[19] 中华人民共和国国家知识产权局

[12] 发明专利说明书

专利号 ZL 02806348.1

[51] Int. Cl.
*G06F 12/14 (2006.01)*
*G06F 15/16 (2006.01)*
*G06F 15/163 (2006.01)*
*G06F 15/167 (2006.01)*
*G06F 15/17 (2006.01)*
*G06F 15/173 (2006.01)*

[45] 授权公告日 2006 年 12 月 20 日

[11] 授权公告号 CN 1291327C

[51] Int. Cl. (续)
*G06F 15/177 (2006.01)*

[22] 申请日 2002.3.19 [21] 申请号 02806348.1

[30] 优先权

[32] 2001.3.22 [33] US [31] 09/816,020

[86] 国际申请 PCT/JP2002/002606 2002.3.19

[87] 国际公布 WO2002/077826 英 2002.10.3

[85] 进入国家阶段日期 2003.9.11

[73] 专利权人 索尼计算机娱乐公司

地址 日本东京

[72] 发明人 铃置雅一 山崎刚

审查员 丁文勋

[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

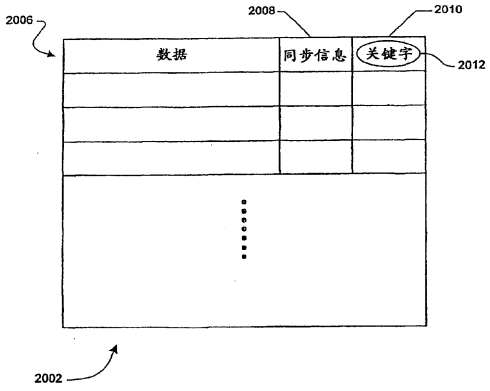
权利要求书 9 页 说明书 31 页 附图 47 页

[54] 发明名称

# 计算机处理系统和计算机处理方法

[57] 摘要

提供一种用于宽带网络上的高速处理的计算机体系结构和编程模型。该体系结构使用一致的模块化结构，公用计算模块和统一的软件单元。公用计算模块包括控制处理器，多个处理单元，由处理单元处理其程序的多个局部存储器，直接存储器存取控制器和共享主存储器。还提供了一种用于协调处理单元从共享主存储器读数据和向其写数据的同步系统和方法。提供了硬件沙箱结构，用于确保处理单元正在处理的程序中间的数据不会被破坏。



1.一种计算机处理系统，所述处理系统包括：

第一存储器区，用于存储程序和与所述程序相关的数据；

多个第一处理单元，用于处理所述程序和与所述程序相关的所述数据；

存储器控制器，用于控制所述第一处理单元对所述第一存储器区的访问；

第二存储器区，用于存储存取表，所述存取表包括多个存取条目，每个所述存取条目包含存取关键字，和所述第一存储器区内与所述存取关键字相关的存储器空间的标识，

第三存储器区，用于存储关键字表，所述关键字表包括多个关键字条目，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字；

第二处理单元，用于控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理，所述第二处理单元可以基于该与所述程序相关的数据构造和维护所述存取表和所述关键字表，所述第二处理单元还可以指示一个所述第一处理单元处理一个所述程序，

该一个所述第一处理单元在处理该一个所述程序时可以向所述存储器控制器发出请求以便访问所述第一存储器区内的存储位置，

所述存储器控制器可以在存储位置对应于和所述存取表中标识的一个所述存取关键字相关的存储器空间的情况下，响应所述请求而基于所述关键字表中与该一个所述第一处理单元相关的请求关键字和所述存取表中的该一个所述存取关键字之间的比较的结果，来执行所述请求。

2.如权利要求1所述的计算机处理系统，还包括多个局部存储器，每个所述局部存储器与所述第一处理单元中的相应一个相关，其中通过指示所述存储器控制器从所述第一存储器区向与该一个所述第一处理单元相关的局部存储器传送该一个所述程序，所述第二处理单

元可以指示该一个所述第一处理单元处理该一个所述程序，该一个所述第一处理单元此后处理来自所述局部存储器的该一个所述程序。

3.如权利要求1所述的计算机处理系统，其中该一个所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，所述存储器控制器可以仅在所有所述多个第一位与所有所述多个第二位匹配时执行所述请求。

4.如权利要求1所述的计算机处理系统，其中该一个所述存取关键字包括多个第一位，和关键字掩码，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，所述存储器控制器可以仅在所有所述多个第一位与所述多个第二位匹配，或者所述多个第一位中不与所述多个第二位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

5.如权利要求1所述的计算机处理系统，其中该一个所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，和关键字掩码，所述存储器控制器可以仅在所有所述多个第二位与所述多个第一位匹配，或者所述多个第二位中不与所述多个第一位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

6.如权利要求1所述的计算机处理系统，其中每个所述关键字条目还包含关键字掩码。

7.如权利要求1所述的计算机处理系统，其中每个所述存取条目还包含关键字掩码。

8.如权利要求1所述的计算机处理系统，其中每个所述关键字条目还包含关键字掩码，而每个所述存取条目还包含关键字掩码。

9.如权利要求1所述的计算机处理系统，其中每个所述存取条目还包含基位置条目和尺寸条目，所述基位置条目提供所述存储器空间中与所述存取条目相关的所述第一存储器区内的起始地址，而所述尺寸条目提供所述存储器空间中与所述存取条目相关的所述第一存储器区内的尺寸。

10.如权利要求1所述的计算机处理系统，其中所述第一存储器区在动态随机存取存储器中，并且在所述计算机处理系统的主存储器中。

11.一种计算机处理方法，所述方法包括：

在第一存储器区中存储程序和与所述程序相关的数据；

通过多个第一处理单元处理所述程序和与所述程序相关的所述数据；

通过存储器控制器控制所述第一处理单元对所述第一存储器区的访问；

通过第二处理单元在第二存储器区中基于该与所述程序相关的数据构造存取表和在第三存储器区中基于该与所述程序相关的数据构造关键字表，所述存取表包括多个存取条目，每个所述存取条目包含存取关键字，和所述第一存储器区内与所述存取关键字相关的存储器空间的标识，所述关键字表包括多个关键字条目，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字；

通过所述第二处理单元控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理；

通过所述第二处理单元指示一个所述第一处理单元处理一个所述程序；

在处理该一个所述程序时从该一个所述第一处理单元发出请求到所述存储器控制器，以便访问所述第一存储器区内的存储位置；

响应所述请求，将所述关键字表中与该一个所述第一处理单元相关的请求关键字与所述存取表中的存取关键字相比较；

如果所述存储位置对应于与一个所述存取关键字相关的存储器空间，则基于所述关键字表中与该一个所述处理单元相关的请求关键字和所述存取表中的该一个所述存取关键字之间的比较的结果，来执行所述请求。

12.如权利要求11所述的计算机处理方法，还包括通过从所述第

二处理单元向所述存储器控制器发出命令以便从所述第一存储器区向与该一个所述第一处理单元相关的局部存储器传送该一个所述程序，通过所述第二处理单元指示该一个所述第一处理单元处理该一个所述程序，并且此后通过该一个所述第一处理单元处理来自所述局部存储器的该一个所述程序。

13.如权利要求11所述的计算机处理方法，其中该一个所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，还包括通过所述存储器控制器仅在所有所述多个第一位与所有所述多个第二位匹配时执行所述请求。

14.如权利要求11所述的计算机处理方法，其中该一个所述存取关键字包括多个第一位，和关键字掩码，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，还包括通过所述存储器控制器仅在所有所述多个第一位与所述多个第二位匹配，或者所述多个第一位中不与所述多个第二位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

15.如权利要求11所述的计算机处理方法，其中该一个所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，和关键字掩码，还包括通过所述存储器控制器仅在所有所述多个第二位与所述多个第一位匹配，或者所述多个第二位中不与所述多个第一位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

16.如权利要求11所述的计算机处理方法，其中每个所述关键字条目还包含关键字掩码。

17.如权利要求11所述的计算机处理方法，其中每个所述存取条目还包含关键字掩码。

18.如权利要求11所述的计算机处理方法，其中每个所述关键字条目还包含关键字掩码，而每个所述存取条目还包含关键字掩码。

19.如权利要求11所述的计算机处理方法，其中每个所述存取条目还包含基位置条目和尺寸条目，所述基位置条目提供所述存储器空

间中与所述存取条目相关的所述第一存储器区内的起始地址，而所述尺寸条目提供所述存储器空间中与所述存取条目相关的所述第一存储器区内的尺寸。

20.如权利要求11所述的计算机处理方法，其中所述第一存储器区在动态随机存取存储器中，并且在所述计算机的主存储器中。

21.一种计算机处理系统，所述处理系统包括:

第一存储器区，用于存储程序和与所述程序相关的数据，所述第一存储器区包括多个可寻址存储位置，每个所述可寻址存储位置包括与所述可寻址存储位置相关的附加存储段，并且包含针对所述可寻址存储位置的存取关键字;

多个第一处理单元，用于处理所述程序和与所述程序相关的所述数据;

存储器控制器，用于控制所述第一处理单元对所述第一存储器区的访问;

第二存储器区，用于存储包括多个关键字条目的关键字表，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字;

第二处理单元，用于控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理，所述第二处理单元可以基于该与所述程序相关的数据分配和维护所述存取关键字，并且基于该与所述程序相关的数据构造和维护所述关键字表，所述第二处理单元还可以指示一个所述第一处理单元处理一个所述程序，

该一个所述第一处理单元在处理该一个所述程序时可以向所述存储器控制器发出请求以便访问一个所述可寻址存储位置，

所述存储器控制器可以响应所述请求，基于所述关键字表中与该一个所述第一处理单元相关的请求关键字和附加存储段中包含的、和该一个所述可寻址存储位置相关的存取关键字之间的比较的结果，来执行所述请求。

22.如权利要求21所述的计算机处理系统，还包括多个局部存储

器，每个所述局部存储器与所述第一处理单元中的相应一个相关，其中通过指示所述存储器控制器从所述第一存储器区向与该一个所述第一处理单元相关的局部存储器传送该一个所述程序，所述第二处理单元可以指示该一个所述第一处理单元处理该一个所述程序，该一个所述第一处理单元此后处理来自所述局部存储器的该一个所述程序。

23.如权利要求21所述的计算机处理系统，其中所述存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，所述存储器控制器可以仅在所有所述多个第一位与所有所述多个第二位匹配时执行所述请求。

24.如权利要求21所述的计算机处理系统，其中所述存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字包括多个第一位，和关键字掩码，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，所述存储器控制器可以仅在所有所述多个第一位与所述多个第二位匹配，或者所述多个第一位中不与所述多个第二位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

25.如权利要求21所述的计算机处理系统，其中所述存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，和关键字掩码，所述存储器控制器可以仅在所有所述多个第二位与所述多个第一位匹配，或者所述多个第二位中不与所述多个第一位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

26.如权利要求21所述的计算机处理系统，其中每个所述关键字条目还包含关键字掩码。

27.如权利要求21所述的计算机处理系统，其中每个所述附加存储段还包含关键字掩码。

28.如权利要求21所述的计算机处理系统，其中每个所述关键字条目还包含关键字掩码，而每个所述附加存储段还包含关键字掩码。

29.如权利要求21所述的计算机处理系统，其中每个所述附加存

储段还包含状态信息，所述状态信息有关与所述附加存储段相关的可寻址存储位置中存储的数据的状态。

30.如权利要求21所述的计算机处理系统，其中所述第一存储器区在动态随机存取存储器中，并且在所述计算机处理系统的主存储器中。

31.一种计算机处理方法，所述方法包括:

在第一存储器区中存储程序和与所述程序相关的数据，所述第一存储器区包含多个可寻址存储位置，每个所述可寻址存储位置包含与所述可寻址存储位置相关的附加存储段;

在每个所述可寻址存储位置的每个所述附加存储段中存储针对所述可寻址存储位置的存取关键字;

通过多个第一处理单元处理所述程序和与所述程序相关的所述数据;

通过存储器控制器控制所述第一处理单元对所述第一存储器区的访问;

在第二存储器区中存储包括多个关键字条目的关键字表，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字;

通过所述第二处理单元控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理;

通过所述第二处理单元基于该与所述程序相关的数据分配每个所述存取关键字;

通过所述第二处理单元基于该与所述程序相关的数据构造所述关键字表;

通过所述第二处理单元指示一个所述第一处理单元处理一个所述程序;

在处理该一个所述程序时从该一个所述第一处理单元发出请求到所述存储器控制器，以便访问一个所述可寻址存储位置;

响应所述请求，将所述关键字表中与该一个所述第一处理单元

相关的请求关键字，与附加存储段中包含的、与该一个所述可寻址存储位置相关的存取关键字相比较；

基于所述关键字表中与该一个所述第一处理单元相关的请求关键字和所述附加存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字之间的比较的结果，执行所述请求。

32.如权利要求31所述的计算机处理方法，还包括通过指示所述存储器控制器从所述第一存储器区向与该一个所述第一处理单元相关的局部存储器传送该一个所述程序，通过所述第二处理单元指示该一个所述第一处理单元处理该一个所述程序，并且此后通过该一个所述第一处理单元处理来自所述局部存储器的该一个所述程序。

33.如权利要求31所述的计算机处理方法，其中所述附加存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，还包括通过所述存储器控制器仅在所有所述多个第一位与所有所述多个第二位匹配时执行所述请求。

34.如权利要求31所述的计算机处理方法，其中所述附加存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字包括多个第一位，和关键字掩码，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，还包括通过所述存储器控制器仅在所有所述多个第一位与所述多个第二位匹配，或者所述多个第一位中不与所述多个第二位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

35.如权利要求31所述的计算机处理方法，其中所述附加存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字包括多个第一位，并且与该一个所述第一处理单元相关的所述请求关键字包括多个第二位，和关键字掩码，还包括通过所述存储器控制器仅在所有所述多个第二位与所述多个第一位匹配，或者所述多个第二位中不与所述多个第一位匹配的所有位被所述关键字掩码屏蔽时，执行所述请求。

36.如权利要求31所述的计算机处理方法，其中每个所述关键字条目还包含关键字掩码。

37.如权利要求31所述的计算机处理方法，其中每个所述附加存储段还包含关键字掩码。

38.如权利要求31所述的计算机处理方法，其中每个所述关键字条目还包含关键字掩码，而每个所述附加存储段还包含关键字掩码。

39.如权利要求31所述的计算机处理方法，其中每个所述附加存储段还包含状态信息，所述状态信息有关与所述附加存储段相关的可寻址存储位置中存储的数据的状态。

40.如权利要求31所述的计算机处理方法，其中所述第一存储器区在动态随机存取存储器中，并且在所述计算机的主存储器中。

# 计算机处理系统和计算机处理方法

## 技术领域

本发明涉及计算机处理器和计算机网络的体系结构，特别涉及宽带环境中计算机处理器和计算机网络的体系结构。本发明还涉及这种体系结构的编程模型。

## 背景技术

对于当前计算机网络，例如办公室网络中使用的局域网(LAN)，以及象 Internet 等这样的全球网络，其计算机和计算设备主要是为单机计算设计的。通过计算机网络共享数据和应用程序不是这些计算机和计算设备的主要设计目的。通常还使用各种类型的、由不同生产商生产的处理器来设计这些计算机和计算设备，这些生产商例如是摩托罗拉、英特尔、德州仪器，索尼以及其它一些公司。这些处理器中的每一个都具有其独特的指令集和指令集体系结构(ISA)，即，具有其独特的汇编语言指令集，和用于执行这些指令的基本计算单元和存储单元的结构。因此，程序员需要理解每个处理器的指令集和 ISA，以便写出针对这些处理器的应用程序。当前计算机网络上计算机和计算设备的这种异构组合使得数据及应用程序的处理和共享复杂化。而且，经常需要同一应用程序的多个版本，以适应这种异构环境。

连接到全球网络，尤其是连接到 Internet 的计算机和计算设备的种类是非常广泛的。除个人计算机(PC)和服务器外，这些计算设备还包括蜂窝电话、移动计算机、个人数字助理(PDA)、机顶盒、数字电视以及许多其它设备。在如此多样的计算机和计算设备之间共享数据和应用程序带来了相当大的问题。

为克服这些问题，已使用了很多技术。尤其是，这些技术包括高级的接口和复杂的编程技术。这些解决方案的实现通常需要处理能力有相当大的增加。它们还经常大大增加在网络上处理应用程序和传输数据所需的时间。

通常，通过 Internet 从相应应用程序分别发送数据。这个解决

方案不必发送应用程序和与该应用程序对应的每组发送数据。虽然该方案使得带宽需求达到最小，但它还是经常引起用户间的干扰(frustration)。可能在客户端计算机上得不到用于所发送数据的正确应用程序或者最新应用程序。该方案还需要针对网络上处理器的不同ISA和指令集的多样性编写每个应用程序的多个版本。

Java模型试图解决这个问题。该模型使用服从严格安全协议的小应用程序(“applet”)。applet由服务器通过网络发送，以便在客户端计算机(“client”)上运行。为避免必须向使用不同ISA的客户端发送相同applet的不同版本，所有applet均在客户端的Java虚拟机上运行。Java虚拟机是模拟具有Java ISA和Java指令集的计算机的软件。然而，该软件运行在客户端的ISA和客户端的指令集上。为客户端的每个不同ISA和指令集提供一个版本的Java虚拟机。这样，就不需要每个applet的不同版本的多样性。每个客户端仅下载针对其特定ISA和指令集的正确Java虚拟机，以运行所有Java applet。

虽然对必须为每个不同ISA和指令集编写应用程序的不同版本的问题提供了解决方案，然而Java处理模型在客户端计算机上仍需要一个附加软件层。这个附加软件层极大地降低了处理器的处理速度。这种速度降低对于实时、多媒体应用尤其明显。下载的Java applet还可能含有病毒、处理故障等。这些病毒和故障会破坏客户端的数据库，并造成其它损坏。虽然Java模型中使用的安全协议试图通过实现软件“沙箱（sandbox）”来解决这个问题(“沙箱”是客户端存储器中的一个空间，如果超出此空间，则Java applet不能写数据)，但是这个软件驱动的安全模型在其执行过程中经常是不安全的，并且需要更多的处理。

实时、多媒体、网络应用变得越来越重要。这些网络应用需要极快的处理速度。将来，这类应用每秒钟可能需要若干千兆位的数据。目前的网络体系结构，特别是Internet的体系结构，以及当前在例如Java模型中实现的编程模型要达到这样的处理速度是极其困难的。

因此，需要一种新的计算机体系结构、一种新的计算机网络体系结构和一种新的编程模型。这种新的体系结构和编程模型应当解决

在各种网络成员中间共享数据和应用程序的问题，而无须增加额外的计算负担。这种新的计算机体系结构和编程模型还应当解决在网络成员中间共享应用程序和数据所固有的安全问题。

发明内容

一方面，本发明提供一种用于计算机、计算设备和计算机网络的新型体系结构。另一方面，本发明提供一种用于这些计算机、计算设备和计算机网络的新型编程模型。

根据本发明，计算机网络的所有成员，即网络中所有计算机和计算设备，都用公用计算模块构建。这种公用计算模块具有一致的结构，并且最好使用相同的 ISA。网络的成员可以是例如客户机、服务器、PC、移动计算机、游戏机、PDA、机顶盒、电器、数字电视、以及其它使用计算机处理器的设备。一致的模块化结构允许网络成员有效、高速地处理应用程序和数据，并允许在网络上快速传输应用程序和数据。这种结构还简化了具有各种规模和处理能力的网络成员的构建，以及这些成员处理的应用程序的准备。另外，根据本发明的一个实施例，提供了一个计算机网络，该计算机网络包含连接到所述网络的多个处理器，每个所述处理器包含具有相同指令集体系结构的多个第一处理单元，以及用于控制所述第一处理单元的第二处理单元，所述第一处理单元可处理通过所述网络传输的软件单元，每个所述软件单元包含与所述指令集体系结构兼容的程序，与所述程序相关的数据，以及在通过所述网络传输的全部所述软件单元中唯一标识所述软件单元的标识符。标识符最好是在通过所述网络传输的全部所述软件单元中唯一标识所述软件单元的标识号。

另一方面，本发明提供了一种用于通过网络传输数据和应用程序，以及用于在网络成员中间处理数据和应用程序的新型编程模型。该编程模型使用通过网络传输的、可由任何网络成员处理的软件单元。每个软件单元均具有相同的结构，并可包含应用程序和数据。由于模块化计算机体系结构提供的高速处理和传输速度，这些单元可以被快速处理。应用程序代码最好基于相同的公用指令集和 ISA。每个软件单元最好包含全局标识(global ID)，和说明该单元的处理所需计算资源数量的信息。由于所有计算资源具有相同的基本结构，并使用相同的 ISA，所以执行这个处理的特定资源可以位于网络上的任何地

方，并且可以动态分配。

基本的处理模块是处理器元件(PE)。PE 最好包括处理单元(PU)，直接存储器存取控制器(DMAC)，以及多个附连处理单元(APU)。在优选实施例中，PE 包含八个 APU。PU 和 APU 与最好具有交叉(cross-bar)体系结构的动态随机存取存储器(DRAM)交互。PU 计划和编排 APU 对数据和应用程序的处理。APU 以并行和独立的方式执行这种处理。DMAC 控制 PU 和 APU 对共享 DRAM 中存储的数据和应用程序的存取。

根据这种模块化的结构，网络成员使用的 PE 的数目取决于该成员所需要的处理能力。例如，服务器可能使用四个 PE，工作站可能使用两个 PE，PDA 可能使用一个 PE。PE 中被分配用来处理特定软件单元的 APU 的数目取决于单元中的程序和数据的复杂度和量级。

在优选实施例中，多个 PE 与共享 DRAM 相关。最好将 DRAM 分隔成多个区（section），并将这些区中的每一个分隔成多个存储体（memory bank）。在一个特别的优选实施例中，DRAM 包含 64 个存储体，并且每个存储体具有一兆字节的存储容量。DRAM 的每个区最好由存储体控制器控制，并且 PE 的每个 DMAC 最好访问每个存储体控制器。因此，该实施例中每个 PE 的 DMAC 可访问共享 DRAM 的任何部分。

另一方面，本发明提供使 APU 从共享 DRAM 读取数据，以及向其写入数据的同步系统和方法。该系统避免了共享 DRAM 的多个 APU 和多个 PE 中间的冲突。根据该系统和方法，将 DRAM 的一个区域指定为用于存储多个全空位。这些全空位中的每一个对应于 DRAM 的一个指定区域。将同步系统集成到 DRAM 的硬件中，因此避免了用软件实现的数据同步模式的计算开销。

本发明还在 DRAM 中实现沙箱以提供安全性，从而防止一个 APU 处理的程序的数据被另一个 APU 处理的程序的数据破坏。每个沙箱定义了共享 DRAM 中的一个区域，如果超出此区域，特定 APU 或 APU 组不能读数据或写数据。

另一方面，本发明提供用于 PU 向 APU 发出命令以启动 APU 对应用程序和数据的处理的系统和方法。这些命令被称为 APU 远程

过程调用(ARPC)，使 PU 能够编排和协调 APU 对应用程序和数据的并行处理，而无须 APU 充当协处理器的角色。

在另一方面，本发明提供用于建立专用流水线结构以处理流数据的系统和方法。根据该系统和方法，PU 建立协同的 APU 组，以及与这些 APU 相关的协同存储器沙箱组，以用于这些数据的处理。在不处理数据的时候，流水线的专用 APU 和存储器沙箱仍保留为流水线专用。换句话说，在这些期间中，将专用 APU 及其相关沙箱置于保留状态。

在另一方面，本发明提供用于处理任务的绝对定时器。该绝对定时器独立于处理应用程序和数据的 APU 所使用的时钟频率。根据绝对定时器定义的任务时间周期来写应用程序。如果因 APU 的增强(例如)而使 APU 使用的时钟频率提高，由绝对定时器定义的给定任务的时间周期保持不变。该方案允许以更新版本的 APU 实现增强的处理时间，而无须禁止这些新 APU 处理针对较早 APU 的较慢处理时间而编写的较早应用程序。

本发明还提供使具有更快处理速度的更新 APU 能够处理针对较早 APU 的较慢处理速度而编写的较早应用程序的一种可选方案。在这个可选方案中，针对因速度增强而在协调 APU 的并行处理方面所产生的问题，在处理期间对 APU 在处理这些较早应用程序时所使用的特定指令或微代码进行分析。在由这些 APU 中的某一些执行的指令中插入“无操作”（“NOOP”）指令，以使 APU 的处理按程序预期的顺序完成。通过在这些指令中插入这些 NOOP，保持了 APU 执行所有指令的正确定时。

在另一方面，本发明提供包含集成了光波导的集成电路的芯片封装（chip package）。

本发明提供了一种计算机处理系统，所述处理系统包括：第一存储器区，用于存储程序和与所述程序相关的数据；多个第一处理单元，用于处理所述程序和与所述程序相关的所述数据；存储器控制器，用于控制所述第一处理单元对所述第一存储器区的访问；第二存储器区，用于存储存取表，所述存取表包括多个存取条目，每个所述存取条目包含存取关键字，和所述第一存储器区内与所述存取关键字相关的存储器空间的标识，第三存储器区，用于存储关键字表，所述

关键字表包括多个关键字条目，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字；第二处理单元，用于控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理，所述第二处理单元可以基于该与所述程序相关的数据构造和维护所述存取表和所述关键字表，所述第二处理单元还可以指示一个所述第一处理单元处理一个所述程序，该一个所述第一处理单元在处理该一个所述程序时可以向所述存储器控制器发出请求以便访问所述第一存储器区内的存储位置，所述存储器控制器可以在存储位置对应于和所述存取表中标识的一个所述存取关键字相关的存储器空间的情况下，响应所述请求而基于所述关键字表中与该一个所述第一处理单元相关的请求关键字和所述存取表中的该一个所述存取关键字之间的比较的结果，来执行所述请求。

本发明提供了一种计算机处理方法，所述方法包括：在第一存储器区中存储程序和与所述程序相关的数据；通过多个第一处理单元处理所述程序和与所述程序相关的所述数据；通过存储器控制器控制所述第一处理单元对所述第一存储器区的访问；通过第二处理单元在第二存储器区中基于该与所述程序相关的数据构造存取表和在第三存储器区中基于该与所述程序相关的数据构造关键字表，所述存取表包括多个存取条目，每个所述存取条目包含存取关键字，和所述第一存储器区内与所述存取关键字相关的存储器空间的标识，所述关键字表包括多个关键字条目，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字；通过所述第二处理单元控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理；通过所述第二处理单元指示一个所述第一处理单元处理一个所述程序；在处理该一个所述程序时从该一个所述第一处理单元发出请求到所述存储器控制器，以便访问所述第一存储器区内的存储位置；响应所述请求，将所述关键字表中与该一个所述第一处理单元相关的请求关键字与所述存取表中的存取关键字相比较；如果所述存储位置对应于与一个所述存取关键字相关的存储器空间，则基于所述关键字表中与该一个所述处理单元相关的请求关键字和所述存取表中的该一个所述存取关键字之间的比较的结果，来执行所述请求。

本发明提供了一种计算机处理系统，所述处理系统包括：第一存储器区，用于存储程序和与所述程序相关的数据，所述第一存储器区包括多个可寻址存储位置，每个所述可寻址存储位置包括与所述可寻址存储位置相关的附加存储段，并且包含针对所述可寻址存储位置的存取关键字；多个第一处理单元，用于处理所述程序和与所述程序相关的所述数据；存储器控制器，用于控制所述第一处理单元对所述第一存储器区的访问；第二存储器区，用于存储包括多个关键字条目的关键字表，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字；第二处理单元，用于控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理，所述第二处理单元可以基于该与所述程序相关的数据分配和维护所述存取关键字，并且基于该与所述程序相关的数据构造和维护所述关键字表，所述第二处理单元还可以指示一个所述第一处理单元处理一个所述程序，该一个所述第一处理单元在处理该一个所述程序时可以向所述存储器控制器发出请求以便访问一个所述可寻址存储位置，所述存储器控制器可以响应所述请求，基于所述关键字表中与该一个所述第一处理单元相关的请求关键字和附加存储段中包含的、和该一个所述可寻址存储位置相关的存取关键字之间的比较的结果，来执行所述请求。

本发明提供了一种计算机处理方法，所述方法包括：在第一存储器区中存储程序和与所述程序相关的数据，所述第一存储器区包含多个可寻址存储位置，每个所述可寻址存储位置包含与所述可寻址存储位置相关的附加存储段；在每个所述可寻址存储位置的每个所述附加存储段中存储针对所述可寻址存储位置的存取关键字；通过多个第一处理单元处理所述程序和与所述程序相关的所述数据；通过存储器控制器控制所述第一处理单元对所述第一存储器区的访问；在第二存储器区中存储包括多个关键字条目的关键字表，每个所述关键字条目包含所述第一处理单元之一的标识，和与该所述第一处理单元之一相关的请求关键字；通过所述第二处理单元控制所述第一处理单元对所述程序和与所述程序相关的所述数据的所述处理；通过所述第二处理单元基于该与所述程序相关的数据分配每个所述存取关键字；通过所述第二处理单元基于该与所述程序相关的数据构造所述关键字表；通

过所述第二处理单元指示一个所述第一处理单元处理一个所述程序；在处理该一个所述程序时从该一个所述第一处理单元发出请求到所述存储器控制器，以便访问一个所述可寻址存储位置；响应所述请求，将所述关键字表中与该一个所述第一处理单元相关的请求关键字，与附加存储段中包含的、与该一个所述可寻址存储位置相关的存取关键字相比较；基于所述关键字表中与该一个所述第一处理单元相关的请求关键字和所述附加存储段中包含的、与该一个所述可寻址存储位置相关的所述存取关键字之间的比较的结果，执行所述请求。

附图说明

图 1 说明了本发明的计算机网络的整体体系结构。

图 2 说明了本发明的处理器元件(PE)的结构。

图 3 说明了本发明的宽带引擎(BE)的结构。

图 4 说明了本发明的附连处理单元(APU)的结构。

图 5 说明了本发明的处理器元件，观察器(VS，visualizer)和光接口的结构。

图 6 说明了本发明的处理器元件的一种组合。

图 7 说明了本发明的处理器元件的另外一种组合。

图 8 说明了本发明的处理器元件的另外一种组合。

图 9 说明了本发明的处理器元件的另外一种组合。

图 10 说明了本发明的处理器元件的另外一种组合。

图 11A 说明了本发明的芯片封装中光接口的集成。

图 11B 是使用图 11A 的光接口的处理器的一种结构的图。

图 11C 是使用图 11A 的光接口的处理器的另外一种结构的图。

图 12A 说明了本发明的存储器系统的结构。

图 12B 根据本发明说明了从第一宽带引擎向第二宽带引擎的数据写入。

图 13 是本发明的处理器元件的共享存储器的结构图。

图 14A 说明了图 13 中的存储器的存储体的一种结构。

图 14B 说明了图 13 中的存储器的存储体的另一种结构。

图 15 说明了本发明的直接存储器存取控制器的结构。

图 16 说明了本发明的直接存储器存取控制器的另一种结构。

图 17A-17O 说明了本发明的数据同步操作。

图 18 是根据本发明的数据同步模式说明存储器位置的不同状态的三状态存储器图。

图 19 说明了本发明的硬件沙箱的关键字控制表的结构。

图 20 说明了用于存储本发明的硬件沙箱的存储器存取关键字的方案。

图 21 说明了本发明的硬件沙箱的存储器存取控制表的结构。

图 22 是使用图 19 的关键字控制表和图 21 的存储器存取控制表访问存储器沙箱的步骤的流程图。

图 23 说明了本发明的软件单元的结构。

图 24 是根据本发明向 APU 发出远程过程调用的步骤的流程图。

图 25 说明了本发明用于处理流数据的专用流水线的结构。

图 26A-26B 是本发明如图 25 所示的专用流水线在处理流数据时执行的步骤的流程图。

图 27 说明了本发明用于处理流数据的专用流水线的可选结构。

图 28 说明了本发明用于协调 APU 对应用程序和数据的并行处理的绝对定时器的方案。

具体实施方式

图 1 示出了本发明的计算系统 101 的整体体系结构。

如图中所示，系统 101 包括网络 104，其上连接了多个计算机和计算设备。网络 104 可以是 LAN、像 Internet 这样的全球网、或者任何其它的计算机网络。

连接到网络 104 的计算机和计算设备(网络“成员”)包括例如客户端计算机 106、服务器计算机 108、个人数字助理(PDA) 110、数字电视(DTV)112、以及其它有线或无线计算机和计算设备。网络 104 的成员使用的处理器由相同的公用计算模块构建。这些处理器最好还都具有相同的 ISA，并按照相同的指令集执行处理。任何一个特定的处理器中包含的模块的数目取决于该处理器所需的处理能力。

例如，由于系统 101 中的服务器 108 比客户端 106 执行更多的

数据和应用程序处理，所以服务器 108 比客户端 106 包含更多的计算模块。另一方面，PDA 110 执行的处理量最少，因此 PDA 110 包含的计算模块最少。DTV 112 执行的处理量介于客户端 106 的和服务器 108 之间，因此，它包含的计算模块的数目介于客户端 106 和服务器 108 之间。如下面讨论的，每个计算模块包含一个处理控制器，和多个同样的处理单元，用于对网络 104 上传输的数据和应用程序执行并行处理。

系统 101 的这种同构结构利于提高适应性、处理速度和处理效率。因为系统 101 的每个成员使用一或多个(或某些部分的)相同计算模块来执行处理，所以执行数据和应用程序的实际处理的特定计算机或计算设备是不重要的。而且，特定应用程序和数据的处理可以在网络成员间分享。通过在整个系统中唯一标识包含由系统 101 处理的数据和应用程序的单元，则无论处理在哪里进行，都可以将该处理结果传输到请求处理的计算机或计算设备。因为执行该处理的模块具有公用结构并使用公用 ISA，所以避免了为实现处理器之间的兼容性而附加的软件层的计算负担。该体系结构和编程模型利于得到执行例如实时、多媒体应用程序所必需的处理速度。

为了进一步利用系统 101 带来的处理速度和效率，将该系统处理的数据和应用程序封装到具有唯一标识和统一格式的软件单元 102 中。每个软件单元 102 包含或可以包含应用程序和数据。每个软件单元还包含在整个网络 104 和系统 101 中全局性地标识该单元的 ID。软件单元结构的这种一致性以及软件单元在整个网络中的唯一标识，有利于在网络中的任何一个计算机或计算设备上处理应用程序和数据。例如，客户端 106 可以形成软件单元 102，但是因为客户端 106 的处理能力有限，可以将该软件单元发送到服务器 108 来进行处理。因此，软件单元可在整个网络 104 中迁移，以便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还避免了现今的异构网络的许多问题。例如，设法允许在使用任何指令集的任何 ISA，例

如 Java 虚拟机这样的虚拟机上进行应用程序处理的低效编程模型得以避免。因此，系统 101 可实现远比现今网络能有效和高效的宽带处理。

用于网络 104 的所有成员的基本处理模块是处理器元件(PE)。图 2 说明了 PE 的结构。如图所示，PE 201 包含处理单元(PU)203、直接存储器存取控制器(DMAC) 205 和多个附连处理单元(APU)，即 APU 207、APU 209、APU 211、APU 213、APU 215、APU 217、APU 219 和 APU 221。局部 PE 总线 223 在 APU、DMAC 205 和 PU 203 间传输数据和应用程序。局部 PE 总线 223 可具有例如传统体系结构，或者可以被实现成分组交换网络。虽然需要更多的硬件，然而实现成分组交换网络可增加可用带宽。

可使用用于实现数字逻辑的各种方法来构建 PE 201。然而，PE 201 最好被构造成使用硅衬底上的互补金属氧化物半导体（CMOS）的单个集成电路。可选的衬底材料包括砷化镓，镓铝砷化物，以及其它使用各种掺杂物的所谓 III–B 化合物。还可使用超导材料，例如快速单磁通量子(RSFQ)逻辑来实现 PE 201。

PE 201 通过高带宽存储器连接 227 与动态随机存取存储器(DRAM) 225 紧密相关。DRAM 225 用作 PE 201 的主存储器。虽然 DRAM 225 最好是动态随机存取存储器，但也可使用其它装置将 DRAM 225 实现成例如静态随机存取存储器(SRAM)、磁性随机存取存储器(MRAM)、光存储器或全息存储器。DMAC 205 利于在 DRAM 225 与 PE 201 的 APU 和 PU 之间传输数据。如下面进一步讨论的，DMAC 205 在 DRAM 225 中为每个 APU 指定一个独占区域，只有该 APU 可以往里写数据或从中读数据。这个独占区域被称为“沙箱”。

PE 203 可以是例如能够独立处理数据和应用程序的标准处理器。在操作中，PU 203 调度并编排 APU 对数据和应用程序的处理。APU 最好是单指令多数据(SIMD)处理器。在 PU 203 的控制下，APU 以并行和独立的方式执行这些数据和应用程序的处理。DMAC 205 控

制 PU 203 和 APU 对存储在共享 DRAM 225 中的数据和应用程序的存取。虽然 PE 201 最好包括八个 APU，但也可根据所需的处理能力在 PE 中使用更多或更少数目的 APU。并且，可将许多的像 PE 201 这样的 PE 连接或封装在一起以提供增强的处理能力。

例如，如图 3 所示，可将四个 PE 一起封装或连接在例如一个或多个芯片封装内，以形成用于网络 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 PE，即 PE 303、PE 305、PE 307 和 PE 309。通过 BE 总线 311 进行这些 PE 之间的通信。宽带宽存储器连接 313 提供共享 DRAM 315 与这些 PE 之间的通信。作为 BE 总线 311 的替代，可通过 DRAM 315 和这个存储器连接进行 BE 301 的 PE 之间的通信。

输入/输出(I/O)接口 317 和外部总线 319 提供宽带引擎 301 和网络 104 的其它成员之间的通信。与 PE 的 APU 执行的并行与独立的应用程序和数据处理相类似，BE 301 的每个 PE 以并行和独立的方式执行数据和应用程序的处理。

图 4 说明了 APU 的结构。APU 402 包括局部存储器 406，寄存器 410，四个浮点单元 412 和四个整数单元 414。然而，根据所需要的处理能力，可以使用更多或更少数目的浮点单元 412 和整数单元 414。在优选实施例中，局部存储器 406 包含 128 千字节的存储量，并且寄存器 410 的容量是 128×128 位。浮点单元 412 最好以 320 亿(32 billion)次浮点运算每秒(32 GFLOPS)的速度工作，整数单元 414 最好以 320 亿次运算每秒(32 GOPS)的速度工作。

局部存储器 402 不是高速缓冲存储器。局部存储器 402 最好被构造成 SRAM。对 APU 的高速缓存一致性支持是没有必要的。对于由 PU 启动的直接存储器存取，PU 会需要高速缓存一致性支持。然而，对于由 APU 启动的直接存储器存取或针对外部设备的存取，则不需要高速缓存一致性支持。

APU 402 还包含总线 404，用于针对 APU 传送应用程序和数据。在优选实施例中，该总线的宽度是 1024 位。APU 402 还包括内

部总线 408、420 和 418。在优选实施例中，总线 408 的宽度是 256 位，提供局部存储器 406 与寄存器 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，寄存器 410 和整数单元 414 之间的通信。在优选实施例中，从寄存器 410 到浮点单元或整数单元的总线 418 和 420 的宽度是 384 位，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。这些总线中，从寄存器 410 到浮点或整数单元的宽度比从这些单元到寄存器 410 的宽度大，这样可适应处理期间来自寄存器 410 的较大数据流。每次计算最多需要三个字。然而，每次计算的结果通常只是一个字。

图 5-10 进一步说明了网络 104 的成员的处理器的模块化结构。例如，如图 5 所示，处理器可以包含单个 PE 502。如上所述，该 PE 通常包含一个 PU、一个 DMAC 和八个 APU。每个 APU 包括局部存储器(LS)。另一方面，处理器可包含观察器(VS，visualizer) 505 的结构。如图 5 所示，VS 505 包含 PU 512、DMAC 514 和四个 APU，即 APU 516、APU 518、APU 520 和 APU 522。在这种情况下，通常由 PE 的其它四个 APU 占有的芯片封装内空间则由像素引擎 508，图象高速缓存 510，以及阴极射线管控制器(CRTC)504 占用。根据 PE 502 或 VS 505 所需要的通信速度，光接口 506 也可被包括在芯片封装内。

使用这种标准模块化的结构，可以很容易和有效地构建处理器的许多其他变种。例如，图 6 所示的处理器包含两个芯片封装，即包含一个 BE 的芯片封装 602 和包含四个 VS 的芯片封装 604。输入/输出(I/O) 606 提供芯片封装 602 的 BE 与网络 104 之间的接口。总线 608 提供芯片封装 602 与芯片封装 604 之间的通信。输入输出处理器(IOP)610 控制数据流入和流出 I/O 606。I/O 606 可以制造成专用集成电路(ASIC)。从 VS 输出的是视频信号 612。

图 7 说明了带有两个光接口 704 和 706 的 BE 702 的芯片封装，所述光接口用于提供与网络 104 的其它成员(或本地连接的其它芯片封装)的甚高速通信。BE 702 可用作例如网络 104 上的服务器。

图 8 的芯片封装包含两个 PE 802 和 804，以及两个 VS 806 和 808。I/O 810 提供芯片封装与网络 104 间的接口。从芯片封装输出的是视频信号。这种结构可用作例如图形工作站。

图 9 说明了另外一种结构。该结构包含图 8 中说明的结构的处理能力的一半。提供一个 PE 902，而不是两个 PE，并且提供一个 VS 904，而不是两个 VS。I/O 906 的带宽是图 8 中说明的 I/O 带宽的一半。然而，这样的处理器也可用作图形工作站。

最后一种结构如图 10 所示。该处理器仅由一个单一的 VS 1002 和 I/O 1004 组成。这种结构可用作例如 PDA。

图 11A 说明了光接口在网络 104 的处理器的芯片封装中的集成。这些光接口将光信号转换成电信号，并将电信号转换成光信号，它们可以用各种材料制成，包括例如砷化镓、镓铝砷化物、锗以及其它元素和化合物。如图中所示，光接口 1104 和 1106 被做在 BE 1102 的芯片封装上。BE 总线 1108 提供 BE 1102 的多个 PE，即 PE 1110、PE 1112、PE 1114、PE 1116 与这些光接口之间的通信。光接口 1104 包括两个端口，即端口 1118 和端口 1120，并且光接口 1106 也包括两个端口，即端口 1122 和端口 1124。端口 1118、1120、1122 和 1124 分别连接到光波导 1126、1128、1130 和 1132。通过光接口 1104 和 1106 的端口，经由这些光波导针对 BE 1102 传送光信号。

使用这种光波导和每个 BE 的四个光端口可将多个 BE 按各种结构连接在一起。例如，如图 11B 所示，通过这种光端口可将两个或多个 BE，例如 BE 1152、BE 1154 和 BE 1156 串行连接在一起。在该例中，BE 1152 的光接口 1166 通过其光端口连接到 BE 1154 的光接口 1160 的光端口。以类似的方式，BE 1154 上光接口 1162 的光端口连接到 BE 1156 上光接口的光端口。

图 11C 说明了一种矩阵结构。在该结构中，每个 BE 的光接口连接到两个其它的 BE 上。如图中所示，BE 1172 的光接口 1188 的一个光端口连接到 BE 1176 的光接口 1182 的光端口。光接口 1188 的另一个光端口连接到 BE 1178 的光接口 1184 的光端口。以类似的方

式，BE 1174 的光接口 1190 的一个光端口连接到 BE 1178 的光接口 1184 的另一个光端口。光接口 1190 的另一个光端口连接到 BE 1180 的光接口 1186 的光端口。这种矩阵结构可按类似的方式延及其它 BE。

通过使用串联结构或矩阵结构，可构建网络 104 的具有任何期望大小和能力的处理器。当然，可以向 BE 的光接口，或其 PE 数目多于或少于 BE 的 PE 数目的处理器增加附加端口，以形成其它的结构。

图 12A 说明了用于 BE 的 DRAM 的控制系统和结构。类似的控制系统和结构被用于具有其它大小和包含或多或少的 PE 的处理器中。如图中所示，交叉开关将构成 BE 1201 的四个 PE 的各个 DMAC 1210 连接到八个存储体控制器 1206。每个存储体控制器 1206 控制 DRAM 1204 的八个存储体 1208(图中只画出了四个)。因此，DRAM 1204 总共包含 64 个存储体。在优选实施例中，DRAM 1204 具有 64 兆字节的容量，并且每个存储体具有 1 兆字节的容量。在该优选实施例中，每个存储体中的最小可寻址单位是 1024 位块。

BE 1201 还包括开关单元 1212。开关单元 1212 能够使 BE 上的其它 APU 紧密耦合到 BE 1201，以便访问 DRAM 1204。因此，第二 BE 可以紧密耦合到第一 BE，并且每个 BE 的每个 APU 可以寻址的存储器位置的数量是通常可由 APU 访问的存储器位置的数量的两倍。通过例如开关单元 1212 的开关单元，可以直接从第一 BE 的 DRAM 到第二 BE 的 DRAM，或从第二 BE 的 DRAM 到第一 BE 的 DRAM 的读取或写入数据。

例如，如图 12B 所示，为完成这种写操作，第一 BE 的 APU，例如，BE 1222 的 APU 1220 向第二 BE 的 DRAM，例如 BE 1226 的 DRAM 1228(而不是象通常的情况那样向 BE 1222 的 DRAM 1224)的存储器位置发出写命令。BE 1222 的 DMAC 1230 通过交叉开关 1221 向存储体控制器 1234 发送写命令，并且存储体控制器 1234 向连接到存储体控制器 1234 的外部端口 1232 发送该命令。BE 1226 的 DMAC

1238 接收该写命令，并将该命令传送到 BE 1226 的开关单元 1240。开关单元 1240 识别包含在写命令中的 DRAM 地址，并通过 BE 1226 的存储体控制器 1242 向 DRAM 1228 的存储体 1244 发送存储在该地址中的数据。因此，开关单元 1240 使 DRAM 1224 和 DRAM 1228 能够充当 BE 1222 的 APU 的单独存储空间。

图 13 说明了 DRAM 的 64 存储体的结构。这些存储体排列成八行，即行 1302、1304、1306、1308、1310、1312、1314 和 1316，和八列，即列 1320、1322、1324、1326、1328、1330、1332 和 1334。每行由一个存储体控制器控制。因此，每个存储体控制器控制八兆字节的存储器。

图 14A 和 14B 说明了用于存储和访问 DRAM 的最小可寻址存储单位，例如 1024 位块的不同结构。在图 14A 中，DMAC 1402 在单个存储体 1404 中存储八个 1024 位块 1406。另一方面，在图 14B 中，当 DMAC 1412 读和写包含 1024 位的数据块时，这些块在两个存储体，即存储体 1414 和存储体 1416 之间交错。因此，这些存储体中的每一个包含 16 个数据块，并且每个数据块包含 512 位。这种交错可有利于更快速地访问 DRAM，并且在某些应用程序的处理中是非常有用的。

图 15 说明了 PE 中的 DMAC 1506 的体系结构。如图中所示，构成 DMAC 1506 的结构硬件在整个 PE 中分布，使得每个 APU 1502 直接访问 DMAC 1506 的结构节点 1504。每个节点执行适合于由该节点直接访问的 APU 进行存储器访问的逻辑。

图 16 示出了 DMAC 的一个可选实施例，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。APU 1602 和 PU 1604 通过局部 PE 总线 1607 与 DMAC 1606 通信。DMAC 1606 通过交叉开关连接到总线 1608。总线 1608 连接到 DRAM 1610。

如上所述，PE 的所有多个 APU 可独立访问共享 DRAM 中的数据。结果，在第二 APU 请求第一 APU 的局部存储器中的特定数据

时，第一 APU 可能正对该数据进行操作。如果此时将数据从共享 DRAM 提供到第二 APU，则数据可能是无效的，因为第一 APU 正在进行的处理可能改变数据的值。因此，如果此时第二处理器接收到了来自共享 DRAM 的数据，则第二处理器可能会产生错误的结果。例如，数据可能是全局变量的特定值。如果第一处理器在其处理过程中改变了该值，则第二处理器会接收到一个过期值。因此，必须有一种方案来同步 APU 对共享 DRAM 中的存储器位置进行的数据读出和写入。该方案必须防止从这样的存储器位置读取数据，其中另一 APU 当前正在其局部存储器中对该存储器位置进行操作，因此此时读取的数据不是最新的，并且防止向存储当前数据的存储器位置写数据。

为了解决这些问题，对于 DRAM 的每个可寻址存储器位置，在 DRAM 中分配一个附加存储段以存储有关该存储器位置中存储的数据的状态信息。这种状态信息包括满/空(F/E)位，向存储器位置请求数据的 APU 的标识(APU ID)，以及从其读取所请求数据的 APU 局部存储器的地址(LS 地址)。DRAM 的可寻址存储器位置可以具有任意大小。在优选实施例中，该大小是 1024 位。

将 F/E 位设成 1 表示存储在相关存储器位置中的数据是最新的。相反，将 F/E 位设成 0 表示存储在相关存储器位置中的数据不是最新的。如果 APU 在该位被设成 0 时请求数据，则禁止 APU 立即读取数据。在这种情况下，标识请求数据的 APU 的 APU ID，和 LS 地址被记入到附加存储段中，其中 LS 地址标识在数据变为最新数据时，向读入数据的 APU 的局部存储器中的存储器位置。

对于 APU 的局部存储器中的每个存储器位置，也分配附加存储段。该附加存储段存储一个位，称为“忙位（busy bit）”。该忙位用于保留相关的 LS 存储器位置，用于存储从 DRAM 取回的特定数据。如果针对局部存储器中特定存储器位置的忙位被设成 1，则 APU 只能将该存储器位置用于写入这些特定数据。反之，如果针对局部存储器中特定存储器位置的忙位被设成 0，则 APU 可将该存储器位置用于写入任何数据。

图 17A-17O 说明了使用 F/E 位，APU ID，LS 地址以及忙位来同步针对 PE 的共享 DRAM 的数据读写的方式的几个例子。

如图 17A 所示，一或多个例如 PE 1720 的 PE 与 DRAM 1702 交互。PE 1720 包括 APU 1722 和 APU 1740。APU 1722 包括控制逻辑 1724，而 APU 1740 包括控制逻辑 1742。APU 1722 还包括局部存储器 1726。该局部存储器包括多个可寻址存储器位置 1728。APU 1740 包括局部存储器 1744，该局部存储器还包括多个可寻址存储器位置 1746。所有这些可寻址存储器位置的大小最好是 1024 位。

附加存储段与每个 LS 可寻址存储器位置相关。例如，存储段 1729 和 1734 分别与局部存储器位置 1731 和 1732 相关，并且，存储段 1752 与局部存储器位置 1750 相关。如上所述，“忙位”存储在这些附加存储段的每一个中。局部存储器位置 1732 如图所示具有几个 X，表示该存储器位置包含数据。

DRAM 1702 包含多个可寻址存储器位置 1704，包括存储器位置 1706 和 1708。这些存储器位置的大小最好也是 1024 位。附加存储段也与这些存储器位置中的每一个相关。例如，附加存储段 1760 与存储器位置 1706 相关，并且附加存储段 1762 与存储器位置 1708 相关。与存储在每个存储器位置的数据相关的状态信息被存储在与该存储器位置相关的存储段中。如上所述，该状态信息包括 F/E 位，APU ID 和 LS 地址。例如，对于存储器位置 1708，该状态信息包括 F/E 位 1712，APU ID 1714 和 LS 地址 1716。

通过使用状态信息和忙位，能够实现在 PE 或一组 PE 的 APU 中间对共享 DRAM 的数据读写同步。

图 17B 说明了从 APU 1722 的 LS 存储器位置 1732 向 DRAM 1702 的存储器位置 1708 的数据同步写入的启动。APU 1722 的控制器 1724 启动这些数据的同步写入。因为存储器位置 1708 是空的，所以 F/E 位 1712 设为 0。结果，可以将 LS 存储器位置 1732 中的数据写入存储器位置 1708 中。另一方面，如果该位设为 1 以指示存储器位置 1708 是满的，并且包含的数据是最新和有效的，则控制器 1722

将收到错误消息，并且被禁止向该存储器位置写入数据。

图 17C 示出了成功地向存储器位置 1708 同步写入数据的结果。写入的数据被存储在存储器位置 1708 中，并将 F/E 位 1712 设成 1。该设定指示存储器位置 1708 是满的，并且该存储器位置中的数据是最新和有效的。

图 17D 说明了从 DRAM 1702 的存储器位置 1708 同步读取数据到局部存储器 1744 的 LS 存储器位置 1750 的启动。为启动该读取操作，将 LS 存储器位置 1750 的存储段 1752 中的忙位设成 1，以为这些数据保留该存储器位置。将该忙位设成 1 可禁止 APU 1740 在该存储器位置存储其它数据。

接着如图 17E 所示，控制逻辑 1742 针对 DRAM 1702 的存储器位置 1708 发出同步读命令。因为与该存储器位置相关的 F/E 位 1712 被设成 1，所以存储在存储器位置 1708 中的数据被认为是最新和有效的。因此，将 F/E 位 1712 设成 0，以准备将数据从存储器位置 1708 传送到 LS 存储器位置 1750。该设定如图 17F 所示。将该位设成 0 表示：在读取这些数据之后，存储器位置 1708 中的数据无效。

接下来，如图 17G 所示，将存储器位置 1708 中的数据从存储器位置 1708 读到 LS 存储器位置 1750 中。图 17H 示出了最终的状态。将存储器位置 1708 中的数据的副本存储到 LS 存储器位置 1750 中。F/E 位被设成 0 以指示存储器位置 1708 中的数据无效。这种无效是 APU 1740 对这些数据进行的修改的结果。存储段 1752 中的忙位也被设成 0。该设定指示 LS 存储器位置 1750 现在可被 APU 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，APU 1740 可对 LS 存储器位置 1750 进行任何目的的访问。

图 17I-17O 说明了当针对 DRAM 1702 的存储器位置的 F/E 位设成 0 以指示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)同步读数据到 APU 的局部存储器的 LS 存储器位置(例如局部存储器 1744 的 LS 存储器位置

1752)的操作。如图 17I 所示，为启动这种传送，将 LS 存储器位置 1750 的存储段 1752 中的忙位设成 1，以保留该 LS 存储器位置用于该数据传输。接下来，如图 17J 所示，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 被设成 0，所以存储器位置 1708 中存储的数据是无效的。结果，向控制逻辑 1742 发送信号，以阻止从该存储器位置立即读取数据。

接着，如图 17K 所示，将用于该读命令的 APU ID 1714 和 LS 地址 1716 写入到存储段 1762 中。在这种情况下，将针对 APU 1740 的 APU ID 和针对 LS 存储器位置 1750 的 LS 存储器位置写入到存储段 1762 中。因此，当存储器位置 1708 中的数据变成最新时，使用该 APU ID 和 LS 存储器位置来确定当前数据要发送到的位置。

当 APU 将数据写入该存储器位置时，存储器位置 1708 中的数据变为有效和最新的。图 17L 说明了从例如 APU 1722 的存储器位置 1732 到存储器位置 1708 的数据同步写操作。因为针对该存储器位置的 F/E 位被设为 0，所以这些数据的这种同步写是许可的。

如图 17M 所示，在写操作之后，存储器位置 1708 中的数据变成最新和有效的。因此，立即从存储段 1762 读取来自存储段 1762 的 APU ID 1714 和 LS 地址，然后将该信息从该存储段中删除。还将 F/E 位 1712 设成 0，以期立即读取存储器位置 1708 中的数据。如图 17N 所示，当读取 APU ID 1714 和 LS 地址 1716 后，立即将该信息用于将存储器位置 1708 中的有效数据读到 APU 1740 的 LS 存储器位置 1750 中。最终状态示于图 17O。该图示出从存储器位置 1708 复制到存储器位置 1750 的有效数据，存储段 1752 中的忙位设为 0，并且存储段 1762 中的 F/E 位 1712 设为 0。将此忙位设成 0 允许 LS 存储器位置 1750 现在可被 APU 1740 进行任何目的的访问。将 F/E 位设成 0 指示存储器位置 1708 中的数据不再是最新和有效的。

图 18 根据对应于存储器位置的存储段中存储的 F/E 位的状态，APU ID 和 LS 地址，概括了上述操作和 DRAM 的存储器位置的各种

状态。存储器位置可有三种状态。这三种状态是：空状态 1880，其中 F/E 位设成 0，并且不针对 APU ID 或 LS 地址提供信息；满状态 1882，其中 F/E 位设成 1，并且不针对 APU ID 或 LS 地址提供信息；阻塞状态 1884，其中 F/E 位设成 0，并且针对 APU ID 和 LS 地址提供信息。

如该图所示，在空状态 1880 中，允许同步写操作，并且导致切换到满状态 1882。而同步读操作导致切换到阻塞状态 1884，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 1882 中，允许同步读操作，并导致切换到空状态 1880。另一方面，满状态 1882 下的同步写操作被禁止，以防止覆盖有效数据。如果在这种状态中试图进行这种写操作，则不发生状态改变，并且向 APU 的相应控制逻辑发送错误消息。

在阻塞状态 1884 中，允许对存储器位置进行同步数据写操作，并导致切换到空状态 1880。另一方面，阻塞状态 1884 下的同步读操作被禁止，以防止与较早的、导致这种状态的同步读操作发生冲突。如果在阻塞状态 1884 下试图进行同步读操作，则不发生状态改变，并且向 APU 的对应控制逻辑发送错误消息。

上述用于针对共享 DRAM 的同步读和写数据操作的方案还可用于节省通常被处理器专用于针对外部设备读写数据的计算资源。该输入/输出(I/O)功能可由 PU 执行。然而，通过使用该同步方案的修改方案，运行适当程序的 APU 可执行该功能。例如，通过使用该方案，接收由外部设备发出的、用于从 I/O 接口传输数据的中断请求的 PU 可以委托该 APU 来处理这个请求。然后，APU 向 I/O 接口发出同步写命令。该接口则通知外部设备：现在可以向 DRAM 写入数据。接下来，APU 向 DRAM 发出同步读命令，以将 DRAM 的有关存储空间设成阻塞状态。APU 还将 APU 局部存储器中接收数据所需的存储器位置的忙位设成 1。在阻塞状态中，与 DRAM 的有关存储空间相关的附加存储段包含 APU 的 ID 和 APU 局部存储器的有关存储器位置的地址。接着，外部设备发出同步写命令，以直接向

DRAM 的有关存储空间写入数据。由于该存储空间处于阻塞状态，所以立即将数据从该空间读到附加存储段中所标识的 APU 局部存储器的存储器位置上。然后将针对这些存储器位置的忙位设成 0。当外部设备完成数据的写操作时，APU 向 PU 发出表明传输完成的信号。

因此，通过使用这个方案，可处理来自外部设备的数据传输，并使 PU 的计算负荷最小。然而，被委派此功能的 APU 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 PE 的 DRAM 包括多个“沙箱”。沙箱定义了共享 DRAM 的一个区域，如果超出此区域，特定 APU 或特定一组 APU 不能读或写数据。这些沙箱提供了防止正被一个 APU 处理的数据遭到正被另一个 APU 处理的数据破坏的安全性。这些沙箱还允许从网络 104 向特定沙箱下载软件单元，而不存在软件单元破坏整个 DRAM 中的任何数据的可能性。在本发明中，用 DRAM 和 DMAC 的硬件来实现沙箱。通过用硬件而不是软件来实现沙箱，可获得速度和安全性方面的优势。

PE 的 PU 控制分配给 APU 的沙箱。由于 PU 通常只操作受信的程序，例如操作系统，该方案不会危及安全性。根据该方案，PU 建立并维护关键字控制表。图 19 示出了该关键字控制表。如该图所示，关键字控制表 1902 的每个条目包含针对 APU 的标识(ID)1904，针对该 APU 的 APU 关键字 1906，和关键字掩码 1908。该关键字掩码的使用将在后面进行说明。关键字控制表 1902 最好存储在相对较快的存储器中，例如静态随机存取存储器(SRAM)，并且与 DMAC 相关。关键字控制表 1902 中的条目由 PU 控制。当 APU 请求针对 DRAM 的特定存储器位置读写数据时，DMAC 对照与该存储器位置相关的存储器存取关键字，来评估关键字控制表 1902 中分配给该 APU 的 APU 关键字 1906。

如图 20 所示，给 DRAM 2002 的每个可寻址存储器位置 2006 分配一个专用存储段 2010。在该专用存储段中存储针对存储器位置的存储器存取关键字 2012。如上所述，另一个附加专用存储段 2008 也

与每个可寻址存储器位置 2006 相关，并且存储用于针对存储器位置读写数据的同步信息。

在操作中，APU 向 DMAC 发出 DMA 命令。该命令包括 DRAM 2002 的存储器位置 2006 的地址。在执行该命令之前，DMAC 使用 APU 的 ID 1904 在关键字控制表 1902 中查找发出请求的 APU 的关键字 1906。然后，DMAC 将发出请求的 APU 的 APU 关键字 1906 与专用存储段 2010 中存储的、和 APU 试图访问的 DRAM 存储器位置相关的存储器存取关键字 2012 进行比较。如果这两个关键字不匹配，则不执行 DMA 命令。另一方面，如果这两关键字匹配，则 DMA 命令继续进行，并且执行所请求的存储器访问。

图 21 说明了另一可选实施例。在该实施例中，PU 同样也维护存储器存取控制表 2102。存储器存取控制表 2102 包含针对 DRAM 中每个沙箱的条目。在图 21 的特定例子中，DRAM 包含 64 个沙箱。存储器存取控制表 2102 中的每个条目包含沙箱的标识(ID)2104，存储器基地址 2106，沙箱大小 2108，存储器存取关键字 2110 和存取关键字掩码 2112。存储器基地址 2106 提供 DRAM 中的、开始特定存储器沙箱的地址。沙箱大小 2108 提供沙箱的大小，并且因此提供特定沙箱的终点。

图 22 是使用关键字控制表 1902 和存储器存取控制表 2102 执行 DMA 命令的步骤的流程图。在步骤 2202 中，APU 向 DMAC 发出用于访问沙箱中的一个特定存储器位置或多个位置的 DMA 命令。该命令包括沙箱 ID 2104，用于标识被请求访问的特定沙箱。在步骤 2204 中，DMAC 使用 APU 的 ID 1904 在关键字控制表 1902 中查找发出请求的 APU 的关键字 1906。在步骤 2206 中，DMAC 使用命令中的沙箱 ID 2104 在存储器存取控制表 2102 中查找与该沙箱相关的存储器存取关键字 2110。在步骤 2208 中，DMAC 将分配给发出请求的 APU 的 APU 关键字 1906 与和沙箱相关的存取关键字 2110 进行比较。在步骤 2210 中，确定这两个关键字是否匹配。如果这两个关键字不匹配，则处理转到步骤 2212，其中 DMA 命令不继续执行，并向

发出请求的 APU，PU 或二者发送错误消息。另一方面，如果在步骤 2210 中发现两个关键字匹配，则处理进行到步骤 2214，其中 DMAC 执行 DMA 命令。

针对 APU 关键字和存储器存取关键字的关键字掩码为该系统提供了更大的灵活性。针对关键字的关键字掩码将被屏蔽位转换成通配符。例如，如果与 APU 关键字 1906 相关的关键字掩码 1908 的最后两位被设成 "掩码"(例如通过将关键字掩码 1908 中的这些位设为 1 来指定)，那么 APU 关键字可以是 1 或 0，并且仍然与存储器存取关键字匹配。例如，APU 关键字可以是 1010。该 APU 关键字通常只允许访问具有存取关键字 1010 的沙箱。然而，如果将用于该 APU 关键字的 APU 关键字掩码设成 0001，则可使用该 APU 关键字来实现对存取关键字为 1010 或 1011 的沙箱的访问。类似地，掩码设为 0001 的存取关键字 1010 可被 APU 关键字为 1010 或 1011 的 APU 访问。因为可同时使用 APU 关键字掩码和存储器关键字掩码，所以可确定 APU 对沙箱的各种可访问性。

本发明还为系统 101 的处理器提供了新的编程模型。该编程模型使用软件单元 102。可以将这些单元发送到网络 104 上的任何处理器来进行处理。这种新的编程模型还利用系统 101 的独特模块化体系结构和系统 101 的处理器。

APU 直接从 APU 局部存储器处理软件单元。APU 不直接对 DRAM 中的任何数据或程序进行操作。在 APU 处理这些数据和程序之前，将 DRAM 中的数据和程序读取到 APU 的局部存储器中。因此，APU 的局部存储器包括程序计数器，堆栈，和用于执行这些程序的其它软件组件。PU 通过向 DMAC 发出直接存储器存取(DMA)命令来控制 APU。

图 23 示出了软件单元 102 的结构。如该图所示，软件单元(例如软件单元 2302)包含路由信息部份 2304 和主体 2306。路由信息部份 2304 中包含的信息取决于网络 104 的协议。路由信息部份 2304 包含标头（header）2308，目的 ID 2310，源 ID 2312 和回复 ID 2314。目

的 ID 包括网络地址。例如，在 TCP/IP 协议下，网络地址是网际协议(IP)地址。目的 ID 2310 还包括 PE 和 APU 的身份信息，其中应当向该 PE 和 APU 发送单元以进行处理。源 ID 2314 包含网络地址，并标识发出单元的 PE 和 APU，以便在必要时允许目的 PE 和 APU 得到关于单元的附加信息。回复 ID 2314 包含网络地址，并标识关于单元的查询，和单元处理的结果应当被送往的 PE 和 APU。

单元主体 2306 包含与网络协议无关的信息。图 23 的分解部分示出了单元主体 2306 的详细情况。单元主体 2306 的标头 2320 标识单元主体的开始。单元接口 2322 包含使用单元所必需的信息。该信息包括全局唯一 ID 2324，所需的 APU 2326，沙箱大小 2328，以及前一个单元 ID 2330。

全局唯一 ID 2324 在整个网络 104 中唯一标识软件单元 2302。根据源 ID 2312，例如源 ID 2312 中的 PE 或 APU 的唯一标识，以及软件单元 2302 的产生或发送时间和日期产生全局唯一 ID 2324。所需的 APU 2326 提供执行该单元所需的最小数量的 APU。沙箱大小 2328 提供所需 APU 中为执行单元而需要的相关 DRAM 中的受保护存储器的数量。前一个单元 ID 2330 提供需要顺序执行(例如流数据)的一组单元中的前一个单元的身份信息。

实现部份 2332 包含单元的核心信息。该信息包括 DMA 命令表 2334，程序 2336，以及数据 2338。程序 2336 包含要由 APU 运行的程序（称为“apulets”），例如 APU 程序 2360 和 2362，而数据 2338 包含要通过这些程序处理的数据。DMA 命令表 2334 包含开始程序所需要的一系列 DMA 命令。这些 DMA 命令包括 DMA 命令 2340、2350、2355 和 2358。PU 向 DMAC 发出这些 DMA 命令。

DMA 命令 2340 包括 VID 2342。VID 2342 是 APU 的虚拟 ID，它在 DMA 命令发出时被映射到物理 ID。DMA 命令 2340 还包括装入命令 2344 和地址 2346。装入命令 2344 指示 APU 将特定信息从 DRAM 读取到局部存储器中。地址 2346 提供 DRAM 中包含该信息的虚拟地址。该信息可以是例如来自程序部份 2336 的程序，来自数

据部份 2338 的数据，或者其它数据。最后，DMA 命令 2340 包括局部存储器地址 2348。该地址标识局部存储器中应当装入信息的地址。DMA 命令 2350 包含类似的信息。也可能有其它的 DMA 命令。

DMA 命令表 2334 还包括一系列的启动(kick)命令，例如启动命令 2355 和 2358。启动命令是由 PU 向 APU 发出的、用来启动单元的处理的命令。DMA 启动命令 2355 包括虚拟 APU ID 2352，启动命令 2354，以及程序计数器 2356。虚拟 APU ID 2352 标识要被启动的 APU，启动命令 2354 提供有关的启动命令，程序计数器 2356 提供针对用于执行程序的程序计数器的地址。DMA 启动命令 2358 提供针对同一 APU 或另一 APU 的类似信息。

如前面提到的，PU 把 APU 看作独立的处理器，而不是协处理器。因此，为了控制 APU 执行的处理，PU 使用类似于远程过程调用的命令。这些命令被叫做 "APU 远程过程调用"(ARPC)。PU 通过向 DMAC 发出一系列 DMA 命令来执行 ARPC。DMAC 将 APU 程序及其相关的栈帧装入到 APU 的局部存储器中。然后，PU 向 APU 发出初始启动来执行 APU 程序。

图 24 说明了用于执行 apulet 的 ARPC 的步骤。图 24 的第一部分 2402 说明了在启动指定 APU 对 apulet 的处理时 PU 执行的步骤，并且图 24 的第二部分 2404 说明了指定 APU 在处理 apulet 时执行的步骤。

在步骤 2410 中，PU 对 apulet 进行评估，并且然后指定用于处理 apulet 的 APU。在步骤 2412 中，PU 通过向 DMAC 发出设定用于所需的一或多个沙箱的存储器存取关键字的命令，在 DRAM 中分配用于执行 apulet 的空间。在步骤 2414 中，PU 使能针对指定 APU 的中断请求，以通知 apulet 的完成。在步骤 2418 中，PU 向 DMAC 发出将 apulet 从 DRAM 装入到 APU 的局部存储器中的 DMA 命令。在步骤 2420 中，执行该 DMA 命令，并且将 apulet 从 DRAM 读取到 APU 的局部存储器中。在步骤 2422 中，PU 向 DMAC 发出将与 apulet 相关的栈帧从 DRAM 装入到 APU 的局部存储器中的 DMA 命

令。在步骤 2423 中，执行该 DMA 命令，并且将栈帧从 DRAM 读取到 APU 的局部存储器中。在步骤 2424 中，PU 发出使 DMAC 向 APU 分配关键字的 DMA 命令，以允许 APU 针对在步骤 2412 中指定的一或多个硬件沙箱读/写数据。在步骤 2426 中，DMAC 用分配给 APU 的关键字更新关键字控制表(KTAB)。在步骤 2428 中，PU 向 APU 发出启动程序处理的 DMA 命令“启动”。依据特定的 apulet，PU 在执行特定 ARPC 时可以发出其它 DMA 命令。

如上面所指出的，图 24 中的第二部分 2404 说明了 APU 在执行 apulet 时所执行的步骤。在步骤 2430 中，APU 响应在步骤 2428 发出的启动命令而开始执行 apulet。在步骤 2432 中，在 apulet 的指示下，APU 评估 apulet 的相关栈帧。在步骤 2434 中，APU 向 DMAC 发出多个 DMA 命令，以便将根据栈帧的需要而指定的数据从 DRAM 装入到 APU 的局部存储器中。在步骤 2436 中，执行这些 DMA 命令，并且将数据从 DRAM 读取到 APU 的局部存储器中。在步骤 2438 中，APU 执行 apulet 并产生结果。在步骤 2440 中，APU 向 DMAC 发出把结果存储在 DRAM 中的 DMA 命令。在步骤 2442 中，执行该 DMA 命令，并将 apulet 的结果从 APU 的局部存储器写到 DRAM 中。在步骤 2444 中，APU 向 PU 发出中断请求，以通知 ARPC 已经完成。

APU 在 PU 的指示下独立执行任务的能力允许 PU 将一组 APU，以及与一组 APU 相关的存储器资源专用于执行长时间（extended）任务。例如，PU 可以将一或多个 APU，以及与这一或多个 APU 相关的一组存储器沙箱专用于接收在长时间段内通过网络 104 发送的数据，并且将该期间内接收到的数据指引到一或多个 APU 及其相关的存储器沙箱，以便进行进一步的处理。该能力尤其利于处理通过网络 104 传输的流数据，例如 MPEG 流或流式 ATRAC 音频或视频数据。PU 可将一或多个 APU 及其相关的存储器沙箱专用于接收这些数据，并将一或多个其它 APU 及其相关的存储器沙箱专用于解压缩和进一步处理这些数据。换句话说，PU 可在一组 APU 及其相

关的存储器沙箱中间建立用于处理这些数据的专用流水线关系。

然而，为了有效地执行这种处理，在没有发生包含数据流的apulet 处理时，流水线的专用 APU 和存储器沙箱应当保持为该流水线专用。换句话说，在此期间专用 APU 及其相关的沙箱应处于保留状态。在完成 apulet 处理时进行的 APU 及其相关一或多个存储器沙箱的保留被称为“常驻终止（resident termination）”。响应来自 PU 的指令发生常驻终止。

图 25、图 26A 和图 26B 说明了包含用于处理流数据(例如 MPEG 流数据)的一组 APU 及其相关沙箱的专用流水线结构的建立。如图 25 所示，流水线结构的部件包括 PE 2502 和 DRAM 2518。PE 2502 包括 PU 2504、DMAC 2506 和多个 APU，包括 APU 2508、APU 2510 和 APU 2512。PU 2504、DMAC 2506 和这些 APU 中间的通信通过 PE 总线 2514 进行。宽带总线 2516 将 DMAC 2506 连接到 DRAM 2518。DRAM 2518 包括多个沙箱，例如沙箱 2520、沙箱 2522、沙箱 2524 和沙箱 2526。

图 26A 说明了建立专用流水线的步骤。在步骤 2610 中，PU 2504 分配 APU 2508 以处理网络 apulet。网络 apulet 包含用于处理网络 104 的网络协议的程序。在这种情况下，该协议是传输控制协议/网际协议(TCP/IP)。符合该协议的 TCP/IP 数据包通过网络 104 进行传输。一旦接收到数据包，APU 2508 处理这些数据包，并将包中的数据装配到软件单元 102 中。在步骤 2612 中，在网络 apulet 的处理完成时，PU 2504 便命令 APU 2508 执行常驻终止。在步骤 2614 中，PU 2504 指定 APU 2510 和 2512 来处理 MPEG apulet。在步骤 2615 中，当 MEPG apulet 处理完成时，PU 2504 也命令 APU 2510 和 2512 执行常驻终止。在步骤 2616 中，PU 2504 指定沙箱 2520 作为由 APU 2510 访问的源沙箱，并且作为由 APU 2508 访问的目的沙箱。在步骤 2618 中，PU 2504 指定沙箱 2522 作为由 APU 2510 访问的目的沙箱，并且作为由 APU 2512 访问的源沙箱。在步骤 2620 中，PU 2504 指定沙箱 2524 作为由 APU 2512 访问的目的沙箱，并且作为由流水线中的另一 APU 访问的源沙箱。在步骤 2622 中，PU

2504 指定沙箱 2526 作为由流水线中的其它 APU 访问的目的和源沙箱。在步骤 2624 中，APU 2510 和 APU 2512 分别向源沙箱 2520 和源沙箱 2522 中的存储器块发送同步读命令，以便将这些存储器块设成阻塞状态。最后，处理转到步骤 2628，其中完成专用流水线的建立，并且保留专用于该流水线的资源。因此，APU 2508、2510、2512 等及其相关沙箱 2520、2522 和 2524 进入保留状态。

图 26B 说明了通过这个专用流水线处理 MPEG 流数据的步骤。在步骤 2630 中，处理网络 apulet 的 APU 2508 在其局部存储器中接收来自网络 104 的 TCP/IP 数据包。在步骤 2632 中，APU 2508 处理这些 TCP/IP 数据包，并将这些包中的数据装配到软件单元 102 中。在步骤 2634 中，APU 2508 检查软件单元的标头 2320(图 23)，以确定单元中是否包含 MEPG 数据。如果单元中不包含 MPEG 数据，则在步骤 2636 中，APU 2508 将单元发送到 DRAM 2518 中指定的通用沙箱，以便由专用流水线中未包含的其它 APU 处理其它数据。APU 2508 还将此次发送通知给 PU 2504。

另一方面，如果软件单元包含 MPEG 数据，则在步骤 2638 中，APU 2508 检查单元的前一个单元 ID 2330(图 23)，以识别单元所属的 MPEG 数据流。在步骤 2640 中，APU 2508 选择专用流水线中用于处理该单元的 APU。在这种情况下，APU 2508 选择 APU 2510 来处理这些数据。该选择是根据前一个单元 ID 2330 和负荷平衡因素来进行的。例如，如果前一个单元 ID 2330 表明该软件单元所属的 MPEG 数据流的前一个软件单元被发送到 APU 2510 进行处理，则通常也会将当前软件单元发送到 APU 2510 进行处理。在步骤 2642 中，APU 2508 发出将 MPEG 数据写到沙箱 2520 的同步写命令。由于该沙箱在之前被设成阻塞状态，所以在步骤 2644 中，自动将 MPEG 数据从沙箱 2520 读到 APU 2510 的局部存储器中。在步骤 2646 中，APU 2510 在其局部存储器中处理 MPEG 数据以产生视频数据。在步骤 2648 中，APU 2510 将视频数据写到沙箱 2522 中。在步骤 2650 中，APU 2510 向沙箱 2520 发出同步读取命令，以使该沙箱准备接收另外的 MPEG 数据。在步骤 2652 中，APU 2510 处理常

驻终止。这种处理使这个 APU 进入保留状态，在保留状态期间，该 APU 等待处理 MPEG 数据流中另外的 MPEG 数据。

在一组 APU 及其相关沙箱中间可建立用于处理其它类型据的其它专用结构。例如，如图 27 所示，可以建立专用 APU 组，例如 APU 2702、2708 和 2714，用于对三维物体执行几何变换以产生二维显示表。其它 APU 可进一步处理（呈现）这些二维显示表以产生像素数据。为执行这种处理，将沙箱专用于 APU 2702、2708 和 2414，以存储三维物体，以及对这些物体进行处理而产生的显示表。例如，源沙箱 2704、2710 和 2716 专用于存储分别由 APU 2702、APU 2708 和 APU 2714 处理的三维物体。以类似的方式，目的沙箱 2706、2712 和 2718 专用于存储分别由 APU 2702、APU 2708 和 APU 2714 对这些三维物体进行处理而得到的显示表。

协调 APU 2720 专用于在其局部存储器中接收来自目的沙箱 2706、2712 和 2718 的显示表。APU 2720 在这些显示表中进行仲裁，并把它们发送到用于呈现像素数据的其他 APU。

系统 101 的处理器还使用绝对定时器。绝对定时器向 APU 及 PE 的其它单元提供时钟信号，该时钟信号独立于驱动这些单元的时钟信号，并比其更快。绝对定时器的使用示于图 28。

如图所示，绝对定时器为 APU 执行任务确定时间预算。该时间预算提供完成这些任务的时间，该时间要比 APU 处理这些任务所需的时间要长。结果，对于每项任务，在时间预算内存在忙时间段和等待时间段。根据这个时间预算编写所有 apulet 以用于处理，而不管 APU 的实际处理时间和速度如何。

例如，对于 PE 的特定 APU，特定任务可在时间预算 2804 的忙时间段 2802 中被执行。由于忙时间段 2802 比时间预算 2804 短，所以在时间预算中出现等待时间段 2806。在该等待时间段中，APU 进入休眠模式，期间 APU 消耗的功率较小。

在时间预算 2804 期满之前，其它 APU 或 PE 的其它单元不期望得到任务处理的结果。因此，通过使用由绝对定时器确定的时间预

算，不管 APU 的实际处理速度如何，APU 的处理结果总是协调的。

将来，APU 的处理速度会变得更快。然而，由绝对定时器确定的时间预算将保持相同。例如，如图 28 所示，将来的 APU 将在更短的时间内执行任务，因而将有更长的等待时间段。因此，忙时间段 2808 比忙时间段 2802 短，而等待时间段 2810 比等待时间段 2806 长。然而，由于根据由绝对定时器确定的相同时间预算编写程序以进行处理，因此，仍然保持 APU 间处理结果的协调。结果，较快的 APU 可以处理为较慢 APU 编写的处理程序，而不会在期待该处理结果时引起冲突。

作为对用于建立 APU 之间的协调的绝对定时器的代替，对于因增强的或不同的运算速度而产生的 APU 并行处理的协调问题，PU 或一或多个指定 APU 可分析 APU 在处理 apulet 时执行的特定指令或微代码。可以在指令中插入“无操作”(“NOOP”)指令，并由某些 APU 来执行，以便保持 APU 的处理按 apulet 预期的正确顺序完成。通过在指令中插入这些 NOOP，可以保持 APU 执行所有指令的正确定时。

虽然这里参照特定实施例对本发明进行了说明，但是我们应该理解，这些实施例只是为了示例性地说明本发明的原理和应用。因此，我们应该理解，在不脱离由所附权利要求限定的本发明的精神和范围的条件下，可以对示例性的实施例进行许多种修改，并且可以设计出其它的方案。

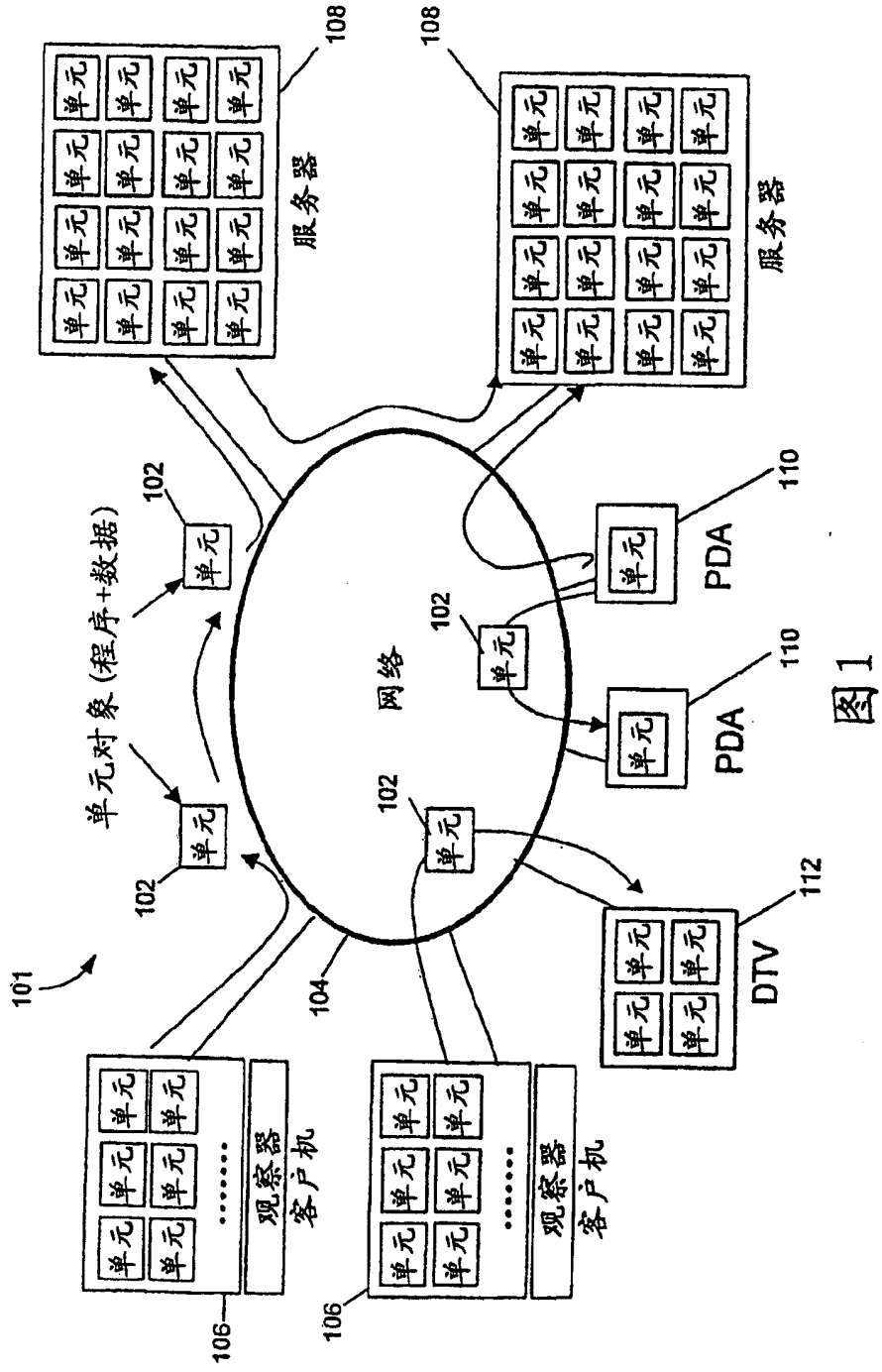

服务器
108
服务器
108
单元对象(程序+数据)
102
网络
PDA
110
PDA
110
DTV
112
观察器
客户机
106
观察器
客户机
106
101
104

图 1

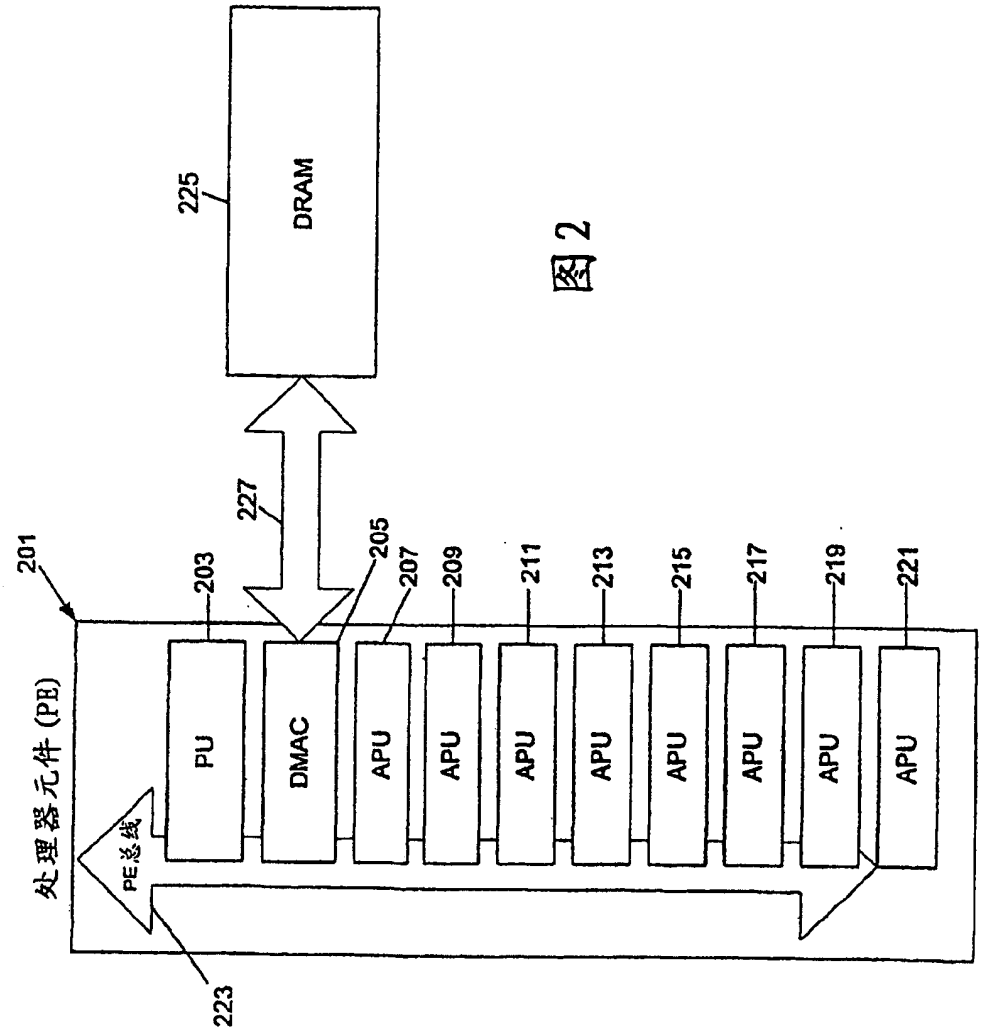

225
DRAM
227
201
203
205
207
209
211
213
215
217
219
221
处理器元件(PE)
PU
DMAC
APU
APU
APU
APU
APU
APU
APU
APU
PE总线
223

图2

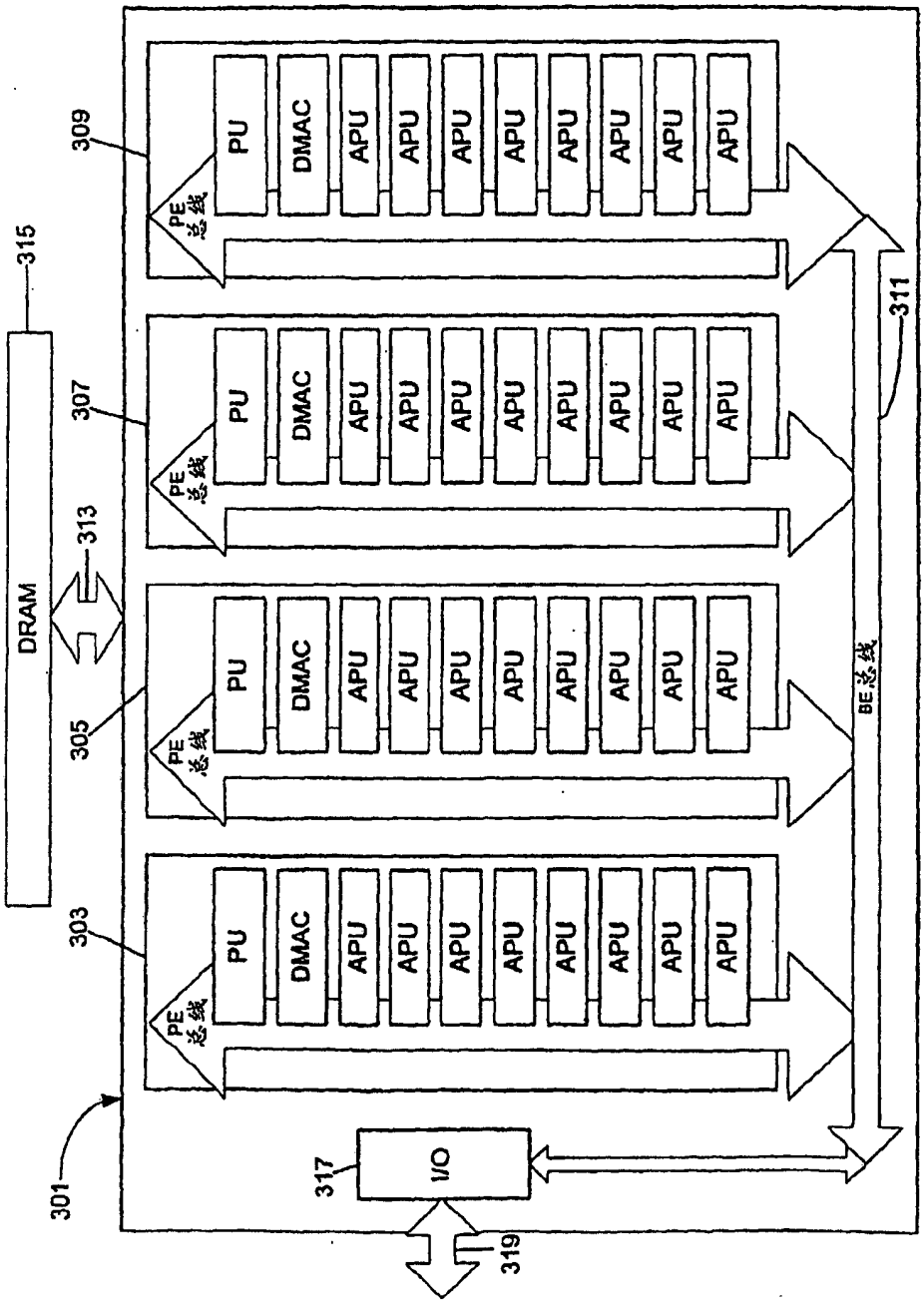

301
303
305
307
309
311
313
315
317
319
DRAM
I/O
PU
DMAC
APU
PE总线
BE总线

图 3

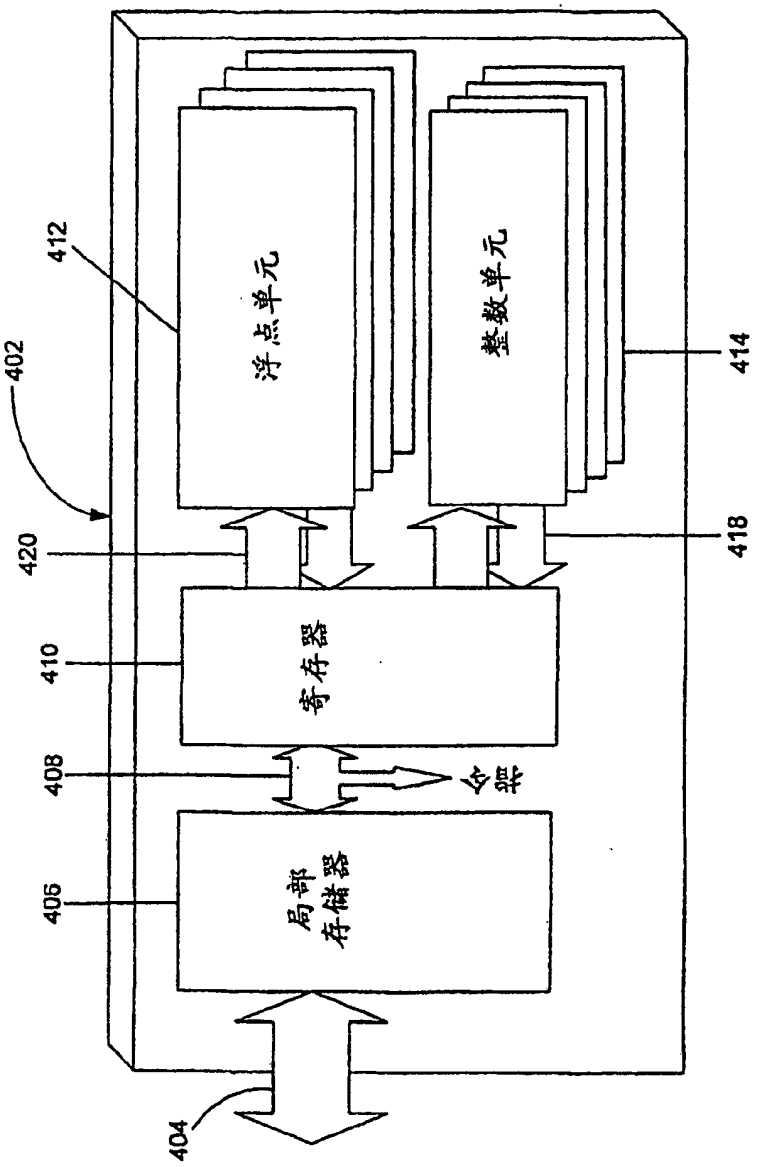



图4

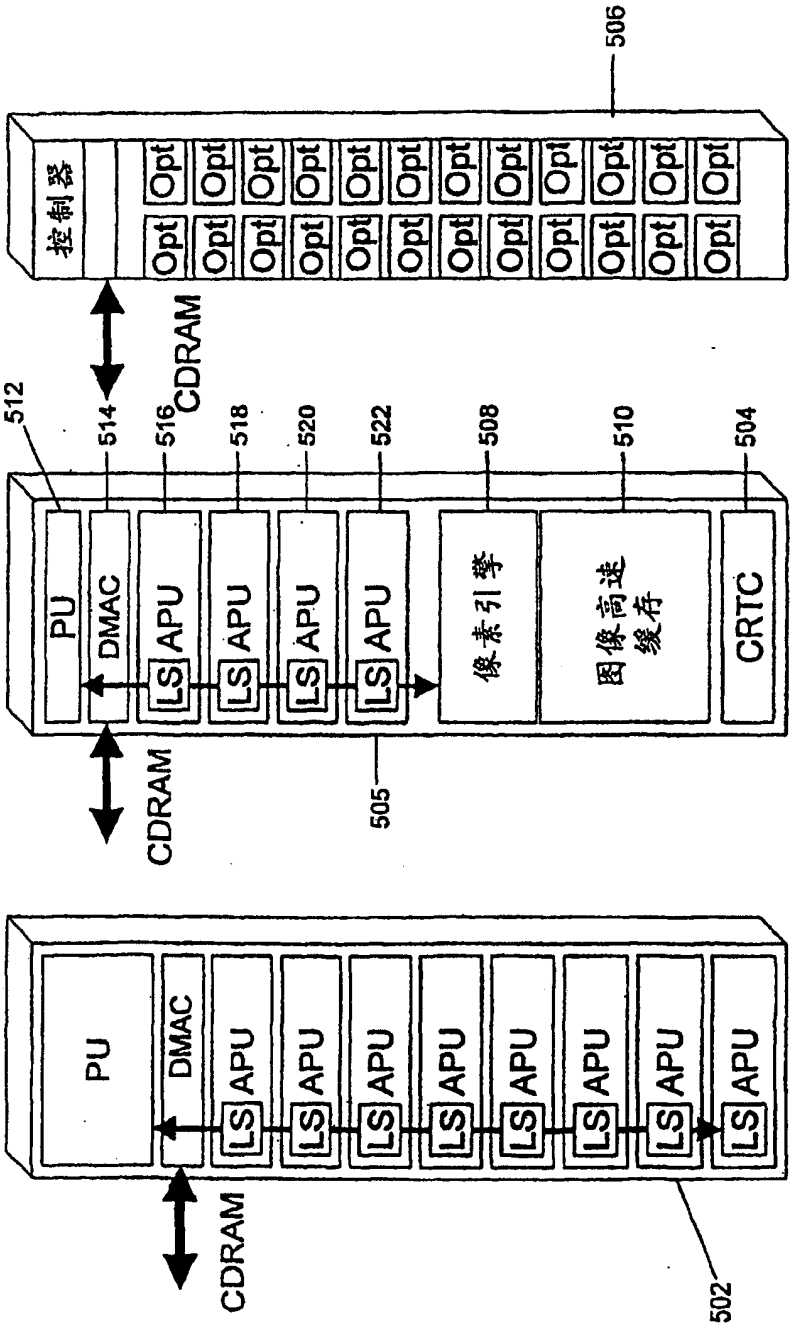

506
控制器
Opt
512
514
516
CDRAM
518
520
522
508
510
504
PU
DMAC
APU
LS
像素引擎
图像高速缓存
CRTC
CDRAM
505
CDRAM
502

图 5

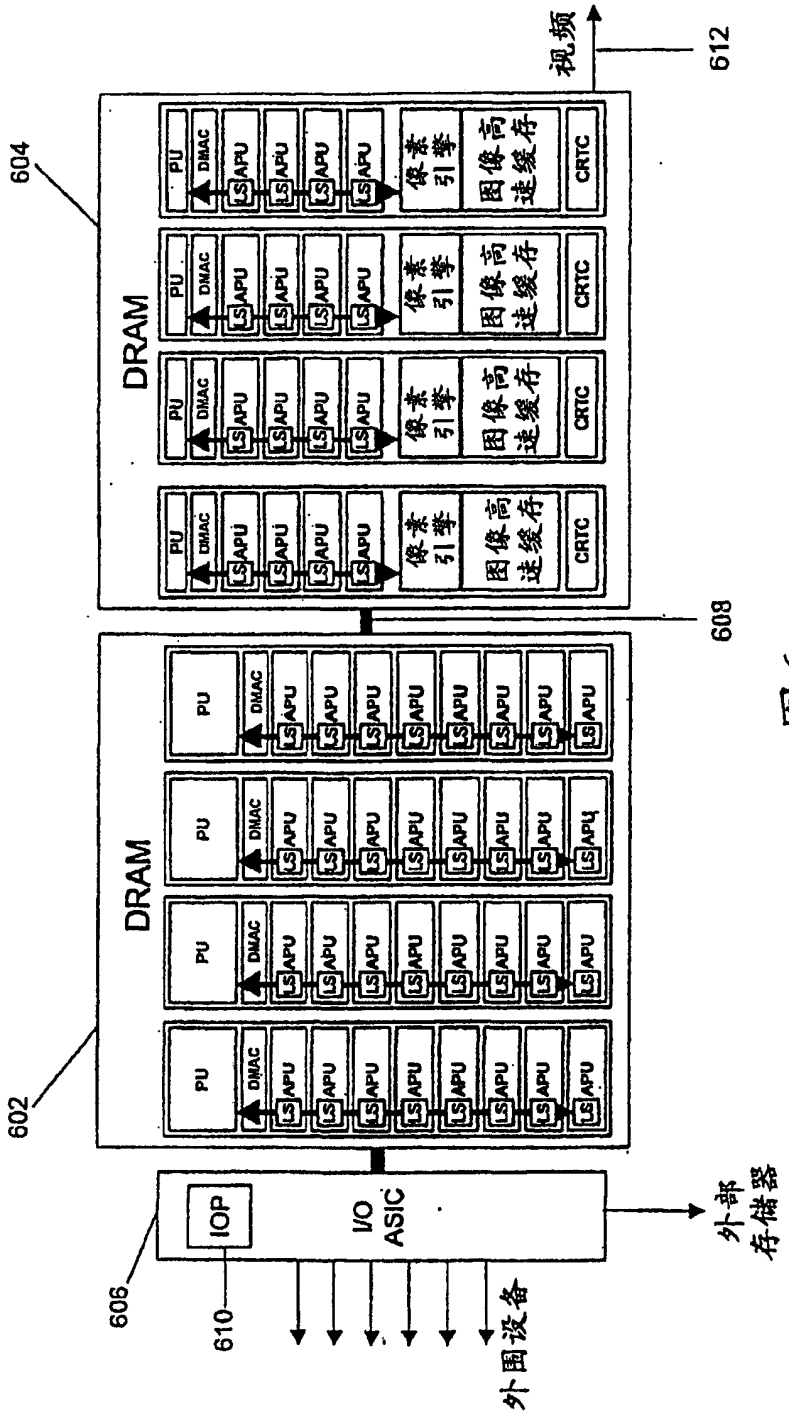

视频
612
604
DRAM
PU
DMAC
APU
像素引擎
图像高速缓存
CRTC
608
602
DRAM
606
IOP
I/O ASIC
610
外围设备
外部存储器

图 6

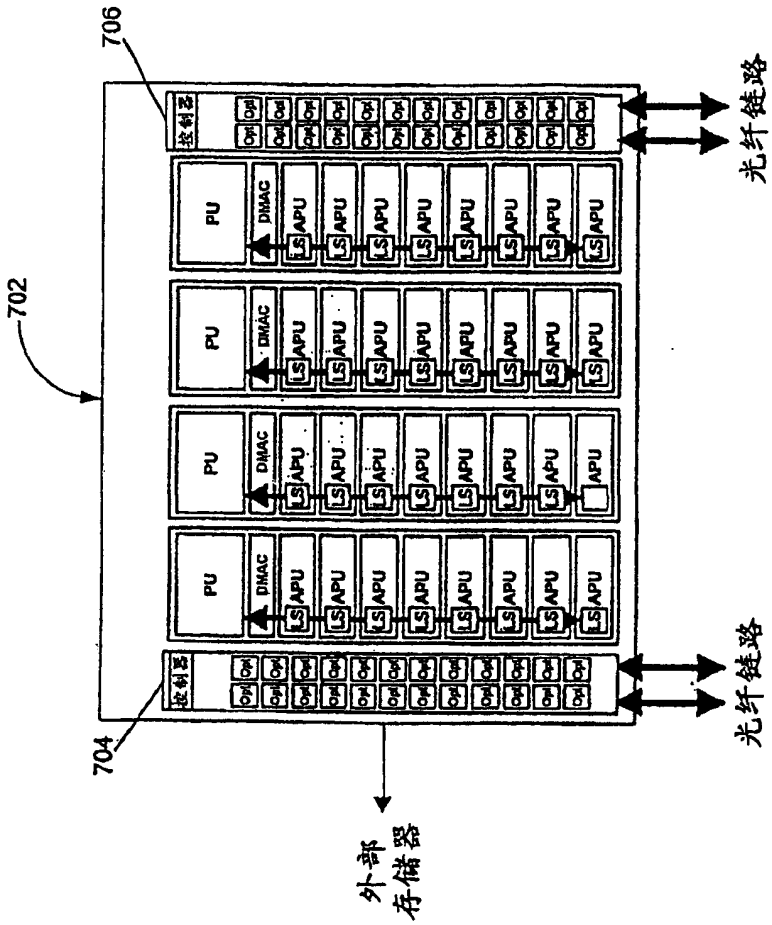



图 7

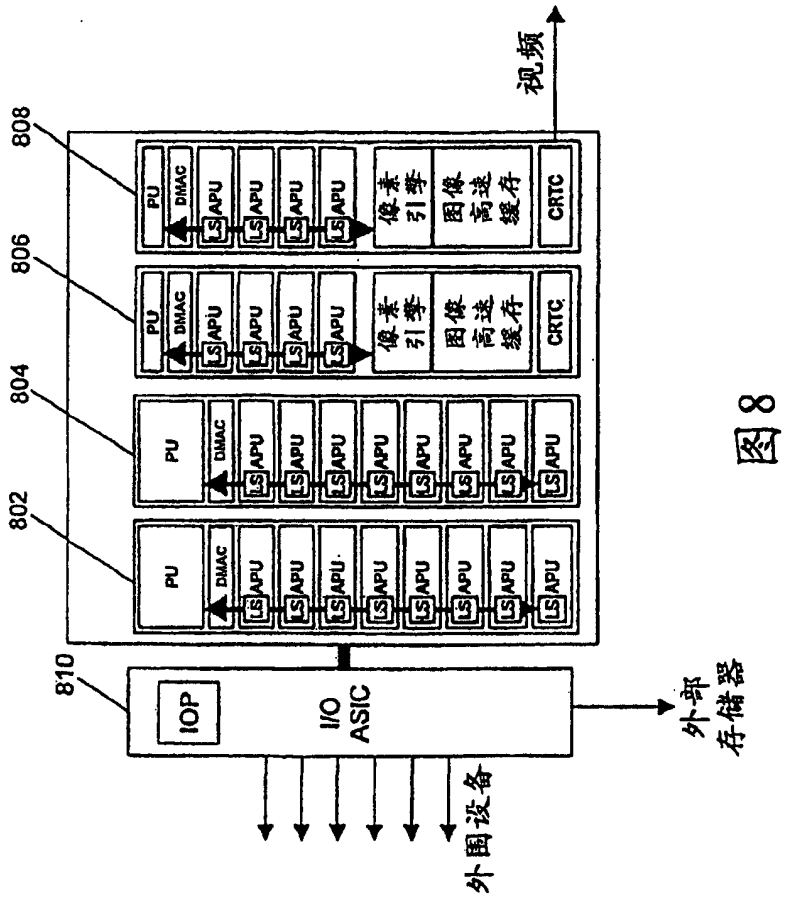

视频
808
806
804
802
810
PU
DMAC
APU
LS
像素引擎
图像高速缓存
CRTC
IOP
I/O ASIC
外部存储器
外围设备

图8

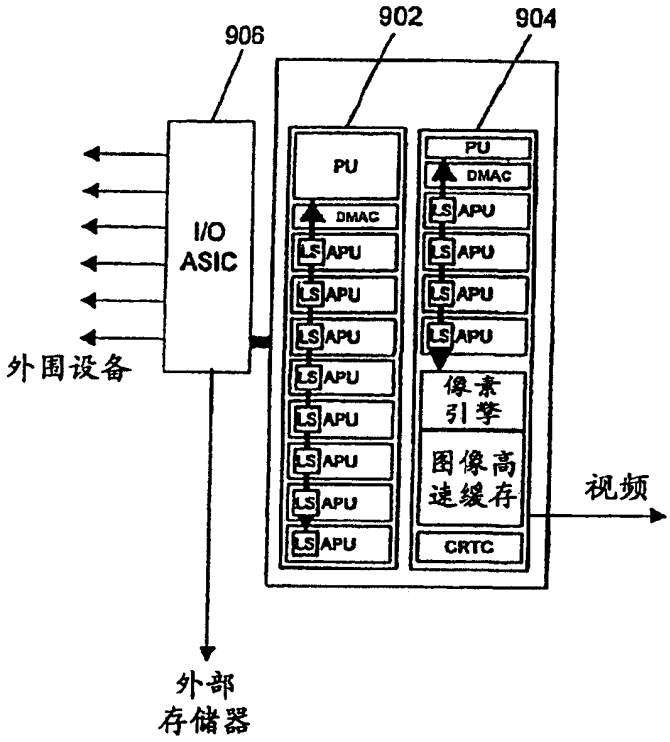

906
902
904
I/O
ASIC
外围设备
PU
DMAC
LS APU
LS APU
LS APU
LS APU
LS APU
LS APU
LS APU
LS APU
PU
DMAC
LS APU
LS APU
LS APU
LS APU
像素
引擎
图像高
速缓存
CRTC
视频
外部
存储器

图9

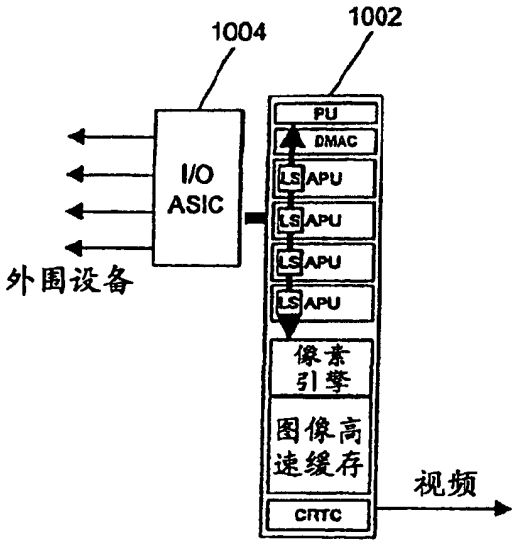

1004
1002
I/O
ASIC
外围设备
PU
DMAC
LS
APU
LS
APU
LS
APU
LS
APU
像素
引擎
图像高
速缓存
视频
CRTC

图10

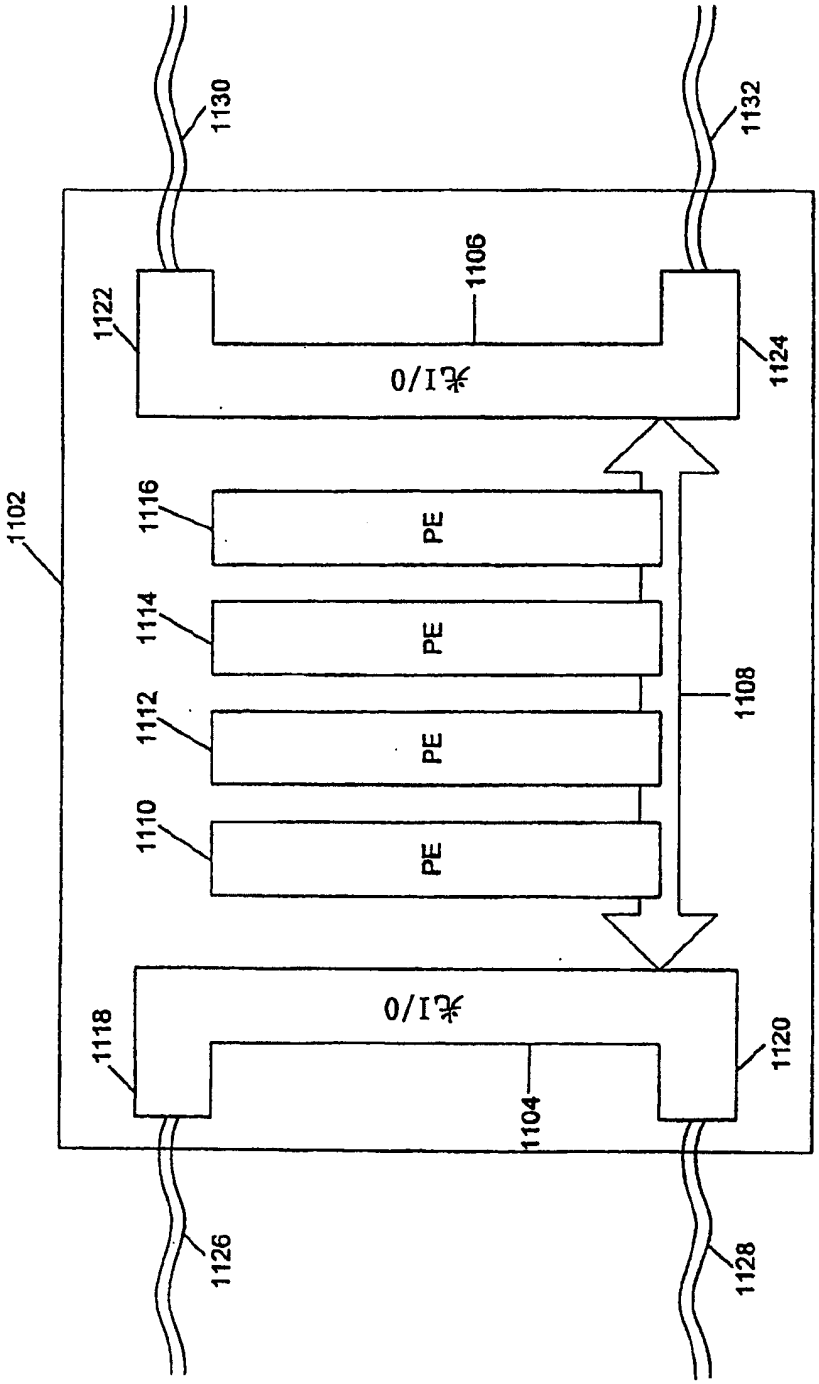

1130
1132
1106
1122
1124
I/O
1102
1116
PE
1114
PE
1112
PE
1110
PE
1108
1118
I/O
1120
1104
1126
1128

图11A

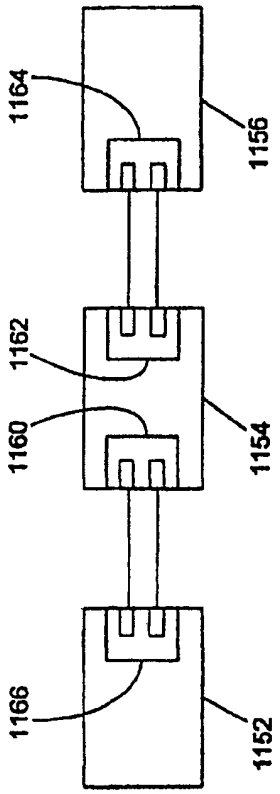



图11B

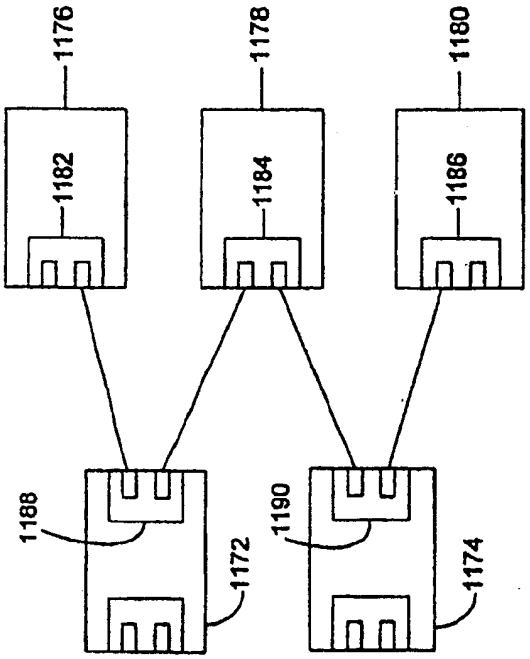



图11C

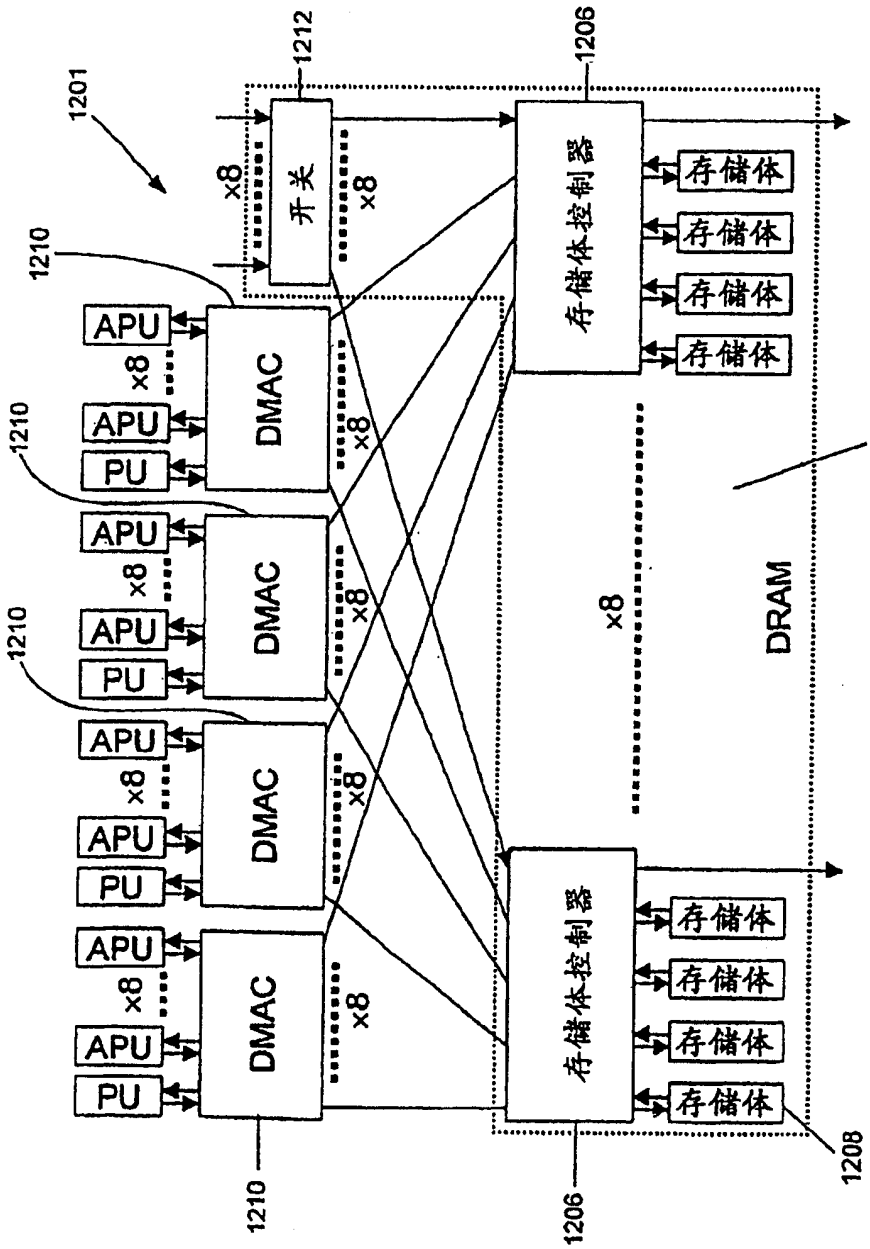

1201
1212
1206
开关
x8
1210
APU
PU
DMAC
存储体控制器
存储体
1204
DRAM
1208

图12A

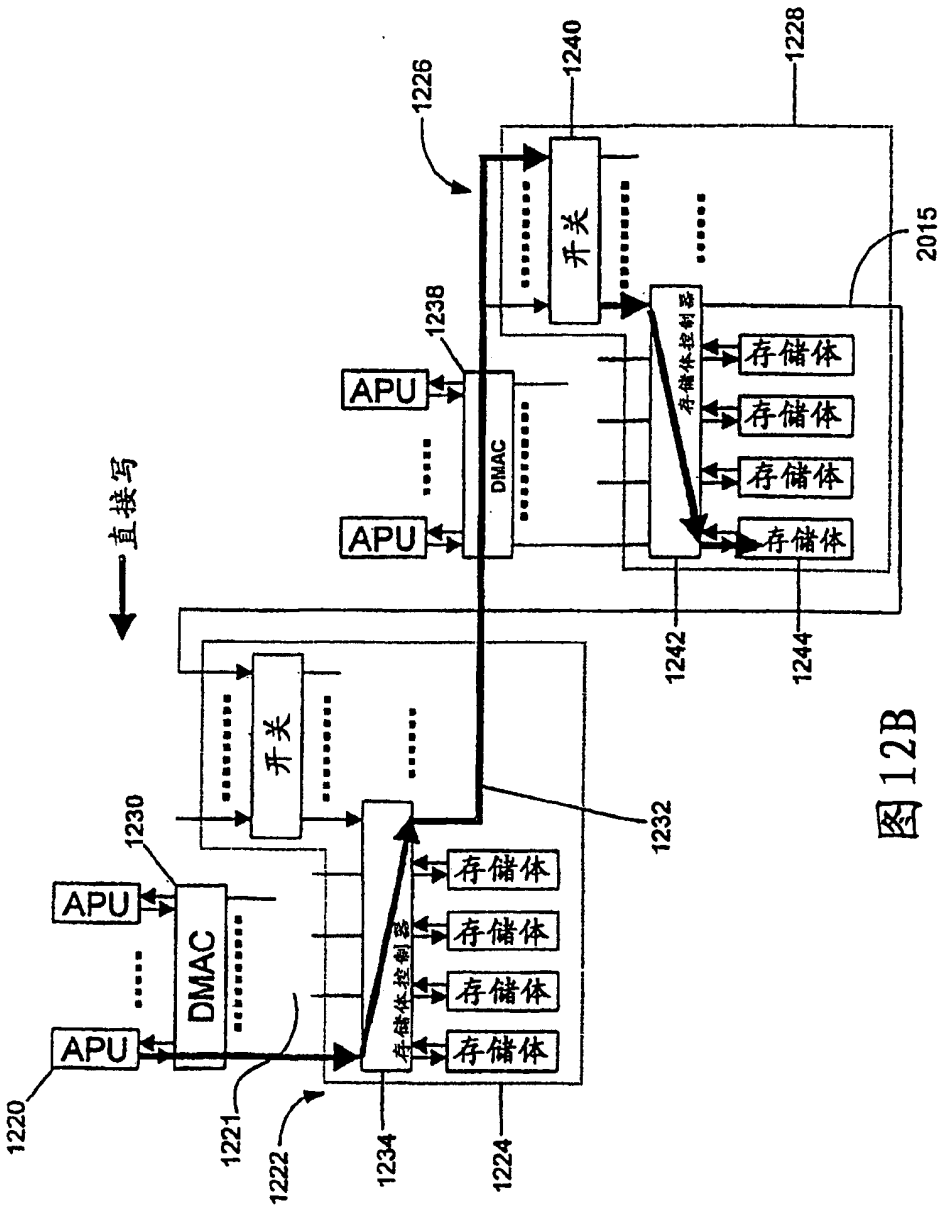

直接写
APU
APU
DMAC
1230
1220
1221
1222
开关
存储体控制器
存储体
存储体
存储体
存储体
1234
1224
1232
1226
1240
1228
2015
1238
APU
APU
DMAC
开关
存储体控制器
存储体
存储体
存储体
存储体
1242
1244

图 12B

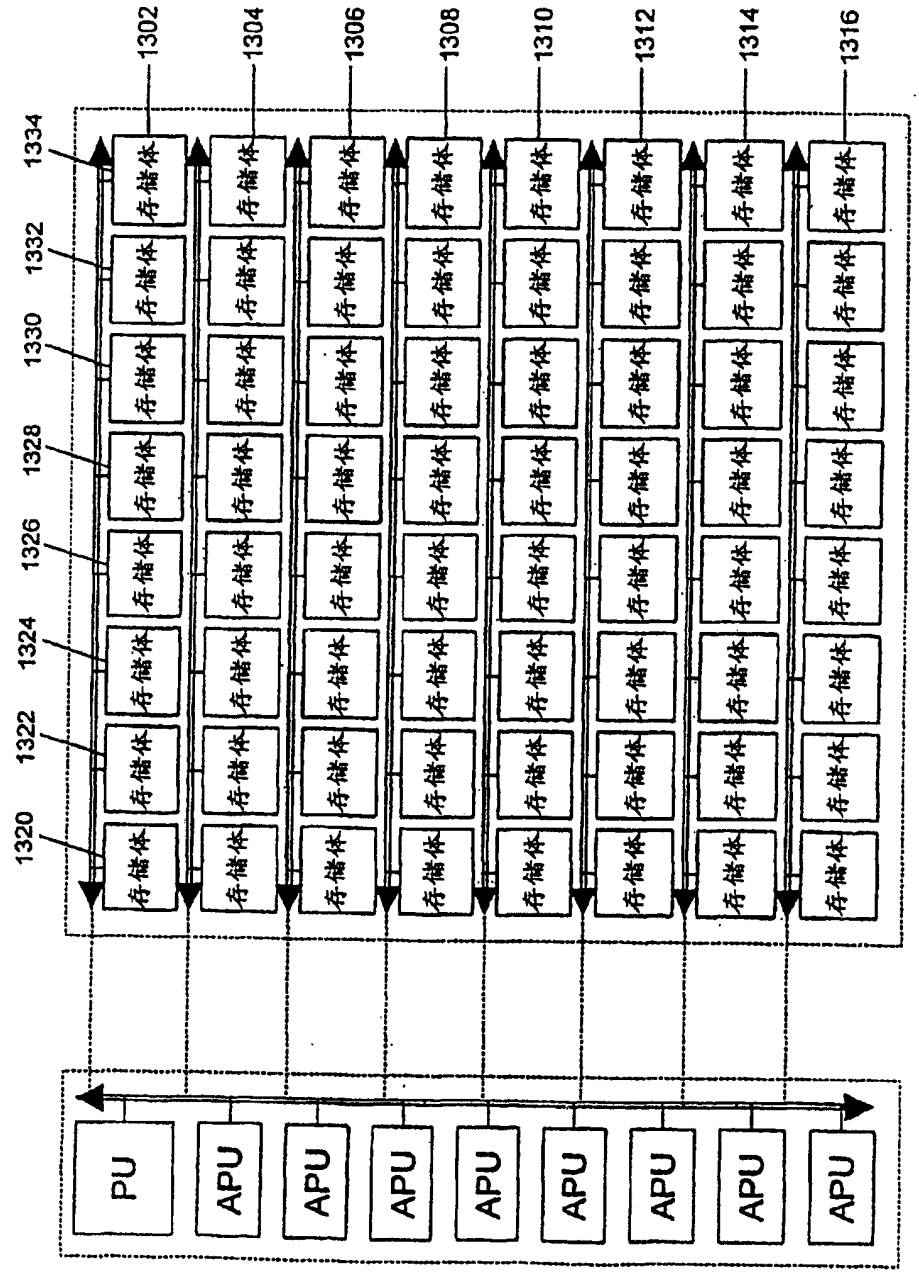

1302
1304
1306
1308
1310
1312
1314
1316
1334
1332
1330
1328
1326
1324
1322
1320
存储体
DRAM
PU
APU

图13

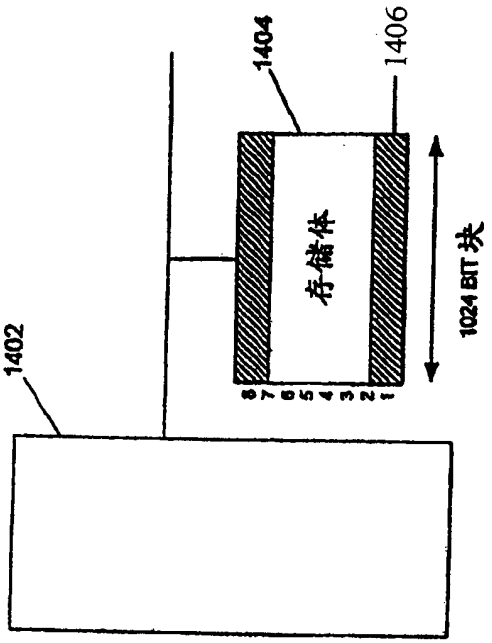



图14A

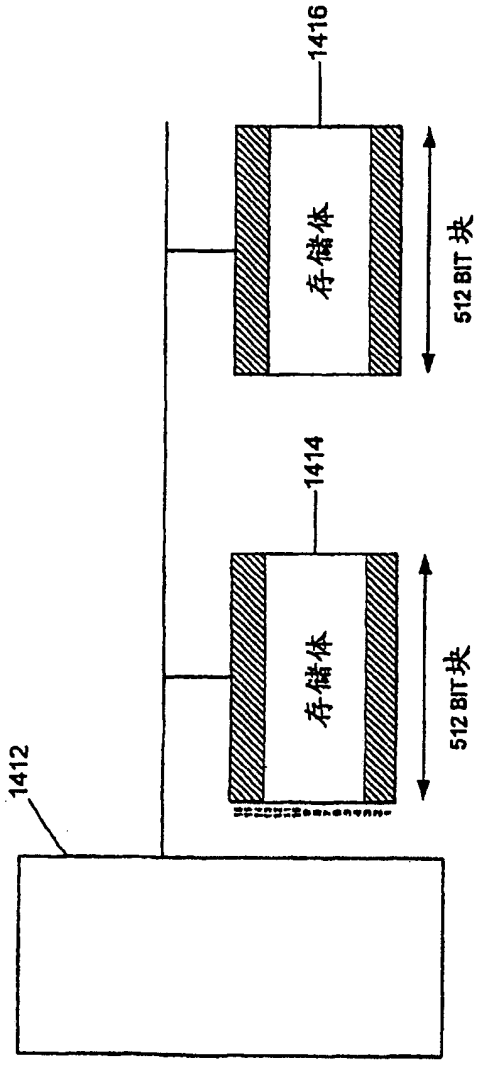



图14B

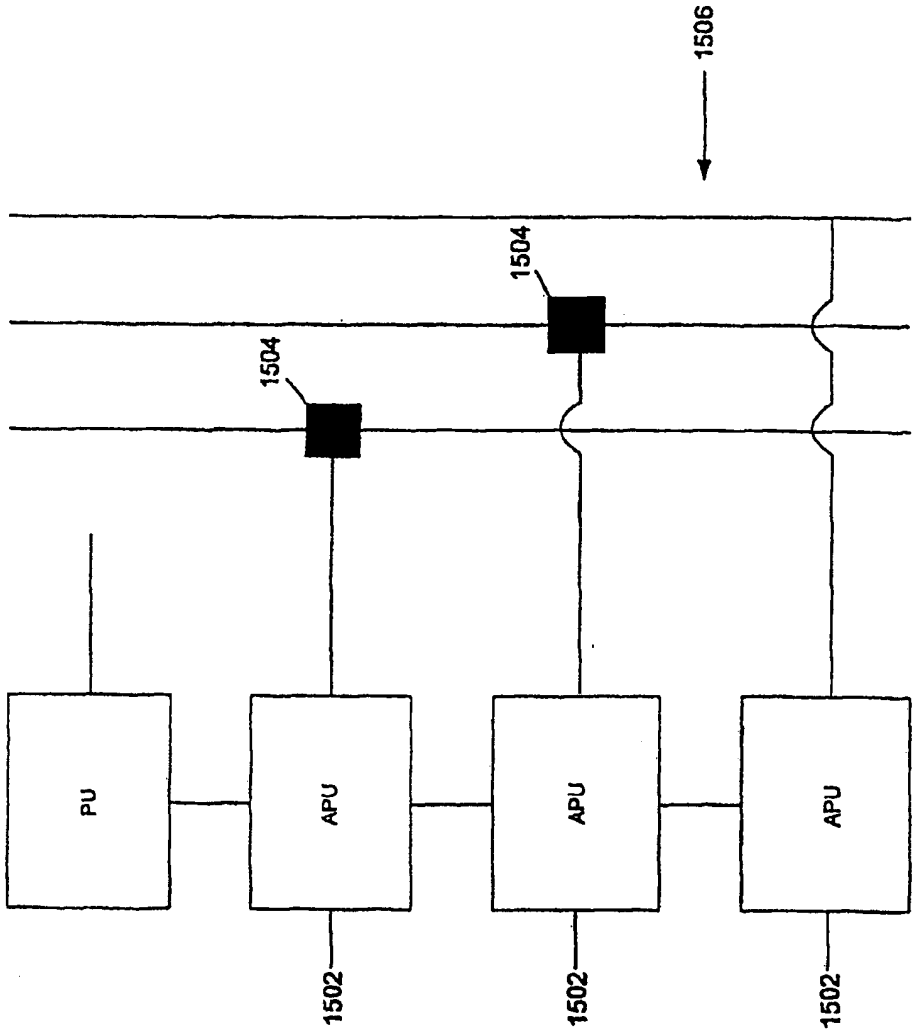



图15

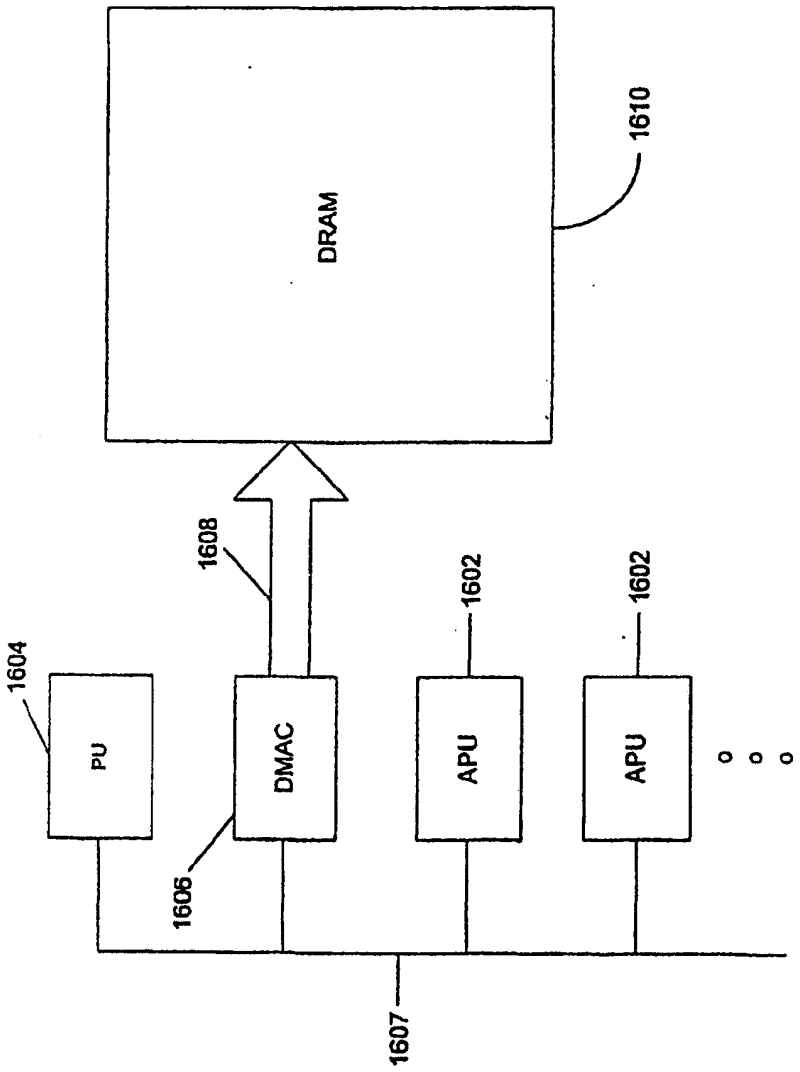



图16

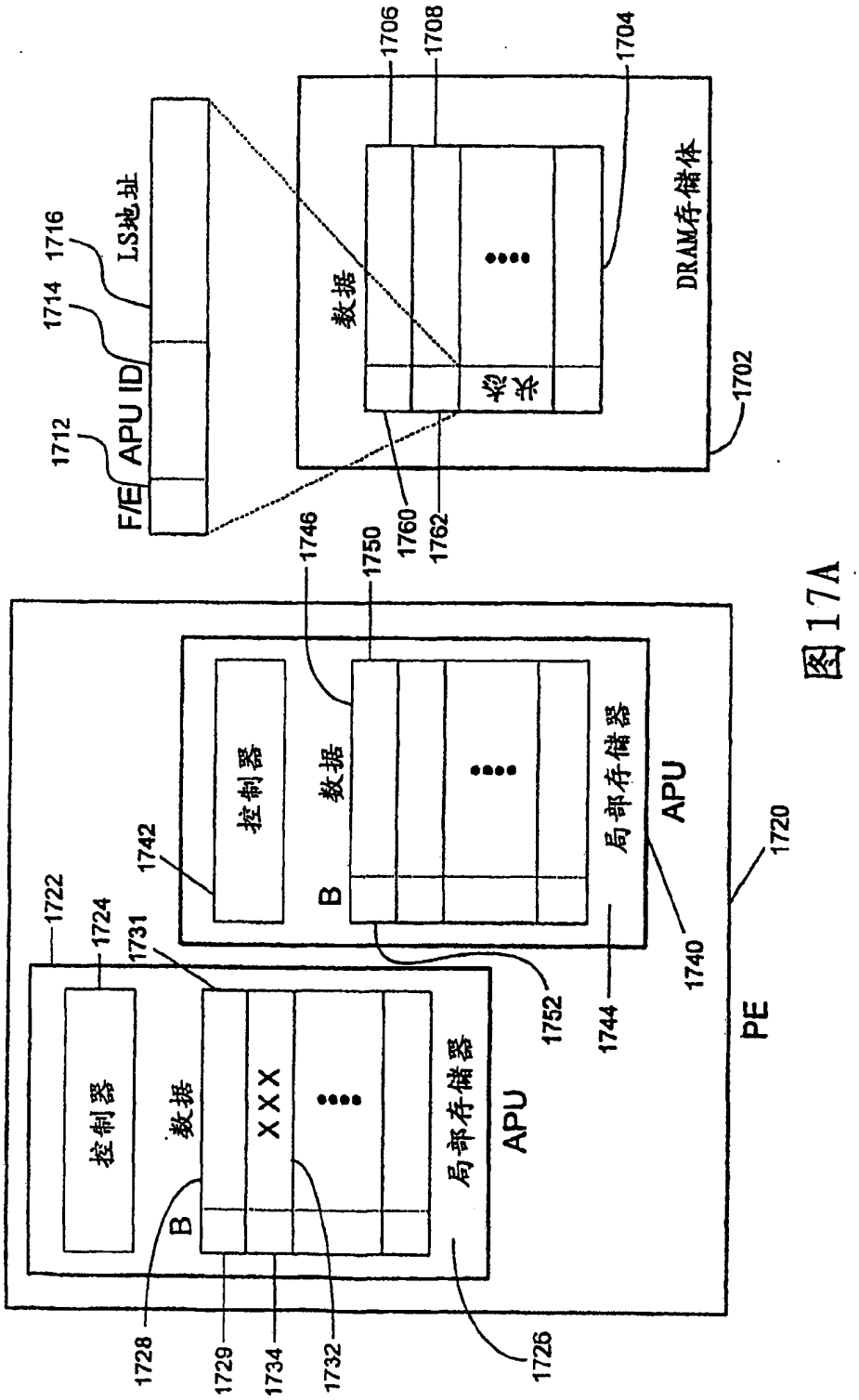

1706
1708
1704
1716
LS地址
1714
APU ID
1712
F/E
数据
DRAM存储体
1702
1746
1750
1760
1762
控制器
数据
B
1742
局部存储器
APU
1720
1722
1724
1731
1740
1752
1744
PE
XXX
1728
1729
1734
1732
1726

图 17A

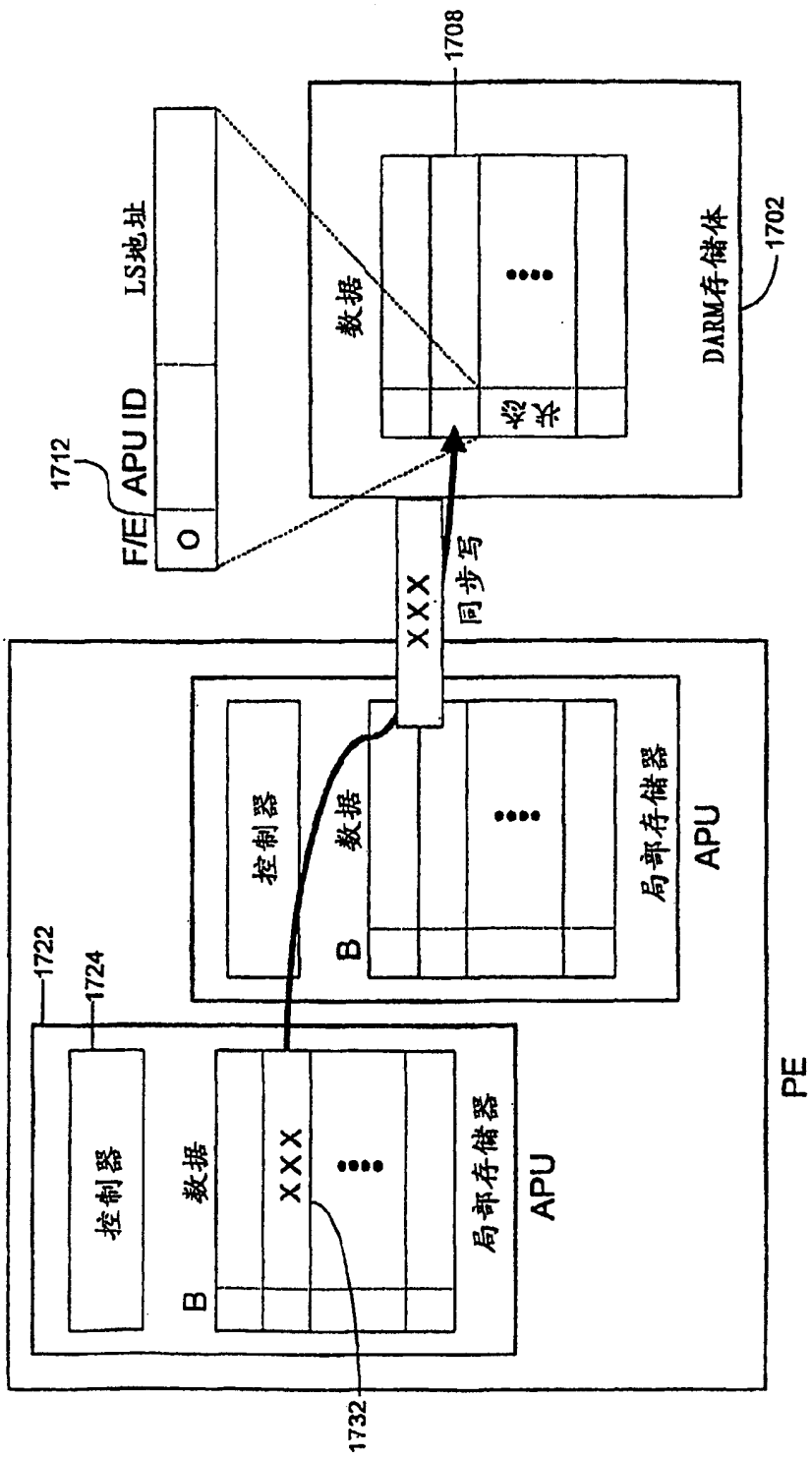

1708
LS地址
APU ID
F/E
O
1712
数据
DARM存储体
1702
同步写
XXX
控制器
数据
B
局部存储器
APU
1722
1724
XXX
1732
PE

图17B

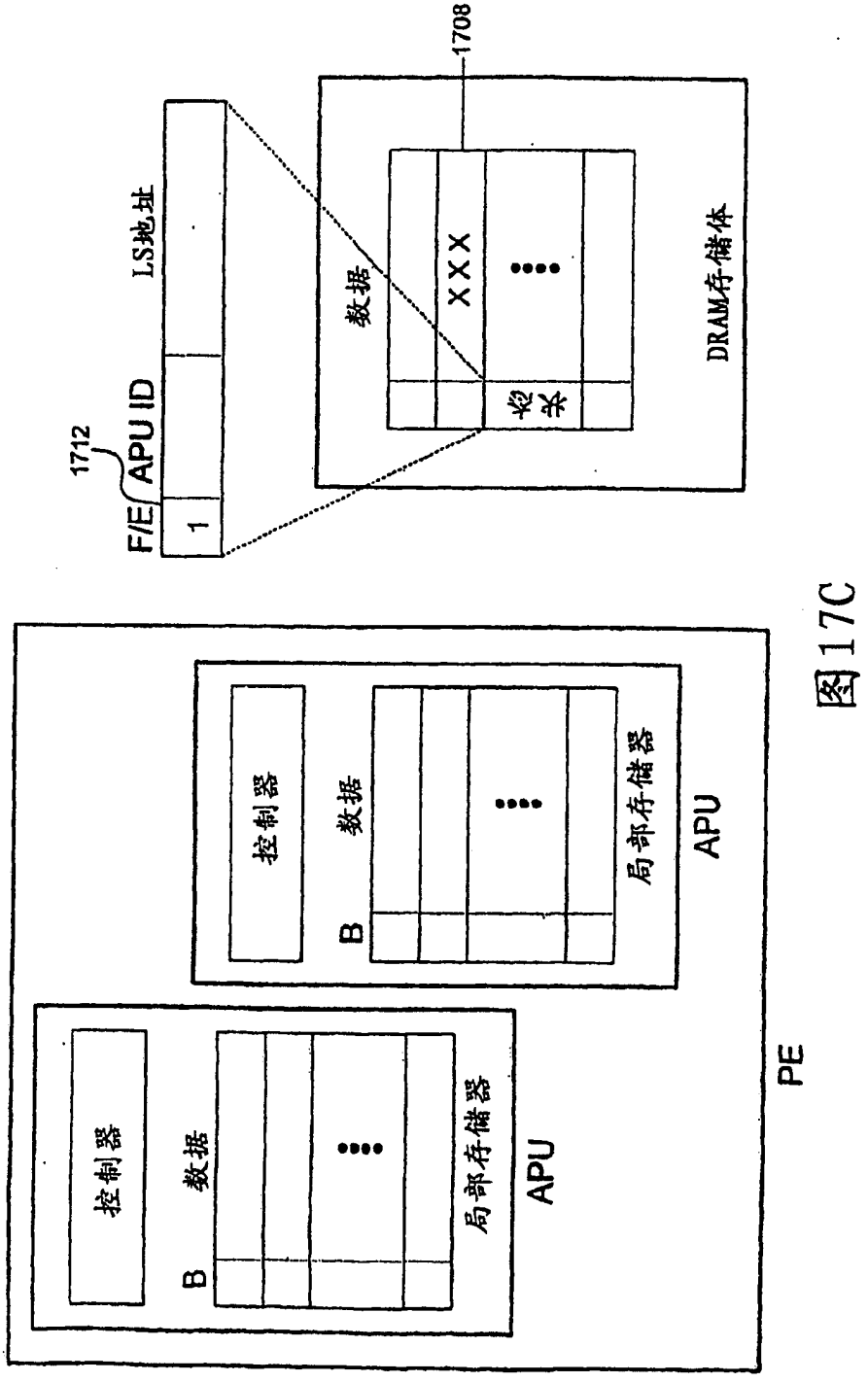

PE
APU
控制器
B
数据
局部存储器
APU
控制器
B
数据
局部存储器
F/E
APU ID
LS地址
1712
1
1708
数据
XXX
DRAM存储体

图17C

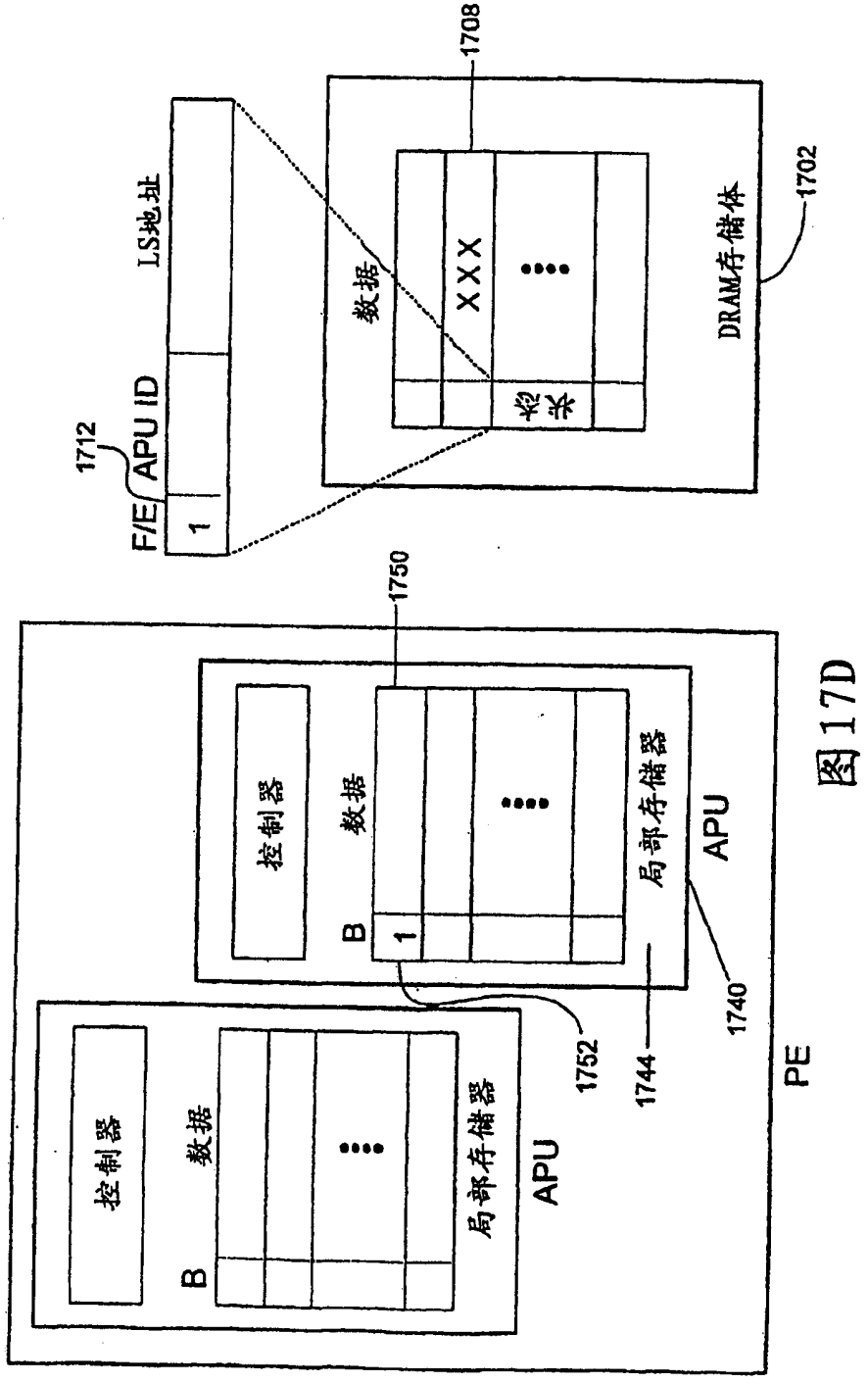

1708
LS地址
APU ID
F/E
1712
1
数据
XXX
DRAM存储体
1702
1750
控制器
数据
B
1
局部存储器
APU
1740
1752
1744
PE

图 17D

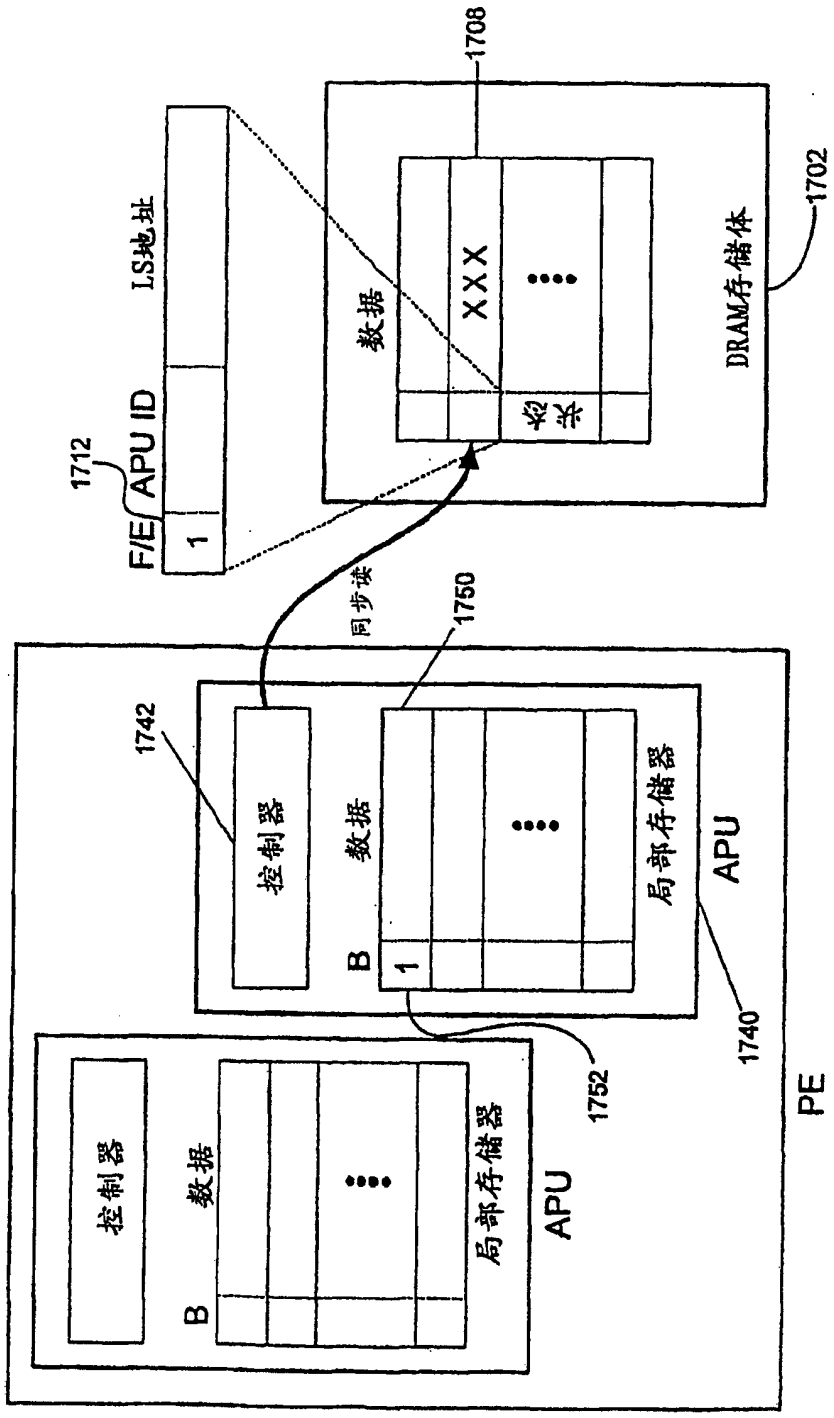

1708
LS地址
APU ID
F/E
1712
1
数据
XXX
DRAM存储体
1702
同步读
1750
1742
控制器
B
数据
1
局部存储器
APU
1740
1752
控制器
B
数据
局部存储器
APU
PE

图17E

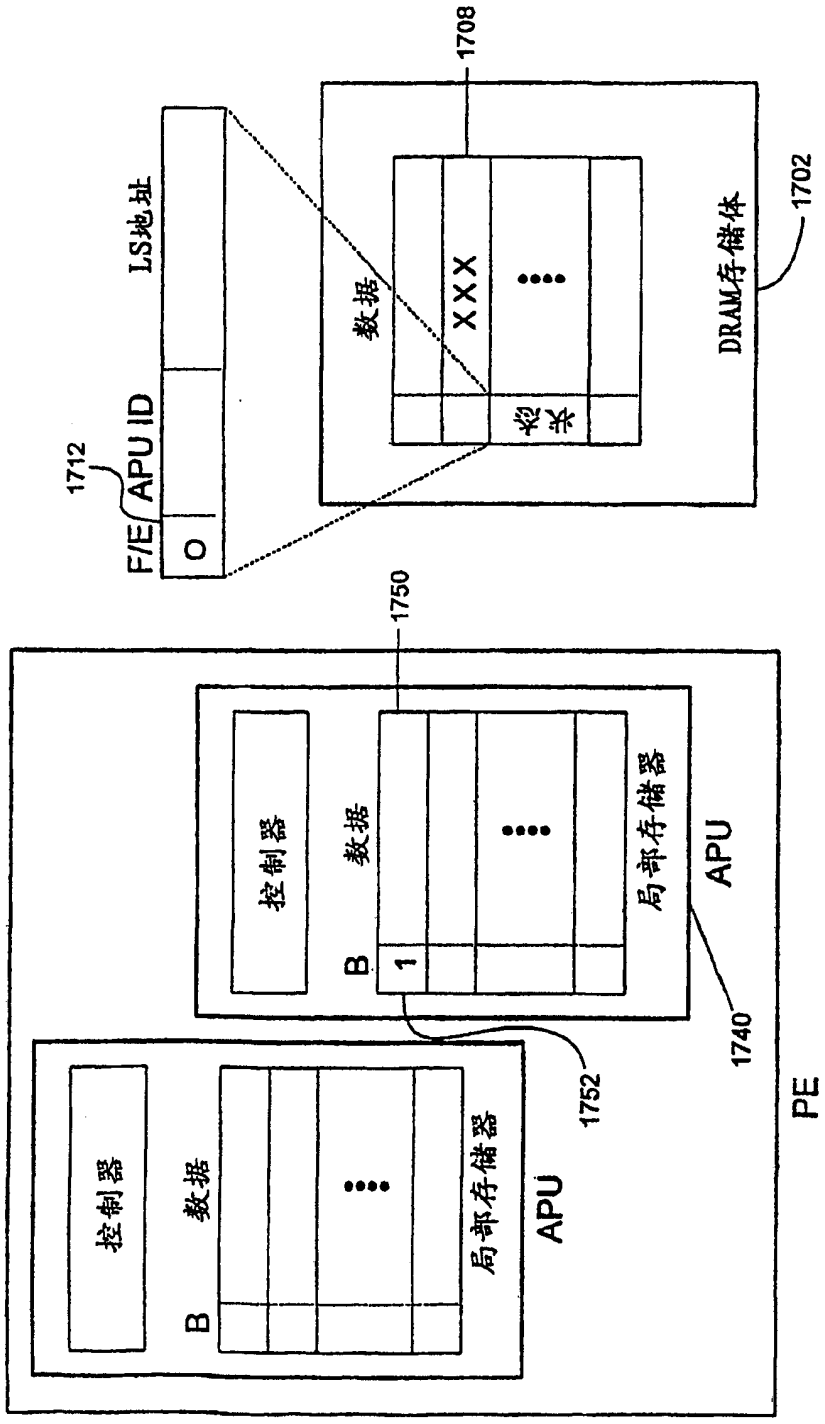

1708
LS地址
F/E
APU ID
1712
O
数据
XXX
DRAM存储体
1702
1750
控制器
数据
B
1
局部存储器
APU
1740
1752
控制器
数据
B
局部存储器
APU
PE

图17F

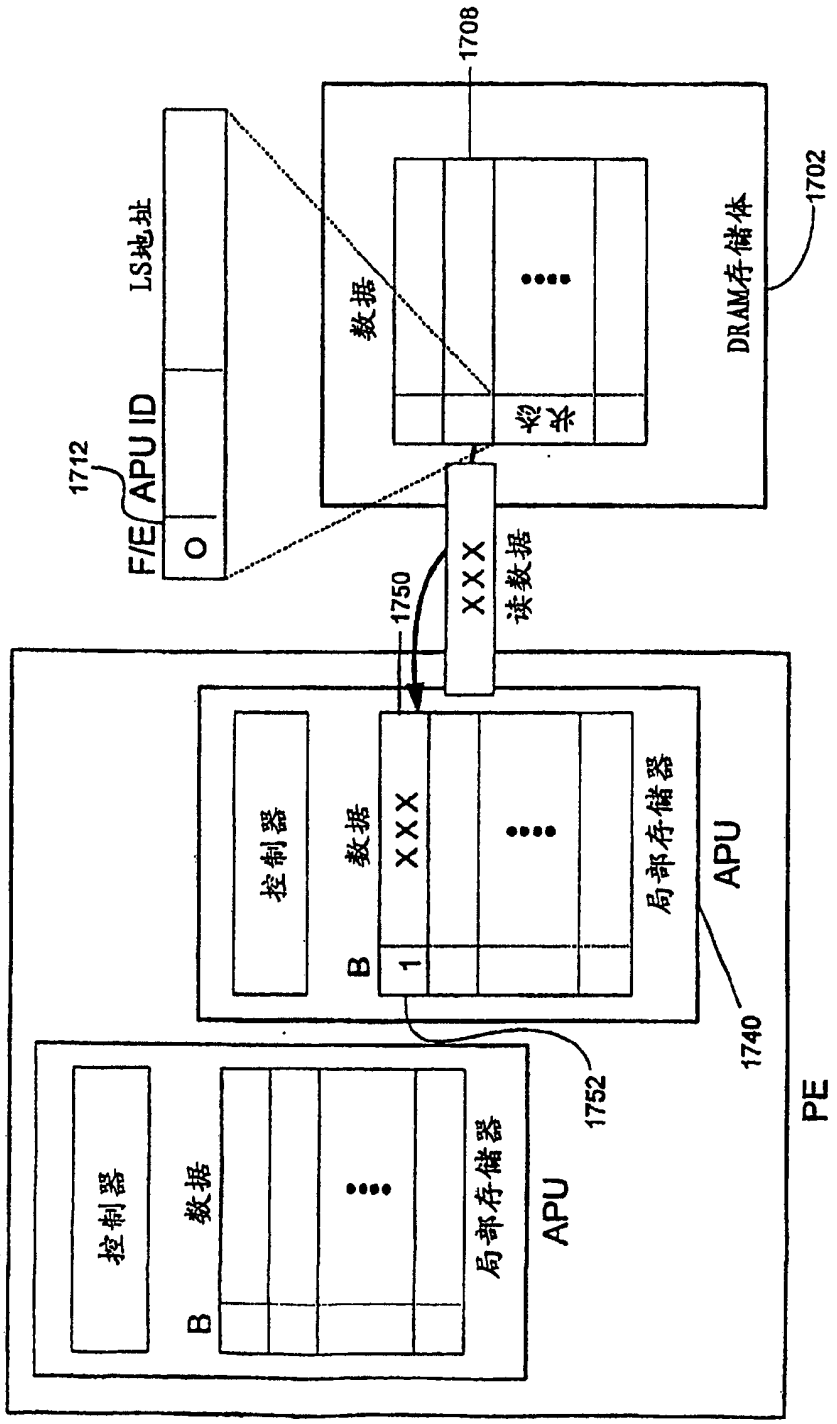

1708
1712
F/E
APU ID
LS地址
O
数据
DRAM存储体
1702
XXX
读数据
1750
控制器
B
数据
1
XXX
局部存储器
APU
1740
1752
控制器
B
数据
局部存储器
APU
PE

图17G

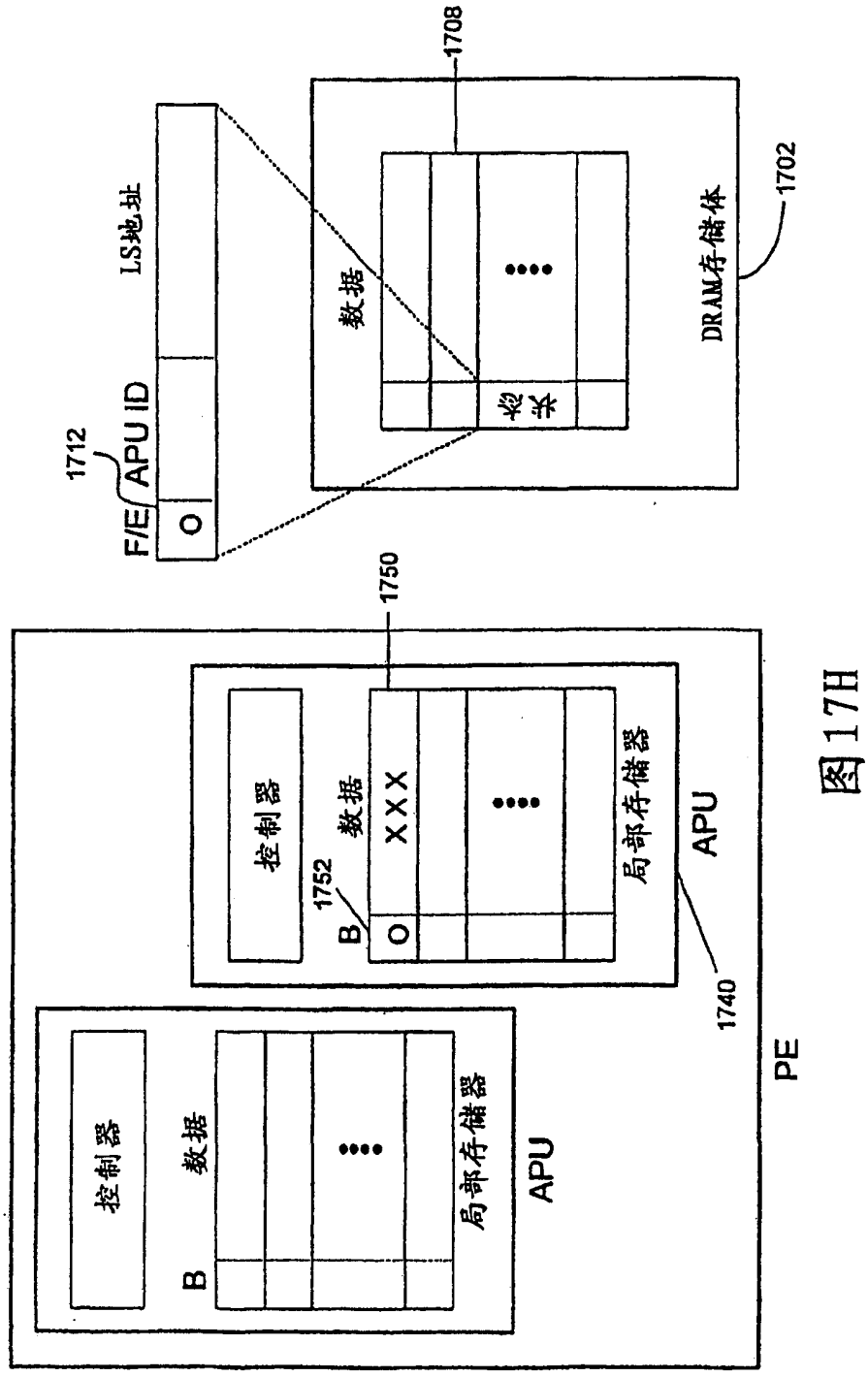

1708
LS地址
1712
F/E
APU ID
O
数据
必杀
DRAM存储体
1702
1750
控制器
1752
B
数据
O
XXX
局部存储器
APU
1740
控制器
B
数据
局部存储器
APU
PE

图17H

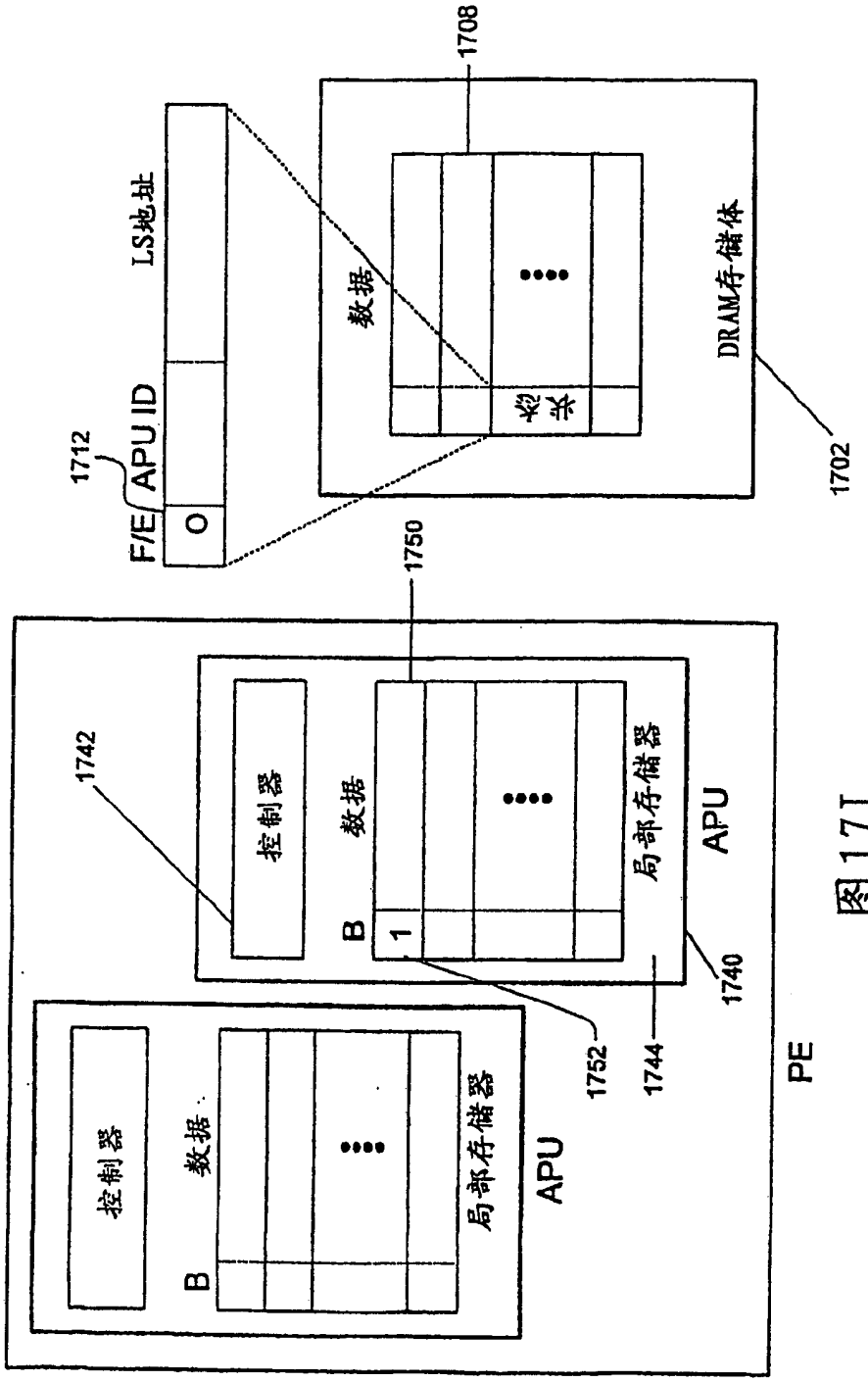

1708
LS地址
APU ID
F/E
O
1712
数据
DRAM存储体
1702
1750
控制器
1742
数据
B
1
局部存储器
APU
1740
1752
1744
控制器
数据
B
局部存储器
APU
PE

图17I

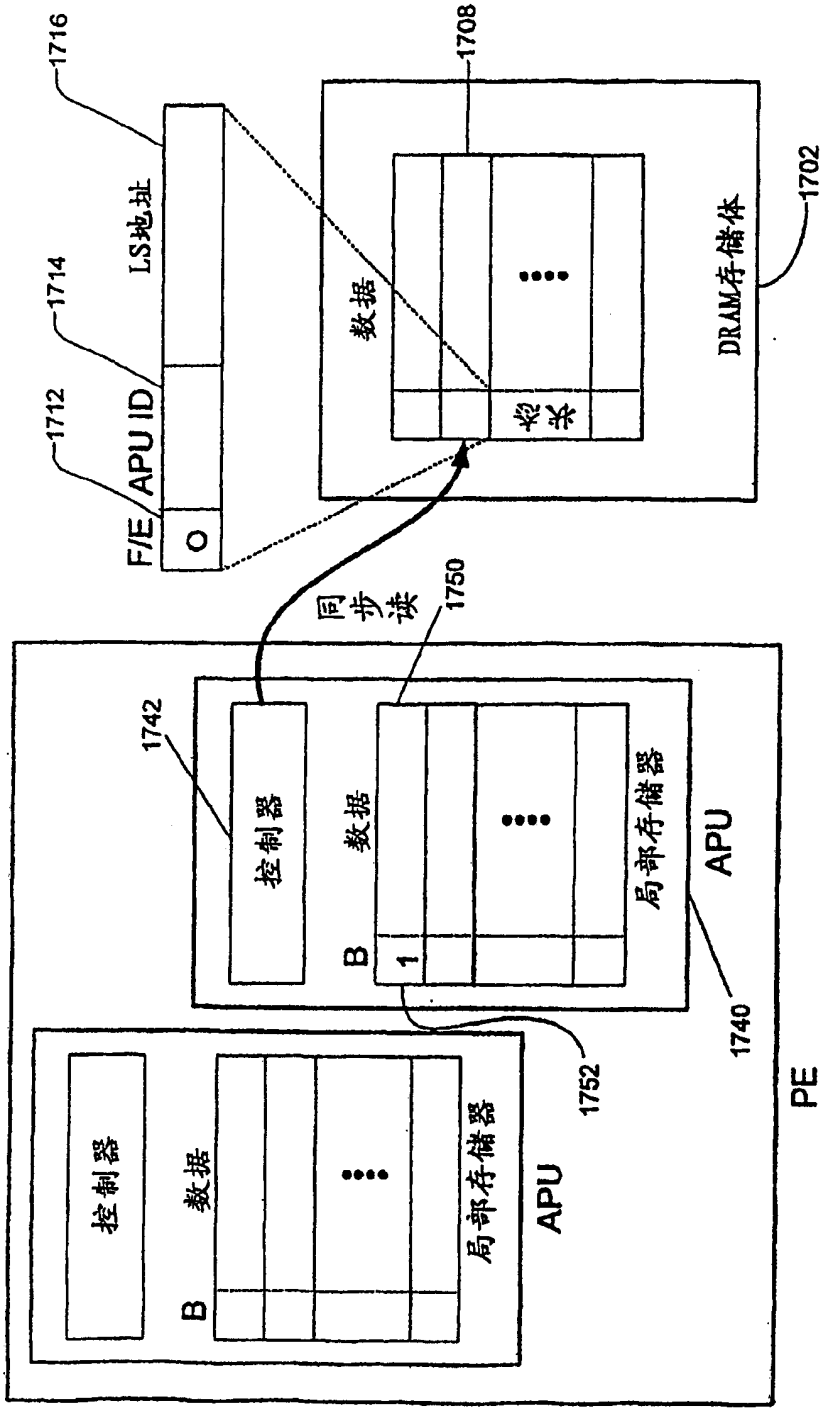

1716
LS地址
1714
APU ID
1712
F/E
O
1708
数据
DRAM存储体
1702
同步读
1750
1742
控制器
B
数据
1
局部存储器
APU
1740
1752
PE
控制器
B
数据
局部存储器
APU

图17J

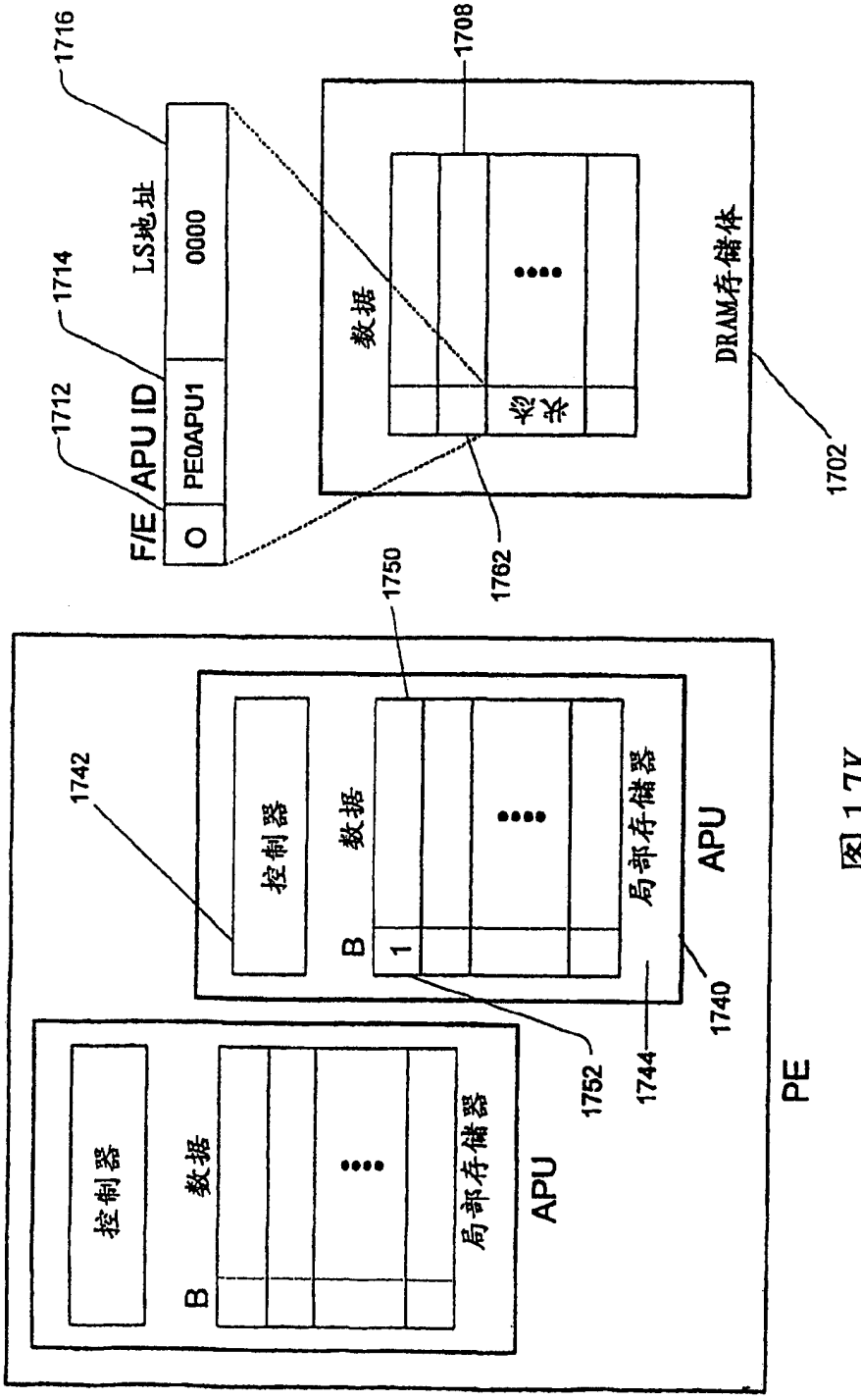

1716
1708
LS地址
0000
1714
APU ID
PE0APU1
1712
F/E
0
数据
DRAM存储体
1702
1762
1750
1742
控制器
数据
B
1
局部存储器
APU
1740
1752
1744
控制器
数据
B
局部存储器
APU
PE

图17K

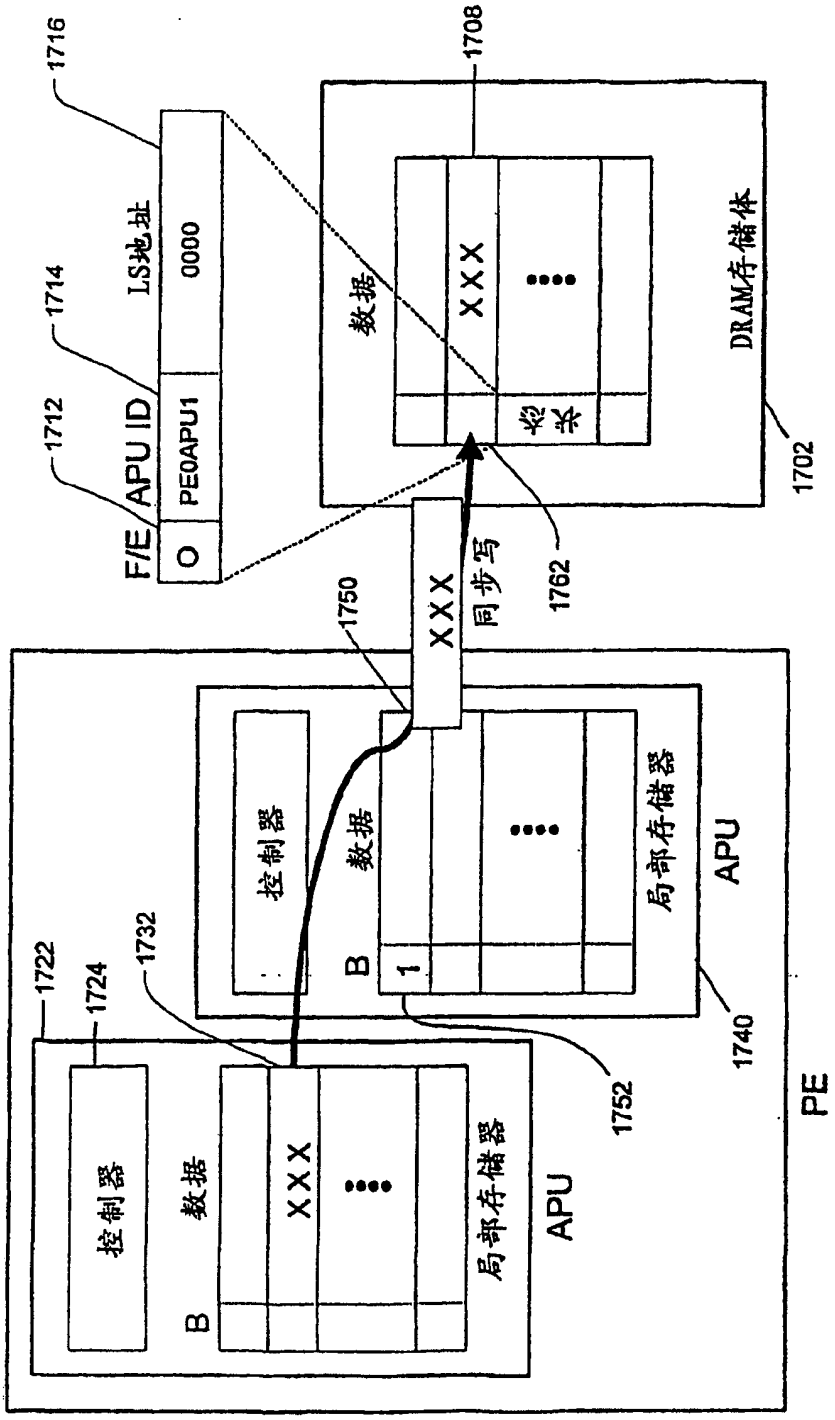

1716
1708
LS地址
0000
1714
APU ID
PE0APU1
1712
F/E
0
数据
XXX
DRAM存储体
1702
1750
XXX
同步写
1762
控制器
B
数据
1
局部存储器
APU
1732
1722
1724
1740
1752
PE

图17L

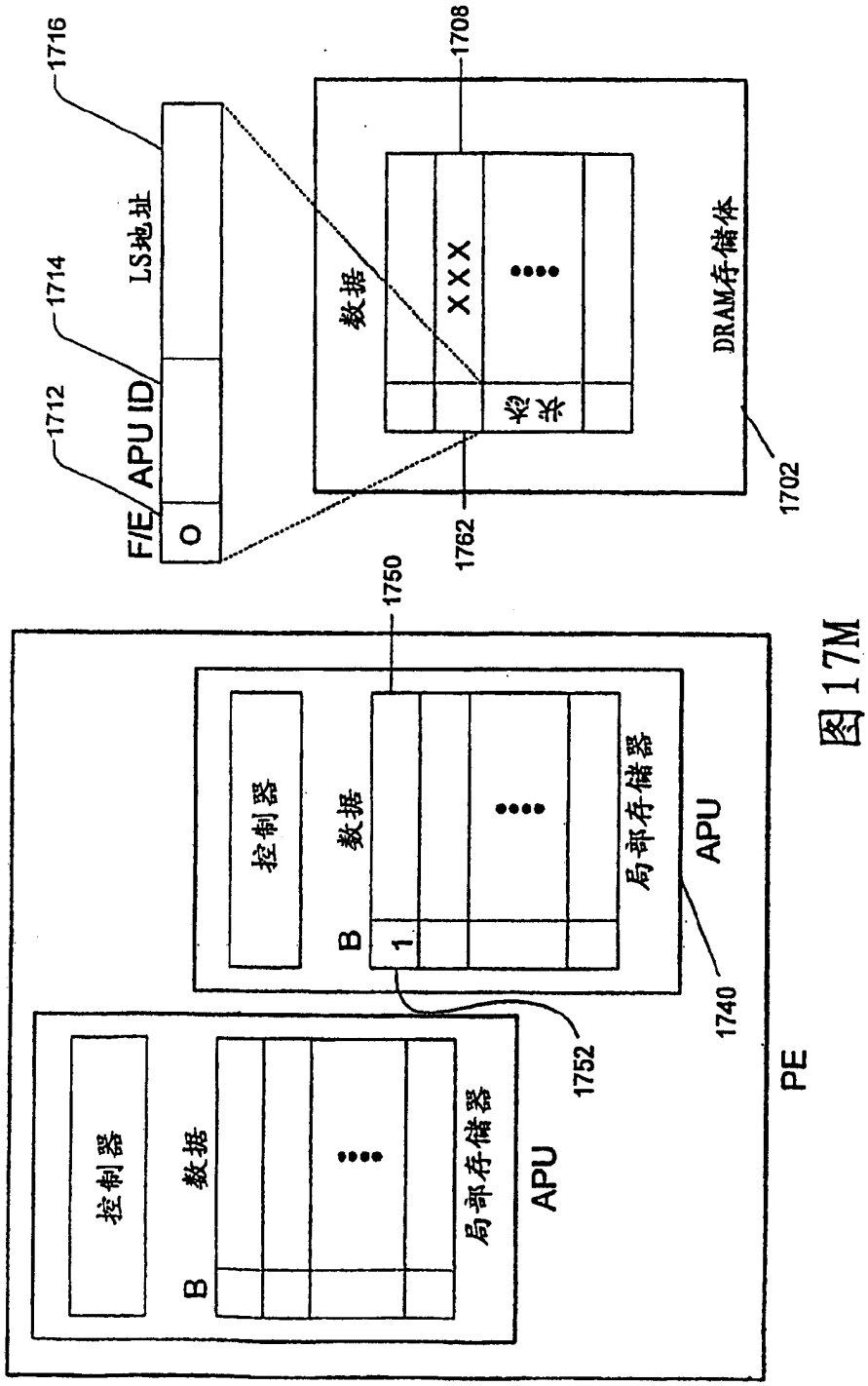

1716
1708
LS地址
1714
APU ID
1712
F/E
O
数据
XXX
DRAM存储体
1702
1762
1750
控制器
数据
B
1
局部存储器
APU
1740
1752
控制器
数据
B
局部存储器
APU
PE

图17M

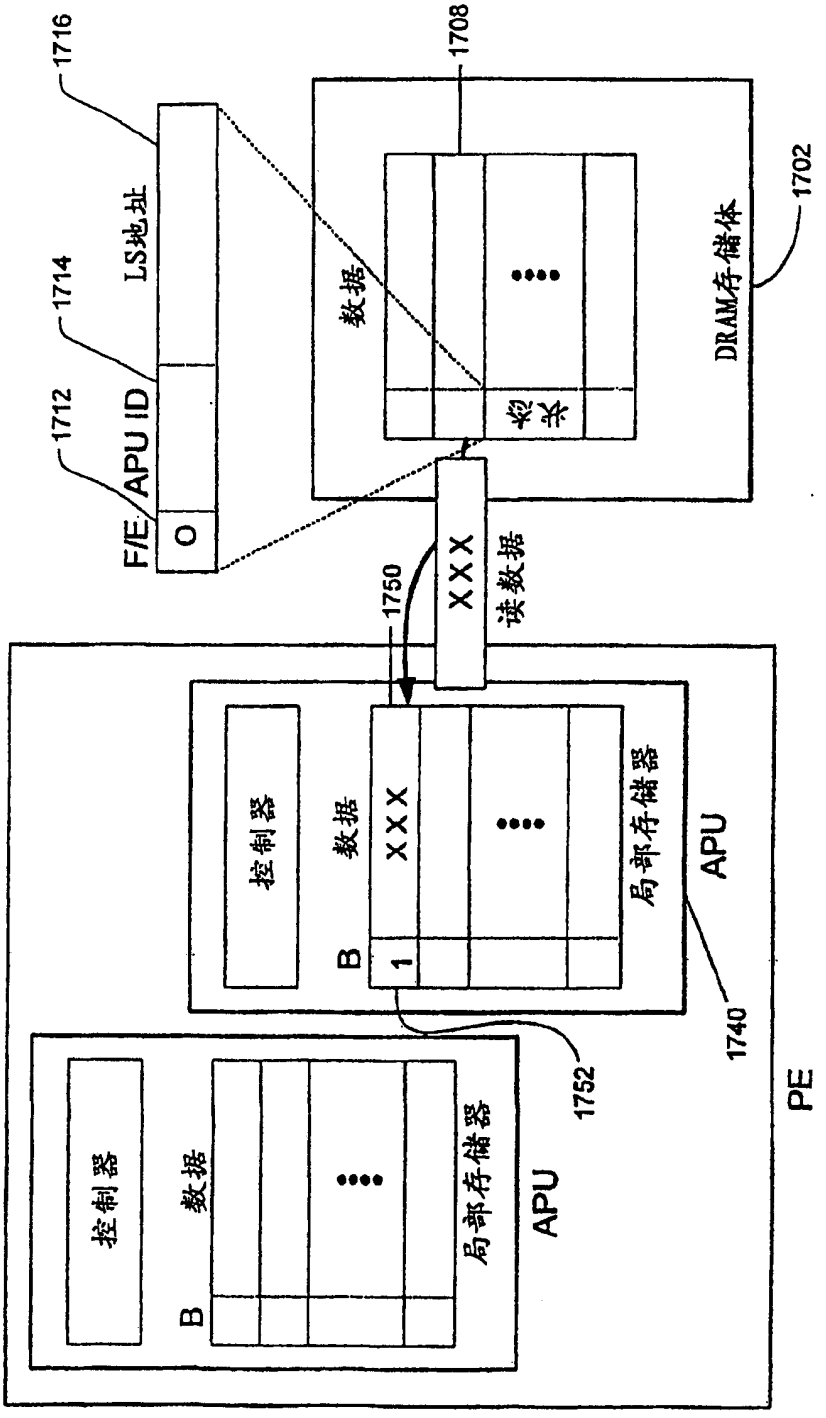

1716
LS地址
1714
APU ID
1712
F/E
0
1708
数据
DRAM存储体
1702
1750
XXX
读数据
控制器
数据
B
1
XXX
局部存储器
APU
1740
1752
控制器
数据
B
局部存储器
APU
PE

图17N

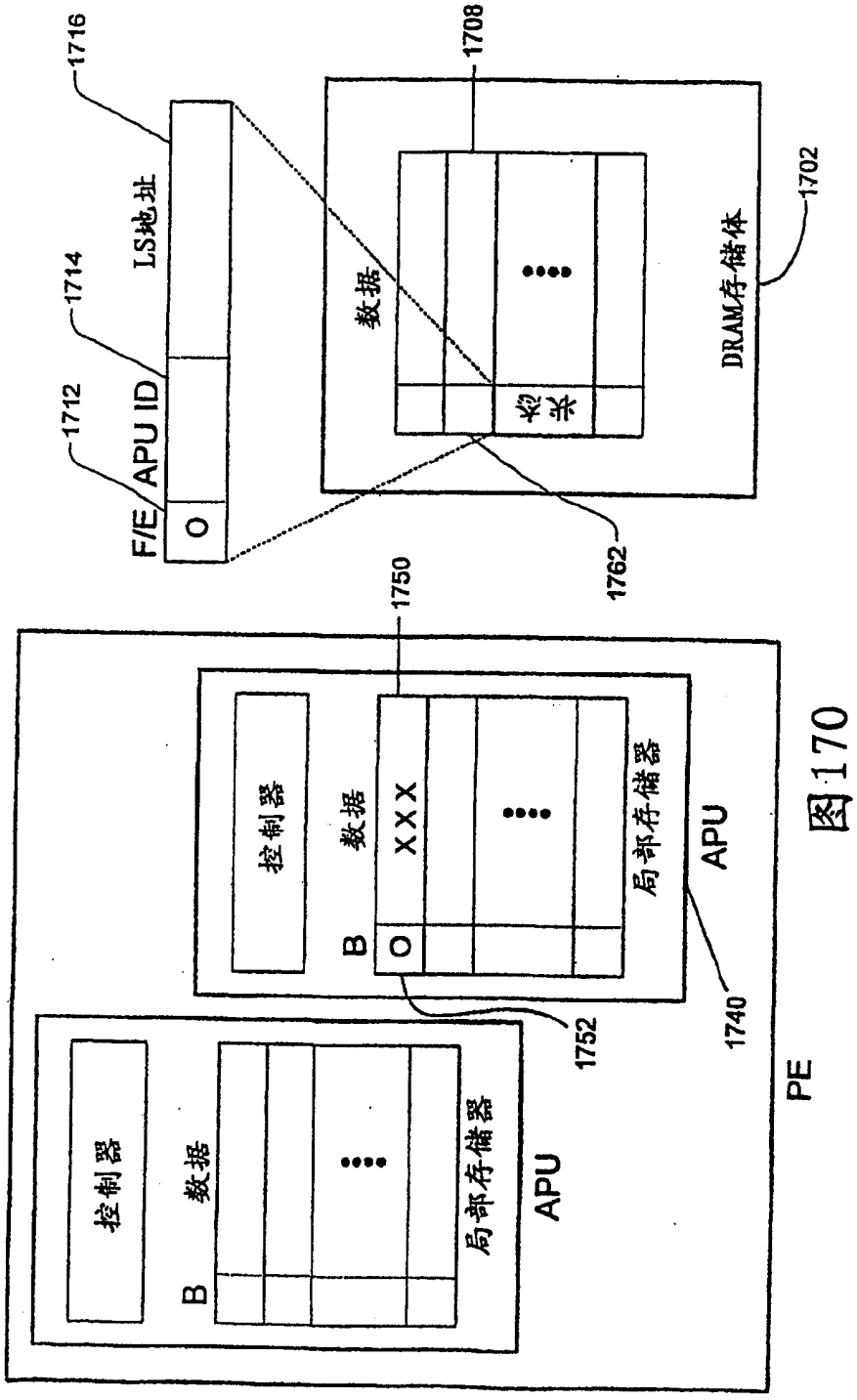

1716
1708
LS地址
1714
APU ID
1712
F/E
O
数据
DRAM存储体
1702
1762
1750
控制器
B
数据
O
XXX
局部存储器
APU
1752
1740
控制器
B
数据
局部存储器
APU
PE

图170

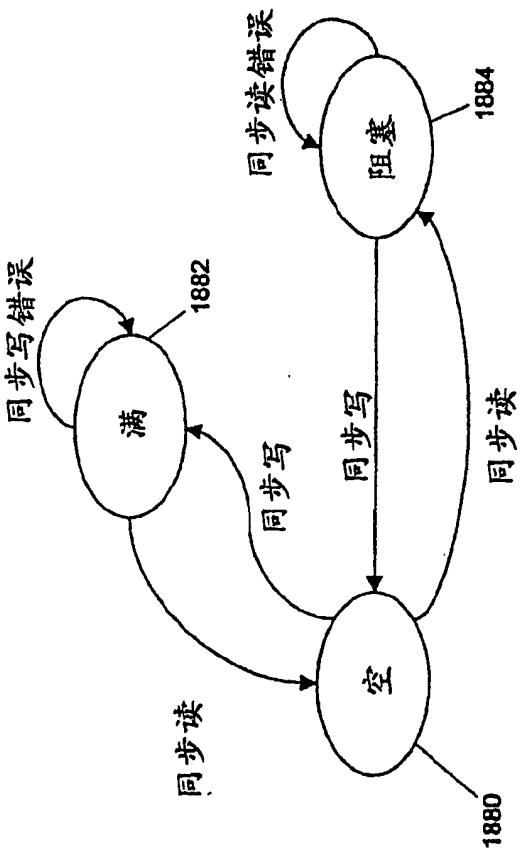



图18

关键字控制表

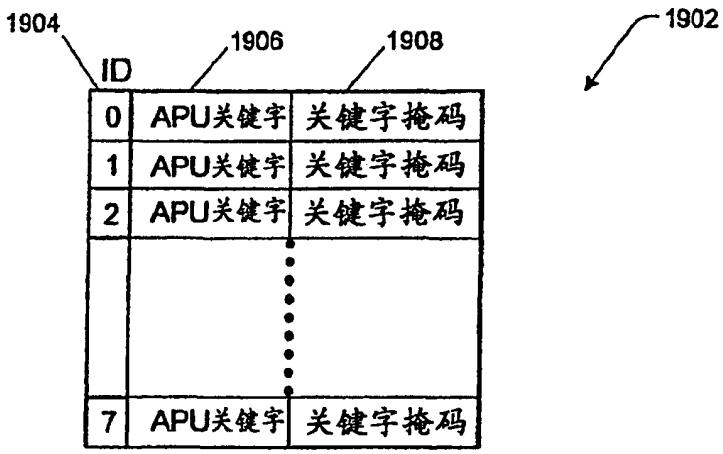

1904
ID
1906
1908
1902
0
APU关键字
关键字掩码
1
APU关键字
关键字掩码
2
APU关键字
关键字掩码
7
APU关键字
关键字掩码

图19

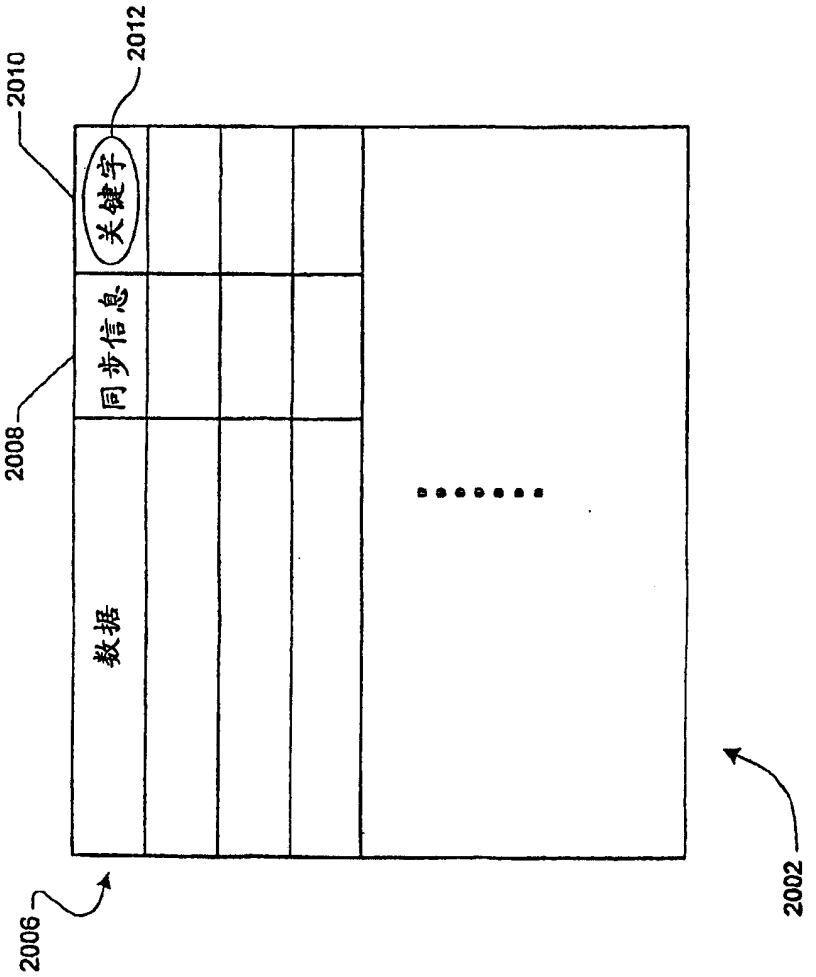



图20

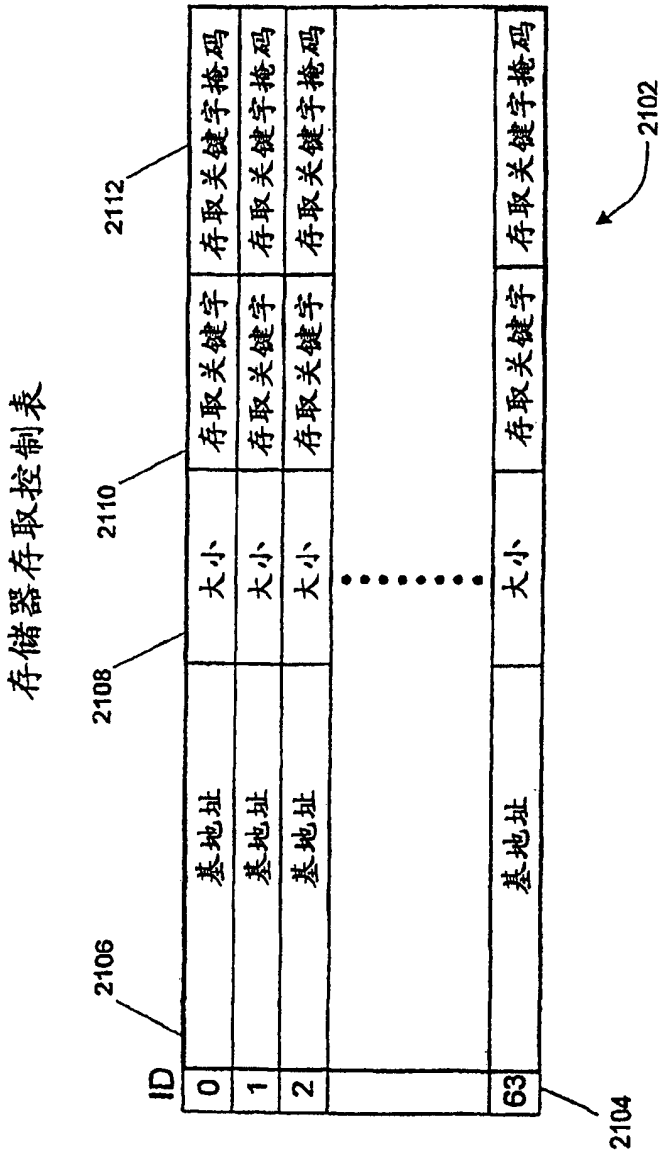

存储器存取控制表
ID
基地址
大小
存取关键字
存取关键字掩码
0
1
2
63
2106
2108
2110
2112
2102
2104

图21

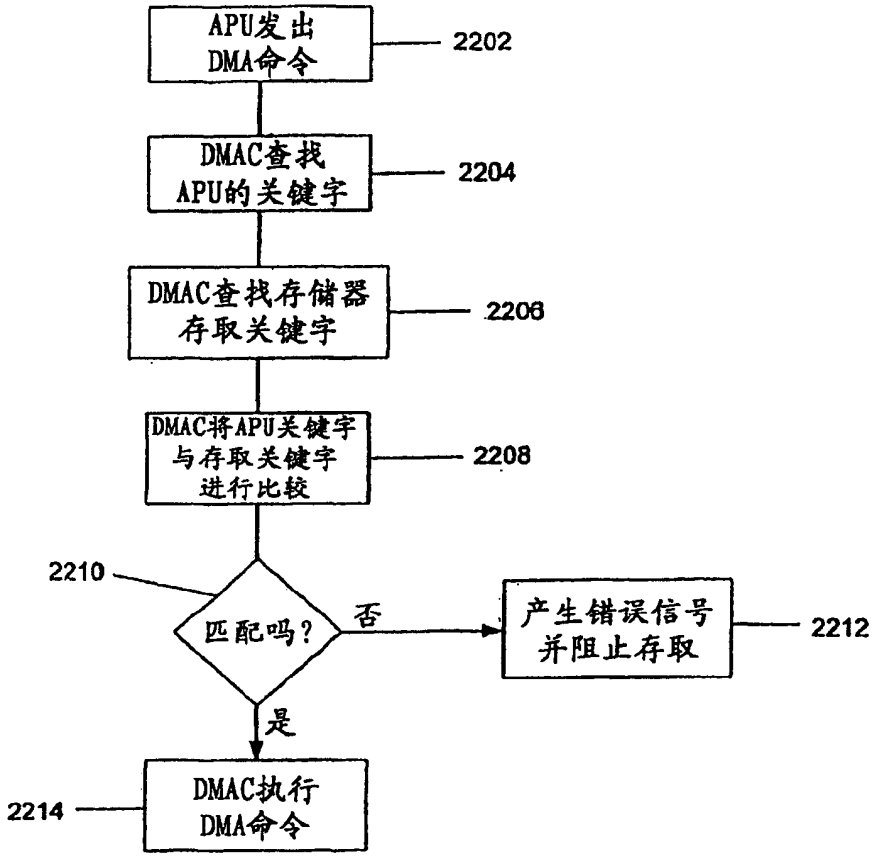

APU发出
DMA命令
2202
DMAC查找
APU的关键字
2204
DMAC查找存储器
存取关键字
2206
DMAC将APU关键字
与存取关键字
进行比较
2208
2210
匹配吗?
否
产生错误信号
并阻止存取
2212
是
DMAC执行
DMA命令
2214

图22

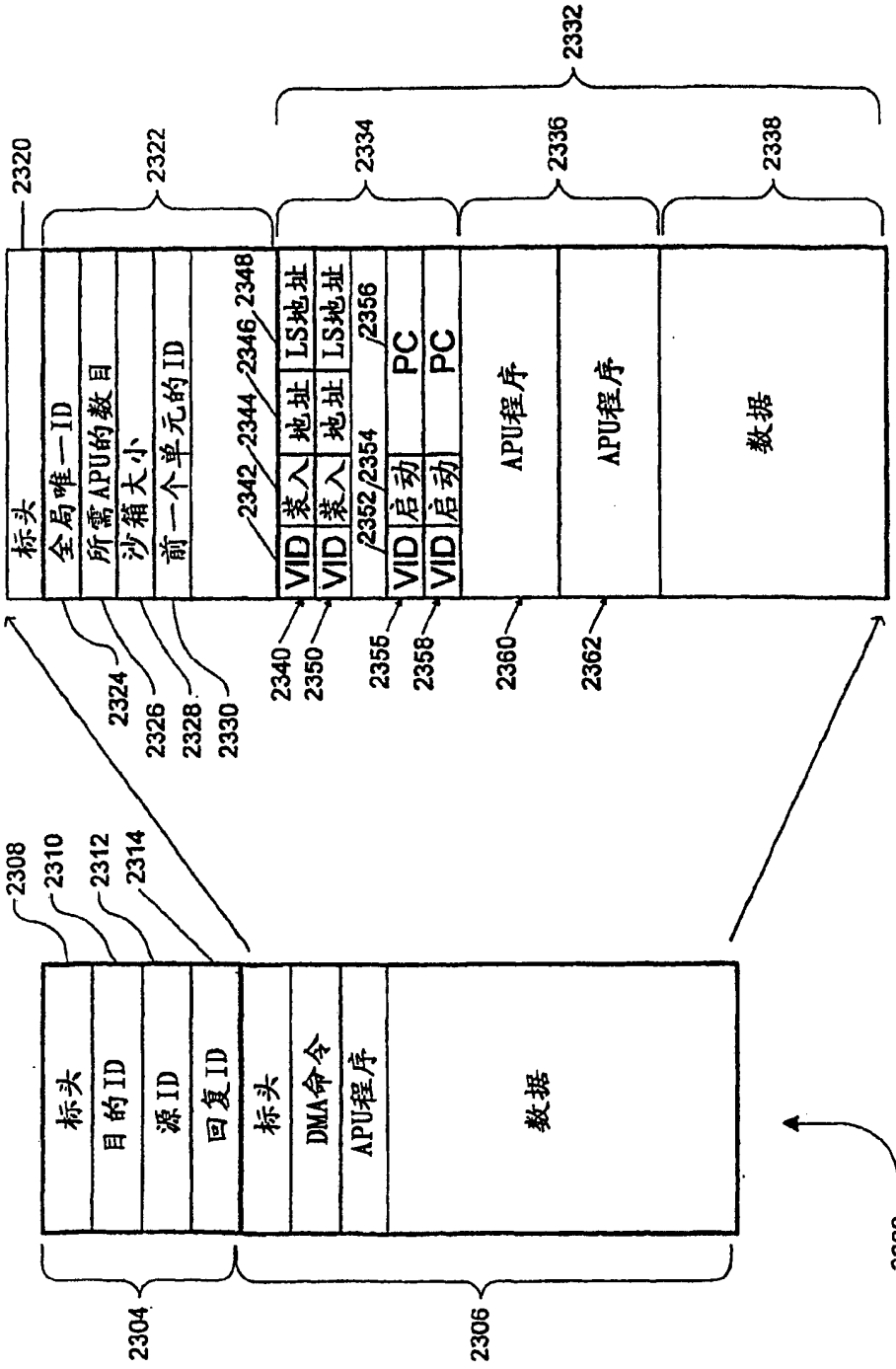

2320
2322
2332
2334
2336
2338
标头
全局唯一ID
所需APU的数目
沙箱大小
前一个单元的ID
2342
2344
2346
2348
VID
装入
地址
LS地址
2352
2354
2356
启动
PC
APU程序
数据
2324
2326
2328
2330
2340
2350
2355
2358
2360
2362
2308
2310
2312
2314
标头
目的ID
源ID
回复ID
DMA命令
APU程序
数据
2304
2306
2302

图23

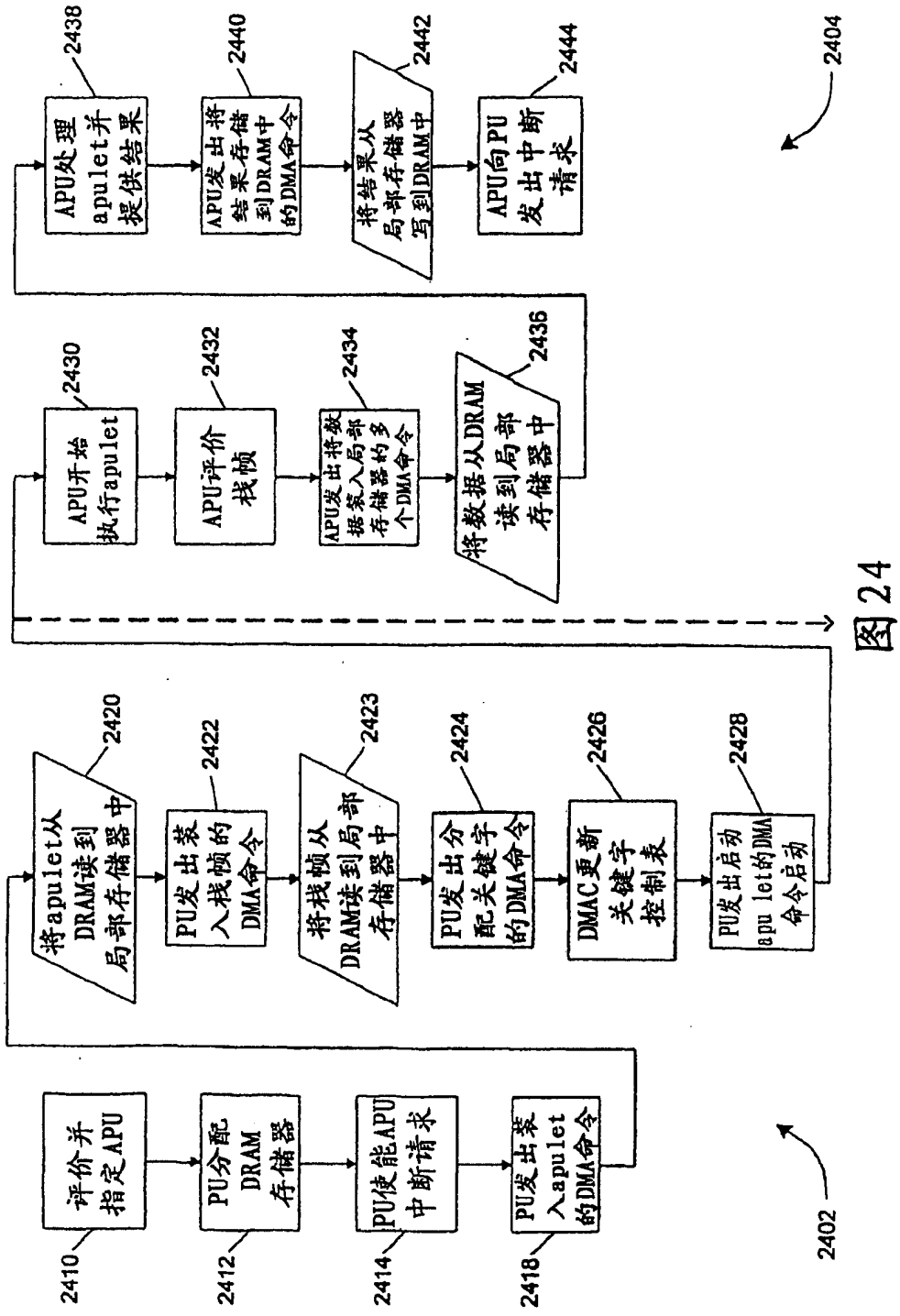

评价并指定APU
2410
PU分配DRAM存储器
2412
PU使能APU中断请求
2414
PU发出装入apulet的DMA命令
2418
将apulet从DRAM读到局部存储器中
2420
PU发出装入找帧的DMA命令
2422
将找帧从DRAM读到局部存储器中
2423
PU发出分配关键字的DMA命令
2424
DMAC更新关键字控制表
2426
PU发出启动apulet的DMA命令启动
2428
2402
APU开始执行apulet
2430
APU评价找帧
2432
APU发出将数据入局部存储器的多个DMA命令
2434
将数据从DRAM读到局部存储器中
2436
APU处理apulet并提供结果
2438
APU发出将结果存储到DRAM中的DMA命令
2440
将结果从局部存储器写到DRAM中
2442
APU向PU发出中断请求
2444
2404

图 24

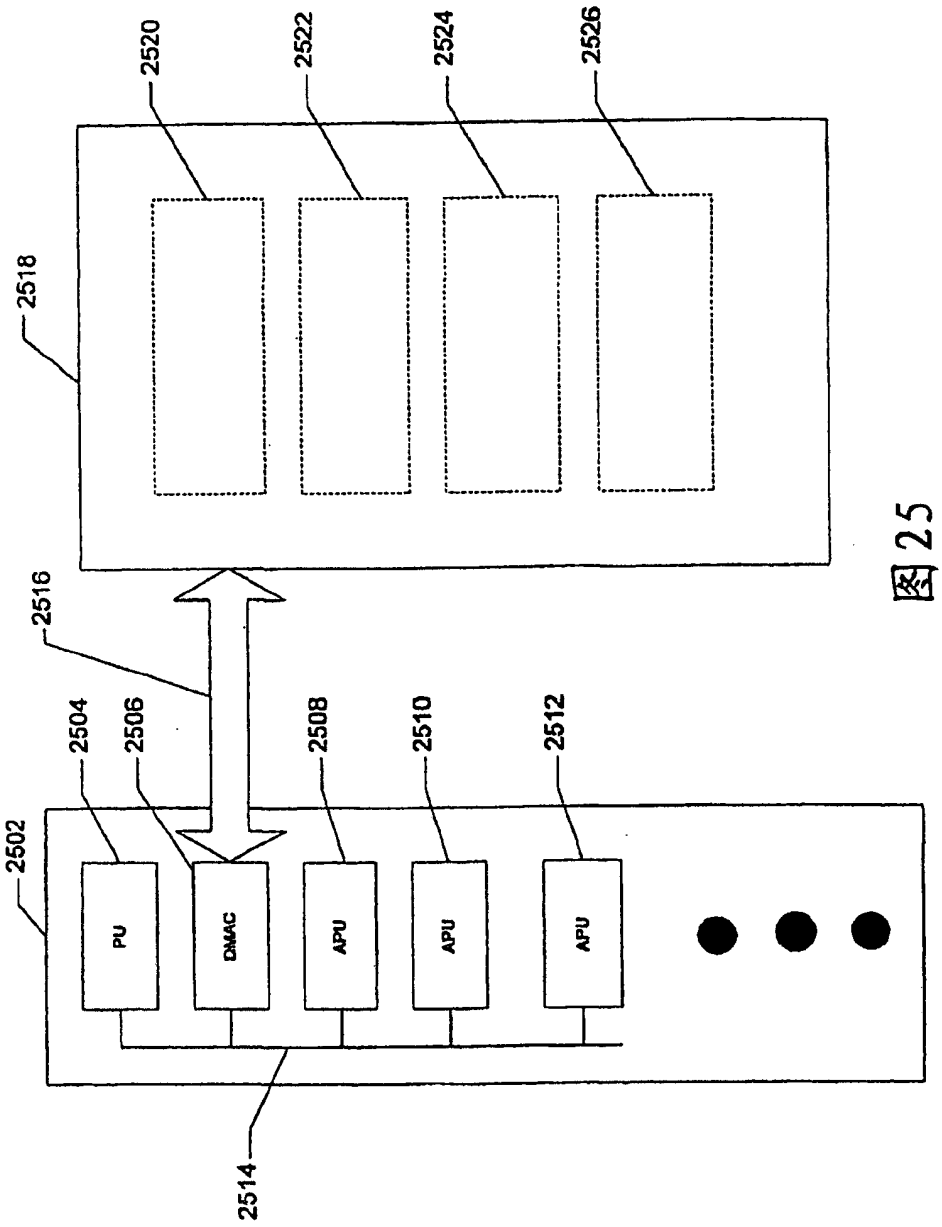

2520
2522
2524
2526
2518
2516
2504
2506
2508
2510
2512
2502
PU
DMAC
APU
APU
APU
2514

图25

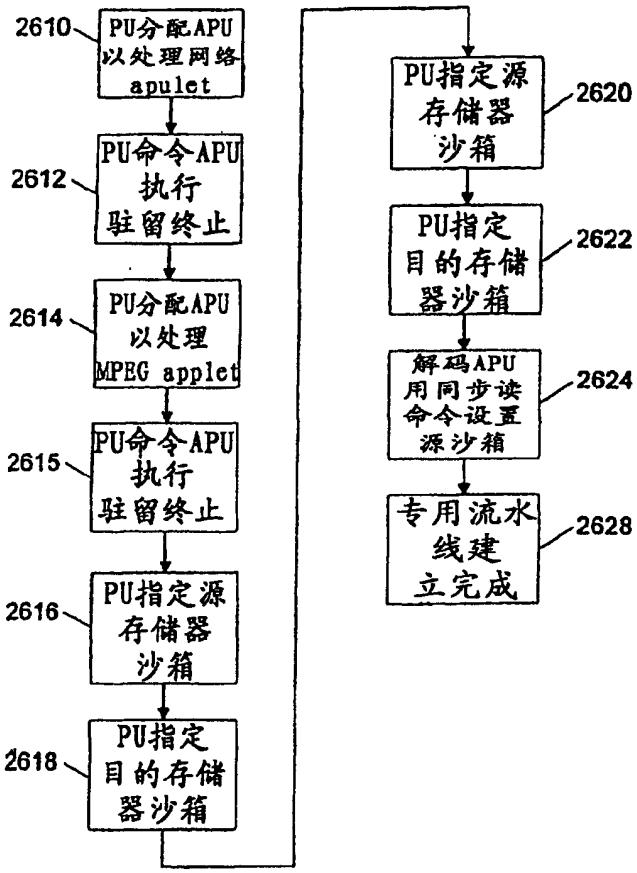

2610
PU分配APU以处理网络apulet
2612
PU命令APU执行驻留终止
2614
PU分配APU以处理MPEG applet
2615
PU命令APU执行驻留终止
2616
PU指定源存储器沙箱
2618
PU指定目的存储器沙箱
PU指定源存储器沙箱
2620
PU指定目的存储器沙箱
2622
解码APU用同步读命令设置源沙箱
2624
专用流水线建立完成
2628

图26A

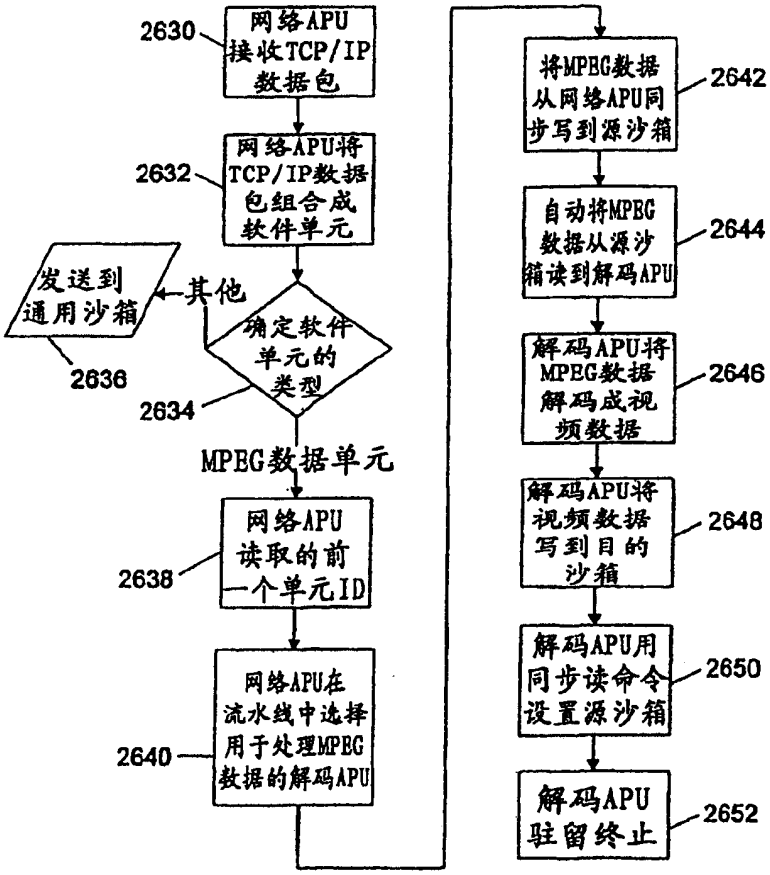

网络APU接收TCP/IP数据包
2630
网络APU将TCP/IP数据包组合成软件单元
2632
确定软件单元的类型
2634
其他
发送到通用沙箱
2636
MPEG数据单元
网络APU读取的前一个单元ID
2638
网络APU在流水线中选择用于处理MPEG数据的解码APU
2640
将MPEG数据从网络APU同步写到源沙箱
2642
自动将MPEG数据从源沙箱读到解码APU
2644
解码APU将MPEG数据解码成视频数据
2646
解码APU将视频数据写到目的沙箱
2648
解码APU用同步读命令设置源沙箱
2650
解码APU驻留终止
2652

图26B

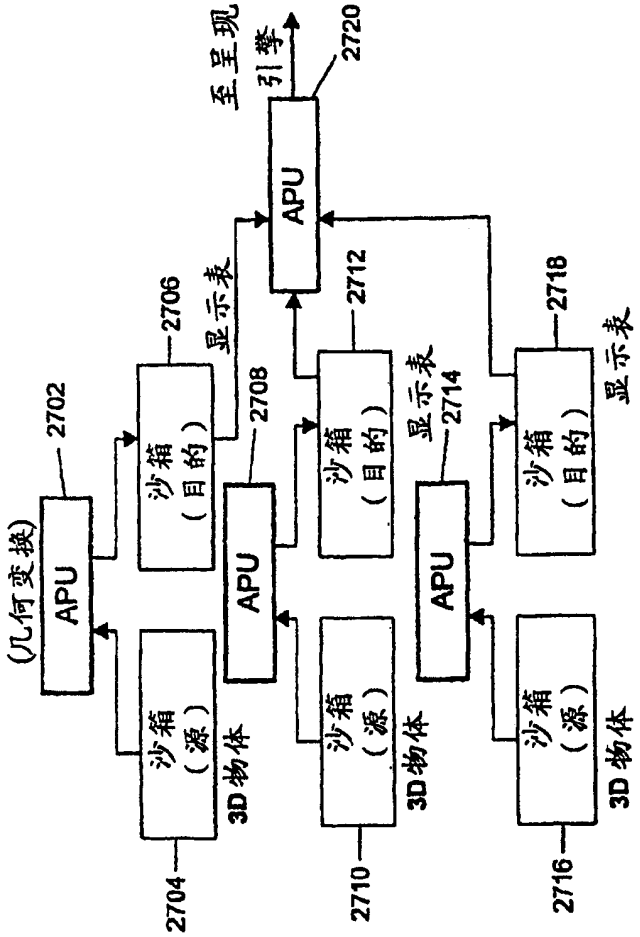

至呈现引擎
2720
APU
显示表
2706
2708
2712
2718
显示表
显示表
2714
(几何变换)
2702
APU
APU
APU
沙箱
(目的)
沙箱
(目的)
沙箱
(目的)
沙箱
(源)
沙箱
(源)
沙箱
(源)
3D物体
3D物体
3D物体
2704
2710
2716

图 27

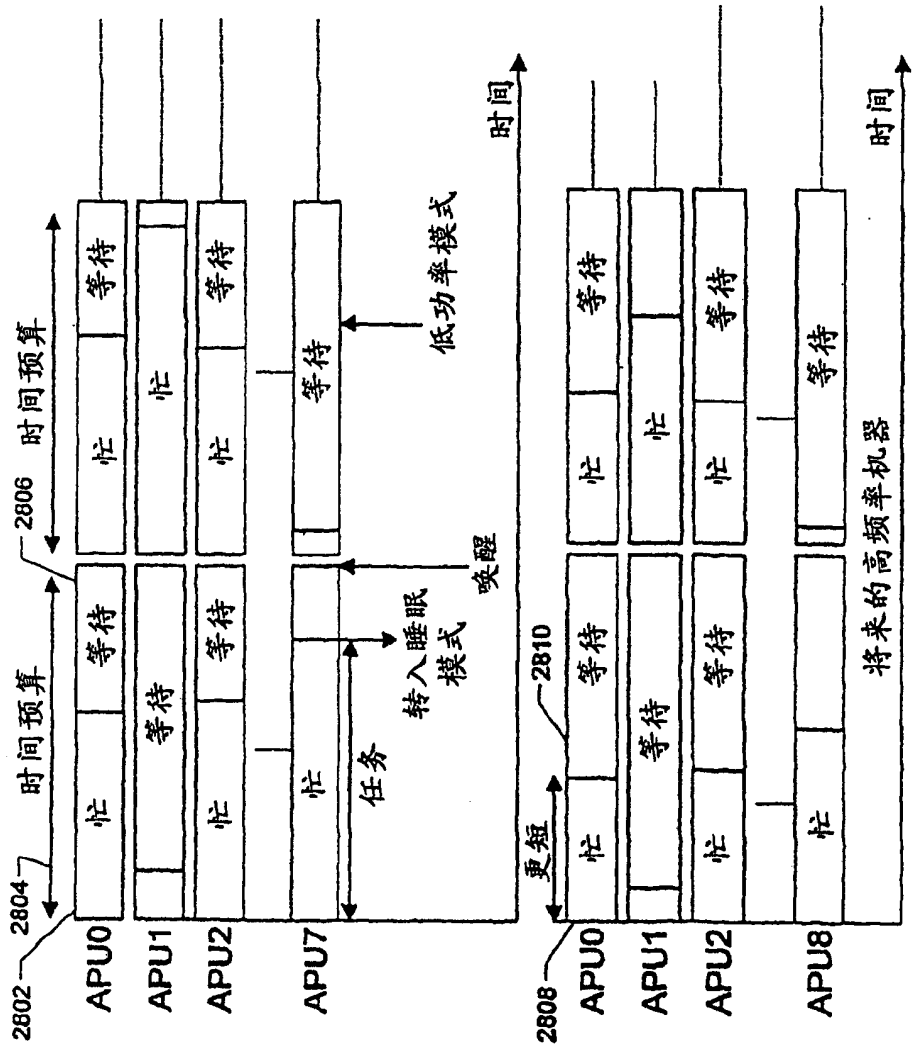

2802
2804
时间预算
2806
时间预算
APU0
APU1
APU2
APU7
忙
等待
任务
转入睡眠模式
唤醒
低功率模式
时间
2808
2810
更短
APU0
APU1
APU2
APU8
将来的高频率机器
时间

图 28