[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 15/167 (2006.01)

[12] 发明专利说明书

专利号 ZL 200410079705.3

[45] 授权公告日 2007 年 2 月 14 日

[11] 授权公告号 CN 1300723C

[22] 申请日 2004.9.17
[21] 申请号 200410079705.3
[30] 优先权
[32] 2003.9.25 [33] US [31] 10/670,835
[73] 专利权人 国际商业机器公司
地址 美国纽约
[72] 发明人 丹尼尔·艾伦·布肯谢尔
迈克尔·诺曼·戴 巴里·L·麦纳
马克·理查德·纳特
[56] 参考文献
US4485438A 1984.11.27
CN1391178A 2003.1.15
审查员 陈晓华

[74] 专利代理机构 中国国际贸易促进委员会专利商标事务所
代理人 李德山

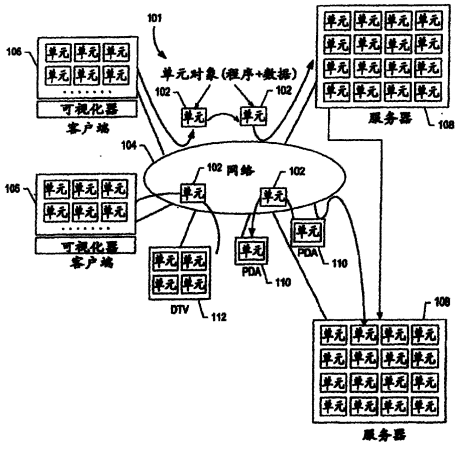
权利要求书 6 页 说明书 40 页 附图 57 页

[54] 发明名称

将处理器用作虚拟设备的方法和信息处理系统

[57] 摘要

提供系统和方法以允许在多处理器系统，例如 BE 环境中使用多个处理器作为虚拟设备。使用此方法，协处理单元(SPU)可以被专门用于执行具体功能(即，音频，视频等)，或者单个 SPU 可以被编程以代表系统中其它处理器执行若干功能。最好运行在主(PU)处理器之一上的应用程序通过对应于 SPU 的设备驱动程序发出 IOCTL 命令。管理主处理器的内核通过向执行专门功能的 SPU 发送适当消息来作出应答。使用此方法，SPU 可以被虚拟化以交换多个任务或专门执行具体任务。



1.一种计算机实现的用于将处理器用作虚拟设备的方法，所述方法包括:

在计算机系统的多个处理器中，从第一处理器向第二处理器通知请求，其中所述多个处理器共享公共存储器，并且其中至少两个处理器是不同的;

把对应于请求的数据存储在对应于第二处理器的本地存储器中;以及

由第二处理器使用存储在第二处理器的本地存储器中的软件代码处理数据。

2.如权利要求1所述的方法，还包括:

从在第一处理器上执行的软件程序把数据写入到存储在公共存储器中的输入缓冲区中，其中存储还包含把数据从输入缓冲区发送到第二处理器的本地存储器;以及

在处理之前，把软件代码从公共存储器加载到第二处理器的本地存储器。

3.如权利要求2所述的方法，其中发送数据到第二处理器的本地存储器和加载软件代码的步骤都使用DMA操作来执行。

4.如权利要求3所述的方法，其中使用多个DMA控制器之一执行DMA操作，所述多个DMA控制器中的一个被分配给第一处理器，并且其中的第二个被分配给第二处理器。

5.如权利要求1所述的方法，还包括:

响应处理产生存储在第二处理器的本地存储器中的结果数据。

6.如权利要求5所述的方法，还包括:

把结果数据写入到存储在公共存储器中的输出缓冲区中。

7.如权利要求5所述的方法，还包括:

把结果数据写入到对应于另一个设备的输入缓冲区中。

8.如权利要求5所述的方法，还包括:

从第二处理器通知多个处理器中的第三处理器，其中第三处理器包含本地存储器；

把结果数据存储在第三处理器的本地存储器中；以及

由第三处理器使用存储在第三处理器的本地存储器中的软件代码处理结果数据。

9.如权利要求5所述的方法，还包括：

把结果数据写入到物理设备。

10.如权利要求1所述的方法，其中存储在第二处理器的本地存储器中的软件代码包含多个代码例程，该方法还包括：

根据请求识别代码例程之一；以及

执行所识别的代码例程。

11.如权利要求1所述的方法，还包括：

针对由多个处理器的一或多个执行的一或多个设备功能初始化任务队列，其中通知还包含：

把请求写入到任务队列；

由第二处理器确定请求在任务队列中；以及

响应确定从任务队列读取请求。

12.如权利要求1所述的方法，还包括：

在通知之前：

把数据加载到公共存储器中的第一位置；

把适于执行处理的软件代码加载到公共存储器中的第二位置；

把指令块写入到公共存储器中的第三位置，其中指令块包含第一和第二位置；

其中通知包含把第三位置写入到对应于第二处理器的邮箱。

13.如权利要求12所述的方法，还包括：

由第二处理器从第二处理器的邮箱读取第三位置；

由第二处理器从存储在第三位置的指令块中读取第一和第二位置；以及

在把数据存储在第二处理器的本地存储器之前，由第二处理器从

公共存储器中的第一位置读取数据。

14.如权利要求12所述的方法，还包括:

由第二处理器读取存储在公共存储器中的第二位置的软件代码;以及

把软件代码存储在第二处理器的本地存储器中。

15.如权利要求14所述的方法，其中，响应于确定存储在公共存储器的第二位置的软件代码尚未被存储在第二处理器的本地存储器中，执行软件代码的读取和存储。

16.一种信息处理系统，包括:

多个异构处理器;

由多个异构处理器共享的公共存储器;

从多个处理器中选择的第一处理器，用于发送请求到第二处理器，该第二处理器也是从多个处理器中选择的;

对应于第二处理器的本地存储器;

与第二处理器相关的DMA控制器，DMA控制器适于在公共存储器和第二处理器的本地存储器之间传送数据;以及

虚拟设备工具，用于把第二处理器作为虚拟设备来操作，虚拟设备工具包含用于以下操作的软件:

从第一处理器通知第二处理器;

把对应于请求的数据存储在第二处理器的本地;以及

由第二处理器使用存储在第二处理器的本地存储器中的软件代码处理数据。

17.如权利要求16所述的信息处理系统,还包括用于以下操作的软件:

从在第一处理器上执行的软件程序把数据写入到存储在公共存储器中的输入缓冲区中，其中存储还包含把数据从输入缓冲区发送到第二处理器的本地存储器;以及

在处理之前，把软件代码从公共存储器加载到第二处理器的本地存储器。

18.如权利要求17所述的信息处理系统，其中发送数据到第二处理器的本地存储器和加载软件代码的操作都使用DMA操作来执行。

19.如权利要求16所述的信息处理系统，还包括用于以下操作的软件：

响应于使用软件代码处理数据，产生存储在第二数据处理器的本地存储器中的结果数据。

20.如权利要求19所述的信息处理系统，还包括用于以下操作的软件：

把结果数据写入到存储在公共存储器中的输出缓冲区中。

21.如权利要求19所述的信息处理系统，还包括用于以下操作的软件：

把结果数据写入到对应于另一个设备的输入缓冲区中。

22.如权利要求21所述的信息处理系统，还包括用于以下操作的软件：

从第二处理器通知多个处理器中的第三处理器，其中第三处理器包含本地存储器；

把结果数据存储在第三处理器的本地存储器中；以及

由第三处理器使用存储在第三处理器的本地存储器中的软件代码处理结果数据。

23.如权利要求19所述的信息处理系统，还包括用于以下操作的软件：

把结果数据写入到物理设备。

24.如权利要求16所述的信息处理系统，其中存储在第二处理器的本地存储器中的软件代码包含多个代码例程，该信息处理系统还包括用于以下操作的软件：

根据请求识别代码例程之一；以及

执行所识别的代码例程。

25.如权利要求16所述的信息处理系统，还包括用于以下操作的软件：

针对由多个处理器的一或多个执行的一或多个设备功能初始化任务队列，其中第二处理器的通知还包含用于以下操作的软件：

把请求写入到任务队列；

由第二处理器确定请求在任务队列中；以及

响应确定从任务队列读取请求。

26.如权利要求**16**所述的信息处理系统，还包括用于以下操作的软件：

响应于使用软件代码对数据的数据，产生存储在第二处理器的本地存储器中的结果数据。

27.如权利要求**16**所述的信息处理系统，还包括用于在通知第二处理器之前进行以下操作的软件：

把数据加载到公共存储器中的第一位置；

把适于执行处理的软件代码加载到公共存储器中的第二位置；

把指令块写入到公共存储器中的第三位置，其中指令块包含第一和第二位置；

其中第二处理器的通知包含用于把第三位置写入到对应于第二处理器的邮箱的软件代码。

28.如权利要求**27**所述的信息处理系统，还包括用于以下操作的软件：

由第二处理器从第二处理器的邮箱读取第三位置；

由第二处理器从存储在第三位置的指令块中读取第一和第二位置；以及

在把数据存储在第二处理器的本地存储器之前，由第二处理器从公共存储器中的第一位置读取数据。

29.如权利要求**27**所述的信息处理系统，还包括用于以下操作的软件：

由第二处理器读取存储在公共存储器的第二位置的软件代码；以及

把软件代码存储在第二处理器的本地存储器中。

30.如权利要求29所述的信息处理系统，其中，响应于确定存储在公共存储器的第二位置的软件代码尚未被存储在第二处理器的本地存储器中，执行软件代码的读取和存储。

将处理器用作虚拟设备的方法和信息处理系统

技术领域

本发明通常涉及使用多个处理器作为虚拟设备的系统和方法。更具体地，本发明涉及使用共享共用存储器的异构处理器以充当虚拟设备的系统和方法。

背景技术

计算机系统已经被广泛用于提供计算机能力给当今现代社会的许多部分。个人计算机系统可以通常定义为桌面，立式或便携微型计算机，包含具有系统处理器和相关易失以及非易失存储器的系统单元，显示监视器，键盘，一或多个磁盘驱动器，固定的磁盘存储设备以及可选的打印机。这些系统的区别特征之一是使用系统板把这些部件电连接在一起。

为执行附加功能或增加其它设备，通常增加固件卡。例如，视频卡通常被加入以执行2D以及3D几何函数以及在显示监视器上显示图形。通常加入计算机系统的其它设备包含调制解调器，LAN卡，驱动控制器以及执行数学函数的专用卡。向计算机系统增加固件卡需要系统板上有额外空间以连接卡。另外，固件卡需要从计算机系统提供的功率并且对计算机系统产生额外的热。通常遇到额外功率和热，不管卡是否充分使用。

使用由固件卡提供的设备的一个挑战是热和功率方面的要求。另一挑战是如果不把旧卡从系统板上去除并且用新卡替换它，则固件卡通常难以升级。为了消除这些挑战，由固件卡执行的一些功能由计算机系统的中央处理单元(CPU)执行。 例如，“软调制解调器”通过软件提供通常由硬件调制解调器执行的调制解调功能。然而，这些“软”设备的问题是其为已经很繁忙的CPU提供了额外的计算负担，并且执行速度通常比基于固件的设备更慢。

多处理器通常用于正变得越来越复杂的信息处理系统。多处理器提供的系统具有比使用单微处理器的系统更强的计算机功率。然而，使用多处理器系统的设计人员面对的一个挑战是内部和外部设备的使用。每个处理器通常使用诸如PCI总线的总线与内部和外部设备通信。具有多处理器的计算机通常具有同构处理器组。换句话说，CPU全部

具有同样的类型，使得它们能够分享工作和执行同样的指令集。因此，具有多处理器的计算机系统的挑战是处理器的缺点在整个处理器组合中是共同的。

因此，需要一种包含异构处理器组合的系统和方法，其中一些处理器适于代表其它处理器执行高速处理，例如仿真设备功能。还需要一种公共存储器，其由异构处理器组共享，以利于各种处理器之间的数据通信。

发明内容

提供一种系统和方法以允许在多处理器系统，例如BE环境中使用多个处理器的虚拟设备。使用此方法，协处理单元(SPU)可以被专用于执行具体功能（即，音频，视频等），或者单个SPU可以被编程以代表系统中的其它处理器执行若干功能。最好运行在主(PU)处理器之一上的应用程序通过对应于SPU的设备驱动程序发出IOCTL命令。管理主处理器的内核通过向正执行专用功能的SPU发送适当消息来作出应答。使用此方法，SPU可以被虚拟化以交换多个任务或专用于执行具体任务。

第一处理器，例如PU，通知SPU之一执行虚拟设备任务。第二处理器(SPU)接收请求，并且在必要时，通过从异构处理器共享的公共存储器读取代码来检索适于执行设备任务的代码，所述异构处理器包含将执行此任务的PU和SPU。SPU还读取数据，例如图形命令。在SPU使用代码完成数据处理之后，SPU可以向另一设备发送数据，例如在另一SPU中运行的虚拟设备，或实际的硬件设备，例如硬件渲染引擎(rasterizer)。如果结果数据将被提供给运行在请求方处理器上的进程（即，PU进程），则使用DMA命令把数据回写到公共存储器。在一个实施例中，每个SPU和PU处理器包含用于写入公共存储器的DMA控制器。

可以建立分立的任务队列，其中每个任务队列对应于一个虚拟设备功能。SPU周期性检查任务队列以确定是否存在等待的请求。当SPU发现需要执行的任务时，SPU获得任务队列的锁，使得虚拟设备的后续任务将由相同SPU处理。

本发明提供了一种计算机实现的用于将处理器用作虚拟设备的方法，所述方法包括：在计算机系统的多个处理器中，从第一处理器向第二处理器通知请求，其中所述多个处理器共享公共存储器，并且其中至少两个处理器是不同的；把对应于请求的数据存储在对应于第二处理器的本地存储器中；以及由第二处理器使用存储在第二处理器

的本地存储器中的软件代码处理数据。

本发明提供了一种信息处理系统，包括：多个异构处理器；由多个异构处理器共享的公共存储器;从多个处理器中选择的第一处理器，用于发送请求到第二处理器，该第二处理器也是从多个处理器中选择的；对应于第二处理器的本地存储器；与第二处理器相关的DMA控制器，DMA控制器适于在公共存储器和第二处理器的本地存储器之间传送数据；以及虚拟设备工具，用于把第二处理器作为虚拟设备来操作，虚拟设备工具包含用于以下操作的软件：从第一处理器通知第二处理器；把对应于请求的数据存储在第二处理器的本地；以及由第二处理器使用存储在第二处理器的本地存储器中的软件代码处理数据。

上述只是一个概述，因而肯定包含对细节的简化，概括和省略；因此，本领域的技术人员会理解，这种概述只是说明性的，并非旨在进行任何方式的限制。本发明的如权利要求单独限定的其它方面，发明特性及优点将在下面提供的非限制性详细描述中变得清楚。

附图说明

通过参照附图，本领域技术人员可更好地理解本发明及其各种目的，特性及优点。在不同附图中使用相同的附图标记指示类似或相同的组成部分。

图1图解了根据本发明的计算机网络的总体体系结构；

图2的图例图解了根据本发明的处理单元(PU)的结构；

图3的图例图解了根据本发明的宽带引擎引擎(BE)的结构；

图4的图例图解了根据本发明的协处理单元(SPU)的结构；

图5的图例图解了根据本发明的处理单元，可视化器(visualizer)(VS)及光学接口的结构；

图6的图例图解了根据本发明的一个处理单元组合；

图7图解了根据本发明的另一个处理单元组合；

图8图解了根据本发明的另一个处理单元组合；

图9图解了根据本发明的另一个处理单元组合；

图10图解了根据本发明的另一个处理单元组合；

图11A图解了根据本发明的芯片封装内的光学接口的集成；

图11B的图例图解了使用图11A的光学接口的处理器的一个配置；

图11C的图例图解了使用图11A的光学接口的处理器的另一配

置;

图12A图解了根据本发明的存储器系统的结构;

图12B图解了根据本发明从第一宽带引擎到第二宽带引擎的数据写入;

图13的图例图解了根据本发明的处理单元的共享存储器的结构;

图14A图解在图13中示出的存储器组的一个结构;

图14B图解在图13中示出的存储器组的另一结构;

图15图解了根据本发明的直接存储器访问控制器的结构;

图16图解了根据本发明的直接存储器访问控制器的可选结构;

图17-31图解了根据本发明的数据同步操作;

图32的三态存储器图根据本发明的数据同步模式图解存储器位置的各种状态;

图33根据本发明图解了硬件沙箱(sandbox)的密钥控制表(key control table)的结构;

图34根据本发明图解了存储硬件沙箱的存储器访问密钥的模式;

图35根据本发明图解了硬件沙箱的存储器访问控制表的结构;

图36是使用图33的密钥控制表及图35的存储器访问控制表访问存储器沙箱的步骤的流程图;

图37根据本发明图解了软件单元的结构;

图38是根据本发明向SPU发出远程过程调用的步骤的流程图;

图39根据本发明图解了用于处理流数据的专用管道的结构;

图40是根据本发明在流数据的处理中由图39的专用管道完成的步骤的流程图;

图41根据本发明图解了用于流数据处理的专用管道的可选结构;

图42根据本发明图解了用于协议SPU对应用及数据的并行处理的绝对定时器的模式;

图43的图例示出了初始化程序的处理单元,及程序的加载相关文件到协处理单元的运行时刻加载程序;

图44-46示出了载入SPU的本地存储器的各种设备代码模块;

图47的流程图示出了使用适于SPU执行的设备代码对计算机系统的初始化；

图48的流程图示出了由SPU在管理多个设备代码文件时采取的步骤；

图49的图例示出了用于管理能够由SPU之一执行多个虚拟设备的数据结构；

图50的流程图示出了进程调用由SPU之一执行的虚拟设备时所采取的步骤；

图51的流程图示出了由非专用SPU在识别和执行请求的虚拟设备任务时采取的步骤；

图52的流程图示出了由专用SPU在执行请求的虚拟设备任务时采取的步骤；

图53的图例示出了用于帮助SPU对虚拟设备任务的处理的任务队列管理器；

图54的流程图示出了由任务队列管理器在帮助SPU对设备任务的处理时采取的步骤；

图55的流程图示出了任务队列管理器通知应用程序预先请求的设备进行请求；

图56的流程图示出了由任务队列管理器管理的SPU采取的步骤；以及

图57的模块图图解了具有共享系统存储器的主处理器和多个辅助处理器的处理元件。

具体实施方式

以下描述旨在提供本发明例子的详细描述，而不是对本发明自身的限制。而是，属于本发明范围的任何数量的变化在后面的权利要求中定义。

根据本发明的计算机系统101的总体体系结构在图1中示出。

如此图所示，系统101包含连接多个计算机及计算设备的网络104。网络104可以是LAN，全球网络，例如因特网，或任何其他计算

机网络。

连接到网络104的计算机及计算设备（网络的“成员”）包含例如客户端计算机106，服务计算机108，个人数字助理(PDA)110，数字电视(DTV)112及其它有线或无线计算机及计算设备。由网络104的成员使用的处理器是由相同的通用计算机模块构造。这些处理器也最好全部具有相同ISA并且根据相同指令集执行处理。包含在任何特定处理器内的模块的数量取决于该处理器所需的处理能力。

例如，由于系统101的服务器108比客户端106执行更多的数据及应用处理，服务器108比客户端106包含更多的计算模块。另一方面，PDA 110执行处理的量最少。因此，PDA 110包含最小量的计算模块。DTV 112执行的处理量介于客户端106及服务器108之间。因此，DTV 112包含介于客户端106及服务器108之间的数量的计算模块。如下所述，各个计算模块包含处理控制器及多个相同处理单元，用于执行通过网络104发送的数据及应用的并行处理。

系统101的这种同构结构利于可适应性，处理速度及处理效率。由于系统101的各个成员使用一或多个相同计算模块（或某个部分）执行处理，执行数据及应用的实际处理的特定计算机或计算设备是不重要的。此外，具体应用及数据的处理可以在网络的成员中间分享。通过唯一标识全系统中包括由系统101处理的数据及应用的单元(cell)，处理结果可以被发送到请求处理的计算机或计算设备，而无需考虑这个处理发生的位置。由于执行这个处理的模块具有通用结构并且使用通用ISA，避免了为在处理器中间实现兼容而增加的软件层的计算负担。这个体系结构及编程模型利于得到执行例如实时多媒体应用所需的处理速度。

为了进一步利用系统101所带来的处理速度及效率，由这个系统处理的数据及应用被封装到唯一标识别的、统一格式化的软件单元(software cell)102中。

每个软件单元102包含或能够包含应用程序和数据。每个软件单元也包含一个ID以全局标识遍及网络104及系统101的单元。软件单元

的这种统一结构和遍及网络的软件单元的唯一标识，利于网络的任何计算机或计算设备上对应用及数据的处理。例如，客户端106可以设定软件单元102，但由于客户端106受处理能力的限制，将这个软件单元发送到服务器108进行处理。因此，软件单元可以在全网络104迁移以根据网络上处理资源的可用性进行处理。

系统101的处理器和软件单元的同构结构也避免了现在异构网络的许多问题。例如，避免了设法允许在使用任何指令集的任何ISA上处理应用的低效编程模型，例如，例如Java虚拟机的虚拟机。因此，系统101可以进一步实现比现在的网络更有效和高效的宽带处理。

针对网络104的所有成员的基本处理模块是处理单元(PU)。图2图解了PU的结构。如该图所示，PE 201包括处理单元(PU) 203，直接存储器访问控制器(DMAC) 205及多个协处理单元(SPU)，即，SPU 207，SPU 209，SPU 211，SPU 213，SPU 215，SPU 217，SPU 219及SPU 221。本地PE总线223在SPU，DMAC 205及PU 203中间发送数据及应用。例如，本地PE总线223可以具有传统体系结构或被实现为分组交换网络。分组交换网络的实现在需要更多硬件的同时，增加了可用带宽。

可以使用各种方法构造PE 201以实现数字逻辑。然而，PE 201最好被构造成在硅质基底上使用互补金属氧化物半导体(CMOS)的单个集成电路。基底的可选材料包含镓砷化物(gallium arsinide)，镓铝砷化物(gallium aluminum arsinide)及其它使用各种掺杂剂的所谓III-B化合物。也可以通过使用例如快速单通量量子(RSFQ)逻辑的超导材料实现PE 201。

PE 201通过高带宽存储器连接227与动态随机访问存储器(DRAM) 225紧密关联。DRAM 225作为PE 201的主存储器。尽管DRAM 225最好是动态随机访问存储器，但是DRAM 225可以通过使用例如作为静态随机访问存储器(SRAM)，磁性随机访问存储器(MRAM)，光学存储器或全息存储器的其它装置实现。DMAC 205利于DRAM 225及PE 201的SPU和PU之间的数据传送。如下面进一步讨论的，DMAC 205为各个SPU指定DRAM 225内的专用区域，在该区域

只有SPU可以写数据并且只有SPU可以从该区域读取数据。这个专用区域被称为“沙箱”。

例如，PU 203可以是能独立处理数据及应用的标准处理器。在操作中，PU 203安排及协调SPU对数据及应用的处理。SPU最好是单指令、多数据(SIMD)处理器。

在PU 203的控制下，SPU以并行且独立的方式执行这些数据及应用的处理。DMAC 205控制PU 203及SPU对存储在共享DRAM 225中的数据及应用的访问。尽管PE 201最好包含8个SPU，但是根据所需处理能力，在PU中可使用更多或更少数量的SPU。同时，例如PE 201的若干PU可以加入或封装到一起以提供增强的处理能力。

例如，如图3所示，四个PU可以封装或加入到一起，例如，在一或多个芯片封装内，以构成网络104的成员的单个处理器。这种结构被称为宽带引擎(BE)。如图3所示，BE 301包含四个PU，即，PE 303，PE 305，PE 307及PE 309。这些PU间的通信通过BE总线 311进行。宽带宽存储器连接313提供共享DRAM 315及这些PU之间的通信。代替BE总线 311地，BE 301的PU间的通信可以通过DRAM 315及该存储器连接进行。

输入/输出(I/O)接口317及外部总线319提供宽带引擎301和网络104的其它成员之间的通信。BE 301的各个FU以并行及独立的方式执行数据及应用的处理，该方式类似于由PU的SPU执行的并行及独立的应用及数据的处理。

图4图解了SPU的结构。SPU 402包含本地存储器406，寄存器410，四个浮点单元412及四个整数单元414。然而，仍然根据所需的处理能力，可以使用更多或更低数量的浮点单元412及整数单元414。在一个最优实施例中，本地存储器406包含128千字节的存储器，并且寄存器410的容量是128乘128位。浮点单元412最好以每秒32千兆(billion)浮点运算的速度（32 GFLOPS）操作，而整数单元414最好以每秒32千兆运算的速度(32 GOPS)操作。

本地存储器406不是高速缓冲存储器。本地存储器406最好被构造

为SRAM。对于SPU的超高速缓存一致性支持是不必要的。PU可以需要针对由PU初始化的直接存储器访问的超高速缓存一致性支持。然而，对于由SPU初始化的直接存储器访问或对于来自和到外部设备的访问，不需要超高速缓存一致性支持。

SPU 402还包含用于针对SPU发送应用及数据的总线404。在一个最优实施例中，该总线是1024位宽。SPU 402还包含内部总线408，420和418。在一个最优实施例中，总线408具有256位的宽度，并且提供本地存储器406和寄存器410之间的通信。总线420和418分别提供寄存器410和浮点单元412之间，及寄存器410和整数单元414之间的通信。在一个最优实施例中，从寄存器410到浮点或整数单元的总线418和420的宽度是384位，并且从浮点或整数单元到寄存器410的总线418和420的宽度是128位。从寄存器410到浮点或整数单元的这些总线的宽度比从这些单元到寄存器410的宽度更大，使得处理期间可容纳更大的来自寄存器410的数据流。对于每个计算最多需要三个字。然而，每个计算的结果通常只是一个字。

图5-10进一步图解了网络104的成员的处理器的模块结构。 例如，如图5所示，处理器可以包括单个PU 502。如上所述，该PU通常包括PU，DMAC和8个SPU。每个SPU包含本地存储器(LS)。另一方面，处理器可以包括可视化器(VS) 505的结构。如图5所示，VS 505包括PU 512，DMAC 514和四个SPU，即，SPU 516，SPU 518，SPU 520和SPU 522。通常由PU的其它四个SPU占用的芯片封装内的空间在这种情况下由像素引擎 508，图像高速缓存510和阴极射线管控制器(CRTC)504占用。根据PU 502或VS 505需要的通信速度，光学接口506也可以包含在芯片封装上。

使用该标准化的模块结构，能容易和有效地构造处理器的许多其它变型。例如，如图6示出的处理器包括两个芯片封装，即，包括BE的芯片封装602和包括四个VS的芯片封装604。输入/输出(I/O) 606提供芯片封装602的BE和网络104之间的接口。总线608提供芯片封装602和芯片封装604之间的通信。输入输出处理器(IOP) 610控制进出I/O 606

的数据流。I/O 606可以制造成专用集成电路(ASIC)。来自VS的输出是视频信号612。

图7图解了针对具有两个光学接口704和706的BE 702的芯片封装，所述接口用于向网络104的其它成员(或本地连接的其它芯片封装)提供超高速通信。例如，BE 702可以作为网络104上的服务器。

图8的芯片封装包括两个PE 802和804以及两个VS 806和808。I/O 810提供芯片封装602和网络104之间的接口。来自芯片封装的输出是视频信号。例如，该结构可以作为图形工作站。

图9图解了另一个结构。该结构包含图8中图解的结构的处理能力的一半。 代替两个PU地，提供一个PE 902，并且代替两个VS地，提供一个VS 904。I/O 906具有图8中图解的I/O的带宽的一半。 然而，这种处理器也可以作为图形工作站。

最后的结构在图10中示出。该处理器仅由单个VS 1002和I/O 1004构成。例如，该结构可以作为PDA。

图11A图解了把光学接口集成到网络104的处理器的芯片封装。这些光学接口将光学信号转换为电信号以及将电信号转换为光学信号，并且光学接口可以由包含例如镓砷化物，铝镓砷化物，锗和其它元素或化合物的各种材料构成。如此图所示，光学接口1104和1106被制造在BE 1102的芯片封装上。BE总线1108提供BE 1102的PU，即，PE 1110，PE 1112，PE 1114，PE 1116和这些光学接口间的通信。光学接口1104包含两个端口，即，端口1118和端口1120，并且光学接口1106也包含两个端口，即，端口1122和端口1124。端口1118，1120，1122和1124被分别连接到光波导1126，1128，1130和1132。通过光学接口1104和1106的端口，经由这些光波导针对BE 1102收发光信号。

通过使用这种光波导和各个BE的四个光学端口，多个BE可以在各种结构中被连接在一起。 例如，如图11B所示，两个或更多BE，例如BE 1152，BE 1154和BE 1156，可以通过这种光学端口被串行连接。在这个例子中，BE 1152的光学接口1166通过其光学端口连接到BE 1154的光学接口1160的光学端口。以类似方式，BE 1154上的光学接

口1162的光学端口被连接到BE 1156的光学接口1164的光学端口。

在图 11C 中图解了矩阵结构。在该结构中，每个 BE 的光学接口被连接到其它两个 BE。如该图所示，BE 1172 的光学接口 1188 的光学端口之一被连接到 BE 1176 的光学接口 1182 的光学端口。光学接口 1188 的其它光学端口被连接到 BE 1178 的光学接口 1184 的光学端口。以类似方式，BE 1174 的光学接口 1190 的光学端口之一被连接到 BE 1178 的光学接口 1184 的另一个光学端口。光学接口 1190 的另一个光学端口被连接到 BE 1180 的光学接口 1186 的光学端口。该矩阵结构能以类似方式扩展到其它 BE。

使用串行结构或矩阵结构，网络104的处理器可以被构造成具有任何期望的规模和能力。当然，其它端口可以加到BE的光学接口上，或加到具有比BE更多或更低数量的PU的处理器，以构成其它结构。

图12A图解了BE的DRAM的控制系统和结构。 在具有其它规模以及包含或多或少PU的处理器中使用类似控制系统和结构。如此图所示，交叉交换装置(cross-bar switch)把具有包括BE 1201的四个PU的每个DMAC 1210连接到8个组控制1206。每个组控制1206控制DRAM 1204的8个组1208（在此图中只示出四个）。因此，DRAM 1204包括总共64个组。在一个最优实施例中，DRAM 1204具有64兆字节的容量，并且各个组具有1兆字节的容量。在此优选实施例中，各组内最小可寻址单元是 1024位块。

BE 1201还包含交换单元1212。交换单元1212允许BE上与BE 1201紧耦合的其它SPU访问DRAM 1204。因此，第二个BE可以紧耦合到第一个BE，并且每个BE的每个SPU可以寻址的存储器位置数量两倍于SPU通常可访问的存储器位置数量。对第一BE的DRAM和对第二BE的DRAM的数据直接读或写可以通过例如交换单元1212的交换单元进行。

例如，如图12B所示，为了实现此写入，第一BE的SPU，例如，BE 1222的SPU 1220，向第二BE的DRAM，例如，BE 1226的DRAM 1228的存储器位置发出写命令（不象通常那样向BE 1222的DRAM

1224发出）。BE 1222的DMAC 1230通过交叉交换装置1221向组控制1234发送写命令，并且组控制1234向连接到组控制1234的外部端口1232发送该命令。BE 1226的DMAC 1238接收写命令，并且传送该命令到BE 1226的交换单元1240。交换单元1240识别包含在写命令中的DRAM地址，并且通过BE 1226的组控制1242向DRAM 1228的组1244发送数据用于存储在该地址。因此，交换单元1240允许DRAM 1224和DRAM 1228作为BE 1226的SPU的单个存储器空间。

图13示出了DRAM的64组的结构。这些组被排列成8行，即，行1302，1304，1306，1308，1310，1312，1314和1316，以及8列，即，列1320，1322，1324，1326，1328，1330，1332和1334。每行由组控制器控制。因此，每个组控制器控制8兆字节的存储器。

图14A和14B图解了用于存储及访问DRAM的最小可寻址存储器单元，例如1024位块，的不同结构。 在图14A中，DMAC 1402在单个组1404中存储8个1024位块1406。另一方面，在图14B中，当DMAC 1412读写包含1024位块的数据时，这些块在两个组，即，组1414和组1416之间交错。因此，每个这样的组包含十六个数据块，并且每个数据块包含512位。这种交错可以利于DRAM的快速访问，并且在某些应用的处理中有用。

图15图解了PE内DMAC 1504的体系结构。如此图所示，包括DMAC 1506的结构硬件被分布在整个PE，使得每个SPU 1502直接访问DMAC 1506的结构节点1504。每个节点通过该节点直接访问的SPU执行适于存储器访问的逻辑。

图16示出了DMAC的可选实施例，即，非分布式体系结构。在这种情况下，DMAC 1606的结构硬件是集中式的。SPU 1602和PU 1604通过本地PE总线1607与DMAC 1606通信。DMAC 1606通过交叉交换装置被连接到总线1608。总线1608被连接到DRAM 1610。

如上所述，PU的所有多个SPU可以独立访问共享DRAM中的数据。结果，第一个SPU可以在第二个SPU请求时操作本地存储器中的特定数据。如果数据在该时刻被从共享DRAM提供给第二SPU，则由

于第一SPU正在进行可改变数据值的处理，所以该数据无效。因此，如果第二处理器在该时刻从共享DRAM接收数据，第二处理器可能产生错误结果。例如，数据可以是全局变量的具体值。如果第一处理器在其处理期间改变此值，则第二处理器将接收过期的值。因此，需要一种同步SPU针对共享DRAM内存储器位置的数据读写数据的模式(scheme)。该模式必须防止从这样的存储器位置读取数据，其中另一SPU正在其本地存储器中操作该存储器位置的数据，因此该存储器位置的数据不是最新的，并且防止把数据写入到存储当前数据的存储器位置。

为了克服这些问题，针对DRAM的每个可寻址存储器位置，附加存储器分段被分配在DRAM中，用于存储涉及存储在存储器位置中的数据的状态信息。此状态信息包含满/空(F/E)位，向存储器位置请求数据的SPU的标识(SPU ID)，以及应当从中读取所请求数据的SPU本地存储器的地址(LS地址)。DRAM的可寻址存储器位置可以是任何长度。在一个最优实施例中，此长度是1024位。

F/E位设置为1指示存储在相关存储器位置的数据是最新的。另一方面，F/E位设置为0指示存储在相关存储器位置的数据不是最新的。如果当此位被设置成0时SPU请求数据，则防止SPU立即读取数据。在这种情况下，标识请求数据的SPU的SPU ID和标识此SPU的本地存储器内的存储器位置的LS地址被输入到附加存储器分段，其中SPU当数据变成最新时读取数据到该存储器位置。

附加存储器分段还分配给SPU的本地存储器内的每个存储器位置。此附加存储器分段存储一个位，被称为“忙位”。忙位用于保留用于存储从DRAM检索的特定数据的相关LS存储器位置。如果忙位针对本地存储器中特定存储器位置被设置成1，则SPU只能使用该存储器位置用于这些特定数据的写入。另一方面，如果忙位针对本地存储器中特定存储器位置被设置成0，则SPU能够使用该存储器位置用于任何数据的写入。

在图17-31中图解了该方式的例子，其中F/E位，SPU ID，LS地

址和忙位被用于同步针对PU的共享DRAM的数据的读、写。如图17所示，一或多个PU，例如，PE 1720，与DRAM 1702交互。PE 1720包含SPU 1722和SPU 1740. SPU 1722包含控制逻辑1724，并且SPU 1740包含控制逻辑1742。SPU 1722还包含本地存储器1726。该本地存储器包含多个可寻址存储器位置1728。SPU 1740包含本地存储器1744，并且此本地存储器还包含多个可寻址存储器位置1746。所有这些可寻址存储器位置的长度最好是1024位。

附加存储器分段与每个LS可寻址存储器位置相关。例如，存储器分段1729和1734分别与本地存储器位置1731和1732相关，并且存储器分段1752与本地存储器位置1750相关。如上所述，“忙位”被存储在所有这些其它存储器分段中。用若干X示出本地存储器位置1732以指示该位置包含数据。

DRAM 1702包含多个可寻址存储器位置1704，其中包含存储器位置1706和1708。这些存储器位置的长度最好也是1024位。附加存储器分段同样与所有这些存储器位置相关。例如，附加存储器分段1760与存储器位置1706相关，并且附加存储器分段1762与存储器位置1708相关。涉及存储在每个存储器位置中的数据的状态信息被存储在与该存储器位置相关的存储器分段中。如上所述，这个状态信息包含F/E位，SPU ID和LS地址。例如，针对存储器位置1708，这个状态信息包含F/E位1712，SPU ID 1714和LS地址1716。

使用状态信息和忙位，能够实现PU的SPU，或一组PU中间针对共享DRAM的数据同步读、写。

图18图解了从SPU 1722的LS存储器位置1732向DRAM 1702的存储器位置1708同步写入数据的发起。SPU 1722的控制1724发起这些数据的同步写入。由于存储器位置1708为空，F/E位1712被设置成0。 结果，在LS位置1732中的数据能够被写入存储器位置1708。另一方面，如果该位被设置成1以指示存储器位置1708已满并且包含最新有效数据，则控制1722会接收差错消息并且被禁止向这个存储器位置写数据。

在图19中示出了向存储器位置1708成功同步写入数据的结果。

写数据被存储在存储器位置1708，并且F/E位1712被设置成1。该设置指示存储器位置1708满，并且在该存储器位置中的数据是最新及有效的。

图20图解了从DRAM 1702的存储器位置1708向本地存储器1744的LS存储器位置1750同步读取数据的发起。为了发起这个读取，在LS存储器位置1750的存储器分段1752中的忙位被设置成1，以保留该储器位置给这些数据。该忙位设置为1防止SPU 1740在此存储器位置存储其它数据。

如图21所示，控制逻辑1742接着针对DRAM 1702的存储器位置1708发出同步读命令。由于与这个存储器位置相关的F/E位1712被设置成1，存储在存储器位置1708中的数据被认为是最新及有效的。结果，为准备把数据从存储器位置1708传送到LS存储器位置1750，F/E位1712被设置成0。在图22中示出该设置。该位设置为0指示在读取这些数据之后，存储器位置1708中的数据将无效。

如图23所示，接着，存储器位置1708内的数据被从存储器位置1708读取到LS存储器位置1750。图24示出最后状态。在存储器位置1708中的数据的复本被存储在LS存储器位置1750。F/E位1712被设置成0以指示在存储器位置1708中的数据无效。该无效是SPU 1740对这些数据进行改变的结果。存储器分段1752中的忙位也被设置成0。该设置指示现在LS存储器位置1750可被SPU 1740用于任何目的，即，该LS存储器位置不再处于等待接收特定数据的保留状态。因此，现在SPU 1740能够为任何目的访问LS存储器位置1750。

图25-31图解了当DRAM 1702的存储器位置的F/E位被设置成0以指示在该存储器位置中的数据不是最新或有效时，从DRAM 1702的存储器位置，例如存储器位置1708，到SPU的本地存储器的LS存储器位置，例如本地存储器1744的LS存储器位置1752的同步数据读取。如图25所示，为了启动这个传送，在LS存储器位置1750的存储器分段1752中的忙位被设置成1，以保留该LS存储器位置用于该数据传送。如图26所示，接着，控制逻辑1742针对DRAM 1702的存储器位置1708

发出同步读命令。由于与该存储器位置相关的F/E位，即F/E位1712，被设置成0，则存储在存储器位置1708中的数据无效。结果，向控制逻辑1742发送信号以阻塞从该存储器位置的即时数据读取。

如图27所示，接着，该读命令的SPU ID 1714及LS地址1716被写入存储器分段1762。在这种情况下，SPU 1740的SPU ID和LS存储器位置1750的LS存储器位置被写入存储器分段1762。因此当存储器位置1708内的数据变成最新时，该SPU ID和LS存储器位置被用于确定当前数据被发送到的位置。

当SPU写数据到该存储器位置时，存储器位置1708中的数据变得有效及最新。在图28中图解了数据从例如SPU 1722的存储器位置1732到存储器位置1708的同步写入。 由于该存储器位置的F/E位1712被设置成0，因此允许这些数据的同步写入。

如图29所示，在此写入之后，存储器位置1708中的数据变成最新及有效。因此，来自存储器分段1762的SPU ID 1714和LS地址1716被立即从存储器分段1762读取，并且该信息接着被从此分段删除。F/E位1712也被设置成0，以期即时读取存储器位置1708中的数据。如图30所示，当读取SPU ID 1714和LS地址1716时，该信息被立即用于把存储器位置1708中的有效数据读取到SPU 1740的LS存储器位置1750。图31中示出了最后状态。该图示出了从存储器位置1708复制到存储器位置1750的有效数据，存储器分段1752中的忙位设置为0并且存储器分段1762中的F/E位1712设置为0。 设置该忙位为0使LS存储器位置1750现在能够被SPU 1740出于任何目的访问。设置该F/E位为0指示存储器位置1708中的数据不再最新及有效。

图32根据存储在对应于一存储器位置的存储器分段中的F/E位，SPU ID和LS地址的状态，总结了上面描述的操作，以及DRAM的该存储器位置的各种状态。存储器位置可以具有三个状态。这三个状态是空状态3280，其中F/E位被设置成0并且没有提供针对SPU ID或LS地址的信息；满状态3282，其中F/E位被设置成1并且没有提供针对SPU ID或LS地址的信息；和阻塞状态3284，其中F/E位被设置成0并且提供针

对SPU ID和LS地址的信息。

如此图所示，在空状态3280，允许同步写操作，并且导致转变到满状态3282。然而，同步读取操作导致向阻塞状态3284转变，因为当存储器位置处于空状态时，存储器位置中的数据不是最新的。

在满状态3282，允许同步读取操作并且导致向空状态3280转变。另一方面，禁止满状态3282中的同步写操作，以防止覆盖有效数据。如果在此状态中尝试这种写操作，则不会出现状态改变，并且差错消息被发送到SPU的相应控制逻辑。

在阻塞状态3284，允许把数据同步写入到存储器位置，并且导致向空状态3280转变。另一方面，禁止阻塞状态3284中的同步读取操作，以防止与在前的导致该状态的同步读取操作冲突。如果在阻塞状态3284中尝试同步读取操作，则不会出现状态改变，并且差错消息被发送到SPU的相应控制逻辑。

用于针对共享DRAM同步读、写数据的上述模式也可以被用于免除通常由处理器专用于从外部设备读取数据和向外部设备写数据的计算资源。此输入/输出(I/O)功能可以由PU执行。然而，使用该同步模式的修改，运行适当程序的SPU能够执行此功能。例如，通过使用此模式，接收由外部设备发起的用于从I/O接口发送数据的中断请求的PU，可以把此请求的处理分配给此SPU。接着，SPU向I/O接口发出同步写命令。此接口接着通知外部设备数据现在可以被写入DRAM。SPU接着向DRAM发出同步读命令以把DRAM的相关存储器空间设置成阻塞状态。对于SPU的本地存储器中接收数据所需的存储器位置，SPU也将其忙位设置为1。在阻塞状态，与DRAM的相关存储器空间相关的附加存储器分段包含SPU的ID和SPU的本地存储器的相关存储器位置的地址。外部设备接着发出同步写命令以把数据直接写入DRAM的相关存储器空间。由于此存储器空间处于阻塞状态，数据被立即从此空间读到附加存储器分段中所标识的SPU的本地存储器的存储器位置。接着这些存储器位置的忙位被设置成0。 当外部设备完成数据写入时，SPU向PU发出有关发送已完成的通知信号。

因此，通过使用此模式，源自外部设备的数据传送可以用PU上最小计算负载来处理。然而，分配此功能的SPU应当能够向PU发出中断请求，并且外部设备应当直接访问DRAM。

每个PU的DRAM包含多个“沙箱”。 沙箱定义共享DRAM的区域，在该区域之外，特定SPU或SPU组不能读或写数据。这些沙箱提供安全性以防止一个SPU处理的数据被另一SPU处理的数据破坏。这些沙箱也允许从网络104下载软件单元到特定沙箱，并且软件单元没有可能破坏整个DRAM的数据。在本发明中，在DRAM和DMAC的硬件中实现沙箱。通过用硬件而不是软件实现这些沙箱，获得速度和安全性的优势。

PU的PU控制分配给SPU的沙箱。由于PU通常只操作信任的例如操作系统的程序，此模式不会危害安全性。根据此模式，PU构建并且维护密钥控制表(key control table)。在图33中图解了此密钥控制表。如此图所示，每个密钥控制表3302中的表项包含SPU的标识(ID)3304，该SPU的SPU密钥3306以及密钥掩码3308。下面说明此密钥掩码的使用。密钥控制表3302最好被存储在例如静态随机访问存储器(SRAM)的相对快速的存储器中，并且与DMAC相关联。密钥控制表3302中的表项由PU控制。当SPU请求向DRAM的特定存储器位置写入数据，或从DRAM的特定存储器位置读取数据时，DMAC相对于与该存储器位置相关的存储器访问密钥，评估密钥控制表3302中分配给该SPU的SPU密钥3306。

如图34所示，专用存储器分段3410被分配给DRAM 3402的每个可寻址存储器位置3406。针对存储器位置的存储器访问密钥3412被存储在此专用存储器分段。如上所述，也与每个可寻址存储器位置3406相关的另一个附加专用存储器分段3408存储用于写数据到存储器位置以及从存储器位置读取数据的同步信息。

在操作中，SPU向DMAC发出DMA命令。此命令包含DRAM 3402的存储器位置3406的地址。在执行此命令之前，DMAC使用SPU的ID 3304在密钥控制表3302中查找请求方SPU的密钥3306。接着DMAC把

请求方SPU的SPU密钥3306与存储在和SPU寻求访问的DRAM的存储器位置相关的专用存储器分段3410中的存储器访问密钥3412比较。如果两个密钥不匹配，则不执行DMA命令。另一方面，如果两个密钥匹配，则执行DMA命令以及请求的存储器访问。

在图35中图解了可选实施例。 在这个实施例中，PU还维护存储器访问控制表3502。存储器访问控制表3502包含针对DRAM内每个沙箱的表项。在图35的特定例子中，DRAM包含64个沙箱。存储器访问控制表3502中的每个表项包含沙箱的标识(ID)3504，基存储器地址3506，沙箱尺寸3508，存储器访问密钥3510以及访问密钥掩码3512。基存储器地址3506提供DRAM中特定存储器沙箱起始的地址。因此，沙箱尺寸3508提供沙箱的大小以及特定沙箱的端点。

图36是使用密钥控制表3302以及存储器访问控制表3502执行DMA命令的步骤的流程图。在步骤3602，SPU向DMAC发出DMA命令以访问沙箱内的一或多个特定存储器位置。此命令包含标识请求访问的特定沙箱的沙箱ID 3504。在步骤3604，DMAC使用SPU的ID 3304在密钥控制表3302中查找请求方SPU的密钥3306。在步骤3606，DMAC使用命令中的沙箱ID 3504在存储器访问控制表3502中查找与该沙箱相关的存储器访问密钥3510。在步骤3608，DMAC把分配给请求方SPU的SPU密钥3306与和沙箱相关的访问密钥3510比较。在步骤3610，确定两个密钥是否匹配。如果两个密钥不匹配，处理进行到步骤3612，在此步骤不执行DMA命令，并且差错消息被发送到请求方SPU或PU，或二者。另一方面，如果在步骤3610发现两个密钥匹配，则处理执行到步骤3614，其中DMAC执行DMA命令。

SPU密钥的密钥掩码以及存储器访问密钥向这个系统提供更大灵活性。密钥的密钥掩码把被屏蔽位转换成通配符。例如，如果与SPU密钥3306相关的密钥掩码3308设置其最后两位为“屏蔽”(例如通过设置密钥掩码3308中的这些位为1来指定)，则SPU密钥可以是1或0并且仍然匹配存储器访问密钥。例如，SPU密钥可以是1010。此SPU密钥通常只允许访问具有1010的访问密钥的沙箱。然而，如果此SPU密钥

的SPU密钥掩码被设置成0001，那么此SPU密钥可以被用于获得对具有1010或者1011的访问密钥的沙箱的访问。类似地，具有设置为0001的掩码的访问密钥1010可以通过具有1010或者1011的SPU密钥的SPU访问。由于SPU密钥掩码和存储器密钥掩码可以被同时使用，能够建立由SPU到沙箱的可访问性的许多变化。

本发明还针对系统101的处理器提供了新编程模型。此编程模型使用软件单元102。这些单元能够被发送到网络104上的任何处理器进行处理。此新编程模型还使用系统101的独特模块化结构和系统101的处理器。

由SPU从SPU的本地存储器直接处理软件单元。在DRAM中，SPU不直接操作任何数据或程序。在SPU处理这些数据和程序之前，DRAM中的数据和程序被读取到SPU的本地存储器。因此，SPU的本地存储器包含程序计数器，堆栈和用于执行这些程序的其它软件元素。PU通过向DMAC发出直接存储器访问(DMA)命令来控制SPU。

图37图解了软件单元102的结构。 如此图所示，例如，软件单元3702的软件单元包含路由信息部分3704和主体3706。包含在路由信息部分3704中的信息依赖于网络104的协议。路由信息部分3704包含头3708，目的ID 3710，源ID 3712和应答ID 3714。目的ID包含网络地址。例如，在TCP/IP协议下，网络地址是网际协议(IP)地址。目的ID 3710还包含PU和SPU的标识，其中单元应当发送到该PU和SPU以进行处理。源ID 3712包含网络地址，并且标识单元所来自的PU和SPU，以允许目的PU和SPU在必要时获得有关单元的附加信息。应答ID 3714包含网络地址，并且标识PU和SPU，其中有关单元的查询以及单元处理的结果应当被导向该PU和SPU。

单元主体3706包含独立于网络协议的信息。图37的分解部分示出了单元主体3706的细节。单元主体3706的头3720标识单元主体3706的起始。单元接口3722包含单元使用所需的信息。此信息包含全局唯一ID 3724，所需的SPU 3726，沙箱尺寸3728以及前一单元ID 3730。

全局唯一ID 3724唯一标识整个网络104的软件单元3702。根据源

ID 3712，例如源ID 3712内的PU或SPU的唯一标识，以及软件单元3702的生成或发送时间和日期，产生全局唯一ID 3724。所需的SPU 3726提供执行单元所需要的SPU的最小数量。沙箱尺寸3728提供执行单元所必须的、所需SPU的相关DRAM中的保护存储器的数量。前一单元ID 3730提供需要顺序执行(例如流数据)的单元组内前一单元的标识。

实现部分3732包含单元的核心信息。此信息包含DMA命令列表3734，程序3736和数据3738。程序3736包含由SPU运行的、例如SPU程序3760和3762的程序（称作“spulet”），并且数据3738包含用这些程序处理的数据。DMA命令列表3734包含开始程序所需要的系列DMA命令。这些DMA命令包含DMA命令3740，3750，3755和3758。PU向DMAC发出这些DMA命令。

DMA命令3740包含VID 3742。VID 3742是当发出DMA命令时SPU的映射到物理ID的虚拟ID。DMA命令3740还包含加载命令3744和地址3746。加载命令3744指示SPU把特定信息从DRAM读取到本地存储器。地址3746提供DRAM中包含此信息的虚拟地址。例如，信息可以是来自程序部分3736的程序，来自数据部分3738的数据或其它数据。最后，DMA命令3740包含本地存储器地址3748。此地址标识本地存储器中应当加载信息的地址。DMA命令3750包含类似信息。也可以是其它DMA命令。

DMA命令列表3734也包含系列踢动(kick)命令，例如，踢动命令3755和3758。踢动命令是由PU向SPU发出的命令，用以启动单元的处理。DMA踢动命令3755包含虚拟SPU ID 3752，踢动命令3754和程序计数器3756。虚拟SPU ID 3752标识要踢动的SPU，踢动命令3754提供相关踢动命令，并且程序计数器3756提供用于执行程序的程序计数器的地址。DMA踢动命令3758向相同SPU或另一SPU提供类似信息。

如上所述，PU把SPU作为独立处理器，而不是协处理器。因此，为了由控制SPU的处理，PU使用类似于远程过程调用的命令。这些命令被称为“SPU远程过程调用”(SRPC)。PU通过向DMAC发出系列DMA命令来实现SRPC。DMAC加载SPU程序及其相关堆栈结构到

SPU的本地存储器。接着PU向SPU发出初始踢动以执行SPU程序。

图38图解了执行spulet的SRPC的步骤。在图38的第一部分3802中示出了在启动指定SPU对spulet的处理时由PU执行的步骤，并且在图38的第二部分3804中示出了在处理spulet时由指定SPU执行的步骤。

在步骤3810，PU评估spulet并且接着指定用于处理spulet的SPU。在步骤3812，通过向DMAC发出MDA命令以便为必要的一或多个沙箱设置存储器访问密钥，PU分配DRAM中的空间用于执行spulet。在步骤3814，PU使能针对指定SPU的中断请求以通知spulet完成。在步骤3818，PU向DMAC发出DMA命令以把spulet从DRAM加载到SPU的本地存储器。在步骤3820，执行DMA命令，并且把spulet从DRAM读取到SPU的本地存储器。在步骤3822，PU向DMAC发出DMA命令以把与spulet相关的堆栈结构从DRAM加载到SPU的本地存储器。在步骤3823，执行DMA命令，并且把堆栈结构从DRAM读取到SPU的本地存储器。在步骤3824，PU发出针对DMAC的DMA命令以向SPU分配密钥，从而允许SPU对在步骤3812指定的一或多个硬件沙箱读、写数据。在步骤3826，DMAC用分配给SPU的密钥更新密钥控制表(KTAB)。在步骤3828，PU向SPU发出DMA命令"踢动"以开始程序的处理。根据特定spulet，在特定SRPC的执行中可由PU发出其它DMA命令。

如上所示，图38的第二部分3804图解了在执行spulet时SPU执行的步骤。在步骤3830，SPU响应在步骤3828发出的踢动命令开始执行spulet。在步骤3832，在spulet的指示下，SPU评估spulet的相关堆栈结构。在步骤3834，SPU向DMAC发出多个DMA命令以把指定为堆栈结构需要的数据从DRAM加载到SPU的本地存储器。在步骤3836，执行这些DMA命令，并且把数据从DRAM读取到SPU的本地存储器。在步骤3838，SPU执行spulet并且产生结果。在步骤3840，SPU向DMAC发出DMA命令以存储DRAM的结果。在步骤3842，执行DMA命令，并且把spulet的结果从SPU的本地存储器写入到DRAM。在步骤3844，SPU向PU发出中断请求以通知SRPC已经完成。

SPU在PU的指示下独立执行任务的能力允许PU将一组SPU，以

及与一组SPU相关的存储器资源专用于执行扩展任务。例如，PU可以将一或多个SPU，以及与此一或多个SPU相关的一组存储器沙箱专用于在扩展周期上接收网络104上发送的数据，以及将在此周期接收的数据导向一或多个其它SPU及其相关的存储器沙箱以进行进一步处理。此能力尤其有利于处理通过网络104发送的流数据，例如，流MPEG或流ATRAC音频或视频数据。PU可以将一或多个SPU及其相关存储器沙箱专用于接收这些数据，以及将一或多个其它SPU及其相关存储器沙箱专用于解压缩并且进一步处理这些数据。换句话说，PU能够在一组SPU及其相关存储器沙箱中间建立专用管道关系以处理这种数据。

然而，为了有效执行这种处理，管道的专用SPU以及存储器沙箱在没有发生包括数据流的spulet的处理期间应当保持专用于管道。换句话说，在这些周期期间，专用SPU及其相关沙箱应当处于保留状态。当spulet的处理完成时，SPU及其一或多个相关存储器沙箱的保留被称作“驻留终止”。驻留终止响应来自PU的指令而发生。

图39，40A以及40B图解了专用管道结构的建立，其包括一组SPU及其相关沙箱，用于处理流数据，例如，流MPEG数据。 如图39所示，此管道结构的部件包含PE 3902和DRAM 3918。PE 3902包含PU 3904，DMAC 3906和包含SPU 3908，SPU 3910和SPU 3912的多个SPU。通过PE总线3914，在PU 3904，DMAC 3906和这些SPU间进行通信。宽带宽总线3916连接DMAC 3906到DRAM 3918。DRAM 3918包含例如沙箱3920，沙箱3922，沙箱3924和沙箱3926的多个沙箱。

图40A图解了建立专用管道的步骤。在步骤4010，PU 3904分配SPU 3908以处理网络spulet。网络spulet包括用于处理网络104的网络协议的程序。在这种情况下，此协议是传输控制协议/网际协议(TCP/IP)。符合此协议的TCP/IP数据包通过网络104发送。在接收时，SPU 3908处理这些包并且将包中的数据装配到软件单元102中。在步骤4012，当网络spulet的处理完成时，PU 3904指示SPU 3908执行驻留终止。在步骤4014，PU 3904分配PU 3910和3912以处理MPEG spulet。

在步骤4015，当MPEG spulet的处理完成时，PU 3904指示SPU 3910和3912也执行驻留终止。在步骤4016，PU 3904指定沙箱3920作为由SPU 3908和SPU 3910访问的源沙箱。在步骤4018，PU 3904指定沙箱3922作为由SPU 3910访问的目的沙箱。在步骤4020，PU 3904指定沙箱3924作为由SPU 3908和SPU 3912访问的源沙箱。在步骤4022，PU 3904指定沙箱3926作为由SPU 3912访问的目的沙箱。在步骤4024，SPU 3910和SPU 3912分别向源沙箱3920和源沙箱3924内的存储器块发送同步读命令，以设置这些存储器块为阻塞状态。处理最后进行到步骤4028，其中专用管道的建立完成并且保留专用于管道的资源。因此，SPU 3908，3910和3912及其相关沙箱3920，3922，3924和3926进入保留状态。

图40B图解了通过此专用管道处理流MPEG数据的步骤。在步骤4030，处理网络spulet的SPU 3908在其本地存储器中接收来自网络104的TCP/IP数据包。在步骤4032，SPU 3908处理这些TCP/IP数据包并且将这些包内的数据装配到软件单元102中。在步骤4034，SPU 3908检查软件单元的头3720（图37）以确定单元是否包含MPEG数据。如果单元不包含MPEG数据，那么，在步骤4036，SPU 3908发送该单元到DRAM 3918内指定的通用沙箱，以通过不包含在专用管道内的其它SPU处理其它数据。SPU 3908还将此发送通知PU 3904。

另一方面，如果软件单元包含MPEG数据，那么，在步骤4038，SPU 3908检查该单元的前一单元ID 3730（图37）以识别该单元属于的MPEG数据流。在步骤4040，SPU 3908选择专用管道的SPU处理该单元。在这种情况下，SPU 3908选择SPU 3910处理这些数据。此选择是以前一单元ID 3730和负载均衡系数为基础的。例如，如果前一单元ID 3730指示软件单元所属的MPEG数据流的前一软件单元被发送到SPU 3910进行处理，则当前软件单元通常也将被发送到SPU 3910进行处理。在步骤4042，SPU 3908发出同步写命令以把MPEG数据写入到沙箱3920。由于此沙箱先前被设置成阻塞状态，在步骤4044，MPEG数据自动被从沙箱3920读取到SPU 3910的本地存储器。在步骤4046，SPU

3910在其本地存储器中处理MPEG数据以产生视频数据。在步骤4048，SPU 3910把视频数据写入沙箱3922。在步骤4050，SPU 3910向沙箱3920发出同步读命令以准备此沙箱来接收其它MPEG数据。在步骤4052，SPU 3910处理驻留终止。此处理导致此SPU进入保留状态，在此期间SPU等待处理MPEG数据流中的其它MPEG数据。

其它专用结构可以在一组SPU及其相关沙箱中间建立以处理其它类型的数据。例如，如图41所示，例如SPU 4102，4108和4114的专用SPU组可以被建立以执行三维对象的几何变换，以产生二维显示列表。这些二维显示列表可以由其它SPU进一步处理（呈现）以产生像素数据。为执行此处理，沙箱被专门用于SPU 4102，4108和4114，用于存储三维对象和这些对象的处理产生的显示列表。例如，源沙箱4104，4110和4116被专门用于存储分别由SPU 4102，SPU 4108和SPU 4114处理的三维对象。以类似方式，目的沙箱4106，4112和4118被专门用于存储分别由SPU 4102，SPU 4108和SPU 4114处理这些三维对象而产生的显示列表。

协同SPU 4120专门用于在其本地存储器中从目的沙箱4106，4112和4118接收显示列表。SPU 4120在这些显示列表中间做出仲裁，并且把它们发送给其它SPU以呈现像素数据。

系统101的处理器还使用绝对定时器。绝对定时器向SPU和PU的其它单元提供时钟信号，该时钟信号既独立于又快于驱动这些单元的时钟信号。图42中图解了此绝对定时器的使用。

如此图所示，绝对定时器建立了针对SPU的任务执行的时间预算。此时间预算提供完成这些任务的时间，该时间长于SPU处理任务所需的时间。结果，针对每个任务，在时间预算内存在忙周期和后备周期。无论SPU的实际处理时间或速度如何，所有spulets被编写成根据此时间预算进行处理。

例如，针对PU的特定SPU，特定任务可以在时间预算4204的忙周期4202期间执行。由于忙周期4202小于时间预算4204，在时间预算期间出现后备周期4206。在此后备周期期间，SPU进入休眠模式，在此

休眠模式期间SPU消耗较少功率。

其它SPU或PU的其它单元不预期任务处理的结果，直到时间预算4204过期。因此，通过使用由绝对定时器建立的时间预算，无论SPU的实际处理速度如何，SPU处理的结果始终是协同的。

在将来，SPU处理的速度将变得更快。然而，由绝对定时器建立的时间预算将保持原样。例如，如图42所示，将来SPU将在较短的周期内执行任务，并且因此将具有较长后备周期。因此，忙周期4208比忙周期4202更短，并且后备周期4210长于后备周期4206。然而，由于程序被编写成根据绝对定时器建立的相同时间预算进行处理，则SPU间处理结果的协同得到维护。结果，更快速的SPU能够处理针对较慢SPU编写的程序，而不会导致处理结果的期待时间的冲突。

在代替用绝对定时器建立SPU之间的协同的一个方式中，PU或一或多个指定SPU可以分析由SPU在处理spulet时执行的特定指令或微码，以解决由增强或不同操作速度产生的SPU并行处理的协同问题。“无操作”(“NOOP”)指令可以被插入到指令中，并且由某些SPU执行以维护由spulet期待的SPU的处理的正确顺序完成。通过插入这些NOOP到指令中，SPU对所有指令的执行的正确定时能够被保持。

图43的系统图例示出了充当虚拟设备的SPU。运行在例如PU处理器的不同处理器上的进程被描述为PU进程4300。虽然在一个实施例中进程4300在PU处理器上运行，但它也可以在象SPU处理器4340的不同SPU处理器上运行。重要的是，运行进程4300的处理器和SPU处理器4340共享公共存储器4310，其中SPU处理器4340能够针对公共存储器保存和检索数据。

在一个使用SPU处理器4340作为虚拟设备的实施例中，进程4300把数据写入缓冲区，在常规系统中该缓冲区被传送到实际设备。在第一发送4315中，例如图形库的进程4300写数据到设备的输入缓冲区(4320)，直到缓冲区满（或几乎满）。设备输入缓冲区4320被存储在公共存储器4310中。公共存储器4310在运行进程4300的处理器和SPU 4340之间共享。

当设备的输入缓冲区满（或几乎满）时，进行第二发送4325以写指令到也存储在公共存储器中的指令块4330。指令块4330指明输入缓冲区的地址，输出缓冲区(如果可用)，和进程正请求对存储在输入缓冲区中的数据执行的设备代码4305的地址。另外，指令块可以包含通知指令，其指示被SPU用来在处理完成时进行通知的方法。如果SPU专用于执行具体设备功能，则当SPU在这种情况下执行相同代码以处理指定输入缓冲区时，设备代码的地址也可以被省略。

在第三发送(4335)中，进程4300通过把指令块4330的地址写入SPU的邮箱(4345)来通知SPU 4340。邮箱能够在FIFO队列中存储多个地址，其中每个地址指向不同的指令块。SPU 4340以FIFO方式从邮箱4345中检索记录。在第四发送4355中，SPU 4340检索对应于存储在邮箱4355中的地址的指令块4330，其中使用DMA命令从公共存储器4310读取指令块并且存储指令块在其本地存储器4350中。所检索的指令块指示输入缓冲区4320的地址和代码地址4305。如果设备代码尚未被载入SPU的本地存储器，则在第五发送4360中，使用DMA命令检索设备代码以从公共存储器4310读取设备代码4305并且在本地存储器位置4365将其存储在SPU的本地存储器4340中。

在第六发送(4370)期间，使用DMA命令从公共存储器4310读取由检索的指令块中的地址指示的输入缓冲区4320，并且在位置4375将其存储在SPU的本地存储器中。如果输入缓冲区太大以致不能被完全读取到分配给输入数据的SPU的本地存储器区中，则通过相继的块取得数据。存储在SPU的本地存储器中的设备代码4365用于处理存储在SPU的本地存储器中的输入数据(4375)，并且在位置4380把结果存储在SPU的本地存储器中。有一个例子是使用SPU作为几何引擎以处理图形命令。当数据已经由SPU处理时，在第七发送(4390)中，几何引擎产生的例如图形原语数据的输出数据被发送到输出设备。输出设备也可以是充当另一虚拟设备的另一SPU，例如硬件渲染引擎，其中SPU 4340建立指示下一个SPU处理输出数据4380所需的设备代码地址和输入代码地址的指令块，并且通过把指令块的地址写入到下一个SPU的邮箱

来通知下一个SPU。输出设备也可以是实际的硬件设备，例如硬件渲染引擎，其中SPU 4340使用DMA写命令把输出数据4380写入到硬件设备。

图44-46示出了载入SPU的本地存储器的各种设备代码模块。用四个不同大小的不同设备代码说明公共存储器4400。在给出的例子中，公共存储器4400包含16千字节(16K)的设备代码4405，32K的设备代码4410，16K的设备代码4415，以及16K的设备代码4420。在图44中，示出的SPU 4430用设备代码4405初始化，其中设备代码4405使用DMA命令读取并且存储在SPU的本地存储器4435中。在示出的例子中，SPU的本地存储器是128K，其中32K被留给输入数据的存储（输入数据区4450），并且另32K被留给结果数据的存储（输出数据区4455）。因此，有64K非被保留并且能够被用于存储设备代码。在第一次加载设备代码4405（DMA读取4425）之后，16K的非保留存储器被分配给加载的设备代码（SPU本地数据区4440）其中48K保持不用（未使用数据区4445）。

在图45中，加载(DMA读取4460)设备代码4410(32K)和设备代码4415(16K)，从而填充SPU 4430中剩余未使用的本地存储器。这里，如果接收到针对设备代码4405，4410或4415的请求，则对应设备代码4440，4465和4470分别可以被立即执行，而不用等待从公共存储器4400加载设备代码。

然而，在图46中，请求SPU 4430执行16K的附加设备代码功能（设备代码4420）。由于SPU本地存储器4435中没有足够的未使用存储器来加载所请求的设备代码，所以预先存储在SPU的本地存储器中的设备代码被覆盖以满足该请求。在示出的例子中，用从公共存储器4600读取的设备代码4420（DMA读取4480）覆盖存储在SPU的本地存储器中的设备代码4440。目前SPU加载了设备代码4485，4465和4470，并且可以依据请求立即执行任何这些设备功能。如果设备代码4405被再次请求，则当前加载的设备代码(4485，4465，或4470)之一将被覆盖以满足请求。

图47的流程图示出了使用适于SPU执行的设备代码对计算机系统的初始化。处理在4700开始，从而在步骤4710，从非易失存储设备4720加载计算机系统的操作系统。在步骤4725，第一设备代码被从非易失存储设备4720加载并且存储在公共存储器中，使得设备代码可以被SPU之一随后检索和加载。

确定SPU是否专用于执行所加载的设备代码（判决4730）。如果SPU将被专用，则判定4730分支到"是"分支4735，由此在步骤4740识别空闲(即可用)SPU。确定可用SPU是否能够被识别(判定4750)。例如，所有SPU可能已经被分配了不同任务。如果可用SPU被识别，则判定4750分支到"是"分支4755，由此所识别的SPU被分配给设备功能。另一方面，如果可用SPU不能被识别，则判定4750分支到"否"分支4765，由此在步骤4765产生指示系统不能将SPU专用于执行功能的错误，并且加入数据结构以在一或多个非专用SPU中管理设备（预定的处理4780，参见图49及对应文本以得到处理细节）。回到判定4730，如果将由非专用SPU执行设备代码，则判定4730分支到"否"分支4775，由此也加入数据结构以在一或多个非专用SPU中管理设备（预定的处理4780，参见图49及对应文本以得到处理细节）。

确定是否存在需要处理的附加设备代码功能（判定4785）。如果存在更多设备代码功能，则判定4785分支到"是"分支4788，由此下一个虚拟设备的代码在步骤4790被从非易失存储器4720读取，并且处理循环返回以处理新读取的设备代码。此循环继续执行，直到不再有更多需要处理的设备代码功能为止，在此判定4785分支到"否"分支4792，并且在4795初始化处理结束。

图48的流程图示出了SPU在管理多个设备代码文件时采取的步骤。处理在4800开始,由此在步骤4810，由SPU接收请求（即，通过将指令块的地址通知SPU的邮箱）。确定设备代码是否已经被加载在SPU的本地存储器中（判定4820）。如果设备代码未加载在SPU的本地存储器中，则判定4820分支到"否"分支4825，由此另外确定在SPU的本地存储器中是否存在足够空闲（即，未分配）空间来加载设备代码（判

定4830）。如果存在足够空闲空间，则判定4830分支到"是"分支4835，由此在步骤4840，设备代码被载入SPU的本地存储器中的空闲空间（即，使用DMA读命令）。 另一方面，如果不存在可用于设备代码的足够空闲空间，则判定4830分支到"否"分支4845，由此在步骤4850，加载所请求的设备代码（即，使用DMA读命令）并且覆盖预先加载在SPU中的设备代码。 一旦设备代码被加载，在步骤4870执行代码以处理请求。返回到判定4820，如果设备代码已经在SPU的本地存储器中，则判定4820分支到"是"分支4860，并且在步骤4870执行代码。其后在4895处理结束。

图49的图例示出了用于管理可由多个SPU之一执行的虚拟设备的数据结构。共享公共存储器4900包含在SPU上执行的针对不同设备功能的设备代码（设备代码4905，4910和4915）。数据结构4920被初始化以管理设备。针对每个设备建立数据结构（数据结构4930，4950和4970分别对应于设备代码4905，4910和4915）。所有这些数据结构均包含任务队列和锁定结构（任务队列4935，4955和4975分别对应于设备代码4905，4910和4915，并且锁定结构4940，4960和4980分别对应于设备代码4905，4910和4915）。请求被存储在指定设备的任务队列中。例如，如果进程正请求第一设备代码(4905)，则指令块的地址被写入已经建立以管理第一设备代码的任务队列（任务队列4935）。锁定结构包含指示已经获得锁并因此正在执行设备代码的SPU的SPU标识符（SPU标识符4945，4965和4985分别对应于设备代码4905，4910和4915）。周期性地，当SPU没有可执行的设备代码任务时，SPU检查各个数据结构以确定是否存在任何已经请求但未分配SPU的设备代码。当SPU识别这种数据结构时，SPU通过把其标识符写入对应锁定结构来获得锁，并且处理存储在任务队列中的等待请求。当所有请求已经处理时，SPU有空释放锁并且搜索已经请求但未分配SPU的另一设备代码。

图50的流程图示出了SPU之一执行的处理在调用虚拟设备时所采取的步骤。当运行在PU或SPU之一上的进程需要调用虚拟设备时，

执行图50中的步骤。运行在PU或SPU上的实际应用程序可以实际调用包含在例如图形库的库中的API，其中库API代码实际调用一个SPU上加载的虚拟设备。

处理在5000开始，由此在步骤5010接收设备请求(即，通过库API代码）。在步骤5020，要处理的输入数据被载入位于公共（共享)存储器中的输入缓冲区。在步骤5030，初始化输出缓冲区（如果有）。对于某些虚拟设备，返回数据，而对于其它设备请求，只返回一个返回代码。例如，如果虚拟设备是其输出被发送到硬件渲染引擎的几何引擎，输出缓冲区可能不需要，或可能只被用于存储返回代码或错误值。在步骤5040，指令块被写入共享存储器，以指示输入缓冲区地址，输出缓冲区（如果有）的地址，设备代码地址，通知指令（例如回写地址），以及任何其他执行设备请求所需的参数数据。

确定所请求的设备代码是否由专用SPU执行（判定5050）。如果设备代码由专用SPU执行，则判定5050分支到"是"分支5055，由此在步骤5060，指令块的地址被写入专用SPU的邮箱。另一方面，如果设备代码不由专用SPU执行，则判定5050分支到"否"分支5065，由此在步骤5070，指令块的地址被写入设备任务队列数据结构，使得非专用SPU将找到请求并且执行所请求的设备代码。

在请求已经进行之后，通过SPU的邮箱或设备的任务队列，处理等待指示SPU已经完成所请求处理的完成通知（步骤5080）。在步骤5090，读取输出缓冲区或回写地址，并且结果得到相应处理（即，错误处理(如果错误出现)，从虚拟设备产生的数据的进一步使用或处理等）。 其后在5095处理结束。

图51的流程图示出了由非专用SPU在识别和执行请求的虚拟设备任务时采取的步骤。处理在5100开始，由此非专用SPU获得数据结构的针对具有任务队列项的第一可用（即，仍未分配）设备的锁（步骤5105）。在步骤5110，读取所获得任务队列中的第一队列项。读取的任务队列项指示在步骤5115读取的指令块的地址，因而提供设备代码地址，输入缓冲区地址，输出缓冲区（如果有）地址，通知指令（如

果有），以及任何执行设备请求所需的附加参数。确定设备代码是否已经被加载在SPU的本地存储器中（判定5120）。如果设备代码还未被加载在SPU的本地存储器中，则判定5120分支到"否"分支5122，由此使用DMA命令（步骤5125）从共享存储器读取设备代码到SPU本地存储器5130，导致设备代码5135被存储在本地存储器中。另一方面，如果设备代码已经加载在SPU的本地存储器中，则判定5120分支到"是"分支5128，从而略过步骤5125。

使用DMA命令（步骤5140）从共享存储器读取位于输入缓冲区中的数据并且将其存储在SPU的本地存储器中，导致输入数据5145被存储在SPU的本地存储器5130中。设备代码被执行(步骤5150)，并且代码的结果被写入到存储在SPU本地存储器5130中的输出数据区5155中。如果输入数据或输出数据对于SPU本地存储器来说太大，则输入数据可以分块读取，存储在SPU本地存储器中并且加以处理。另外，可以写入输出数据，直到输出数据区满，并且接着可以间歇地把输出数据写入输出缓冲区（即，共享存储器中的缓冲区空间，或发送到实际硬件设备）。

确定所输入数据是否由设备代码完成处理（判定5160）。如果未完成输入数据处理，则判定5160分支到"否"分支5162，从而循环返回并且继续处理输入数据。此循环继续执行，直到完成输入数据处理，在此判定5160分支到"是"分支5164。

在步骤5165，结果（存储在SPU的本地存储器内的位置5155上）被写入到输出缓冲区位置，该位置可以是存储在共享存储器中的输出缓冲区（例如缓冲区5170），或可以是实际硬件设备，例如硬件渲染引擎。确定是否存在针对任务队列的更多请求，其中该任务队列的锁正由SPU保持（判定5175）。如果存在更多排在任务队列中的请求，则判定5175分支到"是"分支5178，由此所获得任务队列中的下一个队列项被读取(步骤5180)，并且处理循环返回以处理下一个队列项。此循环继续执行，直到任务队列中不存在更多队列项（即，指示现在没有请求设备的进程），在此判定5175分支到"否"分支5185，由此对应

于任务队列的锁被释放，并且SPU寻找另一个具有等待队列项但尚未由被一个SPU获得的设备任务队列。

图52的流程图示出了由专用SPU在执行请求的虚拟设备任务时采取的步骤。处理在5200开始，由此在步骤5205，检索指示指令块地址的请求。在一个实施例中，指令块的请求被写入到专用任务队列数据结构（参见图49），而在另一个实施例中，指令块地址被写入到专用SPU的邮箱。在步骤5210读取指令块，因而提供设备代码地址，输入缓冲区地址，输出缓冲区（如果有）地址，通知指令（如果有），以及任何执行设备请求所需的附加参数。确定设备代码是否已经被加载在SPU的本地存储器中（判定5215）。如果设备代码还未被加载在SPU的本地存储器中，则判定5215分支到"否"分支5218，由此使用DMA命令（步骤5220）从共享存储器读取设备代码到SPU本地存储器5230，导致设备代码5235被存储在本地存储器中。另一方面，如果设备代码已经加载在SPU的本地存储器中，则判定5215分支到"是"分支5238，从而略过步骤5220。

使用DMA命令（步骤5240）从共享存储器读取位于输入缓冲区的数据并且将其存储在SPU的本地存储器中，导致输入数据5245被存储在SPU本地存储器5230中。设备代码被执行(步骤5250)，并且代码的结果被写入到存储在SPU本地存储器5230中的输出数据区域5255中。如果输入数据或输出数据对于SPU本地存储器来说太大，则输入数据可以分块读取，存储在SPU本地存储器中并且加以处理。另外，可以写入输出数据，直到输出数据区满，并且接着可以间歇地把输出数据写入输出缓冲区（即，共享存储器中的缓冲区空间，或发送到实际硬件设备）。

确定所输入数据是否由设备代码完成处理（判定5260）。如果未完成输入数据处理，则判定5260分支到"否"分支5262，其循环返回并且继续处理输入数据。此循环继续执行，直到完成输入数据处理，在此判定5260分支到"是"分支5264。

在步骤5265，结果（存储在SPU的本地存储器内的位置5255上）

被写入输出缓冲区位置，该位置可以是存储在共享存储器中的输出缓冲区（例如缓冲区5270），或可以是实际硬件设备，例如硬件渲染引擎。确定是否存在针对虚拟设备的更多请求(判定5275)。如果存在更多请求，判定5275分支到"是"分支5278，由此处理循环返回以处理请求。如果不存在附加的排队请求，则判定5275分支到"否"分支5285，由此SPU进入低功率状态，并且等待写入到SPU的邮箱的新请求（步骤5290）。

图53的图例示出了用于帮助SPU对虚拟设备任务的处理的任务队列管理器。应用程序5300请求经常被库，例如API库5305中的API执行的功能。这些功能可以包含设备指令和请求。库中的API可以被编程以发送请求到物理设备5310或正执行设备代码的SPU，例如用于图形应用的几何引擎。当SPU执行功能时，请求被发送到代表请求方应用程序和API提供服务的任务队列管理器5315。这些服务包含把所请求任务送入适当队列(进程5320)，以及发送请求到已经识别的SPU(进程5325)。任务队列管理器也把完成通知发送回到请求方API/应用程序。

在送入任务时，任务队列管理器写入指令块5330，其包含所请求设备代码的地址，输入和输出缓冲区的地址，通知指令(如果需要)和任何执行所请求设备代码所需的参数。另外，指令块的地址被写入到FIFO任务队列5335，使得请求将被所识别的SPU记录和处理。

在识别执行请求的SPU时，检查任务队列和设备历史记录以确定SPU现在是否正执行设备代码，并且如果SPU现在没有正执行设备代码，则基于设备历史数据5340选择最近执行代码，并且因此仍然在SPU的本地存储器中具有可用代码的拷贝的SPU。

SPU 5360包含若干SPU，其中每个SPU具有本地存储器和邮箱。另外，每个SPU能够使用DMA命令针对公共（共享）存储器5328写/读数据。在示出的例子中，SPU包含SPU 5370，5380和5390。所有这些SPU分别具有本地存储器5372，5382和5392。所有这些SPU也分别具有邮箱5376，5386和5396。当SPU接收请求时，SPU检索具有涉及

请求的细节信息的对应指令块5330。SPU也检索设备代码5345，输入缓冲区数据5350，以及输出缓冲区地址5355(可选)。SPU使用DMA命令从指令块和输入缓冲区读取数据，并且也使用DMA命令把数据写入输出缓冲区（或另一个SPU或物理设备）。

图54的流程图示出了由任务队列管理器在帮助SPU处理设备任务时采取的步骤。任务队列管理器和处理在5400开始。任务队列管理器可以作为PU进程或作为SPU进程执行。

任务队列管理器通过API库5418中包含的API接收来自应用程序的请求（预定进程5410，参见图55和对应文本以得到处理细节)。这种API库的例子是用于执行图形功能的图形库。在步骤5420，建立输出缓冲区（或回写地址）以检索数据，或由设备代码处理产生的返回代码，如果应用程序（即，API）尚未提供输出缓冲区的话。

在步骤5425，用SPU处理请求所需的数据创建任务数据分组（即，信息块），所述数据例如是设备代码地址，输入缓冲区地址，输出缓冲区地址(如果需要)，通知指令(例如回写地址)，以及任何执行设备代码请求可能需要的附加参数。通过把创建的信息块的地址写入对应于所请求设备代码的任务队列，把请求加入任务队列。

任务队列管理器确定SPU之一现在是否被分配给所请求设备任务(判定5435)。如果SPU现在未被分配给所请求任务，则判定5435分支到"否"分支5440，由此在步骤5445，任务队列管理器分析设备历史数据以及现有的任务队列。基于此分析，在步骤5450，任务队列管理器识别最不繁忙和最近执行所请求设备代码的SPU。分析的最不繁忙方面将倾向于当前未指定给具体设备代码的SPU，而分析的最近执行方面倾向于那些在SPU的本地存储器中可能仍然具有可用的所请求设备代码的SPU。在步骤5455，当SPU之一已经被任务队列管理器识别时，设备代码的任务队列被分配给所识别的SPU。在步骤5460，历史数据被更新以反映分配，使得在后续分析期间，将知道所识别的SPU曾经把设备代码加载到SPU的本地存储器。

返回到判定5435，如果SPU之一现在被分配给(即，执行)设备代

码，则判定5435分支到"是"分支5465，从而略过步骤5445-5460。

在步骤5470，通过把在步骤5425准备的指令块的地址写入到邮箱，通知已经分配给设备代码任务的SPU的邮箱。在一个实现中，每个SPU具有容纳四个项的有限邮箱容量。在此实现中，任务队列管理器查询所分配的SPU的邮箱以保证SPU的邮箱中有空间。如果没有空间，则任务队列管理器对请求进行排队，并且周期性查询SPU的邮箱，由此仅当有空位可用时，请求才被加入邮箱。

确定是否存在更多需要任务队列管理器处理的请求(判定5475)。如果存在附加请求，则判定5475分支到"是"分支5278，从而循环返回以处理下一个请求。另一方面，当没有更多请求(即，系统关机)时，判定5475分支到"否"分支5485，由此任务队列管理器的处理在5495结束。

图55的流程图示出了任务队列管理器将预先请求的设备请求通知应用程序。此流程图示出了图54中示出的预定处理5410内出现的处理的细节。

处理在5500开始，由此分析从应用/API接收的请求(步骤5510)。确定应用/API是否提供用于当请求已经完成时通知应用/API的数据结构的地址（判定5520）。如果应用/API没有提供数据结构，则判定5520分支到"否"分支5525，由此创建用于存储完成信息的数据结构(步骤5530)，并且在步骤5535把数据结构的地址返回给应用/API。另一方面，如果应用/API提供用于返回数据的数据结构，则判定5520分支到"是"分支5545，从而略过步骤5530和5535。

在步骤5550，数据结构与发送到SPU的请求相关联。在步骤5555，任务管理器从执行请求的SPU 5560接收响应。在一个实施例中，SPU把地址写入到队列管理器的邮箱(5565)，在另一个实施例中，SPU把地址回写入包含有被任务管理器用来管理虚拟设备的数据结构的回写队列。无论如何，在步骤5555，任务管理器从SPU接收完成通知。在步骤5570，通过读取请求数据结构5575来识别与原始请求相关的输出数据结构。在步骤5580，从SPU接收的完成数据被写入输出数据结构。

输出数据结构在步骤5590被解锁（即，通知等待锁或信号灯的应用/API），使得应用/API 5540从适当数据结构接收结果数据。接着在5595处理返回到调用例程。

图56的流程图示出了由任务队列管理器管理的SPU采取的步骤。SPU处理在5600开始，由此在步骤5610，SPU从队列管理器接收写入到SPU邮箱的邮箱请求（5615）。

SPU的邮箱的第一项在步骤5620被读取。此项是位于共享存储器的指令块的地址。SPU通过使用DMA命令读取指令块，以检索所识别的指令块(步骤5625)。指令块指示SPU正被请求处理的代码的代码地址，输入和输出缓冲区的地址，通知指令(即，回写地址)，和执行请求所需要的任何附加参数。

确定指令块中标识的代码是否已经加载在SPU的本地存储器中（判定5630）。如果代码现在未加载在SPU的本地存储器中，则判定5630分支到"否"分支5632，由此使用DMA命令从共享存储器读取代码并且把代码存储在SPU的本地存储器中。另一方面，如果代码已经在SPU的本地存储器中，则判定5630分支到"是"分支5638，从而略过步骤5635。

使用DMA命令（步骤5640）从共享存储器读取位于输入缓冲区的数据并且将数据存储在SPU的本地存储器中，从而导致输入数据5660被存储在SPU本地存储器5650中。设备代码被执行(步骤5645)，并且代码的结果被写入到存储在SPU本地存储器5650中的输出数据区5665中。如果输入数据或输出数据对于SPU本地存储器来说太大，则输入数据可以分块读取，存储在SPU本地存储器中并且加以处理。另外，可以写入输出数据，直到输出数据区满，并且接着可以间歇地把输出数据写入输出缓冲区（即，共享存储器中的缓冲区空间，或发送到实际硬件设备）。

确定所输入数据是否由设备代码完成处理（判定5670）。如果未完成输入数据处理，则判定5670分支到"否"分支5672，从而循环返回并且继续处理输入数据。此循环继续执行，直到完成输入数据处理，

在此判定5670分支到"是"分支5674。

在步骤5675，结果（存储在SPU的本地存储器内的位置5665）被写入输出缓冲区位置，该位置可以是存储在共享存储器中的输出缓冲区（例如缓冲区5270），或可以是实际硬件设备，例如硬件渲染引擎。确定是否有更多的请求在SPU的邮箱中等待(判定5685)。如果在SPU的邮箱中有更多的请求，则判定5685分支到"是"分支5690，由此SPU的邮箱中的下一项（即，地址）被读取，并且处理循环返回以处理请求。此循环继续执行，直到邮箱中没有其它项，在此判定5685分支到"否"分支5695，由此SPU进入低功率状态并且等待写入到SPU的邮箱的新请求（步骤5698）。

图57的模块图图解了具有共享系统存储器的主处理器和多个辅助处理器的处理单元。处理器单元(PE) 5705包含在一个实施例中充当主处理器并且运行操作系统的处理单元(PU) 5710。例如，处理单元5710可以是执行Linux操作系统的Power PC核心。PE 5705还包含多个协处理结构(SPC)，例如SPC 5745，5765和5785。SPC包含充当PU 5710的辅助处理单元的协处理单元(SPU)，存储器存储单元，以及本地存储器。例如，SPC 5745包含SPU 5760，MMU 5755和本地存储器5759；SPC 5765包含SPU 5770，MMU 5775和本地存储器5779；以及SPC 5785包含SPU 5790，MMU 5795和本地存储器5799。

每个SPC可以被配置成用于执行不同任务，并且因此，在一个实施例中，每个SPC可以使用不同指令集访问。例如，如果PE 5705被用于无线通信系统中，则每个SPC可负责分立的处理任务，例如调制，码片速率处理，编码，网络接口等。在另一个实施例中，SPC可以具有相同指令集，并且可以彼此并行使用以执行因并行处理而获益的操作。

PE 5705也可以包含2级高速缓存，例如L2高速缓存5715，以用于PU 5710。另外，PE 5705包含在PU 5710和SPU之间共享的系统存储器5720。系统存储器5720可以存储例如正在运行的操作系统(包含内核)，设备驱动程序，I/O配置等的映像，执行的应用程序，以及其它数据。

系统存储器5720包含映射到系统存储器5720的区域的一或多个SPC的本地存储器单元。例如，本地存储器5759可以映射到被映射区域5735，本地存储器5779可以映射到被映射区域5740，并且本地存储器5799可以映射到被映射区域5742。通过总线5717，PU 5710和SPC互相以及与系统存储器5720通信，该总线被构造成用于在这些设备之间传递数据。

MMU负责在SPU的本地存储器和系统存储器之间传送数据。在一个实施例中，MMU包含被构造成用于执行此功能的直接存储器访问(DMA)控制器。PU 5710可以编程MMU以控制每个MMU可使用哪个存储器区域。通过改变每个MMU可用的映射，PU可以控制哪个SPU访问系统存储器5720的哪个区域。 通过这种方式，例如，PU可以指定系统存储器的区域专门为特定SPU所独享。在一个实施例中，可以通过PU 5710以及使用存储器映射的其它SPU访问SPU的本地存储器。在一个实施例中，PU 5710管理所有SPU的公共系统存储器5720的存储器映射。存储器映射表可以包含PU 5710的L2高速缓存5715，系统存储器5720，以及SPU 的共享本地存储器。

在一个实施例中，SPU在PU 5710的控制下处理数据。例如，SPU可以是数字信号处理核心，微处理器核心，微控制器核心等，或是上述核心的组合。每一个本地存储器是与特定SPU相关的存储区。在一个实施例中，每个SPU能够将其本地存储器配置为专有存储区，共享存储区，或SPU可以将其本地存储器配置为部分专有和部分共享的存储器。

例如，如果SPU需要大量的本地存储器，则SPU可以将其本地存储器100%分配给仅可由该SPU访问的专有存储器。另一方面，如果SPU需要最小量的本地存储器，则SPU可以将其10%的本地存储器分配给专有存储器，而剩余的90%分配给共享存储器。可由PU 5710和其它SPU访问共享存储器。为了使SPU在执行需要快速访问的任务时具有快速有保证的存储器访问，SPU可以保留其部分本地存储器。SPU也可以在处理敏感数据时，例如在SPU执行加密/解密时保留某些其本地存储器为专有的。

尽管这里参照具体实施例描述了本发明，但是应当理解，这些实施例仅仅是本发明的原理和应用的说明。因此应当理解，在不偏离根据所附权利要求书的限定的本发明的宗旨和范围的前提下，可以对说明性的实施例进行许多修改，并且可以得出其它方案。

本发明的一个优选实现是应用程序，即例如可以驻留在计算机的随机访问存储器中的代码模块的一组指令 (程序代码)。在计算机需要之前，该指令组可以被存储在另一个计算机存储器中，例如存储在硬盘驱动器中，或存储在例如光盘 (最终在 CD ROM中使用)或软盘最终在软盘驱动器中使用)的可移动存储器中，或者经由因特网或其它计算机网络下载。由此，本发明可以被实现成用于计算机的计算机程序产品。另外，虽然描述的各种方法被方便地实现在通过软件有选择地启动或重新配置的通用计算机中，然而本领域的普通技术人员也会认识到，这种方法可以在硬件，固件或被构造成用于执行所需方法步骤的更加专用的设备中执行。

虽然已经示出和描述了本发明的特定实施例，然而本领域的技术人员显然明白，根据这里的教导，可以在不偏离本发明及其更宽的方面的情况下进行改变和修改，因此所附权利要求书将把所有这种在本发明的真实实质和范围内的的改变和修改包括在其范围内。此外应当理解，本发明完全由所附权利要求来限定。本领域技术人员会理解，如果旨在所引入权利要求元素的具体数量，则这种意图会在权利要求中明确表述，并且在没有这种表述的情况下，则不存在这种限制。对于非限制性例子，为帮助理解，以下所附权利要求包含使用介绍性短语"至少一个"和"一或多个"来介绍权利要求元素。然而,这种短语的使用不应该被解释为意味着通过不定冠词" a "或" an "引入的权利要求元素，将包含这种引入的权利要求元素的任何特定权利要求限制于仅包含一个这种元素的发明，即使在相同权利要求包含引入性短语"一或多个"或"至少一个"和例如" a "或" an "的不定冠词时；相同道理也适用于在权利要求中使用定冠词。

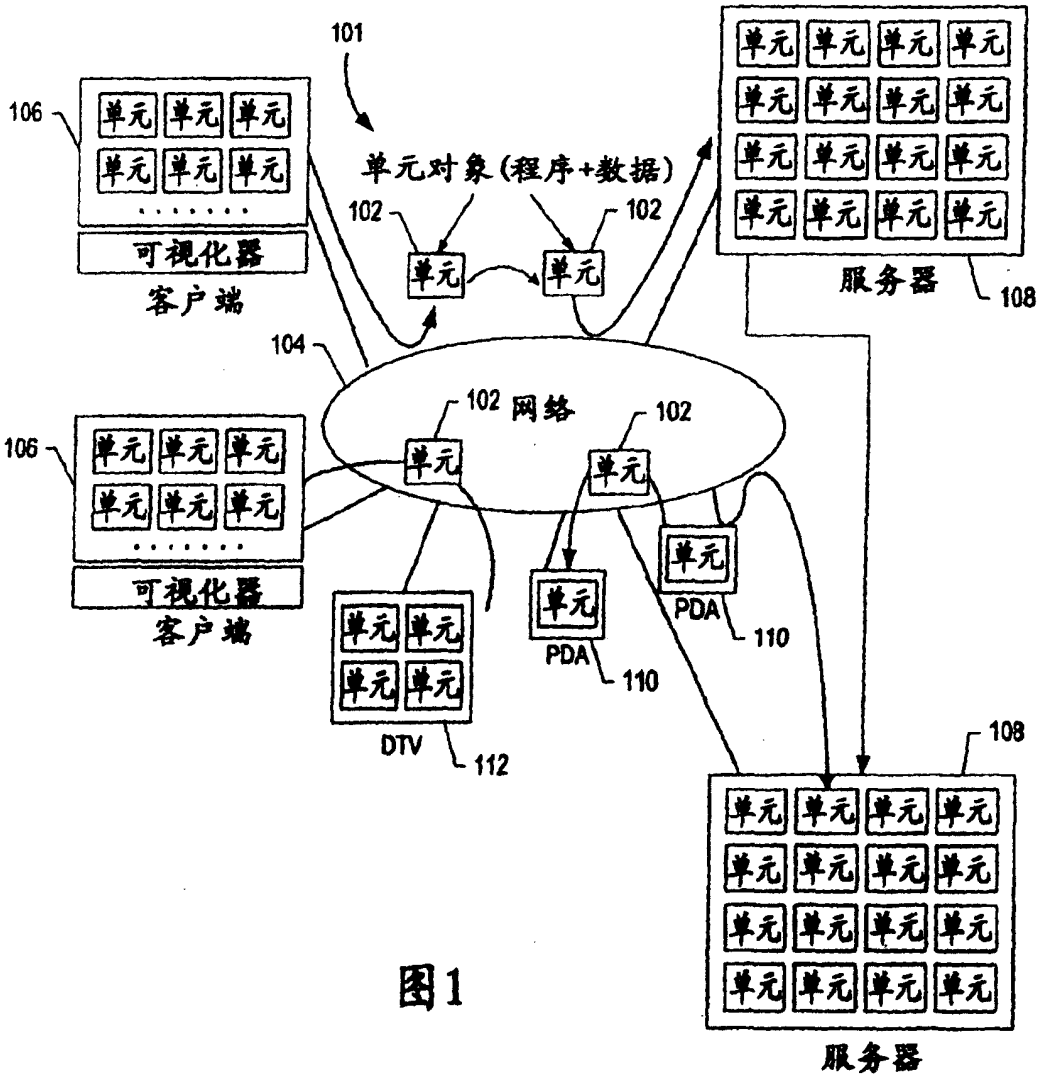

101
单元对象(程序+数据)
102
102
106
单元 单元 单元
单元 单元 单元
可视化器
客户端
104
102 网络
102
106
单元 单元 单元
单元 单元 单元
可视化器
客户端
单元 单元
单元 单元
DTV
112
单元
PDA
110
单元
PDA
110
单元 单元 单元 单元
单元 单元 单元 单元
单元 单元 单元 单元
单元 单元 单元 单元
服务器
108
单元 单元 单元 单元
单元 单元 单元 单元
单元 单元 单元 单元
单元 单元 单元 单元
服务器
108

图1

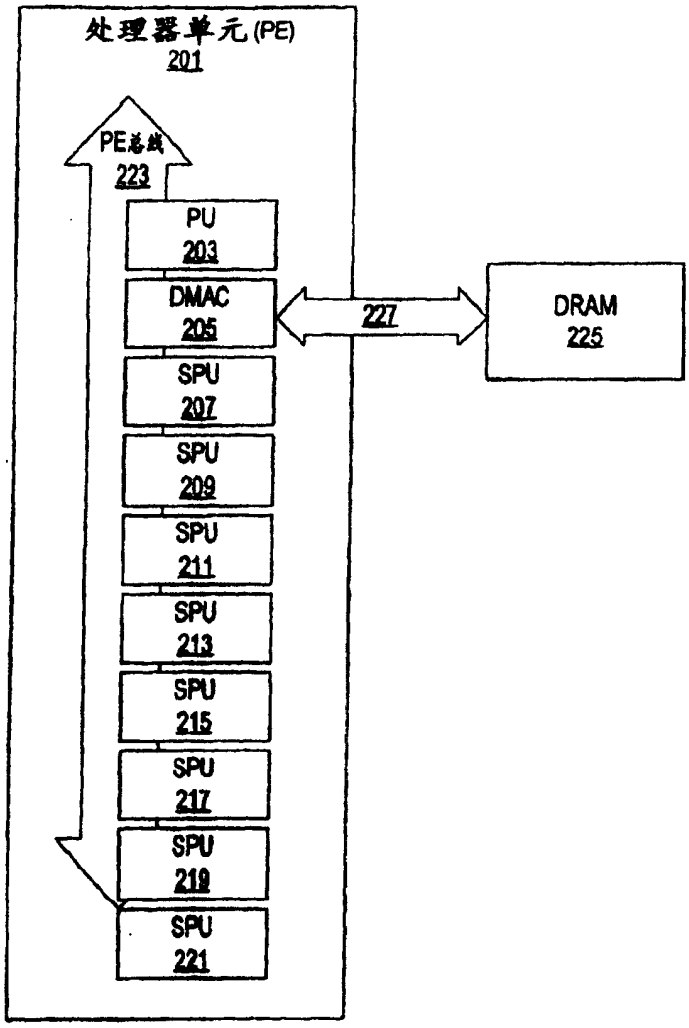

处理器单元(PE)
201
PE总线
223
PU
203
DMAC
205
227
DRAM
225
SPU
207
SPU
209
SPU
211
SPU
213
SPU
215
SPU
217
SPU
219
SPU
221

图2

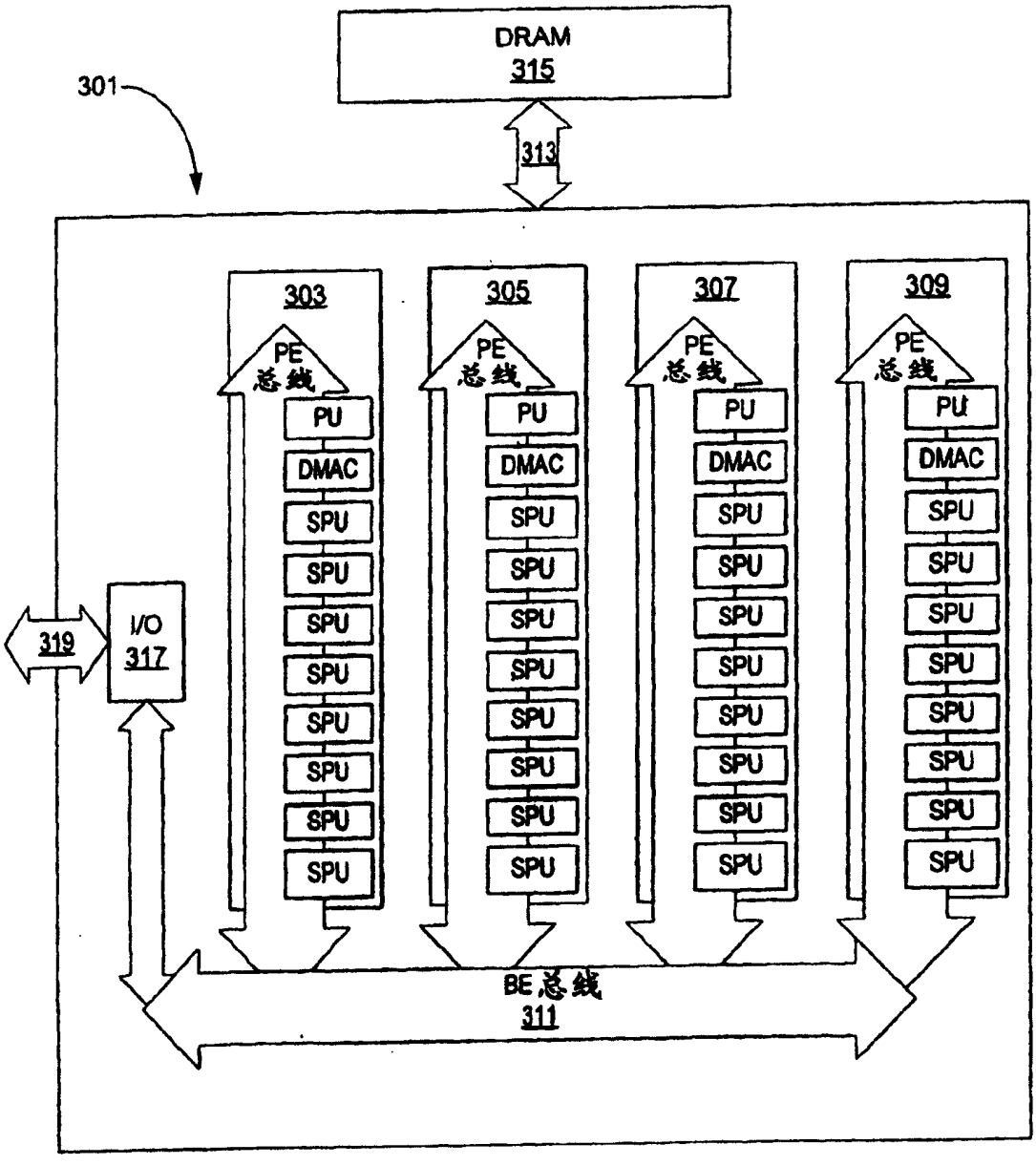

DRAM
315
301
313
303
305
307
309
PE
总线
PU
DMAC
SPU
I/O
317
319
BE总线
311

图3

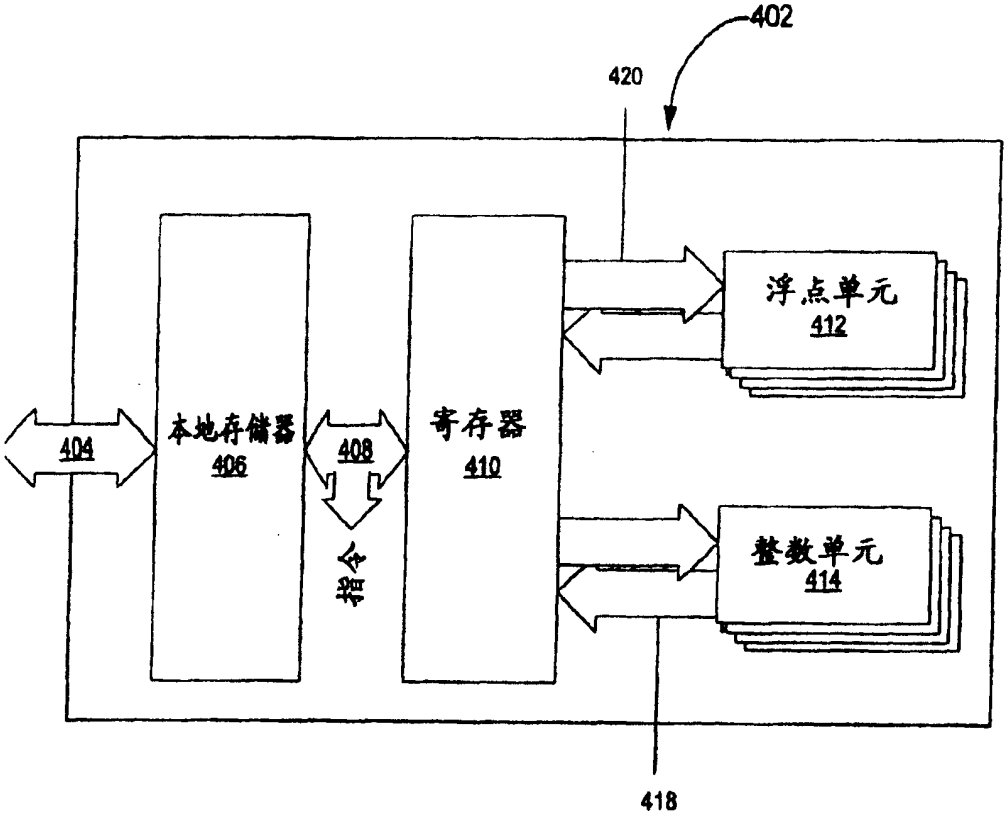


图4

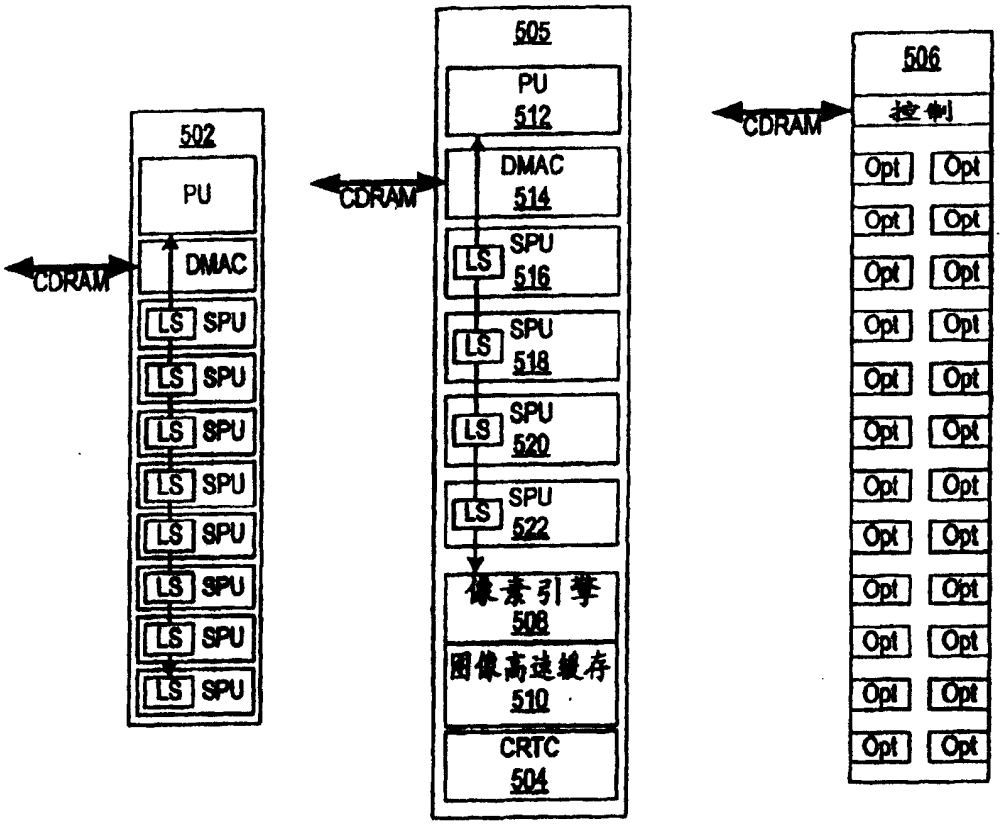

502
PU
DMAC
CDRAM
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
505
PU
512
DMAC
514
CDRAM
LS SPU
516
LS SPU
518
LS SPU
520
LS SPU
522
像素引擎
508
图像高速缓存
510
CRTC
504
506
控制
CDRAM
Opt

图5

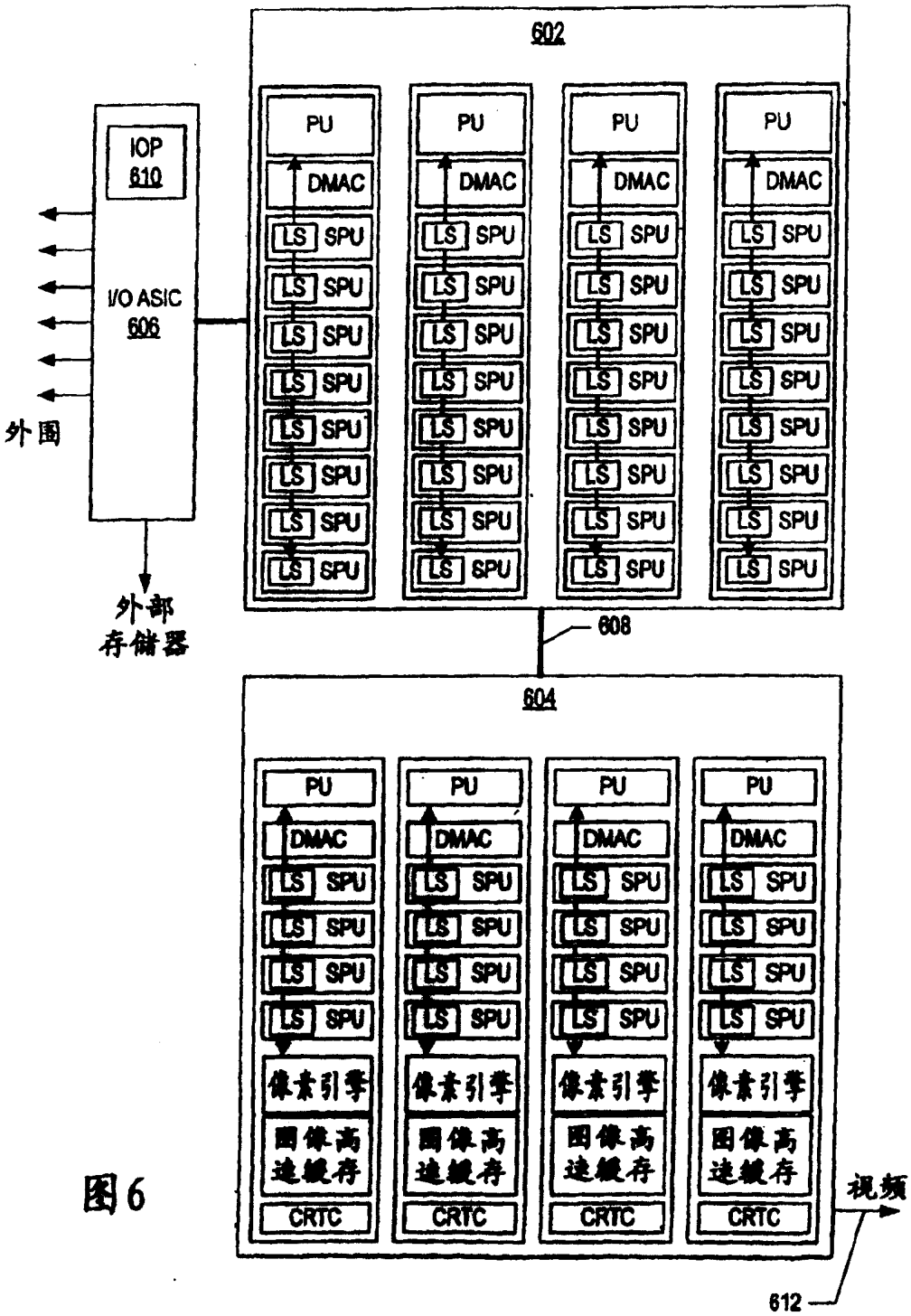

602
IOP
610
I/O ASIC
606
外围
外部
存储器
PU
DMAC
LS
SPU
608
604
像素引擎
图像高速缓存
CRTC
视频
612

图6

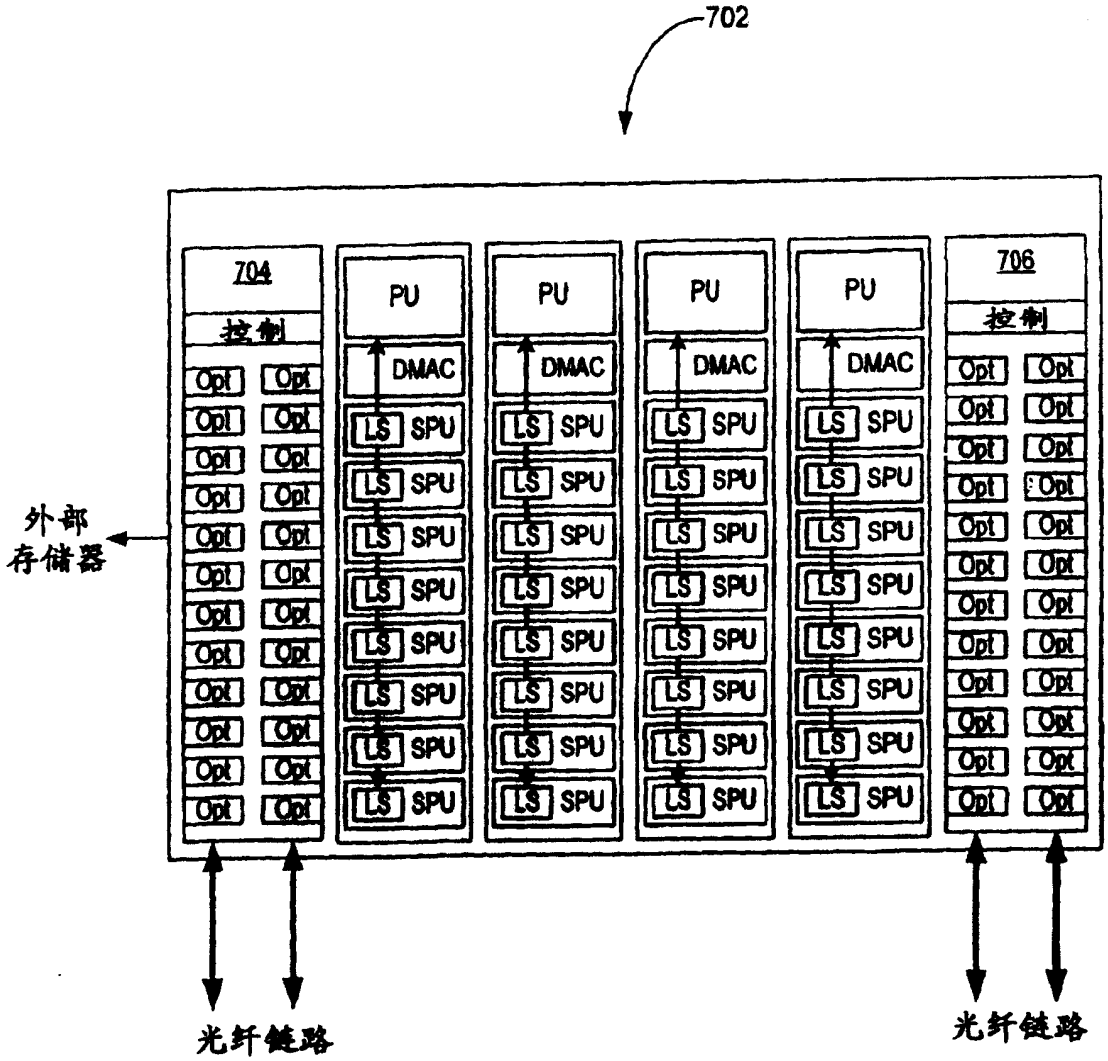


图7

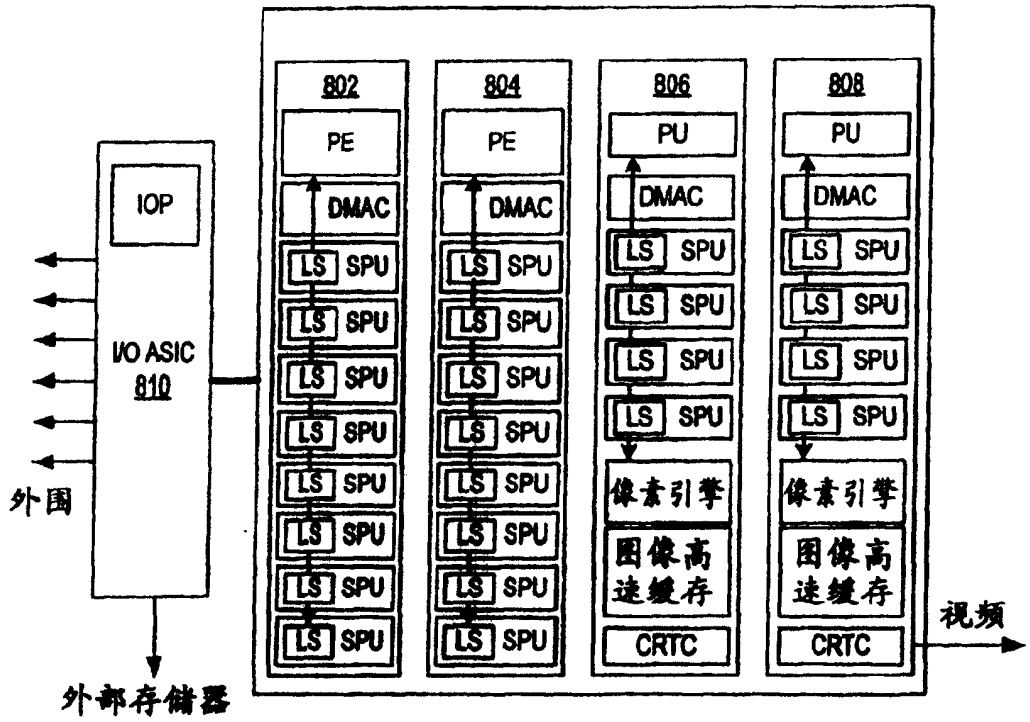

802
804
806
808
PE
PE
PU
PU
IOP
DMAC
DMAC
DMAC
DMAC
LS SPU
I/O ASIC
810
像素引擎
像素引擎
图像高速缓存
图像高速缓存
CRTC
CRTC
外围
外部存储器
视频

图8

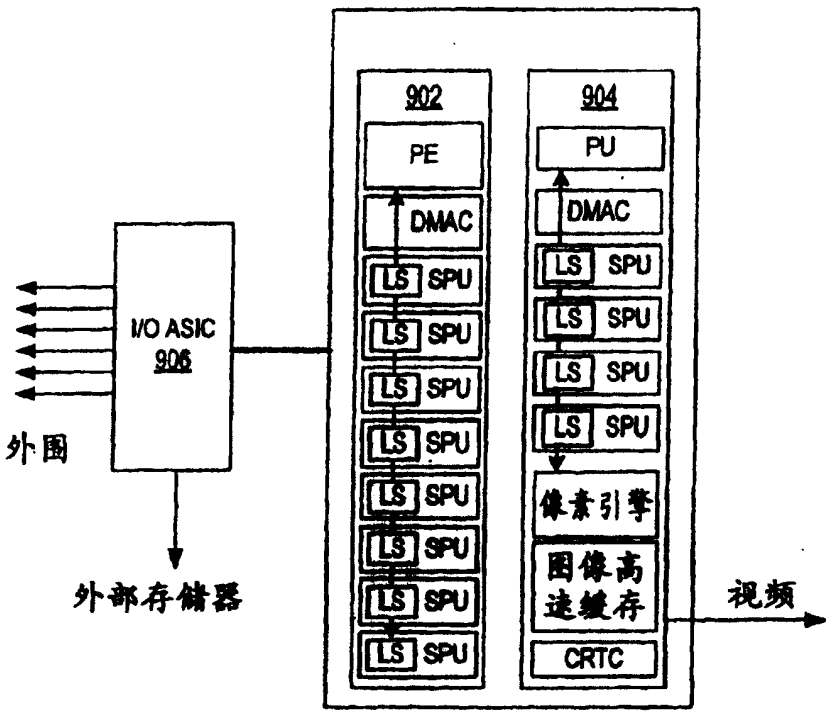

902
PE
DMAC
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
LS SPU
904
PU
DMAC
LS SPU
LS SPU
LS SPU
LS SPU
像素引擎
图像高速缓存
CRTC
视频
I/O ASIC
906
外围
外部存储器

图9

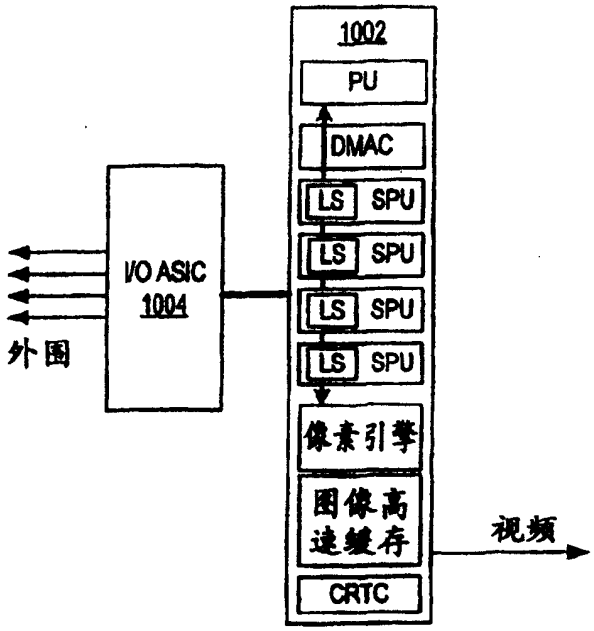

1002
PU
DMAC
LS SPU
LS SPU
LS SPU
LS SPU
像素引擎
图像高速缓存
CRTC
I/O ASIC
1004
外围
视频

图10

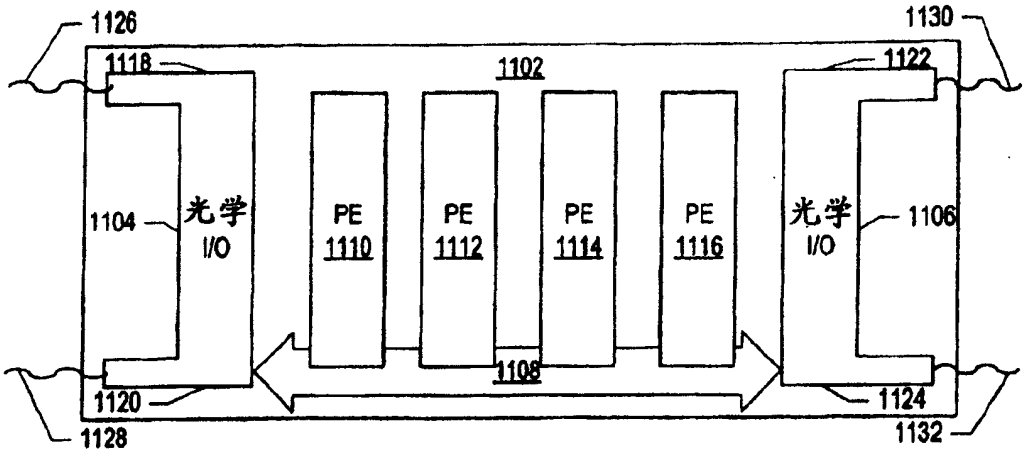


图11A

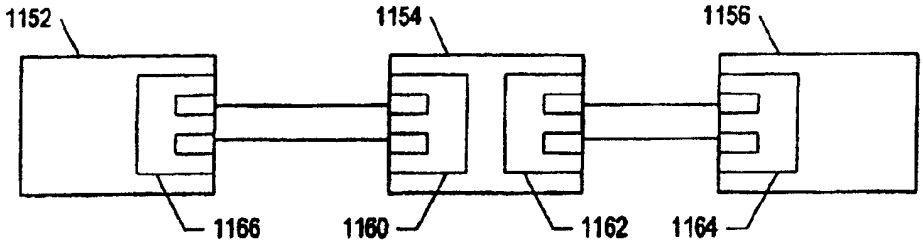


图11B

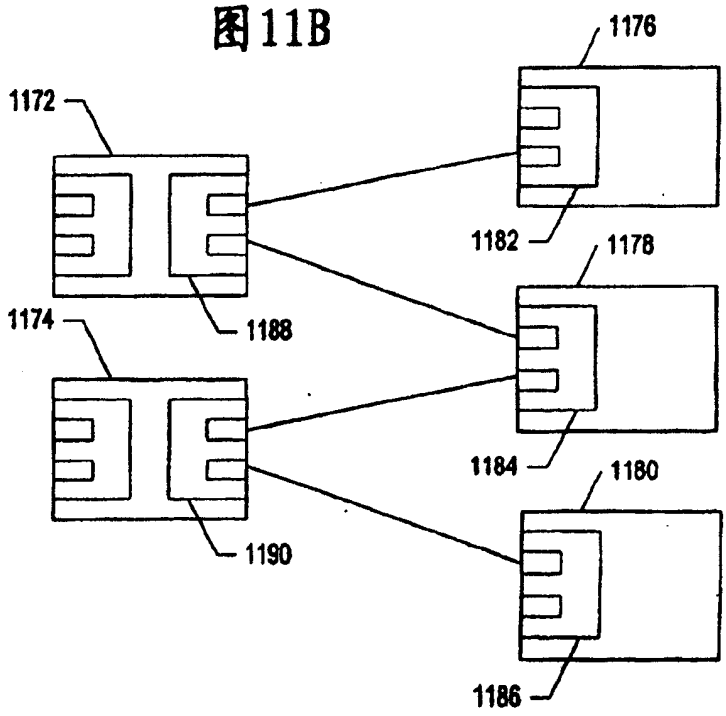


图11C

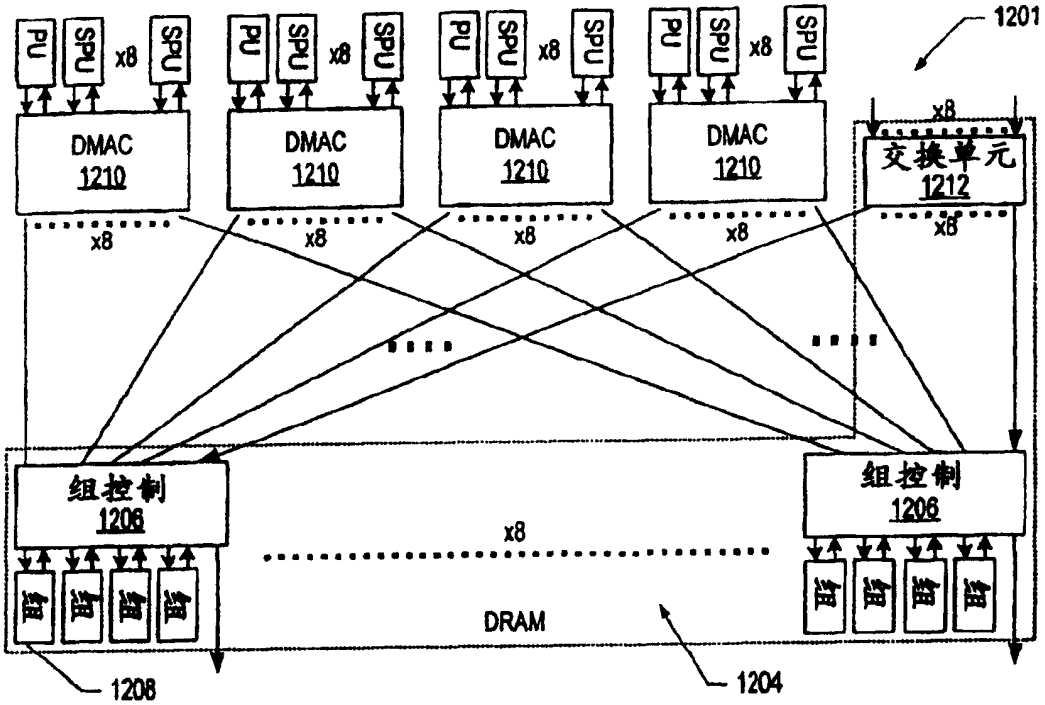

1201
PU
SPU
x8
SPU
DMAC
1210
交换单元
1212
组控制
1206
DRAM
1208
1204

图12A

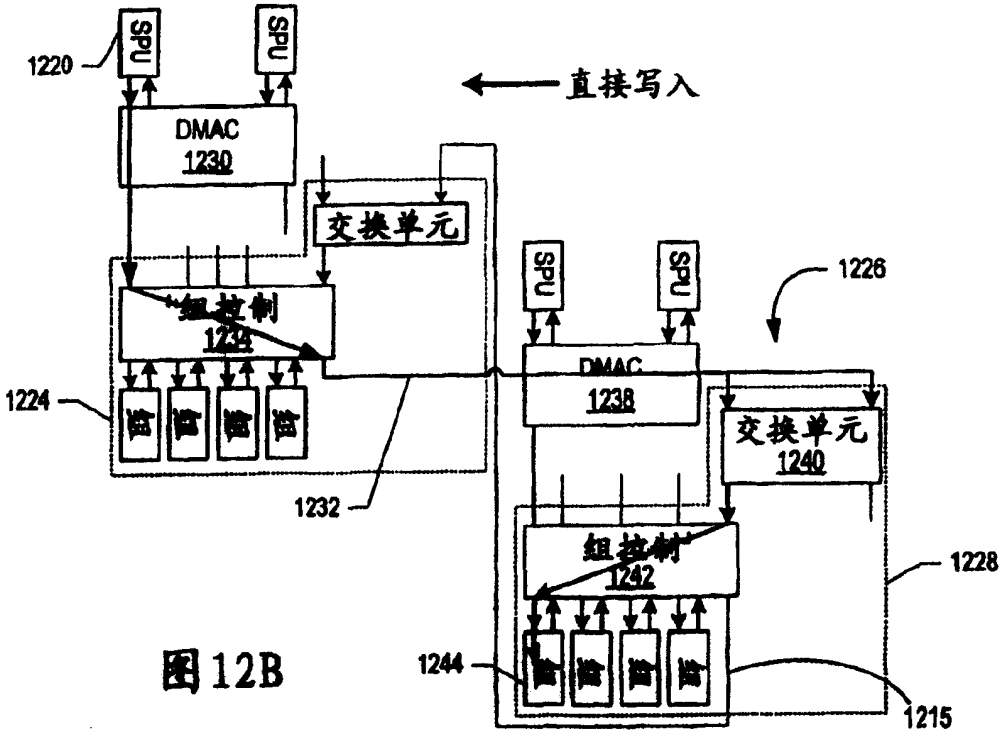

1220
SPU
直接写入
DMAC
1230
交换单元
组控制
1234
1224
1232
1226
DMAC
1238
交换单元
1240
组控制
1242
1228
1244
1215

图12B

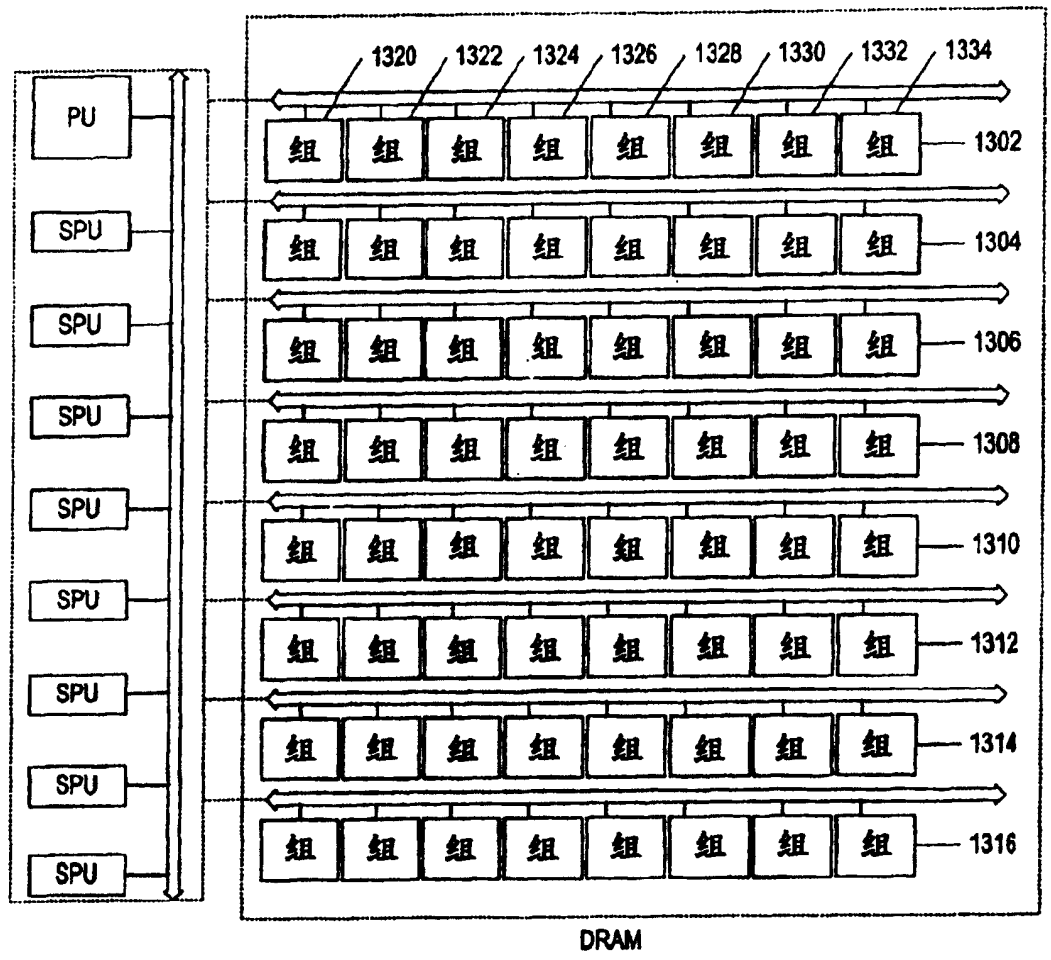

1320
1322
1324
1326
1328
1330
1332
1334
PU
SPU
SPU
SPU
SPU
SPU
SPU
SPU
SPU
组
1302
1304
1306
1308
1310
1312
1314
1316
DRAM

图13

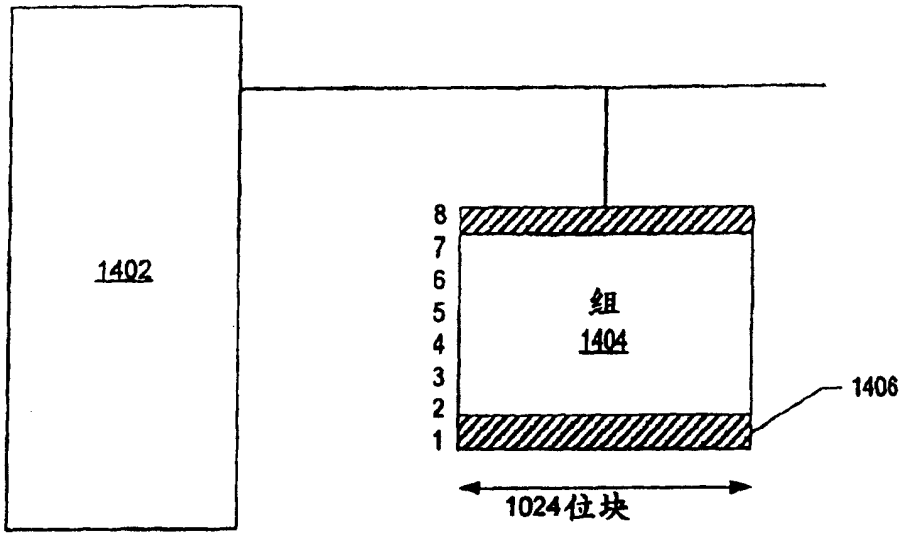

1402
8
7
6
5
4
3
2
1
组
1404
1406
1024位块

图14A

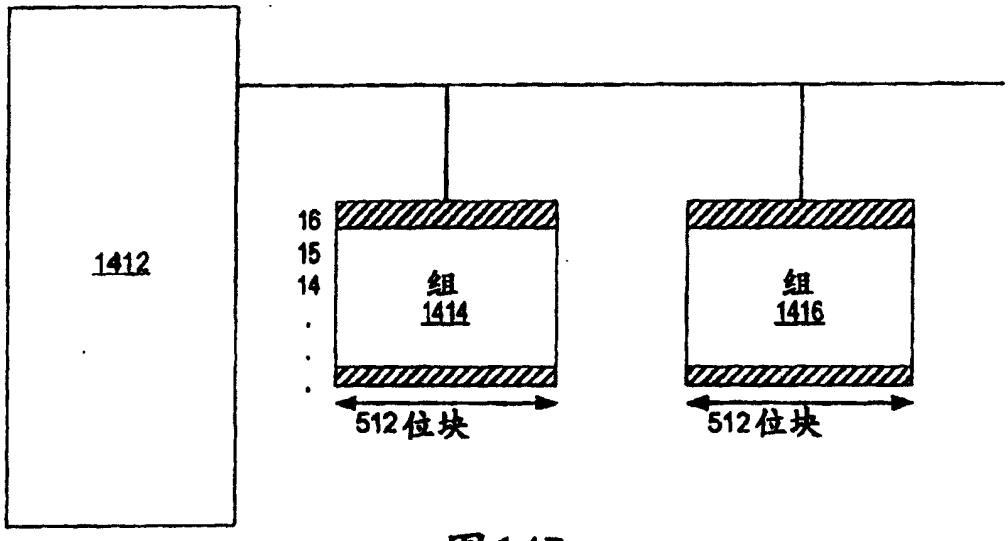

1412
16
15
14
组
1414
组
1416
512位块
512位块

图14B

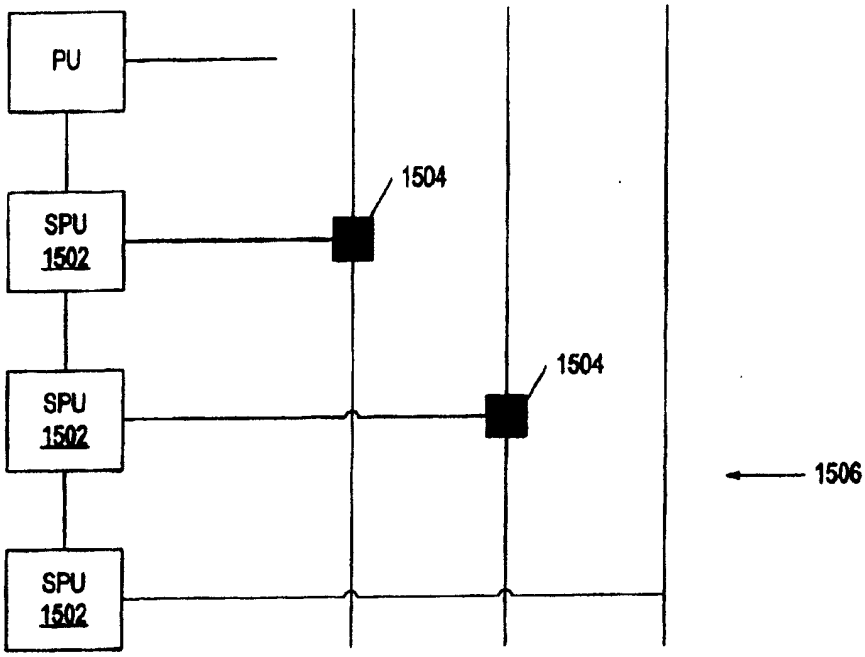


图15

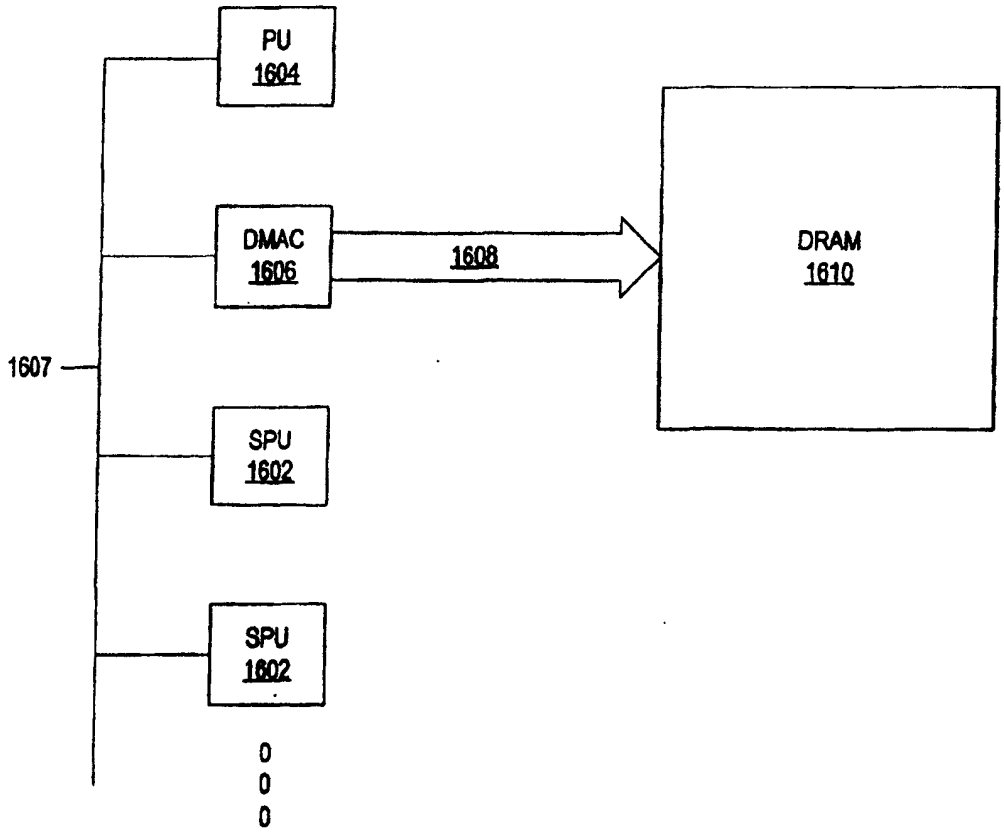


图16

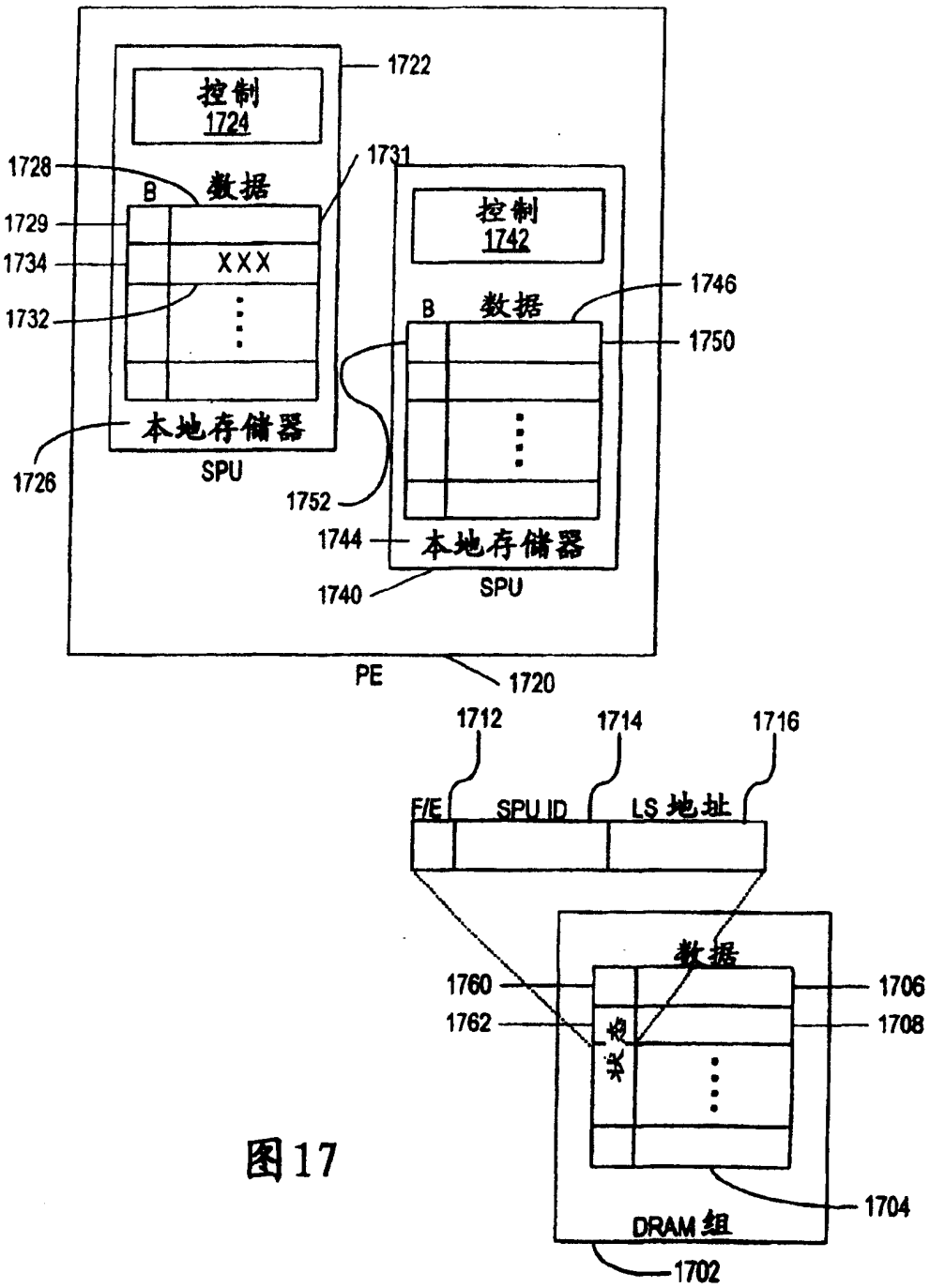

控制
1724
1722
1728
数据
B
1731
1729
1734
XXX
1732
1726
本地存储器
SPU
控制
1742
1746
B
数据
1750
1752
1744
本地存储器
1740
SPU
PE
1720
1712
1714
1716
F/E
SPU ID
LS 地址
数据
1760
1762
1706
1708
1704
DRAM 组
1702

图17

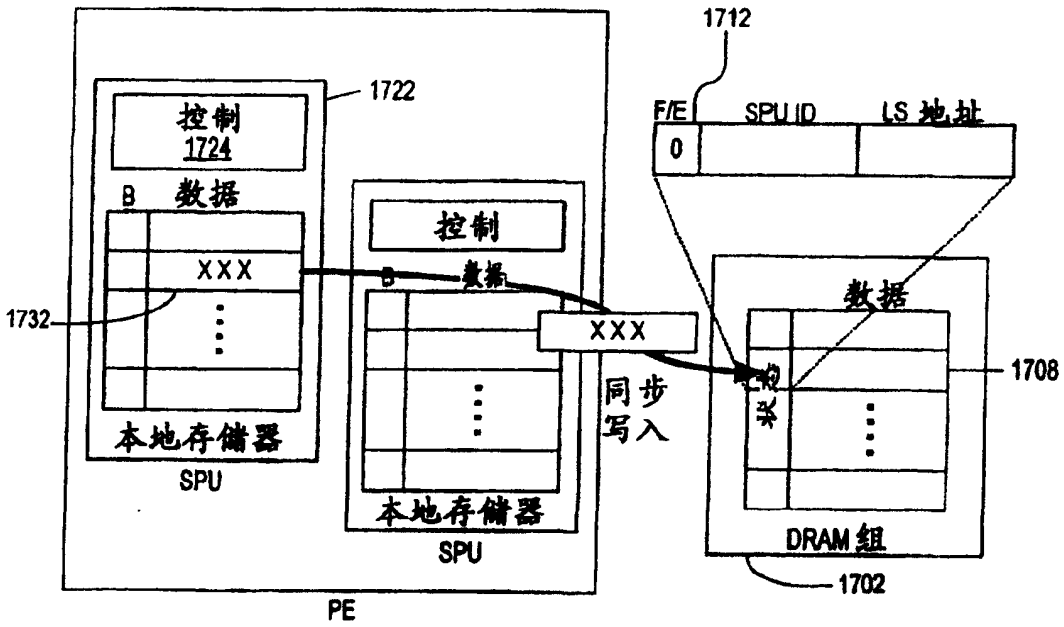

1712
F/E
SPU ID
LS 地址
0
1722
控制
1724
B
数据
XXX
1732
本地存储器
SPU
控制
B
数据
本地存储器
SPU
XXX
同步
写入
数据
1708
DRAM组
1702
PE

图18

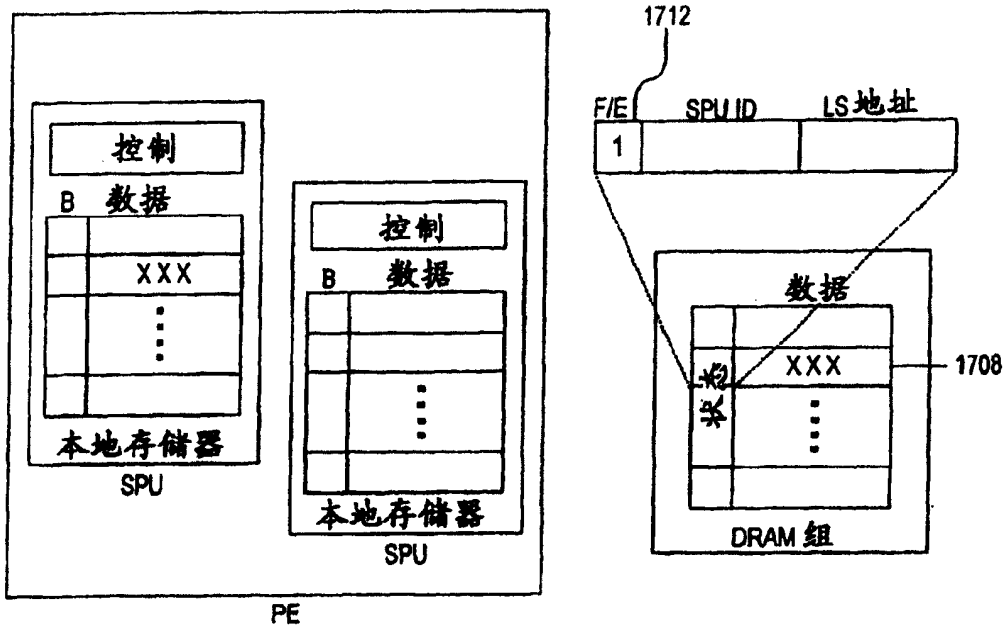

1712
F/E
SPU ID
LS地址
1
控制
B
数据
XXX
本地存储器
SPU
控制
B
数据
本地存储器
SPU
PE
数据
XXX
1708
DRAM 组

图19

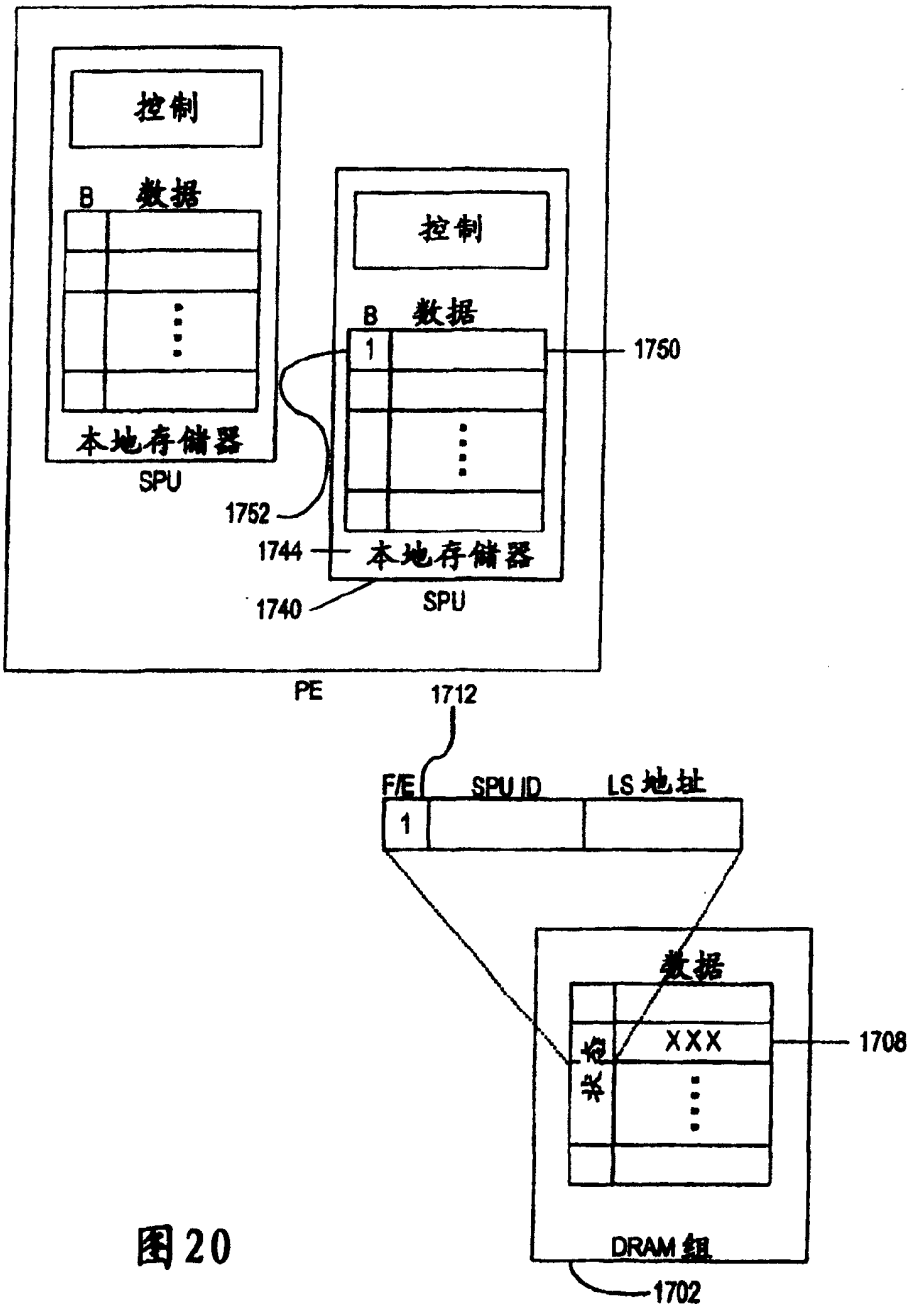

控制
B 数据
本地存储器
SPU
控制
B 数据
1
1750
1752
1744
本地存储器
1740
SPU
PE
1712
F/E
SPU ID
LS 地址
1
数据
XXX
1708
DRAM 组
1702

图20

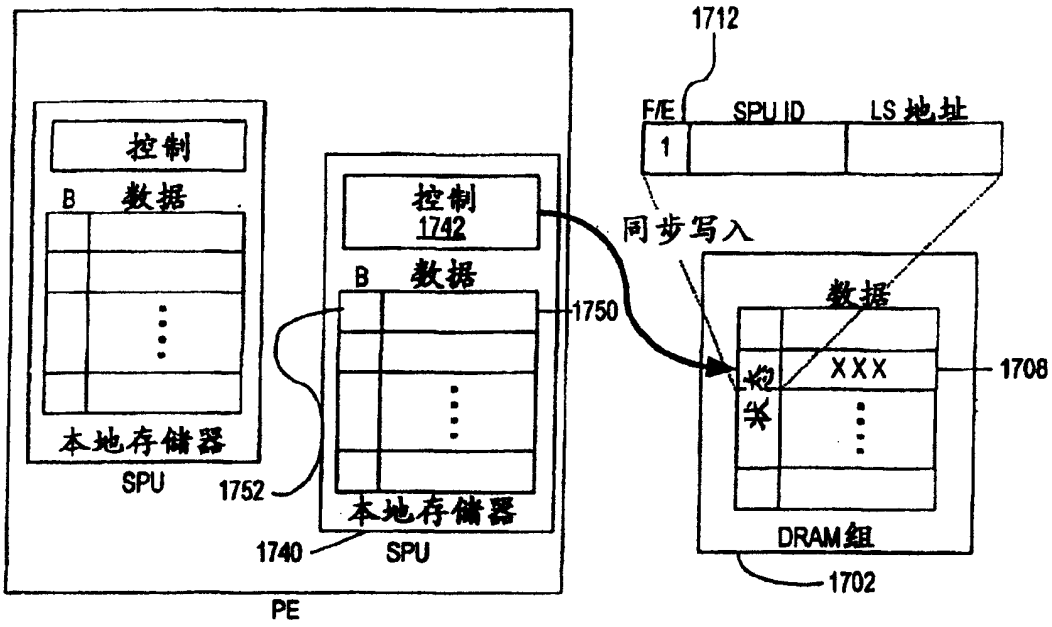

1712
F/E
SPU ID
LS 地址
1
控制
B
数据
本地存储器
SPU
1752
控制
1742
B
数据
1750
本地存储器
1740
SPU
同步写入
数据
XXX
1708
DRAM组
1702
PE

图21

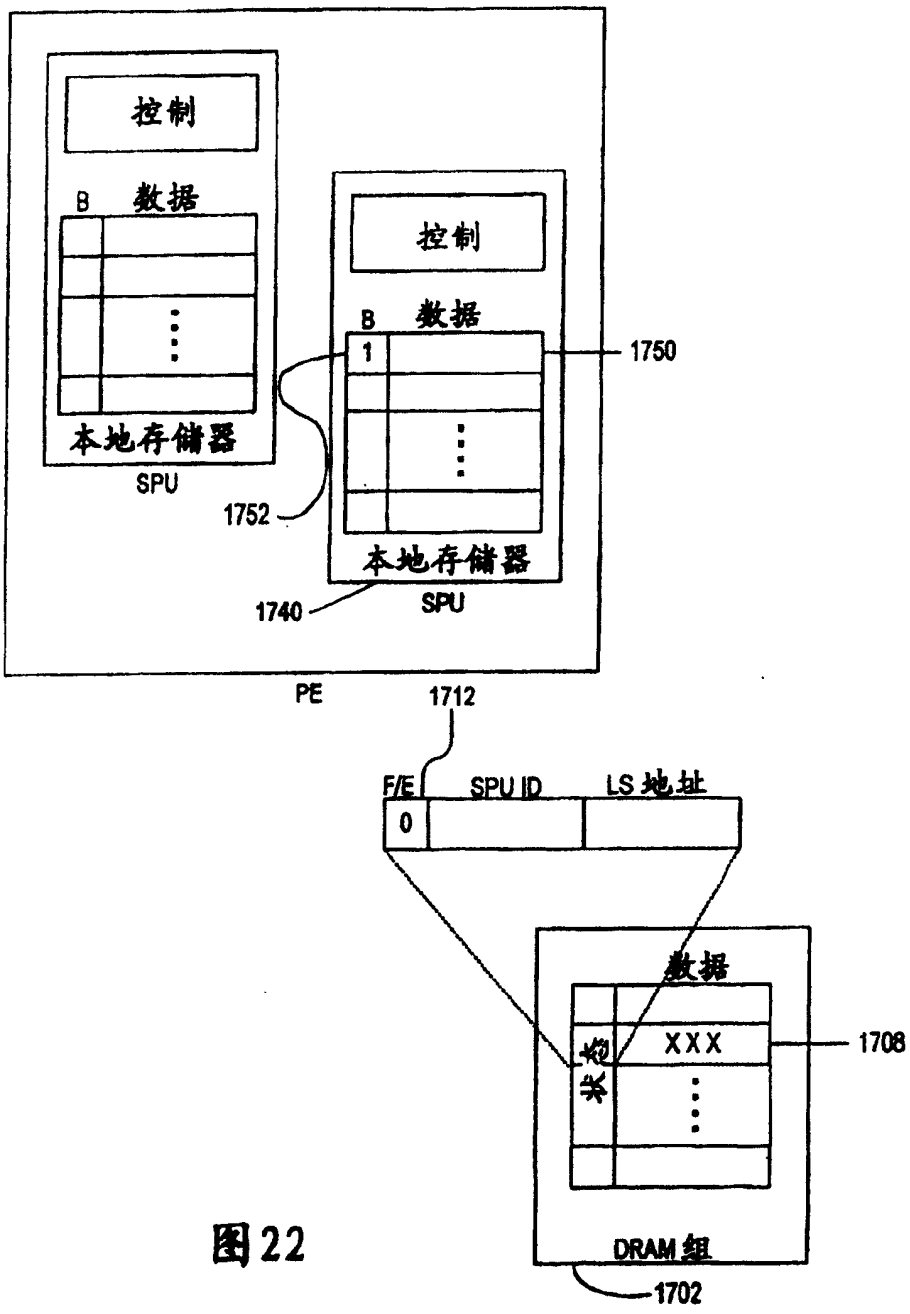

控制
B 数据
本地存储器
SPU
控制
B 数据
1
1750
1752
本地存储器
SPU
1740
PE
1712
F/E
SPU ID
LS 地址
0
数据
XXX
1708
DRAM 组
1702

图22

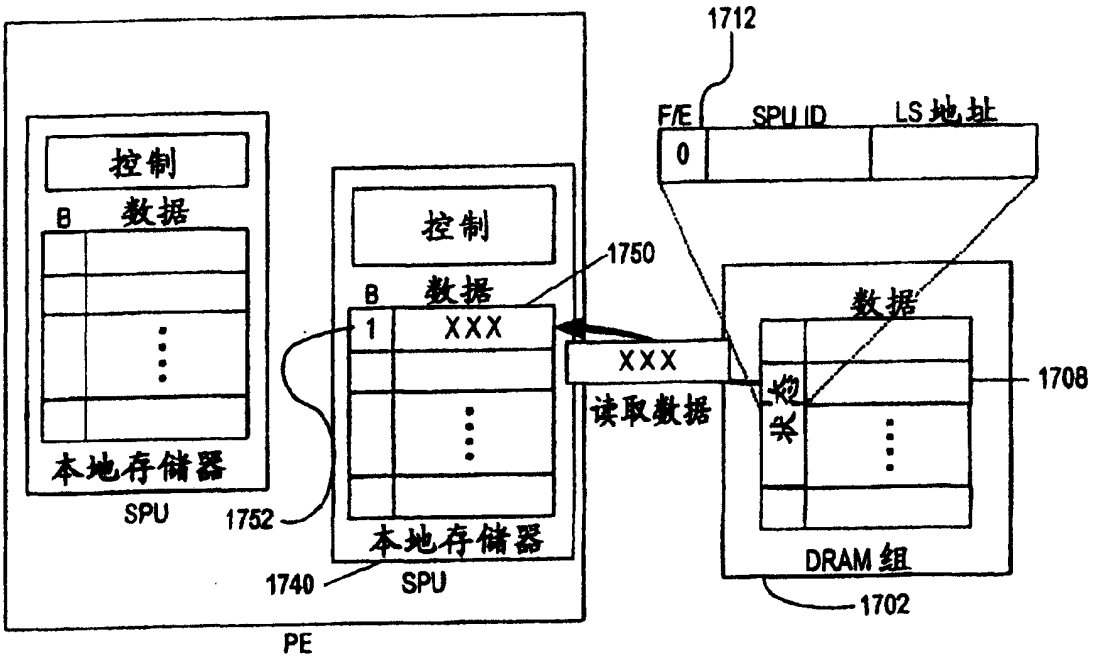

1712
F/E
SPU ID
LS 地址
0
控制
B
数据
本地存储器
SPU
1752
控制
B
数据
1
XXX
1750
本地存储器
1740
SPU
XXX
读取数据
数据
物
1708
DRAM 组
1702
PE

图23

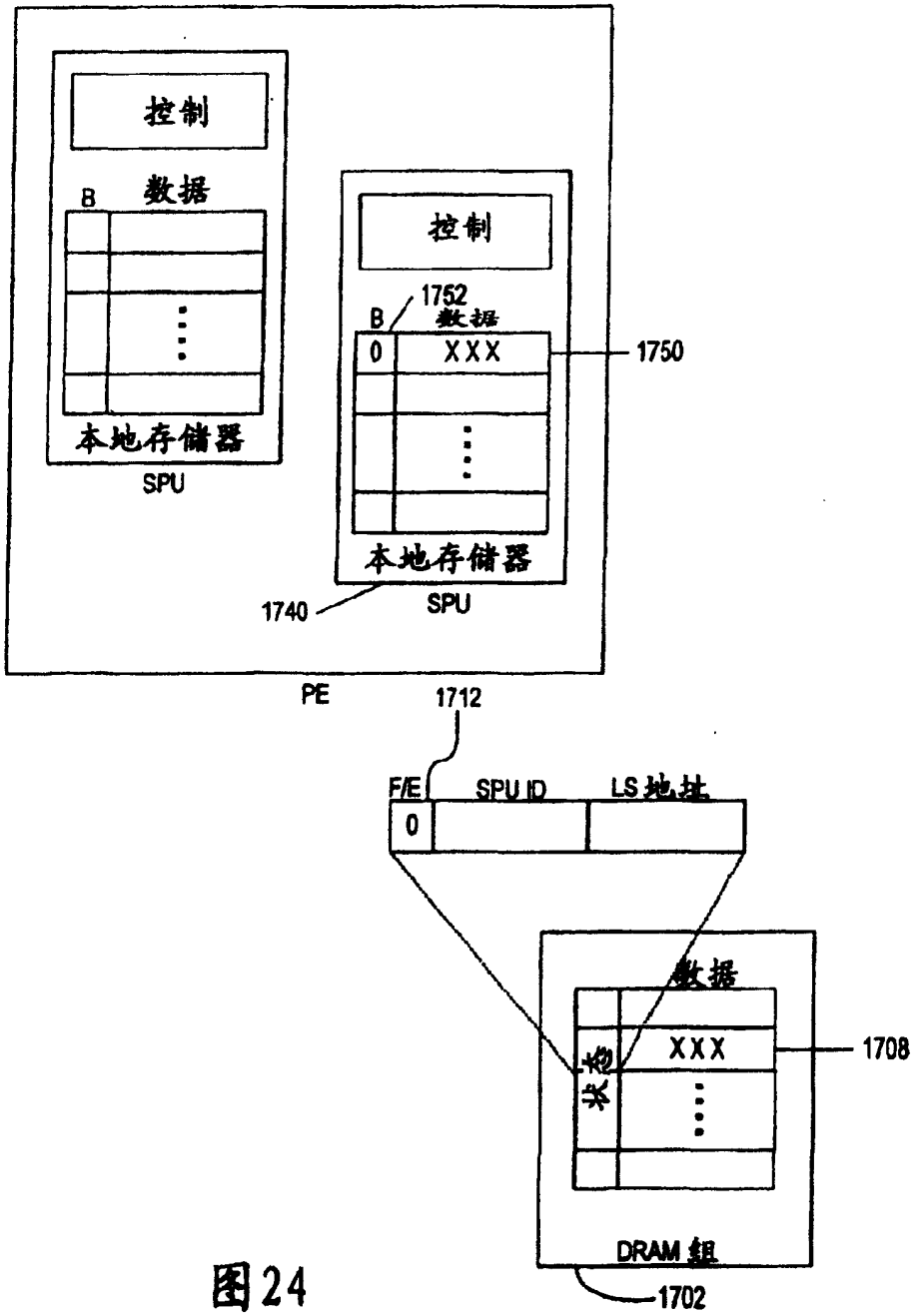

控制
B
数据
本地存储器
SPU
控制
1752
B
数据
0
XXX
1750
本地存储器
SPU
1740
PE
1712
F/E
SPU ID
LS 地址
0
数据
XXX
1708
DRAM 组
1702

图24

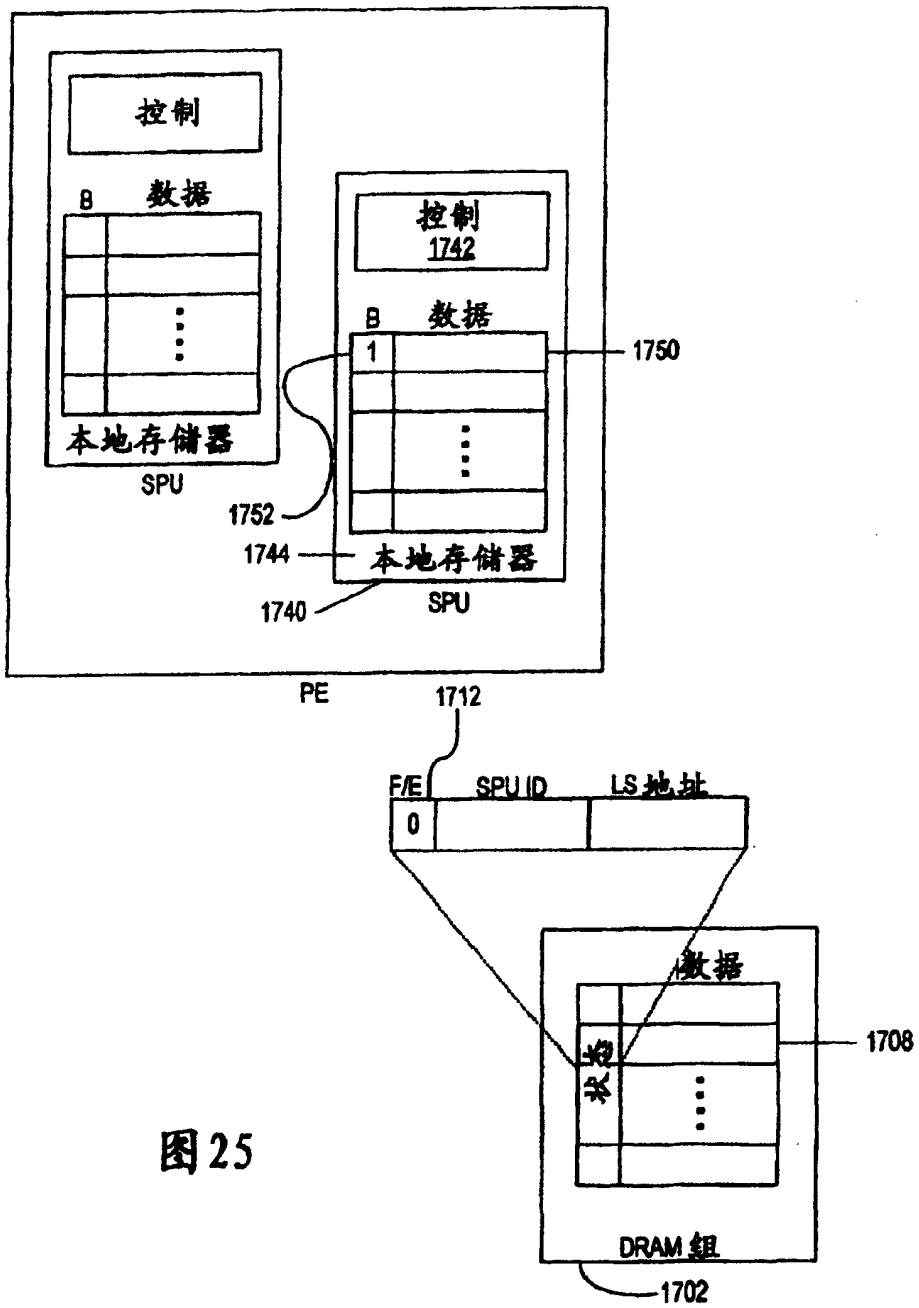

控制
B
数据
本地存储器
SPU
控制
1742
B
数据
1
1750
1752
1744
本地存储器
1740
SPU
PE
1712
F/E
SPU ID
LS 地址
0
数据
1708
DRAM 组
1702

图25

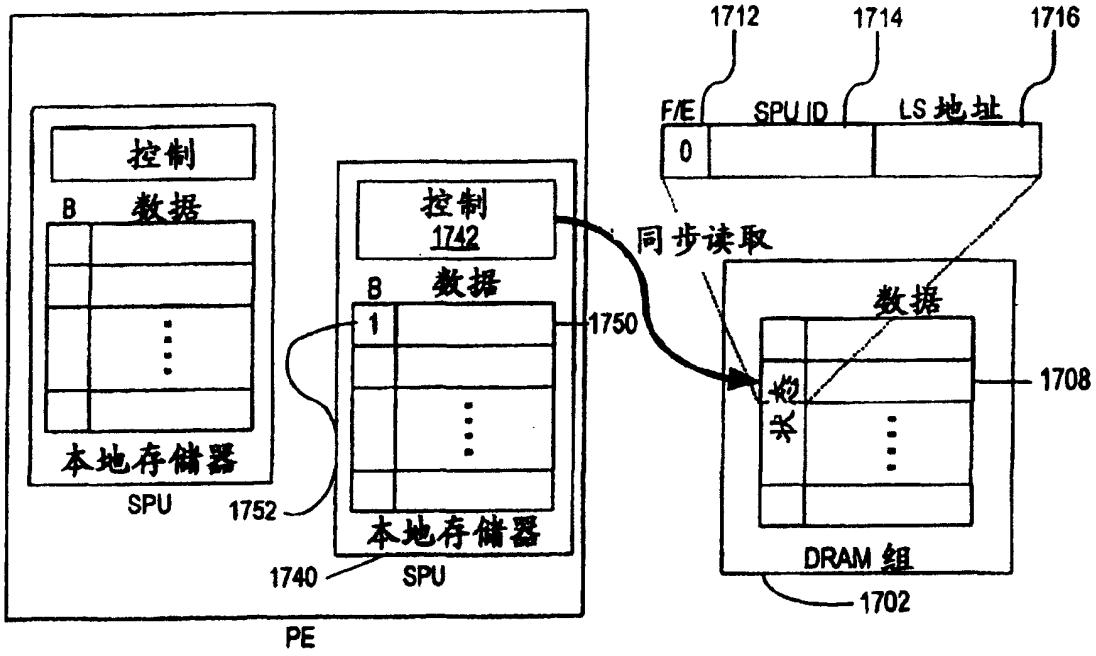

1712
1714
1716
F/E
SPU ID
LS 地址
0
控制
B
数据
本地存储器
SPU
1752
控制
1742
B
数据
1
1750
本地存储器
1740
SPU
同步读取
数据
1708
DRAM 组
1702
PE

图26

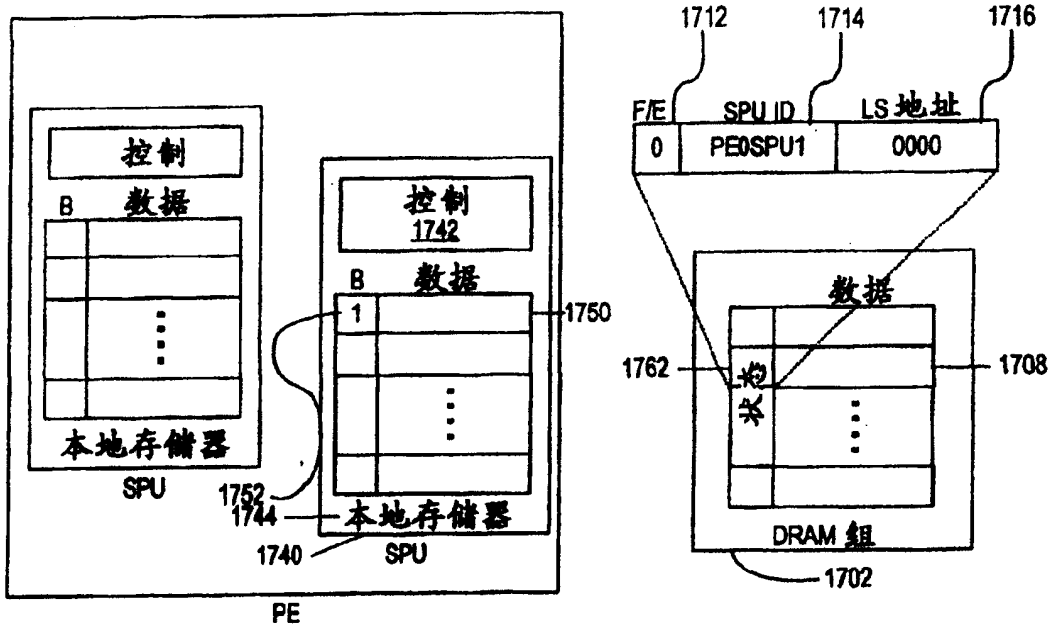

1712
1714
1716
F/E
SPU ID
LS 地址
0
PE0SPU1
0000
控制
B
数据
本地存储器
SPU
控制
1742
B
数据
1
1750
1752
1744
本地存储器
1740
SPU
PE
数据
1762
1708
DRAM 组
1702

图27

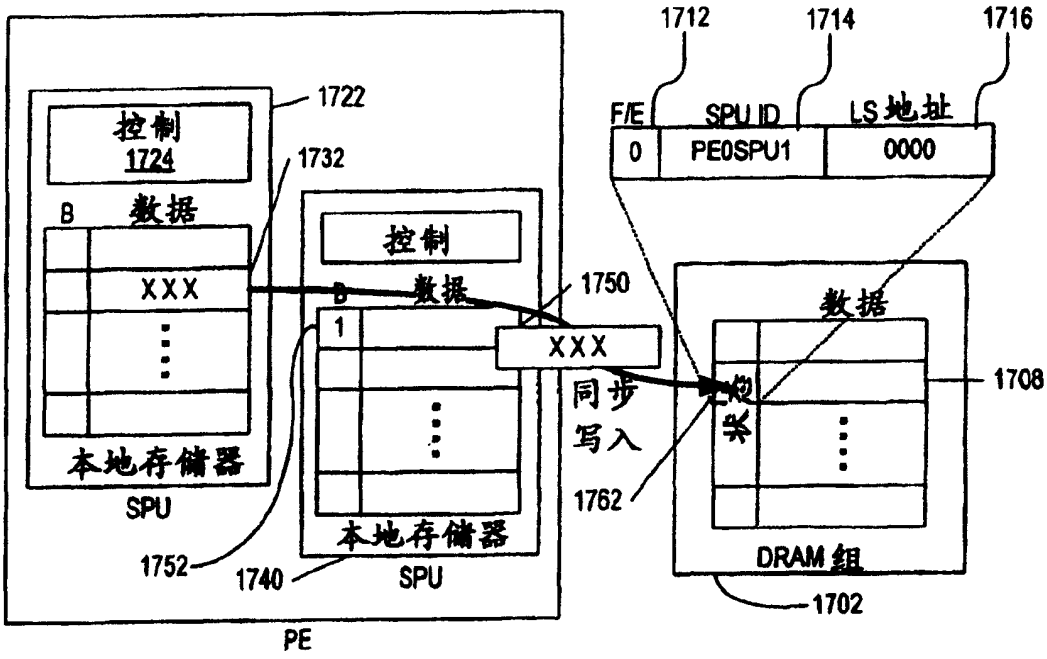

1712
1714
1716
F/E
SPU ID
LS 地址
0
PE0SPU1
0000
1722
控制
1724
1732
B
数据
XXX
本地存储器
SPU
控制
B
数据
1
本地存储器
SPU
1750
XXX
同步
写入
1762
数据
1708
DRAM 组
1702
1752
1740
PE

图28

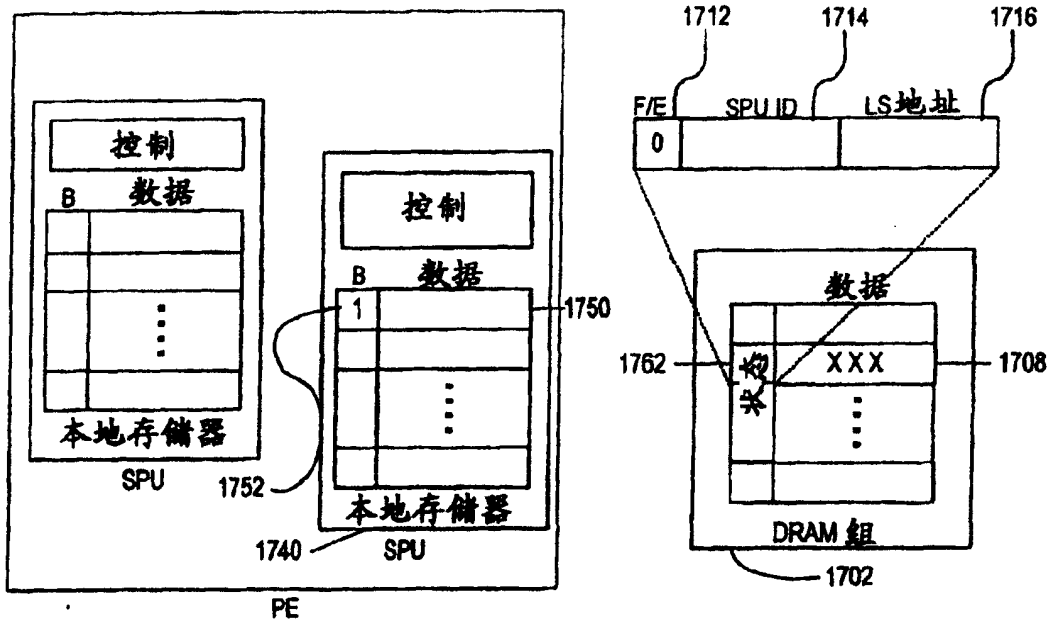

1712
1714
1716
F/E
SPU ID
LS地址
0
控制
B
数据
本地存储器
SPU
1752
控制
B
数据
1
1750
本地存储器
1740
SPU
PE
数据
1762
XXX
1708
DRAM组
1702

图29

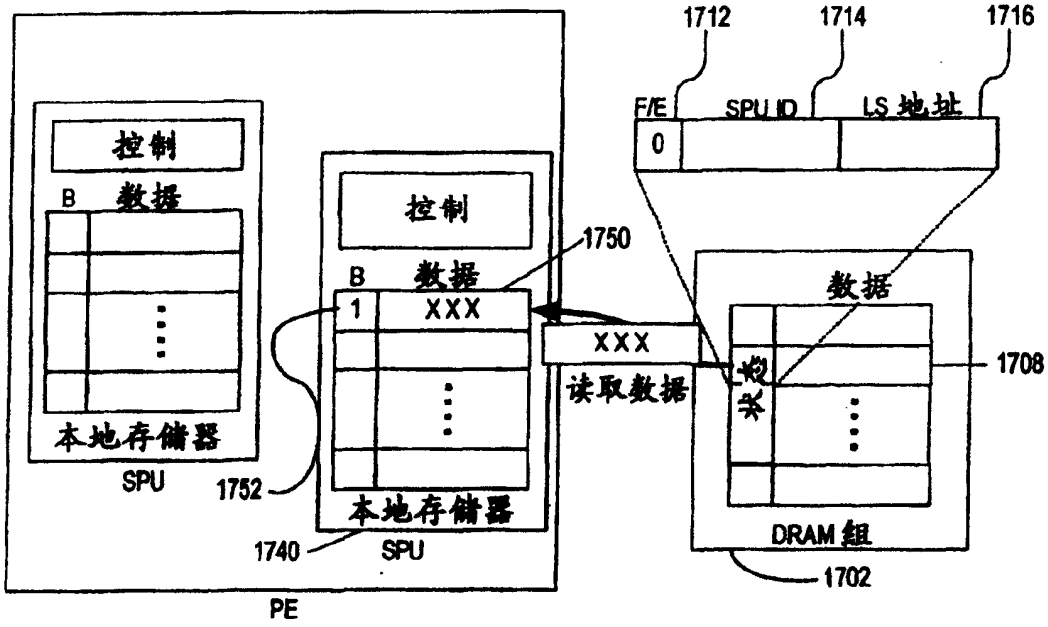

1712
1714
1716
F/E
SPU ID
LS 地址
0
控制
B
数据
本地存储器
SPU
1752
控制
B
数据
1
XXX
1750
本地存储器
1740
SPU
PE
XXX
读取数据
数据
1708
DRAM 组
1702

图 30

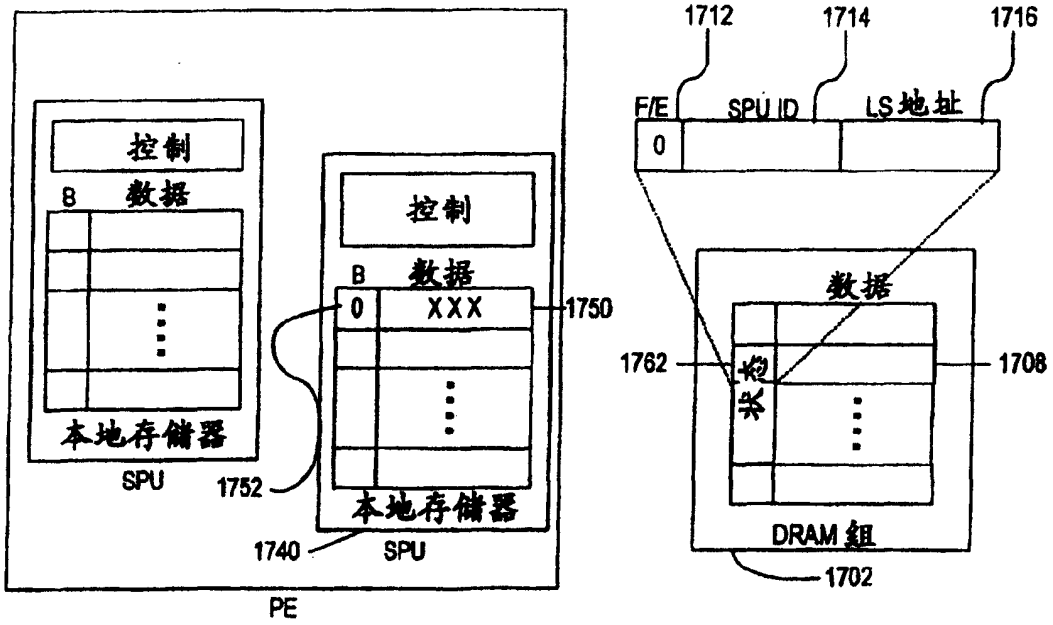

1712
1714
1716
F/E
SPU ID
LS 地址
0
控制
B
数据
本地存储器
SPU
1752
控制
B
数据
0
XXX
1750
本地存储器
1740
SPU
PE
数据
1762
1708
DRAM 组
1702

图 31

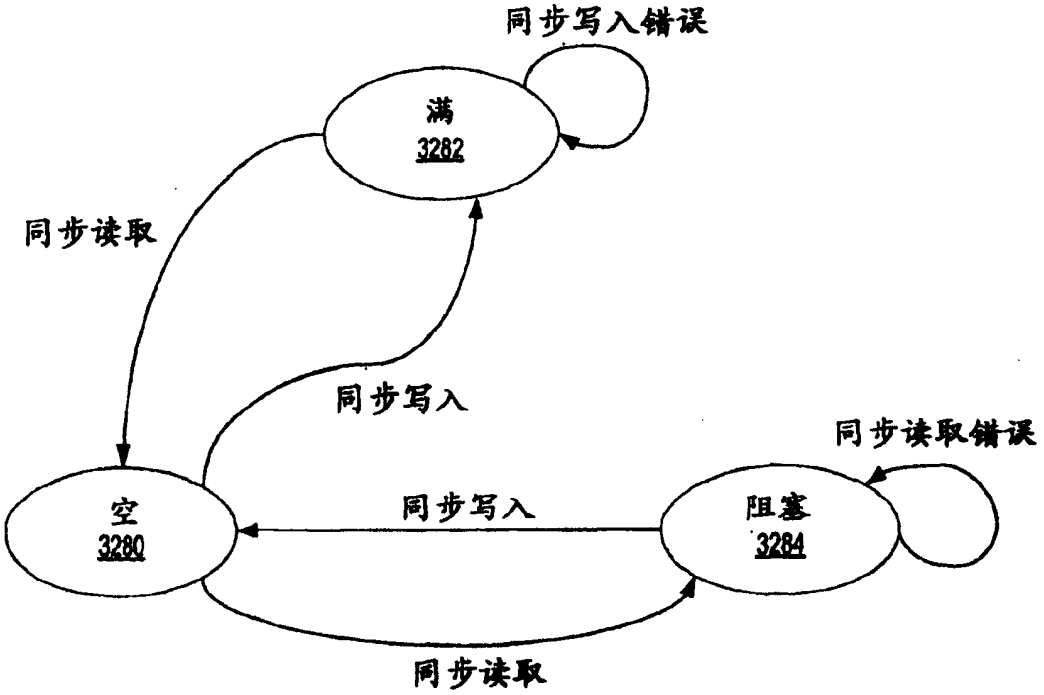


图32

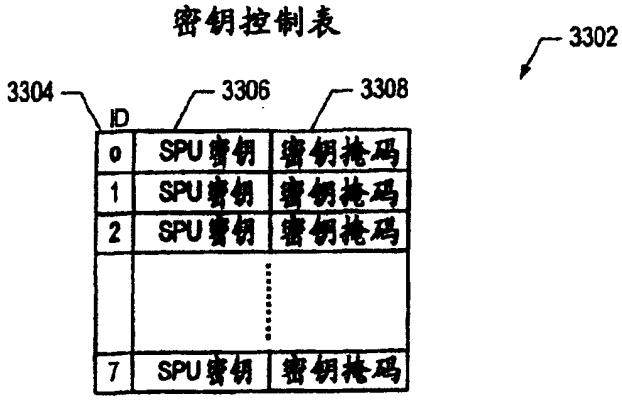

密钥控制表
3302
3304
3306
3308
ID
0
SPU密钥
密钥掩码
1
SPU密钥
密钥掩码
2
SPU密钥
密钥掩码
7
SPU密钥
密钥掩码

图33

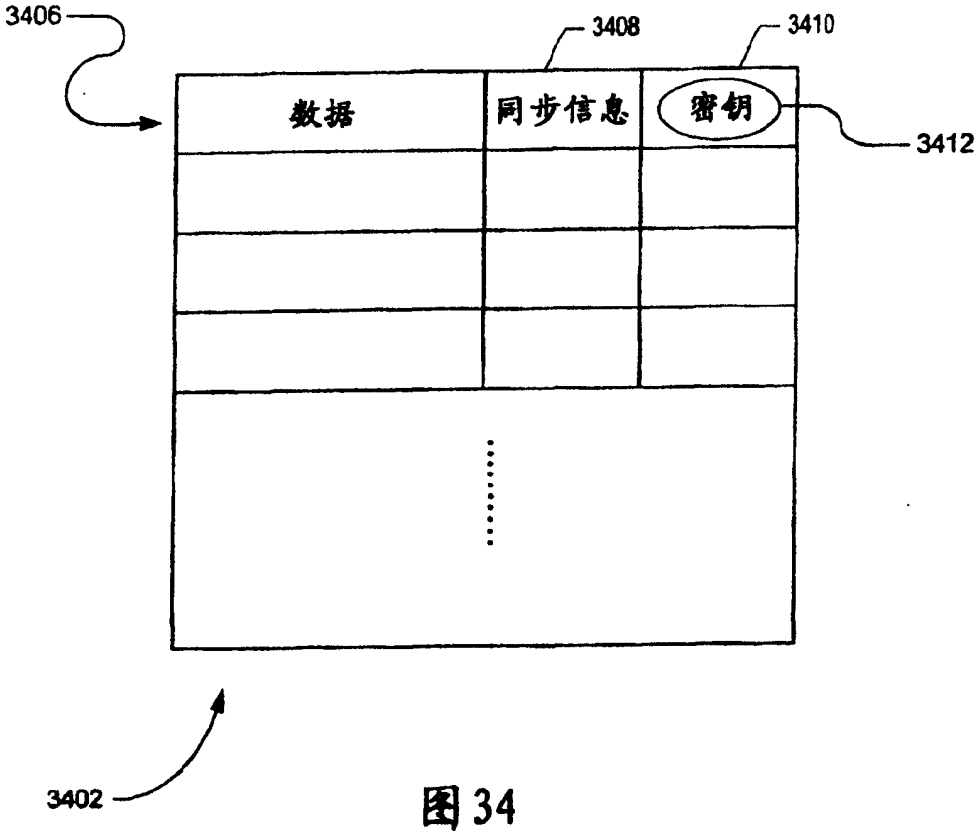


图 34

存储器访问控制表

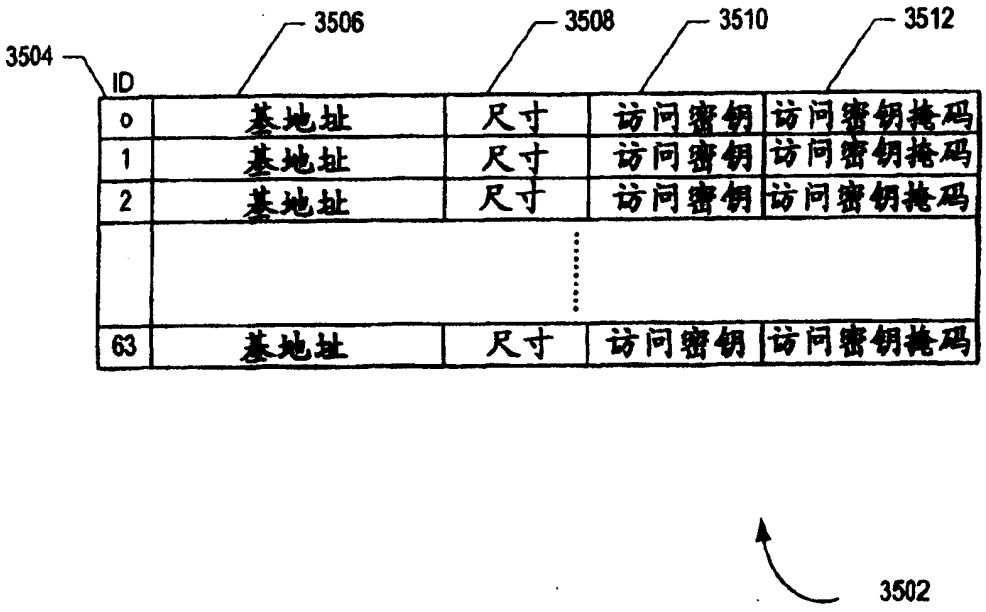


图35

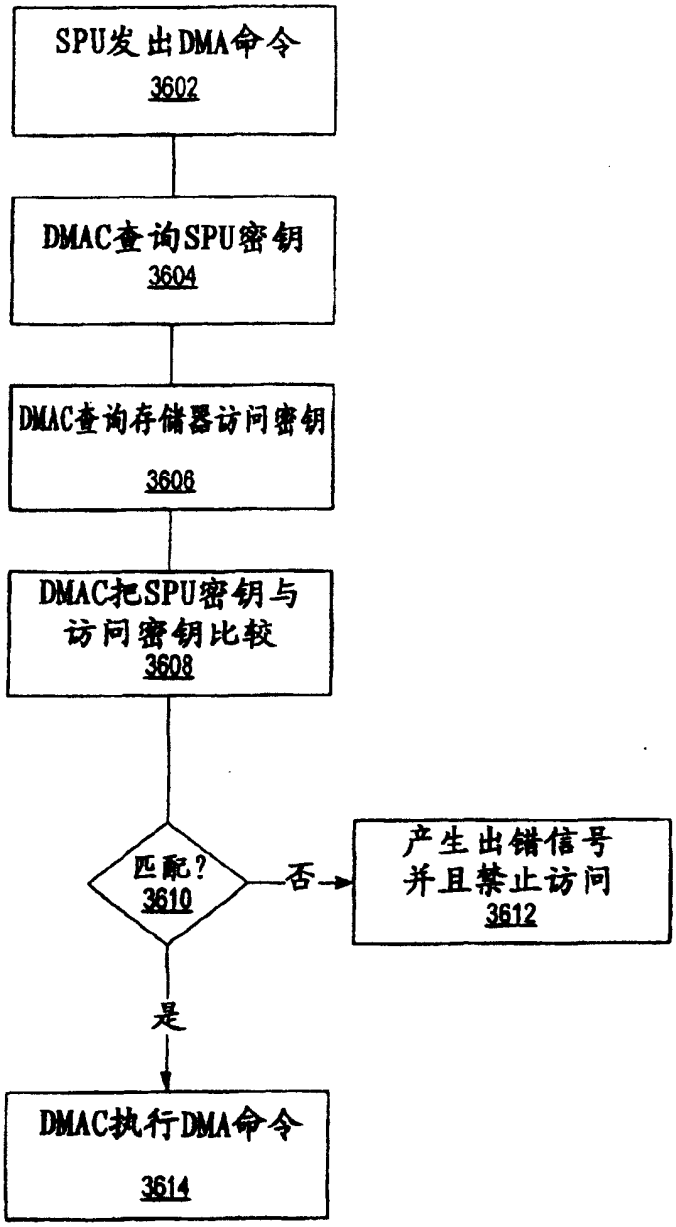

SPU发出DMA命令
3602
DMAC查询SPU密钥
3604
DMAC查询存储器访问密钥
3606
DMAC把SPU密钥与
访问密钥比较
3608
匹配?
3610
否
产生出错信号
并且禁止访问
3612
是
DMAC执行DMA命令
3614

图36

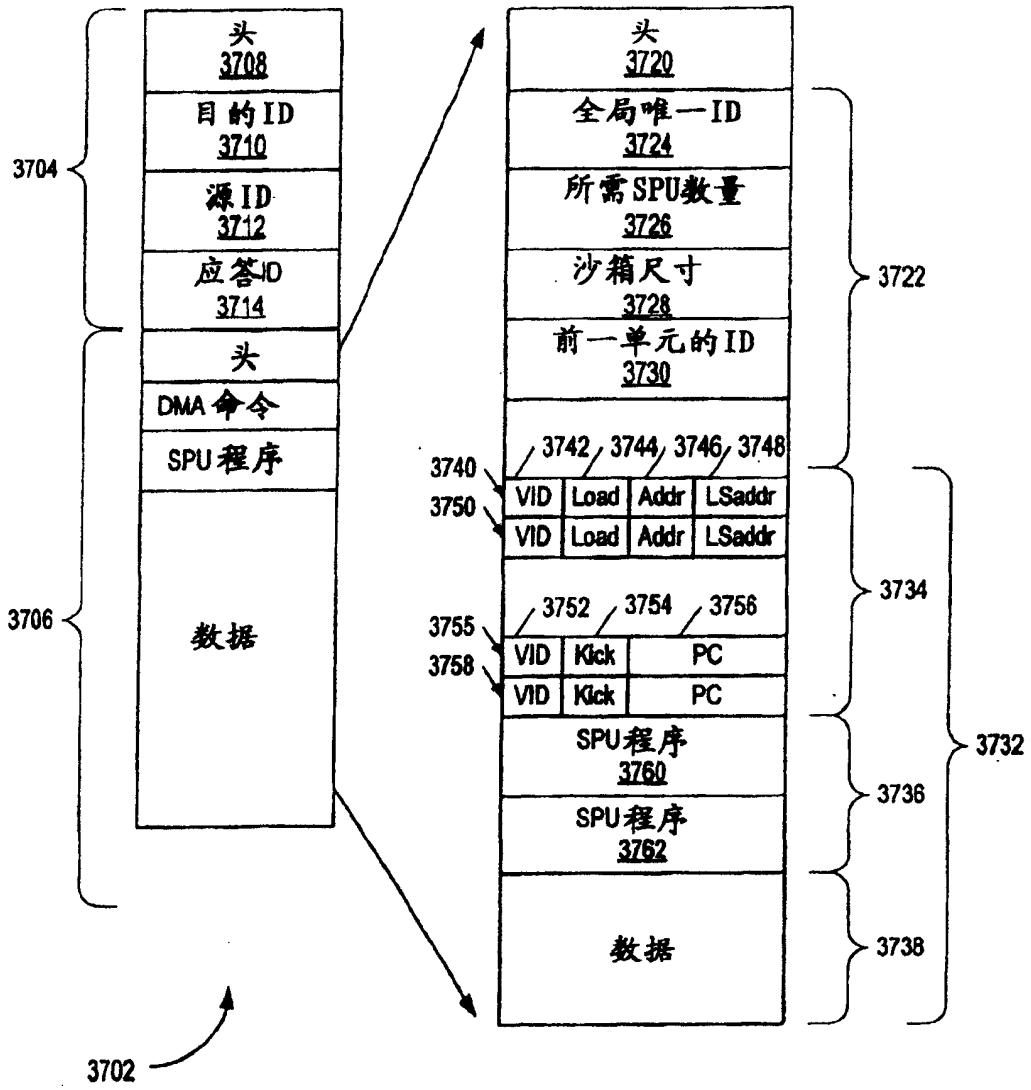

头
3708
目的ID
3710
源ID
3712
应答ID
3714
3704
头
DMA命令
SPU程序
数据
3706
3702
头
3720
全局唯一ID
3724
所需SPU数量
3726
沙箱尺寸
3728
前一单元的ID
3730
3722
3740
3750
3742
3744
3746
3748
VID
Load
Addr
LSaddr
VID
Load
Addr
LSaddr
3734
3755
3758
3752
3754
3756
VID
Kick
PC
VID
Kick
PC
SPU程序
3760
SPU程序
3762
3736
3732
数据
3738

图37

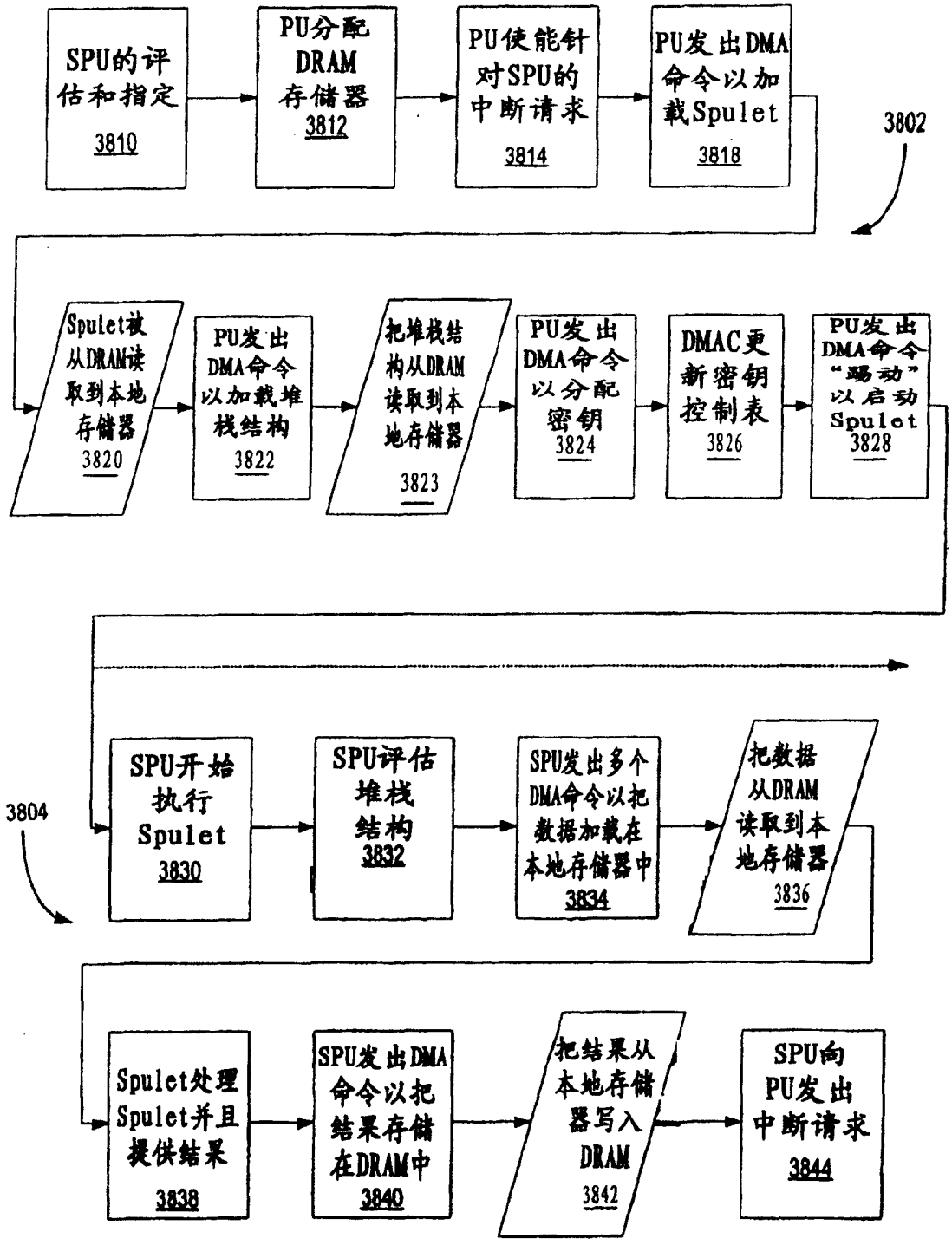

SPU的评估和指定
3810
PU分配DRAM存储器
3812
PU使能针对SPU的中断请求
3814
PU发出DMA命令以加载Spulet
3818
3802
Spulet被从DRAM读取到本地存储器
3820
PU发出DMA命令以加载堆栈结构
3822
把堆栈结构从DRAM读取到本地存储器
3823
PU发出DMA命令以分配密钥
3824
DMAC更新密钥控制表
3826
PU发出DMA命令"踢动"以启动Spulet
3828
3804
SPU开始执行Spulet
3830
SPU评估堆栈结构
3832
SPU发出多个DMA命令以把数据加载在本地存储器中
3834
把数据从DRAM读取到本地存储器
3836
Spulet处理Spulet并且提供结果
3838
SPU发出DMA命令以把结果存储在DRAM中
3840
把结果从本地存储器写入DRAM
3842
SPU向PU发出中断请求
3844

图38

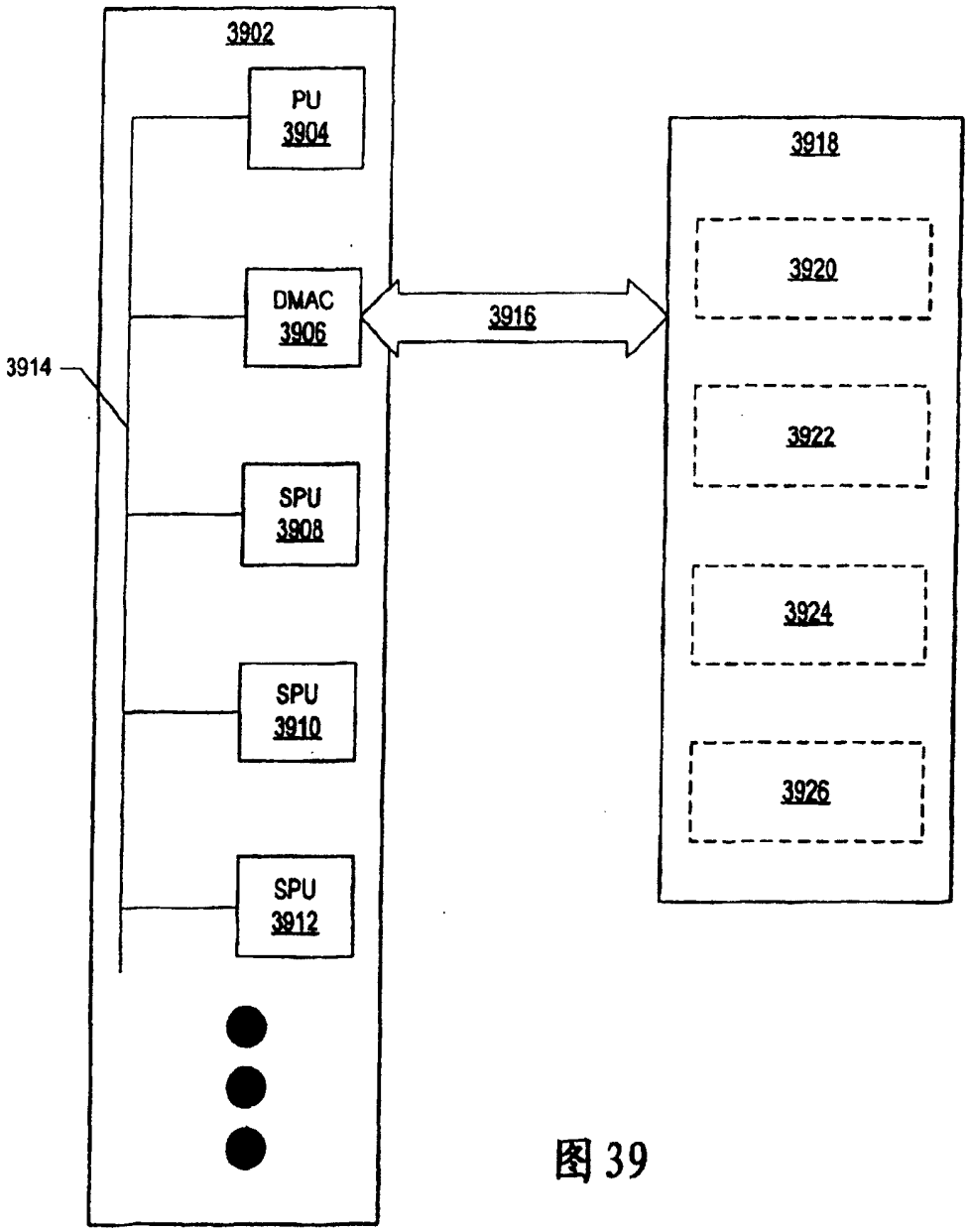

3902
PU
3904
DMAC
3906
3914
SPU
3908
SPU
3910
SPU
3912
3916
3918
3920
3922
3924
3926

图 39

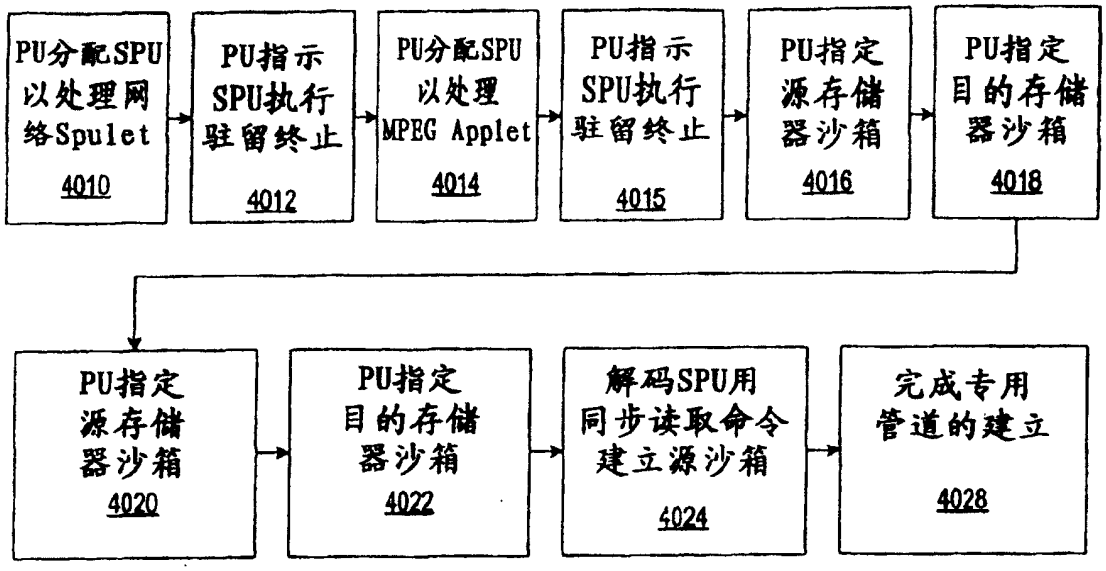

PU分配SPU以处理网络Spulet
4010
PU指示SPU执行驻留终止
4012
PU分配SPU以处理MPEG Applet
4014
PU指示SPU执行驻留终止
4015
PU指定源存储器沙箱
4016
PU指定目的存储器沙箱
4018
PU指定源存储器沙箱
4020
PU指定目的存储器沙箱
4022
解码SPU用同步读取命令建立源沙箱
4024
完成专用管道的建立
4028

图40A

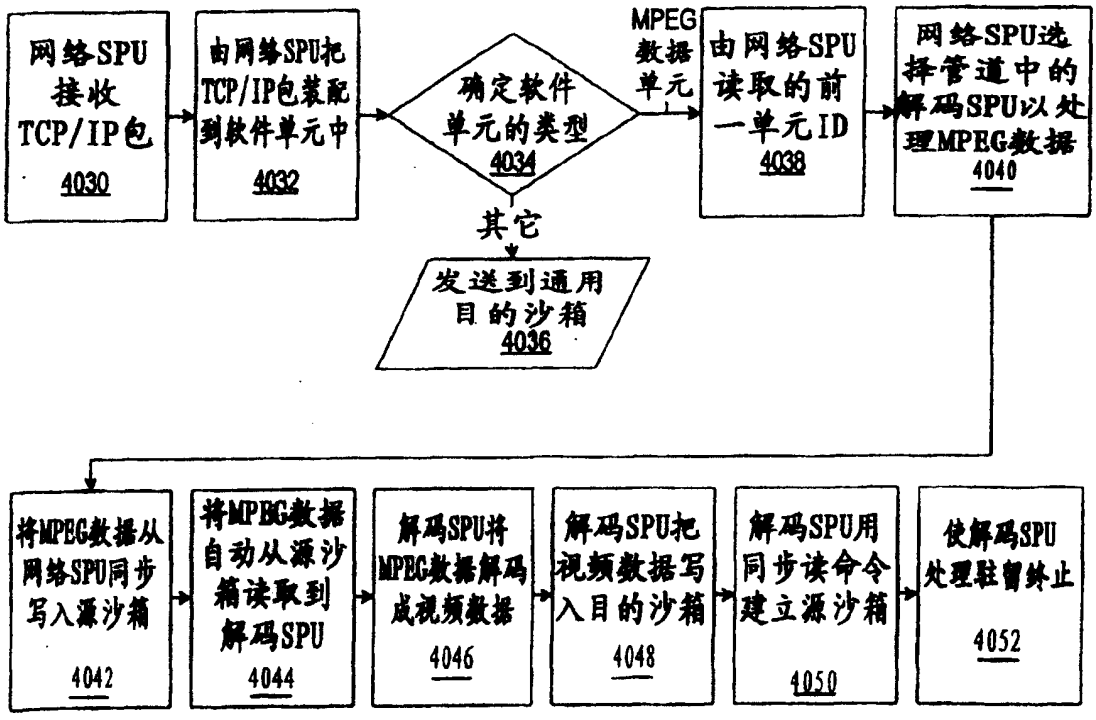

网络SPU接收TCP/IP包
4030
由网络SPU把TCP/IP包装配到软件单元中
4032
确定软件单元的类型
4034
MPEG数据单元
其它
发送到通用目的沙箱
4036
由网络SPU读取的前一单元ID
4038
网络SPU选择管道中的解码SPU以处理MPEG数据
4040
将MPEG数据从网络SPU同步写入源沙箱
4042
将MPEG数据自动从源沙箱读取到解码SPU
4044
解码SPU将MPEG数据解码成视频数据
4046
解码SPU把视频数据写入目的沙箱
4048
解码SPU用同步读命令建立源沙箱
4050
使解码SPU处理驻留终止
4052

图40B

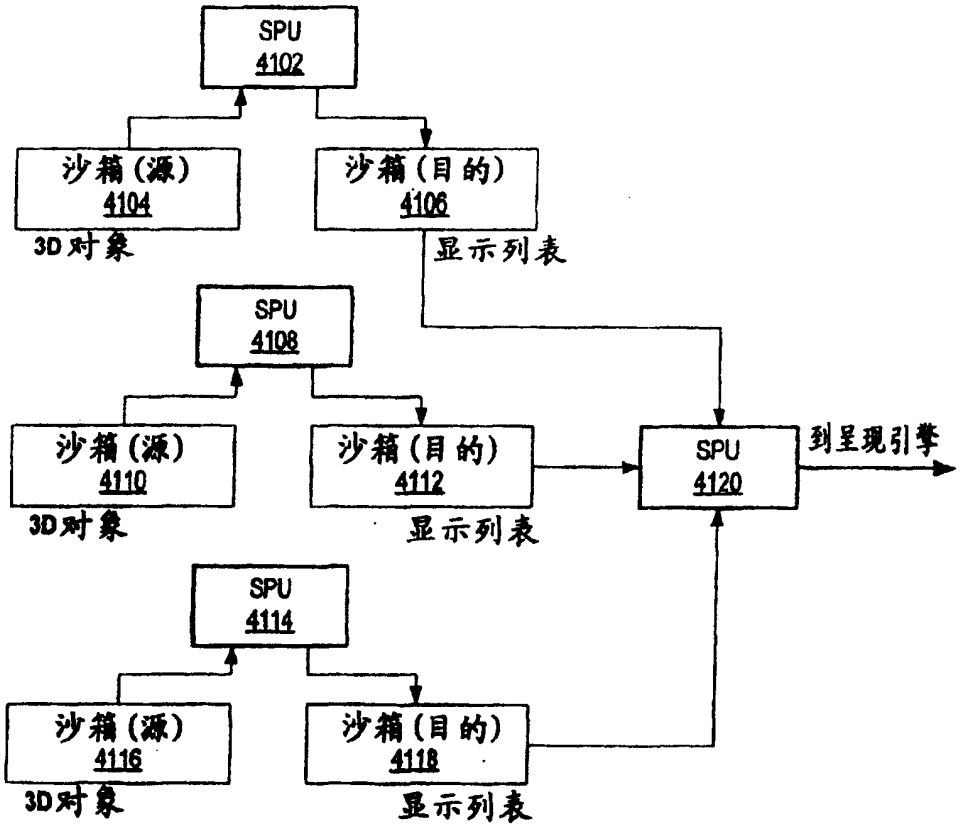

SPU
4102
沙箱(源)
4104
3D对象
沙箱(目的)
4106
显示列表
SPU
4108
沙箱(源)
4110
3D对象
沙箱(目的)
4112
显示列表
SPU
4120
到呈现引擎
SPU
4114
沙箱(源)
4116
3D对象
沙箱(目的)
4118
显示列表

图41

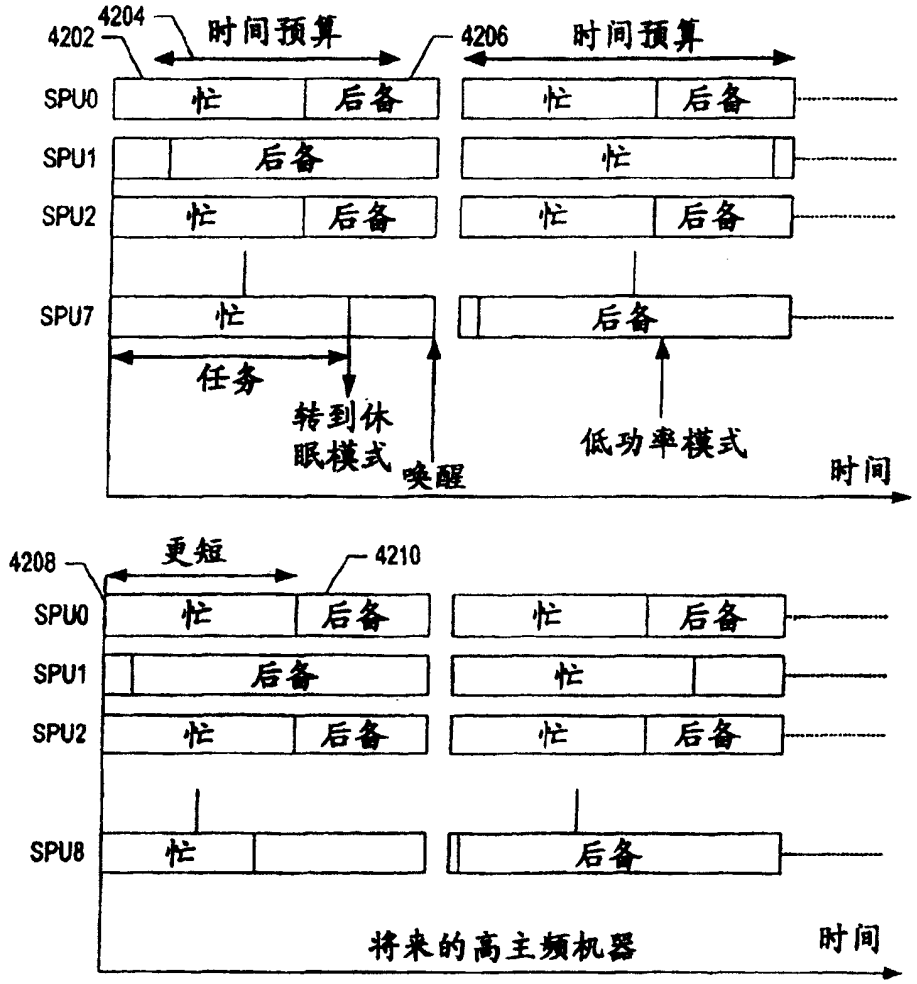

4202
4204
时间预算
4206
时间预算
SPU0
忙
后备
忙
后备
SPU1
后备
忙
SPU2
忙
后备
忙
后备
SPU7
忙
后备
任务
转到休眠模式
唤醒
低功率模式
时间
4208
更短
4210
SPU0
忙
后备
忙
后备
SPU1
后备
忙
SPU2
忙
后备
忙
后备
SPU8
忙
后备
将来的高主频机器
时间

图42

图43

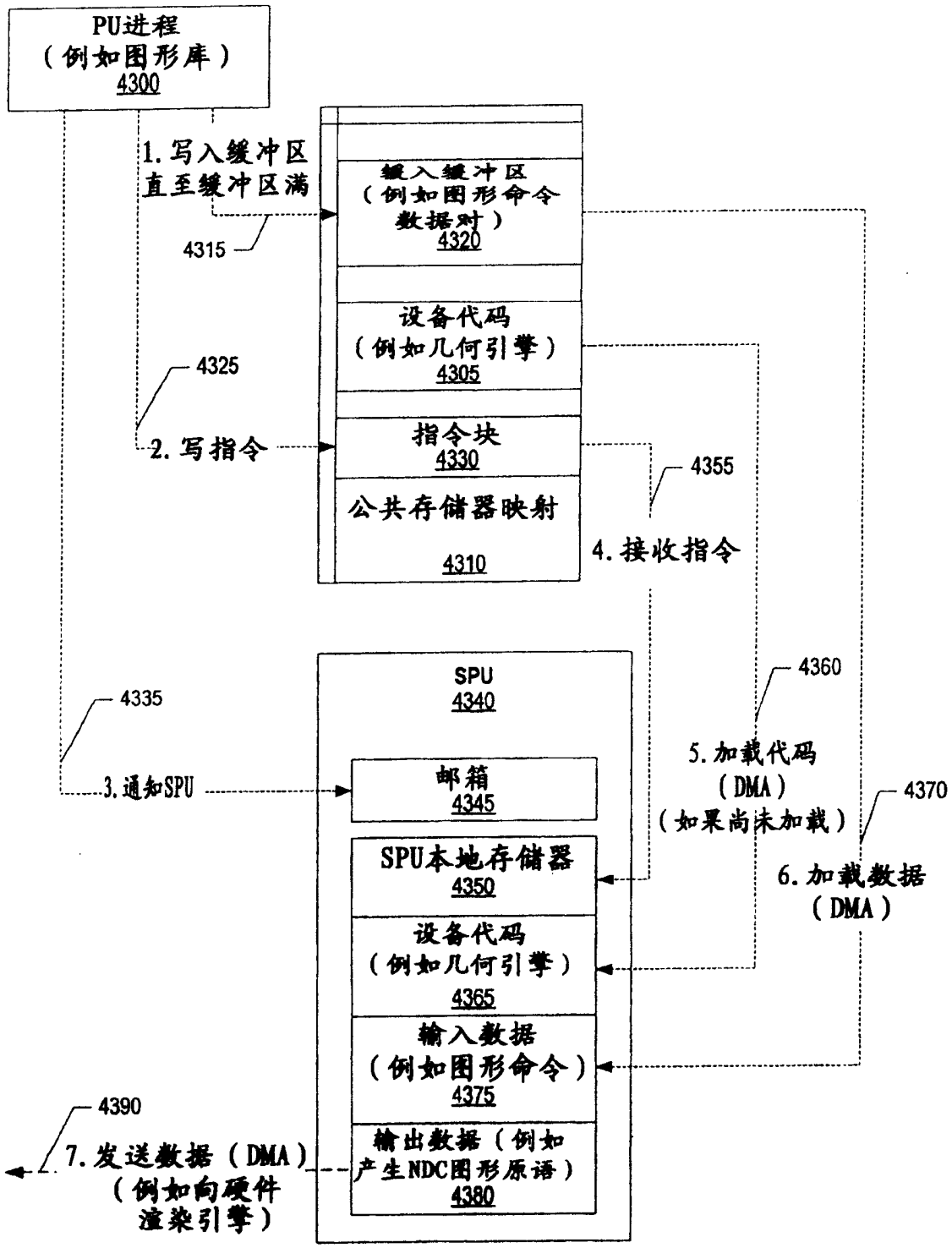

PU进程
（例如图形库）
4300
1.写入缓冲区
直至缓冲区满
4315
输入缓冲区
（例如图形命令
数据对）
4320
设备代码
（例如几何引擎）
4305
4325
2.写指令
指令块
4330
公共存储器映射
4310
4355
4.接收指令
SPU
4340
4335
3.通知SPU
邮箱
4345
4360
5.加载代码
（DMA）
（如果尚未加载）
4370
6.加载数据
（DMA）
SPU本地存储器
4350
设备代码
（例如几何引擎）
4365
输入数据
（例如图形命令）
4375
4390
7.发送数据（DMA）
（例如向硬件
渲染引擎）
输出数据（例如
产生NDC图形原语）
4380

图44

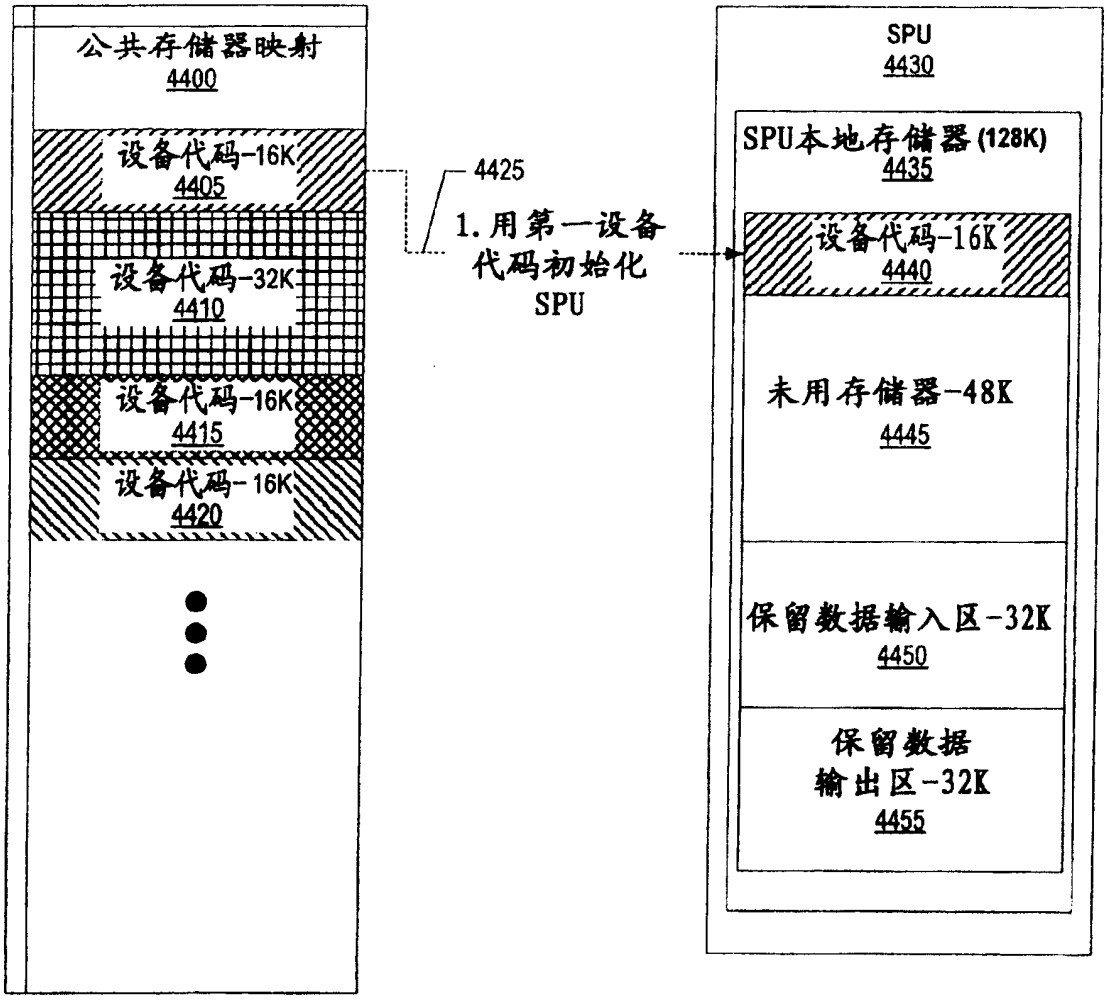

公共存储器映射
4400
设备代码-16K
4405
设备代码-32K
4410
设备代码-16K
4415
设备代码-16K
4420
4425
1.用第一设备代码初始化SPU
SPU
4430
SPU本地存储器(128K)
4435
设备代码-16K
4440
未用存储器-48K
4445
保留数据输入区-32K
4450
保留数据输出区-32K
4455

图45

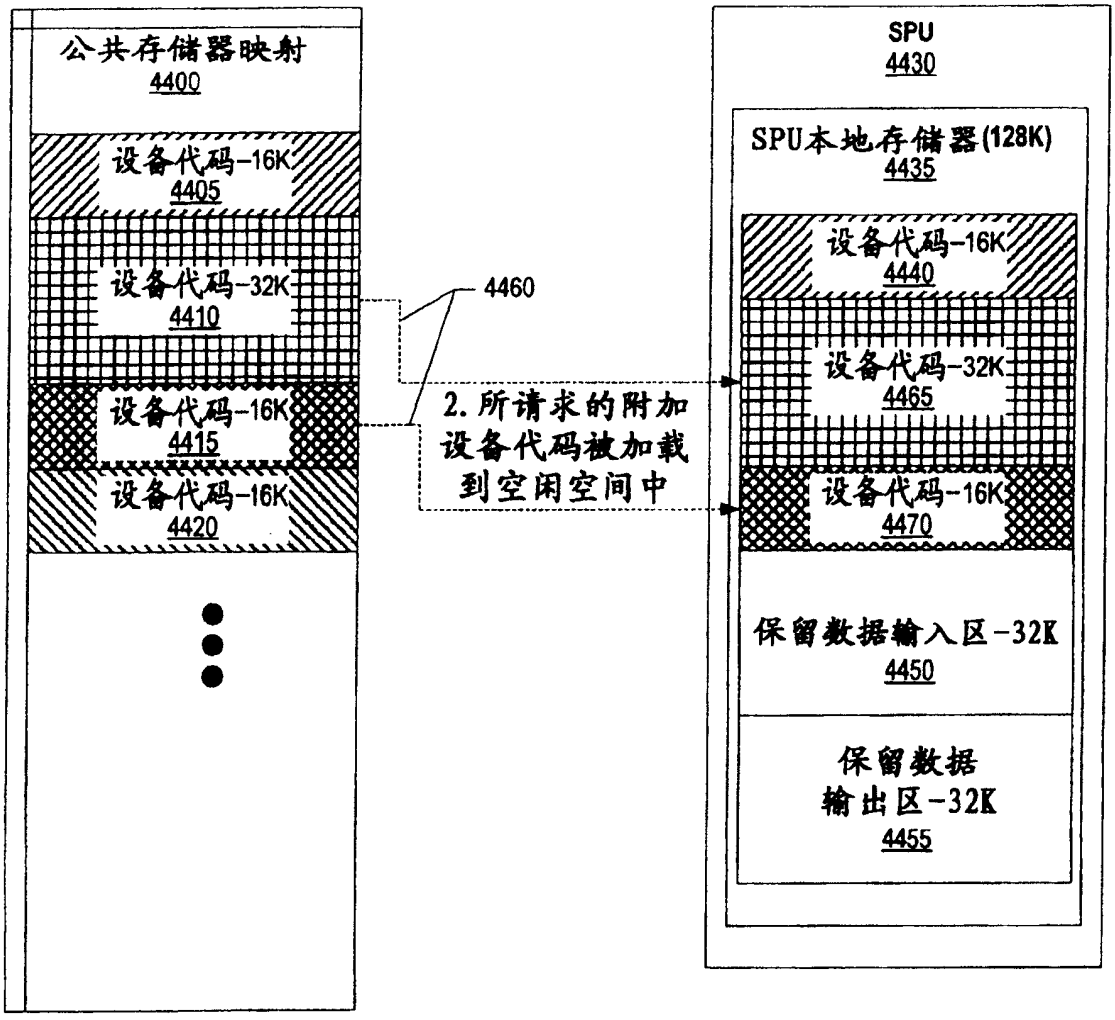

公共存储器映射
4400
设备代码-16K
4405
设备代码-32K
4410
设备代码-16K
4415
设备代码-16K
4420
4460
2. 所请求的附加设备代码被加载到空闲空间中
SPU
4430
SPU本地存储器(128K)
4435
设备代码-16K
4440
设备代码-32K
4465
设备代码-16K
4470
保留数据输入区-32K
4450
保留数据输出区-32K
4455

图46

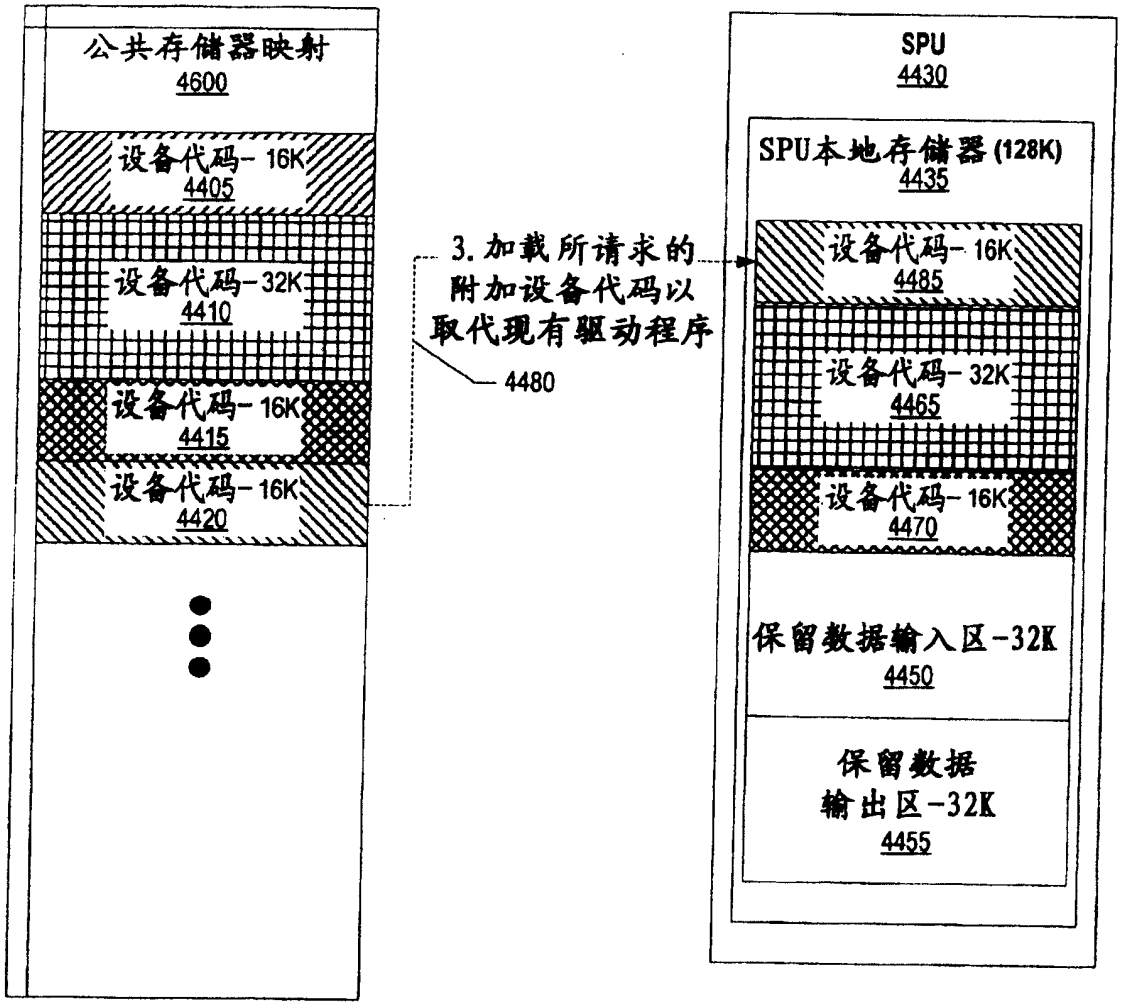

公共存储器映射
4600
设备代码- 16K
4405
设备代码-32K
4410
设备代码- 16K
4415
设备代码- 16K
4420
3. 加载所请求的
附加设备代码以
取代现有驱动程序
4480
SPU
4430
SPU本地存储器(128K)
4435
设备代码- 16K
4485
设备代码- 32K
4465
设备代码- 16K
4470
保留数据输入区-32K
4450
保留数据
输出区-32K
4455

图47

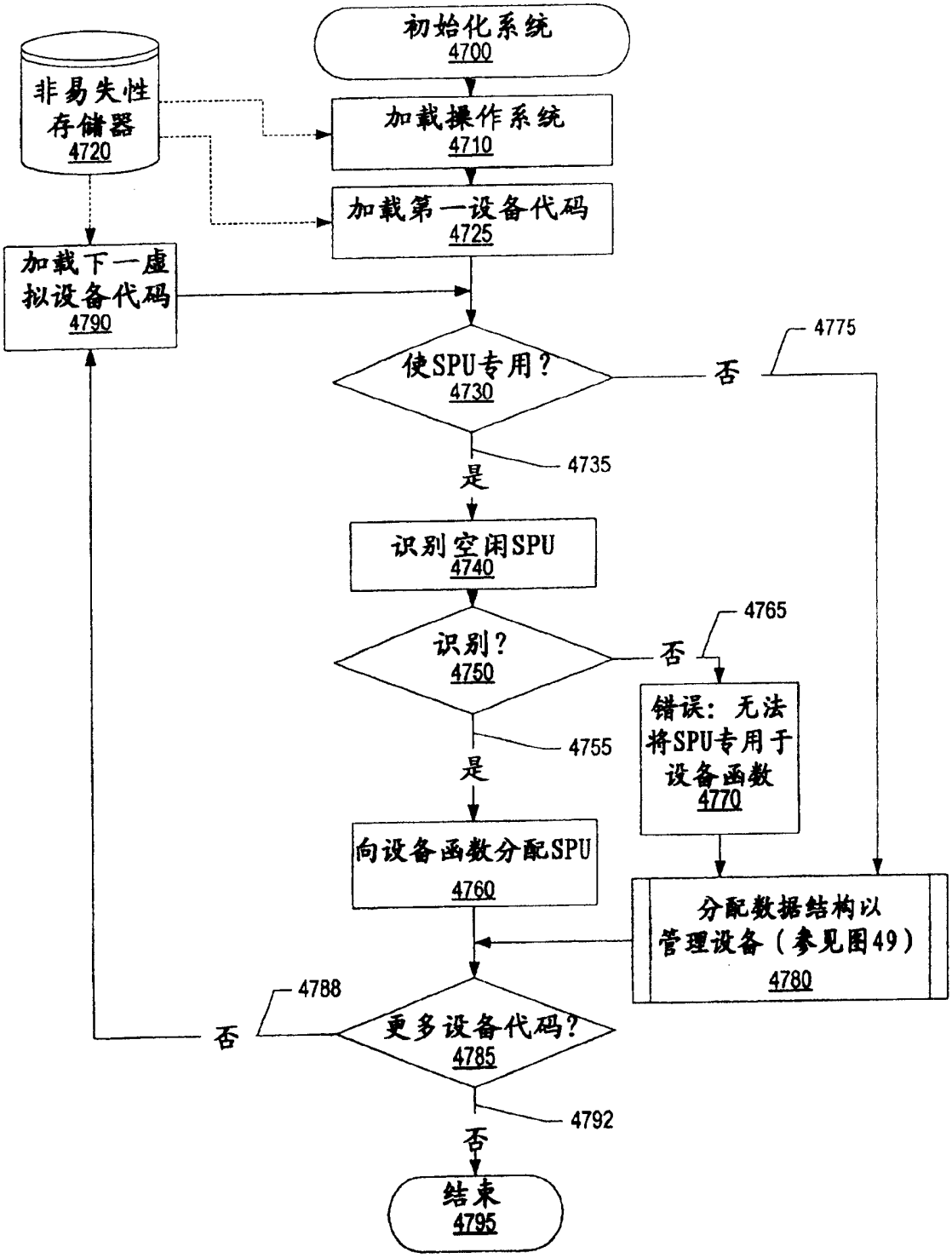

初始化系统
4700
非易失性存储器
4720
加载操作系统
4710
加载第一设备代码
4725
加载下一虚拟设备代码
4790
使SPU专用?
4730
否
4775
是
4735
识别空闲SPU
4740
识别?
4750
否
4765
错误：无法将SPU专用于设备函数
4770
是
4755
向设备函数分配SPU
4760
分配数据结构以管理设备（参见图49）
4780
更多设备代码?
4785
否
4788
否
4792
结束
4795

图48

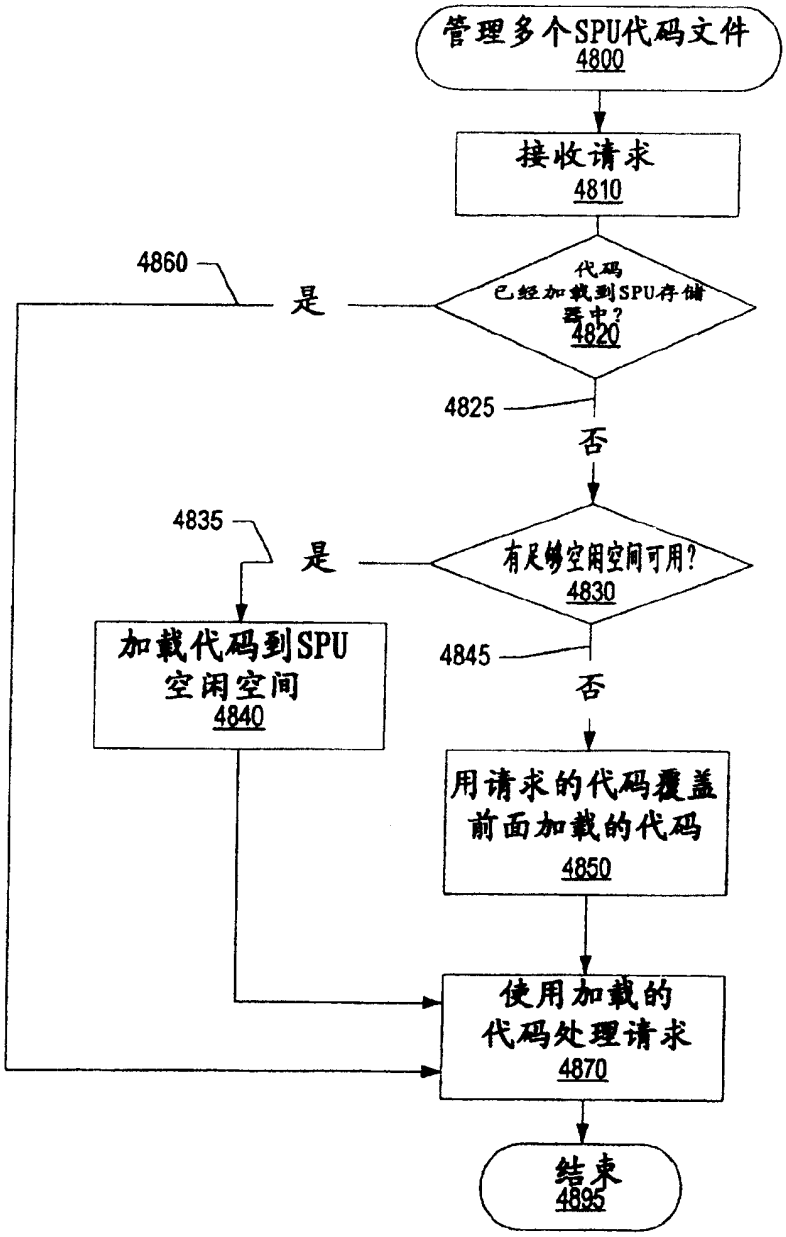

管理多个SPU代码文件
4800
接收请求
4810
代码已经加载到SPU存储器中?
4820
4860
是
4825
否
有足够空闲空间可用?
4830
4835
是
加载代码到SPU空闲空间
4840
4845
否
用请求的代码覆盖前面加载的代码
4850
使用加载的代码处理请求
4870
结束
4895

图49

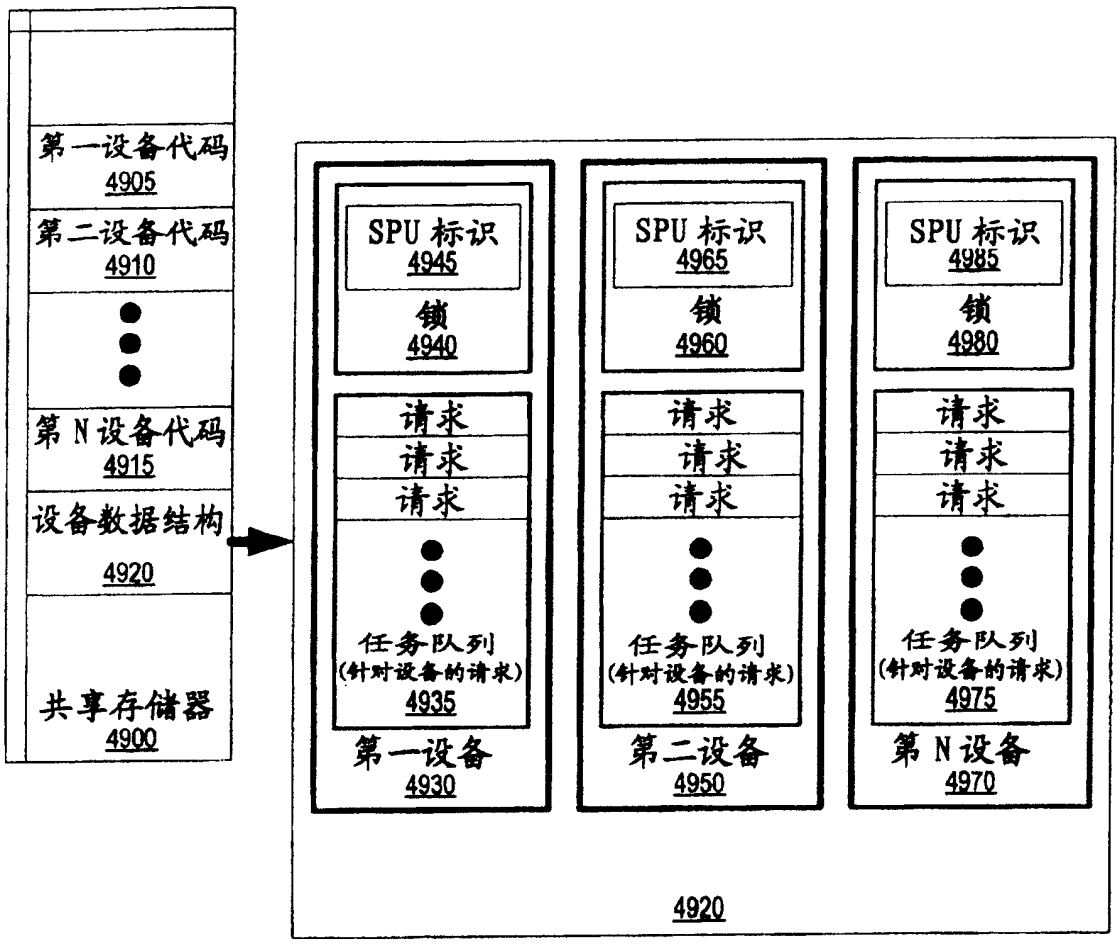

第一设备代码
4905
第二设备代码
4910
第N设备代码
4915
设备数据结构
4920
共享存储器
4900
SPU 标识
4945
锁
4940
请求
请求
请求
任务队列
(针对设备的请求)
4935
第一设备
4930
SPU 标识
4965
锁
4960
请求
请求
请求
任务队列
(针对设备的请求)
4955
第二设备
4950
SPU 标识
4985
锁
4980
请求
请求
请求
任务队列
(针对设备的请求)
4975
第N设备
4970
4920

图50

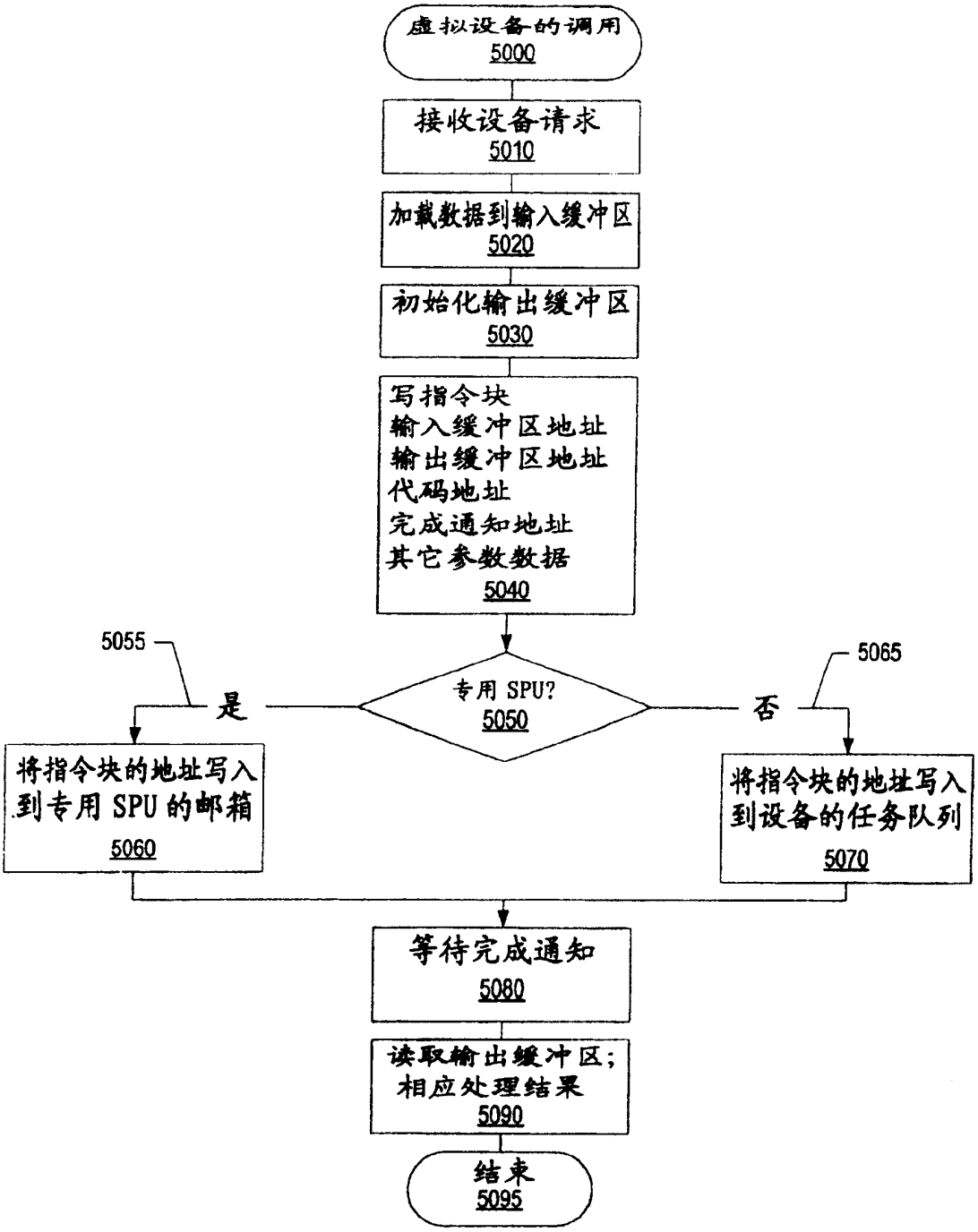

虚拟设备的调用
5000
接收设备请求
5010
加载数据到输入缓冲区
5020
初始化输出缓冲区
5030
写指令块
输入缓冲区地址
输出缓冲区地址
代码地址
完成通知地址
其它参数数据
5040
5055
是
专用 SPU?
5050
否
5065
将指令块的地址写入到专用 SPU 的邮箱
5060
将指令块的地址写入到设备的任务队列
5070
等待完成通知
5080
读取输出缓冲区；相应处理结果
5090
结束
5095

图51

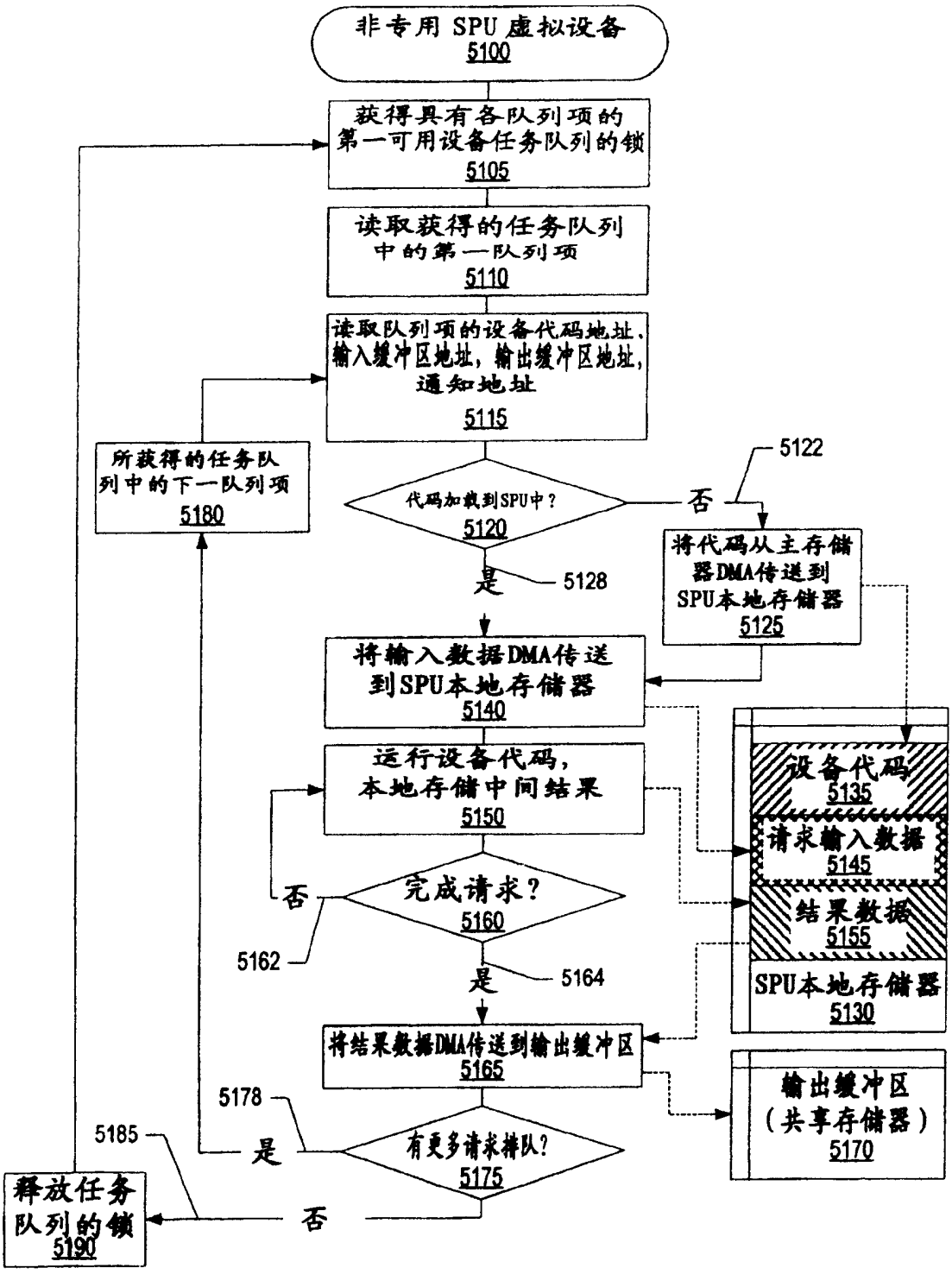

非专用 SPU 虚拟设备
5100
获得具有各队列项的第一可用设备任务队列的锁
5105
读取获得的任务队列中的第一队列项
5110
读取队列项的设备代码地址，输入缓冲区地址，输出缓冲区地址，通知地址
5115
所获得的任务队列中的下一队列项
5180
代码加载到SPU中？
5120
否
5122
是
5128
将代码从主存储器DMA传送到SPU本地存储器
5125
将输入数据DMA传送到SPU本地存储器
5140
运行设备代码，本地存储中间结果
5150
完成请求？
5160
否
5162
是
5164
将结果数据DMA传送到输出缓冲区
5165
有更多请求排队？
5175
5178
是
5185
否
释放任务队列的锁
5190
设备代码
5135
请求输入数据
5145
结果数据
5155
SPU本地存储器
5130
输出缓冲区（共享存储器）
5170

图 52

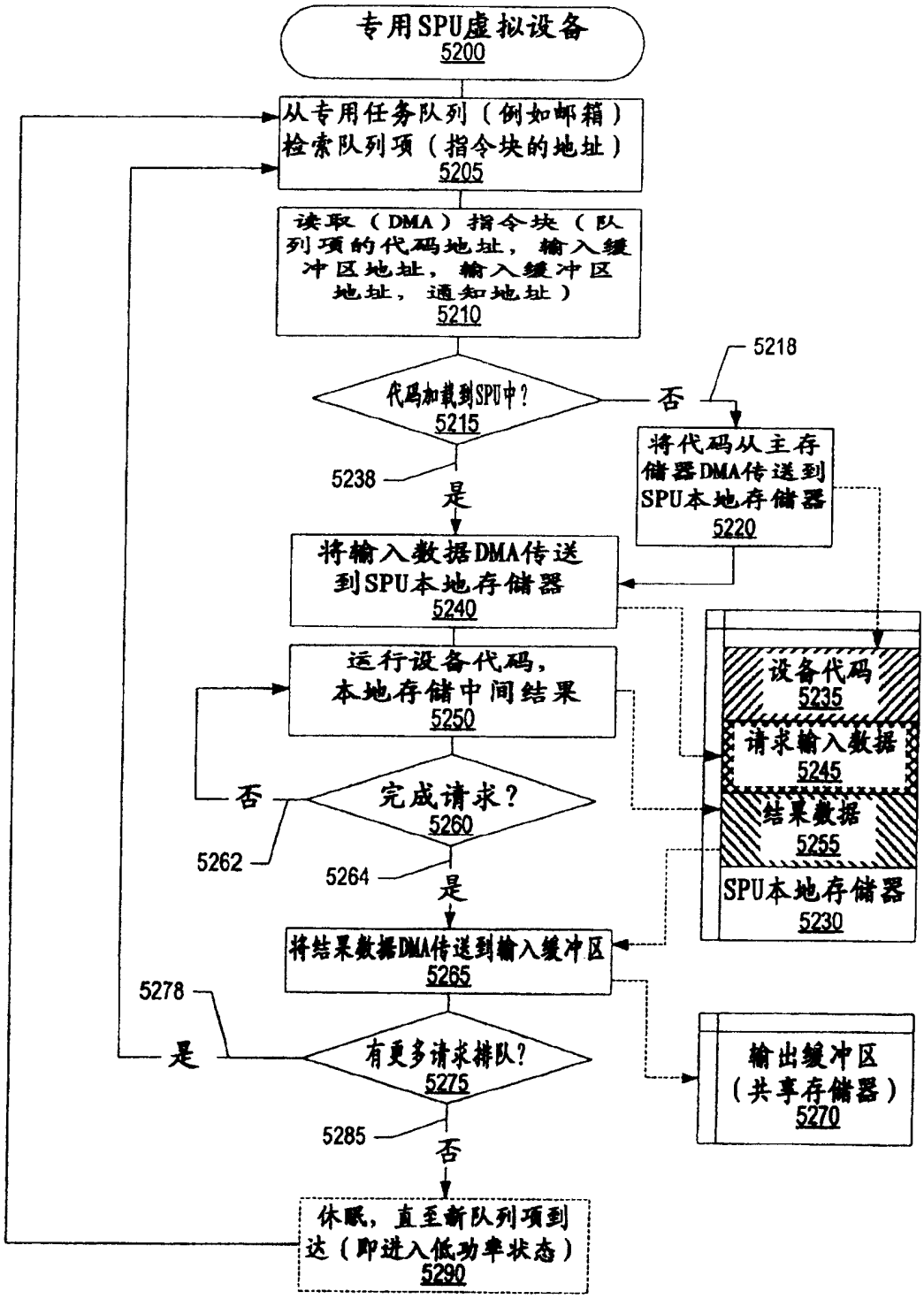

专用SPU虚拟设备
5200
从专用任务队列（例如邮箱）检索队列项（指令块的地址）
5205
读取（DMA）指令块（队列项的代码地址，输入缓冲区地址，输入缓冲区地址，通知地址）
5210
代码加载到SPU中？
5215
否
5218
将代码从主存储器DMA传送到SPU本地存储器
5220
5238
是
将输入数据DMA传送到SPU本地存储器
5240
运行设备代码，本地存储中间结果
5250
完成请求？
5260
否
5262
5264
是
将结果数据DMA传送到输入缓冲区
5265
5278
是
有更多请求排队？
5275
5285
否
休眠，直至新队列项到达（即进入低功率状态）
5290
设备代码
5235
请求输入数据
5245
结果数据
5255
SPU本地存储器
5230
输出缓冲区（共享存储器）
5270

图53

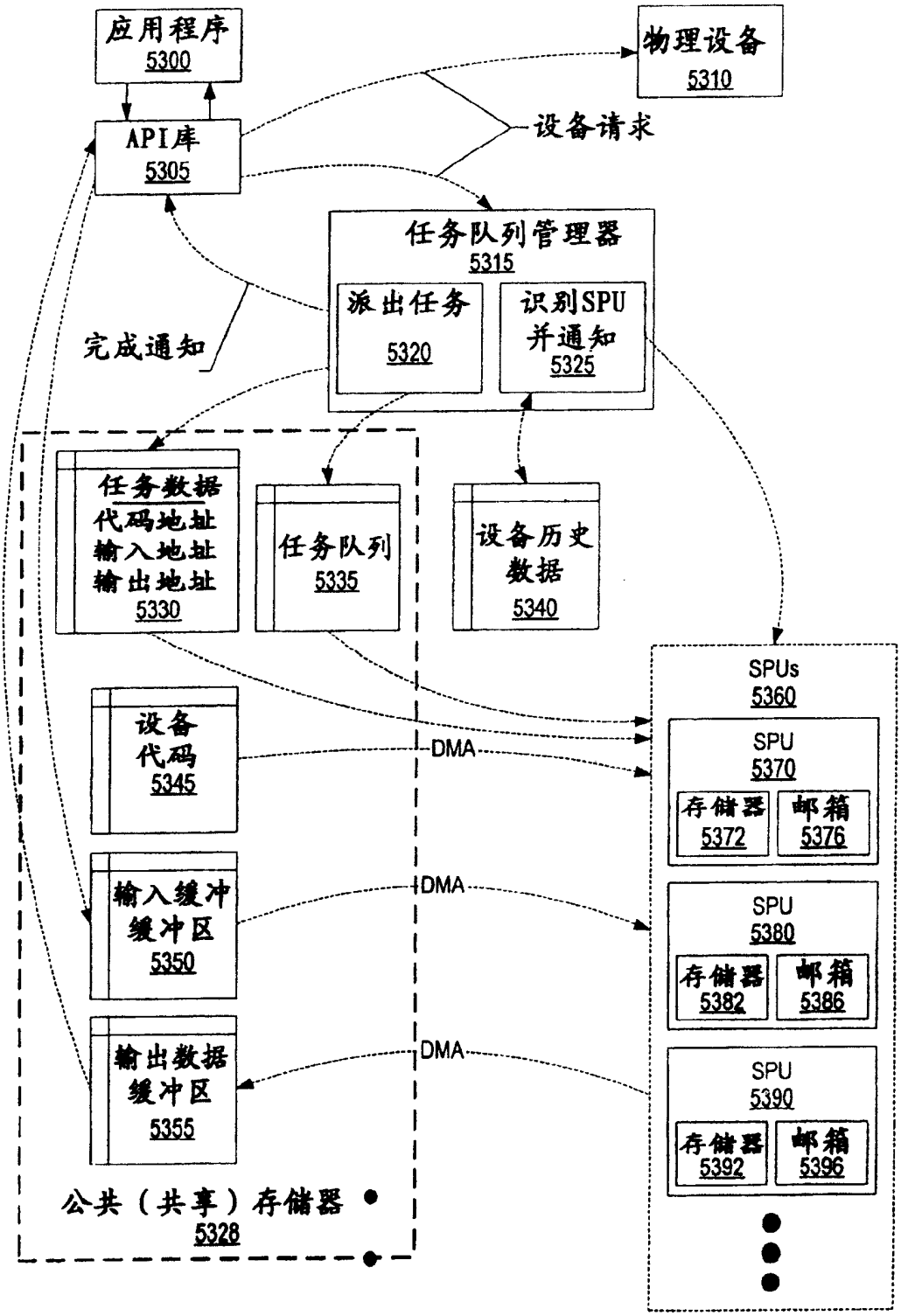

应用程序
5300
物理设备
5310
设备请求
API库
5305
任务队列管理器
5315
派出任务
5320
识别SPU
并通知
5325
完成通知
任务数据
代码地址
输入地址
输出地址
5330
任务队列
5335
设备历史
数据
5340
SPUs
5360
SPU
5370
存储器
5372
邮箱
5376
设备
代码
5345
DMA
输入缓冲
缓冲区
5350
SPU
5380
存储器
5382
邮箱
5386
输出数据
缓冲区
5355
SPU
5390
存储器
5392
邮箱
5396
公共（共享）存储器
5328

图54

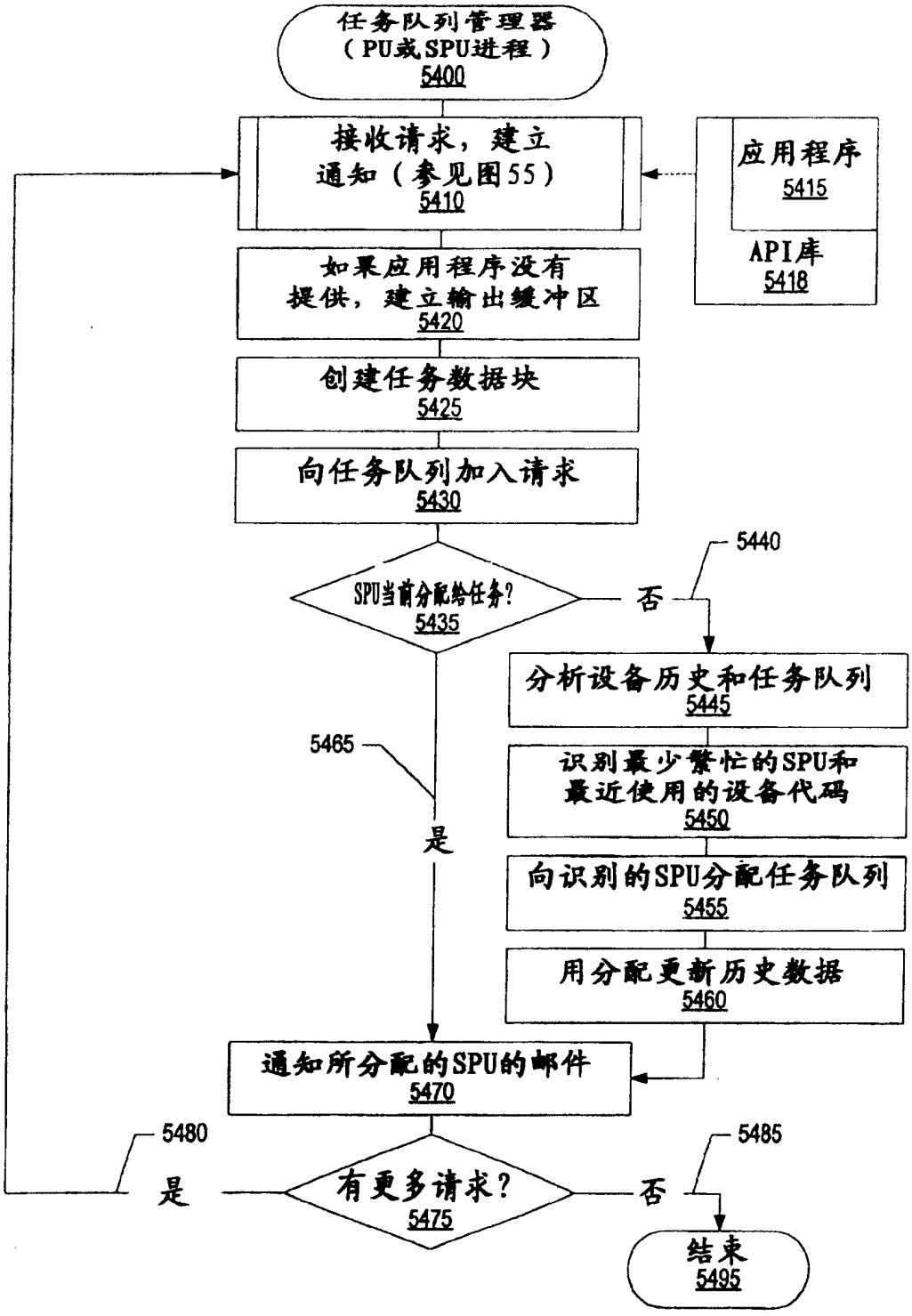

任务队列管理器
（PU或SPU进程）
5400
接收请求，建立
通知（参见图55）
5410
应用程序
5415
API库
5418
如果应用程序没有
提供，建立输出缓冲区
5420
创建任务数据块
5425
向任务队列加入请求
5430
SPU当前分配给任务?
5435
否
5440
分析设备历史和任务队列
5445
识别最少繁忙的SPU和
最近使用的设备代码
5450
向识别的SPU分配任务队列
5455
用分配更新历史数据
5460
5465
是
通知所分配的SPU的邮件
5470
有更多请求?
5475
否
5485
结束
5495
5480
是

图55

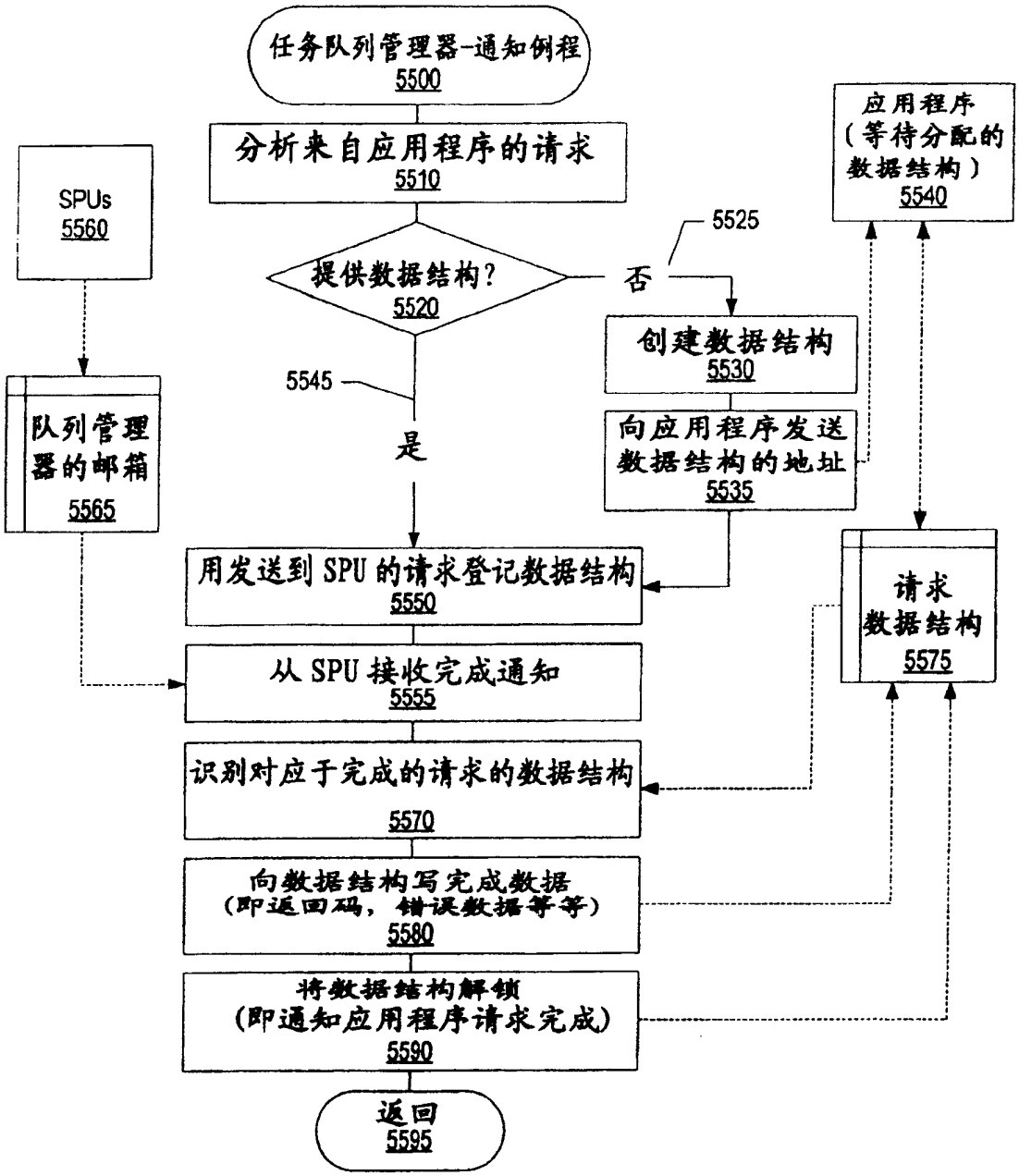

任务队列管理器-通知例程
5500
分析来自应用程序的请求
5510
提供数据结构?
5520
否
5525
是
5545
创建数据结构
5530
向应用程序发送数据结构的地址
5535
应用程序（等待分配的数据结构）
5540
SPUs
5560
队列管理器的邮箱
5565
用发送到 SPU 的请求登记数据结构
5550
从 SPU 接收完成通知
5555
识别对应于完成的请求的数据结构
5570
向数据结构写完成数据（即返回码，错误数据等等）
5580
将数据结构解锁（即通知应用程序请求完成）
5590
返回
5595
请求数据结构
5575

图56

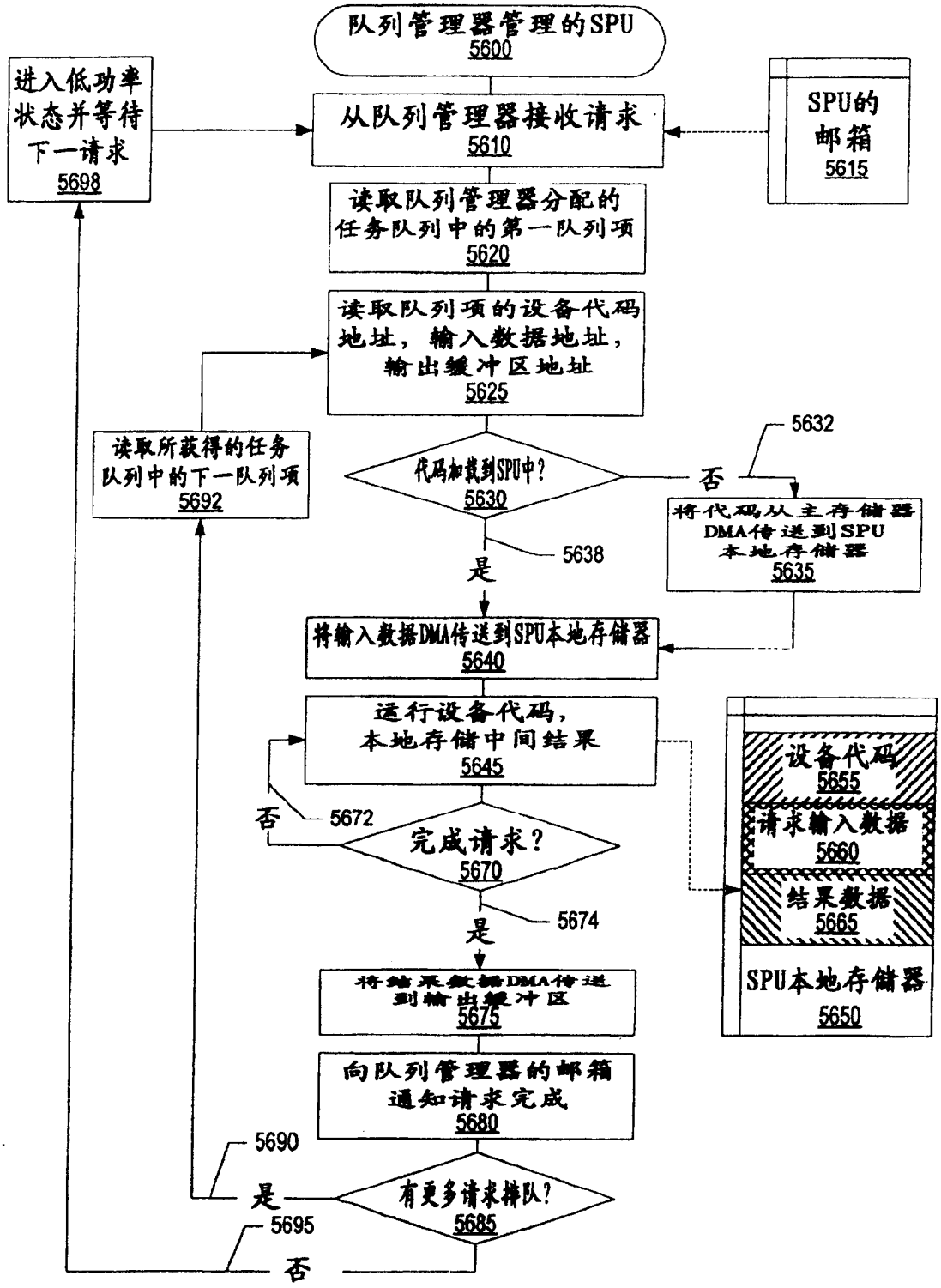
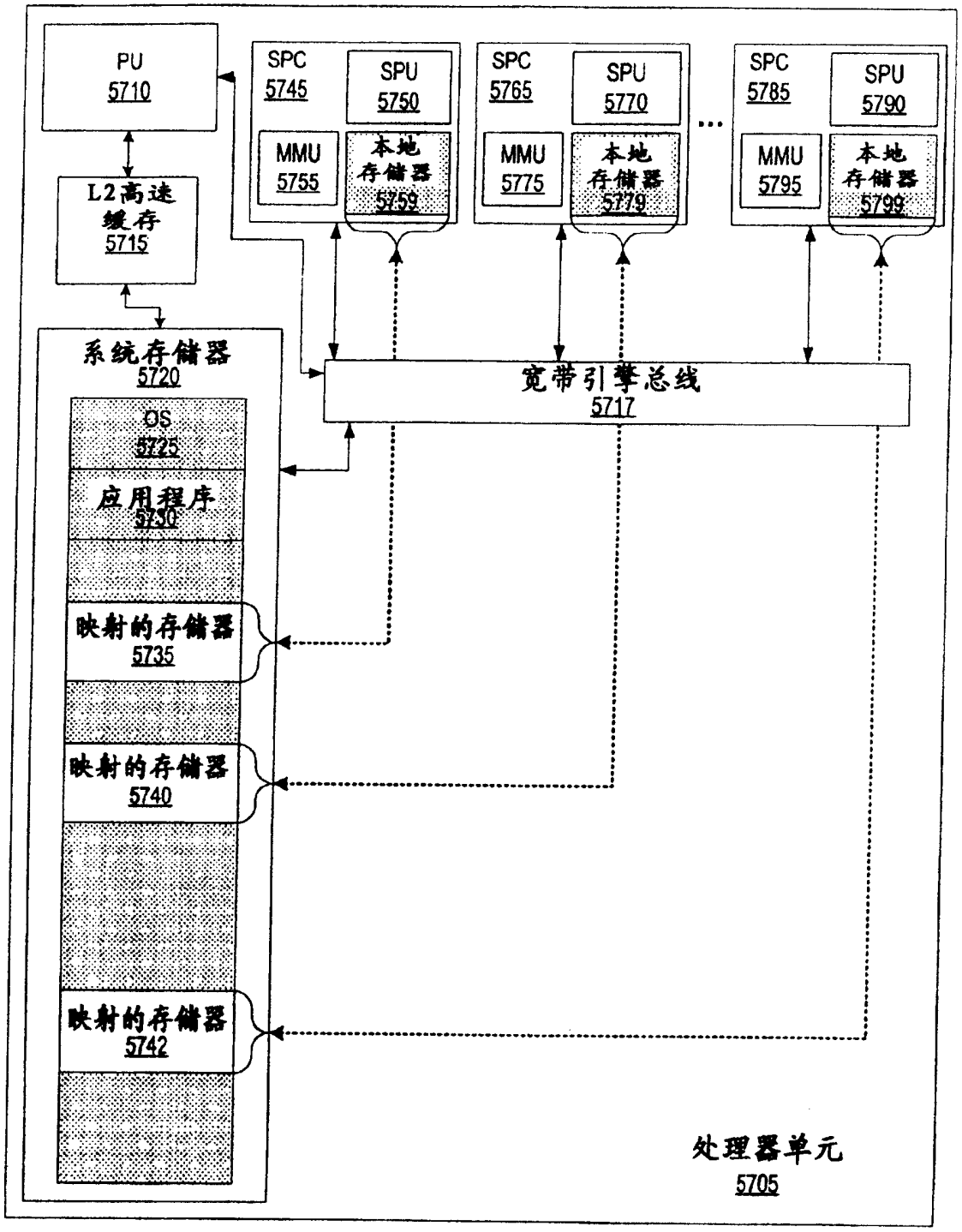

队列管理器管理的SPU
5600
进入低功率状态并等待下一请求
5698
从队列管理器接收请求
5610
SPU的邮箱
5615
读取队列管理器分配的任务队列中的第一队列项
5620
读取队列项的设备代码地址，输入数据地址，输出缓冲区地址
5625
读取所获得的任务队列中的下一队列项
5692
代码加载到SPU中?
5630
否
5632
将代码从主存储器DMA传送到SPU本地存储器
5635
5638
是
将输入数据DMA传送到SPU本地存储器
5640
运行设备代码，本地存储中间结果
5645
否
5672
完成请求?
5670
5674
是
将结果数据DMA传送到输出缓冲区
5675
向队列管理器的邮箱通知请求完成
5680
5690
是
有更多请求排队?
5685
5695
否
设备代码
5655
请求输入数据
5660
结果数据
5665
SPU本地存储器
5650

图57


PU
5710
L2高速
缓存
5715
SPC
5745
SPU
5750
MMU
5755
本地
存储器
5759
SPC
5765
SPU
5770
MMU
5775
本地
存储器
5779
SPC
5785
SPU
5790
MMU
5795
本地
存储器
5799
系统存储器
5720
宽带引擎总线
5717
OS
5725
应用程序
5730
映射的存储器
5735
映射的存储器
5740
映射的存储器
5742
处理器单元
5705