可以将输出数据传输到纹理单元340和/或preROP370。在一些实施例中,preROP 370可以实现在3D图形API中指定的部分或全部光栅操作,在这种情况下,preROP 370实现通过ROP 410进行的部分或全部操作。纹理单元340通常执行纹理映射操作,包括例如确定纹理样本位置、读取纹理数据以及过滤纹理数据等。PreROP 370通常执行面向光栅的操作,包括例如组织像素颜色数据并执行颜色混合优化。PreROP 370还可以执行地址转换,并将从SM 332接收的输出数据定向到分区单元262中的一个或更多个光栅运算处理器(ROP)单元。

[0090] 在以上任何配置中,一个或更多个PM 360监视GPC 242的各个组件的性能,以便向用户提供性能数据,和/或平衡跨线程组的计算、图形和/或存储器资源的利用,和/或平衡这些资源与其他GPC 242的资源的利用。此外,在以上任何配置中,SM 332和GPC 242中的其他组件都可以通过MMU 300与存储器接口260进行存储器访问操作。MMU 300通常代表GPC 242及其中包括的组件将输出数据写入各种存储空间和/或从各种存储空间读取输入数据。MMU 300被配置为经由一组页表项(PTE)和一个或更多个可选地址转换后备缓冲器(TLB)将虚拟地址映射为物理地址。MMU 300可以在L1.5高速缓存350中高速缓存各种数据,包括从存储器接口260返回的读取数据。在所示的实施例中,MMU 300在外部耦合到GPC 242并且可

以潜在地与其他GPC 242共享。在其他实施例中, GPC 242可以包括MMU 300的专用实例, 其提供对包括在存储器接口260中的一个或多个分区单元262的访问。

[0091] 图4是根据各种实施例的包括在图2的PPU 200中的分区单元262的框图。如图所示, 分区单元262包括L2高速缓存400、帧缓冲器 (FB) DRAM接口410、光栅运算处理器 (ROP) 420和一个或多个PM 430。L2高速缓存400耦合在FB DRAM接口410、ROP 420和PM 430之间。

[0092] L2高速缓存400是执行从交叉开关单元250和ROP 420接收的加载和存储操作的读/写高速缓存。L2高速缓存400将读取未中和紧急写回请求输出到FB DRAM接口410以处理。L2高速缓存400还将脏更新发送到FB DRAM接口410以机会处理。在一些实施例中, 在操作期间, PM 430监视L2高速缓存400的利用, 以便在不同的GPC 242和PPU 200的其他组件之间公平地分配存储器访问带宽。FB DRAM接口410直接与特定的DRAM 272接口以执行存储器访问操作, 包括将数据写入DRAM 272和从DRAM 272读取数据。在一些实施例中, 这组DRAM 272被分在多个DRAM芯片中, 其中多个DRAM芯片的一部分对应于每个DRAM 272。

[0093] 在与图形处理有关的配置中, ROP 420执行光栅操作以生成图形数据。例如, ROP 420可以对z或颜色数据执行模板操作、z测试操作、混合操作以及压缩和/或解压缩操作等。ROP 420可以被配置为生成各种类型的图形数据, 包括像素数据、图形对象、片段数据等。ROP 420还可以将图形处理任务分配给其他计算单元。在一个实施例中, 每个GPC 242包括专用ROP 420, 其代表相应的GPC 242执行光栅操作。

[0094] 本领域技术人员将理解, 图1-图4中描述的架构绝不限本实施例的范围, 并且本文公开的技术可以在任何适当配置的处理单元上实施, 该处理单元包括但不限于, 一个或多个CPU、一个或多个多核CPU、一个或多个PPU 200、一个或多个GPC 242、一个或多个GPU或其他专用处理单元等, 而不脱离本发明实施例的范围和精神的情况下。

[0095] 硬件资源的逻辑分组

[0096] 图5是根据各种实施例的图2的PPU中包括的各种PPU资源的框图。如图所示, PPU资源500包括系统管道230 (0) 至230 (7)、控制交叉开关和SMC仲裁器510、特权寄存器接口 (PRI) 集线器512、GPC 242、交叉开关单元250和L2高速缓存400。L2高速缓存400此处描绘为“L2缓存切片”的集合, 每个切片对应于DRAM 262的不同区域。系统管道230、GPC 242和PRI集线器512通过控制交叉开关和SMC仲裁器510耦合在一起。GPC242和L2高速缓存400的各个片段经由交叉开关单元250耦合在一起。在本文讨论的示例中, PPU资源500包括八个系统管道230、八个GPC 242以及特定数量的其他组件。然而, 本领域技术人员将理解, PPU资源500可以包括这些组件的任何技术上可行的数量。

[0097] 每个系统管道230通常包括PBDMA 520和522、前端上下文交换机 (FECS) 530、计算 (COMP) 前端 (FE) 540、调度器 (SKED) 550和CUDA工作分配器 (CWD) 560。PBDMA 520和522是管理设备驱动器122和PPU 200之间的通信的硬件存储控制器。FECS 530是管理上下文切换的硬件单元。计算FE 540是准备处理计算任务以供执行的硬件单元。SKED 550是调度处理任务以执行的硬件单元。CWD 560是被配置为将一个或多个线程网格排队并分派到一个或多个GPC 242以执行一个或多个处理任务的硬件单元。在一个实施例中, 可以在CUDA程序中指定给定的处理任务。通过以上组件, 系统管道230可以被配置为执行和/或管理通用计算操作。

[0098] 系统管道230 (0) 还包括图形前端 (FE) 单元542 (显示为GFX FE 542)、状态改变控制器SCC 552和图元分配器阶段A/阶段B单元 (PDA/PDB) 562。图形FE 542是准备图形处理任务以执行的硬件单元。SCC 552是管理具有不同API状态 (例如,着色器程序、着色器使用的常量以及如何采样纹理) 的工作并行化,以维护API状态的有序应用的硬件单元,即使图元未按顺序处理。PDA/PDB 562是将图元 (例如,三角形、直线、点、四边形、网格等) 分布到GPC 242的硬件单元。通过这些附加组件,系统管道230 (0) 可以进一步被配置为执行图形处理操作。在各种实施例中,部分或全部系统管道230可以被配置为包括与系统管道230 (0) 类似的组件,并且因此能够执行通用计算操作或图形处理操作。可选地,在各种其他实施例中,部分或全部系统管道230可以被配置为包括与系统管道230 (1) 至230 (7) 类似的组件,并且因此能够仅执行通用计算操作。通常,图2的前端232可以被配置为包括计算FE 540、图形FE 542或计算FE 540和图形FE 542两者。因此,出于一般考虑,下文中参考计算FE 540和图形FE 542中的一个或两个来引用前端232。

[0099] 控制交叉开关和SMC仲裁器510促进系统管道230与GPC 242之间的通信。在某些配置中,一个或更多个特定的GPC 242被可编程分配以代表特定的系统管道230执行处理任务。在这样的配置中,控制交叉开关和SMC仲裁器510被配置为在任何给定的GPC 242和对应的系统管道230之间路由数据。PRI集线器512通过CPU 110和/或PPU 200单元提供对一组特权寄存器的访问,以控制PPU 200的配置。可以通过PRI寄存器来配置PPU 200的寄存器地址空间,并且这样,PRI集线器212用于配置通用PRI地址空间和为每个系统管道230分别定义的PRI地址空间之间的PRI寄存器地址的映射。此PRI地址空间配置提供了从SMC引擎向多个PRI寄存器进行广播的功能,以下结合图7进行描述。GPC 242以前述方式经由交叉开关单元250将数据写入L2高速缓存400并且从L2高速缓存400读取数据。在一些配置中,每个GPC 242被分配从L2高速缓存400派生的L2切片的单独集合,并且任何给定的GPC 242可以对相应的L2切片的集合执行写/读操作。

[0100] 以上讨论的任何PPU资源500可以被逻辑地分组或划分为一个或更多个PPU分区,每个分区以与PPU 200整体相同的方式操作。具体而言,给定的PPU分区可以配置有足够的计算、图形和存储器资源,以执行可以由PPU 200执行的任何技术上可行的操作。如何将PPU资源500逻辑地分组为分区的示例在下面结合图6进行了详细说明。

[0101] 图6是根据各种实施例的图1的管理程序如何将PPU资源逻辑地分组为PPU分区集的示例。如图所示,PPU分区600包括一个或更多个PPU切片610。具体而言,PPU分区600 (0) 包括PPU切片610 (0) 至610 (3), PPU分区600 (4) 包括PPU切片610 (4) 和610 (5), PPU分区600 (6) 包括PPU切片610 (6), 并且PPU分区600 (7) 包括PPU切片610 (7)。在本文讨论的示例中,PPU分区600包括所示的特定数量的PPU切片610。但是,在其他配置中,PPU分区600可以包括其他数量的PPU切片610。

[0102] 每个PPU切片610包括从一个系统管道230派生的各种资源,包括PBDMA 520和522、FECS 530、前端232、SKED 550和CWD 560。每个PPU切片610还包括GPC 242、L2切片集620以及DRAM 272的对应部分 (这里未示出)。给定的PPU切片610内包括的各种资源赋予足够的功能,以使任何给定的PPU切片610可以执行PPU 200能够执行的至少一些通用计算和/或图形处理操作。

[0103] 例如,PPU切片610可以经由前端232接收处理任务,然后调度那些处理任务以经由

SKED 550执行。然后,CWD 560可以发布线程网格以在GPC 242上执行那些处理任务。GPC 242可以以上文结合图3所述的方式并行执行多个线程组。PBDMA 520和522可以代表PPU切片610中包含的各种组件执行存储器访问操作。在某些实施例中,PBDMA 520和522从存储器中获取命令,并将命令发送到FE 232以处理。根据需要,PPU切片610的各种组件可以向对应的L2缓存切片集620写入数据和从中读取数据。PPU切片610的组件还可以根据需要进行接口,包括I/O单元210和/或PCE 222等。当在PPU切片610中包含的各种资源上对一个或更多个VM进行时间切片时,FECS 530可以执行上下文切换操作。

[0104] 在所示的实施例中,每个PPU切片610包括从系统管道230派生的资源,其被配置为协调通用计算操作。因此,PPU切片610被配置为仅执行通用处理任务。然而,在其他实施例中,每个PPU切片610可以进一步包括从系统管道230派生的资源,其被配置为协调图形处理操作,例如系统管道230(0)。在这些实施例中,PPU切片610可以被配置为另外执行图形处理任务。

[0105] 通常,每个PPU分区600是资源的硬分区,其为一个或更多个用户提供与其他PPU分区600隔离的专用并行计算环境。给定的PPU分区600包括如图所示的一个或更多个专用PPU切片610,其共同地提供了至少在某种程度上模仿PPU 200整体的总体功能所需的各种通用计算、图形处理和存储器资源。因此,给定用户可以以与当未对PPU 200进行分区时在PPU 200上执行那些相同并行处理操作的相似用户的方式在给定的PPU分区600内执行并行处理操作。每个PPU分区600对于其他PPU 600是故障不敏感的,并且每个PPU分区可以独立于其他PPU分区600并且在不中断其他PPU分区600的操作的情况下复位。如下面更详细描述,这里未具体示出的各种资源与那些不同的PPU分区600的大小成比例地公平地分布在不同的PPU分区600上。

[0106] 在本文讨论的PPU分区600的示例配置中,PPU分区600(0)被分配八个PPU切片610中的四个,并且因此被提供了PPU资源500的一半,包括各种类型的带宽,例如存储器带宽。因此,PPU分区610(0)将被约束为消耗可用系统存储带宽的一半、可用PPU存储带宽的一半、可用PCE 212带宽的一半等。类似地,PPU分区600(4)被分配了八个PPU切片610中的两个,因此被提供了PPU资源500的四分之一。因此,PPU分区610(4)将被限制为消耗可用系统存储带宽的四分之一、可用PPU存储带宽的四分之一、可用PCE 212带宽的四分之一等。其他PPU分区600(6)和600(7)将以类似的方式被约束。本领域技术人员将理解如何利用PPU分区600的任何其他技术上可行的配置来实现上述示例性分区和相关联的资源供应。

[0107] 在一些实施例中,每个PPU分区600为一个虚拟机(VM)执行上下文。在一个实施例中,PPU 200可以实现各种性能监视器和节流计数器,其记录每个PPU分区600正在消耗的本地和/或系统范围的资源量,以便在所有PPU分区600上保持成比例的资源消耗。可以通过将相同部分的L2切片400分配给PPU分区600来实现将适当部分的PPU存储带宽分配给PPU分区600。

[0108] 通常,PPU分区600可以被配置为相对于彼此以功能隔离的方式操作。如本文所提及的,应用于PPU分区集600的术语“功能隔离”通常表示任何PPU分区600可以独立于PPU分区集600中的任何其他PPU分区600所执行的任何操作来执行一个或更多个操作,而不干扰任何PPU分区集600中的任何其他PPU分区600所执行的任何操作,并且不受PPU分区集600中的任何其他PPU分区600所执行的任何操作干扰。

[0109] 给定的PPU分区600可以配置为同时执行与多个处理上下文相关联的处理任务。术语“处理上下文”或“上下文”通常指在执行一个或多个线程期间的硬件、软件和/或存储器资源的状态,并且通常对应于CPU 110上的一个进程。与给定的PPU分区600相关联的多个处理上下文可以是不同的处理上下文或相同处理上下文的不同实例。当以这种方式配置时,分配给给定的PPU分区600的特定PPU资源被逻辑地分组到单独的“SMC引擎”,其执行与单独的处理上下文相关联的单独的处理任务,如下面结合图7所更详细地描述的。因此,给定的处理上下文可以包括在SMC引擎700中执行的硬件设置、每线程指令和/或与线程相关联的寄存器内容。

[0110] 图7示出了根据各种实施例的图1的管理程序如何配置PPU分区集以实现一个或多个同时多上下文(SMC)引擎的示例。如图所示,PPU分区600包括一个或多个SMC引擎700。特别地,PPU分区600(0)包括SMC引擎700(0)和700(2),PPU分区600(4)包括SMC引擎700(4),PPU分区600(6)包括SMC引擎700(6),并且PPU分区600(7)包括SMC引擎700(7)。每个SMC引擎700可以被配置为执行一个或多个处理上下文和/或被配置为执行与给定的处理上下文相关联的一个或多个处理任务,以类似于PPU 200整体的方式。

[0111] 给定的SMC引擎700通常包括与至少一个PPU切片610相关联的计算和存储器资源。例如,SMC引擎700(6)和700(7)包括分别与PPU切片610(6)和610(7)相关联的计算和存储器资源。每个SMC引擎700还包括一组虚拟引擎标识符(VEID)702,它们在本地产引用一个或多个子上下文,其中VEID与用于选择虚拟地址空间的虚拟地址空间标识符关联,并且可以与其相同,其中虚拟地址空间的页面由MMU1600管理的页面表描述。给定的SMC引擎700还可以包括与多个PPU切片610相关联的计算和存储器资源。例如,SMC引擎700(0)包括与PPU切片610(0)和610(1)相关联的计算资源,但不利用系统管道230(1)。SMC引擎700(0)包括并利用四个PPU切片610(0)、610(1)、610(2)和610(3)中的L2切片。在一些实施例中,相同PPU分区600内的SMC引擎700共享PPU分区600内的L2切片。在该配置中,所示的PPU分区600(1)的系统管道230(1)未使用,因为SMC引擎通常一次运行一个处理环境,并且一个处理环境仅需要一个系统管道230。SMC引擎700(2)以与SMC引擎700(0)类似的方式配置。可以将任何特定PPU分区600中包含的存储器资源示为PPU存储器分区710,该存储器资源可以分配给和/或分布在该特定PPU分区700中的任何一个或多个SMC引擎700。

[0112] 给定的PPU存储器分区710包括PPU分区600中包括的L2切片集合以及DRAM 272的对应部分。通常,如果那些SMC引擎700包含在同一PPU分区600中,则多个SMC引擎700共享一个PPU内存分区710。每个SMC引擎700的分配被提供给在那些SMC引擎700上运行的上下文,并且基于页面实现PPU存储器分区710内的分配。

[0113] 每个SMC引擎700可以被配置为在任何给定时间独立执行与一个处理上下文相关联的处理任务。因此,具有两个SMC引擎700(0)和700(2)的PPU分区600(0)可以被配置为在任何给定时间同时执行与两个单独的处理上下文相关联的处理任务。另一方面,每个分别包括一个SMC引擎700(4)、700(6)和700(7)的PPU分区600(4)、600(6)和600(7)可以被配置为一次执行与一个处理上下文相关联的处理任务。在一些实施例中,在不同的PPU分区600中的SMC引擎700上运行的上下文可以通过共享一个或两个PPU分区600中的一个或多个页面来共享数据。

[0114] 任何给定的SMC引擎700可以进一步被配置为在不同的时间间隔内对不同的处理

上下文进行时间切片。因此,每个SMC引擎700可以独立地支持与多个处理上下文相关联的处理任务的执行,尽管不必同时执行。例如,SMC引擎700(6)可以在四个不同的时间间隔上对四个不同的处理上下文进行时间切片,从而允许与这四个处理上下文相关联的处理任务在PPU分区600(6)内执行。在一些实施例中,VM在一个或多个PPU分区600上按时间切片。例如,PPU分区600(0)可以在两个VM之间进行时间切片,其中每个VM同时执行两个处理上下文,每个SMC引擎700(0)和700(1)上一个处理上下文。在这些实施例中,优选地,在从第二VM在处理上下文中上下文切换之前,上下文从第一VM切换出所有处理上下文,当在PPU分区600(0)上运行的处理上下文共享PPU分区600(0)内的L2切片400时,这是有利的。

[0115] 在一个实施例中,给定的VM可以与GPU功能ID(GFID)相关联。给定的GFID可以包含一个或多个位(bit),这些位对应于与VM在其中执行的硬件相关联的物理功能(PF)。给定的GFID还可以包括一组位,这些位对应于唯一分配给VM的虚拟功能(VF)。除其他用途外,给定的GFID可用于将错误路由到与VM的访客操作系统相对应的位置。

[0116] 不同的PPU分区600内的SMC引擎700通常彼此隔离运行,因为如前所述,每个PPU分区600是PPU资源500的硬分区。同一PPU分区600内的多个SMC引擎700通常可以彼此独立地运行,特别是可以彼此独立地上下文切换。例如,PPU分区600(0)内的SMC引擎700(0)相对于SMC引擎700(2)可以独立且异步地进行上下文切换。在一些实施例中,同一PPU分区600内的多个SMC引擎700可以同步上下文切换,以支持某些操作模式,例如两个VM之间的时间切片。

[0117] 通常,图1的设备驱动器122和管理程序124以到目前为止描述的方式互操作以对PPU 200分区。此外,设备驱动器122和管理程序124互操作以将每个PPU分区600配置成一个或多个SMC引擎700。这样,设备驱动器122和管理程序124配置DRAM 272和/或L2高速缓存400以便将L2切片集合分区成每个都是SMC存储器分区710的组,如下面结合图8更详细地描述的。在一些实施例中,管理程序124响应于系统管理员的控制,以允许系统管理员创建PPU分区的配置。这些PPU分区600被切换到VM的访客OS 916,并且访客OS 916随后向管理程序124发送请求以将相关联的PPU分区600配置为一个或多个SMC引擎700。在一些实施例中,因为添加了足够的隔离以防止一个访客OS影响另一个访客OS的PPU分区700,访客OS可以在PPU分区700内直接配置SMC引擎700。

[0118] 图8A是根据各种实施例的图7的DRAM的更详细的图示。如图所示,可通过L2切片800访问DRAM 272,其包括图7的DRAM 272(0)至272(7)中的每一个。每个L2切片800对应于L2高速缓存400的不同部分,并被配置为访问DRAM 272中的位置的对子集。通常,DRAM 272的分区对应于原始2D地址空间,其组织方式与本文所示的DRAM 272相似。

[0119] 同样如图所示,DRAM 272被分成顶部部分810、可分区部分820和底部部分830。顶部部分810和底部部分830是分别从所有DRAM 272(0)到272(7)的顶部部分和底部部分派生的存储分割。设备驱动器122、管理程序124和其他系统级实体可以访问顶部部分810和/或底部部分830,在某些实施例中,PPU分区600无法访问这些部分。另一方面,可分区部分820通常被指定供一般PPU分区600使用,并且特别是由SMC引擎700使用。在一些实施例中,安全数据驻留在顶部部分810或底部部分830中,并且可由所有PPU分区600访问。在一些实施例中,顶部部分810或底部部分830用于VM无法访问的管理程序数据。

[0120] 在所示的示例性存储器分区中,可分区部分820包括与在PPU分区600(0)内的PPU存储器分区710(0)相对应的DRAM部分822(0),与在PPU分区600(2)内的PPU存储器分区710

(4) 相对应的DRAM部分822 (4), 与在PPU分区600 (6) 内的PPU存储器分区710 (6) 相对应的DRAM部分822 (6), 和与在PPU分区600 (7) 内的PPU存储器分区710 (7) 相对应的DRAM部分822 (7)。每个DRAM部分822与对应于一组L2高速缓存切片800的地址的中间部分相对应。给定的DRAM部分822可以进一步细分, 以便为执行与不同的处理上下文相关联的处理任务的不同VM提供单独的L2高速缓存切片集。例如, DRAM部分822 (4) 可以被细分为两个或更多个区域, 以支持执行与两个或更多个处理上下文相关联的处理任务的两个或更多个VM。一旦配置并使用了DRAM部分, 通常一次由一个运行在PPU分区600上的VM使用它。

[0121] 在操作中, 设备驱动器122和管理程序124以相对平衡的方式经由对应于所有L2高速缓存切片800的地址范围的顶部部分和底部部分在顶部部分810和/或底部部分830内执行存储器访问操作, 从而按比例惩罚每个L2切片800上的存储带宽。在一些实施例中, SMC引擎700经由L2高速缓存切片800执行对系统存储器120的存储器访问操作, 其吞吐量由节流计数器840控制。每个节流计数器840监视当SMC引擎700经由与对应的PPU存储器分区710相关联的L2高速缓存切片800访问系统存储器120时消耗的存储器带宽, 以便为每个PPU分区600提供成比例的存储器带宽。如所讨论的, 与那些PPU分区600的配置成比例地向PPU分区提供了对各种系统范围的资源的访问。在所示的示例中, PPU分区600 (0) 被分配了PPU资源500的一半, 并且因此被分配了可分区部分820的一半 (显示为DRAM部分822 (0)), 并且相应地, 将可用的存储器带宽的一半分配给系统存储器120。下面结合图19-图24更详细地描述DRAM 272的分区。

[0122] 图8B示出了根据各种实施例的如何寻址图8B的各种DRAM部分。如图所示, 一维 (1D) 系统物理地址 (SPA) 空间850包括对应于顶部部分810的顶部地址852, 可分区地址854分为地址区域856并且对应于DRAM部分822, 以及对应于底部部分830的底部地址858。顶部地址852在所有L2切片800上被模糊化 (即, 基于SPA地址的伪随机交织), 并且对应于那些L2切片的顶部部分。底部地址858在所有的L2切片800上被模糊化, 并且对应于那些L2切片的底部部分。通常, 只有系统级实体 (例如, 管理程序124) 和/或通过物理功能 (PF) 进行操作的任何实体可以访问顶部地址852和底部地址858。可分区地址854被分配给PPU分区600。特别地, 地址区域856 (0) 被分配给PPU分区600 (0), 地址区域856 (4) 被分配给PPU分区600 (4), 地址区域856 (6) 被分配给PPU分区600 (6), 地址区域856 (7) 被分配给PPU分区600 (7)。地址区域856只能由在相应的PPU分区600中执行的一个或更多个SMC引擎700访问。

[0123] 总体上参考图5-图8B, 以上对PPU资源500进行分区的方法支持多种使用场景, 其中包括单租户和多租户使用场景。在单租户使用场景中, 可以对PPU 200进行分区, 以为与单个租户相关联的不同用户提供对PPU资源的独立访问。例如, 与给定租户相关联的不同用户可以在不同的PPU分区600上执行不同的预定工作量。在单租户使用场景中, 可以向单个实体提供对整个PPU资源500的访问。在多个租户使用场景中, 可以对PPU 200进行分区, 以为与一个或更多个不同租户相关联的一个或更多个用户提供对PPU资源的独立访问。在多租户使用场景中, 可以为多个实体提供对不同的PPU分区600的访问, 这些分区包括PPU资源500的不同部分。

[0124] 在任何使用场景下, 设备驱动器122和管理程序124互操作以执行两步过程, 该过程首先涉及将PPU 200分为PPU分区600, 并且其次涉及将这些PPU分区600配置为SMC引擎700。下面结合图9-图10进行更详细的说明。

[0125] 配置硬件资源逻辑分组的技术

[0126] 图9是示出了根据各种实施例的图1的管理程序如何分区和配置PPU的流程图。如图所示,管理程序环境900包括由管理程序信任边界930彼此分隔的访客环境910和主机环境920。访客环境910包括系统管理接口(SMI)912、核心驱动器914和访客操作系统(OS)916。主机环境920包括SMI 922、虚拟GPU(vGPU)插件924、主机OS 926和核心驱动程序928。驻留在管理程序信任边界930上方的访客环境910中包括的模块通常以比驻留在管理程序信任边界930下方的主机环境920中包括的模块更低的权限级别执行,包括同一模块的重复实例,例如SMI 912和SMI 922。管理程序124以核心级权限集执行,并且可以向任何所示的模块授予适当的权限。在一些实施例中,VM与访客环境910之间存在一一对应关系;并且当多个虚拟机没有上下文切换到PPU分区600时,访客环境和PPU分区之间通常存在一一对应关系。

[0127] 在操作中,PPU 200的管理员用户通过主机环境920和主机OS 926与PPU 200进行交互,以配置PPU分区600。特别地,管理员用户将分区输入904提供给SMI 922。作为响应,SMI 922向核心驱动程序928发出“创建分区”命令,指示PPU分区600的目标配置。核心驱动程序928将“创建分区”命令传输到PPU 200中的主机接口220,以对各种PPU资源500进行分区。以这种方式,管理员用户可以将PPU 200初始化为具有PPU分区600的特定配置。通常,管理员用户可以不受限制地访问PPU 200。例如,管理员用户可以是PPU 200所在的数据中心的系统管理员。管理员用户可以是多个PPU 200所驻留的数据中心的系统管理员。管理员用户以所描述的方式分区PPU 200,以准备要由各个访客用户独立配置和使用的各个PPU分区600,如下文更详细地描述的。

[0128] PPU 200的访客用户经由在访客环境910内执行的VM与特定的“访客”PPU分区600交互,以便在该访客PPU分区600内配置SMC引擎700。具体地,访客用户提供配置输入902到SMI 912。然后SMI 912向核心驱动器914发出“配置分区”命令,指示SMC引擎700的目标配置。核心驱动器914通过访客OS 916跨管理程序信任边界930向vGPU插件924发送“配置分区”命令。vGPU插件924向核心驱动程序928发出各种VM调用。核心驱动程序928将“配置分区”命令发送到PPU 200中的主机接口220,以配置访客PPU分区600的各种资源。以这种方式,访客用户可以将给定的PPU分区600配置为具有SMC引擎700的特定配置。通常,访客用户只能访问与访客PPU分区600相关联的PPU资源500的一部分。例如访客用户可以是购买了对一部分PPU 200的访问权的PPU 200所驻留的数据中心的客户。在一个实施例中,访客OS 916可以被配置为具有足够的安全措施以允许每个访客OS 916配置相应的PPU分区600而无需主机环境920和/或管理程序124的参与。

[0129] 图10是根据各种实施例的用于代表一个或更多用户对PPU进行分区和配置的方法步骤的流程图。尽管结合图1至图9的系统描述了方法步骤,但是本领域技术人员将理解,配置成以任何顺序执行该方法步骤的任何系统都落在本实施例的范围内。

[0130] 如图所示,方法从步骤1000开始,其中管理程序124通过主机环境920接收分区输入904。主机环境920以提升的权限级别执行,从而允许管理员用户直接与PPU 200进行交互。分区输入904指定PPU分区600的目标配置,包括PPU分区600的期望大小和布置。在一个实施例中,可以经由主机环境从管理员用户接收分区输入。

[0131] 在步骤1004,管理程序124基于在步骤1002接收的分区输入904在PPU 200内生成

一个或更多个PPU分区600。特别地,管理程序124实施SMI 922以向核心驱动程序928发出“创建分区”命令。作为响应,核心驱动程序928与PPU 200的主机接口220交互以创建具有期望配置的一个或更多个PPU分区600。

[0132] 在步骤1006,管理程序124在步骤1004所生成的一个或更多个PPU分区600上分配存储器资源。特别地,通过以上讨论的“创建分区”命令,管理程序124以结合图8A的上述方式细分DRAM 272,以将DRAM 272和相应的L2高速缓存切片800的不同区域分配给一个或更多个PPU分区600。在一个实施例中,管理程序124还可以配置地址映射单元以执行分区特定的混乱操作,以经由L2高速缓存切片800提供对DRAM 272的那些不同区域的访问。以下结合图22更详细地描述该特定实施例。

[0133] 在步骤1008,管理程序124在一个或更多个PPU分区600上分配PPU计算和/或图形资源。这样做,管理程序124经由“创建分区”命令分配一个或更多个系统管道230和一个或更多个GPC 242到一个或更多个PPU分区600。在一个实施例中,管理程序124可通过将一个或更多个PPU切片610逻辑地分配给一个或更多个PPU分区600来实现步骤1006和1008,从而一起分配存储器资源和计算/图形资源。当方法1000的步骤1002、1004、1006和1008完成时,PPU 200被分区,并且然后访客用户可以配置一个或更多个PPU分区600,如下所述。

[0134] 在步骤1010,管理程序124经由访客环境910接收与第一PPU分区相关联的配置输入902。访客环境910以降低的权限级别执行,从而允许访客用户仅与第一PPU分区600进行交互。配置输入902指定第一PPU分区600内的SMC引擎700的目标配置,包括SMC引擎700的期望大小和布置。在一个实施例中,可以经由访客环境从访客用户接收配置输入。

[0135] 在步骤1012,管理程序124基于在步骤1010接收到的配置输入902,通过“配置分区”命令在第一PPU分区600内生成一个或更多个SMC引擎700。给定的SMC引擎700可以包括从一个系统管道230派生的计算和/或图形资源和从一个或更多个PPU切片610派生的一个或更多个GPC 242。给定的SMC引擎700可以访问SMC引擎700所驻留的PPU分区600中包含的PPU存储器分区710的至少一部分,其中该PPU存储器分区710包括一组或更多组L2高速缓存切片和DRAM 272的相应部分。

[0136] 在步骤1014,管理程序124跨在步骤1012生成的一个或更多个SMC引擎700分发分配给第一PPU分区600的存储器资源。一般而言,一个或更多个SMC引擎700共享第一PPU存储器分区710,如果在相同的PPU分区600中包含了这些SMC引擎700。向在这些SMC引擎700上运行的上下文提供每个SMC引擎700的分配,并基于页面实现PPU存储器分区710中的分配。在一些实施例中,VM的访客OS 916通过将存储器资源分发给在SMC引擎700上运行的上下文来执行步骤1014,SMC引擎700是属于访客环境910的PPU分区600的一部分。在其他实施例中,管理程序124执行使用PPU分区600执行用于多个VM的存储器资源分配1014,并且每个VM也分发存储器资源1024到在SMC引擎700上运行的上下文。

[0137] 在步骤1016,管理程序124在步骤1012生成的一个或更多个SMC引擎700上分发分配给第一PPU分区600的计算和/或图形资源。通过“配置分区”命令,管理程序124将访客PPU分区600中包含的系统管道230分配给每个SMC引擎700。管理程序124还向每个SMC引擎700分配一个或更多个GPC 242。当方法1000的步骤1010、1012、1014和1016完成时,第一PPU分区600被配置,并且然后访客用户可以发起对PPU分区600内的一个或更多个SMC引擎700的处理操作。

[0138] 在步骤1018,管理程序124使第一PPU分区600跨在第一PPU分区600内配置的一个或多个SMC引擎700对一个或多个VM进行时间切片。经过时间切片的VM可以独立于在一个第一PPU分区600内执行的其他VM进行操作,并且与在其他PPU分区600内执行的其他VM隔离地操作。在一个实施例中,可以跨一个或多个SMC引擎700同时对一个或多个VM进行时间切片。以这种方式,公开的技术允许分区化的PPU支持与多个不同处理上下文相关联的处理任务的并行执行。

[0139] 在一些实施例中,本文公开的技术在非虚拟化系统中操作。本领域技术人员将认识到,PPU 200或一组PPU 200上的单个OS使用模型可以使用结合VM描述的所有机制。在一些实施例中,容器对应于VM的描述,这意味着单个OS上的容器可以实现提供给本文所述的VM的处理隔离。

[0140] 划分计算资源以支持同时的多个上下文

[0141] 在各种实施例中,当管理程序124以上述方式代表管理员用户对200PPU进行分区时,管理程序124接收来自管理员用户的输入,其指示PPU分区600之间的各种边界。基于该输入,管理程序124将PPU切片610逻辑地分组到PPU分区600,分配各种硬件资源到每个PPU分区600,并协调各种其他操作,以支持给定的PPU分区600内多个处理上下文的同时实现。管理程序124还执行另外的技术来支持配置在不同PPU 200上的PPU分区600之间的处理上下文的迁移。下面结合图11-图18更详细地描述这些各种技术。

[0142] 图11示出了根据各个实施例的分区配置表的实施例,根据该实施例图1的管理程序124可以配置一个或多个PPU分区。如图所示,分区配置表1100包括分区选项0至14。分区选项0-14被描绘在PPU切片610上方。分区选项0-14中的每个跨越PPU切片610的不同分组,并且以这种方式表示不同的可能分区PPU 600。具体而言,分区选项0跨越PPU切片610(0)到610(7),并且因此表示包括所有八个PPU切片610的PPU分区600。类似地,分区选项1跨越PPU切片610(0)至610(3),并且因此表示仅包括前四个PPU切片610的PPU分区600,类似于图6-图7中所示的PPU分区600(0)。分区选项2跨越PPU切片610(4)至610(7),并且因此表示仅包括最后四个PPU切片610的PPU分区600。分区选项3、4、5和6跨越两个相邻的PPU切片610的不同的分组,而分区选项7、8、9、10、11、12、13和14仅跨越一个相应的PPU切片610。

[0143] 分区配置表1100还包括代表分区边界的不同的可能位置的边界选项1110。具体来说,边界选项1110(1)和1110(9)代表分区选项0的边界。边界选项1110(1)和1110(5)代表分区选项1的边界,而边界选项1110(5)和1110(9)代表分区选项2的边界。边界选项1110(1)、1110(3)、1110(5)、1110(7)和1110(9)表示与分区选项3、4、5和6相关联的边界。边界选项1110(1)至1110(9)代表与分区选项7至14相关联的边界。应当理解,本领域技术人员可以创建许多不同的方案来实现与配置表1100相同的功能,这可能可以是使能位的集合、预定义选择的列表或允许控制如何将PPU切片610分为PPU分区600的任何其他形式。此外,本领域技术人员将理解分区配置表1100可以包括除图11所示之外的任何技术上可行的数字或条目。

[0144] 在分区期间,在管理程序级别运行的管理程序124或设备驱动器122接收来自管理员用户的分区输入,其指示根据其应该对PPU 200进行分区的特定分区选项。管理程序124或设备驱动器122然后激活边界选项1110的特定集,该边界选项集在逻辑上将一组或更多组PPU切片610彼此隔离以实现期望的分区,如下面结合图12的示例更详细地描述的。

[0145] 图12示出了根据各种实施例的图1的管理程序124或设备驱动器122如何分区PPU以生成一个或多个PPU分区。如图所示,在分区期间,以管理程序级别运行的管理程序124或设备驱动器122从管理员用户接收的输入,其指示应该根据分区选项1、5、13和14对PPU 200进行分区(为清楚起见而强调)。作为响应,管理程序124或设备驱动器122激活边界选项1110(1)、1110(5)、1110(7)、1110(8)和1110(9)并停用其他边界选项,以便生成PPU分区600(0)、600(4)、600(6)和600(7)。在图6-图7中还示出了PPU分区600的该示例性配置。

[0146] 总体上参考图11-图12,管理程序124或设备驱动器122通过将分区选项的每个选择映射到特定的二进制值来实现上述技术,该特定的二进制值随后用于启用和禁用边界选项1110。与给定分区选项相关联的二进制值在本文中称为“混乱标识符”(swizID)。由管理程序124实现的各种swizID列在下表1中:

[0147]	分区选项	SwizID
	0	11000000011
	1	10000100011
	2	11000100001
[0148]	3	10000001011
	4	10000101001
	5	10010100001
	6	11010000001
	7	10000000111
	8	10000001101
	9	10000011001
	10	10000110001
	11	10001100001
	12	10011000001
	13	10110000001
	14	11100000001

[0149] 表1

[0150] 管理程序124基于与给定分区选项相关联的swizID为给定分区选项激活或停用边界选项1110。例如,管理程序124可以激活边界选项1110(1)和1110(3),以基于相应的swizID 10000001011根据分区选项3配置PPU 200。此swizID的位1和3分别激活边界选项1110(1)和1110(3),并且位2和4-9停用其余的边界选项。所有swizID的位0和10被设置为1以激活L2高速缓存400内的边界,如下面结合图19-图20更详细描述。管理程序124为不同的所选配置选项收集各种swizID,并跨所有收集的swizID计算OR操作,以生成定义PPU分区600的期望配置的配置swizID。配置swizID指示应激活和停用的所有边界选项1110以实现PPU分区600的期望配置。

[0151] 本领域技术人员将认识到分区选项的某些组合是不可行的。例如,分区选项0和1不能相互结合实现,因为分区选项0和1相互重叠。在分区期间,管理程序124通过自动检测分区选项的不可行组合,并且通过修改一个或多个分区选项和/或相应的swizID或省略

一个或多个分区选项和/或相应的swizID来纠正这些组合。

[0152] 另外,管理程序124可以动态检测导致某些分区选项不可行的硬件故障。例如,假设PPU切片610(0)包括非功能性GPC 242,其在制造期间被地板擦伤并熔断。在这种情况下,仅包括PPU切片610(0)的PPU分区600将缺少足够的计算资源来运行,因此将难以实施。在这种情况下,管理程序124将不允许选择分区选项7和/或使用相应的swizID,因为根据该分区选项配置的任何PPU分区600都将无法执行计算操作,因此将无法正常运行。

[0153] 在某些情况下,管理程序124可以允许某些配置选项,其中包括一定数量的非功能硬件,只要根据这种配置选项配置的PPU分区600仍然可以在一定程度上起作用。在以上示例中,管理程序124可以允许选择配置选项3,只要PPU切片610(1)包括功能GPC 242。根据配置选项3配置的任何PPU分区600仍将起作用,但与不包含任何非功能性硬件的类似PPU分区600相比,它仅包括计算资源的一半。

[0154] 在以上述方式对PPU 200进行分区之后,管理程序124将各种硬件资源分配给所得的PPU分区600。这些资源中的一些静态地分配给各个PPU切片610,并为特定操作提供专用支持,而其他资源在相同PPU分区600内或不同PPU分区600内的不同PPU切片610之间共享,如下面结合图13更详细地描述的。

[0155] 图13示出了根据各种实施例的图1的管理程序在分区期间如何分配各种PPU资源。如图所示,PCE 222(0)至222(7)耦合到PPU切片610(0)至610(7)。在该示例中,PPU 200包括PCE 222的数量等于PPU切片610的数量。因此,管理程序124可以将每个PCE 222静态地分配给不同的PPU切片610,并且配置那些PCE 222以专用方式代表相应的PPU切片610执行复制操作。

[0156] PPU 200中包括的其他硬件资源不能以上述方式静态地分配,因为这些资源可能相对稀缺。在所示的示例中,PPU 200仅包括需要在八个PPU切片610上分配的两个解码器1300。因此,管理程序124将解码器1300(0)动态地分配给包括在PPU分区600(0)中的PPU切片610(0)至610(3)。管理程序124还动态地将解码器1300(1)分配给PPU分区600(4)中包括的PPU切片610(4)和610(5)、PPU分区600(6)中包括的PPU切片610(6)和PPU分区600(7)中包括的PPU切片610(7)。

[0157] 在所示的配置中,解码器1300(0)被动态分配以专用方式对PPU分区600(0)执行解码操作,但是解码器1300(1)在PPU分区600(4),600(6)和600(7)之间共享。在各种实施例中,一个或多个性能监视器可以管理以所描述的方式共享的硬件资源的使用,以便在不同的PPU切片610之间负载平衡资源使用。管理程序124执行以上技术,以便分配PPU 200的任何技术上可行的资源到PPU分区600。

[0158] 当已经执行分区并且将PPU 200的各种资源静态或动态地分配给各个PPU切片610时,管理程序124准备允许VM开始在那里PPU分区600内执行处理任务。这样,VM可以在给定的PPU分区600内同时启动与关联于其他PPU分区600的其他处理上下文隔离的多个处理上下文,如上所述并且如下面结合图14A-图14B详细描述的。可以将未使用的PPU切片610重新分区到其他PPU分区600中,而将其他PPU切片在活动PPU分区600中使用。

[0159] 图14A根据各种实施例示出了运行多个VM的多个访客OS 916如何在一个或多个PPU分区内同时启动多个处理上下文。如图所示,访客OS 916包括与不同的PPU分区600相关联的各种处理上下文1400。处理上下文1400(0)和1400(1)与PPU分区600(0)相关联并且可

以在SMC引擎700 (0) 或SMC引擎700 (1) 上启动。在一些实施例中,一旦将处理上下文分配给SMC引擎700,它就保留在该smc引擎700上直到完成。处理上下文1400 (4) 与PPU分区600 (4) 相关联,并且可以在SMC引擎700 (4) 上启动。处理上下文1400 (6) 和1400 (6) 分别与PPU分区600 (6) 和600 (7) 相关联,并且可以分别在SMC引擎700 (6) 和700 (7) 上启动。

[0160] 如先前结合图7所述,每个SMC引擎700可以独立于执行与任何给定处理上下文1400相关联的处理任务的其他SMC引擎700来执行与给定处理上下文1400相关联的处理任务。由给定SMC引擎700结合给定处理上下文1400执行的所述处理任务,独立于由其他SMC引擎700结合任何其他处理上下文1400执行的其他处理任务进行调度。此外,如下面结合图15-图16更详细地描述的,SMC引擎700可以彼此独立地经历故障和/或错误,并且可以在不中断其他SMC引擎700的操作的情况下进行重置。

[0161] 此外,每个SMC引擎700可以被配置为执行与一个或更多个处理子上下文1410相关联的处理任务,所述一个或更多个处理子上下文1410被包含在/或派生于单个父处理上下文1400。如图所示,给定的处理上下文1400 (0) 包括一个或更多个处理子上下文1410 (0),并且给定处理上下文1400 (1) 包括一个或更多个处理子上下文1410 (1)。管理程序124配置处理子上下文1410和相应的设备驱动器。与给定的父处理上下文1400相关联的处理子上下文1410在启动了父处理上下文1400的同一SMC引擎700上启动。因此,在所示的示例中,在SMC引擎700 (0) 上启动处理子上下文1410 (0),并且在SMC引擎700 (1) 上启动处理子上下文1410 (1)。在一个实施例中,每个访客OS 916能够独立于管理程序124来配置各自的PPU分区600,并且能够不干扰其他PPU分区600的配置。

[0162] 在不使用虚拟化的某些实施例中,可以不存在管理程序124和访客OS 916,并且主机OS 926可以配置和启动处理上下文1400和处理子上下文1410,如下面结合图14B更详细地描述的。

[0163] 图14B示出了根据各种实施例的主机OS如何在一个或更多个PPU分区内同时启动多个处理上下文。如图所示,主机OS 926包括处理上下文1400和处理子上下文1410。在所示的实施例中,主机OS 926被配置为在SMC引擎700上启动处理上下文1400和处理子上下文1410,而无需涉及管理程序或其他虚拟化软件。所示的实施例可以在“裸金属”场景中实现。

[0164] 总体上参考图14A-图14B,通常不彼此独立地调度与同一父处理上下文1400中的处理子上下文1410相关联的处理任务,并且通常共享相应的SMC引擎700的资源。此外,在某些情况下,在给定的SMC引擎700中启动的处理子上下文1410可能导致故障和/或错误,其使SMC引擎700重置任何相关处理上下文1400和/或要重新启动的处理子上下文1410。向处理上下文1400和/或处理子上下文1410分配本地虚拟地址空间标识符,其派生于与PPU 200整体相关联的全局虚拟地址空间标识符1510,如下面结合图15更详细描述的。

[0165] 在一些实施例中,没有虚拟化,并且因此没有管理程序,但是对于本领域技术人员来说清楚的是,PPU 200或一组PPU 200上的单个OS使用模型可以使用所描述的所有属于VM的机制。在一些实施例中,容器对应于VM的描述,这意味着单个OS上的容器可获得本文描述中提供给VM的处理隔离。容器的示例是LXC (Linux容器) 和Docker容器,这在计算机行业中是众所周知的。例如,每个Docker容器可以对应于一个PPU分区600,因此本发明提供了在一个OS下运行的多个Docker容器之间的隔离。

[0166] 图15示出了根据各种实施例的图1的管理程序如何将虚拟地址空间标识符分配给

不同的SMC引擎。如图所示,虚拟地址空间标识符1500包括用于每个SMC引擎700的单独的虚拟地址范围。每个虚拟地址范围以零(0)开始以维持SMC引擎700之间的一致性,但是每个虚拟地址范围都对应于全局虚拟地址空间标识符1510的不同的部分。例如,分配给SMC引擎700(0)的虚拟地址空间标识符0-15对应于全局虚拟地址空间标识符0-15,但是分配给SMC引擎700(1)的虚拟地址空间标识符0-15对应于全局虚拟地址空间标识符16-31。在一个实施例中,虚拟地址空间1510的全局集合可以是虚拟地址空间或物理地址空间。在一些实施例中,还存在每个PPU分区的虚拟地址空间标识符,以便VM的客户OS对其拥有的所有SMC引擎700具有从零开始的虚拟地址空间标识符集合。

[0167] 管理程序124根据SMC引擎700从其分配了资源的PPU切片610的数量,向给定的SMC引擎700分配一定范围的虚拟地址空间标识符。在所示的示例中,管理程序124将虚拟地址空间标识符0-15分配给SMC引擎700(0),将虚拟地址空间标识符0-15分配给SMC引擎700(1),并将虚拟地址空间标识符0-15分配给SMC引擎700(4)。管理程序124向SMC引擎700(0)、700(1)和700(4)分配16个虚拟地址空间标识符,因为这些SMC引擎从两个PPU切片610提取资源,如图7所示。相比之下,管理程序124将虚拟地址空间标识符0-7分配给SMC引擎700(6)和700(7),因为这些SMC引擎700分别从一个PPU切片610提取资源。管理程序124可以进一步将分配给给定的SMC引擎700的虚拟地址空间标识符细分,以支持多个处理上下文1400。例如,管理程序124可以将分配给SMC引擎700(0)的虚拟地址空间标识符0-15细分为两个范围0-7和0-7,每个都可以分配给不同的处理上下文1400。此示例示出了全局虚拟地址空间标识符1510如何按0-15、16-31、32-47、48-55和55-63的组成比例地分布。在一些实施例中,虚拟地址空间标识符是唯一的,因此上述示例将具有虚拟空间标识符0-15、16-31、32-47、48-55和55-63,而不是0-15、0-15、0-15、0-7和0-7,如图15所示。在某些实施例中,全局虚拟地址空间标识符1510的分配与PPU切片610的数量不成比例,并且管理程序可以自由分配全局虚拟地址空间标识符1510的任何子集到PPU分区600或SMC引擎700。

[0168] 管理程序124以所描述的方式分配虚拟地址空间标识符,以允许不同的SMC引擎700执行与任何给定处理上下文1400相关联的处理任务,而无需重新映射那些处理任务所指定的虚拟地址。因此,管理程序124可以在SMC引擎700之间动态地迁移处理上下文1400,而无需对那些处理上下文进行重大改变。在执行与给定处理上下文1400相关联的各种处理任务期间,任何给定的SMC引擎700有时会遇到故障,并被配置为使用本地分配的虚拟地址报告这些故障,如下面结合图16所更详细描述。迁移发生之后,迁移的处理上下文仍使用相同的虚拟地址空间标识符1500,但是这些标识符可能对应于不同的全局虚拟地址空间标识符1510。

[0169] 图16示出了根据各种实施例的在减轻故障时存储器管理单元如何将本地虚拟地址空间标识符1500转换为全局虚拟地址空间标识符1510。如图所示,如先前所讨论的,在执行期间,SMC引擎700可以经历故障和/或错误并且彼此独立地崩溃。在所示的示例中,SMC引擎700(1)遇到错误,并导致将本地故障标识符1610输出到存储器管理单元(MMU)1600。SMC引擎700对未映射页面的访问会导致MMU生成故障,还会导致本地故障标识符。

[0170] MMU 1600维护本地虚拟地址空间标识符1500与全局虚拟地址空间标识符1510之间的映射。基于该映射,MMU 1600生成全局故障标识符1620并将全局故障标识符1620发送给客户OS 916(0)。响应于接收全局故障标识符1620,访客OS 916(0)可以在不中断任何其

他SMC引擎700的操作的情况下重置SMC引擎700 (1), 并且然后重新启动处理上下文1400 (1)。通过这种方法, 每个SMC引擎700以不同的虚拟地址空间标识符集合运行, 其从零开始并且跨越可能相似的范围, 但是对应于全局存储器的不同部分。因此, 全局虚拟地址空间标识符1510可以在SMC引擎700之间划分, 但是保留专用地址空间的外观。在一些实施例中, 对于整个PPU分区600, 故障标识符1620可以是从小开始的。在其他实施例中, 故障标识符1620可以是SMC引擎700的标识符和虚拟地址空间标识符1500。

[0171] 在一个实施例中, 全局故障标识符1620可以被报告给管理程序124, 并且管理程序124可以执行各种操作来解决相关联的故障。在另一个实施例中, 一些类型的故障可以被报告给相关联的客户OS 916, 而其他类型的故障 (例如在DRAM 272的顶部部分810或底部部分830内发生的硬错误) 可以被报告给管理程序124。响应于这种故障, 管理程序124可以重置部分或全部SMC引擎700。在各种其他实施例中, 给定的全局故障标识符1620可以被虚拟化, 因此不直接对应于真实的全局标识符。在操作中, MMU 1600可以基于与那些VM相关联的GFID将故障路由到适当的VM。上面结合图7讨论了GFID。

[0172] 总体上参考图15-图16, 管理程序124可以实现与上述技术类似的技术, 以将标识符分配给与每个PPU分区600和/或每个SMC引擎700相关联的各种硬件资源。例如, 管理程序124可以为给定的PPU分区600中包括的每个GPC 242分配从小 (0) 开始的本地GPC ID范围的本地GPC标识符 (GPC ID)。每个本地GPC ID将对应于不同的全局GPC ID。可以用任何PPU资源来实现该方法, 以便维护在任何给定的PPU分区600和/或SMC引擎700中内部一致的一组标识符。如上所述, 该方法促进在SMC引擎700之间迁移处理上下文1400, 并且进一步允许处理上下文1400在不同的PPU 200之间迁移。

[0173] 当管理程序124在驻留在不同PPU 200上的不同SMC引擎700之间迁移处理上下文1400时, 管理程序124执行本文中称为“软地板清扫”的技术, 以便为目标PPU 200配置与源PPU 200相似的硬件资源。如下面结合图17更详细地描述的。

[0174] 图17示出了根据各种实施例的当在不同PPU上的SMC引擎之间迁移处理上下文时, 图1的管理程序如何实现软地板清扫。如图所示, 计算环境1700 (0) 包括管理程序124 (0) 的实例和PPU分区600 (0)。PPU分区600 (0) 配置有SMC引擎700 (0)。SMC引擎700 (0) 执行与处理上下文1710相关联的处理任务。向SMC引擎700 (0) 分配资源1720 (0) 和1720 (1), 但是资源1720 (1) 不起作用。这样, 在制造期间, 资源1720 (1) 被熔断 (“地板清扫”)。资源1720可以是到目前为止描述的任何计算、图形或存储器资源。例如, 给定资源1720可以是GPC 242、GPC 242内的TPC 330、TPC 330内的SM 332、GFX FE 542或L2高速缓存切片800等。

[0175] 在各种情况下, 管理程序124 (0) 可以确定应将处理上下文1710从计算上下文1700 (0) 迁移到计算上下文1700 (1)。例如, 计算上下文1700 (0) 可以被调度用于计划的停机时间, 并且为了维持连续的服务, 管理程序124 (0) 确定在计算上下文1700 (0) 不可用时, 处理上下文1710应该至少暂时迁移到不同的计算环境。

[0176] 在这样的情况下, 管理程序124 (0) 与在计算环境1700 (1) 中执行的对应的管理程序124 (1) 交互以配置PPU分区600 (1), 以提供与PPU分区600 (0) 相同或相似的资源。如图所示, PPU分区600 (1) 包括资源1720 (2) 和1720 (3), 但是为了模仿PPU分区600 (0) 提供的资源量, 使得1720 (3) 不可用。这样, 处理上下文1710可以从PPU分区600 (0) 内的SMC引擎700 (0) 迁移到PPU分区600 (1) 内的SMC引擎700 (1), 而服务质量没有明显变化。该方法通过提供对

一致的资源集合的访问,同时还允许在不同的硬件之间迁移处理上下文,来帮助维持任何给定的PPU分区600以与PPU 200类似的方式运行的外观。管理程序124也可以实施上述方法以在同一PPU 200内的分区600之间迁移SMC引擎700。在一个实施例中,管理程序124 (0) 和124 (1) 可以作为管理在不同计算环境1700中的多个PPU 200的操作的统一软件实体执行。

[0177] 总体上参考图11-图17,管理程序124以支持同时执行与多个处理上下文相关联的处理任务的方式来实现上述技术以分区PPU资源。下面结合图18更详细地描述这些技术。

[0178] 图18是根据各种实施例的用于在PPU内配置计算资源以同时支持与多个处理上下文相关联的操作的方法步骤的流程图。尽管结合图1-图17的系统描述了方法步骤,但是本领域技术人员将理解,配置成以任何顺序执行方法步骤的任何系统都落在本实施例的范围内。

[0179] 如图所示,方法1800开始于步骤1802,其中图1的管理程序124评估PPU 200以确定一组可用的硬件资源。某些硬件资源有时在给定的PPU 200的制造期间无法正确制造,并且可能是非功能性的。实际上,这些非功能性硬件资源已被熔断并且不使用。然而,给定的PPU 200内的其他硬件资源是功能性的并且因此整个PPU 200仍可以运行,因此尽管性能较低。以所描述的方式抢救部分功能性PPU和其他类型的单元在本领域中称为“地板清扫”。

[0180] 在步骤1804,管理程序124基于在步骤1802确定的可用硬件资源确定一组可用swizID。如以上结合图11所述,给定的swizID定义了一组硬件边界,其可以被启用和禁用以隔离PPU 200中不同的PPU切片610组以形成PPU分区600。在某些硬件资源不可用的情况下,管理程序124确定某些swizID对应于不可行的分区配置,因此应使其不可用。

[0181] 在步骤1806,管理程序124基于分区输入生成一组swizID。例如,管理程序124可以从管理员用户接收指示一组分区选项的输入,然后将那些分区选项映射到从在步骤1804确定的可用swizID组中派生的相应swizID组。可选地,管理程序124可以直接从管理员用户接收该swizID组,然后修改未包含在该可用swizID组中的任何swizID。

[0182] 在步骤1808中,管理程序124基于在步骤1806中生成的swizID组在硬件资源之间配置一组边界。在这种情况下,管理程序124跨该swizID组计算逻辑OR以生成配置swizID (或“本地”swizID),其指示应将哪些边界选项激活为边界,并禁用哪些边界选项。上面结合图12描述了示例性的分区选项集和相应的边界选项。

[0183] 在步骤1810,访客OS 916至少部分地基于一个或更多个swizID在分配给访客用户的PPU分区600中启动一组处理上下文。管理程序124将一组虚拟地址空间标识符1500分配给对应于图15的全局虚拟地址空间标识符1510的一部分的PPU分区600。管理程序124或该PPU分区600内的SMC引擎700可以将该组虚拟地址空间标识符1500细分为不同范围,这些范围又分配给不同的处理上下文。这种方法允许每个处理上下文在所有SMC引擎700上使用一组一致的虚拟地址空间进行操作,从而允许更容易地迁移处理上下文。

[0184] 在步骤1812,管理程序124或相应的访客OS 916响应于一个或更多个故障而重置在步骤1810启动的处理上下文的子集。这些故障可能发生在执行单元级别、SMC引擎级别或VM级别等。重要的是,在执行与一个处理上下文相关联的处理任务期间产生的故障通常不会影响与其他处理上下文相关联的处理任务的执行。处理上下文之间的这种故障隔离专门解决了依赖于处理子上下文的现有技术方法中发现的问题。可选地,在步骤1810和1812之间,可以调用调试器以控制遇到故障的SMC引擎700。

[0185] 在步骤1814,管理程序124基于与PPU 200相关联的可用硬件资源来配置迁移目标。迁移目标可以是另一个PPU 200,但是在某些情况下,迁移目标200是给定PPU分区600内的另一个SMC 700或PPU 200中的另一个PPU分区600。在配置迁移目标时,管理程序124可以执行本文中称为“软地板清扫”的技术,以使迁移目标提供与由一组处理上下文所使用的硬件资源相似的硬件资源。。

[0186] 在步骤1816,管理程序124将一组处理上下文迁移到迁移目标。与那些处理上下文相关联的处理任务可以在中断很少或没有中断的情况下继续进行,并且可以使用相似的可用硬件资源继续进行。因此,这些技术允许在处理上下文需要在不同的PPU分区600或不同的PPU 200之间移动的情况下,提供平衡的服务质量。

[0187] 总体上参考图11-图18,管理程序124、访客OS 916和/或主机OS 926执行所公开的技术,以将与PPU 200相关联的各种计算资源分为隔离的和独立的PPU分区600,在其中不同的处理上下文可以同时激活。因此,与在可能不完全利用PPU的时间支持一个处理上下文的传统方法相比,可以更有效地利用PPU 200的资源。也可以由多个不同的租户彼此独立地访问和配置不同的PPU分区600。因此,所公开的技术为多租户提供了可靠的支持,因此可以满足消费者对有效的基于云的并行处理平台的需求。

[0188] 分区存储器资源以支持同时的多个上下文

[0189] 除了对与PPU 200相关联的计算资源进行分区以同时支持多个处理上下文之外,管理程序124还对与PPU 200相关联的存储器资源进行分区以同时支持多个上下文,因此为多租户提供了可靠的支持。管理程序124在分区与PPU 200相关联的存储器资源时实现各种技术,下面结合图19-图24更详细地描述。

[0190] 图19示出了根据各种实施例的边界选项集,根据这些边界选项,图1的管理程序可以生成一个或更多个PPU存储器分区。如图所示,DRAM 272包括边界选项集1900,其可以在分区期间被激活以将L2高速缓存分为各个部分和分区。

[0191] 特别地,边界选项1900 (0)、1900 (1)、1900 (9) 和1900 (10) 将DRAM 272分为图8A的顶部部分810、可分区部分820和底部部分830。边界选项1900 (0) 形成底部部分830的下边界,并且边界选项1900 (1) 形成底部部分830的上边界。边界选项1900 (1) 还形成可分区部分820的下边界以及可分区部分820的左边界。边界选项1900 (9) 形成可分区部分820的右边界以及可分区部分820的上边界。边界选项1900 (9) 还形成了顶部部分810的下边界,并且边界选项1900 (10) 形成顶部部分810的上边界。边界选项1900 (1) 到1900 (8) 将可分区部分820进一步细分为各种存储器分区,下面将结合图20进行更详细地描述。

[0192] 同样如图所示,DRAM 272的总大小为M,顶部部分810的总大小为T,可分区部分820的总大小为P,并且底部部分830的总大小为B。进一步,与可分区部分820对应的给定缓存切片的部分由F给出,并且与底部部分对应的给定缓存切片的部分由W给出。F和W是可通过管理程序124设置的可配置参数,并且在一些实施例中,其可以相对于M完全约束T、P和B的值。

[0193] 在配置期间,管理程序124基于M、F和W将DRAM 272配置到顶部部分810、可分区部分820和底部部分830。这样,管理程序124基于M、F和W确定T、P和B的值。管理程序124还基于经由与管理用户交互而生成的配置swizID来激活特定边界选项1900,如以上结合图11-图12所述。下面结合图20描述边界选项1900的示例性激活。

[0194] 图20示出了根据各种实施例的图1的管理程序如何分区PPU存储器以生成一个或

更多个PPU存储器分区的示例。如图所示,边界选项1900(0)、1900(1)、1900(9)和1900(10)被激活,从而形成DRAM 272的顶部部分810、可分区部分820和底部部分830。边界选项1900(1)、1900(5)、1900(7)和1900(8)也被激活,从而形成分别对应于可分区部分820内的DRAM分区822(0)、822(4)、822(6)和822(7)的PPU存储分区710(0)、710(4)、710(6)和710(7)。边界选项1900(2)、1900(3)、1900(4)和1900(6)未激活,并且因此已被省略。管理程序124基于等于“11110100011”的配置swizID,以所示方式配置DRAM 272。

[0195] 与DRAM 272相关联的边界选项1900逻辑地对应于图11-图12中所示的边界选项1110。如以上结合图11-图12所讨论的,给定配置swizID的每个位指示是否应当激活或停用对应的边界选项1110以将PPU切片610分组在一起。以类似的方式,如图20所示,示例性配置swizID

[0196] “11110100011”中每个位指示与DRAM 272相关联的对应边界选项1900是否应当被激活或停用。

[0197] 默认情况下,示例性swizID的位0和10设置为1,以激活边界选项1900(0)和1900(10)。将示例性swizID的位1和9设置为1以激活边界选项1900(1)和1900(9),并建立可分区部分820。将示例性swizID的位5、7和8设置为1以激活边界选项1900(5)、1900(7)和1900(8),并将可分区部分820分为与PPU存储器分区710相关联的DRAM部分822。swizID的其他位设置为零以停用相应的边界选项。此处所示的DRAM 272的分区对应于图12中所示的PPU切片610的示例性配置。一旦通过管理程序124以这种方式进行分区,在PPU分区600内执行的SMC引擎700可以通过L2高速缓存切片800以下面结合图21-23所描述的方式执行存储器访问操作。

[0198] 图21示出了根据各种实施例的图16的存储器管理单元如何提供对不同的PPU存储器分区的访问。如图所示,图16的MMU 1600耦合在DRAM 272和1D SPA空间850之间。1D SPA空间850被分为与顶部部分810相对应的顶部地址852,与可分区部分820相对应的可分区地址854以及与底部部分830相对应的底部地址858,也如图8B所示。在分区期间,管理程序124基于DRAM 272的配置生成1D SPA空间850。

[0199] MMU 1600包括地址映射单元(AMAP)2110,其被配置为将顶部地址852、可分区地址854和底部地址858映射到分别与顶部部分810、可分区部分820和底部部分830相关联的原始地址。以这种方式,MMU 1600服务于从管理程序124接收的存储器访问请求,其以顶部部分810和/或底部830部分为目标,以及从SMC引擎700接收的存储器访问请求,其以可分区部分820为目标,如下面结合图22更详细地描述的。

[0200] 图22示出了根据各种实施例的图16的存储器管理单元如何执行各种地址转换。如图所示,可分区地址854包括地址区域856(0),其包括对应于PPU存储器分区710(0)的地址,如以上结合图8B所讨论的。MMU 1600经由AMAP 2110将包括在地址区域856(0)中的物理地址转换为与DRAM部分822(0)相关联的原始地址。AMAP2110被配置为在PPU存储器分区710(0)中包括的L2高速缓存切片800(0)上对来自地址区域856(0)的地址进行重排,以避免跨步导致相同的L2缓存切片800(0)被重复访问(也称为“露营”)的情况。

[0201] 在一个实施例中,AMAP 2110可以实现“存储器访问”swizID,其识别用于给定存储器区域的存储器交织因子。给定的存储器访问swizID确定了一组L2高速缓存切片,它们交叉交错以用于各种类型的存储器访问,包括视频存储器、系统存储器和对等存储器访问。不

同的PPU分区600通常在可分区部分829内实现不同且不重叠的存储器区域822,以最小化同时执行的作业之间的干扰。管理程序124可以使用零的存储器访问swizID,以平衡跨L2高速缓存切片的存储器访问操作,这通常将访问顶部部分810或底部部分830。

[0202] 给定的存储器访问swizID可以是“本地”swizID,其是基于系统物理地址计算的,并且用于在相关L2切片和DRAM的相应部分上交错或混淆存储器访问请求。与给定的PPU分区600相关联的给定的本地swizID可以对应于用于配置该PPU分区的swizID。通过这种方法,AMAP 2110可以根据用于激活那些边界的swizID在给定PPU存储器分区的边界内混淆地址。基于存储器访问swizID的混淆地址允许MMU 1600交错DRAM 272,以使每个PPU分区600在线性系统物理地址空间850中将其可分区部分820视为连续的。这种方法可以保持PPU分区600之间的隔离和与那些PPU分区600相关联的完整性。

[0203] 给定的存储器访问swizID可以替代地是由设备驱动器122或管理程序124提供的“远程”swizID,并用于在L2切片上交错存储器访问请求以进行系统存储器访问操作。对于在给定的PPU分区600内发生的处理操作,本地swizID和远程swizID可以相同。不同的PPU分区600通常具有不同的远程swizID,以允许系统存储器访问操作仅通过属于PPU分区600的L2切片800。

[0204] MMU 1600还提供用于将与虚拟地址空间标识符1500相关联的虚拟地址转换为1D系统物理地址空间850中的系统物理地址的支持。例如,假设图7的SMC引擎700 (0) 使用PPU存储器分区710 (0) 和对应的DRAM部分822 (0) 执行,并且这样做会引起存储器故障。MMU 1600会发出带有本地故障标识符1610的故障。MMU 1600又会将本地故障标识符1610转换为全局故障标识符1620。可以根据SR-IOV公共规范将故障和错误报告给虚拟功能。

[0205] MMU 1600还有助于细分地址区域856和PPU存储器分区710,以提供对在给定PPU分区600内执行的多个SMC引擎700、多个VM和/或多个处理上下文1400的支持,如下面结合图23更详细地描述的。

[0206] 图23示出了根据各种实施例的图16的存储器管理单元如何同时提供对与多个处理上下文相关联的操作的支持。如图所示,地址区域856 (0) 包含大小不同的多个虚拟存储器页面2310。对于SMC引擎700 (0),虚拟存储器空间标识符1500映射到全局虚拟存储器空间标识符1510,其为SMC引擎700 (0) 上的处理上下文使用的特定虚拟地址空间选择页表。由页表A指定的页面选择DRAM部分822 (0) 内的页面2310 (A)。同时,SMC引擎700 (2) 可以使用由页表B指定的页面,其也在DRAM部分822 (0) 内选择页面2310 (B)。通过基于页面的虚拟存储器管理方案,可以将DRAM部分822 (0) 中的页面分配给不同的子上下文或不同的处理上下文。注意,处理上下文可以使用多个虚拟地址空间标识符1500,因为它可以执行许多子上下文。

[0207] 以所示方式细分对应于PPU存储器分区710 (0) 的地址区域856 (0) 和DRAM部分822 (0) 为在相应PPU分区600内执行的不同SMC引擎700提供PPU存储器分区822 (0) 内的专用存储器资源。因此,不同PPU 600分区中的多个SMC引擎700可以同时在不同处理上下文内执行处理任务,而不会在带宽方面相互干扰。

[0208] 上述的基于页面的方法也可以应用于执行多个处理子上下文的单个SMC引擎700,其中每个处理子上下文都需要PPU存储器分区710 (0) 的专用部分。同样地,以上方法可以应用于在一个或更多个SMC引擎700上执行的不同VM并且需要PPU存储器分区710 (0) 的专用部分。

[0209] 总体上参考图19-图23,所公开的技术允许以以上结合图11-图12所描述的方式配置给定PPU分区600,以同时安全地启动多个处理上下文。特别地,如所描述的,对L2高速缓存的分区公平地将DRAM部分822分配给不同的PPU分区600。此外,经由MMU 1600和AMAP 2110实现的各种地址转换有效且公平地利用了存储器带宽,从而向PPU 200的所有租户提供一致的服务质量。下面还结合图24更详细地描述结合图19-图23描述的技术。

[0210] 图24是根据各种实施例的用于在PPU内配置存储器资源以同时支持与多个处理上下文相关联的操作的方法步骤的流程图。尽管结合图1-图23的系统描述了方法步骤,但是本领域技术人员将理解,被配置为以任何顺序执行方法步骤的任何系统都落在本实施例的范围内。

[0211] 如图所示,方法2400在步骤2402开始,其中图1的管理程序124确定用于对DRAM 272进行分区的一组存储器配置参数。该组存储器配置参数可以指示描述DRAM 272的任何属性的任何技术上可行的参数组,包括DRAM 272的总大小(M)、顶部部分810的大小T、可分区部分820的大小P、底部部分830的大小B、L2缓存切片800的数量(例如,96)、对应于可分区部分820的每个高速缓存切片部分的大小F和/或对应于底部部分830的每个高速缓存切片部分的大小W。在一个实施例中,一组配置参数仅需要包括参数M、F和W。

[0212] 在步骤2404,管理程序124基于在步骤2402中确定的一组存储器配置参数来激活第一组边界选项,以将DRAM 272分为多个部分。特别地,管理程序124激活图19所示的边界选项1900(0)、1900(1)、1900(9)和1900(10),以便将DRAM 272分为顶部部分810、可分区部分820和底部部分830。在一个实施例中,管理程序124可以修改给定的边界选项的位置来调整DRAM 272的相应部分的大小。

[0213] 在步骤2406,管理程序124基于分区输入来确定配置swizID。可以经由上文结合图10所描述的方法1000的步骤1002获得分区输入。分区输入指示PPU 200的一组目标PPU分区。管理程序124可以通过以上结合图11-图12描述的技术确定目标分区集的配置swizID。在一个实施例中,可以在步骤2404之前获得配置swizID,然后可以基于该配置swizID来激活第一组边界选项。

[0214] 在步骤2408中,管理程序基于配置swizID激活第二组边界选项,以在DRAM 272的可分区部分820内生成一个或多个PPU存储器分区710。第二组边界选项可以包括图19中示出的任何边界选项1900(2)至1900(8)。这些不同的边界选项可以将可分区部分820细分为与PPU存储器分区710相对应的多个DRAM部分822,其等于或小于PPU切片610的数目。在各种实施例中,方法2400的步骤2404和2408可以基于经由管理员用户输入获得或基于管理员用户输入生成的配置swizID而彼此结合地执行。

[0215] 在步骤2410,管理程序124基于存储器配置参数的集合和/或配置swizID来确定可分区地址的集合854。这样,管理程序124将ID SPA空间850分为顶部地址852、可分区地址854和底部地址858,如图21所示。顶部地址852可以被转换为与顶部部分810相关联的原始地址,可分区地址854可以被转换为与可分区部分820相关联的原始地址,并且底部地址858可以被转换为与底部部分830相关联的原始地址。

[0216] 在步骤2412中,MMU 1600通过在与DRAM部分822相对应的PPU存储器分区710内的L2高速缓存切片800上混淆可分区地址854,来服务于存储器访问请求。MMU 1600基于与PPU存储器分区710相关联的存储器访问swizID(或“远程”swizID)通过AMAP 2110混淆可分配

地址。在一个实施例中,从根据其配置了PPU存储器分区710的swizID导出存储器访问swizID。以这种方式混淆可分区的地址854可以减少对各个L2高速缓存切片800(也被称为“露营”)的重复访问。

[0217] 在步骤2414,MMU 1600接收与存储器故障相关联的本地故障标识符,并将本地故障标识符转换为全局故障标识符。这样,MMU 1600可以将与本地故障标识符相关联的虚拟地址转换成与全局故障标识符相关联的全局地址。例如,当给定的SMC引擎700在用PPU存储器分区710执行存储器读取操作或存储器写入操作期间遇到错误时,可能导致存储器故障。在本地虚拟地址空间中实现故障ID允许SMC引擎700以类似的地址空间来操作,从而允许在PPU分区600之间的处理上下文的更简单的迁移,如上面结合图17所描述的。将那些故障ID转换为全局故障标识符1620允许管理程序124解决与来自全局虚拟地址空间标识符1510透视图的故障ID相对应的故障。

[0218] 总体上参考图19-图24,所公开的用于分区PPU存储器资源的技术完成了以上结合图11-图18所描述的用于分区PPU计算资源的技术。通过这些技术,管理程序124可以配置和分区PPU 200以同时支持多个处理上下文。借助这一功能,PPU 200可以代表多个CPU进程安全地执行各种不同的处理任务,而又不让这些CPU进程相互干扰,从而可以更有效地利用PPU资源。另外,所公开的技术可以应用于为基于云的PPU部署中的多租户提供可靠的支持,从而满足历史上以前的方法尚未满足的产品需求。

[0219] 对多个VM和处理上下文进行时间切片

[0220] 如本文进一步所讨论的,图2的PPU 200支持两个级别的分区。在第一分区级别中,在本文中被称作“PPU分区”,将PPU 200的PPU资源500分为PPU分区600,在本文中也被称为“分区的PPU”。在一些实施例中,PP存储器270和DRAM 272中的一个或两个可以被分为SMC存储器分区710。通过这种分区级别,每个PPU分区600在任何给定时间执行一个VM。在第二分区级别中,在本文中被称作“SMC分区”,每个PPU分区600进一步被分为SMC引擎700。每个PPU分区600包括一个或多个SMC引擎700。在该分区级别中,每个SMC引擎700在任何给定时间为一个VM执行一个处理上下文。

[0221] 随着时间,SMC引擎700从执行针对VM的特定处理上下文切换为执行针对相同VM的不同处理上下文或执行针对不同VM的不同处理上下文。由于SMC引擎700的执行时间在与一个或多个VM相对应的多个处理上下文中被“切片”,因此该过程在本文中被称作“时间切片”。

[0222] 每个SMC引擎700在运行列表上列出的处理上下文之间时间切片,如由SMC引擎700的PBDMA 520和522所管理的。通常,当在VM之间切换时,所有受影响的SMC引擎700上的运行列表被替换,以便对不同组的处理上下文进行时间切片。如果多个SMC引擎700是活动的,则运行列表被同时替换。这种通过运行列表替换进行的调度类型在本文中被称作“软件调度”。相同VM内的处理上下文之间的切换类似地也可能涉及替换运行列表,并且与切换VM非常相似,不同之处在于,VM不会由于上下文切换而改变。结果,这种类型的上下文切换不需要额外的硬件支持。在一些实施例中,VM可以具有各种大小的许多不同的处理上下文。在这样的实施例中,软件调度可以考虑如何将它们打包到SMC引擎700中以进行正确和有效地执行。此外,软件调度可以重新配置PPU分区600的数量和每个PPU分区600内的SMC引擎的数量,以正确且有效地执行用于VM的处理上下文。

[0223] 为了使PPU资源500支持上述两个分区级别,PPU 200相应地支持两个时间切片级别。对应于PPU分区,PPU 200执行VM级时间切片,其中每个PPU分区600在多个虚拟机之间进行时间切片。对应于SMC分区,PPU 200执行SMC级时间切片,其中每个VM 600在多个处理上下文之间进行时间切片。在各种实施例中,两个时间分片级别随时间保持每个GPC 242中相同数量的TPC。在各种实施例中,时间切片可以涉及随时间改变一个或更多个GPC 242中的TPC的数量。现在描述时间切片的两个级别。

[0224] 图25是时间线集合2500,其示出了根据各种实施例的与图2的PPU 200的PPU分区600相关联的VM级时间切片。如图所示,时间线集合2500包括但不限于四个PPU分区时间线2502(0)、2502(4)、2502(6)和2502(7)。在一些实施例中,PPU分区时间线2502(0)、2502(4)、2502(6)和2502(7)可以分别对应于图6的PPU分区600(0)、600(4)、600(6)和600(7)。在这样的实施例中,PPU分区时间线2502(0)可以对应于PPU切片610(0)-610(3),并且PPU分区时间线2502(4)可以对应于PPU切片610(4)-610(5)。类似地,PPU分区时间线2502(6)可以对应于PPU切片610(6),并且PPU分区时间线2502(7)可以对应于PPU切片610(7)。结果,PPU分区600(0)可以同时执行多达四个处理上下文,PPU分区600(4)可以同时执行多达两个处理上下文,并且每个PPU分区600(6)和600(7)可以一次执行一个处理上下文。

[0225] 如PPU分区时间线2502(0)所示,PPU分区600(0)在两个VM之间时间切片,被称为VM A和VM B。时间线2502(0)示出了VM A与VM B之间的时间切片,其中与VM A相关联的处理上下文以上下文2510(Ax-y)的形式示出,与VM B相关联的处理上下文以上下文2510(Bx-y)的形式示出。从时间t0到时间t1,PPU分区600(0)执行VM A的处理上下文。PPU分区600(0)的第一SMC引擎700(0)依次执行处理上下文2510(A0-1)、处理上下文2510(A1-1)以及处理上下文2510(A2-1)。同时,PPU分区600(0)的第二SMC引擎700(2)依次执行处理上下文2510(A3-1)、处理上下文2510(A4-1)和处理上下文2510(A5-1)。在时间t1,PPU分区600(0)停止执行与VM A相关联的处理上下文,并且切换到与VM B相关联的处理上下文。处理上下文2510(A2-1)和处理上下文2510(A5-1)被上下文切换出相应的SMC引擎700(0)和700(2)。将PPU分区600(0)从具有两个SMC引擎700(0)(其包括两个GPC 230(0)和230(1))以及700(2)(其包括两个GPC 230(2)和230(3)),重新配置为具有一个SMC引擎700(0)(其包括四个GPC 230(0)、230(1)、230(2)和230(3))。一旦重新配置完成,PPU分区600(0)开始执行VM B的处理上下文2510(B0-1)。

[0226] 从时间t1到时间t4,PPU分区600(0)执行VM B的处理上下文。PPU分区600(0)的第一SMC引擎700(0)依次执行处理上下文2510(B0-1)、处理上下文2510(B1-1)和处理上下文2510(B2-1)。在时间t4,PPU分区600(0)停止执行与VM B相关联的处理上下文,并且切换到与VM A相关联的处理上下文。处理上下文2510(B2-1)被上下文切换出相应的SMC引擎700(0)。将PPU分区600(0)从具有一个SMC引擎700(0)(其包括四个GPC 230(0)、230(1)、230(2)和230(3)),重新配置为具有两个SMC引擎700(0)(其包括两个GPC 230(0)和230(1))以及700(2)(其包括两个GPC 230(2)和230(3))。一旦重新配置完成,PPU分区600(0)开始执行VM A的处理上下文2510(A2-2)和2510(A5-2)。从时间t4开始,PPU分区600(0)再次执行VM A的处理上下文。PPU分区600(0)的第一SMC引擎700(0)依次执行处理上下文2510(A2-2)和处理上下文2510(A0-2)。同时,PPU分区600(0)的第二SMC引擎700(2)依次执行处理上下文2510(A5-2)和处理上下文2510(A3-2)。

[0227] 如PPU分区时间线2502(4)所示,PPU分区600(4)在两个VM之间时间切片,被称为VM C和VM D。时间线2502(4)示出了VM C与VM D之间的时间切片,其中与VM C相关联的处理上下文以上下文2510(Cx-y)的形式示出,与VM D相关联的处理上下文以上下文2510(Dx-y)的形式示出。从时间t0到时间t2,PPU分区600(4)处于空闲状态,并且不执行任何处理上下文。从时间t2到时间t5,PPU分区600(4)执行VM C的处理上下文。PPU分区600(4)的第一SMC引擎700(4)依次执行处理上下文2510(C0-1)和处理上下文2510(C1-1)。在时间t5,PPU分区600(4)停止执行处理上下文2510(C1-1),并重新配置以开始执行VM D的处理上下文。从时间t5开始,PPU分区600(4)执行VM D的处理上下文。PPU分区600(4)的第一SMC引擎700(4)执行处理上下文2510(D1-1)。

[0228] 如PPU分区时间线2502(6)所示,PPU分区600(6)在两个VM之间时间切片,被称为VM E和VM F。时间线2502(6)示出了VM E与VM F之间的时间切片,其中与VM E相关联的处理上下文以上下文2510(Ex-y)的形式示出,与VM F相关联的处理上下文以上下文2510(Fx-y)的形式示出。从时间t0到时间t3,PPU分区600(6)处于空闲状态,并且不执行任何处理上下文。从时间t3到时间t6,PPU分区600(6)执行VM E的处理上下文。PPU分区600(6)的第一SMC引擎700(6)依次执行处理上下文2510(E0-1)和处理上下文2510(E1-1)。在时间t6,PPU分区600(6)停止执行处理上下文2510(E1-1),并重新配置以开始执行VM F的处理上下文。从时间t6开始,PPU分区600(4)执行VM F的处理上下文。PPU分区600(6)的第一SMC引擎700(6)执行处理上下文2510(F0-1)。

[0229] 如PPU分区时间线2502(7)所示,PPU分区600(7)在单个VM内进行时间切片,称为VM G。时间线2502(6)示出了VM G的时间切片,其中与VM G相关联的处理上下文以上下文2510(Gx-y)的形式示出。从时间t0到时间t5,PPU分区600(7)处于空闲状态,并且不执行任何处理上下文。从时间t5开始,PPU分区600(7)执行处理上下文G。PPU分区600(7)的第一SMC引擎700(7)执行处理上下文2510(G0-1)。

[0230] 以这种方式,PPU分区600(0)、600(4)、600(6)和600(7)中的每一个在与一个或多个VM相对应的处理上下文之间进行时间切片。PPU分区600(0)、600(4)、600(6)和600(7)中的每一个彼此独立地从一个处理上下文过渡到另一处理上下文。例如,PPU分区600(0)可以从执行针对特定VM的一个处理上下文切换到针对相同或不同VM的另一处理上下文,而不考虑PPU分区600(4)、600(6)和600(7)中的任何一个或多个是否切换处理上下文。在所示的时间段期间,PPU分区600(0)、600(4)、600(6)和600(7)中的每一个保持恒定数量的PPU切片610。在一些实施例中,每个PPU分区600的PPU切片610的数量可以改变,如现在所描述的。

[0231] 图26是根据各种其他实施例的另一时间线集合2600,其示出了与图2的PPU 200相关联的VM级时间切片。时间线集合2600中所示的处理上下文的功能与图25的时间线集合2500基本相同,除了下面进一步描述的。如图所示,时间线集合2600包括但不限于四个PPU分区时间线2602(0)、2602(4)、2602(6)和2602(7)。在一些实施例中,PPU分区时间线2602(0)、2602(4)、2602(6)和2602(7)可以分别对应于图6的PPU分区600(0)、600(4)、600(6)和600(7)。

[0232] 如PPU分区时间线2602(0)所示,PPU分区600(0)在两个VM之间时间切片,被称为VM A和VM B。时间线2602(0)示出了VM A与VM B之间的时间切片,其中与VM A相关联的处理上下文以上下文2610(Ax-y)的形式示出,与VM B相关联的处理上下文以上下文2610(Bx-y)的

形式示出。从时间t0到时间t1,PPU分区600(0)执行VM B的处理上下文。PPU分区600(0)的第一SMC引擎700(0)依次执行处理上下文2610(B1-1)和处理上下文2610(B2-1)。在时间t1,PPU分区600(0)停止执行与VM B相关联的处理上下文,并且切换到与VM A相关联的处理上下文。处理上下文2610(B2-1)被上下文切换出相应的SMC引擎700(0)。将PPU分区600(0)从具有一个SMC引擎700(0)(其包括四个GPC 230(0)、230(1)、230(2)和230(3)),重新配置为具有两个SMC引擎700(0)(其包括两个GPC 230(0)和230(1))以及700(2)(其包括两个GPC 230(2)和230(3))。一旦重新配置完成,PPU分区600(0)开始执行VM A的处理上下文。从时间t1到时间t3,PPU分区600(0)执行A的处理上下文。PPU分区600(0)的第一SMC引擎700(0)依次执行处理上下文2610(A2-1)、处理上下文2610(A0-1)和处理上下文2610(A1-1)。同时,PPU分区600(0)的第二SMC引擎700(2)依次执行处理上下文2610(A5-1)、处理上下文2610(A3-1)和处理上下文2610(A4-1)。在时间t3,PPU分区600(0)停止执行与VM A相关联的处理上下文,并且切换到与VM B相关联的处理上下文。处理上下文2610(A1-1)和处理上下文2610(A4-1)被上下文切换出相应的SMC引擎700(0)和700(2)。将PPU分区600(0)从具有两个SMC引擎700(0)(其包括两个GPC 230(0)和230(1))以及700(2)(其包括两个GPC 230(2)和230(3)),重新配置为具有一个SMC引擎700(0)(其包括四个GPC 230(0)、230(1)、230(2)和230(3))。一旦重新配置完成,PPU分区600(0)开始执行VM B的处理环境。从时间t3开始,PPU分区600(0)再次执行VM B的处理上下文。PPU分区600(0)的第一SMC引擎700(0)依次执行处理上下文2610(B2-2)和处理上下文2610(B0-1)。

[0233] 如PPU分区时间线2602(4)所示,从时间t0到时间t2,PPU分区600(4)的第一SMC引擎700(4)依次执行处理上下文2610(D0-1)和处理上下文2610(D1-1)。然后,PPU分区600(4)的第一SMC引擎700(4)空闲。如PPU分区时间线2602(6)所示,从时间t0到时间t2,PPU分区600(6)的第一SMC引擎700(6)依次执行处理上下文2610(F0-1)和处理上下文2610(F1-1)。然后,PPU分区600(6)的第一SMC引擎700(6)空闲。如PPU分区时间线2602(7)所示,PPU分区600(7)从时间t0到时间t2空闲。

[0234] 在时间t2,PPU分区600(4)、600(6)和600(7)合并以形成具有四个SMC引擎700(4)-700(7)的单个PPU分区600(4)。如PPU分区时间线2604(4)所示,合并的PPU分区600(4)执行VM H的处理上下文。从时间t2开始,PPU分区600(4)的第一SMC引擎700(4)依次执行处理上下文2610(H0-1)、处理上下文2610(H1-1)、处理上下文2610(H2-1)和处理上下文2610(H0-2)。

[0235] 以这种方式,可以在时间切片期间将PPU分区600合并和/或切分为不同大小的分区。PPU分区600可以彼此独立地合并和/或切分。在图25和26中,每个VM在固定数量的SMC引擎700上执行,从而导致给定VM同时执行恒定数量的处理上下文。如图所示,VM A在两个SMC引擎700上同时执行,而其余VM一次在一个SMC引擎700上执行。在一些实施例中,特定VM可以改变在其上执行VM的SMC引擎700的数量,如现在所描述的。

[0236] 图27是根据各种实施例的时间线2700,其示出了与图2的PPU 200相关联的SMC级时间切片。时间线2700所示的处理上下文分别与图25和26的时间线集合2500和2600基本相同,除了下面进一步描述的。如图所示,时间线2700代表单个PPU分区时间线。在一些实施例中,由时间线2700表示的PPU分区时间线可以对应于图6的任何PPU分区600,其包括至少两个SMC引擎700。

[0237] 如时间线2700所示,单个VM内的PPU分区600的时间切片,被称为VM A。时间线2700图示了VM A的时间切片,其中与VM A相关联的处理上下文以上下文2710 (Ax-y) 的形式示出。从时间t0开始,VM A在两个SMC引擎700上执行。PPU分区600中包含的第一SMC引擎700执行处理上下文2710 (A0-1),然后空闲直到时间t1。同时,包括在PPU分区600中的第二SMC引擎700执行处理上下文2710 (A1-1),然后空闲直到时间t1。时间t0和时间t1之间的持续时间足够长,以确保有足够的时间来处理上下文2710 (A0-1) 和2710 (A1-1) 以完成执行,并使SMC引擎进入空闲状态。在一些实施例中,处理上下文2710 (A0-1) 和处理上下文2710 (A1-1) 可以在两个单独的SMC引擎700上同时执行相同的任务,从而提供空间冗余。在这样的实施例中,处理上下文2710 (A0-1) 和处理上下文2710 (A1-1) 可以在彼此具有相同配置的冗余SMC引擎700上执行任务,然后比较结果的准确性和确定性。

[0238] 在时间t1和时间t2之间,从PPU分区600中移除了用于处理上下文2710 (A0-1) 和2710 (A1-1) 的运行列表。将PPU分区600从两个SMC引擎700重新配置为一个SMC引擎700,其包括两个SMC引擎700的所有资源。然后,PPU分区600使用新的运行列表来执行处理上下文2710 (B2-1)、2710 (B3-1) 和2710 (B4-1)。

[0239] 在时间t2和时间t3之间,VM B在一个SMC引擎700上执行。PPU分区600中包含的SMC引擎700依次执行处理上下文2710 (B2-1)、处理上下文2710 (B3-1) 和处理上下文2710 (B4-1)。在一些实施例中,处理上下文2710 (B2-1)、2710 (B3-1) 和2710 (B4-1) 可以执行性能密集型任务,这可以受益于在单个SMC引擎700上执行,该SMC引擎比执行处理上下文2710 (A0-1) 和2710 (A1-1) 的SMC引擎具有更多的计算资源。SMC引擎700然后空闲直到时间t3。在一些实施例中,SMC引擎700在该空闲时段期间执行离线调度任务。

[0240] 在时间t3和时间t4之间,从PPU分区600中移除了用于处理上下文2710 (B2-1)、2710 (B3-1) 和2710 (B4-1) 的运行列表。将PPU分区600从一个SMC引擎700重新配置为两个SMC引擎700。两个SMC引擎700每个都包含一个SMC引擎700中包含的部分资源。PPU分区600然后使用新的运行列表来执行处理上下文2710 (A0-2) 和2710 (A1-2)。

[0241] 从时间t4开始,VM A再次在两个SMC引擎700上执行。第一SMC引擎700执行处理上下文2710 (A0-2),然后空闲直到时间t5。同时,第二SMC引擎700执行处理上下文2710 (A1-2),然后空闲直到时间t5。时间t4和时间t5之间的持续时间足够长,以确保有足够的时间来处理上下文2710 (A0-2) 和2710 (A1-2) 以完成执行,并使SMC引擎进入空闲状态。在一些实施例中,处理上下文2710 (A0-2) 和处理上下文2710 (A1-2) 可以在两个单独的SMC引擎700上同时执行相同的任务,从而提供空间冗余。在这样的实施例中,处理上下文2710 (A0-2) 和处理上下文2710 (A1-2) 可以在彼此具有相同的配置的冗余SMC引擎700上执行任务,然后比较结果的准确性和确定性。

[0242] 在时间t5和时间t6之间,从PPU分区600中移除了用于处理上下文2710 (A0-2) 和2710 (A1-2) 的运行列表。将PPU分区600从两个SMC引擎700重新配置为一个SMC引擎700,其包括两个SMC引擎700的全部资源。然后PPU分区600使用新的运行列表来执行处理上下文2710 (B3-2)。在时刻t6开始,VM B再次在一个SMC引擎700上执行。SMC引擎700执行处理上下文2710 (B3-2)。

[0243] 在一些实施例中,PPU分区600可以在在一个SMC引擎上执行与在两个SMC引擎上执行之间快速重新配置,该过程在本文中被称为“快速重新配置”。快速重新配置增加了PPU

200资源的利用率,同时提供了用于多个处理上下文以不同模式在单个PPU分区600上执行的机制。PPU 200内的核心驱动器914和硬件微代码之一或二者包括实现快速重新配置的各种优化。现在描述这些优化。

[0244] 在重新配置期间,除非资源产生错误,否则不重置PPU 200中的某些资源,例如FECS 530和GPC 242环境上下文。结果,在重新配置期间将微代码加载到这些资源中可以被分为多个阶段。特别地,FECS 530和GPC 242的微代码加载序列可以分为LOAD阶段和INIT阶段。对于PPU分区600内的所有可用FECS 530和GPC 242环境上下文,并行执行LOAD阶段,从而减少了将微代码加载到这些资源中所需的时间。INIT阶段在重新配置期间执行,从而与重新配置PPU分区600并行执行FECS 530和GPC 242上下文切换的初始化。在初始化阶段期间,PPU 200确保所有FECS 530和GPC 242环境上下文的LOAD阶段已完成。结果,减少了加载和初始化PPU分区600的资源的时间。另外,PPU 200为PPU分区600的每个可能的配置存储块标准化处理上下文图像的高速缓存,其在本文被称为“黄金处理上下文图像”。在LOAD和INIT阶段期间检索并加载适当的黄金处理上下文图像,从而进一步减少了加载和初始化PPU分区600的资源的时间。

[0245] 如本文所述,特定VM可以随着时间改变VM在其上执行的SMC引擎700的数量。在一个特定示例中,VM A包括与自动驾驶车辆相关联的各种任务。自动驾驶车辆的某些任务比其他任务更为关键。例如,与自动驾驶相关联的任务(例如检测交通信号灯和避免撞车)将被认为比与车辆的娱乐系统相关联的任务更为关键。这些更关键的任务可能需要遵守某些法规或行业标准。一种这样的标准分配了被称为汽车安全完整性级别(ASIL)的分类级别。为了提高完整性级别,ASIL包括四个级别,分别称为ASIL-A、ASIL-B、ASIL-C和ASIL-D。检测交通信号灯和避免撞车等任务将归为ASIL-D。不太重要的任务可以归为较低的ASIL级别。与安全性无关的任务(例如与车辆的娱乐系统相关联的任务)可以归类为QM,这表明仅适用标准质量管理规范。

[0246] 在这方面,处理上下文2710(A0-1)和2710(A1-1)可以包括在PPU分区600的两个不同SMC引擎700上同时执行的相同ASIL-D级任务的两个实例。在处理上下文2710(A0-1)和2710(A1-1)完成执行后,将处理上下文2710(A0-1)和2710(A1-1)的结果相互比较。如果处理上下文2710(A0-1)和处理上下文2710(A1-1)生成相同的结果,则已验证结果,车辆将根据结果继续行驶。另一方面,与处理上下文2710(A0-1)或处理上下文2710(A1-1)相关联的一个或更多个组件中的故障可能导致受影响的处理上下文生成不正确的结果。因此,如果处理上下文2710(A0-1)和处理上下文2710(A1-1)生成不同的结果,则结果无效,并且车辆将执行适当的躲避动作,例如从车流中缓慢移向最靠近位置。

[0247] 在完成处理上下文2710(A0-1)和2710(A1-1)的执行之后,PPU分区600重新配置为仅包括一个SMC引擎700。SMC引擎700依次执行QM级处理上下文2710(B2-1)、2710(B3-1)和2710(B4-1)。这些处理上下文包括不太关键的任务,例如与车辆的娱乐系统相关联的任务。在完成处理上下文2710(B2-1)、2710(B3-1)和2710(B4-1)之后,PPU分区600重新配置为包括两个SMC引擎700。SMC引擎700同时执行处理上下文2710(A0-2)和2710(A1-2),它们是相同ASIL-D级任务的两个实例。在处理上下文2710(A0-2)和2710(A1-2)完成执行之后,PPU分区600再次重新配置为仅包括一个SMC引擎700,并执行QM级别处理上下文2710(B3-2)。

[0248] 以这种方式,PPU分区600在执行ASIL-D级任务的多个SMC引擎700和执行QM级任务

的单个SMC引擎700之间动态地重新配置。连续的ASIL-D处理上下文之间的持续时间(例如时间t0与时间t4之间的持续时间)被称为“帧”,其中时间t0与时间t1之间的帧的部分被分配用于执行ASIL-D任务。

[0249] 图28示出了根据各种实施例的VM如何从一个PPU 200(1)迁移到另一PPU 200(2)。如图2800所示,PPU 200(1)执行四个VM 2810A、2810B、2810C和2810D。这些VM 2810A、2810B、2810C和2810D中的每一个在PPU 200(1)中包括的不同的SMC引擎700上执行。类似地,PPU 200(2)执行四个VM 2810E、2810F、2810G和2810H。这些VM 2810E、2810F、2810G和2810H中的每一个在PPU 200(1)中包括的不同SMC引擎700上执行。

[0250] 随着时间,由于各种原因,VM可能从一个PPU 200迁移到另一PPU 200,包括但不限于准备进行系统维护、将VM整合到更少的PPU 200上以提高利用率或节省功率以及通过迁移到不同的数据中心获得效率。在第一示例中,当VM当前正在其上执行的系统将被关闭以用于系统维护时,可以迫使VM迁移到不同的PPU 200。在第二示例中,一个或更多个VM的处理上下文可以在不确定的时间内空闲。如果一个或更多个VM中的所有处理上下文都空闲,则VM可以从一个PPU 200迁移到另一PPU 200,以提高PPU 200的利用率或降低能耗。在第三示例中,可以将与特定用户或用户组相关联的VM从地理上遥远的数据中心迁移到较近的数据中心,以改善通信延迟。更一般地,VM可以被迁移到不同的PPU 200。

[0251] 更一般地,当上下文已被通过上下文保存从硬件上执行移除时,VM可在任何时候从一个PPU 200迁移到另一PPU 200。相关的操作系统,例如图9的访客操作系统916,可以在任何时候抢占上下文并强制保存上下文,而不仅仅是在相应的VM处于空闲状态时。当相应VM的所有工作均已完成时,可以抢占上下文。另外,可以通过强制上下文停止提交进一步的工作并耗尽当前正在进行的工作来抢占上下文,即使VM还有其他工作要执行。在任一种情况下,一旦已经耗尽上下文的进行中的工作并且已经保存上下文,就可以将VM从一个PPU 200迁移到另一PPU 200。在VM迁移期间,VM可能会暂停执行,时间约为几毫秒。

[0252] 在一些实施例中,VM可以仅迁移到另一个PPU 200中的PPU分区600,其具有与当前正在执行VM的PPU分区600相同的配置。例如,VM可以被限制为仅迁移到另一个PPU 200中的PPU分区600,其具有与当前正在执行VM的PPU分区600相同数量的GPC 242。如图2802所示,四个VM处于空闲状态。PPU 200(1)执行两个VM 2810A和2810C。先前在PPU 200(1)上执行的其他两个VM 2810B和2810D处于空闲状态。类似地,PPU 200(2)执行两个VM 2810E和2810H。先前在PPU 200(1)上执行的其他两个VM 2810F和2810G处于空闲状态。结果,PPU 200(1)和200(2)中的每一个未被充分利用。在这种情况下,当前正在执行的VM可能会迁移以更好地利用可用的PPU资源。在一个示例中,在PPU 200(1)上执行的VM可以消耗PPU 200(1)上可用的硬件资源的一半。同样,在PPU 200(2)上执行的VM可能会消耗PPU 200(2)上可用的硬件资源的一半。结果,PPU 200(1)和PPU 200(2)中的每一个将以大约50%的容量运行。如果将在PPU 200(2)上执行的所有VM都迁移到PPU 200(1),则PPU 200(1)将以大约100%的容量运行。PPU 200(2)将以容量的0%运行。结果,为了降低功耗,可以减小到PPU 200(2)的电源电压。如图2804所示,VM 2810E和2810H已经从PPU 200(2)迁移到PPU 200(1)。结果,PPU 200(1)执行四个VM 2810A、2810E、2810C和2810H。因此,PPU 200(1)被更充分地利用。VM迁移后,PPU 200(2)不再执行任何VM。结果,PPU 200(2)可以被断电以减少功耗。如果另外的VM随后开始执行,则PPU 200(2)可以被通电以执行另外的VM。

[0253] 图29是根据各种实施例的时间线集合2900,其示出了与图2的PPU 200相关联的精细VM迁移。时间线集合2900的功能分别与图25和26的时间线2500和2600以及图27的时间线2700基本上相同,除了以下进一步描述的。如图所示,时间线集合2900包括但不限于四个PPU分区时间线2902(0)、2902(1)、2902(2)和2902(3)。在一些实施例中,PPU分区时间线2902(0)、2902(1)、2902(2)和2902(3)可以对应于图6的四个PPU分区600。每个PPU分区600包括一个SMC引擎700。因此,每个PPU分区600可以一次执行一个VM。如PPU分区时间线2902(0)、2902(1)、2902(2)和2902(3)所示,每个PPU分区600在五个VM之间进行时间切片,其被称为VM A到VM F。作为结果,五个VM中的每一个都在四个PPU分区600之间迁移。

[0254] 在图29所示的时间段期间,VM A在第一PPU分区上执行处理上下文2910(A0),如PPU分区时间线2902(3)所示。然后,VM A迁移到第二PPU分区,并在PPU分区时间线2902(2)上执行处理上下文2910(A1)。随后,VM A依次迁移到第三PPU分区和第四PPU分区,并分别在PPU分区时间线2902(1)和2902(0)上执行处理上下文2910(A2)和2910(A3)。然后,VM A迁移回第一PPU分区,并在PPU分区时间线2902(3)上执行处理上下文2910(A4)。最后,VM A再次迁移到第二PPU分区,并在PPU分区时间线2902(2)上执行处理上下文2910(A5)。

[0255] 以类似的方式,执行处理上下文2910(B0)至2910(B4)的VM B在第一PPU分区600上执行,然后在其他三个PPU分区之间迁移,如PPU分区时间线2902(0)、2902(1)、2902(2)和2902(3)所示。其余三个VM同样在四个PPU分区之间迁移,其中VM C执行处理上下文2910(C0)至2910(C5),VM D执行处理上下文2910(D0)至2910(D5),VM E执行处理上下文2910(E0)到2910(E5)。

[0256] 以这种方式,五个VM在四个PPU分区600之间迁移,其中每个VM访问基本上相同数量的PPU资源。总体而言,五个VM各自能够执行大约80%的时间,其中四个PPU分区600除以五个VM等于4/5或80%。结果,精细VM迁移在一组VM之间执行负载平衡,而与VM的数量相对于PPU分区600的数量无关。

[0257] 图30A-图30B示出了根据各种实施例的用于对图2的PPU 200中的VM进行时间切片的方法步骤的流程图。尽管结合图1-图15的系统描述了方法步骤,但是本领域普通技术人员将理解,配置成以任何顺序执行方法步骤的任何系统都在本公开的范围之内。

[0258] 如图所示,方法3000在步骤3002开始,其中PPU 200确定至少一个VM将要从一个或更多个处理上下文的第二集合切换到一个或更多个处理上下文的第二集合。VM可能会出于任何技术上可行的原因而切换一个或更多个处理上下文,包括但不限于VM完成所有任务的执行,VM进入空闲状态,VM已执行了最大分配时间,或VM已经产生错误。

[0259] 在步骤3004,PPU 200确定PPU 200是否需要执行PPU内部分区改变以适应新的一个或更多个处理上下文。当PPU分区600在上下文切换期间维持相同数目的PPU切片610但是改变PPU分区600内的活动SMC引擎700的数目时,发生PPU内部分区改变。如果PPU 200不需要执行PPU内部分区改变以适应新的一个或更多个处理上下文,则方法3000进行到步骤3008。然而,如果PPU 200需要执行PPU内部分区改变以适应新的一个或更多个处理上下文,则方法3000进行到步骤3006,其中PPU 200重新配置PPU分区600以在改变活动SMC引擎700的数量的同时保持相同数量的PPU切片610。

[0260] 在步骤3008,PPU 200确定PPU 200是否需要执行PPU间分区改变以适应新的一个或更多个处理上下文。当在上下文切换期间PPU分区600通过合并或切分一个或更多个PPU

分区600来改变PPU切片610的数量时,发生PPU间分区改变。作为结果,PPU 200还改变700内PPU分区600的数量。取决于新的处理上下文,PPU 200可以更改或可以不更改每个PPU分区600中活动的SMC引擎700的数量。如果PPU 200不需要执行PPU间分区改变以适应新的一个或更多个处理上下文,则方法3000进行到步骤3012。但是,如果PPU 200需要执行PPU间分区改变以适应新的一个或更多个处理上下文,则方法3000进行到步骤3010,其中PPU 200重新配置PPU分区600以改变包括在PPU分区600中的PPU切片610的数量。为了改变PPU分区600中的PPU切片610的数量,PPU 200将两个或更多个PPU分区600合并为单个PPU分区600。附加地或可选地,PPU 200可以将PPU分区600分成两个或多个PPU分区600。

[0261] 在步骤3012,PPU 200确定在给定PPU 200上执行的所有VM是否空闲。如果给定PPU 200上的一个或更多个VM不是空闲的(活动的),则该方法进行到步骤3018。另一方面,如果给定PPU 200上执行的所有VM的空闲,则方法3000进行到步骤3014,其中PPU 200确定一个或更多个其他PPU 200是否具有可用于执行空闲VM的资源。如果资源在一个或更多个其他PPU 200上不可用,则该方法进行到步骤3018。另一方面,如果资源在一个或更多个其他PPU 200上不可用,则该方法进行到步骤3016,其中PPU 200将空闲VM迁移到一个或更多个其他PPU 200。

[0262] 在步骤3018,在执行PPU内部分区改变、PPU间分区改变和/或VM迁移之后,PPU 200开始执行新的处理上下文。然后,方法3000终止。在各种实施例中,PPU 200独立于确定执行PPU间改变的需要来确定执行PPU内部改变的需要。类似地,在各种实施例中,PPU 200独立于确定执行PPU内部改变的需要来确定执行PPU间改变的需要。

[0263] 特权寄存器地址映射

[0264] 如本文进一步所述,图5的PRI集线器512和内部PRI总线(未示出)使CPU 110和/或PPU 200中的任何单元能够读取和写入特权寄存器,也被称为“PRI寄存器”:其分布在整个PPU 200中。这样,PRI集线器212被配置为在覆盖所有PRI总线寄存器的通用地址空间与为每个系统管道230分别定义的地址空间之间映射PRI总线地址。当通过PCIe链接进行通信(通常是从CPU)时,通过PCIe地址空间访问PRI寄存器,在本文中被称作“基地址寄存器0”空间,或更简单地说,是“BAR0”地址空间,通常用于连接到PCIe总线的设备。通常,由于大量设备必须全部容纳在BAR0地址空间中,因此PPU 200的BAR0地址空间的寻址存储器范围被限制为16兆字节(MB)。16MB的地址范围足以访问单个SMC引擎700的特权寄存器。但是,为了支持多个SMC引擎700,地址范围可能会超过16MB。因此,PRI集线器512提供两种寻址模式以便支持用多个SMC引擎300执行。第一寻址模式,在本文中被称作“传统模式”,适用于涉及单个SMC引擎700的操作。第二寻址模式,在本文中被称作“SMC引擎寻址模式”,适用于涉及多个SMC引擎700的操作。现在描述寻址模式。

[0265] 图31是根据各种实施例的存储器映射,其示出了BAR0地址空间3110如何映射到图2的PPU 200内的特权寄存器地址空间3120。BAR0地址空间3110包括但不限于第一地址空间3112、图形寄存器(GFX REG)地址空间3114和第二地址空间3116。特权寄存器地址空间3120包括但不限于第一地址空间3122、传统图形寄存器地址空间3124、第二地址空间3126和SMC图形寄存器地址空间3128(0)-3128(7)。BAR0地址空间3110支持两种寻址模式,传统寻址模式和SMC寻址模式。通常,两种模式的目标是:(1)传统模式,其中整个PPU 200被视为具有一组PRI寄存器的一个引擎;(2)SMC模式,其中每个PPU分区600被寻址为好像每个PPU分区600

是单独的引擎,并且好像每个PPU分区600本身就是整个PPU 200。当在传统模式下处理整个PPU以及当仅在一个PPU分区600上进行处理时,SMC模式允许驱动器软件122相同。也就是说,驱动器可以编写一次,并在传统模式和SMC模式两种情景下使用。。

[0266] 在传统寻址模式中,PPU 200执行任务作为硬件资源的单个集群,而不是作为具有单独的SMC引擎700的单独的PPU分区600。在传统模式下,将对存储器地址的存储器读取或写入定向到BAR0地址空间3110的第一地址空间3112或第二地址空间3116,分别访问特权寄存器地址空间3120的第一地址空间3122或第二地址空间3126中的相应的存储器地址。类似地,将对存储器地址的存储器读取或写入定向到BAR0地址空间3110的图形寄存器地址空间3114,访问特权寄存器地址空间3120的传统图形寄存器地址空间3124内相应的存储器地址。传统图形寄存器地址空间3124包括PPU 200内各种组件的地址范围,包括但不限于,计算FE 540、图形FE 542、SKED 550、CWD 560和PDA/PDB 562。此外,传统图形寄存器地址空间3124包括用于每个的GPC 242的地址范围。GPC 242可以经由传统图形寄存器地址空间3124内的专用地址范围单独寻址的,减去由于地板清扫而被移除的任何的GPC 242。附加地或可选地,传统图形寄存器地址空间3124包括用于同时向所有GPC 242广播数据的地址范围。当相同地配置所有GPC 242时,这些GPC广播地址空间可能是有用的。

[0267] 在SMC寻址模式中,PPU 200执行任务作为具有单独的SMC引擎700的单独的PPU分区600。如在传统模式中一样,将对存储器地址的存储器读取或写入定向到BAR0地址空间3110的第一地址空间3112或第二地址空间3116,分别访问特权寄存器地址空间3120的第一地址空间3122或第二地址空间3126中的相应的存储器地址。在SMC模式下,提供SMC图形寄存器地址空间3128 (0) - 3128 (7) 以分别访问SMC引擎700 (0) - 700 (7) 的每一个中的各个组件。这些组件包括但不限于,计算FE 540 (0) - 540 (7)、图形FE 542 (0) - 542 (7)、SKED 550 (0) - 550 (7)、CWD 560 (0) - 560 (7) 以及PDA/PDB 562 (0) - 562 (7)。特定的相应的SMC引擎700的GPC 242是可以通过传统图形寄存器地址空间3124内专用地址范围单独寻址的,减去由于地板清扫而被移除的任何的GPC 242。附加地或可选地,传统图形寄存器地址空间3124包括用于同时将数据广播到特定的相应的SMC引擎700的所有GPC 242的地址范围。BAR0地址空间3110提供两种机制用于访问SMC图形寄存器地址空间3128 (0) - 3128 (7)。

[0268] 在第一种机制中,BAR0地址空间3110的图形寄存器地址空间3114映射到特权寄存器地址空间3120中的SMC图形寄存器地址空间3128 (0) - 3128 (7) 中的一个。BAR0地址空间3110内的特定地址访问SMC窗口寄存器。SMC窗口寄存器包括两个字段。这两个字段包括SMC启用字段和SMC索引字段。SMC启用字段包含一个二进制逻辑值,该值为FALSE或TRUE。如果SMC启用字段为FALSE,则BAR0地址空间3110以传统寻址模式访问特权寄存器地址空间3120,如本文所述。如果SMC启用字段为TRUE,则BAR0地址空间3110基于SMC索引字段的值以SMC寻址模式访问特权寄存器地址空间3120。SMC索引字段的值指定当前将哪个SMC引擎700映射到BAR0地址空间3110。例如,如果SMC索引字段的值为0,则特权寄存器地址空间3120的SMC图形寄存器地址空间3128 (0) 将被映射到BAR0地址空间3110的图形寄存器地址空间3114。同样,如果SMC索引字段的值为1,则特权寄存器地址空间3120的SMC图形寄存器地址空间3128 (1) 将被映射到图形BAR0地址空间3110的寄存器地址空间3114,等等。将对存储器地址的存储器读取或写入定向到BAR0地址空间3110的图形寄存器地址空间3114,访问由SMC索引字段指定的SMC图形寄存器地址空间3128内对应的存储器地址。通过该第一机制,

可以访问由SMC索引字段指定的SMC引擎700的SMC图形寄存器地址空间3128,而禁止访问其余SMC图形寄存器地址空间3128。

[0269] 在第二种机制中,某些特权组件(例如管理程序124)可以访问SMC图形寄存器地址空间3128(0)-3128(7)中的任何一个的单独地址。该第二种机制通过BAR0地址空间3110中的两个特定地址访问SMC图形寄存器地址空间3128(0)-3128(7)。这两个地址之一访问SMC地址寄存器。这两个地址中的另一个访问SMC数据寄存器。可以通过两个步骤访问SMC图形寄存器地址空间3128(0)-3128(7)中任何位置的特定存储器地址。第一步,向SMC地址寄存器写入一个地址,该地址与SMC图形寄存器地址空间3128(0)-3128(7)中的地址相对应。第二步,用数据值读取或写入SMC数据寄存器。在BAR0地址空间3110中读取或写入SMC数据寄存器,导致在SMC地址寄存器中指定的存储器地址处对特权寄存器地址空间3120进行相应的读取或写入。然后取消引用SMC地址寄存器,从而启用SMC地址寄存器和SMC数据寄存器以进行后续事务。读取或写入SMC地址寄存器不会导致对特权寄存器地址空间3120的读取或写入。

[0270] 图32是根据各种实施例的用于在图2的PPU 200中寻址特权寄存器地址空间的方法步骤的流程图。尽管结合图1至图17的系统描述了方法步骤,但是本领域普通技术人员将理解,配置成以任何顺序执行方法步骤的任何系统都在本公开的范围内。

[0271] 如图所示,方法3200在步骤3202开始,其中PPU 200检测针对特权寄存器地址空间3120的存储器访问。更具体地说,PPU 200检测针对BAR0地址空间3110的存储器访问。在步骤3204,PPU 200确定存储器访问是否针对图形寄存器地址空间3114。如果存储器访问未针对图形寄存器地址空间3114,则方法3200前进至步骤3214,其中PPU 200生成到存储器访问所指定的地址的存储器事务。然后,方法3200终止。

[0272] 返回到步骤3204,如果存储器访问针对图形寄存器地址空间3114,则方法3200进行到步骤3206,其中PPU 200确定存储器访问是否处于传统模式。如果存储器访问处于传统模式,则方法3200前进至步骤3214,其中PPU 200生成到存储器访问所指定的地址的存储器事务。然后,方法3200终止。另一方面,如果存储器访问不处于传统模式,则方法3200进行到步骤3208,在此PPU 200确定存储器访问是否处于窗口模式。

[0273] 如果存储器访问处于窗口模式,则该方法进行到步骤3212,其中PPU 200基于SMC窗口寄存器中指定的SMC索引字段的值来生成存储器事务。SMC索引字段的值指定当前哪个SMC引擎700映射到BAR0地址空间3110。例如,如果SMC索引字段的值为0,则特权寄存器地址空间3120的SMC图形寄存器地址空间3128(0)将被映射到BAR0地址空间3110的图形寄存器地址空间3114。同样,如果SMC索引字段的值为1,则特权寄存器地址空间3120的SMC图形寄存器地址空间3128(1)将被映射到BAR0地址空间3110的图形寄存器地址空间3114,等等。将对存储器地址的存储器读取或写入定向到BAR0地址空间3110的图形寄存器地址空间3114,访问由SMC索引字段指定的SMC图形寄存器地址空间3128内的相应的存储器地址。通过该第一机制,可以访问由SMC索引字段指定的SMC引擎700的SMC图形寄存器地址空间3128,而禁止访问其余SMC图形寄存器地址空间3128。然后,方法3200终止。

[0274] 返回到步骤3208,如果存储器访问不处于窗口模式,则该方法进行到步骤3210,其中PPU 200基于SMC地址寄存器和SMC数据寄存器的值生成存储器事务。更具体地说,PPU 200通过BAR0地址空间3110内的两个特定地址访问SMC图形寄存器地址空间3128(0)-3128

(7)。这两个地址之一访问SMC地址寄存器。这两个地址中的另一个访问SMC数据寄存器。可以通过两个步骤访问SMC图形寄存器地址空间3128(0) - 3128(7)中任何位置的特定存储器地址。第一步,向SMC地址寄存器写入一个地址,该地址与SMC图形寄存器地址空间3128(0) - 3128(7)中的地址相对应。第二步,用数据值读取或写入SMC数据寄存器。在BAR0地址空间3110中读取或写入SMC数据寄存器,导致在SMC地址寄存器中指定的存储器地址处对特权寄存器地址空间3120进行相应的读取或写入。然后取消引用SMC地址寄存器,从而启用SMC地址寄存器和SMC数据寄存器以进行后续事务。读取或写入SMC地址寄存器不会引起对特权寄存器地址空间3120的读取或写入。然后,方法3200终止。

[0275] 使用多个SMC引擎进行性能监视

[0276] 如本文中进一步讨论的,性能监视器(PM),例如图2的PM 236、图3的PM 360和图4的PM 430,监视包括在PPU200中的相应的组件的整体性能和/或资源消耗。性能监视器(PM)包括在性能监视系统中,该系统提供跨多个SMC引擎700的性能监视和性能分析。性能监视系统同时或基本上同时配置文件多个VM以及在VM中执行的处理上下文。性能监视系统关于如何生成和捕获性能数据将多个虚拟机和VM中执行的多个处理上下文彼此隔离,以防止VM之间的性能数据的泄漏。性能数据监视系统中的PM 232和相关联的计数器跟踪性能数据对特定SMC引擎700的属性。在共享资源和单元的情况下,其中属性不可追溯到特定SMC引擎700,具有较高特权实体的设备(诸如管理程序124)收集共享资源和单元的性能数据。当VM迁移到其他分区PPU 600和/或其他PPU 200时,性能监视系统同时配置文件计算引擎和图形引擎,以及配置文件VM。现在描述性能监视系统。

[0277] 图33是根据各种实施例的用于图2的PPU 200的性能监视系统3300的框图。如图所示,性能监视系统3300包括但不限于性能监视器3310、选择多路复用器3320、监视总线3330和性能多路复用器单元3340。性能监视器3310和选择多路复用器3320共同构成性能监视器模块(PMM)。每个GPC 242、每个分区单元262和每个系统管道230包括至少一个PMM。性能多路复用器单元3340被包括在被监视的每个单元内。性能多路复用器单元3340内的逻辑被包括在FE 540、SKED 550、CWD 560的一个或更多个中和/或其他合适的功能单元中。如本文进一步描述的,各种性能监视系统3300的所有组件与性能监视器聚合器(图33中未示出)通信。除了下面进一步描述的以外,性能监视系统3300的功能与图2的PM 236、图3的PM 360和PM 430基本相同。

[0278] 在操作中,性能多路复用器单元3340(0) - 3340(P)能够对PPU 200中的信号组进行可编程选择,这些信号组可以由相应的性能监视器3310监视。每个性能多路复用器3340可以选择发送到监视总线3330的一组信号。通过选择多路复用器3320选择来自监视总线3330的信号子集进行监视。通过选择多路复用器3320未被选择的来自监视总线3330的信号不受监视。来自PPU 200内部的信号成组连接到性能多路复用器单元3340(0) - 3340(P),以便一次选择一个组进行监视。性能多路复用器单元3340多路复用信号,使得特定信号组中的信号被选择为组。通过包括在特权寄存器地址空间3120中的一个或更多个寄存器来对包括在性能多路复用器单元3340中的多路复用器的选择输入进行编程。结果,由性能多路复用器单元3340传输到监视总线3330的特定信号是可编程的。

[0279] 监视总线3330从性能多路复用器单元3340(0) - 3340(P)接收信号组。发送到监视总线3330的每个信号作为输入连接到每个选择多路复用器3320。

[0280] 选择多路复用器3320包括一组单独的多路复用器3322 (0) - 3322 (M) 和3324 (0) - 3324 (N)。每个多路复用器3322 (0) - 3322 (M) 和3324 (0) - 3324 (N) 的输入侧从监视总线3330接收所有信号,并且选择一个信号进行发送。通过特权寄存器地址空间3120中包含的一个或更多个寄存器,对多路复用器3322 (0) - 3322 (M) 和3324 (0) - 3324 (N) 的选择输入进行编程。结果,由选择多路复用器3320发送的一组特定信号是可编程的。选择多路复用器3320将选择的信号发送到性能监视器3310。

[0281] 由于可编程性能多路复用器单元3340和可编程选择多路复用器3320的组成,由性能监视器3310监视的特定PPU信号是可编程的。

[0282] 性能监视器3310包括性能计数器阵列3312、阴影计数器阵列3314和触发功能表3316。性能监视器3310接收由选择多路复用器3320发送的信号。更具体地,阴影计数器阵列3314接收由多路复用器3322 (0) - 3322 (M) 发送的信号。类似地,触发功能表3316接收由多路复用器3324 (0) - 3324 (N) 发送的信号。如进一步描述的,阴影计数器阵列3314内的计数器基于从多路复用器3322 (0) - 3322 (M) 接收的信号以及各种触发条件来更新。通常,阴影计数器阵列3314包括一组一个或更多个信号计数器,其中每当从对应的多路复用器3322接收的信号处于特定逻辑状态时,每个计数器就递增。基于触发条件形式的某些信号,阴影计数器阵列3314中的值被传送到性能计数器阵列3312。性能计数器阵列3312包括阴影计数器阵列中包括的对应于信号计数器的一组一个或更多个信号计数器。

[0283] 在一种操作模式中,在转移到性能计数器阵列3312之后,阴影计数器阵列3314中的计数器被重置为零,使得阴影计数器阵列3314中存储的阴影计数器值始终对应于自前一次触发以来的活动。

[0284] 性能监视器3310可根据定义了性能计数器阵列3312和阴影计数器阵列3314中包括的计数器的数量的各种计数模式进行配置。计数模式还定义了如何以及何时触发性能计数器阵列3312和阴影计数器阵列3314,以及如何将来自性能计数器阵列3312的数据传输到PPU 200内的其他设备。这些各种计数模式可以分为两个主要性能监视模式-非流性能监视和流性能监视。

[0285] 在非流式性能监视模式下,触发功能表3316被编程为根据某些指定的逻辑信号表达式组合从多路复用器3324 (0) - 3324 (N) 接收的信号。当满足这些逻辑信号表达式中的一个或更多的条件时,触发功能表3316以逻辑触发器3350的形式向性能计数器阵列3312发送信号。响应于接收到逻辑触发器3350,性能计数器阵列3312采样并且存储阴影计数器阵列3314中的当前值。然后,通过特权寄存器地址空间3120读取性能计数器阵列3312中的值。

[0286] 在流式性能监视模式下,性能监视器聚合器 (PMA) 将PMA触发器3352形式的信号发送到性能计数器阵列3312。响应于接收到PMA触发器3352,性能计数器阵列3312采样并存储阴影计数器阵列3314中的当前值。性能监视器3310生成性能监视器 (PMM) 记录,该记录可以包括但不限于在从PMA接收到PMA触发器3352时性能计数器阵列3312中的值、性能监视器3310响应的PMA触发器总数的计数、SMC引擎ID和唯一标识在系统中生成记录的PMM的PMM ID。然后,这些PMM记录被发送到与一个或更多个性能监视器3310相关联的PMM路由器。然后,PMM路由器,发送PMM记录到PMA。在一些实施例,对于每个性能监视器3310的PMM ID可以经由特权寄存器地址空间3120中包含的一个或更多个寄存器进行编程。

[0287] 通常,PPU 200中的特定性能监视系统3300中的特定性能监视器3310与该特定性

能监视器3310所监视的信号位于相同的时钟频域内。然而,相对于PPU 200中的另一个性能监视器而言特定性能监视器3310可以位于相同的时钟频域或不同的时钟频域内。

[0288] 现在描述性能多路复用器单元3340的各种配置。

[0289] 图34A-图34B示出了根据各种实施例的图33的性能多路复用器单元3340的各种配置。

[0290] 如图34A所示,性能多路复用器单元3340 (0) 的第一配置包括但不限于信号组A 3420 (0) -3420 (P)、信号组B 3430 (0) -3430 (Q) 以及多路复用器3412 (0) 和3412 (1)。在操作中,多路复用器3412 (0) 选择信号组A 3420 (0) -3420 (P) 中的一个,其中信号组A 3420 (0) -3420 (P) 中的每一个是较大信号组C中的子组。多路复用器3412 (0) 选择信号组A 3420 (0) -3420 (P) 中的一个,并将选定的信号组传输到监视总线3330。类似地,多路复用器3412 (1) 选择信号组B 3430 (0) -3430 (Q) 中的一个,其中信号组B 3430 (0) -3430 (Q) 中的每一个是较大信号组D中的子组。多路复用器3412 (1) 选择信号组B 3430 (0) -3430 (Q) 中的一个,并将选定的子组传输到监视总线3330。通过特权寄存器地址空间3120中包含的一个或更多个寄存器,对性能多路复用器单元3340 (0) 中包括的多路复用器3412的选择输入进行编程。结果,性能多路复用器单元3340 (0) 发送的信号集是可编程的。

[0291] 如图34B所示,性能多路复用器单元3340 (1) 的第二配置包括但不限于信号组C 3440 (0) -3440 (R) 和多路复用器3412 (2)。在操作中,多路复用器3412 (2) 选择信号组C 3440 (0) -3440 (R) 中的一个,其中信号组C 3440 (0) -3440 (R) 中的每一个是较大信号组E中的子组。多路复用器3412 (2) 选择信号组C 3440 (0) -3440 (R) 中的一个,并将选定的信号组传输到监视总线3330。在性能多路复用器单元3340 (1) 的配置中,几个信号被传输到多个信号组。特别地,信号C1 3450被发送到信号组C 3440 (0) 和信号组C 3440 (1)。类似地,信号C2 3452被发送到信号组C 3440 (1) 和信号组C 3440 (2)。通过包括在特权寄存器地址空间3120中的一个或更多个寄存器,对包括在性能多路复用器单元3340 (1) 中的多路复用器3412 (2) 的选择输入进行编程。结果,由性能多路复用器单元3340 (1) 传输的信号集是可编程的。性能多路复用器单元3340 (1) 的配置在使信号在多个信号组3440中可用促进性能监视系统3300的单次通过中某些信号组的可见性时可能有用。

[0292] 图35是根据各种实施例的用于图2的PPU 200的性能监视器聚合系统3500的框图。如图所示,性能监视器聚合系统3500包括但不限于GPC 242 (0) -242 (M)、分区单元262 (0) -262 (N)、交叉开关单元250、控制交叉开关和SMC仲裁器510、PM管理系统3530和性能分析系统3540。

[0293] 在操作中,GPC 242 (0) -242 (M) 为一个或更多个系统管道230执行各种处理任务。每个GPC 242包括能够同时执行大量线程的多个并行处理核心,并且具有与其他GPC 242任意程度的独立性和/或隔离。每个GPC 242 (0) -242 (M) 包括一个或更多个PM 360 (0) -360 (M) 和GPC PMM路由器3514 (0) -3514 (M)。PM 360 (0) -360 (M) 的功能基本上类似于图33的性能监视器3310。PM 360 (0) -360 (M) 生成PMM记录,其中包括相应GPC 242 (0) -242 (M) 的性能数据。PM 360 (0) -360 (M) 将这些PMM记录发送到相应的GPC PMM路由器3514 (0) -3514 (M) 并从中接收数据。GPC PMM路由器3514 (0) -3514 (M) 通过交叉开关单元250将PMM记录传输到PM管理系统3530。

[0294] 分区单元262 (0) -262 (N) 提供对PPU存储器(图35中未示出)内的DRAM的高带宽存

存储器访问。每个分区单元262利用彼此并行的不同DRAM执行存储器访问操作,从而有效地利用PPU存储器的可用存储器带宽。分区单元262 (0) - 262 (N) 的每一个包括一个或多个PM 430 (0) - 430 (N) 和分区单元 (PU) PMM路由器3524 (0) - 3524 (N) 。PM 430 (0) - 430 (N) 的功能基本上类似于图33的性能监视器3310。PM 430 (0) - 430 (N) 生成PMM记录,其中包括对应的分区单元262 (0) - 262 (N) 的性能数据。PM 430 (0) - 430 (N) 将这些PMM记录发送到相应的PU PMM路由器3524 (0) - 3524 (N) 并从中接收数据。PU PMM路由器3524 (0) - 3524 (N) 又通过控制交叉开关和SMC仲裁器510将PMM记录传输到PM管理系统3530。

[0295] PM管理系统3530控制PMM记录的收集并存储PMM记录以用于报告目的。PM管理系统3530包括但不限于系统性能监视器3532、系统 (SYS) PMM路由器3534、性能监视器聚合器 (PMA) 3536、高速集线器 (HSHUB) 和传输逻辑3539。

[0296] 系统PM 3532的功能基本上类似于图33的性能监视器3310。系统PM 3532生成PMM记录,该记录包含针对特定GPC 242或分区单元262中未包含的系统范围组件的性能数据。系统PM 3532将这些PMM记录发送到系统PMM路由器3534并从中接收数据。系统PMM路由器3534又将PMM记录发送到PMA 3536。

[0297] PMA 3536为包括PM 360 (0) - 360 (M) 、PM 430 (0) - 430 (N) 和系统PM 3532的各种性能监视器生成触发。PMA 3536通过两种技术生成这些触发器。在第一技术中,当系统管道230从主机接口220接收命令时,PMA 3536响应于每个系统管道230发送的信号来生成触发。在第二技术中,PMA 3536通过周期性地将编程控制的触发脉冲发送到PM来生成触发。通常,性能监视器聚合系统3500除了与任何系统管道230独立的另一个触发脉冲发生器之外,还至少包括一个与每个系统管道230相对应的可编程触发脉冲发生器。PMA 3536通过控制交叉开关和SMC仲裁器510将触发信号发送到GPC PMM路由器3514 (0) - 3514 (M) 和PU PM 3524 (0) - 3524 (N) 。PMA 3536直接通过PM管理系统3530内部的通信链路将触发发送到系统PMM路由器3534。PMM路由器将PMA触发器发送到相应的PM。在一些实施例中,这些触发器采取以下结合图36描述的触发分组的形式。

[0298] 图36示出了根据各种实施例的与图35的性能监视器聚合系统3500相关联的触发分组的格式。触发分组的目的是将有关触发源的信息传达给性能监视器3310。性能监视器3310的每一个利用该信息来确定是否响应特定的触发。在这方面,触发分组包含可由每个性能监视器3310用来确定是否响应特定触发分组的信息。与特定SMC引擎700相关联的每个性能监视器3310使用与该SMC引擎700相对应的SMC引擎ID进行编程。这样的性能监视器3310响应包括相同SMC引擎ID的每个SMC触发分组。与特定SMC引擎700不相关联的每个性能监视器3310或使用无效的SMC引擎ID编程的每个性能监视器3310,均不响应每个SMC触发分组。相反,这种性能监视器3310响应共享的触发分组。

[0299] 图3600示出了触发分组的一般格式。如图所示,图3600包括指示分组是PM触发的分组类型3602、触发类型3604和触发有效载荷3606。PM触发类型3604是标识触发格式的类型枚举值。例如,为了标识三种不同的触发分组类型,PM触发类型3604可以是2位值。触发有效载荷3606包括基于PM触发类型3604而不同的数据。现在描述三种不同类型的触发分组,其中三种类型的触发分组对应于性能监视数据的三种类别 (传统数据、每个SMC数据以及共享数据) 。

[0300] 图3610示出了传统触发分组的格式。传统触发分组包括指示分组是PM触发的分组

类型3602和指示触发分组是传统触发分组的触发类型3614。传统触发分组的触发有效载荷3606包括未使用字段3616。

[0301] 图3620图示了每个SMC触发分组的格式。每个SMC触发分组包括指示分组是PM触发的分组类型3602和指示触发分组是SMC触发分组的触发类型3624。每个SMC触发分组的触发有效载荷3606包括SMC引擎ID字段。SMC引擎ID字段3626标识触发器适用的特定SMC引擎700。

[0302] 图3630示出了共享触发分组的格式。共享触发分组包括指示分组是PM触发的分组类型3602和指示触发分组是共享触发分组的触发类型3634。共享触发分组的触发有效载荷3606包括未使用字段。

[0303] 由PMA 3536发送的触发分组的类型由触发源以及包括在对应于每个触发源的特权寄存器地址空间3120中的一个或多个寄存器确定,该寄存器被编程为指示PMA应该为该来源发送的触发分组类型。在一种操作模式中,对PMA进行编程,以使响应于关联于SMC引擎的源而生成的触发分组是将SMC引擎ID设置为相应的SMC引擎的每个SMC触发分组,以及响应于不关联于SMC引擎的源而生成的触发分组是共享触发分组。触发分组可以以任何技术上可行的速率生成,每个计算周期最多包含一个触发分组。

[0304] 响应于从PMA 3536接收到触发分组,每个PM检查触发类型和触发有效载荷,以确定PM是否应响应触发。每个PM无条件响应传统触发分组。在每个SMC触发分组的情况下,仅当触发有效载荷中包含的SMC引擎ID与通过寄存器特权寄存器地址空间3120编程分配给PM的SMC引擎ID匹配时,PM才响应。编程在此寄存器中的无效的SMC引擎ID确保PM不响应每个SMC触发分组。在共享触发分组的情况下,仅当PM已通过特权寄存器地址空间3120中的寄存器编程为做出响应时,PM才响应。在一种操作模式下,作为监视单元均唯一分配给SMC引擎的所有PM被编程为响应具有相应的SMC引擎ID有效载荷的每个SMC触发分组,而所有其他PM都被编程为响应共享的触发分组,但不响应每个SMC触发分组。

[0305] 在PM确定保证对触发的响应的情况下,PM对包括在相应的PM中的计数器进行采样。然后,响应的PM传输PMM记录,其中包括采样的计数器值、响应的触发器总数、分配给PM的SMC引擎ID以及在系统中唯一标识PM的PMM ID。PMM路由器、PMA 3536和/或性能分析系统3540利用PMM ID来识别哪个PM发送了相应的PMM记录。然后,PMM路由器将所述记录发送到PMA 3536。更具体地说,GPC PMM路由器3514 (0) - 3514 (M) 通过交叉开关单元250和高速集线器3538将PMM记录发送到PMA 3536。PU PMM路由器3524 (0) - 3524 (N) 通过控制交叉开关和SMC仲裁器510将PMM记录发送到PMA 3536。系统PMM路由器3534直接通过通信链路将PMM记录传输到PM管理系统3530。以这种方式,PMA 3536从PPU 200中的所有相关PM接收PMM记录。

[0306] 在一些实施例中,当传输触发时,PMA 3536另外生成PMA记录。通常,PMA记录的目的是记录产生特定PMA触发的时间戳,并且将来自于响应于那个时间戳的PMA触发的性能监视器3310对应的PMM记录相关联。PMA记录包括但不限于时间戳、与PMA触发的源相关联的SMC引擎ID、由具有相同SMC引擎ID的源生成的触发总数和相关联的元数据。当性能监视器3310接收到触发时,性能监视器3310也生成具有触发计数的PMM记录。随后,当解析PMA记录和PMM记录时,具有特定触发计数的PMM记录可以与具有相同触发计数的PMA记录相关联。以这种方式,基于相关联的PMA记录的时间戳来建立与PMM记录相对应的时间戳。结果,由PMM记录反映的PPU 200的行为与由来自相同来源的两个相邻的PMA触发器限定的时间范围精

确相关联。

[0307] 在接收PMM记录并生成PMA记录时,PMA 3536通过高速集线器3538将PMM记录和PMA记录以记录缓冲区的形式存储到PPU存储器中的数据存储中。高速集线器3538发送PMM记录和PMA记录到分区单元262(0)-262(N)。然后,分区单元将PMM记录和PMA记录存储在PPU存储器中的记录缓冲区中。附加地或可选地,高速集线器3538经由传输逻辑3539将PMM记录和PMA记录传输到性能分析系统3540。在一些实施例中,高速集线器3538、传输逻辑3539和性能分析系统3540可以经由PCIe链接彼此通信。用户可以在性能分析系统3540上查看PMM记录和PMA记录,以便表征如在PMM记录中反映的PPU 200的行为。性能分析系统3540收集具有相同触发计数的PMM记录和PMA记录。然后,性能分析系统3540使用来自PMA记录的时间戳和来自PMM记录的具有相同触发计数的性能数据,来确定与性能数据相关联的时间戳。在一些实施例中,性能分析系统3540可以将性能记录缓冲区作为虚拟存储器来访问。作为将性能记录缓冲区放置在不同的虚拟地址空间中的结果,不同SMC引擎700的性能监视数据可以彼此隔离,如现在所描述的。

[0308] PMA 3536提供了几个SMC引擎700之间的性能监视数据的隔离。特别地,PMA 3536基于操作模式将PMM记录和PMA记录分类为类别。当PPU 200在传统模式下操作时,PPU 200将任务作为单个硬件资源集群执行,而不是作为具有单独的SMC引擎700的单独PPU分区600执行。在传统模式下,PMA 3536将PMM记录和PMA记录存储在单个类别中,作为单独一组性能监视数据。当PPU 200以SMC模式运行时,PPU 200将任务作为具有单独的SMC引擎700的单独PPU分区600执行。在SMC模式下,PMA 3536使用PMM记录和PMA记录的SMC引擎ID字段将PMM记录和PMA记录分类并存储在不同类别中。具有每个SMC引擎ID的记录以记录缓冲区的形式存储在单独的数据存储中,该记录缓冲区可从与相应的SMC引擎700匹配的不同虚拟地址空间访问。如上所述,不可追溯到特定SMC引擎700的PMM记录和PMA记录包含无效SMC引擎ID。PMA将这些记录以虚拟地址空间中的SMC记录缓冲区的形式存储在单独的数据存储中,虚拟地址空间可由具有足够特权访问所有SMC引擎700的数据的任何授权实体访问。此类授权实体包括但不限于虚拟环境中的管理程序124和非虚拟环境中的root用户或操作系统核心。每个SMC引擎700可以通过从授权实体请求数据来访问在非SMC PMA记录缓冲区中的部分或全部性能监视数据。

[0309] 在一些实施例中,PMA 3536被配置使得生成与每个SMC引擎700相对应的触发,以与同一SMC引擎700的上下文切换事件一致。在这样的实施例中,PM被配置使得阴影计数器阵列3314中的计数器在每次触发后重置为零,以便在启用时间切片的同时,每个SMC引擎ID从PM传输到PMA 3536的数据可归因于单个上下文或VM。

[0310] 图37是根据各种实施例的用于监视图2的PPU 200的性能的方法步骤的流程图。尽管结合图1-图23的系统描述了方法步骤,但是本领域普通技术人员将理解,被配置为以任何顺序执行方法步骤的任何系统都在本公开的范围。

[0311] 如图所示,方法3700在步骤3702开始,其中PMA 3536生成并发送触发至PM 3310。此外,PMA 3536生成对应的PMA记录,其包括时间戳以及,可选地,还包括与触发的源相对应的SMC引擎ID。PM 3310接收触发以采样性能数据。

[0312] 在步骤3704,响应于接收触发,PM 3310确定是否保证对触发的响应。每个PM 3310检查来自触发分组的触发类型和触发有效载荷,以确定PM 3310是否应响应触发。每个PM

3310无条件响应传统触发分组。在每个SMC触发分组的情况下,仅当触发有效载荷中包含的SMC引擎ID与通过特权寄存器地址空间3120中的寄存器编程分配给PM 3310的SMC引擎ID匹配时,PM 3310才响应。编程该寄存器中的无效额SMC引擎ID确保PM 3310不响应每个SMC触发分组。在共享触发分组的情况下,仅当PM 3310已被编程为通过特权寄存器地址空间3120中的寄存器进行响应时,PM 3310才响应。在一种操作模式下,作为监视单元均唯一分配给SMC引擎700的所有PM 3310被编程为响应具有相应的SMC引擎ID有效载荷的每个SMC触发分组。所有其他PM 3310被编程为响应共享的触发分组,但不响应每个SMC触发分组。

[0313] 如果需要响应,则性能计数器阵列3312采样并将当前值存储在阴影计数器阵列3314中。在非流模式下,其他组件可以通过特权寄存器接口集线器512读取性能计数器阵列中的值。

[0314] 在流传输模式下,该方法进行到步骤3706,其中PM 3310发送采样的性能数据,并且PMA 3536从PM 3310接收采样的性能数据。PM 3310生成PMM记录,该记录包括当从PMA 3536接收到PMA触发器3352时,性能计数器阵列3312中的值。然后,这些PMM记录被发送到与特定性能监视器3310相关联的PMM路由器。PMM路由器又将PMM记录发送到PMA 3536。

[0315] 在步骤3708,PMA 3536基于操作模式将PMM记录和PMA记录分类到类别。当PPU 200以传统模式操作时,PMA 3536将PMM记录和PMA记录在单个类别中分类作为单组性能监视数据。当PPU 200在SMC模式下操作时,PMA 3536使用PMM记录和PMA记录的SMC引擎ID字段将PMM记录和PMA记录分类为不同的类别。如上所述,不可追溯到特定SMC引擎700的PMM记录和PMA记录包含无效的SMC引擎ID。PMA 3536将这些PMM记录和PMA记录分类到一个单独的类别中。

[0316] 在步骤3710,PMA 3536将PMM记录和/或PMA记录存储到PPU存储器中的PMA记录缓冲区中。当PPU 200以传统模式操作时,PMA 3536将PMM记录和PMA记录存储在单个类别中作为单组性能监视数据。当PPU 200在SMC模式下操作时,PMA 3536将与每个SMC引擎ID相关联的PMM记录和PMA记录存储在可从与相应的SMC引擎700匹配的不同虚拟地址空间访问的单独数据存储中。PMA 3536将不能追溯到特定SMC引擎700的PMM记录和PMA记录在虚拟地址空间中以非SMC PMA记录缓冲区的形式在单独的数据存储中存储,该虚拟地址空间只有具有足够特权来访问所有SMC引擎700的数据的任何授权实体才能访问。这样的授权实体包括但不限于虚拟化环境中的管理程序124和非虚拟化环境中的root用户或操作系统核心。每个SMC引擎700可以通过从授权实体请求数据来访问非SMC PMA记录缓冲区中的部分或全部性能监视数据。

[0317] 更具体地,PMA 3536将PMM记录和PMA记录流传输到高速集线器3538。高速集线器3538将PMM记录和PMA传输发送到分区单元262 (0) - 262 (N)。然后,分区单元将PMM记录和PMA记录存储在PPU存储器中的PMA记录缓冲区中。基于记录的SMC引擎ID字段选择每个记录的PMA记录缓冲区,以使每个记录最终驻留在PPU存储器中,该PPU存储器可在与PMA记录缓冲区对应的SMC引擎匹配的虚拟地址空间中访问。在单独的数据存储中不可追溯到特定SMC引擎700的PMM记录和PMA记录只能由授权实体访问。

[0318] 在步骤3712,PMA 3536经由高速集线器3538和传输逻辑3539将PMA记录和/或PMM记录发送到性能分析系统3540。另外或可选地,性能分析系统3540通过虚拟地址空间中的一个或多个虚拟地址访问PMA记录和/或PMM记录。通常,性能分析系统3540包括在CPU

110和/或任何其他技术上可行的处理器上执行的软件应用。性能分析系统3540直接访问虚拟存储器以访问PMA记录和/或PMM记录。虚拟存储器可以与PPU 200和/或CPU 110相关联。用户可以在性能分析系统3540上查看PMA记录和/或PMM记录,以便表征如在PMA记录和/或PMM记录中反映的PPU 200的行为。然后,方法3700终止。

[0319] SMC引擎的电源和时钟频率管理

[0320] 复杂系统,诸如图2的PPU 200可能消耗大量功率。更具体地说,PPU 200内的某些组件可以在不同的时间点具有彼此不同的功耗水平。在一个示例中,一个PPU分区600中的组件可以执行计算和/或图形密集型任务,从而相对于其他PPU分区600增加了功耗。在另一示例中,由于泄漏当前和相关因素,即使处于空闲状态时PPU分区600也可能消耗功率。此外,增加的功耗可能导致较高的工作温度,进而可能导致性能降低。结果,PPU 200包括功率和时钟频率管理,其考虑一个PPU分区600内的功耗如何可能负面影响其他PPU分区600的性能。

[0321] 图38是根据各种实施例的用于图2的PPU 200的功率和时钟频率管理系统3800的框图。功率和时钟频率管理系统3800包括但不限于电路子部分3810(0)-3810(N)、功率门控制器3820和时钟频率控制器3830。

[0322] 电路子部分3810(0)-3810(N)的每一个包括任何级别的粒度的PPU 200中包含的任何组件集。在这方面,电路子部分3810(0)-3810(N)的每一个可以包括但不限于系统管道230、PPU分区600、PPU切片610、SMC引擎700或其任何技术上可行的子集。

[0323] 在操作中,功率门控制器3820监视电路子部分3810(0)-3810(N)的活动状态。如果功率门控制器3820确定特定电路子部分(诸如电路子部分3810(2))处于空闲状态,则功率门控制器3820将电路子部分3810(2)的供应电压减小到小于运行电压但维持存储在存储器中的数据电压。可选地,功率门控制器3820可以从电路子部分3810(2)移除电源,从而关闭电路子部分3810(2)。随后,如果需要电路子部分3810(2)来执行某些任务,则功率门控制器3820将电路子部分3810(2)的电源电压增加到适合于运行的电压。

[0324] 时钟频率控制器3830监视电路子部分3810(0)-3810(N)的功耗。如果时钟频率控制器3830确定特定的电路子部分(例如电路子部分3810(3))相对于其他电路子部分3810消耗更多的功率,则时钟频率控制器3830降低与电路子部分3810(3)相关联的时钟信号的频率。结果,减少了电路子部分3810(3)消耗的功率。随后,如果时钟频率控制器3830确定电路子部分3810(3)相对于其他电路子部分3810消耗更少的功率,则时钟频率控制器3830增加与电路子部分3810(3)相关联的时钟信号的频率,从而增加电路子部分3810(3)的性能。

[0325] 以这种方式,功率门控制器3820和时钟频率控制器3830降低了PPU 200的总体功耗,并减少了一个PPU分区600由于温度影响而对另一个PPU分区600所产生的负面影响。

[0326] 图39是根据各种实施例的用于管理图2的PPU 200的功耗的方法步骤的流程图。尽管结合图1至图25的系统描述了方法步骤,但是本领域普通技术人员将理解,配置成以任何顺序执行方法步骤的任何系统都在本公开的范围之内。

[0327] 如图所示,方法3900在步骤3902开始,其中PPU 200的电源和时钟频率管理系统3800监视在PPU 200的各个电路子部分3810上执行的VM的活动状态。在步骤3904,电源和时钟频率管理系统3800确定任何电路子部分3810是否处于空闲状态。如果没有电路子部分3810处于空闲状态,则该方法进行到步骤3908。但是,如果一个或多个电路子部分3810处

于空闲状态,则该方法进行到步骤3906,在此功率和时钟频率管理系统3800降低空闲的电路子部分3810的电源电压。特别地,功率和时钟频率管理系统3800中的功率门控制器3820将电路子部分3810(2)的供应电压减小到小于运行电压但维持存储在存储器中的数据的电压。可选地,功率门控制器3820可以从电路子部分3810(2)移除电源,从而关闭电路子部分3810(2)。

[0328] 在步骤3908,功率和时钟频率管理系统3800监视PPU中每个SMC引擎700的功耗。在步骤3910,电源和时钟频率管理系统3800确定一个或多个SMC引擎700是否相对于其他MC引擎700正在消耗过多的功率。如果没有SMC引擎700正在消耗过多的功率,则该方法进行到步骤3902以继续监视。然而,如果一个或多个SMC引擎700正在消耗过多的功率,则该方法进行到步骤3912,在此功率和时钟频率管理系统3800内的时钟频率控制器3830减小与正在消耗过多功率的SMC引擎700相关联的一个或多个电路子部分3810的时钟频率。然后该方法进行到步骤3902以继续监视。

[0329] 总之,各种实施例包括可以被分为分区的并行处理单元(PPU)。每个分区被配置为同时执行与多个处理上下文相关联的处理任务。给定的分区包括GPU资源的一个或多个逻辑分组或“切片”。每个切片提供足够的计算、图形和存储器资源,以模拟整个PPU的操作。在CPU上执行的管理程序代表管理员用户执行各种技术来对PPU进行分区。访客用户被分配给分区,并且然后可以在该分区内执行与分配给任何其他分区的任何其他访客用户隔离的处理任务。

[0330] 相对于现有技术,所公开的技术的一个技术优势在于,利用所公开的技术,PPU可以同时支持多个处理上下文并且在功能上彼此隔离。因此,多个CPU进程可以经由多个不同的处理上下文有效地利用PPU资源,而不会彼此干扰。所公开的技术的另一技术优势在于,因为可以使用所公开的技术将PPU分区为隔离的计算环境,所以相对于依赖于处理子上下文来提供多租户功能的现有技术方法,PPU可以支持更可靠的多租户形式。因此,当实现所公开的技术时,PPU变得更适合于基于云的部署,其中可以向不同的和潜在竞争的实体提供对同一PPU内不同分区的访问。这些技术优势代表相对于现有技术方法的一种或更多种技术进步。

[0331] 1.一些实施例包括一种计算机实现的方法,包括:对包括在处理器中的硬件资源集合进行分区,以生成包括硬件资源的第一子集的第一逻辑分区;以及在所述第一逻辑分区内生成多个引擎,其中包括在所述多个引擎中的每个引擎被分配所述硬件资源的第一子集的不同部分,并且与包括在所述多个引擎中的所有其他引擎在功能上隔离地执行。

[0332] 2.根据条款1所述的计算机实现的方法,其中对所述硬件资源集合进行分区还包括:对所述硬件资源集合进行分区,以生成包括硬件资源的第二子集的第二逻辑分区,其中所述硬件资源的第一子集包括比所述硬件资源的第二子集更多的硬件资源。

[0333] 3.根据条款1-2中任一项所述的计算机实现的方法,其中所述包括在所述多个引擎中的每个引擎执行与不同的处理上下文相关联的一组处理任务。

[0334] 4.根据条款1-3中的任一项的计算机实现的方法,还包括:使得包括在多个引擎中的第一引擎在给定的时间间隔期间执行与处理上下文相关联的一组处理任务,以及使包括在多个引擎中的第二引擎在给定的时间间隔内执行一个或多个上下文切换操作。

[0335] 5.根据条款1-4中任一项所述的计算机实现的方法,还包括:使包括在所述多个引

擎中的第一引擎在给定的时间间隔内执行与第一处理上下文相关联的第一组处理任务;以及响应于在所述第二引擎执行与第二处理上下文相关联的第二组处理任务时发生的故障,使包括在所述多个引擎中的第二引擎在所述给定的时间间隔期间复位。

[0336] 6. 根据条款1-5中的任一项的计算机实现的方法,还包括:使包括在所述多个引擎中的第一引擎在第一时间间隔期间执行与第一处理子上下文相关联的第一组处理任务;以及使所述第一引擎在所述第一时间间隔期间执行与第二处理子上下文相关联的第二组处理任务,其中所述第一处理子上下文和所述第二处理子上下文均从第一处理上下文导出,并且其中根据所述第一处理上下文配置所述第一引擎。

[0337] 7. 根据条款1-6中任一项所述的计算机实现的方法,还包括:分析所述硬件资源集合以识别第一硬件资源的非功能实例;确定所述硬件资源的子集包括所述第一硬件资源的功能实例;以及电隔离所述第一硬件资源的所述非功能实例。

[0338] 8. 根据条款1-7中的任一项所述的计算机实现的方法,还包括:进一步分区所述硬件资源集合以生成第二逻辑分区;根据所述第一逻辑分区配置所述第二逻辑分区;识别包括在所述第一分区内的执行与第一处理上下文相关联的第一处理任务的第一引擎;以及将所述第一处理上下文迁移到包括在所述第二逻辑分区内的第二引擎。

[0339] 9. 根据条款1-8中的任一项所述的计算机实现的方法,还包括:确定所述第一逻辑分区包括第一硬件资源的非功能实例;确定第二逻辑分区包括所述第一硬件资源的功能实例;停用所述第一硬件资源的所述功能实例;根据所述第一逻辑分区配置所述第二逻辑分区;以及将与所述第一逻辑分区相关联的第一处理上下文迁移到所述第二逻辑分区。

[0340] 10. 根据条款1-9中的任一项的计算机实现的方法,还包括:接收与第一计算环境相关联的第一输入,其中基于第一组许可在所述第一计算环境内的所述处理器上执行第一组操作;以及接收与第二计算环境相关联的第二输入,其中基于第二组许可在所述第二计算环境内的所述处理器上执行第二组操作,并且所述第一组许可包括比所述第二组更多的许可,其中基于所述第一输入分区所述硬件资源集合,以及其中基于所述第二输入配置包括在所述多个引擎中的第一引擎。

[0341] 11. 一些实施例包括一种存储程序指令的非暂时性计算机可读介质,,该程序指令在由处理器执行时,使处理器执行以下步骤:在包括硬件资源集合的处理器内生成第一逻辑分区,其中所述第一逻辑分区包括硬件资源的第一子集;以及在所述第一逻辑分区内生成多个引擎,其中包括在所述多个引擎中的每个引擎被分配所述硬件资源的第一子集的不同部分,并且与包括在所述多个引擎中的所有其他引擎在功能上隔离地执行。

[0342] 12. 根据条款11所述的非暂时性计算机可读介质,其中包括在所述多个引擎中的每个引擎执行与不同的处理上下文相关联的一组处理任务。

[0343] 13. 根据条款11-12中的任一项的非暂时性计算机可读介质,还包括以下步骤:使包括在所述多个引擎中的第一引擎在给定的时间间隔期间执行与处理上下文相关联的一组处理任务;以及使包括在所述多个引擎中的第二引擎在所述给定的时间间隔期间执行一个或更多个上下文切换操作。

[0344] 14. 根据条款11-13中的任一项的非暂时性计算机可读介质,还包括以下步骤:使包括在所述多个引擎中的第一引擎在给定的时间间隔期间执行与第一处理上下文相关联的第一组处理任务;以及响应于在所述第二引擎执行与第二处理上下文相关联的第二组处

理任务时发生的故障,使包括在所述多个引擎中的第二引擎在所述给定的时间间隔期间复位。

[0345] 15.根据条款11-14中的任一项的非暂时性计算机可读介质,还包括以下步骤:使包括在所述多个引擎中的第一引擎在第一时间间隔期间执行与第一处理子上下文相关联的第一组处理任务;以及使所述第一引擎在所述第一时间间隔期间执行与第二处理子上下文相关联的第二组处理任务,其中所述第一处理子上下文和所述第二处理子上下文均从第一处理上下文导出,并且其中根据所述第一处理上下文配置所述第一引擎。

[0346] 16.根据条款11-15中的任一项所述的非暂时性计算机可读介质,还包括以下步骤:确定所述第一逻辑分区包括第一硬件资源的非功能实例;确定第二逻辑分区包括所述第一硬件资源的功能实例;停用所述第一硬件资源的所述功能实例;根据所述第一逻辑分区配置所述第二逻辑分区;以及将与所述第一逻辑分区相关联的第一处理上下文迁移到所述第二逻辑分区。

[0347] 17.根据条款11-16中的任一项所述的非暂时性计算机可读介质,还包括以下步骤:接收与主机计算环境相关联的分区输入;以及接收与访客计算环境相关联的配置输入,其中基于所述分区输入分区所述硬件资源集合,以及其中基于所述配置输入配置包括在所述多个引擎中的第一引擎。

[0348] 18.根据条款11-17中的任一项所述的非暂时性计算机可读介质,其中所述多个引擎被配置为实现多个图形处理管线。

[0349] 19.根据条款11-18中的任一项所述的非暂时性计算机可读介质,其中分区所述硬件资源集合的步骤包括:基于一组位激活与所述处理器相关联的逻辑边界集合,其中所述包括在所述一组位中的每个位对应于包括在所述逻辑边界集合中的不同逻辑边界。

[0350] 20.一些实施例包括一种系统,该系统包括:存储器,用于存储软件应用程序;以及处理器,其在执行所述软件应用程序时,被配置为执行以下步骤:分区处理器中包括的硬件资源集合,以生成包括在硬件资源的第一子集中的第一逻辑分区,以及在所述第一逻辑分区内生成多个引擎,其中包括在所述多个引擎中的每个引擎被分配所述硬件资源的第一子集的不同部分,并且与包括在所述多个引擎中的所有其他引擎在功能上隔离地执行。

[0351] 以任何方式,在任何权利要求中记载的任何权利要求元素和/或在本申请中描述的任何元素的任何和所有组合都落入本实施例和保护的预期范围内。

[0352] 出于说明的目的已经给出了各种实施例的描述,但并不意图是穷举性的或限于所公开的实施例。在不脱离所描述的实施例的范围和精神的情况下,许多修改和变化对于本领域普通技术人员将是显而易见的。

[0353] 本发明实施方案的各方面可以被实现为系统、方法或计算机程序产品。因此,本公开的各方面可以采取完全硬件实施例、完全软件实施例(包括固件、驻留软件、微代码等)或结合了软件和硬件方面的实施例,这些方面在本文中通常都统称为“模块”、“系统”或“计算机”。另外,在本公开中描述的任何硬件和/或软件技术、过程、功能、组件、引擎、模块或系统可以被实现为电路或电路集合。此外,本公开的各方面可以采取体现在其上体现有计算机可读程序代码的一个或更多个计算机可读介质中体现的计算机程序产品的形式。

[0354] 可以利用一种或更多种计算机可读介质的任何组合。所述计算机可读介质可以是计算机可读信号介质或计算机可读存储介质。计算机可读存储介质例如可以是但不限于电

子、磁性、光学、电磁、红外或半导体系统、装置或设备,或前述的任何适当组合。计算机可读存储介质的更具体示例(非详尽列表)将包括以下内容:具有一根或更多根电线的电连接、便携式计算机软盘、硬盘、随机存取存储器(RAM)、只读存储器(ROM)、可擦可编程只读存储器(EPROM或闪存)、光纤、便携式光盘只读存储器(CD-ROM)、光学存储设备、磁存储设备或任何其他前述的适当组合。在本文的环境中,计算机可读存储介质可以是任何有形介质,其可以包含或存储供指令执行系统、装置或设备使用或与其结合使用的程序。

[0355] 上面参考根据本公开的实施例的方法、装置(系统)和计算机程序产品的流程图和/或框图描述了本公开的各方面。将理解的是,流程图图示和/或框图的每个框以及流程图图示和/或框图中的框的组合可以由计算机程序指令来实现。可以将这些计算机程序指令提供给通用计算机、专用计算机或其他可编程数据处理设备的处理器以产生机器。当指令通过计算机或其他可编程数据处理设备的处理器执行时,使得能够实现流程图和/或框图方框中指定的功能/动作。这样的处理器可以是但不限于通用处理器、专用处理器、应用程序专用处理器或现场可编程门阵列。

[0356] 在附图中的流程图和框图示出根据本发明的各种实施例的系统、方法和计算机程序产品可能实现的架构、功能和操作。就这一点而言,流程图或框图中的每个框可以代表代码的模块、片段或部分,其包括用于实现一个或更多个指定的逻辑功能的一个或更多个可执行指令。还应注意,在一些替代实施方式中,方框中指出的功能可以不按图中指出的顺序发生。例如,取决于所涉及的功能,实际上可以基本上同时执行连续示出的两个框,或者有时可以以相反的顺序执行这些框。还应注意,框图和/或流程图说明的每个方框以及框图和/或流程图说明中的方框的组合可以由执行指定功能或动作的基于专用硬件的系统或专用硬件和计算机指令的组合来实现。

[0357] 尽管前面针对本公开的实施例,但是可以在不脱离本公开的基本范围的情况下设计本公开的其他和进一步的实施例,并且本公开的范围由所附权利要求确定。

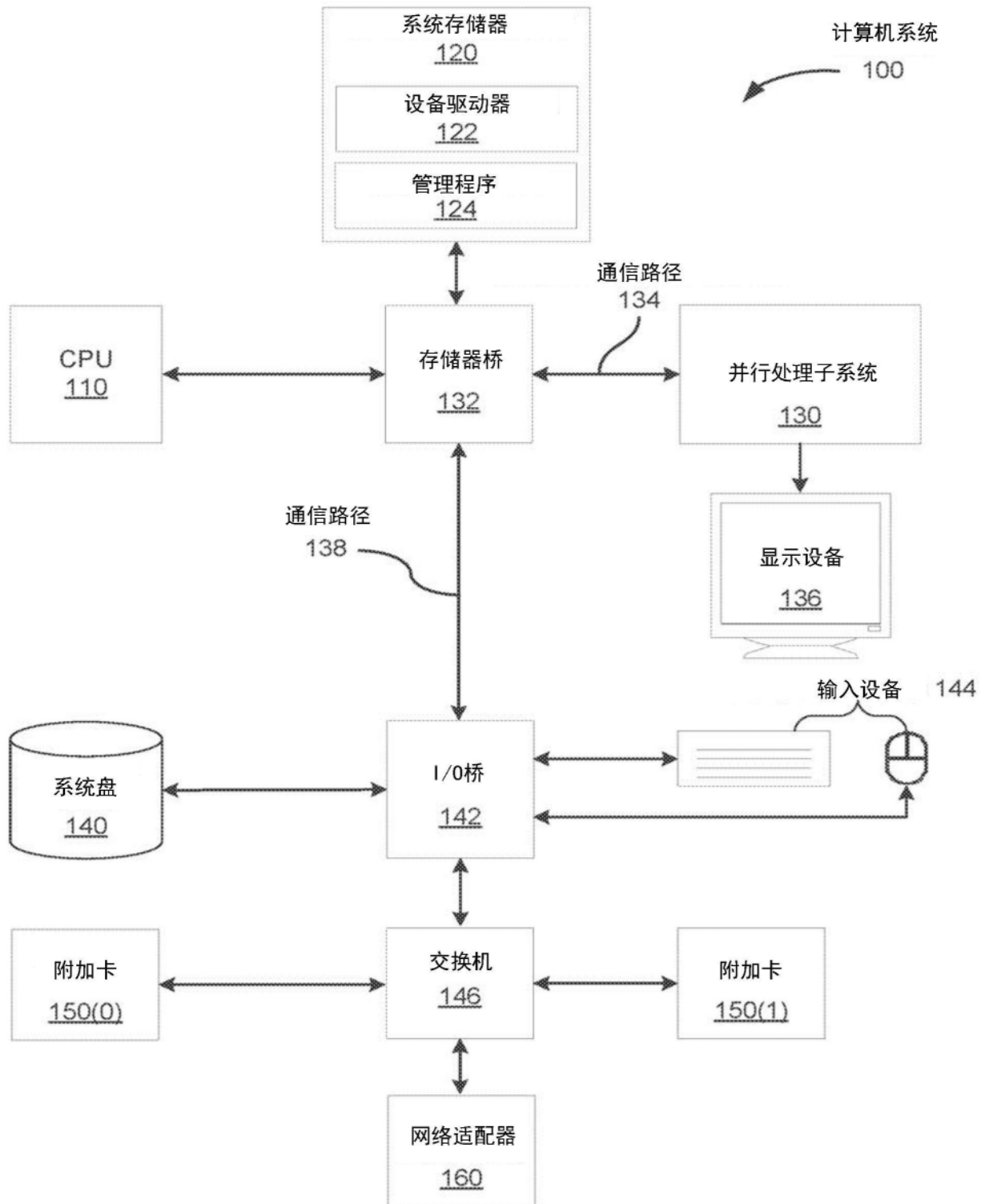


图1

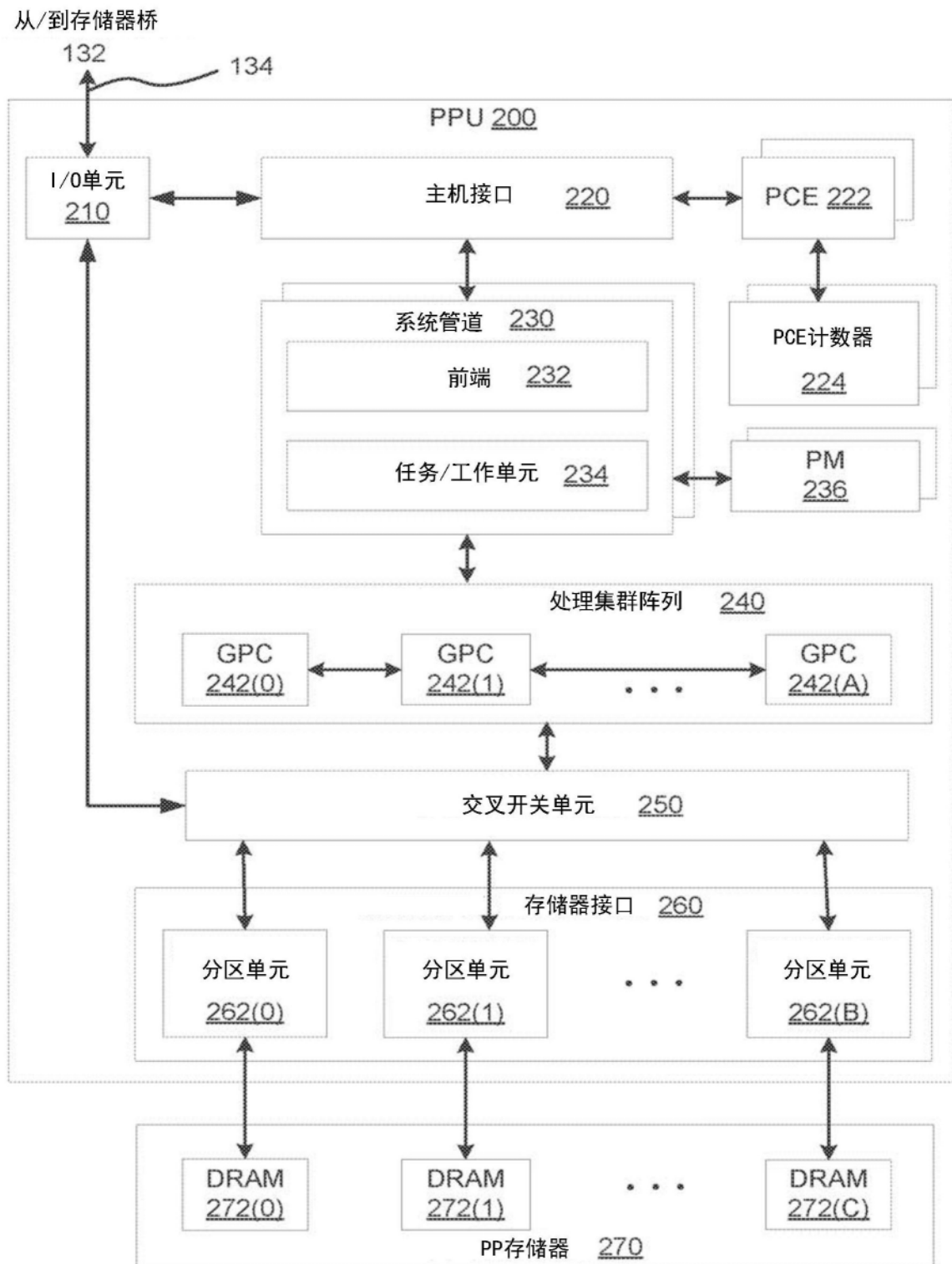


图2

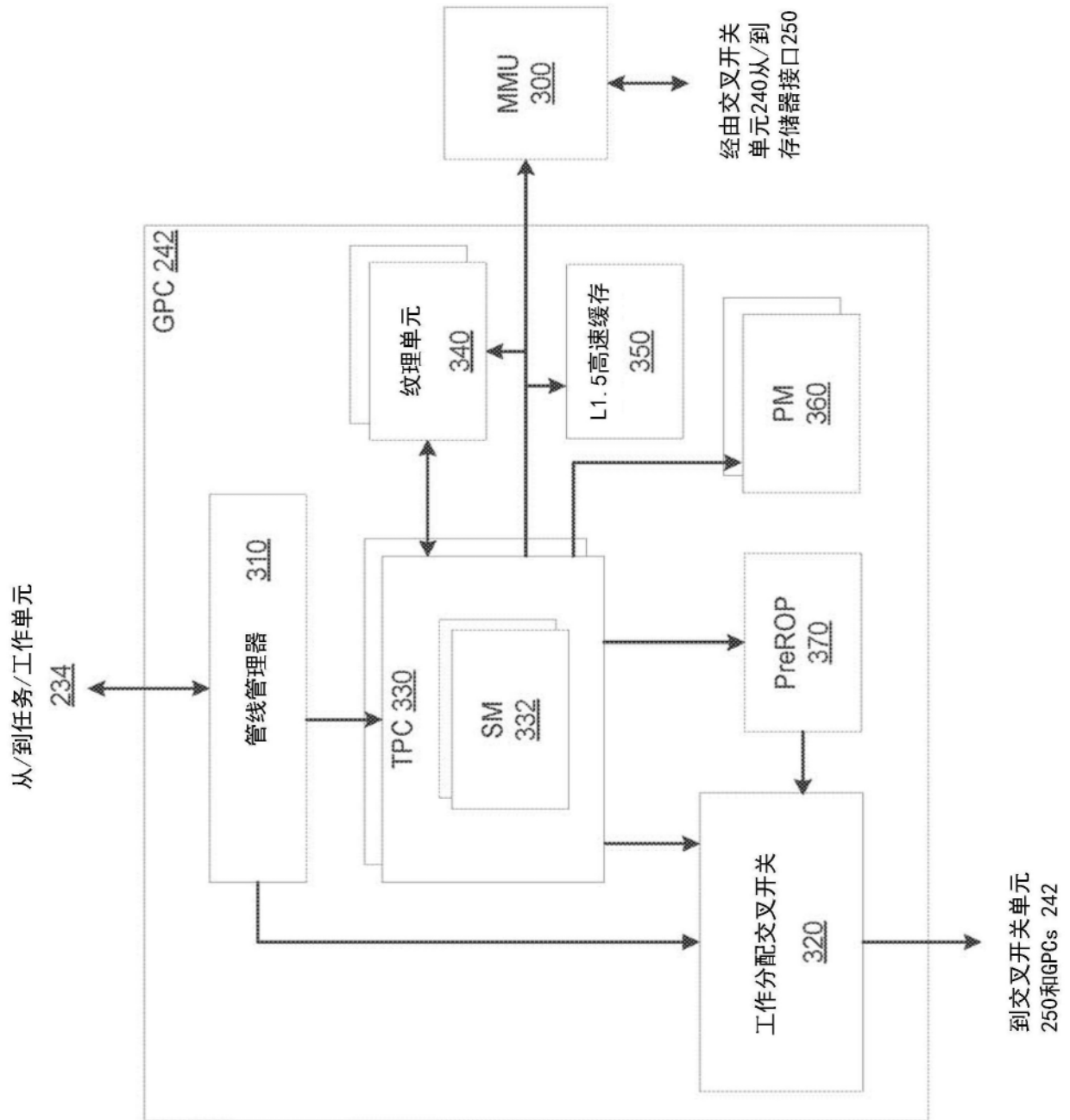


图3

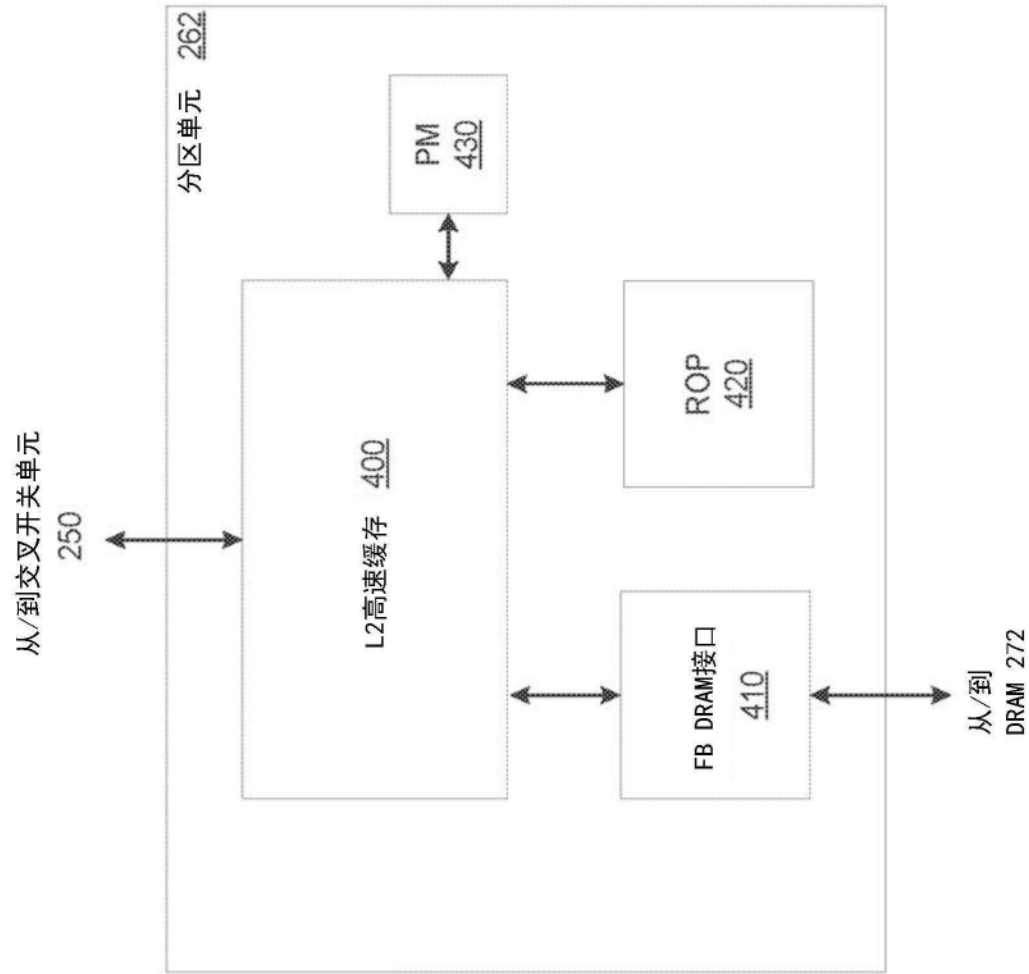


图4

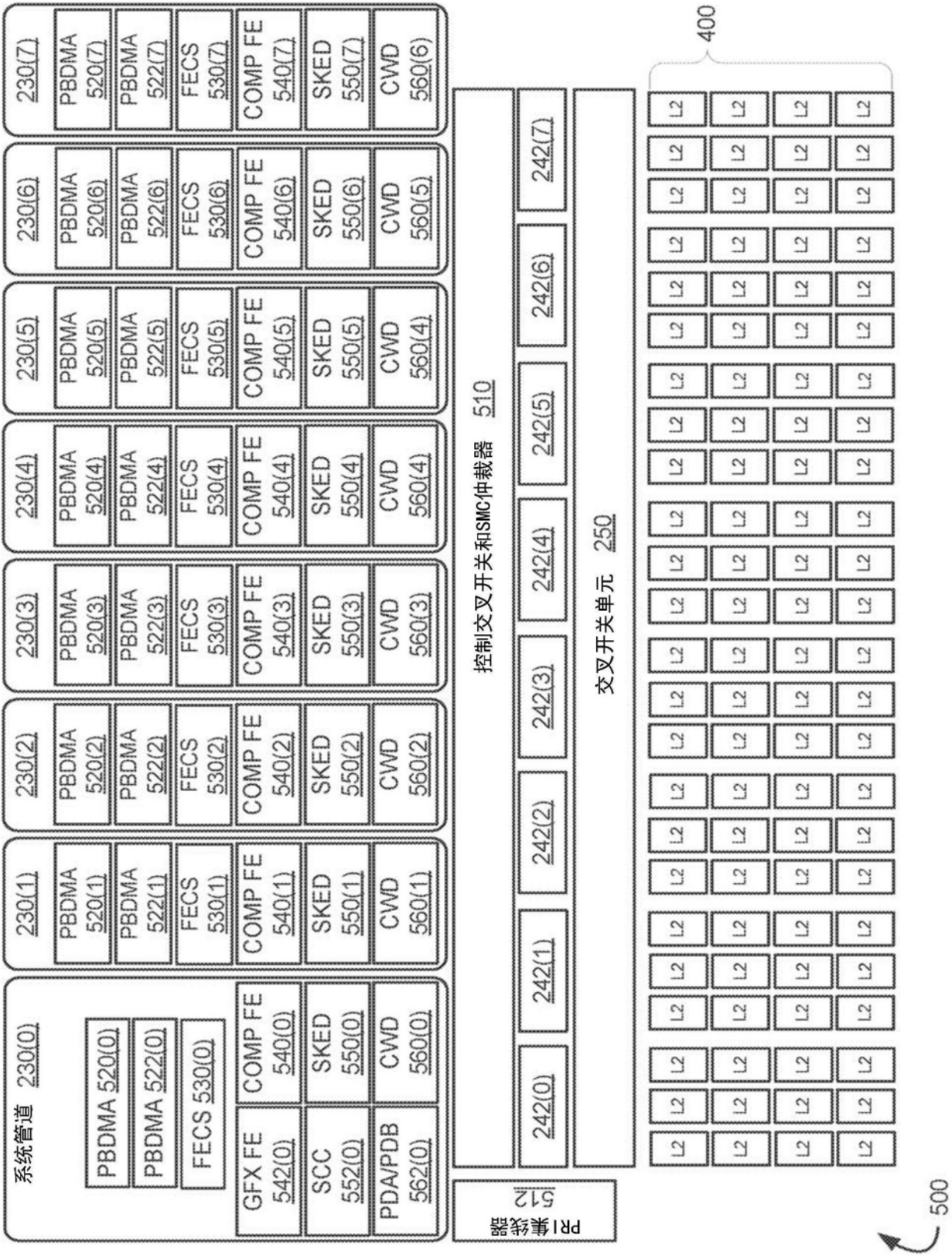


图5

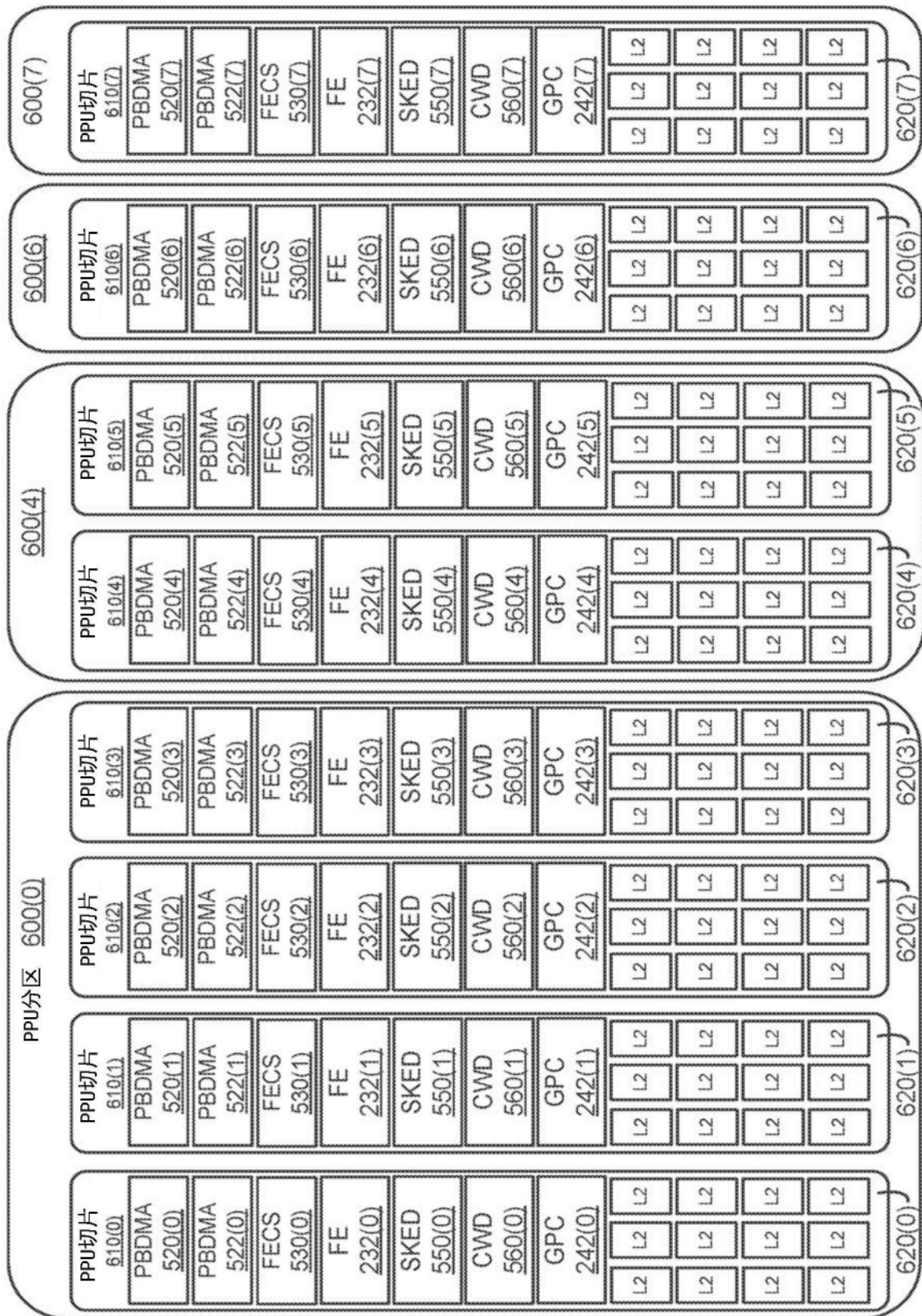


图6

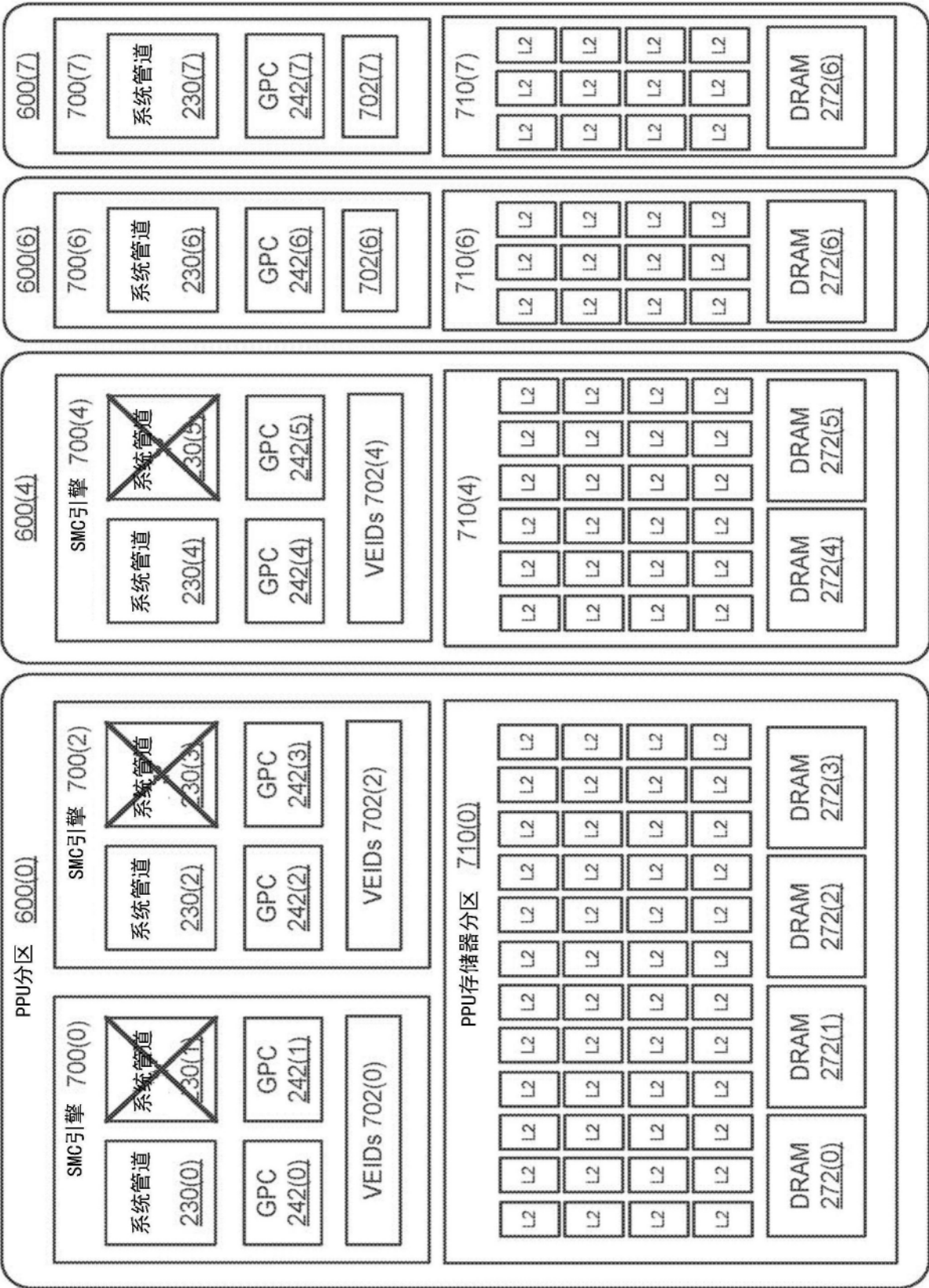


图7

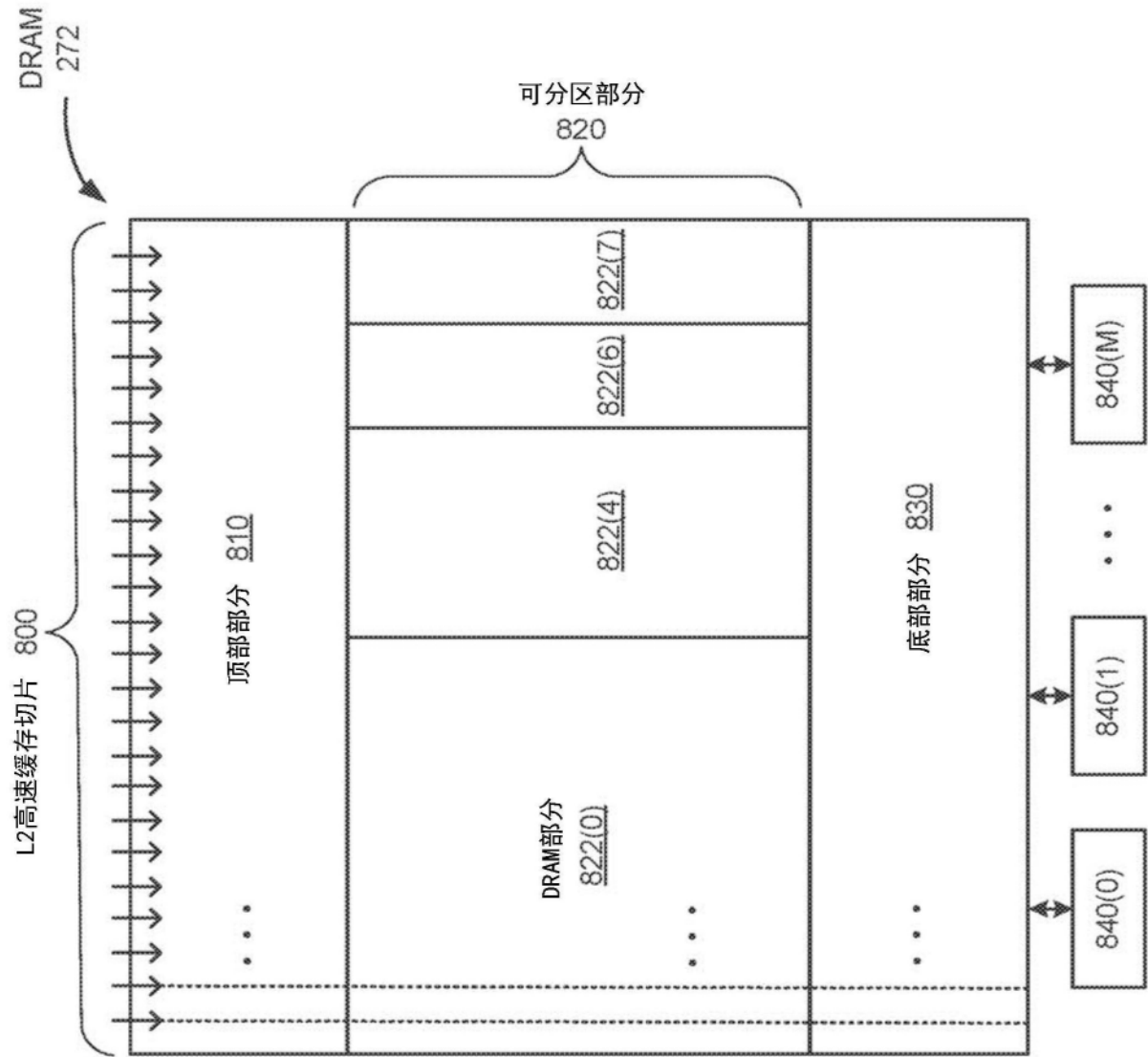


图8A

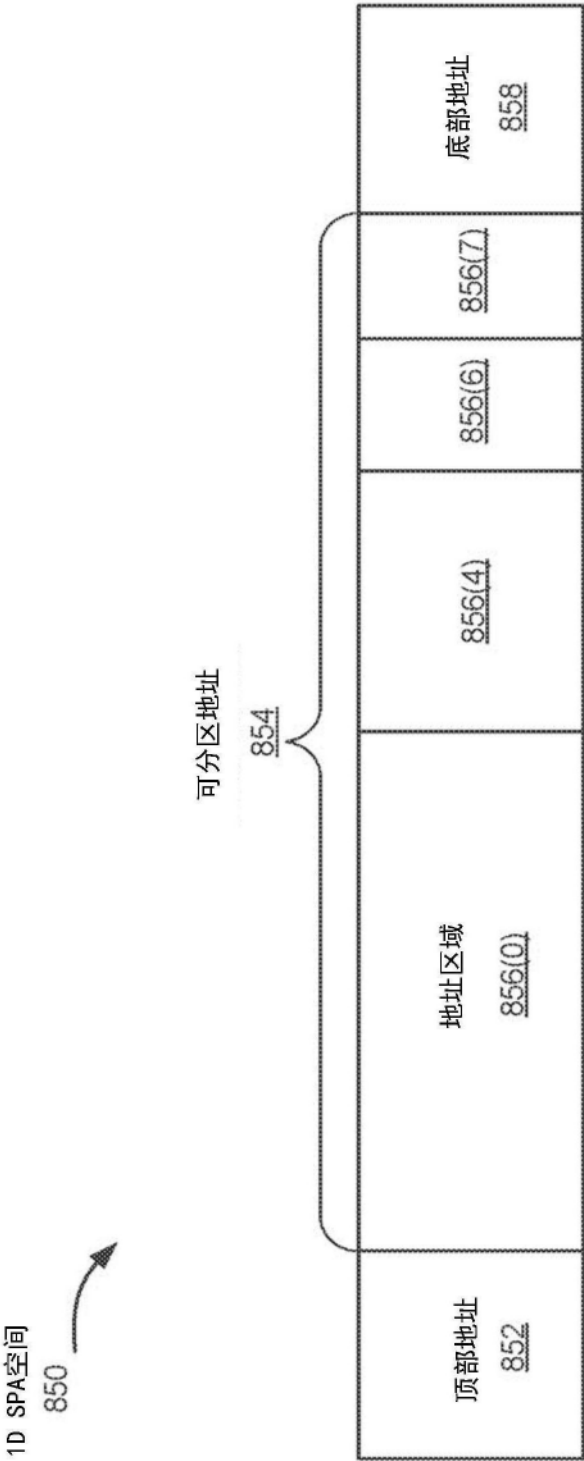


图8B

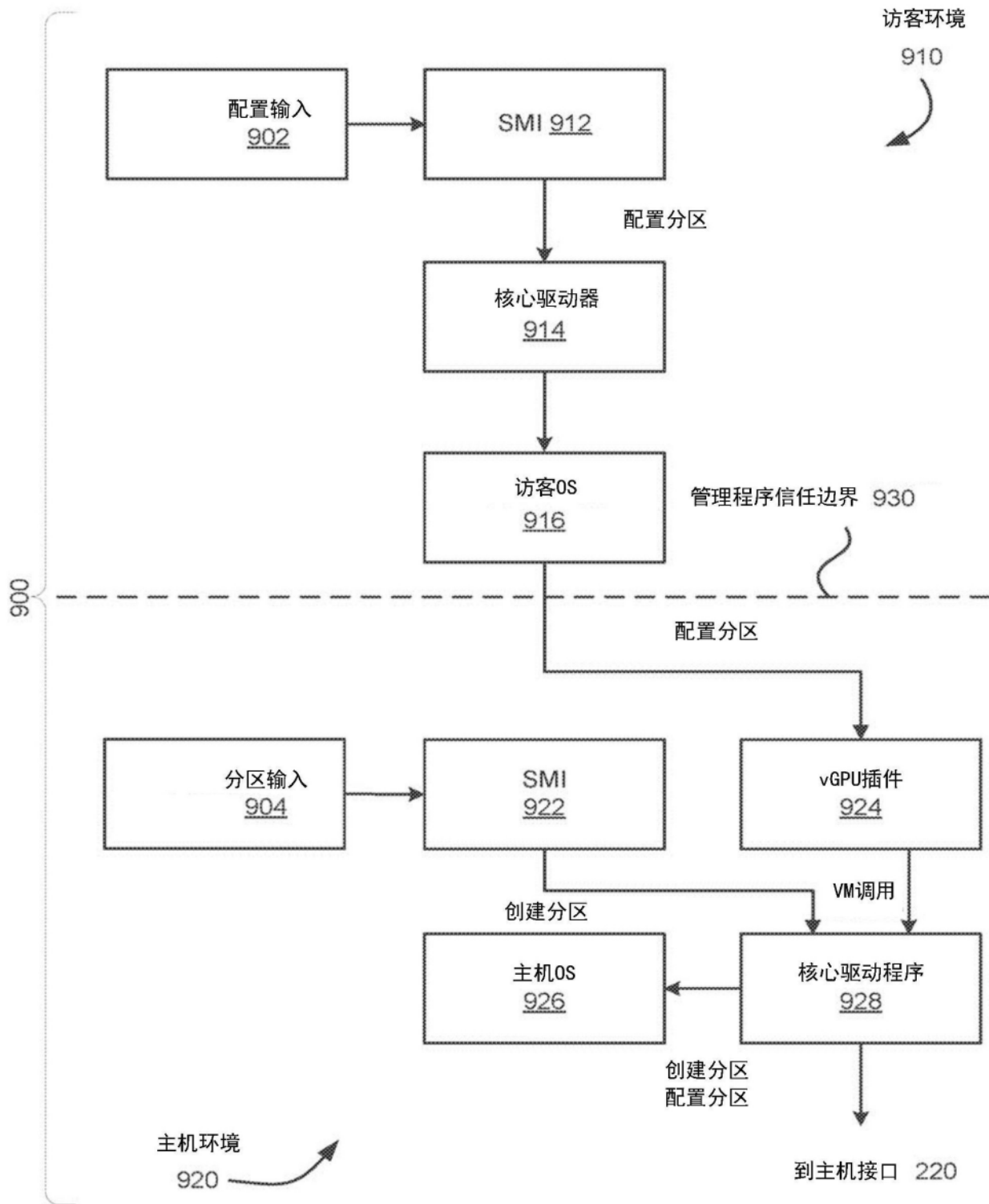


图9

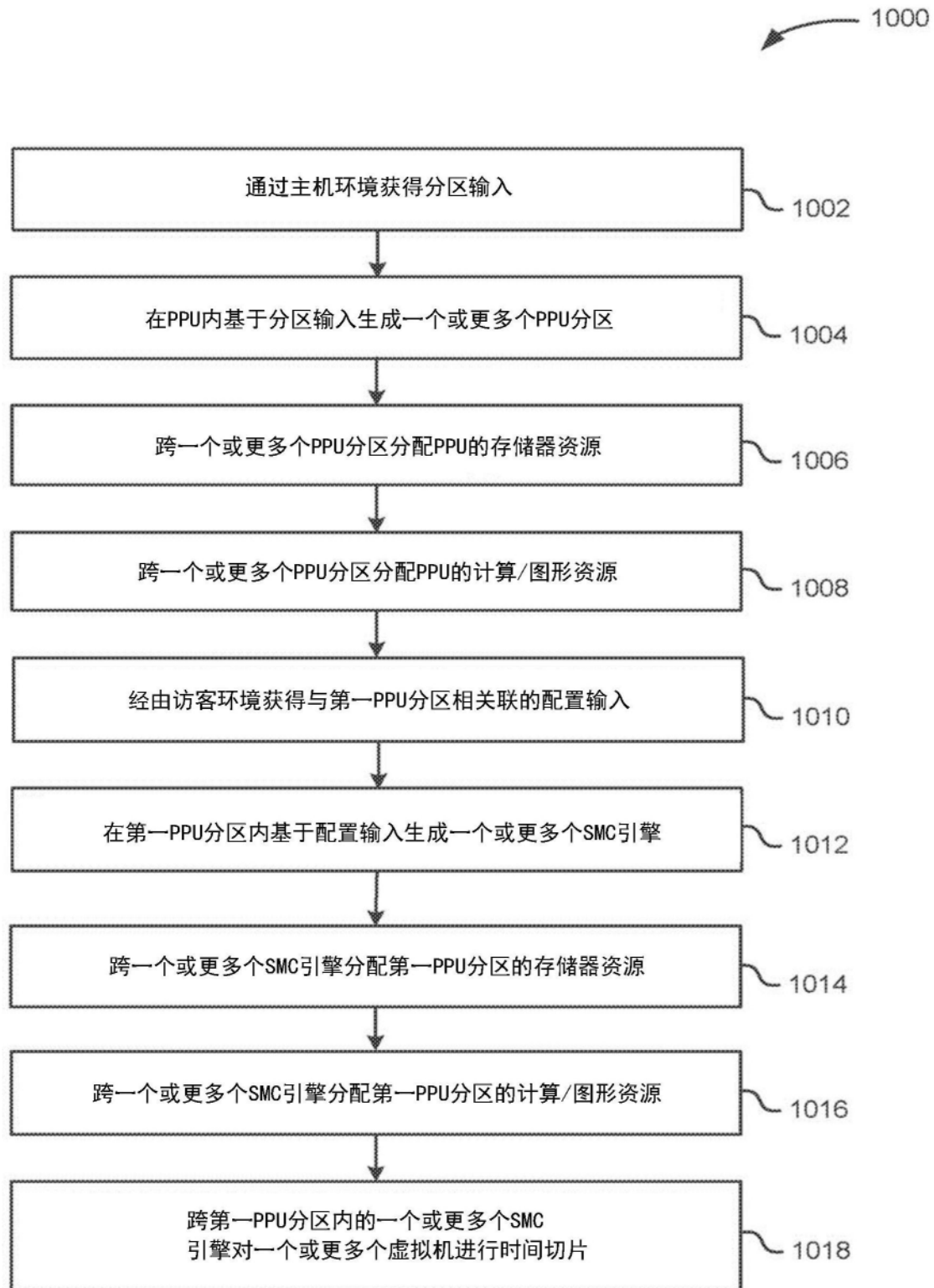


图10

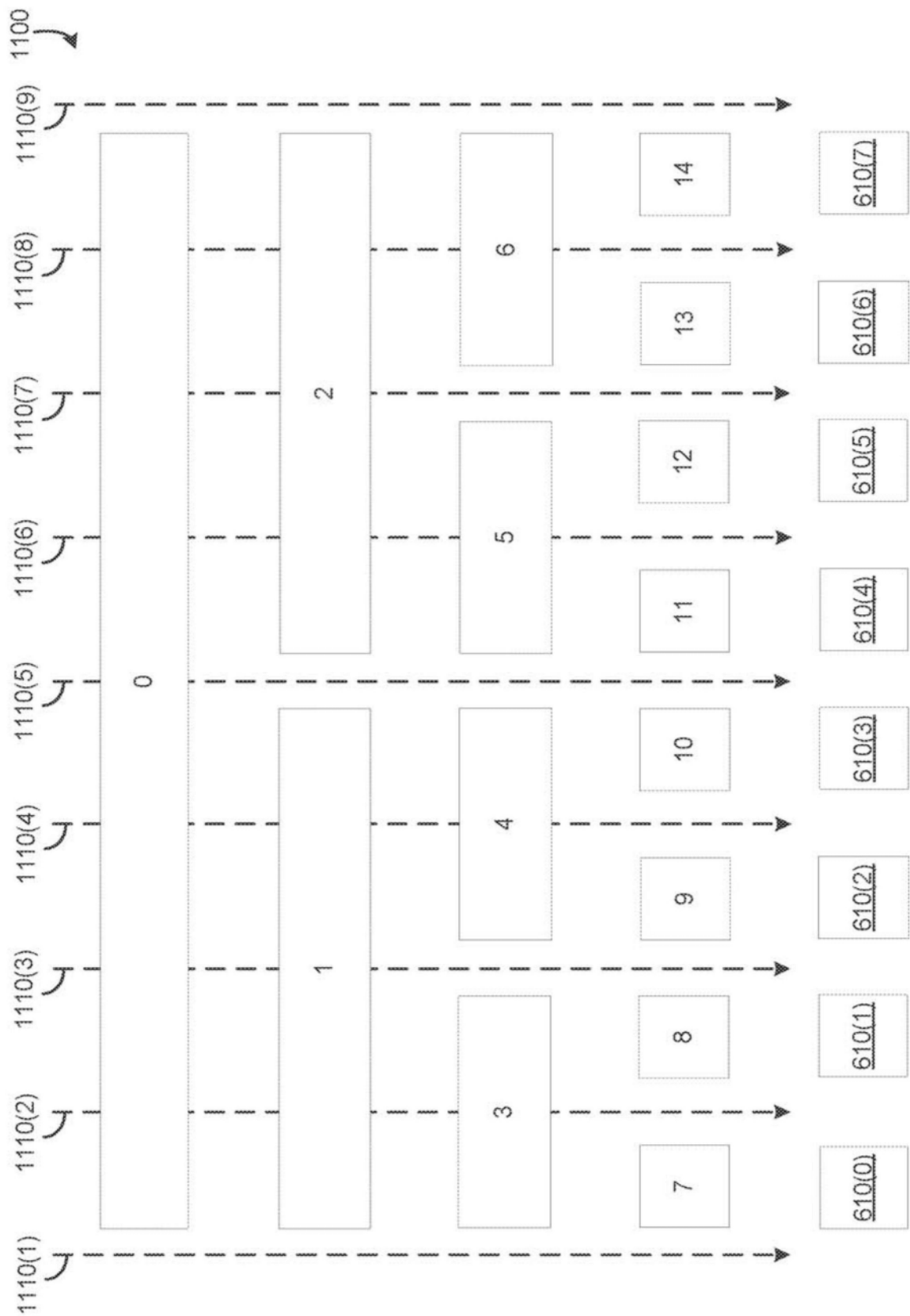


图11

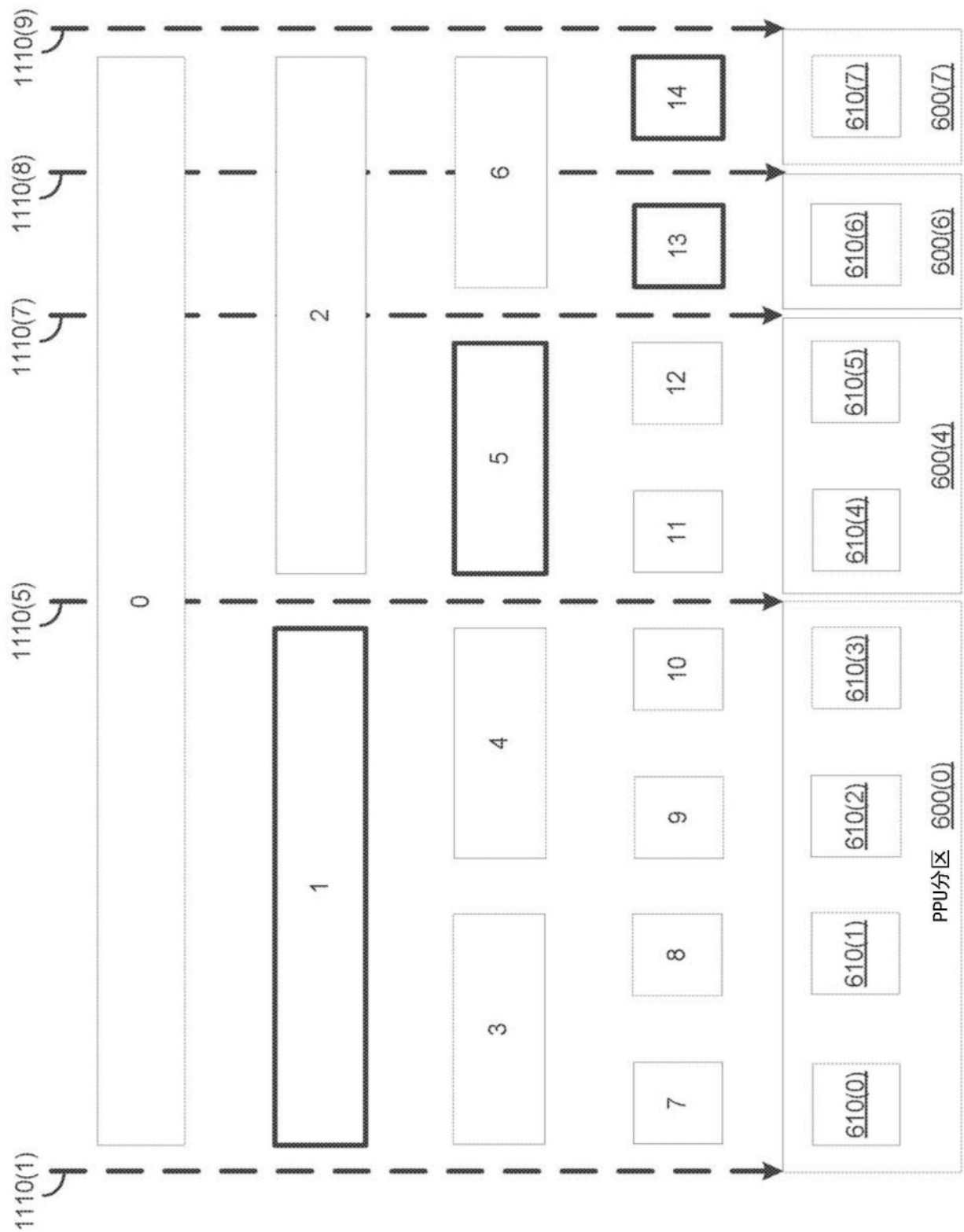


图12

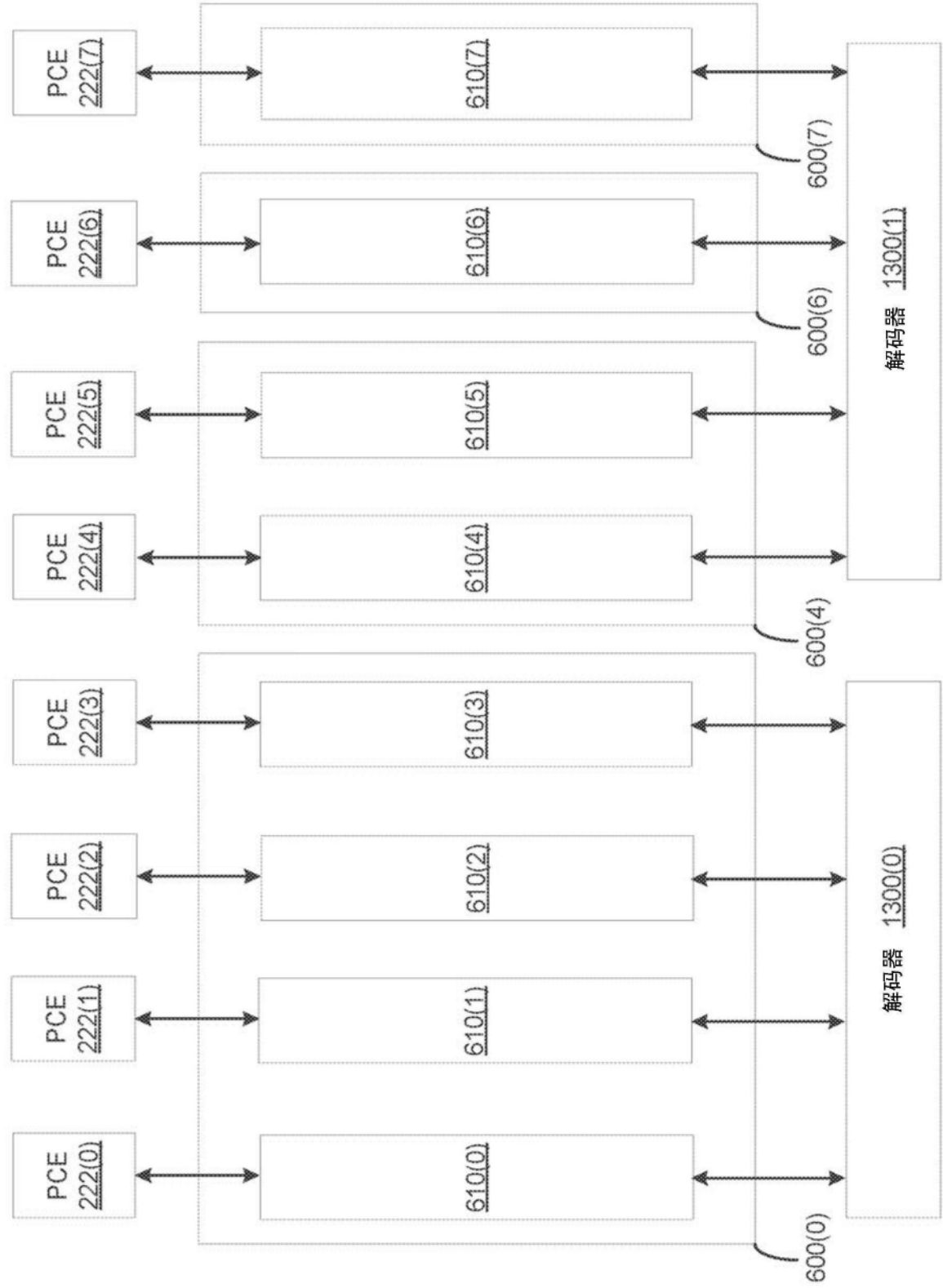


图13

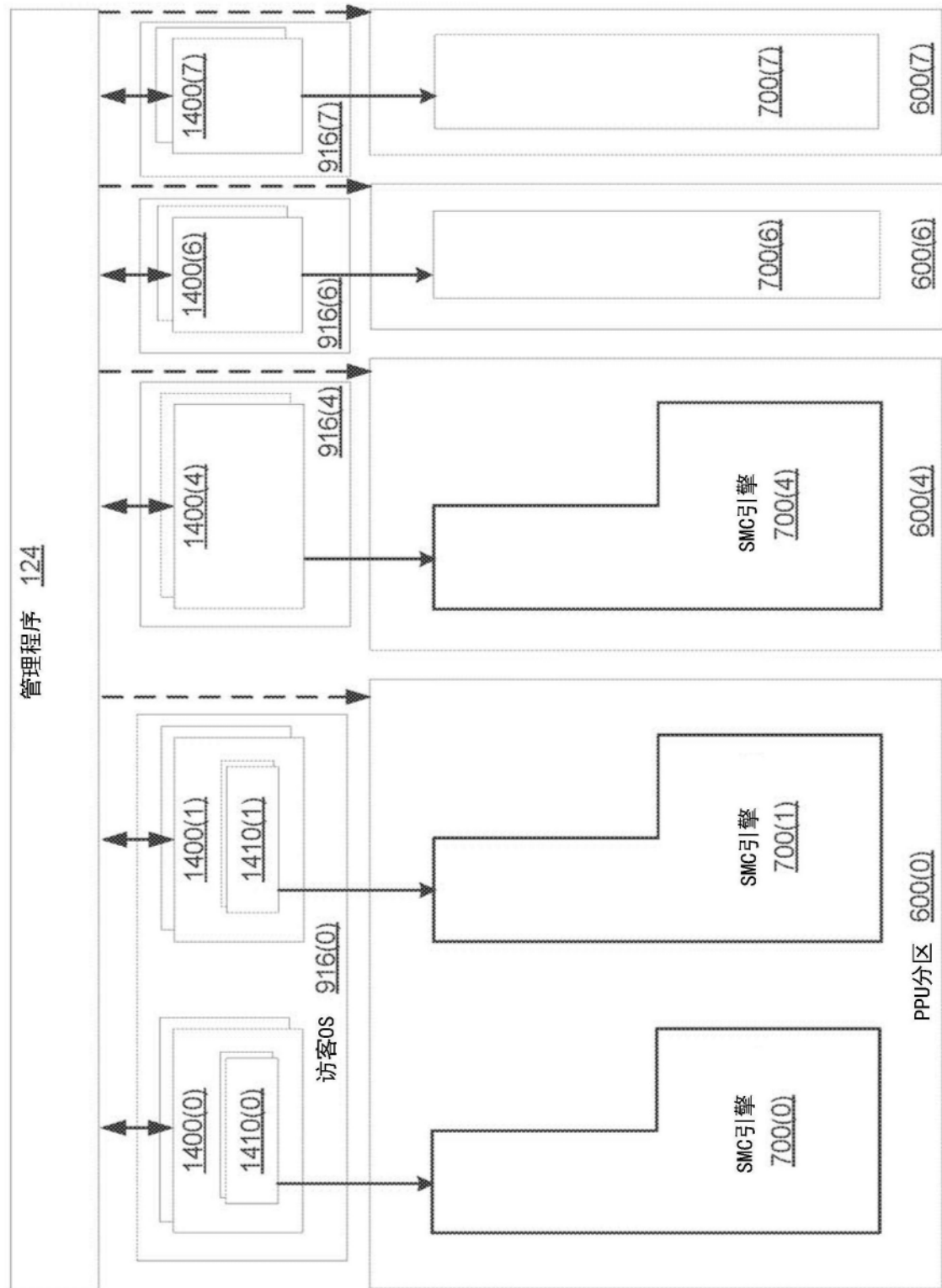


图14A

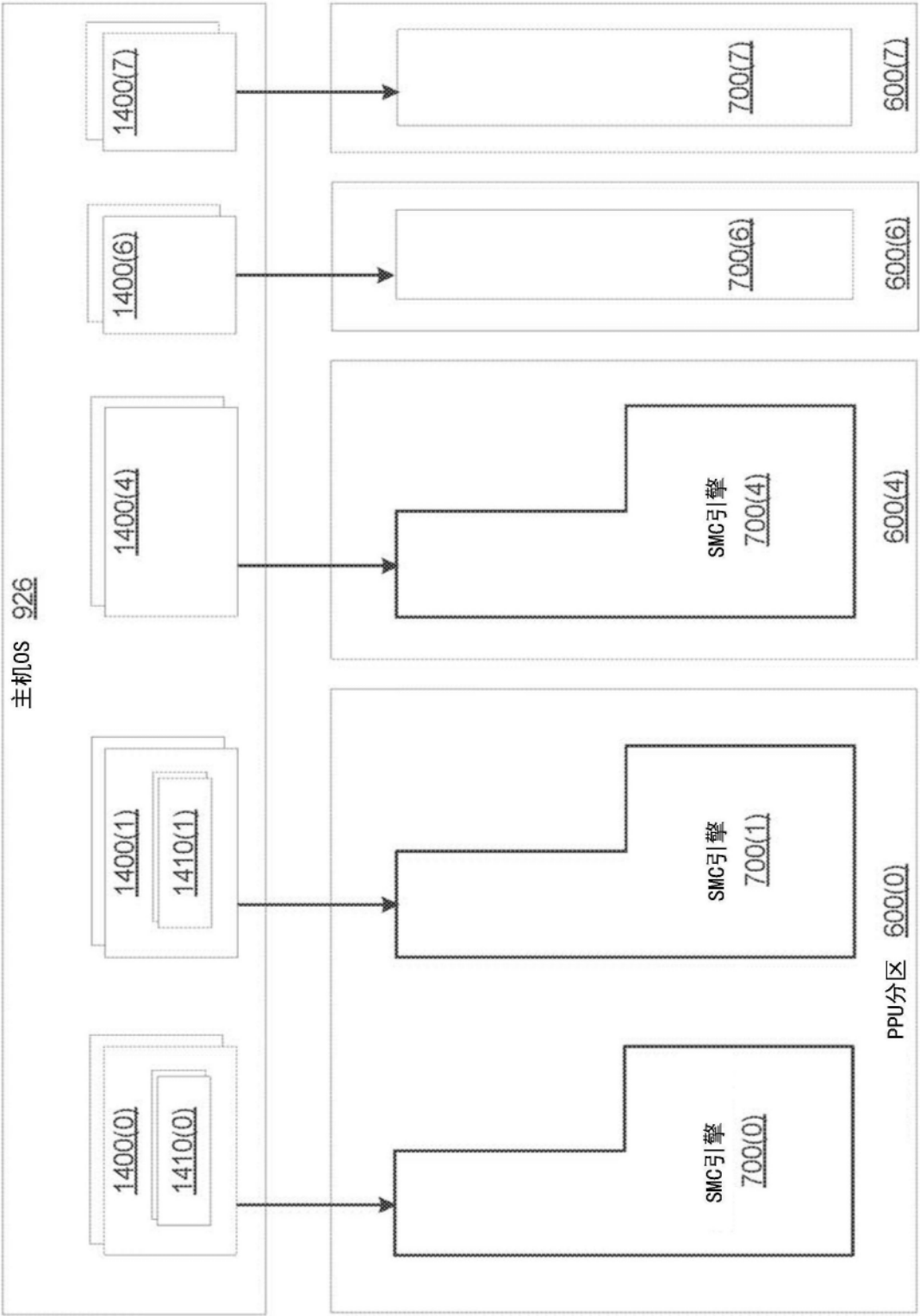


图14B



图15

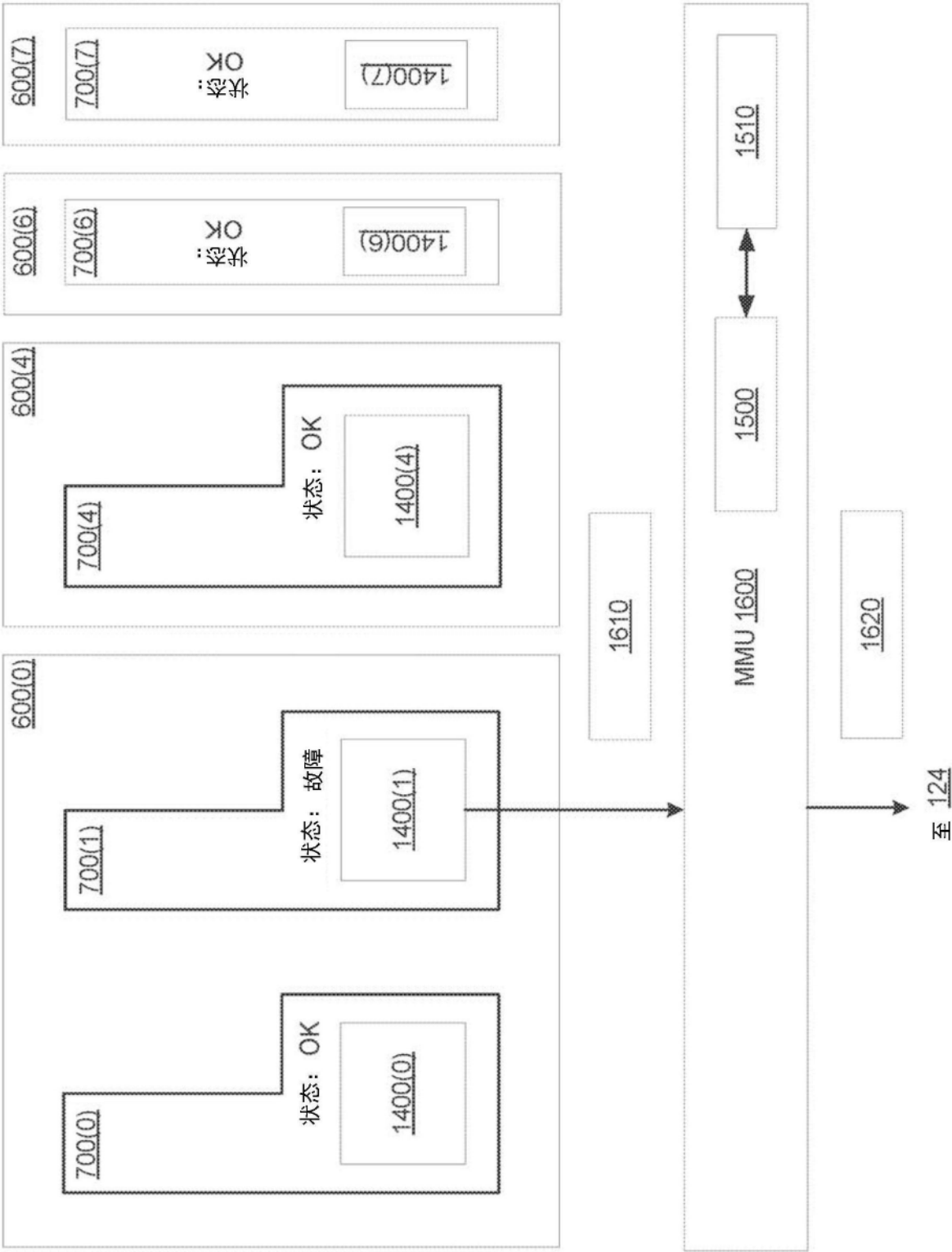


图16

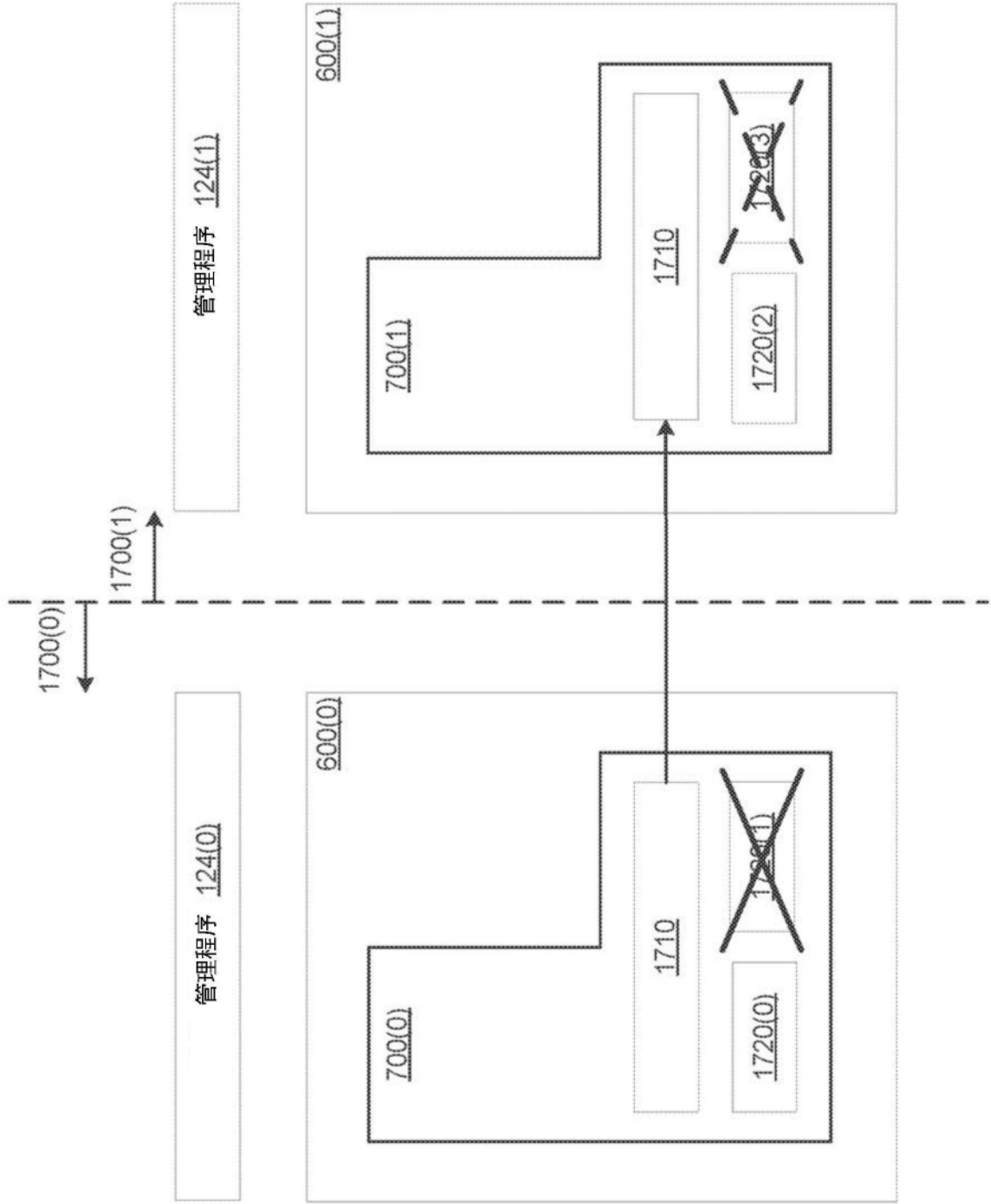


图17

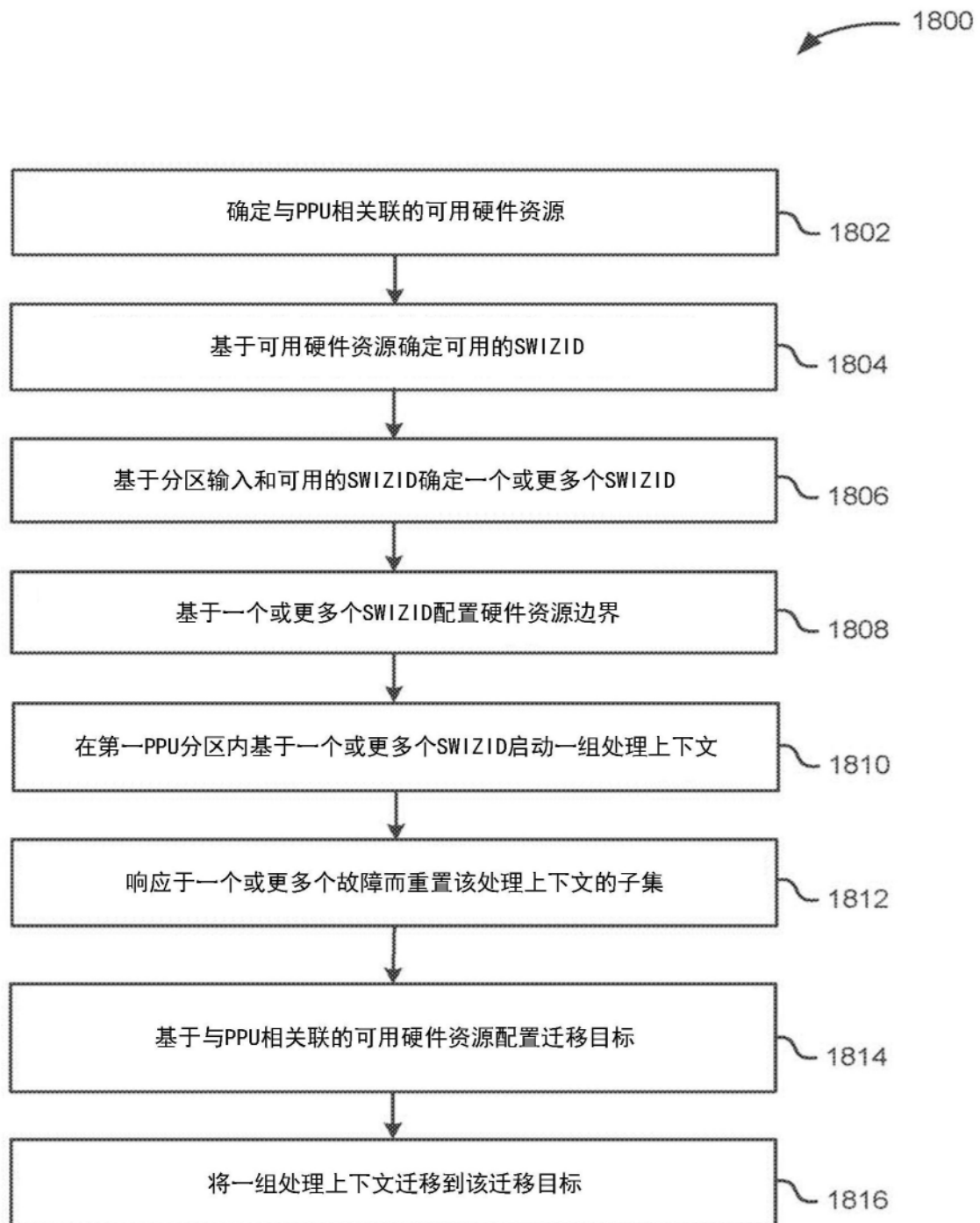


图18

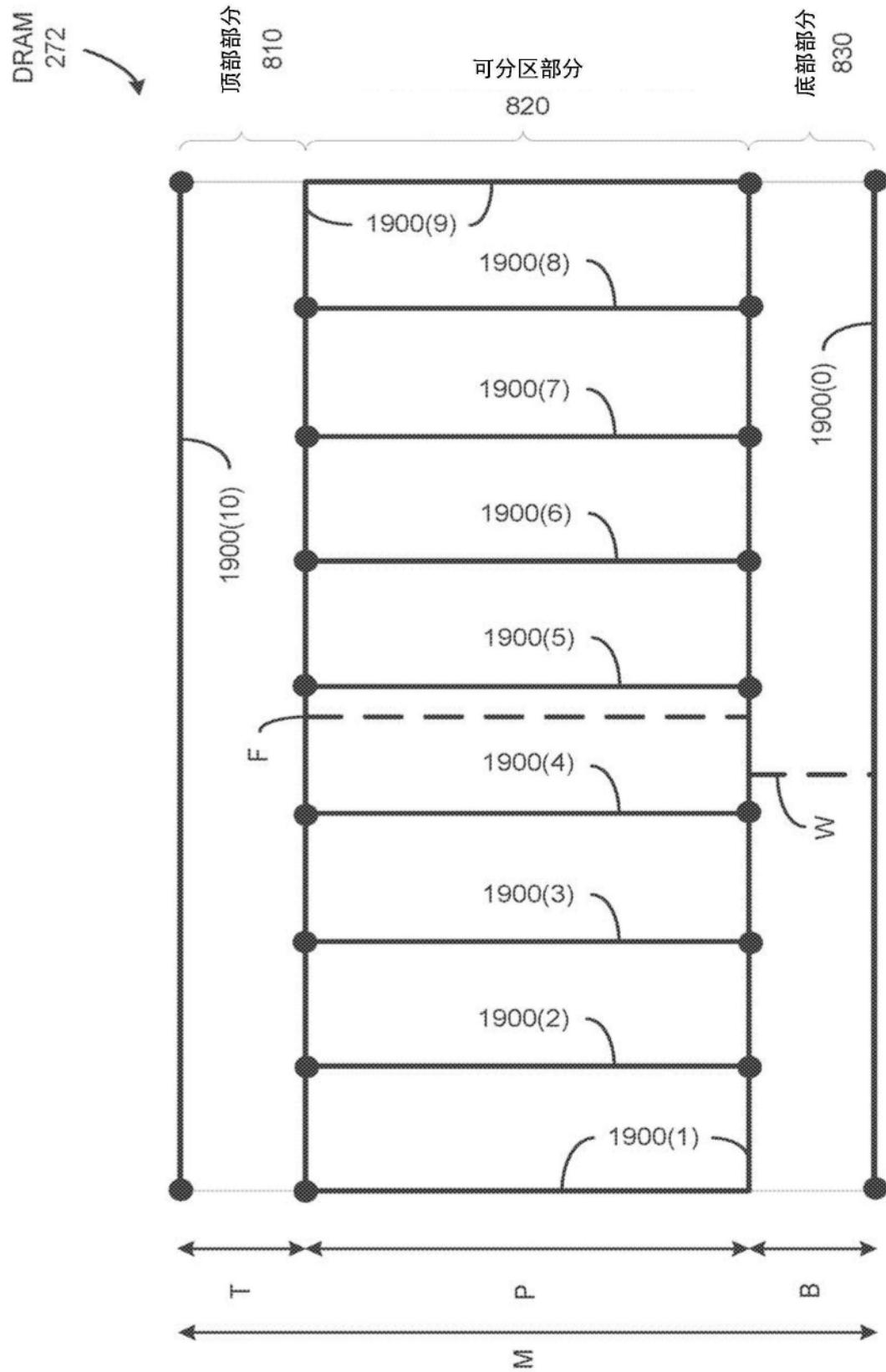


图19

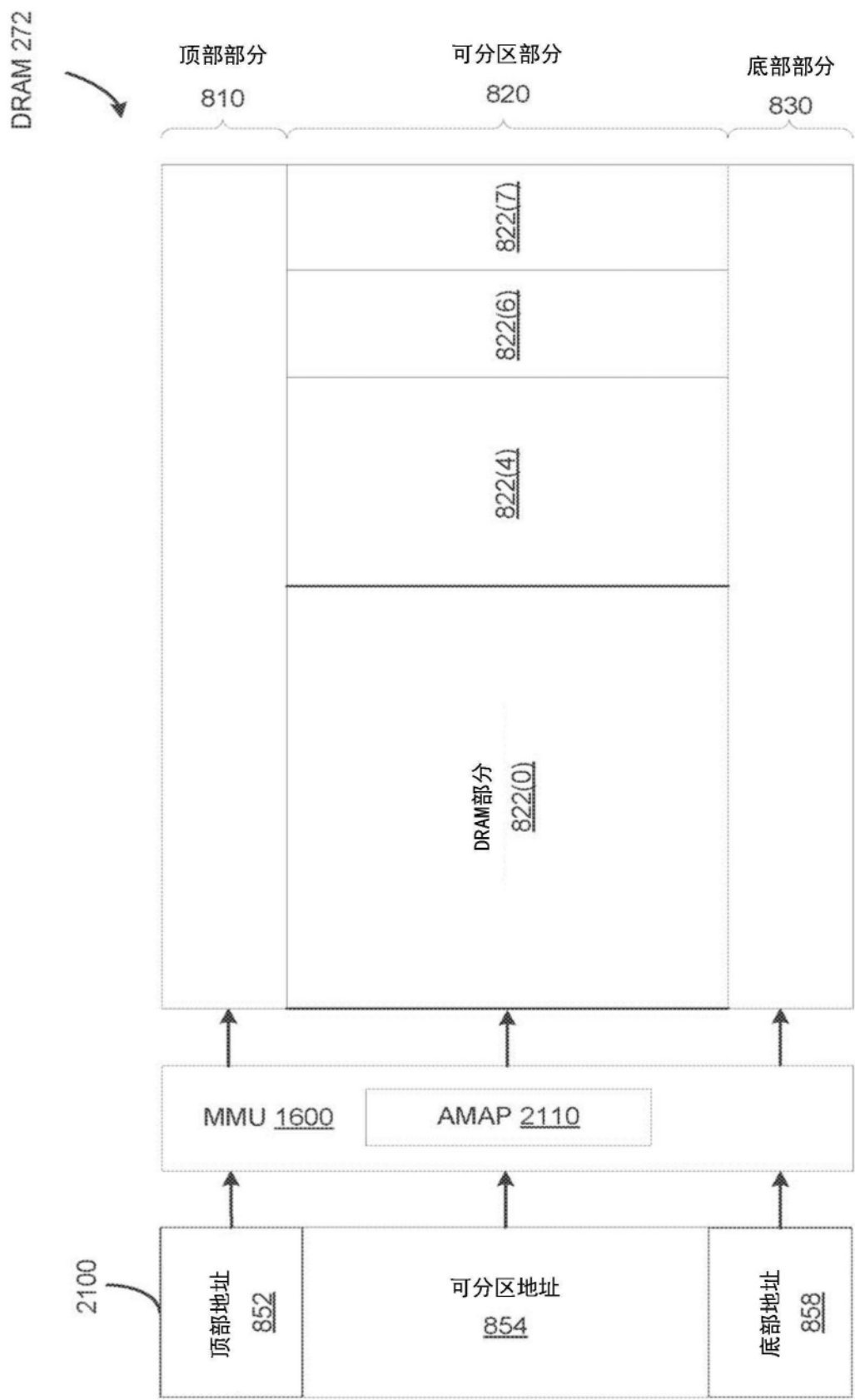


图21

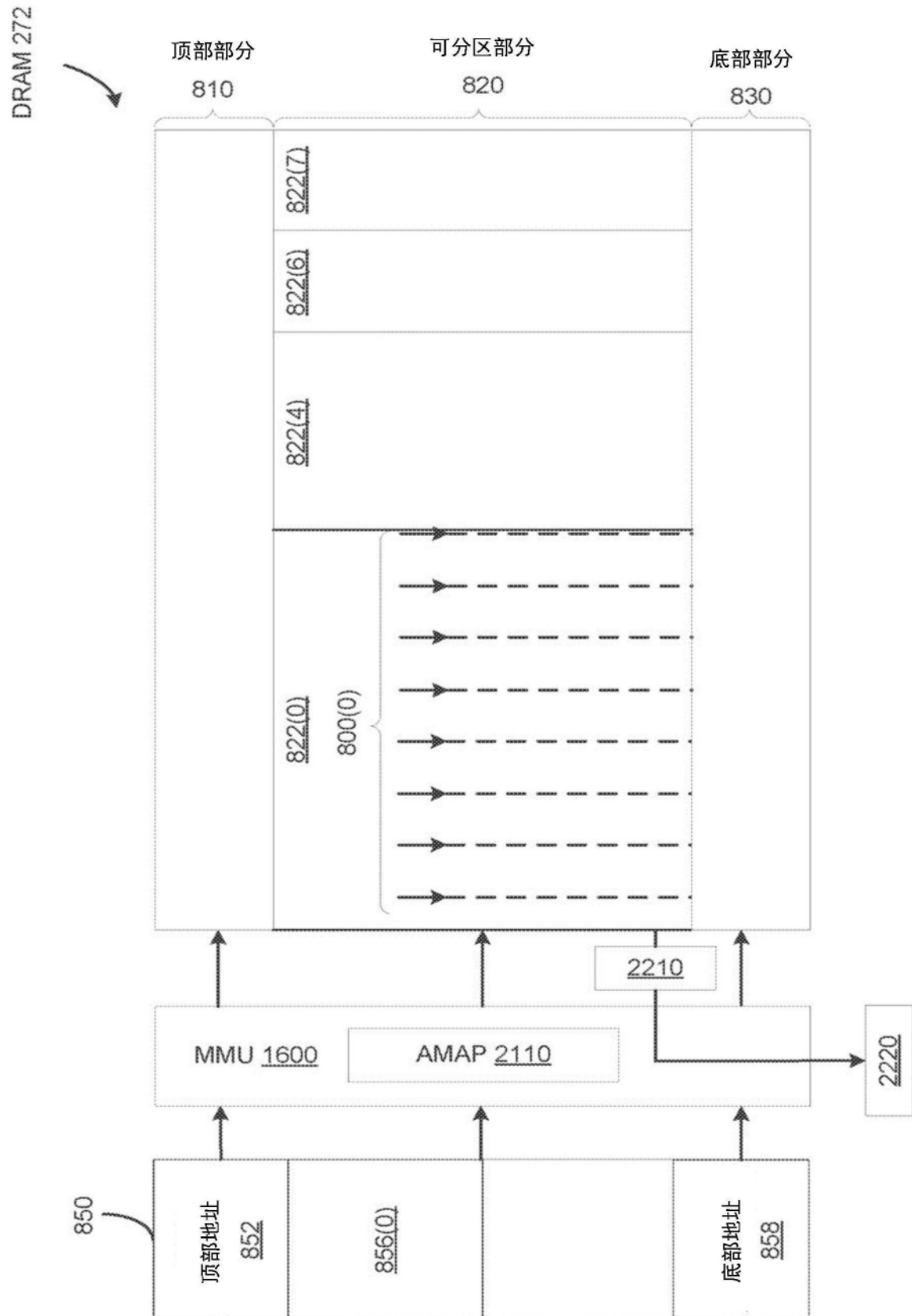


图22

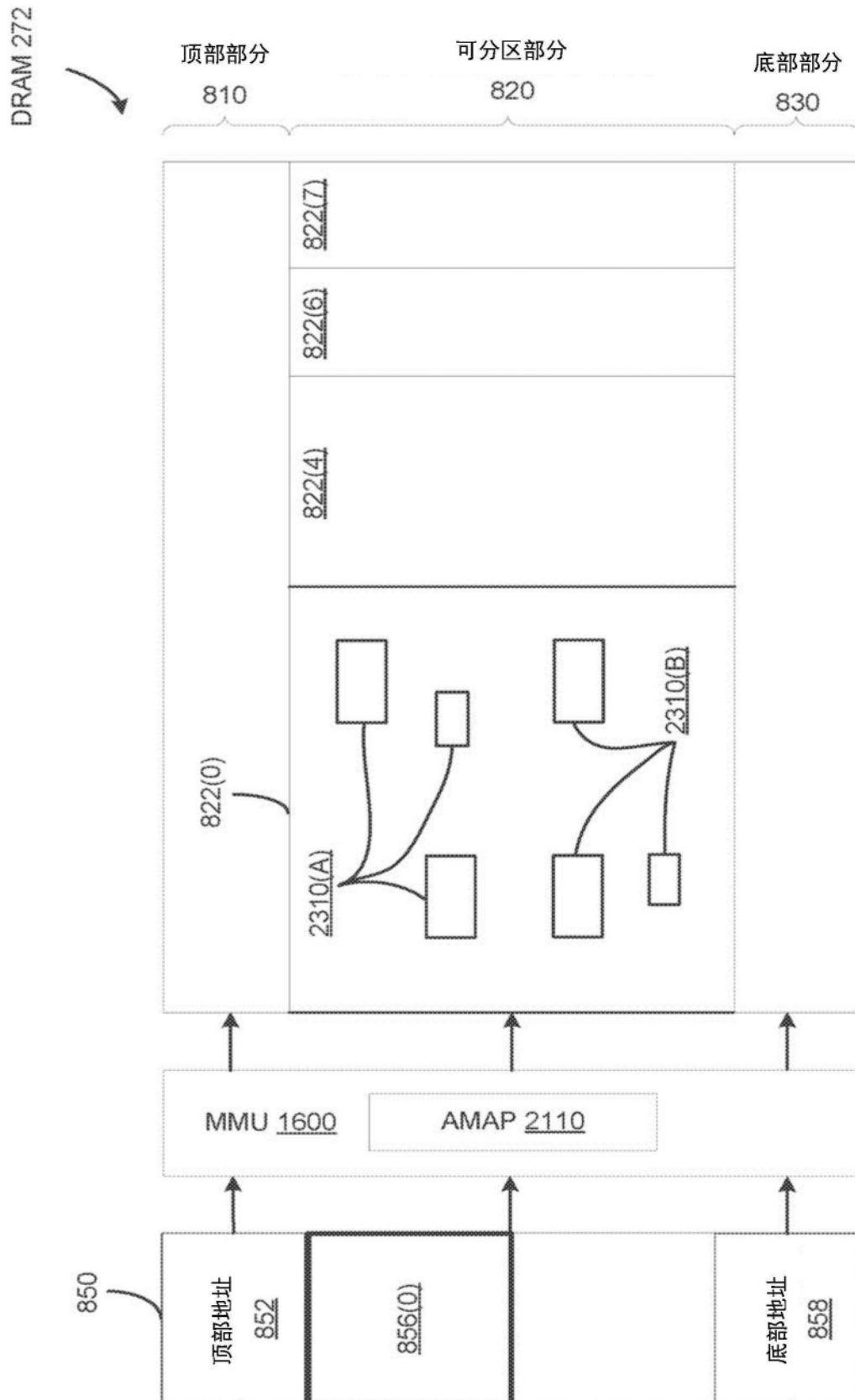


图23

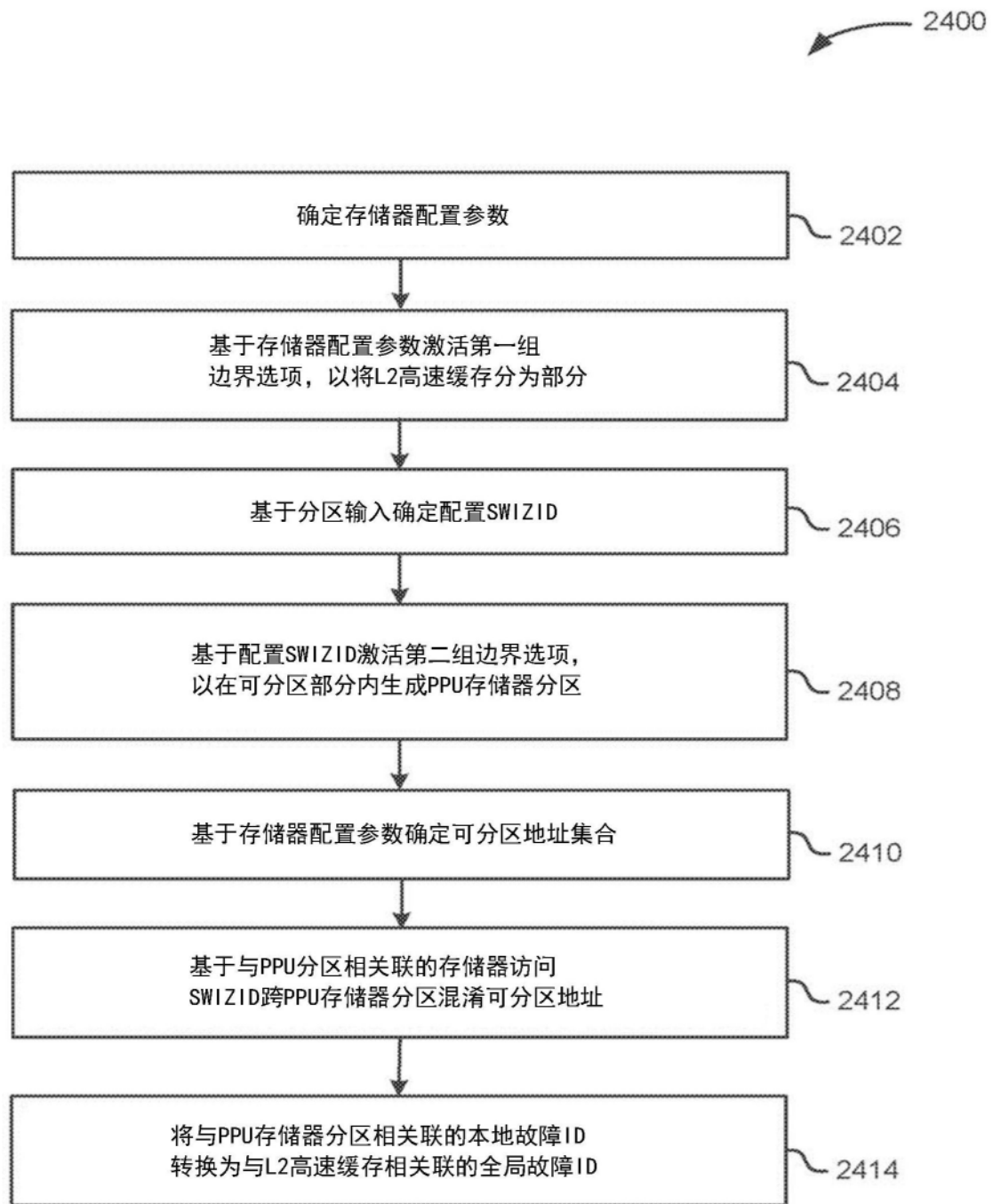


图24

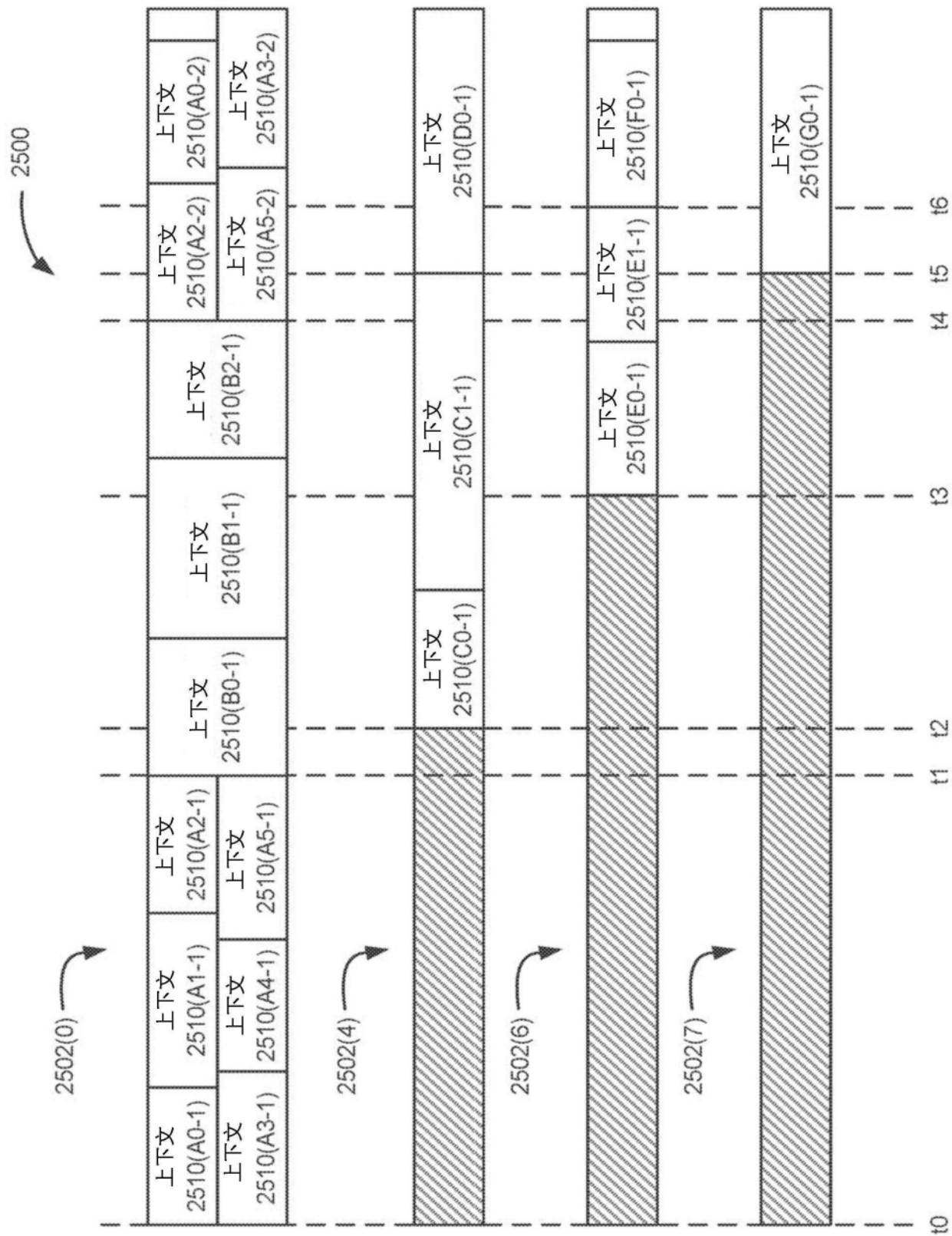


图25

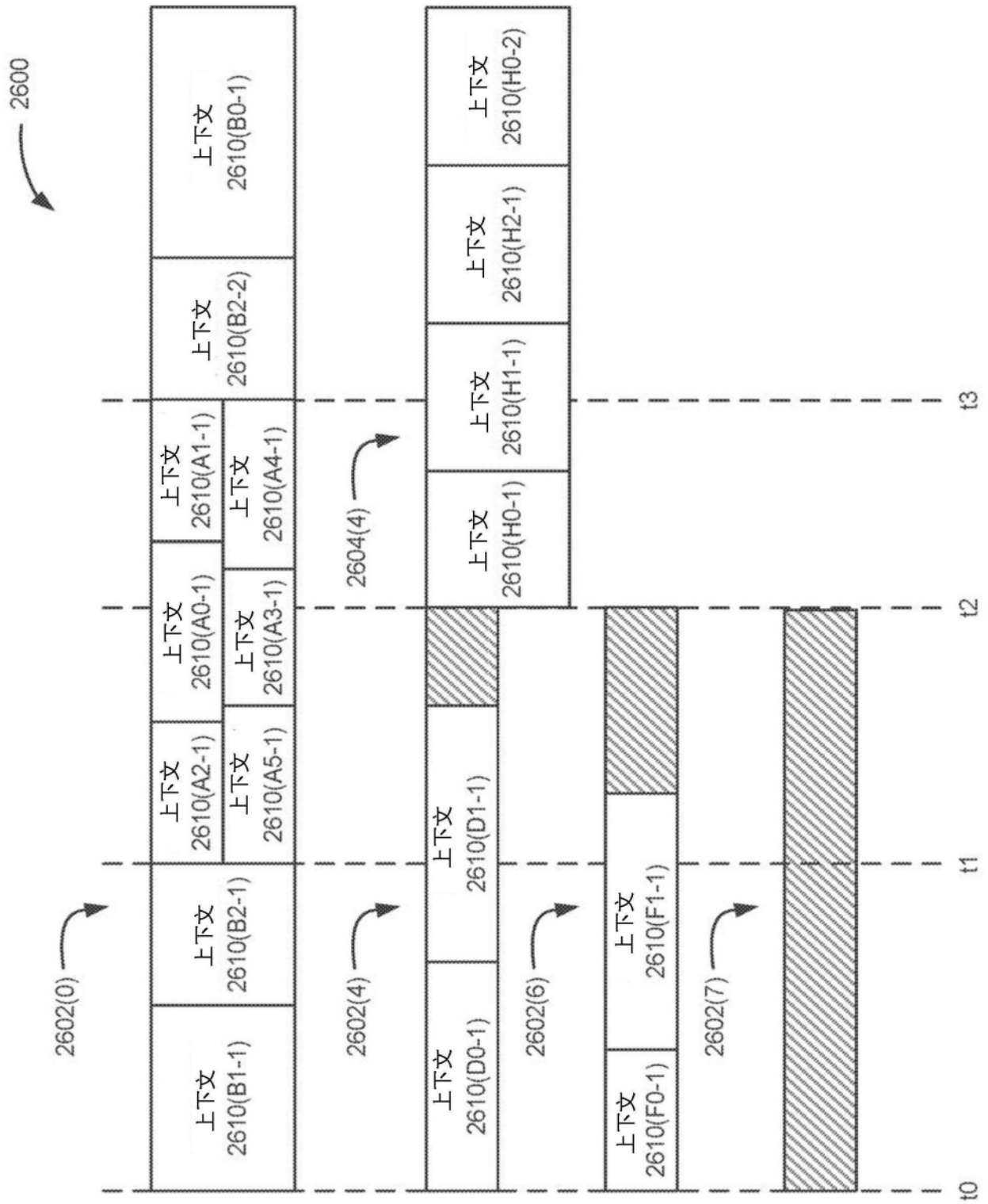


图26

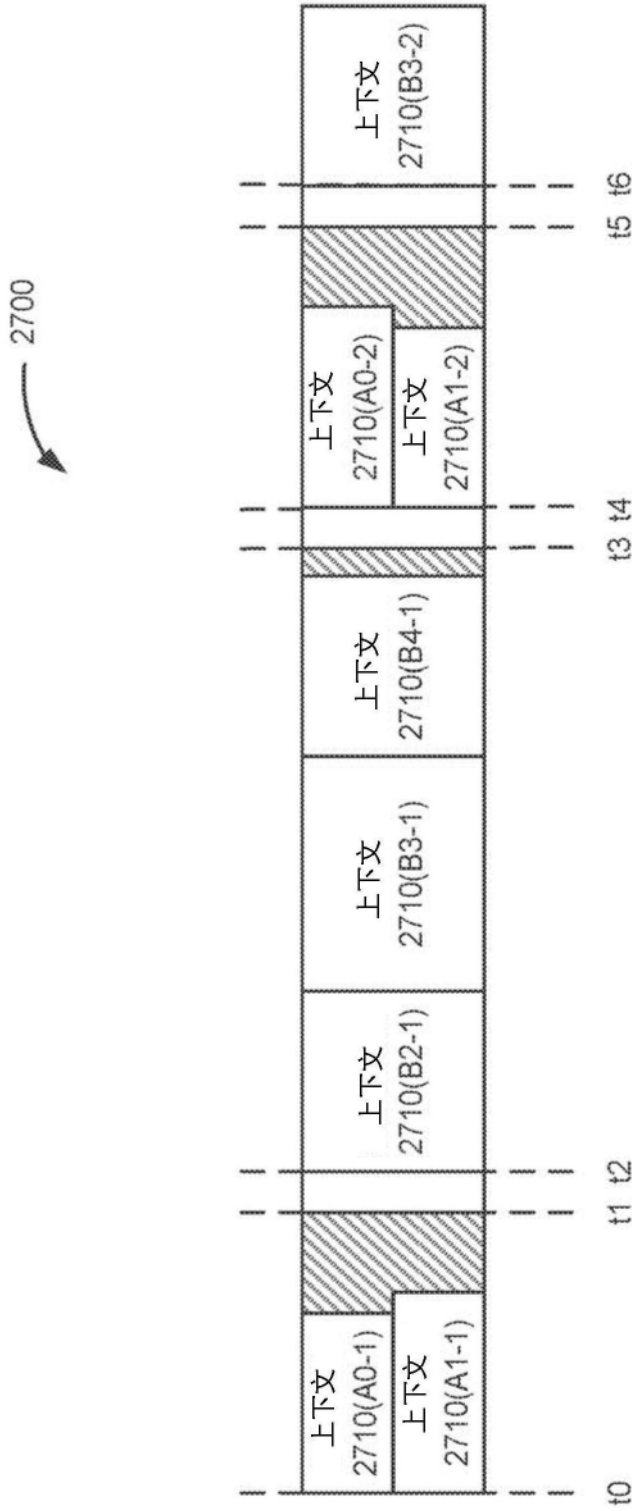


图27

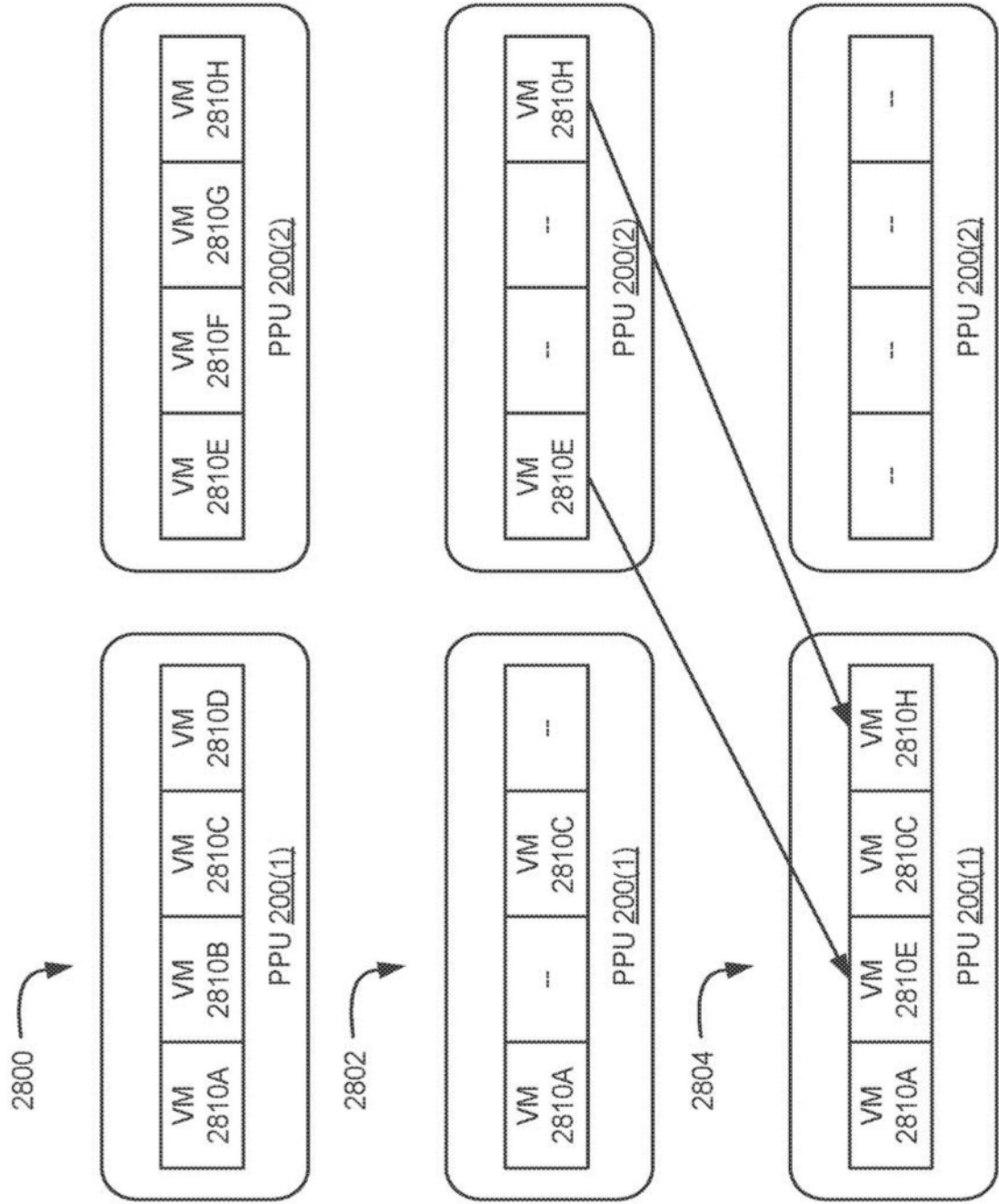


图28

2900

2902(0)

	上下文 2910(C0)	上下文 2910(D1)	上下文 2910(E2)	上下文 2910(A3)	上下文 2910(B3)	上下文 2910(C4)	上下文 2910(D5)
--	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

2902(1)

	上下文 2910(D0)	上下文 2910(E1)	上下文 2910(A2)	上下文 2910(B2)	上下文 2910(C3)	上下文 2910(D4)	上下文 2910(E5)
--	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

2902(2)

	上下文 2910(E0)	上下文 2910(A1)	上下文 2910(B1)	上下文 2910(C2)	上下文 2910(D3)	上下文 2910(E4)	上下文 2910(A5)
--	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

2902(3)

上下文 2910(A0)	上下文 2910(B0)	上下文 2910(C1)	上下文 2910(D2)	上下文 2910(E3)	上下文 2910(A4)	上下文 2910(B4)	上下文 2910(C5)
-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

图29

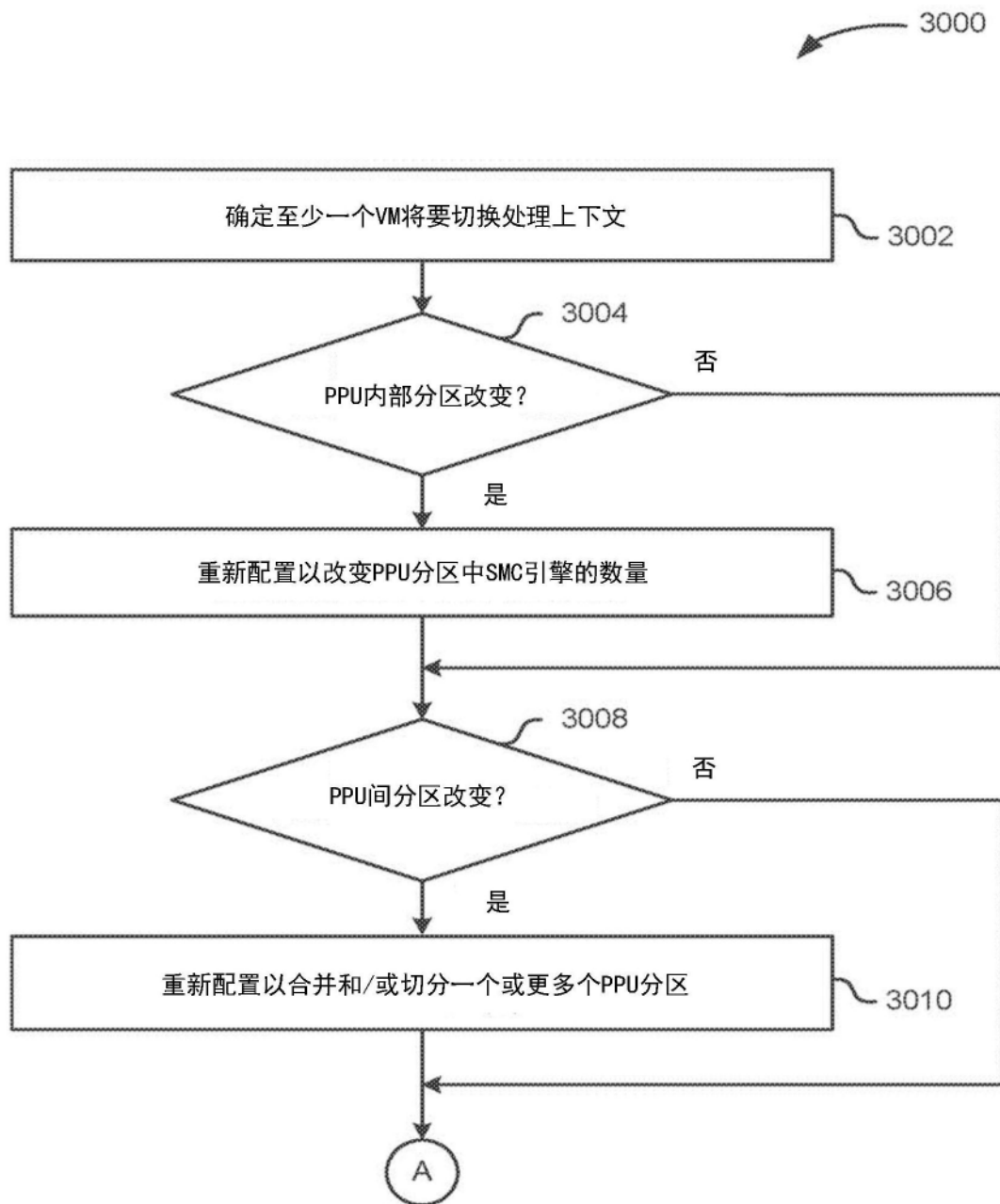


图30A

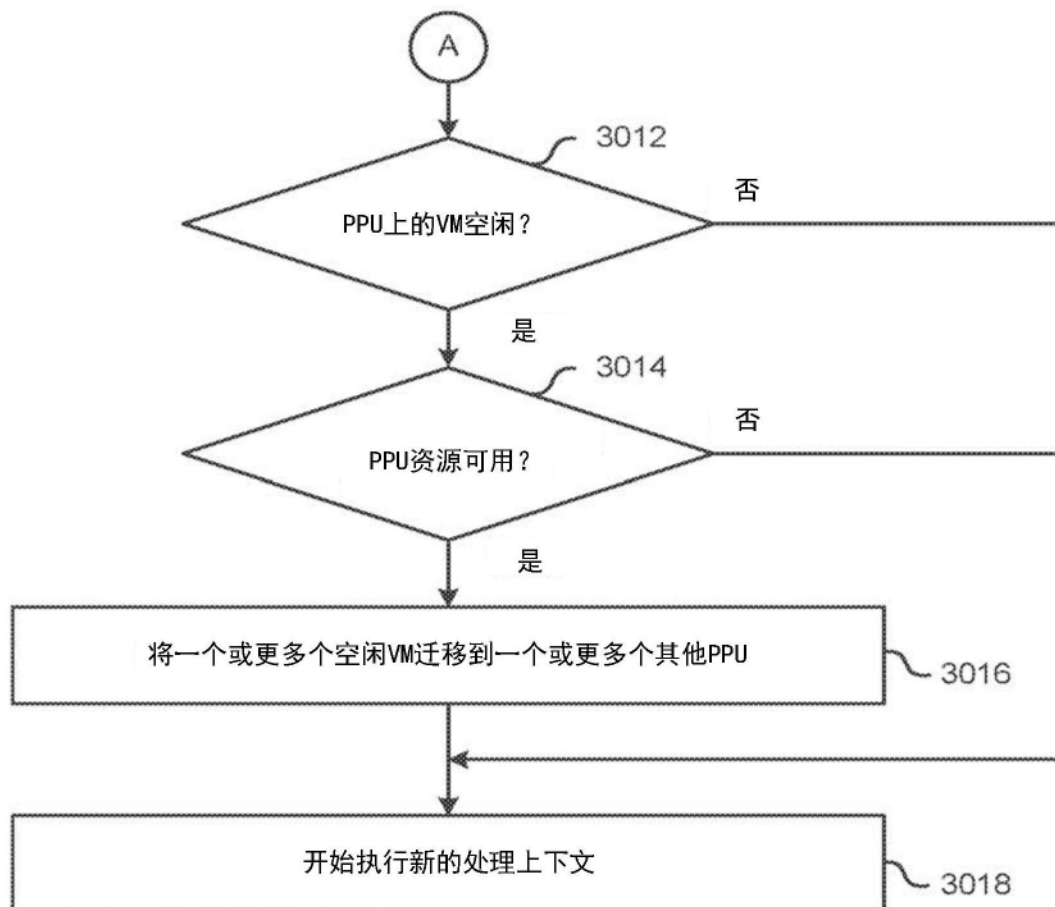


图30B

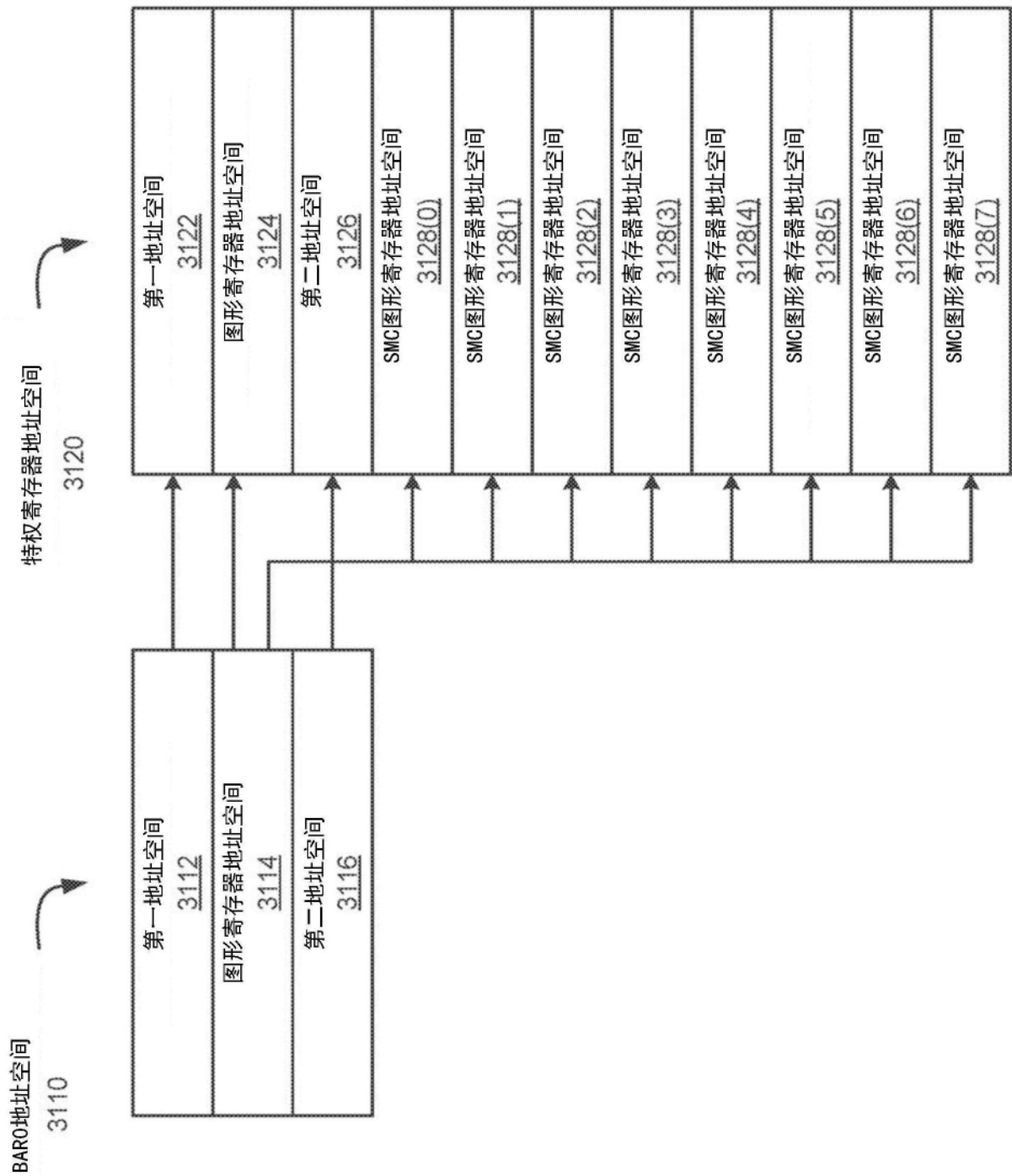


图31

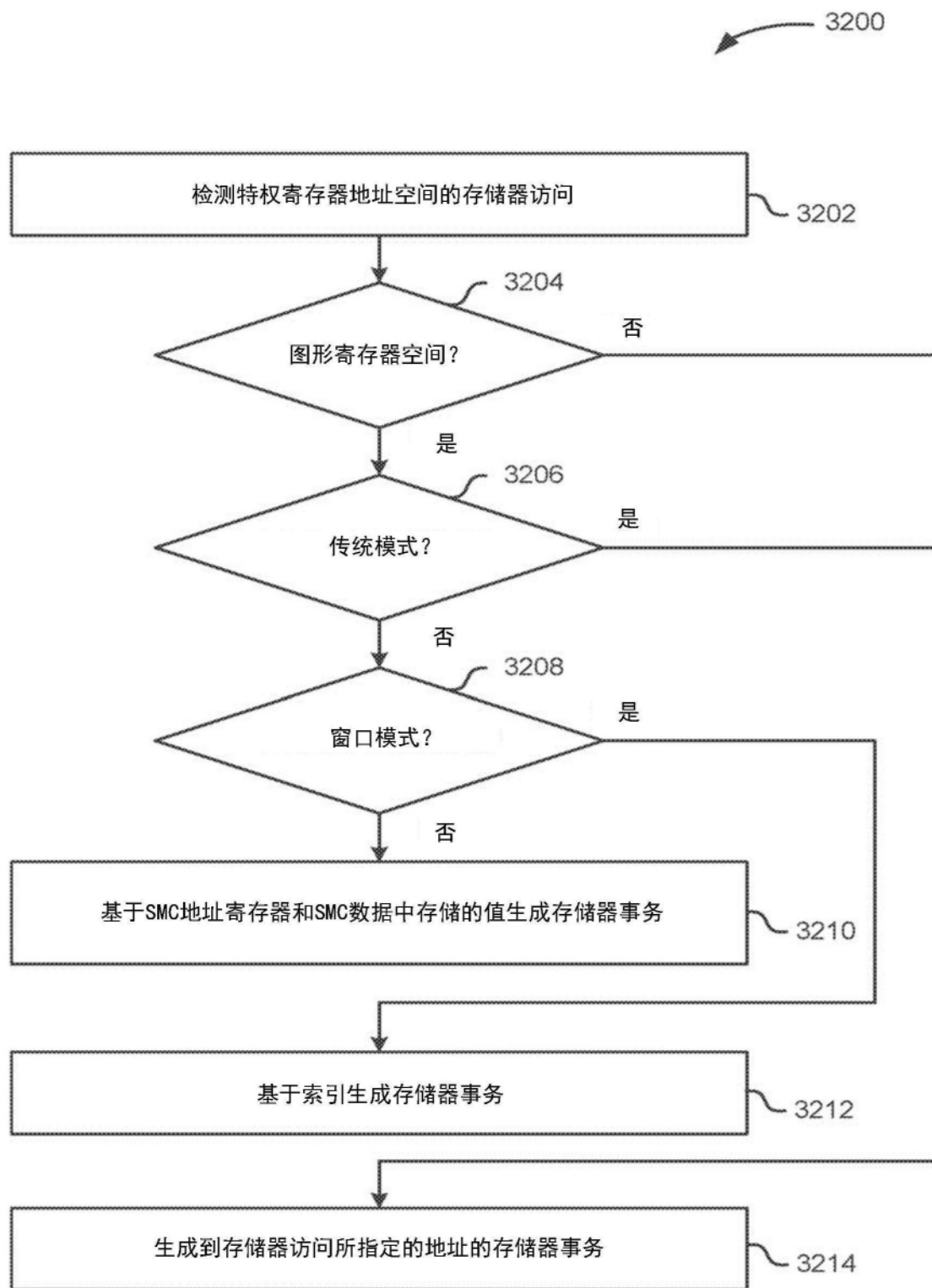


图32

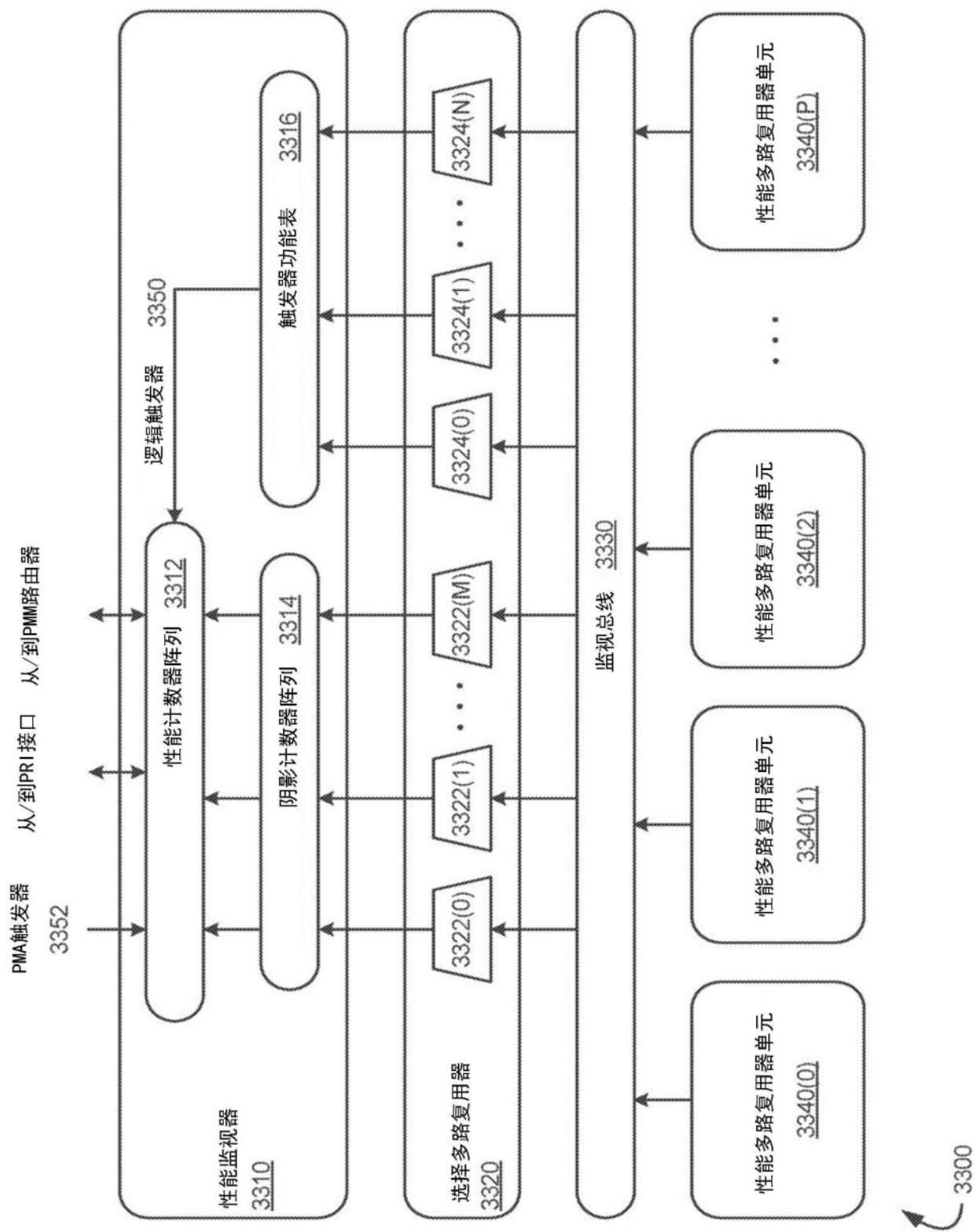


图33

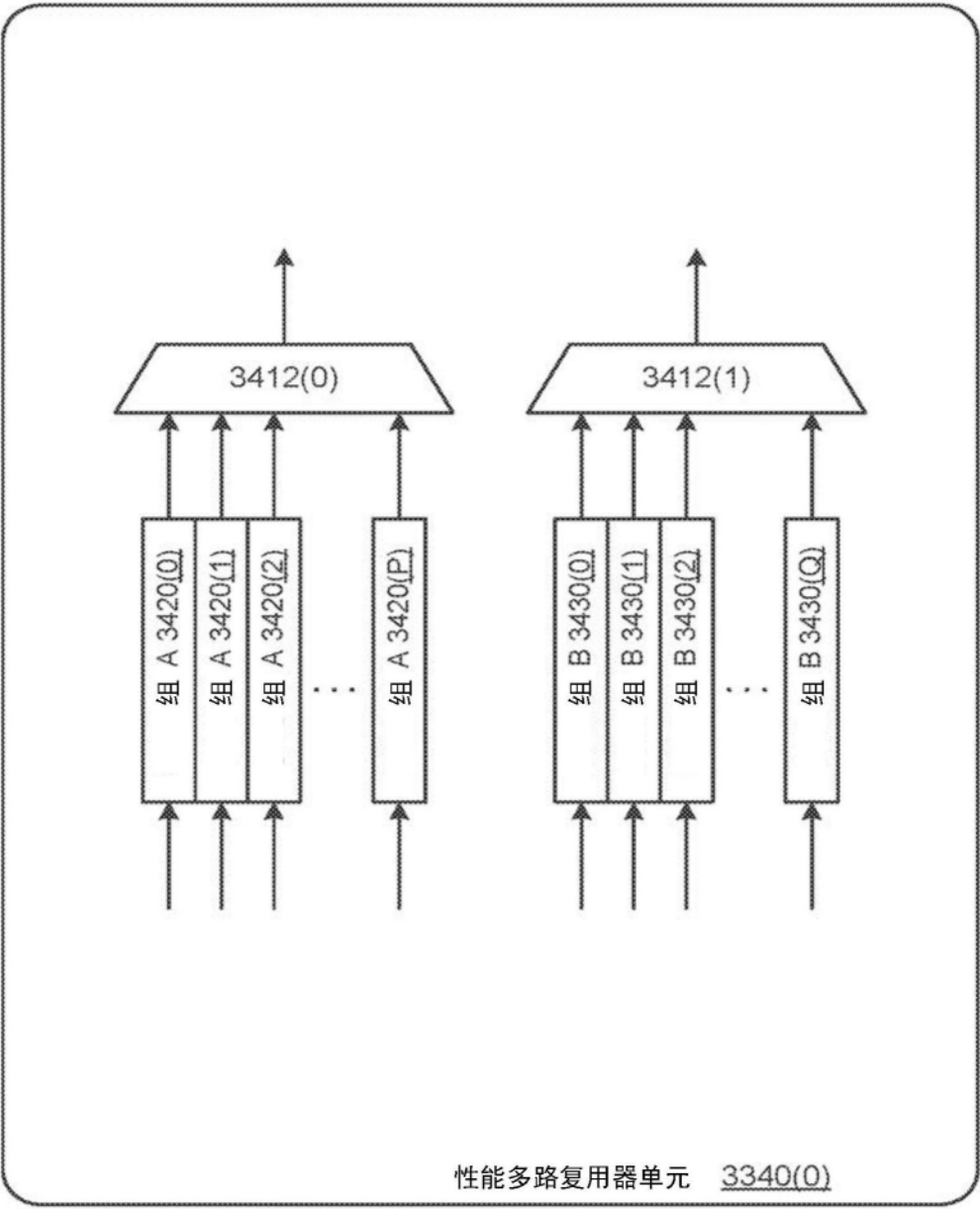


图34A

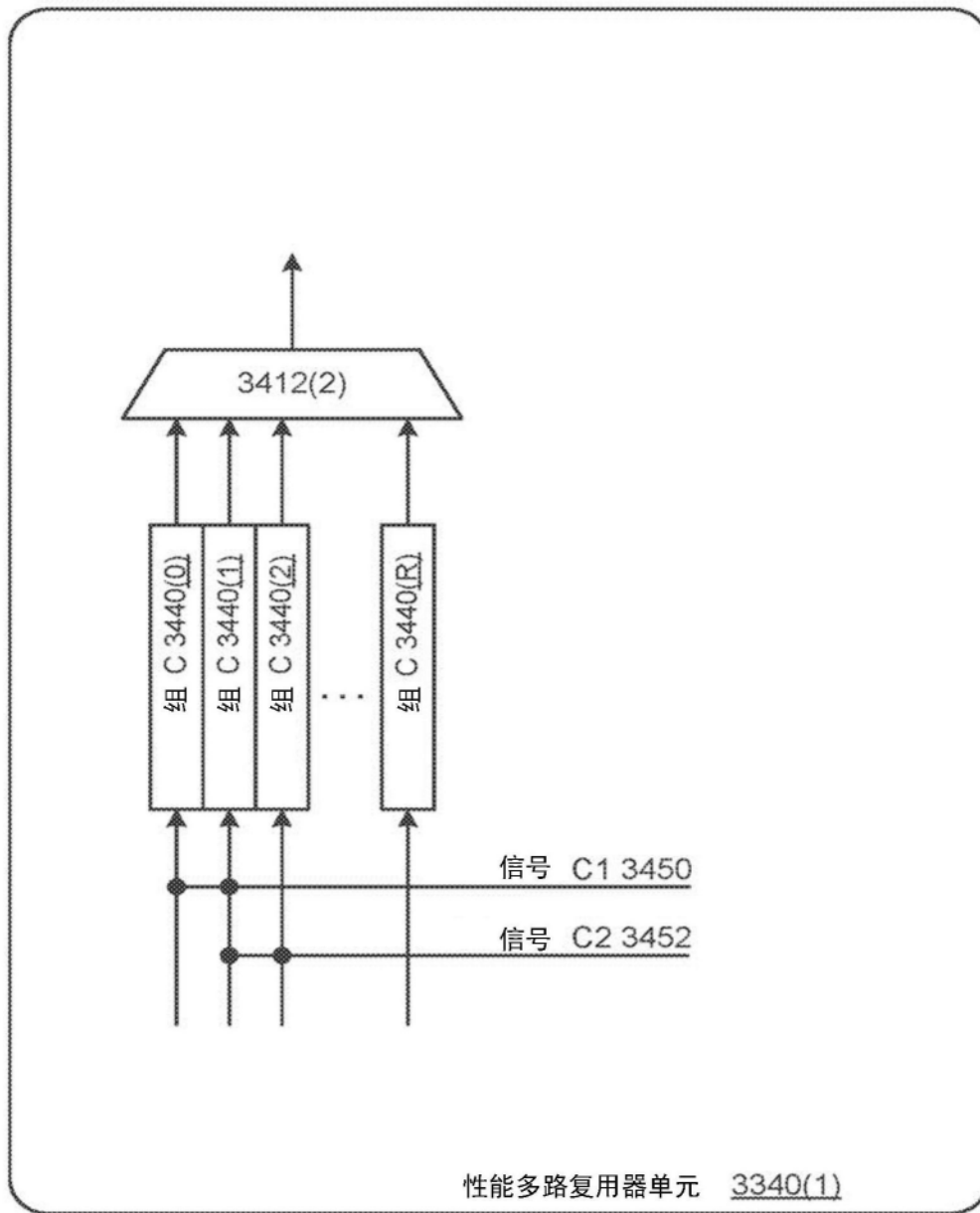


图34B

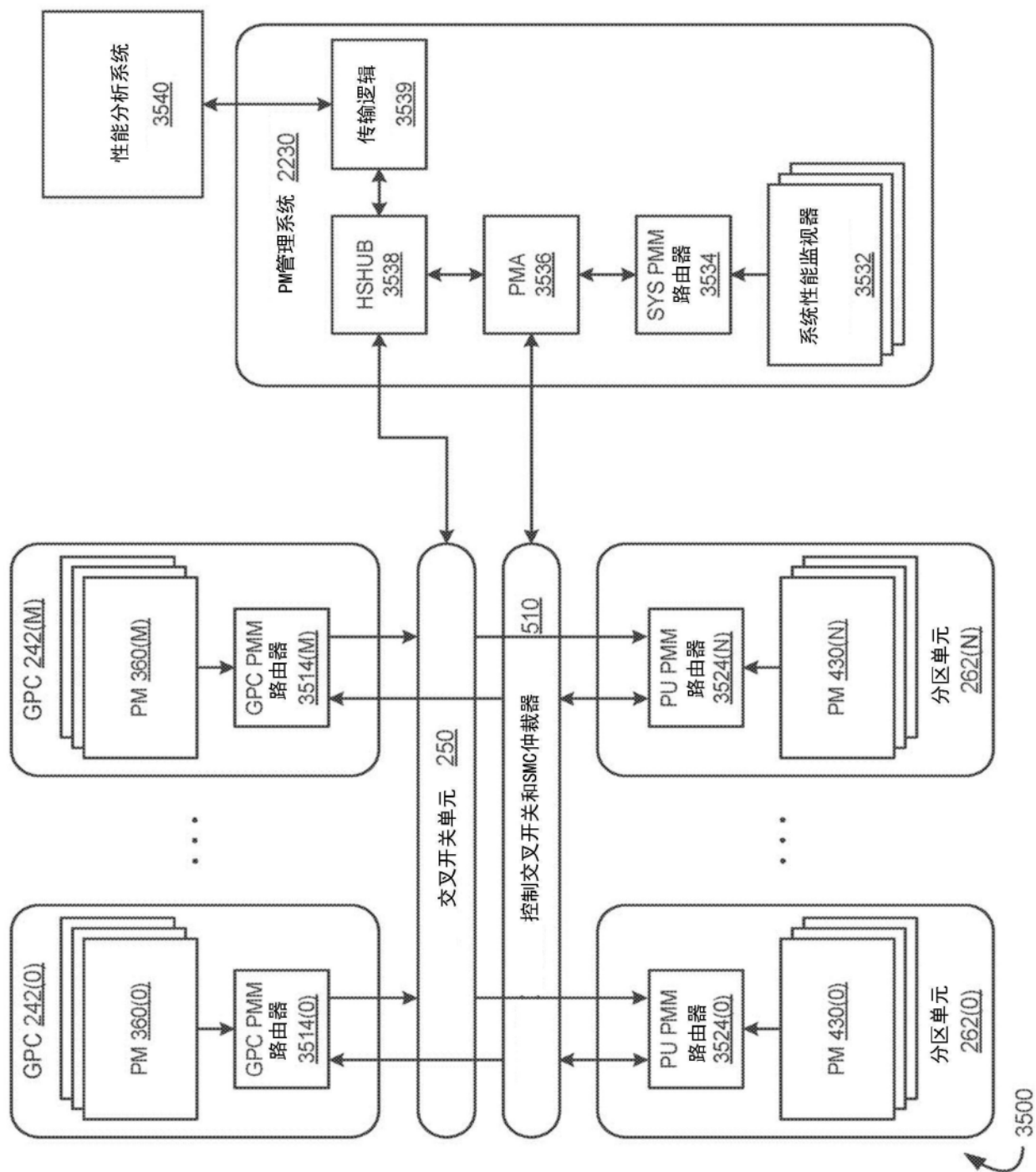


图35

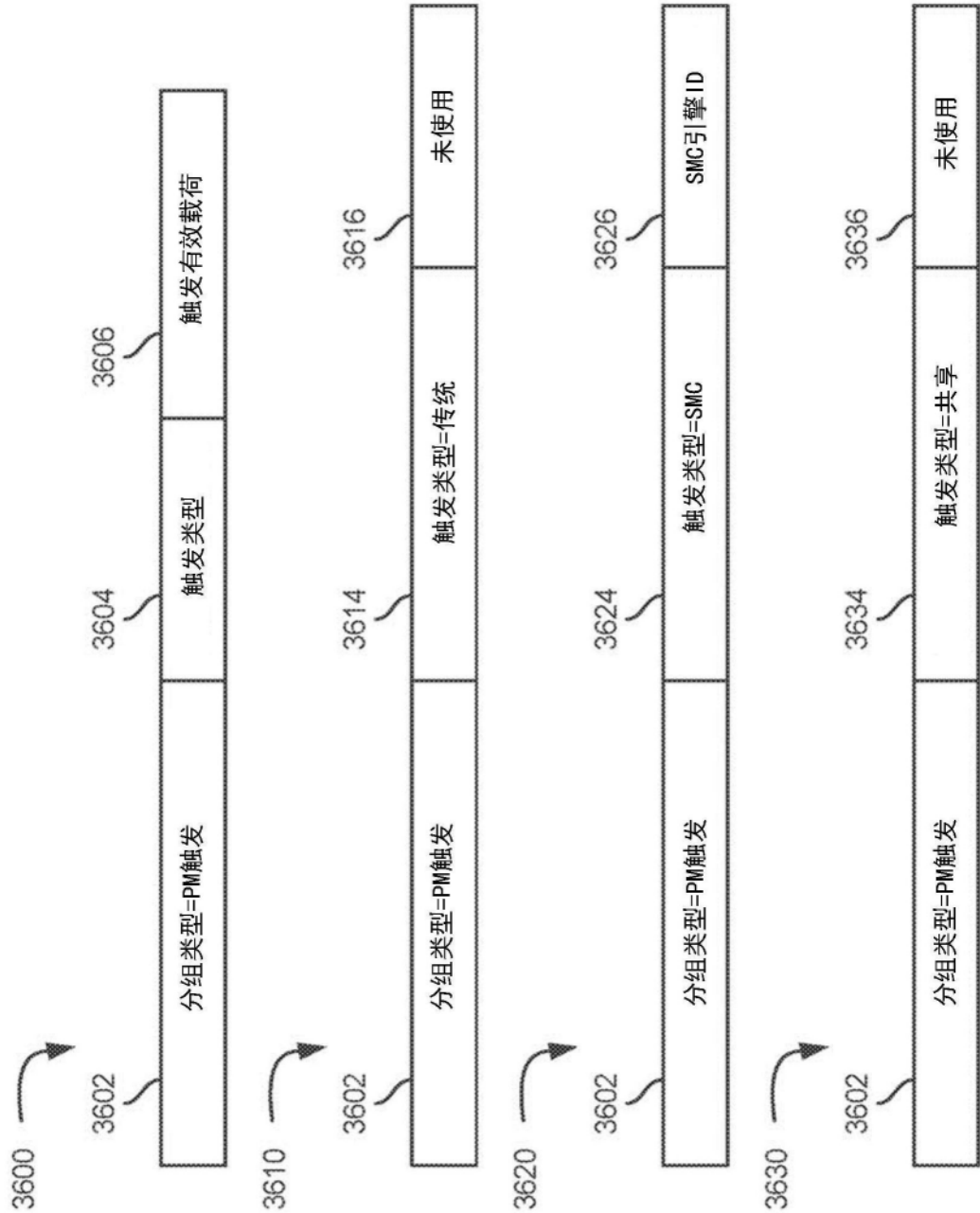


图36

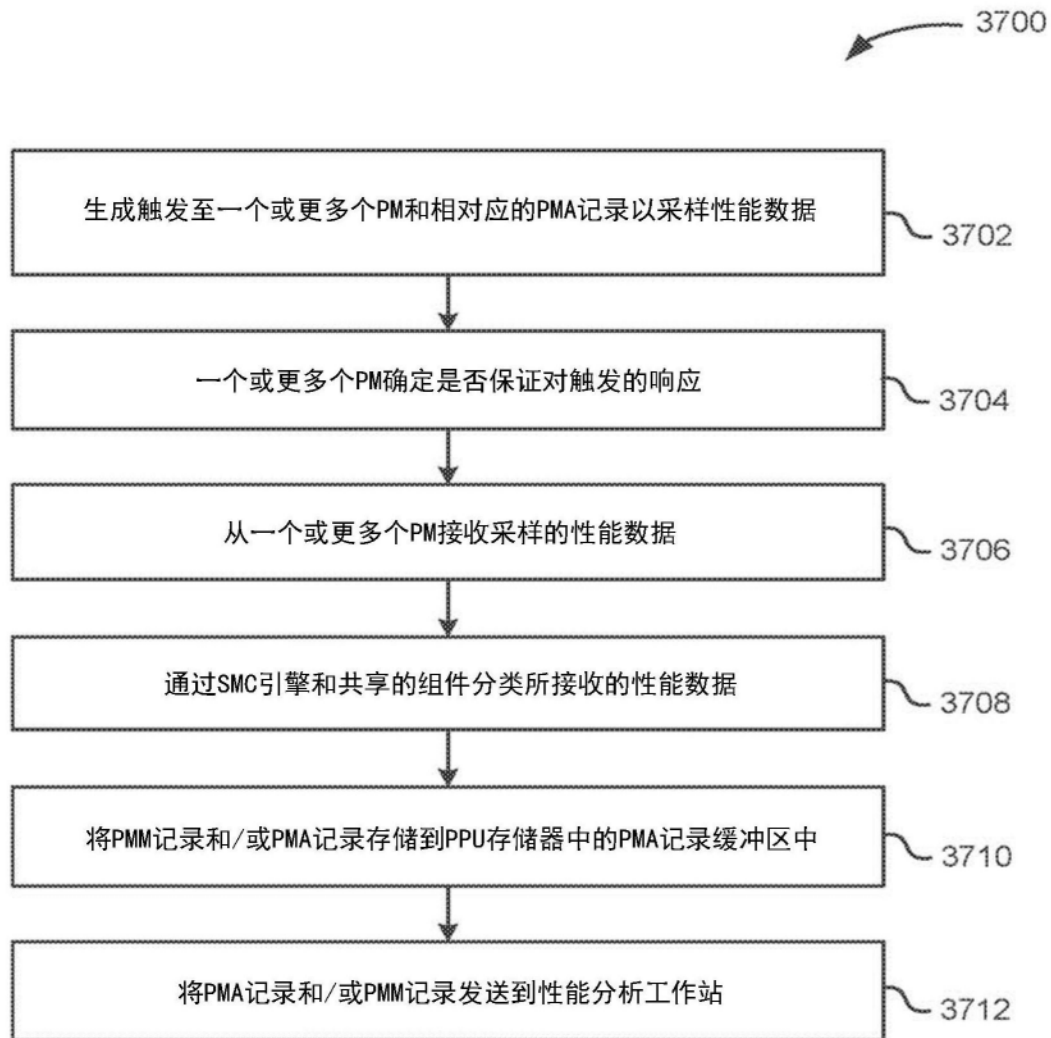


图37

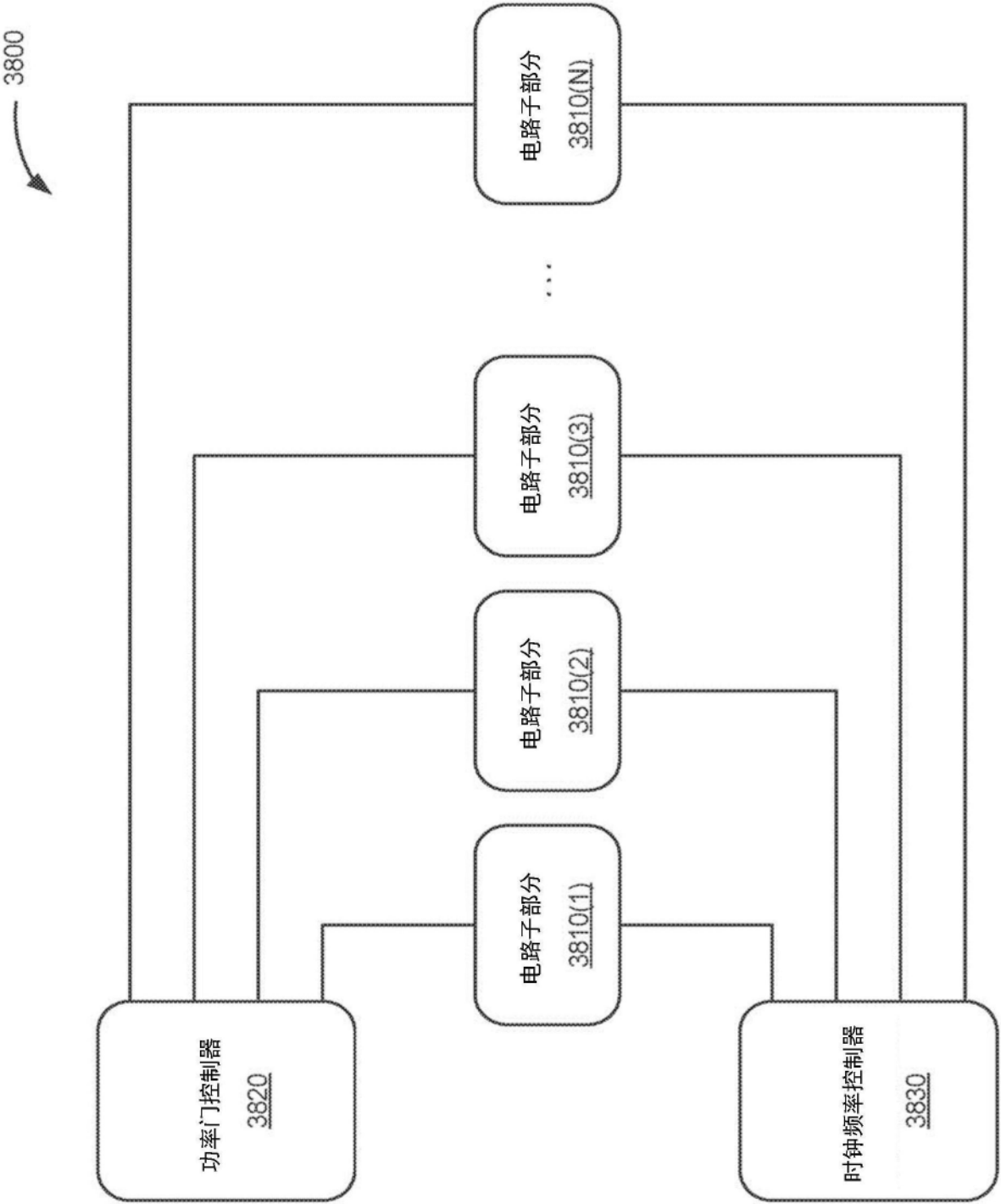


图38

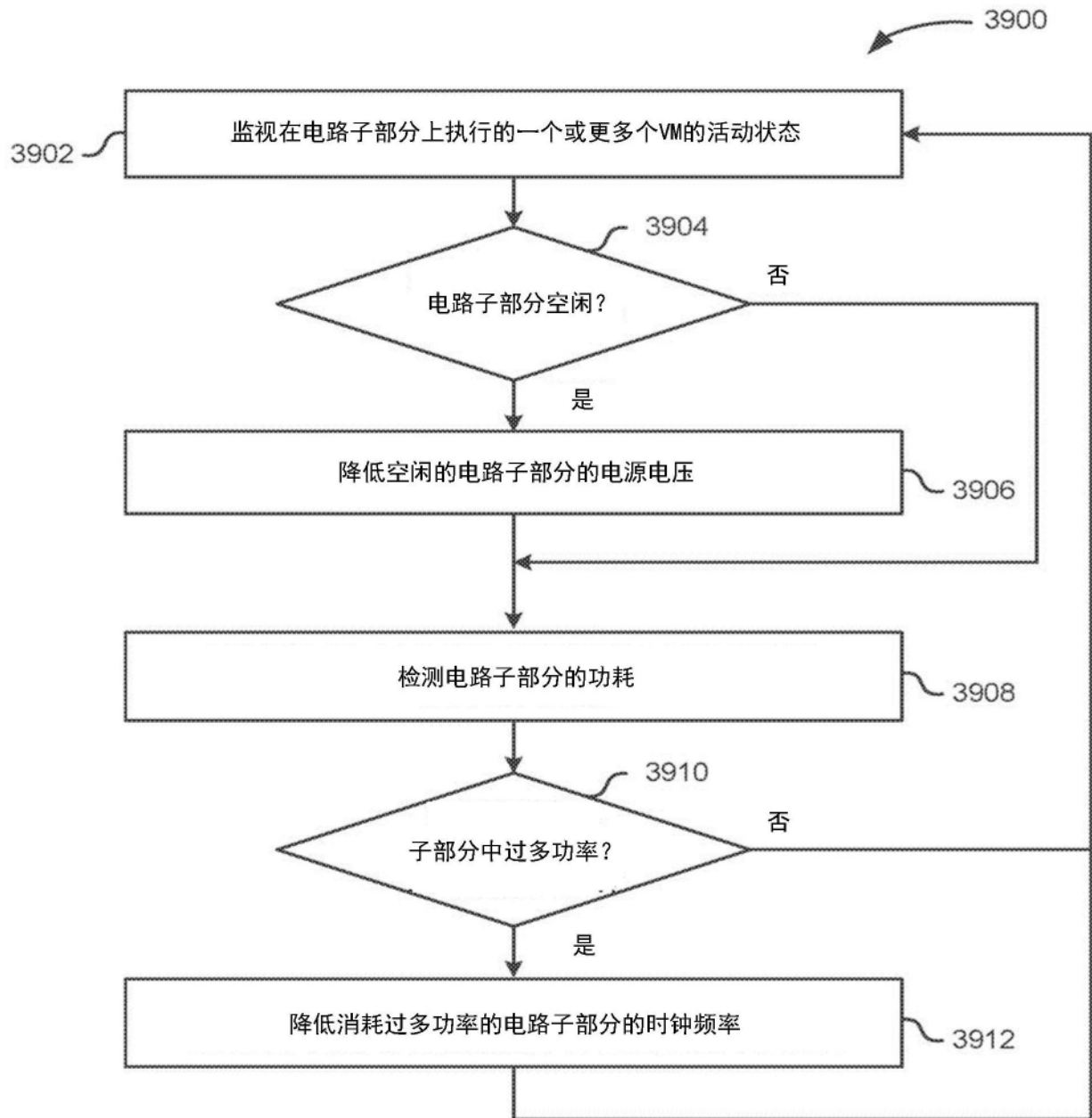


图39