[19]中华人民共和国国家知识产权局

[51]Int.Cl[6]
G06F 12/10
G06F 9/46

[12]发明专利申请公开说明书

[21]申请号 98801271.5

[43]公开日 1999年12月1日

[11]公开号 CN 1237252A

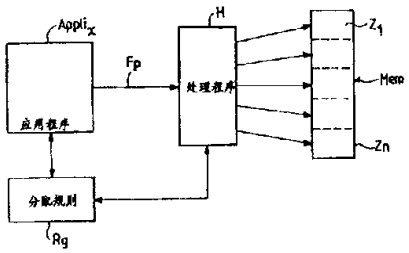
[22]申请日 98.8.26 [21]申请号 98801271.5
[30]优先权
[32]97.9.4 [33]FR [31]97/11025
[32]98.6.25 [33]FR [31]98/08058
[86]国际申请 PCT/FR98/01855 98.8.26
[87]国际公布 WO99/12099 法 99.3.11
[85]进入国家阶段日期 99.4.30
[71]申请人 布尔有限公司
地址 法国卢维耶尼
[72]发明人 希里·伯达茨 帕特里斯·罗曼德
珍－多米尼克·索雷丝
[74]专利代理机构 中国国际贸易促进委员会专利商标事务所
代理人 马 浩

权利要求书5页 说明书19页 附图页数5页

[54]发明名称 用于在多处理器数据处理系统中分配存储器的方法

[57]摘要

本发明涉及一种用于在一个多处理器的数据处理系统中分配物理存储单元的方法，该多处理器数据处理系统包括一个分布在多个模块中的非均匀存取存储器单元(Mem)。软件应用程序($Appli_x$)与一组预先定义的存储器分配规则(Rg)相连接。当一个地址对应表中不存在与一个虚拟地址相关的项时，就产生一个页面错误(Fp)。依据一个作为应用程序($Appli_x$)简要表和页面错误类型(Fp)的函数的预定规则，执行物理存储单元(Mem)的分配。在实施例的一个最佳变形例中。存储器以段的形式进行组织并且段被细分成虚拟地址区域。这些区域可与一个专用存储器分配策略相关联。在相反的情况下，为通用的段策略。



ISSN1008-4274

权 利 要 求 书

1.一种用于通过映射与一个给定的应用程序（ $Appli_x$ ）相关联的一个虚拟地址空间的至少一个具有连续存储地址的区域以分配物理存储单元的方法，该应用程序（ $Appli_x$ ）运行于一个数据处理系统（ 1'）中，所述数据处理系统（ 1'）包括一个非均匀存取存储器单元（ Mem ）并利用多种类型虚拟存储器（ tyM_1~tyM_6 ），通过扫描一个地址对应表来执行所述映射，其特征在于所述方法包括将所述给定的软件应用程序（ $Appli_x$ ）与从一组预先定义的规则中选定的存储器分配规则（ Rg ）相连接的步骤，并且其中，当所述地址对应表中不包含关于与所述给定软件应用程序（ $Appli_x$ ）相关联的一个虚拟地址空间的连续存储器地址区域的项时，所述方法包括生成一个异常信息的步骤和依据所述存储器分配规则（ Rg ）之一分配一个物理存储单元（ Z_1 - Z_n ）的后续步骤，规则的选择是通过给定软件应用程序（ $Appli_x$ ）所用的所述虚拟存储器类型（ tyM_1-tyM_6 ）的简要表（ Pa_x ）进行的。

2.如权利要求 1 所述的方法，其特征在于所述具有连续虚拟存储器地址的区域被细分成多个导致生成不同异常信息类型（ Fp ）的类，它包括确定所述异常信息类型（ Fp ）的附加步骤，其中所述存储器分配规则是所述简要表和所述异常类型（ Fp ）的组合函数。

3.如权利要求 1 或 2 所述的方法，其特征在于所述虚拟地址空间由多个段（ Sg_x， Sg_y ）组成，并且所述段（ Sg_x,Sg_y ）与所述存储器分配规则（ Rg ）相关联。

4.如权利要求 1 - 3 中任一所述的方法，其特征在于所述数据处理系统（ 1',1" ）由至少两个不同的模块（ Ma-Mc ）构成，每个模块包括至少一个处理器（ 10a-13a,10b-13b ）和一个所谓的本地物理存储器单元（ 14a,14b ），位于一个模块之外的存储器单元被称为远程存储器单元，所述预先定义的规则集（ Rg ）包括至少下述依据生成异常信息（ Fp ）的存储器分配专用规则：

第一规则，用于只在所述本地存储器单元中分配一个存储单元；

第二规则，用于在所述本地存储器单元和所述远程存储器单元中的通过片状分布的形式分配一个存储单元；

以及第三规则，用于参照所指定给所述模块的编号，在所述本地存储器单元或所述远程存储器单元中分配一个预先建立的固定存储单元。

5.如权利要求 4 所述的方法，其特征在于它包括至少一个被称为隐含规则的附加预定存储器分配规则，以便于在导致一种异常（ Fp ）的所述给定的软件应用程序（ $Appli_x$ ）不与任何一个所述专用规则相连接时，根据所述隐含规则执行存储器分配。

6.如权利要求 4 所述的方法，其特征在于所述给定的软件应用程序（ $Appli_x$ ）包括至少一个与其它应用程序共享的段，所述的所有模块（ Ma， Mb ）以基本相等的方式访问该段，依据所述第二规则来执行用于分配一个物理存储单元（ Mem ）的所述后续步骤，在第二规则中，通过分布在所述本地和远程存储器单元（ 14a,14b ）中来进行分配。

7.如权利要求 4 所述的方法，其特征在于所述给定的软件应用程序（ $Appli_x$ ）包括至少一个在一个所述模块（ Ma - Mc ）中被本地访问的工作存储器段，根据所述第一规则执行用于分配一个物理存储单元（ Mem)的所述后续步骤，分配只在所述本地存储器单元（ 14a 或 14b ）中进行。

8.如权利要求 4 - 7 中任一所述的方法，其特征在于所述段被细分成多个虚拟地址区域（ Ra_1-Ra_5 ），每个区域被分配给至少一个所述应用程序（ $Appli_x$ ），第一数字数据用于规定是否存在至少一个与一个专用存储器分配规则相关联的虚拟地址区域（ Ra_1-Ra_5 ），并且其中所述策略包括至少一个第二数字数据，用于规定所述存储器分配规则的性质和一个第三数字数据，由用于对一个所述模块（ Ma-Mc ）寻址的可选编号构成。

9.如权利要求 8 所述的方法，其特征在于它包括在发生与所述虚拟地址区域（ Ra_1-Ra_5 ）中的一个地址相关的异常（ Fp ）时，用于读取所述第一数字数据的步骤，和用于在所述异常被翻译成不与任一所述虚拟地址区域（ Ra_1-Ra_5 ）对应的存储器寻址时，依据规定（ ruling ）段（ Sg_x-Sg_y ）的存储器分配规则分配一个物理存储单元的后续步骤。

10.如权利要求 8 所述的方法，其特征在于所述第二数字数据规定了至少一种下述规则：

第一规则，用于只在所述本地存储器单元中分配一个存储单元；

第二规则，用于在所述本地和远程存储器单元中通过片状分布的形式分配一个存储单元；

第三规则，用于在所述本地或远程存储器单元中分配一个预先建立的固定存储单元；

以及所谓隐含规则的第四规则，用于依据所述数据处理系统的一个全局存储器分配策略来分配一个存储单元，所述第四规则是所述第一至第三规则中的一种。

11.如权利要求 10 所述的方法，其特征在于当所述第二数字数据规定了所述第二或第三存储器分配规则时，所述第三数据由一个模块编号（ Ma-Mc ）构成，根据该编号来确定必须执行所述存储器分配的模块（ Ma-Mc ）。

12.如权利要求 10 或 11 所述的方法，其特征在于所述数据处理系统包括一个用于描述段的表（ Scb ），该表包括与所述虚拟空间中的所述段（ Sg_x,Sg_y ）的个数相等的项，它包括一个用于将所述存储器分配策略记录到所述段描述表（ Scb ）中的初始步骤和一个用于将所述简要表（ Pa_x ）记录到被与其关联的所述应用程序（ $Appli_x$ ）所占用的虚拟存储空间中的初始步骤。

13.如权利要求 12 所述的方法，其特征在于每个所述存储器分配策略记录（ MP_y ）包括至少用于存储所述第一数字数据的第一字段，用于存储所述第二数字数据的第二字段和用于存储所述第三数字数据的第三字段。

14.如权利要求 13 所述的方法，其特征在于所述记录包括两个辅助字段，第一字段存储用于规定一个给定段（ Sg_x,Sg_y ）中的所述虚拟地址区域的低界地址的数字数据，第二字段存储用于规定该虚拟地址区域的高界地址的数字数据，并且其中包括下述阶段：

一个初级阶段，用于在所述应用程序的请求下，为每个包括至少一个与一专用存储器分配策略相关联的虚拟区域创建一个列表结构，该列

表结构由多个串接的元素（ LRa_1-LRa_2 ）构成，其个数等于所述段（ Sg_x,Sg_y ）中包含的所述虚拟地址区域（ Ra_1-Ra_5 ）的个数，每个元素用于存储所述存储器分配策略记录以及所述第一和第二辅助地址字段，并与虚拟地址区域（ Ra_1-Ra_5 ）之一相关联；

一个后续阶段，当在一个给定段（ Sg_x,Sg_y ）中的构成地址发生一个异常（ Fp ）时，至少包括下述步骤：

a/ 连续步骤，包括读出存储在列表结构（ LRa_1-LRa_Z ）的所述串接元素中的数字数据；

b/ 对每个列表元素，将导致异常（ Fp ）的所述地址与所述低界和高界地址进行比较的步骤；

c/ 在比较结果为肯定的情况下，读取所述第二和第三数据以便根据所述规则生成一个物理存储器分配指令的步骤，并在其编号已根据该规则被选定的模块（ Ma-Mc ）中或在缺少一个由所述第二数字数据规定的规则时执行该指令，以根据用于管理包括虚拟地址区域（ Ra_1-Ra_5 ）的段（ Sg_x,Sg_y ）的存储器分配规则来分配一个物理存储单元。

15.如权利要求 13 所述的方法，其特征在于所述记录包括两个辅助字段，第一字段存储一个用于规定一个给定段中的所述虚拟地址区域（ Ra_1-Ra_5 ）的低界地址的数字数据，而第二字段存储一个用于规定此虚拟地址区域（ Ra_1-Ra_5 ）的高界地址的数字数据，并且其中包括下述阶段：

包括下述步骤的第一初级辅助阶段：

1/ 第一步骤，用于将每个包括至少一个与一专用存储器分配策略相关联的虚拟区域的所述段（ Sg_y ）细分成具有相同固定长度的子段（ SS_1 - SS_N ）；以及

2/ 第二步骤，用于创建一个表（ Th ），该表中包括与子段（ SS_1 - SS_N ）具有相等个数的项（ e_1-e_N ）；

第二初级辅助阶段，用于为每个与每一项相关联的子段（ SS_1 - SS_N ）创建一个列表结构（ LRa_{12}-LRa_{22},LRa_{14},LRa_{17}-LRa_{37} ），该列表结构包括多个串接的元素，其元素个数等于子段（ SS_1 - SS_N ）中包含的所述虚拟地址区域（ Ra_{02}-Ra_{22} ）的个数，每个元素存储所述存储器分配

策略记录以及所述第一和第二辅助地址字段并与虚拟地址区域（ Ra_{02}-Ra_{22} ）之一相关联;

后续阶段，当在一给定子段（ SS_1 - SS_N ）中的一个构成地址上发生一个异常时，至少包括下述步骤;

a/ 读取表（ Th ）中与导致所述异常的地址相关联的所述项，并根据读出的内容确定所述子段（ SS_1 - SS_N)中是否包括一个由一个专用存储器分配策略管理的虚拟地址区域（ Ra_1-Ra_5 ），在判定结果为否定时，利用管理所述段（ Sg_y ）的规则;

b/ 在判定结果为肯定时，后续步骤包括读取存储在列表结构（ LRa_{12}-LRa_{22},LRa_{14},LRa_{17}-LRa_{37} ）的所述串接元素中的数字数据，所述元素与连接于导致所述异常的地址的项相关联;

c/ 对每个列素元素（ LRa_{12}-LRa_{22},LRa_{14},LRa_{17}-LRa_{37} ），将导致异常的所述地址与所述低界和高界地址进行比较;

d/ 在比较结果为肯定时，读取所述第二和第三数字数据，以根据所述规则生成一个物理存储器分配指令并在其编号根据该规则被选定的模块(Ma-Mc)中执行该指令。

16.如权利要求 15 所述的方法，其特征在于所述虚拟存储器地址区域（ Ra_{02}-Ra_{22})具有可变的长度，当所述长度大于具有定长的子段（ SS_1 - SS_N ）的长度时，所述虚拟存储器地址区域被细分成包括在子段（ SS_1 - SS_N ）中的子空间。

17.如权利要求 15 或 16 所述的方法，其特征在于所述段（ Sg_y ）具有 256MB 的连续虚拟空间，并且所述表（ Th ）包括 256 项（ e_1-e_N ），每一项对应于一个 1MB 的子段（ SS_1 - SS_N ）。

说　明　书

用于在多处理器数据处理系统中分配存储器的方法

本发明涉及一种用于在多处理器数据处理系统中分配存储器的方法，特别涉及一种用于分配具有非均匀存取能力的存储器的方法。

在本发明的范围内，术语“非均匀”是一个暂时的概念，如后面所述。同样，术语“一个存储器”表示广义的概念，它可以表示一个分布式存储器、一个存储器分层体系（例如由多个具有不同存取时间的存储器构成）或一组不同类型的存储器。

在数据处理领域内，通过增加处理器的组成个数可以提高计算机的效力是众所周知的，一种名为“ SMP ”（来源于英文词 Symmetrical Multiprocessor)的公知计算机允许不同的处理器在同一机器内通过一条系统总线对称地访问其存储器。这是具有均匀存取存储器的计算机，因为对所有的被存取数据来说，对存储器的访问时间基本相同。

为此，这种体系结构被称为“ UMA ”（来源于英文词“ Uniform Memory Access ”）。

附属于本发明说明书的图 1 简要地示出了“ UMA ”型体系结构的示例。

此后被称为“ SMP ”模块的数据处理系统 1 包括特定数量的中央处理单元或处理器，或依据英文中的专业术语的“ CPU ”。图 1 所示的例子中示出了四个中央处理单元: 10 至 13。与这些中央处理单元 10 至 13 相关联的是一个主存储器 14，这四个中央处理单元均可对该主存储器进行访问。

由于从局部上说，所有的访问都发生模块 1 中，并且如果全部可用存储空间在访问时间上是均匀分配的（这构成了最初的设想，因为这是一个“ UMA ”体系结构），所以无论中央处理器 10 到 13 中的哪一个发送了一个请求，访问时间都保持基本相等。

虽然图 1 中只示出了四个中央处理器 10 至 13，但是应当清楚中央

处理器的个数是可以任意选择的。可以增加或减少中央处理器的个数。但是，这种类型的计算机的性能曲线不会随着处理器的个数呈线性增长。增加的处理器导致系统将更多的时间消耗在对其可用于正在运行的应用程序的资源的可访问性上。这样考虑的结果是：当处理器个数超过一个最佳值，通常被估计为 4 个左右时，将降低计算机的性能曲线。现有技术中已对此问题提供了不同的解决方案。

一种为公众所知的解决方案是将多台计算机分成群集以使它们彼此之间通过一个网络进行通信。每台计算机具有最佳个数的处理器，例如四个，以及自己的操作系统。每当该机器对保存在另一台计算机中的数据执行一次操作时，就与这另一台计算机建立一次通信。这些通信所需要的时间和对相同的数据进行操作的需求导致了大量（ high-volume ）应用程序，例如多次通信的分布式应用程序中出现的等待时间问题。等待时间是用于分离发送一个访问存储器的请求的瞬间和接收对该请求的响应的瞬间的时间。

另一种公知的解决方案是利用具有 NUMA 类型体系结构的计算机（来源于英文词 Non-uniform Memory Access ）。这些是带有非均匀访问存储器的计算机，因为对存储器的访问时间随着被存取数据的位置而变化。一台“ NUMA ”型计算机由多个模块构成，每个模块包括最佳个数的处理器和计算机中整个存储器的物理部分。这种类型的计算机具有非均匀存储器访问的能力，因为对一个模块来说，与访问与其它模块共享的部分相比，访问不与其它模块共享的存储器物理部分要更加容易和快捷。尽管每个模块都具有连接其处理器和其物理存储器的专用系统总线，但公用于所有模块的操作系统则允许将所有的专用系统总线看作该机器的一条唯一的系统总线。一个逻辑地址为一个模块的给定物理存储单元指定一个驻留位置。对一个特定的处理器来说，对实际上位于该处理器所在的模块中的本地存储器部分的访问不同于对实际上位于该处理器所在的模块之外的一个或多个模块中的远程存储器部分的访问。

附属于本发明说明书的图 2 简略地示出了这种体系结构，即“ NUMA ”体系结构的一个示例。为了简化附图，假设数据处理系统 1’只包括两个上述“ SMP ”类型的模块“ Ma 和 Mb，并且这两个模块是

相同的。但是应当理解数据处理系统1'可以包括更多个模块并且模块Ma和Mb可以是不相同的（尤其在中央处理器的数量上）。

模块Ma包括四个中央处理器10a－13a，和一个主存储器14a。同样，模块Mb也包括四个中央处理器10b－13b以及一个主存储器14b。两个主存储器14a和14b（更一般地为n个主存储器）分别借助于所谓的“连接”2，一般通过所谓的远程高速缓冲存储器15a和15b进行彼此之间的通信。连接2并非对应于简单的物理连接，而是包括各种无需在此进行详述的标准电子电路（控制电路、接口电路等）。

在这种类型的体系结构中，很容易理解如果一个应用程序正在模块Ma中运行，则对“近端”存储器14a（本地存取）的存取时间，先验的少于对位于模块Mb中的“远端”存储器14b的存取时间，而与涉及哪一个中央处理器10a－13a无关。当数据被物理存储在另一模块中时，通过连接2是非常必要的，这实质上增加了传送时间。

在新式的数据处理系统中，在一个虚拟存储空间的基础上执行为一个给定的应用程序分配存储器。这种分配是在操作系统或“OS”的控制下被执行的。然后在虚拟存储空间和物理存储器之间建立动态的对应关系。为此，通常利用地址对应表。按照通用的英文表达方式，将其称为动态“映射”（mapping）。已经提出了各种类型的存储格局，包括通过区域或通过段进行组织。为了解释这些概念而不以任何方式限定本发明范围，下面将描述一种“段”型格局。实际上，一个段就是具有连续的虚拟地址，固定和预定长度的空间。

更准确地，在现有技术中，依据所有应用程序通用的规则来执行上述动态对应关系或“映射”，而与它们的类型无关，无需考虑物理存储器的位置。实际上，如果一个进程试图访问一个虚拟地址并在地址对应表中没有找到该项，则产生一个异常信息，该异常信息是通过“UNIX”（注册商标）术语中的页面错误检测而形式化的。术语“页”可被更通俗地定义为“连续地址构成的范围”。一页构成段的子部分。但是，为了简化起见，下面将使用术语“页”。在页面错误检测之后，一个被称为处理程序（handler）的设备依据上述通用规则来分配物理存储器。这种简单的分配方法完全适用于上述“UMA”类型的标准“SMP”机器，

因为对存储器的平均存取时间是均匀的。

另一方面，对于参照图 2 所述的“ NUMA ”类型的体系结构，其存取时间不再相等，就需要一种能够使对系统性能的负影响减至最小的存储器分配方法。

在现有技术中，已经提出了用于实现此目的的方法。例如，提出了修改存储器分配规则以获得最佳效果，但是一旦进行了修改，则这些规则对于所有应用程序来说都是一样的。而且这种方法还具有其它缺点。修改后的规则对于一个特定的应用程序可能是有益的，但对于另一个应用程序来说可能就是不适用甚至是危险的。

还提出了采用专门的“ API ”（来源于英文词“ Application Program Interface ”）以便于定义一种与一个特定应用程序相关并且用于建立虚拟存储空间和物理存储器之间的对应关系（“映射”）的专用算法。但在这种情况下，需要修改相应的应用程序和操作系统的驻留部分（“核”）。因此，这种方法不能用于已存在的程序本身。在任何情形下，这种方法都缺乏灵活性，并且其有效性受到限制。

本发明的主体是一种用于对主存储器，尤其是上述“ NUMA ”类型主存储器进行非均匀存取的数据处理系统的存储器分配方法，该方法试图满足这种特定体系结构的需求并且不具有现有的处理方法中的缺陷。

为此，将存储器分配根据专用于每个应用程序的简要表，即通过执行一组考虑到此简要表的分配规则来执行，其中在检测到的每个页面错误时自动执行一次检索以确定为哪一个执行物理存储器分配。

在最佳实施例中，执行物理存储器分配还要考虑到页面错误的类型。实际上，在其执行过程中，一个应用程序被细分成不同的目标（ object ），例如文本、数据、共用存储器等，它们以不同的方式利用系统的全局存储空间。本发明还可以根据这个参数来优化存储器存取。

在上述变形例中，本发明的方法提供了一种对现有技术的重大改进，特别是在更好的性能和灵活性方面。而且，该方法不必修改现有的应用程序。但是必须注意的是在实际上，一个多处理器数据处理系统，特别是“ NUMA ”类型的系统的虚拟存储空间通常都非常大。为解释这

个概念，如果操作系统是上述“ UNIX ”环境或其衍生产品之一，那么一个虚拟存储段一般表示 256MB 。很容易理解，在这样的条件下，对于一定数量的应用程序来说，每个段一个规则显然不是最优的，即使这个段只与一个类别（ class ）：例如“数据类型”相关。而且，通常将一个段分成虚拟地址子空间，此后按英文术语将其称为“虚拟区域（ range ）”。以不同的方式利用相应于这些虚拟区域的段中的不同地址区。

与作为一个段的细分本身的页不同，如此定义的“虚拟区域”具有可变的长度。实际上，在最佳实施例应用（“ UNIX ”环境）中，虚拟区域的粒度可能落在页的级别之下。

为了进行解释，可以为一个定义了一个物理缓冲存储器的表保留 50MB 的虚拟区域，以用于通过一个输入－输出控制器读取记录在盘上的数据。

即使在仅限于这一个实施例时，也可以理解在“ NUMA ”体系结构中，就与上述磁盘相连的模块和利用这些数据的具体应用程序而言，在性能方面，缓冲存储器在系统的一个物理存储器中的位置是不重要的。

还有，依据本发明的另一变形例并再次依据一个最佳实施例，一个专用存储器分配策略有选择地涉及每个虚拟“区域”。这个技术特征使得在检测出页面错误的情况下进一步优化存储器分配成为可能，取代了实现此方法的变型而进行的对应用程序的有限修改。

依据此变形例，当产生一个被翻译成页面错误的异常信息时，负责查找将要执行的规则的处理程序扫描一个表以确定是否存在与此虚拟区域相关的专用分配策略。如果不存在此专用策略，则采用用于管理由该段构成的更高层单元的策略。反之，一个存储器分配规则则来源于这个专用策略。换言之，同一段可包括一个或多个虚拟“区域”，其中的每一个在其它的一个或多个虚拟区域遵守该段或系统的通用策略的同时，与一个专用存储器分配策略及其规则相关。应当清楚术语“专用”并不意味着与一个被任意标记为 n 的给定区域（其中如果该段包括 m 个区域，则 n 为 1 至 m 之间的数）相关的分配策略必然不同于与另外一个或

多个虚拟区域，如 n+2 和 n+5 虚拟区域相关的分配策略。

实际上，列表结构被用于确定需要利用哪一个存储器分配规则，该列表的每个元素对应于一个具有连续地址并且由虚拟区域的起始和结束地址限定的区域。实际上，该列表的一个元素就是一个存储单元，用于存储适用于一个给定虚拟区域的存储器分配规则。按顺序扫描该列表的各个元素以确定应用哪些规则。

这种方法还可被加速。依据本发明此方案的另一变形例，段被细分成 N 个具有连续虚拟地址和相同长度的子空间。建立一个所谓的也包括 N 项的“散列”表。与这 N 项相关联的是相同数量的单个列表结构。单个列表结构的长度是不固定的。它取决于与这一项相关联的虚拟区域的数目。当检测到一个页面错误时，地址处理器 H 知道导致该页面错误的地址。因此确定该表中相应项的编号是非常容易的。

“散列”表由 N 个存储单元构成，每个存储单元中至少有一条关于存在一个与这一项相关联的虚拟区域的信息，这对于使用专用存储器分配策略来说是必要的。实际上，在检测到一个页面错误时，读取表中的对应项，并且在存在表示具有一个专用分配策略的指针时，在后面的辅助步骤中确定上述段中的哪一个虚拟区域与这个页面错误相关联。对于不包含任何与一个专用策略相关联的虚拟区域的段来说，所要应用的策略就是整个段的策略。另一方面，对于需要专用存储器分配策略的虚拟区域来说，有必要对与表中一特定项相关联的列表结构中的一个或多个元素进行扫描。但是，与前述情形相比，这个列表结构的长度受到更多的限制，因为它只覆盖了与这一项相关联的虚拟区域。作为这两种特性所产生的结果，所述实施例的变形例实质上加速了存储器分配进程，至少加速了用于确定将要使用的规则的步骤。

这样，本发明的主体为一种用于通过映射在一个与一给定软件应用程序相关联的虚拟地址空间中的至少一个具有连续存储地址的区域来分配物理存储单元的方法，该运行于一个数据处理系统的应用程序包括一个非均匀存取存储器单元，并利用多种虚拟存储器类型，通过扫描一个地址对应表来执行该映射过程，其特征在于该方法至少包括一个将这个给定的应用软件与从一组预定义的规则中选定的存储器分配规则相连接

的步骤，并且其中，当这个地址对应表中不包含与此给定应用软件相关联的虚拟地址中的连续存储地址区域的项时，该方法包括生成异常信息的步骤和依据这些存储器分配规则之一分配物理存储单元的后续步骤，规则的选择是通过给定应用软件所用的虚拟存储器类型简要表进行的。

参照下列附图所进行的下述描述将有宜于理解本发明及其其它特征和优点，其中：

图 1 简略示出了具有均匀存储器存取能力的数据处理系统的一种所谓的“ UMA ”体系结构；

图 2 简略地示出了具有非均匀存储器存取能力的数据处理系统的一种所谓的“ NUMA ”体系结构；

图 3a 和 3b 简略地示出了对于两个应用软件的存储器的访问；

图 4 示出了一个应用软件的细分示例；

图 5 简略地示出了在发生了页面错误时，依据现有技术分配存储单元；

图 6a 和 6b 简略示出了在发生页面错误时，依据本发明来分配存储单元；

图 7a 和 7b 示出了依据本发明的方法的另一实施例；

图 8 简略地示出一个依据本发明的方法在数字数据处理系统中的变形例；

图 9 简略地示出了根据第一变形例，依据本发明的方法的另一实施方案；以及

图 10a - 10c 简略地示出了根据第二变形例，依据本发明的方法的另一实施方案。

为了进行解释而不对本发明的范围造成任何限制，除非另外声明，否则后面的描述都是针对操作系统为“ UNIX ”或类似类型的数据处理系统。

对于在此环境下运行的应用程序来说，虚拟地址空间可被分成如下的不同类型：

- 文本或程序文本（可执行代码）；
- 初始化的数据；

- 修改后的数据;
- “堆栈”;
- “堆阵”；即动态分配（表等）;
- 共用存储器;
- 共享库。

在一个应用程序的运行过程中，应用程序以不同的方式均等地利用这些不同类型的系统存储器。而且，一个应用程序或同一应用程序的新实例可在图 2 所示的系统的两个模块 Ma 和 Mb 之一中进行。对于相同的应用程序为真的情况对于具有不同简要表的两个应用程序也为真。

图 3a 和 3b 简略地示出了两种类型的应用程序，即一个“小型数据库”和一个更常规的数据库。

在这两个图中用标记 Mem 表示系统的全局存储器。

在第一种情况下，如图 3a 所示，在初始化阶段，存取被限制在由任意设在图 3a 左侧的区域 Zini 所表示的一个地址空间内。初始化之后，存取发生在由区域 Zf 表示的地址空间中，在图 3a 所示的例子中，区域 Zf 的扩展也受到限制并假设该区域与区域 Zini 相关。因此，用于此特定应用程序的物理存储单元被包含在一个模块中，更准确地说，是包含在一个本地模块中。

如图 3b 所示，对于一个标准数据库来说，这是一种非一般情况。存取能够扩展到整个存储空间，如图 3b 中的前头所示。因而所占用的物理存储单元通常分布到两个或更多模块中。

而且，如上所述，对于相同的应用程序，虚拟存储空间被细分成不同类型的段：文本、数据等。

作为一个不受限制的例子，当一个给定应用程序 Appli 正在运行时，其各部分被划分成虚拟存储段：文本 T、数据 Da（不同类型的）、堆栈 St、共用存储器 Shm、文件 Fi 等，如图 4 所示。通常，一个管理设备或英文术语中常用的“处理程序”分配系统的全局物理存储器 Mem 中的这些虚拟存储单元段。

下面将描述用于在检测页面错误时分配物理存储器的机构。

上面已经指出了一个处于运行过程中的应用程序以不同的方式均等

地利用不同类型的存储器。同样，一个最初在一给定模块（例如图 2 中的 Ma ）中运行的应用程序可继续在另一模块（例如图 2 中的 Mb ）中运行或在一个不同的模块中建立并运行这个应用的附加实例。

假设一个应用程序试图在一给定虚拟地址，例如任意地址“ O_x 2000 ”上执行一个特定指令，如装载“（ load ）”指令，则其中正在运行当前进程的中央处理器（例如图 2 中的 10a ）对该指令译码并扫描一个地址对应表（图中未示）。如果所查找的项不存在，则发送一个被处理程序 H 翻译成页面错误的异常信息。因此在这些情况下，为上述虚拟地址“ O_x 2000 ”分配一个物理存储单元是必要的。

在现有技术中，只存在一种分配方法。换言之，无论应用程序的简要表和段的类型如何，所用的规则都是相同的。因此，处理程序 H 依据系统所用的分配规则分配一个物理存储单元。例如，依据这些规则，在局部物理存储器中系统地执行分配，即如果该进程在中央处理器 10a - 13a 之一的控制下运行于模块 Ma，则在存储器 14a 中进行分配。这个规则对于一个文本类型的段是有利的，而对其它类型的段则不是最优的。

上述机构如图 5 所示。应用程序 Appli 产生一个页面错误 Fp，并且处理程序 H 按照预定的规则为其分配物理存储器 Mem 中的一个单元（图 5 中的虚线示出了其分区 Z_1 - Z_n ）。

如前所述，可以对这些规则进行修改，但它们对于所有的应用程序和所有的段类型保持一致。

另一方面，依据本发明的方法，按照一组规则执行物理存储器 Mem 的分配，这组规则考虑到了应用程序的简要表，并且在最佳实施例中还考虑到了页面错误的类型。

图 6a 和 6b 示出了依据本发明的用于分配物理存储器的机构。

依据本发明的方法的主要特征，每个应用程序与一组预先定义的分配规则相连接。为此，需要使每个应用程序 $Appli_x$（ x 是任意下标）与一个特定简要表相关联。在存储它的操作系统或“ OS ”的控制之下执行这种关联。图 6a 简略地示出了该关联机构。

图 6a 示出了一个特定应用程序 $Appli_x$。如前所述，这个应用程序 $Appli_x$ 利用了各种类型的存储器：文本、数据等。在图 6a 中，示出了六

种类型的存储器，分别被标记为 tyM_1~tyM_6， tyM_1~tyM_6 中的每种存储器与一组将在后面进行定义的预先定义的规则中的一个分配规则连接。这些规则已被标记为 R_1 到 R_6，应当理解这些规则不一定是一组不相交规则。换言之，例如，即使存储器类型 tyM_2 和 tyM_3 本身是不相同的，规则 R_2 和 R_3 也可以是相同的。

刚刚定义的存储器分配简要表 Pa_x 通过一个关联 A_x 与一个特定应用程序 $Appli_x$ 相连接。简要表 Pa_x 是一个具有两项的表：存储器类型 tyM_i 和从一组预先定义的规则中选出的规则 R_j， i 和 j 均为任意下标。

一般地，一个关联函数可定义如下：

Association_pa($Appli_x$,Pa_x).

可在执行此应用程序初始化时定义与一个给定应用程序相连接的存储器分配简要表，或在执行过程中的任意时刻进行动态地重新定义，这就增加了此方法的灵活性。

在图 6b 中，用通用标记 R_g 表示预先定义的分配规则。在出现了被翻译成一个页面错误 F_p 的异常信息时，即在地址对应表中不包含用于一个虚拟地址的项时，处理程序 H 检索刚定义的应用程序 $Appli_x$ 的简要表 Pa_x。在最佳实施例中，还确定页面错误 F_p 的类型。根据这两个参数，分配物理存储器中的存储单元 Z_1~Z_n，它们可能是本地的（位于同一模块中）或远程的（位于另一个模块中）或分布在整个存储器 Mem 中。在图 6b 中，用多个箭头（与图 5 中现有技术的单箭头相对）示出了这种取决于应用程序 $Appli_x$ 的简要表 Pa_x 和页面错误 F_p 类型的分布。

因此，以下述方式定义寻址函数 F_{ad}：

F_{ad}=F(Pa_x,Type Fp)。

因此，用于存储器分配的规则的选取是这两个参数进行组合的结果。

更准确地，一个给定的应用程序 $Appli_x$ 规定了它需要一组预先定义的规则中的哪些分配规则用于每种类型的虚拟存储段。例如，通常使用下述段类型：“客户”段、“映射”段、“永久”段、“工作存储器”段、以及“共享库”段。

为了进行解释而不以任何方式限定本发明范围，定义以下述代码表

示的规则组:

- “ P_STRIPE”: 按照一种“循环法”方法，在构成一个给定应用程序的资源、分布在本地存储器（同一模块）或远程存储器（不同模块）中的整个物理存储器空间的条状带中进行分配;

- “P_LOCAL”: 分配的物理存储器与导致页面错误的中央处理器位于同一模块中;

- “P_FIXED ”: 分配的物理存储器在一预定模块中;

- “ P_DEFAULT ”（或“ P_NONE ”）: 没有物理存储器分配规则，所以使用系统的隐含规则。

例如，从直觉上如上定义的“ P_STRIPE ”类型的分配适用于“共用存储器”类型的段，而“ P_LOCAL ”类型的分配适用于“工作存储器”类型的段。

这样，依据本发明的方法可以根据应用程序的实际需求，更准确地说是应用程序的特定简要表来最佳地优化物理存储器的分配。由于对存储器资源的存取时间，至少是平均存取次数也被优化，从而改进了数据处理系统的性能，另一个优点是能够将任何应用程序连接到预先定义的存储器分配规则集上，而在使用一个新“ API ”的情况下无须对其进行修改或重新编译。

而且，本发明的方法具有很大的灵活性。在很多方面，它允许按照现有技术的方案起作用。例如，如果一个给定的应用程序没有规定或请求分配规则，则利用来自系统的隐含规则（“ P_DEFAULT ”）。而且，一个“子”应用程序能够“继承”与创建它的“母”应用程序相关的存储器分配规则。在例如一个应用程序运行于另一模块中的其它情况下亦是如此。

因而很容易看到本发明的方法在所述的变形例中提供了对现有技术的重大改进，包括在更好的性能和更大的灵活性方面。

但是，正如本说明书开头部分所述，当前多处理器数据处理系统的虚拟地址空间非常大。在“ UNIX ”环境框架中，一个段表示例如 256MB 或 2^{16} 页。通常还根据需要将这些段细分成具有可变长度的“区域”。甚至在这些虚拟子空间属于一个公用段时，与其连接的各种应用程序也以

不同的方式利用它们。因此至少对于特定类型的应用程序来说，为所利用的所有一个或多个段执行一个规则可能不是最优的。

作为一个非限定性实施例，将参照图 7a 再次描述一个“ NUMA ”类型的系统。这个以 1”标记的系统类似于图 2 所示的系统 1’，但假设该系统包括至少三个通过连接 2”互连的“ SMP ”模块 Ma 、 Mb 和 Mc 。还假设模块 Mc 通过以一个标记 I/O_c 标注的控制器和标准输入－输出电路与一个磁盘驱动器 D 相连。

最后，假设一个给定的应用程序 $Appli_x$ 运行于模块 Ma 的处理器之一中，例如处理器 10a 中。这个应用程序特别地对一个以 Sg_x 任意标记的系统 1”中的虚拟地址空间段进行操作。这个段 Sg_x 如图 7b 所示。为简化起见，假设只包括以 Ra_1 ~ Ra_5 为标记的五个虚拟“区域”。在所述的实施例中，虚拟区域 Ra_2 被分配给上述应用程序 $Appli_x$，并且更精确地，它相关于一个容量例如为 50MB 的表。这个表相关于位于系统 1”的一个物理存储器中的一个相同容量的缓冲存储器。而且，在所述的实施例中，假设由于与段 Sg_x 的类型和应用程序 $Appli_x$ 简要表相关联的规则而导致缓冲存储单元实际上位于模块 Mb 的主存储器 14b 中。因此，该应用程序 $Appli_x$ 的数据读取特别需要通过连接 2”的两次转接，其转接造成了在处理时间上特别不利的操作。

为了避免进行双次转接，对于持续用于虚拟区域 Ra_2 分配规则来说，将一个 50MB 的缓冲存储器单元放入在模块 Mc （电路 I/O_c 和磁盘驱动器 D 附近）的物理主存储器中或正在运行该应用程序 $Appli_x$ 的模块 Ma 的物理主存储器中都是有利的。通常实践表明，从系统性能的角度来说，第一种方案能够提供更好的结果。

在这个简单的例子中可以容易地看到即使依据本发明的方法执行适用于应用程序简要表的存储器分配规则，即使这些规则可进行动态调整，但还存在一些该方法未被完全优化的情况。

根据本发明的方法的另一方面，在最佳实施例中，虚拟“区域”还可选择性地与专用规则相关联。

再次参照图 7b ，只有虚拟区域 Ra_2 和 Ra_4 被认为与专用规则相关联。为了进行解释，虚拟区域 Ra_4 也代表一个表，但这次是一个例如

100MB 的表。其它虚拟区域（图 7b 中阴影线部分表示的）没有与专用规则连接。在这种情况下，为与所应用的虚拟存储器的更高层单元，此时为段 Sg_x 相关联的规则。以上面所述的方式执行此过程。

更精确地，依据此附加实施例，用于定义一个专用的物理存储器分配策略的辅助信息与每个虚拟区域相关联。但是，这个策略是可选的。实际上，辅助信息包括一个被称为指针“ prange ”的第一字段，用于指示对于一个段，是否必须将至少一个专用策略有效用于一个虚拟“区域”（“ prange ”≠0 ）还是它是否必须应用与其全局相关的分配策略（“ prange”=0 ）。

第二字段与策略类型相关。在虚拟区域一级，再次找到上述定义的用于一个全局段的规则集：“ P_STRIPE ”、“ P_FIXED ”等等。

最后，至少对于特定类型的存储器分配策略来说，需要规定其中要分配物理存储器的模块编号或地址。为此，还有一个第三字段或“模块 No ”字段。对于类型“ P_FIXED ”：需要规定预先定义的模块的编号。对于类型“ P_STRIPE ”：也需要知道以前所用的模块编号，以便依据所用的存储器条状分布规则（“循环法”）增加其编号。因此需要在一个存储器位置、寄存器或计数器中保存以前所用的编号。

再次参照图 7b，用于分配虚拟区域 Ra_1~Ra_5 的策略可标识如下：

- 对于虚拟区域 Ra_1、Ra_3 和 Ra_5，没有专用策略，从而规则为用于段 Sg_x，例如类型 = P_FIXED 和模块编号 = Ma 的编号的规则；

- 对于虚拟区域 Ra_2，存在如上所述的类型 = P_FIXED 且模块编号 = Mc 的编号的专用策略；

- 对于虚拟区域 Ra_4，存在例如类型 = P_FIXED 且模块编号 = Mb 的编号的专用策略。

在所述的变形例中，该方法可以使基于页面错误检测的物理存储器分配得到最佳优化。正如前面的变形例所述，它需要对操作系统进行修改，而且还需对使用它的应用程序稍加改进。

例如，在读和写“ UNIX ”中公知的“ bind ”类型的指令，并且该操作必须在具有如前所述的 50MB 的缓冲区的 3 号模块（例如模块 Mc ）中执行的情况下，就需要将一个在系统调用时初始化一个用于所使用的

虚拟区域（例如 Ra_2）的专用策略的指令加到指令流中，其中该指令被称为“策略”。后者具有如下形式:

策略〔地址、大小（例如 50MB），策略（例如 P_FIXE），模块（例如 No.3）〕

尽管所有的应用程序都能受益于这个附加的依据本发明的方法的变形实施例，但不必对所有类型的应用程序进行修改。

必须注意的是那些未经修改的应用程序应当能够正常地运行。以图 6a 和 6b 所示的方式执行物理存储器分配策略。

所用的规则取决于这些应用程序的简要表，并且最好是上面定义过的规则，虽然也可以与之不同。

另一方面，依据该附加变形例的方法最好用于对同一个虚拟存储空间中的数据进行操作的应用程序。可将“驱动程序”类型的应用程序，特别是磁盘和网络“驱动程序”，即外设或网络管理程序引为非限定性例子。通过一个被称为“系统策略”的指令，这些应用程序使用其中具有“缓冲”存储器区域的“数据”类型段，对于所述“缓冲”存储器区域应用程序具有定义专用策略的能力。

其它类型的应用程序尤其受影响。这些应用程序是指那些通过共用存储器进行彼此间通信的应用程序。例如，在第一应用程序运行于图 7a 中模块 Ma 的一个处理器中，而第二应用程序运行于同一模块 Ma 的一个处理器中，它们共享“共用存储器”类型段的第一虚拟区域的情况下，出于系统性能的原因，直觉上对于“P_FIXED”或“P_LOCAL”类型的存储器分配策略和模块编号为 1（模块 Ma）是有利的。从直觉上说，这种配置对于改进系统性能是必要的。因此，为这个第一虚拟区域指定一个专用存储器分配策略是有用的。同样，如果第三应用程序运行于模块 Mb 中并与第一应用程序共享属于同一“共用存储器”类型段中的第二虚拟区域，则对于“P_FIXED”的存储器分配策略类型和模块编号为 2 的模块（模块 Mb）是有利的。从直觉上说，这种配置对于改进系统性能也是必要的。因此，为这个第二虚拟区域指定一个专用存储器分配策略也是有用的。

下面描述在实际实施例中，如何在一台实机中实现本发明的这些不

同变形例所述的方法。为了进行解释而不以任何方式限定本发明范围，将对“UNIX”或类似运行环境下的计算机框架进行描述。

在这种类型的计算机中，存在一个段表 Scb，其中记录了用于定义这些段的数据，特别是关于其类型或类的数据，如图 8 所示。本发明的方法利用了这种特性的优点。

关于一个给定的应用程序 $Appli_x$，其存储器简要表 Pa_x 被存在已创建该应用程序所用的段的结构中。如图 6a 所示，简要表 Pa_x 通常包括多种存储器类型。对于一个给定的段，其类型如段表 Scb 中的一个记录 Sgc_y 所述。该类型涉及简要表 Pa_x 的一个元素，根据本发明，该元素将通过表 Scb 中记录的辅助数据 Mp_y 进行描述。

这些辅助数据 MP_y 准确地表示出在段 Sgc_y 的共用级（global level）或在该段的长度可变的细分的虚拟区域级将要应用的存储器分配策略。图 8 示出了一个给定的应用程序 $Appli_x$ 和段表 Scb 之间的主要相互作用。

实际上，对于已对一个应用程序进行修改以支持一个用于虚拟区域级的专用存储器分配策略的情况，对于每个段都有几组辅助数据。

如上所述，每组辅助数据包括多个字段：用于表示该段中是否具有涉及到的专用策略的指针字段“prange”，用于指示所要应用的规则类型（“P_FIXED”等）的字段“policy_type”，以及指示至少用于特定类型规则（例如规则为“P_FIXED”）的模块编号的字段“module”。为了限定各虚拟区域，还需要用于规定这些虚拟区域的低和高虚拟地址的信息。这些信息分别出现在字段“pno_start 和“pno_end”中。

当检测出一个页面错误时，则需要扫描这些不同的数据以确定所要利用的精确分配策略。依据本发明的一个方面，采用一种列表结构来组织这些辅助数据组，如图 9 所示。

假设引起页面错误的编址段包括 z 个虚拟区域。在表 Scb 中，列表结构的第一存储器位置包括与整个段相关的数据，还特别包括用于此段的全局存储器分配策略。对于未经修改的一个应用程序来说，仅存在这个存储器位置。

列表中以 Ra_1~Ra_z 标记的其它元素与各个连续的虚拟区域相关。这

样，对每个元素来说，都有不同的字段用于指针“ prange ”不为零的段：“ policy_type ”、“ module ”、“ pno_start ”和“ pno_end ”。

引起页面错误的段中的地址是已知的。处理程序 H 提供了分别被称为 idx 和 pno 的这个地址和所涉及的段。这些数据使寻址表 Scb 成为可能。通过将此地址 pno 与 Ra_1~Ra_z 中的每个列表元素的低界和高界（“ pno_start ”和“ pno_end ”地址）相比较，可以确定是否需要将一个专用存储器分配策略应用到所述的虚拟区域中，如果需要，则根据字段“ policy_type ”所指示的类型，确定应该应用哪种类型的策略（“ policy_type ”）和可能涉及哪一个模块（“ module ”）。

在一个段的创建过程中，这些辅助数据根据所涉及的应用程序简要表被初始化。然后通过上述“ system policy ”类型的指令，由应用程序对这些辅助数据进行动态修改。

为了进行解释，以举例方式做出如下假设：

用于该段的全局策略为“ P_LOCAL ”类型，由于分配发生在产生页面错误的模块中，故不需要一个模块编号。

与第一虚拟区域相关联的专用策略，即记录在列表 LRa_1 的第一元素中，包括下述字段：“ policy_type ” = “ P_FIXED ”以及“ module ” = 2；

与第二虚拟区域相关联的专用策略，即记录在列表 LRa_2 的第二元素中，包括下述字段：“ policy_type ” = “ P_STRIPE ”且“ module ” = 3；

一与虚拟区域 z，即记录在列表 LRa_z 相关联的专用策略的最后一个元素中，包括下述字段：“ policy_type ” = “ P_DEFAULT ”且“ module ” = “ ”（即为空）。

当然，地址字段“ pno_start”和“ pno_end ”也被赋予（ document ）给每个虚拟区域 1 ~ z。

如果地址 pno 指向由列表元素 LRa_2 的地址字段确定的空间，处理程序 H 就响应其请求，接收用于指示类型为“ P_STRIPE ”且模块编号为 3，或图 7a 中的 Mc 的数据。由于分配是以划线分块的形式进行循环，所以模块编号必须被增加（或修改）。

依据所述变形例的方法可被进一步改进。实际上可以通过缩短用于根据页面错误的检测而确定用于一个虚拟区域的存储器分配策略的时间来增强系统性能。为此，所述附加实施例的一个变形例利用了一个具有计算寻址能力，即具有一个所谓的“散列”码的虚拟区域表。

依据图 10a~10c 所示，执行一个专用分配策略的确定。段，例如 Sg_y，y 为任意下标，被细分成 N 个定长的子段。在上述“ UNIX ”运行环境中，一个代表 256 MB 虚拟存储器的段被细分成以图 10b 中的 SS_1~SS_N 标记的 256 个 1MB 的子段是比较有益的，因为 256 是 2 的乘方。

“散列”表 Th（图 10a ）还包括与 N 个存储器子段相对应的 N 项。每一项存储专用于子段 SS_1 ~ SS_N 的存储器分配策略。一个单独的列表结构与表 Th 的每一个存储单元相关联。更准确地，在和只有在所述子段包括至少一个属于必须应用专用存储器分配策略的虚拟存储“区域”的区域时，这个列表结构才存在。

参照图 10b，假设段 Sg_y 的“ prange ”不同于 0，由此假设这个段包括至少一个与专用存储器分配策略相关联的虚拟区域。

假设子段 SS_1 不包括任何与一专用策略相关联的虚拟区域。在图 10b 中，这种情况适用于子段 SS_3 和 SS_N。另一方面，假设子段 SS_2 包括三个“区域” Ra_{02}~Ra_{22}。第一区域 Ra_{02} 没有与专用策略相关联，而另外两个区域 Ra_{12} 和 Ra_{22} 分别与专用存储器分配策略，例如“ P_FIXED ”和“ P_STRIPE ”，相关联。

然后“散列”表 Th 的项 e_2 与一个包括两个元素 LRa_{12} 和 LRa_{22} 的列表结构相关联，该列表结构以上述方式存储两个专用策略的特性。

在图 10a 中，假设项 e_1、 e_3、 e_5、 e_6 和 e_N 相应于不包括与专用策略相关联的虚拟区域的子段（同一位置(rank)上）。另一方面，除上面已提到的项 e_2 以外，项 e_4 和 e_7 相应于包括与专用存储器分配策略相关联的虚拟区域的子段。应当注意关联于一项的每个单独列表元素数目，是可变的。实际上，如上所述，虚拟“区域”不具有固定的长度。在图 10a 所示的例子中，与项 e_2 相关联的列表结构包括两个元素 LRa_{12} 和 LRa_{22}，与项 e_4 相关联的列表结构包括一个元素 LRa_{14}，且与项 e_1 相关联的列表结构包括三个元素 LRa_{17} ~ LRa_{37}。

当检测到一个页面错误时，处理程序 H 知道一个给定段，例如具有标引 idx 和地址 pno 的段中的引起引页面错误的虚拟地址，通过该地址可以找到子段的位置，例如子段 SS_2。

在第一步骤中，读取表 Th 的相应项。通过读取记录在表 Th 中的信息组，很可能在这一步骤中直接确定出所要应用的策略。如果情况并非如此，则只读取与相应于引起页面错误的子段的一个给定项相关联的列表元素，即在这种情况下，只读取与项 No.2 相关联的元素。由于这些元素只覆盖了一个子段，即此例中所述的 1MB，因此总是受到限制。如上所述，在“ UNIX ”运行环境下的最佳应用程序中，一个虚拟“区域”的最小粒度为一页，因此每个子段中的最大区域数至多被限定为一个子段的构成页数（例如 256 ）。

因此，所述的这两方面特性加速了用于获取确定所用存储器分配策略所需的数据的进程。

如上所述，虚拟区域不具有固定的长度。特定的区域还可“横跨”两个或多个相邻的子段。这种情况出现在一个扩展为 50MB 以上的区域中，例如若一个子段的长度为 1MB。通过将这些区域本身细分成“子区域”，可以解决这个问题。这种方法不是不利的，因为在其创建过程中，对执行时间的要求不严格。另一方面，在检测到页面错误时，需要很快地分配一个物理存储单元。

在所述附加实施例的变形框架中，参照图 10c ，假设一个页面错误发生在一个给定子段的组成虚拟地址上。假设该虚拟区域在其创建时被细分并分布在位置为 p 和 p+1 的两个相邻子段中。

在这种情况下，在创建该虚拟地址区域时，通过上述的“ system poicy ”类型的指令初始化“散列”表 Th 中的项 p 和 p+1 。在发生了页面错误时，只利用“散列”表 Th 中的一项。该项为与此页面错误相关联，即再次相应于一个子段（地址 pno ）的一项。

如果存在这些项，则分别扫描与这些项 LRa_{1p} 等和 $LRa_{1(p+1)}$等相关联的列表元素，以确定用于引起上述页面错误的地址的存储器分配策略。

更一般地，多项与多个子段相对应，并且当一个给定的虚拟区域被

完全包括在一个子段，即此例中所述的 1MB 虚拟子空间中时，只利用其中一项。

通过上述记载内容，可以很容易看出本发明清楚地达到了所设想的目的。

而且还可以在考虑到所用的存储器类型及其专用简要表的同时，根据应用程序的实际需要求更好地利用存储空间。而且，在附加实施例中，通过下降到更精细的分级，即所谓的“虚拟区域”级来取代对应用程序的稍加修改，可以获得更大程度的优化。但是，即使在计算机中有效地实现了这个最佳实施例，也不必对所有类型的应用程序进行修改。作为受益于由该方法的第一实施方式提供的性能改进的结果，也能够运行未经修改的应用程序，并且存储器分配策略是可应用在段级的策略。通常，只修改那些所期望的性能增益比较具体的应用程序就足够了：在定位很重要的虚拟空间中处理数据的应用程序，如磁盘和网络“驱动程序”等。

但是，必须清楚本发明并不仅限于所述的实施例，尤其是该方法不仅限于说明书所述的预先定义的存储器分配规则集。

还应当清楚虽然本发明特别适用于所述的“ NUMA ”型多处理器系结构，但并不仅限于这种应用。本发明的方法还优先适用于不以均匀的方式进行物理存储器存取的任何数据处理系统，无论此存储器是否分布在多个计算机或模块中。

最后，虽然本发明的方法特别适用于“ UNIX ”或类似运行环境下的操作系统，但应当清楚本发明并不仅限于这种运行环境。

说 明 书 附 图

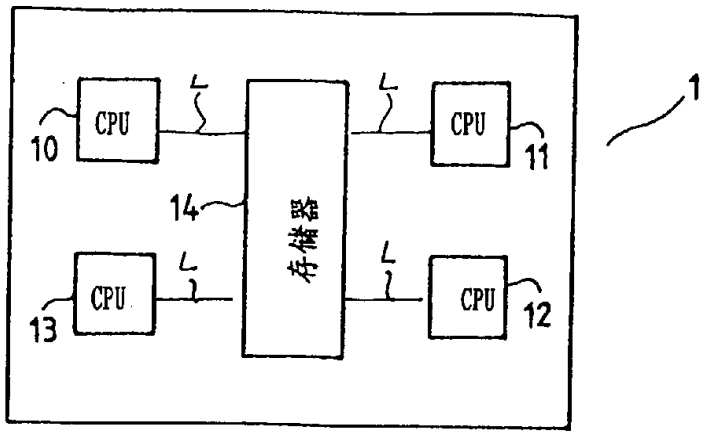


图 1

UMA数据处理系统

NUMA数据处理系统

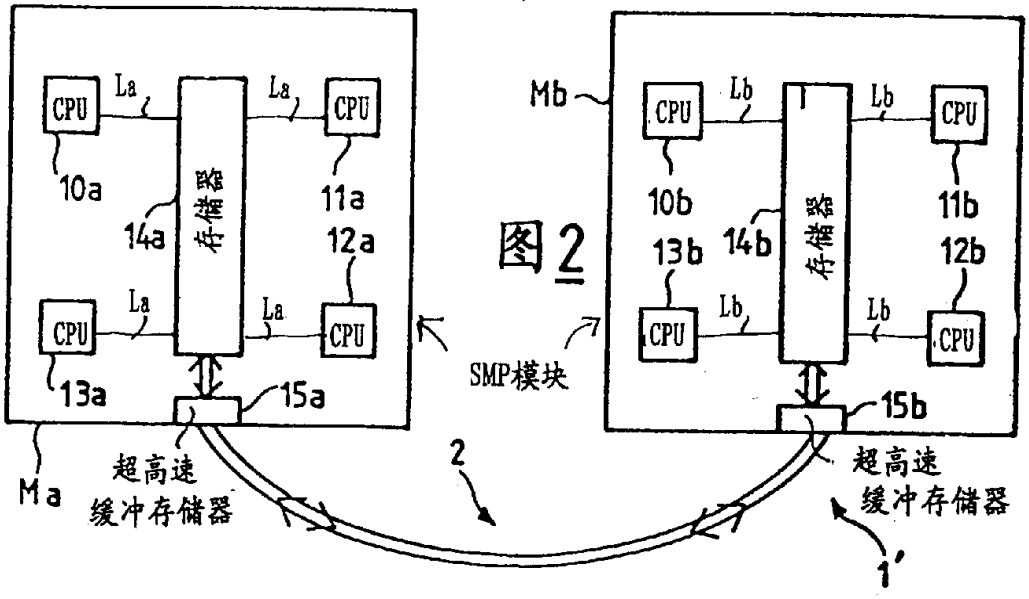


图 2

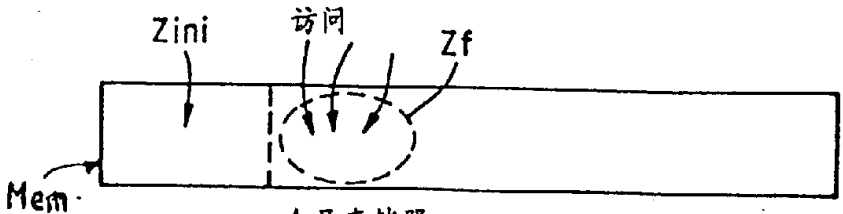


图3a

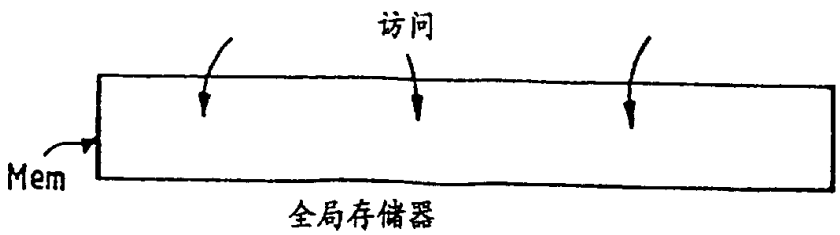


图3b

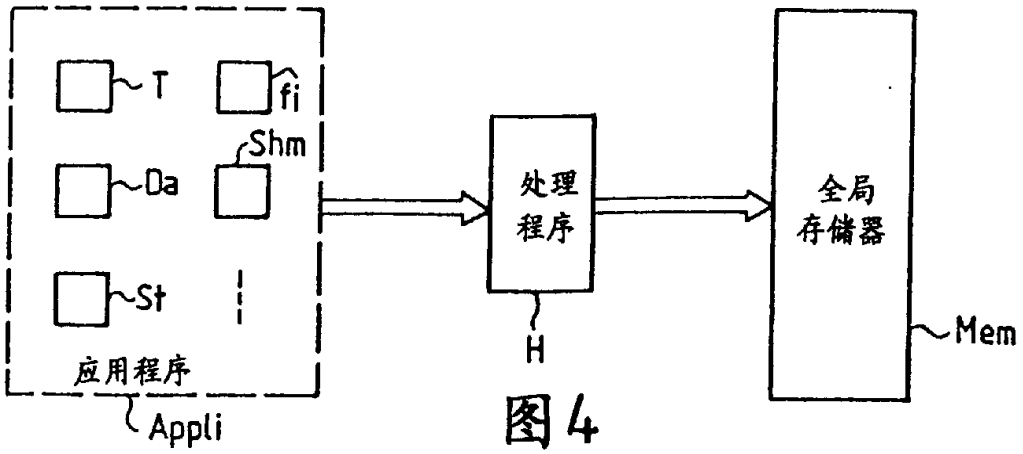


图4

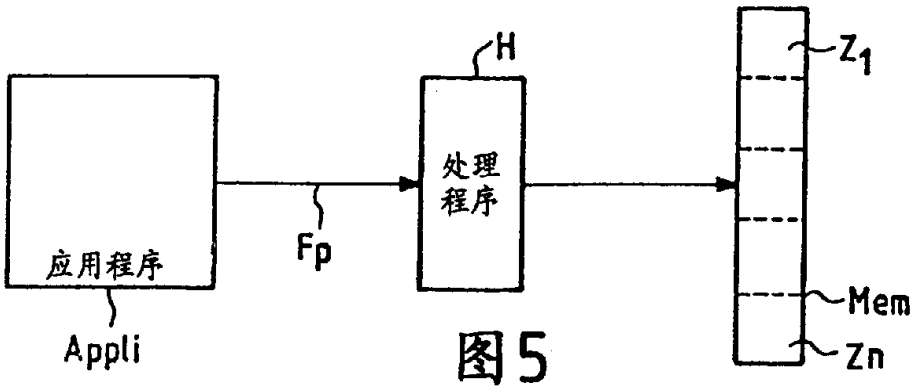


图5

（现有技术）

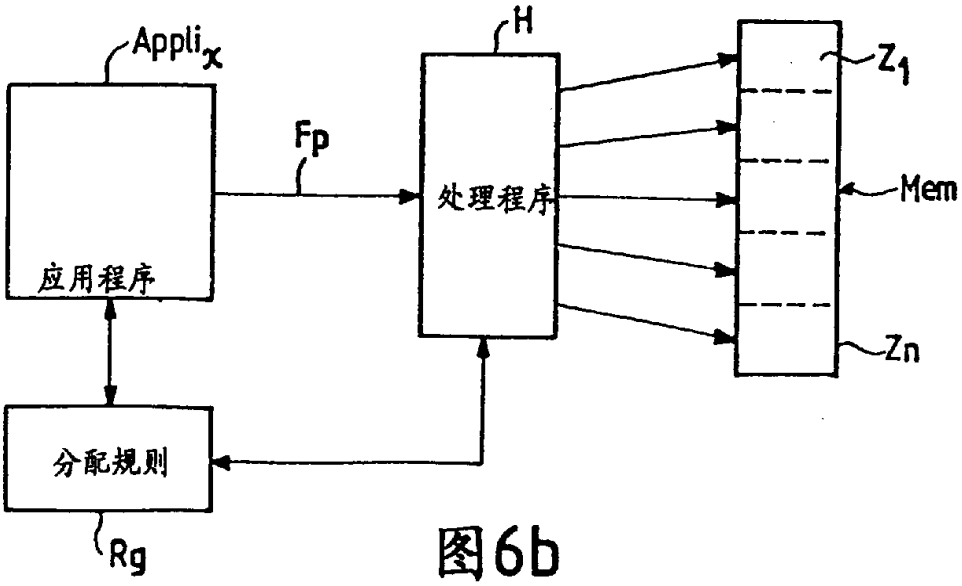


图6b

图 6a

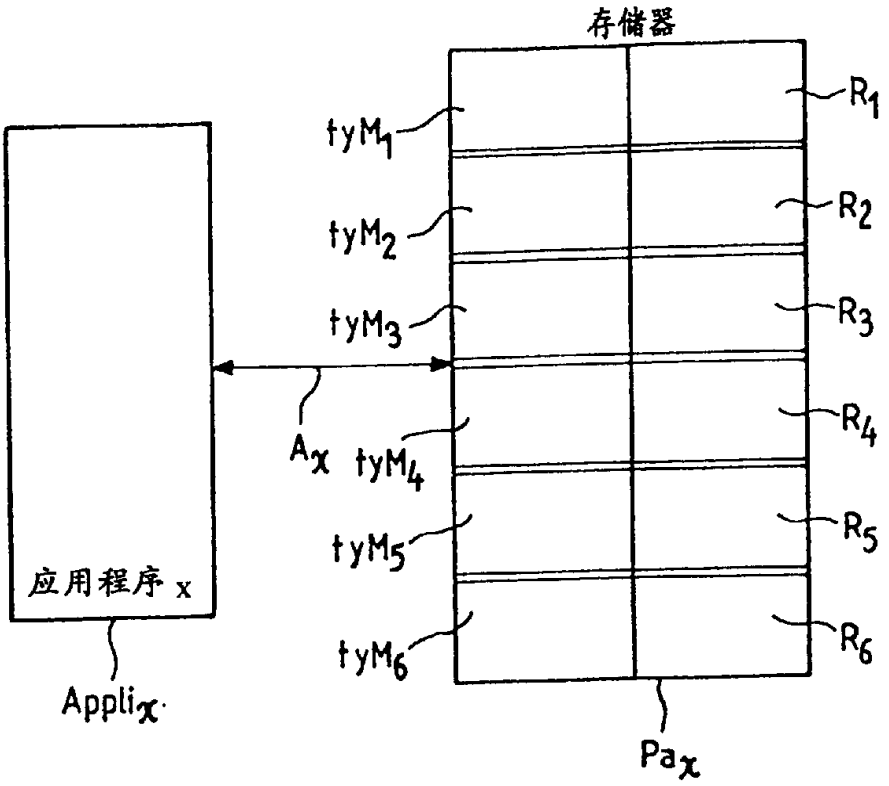
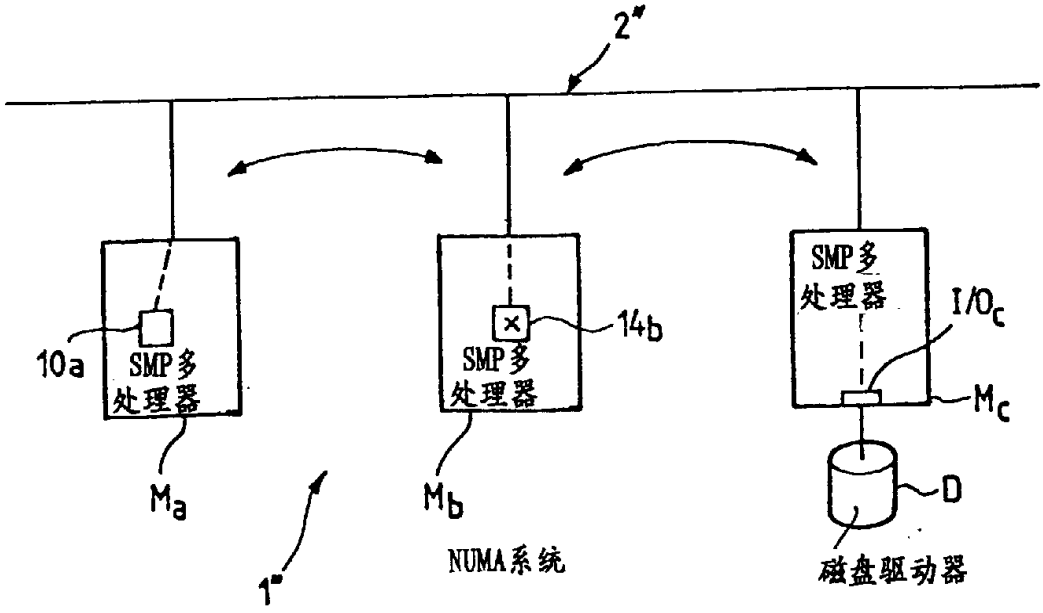

存储器
R1
R2
R3
R4
R5
R6
tyM1
tyM2
tyM3
tyM4
tyM5
tyM6
Ax
应用程序 x
Applix
Pax

图 7a


2"
10a
SMP多处理器
Ma
SMP多处理器
14b
Mb
SMP多处理器
I/Oc
Mc
D
磁盘驱动器
1"
NUMA系统

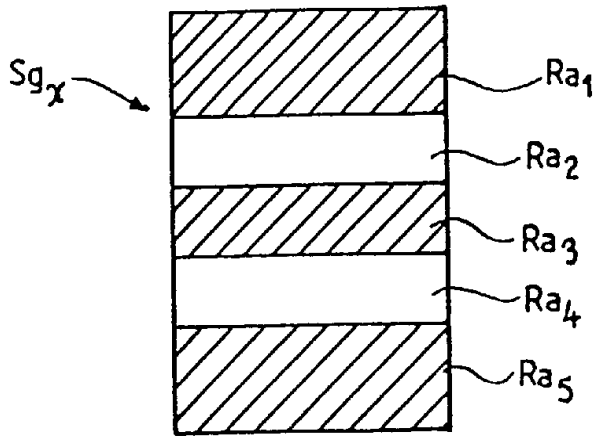

Sg_x
Ra_1
Ra_2
Ra_3
Ra_4
Ra_5
虚拟地址空间

图 7b

图 8

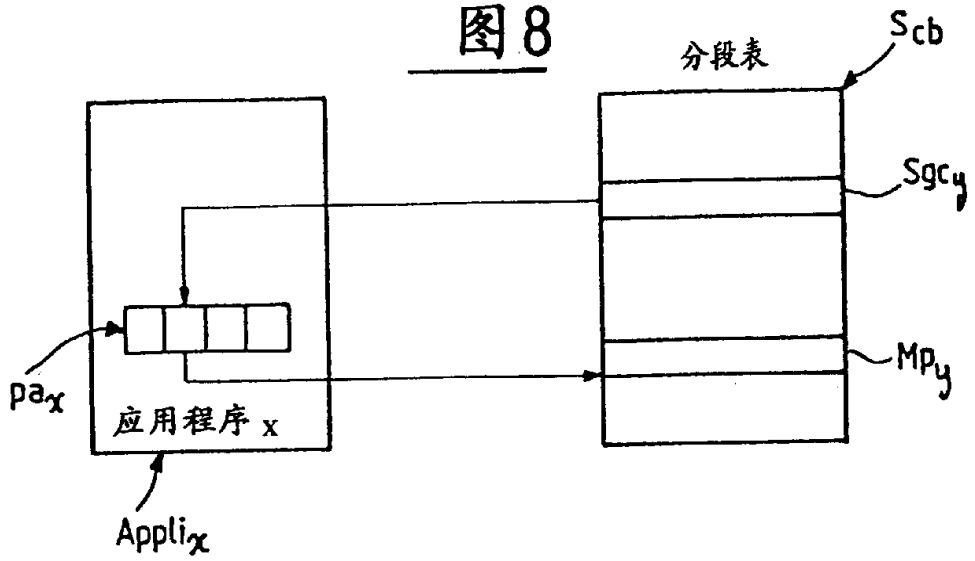
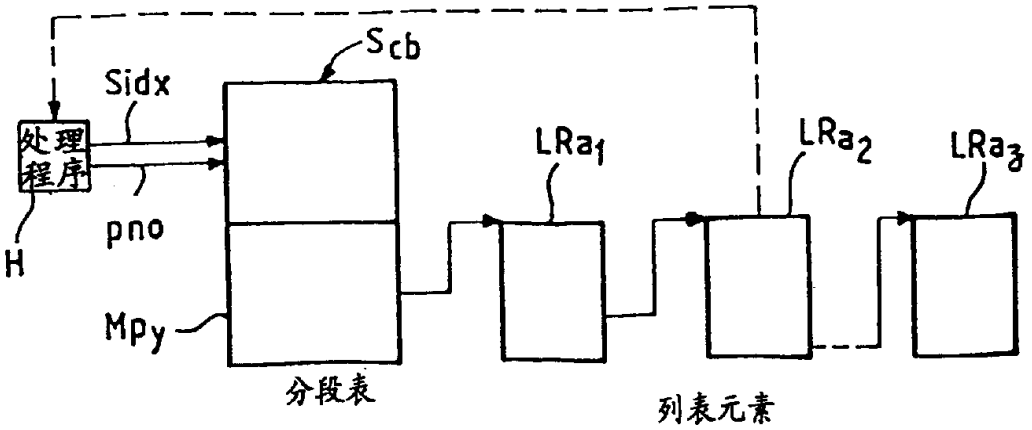

分段表
Scb
Sgc_y
Mp_y
pa_x
应用程序 x
$Appli_x$

图 9


Scb
Sidx
处理程序
pno
H
Mpy
LRa1
LRa2
LRa3
分段表
列表元素

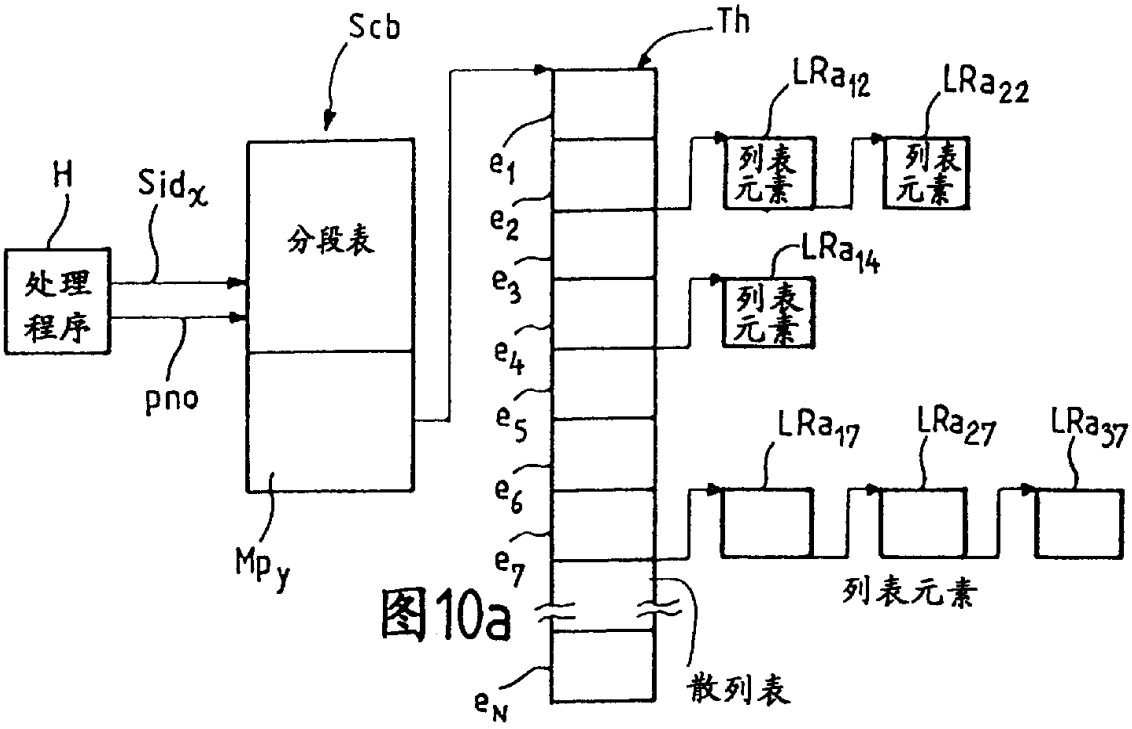


图10a

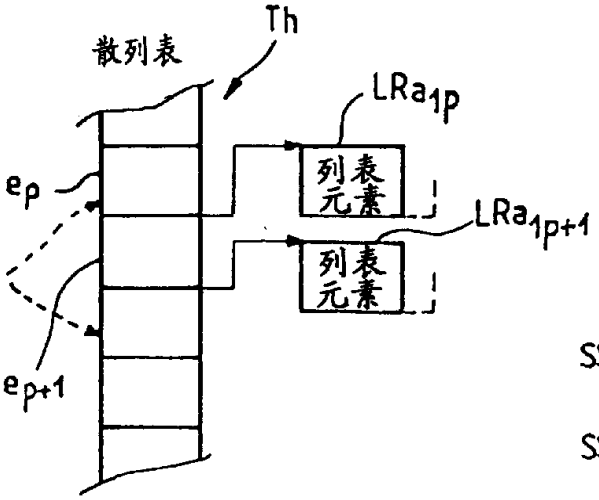


图10c

图10b