[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/46
G06F 15/76

[12] 发明专利申请公开说明书

[21] 申请号 200410011719.1

[43] 公开日 2005 年 3 月 30 日

[11] 公开号 CN 1601474A

[22] 申请日 2004.9.24
[21] 申请号 200410011719.1
[30] 优先权
[32] 2003. 9.26 [33] JP [31] 335498/2003
[71] 申请人 株式会社东芝
地址 日本东京都
[72] 发明人 金井达德 前田诚司 矢野浩邦
吉井谦一郎

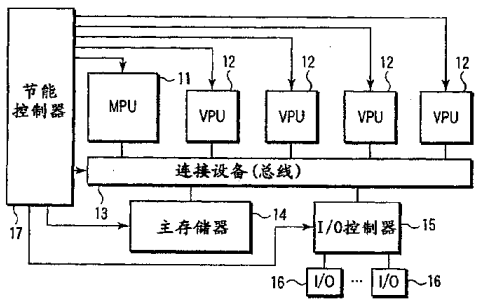
[74] 专利代理机构 中国国际贸易促进委员会专利商标事务所
代理人 马 浩

权利要求书 3 页 说明书 46 页 附图 37 页

[54] 发明名称 执行实时操作的方法和系统

[57] 摘要

一个信息处理系统在一个特定时间间隔内执行多个任务。该系统包括：一条总线；多个通过总线传输数据的处理器；以及用于执行一个时序安排操作的一个单元，该时序安排操作根据关于执行每个任务所需的一个时间的花费信息和关于每个任务所需的一个数据传输带宽的带宽信息确定每个任务的执行开始时间以及至少一个执行任务的处理器，以便在不交叠任务中的至少两个任务的执行期的情况下在特定时间间隔内执行任务，这两个任务要求的数据传输带宽不小于其他任务要求的数据传输带宽。



ISSN1008-4274

1、一种使用通过一条总线传输数据的多个处理器在一个特定时间间隔内执行多个任务的方法，该方法包括:

输入关于执行每个任务所需的一个时间的花费信息和关于每个任务所需的一个数据传输带宽的带宽信息；以及

执行一个时序安排操作，根据输入的花费信息和带宽信息确定每个任务的执行开始时间以及至少一个执行所述多个任务的处理器，以便在不交叠所述多个任务中的至少两个任务的执行期的情况下在所述特定时间间隔内执行所述多个任务，所述两个任务要求的数据传输带宽不小于所述多个任务的其他任务要求的数据传输带宽。

2、根据权利要求 1 的方法，进一步包括:

根据时序安排操作的结果和带宽信息，计算由所述至少一个处理器在所述特定时间间隔内执行数据传输的数据传输带宽的一个峰值；以及

根据计算出的峰值与所述总线的最大数据传输带宽的一个比率，将所述总线的数据传输速率设置为低于所述最大数据传输带宽。

3、根据权利要求 2 的方法，其中所述设置包括控制总线的一个操作频率。

4、在一个特定时间间隔内执行多个任务的一个信息处理系统，包括:

一条总线;

通过所述总线传输数据的多个处理器；以及

用于执行一个时序安排操作的装置，该时序安排操作根据关于执行每个任务所需的一个时间的花费信息和关于每个任务所需的一个数据传输带宽的带宽信息，确定每个任务的执行开始时间以及至少一个执行所述多个任务的处理器，以便在不交叠所述多个任务中的至少两个任务的执行期的情况下，在所述特定时间间隔内执行所述多个任务，所述两个任务要求的数据传输带宽不小于所述多个任务的其他任务要

求的数据传输带宽。

5、根据权利要求 4 的信息处理系统，进一步包括:

用于根据时序安排操作的结果和带宽信息，计算所述至少一个处理器在所述特定时间间隔内执行数据传输的数据传输带宽的一个峰值的装置；以及

用于根据计算出的峰值与所述总线的最大数据传输带宽的一个比率，将所述总线的数据传输速率设置为低于所述最大数据传输带宽的装置。

6、根据权利要求 5 的信息处理系统，其中用于设置数据传输速率的装置包括用于控制总线的一个操作频率的装置。

7、根据权利要求 4 的信息处理系统，其中总线包括一个相互连接的网络。

8、根据权利要求 4 的信息处理系统，进一步包括一个连接到所述总线的存储器，以及

其中所述多个处理器被配置为通过所述总线在所述多个处理器和所述存储器之间传输数据。

9、在一个特定时间间隔内执行多个任务的一个信息处理系统，包括:

一条总线;

通过所述总线传输数据的多个第一处理器；以及

用于执行一个时序安排操作的一个第二处理器，该时序安排操作根据关于执行每个任务所需的一个时间的花费信息和关于每个任务所需的一个数据传输带宽的带宽信息，确定每个任务的执行开始时间以及至少一个执行所述多个任务的第一处理器，以便在不交叠所述多个任务中的至少两个任务的执行期的情况下，在所述特定时间间隔内执行所述多个任务，所述两个任务要求的数据传输带宽不小于所述多个任务的其他任务要求的数据传输带宽。

10、根据权利要求 9 的信息处理系统，其中第二处理器被配置为根据时序安排操作的结果和带宽信息，计算所述至少一个第一处理器

在所述特定时间间隔内执行数据传输的数据传输带宽的一个峰值，并且信息处理系统进一步包括一个数据传输速率控制单元，其被配置为根据计算出的峰值与所述总线的最大数据传输带宽的一个比率将总线的数据传输速率设置为低于所述最大数据传输带宽。

11、根据权利要求 10 的信息处理系统，其中数据传输速率控制单元被配置为控制总线的一个操作频率。

12、根据权利要求 9 的信息处理系统，其中总线包括一个相互连接的网络。

13、根据权利要求 9 的信息处理系统，进一步包括一个连接到总线的存储器，以及

其中所述多个第一处理器被配置为通过总线在所述多个第一处理器和所述存储器之间传输数据。

执行实时操作的方法和系统

相关申请

本申请是基于 2003 年 9 月 26 日提出的现有日本专利申请 2003-335498，并且要求该申请的优先权，并且该申请的全部内容在此处被结合进来做为参考。

技术领域

本发明涉及用于在特定的时间间隔周期性地执行一个实时操作的一个时序安排方法和一个信息处理系统。

背景技术

通常，为了提高吞吐量，诸如服务器计算机这样的计算机系统利用了某种系统结构，例如一个多处理器和一个并行处理器。这两种处理器都用多个处理单元实现了一个并行计算操作。

日本专利申请 KOKAI 公开 10-143380 揭示了一个具有多个处理单元的系统。此系统包括一个高速 CPU、多个低速 CPU 和一个共享的存储器。进程是考虑并行和每个进程的执行时间被分配给高速和低速 CPU 的。

日本专利申请 KOKAI 公开 8-180025 揭示了一个时序安排技术，该技术对线程进行时序安排，以便相同的处理器执行属于相同进程的诸个线程。

近来，不仅计算机系统，一个需要实时处理大量诸如 AV（音频视频）数据这样的数据的嵌入式设备也要求引入诸如一个多处理器和一个并行处理器的系统结构，以提高吞吐量。

但是，在目前的情况下，几乎没有关于以上系统结构中所宣称的实时处理系统的报告。

在实时处理系统中，每个操作需要在允许时间的限制之内完成。为了在特定的时间间隔周期性地执行一个包括多个链接的任务的组合的实时操作，所有链接的任务都需要在每个周期的时间间隔内完成。

由于实时处理系统常被用作一个嵌入式系统，因此其重要问题是降低功耗。系统中包括的处理单元的数目越大，则需要的数据传输速度（数据传输带宽）越高。数据传输带宽越大，则功耗越大。当诸如一个多处理器和一个并行处理器的系统结构被应用到实时处理系统时，在一个给定时间周期内完成一个实时操作的同时，需要一个新的机制来减少所需的数据传输带宽。

发明内容

本发明的一个目标是提供一种方法和一个信息处理系统，它们能在不损害任何实时操作的情况下减少所需的数据传输带宽。

根据本发明的一个实施方式，提供了一种方法，该方法用多个处理器在一个特定的时间间隔内执行多个任务，其中多个处理器通过一条总线传输数据，该方法包括输入关于执行每个任务所需的时间的花费信息以及关于每个任务所需的数据传输带宽的带宽信息，并且执行一个时序安排操作，根据输入花费信息和带宽信息，确定每个任务的执行开始时间以及至少一个执行所述任务的处理器，以便在不交叠多个任务中的至少两个任务的执行时间的情况下，在特定的时间间隔内执行任务，所述两个任务要求的数据传输带宽不少于其他任务要求的数据传输带宽。

附图说明

图1是显示配置根据本发明的一个实施方式一个实时处理系统的一个计算机系统的一个例子的框图。

图 2 是在根据本发明的实施方式的实时处理系统中提供的一个MPU（主处理单元）和VPU（通用处理单元）的框图。

图3是显示用于根据本发明的实施方式的实时处理系统中的一个

虚拟地址翻译机制的一个例子的图。

图4是显示映射在根据本发明的实施方式的实时处理系统中的实际地址空间中的数据的一个例子的图。

图5描述了根据本发明的实施方式的实时处理系统中的有效地址空间、虚拟地址空间和实际地址空间。

图6是一个用于数字电视广播的接收机的框图。

图7是显示由根据本发明的实施方式的实时处理系统执行的一个程序模块的一个例子的图。

图8是显示包含在图7所示的程序模块中的一个结构描述的一个例子的表。

图9是显示对应于图7所示的程序模块的程序间的数据流动的图。

图10是显示图7所示的程序模块的一个并行操作的图，该并行操作是由两个VPU执行的。

图11是显示图7所示的程序模块的一个流水线操作的图，该流水线操作是由两个VPU执行的。

图12是显示一个实时操作的每个任务的一个执行期和一个要求的数据传输带宽之间的一个关系的图。

图13是显示时序安排的一个例子的图，该时序安排考虑到了由每个任务所要求的一个数据传输带宽，以便在一个周期内尽可能统一要求的数据传输带宽。

图14是显示由根据本发明的实施方式的实时处理系统执行的一个节能控制操作的步骤的一个例子的流程图。

图15是由一个VPU周期性地执行一个实时操作的线程的时序安排的图。

图16是由两上VPU同时执行两个实时操作的时序安排的一个例子的图。

图17是根据本发明的实施方式的时序安排方法由两个VPU同时执行两个实时操作的时序安排的一个例子的图。

图 18 是显示根据本发明的实施方式的实时处理系统中的一个操作系统的一个例子的图。

图 19 是显示根据本发明的实施方式的实时处理系统中的一个操作系统的另一个例子的图。

图 20 是显示在根据本发明的实施方式的实时处理系统中的一个虚拟机 OS 和一个子 OS 之间的一个关系的图。

图 21 是显示被时分分配给根据本发明的实施方式的实时处理系统中的多个子 OS 的资源的图。

图 22 是显示由根据本发明的实施方式的实时处理系统中的一个特定子 OS 占用的特定资源的图。

图 23 是用作本发明的实施方式的实时处理系统中的一个时序安排器的 VPU 运行环境的图。

图 24 是显示在用于本发明的实施方式的实时处理系统中的虚拟机 OS 中实现的 VPU 运行环境的一个例子的图。

图 25 是显示在用于本发明的实施方式的实时处理系统中的子 OS 中实现的 VPU 运行环境的一个例子的图。

图 26 是显示在用于本发明的实施方式的实时处理系统中的每个子 OS 中实现的 VPU 运行环境的一个例子的图。

图 27 是显示在用于本发明的实施方式的实时处理系统中的一个子 OS 中实现的 VPU 运行环境的一个例子的图。

图 28 描述了用于根据本发明的实施方式的实时处理系统中的 MPU 侧 VPU 运行环境和 VPU 侧 VPU 运行环境。

图 29 是显示由用于根据本发明的实施方式的实时处理系统中的 VPU 侧 VPU 运行环境执行的一个程序的流程图。

图 30 是显示由用于根据本发明的实施方式的实时处理系统中的 MPU 侧 VPU 运行环境执行的一个程序的流程图。

图 31 描述了在根据本发明的实施方式的实时处理系统中，属于一个紧密耦合线程群组并且被不同处理器执行的线程。

图 32 描述了在根据本发明的实时处理系统中的紧密耦合线程之

间的交互作用。

图 33 描述了在根据本发明的实施方式的实时处理系统中，执行伙伴线程的 VPU 的本地存储在紧密耦合线程的有效地址空间中的映射。

图 34 描述了在根据本发明的实施方式的实时处理系统中属于一个松散耦合线程群组的线程的处理器分配。

图 35 描述了根据本发明的实施方式的实时处理系统中的松散耦合线程之间的交互作用。

图 36 描述了根据本发明的实施方式的实时处理系统中的进程和线程之间的一个关系。

图 37 是显示在根据本发明的实施方式的实时处理系统中用于执行一个时序安排操作的一个程序的流程图。

图 38 是显示在根据本发明的实施方式的实时处理系统中的线程的一个状态转换的图。

图 39 是描述在根据本发明的实施方式的实时处理系统中的一个线程及其执行期之间的一个关系的图。

图 40 是在根据本发明的实施方式的实时处理系统中在一个执行期中同时运行的紧密耦合线程的图。

图 41 是显示在根据本发明的实施方式的实时处理系统中的一个周期执行模型的图。

图 42 是显示在根据本发明的实施方式的实时处理系统中的一个非周期执行模型的图。

图 43 描述了一个任务图。

图 44 描述了用于根据本发明的实施方式的实时处理系统中的一个预定图的原理。

图 45 描述了用于根据本发明的实施方式的实时处理系统中的一个预定图的一个例子。

图 46 是描述用于根据本发明的实施方式的实时处理系统中的一个分级时序安排器的图。

图 47 是描述由根据本发明的实施方式的实时处理系统用于硬实时类中的时序安排的参数的例子的图。

图 48 描述了用于根据本发明的实施方式的实时处理系统中的绝对时间限制。

图 49 描述了用于根据本发明的实施方式的实时处理系统中的相对时间限制。

图 50 描述了用于根据本发明的实施方式的实时处理系统中的互斥限制。

图 51 是描述根据本发明的实施方式的实时处理系统中的同步机制的表。

图 52 是显示一个用于在根据本发明的实施方式的实时处理系统中选择性地使用同步机制的程序的流程图。

图 53 是显示用于根据本发明的实施方式的实时处理系统中的一个预定图的一个例子的图。

图 54 是显示在根据本发明的实施方式的实时处理系统中创建的一个预定请求的一个例子的图。

图 55 是显示由根据本发明的实施方式的实时处理系统根据图 54 所示的预定请求执行的时序安排的一个例子的图。

图 56 是描述由根据本发明的实施方式的实时处理系统执行的软件流水线型时序安排的一个第一例子的图。

图 57 是描述由根据本发明的实施方式的实时处理系统执行的软件流水线型时序安排的一个第二例子的图。

图 58 是用于由根据本发明的实施方式的实时处理系统执行的软件流水线型时序安排的程序的流程图。

图 59 是描述由根据本发明的实施方式的实时处理系统执行的软件流水线型时序安排的一个第三例子的图。

图 60 是显示由两上 VPU 同时执行两个实时操作的时序安排的一个例子的图。

图 61 是显示在根据本发明的实施方式的实时处理系统中由两上

VPU 以流水线模式同时执行两个实时操作的时序安排的一个例子的图。

图 62 是描述通过图 61 所示的时序安排所实现的要求的带宽的减少的图。

图 63 是显示用于根据本发明的实施方式的实时处理系统中的一个具有一个分级结构的预定图的一个例子的图。

图 64 是显示用于根据本发明的实施方式的实时处理系统中的一个预定列表的一个例子的图。

图 65 是显示在根据本发明的实施方式的实时处理系统中用于预定一个执行期的一个程序的流程图。

具体实施方式

现将参照附图说明本发明的一个实施方式。

图 1 显示了实现根据本发明的一个实时处理系统的一个计算机系统的配置的一个例子。计算机系统是一个执行各种操作的信息处理系统，这些操作必须在时间限制下实时完成。计算机系统不仅可被用作一个通用计算机，也可用作各种电子设备的一个嵌入式系统，以执行需要实时完成的操作。参见图 1，计算机系统包括一个 MPU（主处理单元）11、多个 VPU（通用处理单元）12、一个连接设备 13、一个主存储器 14 和一个 I/O（输入/输出）控制器 15。MPU 11、VPU 12、主存储器 14 和 IO 控制器 15 通过连接设备彼此连接。连接设备 13 是一条包括一条总线的数据传输路径。例如，一个环形总线结构或一个互联网络（例如一个纵横交换机）可被用作总线。如果一条总线用于连接设备 13，则其形状可为一个环。MPU 11 是控制计算机系统的一个操作的一个主处理器。MPU 11 主要执行一个 OS（操作系统）。VPU 12 和 IO 控制器 15 可执行 OS 的某些功能。每个 VPU 12 是用于在 MPU 11 的控制下执行多种操作的一个处理器。MPU 11 将操作(任务）分配给 VPU 12，以便并行地执行这些操作（任务）。从而能够高速并高效地执行操作。主存储器 14 是被 MPU 11、VPU 12 和 I/O 控制器

15 共享的一个存储设备（共享存储器）。主存储器 14 存储 OS 和应用程序。I/O 控制器 15 连接到一个或多个 I/O 设备 16。控制器 15 也被称为一个桥接设备。

连接设备 13 具有一个 QoS（服务质量）功能，它保证一个数据传输速率。QoS 功能是通过以一个预定的带宽（传输速率）通过连接设备 13 传输数据来实现的。当写数据以 5 Mbps 的速率从一个 VPU 12 传输到存储器 14 或当其以 100 Mbps 的速率从一个 VPU 12 传输到另一个 VPU 12 时，使用 Qos 功能。每个 VPU 12 指定（预定）连接设备 13 的一个带宽（传输速率）。连接设备 13 按优先级将指定的带宽分配给 VPU 12。如果为一个 VPU 12 的数据传输预定了一个带宽，则即使另一个 VPU 12、MPU 11 或 IO 控制器 15 在前一 VPU 12 的数据传输期间传输大量数据，该带宽也会被保证。QoS 功能对于执行实时操作的计算机尤其重要。

图 1 所示的计算机系统包括一个 MPU 11，四个 VPU 12，一个存储器 14 和一个 IO 控制器 15。VPU 12 的数目不受限制。系统不需要包括 MPU，在此情况下，一个 VPU 12 执行 MPU 11 的操作。换句话说，一个 VPU 12 用作一个虚拟 MPU 11。

计算机系统还包括一个节能控制器 17。此控制 17 完成以下功能，以降低整个或部分系统的功耗。

1. 降低整个计算机系统的时钟频率。

2. 降低整个计算机系统的供电电压。

3. 关闭整个计算机系统的电源。

4. 降低一个或多个模块（MPU、VPU、存储器、I/O 控制器等）的时钟频率。

5. 降低一个或多个模块（MPU、VPU、存储器、I/O 控制器等）的供电电压。

6. 关闭一个或多个模块（MPU、VPU、存储器、I/O 控制器等）的电源。

7. 降低连接设备的时钟频率（操作频率）。

8. 降低连接设备的传输速率。

9. 减小连接设备的带宽。

10. 关闭连接设备的电源。

11. 关闭存储器库的单元的电源。

12. 停止刷新存储器库的单元。

13. 减少 MPU 和 VPU 中同时操作的功能模块。（如果处理器包括多个操作单元，则限制同时使用的操作单元数目。）

以上节能功能可在软件控制下完成。节能功能 1 至 13 可单独或结合被完成。

图 2 显示了一个 MPU 11 和多个 VPU 12。MPU 11 包括一个处理单元 21 和一个存储器管理单元 22。处理单元 21 通过存储器管理单元 22 访问存储器 14。存储器管理单元 22 执行一个虚拟存储器管理功能，并且管理存储器管理单元 22 中的一个高速缓冲存储器。每个 VPU 12 包括一个处理单元 31，一个本地存储（本地存储器）32 和一个存储器控制器 33。处理单元 31 可具有对同一 VPU 12 中的本地存储 32 的直接访问权限。存储器控制器 33 用作一个 DMA（直接存储器访问）控制器，它在本地存储 32 和存储器 14 之间传输数据。存储器控制器 33 利用连接设备 13 的 Qos 功能，并且具有指定一个带宽的功能以及以指定的带宽输入/输出数据的功能。存储器控制器 33 也具有与 MPU 11 的存储器管理单元 22 同样的虚拟存储器管理功能。处理单元 31 将本地存储 32 用作一个主存储器。处理单元 31 不具有对存储器 14 的直接访问权限，而是指示存储器控制器 33 将存储器 14 的内容传输给本地存储 32。处理单元 31 访问本地存储 32 以读/写数据。此外，处理单元 31 指示存储器控制器 33 将本地存储 32 的内容写到存储器 14。

MPU 11 的存储器控制单元 22 以及 VPU 12 的存储器控制器 33 执行如图 3 所示的虚拟存储器管理。由 MPU 11 的处理单元 21 或 VPU 12 的存储器控制器 33 所见的地址是一个 64 位的地址，如图 3 的上部所示。在 64 位地址中，较高的 36 位部分表示一个段号，中间的 16 位部分表示一个页号，而较低的 12 位部分表示一个页偏置。存储器管

理单元 22 和存储器控制器 33 均包括一个段表 50 和一个页表 60。段表 50 和页表 60 将 64 位地址转换为通过连接设备 13 实际寻址的实际地址空间。

例如，如图 4 所示，以下数据映射在由 MPU 11 和每个 VPU 12 所见的实际地址（RA）中。

1. 存储器 14（主存储设备）
2. MPU 11 的控制寄存器
3. 诸个 VPU 12 的控制寄存器
4. 诸个 VPU 12 的本地存储
5. I/O 设备的控制寄存器（包括 I/O 控制器 15 的控制存储器）

MPU 11 和 VPU 12 可访问实际地址空间中的任何地址以读/写数据项 1 至 5。能够从 MPU 11 和 VPU 12 甚至 I/O 控制器 15 访问实际地址空间，进而访问任一 VPU 12 的本地存储 32 是尤其重要的。此外，段表 50 或页表 60 可防止每个 VPU 12 的本地存储 32 的内容被任意读或写。

图 5 显示了由图 3 所示的虚拟存储管理功能所管理的存储器地址空间。由 MPU 11 或 VPU 12 上执行的程序直接看到的是 EA（有效地址）。一个有效地址由段表 50 映射到 VA（虚拟地址）空间中。一个虚拟地址由页表 60 映射到 RA（实际地址）空间中。RA 空间具有如图 4 所示的结构。

MPU 11 可使用诸如一个控制寄存器的硬件机制来管理 VPU 12。例如，MPU 11 可从/向每个 VPU 12 的寄存器读/写数据，并启动/停止每个 VPU 12 执行程序。正如 VPU 12 之间的通信和同步那样，MPU 11 和每个 VPU 12 之间的通信和同步可通过一个硬件机制（例如一个邮箱和一个事件标记）执行。

根据本实施方式的计算机系统允许软件执行一个电子设备的这样一个操作，该操作使得对实时操作的迫切需求按常规的方式实现。例如，一个 VPU 12 执行对应于组成电子设备的某些硬件元件的一个计算，同时另一个 VPU 12 执行对应于组成电子设备的其他硬件元件

的一个计算。

图 6 简单显示了数字电视广播的一个接收机的硬件结构。在此接收机中，一个 DEMUX（多路输出选择器）电路 101 将一个接收到的广播信号分成对应于音频数据、视频数据和字幕数据的压缩编码的数据。一个 A-DEC(音频解码器)电路 102 对压缩编码的音频数据流进行解码。一个 V-DEC（视频解码器）电路 103 对压缩编码的视频数据流进行解码。解码后的视频数据流被发送到 PROG(顺序转换)电路 105，并被转换为一个顺序视频信号。顺序视频信号被发送到一个 BLEND（图像混合）电路 106。一个 TEXT（字幕数据处理）电路 104 将压缩编码的字幕数据转换为一个字幕视频信号，并将它发送给 BLEND 电路 106。BLEND 电路 106 混合 PROG 电路 105 发送的视频信号和 TEXT 电路 104 发送的字幕视频信号，并将混合后的信号作为一个视频流输出。一系列上述操作按照一个视频帧速率（例如，每秒 30、32 或 60 帧）重复。

为了通过软件执行图 6 所示的硬件的操作，本实施方式提供了如图 7 所示的一个程序模块 100。程序模块 100 是一个应用程序，用于使得计算机系统执行图 6 所示的 DEMUX 电路 101，A-DEC 电路 102，V-DEC 电路 103，TEXT 电路 104，PROG 电路 105 和 BLEND 电路 106 的操作。应用程序是由多线程编程所描述的，其结构是执行一个实时操作的线程群组。实时操作包括多个任务的结合。程序模块 100 包括多个程序（多个例程），每个执行为一个线程。具体地说，程序模块 100 包括一个 DEMUX 程序 111，一个 A-DEC 程序 112，一个 V-DEC 程序 113，一个 TEXT 程序 114，一个 PROG 程序 115 和一个 BLEND 程序 116。这些程序 111 至 116 是描述对应于电路 101 至 106 的操作（DMUX 操作，A-DEC 操作，V-DEC 操作，TEXT 操作，PROG 操作，BLEND 操作）的任务过程的程序。更具体地，当程序模块 100 运行时，对应于程序 111 至 116 中每一个程序的一个线程被生成，并被分配给一个或多个 VPU12，并在其上被执行。一个对应于分配给 VPU 12 的线程的程序被加载到 VPU 12 的本地存储 32，并且线程在

本地存储 32 上执行所述程序。程序模块 100 是通过以一个称为结构描述 117 的数据来封装程序 111 至 116 得到的，其中程序 111 至 116 对应于配置一个数据电视广播的接收机的硬件模块。

结构描述 117 是指示程序模块 100 中的程序（线程）是如何被结合并执行的信息。结构描述 117 包括指示链接的程序 111 至 116 之间的输入/输出（链接的）中的关系以及执行程序 111 至 116 中的每一个所需的花费（时间）的信息。图 8 显示了结构描述 117 的一个例子。

结构描述 117 显示了多个模块（程序模块 100 中的程序），每个模块被执行为一个线程，以及其对应的输入、输出、执行花费以及输出所需的缓冲器大小。例如，(3)的 V-DEC 程序接收(1)的 DEMUX 程序的输出，将其作为一个输入，并将其输出发送到(5)的 PROG 程序。V-DEC 程序的输出所需的缓冲器是 1 MB，执行 V-DEC 程序本身的花费是 50。花费可由执行程序或程序的步骤个数所必需的时间单位（时间周期）来描述。它也可以以由一个具有某些虚拟规格的虚拟处理器执行程序所需的时间单位来描述它。由于不同计算机的 VPU 规格和性能可能不同，因此需要以这样的虚拟单元来描述花费。结构描述 117 中的总线带宽是指示由程序 111 至 116 中的每一个程序通过连接设备 13 传输数据所需的一个数据传输带宽（数据传输速率）的信息。数据传输在 VPU 之间、一个 VPU 和存储器 14 之间或者一个 VPU 和 I/O 设备 16 之间执行。以上 QoS 功能使得能够保证执行对应于程序 111 至 116 中每一个程序的操作所需的一个带宽。如果根据图 8 所示的结构描述 117 执行程序，则程序之间的数据流动如图 9 所示。

结构描述 117 还显示了作为线程参数的耦合属性信息，它表示了对应于程序 111 至 116 的线程之间的一个耦合属性。耦合属性包括两个不同的属性：一个紧密耦合属性和一个松散耦合属性。具有紧密耦合属性的多个线程彼此协作地被执行，并被称为一个紧密耦合线程群组。本实施方式的计算机系统对属于每个紧密耦合线程群组的线程进行时序安排，以便属于相同紧密耦合线程群组的线程能够被不同的 VPU 同时执行。具有松散耦合属性的多个线程被称为一个松散耦合线

程群组。编程者可使用线程参数指定对应于程序 11 至 16 的线程之间的一个耦合属性。将参考图 25 及其以下的图详细说明紧密和松散耦合线程群组。包括耦合属性信息的线程参数可被直接描述为程序 111 至 116 中的代码，而不是结构描述 117。

参见图 10 和 11，以下将说明本实施方式的计算机系统如何执行程序 111 至 116。此处假设计算机系统包括两个 VPU: VPU0 和 VPU1。图 10 显示了当每秒显示 30 帧视频数据时将程序分配给每个 VPU 的时间。一帧的音频和视频数据在对应于一个周期的时间间隔中输出。首先，VPU0 执行 DEMUX 程序，以执行 DEMUX 操作，并将其得到的音频、视频和字幕数据写入缓冲器。此后，VPU1 执行 A-DEC 程序和 TEXT 程序，以便顺序执行 A-DEC 操作和 TEXT 操作，并将其结果写入缓冲器。然后，VPU0 执行 V-DEC 程序，以执行 V-DEC 操作，并将其结果写入缓冲器。VPU0 执行 PROG 程序以执行 PROG 操作，并将其结果写入缓冲器。由于此时 VPU1 已完成了 TEXT 程序，因此 VPU0 执行最后的 BLEND 程序以执行 BLEND 操作，以便生成最终的视频数据。每个周期都重复以上处理。

确定每个 VPU 执行哪个程序，以及程序何时完成以便无延迟地执行一个所需操作的一个操作被称为时序安排（scheduling）。一个实现时序安排的模块被称为一个时序安排器。在本实施方式中，时序安排是根据以上包括在程序模块 100 中的结构描述 117 实现的。在时序安排操作中，执行程序 111 至 116 的每个线程的执行开始时间和执行期均是根据结构描述 117 确定的，从而将每个线程分配给一个或多个 VPU 12。以下操作在程序模块 100 将要被执行时执行。

1. 操作系统从一个外部存储或存储器 14 接收程序模块 100，并从程序模块 100 读取多个程序 111 至 116 以及结构描述 117。

2. 根据结构描述 117，操作系统中的时序安排器确定用于执行程序模块 100 中的程序 111 至 116 的每个线程（DEMUX、V-DEC、A-DEC、TEXT、PROG 和 BLEND）的执行开始时间和执行期，以便将线程（DEMUX、V-DEC、A-DEC、TEXT、PROG 和 BLEND）

分配给一个或多个 VPU。

如上所述，在实时处理系统中，执行程序模块 100 中的链接的程序 111 至 116 的每个线程（DEMUX、V-DEC、A-DEC、TEXT、PROG 和 BLEND）的执行开始时间和执行期是根据结构描述 117 确定的。从而，可以不在一个程序的代码中描述每个操作的时间限制条件，对用于执行一个实时操作的线程有效地进行时序安排。

图 11 显示了当每秒显示 60 帧视频数据时执行的程序。图 11 与图 10 的不同之处如下。在图 11 中，每秒需处理 60 帧数据，而在图 10 中，每秒处理 30 帧数据，从而一帧的数据处理可在一个周期(1/30 秒)中完成。换句话说，一帧数据处理不能在一个周期（1/60 秒）中完成，从而在图 11 中执行一个跨多个（两个）周期的软件流水线操作。例如，在周期 1 中，VPU0 对于输入信号执行 DEMUX 程序和 V-DEC 程序。此后，在周期 2 中，VPU1 执行 A-DEC、TEXT、PROG 和 BLEND 程序，并输出最终的视频数据。在周期 2 中，VPU0 在下一帧中执行 DEMUX 和 V-DEC 程序。VPU0 的 DEMUX 和 V-DEC 程序和 VPU1 的 A-DEC、TEXT、PROG 和 BLEND 程序跨两个周期以流水线模式执行。

为实现上述流水线操作，在执行程序模块 100 时执行以下操作：

1. 操作系统从外部存储或存储器 14 接收程序模块 100，并从程序模块 100 读取结构描述 117。

2. 操作系统中的时序安排器根据结构描述 117 确定多个任务 DEMUX、V-DEC、A-DEC、TEXT、PROG 和 BLEND 被程序模块 100 中的程序 111 至 116 执行的顺序。然后时序安排器将任务分成一个第一任务群组和一个第二任务群组。第二任务群组跟随在第一任务群组之后。例如，任务 DEMUX 和 V-DEC 属于第一任务群组，而任务 A-DEC、TEXT、PROG 和 BLEND 属于第二任务群组。

3. 时序安排器使用至少两个处理器 VPU0 和 VPU1，并且周期性地将至少一个处理器分配给第一和第二群组中的每一个，以便以流水线模式周期性地执行第一任务群组（DEMUX 和 V-DEC）和第二任

务群组（A-DEC、TEXT、PROG 和 BLEND）。如果时序安排器用两个处理器 VPU0 和 VPU1 执行一个流水线操作，则它周期性地将第一任务群组（DEMUX 和 V-DEC）分配给 VPU0，以便在 VPU0 上以 1/60 秒的时间间隔周期性地执行第一任务群组。时序安排器周期性地将第二任务群组（A-DEC、TEXT、PROG 和 BLEND）分配给 VPU1，以便在 VPU1 上以相对于第一任务群组延迟一个周期以 1/60 秒的时间间隔周期性地执行第二群组。

两个处理器 VPU1 和 VPU2 可并行执行第二任务群组。例如，当 VPU1 执行任务 A-DEC 和 TEXT 时，VPU2 执行任务 PROG 和 BLEND。

在图 7 所示的程序模块 100 中，多个任务 DEMUX、V-DEC、A-DEC、TEXT、PROG 和 BLEND 由不同的线程执行。从而以上任务群组可被称为线程群组。

图 7 所示的程序模块 100 可被预先记录在结合了本发明的计算机系统的设备的一个闪速 ROM 和一个硬盘中，或者通过一个网络流通。在此情况下，计算机系统执行的操作的内容根据通过网络下载的程序模块的类型而变化。从而，结合了计算机系统的设备可执行对应于每一个不同的专用硬件的实时操作。如果再现新的内容所需的新的播放器软件、解码器软件和加密软件作为计算机系统可执行的程序模块与内容一起被发送，则结合了计算机系统的任何设备可在可接受的能力限制范围内再现所述内容。

节能控制

根据本发明的实施方式的计算机系统在确保一个诸如上述程序模块 100 的实时操作在一个有限的时间周期内完成的同时，执行节能控制以降低功耗。当在特定时间间隔在多个 VPU 上周期性地执行一个实时操作时，则实现一个时序安排操作，以便在一个特定时间间隔内完成实时操作的多个任务，并且尽可能地在一个周期内统一所需的数据传输带宽。根据每个任务所需的数据传输带宽，确定执行任务的一个或多个 VPU 以及每个任务的执行开始时间，以防止具有较大的数据

带宽的至少两个较高级任务的执行期彼此交叠。

图 12 显示了在 VPU0 和 VPU1 上周期性地执行包括三个任务 A、B 和 C 的一个实时操作的时序安排的例子。假设执行任务 A、B 和 C 中每一个所需的总花费（时间）长于对应于一个周期的时间间隔。由于一个 VPU 不能在对应于一个周期的时间间隔内执行三个任务 A、B 和 C，因此任务 A、B 和 C 被分配给 VPU0 和 VPU1。如果要在 VPU0 上执行的一个任务的执行期和要在 VPU1 上执行的一个任务的执行期彼此交叠，则在交叠期内数据传输量（连接设备所需的带宽）增加，如图 12 所示。在图 12 中，任务 A、B 和 C 所需的总线带宽分别为 100 Gbps、90 Gbps 和 20 Gbps。在任务 A 和 B 的执行期交叠期间，需要 190 Gbps 的总线带宽。连接设备（总线）13 的数据传输速率需要被设置为满足每个周期内的一个需要的总线带宽的峰值。峰值越大，则连接设备（总线）13 的数据传输速率必须越高。从而连接设备（总线）13 增加了功耗。

根据本实施方式的时序安排操作是考虑到由结构描述 117 所提供的任务 A、B 和 C 中每一个任务的总线带宽来实现的，以便最小化峰值。操作系统执行任务 A、B 和 C 的时序安排，以便任务 A、B 和 C 在对应于一个周期的一个时间间隔内执行，并且具有大总线带宽的至少两个高级任务（在此情况下是 A 和 B）的执行期不彼此交叠。图 13 显示了防止任务 A 和 B 的执行期彼此交叠的时序安排的一个例子。可使得所需的数据传输带宽在一个周期内几乎统一，并且将其峰值降低到一个较小的值。因此，可在确保包括任务 A、B 和 C 的实时操作在特定时间间隔内被周期性地执行的情况下，将连接设备（总线）13 的数据传输速率设置为低，从而可降低功耗。

将参照图 14 所示的流程图说明一个用于节能控制操作的程序。

步骤 S1：操作系统从外部存储或存储器 14 接收结构描述 117，以检查实时操作的任务的执行顺序、执行每个任务所需的花费以及每个任务所需的数据传输带宽。

步骤 S2：根据以上执行顺序、花费和数据传输带宽，操作系统

执行一个时序安排操作，确定一个或多个执行任务的 VPU 以及每个任务的执行开始时间，以满足三个条件：（1）满足任务的执行顺序限制；（2）在对应于一个周期的时间间隔内执行所有任务；以及（3）防止其数据传输带宽等于或大于其他任务的数据传输带宽的至少两个高级任务的执行期交叠。

步骤 S3：操作系统根据步骤 S2 中的时序安排的结果以及每个任务所需的数据传输带宽，计算（确定）要通过确定的一个或多个 VPU 在时间间隔内执行的数据传输的数据传输带宽的一个峰值。

步骤 S4：操作系统根据连接设备（总线）13 的数据传输能力和计算出的峰值计算所述计算出的峰值与连接设备（总线）13 的最大数据传输带宽（最大总线带宽）的一个比率。

步骤 S5：操作系统根据步骤 S4 中计算出的比率将连接设备（总线）13 的数据传输速率设置为一个低于所述最大数据传输带宽的值。连接设备（总线）13 的数据传输速率可通过将设备 13 的最大数据传输带宽乘以计算出的比率来获得。操作系统向节能控制器 17 发送一个命令，指定设备 13 的操作频率或设备 13 的总线带宽。节能控制器 17 包括一个电路，用于控制设备 13 的操作频率或其总线带宽。节能控制器 17 将操作频率或总线带宽设置为操作系统的命令指定的值。

以下是对由图 7 所示的程序模块所执行的一个数字电视广播接收操作的节能控制的说明。

图 15 显示了当一个 VPU 执行一个数字电视广播接收操作时所需的一个总线带宽。数字电视广播接收操作包括多个 w 任务（D: DEMUX、V: V-DEC、A: A-DEC、T: TEXT、P: PROG、B: BLEND）。这些任务中总线带宽最大的一个（BLEND）被单独执行。因此，每个周期中所需的总线带宽的峰值与任务（BLEND）所需的总线带宽一致。

图 16 显示了当两个频道的数字电视广播接收操作被同时执行时所需的一个总线带宽。VPU0 执行一个频道的一个数字电视广播接收操作，而 VPU1 执行另一频道的一个数字电视广播接收操作。由于总线带宽最大的任务（BLEND）彼此交叠，因此每个周期中所需的总线

带宽的峰值大大增加。

图 17 显示了通过根据本发明的实施方式的时序安排方法对执行两个频道的数字电视广播接收操作进行时序安排的一个例子。在此例子中，用一个周期中未执行任务的空闲时间来移动要被 VPU1 执行的任务（BLEND）的执行期，从而防止总线带宽最大的任务（BLEND）彼此交叠。从而每个周期中所需的总线带宽的峰值可降低到图 16 中的值的一半。

如上所述，考虑到每个任务所需的总线带宽的本实施方式的时序安排操作不仅可应用于对包含在一个实时操作中的任务进行时序安排，也可应用于对一个或多个各自均需要在一个特定时间间隔内执行的实时操作进行时序安排。每个实时操作包含一个或多个任务，并且实时操作之间的执行顺序没有限制。如果两个实时操作均只包含一个任务，则可以只根据执行任务所需的花费和任务的总线带宽来执行时序安排，以防止任务的执行期彼此交叠。

操作系统

当只有一个 OS（操作系统）201 被加载到本实施方式的计算机系统时，它管理所有实际资源（MPU 11、VPU 12、存储器 14、I/O 控制器 15、I/O 设备 16 等），如图 18 所示。

另一方面，一个虚拟机系统可同时执行多个 OS。在此情况下，如图 19 所示，一个虚拟机 OS 301 被加载到计算机系统中，以管理所有的实际资源（MPU 11、VPU 12、存储器 14、I/O 控制器 15、I/O 设备 16 等）。虚拟机 OS 301 也被称为一个主 OS。也被称为子 OS 的一个或多个 OS 302 和 303 被加载到虚拟机 OS 301 上。参见图 20，子 OS 302 和 303 各自在一台包括由虚拟机 OS 301 提供的虚拟机资源的计算机上运行，并向由子 OS 302 和 303 管理的应用程序提供多种服务。在图 20 的例子中，子 OS 302 看起来像是运行在一台包括一个 MPU 11、两个 VPU 12 和一个存储器 14 的计算机上，而子 OS 303 看起来像是运行在一台包括一个 MPU 11、四个 VPU 12 和一个存储器 14 的计算机上。虚拟机 OS 301 管理实际资源中的哪一个 VPU 12 实

际上对应于子 OS 302 所见到的一个 VPU 12 和子 OS 303 所见到的一个 VPU 12。子 OS 302 和 303 不必知道此对应关系。

虚拟机 OS 301 对子 OS 302 和 303 进行时序安排，以便在时分的基础上将计算机系统中的所有资源分配给子 OS 302 和 303。假设子 OS 302 执行一个实时操作。为了以严格的速度每秒三十次执行操作，子 OS 302 将其参数设置到虚拟机 OS 301。虚拟机 OS 301 对子 OS 302 进行时序安排，以便可靠地每 1/30 秒将必要的操作时间分配给子 OS 302 一次。操作时间分配给一个不要求实时操作的子 OS 的优先级低于分配给一个要求实时操作的子 OS 的优先级。图 21 显示子 OS 302 和 303 交替运行，由水平轴表示时间。当子 OS 302（OS1）在运行时，MPU 11 和所有的 VPU 12 被用作子 OS 302（OS1）的资源。而当子 OS 303（OS2）在运行时，MPU 11 和所有的 VPU 12 被用作子 OS 303（OS2）的资源。

图 22 显示了一个不同的操作模式。有这样的情况：希望根据目标应用程序连续使用一个 VPU 12。此情况对应于，例如，一个要求始终连续监控数据和事件的应用程序。虚拟机 OS 301 的时序安排器管理一个特定子 OS 的时序安排，以便该子 OS 占用一个特定的 VPU 12。在图 22 中，一个 VPU 3 被指定为一个子 OS 302（OS1）专用的资源。即使虚拟机 OS 301 将子 OS 302（OS1）和子 OS 303（OS2）彼此交换，VPU 3 还是继续在子 OS 302（OS1）的控制下运行。

为了在本实施方式中使用多个 VPU 12 执行程序，使用了一个软件模块，它被称为一个 VPU 运行环境。该软件模块包括一个时序安排器，用于对要分配给 VPU 12 的线程进行时序安排。当本实施方式的计算机系统上只实现一个 OS 201 时，在 OS 201 上实现了一个 VPU 运行环境 401，如图 23 所示。VPU 运行环境 401 可实现在 OS 201 的内核中或实现在一个用户程序中。它也可以一分为二，用于内核和用户程序，以便彼此协作地运行。当一个或多个子 OS 运行在虚拟机 OS 301 上时，提供以下模式以实现 VPU 运行环境 401：

1. 在虚拟机 OS 301 中实现 VPU 运行环境 401 的模式（图 24）。

2. 将 VPU 运行环境 401 作为一个由虚拟机 OS 301 管理的 OS 的实现模式(图 25)。在图 25 中,运行在虚拟机 OS 301 上的子 OS 304 是 VPU 运行环境 401。

3. 在由虚拟机 OS 301 管理的每个子 OS 中实现一个专用 VPU 运行环境的模式(图 26)。在图 26 中,VPU 运行环境 401 和 402 被实现在各自的子 OS 302 和 303 中。VPU 运行环境 401 和 402 联合运行,如果必要的话,使用由虚拟机 OS 301 提供的子 OS 之间的一个通信功能。

4. 在由虚拟机 OS 301 管理的子 OS 之一中实现 VPU 运行环境 401 的模式(图 27)。一个没有 VPU 运行环境的子 OS 303 通过一个由虚拟机 OS 301 提供的子 OS 之间的通信功能来利用一个子 OS 302 的 VPU 运行环境。

以上模式具有以下优点:

模式 1 的优点

由虚拟机 OS 301 管理的一个子 OS 的时序安排以及诸个 VPU 的时序安排可合并成一个。从而,可以有效地并且精细地完成时序安排,并且有效地使用资源;以及

由于 VPU 运行环境可在多个子 OS 之间共享,因此当引入一个新的子 OS 时不必创建一个新的 VPU 运行环境。

模式 2 的优点

由于诸 VPU 的一个时序安排器可在虚拟机 OS 上的多个子 OS 之间共享,因此可有效地并且精细地执行时序安排,并能有效地使用资源。

由于 VPU 运行环境可在多个子 OS 之间共享,因此当引入一个新的子 OS 时不必创建一个新的 VPU 运行环境;以及

由于可在不依赖于虚拟机 OS 或一个特定的子 OS 的情况下创建 VPU 运行环境,因此它易被标准化,以及彼此取代。如果创建一个适合一个特定的嵌入式设备的一个 VPU 运行环境以便利用设备的特征执行时序安排,则可以有效地完成时序安排。

模式 3 的优点

由于可在每个子 OS 中最优地实现 VPU 运行环境，因此可以有效地并且精细地执行时序安排，并且能够有效地使用资源。

模式 4 的优点

由于不需要在所有子 OS 上实现 VPU 运行环境，因此易于添加一个新的子 OS。

由上可见，模式 1 至 4 均可被用于实现 VPU 运行环境。需要时可使用任何其他的模式。

服务提供者

在根据本实施方式的计算机系统中，VPU 运行环境 401 提供多种服务（一个使用网络的通信功能、一个输入/输出文件的功能、调用一个库函数（例如一个编解码器）、与用户接口、一个使用 I/O 设备进行的输入/输出操作、读取日期和时间等），以及管理与 VPU 12 相关的多种资源（每个 VPU 的操作时间、一个存储器、一个连接设备的带宽等）并对其进行时序安排的功能。这些服务是从运行在 VPU 12 上的应用程序调用的。如果一个简单的服务被调用，则它被 VPU 12 上的服务程序处理。一个不能仅由 VPU 12 处理的服务，例如通信处理和文件处理，则由 MPU 11 上的服务程序处理。提供这些服务的程序被称为一个服务提供者（SP）。

图 28 显示了 VPU 运行环境的一个例子。VPU 运行环境的主要部分位于 MPU 11 上，并且对应于一个 MPU 侧的 VPU 运行环境 501。一个 VPU 侧的 VPU 运行环境 502 位于每个 VPU 12 上，只具有执行一个可由 VPU 12 处理的服务的最小功能。MPU 侧的 VPU 运行环境 501 的功能大体上可分为一个 VPU 控制器 511 和一个服务代理 512。VPU 控制器 511 主要提供对与 VPU 12 相关的各种资源（每个 VPU 的操作时间、一个存储器、一个虚拟空间、一个连接设备的带宽等）的一个管理机制、一个同步机制、一个安全管理机制和一个时序安排机制。根据时序安排的结果将程序分派给 VPU 12 的是 VPU 控制器 511。在接收到一个由每个 VPU 12 上的应用程序调用的服务请求时，

服务代理 512 调用一个适当的服务程序（服务提供者），并且提供服务。

在接收到每个 VPU 12 上的一个应用程序调用的服务请求时，VPU 侧的 VPU 运行环境 502 只处理可在 VPU 12 中提供的服务，并且请求服务代理 512 处理其不可能提供的服务。

图 29 显示了由 VPU 侧的 VPU 运行环境 502 处理一个服务请求的过程。在接收到来自一个应用程序的一个服务调用时（步骤 S101），VPU 侧的 VPU 运行环境 502 确定是否能在其中处理该服务（步骤 S102）。如果该服务能被处理，则 VPU 运行环境 502 执行该服务，并把其结果返回给调用的部分（步骤 S103 和 S107）。如果不能，则 VPU 运行环境 502 确定是否有一个可执行该服务的服务程序作为每个 VPU 12 上的一个可执行程序（步骤 S104）被注册。如果该服务程序已被注册，则 VPU 运行环境 502 执行服务程序并将其结果返回给调用的部分（步骤 S105 和 S107）。如果没有，则 VPU 运行环境 502 请求服务代理 512 执行该服务程序，并将服务的一个结果从服务代理 512 返回给调用部分（步骤 S106 和 S107）。

图 30 显示了由 MPU 侧的 VPU 运行环境 501 的服务代理处理由 VPU 侧的 VPU 运行环境 502 请求的一个服务的过程。在接收到来自 VPU 侧的 VPU 运行环境 502 的一个服务调用时（步骤 S111），服务代理 512 确定 VPU 运行环境 501 是否能够处理该服务（步骤 S112）。如果能够处理该服务，则服务代理 512 执行该服务并将其结果返回给调用部分的 VPU 侧的 VPU 运行环境 502（步骤 S113 和 S114）。如果不能，则服务代理 512 确定是否有一个能够执行该服务的服务程序作为 MPU 11 上的一个可执行程序被注册（步骤 S115）。如果该服务程序已被注册，则服务代理 512 执行该服务程序，并将其结果返回给调用的部分的 VPU 侧的 VPU 运行环境 502（步骤 S116 和 S114）。如果没有，则服务代理 512 将一个错误返回给调用部分的 VPU 侧的 VPU 运行环境 502（步骤 S117）。

结果回复给从每个 VPU 12 执行的程序发出的某些服务请求，并

且没有结果回复给其他服务请求。回复的目标通常是一个发出一个服务请求的线程；但是，另一个线程、一个线程群组或一个进程可以被指定为回复的目标。因此希望目标被包括在一条消息中以请求一个服务。可用一个广泛使用的对象请求代理来实现服务代理 512。

实时操作

根据本实施方式的计算机系统用作一个实时处理系统。由实时处理系统执行的操作大体上分为以下三种类型：

1. 硬实时操作

2. 软实时操作

3. 尽力操作（非实时操作）

硬和软实时操作是所谓的实时操作。本实施方式的实时处理系统具有与多个现有的 OS 相同的线程和进程概念。首先将说明实时处理系统中的线程和进程。

线程具有以下三类

1. 硬实时类

时间要求非常重要。此线程类是用于那种当要求没被满足时，会引起非常危险的情况的重要应用。

2. 软实时类

此线程类是用于那种即使时间要求没被满足质量也只是降低的应用。

3. 尽力类

此线程类是用于一个不包括时间要求的应用。

在本实施方式中，线程是实时操作的一个执行单位。线程具有其相关程序，这些程序将被线程执行。每个线程保存其固有的信息，称为一个线程环境。线程环境包括，例如，一个栈的信息以及存储在处理器寄存器中的值。

在实时处理系统中，有两个不同的线程：MPU 和 VPU 线程。这两个线程是按执行线程的处理器（MPU 11 和 VPU12）分类的，它们的模型是彼此相同的。VPU 线程的线程环境包括 VPU 12 的本地存储

32 的内容，以及存储器控制器 33 的 DMA 控制器的条件。

一组线程被称为一个线程群组。线程群组具有有效并且容易执行诸如赋予该群组的线程相同属性的一个操作的优点。硬或软实时类中的线程群组大体上分为一个紧密耦合线程群组和一个松散耦合线程群组。紧密耦合线程群组和松散耦合线程群组是通过添加到线程群组的属性信息（耦合属性信息）来彼此区别的。线程群组的耦合属性可由应用程序或上述结构描述中的代码明确指定。

紧密耦合线程群组是一个由彼此协作运行的线程组成的线程群组。换句话说，属于紧密耦合线程群组的线程彼此紧密协作。紧密协作是指诸如线程间的频繁通信和同步的交互，或者减少等待时间的交互。属于同一紧密耦合线程群组的线程总是同时被执行。换句话说，松散耦合线程群组是一个排除了属于该群组的线程之间的紧密协作的线程群组。属于松散耦合线程群组的线程执行用于通过存储器 14 上的缓冲器传输数据的通信。

紧密耦合线程群组

如图 31 所示，不同的 VPU 被分配给紧密耦合线程群组的线程，这些线程被同时执行。这些线程被称为紧密耦合线程。紧密耦合线程的执行期在其各自的 VPU 中预定，并且紧密耦合线程被同时执行。在图 31 中，一个紧密耦合线程群组包括两个紧密耦合线程 A 和 B，并且线程 A 和 B 分别被 VPU0 和 VPU1 同时执行。本实施方式的实时处理系统确保线程 A 和 B 被不同的 VPU 同时执行。一个线程可通过执行另一线程的 VPU 的本地存储或控制寄存器与所述另一线程通信。

图 32 描述了线程 A 和 B 之间的通信，该通信是通过分别执行线程 A 和线程 B 的 VPU0 和 VPU1 的各自本地存储执行的。在执行线程 A 的 VPU0 中，对应于执行线程 B 的 VPU1 的本地存储 32 的一个 RA 空间被映射在线程 A 的一个 EA 空间中的一部分。为进行此映射，VPU0 的存储器控制器 33 中提供的一个地址翻译单元 331 使用一个段表和页表执行地址翻译。地址翻译单元 331 将线程 A 的 EA 空间的一部分转换（翻译）为对应于 VPU1 的本地存储 32 的 RA 空间，从而将

对应于 VPU 1 的本地存储 32 的 RA 空间映射在线程 A 的 EA 空间的一部分中。在执行线程 B 的 VPU1 中，对应于执行线程 A 的 VPU0 的本地存储 32 的一个 RA 空间被映射在线程 B 的一个 EA 空间中。为进行此映射，VPU1 的存储器控制器 33 中提供的一个地址翻译单元 331 使用一个段表和页表执行地址翻译。地址翻译单元 331 将线程 B 的 EA 空间的一部分转换（翻译）为对应于 VPU0 的本地存储 32 的 RA 空间，从而将对应于 VPU 0 的本地存储 32 的 RA 空间映射在线程 B 的 EA 空间的一部分中。

图 33 显示了将执行线程 B 的 VPU1 的本地存储(LS1)32 映射到由 VPU0 执行的线程 A 的 EA 空间中，以及将执行线程 A 的 VPU0 的本地存储(LS0)32 映射到由 VPU1 执行的线程 B 的 EA 空间中。例如，当要传输到线程B的数据在本地存储LS0中准备好时，线程A在VPU0的本地存储 LS0 或执行线程 B 的 VPU1 的本地存储 LS1 中设置一个标记，表示此准备好状态。线程 B 响应该标记的设置，从本地存储 LS0 读取数据。

根据上述本实施方式，紧密耦合线程可由耦合属性信息指定，并且紧密耦合线程 A 和 B 肯定分别由不同的 VPU 同时执行。从而，线程A和B之间的通信和同步的交互可在无延迟的情况下更紧密地被执行。

松散耦合线程群组

属于松散耦合线程群组的每个线程的执行期取决于线程之间的输入/输出的关系。即使线程不受执行顺序的限制，也不保证它们被同时执行。属于松散耦合线程群组的线程被称为松散耦合线程。图 34 显示了一个松散耦合线程群组，它包括两个松散耦合线程：线程 C 和 D，这两个线程由各自的 VPU 0 和 VPU1 执行。如图 34 所示，线程 C 和 D 的执行期不同。线程 C 和 D 之间的通信是由主存储器 14 中准备的缓冲器实现的，如图 35 所示。由 VPU0 执行的线程 C 将准备在本地存储 LS0 中的数据通过 DAM 传输写入在主存储器 14 上准备的缓冲器。VPU1 执行的线程 D 从主存储器 14 上的缓冲器读取数据，并在线

程 D 开始运行时通过 DMA 传输将其写入本地存储 LS1.

进程和线程

如图 36 所示，一个进程包括一个地址空间和一个或多个线程。线程可被包括在进程中，不论其数目和类型为何。例如，进程中可只包括 VPU 线程，也可包括 VPU 和 MPU 线程的混合。正如一个线程保存一个线程环境作为其固有信息那样，一个进程保存一个进程环境作为其固有信息。进程环境包括进程固有的一个地址空间以及包括在进程中的所有线程的线程环境。地址空间可在进程的所有线程之间共享。一个进程可包括多个线程群组，但是一个线程群组不能属于多个进程。从而，属于一个进程的一个线程群组是该进程所固有的。

在本实施方式的实时处理系统中，有两种模型作为创建一个新线程的方法：一个线程优先模型和一个地址空间优先模型。地址空间优先模型与现有 OS 中采用的相同，从而可被应用到 MPU 和 VPU 线程。另一方面，线程优先模型只能被应用到 VPU 线程，并且是本实施方式的实时处理系统所特有的。在线程优先模型中，现有的线程（它是用于创建一个新线程的线程，即新线程的一个父线程）首先指定将被一个新线程执行的一个程序，并且使得新线程开始执行该程序。然后该程序被存储在 VPU 的本地存储中，并且从一个指定的地址开始运行。由于此时没有地址空间与新线程相关联，因此新线程可获得对 VPU 的本地存储而不是存储器 14 的访问权限。此后，当需要时，新线程本身调用 VPU 运行环境的一个服务，并创建一个地址空间。地址空间与新线程相关联，并且新的线程可获得对存储器 14 的访问权限。在地址空间优先模型中，现有的线程创建一个新的地址空间或指定现有的地址空间，并且在地址空间中安排要被新线程执行的程序。然后，新线程开始运行程序。线程优先模型的优点是一个线程只能被本地存储执行，因而减少了生成、分派和退出线程所需的开销。

对线程进行时序安排

现将参考图 37 中的流程图说明由 VPU 运行环境 401 执行的一个时序安排操作。VPU 运行环境 401 中的时序安排器根据添加到每个将

被进行时序安排的线程群组的耦合属性信息，检查线程之间的一个耦合属性（步骤 S121）。时序安排器确定每个线程群组是一个紧密耦合线程群组还是一个松散耦合线程群组（步骤 S122）。参考程序代码中的线程描述或以上结构描述 117 中的线程参数来检查耦合属性。如果紧密和松散耦合线程群组被各自指定，则要对其进行时序安排的线程被分成紧密和松散耦合线程群组。

属于紧密耦合线程群组的线程的时序安排如下执行。为了使各自的 VPU 同时执行从要被时序安排的线程中选出的一个紧密耦合线程群组的线程，VPU 运行环境 401 中的时序安排器为每个 VPU 预定一个运行期，VPU 的数目等于线程数目，并且同时将线程分派给 VPU（步骤 S123）。时序安排器使用一个执行线程的 VPU 中的地址翻译单元 331 将一个 RA 空间映射到一个线程的一个 EA 空间的一部分(步骤 S124)，该 RA 空间对应于一个执行与前一线程交互作用的一个伙伴线程的 VPU 的本地存储。对于从要被时序安排的线程中选出的属于松散耦合线程群组的线程，时序安排器根据线程之间的输入/输出的关系按顺序将线程分派给一个或多个 VPU（步骤 S125）。

如果根据耦合属性信息选择了一个紧密耦合线程群组，即彼此协作运行的线程的一个集合，则可确保属于紧密耦合线程群组的线程被不同的处理器同时执行。因此，线程之间的通信可通过彼此获得对执行其伙伴线程的处理器的寄存器的直接访问权限的一个易实现的机制实现。从而可容易并迅速地执行通信。

线程的状态转换

一个线程一般从其被创建直至其被删除期间进行状态转换。如图 38 所示，一个线程进行以下七个状态转换。

1. 不存在状态：此状态是逻辑的，并且不存在于一个有效的线程中。

2. 休眠状态：一个线程被创建并且尚未开始运行。

3. 准备好状态：线程准备好开始运行。

4. 等待状态：线程等待条件符合开始（重新开始）运行。

5. 运行状态：线程在 VPU 或 MPU 上实际运行。

6. 暂停状态：线程被 VPU 运行环境和其他线程强制暂停。

7. 等待-暂停状态：等待和暂停状态彼此交叠。

以上七个状态之间的转换条件和转换涉及的线程环境如下。

[从不存在状态到休眠状态的转换]

此转换是通过创建一个线程完成的。

一个线程环境被创建，但其内容处于初始状态。

[从休眠状态到不存在状态的转换]

此转换是通过删除一个线程完成的。

如果线程被设置为存储其线程环境，则存储的线程环境通过该转换被丢弃。

[从休眠状态到等待状态的转换]

此转换在线程请求运行环境对线程进行时序安排时完成。

[从等待状态到准备好状态的转换]

此转换在线程等待的一个事件（例如，同步、通信、计时器中断）被生成时完成。

[从准备好状态到运行状态的转换]

此转换在线程被运行环境分派给 MPU 或 VPU 时完成。

线程环境被加载。当线程环境被保存时，它被恢复。

[从运行状态到准备好状态的转换]

此转换在线程的运行被先占时完成。

[从运行状态到等待状态的转换]

此转换在线程暂停其自身的运行以等待一个使用同步机制、通信机制等的事件时完成。

每个类中的线程可被设置为存储其线程环境。当一个线程被设置为存储其线程环境时，当线程从运行状态转换到等待状态时，线程环境被运行环境保存。被保存的线程环境被一直保存，除非线程转换到休眠状态，并且在线程转换到运行状态时被恢复。

[从运行状态到暂停状态的转换]

此转换在响应来自运行环境或其他线程的指令强制暂停线程的运行时完成。

每个类中的线程可被设置为存储其线程环境。当一个线程被设置为存储其线程环境时，当线程从运行状态转换到暂停状态时，线程环境被运行环境保存。被保存的线程环境被一直保存，除非线程转换到休眠状态，并且在线程转换到运行状态时被恢复。

[从运行状态到休眠状态的转换]

此转换在线程本身退出其自己的运行时完成。

当线程被设置为存储其线程环境时，线程环境的内容通过该转换被丢弃。

[从等待状态到等待-暂停状态的转换]

此转换在线程在等待状态中等待生成一个事件时被来自外部的指令强制停止时完成。

[从等待-暂停状态到等待状态的转换]

此转换在线程处于等待-暂停状态时被来自外部的指令重新开始运行时完成。

[从等待-暂停状态到暂停状态的转换]

此转换在线程在等待状态中等待的事件被生成时完成。

[从暂停状态到准备好状态的转换]

此转换在线程被来自外部的指令重新开始运行时完成。

[从准备好状态到暂停状态的转换]

此转换在线程被外部环境停止运行时完成。

线程的执行期

向一个 VPU 分配的一个线程的运行状态的期间被称为一个执行期。一般地，从一个线程的创建到删除的期间包括线程的多个执行期。图 39 显示了从创建到删除的不同的线程状态的一个例子。此例子包括线程存在期间的两个执行期。可使用不同的方法保存和恢复线程环境。多数正常的线程运行，使得在一个执行期结束时保存一个环境，并且在下一个执行期开始时恢复该环境。在一个特定的周期性操作中，线

程运行使得在每个周期中，在一个执行期开始时创建一个新的环境，在该执行期期间使用该环境，并且在该执行期结束时丢弃该环境。

属于紧密耦合线程群组的线程的执行期

图 40 显示了属于同一个紧密耦合线程群组的线程的执行期。所有属于一个特定的紧密耦合线程群组的线程由VPU运行环境401进行时序安排，以便它们能在一个执行期中同时运行。此紧密耦合线程群组主要用于硬实时线程。因此，为实现该操作，当为硬实时类预定了一个执行期时，VPU 运行环境 401 指定同时使用的处理器及其数目。此外，VPU 运行环境 401 使得与处理器对应的诸个线程的环境各自同时运行。

属于一个特定的执行期中的紧密耦合线程群组的线程，可通过取消其紧密耦合关系在其它执行期中彼此分离地运行。每个线程必须认识到它是作为一个紧密耦合线程运行的，还是与另一线程分开运行的，并且执行一个与其伙伴线程通信和同步的操作。每个线程具有一个属性，指示是可被先占的或不可被先占的。可被先占的属性允许线程在其执行期被先占，即允许线程停止运行。不可先占的属性确保一个线程在其执行期不能被先占。不同线程类的不可被先占属性的意义不同。在硬实时类中，当一个线程开始运行时，直到其执行期结束，除线程本身外没有什么能停止运行。在软实时类中，可被先占性是基本的，从而不支持不可被先占的属性。在尽力类型中，一个线程可被保护，以便不被其它尽力类先占，但它可被一个诸如硬实时类和软实时类的更高级别的类先占。

线程的执行模型

线程的执行模型大体上可分为两个模型：一个如图 41 所示的周期性执行模型和一个如图 42 所示的非周期性执行模型。在周期性执行模型中，一个线程被周期性地执行。在非周期性运行模型中，一个线程是根据一个事件执行的。周期性执行模型可用一个软件中断或一个诸如同步基元（synchronization primitive）的事件对象实现。在硬实时类中，周期性执行模型是用一个软件中断实现的。换句话说，VPU

运行环境 401 在应当开始一个周期性操作时跳转到由一个指定的方法确定的一个线程的入口点，或者调用由一个指定程序预先注册的一个回调函数。在软实时类中，周期性模型是用一个事件对象实现的。换句话说，由于 VPU 运行环境 401 在每个周期中通知一个预先注册的事件对象的生成，因此一个软实时线程等待每个周期中的一个事件对象，并且在生成事件时执行一个指定的操作，从而实现一个周期性的执行模型。在尽力类中，周期性的执行模型可用一个软件中断或一个事件对象中的任何一个实现。实际执行不总在每个周期开始时开始，而是可以在限制范围内延迟。

通过使用一个事件模型，非周期性的执行模型可被实现为周期性的执行模型。在软实时类和尽力类中，非周期执行模型与周期性执行模型的区别仅在于一个事件被通知的时间，并且这些模型在实现方法上是相同的。在硬实时类中，对于确保时间要求所必需的最小到达间隔时间和最终期限强烈限制着系统的操作；因此，非周期性的执行是受限的。

环境转换

在根据本实施方式的实时处理系统中，可选择在一个 VPU 线程的执行期结束时转换一个环境的方法。由于转换环境的花费是非常高的，因此选择一种方法提高了转换效率。被选中的方法用在一个线程的预定的执行期结束时。当一个线程在执行期期间或在先占时被转换时，在任何情况下当前线程的所有环境均需要被保存，并且在线程下次重新开始运行时被恢复。例如，以下有转换一个 VPU 环境的方法。

1. 丢弃环境

不保存环境。

2. 完全保存环境

保存一个 VPU 的所有环境，包括 VPU 的寄存器和本地存储的状态，以及存储器控制器中的 DMA 控制器的状态。

3. 适度保存环境

环境转换被延迟，直到一个 VPU 的存储器控制器的 DMA 控制

器的所有操作均完成。此后，VPU 中的寄存器和本地存储的内容被保存。在此方法中，VPU 的所有环境和全部存储均被保存。

可实现一个时序安排器对 MPU 和 VPU 线程进行时序安排，也可完成不同的时序安排器以对其各自的 MPU 和 VPU 线程进行时序安排。由于 MPU 和 VPU 线程转换环境花费不同，因此实现不同的时序安排变得更加有效。

硬实时类中的时序安排

硬实时类中的线程的时序安排是用一个扩展的任务图的一个预定图执行的。图 43 显示了任务图的一个例子。任务图表示了任务之间的一个关系。在图 43 中，任务之间的箭头表示任务的依存关系（任务之间的输入/输出中的关系）。根据图 44 的例子，任务 1 和 2 可随意开始运行，一个任务 3 只能在任务 1 和 2 均停止运行后才能开始运行，而任务 4 和 5 只能在任务 3 停止运行后才能开始运行。任务图没有环境的概念。例如，当应该使用相同的环境处理任务 1 和 4 时，不能在任务图中描述这一点。因此以下扩展任务表的预定表被用于本实施方式的实时处理系统中。

首先，考虑任务图不是任务之间而是执行期之间的一个关系。通过将一个环境关联到每个执行期，对应于环境的一个线程在执行期中运行。如果相同的环境被关联到多个执行期，则其对应的线程在每个执行期中运行。在图 44 所示的例子中，线程 1 的环境被关联到执行期 1 和 2，并且线程 1 在执行期 1 和 2 中的每一个中运行。一个指示由运行环境确保的硬实时限制的属性被添加到执行期之间的每个箭头上。通过这样创建的一个预定表，可在不修改实时应用的模型的情况下，描述操作模型和诸如一个实时应用的时间要求的限制。图 45 显示了根据图 44 所示的图创建的预定图的一个例子。图 45 中的环境 1、2 和 3 分别对应于图 44 中的线程 1、2 和 3。

软实时类的时序安排

软实时类中的线程的时序安排是用一个固定优先级的时序安排方法来执行的，以便能够断言线程的运行模式。为所述时序安排方法

准备了两个不同的时序安排算法：一个是固定优先级的 FIFO 时序安排，而另一个固定优先级的循环时序安排。为了优先执行一个较高优先级的线程，即使当一个较低优先级的线程在运行时，较低优先级的线程也被先占，并且较高优先级的线程立即开始运行。为了避免在一个关键部分中出现优先级颠倒问题，需要执行一个同步机制，例如一个优先级继承协议和一个优先级上限协议。

尽力类中的时序安排

尽力类中的线程的时序安排是用动态优先级时序安排或者类似的时序安排执行的。

分级时序安排器

VPU 运行环境 401 中的时序安排功能可由如图 46 所示的一个分级时序安排器实现。换句话说，线程级的时序安排具有两个等级：线程类之间的时序安排和线程类之内的时序安排。从而，VPU 运行环境 401 中的时序安排器具有一个线程类之内的时序安排部件 601 和一个线程类之间的时序安排部件 602。线程类之间的时序安排部件 602 对分布在不同线程类的多个线程进行时序安排。线程类之内的时序安排部件 601 对属于每个线程类的多个线程进行时序安排。部件 601 包括一个硬实时（硬 RT）类时序安排部件 611、一个软实时（软 RT）类时序安排部件 612 和一个尽力类时序安排部件 613。

线程类之间的时序安排和线程类之内的时序安排具有一个分级结构。首先，线程类之间的时序安排运行，以确定执行哪个线程类，然后确定执行线程类中的哪个线程。线程类之间的时序安排采用可先占的固定优先级时序安排。硬实时类具有最高优先级，接下来按顺序是软实时类和尽力类。当一个较高优先级类中的一个线程准备好运行时，一个最低优先级的线程被先占。线程类之间的同步是由一个由 VPU 运行环境 401 提供的同步基元实现的。尤其地，只有基元能被用于一个硬实时线程中以防止在硬实时线程中发生阻塞。当一个尽力线程阻塞一个软实时线程时，它被当作一个软实时线程处理，以防止发生线程类之间的优先级颠倒。此外，例如，优先级继承协议的使用防

止另一个软实时线程阻塞该尽力线程。

线程参数

在根据本实施方式的实时处理系统中，是用多种参数对线程进行时序安排的。各类中的线程公共的参数如下：

线程类（硬实时、软实时、尽力）；

使用资源（MPU 或 VPU 的数目、带宽、物理存储器大小、I/O 设备）；

优先级；以及

可被先占的或不可被先占的。

以下是硬实时类中的线程的参数：

执行期；

最终期限；

周期或最小到达间隔时间；以及

VPU 环境转换方法。

图 47 显示了硬实时类的基本参数的例子。在图 47 的上部分所示的指定一个执行期的例 1 中，在指定的执行期中同时预定了一个 MPU 和两个 VPU，并且每个 VPU 的环境被完全保存。在此情况中，线程在三个处理器上同时运行，并且在执行期之后，所述 VPU 线程的环境以及 MPU 线程的环境被完全保存。在图 47 的右上角，例 2 显示了一种方法，它指定一个最终期限以确保由 VPU 的数目及其执行期所表示的一个操作在最终期限之前执行。最终期限是在做出一个预定请求时由在请求时间处开始的相对时间指定的。在图 47 的下部分，例 3 显示了一种指定一个周期性执行的方法。在此例中，指定两个 VPU 12 的执行期被周期性地重复，并且 VPU 线程的环境在每个周期的执行期之后被丢弃，使得所有操作由新的环境执行。此外，最终期限是由在周期开始处开始的相对时间指定的。

例如，有以下限制作为其他参数用于硬实时类中：

时间限制（绝对时间限制和相对时间限制）；

优先顺序限制；以及

互斥限制。

时间限制提供了延迟执行时间的一个单位。绝对时间限制是指定相对于静止时间（例如一个周期的开始时间）的延迟时间的一种情况，如图 48 所示。相对时间限制是指定相对于动态时间和一个事件（例如一个特定的开始时间和结束时间）的允许的延迟时间的一种情况，如图 49 所示。由于优先顺序限制可通过用相对时间限制指定相对于一个特定执行期的结束时间的延迟时间为 0 或更长来实现，因此它可被视为一个特殊的相对时间限制。

互斥限制是确保执行期彼此不交叠的一种情况，如图 50 所示。互斥限制使得能够减少执行期的不可预计性，所述不可预计性是由一个锁定引起的。换句话说，防止了某些资源公共的所有线程同时运行，以便避免关于资源的锁定。

线程的同步机制

在根据本实施方式的实时处理系统中，以下同步基元被用作线程的同步机制：

信标（semaphore）；

消息队列；

消息缓冲器；

事件标记；

屏障；以及

互斥体。

可使用其他同步基元。本实施方式的实时处理系统提供以下三种方法来实现以上同步机制：

同步机制在一个 VPU 的存储器（主存储）14 或本地存储 32 上用一个诸如一个 TEST&SET 的指令实现；

同步机制由诸如一个邮箱和一个信号寄存器的硬件机制实现；以及

同步机制用一个作为 VPU 运行环境的一个服务提供的机制实现。

由于同步机制具有优点和缺点，因此需要根据线程的属性选择性地使用它们，如图 51 所示。换句话说，一个用由 MPU 和 VPU 共享和访问的存储器（主存储 MS）14 实现的同步机制可用于所有类中的线程。相反，一个实现在一个 VPU 12 的本地存储 LS 上的同步机制只能被属于紧密耦合线程群组的线程使用。这是因为只有属于紧密耦合线程群组的线程确保其同步的伙伴线程同时运行。例如，如果一个属于紧密耦合线程群组的线程被用于实现在执行其伙伴线程的一个 VPU 的本地存储上的一个同步机制，则当使用该同步机制时确保了所述伙伴线程的执行。从而，执行所述伙伴线程的 VPU 的本地存储始终存储所述同步机制的信息。

一个使用一个单元而不是存储器（主存储 MS）和本地存储 LS 的同步机制可由一个硬件机制或者 VPU 运行环境 401 的一个服务实现。由于属于紧密耦合线程或硬实时类中的线程要求高速同步机制，因此需要在线程中使用由硬件机制实现的同步机制。相反，由运行环境提供的同步机制需要用在属于松散耦合线程群组的线程中或属于软实时类和尽力类的线程中。

同步机制的自动选择

在根据本实施方式的实时处理系统中，可根据线程的属性和状态自动选择或转换以上同步机制。此操作由如图 52 所示的一个程序执行。当同步的线程属于紧密耦合线程群组中（步骤 S201 中的“是”），采用由存储器 14、每个 VPU12 的本地存储 32 或硬件机制实现的一个高速同步机制（步骤 S202、S203、S204、S205）。当线程改变状态以取消其紧密耦合关系时（步骤 S201 中的“否”），则高速同步机制被转换为实现为存储器 14 上的一个同步机制或 VPU 运行环境 401 的一个服务的一个同步机制（步骤 S206、S207、S208）。

以上转换可以一个库的形式或者作为每个 VPU12 中的 VPU 运行环境 502 中的一个服务提供给运行在 VPU12 上的程序。多个同步机制可如下转换。可预先确保并且选择性地使用同步机制，或者在执行转换时确保新的同步机制。

对于一个使用 VPU 12 的本地存储的同步机制，线程需要像属于紧密耦合线程群组的线程那样被 VPU 同时执行。此限制可以如下放松。当一个线程未在运行时，本地存储的内容在线程最后运行时被存储在存储器 14 中，并且控制映射使得存储的内容由指示本地存储的页表或段表的条目指示。根据此方法，当伙伴线程未在运行时，线程可继续运行，好像有一个与伙伴线程关联的本地存储一样。当通过向线程分配一个 VPU 12 使之开始运行时，存储器 14 中的内容被恢复到 VPU 12 的本地存储，以改变一个对应的页表或段表的映射。使用 VPU 12 的本地存储的一个备份复本，所述使用 VPU 12 的本地存储的同步机制甚至可用于不属于紧密耦合线程群组的线程。

预定图

图 53 显示了对应于图 9 所示的数据流动的一个预定图。在图 53 中，六个框表示执行期。每个框上左上的数字表示要预定的一个执行期的标识。每个框中的符号表示与执行期相关联的一个线程环境的标识符。每个框的右下的数字表示执行期的长度（花费）。连接框的箭头都表示优先顺序限制。换句话说，从一个框指到另一个框的一个箭头表示后一框的执行期中的操作在前一框的执行期的操作完成后开始。从而可表示一个执行期的链。每个箭头旁的数字表示用于由箭头连接的执行期之间的数据传输的一个缓冲器的标识，而每个数字旁的值表示缓冲器的大小。以下是执行根据图 53 所示的预定图的操作的程序 1 至 7。

1. 创建一个执行 DEMUX 程序 111 的线程环境，并将其标识符称为 DEMUX。

2. 创建一个执行 A-DEC 程序 112 的线程环境，并将其标识符称为 A-DEC。

3. 创建一个执行 V-DEC 程序 113 的线程环境，并将其标识符称为 V-DEC。

4. 创建一个执行 TEXT 程序 114 的线程环境，并将其标识符称为 TEXT。

5. 创建一个执行 PROG 程序 115 的线程环境，并将其标识符称为 PROG。

6. 创建一个执行 BLEND 程序 116 的线程环境，并将其标识符称为 BLEND

7. 创建一个具有如图 54 所示的一个数据结构的预定请求，并将其发送给 VPU 运行环境 401 以做出一个预定。

根据以上程序 1 至 6 中的每一个，如果一个程序被指定为作为一个线程运行，则 VPU 运行环境 401 向该程序分配必要的资源以创建一个线程环境。该线程环境的句柄被返回，并且从而被称为一个标识符。

图 54 显示一个预定请求，它包括写为“缓冲器”的缓冲器数据和写为“任务”的执行期。缓冲器数据用于在存储器 14 上宣称一个缓冲器，用于执行期之间的数据传输。在缓冲器数据中，“标识”表示缓冲器号，“大小”表示缓冲器大小，“源任务”表示写数据的执行期号，而“目标任务”表示读数据的执行期号。在执行期数据中，“标识”代表执行期号，“类”表示线程类（VPU 表示 VPU 线程以及 HRT 表示硬实时类。此外，有 MPU 表示 MPU 线程，SRT 表示软实时类，BST 表示尽力类等），“线程环境”表示对应于执行期的线程环境，“花费”表示执行期的长度或花费，”限制”表示基于执行期的多种限制，“输入缓冲器”表示在执行期中读取的缓冲器的标识符的一个列表，而“输出缓冲器”表示执行期中写入的缓冲器的标识符的一个列表。“限制”还可包括表示优先顺序限制的“优先顺序”，表示绝对时间限制的“绝对时间”，表示相对时间限制的“相对时间”以及表示互斥限制的“排斥”。“限制”具有用于限制的伙伴线程的执行期号的一个列表。

被图 54 所示的预定请求预定的缓冲器区域被分配给主存储器 14，并被 VPU 运行环境 401 从中释放出来。缓冲器区域的分配在一个向缓冲器区域写数据的线程开始运行时执行。缓冲器区域的释放在一个从缓冲器区域读数据的线程退出时执行。使用线程开始运行时预定的一个地址、一个线程或一个寄存器来通知线程被分配的缓冲器的地址。在本实施方式的实时处理系统中，当提供如图 7 所示的程序模块

100 时，如图 8 所示的结构描述 117 被读出程序模块 100，并且根据结构描述 117，一个线程环境被以上程序创建，并且如图 54 所示的预定请求被创建并发布，从而提供了个执行程序模块 100 的功能。如图 7 所示的程序模块 100 所描述的专用硬件的操作被多个处理器由处理软件执行。具有如图 7 所示的结构的一个程序模块被创建给每个要实现的硬件，然后被一个具有符合本实施方式的实时处理系统的功能的设备执行，使得设备能够正如需要的硬件那样被操作。另一个例子是，一个创建图 54 所示的预定请求的操作在应用程序中被描述，并且应用程序可自己创建一个预定请求，并且将其传输给 VPU 运行环境 401。

在提供图 54 所示的预定请求的同时，VPU 运行环境 401 确定哪个 VPU 12 在一个周期中的哪个时间执行各个任务。这就是时序安排。实际上，可同时提供多个预定请求；因此确定操作时间以防止其彼此冲突（防止给定的限制未得到满足）。假设当有如图 55 所示的两个 VPU12 时，只做出了图 54 所示的预定请求，则执行时序安排，以使得 VPU 0 顺序执行不能并行完成的 DEMUX、V-DEC、PROG 和 BLEND 操作，并且在 DEMUX 操作后，VPU1 执行可并行完成的 A-DEC 和 TEXT 操作。

软件流水线

如果在一个周期内没有足够的时间顺序执行 DEMUX、V-DEC、PROG 和 BLEND 操作，则跨多个周期实现软件流水线处理。例如，如图 56 所示，VPU 0 在第一周期中执行 DEMUX 和 V-DEC 操作，而 VPU 1 在第二周期中执行 A-DEC、TEXT、PROG 和 BLEND 操作。在第二周期中，VPU 0 执行下一帧中的 DEMUX 和 V-DEC 操作，这与 A-DEC、TEXT、PROG 和 BLEND 操作是并行的。换句话说，如图 57 所示，流水线处理是这样执行的：在 VPU 0 执行 DEMUX 和 V-DEC 操作的同时，VPU 1 在接收到来自前一周期中的 DEMUX 和 V-DEC 操作的输出时执行 A-DEC、TEXT、PROG 和 BLEND 操作。采用流水线操作使得一个实时操作能够在更短的时间里在每个周期中完成。

图 58 是用于时序安排以实现一个软件流水线操作的程序的一个流程图。

VPU 运行环境 401 确定是否所有需要顺序执行的线程 DEMUX、V-DEC、PROG 和 BLEND 能在一个周期内完成（步骤 S401）。

一个周期的长度作为程序模块 100 的一个执行条件被预设给 VPU 运行环境 401。长度可在结构描述 117 中明确描述。在步骤 S401 中，根据线程 DEMUX、V-DEC、PROG 和 BLEND 的花费预计这些线程的总执行期。预计的总执行期与一个周期的长度相比较。

如果 VPU 运行环境 401 确定线程 DEMUX、V-DEC、PROG 和 BLEND 不能在一个周期内执行（步骤 S401 中的“否”），则它根据线程 DEMUX、V-DEC、A-DEC、TEXT、PROG 和 BLEND 的执行顺序将用于执行程序模块 100 的所有线程 DEMUX、V-DEC、A-DEC、TEXT、PROG 和 BLEND 分成两个可顺序执行的群组（以下称为第一和第二线程群组）（步骤 S402）。第一线程群组是在第二线程群组之前执行的一个或多个线程的一个集合，而第二线程群组是在第一线程群组之后执行的一个或多个线程的一个集合。在本实施方式中，线程 DEMUX 和 V-DEC 属于第一线程群组，而线程 A-DEC、TEXT、PROG 和 BLEND 属于第二线程群组，以满足线程之间的优先顺序限制，并且使得每个群组的总执行期不长于对应于一个周期的时间间隔。

VPU 运行环境 401 执行时序安排操作，以便周期性地将属于第一线程群组的每个线程（DEMUX 和 V-DEC）的执行期分配给 VPU0，使得 VPU0 以 1/60 秒的时间间隔周期性地执行第一线程群组（步骤 S403）。在步骤 S403 中，每个线程 DEMUX 和 V-DEC 的周期性的执行是为 VPU0 预定的。然后，VPU 运行环境 401 执行时序安排操作，以便周期性地将属于第二线程群组的每个线程（A-DEC、TEXT、PROG 和 BLEND）分配给 VPU1，使得 VPU1 以相对于第一线程群组一个周期的延迟以 1/60 秒的时间间隔周期性地执行第二线程群组（步骤 S404）。在步骤 S404 中，每个线程 A-DEC、TEXT、PROG 和 BLEND 的周期性的执行是为 VPU1 预定的。

两个处理器 VPU0 和 VPU1 以流水线模式处理第一线程群组（DEMUX 和 V-DEC）和第二线程群组（A-DEC、TEXT、PROG 和 BLEND）。因此，在第二线程群组相对于第一线程群组延迟一个周期的同时，第一线程群组和第二线程群组被并行执行，从而每个 1/60 秒的周期输出帧数据的处理结果。

在以上例子中，VPU0 始终执行第一线程群组（DEMUX 和 V-DEC），而 VPU1 始终执行第二线程群组（A-DEC、TEXT、PROG 和 BLEND）。但是，如图 59 所示，可执行时序安排，以便周期性地替换第一线程群组所分配的处理器和第二线程群组所分配的处理器。在时序安排操作中，在每个周期中确定第一和第二线程群组中的每一个的执行时间以及用于执行第一和第二线程群组的不同处理器，以便在第二线程群组相对于第一线程群组延迟一个周期的同时，在处理器上并行执行第一和第二线程群组。

使用流水线操作的节能控制

通过上述流水线操作可在满足任务的执行顺序限制的范围内减轻对每个任务的执行时间的限制。即使每个周期没有空闲时间，也可通过流水线操作执行时序安排以防止总线带宽大的任务的执行期彼此交叠。

图 60 显示了同时执行两个频道的数字电视广播接收操作时所需的一个总线带宽。如果每个周期没有空闲时间，则由 VPU0 执行的 BLEND 的执行期和由 VPU1 执行的 BLEND 的执行期不能简单地彼此移动。

图 61 显示了以上 BLEND 的执行期被流水线操作移动的一个例子。要由 VPU1 执行的实时操作（D2: DEMUX、V2: V-DEC、A2: A-DEC、T2: TEXT、P2: PROG、B2: BLEND）被分类成一个第一线程群组（V2、A2、T2、D2）和一个第二线程群组（P2、B2）。如图 61 所示，第二线程群组（P2、B2）以相对于第一线程群组（V2、A2、T2、D2）一个周期的延时被执行，而在周期 2 中，第二线程群组（P2、B2）在第一线程群组（V2、A2、T2、D2）之前被执行。由

于实时操作（D2: DEMUX、V2: V-DEC、A2: A-DEC、T2: TEXT、P2: PROG、B2: BLEND）是由 VPU1 跨两个周期执行的，因此可防止要由 VPU0 和 VPU1 执行的 BLEND 的执行期彼此交叠。因此，如图 62 所示，每个周期中所需的总线带宽的峰值可减少为图 60 中的值的一半。

具有分层结构的预定图

虽然图 53 所示的预定图不具有分层结构，但也可以使用一个具有分层结构的预定图，如图 63 所示。在图 59 中，执行期 A 先于执行期 B，而执行期 B 先于执行期 C。在执行期 B 中，执行期 D 先于执行期 E 和 F。解析此分层结构可得，执行期 A 先于执行期 D，而执行期 E 和 F 先于执行期 C。

基于结构描述的时序安排算法

以下说明了一个用于根据结构在程序模块中的结构描述预定每个线程的一个执行期的程序。

图 8 显示了结合在图 7 所示的程序模块 100 中的结构描述 117 的一个例子。利用结构描述 117，VPU 运行环境 401 执行以下步骤。

1. 写在结构描述 117 的模块域中的程序被加载以生成执行程序的线程。在本实施方式中，为结构描述 117 的每个条目生成一个线程。如果结构描述 117 包括具有相同模块名称的条目，则生成执行相同模块的多个线程，以便对应于其各自的条目。在图 8 的例子中，所有的线程被生成为属于一个进程；但是，线程可属于不同的进程，或者线程群组可属于不同的进程。

2. 根据结构描述 117 的信息创建具有如图 54 所示的一个数据结构的一个预定请求。

3. 预定请求被发送给 VPU 运行环境，以便对线程进行时序安排，并且开始运行线程。

以上创建预定请求的步骤 2 按以下方式执行。

首先，“缓冲器”记录被创建以便一对一地对应于结构描述 117 的输出域，并且被添加到预定请求。例如，在图 8 的例子中，DEMUX

模块的第二输出数据通过一个 1MB 的缓冲器被提供给 V-DEC，以使得如图 54 所示的一个标识符为 2 的“缓冲器”记录被创建。在此“缓冲器”记录中，缓冲器大小在“大小”域中被描述为 1MB，对标识为 1 并且对应于向缓冲器写数据的一个 DEMUX 模块的“任务”记录的参照在“源任务”域中描述，对标识为 3 并且对应于从缓冲器读数据的一个 V-DEC 模块的“任务”记录的参照在“目标任务”域中描述。

然后，“任务”记录被创建以一对一地对应于结构描述 117 的模块域，并且被添加到预定请求。例如，在图 8 的例子中，如图 54 所示的一个标识为 3 的“任务”记录被创建为对应于 V-DEC 模块。此“任务”记录具有以下信息：

Class 域：指示哪个属性被用于执行“任务”记录中指定的一个线程的标记。

在此域中，“VPU”表示一个在 VPU 上运行的线程，而“HRT”表示硬实时类中的一个线程。这些信息项目是根据图 8 所示的结构描述 117 的线程参数中描述的信息设置的。

线程环境域：指定一个线程的线程环境的标记，该线程的运行将在“任务”记录中被预定。更具体地，结构描述 117 的模块域中指定的一个程序模块被加载，一个执行该程序模块的线程被 VPU 运行环境 401 生成，线程的线程环境的一个标识符（一个指针之类的）被记录在“线程环境”域中。

限制域：记录“任务”记录的限制的标记。当限制为优先顺序限制时，“任务”记录之后的另一个“任务”记录所需的标识号被指定在“优先顺序”域之后。例如，一个标识为 3 的“任务”记录先于一个对应于 PROG 模块的标识为 5 的“任务”记录。

输入缓冲器域：指定被由“任务”记录指定的线程从中读取数据的一个缓冲器的“缓冲器”记录所需的标识号的标记。

输出缓冲器域：指定被由“任务”记录指定的线程向其写入数据的一个缓冲器的“缓冲器”记录所需的标识号的标记。

带宽域：指定“任务”记录指定的线程所需的一个总线带宽的标

记。

如果结构描述按照上述方式被提供，则其对应的预定请求被创建。

当预定请求被发送给 VPU 运行环境 401 中的时序安排器时，时序安排器创建一个执行预定请求所必需的时序安排。此时序安排表示哪个 VPU 在哪个时间被分配给哪个线程，并且在一个周期中 VPU 被分配多长时间，如图 55 所示。实际上，所述时序安排可被图 64 所示的一个预定列表所表示。

图 64 所示的预定列表包括与各 VPU 相关的预定条目。每个预定条目包括一个开始时间域，表示每个周期中一个线程何时被 VPU 执行（线程的执行开始时间），一个执行期域，表示 VPU 被分配给线程多长时间（线程的执行期），以及一个运行线程域，表示线程的一个标识符。预定条目按照 VPU 的开始时间的顺序排序，并被链接到预定列表。

从图 54 所示的预定请求创建一个如图 64 所示的预定例的程序可由图 65 所示的流程图实现。

基本上，预定请求中的“任务”记录只需要按照使用“缓冲器”输入/输出的关系排序，而 VPU 的运行时间只需要按照数据流顺序分配给每个“任务”记录。然后需要将 VPU 分配给属于紧密耦合线程群组的任务。当使用两个或多 VPU 时，考虑每个“任务”记录的总线带宽，对“任务”排序以防止总线带宽较大的至少两个高级“任务”的执行期彼此交叠。

程序如图 65 所示。在接收到一个预定请求时，VPU 运行环境 401 通过以下步骤对由预定请求中的“任务”记录指定的所有任务进行时序安排（换句话说，VPU 运行环境 401 创建一个预定列表，用于预定每个任务被分配给的 VPU，以及任务的执行开始时间和执行期）。

步骤 S301：VPU 运行环境 401 从未进行时序安排的任务中选择一个任务，它之前的所有任务（输入任务）已经被进行了时序安排。并且它没有紧密耦合属性。如果一个任务之前没有输入任务，则它被

确定为输入任务已经被进行了时序安排的任务。

如果有一个任务，它的输入任务已经被进行了时序安排，并且它没有紧密耦合属性，则 VPU 运行环境 401 选择它并转至步骤 S302。如果没有，则转至步骤 S304。

步骤 S302：如果有一个 VPU 在符合要求的限制下能分配选定的任务的执行开始时间和执行期，则 VPU 运行环境 401 转至步骤 S303。如果没有，则 VPU 运行环境 401 的时序安排失败，并且做出一个失败通知。

步骤 S303：VPU 运行环境 401 创建选定的任务的预定条目，并且把它们链接到预定列表。任务的执行期是在考虑到其总线带宽的情况下被确定的，如上所述。

步骤 S304：VPU 运行环境 401 从未被进行时序安排的任务中选择其所有输入任务已经被进行了时序安排、并且属于一个紧密耦合群组的任务。如果一个任务之前没有输入任务，则它们被确定为输入任务已经被进行了时序安排的任务。

如果存在其输入任务已经被进行了时序安排，并且属于紧密耦合群组的任务，则 VPU 运行环境 401 选择它们并转至步骤 S305。如果没有，则结束时序安排。

步骤 S305：如果有 VPU 能够同时预定所选中的任务中的所有任务（具有相同的执行开始时间和相同的执行期），则 VPU 运行环境 401 转至步骤 S306。如果没有，则 VPU 运行环境 401 的时序安排失败，并且做出一个失败通知。

步骤 S306：选中的任务集合的所有任务的预定条目被创建，并被链接到预定列表。

已说明了对一个预定请求进行时序安排的步骤。实际上，通常一个系统中会同时出现多个预定请求。在此情况下，可通过上述步骤对预定请求进行时序安排，更有利地是，可通过上述步骤同时完成它们。

现在已经以描述一个数字电视广播接收机的操作的程序模块为例说明了本实施方式。但是，如果准备了描述多种类型硬件的操作的

一个程序模块，则可由软件执行硬件的操作。

图 1 所示的计算机系统中提供的 MPU 11 和 VPU 12 可实现为混合在一个芯片上的并行处理器。在此情况下，同样，由 MPU 11 执行的 VPU 运行环境或者由一个特定的 VPU 执行的 VPU 运行环境或者类似的能够控制 VPU 12 的时序安排和总线 13 的数据传输速率。

如果作为 VPU 运行环境运行的程序或者包括 VPU 运行环境的操作系统的程序被存储在一个计算机可读存储介质中，然后被引入到一个包括多个处理器的计算机中，其中每个处理器包括一个本地存储器，并且在该计算机中执行，则可获得与本发明的前述实施方式相同的优点。

本领域技术熟练者易实现其他优点和修改。因此，在更广泛的方面中的本发明并不限于特定的细节和此处显示和说明的代表性实施方式。因此，可以在不背离附录的权利要求书或其等价物所定义的一般发明概念的精神或范围的情况下做出多种修改。

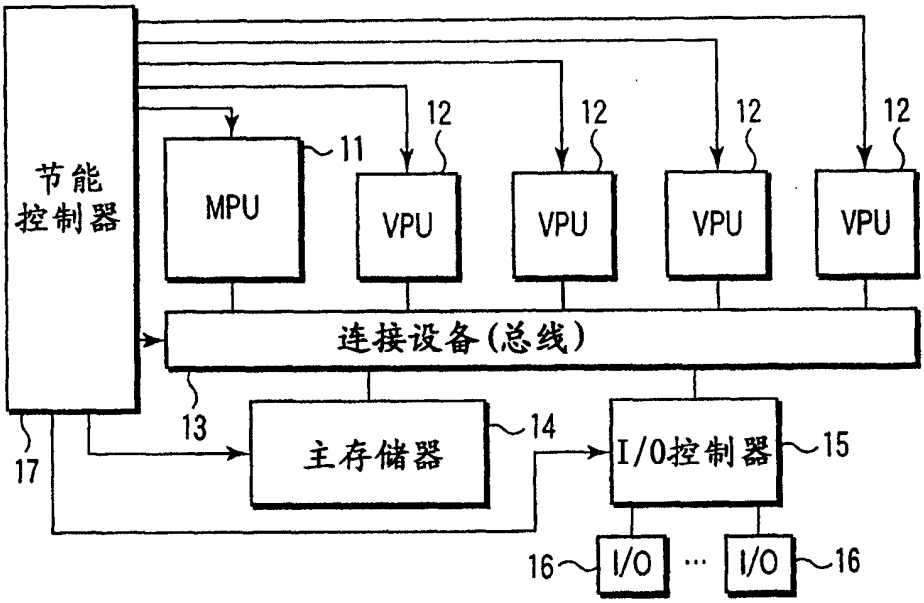

节能
控制器
MPU
11
12
VPU
VPU
VPU
VPU
连接设备(总线)
13
主存储器
14
I/O控制器
15
17
16
I/O
I/O
16

图1

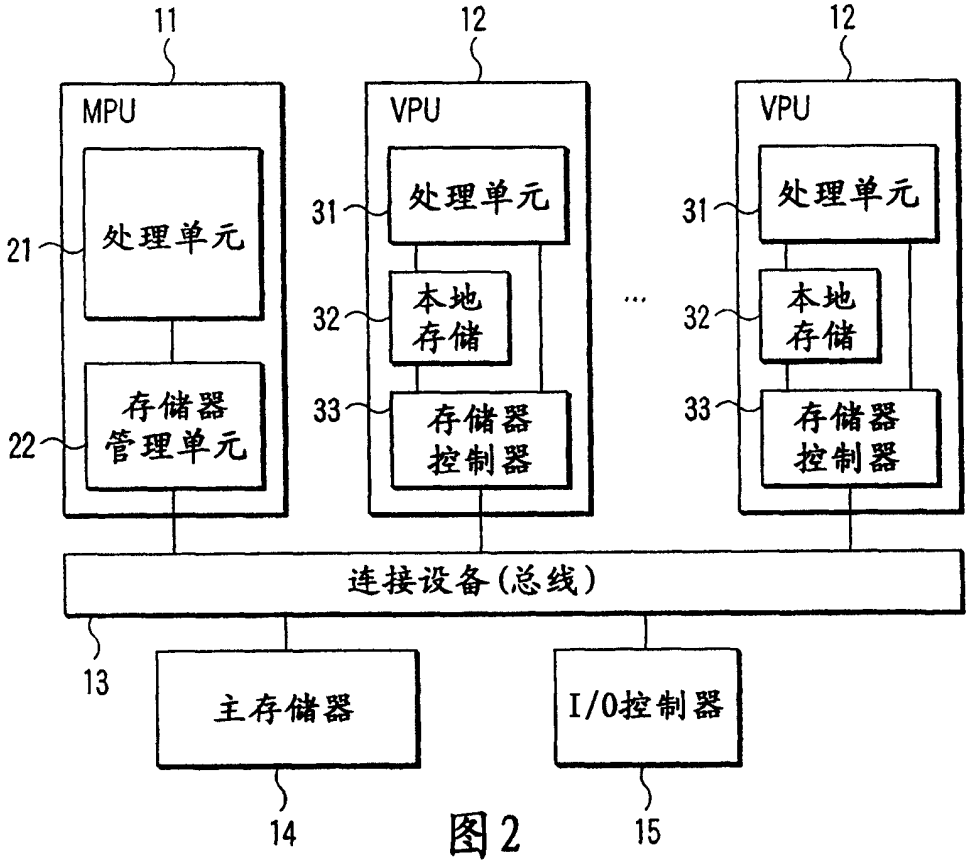

11
12
12
MPU
VPU
VPU
处理单元
处理单元
处理单元
21
31
31
本地
存储
本地
存储
32
32
存储器
管理单元
存储器
控制器
存储器
控制器
22
33
33
连接设备(总线)
13
主存储器
I/O控制器
14
15

图2

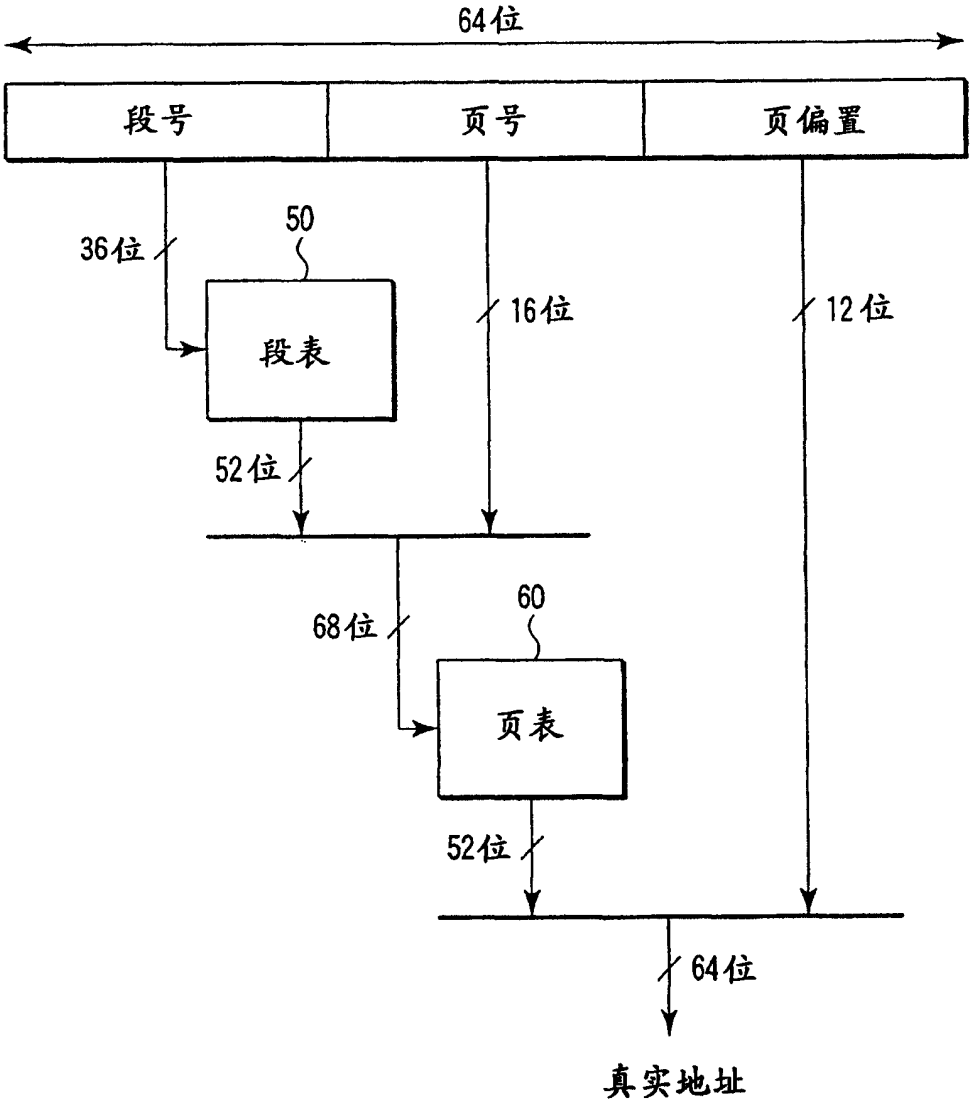


图3

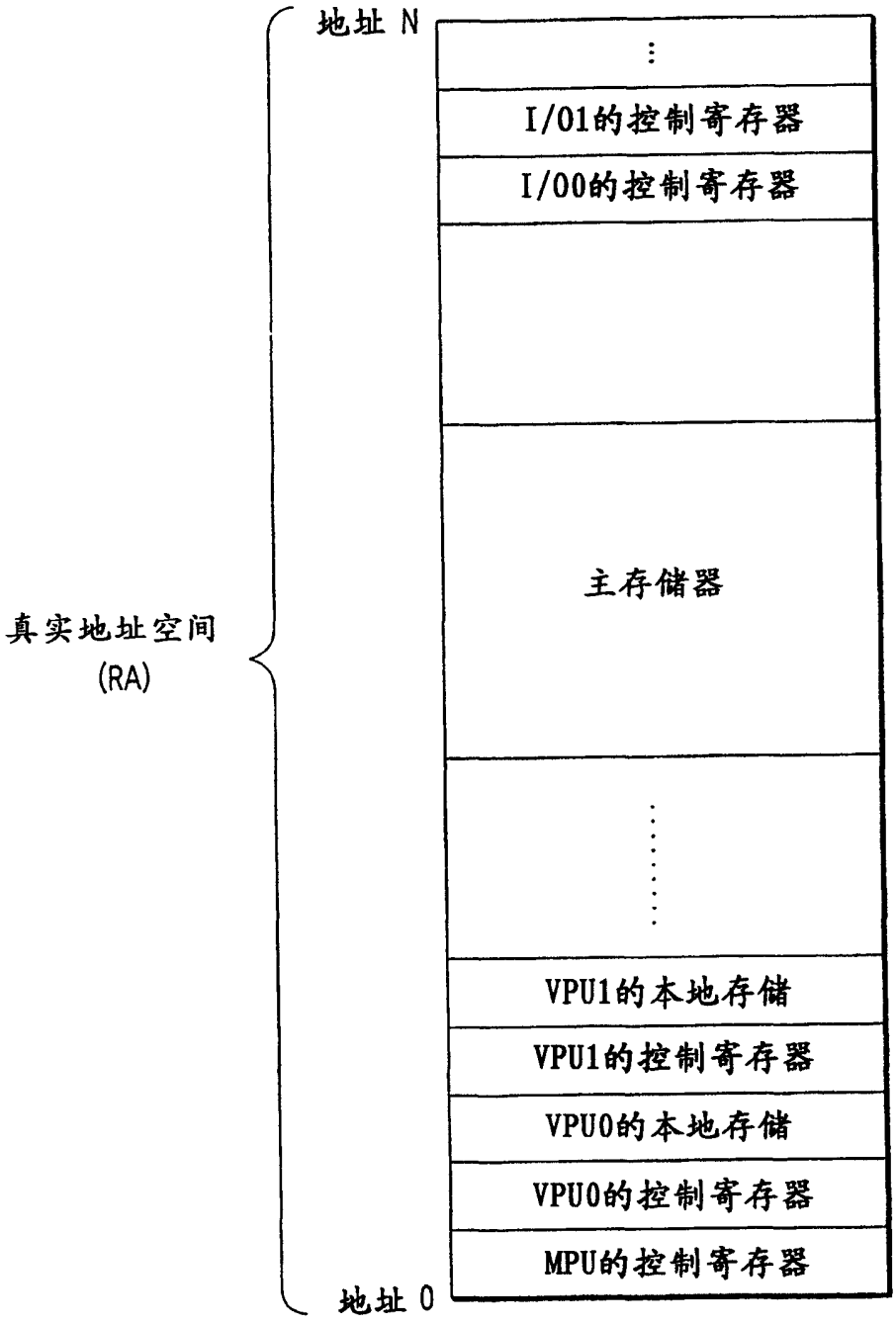

地址 N
I/O1的控制寄存器
I/O0的控制寄存器
主存储器
VPU1的本地存储
VPU1的控制寄存器
VPU0的本地存储
VPU0的控制寄存器
MPU的控制寄存器
地址 0
真实地址空间
(RA)

图4

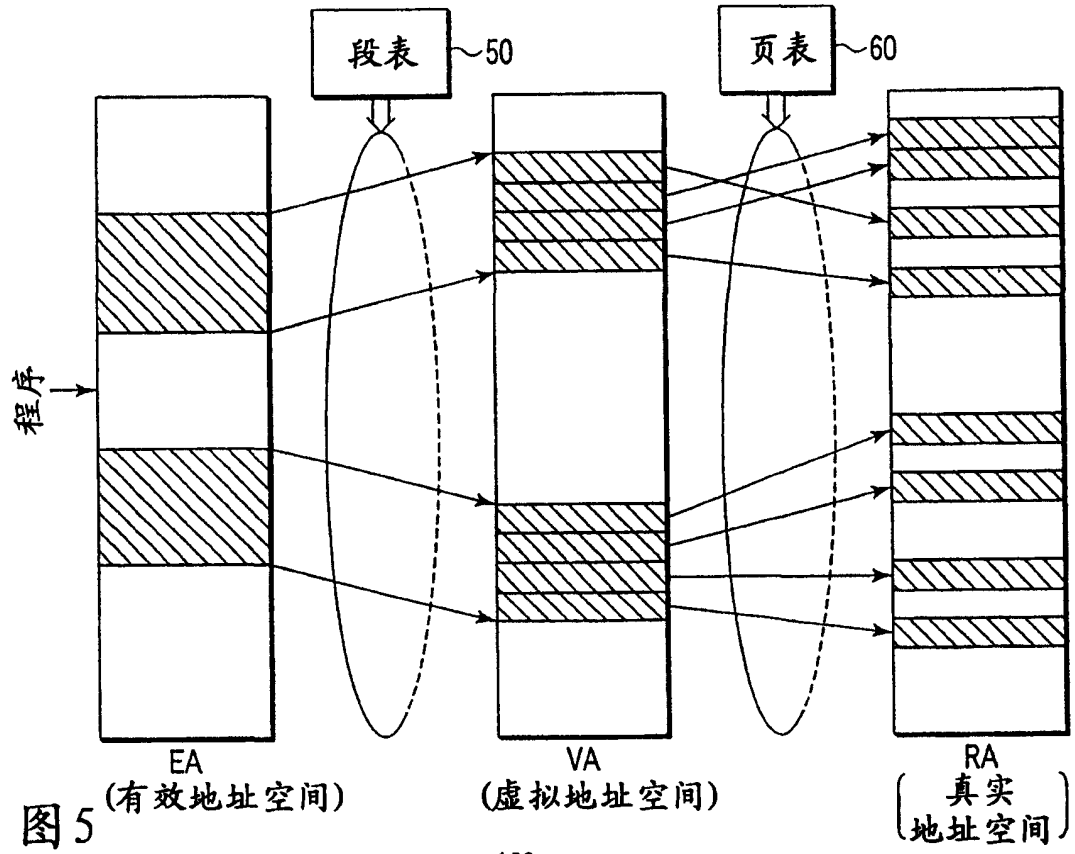

段表
50
页表
60
程序
EA
(有效地址空间)
VA
(虚拟地址空间)
RA
真实
地址空间

图5

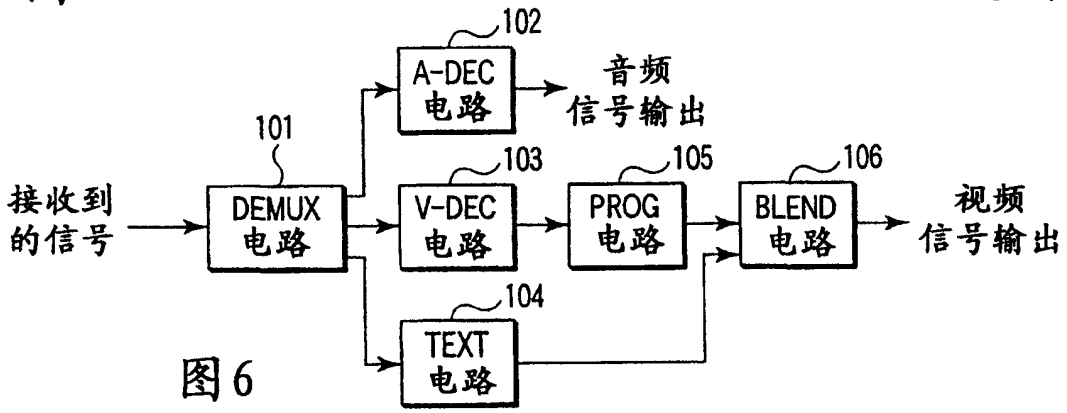

102
A-DEC
电路
音频
信号输出
101
接收到
的信号
DEMUX
电路
103
V-DEC
电路
105
PROG
电路
106
BLEND
电路
视频
信号输出
104
TEXT
电路

图6

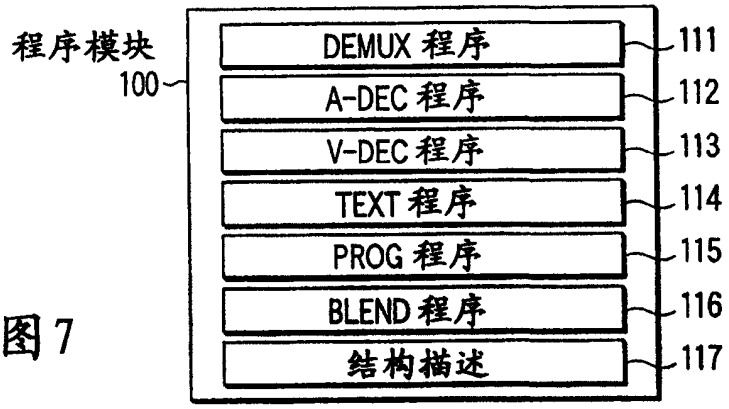

程序模块
100
DEMUX 程序
111
A-DEC 程序
112
V-DEC 程序
113
TEXT 程序
114
PROG 程序
115
BLEND 程序
116
结构描述
117

图7

结构描述 117

编号	程序	输入	输出	花费	缓冲器	总线带宽
(1)	DEMUX	接收到的信号	(2) (3) (4)	5	100KB 1MB 10KB	962Mbps
(2)	A-DEC	(1)	音频输出	10	———	25Mbps
(3)	V-DEC	(1)	(5)	50	1MB	686Mbps
(4)	TEXT	(1)	(6)	5	10KB	2002Mbps
(5)	PROG	(3)	(6)	20	1MB	3200Mbps
(6)	BLEND	(4) (5)	视频输出	10	———	7400Mbps
线程参数 · 紧密耦合线程群组 : · 松散耦合线程群组 :						
其他						

图8

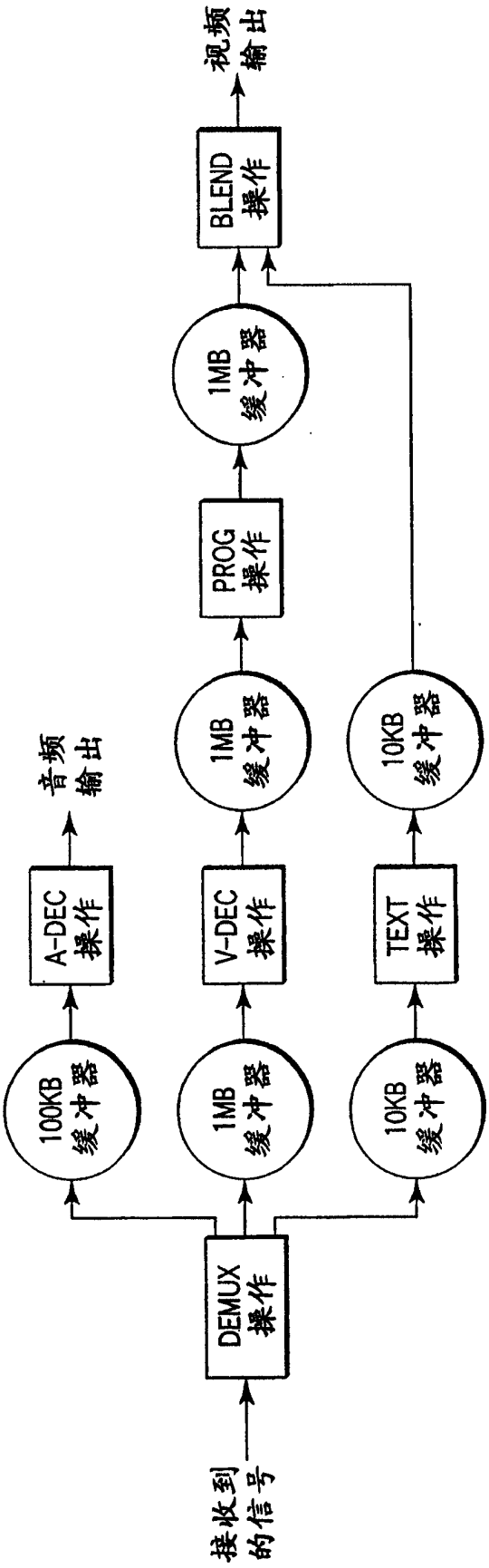

接收到的信号
DEMUX 操作
100KB 缓冲器
1MB 缓冲器
10KB 缓冲器
A-DEC 操作
V-DEC 操作
TEXT 操作
音频输出
1MB 缓冲器
10KB 缓冲器
PROG 操作
1MB 缓冲器
BLEND 操作
视频输出

图 9

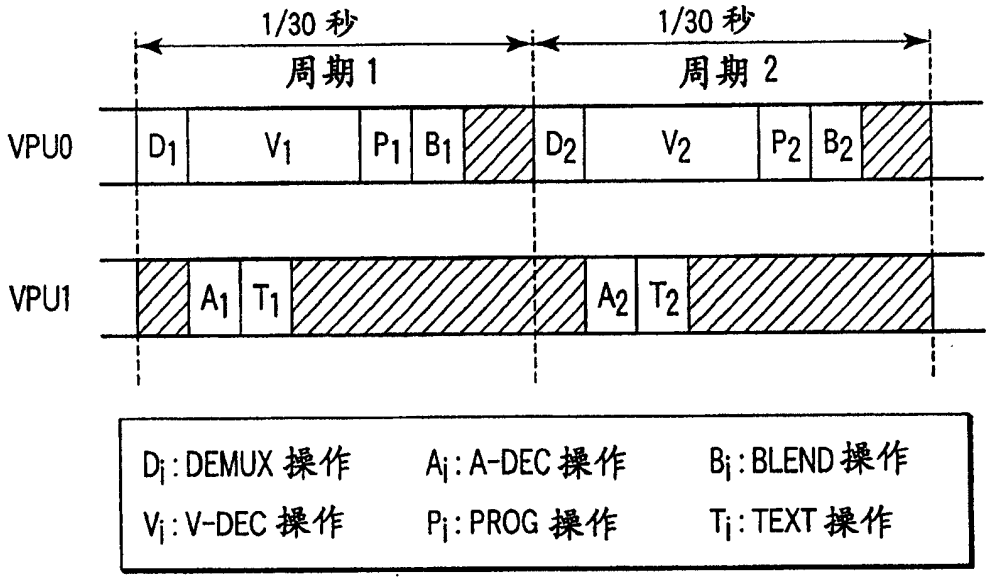

1/30 秒
1/30 秒
周期 1
周期 2
VPU0
D1
V1
P1
B1
D2
V2
P2
B2
VPU1
A1
T1
A2
T2
Di : DEMUX 操作
Ai : A-DEC 操作
Bi : BLEND 操作
Vi : V-DEC 操作
Pi : PROG 操作
Ti : TEXT 操作

图10

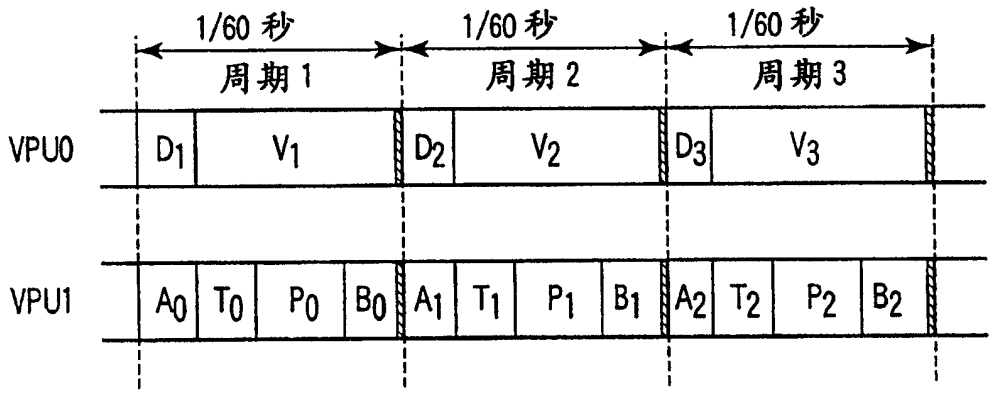

1/60 秒
1/60 秒
1/60 秒
周期 1
周期 2
周期 3
VPU0
D1
V1
D2
V2
D3
V3
VPU1
A0
T0
P0
B0
A1
T1
P1
B1
A2
T2
P2
B2

图11

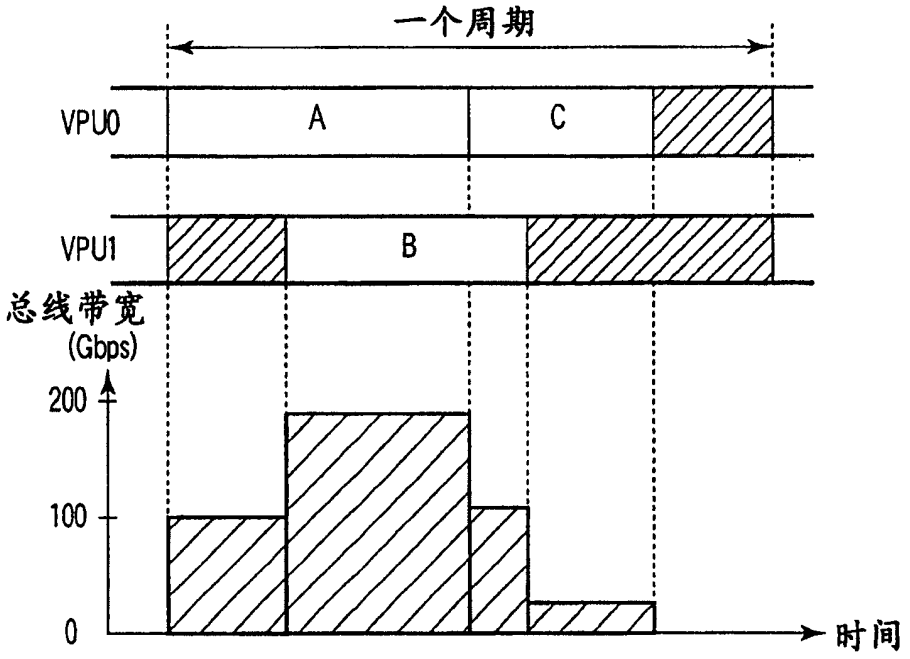

一个周期
VPU0
A
C
VPU1
B
总线带宽
(Gbps)
200
100
0
时间

图12

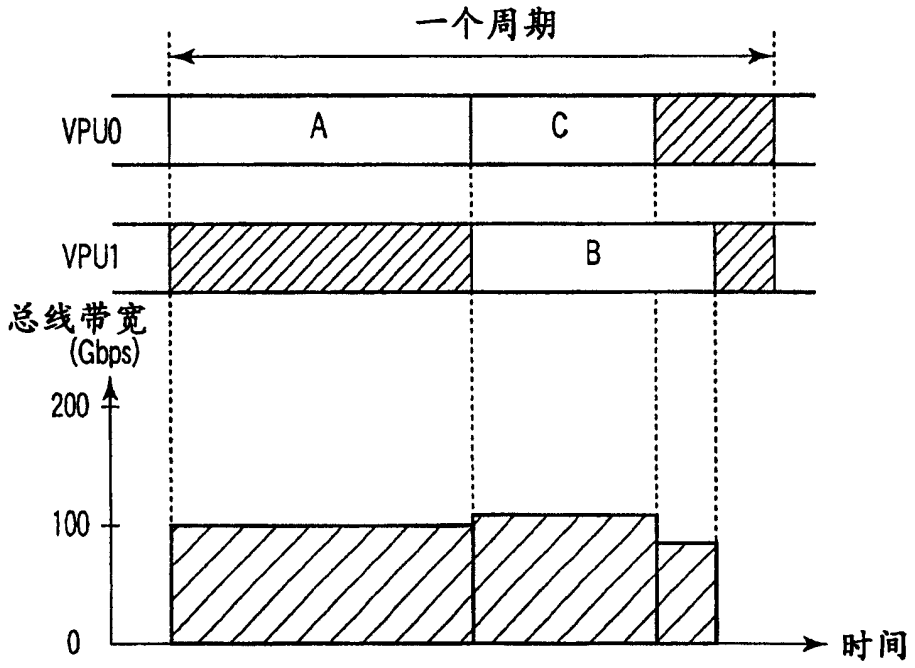

一个周期
VPU0
A
C
VPU1
B
总线带宽
(Gbps)
200
100
0
时间

图13

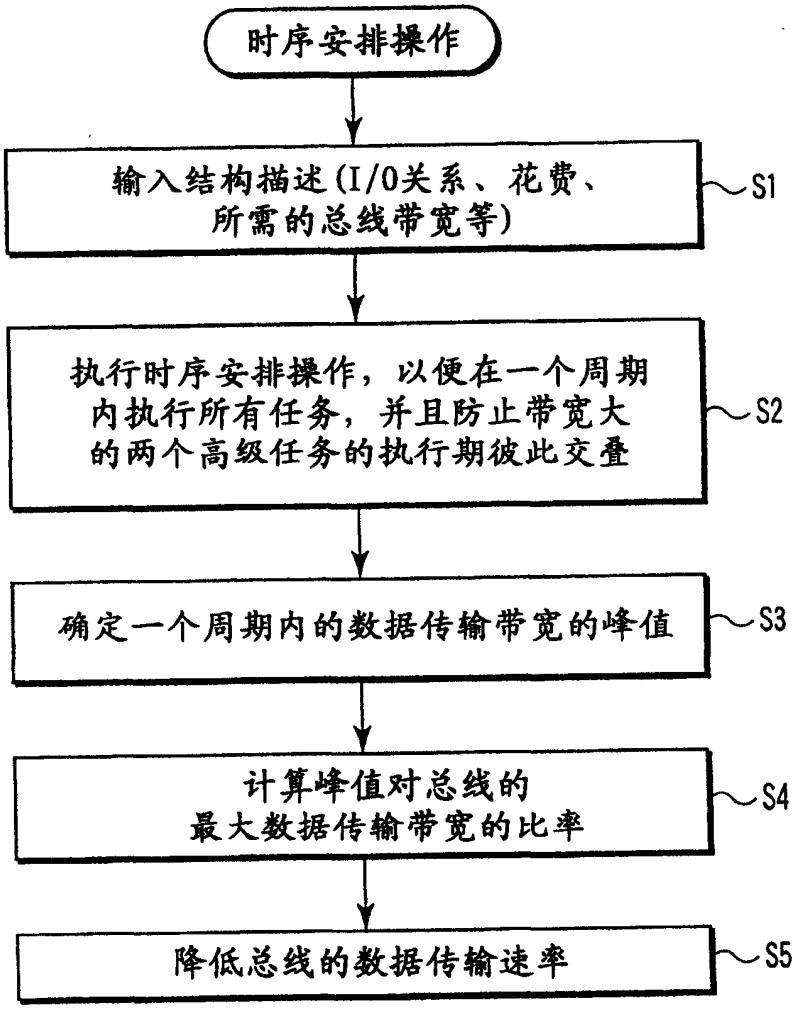

时序安排操作
输入结构描述(I/O关系、花费、所需的总线带宽等)
S1
执行时序安排操作，以便在一个周期内执行所有任务，并且防止带宽大的两个高级任务的执行期彼此交叠
S2
确定一个周期内的数据传输带宽的峰值
S3
计算峰值对总线的最大数据传输带宽的比率
S4
降低总线的数据传输速率
S5

图14

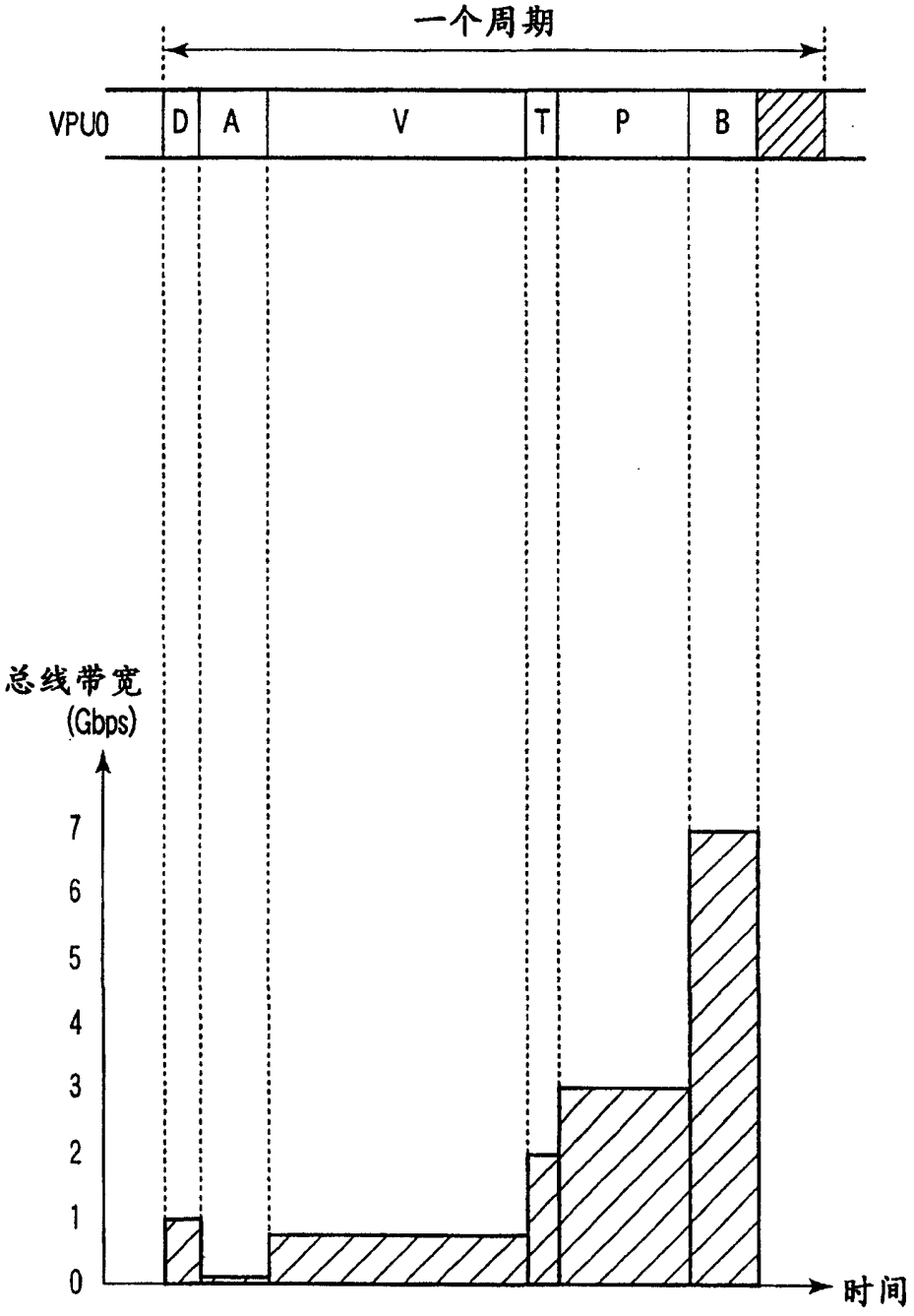


图15

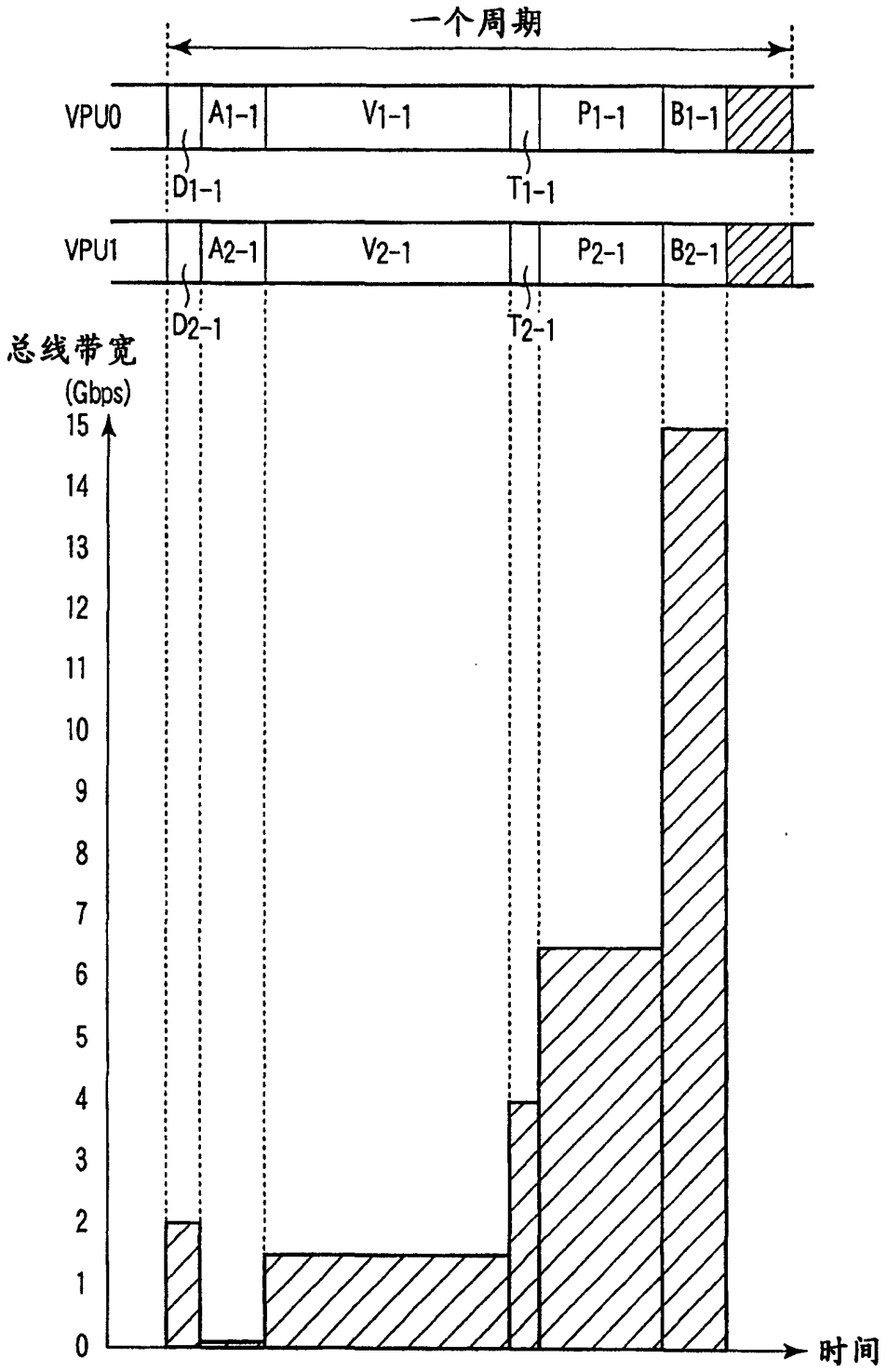

一个周期
VPU0
A1-1
V1-1
P1-1
B1-1
D1-1
T1-1
VPU1
A2-1
V2-1
P2-1
B2-1
D2-1
T2-1
总线带宽
(Gbps)
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
时间

图16

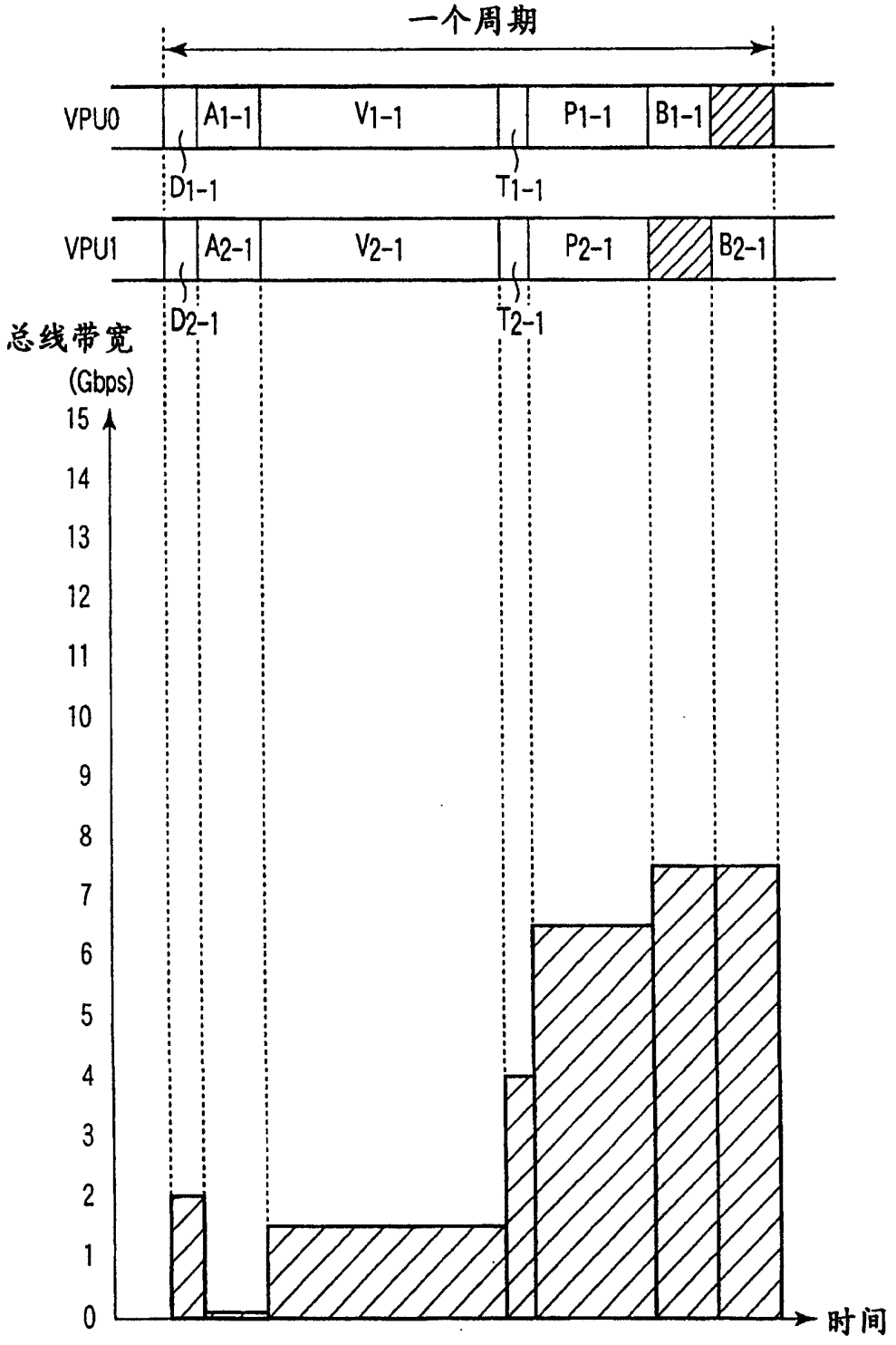

一个周期
VPU0
A1-1
V1-1
P1-1
B1-1
D1-1
T1-1
VPU1
A2-1
V2-1
P2-1
B2-1
D2-1
T2-1
总线带宽
(Gbps)
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
时间

图17

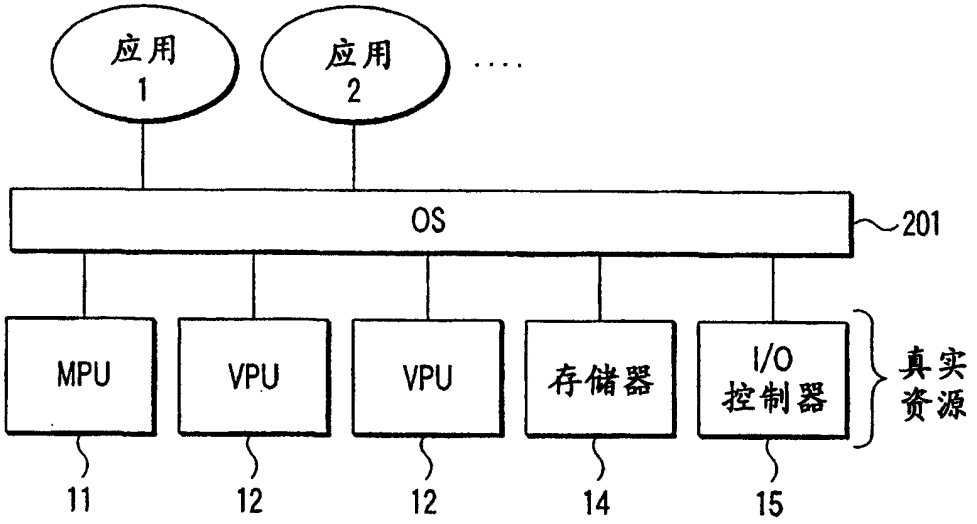

应用
1
应用
2
....
OS
201
MPU
VPU
VPU
存储器
I/O
控制器
真实
资源
11
12
12
14
15

图18

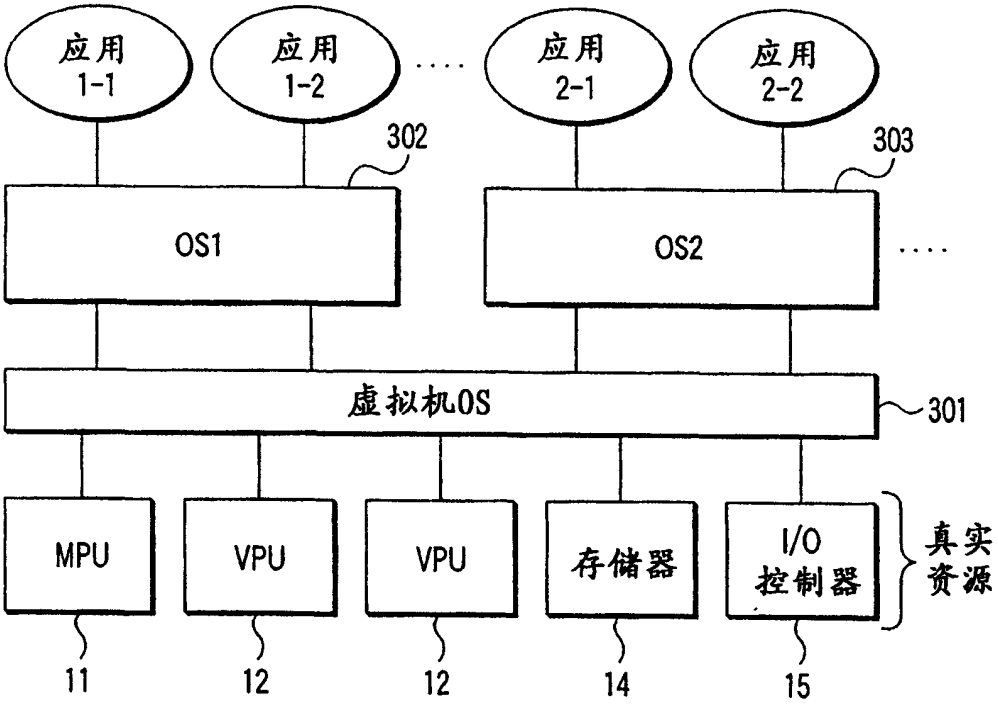

应用
1-1
应用
1-2
....
应用
2-1
应用
2-2
302
303
OS1
OS2
....
虚拟机OS
301
MPU
VPU
VPU
存储器
I/O
控制器
真实
资源
11
12
12
14
15

图19

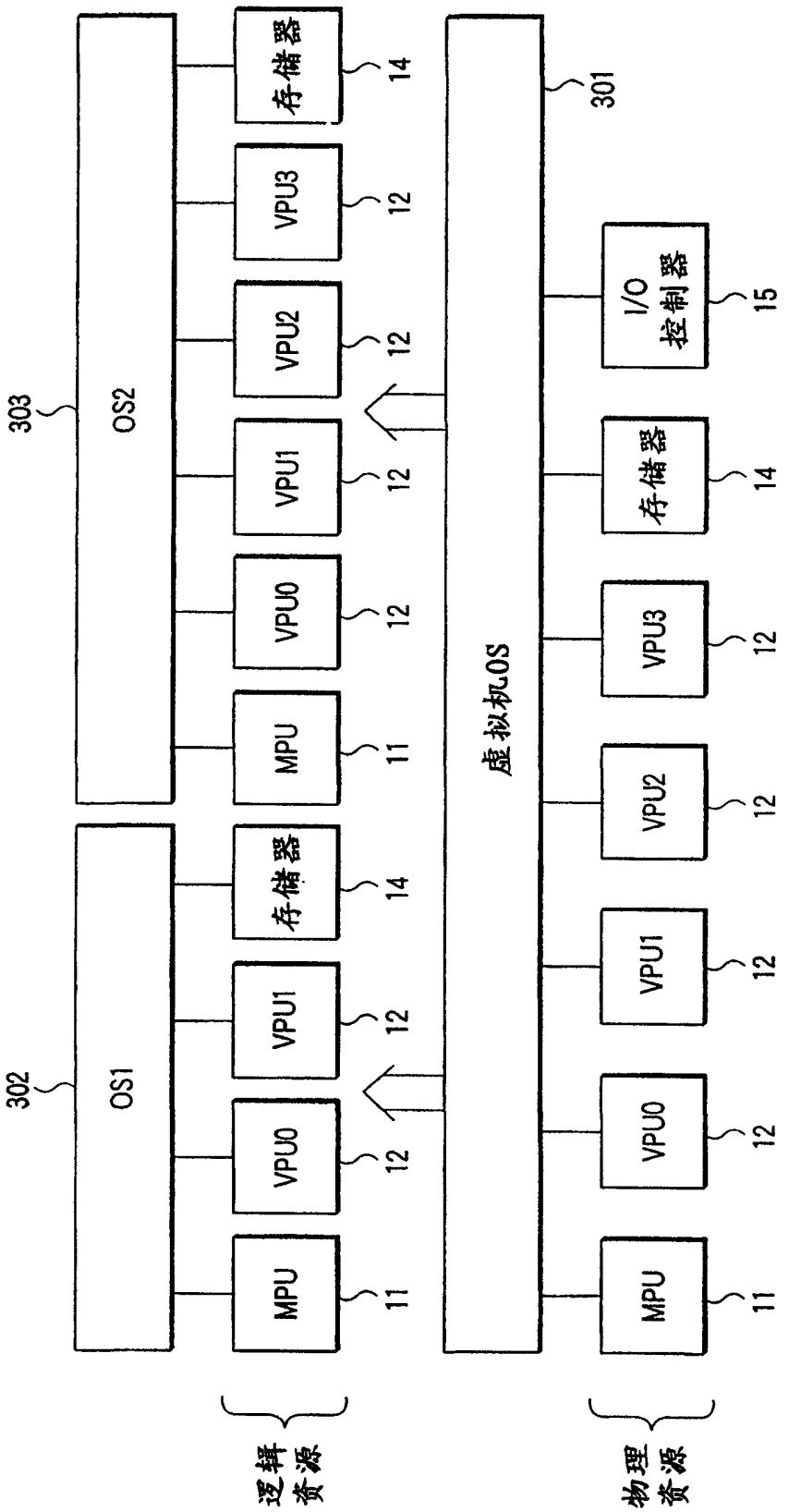

302
OS1
303
OS2
301
虚拟机OS
MPU
VPU0
VPU1
存储器
VPU2
VPU3
I/O
控制器
11
12
14
15
逻辑
资源
物理
资源

图20

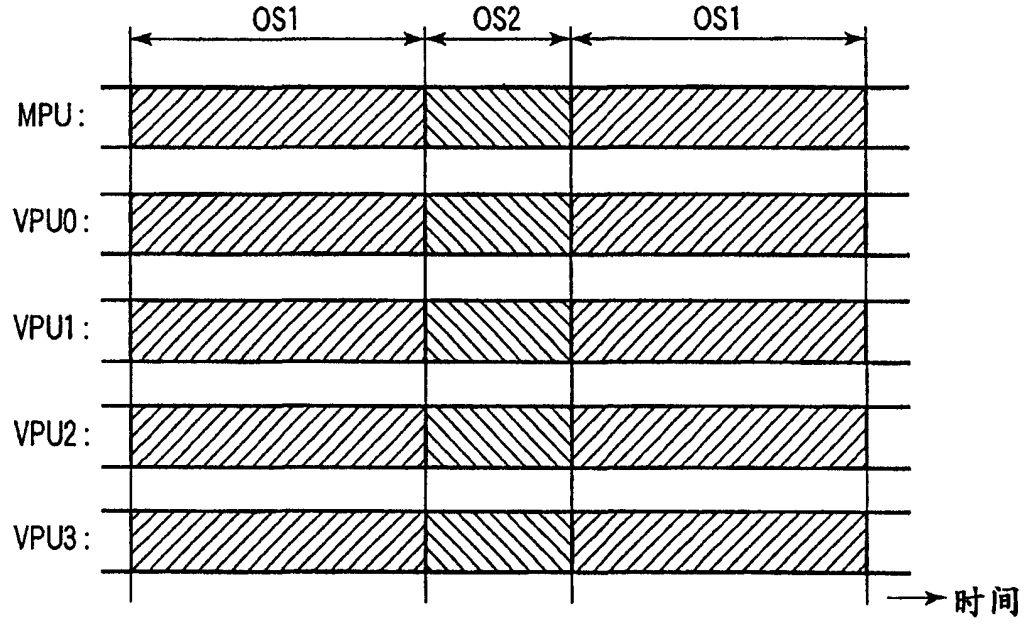

OS1
OS2
OS1
MPU:
VPU0:
VPU1:
VPU2:
VPU3:
时间

图21

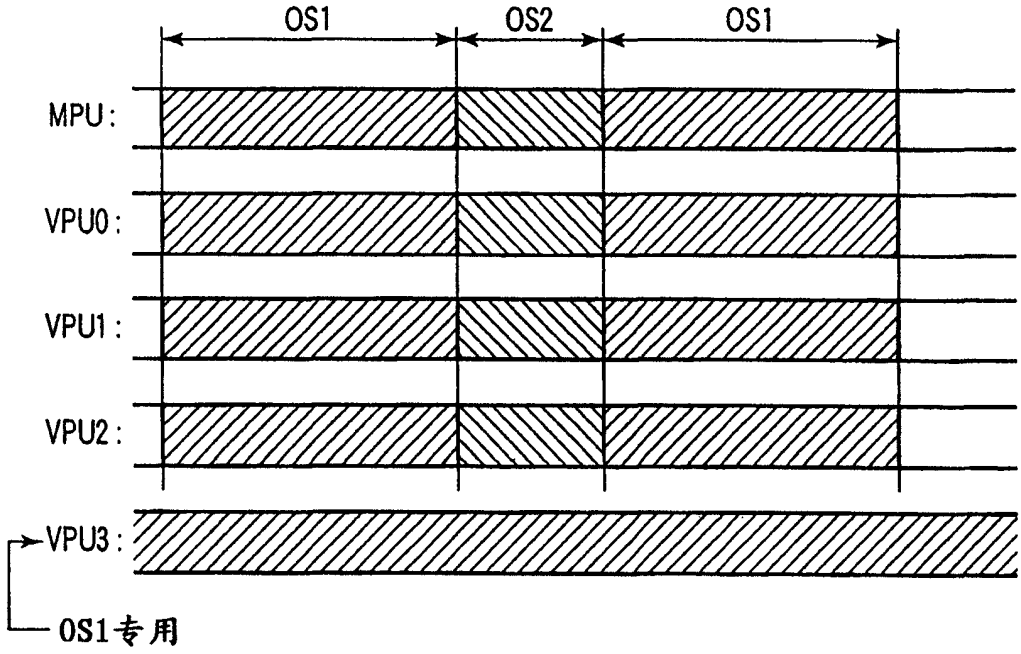

OS1
OS2
OS1
MPU:
VPU0:
VPU1:
VPU2:
VPU3:
OS1专用

图22

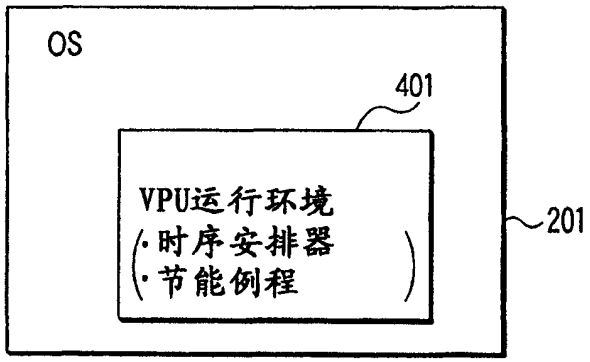

OS
401
VPU运行环境
·时序安排器
·节能例程
201

图23

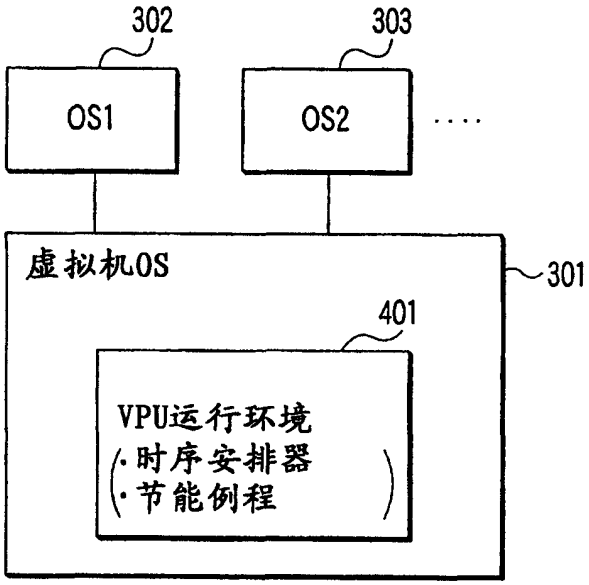

302
303
OS1
OS2
....
虚拟机OS
301
401
VPU运行环境
·时序安排器
·节能例程

图24

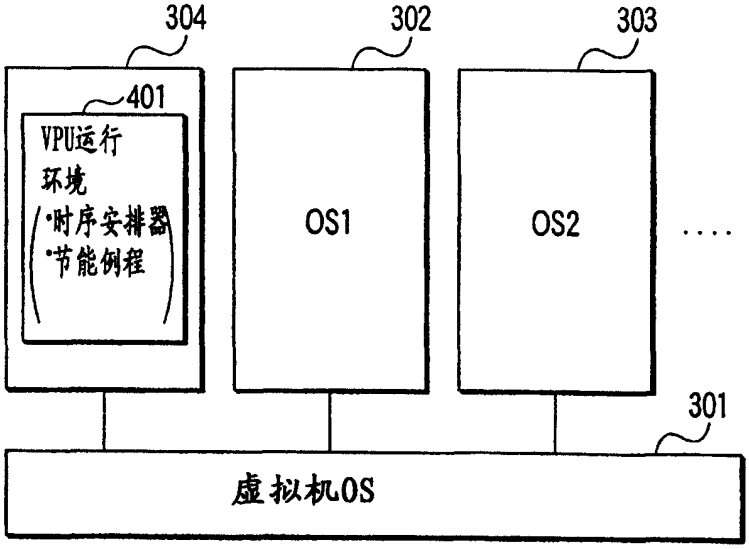

304
302
303
401
VPU运行
环境
·时序安排器
·节能例程
OS1
OS2
....
301
虚拟机OS

图25

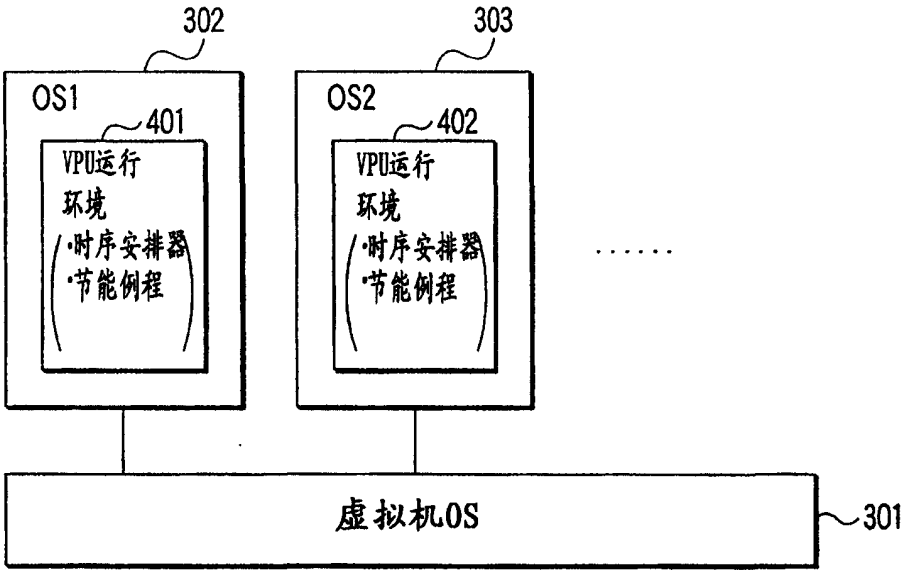

302
303
OS1
401
VPU运行
环境
·时序安排器
·节能例程
OS2
402
VPU运行
环境
·时序安排器
·节能例程
.....
虚拟机OS
301

图26

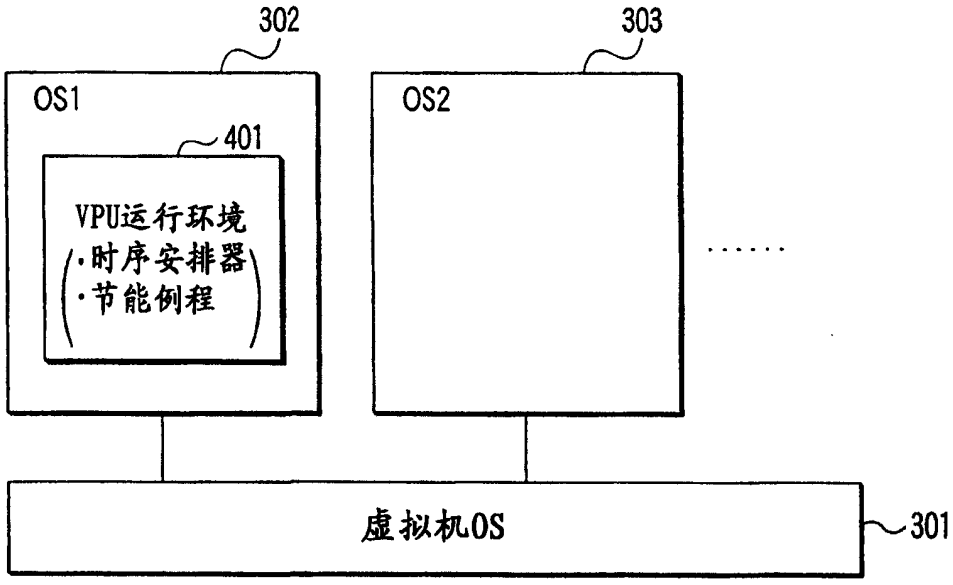

302
303
OS1
401
VPU运行环境
·时序安排器
·节能例程
OS2
.....
虚拟机OS
301

图27

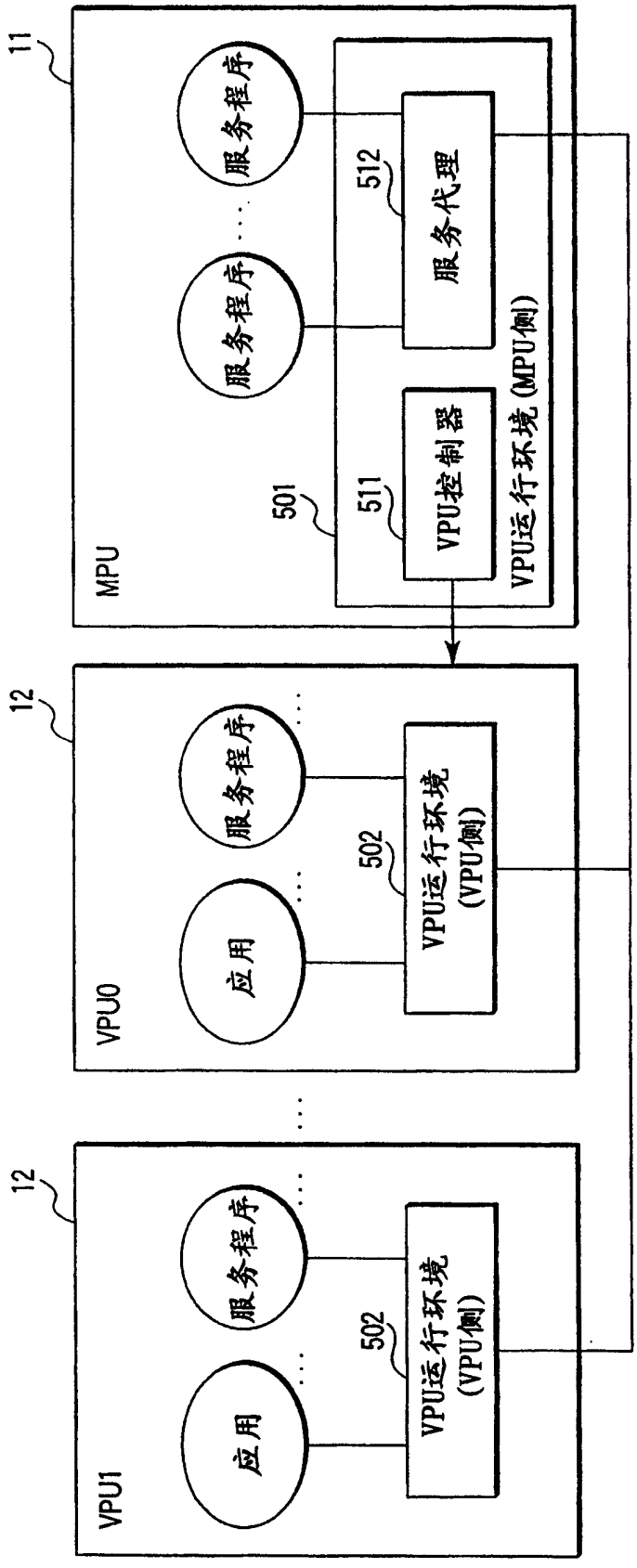

11
MPU
服务程序
服务程序
512
服务代理
501
511
VPU控制器
VPU运行环境(MPU侧)
12
VPU0
应用
服务程序
502
VPU运行环境
(VPU侧)
12
VPU1
应用
服务程序
502
VPU运行环境
(VPU侧)

图28

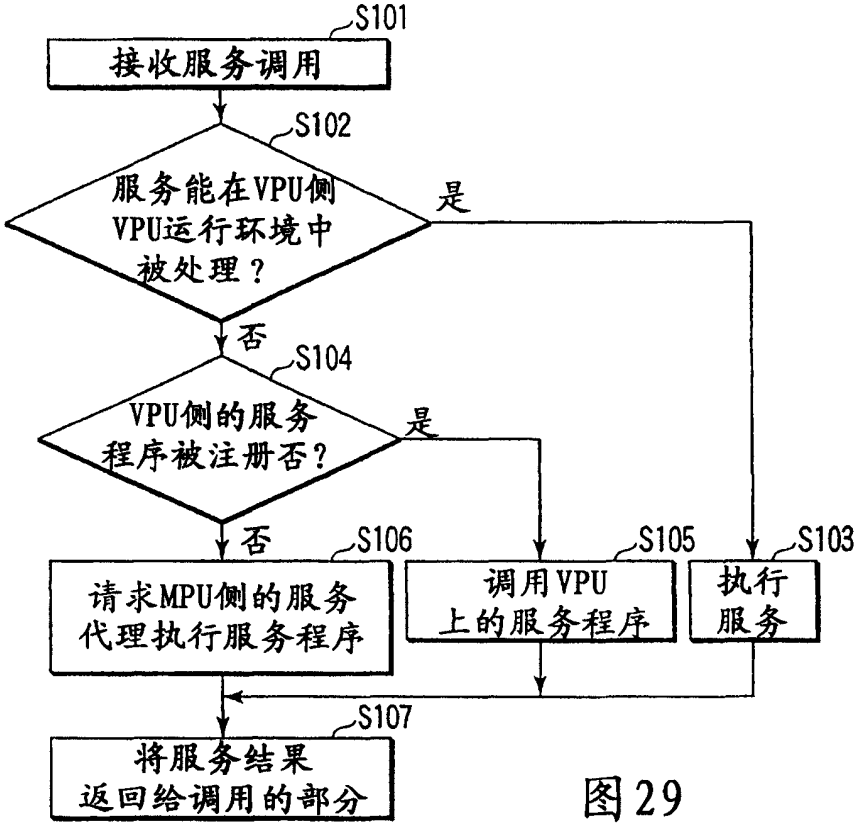

S101
接收服务调用
S102
服务能在VPU侧VPU运行环境中被处理？
是
否
S104
VPU侧的服务程序被注册否？
是
否
S106
请求MPU侧的服务代理执行服务程序
S105
调用VPU上的服务程序
S103
执行服务
S107
将服务结果返回给调用的部分

图29

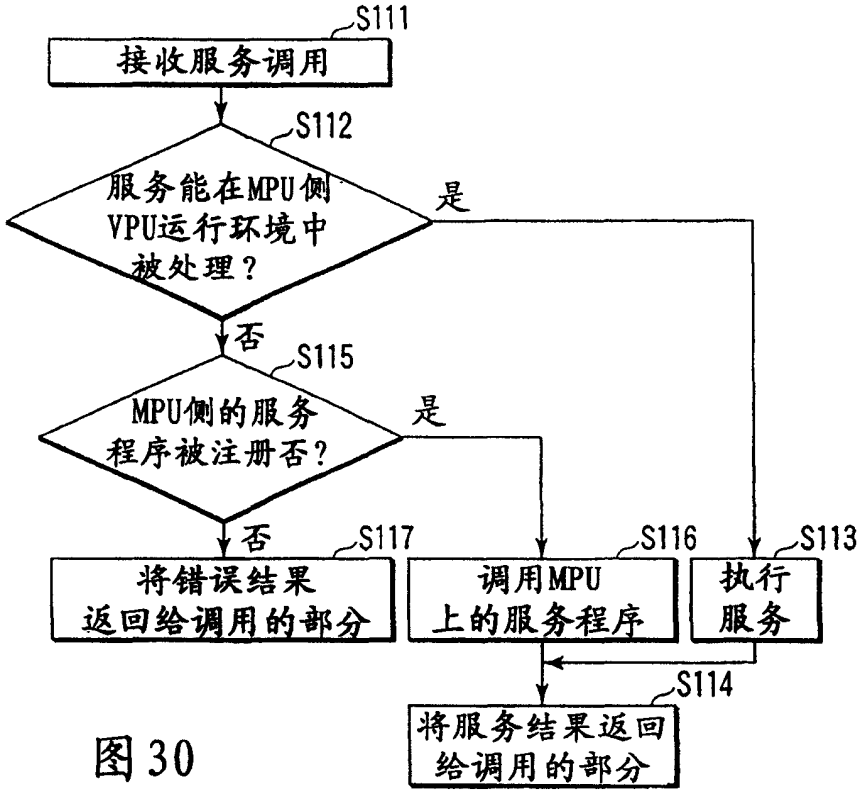

S111
接收服务调用
S112
服务能在MPU侧VPU运行环境中被处理？
是
否
S115
MPU侧的服务程序被注册否？
是
否
S117
将错误结果返回给调用的部分
S116
调用MPU上的服务程序
S113
执行服务
S114
将服务结果返回给调用的部分

图30

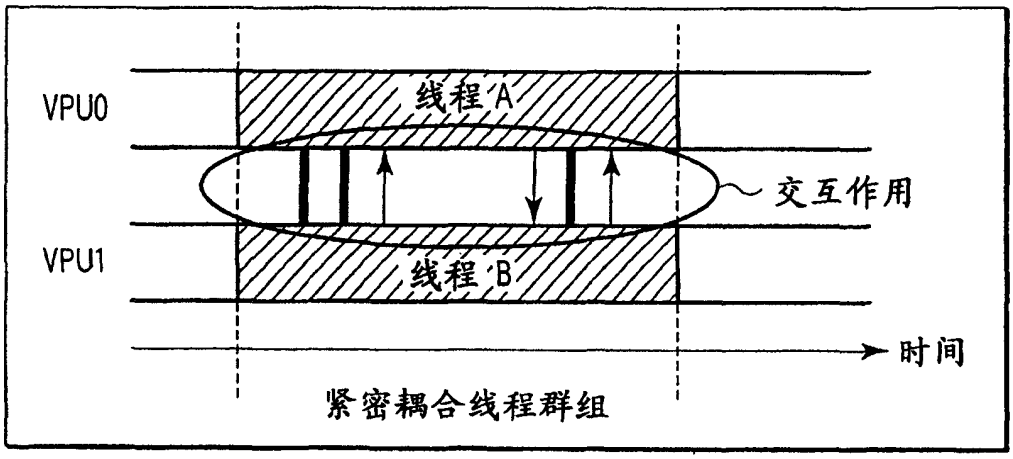

VPU0
线程A
交互作用
VPU1
线程B
时间
紧密耦合线程群组

图31

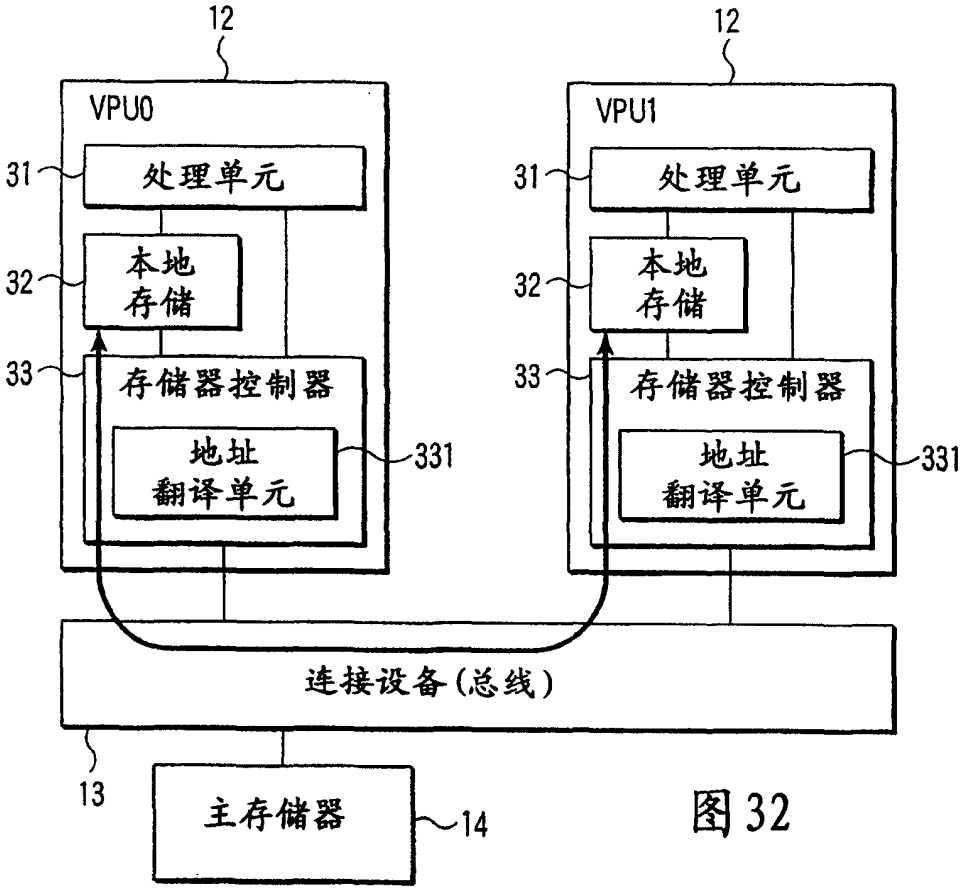

12
VPU0
31
处理单元
32
本地存储
33
存储器控制器
地址翻译单元
331
12
VPU1
31
处理单元
32
本地存储
33
存储器控制器
地址翻译单元
331
连接设备(总线)
13
主存储器
14

图32

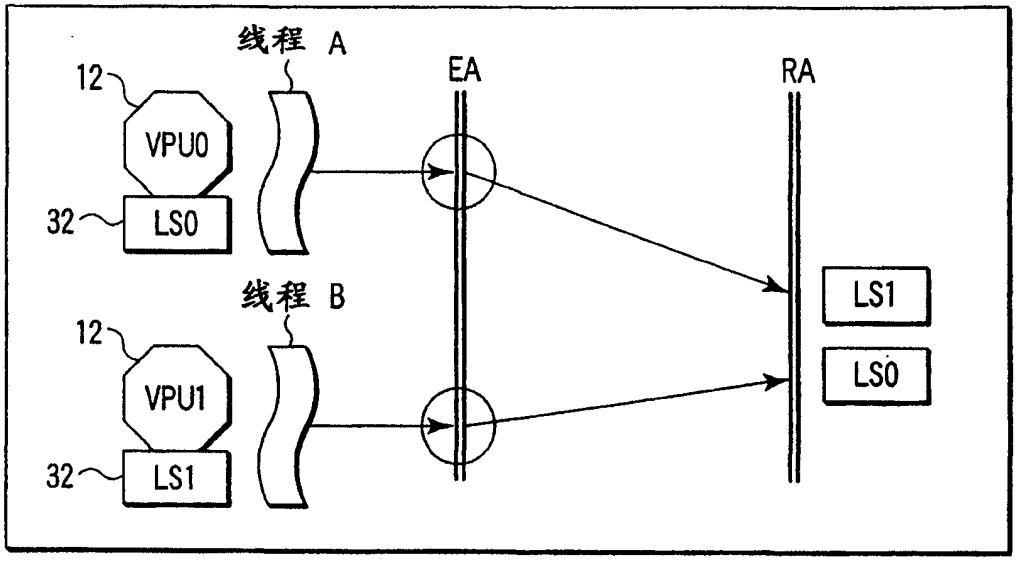

线程 A
12
VPU0
32
LS0
EA
RA
LS1
LS0
线程 B
12
VPU1
32
LS1

图33

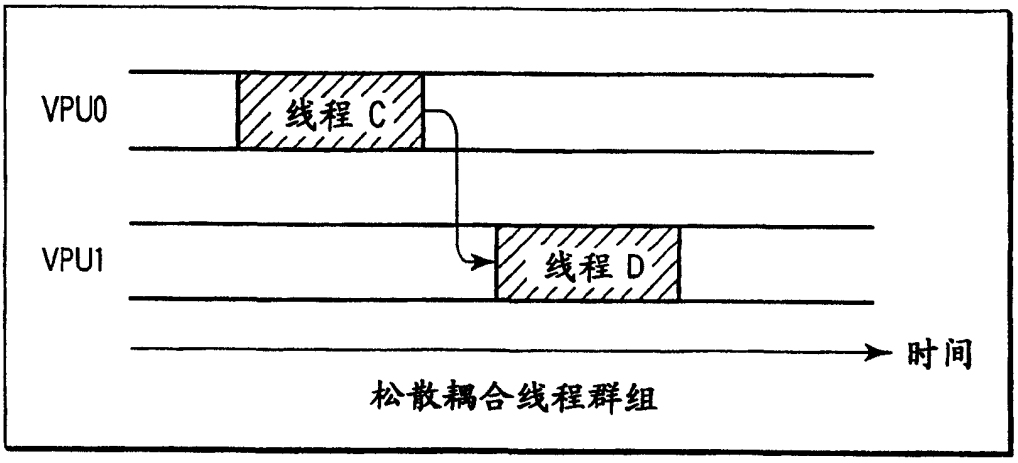

VPU0
线程 C
VPU1
线程 D
时间
松散耦合线程群组

图34

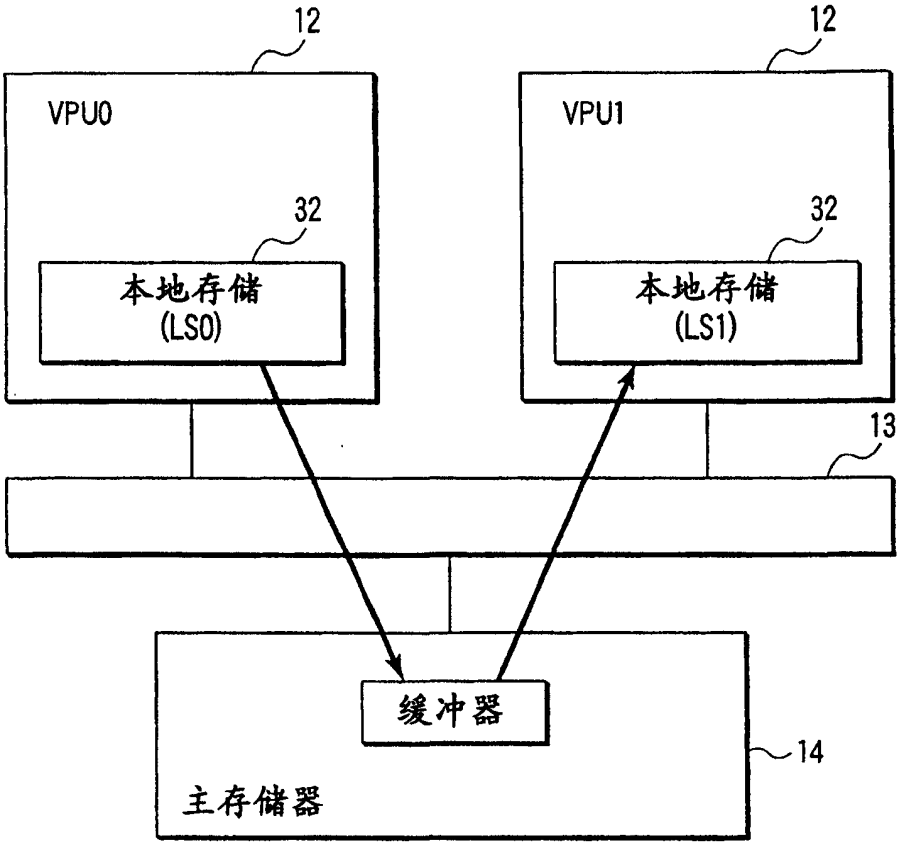

12
12
VPU0
VPU1
32
32
本地存储
(LS0)
本地存储
(LS1)
13
缓冲器
14
主存储器

图35

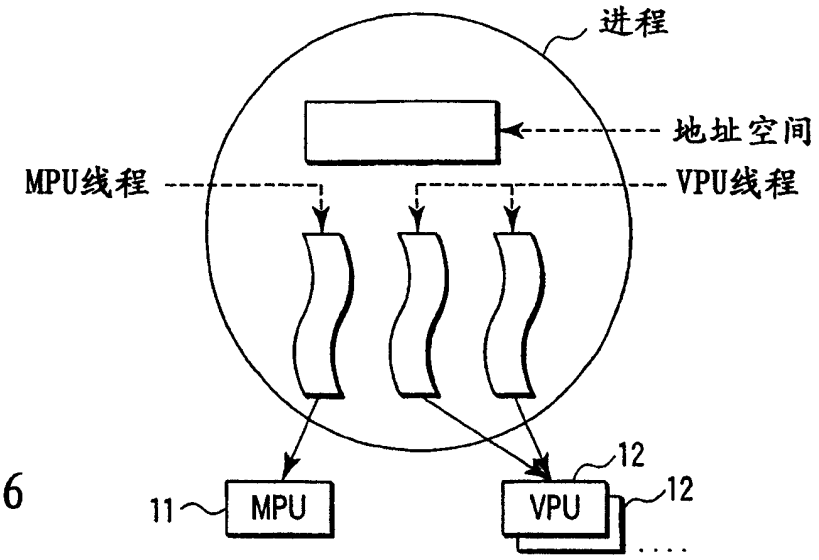

进程
地址空间
MPU线程
VPU线程
12
11
MPU
VPU
12
....

图36

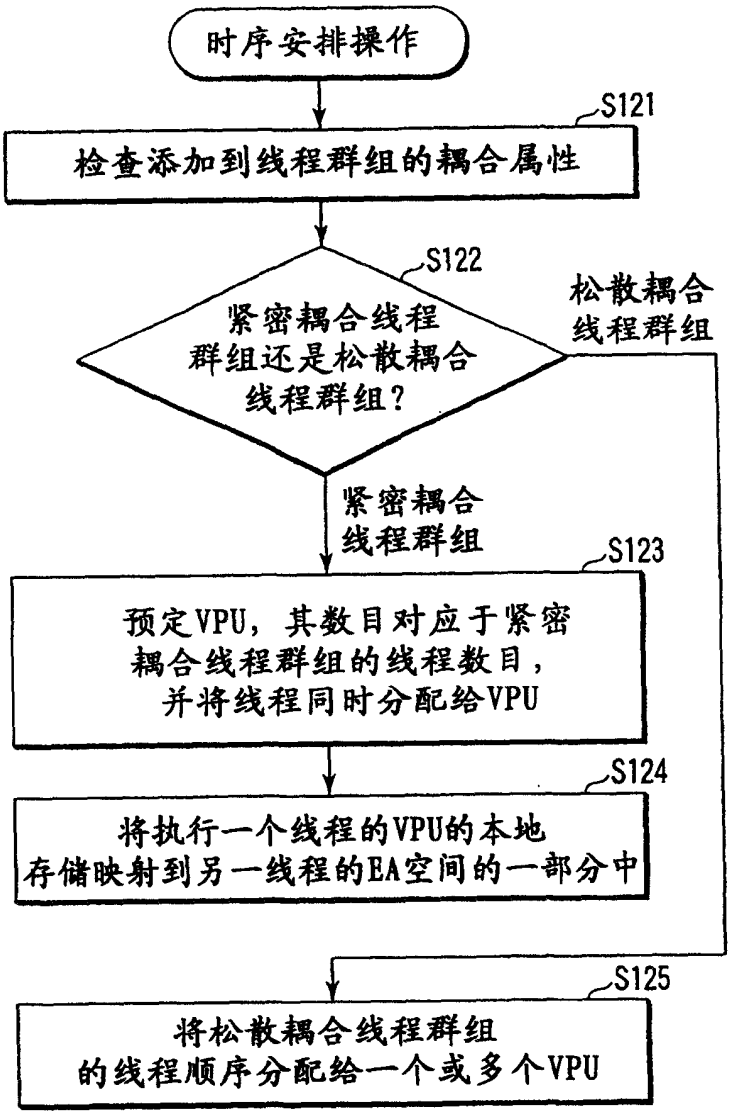

时序安排操作
S121
检查添加到线程群组的耦合属性
S122
紧密耦合线程群组还是松散耦合线程群组？
松散耦合线程群组
紧密耦合线程群组
S123
预定VPU，其数目对应于紧密耦合线程群组的线程数目，并将线程同时分配给VPU
S124
将执行一个线程的VPU的本地存储映射到另一线程的EA空间的一部分中
S125
将松散耦合线程群组的线程顺序分配给一个或多个VPU

图37

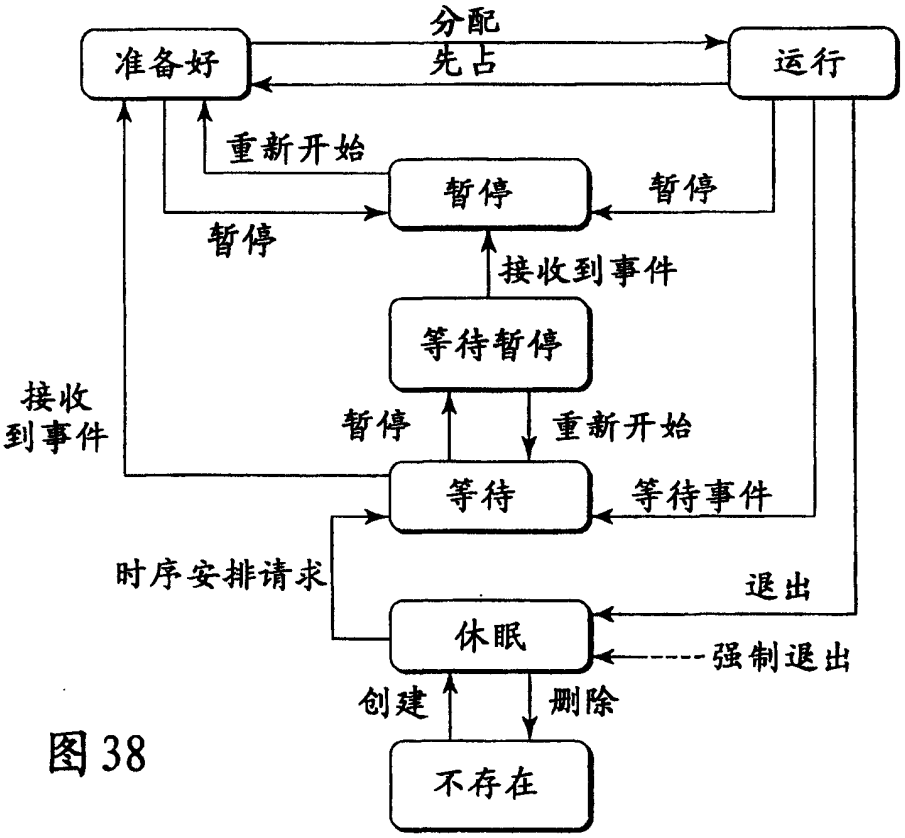

分配
准备好
先占
运行
重新开始
暂停
暂停
暂停
接收到事件
等待暂停
接收
到事件
暂停
重新开始
等待
等待事件
时序安排请求
退出
休眠
强制退出
创建
删除
不存在

图 38

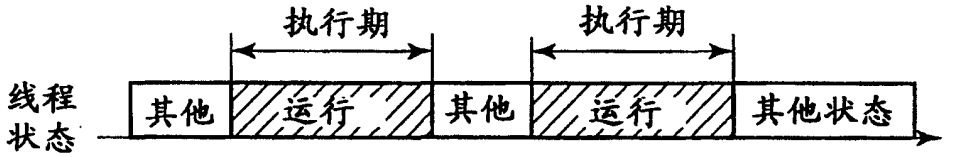

执行期
执行期
线程
状态
其他
运行
其他
运行
其他状态
时间

图 39

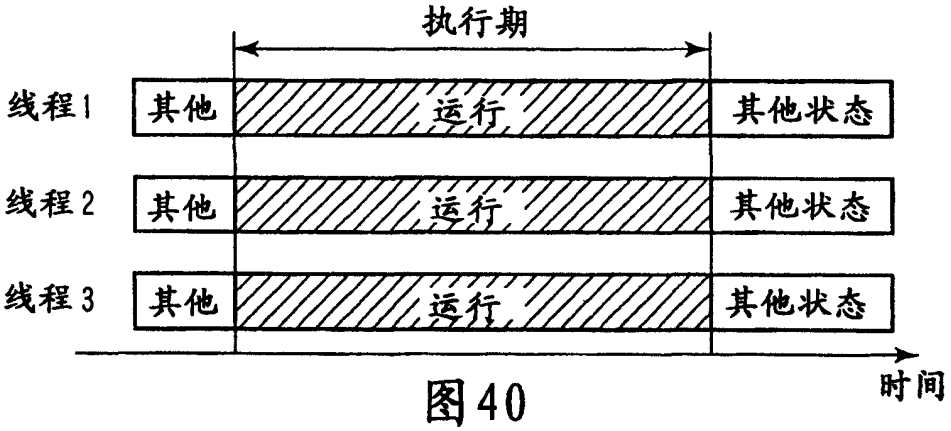

执行期
线程1
其他
运行
其他状态
线程2
其他
运行
其他状态
线程3
其他
运行
其他状态
时间

图 40

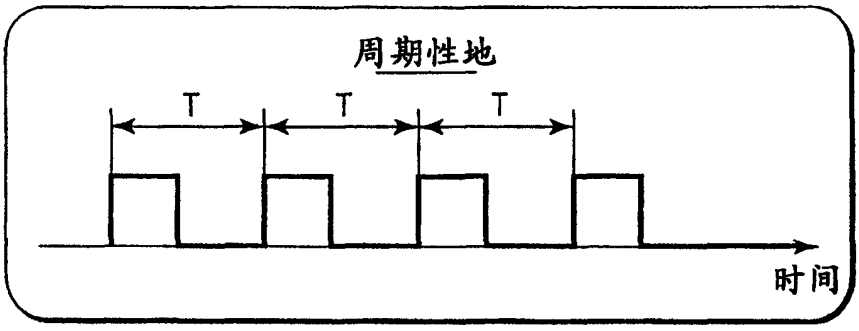


图41

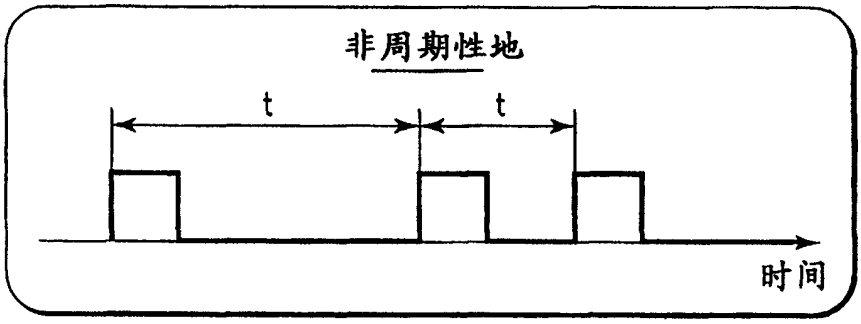


图42

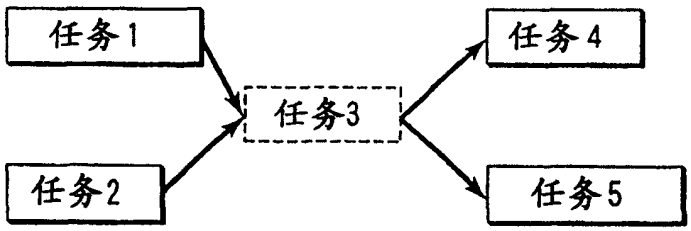


图43

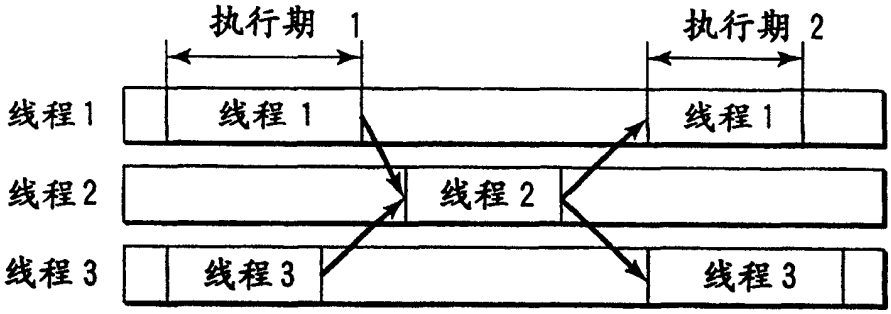


图44

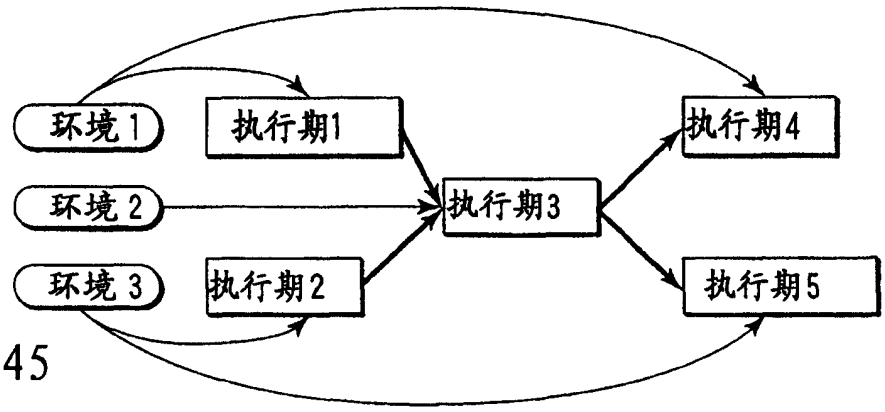

环境 1
环境 2
环境 3
执行期1
执行期2
执行期3
执行期4
执行期5

图 45

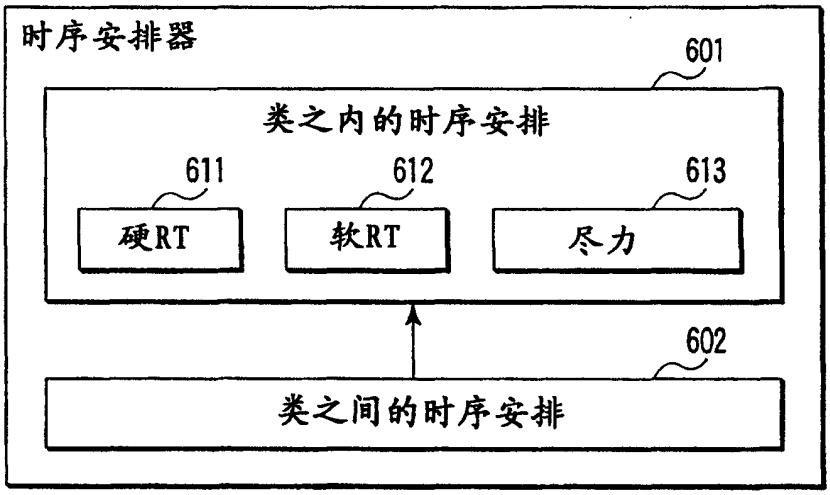

时序安排器
601
类之内的时序安排
611
612
613
硬RT
软RT
尽力
602
类之间的时序安排

图 46

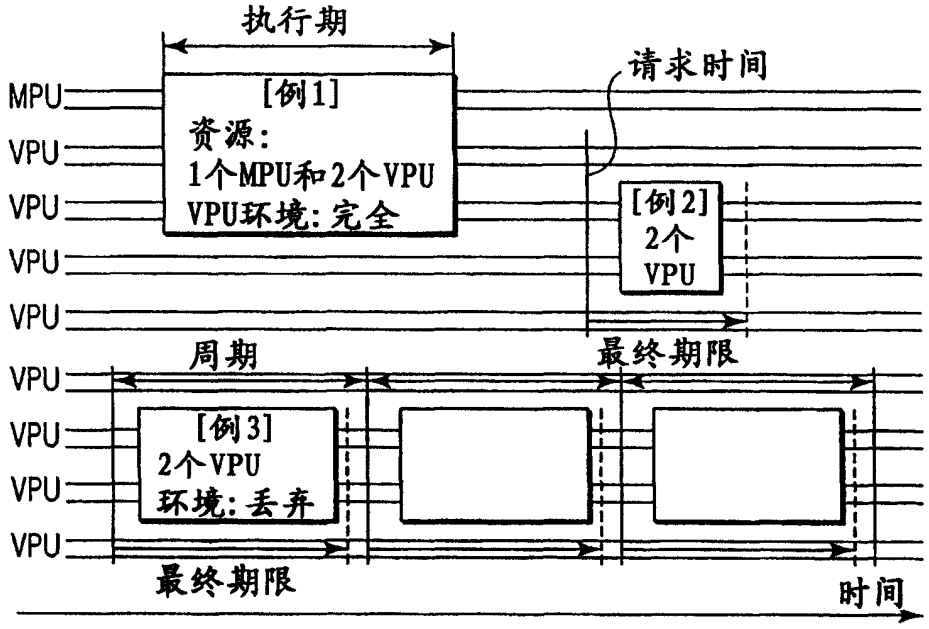

执行期
请求时间
MPU
VPU
VPU
VPU
VPU
VPU
VPU
VPU
VPU
VPU
[例1]
资源:
1个MPU和2个VPU
VPU环境: 完全
[例2]
2个
VPU
周期
最终期限
[例3]
2个VPU
环境: 丢弃
最终期限
时间

图 47

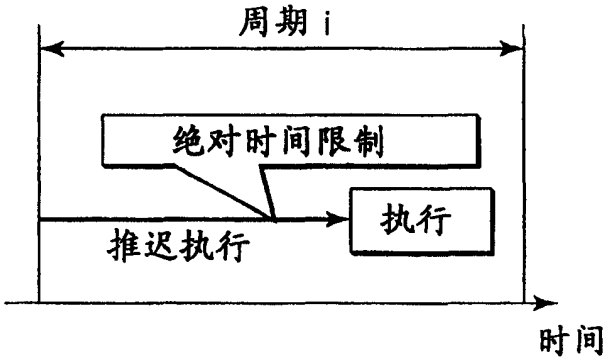


图 48

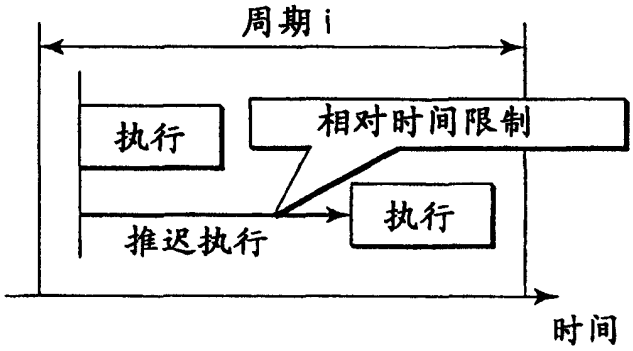


图 49

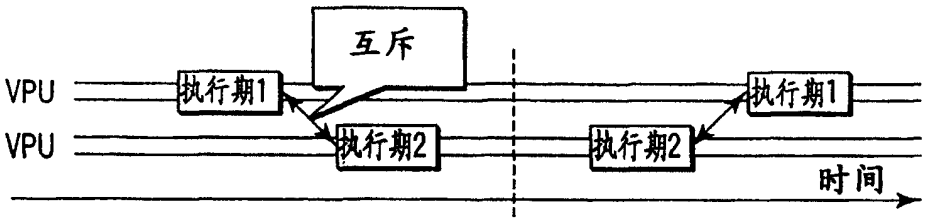


图 50

		紧密耦合线程群组	松散耦合线程群组
在存储器上	LS	能使用	不能使用
	MS	能使用	
其他		应该使用 硬件基元	应该使用由 VPU运行环境提供的机制

图 51

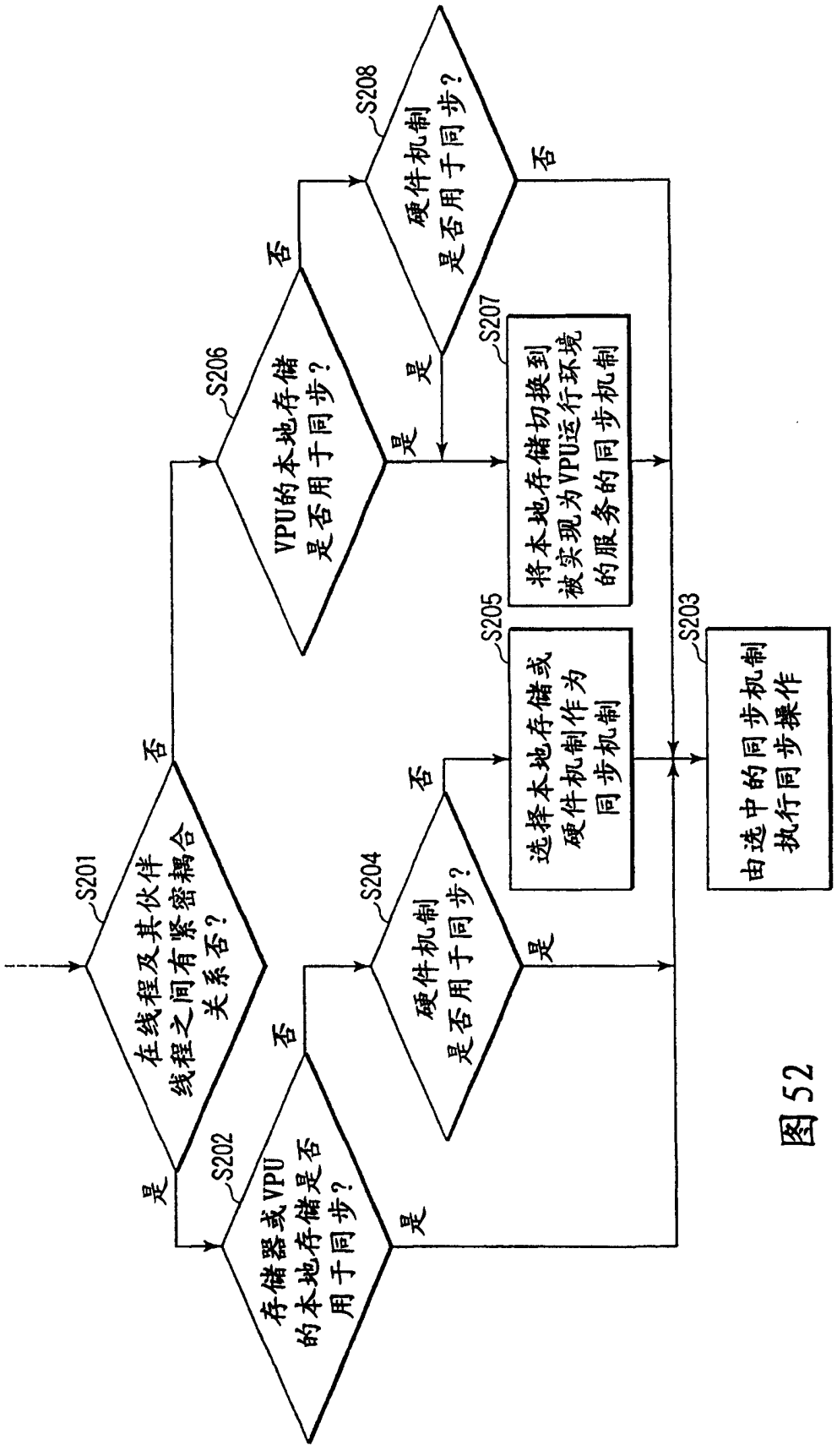

S201
在线程及其伙伴线程之间有紧密耦合关系否?
是
否
S202
存储器或VPU的本地存储是否用于同步?
是
否
S204
硬件机制是否用于同步?
是
否
S205
选择本地存储或硬件机制作为同步机制
S206
VPU的本地存储是否用于同步?
是
否
S207
将本地存储切换到被实现为VPU运行环境的服务的同步机制
S208
硬件机制是否用于同步?
是
否
S203
由选中的同步机制执行同步操作

图52

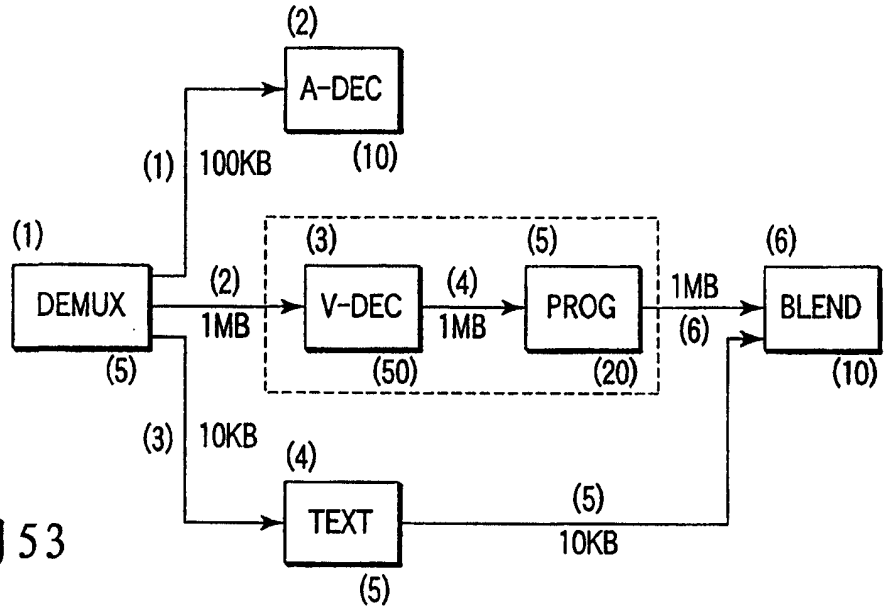

(2)
A-DEC
(1) 100KB
(10)
(1)
(3)
(5)
(6)
DEMUX
(2)
1MB
V-DEC
(4)
1MB
PROG
1MB
(6)
BLEND
(5)
(50)
(20)
(10)
(3) 10KB
(4)
(5)
TEXT
10KB
(5)

图53

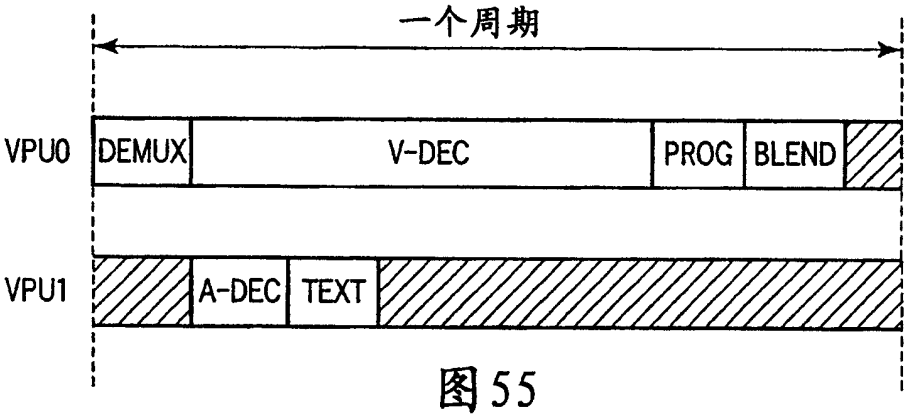

一个周期
VPU0
DEMUX
V-DEC
PROG
BLEND
VPU1
A-DEC
TEXT

图55

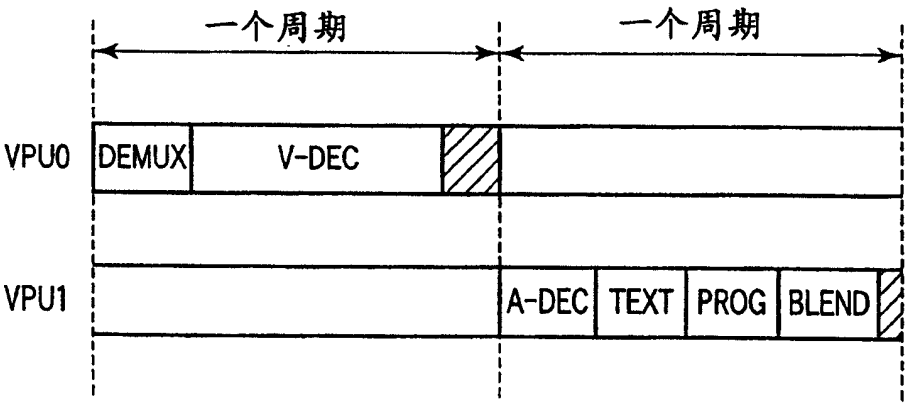

一个周期
一个周期
VPU0
DEMUX
V-DEC
VPU1
A-DEC
TEXT
PROG
BLEND

图56

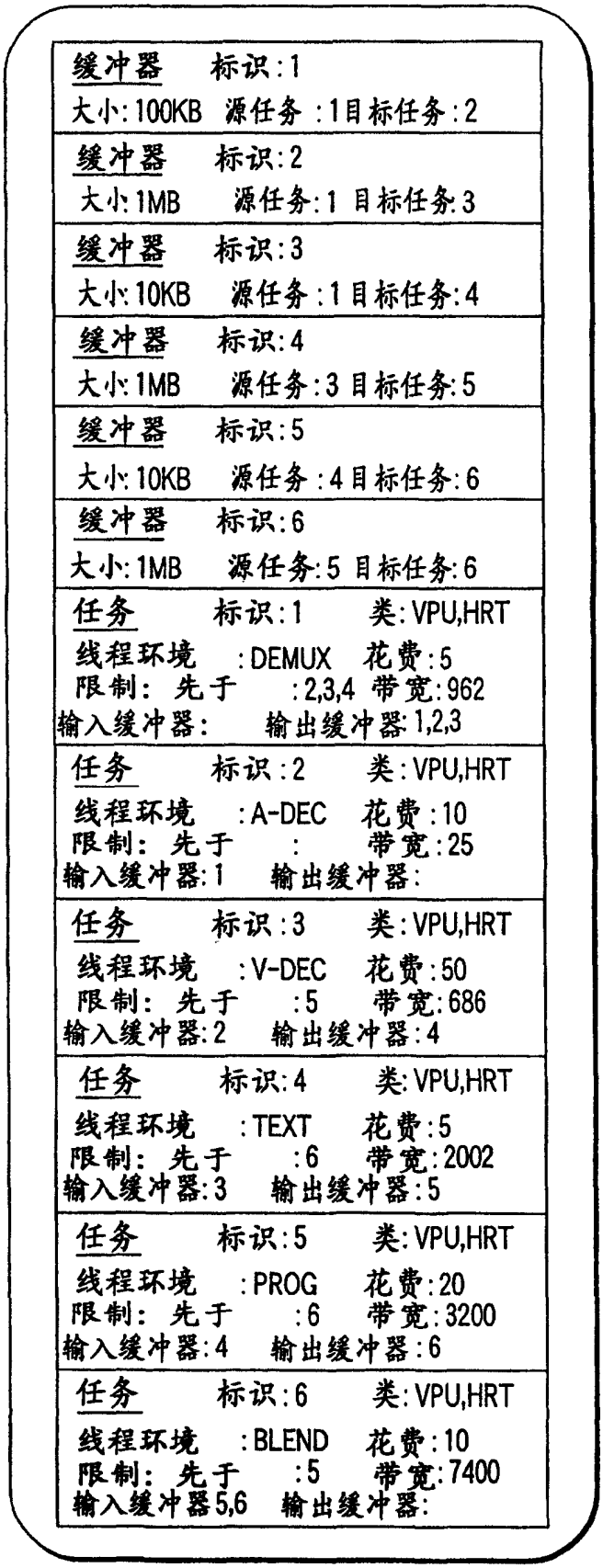

缓冲器 标识:1
大小:100KB 源任务 :1目标任务:2
缓冲器 标识:2
大小:1MB 源任务:1 目标任务:3
缓冲器 标识:3
大小:10KB 源任务 :1目标任务:4
缓冲器 标识:4
大小:1MB 源任务:3 目标任务:5
缓冲器 标识:5
大小:10KB 源任务 :4目标任务:6
缓冲器 标识:6
大小:1MB 源任务:5 目标任务:6
任务 标识:1 类:VPU,HRT
线程环境 :DEMUX 花费:5
限制: 先于 :2,3,4 带宽:962
输入缓冲器: 输出缓冲器:1,2,3
任务 标识:2 类:VPU,HRT
线程环境 :A-DEC 花费:10
限制: 先于 : 带宽:25
输入缓冲器:1 输出缓冲器:
任务 标识:3 类:VPU,HRT
线程环境 :V-DEC 花费:50
限制: 先于 :5 带宽:686
输入缓冲器:2 输出缓冲器:4
任务 标识:4 类:VPU,HRT
线程环境 :TEXT 花费:5
限制: 先于 :6 带宽:2002
输入缓冲器:3 输出缓冲器:5
任务 标识:5 类:VPU,HRT
线程环境 :PROG 花费:20
限制: 先于 :6 带宽:3200
输入缓冲器:4 输出缓冲器:6
任务 标识:6 类:VPU,HRT
线程环境 :BLEND 花费:10
限制: 先于 :5 带宽:7400
输入缓冲器5,6 输出缓冲器:

图54

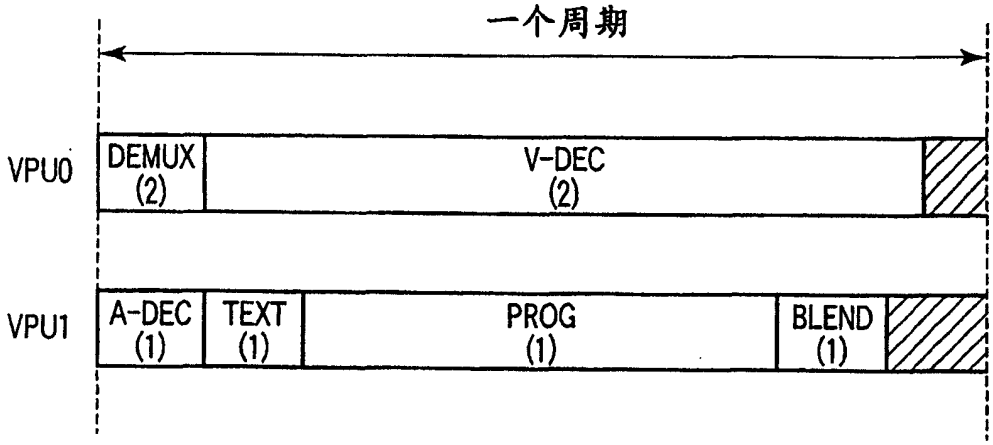

一个周期
VPU0
DEMUX
(2)
V-DEC
(2)
VPU1
A-DEC
(1)
TEXT
(1)
PROG
(1)
BLEND
(1)

图57

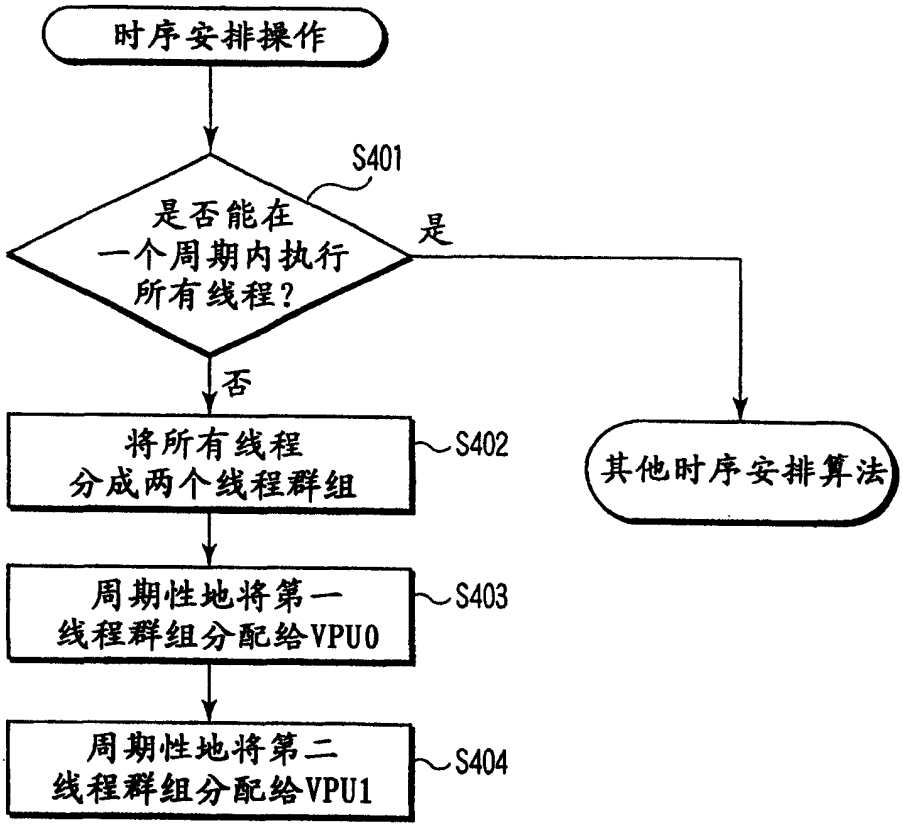

时序安排操作
S401
是否能在
一个周期内执行
所有线程?
是
否
将所有线程
分成两个线程群组
S402
其他时序安排算法
周期性地将第一
线程群组分配给VPU0
S403
周期性地将第二
线程群组分配给VPU1
S404

图58

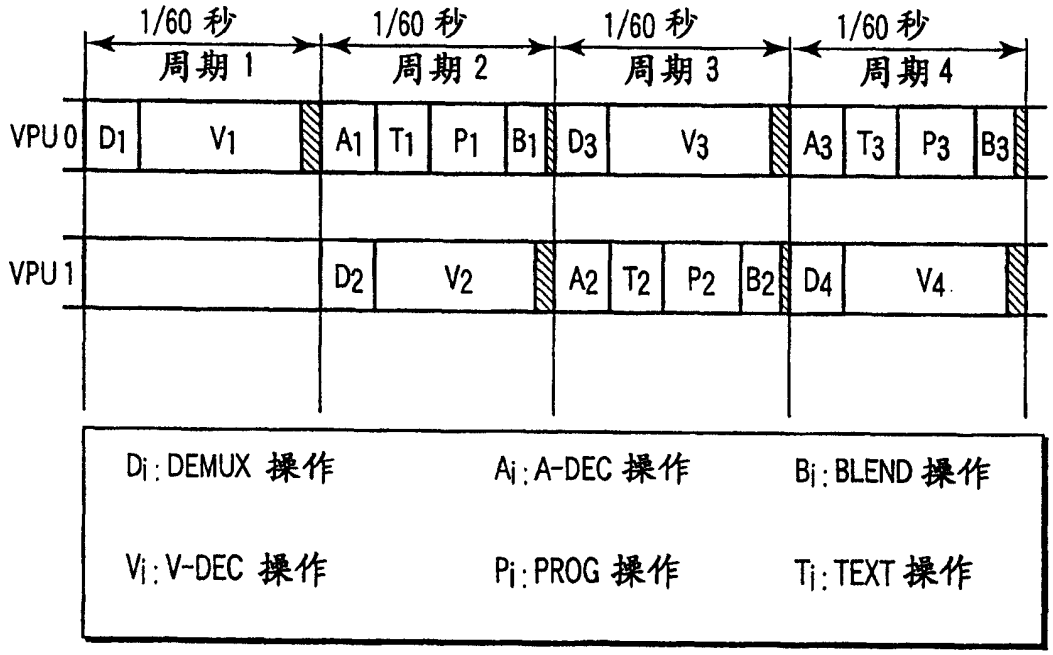

1/60 秒
周期 1
1/60 秒
周期 2
1/60 秒
周期 3
1/60 秒
周期 4
VPU 0
D1
V1
A1
T1
P1
B1
D3
V3
A3
T3
P3
B3
VPU 1
D2
V2
A2
T2
P2
B2
D4
V4
Di: DEMUX 操作
Ai: A-DEC 操作
Bi: BLEND 操作
Vi: V-DEC 操作
Pi: PROG 操作
Ti: TEXT 操作

图59

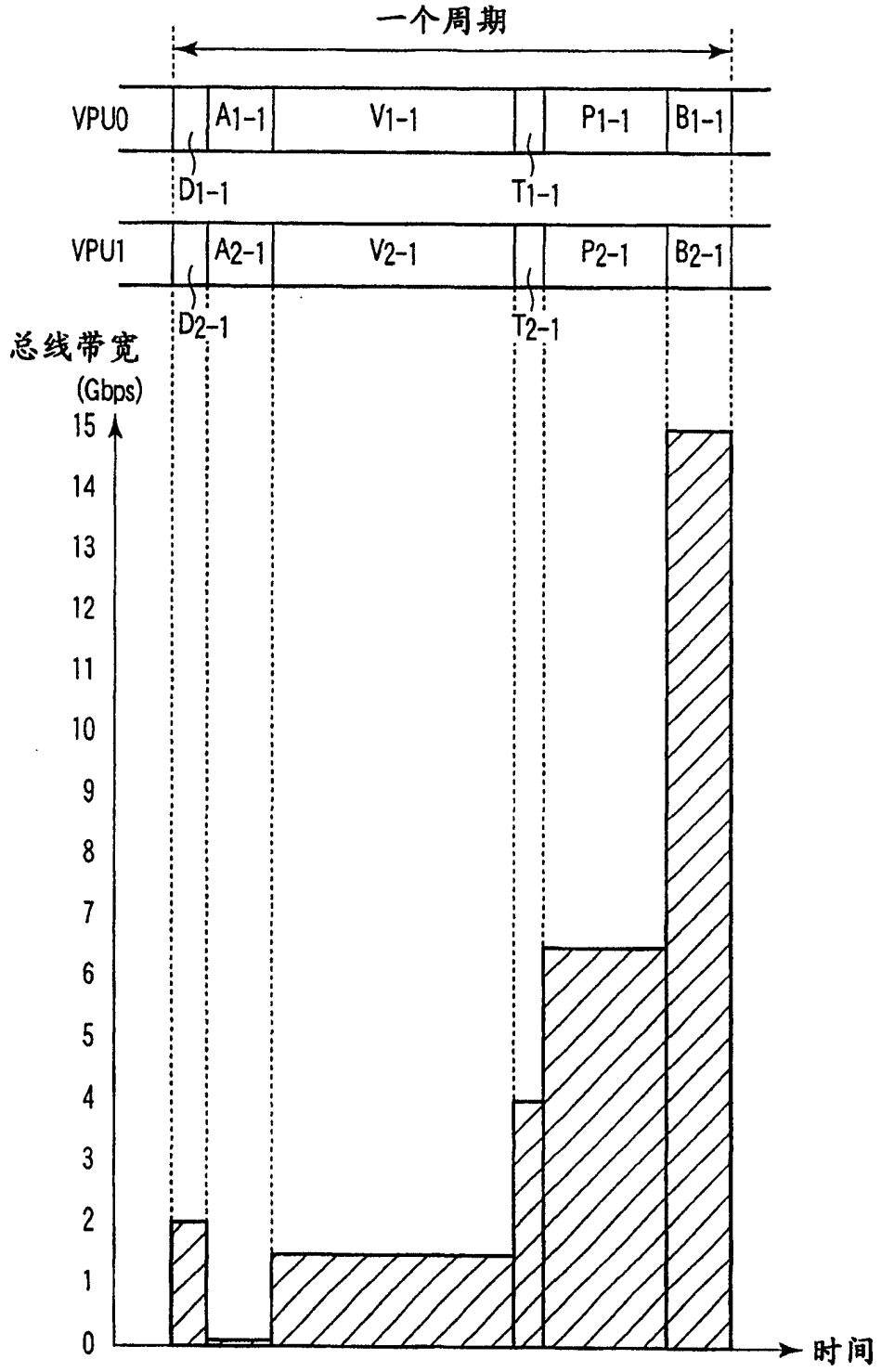

一个周期
VPU0
A1-1
V1-1
P1-1
B1-1
D1-1
T1-1
VPU1
A2-1
V2-1
P2-1
B2-1
D2-1
T2-1
总线带宽
(Gbps)
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
时间

图60

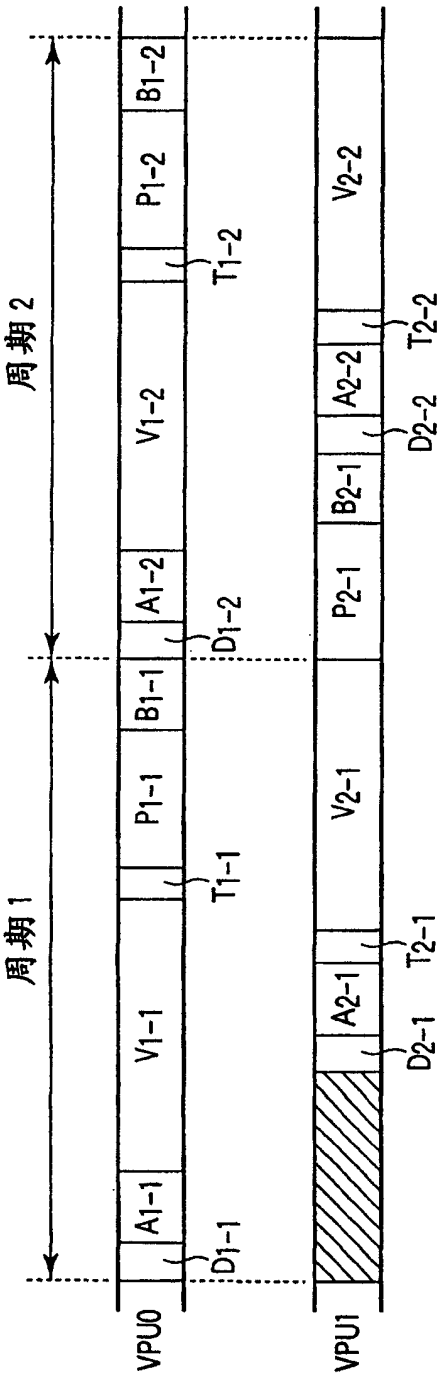

周期1
周期2
VPU0
VPU1
A1-1
V1-1
P1-1
B1-1
A1-2
V1-2
P1-2
B1-2
D1-1
T1-1
D1-2
T1-2
A2-1
V2-1
P2-1
B2-1
A2-2
V2-2
D2-1
T2-1
D2-2
T2-2

图61

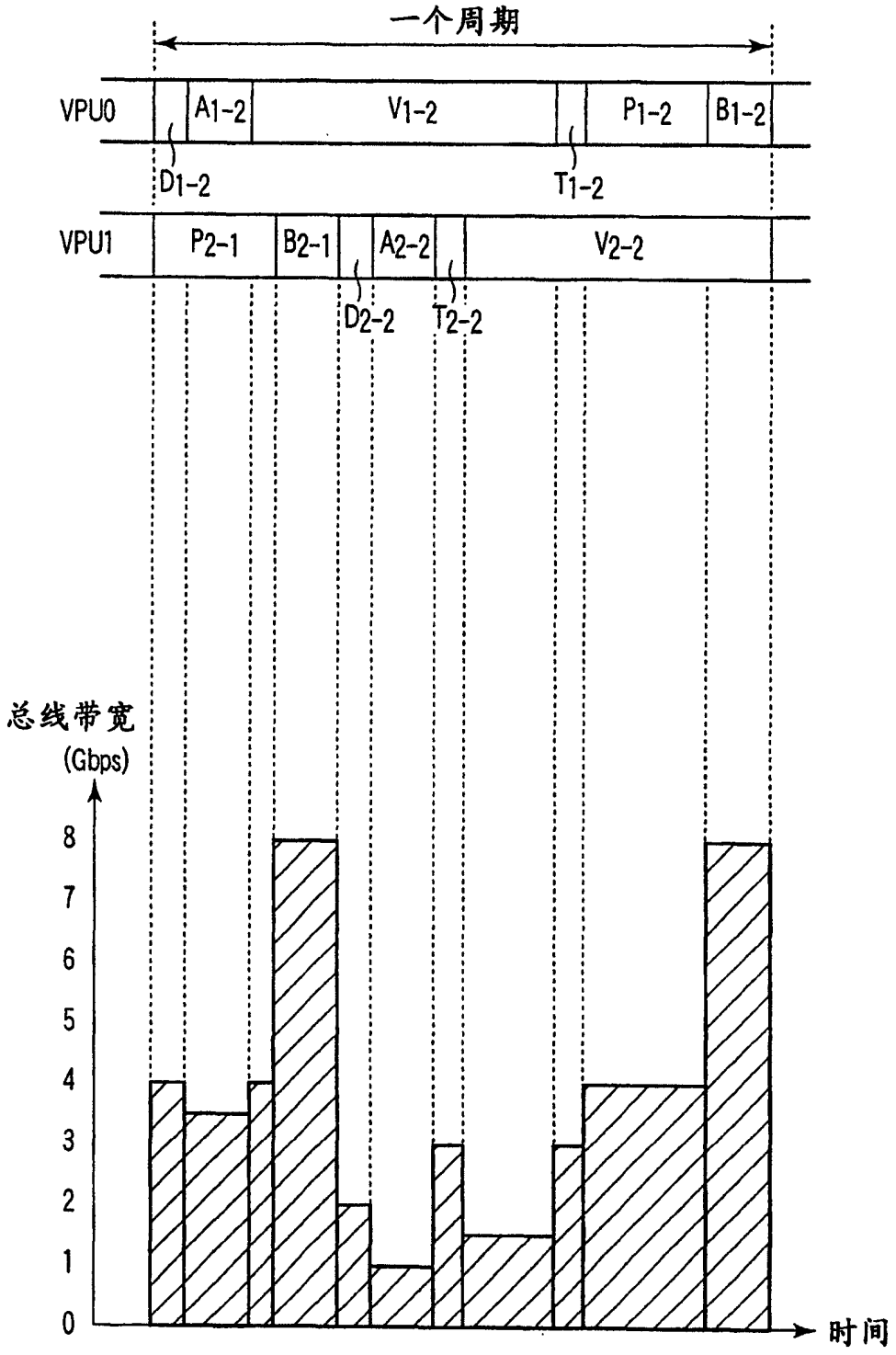

一个周期
VPU0
A1-2
V1-2
P1-2
B1-2
D1-2
T1-2
VPU1
P2-1
B2-1
A2-2
V2-2
D2-2
T2-2
总线带宽
(Gbps)
8
7
6
5
4
3
2
1
0
时间

图62

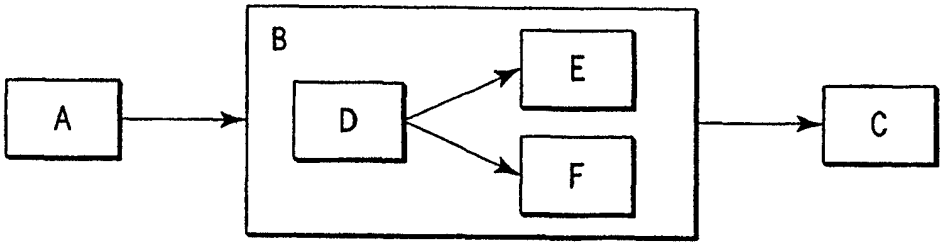

A
B
D
E
F
C

图63

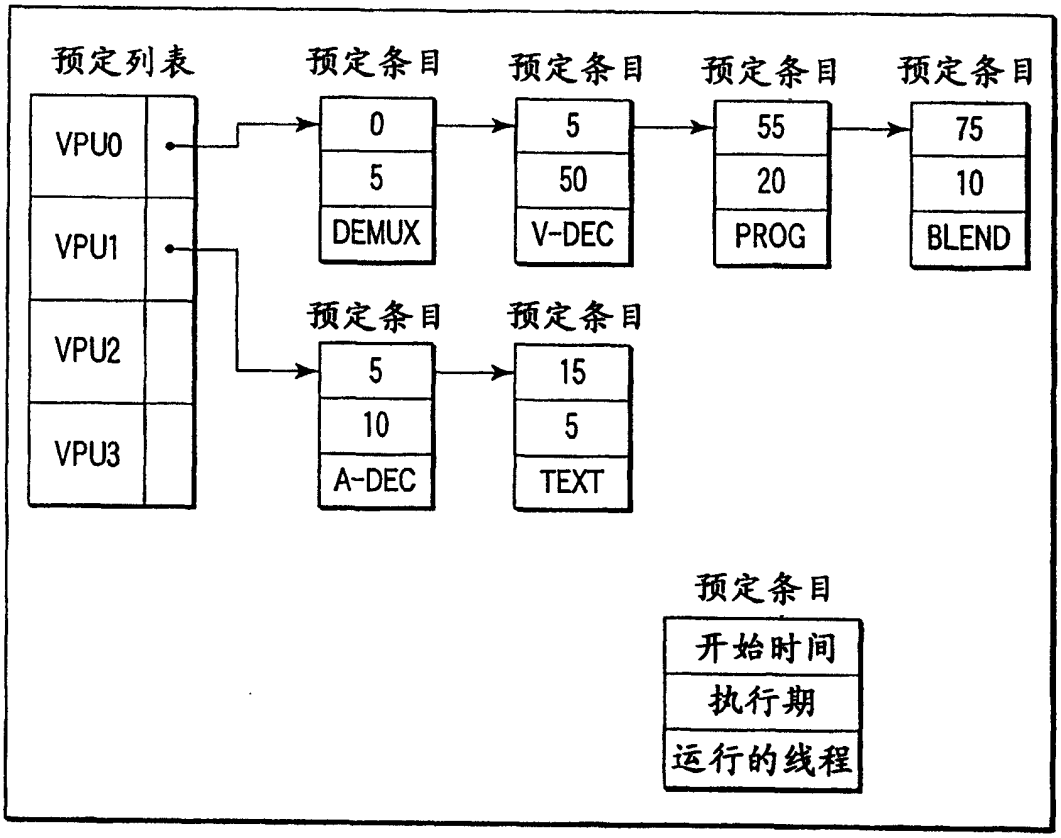

预定列表
VPU0
VPU1
VPU2
VPU3
预定条目
0
5
DEMUX
预定条目
5
50
V-DEC
预定条目
55
20
PROG
预定条目
75
10
BLEND
预定条目
5
10
A-DEC
预定条目
15
5
TEXT
预定条目
开始时间
执行期
运行的线程

图64

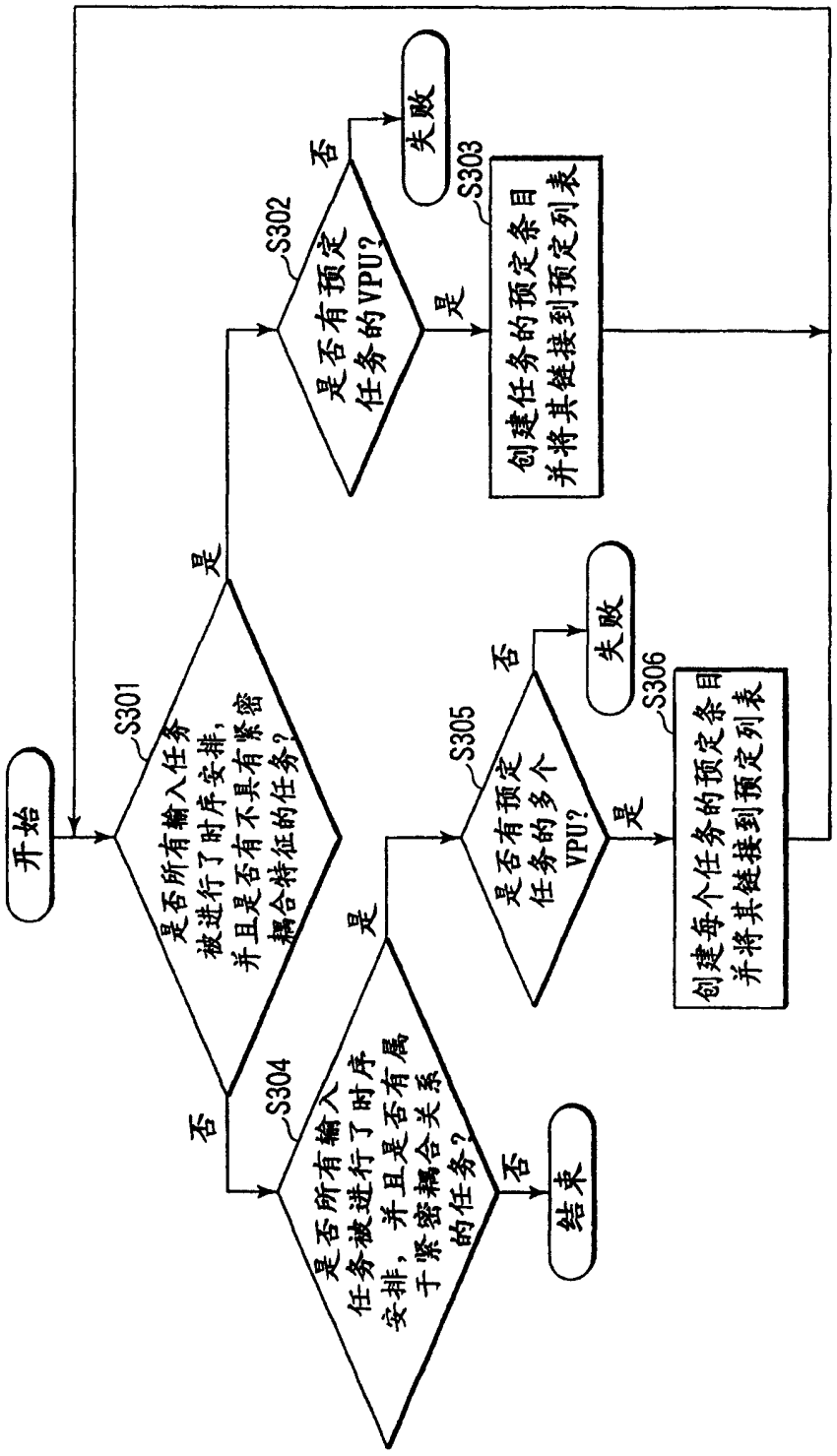

开始
S301
是否所有输入任务被进行了时序安排，并且是否有不具有紧密耦合特征的任务?
是
否
S302
是否有预定任务的VPU?
否
失败
是
S303
创建任务的预定条目并将其链接到预定列表
S304
是否所有输入任务被进行了时序安排，并且是否有属于紧密耦合关系的任务?
是
否
结束
S305
是否有预定任务的多个VPU?
否
失败
是
S306
创建每个任务的预定条目并将其链接到预定列表

图 65