(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 113934459 A
(43)申请公布日 2022.01.14

(21)申请号 202111143789.2

(22)申请日 2016.06.10

(30)优先权数据
62/173,389 2015.06.10 US
62/173,392 2015.06.10 US
62/290,383 2016.02.02 US
62/290,389 2016.02.02 US
62/290,392 2016.02.02 US
62/290,395 2016.02.02 US
62/290,400 2016.02.02 US
62/293,145 2016.02.09 US
62/293,147 2016.02.09 US
62/293,908 2016.02.11 US

(62)分案原申请数据
201680046182.3 2016.06.10

(71)申请人 无比视视觉技术有限公司
地址 以色列耶路撒冷

(72)发明人 约瑟夫·克雷宁 尤斯·阿尔贝里吉尔·伊斯雷尔·多贡

(74)专利代理机构 北京安信方达知识产权代理有限公司 11262
代理人 陆建萍 杨明钊

(51)Int.Cl.
G06F 9/38(2006.01)
G06F 9/30(2006.01)
G06F 9/345(2006.01)
G06F 9/52(2006.01)
G06F 11/10(2006.01)
G06F 12/084(2016.01)
G06F 12/0842(2016.01)
G06F 12/0875(2016.01)
G06T 1/20(2006.01)

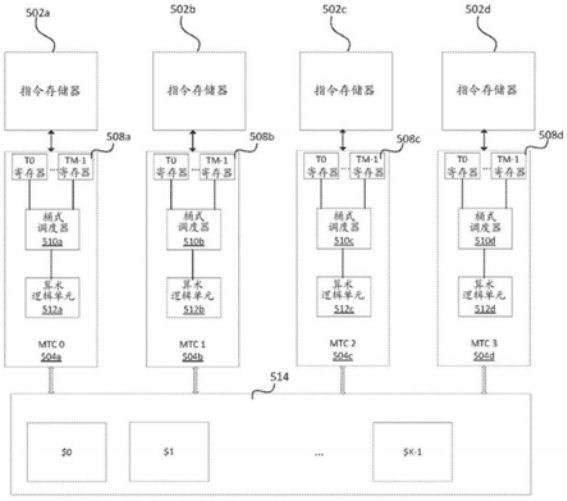
权利要求书15页 说明书94页 附图60页

(54)发明名称

使用多线程处理的多核处理器设备

(57)摘要

本发明涉及使用多线程处理的多核处理器设备。提供配置成在某些计算情境中提高处理性能的多核处理器。多核处理器包括:多个处理核心,其实现桶式线程处理以在确保在处理器的性能被最小化时空闲指令或线程的效果的同时并行地执行多个指令线程。多个核心也可共享公共数据高速缓存,从而最小化对昂贵和复杂机构的需要以减轻高速缓存间一致性问题。桶式线程处理可最小化与共享的数据高速缓存相关联的时延影响。在一些例子中,多核处理器还可包括串行处理器,其配置成执行可以在采用桶式线程处理的处理环境中不产生令人满意的性能的单线程程序代码。



CN 113934459 A

1.一种设备,包括:

一个或更多个存储器组;

一个或更多个寄存器;

连接器,所述连接器将所述一个或更多个存储器组与所述一个或更多个寄存器连接;以及

一个或更多个队列,所述一个或更多个队列存储用于访问来自所述一个或更多个存储器组的数据的一组地址,所述一组地址基于所选择的地址是否与在数据请求中的其他地址是几乎相同的来从所述数据请求进行选择。

2.根据权利要求1所述的设备,其中,所述选择包括基于指示符矩阵的对角元素来选择所述一组地址,以及所述对角元素指示在所述数据请求中的地址是最低索引近邻还是没有近邻。

3.根据权利要求2所述的设备,其中,所述选择包括选择下列项中的至少一个:与所述数据请求中的任何其他地址不是几乎相同的地址,以及与所述数据请求中的一组地址几乎相同并且具有在所述一组地址中的地址的最低索引的地址。

4.根据权利要求2所述的设备,其中,所述选择包括确定所述数据请求中的所述地址中的任何地址是否对应于存储在所述一个或更多个存储器组中的同一个存储器组中的同一个字。

5.根据权利要求2所述的设备,其中,所述连接器基于所述指示符矩阵向所述一个或更多个寄存器提供数据。

6.根据权利要求5所述的设备,其中,基于所述指示符矩阵向所述一个或更多个寄存器提供数据包括:

经由所述指示符矩阵确定与所述数据请求中的任何其他地址不几乎相同的或者与所述数据请求中的一组其他地址几乎相同并且具有在所述一组其他地址中的地址的最低索引的在所述数据请求中的一个或更多个地址;以及

将与所确定的一个或更多个地址相关联的数据提供给所述一个或更多个寄存器。

7.根据权利要求6所述的设备,还包括一个或更多个复用器,并且其中所述确定由所述一个或更多个复用器执行。

8.一种用于处理数据的方法,包括:

确定指示数据请求中的地址是否与在所述数据请求中的其他地址几乎相同的指示符矩阵;

基于所述指示符矩阵来选择用于访问来自一个或更多个存储器组的数据的一组地址,将所选择的一组地址存储在一个或更多个队列中;以及

基于所述一个或更多个队列从所述一个或更多个存储器组检索对应于所述选择的一组地址的数据。

9.根据权利要求8所述的方法,其中,所述选择包括基于所述指示符矩阵的对角元素来选择所述一组地址,以及所述对角元素指示在所述数据请求中的地址是最低索引近邻还是没有近邻。

10.根据权利要求8所述的方法,其中,所述选择包括选择下列项中的至少一个:与所述数据请求中的任何其他地址不几乎相同的地址,以及与所述数据请求中的一组地址几乎相

同并且具有在所述一组地址中的地址的最低索引的地址。

11.根据权利要求8所述的方法,其中,所述选择包括确定所述数据请求中的所述地址中的任何地址是否对应于存储在所述一个或更多个存储器组中的同一个存储器组中的同一个字。

12.根据权利要求8所述的方法,还包括经由连接器基于所述指示符矩阵向一个或更多个寄存器提供数据。

13.根据权利要求12所述的方法,其中,基于所述指示符矩阵向所述一个或更多个寄存器提供数据包括:

经由所述指示符矩阵确定与所述数据请求中的任何其他地址不几乎相同的或者与所述数据请求中的一组其他地址几乎相同并且具有在所述一组其它地址中的所述地址的最低索引的在所述数据请求中的一个或更多个地址;以及

将与所确定的一个或更多个地址相关联的数据提供给所述一个或更多个寄存器。

14.根据权利要求13所述的方法,其中,确定所述一个或更多个地址由一个或更多个复用器执行。

15.一种非临时计算机可读存储介质,其上存储有软件程序的指令集,所述指令集在由计算设备执行时使所述计算设备:

确定指示数据请求中的地址是否与所述数据请求中的其他地址几乎相同的指示符矩阵;

基于所述指示符矩阵来选择用于访问来自一个或更多个存储器组的数据的一组地址;

将所选择的一组地址存储在一个或更多个队列中;以及

基于所述一个或更多个队列来从所述一个或更多个存储器组检索相应于所述选择的一组地址的数据。

16.根据权利要求15所述的非临时计算机可读存储介质,其中,所述选择包括基于所述指示符矩阵的对角元素来选择所述一组地址,以及所述对角元素指示所述数据请求中的地址是最低索引近邻还是没有近邻。

17.根据权利要求15所述的非临时计算机可读存储介质,其中,所述指令集使所述计算设备选择下列项中的至少一个:与所述数据请求中的任何其他地址不几乎相同的地址,或者与所述数据请求中的一组地址几乎相同并且具有所述一组地址中的地址的最低索引的地址。

18.根据权利要求15所述的非临时计算机可读存储介质,其中,所述选择包括确定所述数据请求中的地址中的任何地址是否对应于存储在所述一个或更多个存储器组中的同一个存储器组中的同一个字。

19.根据权利要求15所述的非临时计算机可读存储介质,其中,所述指令集使所述计算设备基于所述指示符矩阵经由连接器来向一个或更多个寄存器提供数据。

20.根据权利要求19所述的非临时计算机可读存储介质,其中,基于所述指示符矩阵来向所述一个或更多个寄存器提供数据包括:

经由所述指示符矩阵来确定与所述数据请求中的任何其他地址不几乎相同的或者与所述数据请求中的一组其他地址几乎相同并且具有所述一组其他地址中的地址的最低索引的在所述数据请求中的一个或更多个地址;以及

将与所确定的一个或更多个地址相关联的数据提供给所述一个或更多个寄存器。

21.根据权利要求20所述的非临时计算机可读存储介质,其中,确定所述一个或更多个地址由一个或更多个复用器执行。

22.一种设备,包括:

存储器;以及

数据处理子系统,其包括:

至少一个数据处理单元;

至少一个地址生成单元,其基于参数的第一集合来在所述存储器中生成地址的第一集合;

控制子系统,其将参数的所述第一集合和参数的第二集合提供给所述数据处理子系统;以及

队列,其存储参数的所述第一集合和参数的所述第二集合并且在所述数据处理单元使用所述地址的第一集合来处理数据时向所述地址生成单元提供参数的所述第二集合。

23.根据权利要求22所述的设备,其中:

由所述数据处理单元处理的所述数据包括至少一个图像窗口,所述地址的第一集合中的至少一个对应于在所述存储器中的存储所述至少一个图像窗口的第一部分的位置;以及

所述地址生成单元通过使用图像参数对所述至少一个图像窗口进行迭代来生成所述地址的第一集合,所述图像参数包括在所述存储器中的存储所述至少一个图像窗口的第二部分的地址。

24.根据权利要求23所述的设备,其中,所述数据处理子系统包括将所述图像参数提供给所述地址生成单元的标量处理器。

25.根据权利要求22所述的设备,其中,所述数据处理单元包括至少一个向量处理器,并且由所述数据处理单元处理的所述数据包括向量数据。

26.根据权利要求23所述的设备,其中,所述数据处理子系统以增量的方式从所述存储器中检索数据,并且所述数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据,所述偏移量指示所述未对齐的向量数据在存储器中关于所述增量是如何未对齐的。

27.根据权利要求26所述的设备,其中,所述地址生成单元计算由所述数据处理单元在所述串接中使用的所述偏移量。

28.根据权利要求27所述的设备,其中,所述地址生成单元通过对所述至少一个图像窗口的部分进行迭代来生成所述地址的第一集合,并且所述偏移量包括迭代不变参数。

29.一种用于处理数据的方法,包括:

经由控制子系统将参数的第一集合和参数的第二集合提供到数据处理子系统,所述数据处理子系统包括至少一个数据处理单元和至少一个地址生成单元;

将参数的所述第一集合和参数的所述第二集合存储在队列中;

基于参数的所述第一集合经由所述地址生成单元来生成在存储器中的地址的第一集合;

由所述数据处理单元使用所述地址的第一集合来处理数据;以及

当使用所述地址的第一集合来处理所述数据时,经由所述队列将所述参数的第二集合

提供到所述地址生成单元。

30.根据权利要求29所述的方法,其中,由所述数据处理单元处理的所述数据包括至少一个图像窗口,其中所述地址的第一集合中的至少一个对应于在所述存储器中的存储所述至少一个图像窗口的第一部分的位置;以及

所述方法还包括通过使用图像参数对所述至少一个图像窗口进行迭代来经由所述地址生成单元生成所述地址的第一集合,所述图像参数包括在所述存储器中的存储所述至少一个图像窗口的第二部分的地址。

31.根据权利要求30所述的方法,还包括通过所述数据处理子系统的标量处理器将所述图像参数提供给所述地址生成单元。

32.根据权利要求29所述的方法,其中,所述数据的处理包括由至少一个向量处理器处理向量数据。

33.根据权利要求30所述的方法,还包括:

经由所述数据处理子系统以增量的方式从所述存储器中检索数据;以及

经由所述数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据,所述偏移量指示所述未对齐的向量数据在所述存储器中关于所述增量是如何未对齐的。

34.根据权利要求33所述的方法,还包括经由所述地址生成单元来计算由所述数据处理单元在所述串接中使用的所述偏移量。

35.根据权利要求33所述的方法,还包括经由所述地址生成单元通过对所述至少一个图像窗口的部分进行迭代来生成所述地址的第一集合,并且所述偏移量包括迭代不变参数。

36.一种非临时计算机可读存储介质,其上存储有软件程序的指令集,所述指令集在由计算设备执行时使所述计算设备:

经由控制子系统将参数的第一集合和参数的第二集合提供到数据处理子系统,所述数据处理子系统包括至少一个数据处理单元和至少一个地址生成单元;

将参数的所述第一集合和参数的所述第二集合存储在队列中;

基于参数的所述第一集合经由所述地址生成单元来生成在存储器中的地址的第一集合;以及

当由所述数据处理单元使用所述地址的第一集合处理数据时,经由所述队列将参数的所述第二集合提供到所述地址生成单元。

37.根据权利要求36所述的非临时计算机可读存储介质,其中,由所述数据处理单元处理的数据包括至少一个图像窗口,所述地址的第一集合中的至少一个对应于在所述存储器中的存储所述至少一个图像窗口的第一部分的位置;以及

所述指令集进一步使所述计算设备经由所述地址生成单元通过使用图像参数对所述至少一个图像窗口进行迭代来生成所述地址的第一集合,所述图像参数包括在所述存储器中的存储所述至少一个图像窗口的第二部分的地址。

38.根据权利要求37所述的非临时计算机可读存储介质,其中,所述数据处理子系统包括将所述图像参数提供给所述地址生成单元的标量处理器。

39.根据权利要求36所述的非临时计算机可读存储介质,其中,所述数据处理单元包括至少一个向量处理器,并且由所述数据处理单元处理的所述数据包括向量数据。

40.根据权利要求37所述的非临时计算机可读存储介质,其中,所述指令集使所述计算设备:

经由所述数据处理子系统以增量的方式从所述存储器中检索数据,并且

经由所述数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据,所述偏移量指示所述未对齐的向量数据在所述存储器中关于所述增量是如何未对齐的。

41.根据权利要求40所述的非临时计算机可读存储介质,其中,所述指令集使所述计算设备经由所述地址生成单元来计算由所述数据处理单元在所述串接中使用的所述偏移量。

42.根据权利要求41所述的非临时计算机可读存储介质,其中,所述指令集使所述计算设备经由所述地址生成单元通过对所述至少一个图像窗口的部分进行迭代来生成所述地址的第一集合,并且所述偏移量包括迭代不变参数。

43.一种设备,包括:

至少一个输入寄存器;

至少一个可寻址寄存器;以及

至少一个处理单元,所述处理单元被配置为通过以下操作来生成直方图:

经由所述输入寄存器接收数据向量,所述数据向量包括多个数据点;

使所述多个数据点与所述直方图的多个直条匹配;

确定在所匹配的多个数据点中是否存在竞争;

如果在所匹配的多个数据点中存在竞争,则将所述数据向量的数据点的至少两个串行地分类到多个存储器位置中;以及

如果在所匹配的多个数据点中不存在竞争,则将所述数据向量的数据点的至少两个并行地分类到多个存储器位置中。

44.根据权利要求43所述的设备,其中,如果在所匹配的多个数据点中存在竞争则将所述数据向量的数据点的两个或更多个串行地分类到所述多个存储器位置中包括:

在第一时间段期间更新与所述直方图的至少一个直条相关联的直方图阵列;

生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于在所述第一时间段期间更新的经更新的直方图阵列;

基于所生成的第一地址位置将所述第一数据点写到所述可寻址寄存器;

在第二时间段期间更新所述直方图阵列;生成第二地址位置,所述第二地址位置对应于所述数据向量的第二数据点,所述第二地址位置基于在所述第二时间段期间更新的经更新的直方图阵列;以及

基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

45.根据权利要求43所述的设备,其中,如果在所匹配的多个数据点中不存在竞争则将所述数据向量的数据点的两个或更多个并行地分类包括:

在第一时间段期间更新与所述直方图的第一直条相关联的第一直方图阵列;

在所述第一时间段期间更新与所述直方图的第二直条相关联的第二直方图阵列;

生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于经更新的第一直方图阵列;

生成第二地址位置,所述第二地址位置对应于所述数据向量的第二数据点,所述第二地址位置基于经更新的第二直方图阵列;

在第二时间段期间基于所生成的第一地址位置来将所述第一数据点写到所述可寻址寄存器;以及

在所述第二时间段期间基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

46.根据权利要求43所述的设备,其中,确定在所匹配的多个数据点内是否存在地址竞争包括确定所述多个数据点中的两个或更多个是否与所述直方图的所述多个直条的公共直条匹配。

47.根据权利要求44所述的设备,其中,生成所述第一地址位置并生成所述第二地址位置包括将所述直方图阵列的值添加到与所述直方图的至少一个直条相关联的基地址。

48.根据权利要求47所述的设备,其中,生成所述第一地址位置包括将经更新的第一直方图阵列的值添加到与所述直方图的第一直条相关联的基地址,并且其中生成所述第二地址位置包括将经更新的第二直方图阵列的值添加到与所述直方图的第二直条相关联的基地址。

49.根据权利要求43所述的设备,其中,使所述多个数据点与所述直方图的多个直条匹配包括将所述多个数据点中的每个数据点关联到所述直方图的所述多个直条中的直条。

50.一种方法,包括:

经由输入寄存器接收数据向量,所述数据向量包括多个数据点;

使所述多个数据点与直方图的多个直条匹配;

确定在所匹配的多个数据点中是否存在竞争;

如果在所匹配的多个数据点中存在竞争,则将所述数据向量的数据点的至少两个串行地分类到多个存储器位置中;以及

如果在所匹配的多个数据点中不存在竞争,则将所述数据向量的数据点的至少两个并行地分类到多个存储器位置中。

51.根据权利要求50所述的方法,其中,如果确定在所匹配的多个数据点中存在竞争则将所述数据向量的数据点的两个或更多个串行地分类到所述多个存储器位置中包括:

在第一时间段期间更新与所述直方图的至少一个直条相关联的直方图阵列;

生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于在所述第一时间段期间更新的经更新的直方图阵列;

基于所生成的第一地址位置来将所述第一数据点写到可寻址寄存器;在第二时间段期间更新所述直方图阵列;

生成第二地址位置,所述第二地址位置对应于所述数据向量的第二数据点,所述第二地址位置基于在所述第二时间段期间更新的经更新的直方图阵列;以及

基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

52.根据权利要求50所述的方法,其中,如果在所匹配的多个数据点中不存在竞争则将所述数据向量的数据点的两个或更多个并行地分类包括:

在第一时间段期间更新与所述直方图的第一直条相关联的第一直方图阵列;

在所述第一时间段期间更新与所述直方图的第二直条相关联的第二直方图阵列;

生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于经更新的第一直方图阵列;

生成第二地址位置，所述第二地址位置对应于所述数据向量的第二数据点，所述第二地址位置基于经更新的第二直方图阵列；

在第二时间段期间基于所生成的第一地址位置将所述第一数据点写到可寻址寄存器；以及

在所述第二时间段期间基于所生成的第二地址位置将所述第二数据点写到所述可寻址寄存器。

53.根据权利要求50所述的方法，其中，确定在所匹配的多个数据点内是否存在地址竞争包括确定所述多个数据点中的两个或更多个是否与所述直方图的所述多个直条的公共直条匹配。

54.根据权利要求51所述的方法，其中，生成所述第一地址位置并生成所述第二地址位置包括将所述直方图阵列的值添加到与所述直方图的至少一个直条相关联的基地址。

55.根据权利要求54所述的方法，其中，生成所述第一地址位置包括将经更新的第一直方图阵列的值添加到与所述直方图的第一直条相关联的基地址，并且其中生成所述第二地址位置包括将经更新的第二直方图阵列的值添加到与所述直方图的第二直条相关联的基地址。

56.根据权利要求50所述的方法，其中，使所述多个数据点与所述直方图的多个直条匹配包括将所述多个数据点中的每个数据点关联到所述直方图的所述多个直条中的直条。

57.一种非临时计算机可读存储介质，其上存储有用于处理软件程序的指令的指令集，所述指令集在由计算设备执行时使所述计算设备：

经由输入寄存器接收数据向量，所述数据向量包括多个数据点；

使所述多个数据点与直方图的多个直条匹配；

确定在所匹配的多个数据点中是否存在竞争；

如果在所匹配的多个数据点中存在竞争，则将所述数据向量的数据点的至少两个串行地分类到多个存储器位置中；以及

如果在所匹配的多个数据点中不存在竞争，则将所述数据向量的至少两个数据点并行地分类到多个存储器位置中。

58.根据权利要求57所述的非临时计算机可读存储介质，其中，如果确定在所匹配的多个数据点中存在竞争则将所述数据向量的数据点的两个或更多个串行地分类到所述多个存储器位置中包括：

在第一时间段期间更新与所述直方图的至少一个直条相关联的直方图阵列；

生成第一地址位置，所述第一地址位置对应于所述数据向量的第一数据点，所述第一地址位置基于在所述第一时间段期间更新的经更新的直方图阵列；

基于所生成的第一地址位置来将所述第一数据点写到可寻址寄存器；

在第二时间段期间更新所述直方图阵列；

生成第二地址位置，所述第二地址位置对应于所述数据向量的第二数据点，所述第二地址位置基于在所述第二时间段期间更新的经更新的直方图阵列；以及

基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

59.根据权利要求57所述的非临时计算机可读存储介质，其中，如果在所匹配的多个数据点中不存在竞争则将所述数据向量的数据点的两个或更多个并行地分类包括：

第一时间段期间更新与所述直方图的第一直条相关联的第一直方图阵列；

在所述第一时间段期间更新与所述直方图阵列的第二直条相关联的第二直方图阵列；

生成第一地址位置，所述第一地址位置对应于所述数据向量的第一数据点，所述第一地址位置基于经更新的第一直方图阵列；

生成第二地址位置，所述第二地址位置对应于所述数据向量的第二数据点，所述第二地址位置基于经更新的第二直方图阵列；

在第二时间段期间基于所生成的第一地址位置将所述第一数据点写到可寻址寄存器；以及

在所述第二时间段期间基于所生成的第二地址位置将所述第二数据点写到所述可寻址寄存器。

60.根据权利要求57所述的非临时计算机可读存储介质，其中，确定在所匹配的多个数据点内是否存在地址竞争包括确定所述多个数据点中的两个或更多个是否与所述直方图的所述多个直条的公共直条匹配。

61.根据权利要求58所述的非临时计算机可读存储介质，其中，生成所述第一地址位置并生成所述第二地址位置包括将所述直方图阵列的值添加到与所述直方图的至少一个直条相关联的基地址。

62.根据权利要求61所述的非临时计算机可读存储介质，其中，生成所述第一地址位置包括将经更新的第一直方图阵列的值添加到与所述直方图的第一直条相关联的基地址，并且其中生成所述第二地址位置包括将经更新的第二直方图阵列的值添加到与所述直方图的第二直条相关联的基地址。

63.根据权利要求57所述的非临时计算机可读存储介质，其中，使所述多个数据点与所述直方图的多个直条匹配包括将所述多个数据点中的每个数据点关联到所述直方图的所述多个直条中的直条。

64.一种存储器设备，包括：

存储器模块，其是易失性的并且被配置为在降低功率模式中操作并且退出所述降低功率模式并且在正常功率模式中操作；

错误控制电路；以及

存储器控制器，其被配置为：

在所述存储器模块返回到所述正常功率模式之后将第一大小的数据单元写到所述存储器模块，其中所述数据单元位于第二大小的存储器分段内；

接收读命令以从所述存储器模块读取所述数据单元；

读取存储在所述存储器分段中的存储器分段内容；

将所述存储器分段内容发送给所述错误控制电路；以及

将读掩码传输到所述错误控制电路；

其中所述读掩码指示与所述数据单元有关的一个或更多个冗余位；

其中所述错误控制电路被配置为：接收所述存储器分段内容和所述读掩码；以及基于与所述数据单元有关的所述一个或更多个冗余位来检查所述数据单元中的一个或更多个错误，同时忽略与所述存储器分段内容的、与所述数据单元不同的一个或更多个部分有关的冗余位；以及

其中从所述降低功率模式退出并不跟随有初始化整个存储器模块。

65.根据权利要求64所述的存储器设备，其中，所述数据单元具有一个字节的宽度，以及其中所述存储器分段内容具有一个字的宽度。

66.一种用于错误校正的方法，所述方法包括：

在降低功率模式中操作存储器模块；

在从在所述降低功率模式中操作所述存储器模块退出之后，在正常功率模式中操作所述存储器模块；

将第一大小的数据单元写到所述存储器模块，其中所述数据单元位于第二大小的存储器分段内；

接收读命令以从所述存储器模块读取所述数据单元；

读取存储在所述存储器分段中的存储器分段内容；

将所述存储器分段内容发送给错误控制电路；

将读掩码传输到所述错误控制电路，其中所述读掩码指示与所述数据单元有关的一个或更多个冗余位；

由错误控制模块接收所述存储器分段内容和所述读掩码；以及

基于与所述数据单元有关的所述一个或更多个冗余位来检查所述数据单元中的一个或更多个错误，同时忽略与所述存储器分段内容的、与所述数据单元不同的一个或更多个部分有关的冗余位；

其中从在所述降低功率模式中操作所述存储器模块退出并不跟随有初始化全部的存储器模块。

67.根据权利要求66所述的方法，其中，所述数据单元具有一个字节的宽度，以及其中所述存储器分段内容具有一个字的宽度。

68.一种非临时计算机可读存储介质，其上存储有用于处理软件程序的多个线程的指令集，所述指令集在由计算设备执行时使所述计算设备：

在降低功率模式中操作存储器模块；

在从在所述降低功率模式中操作所述存储器模块退出之后，在正常功率模式中操作所述存储器模块；

将第一大小的数据单元写到所述存储器模块，其中所述数据单元位于第二大小的存储器分段内；

接收读命令以从所述存储器模块读取所述数据单元；

读取存储在所述存储器分段中的存储器分段内容；

将所述存储器分段内容发送给错误控制电路；

将读掩码传输到所述错误控制电路，其中所述读掩码指示与所述数据单元有关的一个或更多个冗余位；

由错误控制模块接收所述存储器分段内容和所述读掩码；以及

基于与所述数据单元有关的所述一个或更多个冗余位来检查所述数据单元中的一个或更多个错误，同时忽略与所述存储器分段内容的、与所述数据单元不同的一个或更多个部分有关的冗余位；

其中从在所述降低功率模式中操作所述存储器模块退出并不跟随有初始化全部的存

储器模块。

69.根据权利要求68所述的非临时计算机可读存储介质,其中,所述数据单元具有一个字节的宽度,以及其中所述存储器分段内容具有一个字的宽度。

70.一种多核处理器设备,包括:

多个处理核心,其被配置为使用桶式线程处理来执行软件程序的多个线程;

一个或更多个数据高速缓存;

一个或更多个指令存储器;以及

线程间通信模块,其包括原子计数器部分和用于在所述多个线程之间同步的数据锁部分。

71.根据权利要求70所述的多核处理器设备,其中,所述线程间通信模块不属于所述多个处理核心、所述一个或更多个数据高速缓存和所述一个或更多个指令存储器。

72.根据权利要求70所述的多核处理器设备,其中,所述数据锁部分包括多个数据锁,其中每个数据锁包括用于掩蔽对锁定所述数据锁的请求的掩码、用于随着时间的过去而改变所述掩码的掩码编码器、用于选择选定的掩码请求以锁定所述数据锁的第一优先级编码器以及用于选择锁定所述数据锁而不考虑所述掩码的请求的第二优先级编码器。

73.根据权利要求72所述的多核处理器设备,其中,所述第一优先级编码器和第二优先级编码器应用相同的仲裁方案。

74.根据权利要求72所述的多核处理器设备,其中,所述第一优先级编码器和第二优先级编码器应用不同的仲裁方案。

75.根据权利要求70所述的多核处理器设备,其中,所述多个处理核心包括数据锁请求器;并且其中每个数据锁请求器被配置为执行状态机,所述状态机用于控制锁定数据锁的请求的发送以及解锁所述数据锁的请求的发送以及用于选择性地阻止线程的执行。

76.根据权利要求70所述的多核处理器设备,其中,不同的数据锁指向不同的线程的起始。

77.根据权利要求70所述的多核处理器设备,其中,所述原子计数器部分包括存储器模块、判别器和处理器,所述存储器模块包括用于存储多个原子计数器值的多个条目,所述判别器用于按周期选择用于更新给定原子计数器的多达单个请求,以及所述处理器用于更新所述给定原子计数器。

78.根据权利要求77所述的多核处理器设备,其中,所述多个条目的数量小于所述数据锁部分的数据锁的数量。

79.一种方法,包括:

由多个处理核心使用桶式线程处理执行作为指令线程的多个线程;

其中所述多个线程的执行包括通过利用线程间通信模块的数据锁和原子计数器来在线程的执行之间同步。

80.根据权利要求79所述的方法,其中,所述线程间通信模块不属于所述多个处理核心、一个或更多个数据高速缓存以及一个或更多个指令存储器。

81.根据权利要求79所述的方法,其中,在所述线程的执行之间的同步包括:

管理多个数据锁,其中数据锁的管理包括通过掩码对用于锁定所述数据锁的请求进行掩蔽;

随时间的过去由掩码编码器改变所述掩码；

由第一优先级编码器选择用于锁定所述数据锁的选定的掩码请求，并且

由第二优先级编码器选择锁定所述数据锁而不考虑所述掩码的请求。

82.根据权利要求81所述的方法，包括由所述第一优先级编码器和第二优先级编码器应用相同的仲裁方案。

83.根据权利要求81所述的方法，包括由所述第一优先级编码器和第二优先级编码器应用不同的仲裁方案。

84.根据权利要求79所述的方法，其中，所述多个处理核心包括数据锁请求器，并且其中所述方法还包括由每个数据锁请求器执行状态机，所述状态机用于控制锁定数据锁的请求的发送和解锁所述数据锁的请求的第二发送并用于选择性地阻止线程的执行。

85.根据权利要求79所述的方法，其中，不同的数据锁指向不同线程的起始。

86.根据权利要求79所述的方法，其中，所述线程间通信模块的原子计数器部分包括存储器模块、判别器和处理器，所述存储器模块包括用于存储多个原子计数器值的多个条目、所述判别器用于按周期选择用于更新给定原子计数器的多达单个请求，所述处理器用于更新所述给定原子计数器。

87.根据权利要求86所述的方法，其中，所述多个条目的数量小于所述线程间通信模块的数据锁部分的数据锁的数量。

88.一种非临时计算机可读存储介质，其上存储有指令集，所述指令集用于由多个处理核心使用桶式线程处理来执行，所述指令包括作为指令线程的多个线程；

其中所述多个线程的执行包括通过利用线程间通信模块的数据锁和原子计数器在线程的执行之间同步。

89.根据权利要求88所述的非临时计算机可读存储介质，其中，所述线程间通信模块不属于所述多个处理核心、一个或更多个数据高速缓存和一个或更多个指令存储器。

90.根据权利要求88所述的非临时计算机可读存储介质，其中，在所述线程的执行之间的同步包括管理多个数据锁，其中数据锁的管理包括：

通过掩码对用于锁定所述数据锁的请求进行掩蔽；

随时间的过去由掩码编码器改变所述掩码；

由第一优先级编码器选择用于锁定所述数据锁的选定的掩码请求；以及

由第二优先级编码器选择锁定所述数据锁而不考虑所述掩码的请求。

91.根据权利要求90所述的非临时计算机可读存储介质，其具有存储在其上的用于由所述第一优先级编码器和第二优先级编码器应用相同的仲裁方案的指令集。

92.根据权利要求90所述的非临时计算机可读存储介质，其具有存储在其上的用于由所述第一优先级编码器和第二优先级编码器应用不同的仲裁方案的指令集。

93.根据权利要求88所述的非临时计算机可读存储介质，其中，所述多个处理核心包括数据锁请求器；其中所述非临时计算机可读存储介质具有存储在其上的用于由每个数据锁请求器执行状态机的指令集，该状态机用于控制锁定数据锁的请求的发送且用于解锁所述数据锁的请求的发送并用于选择性地阻止线程的执行。

94.根据权利要求88所述的非临时计算机可读存储介质，其中，不同的数据锁指向不同线程的起始。

95.根据权利要求88所述的非临时计算机可读存储介质,其中,所述原子计数器部分包括存储器模块、判别器以及处理器,所述存储器模块包括用于存储多个原子计数器值的多个条目,所述判别器用于按周期选择用于更新给定原子计数器的多达单个请求,所述处理器用于更新所述给定原子计数器。

96.根据权利要求95所述的非临时计算机可读存储介质,其中,所述多个条目的数量小于所述线程间通信模块的数据锁部分的数据锁的数量。

97.一种浮点处理器,包括:

加法和减法单元;以及

格式转换器;

其中:

所述格式转换器被配置为使用指数偏差来将整数转换成浮点数;并且

所述加法和减法单元包括被配置为当接收到作为非规格化数的输入操作数时生成无效异常的异常块。

98.根据权利要求97所述的浮点处理器,其中,所述异常块被配置为将超过大小阈值的输入变量视为无效数字。

99.根据权利要求97所述的浮点处理器,其被配置为:

通过应用加法或减法运算来计算结果;

当所述结果是非规格化数时生成异常;并且

输出调到零位的数字和异常标志。

100.根据权利要求97所述的浮点处理器,还包括:

指数比较器,其被配置为比较两个输入操作数的指数;

初始尾数计算器,其用于在基于所述两个输入操作数的指数在所述两个输入操作数的尾数之间对齐之后加上或减去所述两个输入操作数的所述尾数以提供初始尾数结果;以及

尾数格式化单元,其用于在所述初始尾数结果中找到前导设置位,并且用于当所述前导设置位不是所述初始尾数结果的最高有效位时移位所述初始尾数结果,使得所述前导设置位是结果的尾数的最高有效位。

101.根据权利要求100所述的浮点处理器,还包括:

指数选择器,其用于在两个输入操作数的指数之间进行选择以提供所选择的指数;

指数修改器,其用于修改所述选择的指数以便补偿所述初始尾数结果的移位以提供结果指数;以及

用于确定结果符号的符号选择器。

102.根据权利要求97所述的浮点处理器,其中,所述格式转换器被配置为当浮点数是非规格化数时将该浮点数转换为整数而不生成异常。

103.一种用于由浮点处理器执行浮点运算的方法,其中,所述方法包括:

由加法和减法单元加上或减去浮点格式的两个输入操作数,其中:

加上或减去包括:当所述两个输入操作数中的至少一个是非规格化数时,由所述加法和减法单元的异常块生成无效数字异常;以及

无效数字异常的生成包括输出默认值并设置异常标志;以及使用指数偏差来将整数转换为浮点数。

104.根据权利要求103所述的方法,包括由所述异常块将超过大小阈值的输入变量视为无效数字。

105.根据权利要求103所述的方法,包括:

通过应用加法或减法运算来计算结果;以及

当所述结果是非规格化数时生成异常并输出调到零位的数字和异常标志。

106.根据权利要求103所述的方法,其中,所述加上或减去包括:

由指数比较器比较两个输入操作数的指数;

在基于所述两个输入操作数的指数在所述两个输入操作数的尾数之间对齐之后由初始尾数计算器加上或减去所述两个输入操作数的尾数以提供初始尾数结果;以及

通过以下操作由尾数格式化单元进行对所述初始尾数结果的尾数格式化:在所述初始尾数结果中找到前导设置位,当所述前导设置位不是所述初始尾数结果的最高有效位时移位所述初始尾数结果,使得所述前导设置位是输出尾数的最高有效位。

107.根据权利要求106所述的方法,包括:

由指数选择器在两个输入操作数的指数之间进行选择以提供所选择的指数;

由指数修改器修改所选择的指数以便补偿所述初始尾数结果的移位以提供输出指数;以及

由符号选择器确定输出信号的符号。

108.根据权利要求103所述的方法,包括:当浮点数是非规格化数时,由格式转换器将该浮点数转换为整数而不生成异常。

109.一种非临时计算机可读存储介质,其上存储有指令集,所述指令集用于由浮点处理器执行浮点运算,所述运算包括:

由加法和减法单元加上或减去浮点格式的两个输入操作数;

由格式转换器使用指数偏差来将整数转换为浮点数;

其中:

所述加上或减去包括:当所述两个输入操作数中的至少一个是非规格化数时,生成无效数字异常;以及

所述无效数字异常的生成包括输出默认值并设置异常标志。

110.根据权利要求109所述的非临时计算机可读存储介质,具有存储在其上的用于由所述加法和减法单元的异常块将超过大小阈值的输入变量视为无效数字的指令集。

111.根据权利要求109所述的非临时计算机可读存储介质,具有存储在其上的用于进行以下操作的指令集:

通过应用加法或减法运算来计算结果;以及

当所述结果是非规格化数时生成异常并输出调到零位的数字和异常标志。

112.根据权利要求109所述的非临时计算机可读存储介质,其中,所述加法或减法包括:

由指数比较器比较两个输入操作数的指数;

由初始尾数计算器在基于所述两个输入操作数的指数在所述两个输入操作数的尾数之间对齐之后加上或减去所述两个输入操作数的尾数以提供初始尾数结果;以及

通过以下操作由尾数格式化单元进行对所述初始尾数结果的尾数格式化:在所述初始

尾数结果中找到前导设置位，当所述前导设置位不是所述初始尾数结果的最高有效位时移位所述初始尾数结果，使得所述前导设置位是输出尾数的最高有效位。

113.根据权利要求112所述的非临时计算机可读存储介质，具有存储在其上的用于进行以下操作的指令集：

由指数选择器在所述两个输入操作数的指数之间进行选择以提供所选择的指数；

由指数修改器修改所述选择的指数以便补偿所述初始尾数结果的移位以提供结果指数；以及

由符号选择器确定输出信号的符号。

114.根据权利要求109所述的非临时计算机可读存储介质，具有存储在其上的用于进行以下操作的指令集：当浮点数是非规格化数时由所述格式转换器将该浮点数转换为整数而不生成异常。

115.一种设备，包括：

至少一个输入寄存器；

至少一个可寻址寄存器；以及

至少一个处理单元，所述处理单元被配置为通过以下操作来生成直方图：

经由所述输入寄存器接收数据向量，所述数据向量包括多个数据点；

使所述多个数据点与所述直方图的多个直条匹配；以及

搜索当所述多个数据点包括属于同一直条的数据点时发生的竞争；

其中，当检测到所述竞争时，则串行地计算用于存储属于同一直条的所述数据点的给定存储器条目的地址，在所述给定存储器条目处存储属于同一直条的数据点；以及

其中，当没有检测到所述竞争时，则并行地计算用于存储所述多个数据点的某些存储器条目的地址，并在所述某些存储器条目处存储所述多个数据点。

116.一种方法，包括：

通过以下操作来生成直方图：

经由输入寄存器接收数据向量，所述数据向量包括多个数据点；

使所述多个数据点与所述直方图的多个直条匹配；以及

搜索当所述多个数据点包括属于同一直条的数据点时发生的竞争；

其中，当检测到所述竞争时，则串行地计算用于存储属于同一直条的所述数据点的给定存储器条目的地址，在所述给定存储器条目处存储属于同一直条的数据点；以及

其中，当没有检测到所述竞争时，则并行地计算用于存储所述多个数据点的某些存储器条目的地址，并在所述某些存储器条目处存储所述多个数据点。

117.一种非临时计算机可读存储介质，其上存储有用于处理软件程序的指令的指令集，所述指令集在由计算设备执行时使所述计算设备：

通过以下操作生成直方图：

经由输入寄存器接收数据向量，所述数据向量包括多个数据点；

使所述多个数据点与所述直方图的多个直条匹配；以及

搜索当所述多个数据点包括属于同一直条的数据点时发生的竞争；

其中，当检测到所述竞争时，则串行地计算用于存储属于同一直条的数据点的给定存储器条目的地址，在所述给定存储器条目处存储属于同一直条的数据点；以及

其中，当没有检测到所述竞争时，则并行地计算用于存储所述多个数据点的某些存储器条目的地址，并在所述某些存储器条目处存储所述多个数据点。

118.一种多核处理器设备，包括：

多个处理核心，其是硬件加速器内的全部处理核心，每个处理核心包括桶式调度器，所述桶式调度器被配置成使用桶式线程处理检索和执行与软件程序的多个指令线程相关联的一个或更多个指令；以及

共享数据高速缓存，其由全部的所述多个处理核心共享，并且是一级别数据高速缓存，并被配置成向所述多个处理核心中的每个处理核心发送并接收数据，所述共享数据高速缓存存储与所述多个指令线程中的至少一个中的第一指令的执行相关联的数据，所述共享数据高速缓存和所述多个处理核心以一致方式相互作用，而不依赖于所述多个处理核心中任一个的数据高速缓存。

119.一种用于处理多个指令线程的方法，所述方法包括：

使用多个处理核心执行所述多个指令线程，所述多个处理核心是硬件加速器内的全部处理核心，其中所述多个处理核心中的每个处理核心包括桶式调度器，所述桶式调度器使用桶式线程处理来执行与所述多个指令线程相关联的一个或更多个指令；以及

使用共享数据高速缓存来向所述多个处理核心中的每个处理核心发送和接收数据，所述共享数据高速缓存由全部的所述多个处理核心共享，并且是一级别数据高速缓存，所述共享数据高速缓存存储与所述多个指令线程中的至少一个中的第一指令的执行相关联的数据，所述共享数据高速缓存和所述多个处理核心以一致方式相互作用，而不依赖于所述多个处理核心中任一个的数据高速缓存。

120.一种非临时计算机可读存储介质，其上存储有用于处理软件程序的多个线程的指令的集合，所述指令在由计算设备执行时使所述计算设备：

使用多个处理核心执行所述多个指令线程，所述多个处理核心是硬件加速器内的全部处理核心，其中所述多个处理核心中的每个处理核心包括桶式调度器，所述桶式调度器使用桶式线程处理来执行与所述多个指令线程相关联的一个或更多个指令；以及

使用共享数据高速缓存来向所述多个处理核心中的至少两个处理核心发送和接收数据，所述共享数据高速缓存由全部的所述多个处理核心共享，并且是一级别数据高速缓存，所述共享数据高速缓存存储与所述多个指令线程中的至少一个中的第一指令的执行相关联的数据，所述共享数据高速缓存和所述多个处理核心以一致方式相互作用，而不依赖于所述多个处理核心中任一个的数据高速缓存。

121.一种多核处理器设备，包括：

多个处理核心，其是硬件加速器内的全部处理核心，每个处理核心包括桶式调度器，所述桶式调度器被配置成使用同时多线程处理检索和执行与软件程序的多个指令线程相关联的一个或更多个指令；以及

共享数据高速缓存，其由全部的所述多个处理核心共享，并且是一级别数据高速缓存，并被配置成向全部的所述多个处理核心发送并接收数据，所述共享数据高速缓存存储与所述多个指令线程中的至少一个中的第一指令的执行相关联的数据，所述共享数据高速缓存和所述多个处理核心以一致方式相互作用，而不依赖于所述多个处理核心中任一个的私有数据高速缓存。

使用多线程处理的多核处理器设备

[0001] 本申请是申请日为2016年6月10日，申请号为201680046182.3，发明名称为“使用多线程处理的多核处理器设备”的申请的分案申请。

[0002] 相关申请的交叉引用

[0003] 本申请要求下列专利的优先权：于2015年6月10日提交的美国临时专利序列号62/173,389；于2015年6月10日提交的美国临时专利序列号62/173,392；于2016年2月2日提交的美国临时专利序列号62/290,383；于2016年2月2日提交的美国临时专利序列号62/290,389；于2016年2月2日提交的美国临时专利序列号62/290,392；于2016年2月2日提交的美国临时专利序列号62/290,395；于2016年2月2日提交的美国临时专利序列号62/290,400；于2016年2月9日提交的美国临时专利序列号62/293,145；于2016年2月9日提交的美国临时专利序列号62/293,147；以及于2016年2月11日提交的美国临时专利62/293,908，其全部内容通过引用并入本文。

[0004] 本公开的背景

[0005] 许多类型的计算机体系结构都是可用的。处理器性能通常关于它的速度、效率、存储器消耗和功率消耗被评估。但是，根据这些因素，处理器的性能是高度依赖于上下文的。一个处理器在计算环境中可以具有优越的性能，其中在任何给定时间只有一个指令线程可用于被处理。相同的处理器可能在要求多个指令线程同时被处理的计算环境中展示低于标准的性能。因此，使处理器体系结构适应它将在其中操作的计算环境可以产生提高的性能。这个提高的性能可以包括例如增加的计算速度以及存储器和功率的更有效使用。

[0006] 存储器可被访问的速度常常与存储器的容量反向地相关。因此，使用相对快速但容量有限的存储器(例如，高速缓存)来为多处理器系统中的处理器存储共享数据的本地副本可能是有利的。这样做可以允许处理器与如果它们不得不依赖于从相对大的缓慢的中央存储器访问数据相比将快得多地执行操作。但是，当创建共享数据的本地副本时，存在这些副本将变得彼此不同步(或“不一致”)的重大风险。这是因为处理器常常运行修改在它们的本地高速缓存中的数据副本的过程，使存储在较慢的中央存储器中的原始数据变得过时。尤其是，在其他过程从主存储器访问该数据之前，过程可能不会用它们的变化来更新在外部或主存储器中的数据的原始副本。这可能导致失去的无功效和低效率。可能更糟糕的是，过程可以对过时或不正确的数据执行操作，且然后将不正确的结果写到主或外部存储器。然后，其他过程可以将不正确的数据保存到它们各自的处理器的本地高速缓存，对不正确的数据进行操作，并在系统中传播错误。这可产生很难或不可能检测到的错误。

[0007] 硬件和软件系统可用于解决这些高速缓存一致性问题。虽然这样的系统可以成功地处理数据一致性，但是它们常常将低效率引入到多处理器系统中。例如，“全高速缓存一致性”系统需要额外的硬件支持来适应数据的各种频繁的共享。全高速缓存一致性系统通常被设计为它们在数据共享可能发生的不同情况的适应中是灵活的。额外的灵活性需要额外的复杂硬件。那个额外的硬件有时可降低不强烈地或频繁地依赖于在不同处理器上运行的过程之间的数据共享的操作的速度。

[0008] 另外，处理器性能通常关于它的速度、效率、存储器消耗和功率消耗被评估。但是，

根据这些因素的处理器的性能是高度依赖于上下文的。一个处理器在其中在任何给定时间处只有一个指令线程可用于被处理的计算环境中可以具有优越的性能。相同的处理器可能在要求多个指令线程同时被处理的计算环境中展示低于标准的性能。因此，使处理器体系结构适应它将在其中操作的计算环境可以产生提高的性能。这个提高的性能可以包括例如增加的计算速度以及存储器和功率的更有效使用。

[0009] 当电子设备变得更小和更加移动时，对设备是功率高效的需要(即能够以最少量的功率消耗执行它的功能)变得越来越重要。可使电子设备更加功率高效的一种方式是对设备进行功率门控。功率门控可以指在设备闲置的时间段期间使设备完全或部分地的实践。在具有嵌入式存储器的电子设备中，每当设备掉电之后被加电时，对设备进行功率门控可使存储器变得毁坏。加电过程可能导致存储器错误，并且还可能导致与存储器交互的任何软件/固件的缓慢操作，因为软件可能必须等待存储器重新初始化。因此，为了允许在包括存储器的设备上的功率门控，可以修改存储器读操作以最小化存储器错误，而同时允许在存储器被加电之后快速读取存储器。

[0010] 本公开的概述

[0011] 可以提供任何权利要求的任何主题的任何组合。

[0012] 可以提供在任何附图中和/或在说明书中公开的任何方法和/或方法步骤的任何组合。

[0013] 可以提供在任何附图中和/或在说明书中公开的任何单元、设备和/或部件的任何组合。这种单元的非限制性例子包括收集单元、图像处理器等。

[0014] 本公开涉及利用桶式(barrel)线程处理和共享数据高速缓存来提高处理器的性能的多核计算机处理器。该系统可以例如被应用在其中存在具有很大程度上独立的迭代的多个软件循环的计算环境中。多核处理器可以实现优于其他多线程处理体系结构的性能增益，部分地因为每个核心使用桶式线程处理方式在线程上操作。桶式线程处理可以特别用于掩蔽或隐藏存储器时延(即，将它们对效率的影响最小化)。最小化时延的影响可以允许使用共享数据高速缓存。在多核体系结构中的桶式线程处理和共享数据高速缓存的组合因此可以提供改进的计算环境。

[0015] 本公开涉及可结合例如桶式线程处理和共享数据高速缓存来利用指令链编码的多核计算机处理器。经由指令链编码来减少处理多个相互依赖的指令所需的时钟周期的数量可能导致更快的处理时间。该系统可以被应用在其中存在具有独立迭代的多个软件循环的计算环境中。多核处理器可以实现优于其他多线程处理体系结构的性能增益，部分地因为每个核心使用桶式线程处理方式在线程上操作。桶式线程处理也可以掩蔽或隐藏存储器时延(即，将它们对效率的影响最小化)。最小化时延的影响可以允许使用更有效地访问的存储器，诸如共享数据高速缓存。在多核体系结构中的桶式线程处理、共享数据高速缓存和指令链编码的组合因此可以提供改进的计算环境。

[0016] 本公开涉及包括可促进在多个处理器设备中的数据的一致共享的硬件和/或软件的系统。该系统可包括对同时运行的过程实现对数据的受限访问。该系统还可以包括用于运行具有这种限制性数据访问的系统的相对有限或简化的硬件。另外，系统可以包括各种硬件实现，其存储保存到本地存储器的数据的副本的写状态的指示符。这些系统可以在可以利用桶式线程处理和共享数据高速缓存的多核计算机处理器上实现。该系统可以被应用

在其中存在具有独立迭代的多个软件循环的计算环境中。因此，在多核体系结构中的高速缓存一致性系统、桶式线程处理和共享数据高速缓存的组合可以提供改进的计算环境。

[0017] 本公开涉及用于从存储器检索数据的体系结构和方法。本公开可以结合向量处理器来使用以执行聚集型存储器检索，其除了别的以外还可以检索存储在多个存储器位置处的数据。该系统可以结合许多其他计算系统来使用。例如，该系统可以结合计算系统来使用，在该计算系统中存在具有独立迭代的多个软件循环。该体系结构和方法可以部分地通过促进可能导致增加的吞吐量和/或效率的近邻存储器访问来实现优于其他体系结构和方法的性能增益。

[0018] 本公开涉及可以使用具有地址生成能力的数据处理子系统的处理器。处理器可以是向量处理器。该系统可以被应用在例如其中存在具有独立迭代的多个软件循环的计算环境中。处理器可以实现优于其他体系结构的性能增益，部分地因为专用地址生成单元可以向数据处理子系统提供地址，用于在循环中的数据的高效评估。该系统可以在数据处理子系统中以更高的效率和更高的吞吐量实现指令的流水线评估。

[0019] 本公开涉及利用桶式线程处理和共享数据高速缓存来提高处理器的性能的多核计算机处理器。该系统可以被应用在例如其中存在具有独立迭代的多个软件循环的计算环境中。多核处理器可以实现优于其他多线程处理体系结构的性能增益，部分地因为每个核心使用桶式线程处理方式在线程上操作。桶式线程处理也可以用于掩蔽或隐藏存储器时延(即，将它们对效率的影响最小化)。最小化时延的影响可以允许使用更有效地访问的存储器，如共享数据高速缓存存储器。多核体系结构中的桶式线程处理和共享数据高速缓存器的组合因此可以提供改进的计算环境。

[0020] 本公开涉及一种可以利用读掩码的存储器设备，该读掩码可以指定要从中读取的字内的特定字节。以这种方式，当由读掩码指定的存储器的字节被读取时，可以检查与存储器中的特定字节相关联的奇偶校验位，以确定该字节是否由于软错误而被破坏。读掩码可以最小化由于存储器在掉电后未被初始化而引起的软错误，从而允许在掉电之后不需要存储器重新初始化，其可消耗相当大量的时间。因此，读掩码可以在允许存储器更有效地加电的同时最小化存储器软错误。

[0021] 根据本发明的实施例，可以提供存储器设备，其可以包括存储器模块、错误控制电路和存储器控制器。存储器设备可以包括存储器模块，其是易失性的并且被配置为在降低功率模式中操作、且退出降低功率模式并在正常功率模式中操作；错误控制电路；以及存储器控制器，其被配置为：在存储器模块从降低功率模式退出之后将第一大小的数据单元写到存储器模块，其中数据单元位于第二大小的存储器分段内，接收读命令以从存储器模块读取数据单元，读取存储在存储器分段中的存储器分段内容，将存储器分段内容发送给错误控制电路；以及将读掩码传输到错误控制电路，其中读掩码指示与数据单元有关的一个或更多个冗余位；其中错误控制电路被配置为：接收存储器分段内容和读掩码；以及基于与数据单元有关的一个或更多个冗余位来检查数据单元中的一个或更多个错误，同时忽略与存储器分段内容的与数据单元不同的一个或更多个部分有关的冗余位。从降低功率模式退出并不跟随有初始化整个存储器模块。

[0022] 读掩码也可以指示数据单元的位置。

[0023] 根据本发明的实施例，可以提供一种存储器设备，其可以包括存储器模块、错误控

制电路和存储器控制器，被配置为：在降低功率模式中操作，返回到正常功率模式，将数据字节写到存储器模块，其中数据字节位于存储在存储器模块上的数据字中，接收读命令以从存储器模块读取数据字节，读取存储在存储器模块上的数据字，将读掩码传输到错误控制电路，从错误控制电路接收软错误指示，其中软错误指示基于与数据字节相关联的一个或更多个奇偶校验位，并且其中与数据字节相关联的一个或更多个奇偶校验位使用与数据字相关联的一个或更多个奇偶校验位和所传输的读掩码来确定。

[0024] 根据本发明的实施例，可以提供一种用于实现存储器设备的方法，该方法可以包括：在降低功率模式中操作存储器控制器和存储器模块；将存储器控制器和存储器模块返回到正常功率模式；将数据字节写到存储器模块，其中数据字节位于存储在存储器模块上的数据字中；接收读命令以从存储器模块读取数据字节；读取存储在存储器模块上的数据字；将读掩码传输到错误控制电路；从错误控制电路接收软错误指示，其中软错误指示基于与数据字节相关联的一个或更多个奇偶校验位，并且其中与数据字节相关联的一个或更多个奇偶校验位使用与数据字相关联的一个或更多个奇偶校验位和所传输的读掩码来确定。

[0025] 根据本发明的实施例，可以提供一种非临时计算机可读存储介质，其上存储有用于处理软件程序的多个线程的指令集，该指令集在由计算设备执行时使计算设备：在降低功率模式中操作存储器控制器和存储器模块，使存储器控制器和存储器模块返回到正常功率模式，将数据字节写到存储器模块，其中数据字节位于存储在存储器模块上的数据字内，接收读命令以从存储器模块读取数据字节，读取存储在存储器模块上的数据字，将读掩码传输到错误控制电路，从错误控制电路接收软错误指示，其中软错误指示基于与数据字节相关联的一个或更多个奇偶校验位，且其中与数据字节相关联的一个或更多个奇偶校验位使用与数据字相关联的一个或更多个奇偶校验位和所传输的读掩码来确定。

[0026] 可以提供一种设备，其包括：至少一个输入寄存器、至少一个可寻址寄存器以及至少一个处理单元，该处理单元被配置为通过以下操作来生成直方图：经由输入寄存器接收数据向量，数据向量包括多个数据点；使多个数据点与直方图的多个直条(bin)匹配；搜索当多个数据点包括属于同一直条的数据点时出现的竞争；当检测到竞争时，则串行地计算用于存储属于同一直条的数据点的给定存储器条目的地址并在给定存储器条目处存储属于同一直条的数据点；以及当没有检测到竞争时，则并行地计算用于存储多个数据点的某些存储器条目的地址，并在某些存储器条目处存储多个数据点。

[0027] 可以提供一种方法，该方法包括通过以下操作来生成直方图：经由输入寄存器接收数据向量，数据向量包括多个数据点；使多个数据点与直方图的多个直条匹配；搜索当多个数据点包括属于同一直条的数据点时出现的竞争；当检测到竞争时，则串行地计算用于存储属于同一直条的数据点的给定存储器条目的地址并在给定存储器条目处存储属于同一直条的数据点；以及当没有检测到竞争时，则并行地计算用于存储多个数据点的某些存储器条目的地址，并在某些存储器条目处存储多个数据点。

[0028] 可以提供一种非临时计算机可读存储介质，其上存储有用于处理软件程序的指令的指令集，该指令集在由计算设备执行时使计算设备通过以下操作生成直方图：经由输入寄存器接收数据向量，数据向量包括多个数据点；使多个数据点与直方图的多个直条匹配；搜索当多个数据点包括属于同一直条的数据点时出现的竞争；当检测到竞争时，则串行地计算用于存储属于同一直条的数据点的给定存储器条目的地址并在给定存储器条目处存

储属于同一直条的数据点;以及当没有检测到竞争时,则并行地计算用于存储多个数据点的某些存储器条目的地址,并在某些存储器条目处存储多个数据点。

[0029] 根据本发明的实施例,还涉及以下内容:

[0030] 1)一种多核处理器设备,包括:多个处理核心,其配置成使用桶式线程处理来执行软件程序的多个线程;以及共享数据高速缓存,其配置成将数据发送到所述多个处理核心中的至少两个并从所述多个处理核心中的至少两个接收数据,所述共享数据高速缓存存储与在所述多个线程中的至少一个中的第一指令的执行相关联的数据,所述共享数据高速缓存和所述多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。

[0031] 2)根据1)所述的设备,其中,所述多个处理核心包括:多个线程寄存器文件,其中所述多个线程寄存器文件中的至少一个与所述多个线程中的线程相关联;指令存储器,其中所述指令存储器存储与所述多个线程相关联的一个或更多个指令;一个或更多个算术逻辑单元以及桶式调度器,其中所述桶式调度器接收第一时钟信号,从所述指令存储器加载所述一个或更多个指令中的第一指令,并且通过以下操作来执行所述第一指令:从第一线程寄存器文件检索数据,其中所述第一线程寄存器文件与和所述第一指令相关联的线程相关联;以及将接收到的数据发送到所述一个或更多个算术逻辑单元用于处理。

[0032] 3)根据2)所述的设备,其中,所述桶式调度器接收第二时钟信号,从所述指令存储器加载所述一个或更多个指令中的第二指令,并且通过以下操作来执行所述第二指令:从第二线程寄存器文件检索数据,其中所述第二线程寄存器文件与和所述第二指令相关联的线程相关联;以及将接收到的数据发送到所述一个或更多个算术逻辑单元用于处理。

[0033] 4)根据3)所述的设备,其中,所述多个线程寄存器文件中的每个线程寄存器文件被配置为加载与和所述线程寄存器文件相关联的线程相关联的数据。

[0034] 5)根据3)所述的设备,其中,所述多个处理核心同时执行所述多个线程。

[0035] 6)根据1)所述的设备,其中,所述多个处理核心中的一个或更多个向所述共享数据高速缓存传输数据并从所述共享数据高速缓存接收数据,并且其中所述共享数据高速缓存被配置为存储用于执行所述多个线程的数据。

[0036] 7)根据1)所述的设备,包括被配置为执行单线程程序代码的串行处理器。

[0037] 8)一种用于处理多个线程的方法,所述多个线程是指令程序线程,所述方法包括:使用多个处理核心执行所述多个线程,其中所述多个处理核心使用桶式线程处理来执行所述多个线程;以及,使用共享数据高速缓存将数据发送到所述多个处理核心中的至少两个并从所述多个处理核心中的至少两个接收数据,所述共享数据高速缓存存储与在所述多个线程中的至少一个中的第一指令的执行相关联的数据,所述共享数据高速缓存和所述多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。

[0038] 9)根据8)所述的方法,其中,所述方法包括:在所述多个处理核心的桶式调度器处接收第一时钟信号;从指令存储器加载第一指令,其中所述指令存储器存储与所述多个线程相关联的一个或更多个指令;并且通过以下操作来执行所述第一指令:从第一线程寄存器文件检索数据,其中所述第一线程寄存器文件与和所述第一指令相关联的指令线程相关联,并且将接收到的数据发送到一个或更多个算术逻辑单元用于处理。

[0039] 10)根据9)所述的方法,其中,所述桶式调度器接收第二时钟信号,从所述指令存储器加载所述一个或更多个指令中的第二指令,并且通过以下操作来执行所述第二指令:

从第二线程寄存器检索数据，其中第二线程寄存器文件与和所述第二指令相关联的线程相关联，并将接收到的数据发送到所述一个或更多个算术逻辑单元用于处理。

[0040] 11)根据10)所述的方法，其中，多个线程寄存器文件中的每个线程寄存器文件加载与和所述线程寄存器文件相关联的线程相关联的数据。

[0041] 12)根据10)所述的方法，其中，所述多个处理核心同时执行所述多个线程。

[0042] 13)根据8)所述的方法，其中，所述多个处理核心中的一个或更多个向所述共享数据高速缓存传输数据并从所述共享数据高速缓存接收数据，并且其中所述共享数据高速缓存被配置为存储对所述多个线程的执行所必需的数据。

[0043] 14)根据8)所述的方法，包括使用串行处理器执行一个或更多个单线程程序代码。

[0044] 15)一种非临时计算机可读存储介质，其上存储有用于处理软件程序的多个线程的指令集，所述指令集在由计算设备执行时使所述计算设备：使用多个处理核心执行所述多个线程，其中所述多个处理核心使用桶式线程处理来执行所述多个线程，并使用共享数据高速缓存将数据发送到所述多个处理核心中的至少两个并从所述多个处理核心中的至少两个接收数据，所述共享数据高速缓存存储与在所述多个线程中的至少一个中的第一指令的执行相关联的数据，所述共享数据高速缓存和所述多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。

[0045] 16)根据15)所述的非临时计算机可读存储介质，其中，所述计算设备被使得：在所述多个处理核心的桶式调度器处接收第一时钟信号，从指令存储器加载第一指令，其中所述指令存储器存储与所述多个线程相关联的一个或更多个指令，通过以下操作来执行所述第一指令：从第一线程寄存器检索数据，其中所述第一线程寄存器与和所述第一指令相关联的指令线程相关联，并且将接收到的数据发送到一个或更多个算术逻辑单元用于处理。

[0046] 17)根据16)所述的非临时计算机可读存储介质，其中，所述桶式调度器接收第二时钟信号，从所述指令存储器加载所述一个或更多个指令中的第二指令，并且通过以下操作来执行所述第二指令：从第二线程寄存器检索数据，其中所述第二线程寄存器与和所述第二指令相关联的指令线程相关联，并将接收到的数据发送到所述一个或更多个算术逻辑单元用于处理。

[0047] 18)根据17)所述的非临时计算机可读存储介质，其中，多个线程寄存器中的每个线程寄存器加载与和所述线程寄存器相关联的指令线程相关联的数据。

[0048] 19)根据17)所述的非临时计算机可读存储介质，其中，所述多个处理核心同时执行所述多个线程。

[0049] 20)根据15)所述的非临时计算机可读存储介质，其中，所述多个处理核心中的一个或更多个向所述共享数据高速缓存传输数据并从所述共享数据高速缓存接收数据，并且其中所述共享数据高速缓存被配置为存储用于执行所述多个线程的数据。

[0050] 21)根据15)所述的非临时计算机可读存储介质，其中，所述计算设备被使得：使用串行处理器执行一个或更多个单线程程序代码。

[0051] 22)一种多核处理器设备，包括：多个处理核心，其配置成使用同时多线程处理来执行软件程序的多个线程；以及共享数据高速缓存，其配置成将数据发送到所述多个处理核心中的至少两个并从所述多个处理核心中的至少两个接收数据，所述共享数据高速缓存存储与在所述多个线程中的至少一个中的第一指令的执行相关联的数据，所述共享数据高

速缓存和所述多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。

[0052] 23)一种多核处理器设备,包括:多个处理核心,其被配置为使用桶式线程处理来执行软件程序的多个指令线程,其中所述多个处理核心被配置为通过下列操作来执行与所述多个指令线程的第一线程相关联的链式指令:在第一时间段期间将所述链式指令加载到第一指令存储器,其中所述链式指令包括独立指令部分和从属指令部分;在所述第一时间段期间将所述独立指令部分和与所述独立指令部分相关联的一个或更多个操作数传输到一个或更多个执行单元;在所述第一时间段期间将所述从属指令部分传输到链式指令寄存器;在所述第一时间段期间由所述一个或更多个执行单元执行所述独立指令部分;在所述第一时间段期间将所述独立指令部分的结果写入到第一中间寄存器;在第二时间段期间将所述从属指令部分、与所述从属指令部分相关联的一个或更多个操作数以及所述独立指令部分的结果传输到所述一个或更多个执行单元;以及在所述第二时间段期间执行所述从属指令部分。

[0053] 24)根据23)所述的设备,其中,所述设备在所述第二时间段期间与所述链式指令的所述从属指令部分的执行并行地处理与所述多个指令线程中的第二线程相关联的指令。

[0054] 25)根据24)所述的设备,其中,与所述从属指令部分相关联的一个或更多个操作数包括输入操作数,并且其中执行所述从属指令部分包括用所述从属指令部分的执行的结果覆写所述输入操作数。

[0055] 26)根据24)所述的设备,包括复用器,所述复用器被配置为确定所述一个或更多个执行单元中的哪个将执行所述链式指令的所述从属指令部分,以及所述一个或更多个执行单元中的哪个将执行与所述第二线程相关联的所述指令。

[0056] 27)根据24)所述的设备,包括共享数据高速缓存,所述共享数据高速缓存被配置为向所述多个处理核心中的至少两个发送数据并从所述多个处理核心中的至少两个接收数据,所述共享数据高速缓存存储与所述多个指令线程中的至少一个中的指令的执行相关联的数据,所述共享数据高速缓存和所述多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。

[0057] 28)根据24)所述的设备,其中,与所述从属指令部分相关联的所述一个或更多个操作数在所述第二时间段期间被存储在第一寄存器文件中,并且其中与所述第二线程相关联的所述指令包括在所述第二时间段期间存储在第二寄存器文件中的一个或更多个操作数。

[0058] 29)根据23)所述的设备,包括寄存器文件、链式指令寄存器和与所述多个指令线程中的每个线程相关联的中间寄存器。

[0059] 30)根据23)所述的设备,其中,所述设备在所述第二时间段期间与所述链式指令的所述从属指令部分的执行并行地处理与所述第一线程相关联的另一个独立指令部分。

[0060] 31)根据30)所述的设备,其中,每个处理核心包括用于执行所述第一线程的独立指令部分的第一执行单元和用于执行所述第一线程的从属指令部分的第二执行单元。

[0061] 32)一种用于在被配置为执行多个指令线程的桶式线程化处理器中执行链式指令的方法,所述方法包括:在第一时间段期间将所述链式指令加载到第一指令存储器,其中所述链式指令包括独立指令部分和从属指令部分;在所述第一时间段期间将所述独立指令部分和与所述独立指令部分相关联的一个或更多个操作数传输到一个或更多个执行单元;在

所述第一时间段期间将所述从属指令部分传输到链式指令寄存器;在所述第一时间段期间由所述一个或更多个执行单元执行所述独立指令部分;在所述第一时间段期间将所述独立指令部分的结果写到第一中间寄存器;在第二时间段期间将所述从属指令部分、与所述从属指令部分相关联的一个或更多个操作数以及所述独立指令部分的结果传输到所述一个或更多个执行单元;以及在所述第二时间段期间执行所述从属指令部分。

[0062] 33)根据32)所述的方法,其中,所述链式指令与所述多个指令线程的第一线程相关联,以及其中所述方法包括在所述第二时间段期间与所述链式指令的所述从属指令部分的执行并行地处理与所述多个指令线程中的第二线程相关联的指令。

[0063] 34)根据33)所述的方法,其中,与所述从属指令部分相关联的一个或更多个操作数包括输入操作数,并且其中执行所述从属指令部分包括用所述从属指令部分的执行的结果覆写所述输入操作数。

[0064] 35)根据33)所述的方法,包括确定所述一个或更多个执行单元中的哪个将执行所述链式指令的所述从属指令部分以及与所述第二线程相关联的所述指令。

[0065] 36)根据33)所述的方法,包括使用共享数据高速缓存向多个处理核心中的至少两个发送数据并从所述多个处理核心中的至少两个接收数据,所述共享数据高速缓存存储与所述多个指令线程中的至少一个中的第一指令的执行相关联的数据,所述共享数据高速缓存和所述多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。

[0066] 37)根据33)所述的方法,其中,与所述从属指令部分相关联的所述一个或更多个操作数在所述第二时间段期间被存储在第一寄存器文件中,并且其中与所述第二线程相关联的指令部分包括在所述第二时间段期间存储在第二寄存器文件中的一个或更多个操作数。

[0067] 38)根据37)所述的方法,其中,所述多个指令线程中的每个线程具有它自己的寄存器文件、链式指令寄存器和与它相关联的中间寄存器。

[0068] 39)根据33)所述的方法,包括在所述第二时间段期间与所述链式指令的所述从属指令部分的执行并行地处理与所述第一线程相关联的另一个独立指令部分。

[0069] 40)根据39)所述的方法,其中,每个处理核心包括第一执行单元和第二执行单元,其中所述方法包括由所述第一执行单元执行所述第一线程的独立指令部分以及由所述第二执行单元执行所述第一线程的从属指令部分。

[0070] 41)一种非临时计算机可读存储介质,其上存储有用于处理软件程序的链式指令的指令集,所述指令集在由计算设备执行时使所述计算设备:在第一时间段期间将所述链式指令加载到第一指令存储器,其中所述链式指令包括独立指令部分和从属指令部分;在所述第一时间段期间将所述独立指令部分和与所述独立指令部分相关联的一个或更多个操作数传输到一个或更多个执行单元以用于执行;在所述第一时间段期间将所述从属指令部分传输到链式指令寄存器;在所述第一时间段期间由所述一个或更多个执行单元执行所述独立指令部分;在所述第一时间段期间将所述独立指令部分的结果写到第一中间寄存器;在第二时间段期间将所述从属指令部分、与所述从属指令部分相关联的一个或更多个操作数以及所述独立指令部分的结果传输到所述一个或更多个执行单元;以及在所述第二时间段期间执行所述从属指令部分。

[0071] 42)根据41)所述的非临时计算机可读存储介质,其中,所述链式指令与多个线程

的第一线程相关联,以及其中所述计算设备被使得在所述第二时间段期间与所述链式指令的所述从属指令部分的执行并行地处理与所述多个线程中的第二线程相关联的指令。

[0072] 43)根据42)所述的非临时计算机可读存储介质,其中,与所述从属指令部分相关联的一个或更多个操作数包括输入操作数,并且其中执行所述从属指令部分包括用所述从属指令部分的执行的结果覆写所述输入操作数。

[0073] 44)根据42)所述的非临时计算机可读存储介质,其中,所述计算设备被使得确定所述一个或更多个执行单元中的哪个将执行所述链式指令的所述从属指令部分以及与所述第二线程相关联的所述指令。

[0074] 45)根据42)所述的非临时计算机可读存储介质,其中,所述计算设备被使得使用共享数据高速缓存向多个处理核心中的至少两个发送数据并从所述多个处理核心中的至少两个接收数据,所述共享数据高速缓存存储与所述多个线程中的至少一个中的第一指令的执行相关联的数据,所述共享数据高速缓存和所述多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。

[0075] 46)根据42)所述的非临时计算机可读存储介质,其中,与所述从属指令部分相关联的所述一个或更多个操作数在所述第二时间段期间被存储在第一寄存器文件中,并且其中与所述第二线程相关联的所述指令包括在所述第二时间段期间存储在第二寄存器文件中的一个或更多个操作数。

[0076] 47)根据42)所述的非临时计算机可读存储介质,其中,所述多个线程中的每个线程具有它自己的寄存器文件、链式指令寄存器和与它相关联的中间寄存器。

[0077] 48)一种设备,包括:执行多个过程的多个处理器;与所述多个处理器相关联的多个高速缓存,所述多个高速缓存至少包括与第一处理器相关联的第一高速缓存;由所述多个处理器访问并存储多个变量的存储器,其中所述多个处理器通过下列操作来执行所述多个过程中的至少一个:通过将所述多个变量的子集存储到所述第一高速缓存来在所述第一处理器上发起所述多个过程的第一过程;通过将所述子集的更新副本写到所述存储器来终止所述第一过程;以及在所述第一过程的执行期间限制由除了所述第一过程以外的所述多个过程对所述子集的访问。

[0078] 49)根据48)所述的设备,其中:所述多个过程包括第二过程,在所述第一处理器上的所述第一过程的执行由在第二处理器上执行的所述第二过程启动,终止所述第一过程的所述执行包括所述第一过程使所述第一高速缓存无效并向所述第二过程用信号通知完成,并且在接收到所述信号时,所述第二过程访问所述子集。

[0079] 50)根据48)所述的设备,其中,所述设备包括指示符,所述指示符提供至少第一状态和第二状态的指示,在所述第一状态中所述第一处理器未对在所述第一高速缓存中的数据的字节进行写入,以及在所述第二状态中所述第一处理器已经对在所述第一高速缓存中的数据的字节进行写入。

[0080] 51)根据50)所述的设备,其中,所述指示符包括单个位。

[0081] 52)根据50)所述的设备,其中,所述指示符被存储在所述第一高速缓存中。

[0082] 53)根据50)所述的设备,其中,所述设备布置成通过下列操作通过写所述子集的更新副本来终止所述第一过程的执行:生成对包括在所述第一高速缓存中的数据的字节的所述第一高速缓存的行的请求,在所述第一状态期间,将所述行写到所述存储器使得与在

所述第一高速缓存中的高速缓存数据的字节对应的在所述存储器中的高速缓存数据的字节实质上是不变的，以及在所述第二状态期间将所述行写到所述存储器以覆写对应于在所述第一高速缓存中的数据的字节的在所述存储器中的数据的字节。

[0083] 54)根据48)所述的设备，其中：所述多个处理器包括多个核心，并且所述多个核心共享所述多个高速缓存。

[0084] 55)根据48)所述的设备，其中，I/O一致性协议使所述设备能够与另一设备通过接口连接。

[0085] 56)一种用于在具有多个处理器和多个高速缓存的设备中执行多个过程的方法，所述多个高速缓存包括至少第一高速缓存，所述方法包括：将多个变量存储在由所述多个处理器访问的存储器中，通过将所述多个变量的子集存储到所述第一高速缓存来在第一处理器上发起所述多个过程的第一过程，通过将所述子集的更新副本写到所述存储器来终止所述第一过程，以及在所述第一过程的执行期间限制由除了所述第一过程以外的所述多个过程对所述子集的访问。

[0086] 57)根据56)所述的方法，包括：在第二处理器上执行所述多个过程的第二过程，经由所述第二过程启动所述第一过程，通过使所述第一高速缓存无效并且向所述第二过程发信号通知完成以及在接收到所述信号时通过所述第二过程来访问所述子集来终止所述第一过程。

[0087] 58)根据56)所述的方法，包括：提供指示符，所述指示符提供至少第一状态和第二状态的指示，在所述第一状态中所述第一处理器未对在所述第一高速缓存中的数据的字节进行写入，以及在所述第二状态中所述第一处理器已经对在所述第一高速缓存中的数据的字节进行写入。

[0088] 59)根据58)所述的方法，其中，所述指示符包括单个位。

[0089] 60)根据58)所述的方法，其中，通过写所述子集的更新副本来终止所述第一过程的执行包括：生成对包括在所述第一高速缓存中的数据的字节的所述第一高速缓存的行的请求，在所述第一状态期间，将所述行写到外部存储器使得与在所述第一高速缓存中的高速缓存数据的字节对应的在所述存储器中的高速缓存数据的字节实质上是不变的，并且在所述第二状态期间将所述行写到所述存储器以覆写对应于在所述第一高速缓存中的数据的字节的在所述存储器中的数据的字节。

[0090] 61)根据56)所述的方法，所述方法包括在所述多个处理器的多个核心当中共享所述多个高速缓存。

[0091] 62)根据56)所述的方法，包括使所述设备经由I/O一致性协议来与另一个设备通过接口连接。

[0092] 63)一种非临时计算机可读存储介质，其上存储有用于处理软件程序的指令的指令集，所述指令集在由具有多个处理器和多个高速缓存的设备执行时使所述设备：将多个变量存储在由所述多个处理器访问的存储器中，通过将所述多个变量的子集存储到第一高速缓存来在第一处理器上发起多个过程中的第一过程，所述第一高速缓存是所述设备中的多个高速缓存中的一个，通过将所述子集的更新副本写到所述存储器来终止所述第一过程，并且在所述第一过程的执行期间限制由除了所述第一过程以外的所述多个过程对所述子集的访问。

[0093] 64)根据63)所述的非临时计算机可读存储介质,其中,所述指令使所述设备:在第二处理器上执行所述多个过程中的第二过程,由所述第二过程启动所述第一过程,通过使所述第一高速缓存无效并且向所述第二过程用信号通知完成,以及当接收到所述信号时通过所述第二过程来访问所述子集来终止所述第一过程。

[0094] 65)根据63)所述的非临时计算机可读存储介质,其中,所述指令使所述设备:存储指示符,所述指示符提供至少第一状态和第二状态的指示,在所述第一状态中所述第一处理器未对在所述第一高速缓存中的数据的字节进行写入,以及在所述第二状态中所述第一处理器已经对在所述第一高速缓存中的数据的字节进行写入。

[0095] 66)根据65)所述的非临时计算机可读存储介质,其中,所述指示符包括单个位。

[0096] 67)根据65)所述的非临时计算机可读存储介质,其中,所述指示符被存储在所述第一高速缓存中。

[0097] 68)根据65)所述的非临时计算机可读存储介质,其中,通过写所述子集的更新副本来终止所述第一过程的执行包括:生成对包括在所述第一高速缓存中的数据的字节的所述第一高速缓存的行的请求,在所述第一状态期间,将所述行写到所述存储器使得与在所述第一高速缓存中的高速缓存数据的字节对应的在所述存储器中的高速缓存数据的字节实质上是不变的,并且在所述第二状态期间将所述行写到所述存储器以覆写与在所述第一高速缓存中的数据的字节对应的在外部存储器中的数据的字节。

[0098] 69)根据63)所述的非临时计算机可读存储介质,其中,所述指令使所述设备在所述多个处理器的多个核心当中共享所述多个高速缓存。

[0099] 70)根据63)所述的非临时计算机可读存储介质,其中,所述指令使所述设备经由I/O一致性协议与另一个设备通过接口连接。

[0100] 71)一种设备,包括:一个或更多个存储器组、一个或更多个寄存器、将所述一个或更多个存储器组与所述一个或更多个寄存器连接的连接器、以及存储用于访问来自所述一个或更多个存储器组的数据的一组地址的一个或更多个队列,所述一组地址基于所选择的地址是否与在数据请求中的其他地址是几乎相同的来从所述数据请求进行选择。

[0101] 72)根据71)所述的设备,其中:所述选择包括基于指示符矩阵的对角元素来选择所述一组地址,以及所述对角元素指示在所述数据请求中的地址是最低索引近邻还是没有近邻。

[0102] 73)根据72)所述的设备,其中,所述选择包括选择下列项中的至少一个:与所述数据请求中的任何其他地址不是几乎相同的地址,或者与所述数据请求中的一组地址几乎相同并且具有在所述一组地址中的地址的最低索引的地址。

[0103] 74)根据72)所述的设备,其中,所述选择包括确定所述数据请求中的所述地址中的任何地址是否对应于存储在所述一个或更多个存储器组中的同一个存储器组中的同一个字。

[0104] 75)根据72)所述的设备,其中,所述连接器基于所述指示符矩阵向所述一个或更多个寄存器提供数据。

[0105] 76)根据75)所述的设备,其中,基于所述指示符矩阵向所述一个或更多个寄存器提供数据包括:经由所述指示符矩阵确定与所述数据请求中的任何其它地址不几乎相同的或者与所述数据请求中的一组其它地址几乎相同并且具有在所述一组其它地址中的地址

的最低索引的在所述数据请求中的一个或更多个地址，以及将与所确定的一个或更多个地址相关联的数据提供给所述一个或更多个寄存器。

[0106] 77)根据76)所述的设备，包括一个或更多个复用器，并且其中所述确定由所述一个或更多个复用器执行。

[0107] 78)一种用于处理数据的方法，包括：确定指示数据请求中的地址是否与在所述数据请求中的其他地址几乎相同的指示符矩阵，基于所述指示符矩阵来选择用于访问来自一个或更多个存储器组的数据的一组地址，将所选择的一组地址存储在一个或更多个队列中，以及基于所述一个或更多个队列从所述一个或更多个存储器组检索对应于所述选择的一组地址的数据。

[0108] 79)根据78)所述的方法，其中，所述选择包括基于所述指示符矩阵的对角元素来选择所述一组地址，以及所述对角元素指示在所述数据请求中的地址是最低索引近邻还是没有近邻。

[0109] 80)根据78)所述的方法，其中，所述选择包括选择下列项中的至少一个：与所述数据请求中的任何其他地址不几乎相同的地址，或者与所述数据请求中的一组地址几乎相同并且具有在所述一组地址中的地址的最低索引的地址。

[0110] 81)根据78)所述的方法，其中，所述选择包括确定所述数据请求中的所述地址中的任何地址是否对应于存储在所述一个或更多个存储器组中的同一个存储器组中的同一个字。

[0111] 82)根据78)所述的方法，还包括经由连接器基于所述指示符矩阵向一个或更多个寄存器提供数据。

[0112] 83)根据82)所述的方法，其中，基于所述指示符矩阵向所述一个或更多个寄存器提供数据包括：经由所述指示符矩阵确定与所述数据请求中的任何其它地址不几乎相同的或者与所述数据请求中的一组其它地址几乎相同并且具有在所述一组其它地址中的所述地址的最低索引的在所述数据请求中的一个或更多个地址，以及将与所确定的一个或更多个地址相关联的数据提供给所述一个或更多个寄存器。

[0113] 84)根据83)所述的方法，其中，所述确定所述一个或更多个地址由一个或更多个复用器执行。

[0114] 85)一种非临时计算机可读存储介质，其上存储有软件程序的指令集，所述指令集在由计算设备执行时使所述计算设备：确定指示数据请求中的地址是否与所述数据请求中的其他地址几乎相同的指示符矩阵，基于所述指示符矩阵来选择用于访问来自一个或更多个存储器组的数据的一组地址，将所选择的一组地址存储在一个或更多个队列中，以及基于所述一个或更多个队列来从所述一个或更多个存储器组检索相应于所述选择的一组地址的数据。

[0115] 86)根据85)所述的非临时计算机可读存储介质，其中：所述选择包括基于所述指示符矩阵的对角元素来选择所述一组地址，以及所述对角元素指示所述数据请求中的地址是最低索引近邻还是没有近邻。

[0116] 87)根据85)所述的非临时计算机可读存储介质，其中，所述指令集使所述计算设备选择下列项中的至少一个：与所述数据请求中的任何其它地址不几乎相同的地址，或者与所述数据请求中的一组地址几乎相同并且具有所述一组地址中的地址的最低索引的地

址。

[0117] 88)根据85)所述的非临时计算机可读存储介质,其中,所述选择包括确定所述数据请求中的地址中的任何地址是否对应于存储在所述一个或更多个存储器组中的同一个存储器组中的同一个字。

[0118] 89)根据85)所述的非临时计算机可读存储介质,其中,所述指令集使所述计算设备基于所述指示符矩阵经由连接器来向一个或更多个寄存器提供数据。

[0119] 90)根据89)所述的非临时计算机可读存储介质,其中,基于所述指示符矩阵来向所述一个或更多个寄存器提供数据包括:经由所述指示符矩阵来确定与所述数据请求中的任何其它地址不几乎相同的或者与所述数据请求中的一组其它地址几乎相同并且具有所述一组其它地址中的地址的最低索引的在所述数据请求中的一个或更多个地址,并且将与所确定的一个或更多个地址相关联的数据提供给所述一个或更多个寄存器。

[0120] 91)根据90)所述的非临时计算机可读存储介质,其中,所述确定所述一个或更多个地址由一个或更多个复用器执行。

[0121] 92)一种设备,包括:存储器;数据处理子系统,其包括至少一个数据处理单元以及基于参数的第一集合来在所述存储器中生成地址的第一集合的至少一个地址生成单元;控制子系统,其将参数的所述第一集合和参数的第二集合提供给所述数据处理子系统;以及队列,其存储参数的所述第一集合和参数的所述第二集合并且在所述数据处理单元使用所述地址的第一集合来处理数据时向所述地址生成单元提供参数的所述第二集合。

[0122] 93)根据92)所述的设备,其中:由所述数据处理单元处理的所述数据包括至少一个图像窗口,所述地址的第一集合中的至少一个对应于在所述存储器中的存储所述至少一个图像窗口的第一部分的位置,以及所述地址生成单元通过使用图像参数对所述至少一个图像窗口进行迭代来生成所述地址的第一集合,所述图像参数包括在所述存储器中的存储所述至少一个图像窗口的第二部分的地址。

[0123] 94)根据93)所述的设备,其中,所述数据处理子系统包括将所述图像参数提供给所述地址生成单元的标量处理器。

[0124] 95)根据92)所述的设备,其中,所述数据处理单元包括至少一个向量处理器,并且由所述数据处理单元处理的所述数据包括向量数据。

[0125] 96)根据93)所述的设备,其中,所述数据处理子系统以增量的方式从所述存储器中检索数据,并且所述数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据,所述偏移量指示所述未对齐的向量数据在所述存储器中关于所述增量是如何未对齐的。

[0126] 97)根据96)所述的设备,其中,所述地址生成单元计算由所述数据处理单元在所述串接中使用的所述偏移量。

[0127] 98)根据97)所述的设备,其中:所述地址生成单元通过对所述至少一个图像窗口的部分进行迭代来生成所述地址的第一集合,并且所述偏移量包括迭代不变参数。

[0128] 99)一种用于处理数据的方法,包括:经由控制子系统将参数的第一集合和参数的第二集合提供到数据处理子系统,所述数据处理子系统包括至少一个数据处理单元和至少一个地址生成单元;将参数的所述第一集合和参数的所述第二集合存储在队列中;基于参数的所述第一集合经由所述地址生成单元来生成在存储器中的地址的第一集合;当由所述

数据处理单元使用所述地址的第一集合来处理数据时，经由所述队列将参数的所述第二集合提供到所述地址生成单元。

[0129] 100)根据99)所述的方法，其中，由所述数据处理单元处理的所述数据包括至少一个图像窗口，其中所述地址的第一集合中的至少一个对应于在所述存储器中的存储所述至少一个图像窗口的第一部分的位置，以及所述方法包括通过使用图像参数对所述至少一个图像窗口进行迭代来经由所述地址生成单元生成所述地址的第一集合，所述图像参数包括在所述存储器中的存储所述至少一个图像窗口的第二部分的地址。

[0130] 101)根据100)所述的方法，包括通过所述数据处理子系统的标量处理器将所述图像参数提供给所述地址生成单元。

[0131] 102)根据99)所述的方法，其中，所述数据的处理包括由至少一个向量处理器处理向量数据。

[0132] 103)根据100)所述的方法，包括：经由所述数据处理子系统以增量的方式从所述存储器中检索数据，并且经由所述数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据，所述偏移量指示未对齐的向量数据在所述存储器中关于所述增量是如何未对齐的。

[0133] 104)根据103)所述的方法，包括经由所述地址生成单元来计算由所述数据处理单元在所述串接中使用的所述偏移量。

[0134] 105)根据103)所述的方法，包括经由所述地址生成单元通过对所述至少一个图像窗口的部分进行迭代来生成所述地址的第一集合，并且所述偏移量包括迭代不变参数。

[0135] 106)一种非临时计算机可读存储介质，其上存储有软件程序的指令集，所述指令集在由计算设备执行时使所述计算设备：经由控制子系统将参数的第一集合和参数的第二集合提供到数据处理子系统，所述数据处理子系统包括至少一个数据处理单元和至少一个地址生成单元；将参数的所述第一集合和参数的所述第二集合存储在队列中；基于参数的所述第一集合经由所述地址生成单元来生成在存储器中的地址的第一集合；以及当由所述数据处理单元使用所述地址的第一集合处理数据时，经由所述队列将参数的所述第二集合提供到所述地址生成单元。

[0136] 107)根据106)所述的非临时计算机可读存储介质，其中：由所述数据处理单元处理的数据包括至少一个图像窗口，所述地址的第一集合中的至少一个对应于在所述存储器中的存储所述至少一个图像窗口的第一部分的位置，以及所述指令集使所述计算设备经由所述地址生成单元通过使用图像参数对所述至少一个图像窗口进行迭代来生成所述地址的第一集合，所述图像参数包括在所述存储器中的存储所述至少一个图像窗口的第二部分的地址。

[0137] 108)根据107)所述的非临时计算机可读存储介质，其中，所述数据处理子系统包括将所述图像参数提供给所述地址生成单元的标量处理器。

[0138] 109)根据106)所述的非临时计算机可读存储介质，其中，所述数据处理单元包括至少一个向量处理器，并且由所述数据处理单元处理的所述数据包括向量数据。

[0139] 110)根据107)所述的非临时计算机可读存储介质，其中，所述指令集使所述计算设备：经由所述数据处理子系统以增量的方式从所述存储器中检索数据，并且经由所述数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据，所述偏移量指示未

对齐的向量数据在所述存储器中关于所述增量是如何未对齐的。

[0140] 111)根据110)所述的非临时计算机可读存储介质，其中，所述指令集使所述计算设备经由所述地址生成单元来计算由所述数据处理单元在所述串接中使用的所述偏移量。

[0141] 112)根据111)所述的非临时计算机可读存储介质，其中：所述指令集使所述计算设备经由所述地址生成单元通过对所述至少一个图像窗口的部分进行迭代来生成所述地址的第一集合，并且所述偏移量包括迭代不变参数。

[0142] 113)一种设备，包括：至少一个输入寄存器、至少一个可寻址寄存器以及至少一个处理单元，所述处理单元被配置为通过以下操作来生成直方图：经由所述输入寄存器接收数据向量，所述数据向量包括多个数据点；使所述多个数据点与所述直方图的多个直条匹配；确定在所匹配的多个数据点中是否存在竞争；如果在所匹配的多个数据点中存在竞争，则将所述数据向量的数据点的至少两个串行地分类到多个存储器位置中；以及如果在所匹配的多个数据点中不存在竞争，则将所述数据向量的数据点的至少两个并行地分类到多个存储器位置中。

[0143] 114)根据113)所述的设备，其中，如果在所匹配的多个数据点中存在竞争则将所述数据向量的数据点的两个或更多个串行地分类到所述多个存储器位置中包括：在第一时间段期间更新与所述直方图的至少一个直条相关联的直方图阵列；生成第一地址位置，所述第一地址位置对应于所述数据向量的第一数据点，所述第一地址位置基于在所述第一时间段期间更新的经更新的直方图阵列；基于所生成的第一地址位置将所述第一数据点写到所述可寻址寄存器；在第二时间段期间更新所述直方图阵列；生成第二地址位置，所述第二地址位置对应于所述数据向量的第二数据点，所述第二地址位置基于在所述第二时间段期间更新的经更新的直方图阵列；以及基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

[0144] 115)根据113)所述的设备，其中，如果在所匹配的多个数据点中不存在竞争则将所述数据向量的数据点的两个或更多个并行地分类包括：在第一时间段期间更新与所述直方图的第一直条相关联的第一直方图阵列；在所述第一时间段期间更新与所述直方图的第二直条相关联的第二直方图阵列；生成第一地址位置，所述第一地址位置对应于所述数据向量的第一数据点，所述第一地址位置基于经更新的第一直方图阵列；生成第二地址位置，所述第二地址位置对应于所述数据向量的第二数据点，所述第二地址位置基于经更新的第二直方图阵列；在第二时间段期间基于所生成的第一地址位置来将所述第一数据点写到所述可寻址寄存器；以及在所述第二时间段期间基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

[0145] 116)根据113)所述的设备，其中，确定在所匹配的多个数据点内是否存在地址竞争包括确定所述多个数据点中的两个或更多个是否与所述直方图的所述多个直条的公共直条匹配。

[0146] 117)根据114)所述的设备，其中，生成所述第一地址位置并生成所述第二地址位置包括将所述直方图阵列的值添加到与所述直方图的至少一个直条相关联的基地址。

[0147] 118)根据117)所述的设备，其中，生成所述第一地址位置包括将经更新的第一直方图阵列的值添加到与所述直方图的第一直条相关联的基地址，并且其中生成所述第二地址位置包括将经更新的第二直方图阵列的值添加到与所述直方图的第二直条相关联的基

地址。

[0148] 119)根据113)所述的设备,其中,使所述多个数据点与所述直方图的多个直条匹配包括将所述多个数据点中的每个数据点关联到所述直方图的所述多个直条中的直条。

[0149] 120)一种方法,包括:经由输入寄存器接收数据向量,所述数据向量包括多个数据点;使所述多个数据点与直方图的多个直条匹配;确定在所匹配的多个数据点中是否存在竞争;如果在所匹配的多个数据点中存在竞争,则将所述数据向量的数据点的至少两个串行地分类到多个存储器位置中;以及如果在所匹配的多个数据点中不存在竞争,则将所述数据向量的数据点的至少两个并行地分类到多个存储器位置中。

[0150] 121)根据120)所述的方法,其中,如果确定在所匹配的多个数据点中存在竞争则将所述数据向量的数据点的两个或更多个串行地分类到所述多个存储器位置中包括:在第一时间段期间更新与所述直方图的至少一个直条相关联的直方图阵列;生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于在所述第一时间段期间更新的经更新的直方图阵列;基于所生成的第一地址位置来将所述第一数据点写到可寻址寄存器;在第二时间段期间更新所述直方图阵列;生成第二地址位置,所述第二地址位置对应于所述数据向量的第二数据点,所述第二地址位置基于在所述第二时间段期间更新的经更新的直方图阵列;以及基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

[0151] 122)根据120)所述的方法,其中,如果在所匹配的多个数据点中不存在竞争则将所述数据向量的数据点的两个或更多个并行地分类包括:在第一时间段期间更新与所述直方图的第一直条相关联的第一直方图阵列;在所述第一时间段期间更新与所述直方图的第二直条相关联的第二直方图阵列;生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于经更新的第一直方图阵列;生成第二地址位置,所述第二地址位置对应于所述数据向量的第二数据点,所述第二地址位置基于经更新的第二直方图阵列;在第二时间段期间基于所生成的第一地址位置将所述第一数据点写到可寻址寄存器;以及在所述第二时间段期间基于所生成的第二地址位置将所述第二数据点写到所述可寻址寄存器。

[0152] 123)根据120)所述的方法,其中,确定在所匹配的多个数据点内是否存在地址竞争包括确定所述多个数据点中的两个或更多个是否与所述直方图的所述多个直条的公共直条匹配。

[0153] 124)根据121)所述的方法,其中,生成所述第一地址位置并生成所述第二地址位置包括将所述直方图阵列的值添加到与所述直方图的至少一个直条相关联的基地址。

[0154] 125)根据124)所述的方法,其中,生成所述第一地址位置包括将经更新的第一直方图阵列的值添加到与所述直方图的第一直条相关联的基地址,并且其中生成所述第二地址位置包括将经更新的第二直方图阵列的值添加到与所述直方图的第二直条相关联的基地址。

[0155] 126)根据120)所述的方法,其中,使所述多个数据点与所述直方图的多个直条匹配包括将所述多个数据点中的每个数据点关联到所述直方图的所述多个直条中的直条。

[0156] 127)一种非临时计算机可读存储介质,其上存储有用于处理软件程序的指令的指令集,所述指令集在由计算设备执行时使所述计算设备:经由输入寄存器接收数据向量,所

述数据向量包括多个数据点;使所述多个数据点与直方图的多个直条匹配;确定在所匹配的多个数据点中是否存在竞争;如果在所匹配的多个数据点中存在竞争,则将所述数据向量的数据点的至少两个串行地分类到多个存储器位置中;以及如果在所匹配的多个数据点中不存在竞争,则将所述数据向量的至少两个数据点并行地分类到多个存储器位置中。

[0157] 128)根据127)所述的非临时计算机可读存储介质,其中,如果确定在所匹配的多个数据点中存在竞争则将所述数据向量的数据点的两个或更多个串行地分类到所述多个存储器位置中包括:在第一时间段期间更新与所述直方图的至少一个直条相关联的直方图阵列;生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于在所述第一时间段期间更新的经更新的直方图阵列;基于所生成的第一地址位置来将所述第一数据点写到可寻址寄存器;在第二时间段期间更新所述直方图阵列;生成第二地址位置,所述第二地址位置对应于所述数据向量的第二数据点,所述第二地址位置基于在所述第二时间段期间更新的经更新的直方图阵列;以及基于所生成的第二地址位置来将所述第二数据点写到所述可寻址寄存器。

[0158] 129)根据127)所述的非临时计算机可读存储介质,其中,如果在所匹配的多个数据点中不存在竞争则将所述数据向量的数据点的两个或更多个并行地分类包括:在第一时间段期间更新与所述直方图的第一直条相关联的第一直方图阵列;在所述第一时间段期间更新与所述直方图的第二直条相关联的第二直方图阵列;生成第一地址位置,所述第一地址位置对应于所述数据向量的第一数据点,所述第一地址位置基于经更新的第一直方图阵列;生成第二地址位置,所述第二地址位置对应于所述数据向量的第二数据点,所述第二地址位置基于经更新的第二直方图阵列;在第二时间段期间基于所生成的第一地址位置将所述第一数据点写到可寻址寄存器;以及在所述第二时间段期间基于所生成的第二地址位置将所述第二数据点写到所述可寻址寄存器。

[0159] 130)根据127)所述的非临时计算机可读存储介质,其中,确定在所匹配的多个数据点内是否存在地址竞争包括确定所述多个数据点中的两个或更多个是否与所述直方图的所述多个直条的公共直条匹配。

[0160] 131)根据128)所述的非临时计算机可读存储介质,其中,生成所述第一地址位置并生成所述第二地址位置包括将所述直方图阵列的值添加到与所述直方图的至少一个直条相关联的基地址。

[0161] 132)根据131)所述的非临时计算机可读存储介质,其中,生成所述第一地址位置包括将经更新的第一直方图阵列的值添加到与所述直方图的第一直条相关联的基地址,并且其中生成所述第二地址位置包括将经更新的第二直方图阵列的值添加到与所述直方图的第二直条相关联的基地址。

[0162] 133)根据127)所述的非临时计算机可读存储介质,其中,使所述多个数据点与所述直方图的多个直条匹配包括将所述多个数据点中的每个数据点关联到所述直方图的所述多个直条中的直条。

[0163] 134)一种存储器设备,包括:存储器模块,其是易失性的并且被配置为在降低功率模式中操作并且退出所述降低功率模式并且在正常功率模式中操作;错误控制电路;以及存储器控制器,其被配置为:在所述存储器模块返回到所述正常功率模式之后将第一大小的数据单元写到所述存储器模块,其中所述数据单元位于第二大小的存储器分段内;接收

读命令以从所述存储器模块读取所述数据单元，读取存储在所述存储器分段中的存储器分段内容，将所述存储器分段内容发送给所述错误控制电路；以及将读掩码传输到所述错误控制电路，其中所述读掩码指示与所述数据单元有关的一个或更多个冗余位；其中所述错误控制电路被配置为：接收所述存储器分段内容和所述读掩码；以及基于与所述数据单元有关的所述一个或更多个冗余位来检查所述数据单元中的一个或更多个错误，同时忽略与所述存储器分段内容的、与所述数据单元不同的一个或更多个部分有关的冗余位；其中从所述降低功率模式退出并不跟随有初始化整个存储器模块。

[0164] 135)根据134)所述的存储器设备，其中，所述数据单元具有一个字节的宽度，以及其中所述存储器分段内容具有一个字的宽度。

[0165] 136)一种用于错误校正的方法，所述方法包括：在降低功率模式中操作存储器模块，在从在所述降低功率模式中操作所述存储器模块退出之后，在正常功率模式中操作所述存储器模块；将第一大小的数据单元写到所述存储器模块，其中所述数据单元位于第二大小的存储器分段内；接收读命令以从所述存储器模块读取所述数据单元；读取存储在所述存储器分段中的存储器分段内容，将所述存储器分段内容发送给错误控制电路；将读掩码传输到所述错误控制电路，其中所述读掩码指示与所述数据单元有关的一个或更多个冗余位；由错误控制模块接收所述存储器分段内容和所述读掩码；以及基于与所述数据单元有关的所述一个或更多个冗余位来检查所述数据单元中的一个或更多个错误，同时忽略与所述存储器分段内容的、与所述数据单元不同的一个或更多个部分有关的冗余位；其中从在所述降低功率模式中操作所述存储器模块退出并不跟随有初始化整个存储器模块。

[0166] 137)根据136)所述的方法，其中，所述数据单元具有一个字节的宽度，以及其中所述存储器分段内容具有一个字的宽度。

[0167] 138)一种非临时计算机可读存储介质，其上存储有用于处理软件程序的多个线程的指令集，所述指令集在由计算设备执行时使所述计算设备：在降低功率模式中操作存储器模块，在从在所述降低功率模式中操作所述存储器模块退出之后，在正常功率模式中操作所述存储器模块；将第一大小的数据单元写到所述存储器模块，其中所述数据单元位于第二大小的存储器分段内；接收读命令以从所述存储器模块读取所述数据单元，读取存储在所述存储器分段中的存储器分段内容，将所述存储器分段内容发送给错误控制电路；将读掩码传输到所述错误控制电路，其中所述读掩码指示与所述数据单元有关的一个或更多个冗余位；由错误控制模块接收所述存储器分段内容和所述读掩码；以及基于与所述数据单元有关的所述一个或更多个冗余位来检查所述数据单元中的一个或更多个错误，同时忽略与所述存储器分段内容的、与所述数据单元不同的一个或更多个部分有关的冗余位；其中从在所述降低功率模式中操作所述存储器模块退出并不跟随有初始化整个存储器模块。

[0168] 139)根据138)所述的非临时计算机可读存储介质，其中，所述数据单元具有一个字节的宽度，以及其中所述存储器分段内容具有一个字的宽度。

[0169] 140)一种多核处理器设备，包括：被配置为使用桶式线程处理来执行软件程序的多个线程的多个处理核心；一个或更多个数据高速缓存；一个或更多个指令存储器；以及线程间通信模块，其包括原子计数器部分和用于在所述多个线程之间同步的数据锁部分。

[0170] 141)根据140)所述的多核处理器设备，其中，所述线程间通信模块不属于所述多个处理核心、所述一个或更多个数据高速缓存和所述一个或更多个指令存储器。

[0171] 142)根据140)所述的多核处理器设备,其中,所述数据锁部分包括多个数据锁,其中每个数据锁包括用于掩蔽对锁定所述数据锁的请求的掩码、用于随着时间的过去而改变所述掩码的掩码编码器、用于选择选定的掩码请求以锁定所述数据锁的第一优先级编码器以及用于选择锁定所述数据锁而不考虑所述掩码的请求的第二优先级编码器。

[0172] 143)根据142)所述的多核处理器设备,其中,所述第一优先级编码器和第二优先级编码器应用相同的仲裁方案。

[0173] 144)根据142)所述的多核处理器设备,其中,所述第一优先级编码器和第二优先级编码器应用不同的仲裁方案。

[0174] 145)根据140)所述的多核处理器设备,其中,所述多个处理核心包括数据锁请求器;其中每个数据锁请求器被配置为执行状态机,所述状态机用于控制锁定数据锁的请求的发送,解锁所述数据锁的请求的发送以及选择性地阻止线程的执行。

[0175] 146)根据140)所述的多核处理器设备,其中,不同的数据锁指向不同的线程的起始。

[0176] 147)根据140)所述的多核处理器设备,其中,所述原子计数器部分包括存储器模块、判别器和处理器,所述存储器模块包括用于存储多个原子计数器值的多个条目,所述判别器用于按周期选择用于更新给定原子计数器的多达单个请求,以及所述处理器用于更新所述给定原子计数器。

[0177] 148)根据147)所述的多核处理器设备,其中,所述多个条目的数量小于所述数据锁部分的数据锁的数量。

[0178] 149)一种方法,包括由多个处理核心使用桶式线程处理来执行作为指令线程的多个线程,其中所述多个线程的执行包括通过利用线程间通信模块的数据锁和原子计数器来在线程的执行之间同步。

[0179] 150)根据149)所述的方法,其中,所述线程间通信模块不属于所述多个处理核心、一个或更多个数据高速缓存以及一个或更多个指令存储器。

[0180] 151)根据149)所述的方法,其中,在所述线程的执行之间的同步包括管理多个数据锁,其中数据锁的管理包括通过掩码对用于锁定所述数据锁的请求进行掩蔽,随时间的过去由掩码编码器改变所述掩码,由第一优先级编码器选择用于锁定所述数据锁的选定的掩码请求,并且由第二优先级编码器选择锁定所述数据锁而不考虑所述掩码的请求。

[0181] 152)根据151)所述的方法,包括由所述第一优先级编码器和第二优先级编码器应用相同的仲裁方案。

[0182] 153)根据151)所述的方法,包括由所述第一优先级编码器和第二优先级编码器应用不同的仲裁方案。

[0183] 154)根据149)所述的方法,其中,所述多个处理核心包括数据锁请求器;其中所述方法包括由每个数据锁请求器执行状态机,所述状态机用于控制锁定数据锁的请求的发送、解锁所述数据锁的请求的发送并用于选择性地阻止线程的执行。

[0184] 155)根据149)所述的方法,其中,不同的数据锁指向不同线程的起始。

[0185] 156)根据149)所述的方法,其中,所述线程间通信模块的原子计数器部分包括存储器模块、判别器和处理器,所述存储器模块包括用于存储多个原子计数器值的多个条目,所述判别器用于按周期选择用于更新给定原子计数器的多达单个请求,所述处理器用于更

新所述给定原子计数器。

[0186] 157)根据156)所述的方法,其中,所述多个条目的数量小于所述线程间通信模块的数据锁部分的数据锁的数量。

[0187] 158)一种非临时计算机可读存储介质,其上存储有指令集,所述指令集用于由多个处理核心使用桶式线程处理来执行作为指令线程的多个线程,其中所述多个线程的执行包括通过利用线程间通信模块的数据锁和原子计数器在线程的执行之间同步。

[0188] 159)根据158)所述的非临时计算机可读存储介质,其中,所述线程间通信模块不属于所述多个处理核心、一个或更多个数据高速缓存和一个或更多个指令存储器。

[0189] 160)根据158)所述的非临时计算机可读存储介质,其中,在所述线程的执行之间的同步包括管理多个数据锁,其中数据锁的管理包括通过掩码对用于锁定所述数据锁的请求进行掩蔽,随时间的过去由掩码编码器改变所述掩码,由第一优先级编码器选择用于锁定所述数据锁的选定的掩码请求,并且由第二优先级编码器选择锁定所述数据锁而不考虑所述掩码的请求。

[0190] 161)根据160)所述的非临时计算机可读存储介质,其具有存储在其上的用于由所述第一优先级编码器和第二优先级编码器应用相同的仲裁方案的指令集。

[0191] 162)根据160)所述的非临时计算机可读存储介质,其具有存储在其上的用于由所述第一优先级编码器和第二优先级编码器应用不同的仲裁方案的指令集。

[0192] 163)根据158)所述的非临时计算机可读存储介质,其中,所述多个处理核心包括数据锁请求器;其中所述非临时计算机可读存储介质具有存储在其上的用于由每个数据锁请求器执行状态机的指令集,该状态机用于控制锁定数据锁的请求的发送、解锁所述数据锁的请求的发送并用于选择性地阻止线程的执行。

[0193] 164)根据158)所述的非临时计算机可读存储介质,其中,不同的数据锁指向不同线程的起始。

[0194] 165)根据158)所述的非临时计算机可读存储介质,其中,原子计数器部分包括存储器模块、判别器以及处理器,所述存储器模块包括用于存储多个原子计数器值的多个条目,所述判别器用于按周期选择用于更新给定原子计数器的多达单个请求,所述处理器用于更新所述给定原子计数器。

[0195] 166)根据165)所述的非临时计算机可读存储介质,其中,所述多个条目的数量小于所述线程间通信模块的数据锁部分的数据锁的数量。

[0196] 167)一种浮点处理器,包括加法和减法单元以及格式转换器,其中,所述格式转换器被配置为使用指数偏差来将整数转换成浮点数,并且其中所述加法和减法单元包括被配置为当接收到作为非规格化数的输入操作数时生成无效异常的异常块。

[0197] 168)根据167)所述的浮点处理器,其中,所述异常块被配置为将超过大小阈值的输入变量视为无效数字。

[0198] 169)根据167)所述的浮点处理器,其被配置为通过应用加法或减法运算来计算结果;当所述结果是非规格化数时生成异常;并输出调到零位的数字和异常标志。

[0199] 170)根据167)所述的浮点处理器,还包括:指数比较器,其被配置为比较两个输入操作数的指数;初始尾数计算器,其用于在基于所述两个输入操作数的指数在所述两个输入操作数的尾数之间对齐之后加上或减去所述两个输入操作数的所述尾数以提供初始尾

数结果;尾数格式化单元,其用于在所述初始尾数结果中找到前导设置位,并且用于当所述前导设置位不是所述初始尾数结果的最高有效位时移位所述初始尾数结果,使得所述前导设置位是结果的尾数的最高有效位。

[0200] 171)根据170)所述的浮点处理器,还包括:指数选择器,其用于在两个输入操作数的指数之间进行选择以提供所选择的指数;指数修改器,其用于修改所述选择的指数以便补偿所述初始尾数结果的移位以提供结果指数;以及用于确定结果符号的符号选择器。

[0201] 172)根据167)所述的浮点处理器,其中,所述格式转换器被配置为当浮点数是非规格化数时将该浮点数转换为整数而不生成异常。

[0202] 173)一种用于由浮点处理器执行浮点运算的方法,其中,所述方法包括:由加法和减法单元加上或减去浮点格式的两个输入操作数;其中加上或减去包括:当所述两个输入操作数中的至少一个是非规格化数时,由所述加法和减法单元的异常块生成无效数字异常;其中,无效数字异常的生成包括输出默认值并设置异常标志;以及使用指数偏差来将整数转换为浮点数。

[0203] 174)根据173)所述的方法,包括由所述异常块将超过大小阈值的输入变量视为无效数字。

[0204] 175)根据173)所述的方法,包括通过应用加法或减法运算来计算结果;当所述结果是非规格化数时生成异常并输出调到零位的数字和异常标志。

[0205] 176)根据173)所述的方法,其中,所述加上或减去包括:由指数比较器比较两个输入操作数的指数;在基于所述两个输入操作数的指数在所述两个输入操作数的尾数之间对齐之后由初始尾数计算器加上或减去所述两个输入操作数的尾数以提供初始尾数结果;通过以下操作由尾数格式化单元进行对所述初始尾数结果的尾数格式化:在所述初始尾数结果中找到前导设置位,当所述前导设置位不是所述初始尾数结果的最高有效位时移位所述初始尾数结果,使得所述前导设置位是输出尾数的最高有效位。

[0206] 177)根据176)所述的方法,包括由指数选择器在两个输入操作数的指数之间进行选择以提供所选择的指数;由指数修改器修改所述选择的指数以便补偿所述初始尾数结果的移位以提供输出指数;以及由符号选择器确定输出信号的符号。

[0207] 178)根据173)所述的方法,包括:当浮点数是非规格化数时,由格式转换器将该浮点数转换为整数而不生成异常。

[0208] 179)一种非临时计算机可读存储介质,其上存储有指令集,所述指令集用于由浮点处理器执行浮点运算,所述浮点运算包括:由加法和减法单元加上或减去浮点格式的两个输入操作数;其中所述加上或减去包括:当所述两个输入操作数中的至少一个是非规格化数时,生成无效数字异常;其中,所述无效数字异常的生成包括输出默认值并设置异常标志;以及通过格式转换器使用指数偏差来将整数转换为浮点数。

[0209] 180)根据179)所述的非临时计算机可读存储介质,具有存储在其上的用于由所述加法和减法单元的异常块将超过大小阈值的输入变量视为无效数字的指令集。

[0210] 181)根据179)所述的非临时计算机可读存储介质,具有存储在其上的用于进行以下操作的指令集:通过应用加法或减法运算来计算结果、当所述结果是非规格化数时生成异常并输出调到零位的数字和异常标志。

[0211] 182)根据179)所述的非临时计算机可读存储介质,其中,所述加上或减去包括:由

指数比较器比较两个输入操作数的指数;由初始尾数计算器在基于所述两个输入操作数的指数在所述两个输入操作数的尾数之间对齐之后加上或减去所述两个输入操作数的尾数以提供初始尾数结果;通过以下操作由尾数格式化单元进行对所述初始尾数结果的尾数格式化:在所述初始尾数结果中找到前导设置位,当所述前导设置位不是所述初始尾数结果的最高有效位时移位所述初始尾数结果,使得所述前导设置位是输出尾数的最高有效位。

[0212] 183)根据182)所述的非临时计算机可读存储介质,具有存储在其上的用于进行以下操作的指令集:由指数选择器在所述两个输入操作数的指数之间进行选择以提供所选择的指数、由指数修改器修改所述选择的指数以便补偿所述初始尾数结果的移位以提供输出指数以及由符号选择器确定输出信号的符号。

[0213] 184)根据179)所述的非临时计算机可读存储介质,具有存储在其上的用于进行以下操作的指令集:当浮点数是非规格化数时由所述格式转换器将该浮点数转换为整数而不生成异常。

[0214] 185)一种设备,包括:至少一个输入寄存器、至少一个可寻址寄存器以及至少一个处理单元,所述处理单元被配置为通过以下操作来生成直方图:经由所述输入寄存器接收数据向量,所述数据向量包括多个数据点;使所述多个数据点与所述直方图的多个直条匹配;搜索当所述多个数据点包括属于同一直条的数据点时发生的竞争;当检测到所述竞争时,则串行地计算用于存储属于同一直条的所述数据点的、给定存储器条目的地址并在所述给定存储器条目处存储属于同一直条的数据点;以及当没有检测到所述竞争时,则并行地计算用于存储所述多个数据点的某些存储器条目的地址,并在所述某些存储器条目处存储所述多个数据点。

[0215] 186)一种方法,包括:通过以下操作来生成直方图:经由输入寄存器接收数据向量,所述数据向量包括多个数据点;使所述多个数据点与所述直方图的多个直条匹配;搜索当所述多个数据点包括属于同一直条的数据点时发生的竞争;当检测到所述竞争时,则串行地计算用于存储属于同一直条的所述数据点的给定存储器条目的地址并在所述给定存储器条目处存储属于同一直条的数据点;以及当没有检测到所述竞争时,则并行地计算用于存储所述多个数据点的某些存储器条目的地址,并在所述某些存储器条目处存储所述多个数据点。

[0216] 187)一种非临时计算机可读存储介质,其上存储有用于处理软件程序的指令的指令集,所述指令集在由计算设备执行时使所述计算设备通过以下操作生成直方图:经由输入寄存器接收数据向量,所述数据向量包括多个数据点;使所述多个数据点与所述直方图的多个直条匹配;搜索当所述多个数据点包括属于同一直条的数据点时发生的竞争;当检测到所述竞争时,则串行地计算用于存储属于同一直条的数据点的给定存储器条目的地址并在所述给定存储器条目处存储属于同一直条的数据点;以及当没有检测到所述竞争时,则并行地计算用于存储所述多个数据点的某些存储器条目的地址,并在所述某些存储器条目处存储所述多个数据点。

[0217] 附图简述

[0218] 视为本发明的主题在说明书的结束部分中被特别指出并被清楚地要求保护。然而,关于操作方法和组织的本发明以及其目的、特征和优点在与附图一起阅读时通过参考以下详细描述可得到最好的理解,其中:

[0219] 图1示出根据本公开的例子的示例性片上系统(SoC)；

[0220] 图2示出根据本公开的例子的示例性多核处理器；

[0221] 图3A示出根据本公开的例子的被配置为执行桶式线程处理的示例性多核处理器；

[0222] 图3B示出根据本公开的例子的被配置为执行桶式线程处理的示例性多核处理器；

[0223] 图4示出根据本公开的例子的用于在桶式线程化处理器中的指令线程的执行的示例性定时图；

[0224] 图5A示出根据本公开的例子的被配置为执行桶式线程处理并利用共享数据高速缓存的示例性多核处理器；

[0225] 图6A示出根据本公开的例子的具有串行计算机处理单元的示例性桶式线程化多核处理器；

[0226] 图6B示出根据本公开的例子的具有串行计算机处理单元的示例性桶式线程化多核处理器；

[0227] 图6C示出根据本公开的例子的具有串行计算机处理单元的示例性桶式线程化多核处理器；

[0228] 图6D示出窗口；

[0229] 图6E示出根据本公开的例子的具有串行计算机处理单元的示例性桶式线程化多核处理器；

[0230] 图6F示出根据本公开的例子的多线程通信；

[0231] 图7示出根据本公开的例子的示例性双发射(dual issue)处理器；

[0232] 图8示出根据本公开的例子的配置成接受链式指令的示例性双发射核心处理器；

[0233] 图9示出根据本公开的例子的配置用于桶式线程处理和接受链式指令的示例性双发射核心；

[0234] 图10A示出根据本公开的例子的在第一时钟周期期间的示例性链式指令处理流程；

[0235] 图10B示出根据例子的在第二时钟周期期间的示例性链式指令处理流程。

[0236] 图10C示出根据本公开的例子的配置用于桶式线程处理和接受链式指令的示例性双发射核心；

[0237] 图11示出根据本公开的例子的具有与外部存储器通信的本地高速缓存的示例性多处理器系统；

[0238] 图12示出根据本公开的例子的示例性高速缓存配置或结构；

[0239] 图13A示出根据本公开的例子的在多个处理器当中的真实数据共享；

[0240] 图13B示出在多个处理器当中的伪共享。

[0241] 图14示出根据本公开的例子的解决伪共享的硬件；

[0242] 图15示出根据本公开的例子的“脏位(dirty bits)”的示例性使用；

[0243] 图16A示出根据本公开的例子的具有受限数据共享的多个过程实现；

[0244] 图16B是对于图16A所示的过程的定时示意图；

[0245] 图16C示出根据本公开的例子的数据锁；

[0246] 图16D示出根据本公开的例子的状态机；

[0247] 图16E示出根据本发明的实施例的方法；

[0248] 图17示出根据本公开的例子的示例性向量处理器；

[0249] 图18示出根据本公开的例子的示例性存储器组和互连体系结构；

[0250] 图19A示出根据本公开的例子的示例性体系结构；

[0251] 图19B是根据本公开的例子的对于利用图4的体系结构的流程图；

[0252] 图20示出根据本公开的例子的对于一系列地址之一的指示符阵列的一部分；

[0253] 图21A示出根据本公开的例子的对于一系列几乎相同的地址之一的指示符阵列的一部分；

[0254] 图21B示出根据本公开的例子的对于一系列几乎相同的地址之一的指示符阵列的一部分；

[0255] 图21C示出根据本公开的例子的对于一系列几乎相同的地址之一的指示符阵列的一部分；

[0256] 图22示出根据本公开的例子的表示八个唯一地址的指示符阵列；

[0257] 图23示出根据本公开的例子的表示八个几乎相同的地址的指示符阵列；

[0258] 图24示出根据本公开的例子的表示四对几乎相同的地址的指示符阵列；

[0259] 图25A是根据本公开的例子的用于将值分配给指示符阵列的元素的示例性编码方案；

[0260] 图25B示出根据本发明的实施例的方法；

[0261] 图26示出根据本公开的例子的可以是图17中所示的向量处理器的逻辑和/或处理电路的一部分的示例性控制和处理子系统；

[0262] 图27是示出根据本公开的例子的对地址生成单元的示例性输入和来自地址生成单元的示例性输出；

[0263] 图28示出根据本公开的例子的具有参数队列的示例性控制和处理子系统；

[0264] 图29示出根据本公开的例子的示例性图像；

[0265] 图30示出根据本公开的例子的对于数据处理子系统的执行时间线；

[0266] 图31示出根据本公开的例子的对于图28中的数据处理子系统的执行时间线；

[0267] 图32示出根据本公开的例子的对于图28中的数据处理子系统的执行时间线；

[0268] 图33A示出根据本公开的例子的对于对齐数据的示例性8字节访问；

[0269] 图33B示出根据本公开的例子的对于未对齐数据的示例性8字节访问；

[0270] 图34A示出根据本公开的例子的示例性组合特征；

[0271] 图34B示出根据本公开的例子的用于处理数据的示例性方法；

[0272] 图35示出根据本公开的例子的用于串行化桶式分类的示例性方法；

[0273] 图36示出根据本公开的例子的用于向量化桶式分类的示例性方法；

[0274] 图37示出根据本公开的例子的另一示例性向量化直方图直条分类算法；

[0275] 图38示出根据本公开的例子的能够处理竞争的示例性向量化桶式分类算法；

[0276] 图39示出根据本公开的例子的示例性存储器设备；

[0277] 图40示出根据本公开的例子的具有存储器初始化的示例性存储器设备功率门控方法；

[0278] 图41示出根据本公开的例子的没有存储器初始化的示例性存储器设备功率门控方法；

[0279] 图42示出根据本公开的例子的具有读掩码功能的示例性存储器设备；

[0280] 图43示出根据本公开的例子的没有存储器初始化和具有读掩码功能的示例性存储器设备功率门控方法；

[0281] 图44示出根据本发明的实施例的方法；

[0282] 图45示出根据本发明的实施例的存储器模块、存储器控制器和各种数据结构；

[0283] 图46示出根据本公开的例子的示例性片上系统(SoC)；

[0284] 图47示出根据本发明的实施例的包括加法和减法单元以及格式转换器的浮点处理器；

[0285] 图48示出根据本发明的实施例的方法；以及

[0286] 图49示出根据本发明的实施例的方法。

[0287] 详细描述

[0288] 在例子的以下描述中，参考形成其一部分的附图，并且其中作为例证示出可被实施的特定例子。将理解，其他的例子可以被使用，并且结构改变可以被作出而不偏离所公开的例子的范围。此外，虽然提供了本公开可以被实施的示例背景，但是它们并不意欲将本公开的范围限制到那些背景。

[0289] 在说明书中对方法的任何提及应当加以必要的变动而应用于能够执行该方法的存储器设备，并且应当加以必要的变动而应用于非临时计算机可读介质，该非临时计算机可读介质存储一旦由计算机执行就导致方法的执行的指令。

[0290] 在说明书中对存储器设备的任何提及应当加以必要的变动而应用于可以由存储器设备执行的方法，并且应当加以必要的变动而应用于存储可以由存储器设备执行的指令的非临时计算机可读介质。

[0291] 在说明书中对非临时计算机可读介质的任何提及应当加以必要的变动而应用于能够执行存储在非临时计算机可读介质中的指令的存储器设备，并且应当加以必要的变动而应用于可以由计算机执行的方法，该计算机读取存储在非临时计算机可读介质中的指令。

[0292] 一些计算机处理器可以利用多个核心来使处理器在任何给定时间处理多个指令线程，从而提高处理器可以执行算法或程序的速度。然而，在某些计算情境中，多核处理器所实现的性能增益可能降低。因此，本公开涉及多核处理器，其可以利用桶式线程处理连同在处理器的多个核心之间共享的数据高速缓存，以便改善对于其中算法或程序需要具有独立迭代的多个软件循环的处理的计算环境的多核处理器。

[0293] 图1示出根据本公开的例子的示例性片上系统(SoC)。可以在许多计算情境中利用图1中所示的系统。作为非限制性例子，图1中所示的SoC 102可在计算机视觉系统中被利用，其中外围相机或多个相机捕获图像，图像然后被处理以展现嵌在每个捕获的图像内的特征。参考图1，外围设备106可以表示可以例如捕获某种形式的数据的在系统外部的一个或更多个设备。在计算机视觉情境中，这种外围设备106可以是捕获周围环境的图像的相机。由外围设备106捕获的数据可以经由外围设备控制器108输入到SoC 102内。一个或更多个外围设备控制器108可以从外围设备106接收数据并将该数据存储在SoC 102内部的存储器中。另外，外围设备控制器108可以将输出信号发送到外围设备106。这样的输出信号可以包括使外围设备106执行某些功能的命令，或者被提供给外围设备106的数据。

[0294] 外围设备控制器108可以经由片上互连110连接到SoC 102的其他部件。片上互连可以起到在必要时用于连接SoC 102的部件的开关的作用。作为例子，在外围设备控制器108与计算机处理单元104之间的连接可通过片上互连110来促进。类似地，在加速器114和116与CPU 104之间的连接也可以通过片上互连104来促进。

[0295] CPU 104可以充当协调在SoC 102内的所有各种部件的活动的中央处理器。CPU 104可以确定哪些部件将被分派执行各种程序任务的任务，或者可以处理规定SoC 102的哪个部件将处理各种任务的指令。另外，CPU 104也可以自己执行各种任务。作为例子，在计算机视觉情境中，CPU 104可以独自执行各种图像处理算法，而不必依靠其他部件来执行它们。

[0296] 然而，例如，如果并且当这样的任务更适合在加速器而不是CPU上执行时，CPU 104还可以依靠加速器112和114来执行处理任务。加速器112和114可以是专用计算机硬件，其可以比如果功能由在通用计算机处理器上运行的软件执行更快地执行功能。例如，可以定制加速器112和114以处理可能以另外方式在通用CPU上缓慢操作的计算密集型算法。加速器112和114可以利用各种方法来加速这种处理，例如利用并发性(同时执行多个任务的能力)来加速处理时间。

[0297] SoC 102也可以经由外部存储器控制器116来连接到外部存储器118。外部存储器118可以存储在SoC 102上执行功能和指令所必需的数据。外部存储器可以是固态存储器，但照惯例是可以相对快速地从中读取和写入的随机存取存储器(RAM)。外部存储器118可以被SoC 102中的其他部件(例如CPU 104、外围设备控制器108以及加速器112和114)潜在地访问。

[0298] 像在图1中以112和114所示的加速器一样的加速器可以被定制为执行计算密集型的和/或需要可能在通用CPU上不可用的专用处理的任务。在图2中呈现以根据本公开的例子的多核处理器的形式的硬件加速器的例子。多核处理器是具有彼此并行地读取和执行包含在软件程序内的指令的多个独立处理单元(被称为“核心”)的设备。通过并行地操作，核心可以同时执行算法所需的独立任务，从而减少了在给定数据集上实现算法所必需的时间量。

[0299] 在图2所示的例子中，加速器可以包括四个核心204a-d。每个核心204a-d可以连接到私有指令存储器202a-d。每个指令存储器202a-d被认为是对它相应的核心204a-d是“私有的”，因为它可以专用于该相应的核心。在其他例子中，每个核心204a-d可以共享公共指令存储器(见图3B的公共指令存储器6002)。采用私有或共享指令存储器的决定可以取决于例如存储器需要被核心访问的速度。私有指令存储器可以被实现为私有指令高速缓存(即，快速存储器的小区域)以进一步加速存储器可以被访问的速度。

[0300] 每个指令存储器202a-d可以对应于核心204a-d中的一个。例如，且如图2所示，指令存储器202a耦合到核心204a，指令存储器202b耦合到核心204b，指令存储器202c耦合到核心204c，等等。每个指令存储器202a-d可以存储将要被它相应的核心204a-d执行的指令。当由它相应的核心访问时，指令存储器可以提供将由相应的核心执行的、在指令的序列中的下一个指令。

[0301] 每个核心204a-d也可以耦合到数据高速缓存206a-d。类似于指令存储器202a-d，数据高速缓存206a-206d可以对应于单独的核心204a-204d，从而使数据高速缓存“私有”

(即,每个核具有其自己的对应的数据高速缓存)。数据高速缓存可以包含将要由核心操纵的数据,且还可以存储由每个单独核心处理的结果。

[0302] 为了解释的目的,聚焦于单个核心,核心204a可以从指令存储器202a接收指令。一旦来自指令存储器202a的指令被核心204a加载,核心就可以访问数据高速缓存206a,以检索将由指令操纵的数据。一旦指令已经被执行,指令的结果就可以被写到数据高速缓存206a。然后该过程能够以由核心204a从指令存储器202a加载的下一个指令来重复。在一些例子中,数据高速缓存206a和指令存储器202a可以在加载和存储指令期间交换信息,其中加载指令可以将数据从存储器复制到寄存器,并且存储指令可以将数据从寄存器传输到存储器。

[0303] 上面关于核心204a描述的过程也可以同时在其他三个核心204b-d中运行。由于每个核心都同时运行它自己的指令集,因此可以加快程序被执行的速度,这与其中单个处理核心正在操作的CPU相反。

[0304] 指令的序列可以被称为“程序”,而一个线程可以被定义为可以由调度器独立管理的程序指令的序列。加速器可以依据由加速器处理的程序的特性来定制。作为例子,采用从视觉处理算法中的下面的示例程序伪代码,例子1:

```
for ( obj in array ) {
    if ( obj too small ) {
        if ( obj nearly square ) {
        .....
                                }
[0305]
                            }
    else if ( obj not red enough ) {
                        .....
                                }
                            }
```

[0306] 上面的编程例子示出包括在for循环内的条件句的程序代码。在该例子中,条件句if/else语句(obj too small)嵌在第一个for循环(“objects in array”)中。附加条件句if/else语句(“obj nearly square”)进一步嵌在第一个条件句if/else语句内。上面的编程例子在视觉处理情境中是普遍的,其中分析具有多个像素的图像以例如确定是否存在某些特征或特性。在自主式汽车的领域中,这种视觉处理可能帮助区分在道路上的线路或在汽车附近的其他物体。在诸如视觉处理的计算环境中,其中程序代码将包含程序循环内的条件语句,这样的代码可以在许多并行步骤中被执行的事实可用于定制硬件加速器。

[0307] 图3A示出根据本公开的例子的被配置为执行桶式线程处理的示例性多核处理器。

在图3A中提供的例子中，核心304a-d被配置成使用桶式线程处理来执行程序的线程。可以通过检查在单个核心内的指令执行的时序来说明桶式线程处理的概念。

[0308] 图4示出根据本公开的例子的用于在桶式线程化处理器中的指令线程的执行的示例性时序图。图4中所示的时序图示出在图3A中所示的多核处理器的单个核心内的线程的执行的时序，该核心被配置为多路传输的八个线程(线程1-8)。多核处理器核心可以被配置为多路传输或多或少的指令线程，并且八个线程应该被理解为仅仅用作例子。

[0309] 该图包括10个时间段T1-T10。每个时间段代表处理器核心的单个时钟周期。在时间T1处，第一线程(线程1)的第一指令(可选地被称为命令“cmd”)可以由核心执行。在T2时间处，第二线程(线程2)的第一指令可以由核心执行。在时间T3处，第三线程的第一指令可以由核心执行。在时间T4-T8处，可以执行线程4-8的第一指令。在时间T9处，每个线程都使它们的第一指令被执行。处理器的核心然后可以返回到第一线程并且执行第一线程的第二指令。在时间T10处，过程的核心可以返回到第二线程并执行第二线程的第二指令。这个过程可以重复，直到每一个线程的所有指令都被完成为止。

[0310] 如图4所示，桶式线程处理可以通过消除浪费的时钟周期来提高处理效率。并行地处理多个命令可以减少时延对处理速度的影响。例如，图4中的指令中的任一个(“cmd”)可能导致在处理中的暂停或延迟(“时延”)，其例如由高速缓存未命中、分支、浮点运算和除法运算产生。桶式线程处理并不消除暂停或延迟，但它大大增加了暂停或延迟不使核心保持空闲的可能性。代替是空闲的，核心仅保持空闲一个时钟周期，且然后在随后的周期处将在延迟期间处理来自下一线程的命令，如图4所示。单线程化处理器(即，单个线程的指令被顺序处理的处理器)相反在暂停或延迟的持续时间期间将保持空闲。这可能导致许多浪费的时钟周期(即，其中没有指令的执行发生的时钟周期)。在浪费的时钟周期期间，处理器在它能够继续前进到序列中的下一指令之前必须停止执行程序线程以等待单个指令被完成。这可能导致在对于特定线程的处理时间中的延迟以及在程序或算法的整体执行中的延迟。

[0311] 即使在相比于存在的线程时延持续更多周期的情况下，桶式线程化处理器也可以帮助掩蔽时延的影响。在一个例子中，如果高速缓存未命中导致100个周期的时延，但是在给定的核心中仅有八个硬件线程，则时延可能仅使100/8个周期被错过，而不是在非桶式线程化过程中将错过的100个周期。

[0312] 更具体地，当第一线程正在等待或是空闲时，桶式线程处理可以通过切换到另一线程来减轻浪费的时钟周期。使用图4的例子，在线程1正在运行指令时，只要线程2的指令并不依赖于线程1的结果，则线程2可以执行指令。

[0313] 但是，任何后续指令都必须等待，直到在在它之前的指令已经被执行为止。作为例子，在时间T9处，如果在T1处执行的指令还没有完成执行，则在T9处的指令不能被执行，它必须等待，直到在T1处执行的指令被执行为止。因此，在T9期间，处理器什么也不能做，且然后在T10处将继续前进以执行第二线程的第二指令。

[0314] 在一些例子中，处理器可以用待执行的下一指令填充任何停滞的周期而不是在T9处什么也不做。回来参考上面的例子，在T9处不是什么都不做，处理器可以执行第一线程的第二指令。以这种方式，可以将停滞的线程对吞吐量的影响减到最小。这种处理方法可以被称为同时多线程处理(SMT)。

[0315] 在SMT中，可以以任何给定的时间间隔执行来自多于一个线程的指令。因此，如果

特定的线程以特定的时间间隔被停滞，并不是什么也不做，而是SMT可以允许在停滞的周期期间执行来自另一个线程的指令。

[0316] 以这种方式，如果线程的特定指令花费少于八个周期来执行，那么到核心处理器返回到线程时，指令将被执行，且线程可以继续前进以执行下一指令。在一个例子中，如果高速缓存未命中花费80个周期来被解决，则线程将失去10回合(turn)。这是因为桶式线程化核心每8个周期将返回到该线程。相反，在串行线程化处理器中，核心在能够执行下一指令之前必须等待80个周期并保持空闲。在桶式线程处理的情形中，处理器不是空闲的。桶式线程化系统将在高速缓存未命中被解决时执行其他线程的其他指令。

[0317] 桶式线程处理在有许多活动线程的计算环境中是有利的。返回到图4的例子，在某些情况下，在总共八个线程中只能有四个活动线程。在这种情况下，在时间T5-T8期间，处理器核心将是空闲的。这将导致仅仅50％的效率(即，4/8周期是活动的)。如果存在单个活动线程，则这个低于标准的性能将甚至更差。在这种情况下，处理器将仅仅是12.5％有效的。桶式线程处理可以避免与操作具有串行CPU的处理器相关联的许多成本。因此，在有许多活动线程的计算环境中，桶式线程处理是最合适的。

[0318] 上面描述的示例程序代码(例子1)(具有带有嵌入式条件语句的独立迭代的循环)可以被分配来在几个线程中运行，其中不同的线程执行不同的迭代，且因此桶式线程处理可以提高处理器在这样的计算环境中的性能。

[0319] 在for循环的迭代是独立的计算环境中，于是指令的不同线程可以执行该循环的不同迭代。这种类型的处理环境可以由在图3A中描绘的系统有效地执行。

[0320] 返回到图3A，每个核心304a-d可以实现桶式线程化处理。使用第一核心304a作为例子，核心可以通过利用桶式调度器310a来实现桶式线程处理。桶式调度器310a可以连接到被标记为T0-TM-1的多个线程寄存器308a，其中M表示在关于图4描述的桶式线程处理过程中正在被循环通过的线程的数量。

[0321] 每个线程寄存器文件308a包括可以存储与对应于线程寄存器文件的线程有关的信息的多个寄存器。在每个线程寄存器文件内的寄存器可以包含由处理器根据由线程运行的指令处理的数据，并且寄存器文件还包含程序计数器寄存器，其记录线程中的什么指令接下来将被执行。在图3A中，程序计数器指向存储在核心的指令存储器中的指令。在图3B中，线程寄存器中的程序计数器指向存储在公共指令存储器6002中的指令。

[0322] 来自在多核处理器外部的部件的数据可以经由私有数据高速缓存306a加载到线程寄存器中。

[0323] 桶式调度器310a可以通过首先访问与待处理的多个线程的第一线程相对应的线程寄存器文件308a来开始该过程。在图3A的例子中，第一线程寄存器被标记为T0。位于线程寄存器文件T0的存储器中的程序计数器可以向桶式调度器310a发信号通知哪个指令从指令存储器302a出栈以用于执行。基于从指令存储器302a接收到的指令，桶式调度器可从线程寄存器文件T0加载适当的数据，并将该信息传递给算术逻辑单元312a，然后算术逻辑单元312a执行该指令。

[0324] 在下一个时钟周期上，桶式调度器可以在线程寄存器文件T1上发起相同的过程，并且在每个随后的周期上可以发起对于线程寄存器文件T2到TM-1的过程，其中M等于核心可以处理的线程的数量。一旦来自每个线程的指令已经被处理，桶式调度器302a就可以通

过访问线程寄存器文件T0并确定第一指令是否已经完成而返回到第一线程。如果它已经完成，那么桶式调度器可以处理第一线程的第二指令。如果它没有完成，则桶式调度器可以继续前进到第二线程，在这个周期什么也不执行。

[0325] 该过程可以重复，直到每个线程中的所有指令都由核心304a执行为止。并行地，其他核心304b-d也可以在分配给它们的相应程序线程上执行相同的桶式线程处理过程。因此，作为例子，在四核处理器(每个包含8个线程寄存器)中，核心304a-d可以在32个单独的线程(4个核心×8个线程/核心)上并行地操作。以这种方式，包含嵌在具有在很大程度上独立的迭代的循环内的条件语句的程序代码的处理可以比使用只可在一个线程上以串行方式操作的单核处理器更快地被处理。

[0326] 除了利用桶式线程处理之外，通过采用共享数据高速缓存也可以进一步改善多核处理器的操作。共享数据高速缓存由多核处理器的核心中的每一个访问。图5A示出根据本公开的例子的被配置为执行桶式线程处理并利用共享数据高速缓存的示例性多核处理器。图5A所示的多核处理器类似于图3A的多核处理器。特别地，核心504a-d与图3A的核心304a-d类似地被配置。指令存储器502a-d、线程寄存器文件508a-d、桶式调度器510a-d、算术逻辑单元512a-d可以与在图3A中的它们的对应物相同，且因此在以上关于图3A的讨论中可以找到它们的功能的讨论。

[0327] 与私有数据高速缓存306a-d(其中每个核心具有与它相关联的私有数据高速缓存)相反，图5A的多核处理器包括共享数据高速缓存514。一般来说，私有数据高速缓存可以在没有使存储器对于几个处理器可访问所必需的开销的情况下实现，因此允许多核处理器的每个核心以最小的存储器时延访问数据高速缓存。但是，当私有数据高速缓存可能包含彼此不一致(不同步)的数据版本时，私有数据高速缓存可导致核心间高速缓存一致性问题。解决这些高速缓存一致性问题可能花费额外的芯片空间、功率和时钟周期，且因此使效率降级。

[0328] 当多核处理器执行图像处理时，根据所有核心已知的预定方案来将经处理的图像的像素存储在共享高速缓存存储器中。这使不同的核心能够容易访问相同的像素并实现了减少的高速缓存未命中。预定方案的非限制性例子是在共享存储器高速缓存内连续存储属于该图像的像素。

[0329] 作为例子，在如上面讨论的视觉处理计算环境中，多核处理器的每个核心可以同时处理图像的不同部分，并将图像的经处理的部分保存在它自己的私有数据高速缓存上。但是，如果外部存储器需要最新的图像，则必须访问每一个数据高速缓存以查看哪个私有数据高速缓存具有图像的最新部分。过程必须被放置到适当的地方以确保每个私有数据高速缓存拥有的数据的副本是最新的。确保高速缓存间一致性的这个过程从功率和时延立场来看可能是昂贵的，因为额外的功率和时间必须被花费在高速缓存间一致性上。

[0330] 但是，使用共享数据高速缓存，将避免这些高速缓存间一致性要求。如图5A所示，每个核心504a-d可连接到共享数据高速缓存514。共享数据高速缓存514可以包含多个组$0-$K-1。每个核心504a-d可以将数据写到多个组$0-$K-1并从该多个组$0-$K-1加载数据。由于每个核心现在共享包含在共享数据高速缓存514内的数据，而不是在私有数据高速缓存中维护数据的它自己的单个副本，因此消除了与解决高速缓存间一致性问题相关联的复杂性。

[0331] 虽然共享数据高速缓存可以除去与由私有数据高速缓存造成的高速缓存间一致性相关联的复杂性，但是它们但也可导致更长的存储器时延。换句话说，核心读取和写入共享数据高速缓存所花费的时间可能比读取和写入私有数据高速缓存所花费的时间更长。

[0332] 在利用采用桶式线程处理的多核体系结构的硬件加速器中，通过添加共享数据高速缓存而创建的增加的存储器时延可能不显著使性能降级。如在上面关于图3A和图4所讨论的，在桶式线程化处理器中，在从线程执行了一条指令之后，那个线程必须在桶式调度器返回到线程以执行在指令序列中的下一条指令之前等待预定数量的时钟周期。预定的时间量取决于可由核心处理的线程的数量。在图3A和图4的例子中，包含八个线程寄存器、因此处理八个线程的核心将不会每八个时钟周期返回执行来自线程的指令。

[0333] 在线程之间的后续指令的执行之间存在预定的时间量的事实可能致使与共享数据高速缓存相关联的增加的存储器时延从性能立场看是未决的。这可能是由于在线程中的后续指令的执行之间的时间量可能大于访问共享数据高速缓存所需的时间量的事实。因此，共享数据高速缓存相对于私有数据高速缓存花费较长时间量来访问的事实是不重要的，因为到在线程中的下一条指令准备好被执行的时候，足够的时间已经过去以允许核心访问共享数据高速缓存。

[0334] 采用共享数据高速缓存的另一个可能的优点在于它的使用可以使高速缓存未命中的数量最小化。当处理器的核心不能在它的数据高速缓存上找到所需的数据并且必须从另一级别的高速缓存或外部存储器中检索所需的数据时，高速缓存未命中发生。由于私有数据高速缓存在容量方面比共享数据高速缓存更小，因此在私有数据高速缓存上可能不存在所需数据的机会比如果使用更大的共享数据高速缓存时更高。此外，如果不同的线程处理相关的数据项，则将数据带入到高速缓存中的一个线程可能帮助在同一高速缓存行中需要数据的另一个线程。使用私有数据高速缓存，数据的2个或更多个物理副本将存在，一个副本在需要它的每个核心的高速缓存中，因而浪费了高速缓存存储器空间以及将数据提取到几个高速缓存而不是仅仅一个高速缓存中所必需的时间。

[0335] 虽然具有核心间共享数据高速缓存的桶式线程化多核处理器可以在功率和速度方面产生更大的处理效率，但是如上面讨论的，它们可能对串行计算环境不是理想的。如果程序或代码具有可以被并行化的任务和必须被串行地处理的任务的混合，那么桶式线程化多核处理器可能在最佳效率和低于标准的效率的时期之间振荡。当正在操作可以并行地运行的线程时，可以实现桶式线程化处理器的优点。但是，当串行代码被操作时，处理器可能变得相对低效。因此，包括被优化以对必须串行地运行的代码进行操作的单独的串行处理器可能是有利的。

[0336] 图6A示出根据本公开的例子的具有串行计算机处理单元的示例性桶式线程化多核处理器。图6A的例子与关于图5A讨论的示例多核处理器相同，其中增加有串行处理器616。因此，下面没有描述的各个单元的讨论可关于在上面讨论的图5A中的它们的对应物的讨论被找到。

[0337] 图6A的多核处理器例子包括可用于处理单线程化程序代码的串行处理器616。串行处理器将需要与所有的其他核心进行通信，以便能够交换数据。例如，在一个例子中，串行处理器616可以接收由核心604a-d准备的数据，串行地处理它，并且计算核心604a-d随后可以访问的结果。

[0338] 在串行处理器616和核心604a-d之间的这种数据交换可以以各种方式来实现。例如，核心604a-d和串行处理器616可以共享本地存储器(图6B中的6004)。在另一个例子中，核心604a-d可以将数据写到先进先出(FIFO)队列(图6C中的6006)中。串行处理器616然后可以从FIFO队列中读取数据，并将它的计算的结果写到另一个FIFO队列(未示出)。在另一个例子中，核心604a-d和串行处理器616可以通过使用直接存储器存取控制器(DMA)或它自己的数据高速缓存(未示出)来直接读取它们的数据和将它们的数据写到外部。

[0339] 串行处理器可以通过防止多核处理器在每当其需要执行串行计算时必须与SoC的主CPU进行通信或必须在桶式线程化处理器(其中这种串行计算将低效地运行)上运行串行计算来提高SoC的效率。

[0340] 当不同的线程应用处理重叠或部分重叠的像素的图像处理算法时，使用共享高速缓存是非常有益的。例如，FAST角检测器(见http://www.wikipedia.org;Edward Rosten和Tom Drummond在2015年10月的IEEE International Conference on Computer Vision的第1508-1511页发表的“Fusing points and lines for highperformance tracking”以及Edward Rosten和Tom Drummond在2006年5月的European Conference on Computer Vision的第430-443页上发表的“Machine learning for high-speed corner detection”)使用16像素的圆(半径为3的Bresenham圆，见图6D)来分类候选点p是否实际上是角。圆中的每个像素都从整数1到16顺时针地被标记。如果圆圈中的N个连续的像素的集合都比候选像素p的强度(由I_p表示)加上阈值t更亮或者都比候选像素p的强度减去阈值t更暗，则将p分类为角。

[0341] 条件可以被写成：

[0342] a.条件1:N个连续的像素的集合S，$\forall x \in S$，x的强度(I_x)＞I_p+阈值t

[0343] b.条件2:N个连续的像素的集合S，$\forall x \in S$，I_x<I_p-t

[0344] 所以当两个条件中的任何一个被满足时，候选p可以被分类为角。存在选择N、连续像素的数量和阈值t的折衷。一方面，检测到的角点的数量不应该太多，另一方面，高性能不应通过牺牲计算效率来实现。在没有机器学习的提高的情况下，N通常被选择为12。高速测试方法可应用于排除非角点。

[0345] 用于拒绝非角点的高速测试通过检查4个示例像素(即，像素1、9、5和13)来操作。因为至少应该有12个连续的像素，不管是都比候选角更亮还是更暗，所以在这4个示例像素中至少存在3个像素都比候选角更亮或者更暗。

[0346] 首先检查像素1和9，如果I_1和I_9都在[I_p-t，I_p+t]内，则候选点p不是角。

[0347] 否则，进一步检查像素5和13以检查它们中的三个比I_p+t更亮还是比I_p-t更暗。如果存在它们中的3个更亮或者更暗，则其余像素被检查以得出最终结论。

[0348] 而且根据发明人，在他的第一篇论文中，需要平均3.8个像素来检查候选角像素。与对于每个候选角的8.5个像素比较，3.8是可以大大提高性能的很大的减少。

[0349] 但是，对这个测试方法存在几个弱点：

[0350] a.对于N<12，高速测试不能很好地被一般化。如果N<12，则候选点p是角且4个示例测试像素中只有2个都比I_p+t更亮或比I_p-t更暗将是可能的。

[0351] b.检测器的效率取决于这些所选择的测试像素的选择和定序。然而，所选择的像素不太可能是最佳的，这涉及角外观的分布。

[0352] c.多个特征被检测到彼此相邻。

[0353] 利用对机器学习的改进

[0354] 为了解决高速测试的前两个弱点，引入了机器学习方法来帮助改进检测算法。这种机器学习方法在两个阶段中操作。首先，在从目标应用领域中优选的一组训练图像上处理使用给定的N的角检测。通过最简单的实现来检测角，该实现实际上提取16个像素的环，并将强度值与适当的阈值进行比较。

[0355] 对于候选p，在圆上的每个位置$x\in\{1,2,3,\ldots,16\}$都可以由$p\rightarrow x$表示。每个像素的状态$S_{p\rightarrow x}$必须在以下三种状态之一中：

[0356] d，$I_{p\rightarrow x}\leqslant I_P-t$(更暗)

[0357] s，$I_P-t\leqslant I_{p\rightarrow x}\leqslant I_P+t$(相似)

[0358] b，$I_{p\rightarrow x}\geqslant I_P+t$(更亮)

[0359] 然后选择x(对于所有p都是相同的)将P(所有训练图像的所有像素的集合)分成3个不同的子集P_d、P_s、P_b，其中：

[0360] $P_d=\{p\in P:S_{p\rightarrow x}=d\}$

[0361] $P_s=\{p\in P:S_{p\rightarrow x}=s\}$

[0362] $P_b=\{p\in P:S_{p\rightarrow x}=b\}$

[0363] 其次，决策树算法，ID3算法应用于16个位置，以便达到最大信息增益。令K_p是布尔变量，其指示p是否是角，然后K_p的熵用来测量作为角的p的信息。对于像素的集合Q，K_Q的总熵(未归一化)为：

[0364] $H(Q)=(c+n)\log_2(c+n)-c\log_2 c-n\log_2 n$；

[0365] 其中，$c=|\{i\in Q:K_i\text{为真}\}|$(角的数量)；

[0366] 其中，$n=|\{i\in Q:K_i\text{为假}\}|$(非角的数量)。

[0367] 然后，信息增益可以被表示为：$H_g=H(P)-H(P_b)-H(P_s)-H(P_d)$。

[0368] 递归过程被应用于每个子集，以便选择可以使信息增益最大化的每个x。例如，首先选择x来将P划分为具有最多信息的P_d、P_s、P_b；然后对于每个子集P_d、P_s、P_b，选择另一个y以产生最多的信息增益(注意，y可以与x相同)。这个递归过程在熵为零时结束，使得在该子集中的所有像素是角或非角。

[0369] 这个生成的决策树于是可以被转换成程序代码，比如C和C++，其只是一堆嵌套的if-else语句。为了优化目的，使用配置文件引导的优化来编译代码。编码的代码稍后用作对于其他图像的角检测器。

[0370] 注意，使用此决策树算法检测到的角应与使用分段测试检测器的结果略有不同。这是因为决策树模型取决于训练数据，其不能涵盖所有可能的角。

[0371] 多核处理器可使每个线程(不管该线程在哪个核心上)处理图像行的子集：

[0372] parallel_for(y=0:height)

[0373] {for(x=0:width)

[0374] {corner[x,y]=evaluate_decision_tree(x,y);//evaluate_decision_tree是if/else嵌套的东西。}

[0375] }

[0376] 应注意，内部“for”循环以及在内部循环内的“if/else”是编译成标准类型的汇编

指令的标准C。

[0377] 使用并行的for是一种不太普遍类型的构造,但是存在对这种类型的东西的标准(例如,OpenMP)。

[0378] 关于并行的for的两个有趣的可选方案是静态vs动态工作划分。

[0379] 静态划分通过将并行的for实现为“for(y＝my_thread_id();y<height;y+＝total_num_threads)”来起作用。这样,使用总共32个线程(在MPC上,这是4个核心×每核心8个线程的结果),线程0处理行0、32、64.....;线程1处理行1、33、65、……等。

[0380] 动态划分通过将并行_for实现为“for(y＝atomic_add(counter,1);y<height;y＝atomic_add(counter,1)”来起作用。

[0381] 当应用动态划分时,SoC直到运行时间时才知道哪个线程处理哪个行(因此名称为“动态划分”),它们都共享原子计数器,并且每当线程完成一行时,它就使计数器递增。

[0382] 动态划分的优点是,如果在图像的一些部分中存在比其他部分少得多的工作,那么获得“容易的”行的线程将总体上简单地处理更多行,而获得更多的“难的”行的线程将总体上处理更少的行,但是它们都将在大约同一时间完成,而没有快速线程空闲地等待落后者。所以动态划分对于动态工作负载更好。

[0383] 然而,静态划分具有下面的优点:SoC没有原子计数器,但使用线程本地简单计数器,这些是非常便宜的(1-2个周期来使计数器递增vs可能几十或几百个周期)。

[0384] 每个处理器稍微不同地支持原子计数器。

[0385] 多核处理器可以在专用硬件中具有固定数量的原子计数器。根据本发明的另一个实施例,原子计数器可以通过将计数器存储在共享存储器中来实现。

[0386] 您可能更喜欢具有角-x、y坐标的列表而不是产生宽x高布尔矩阵角[x,y]。这在其中大多数像素不是角所以坐标列表是比布尔矩阵更紧凑的表示的常见情况下在存储器使用方面来说是更好的选择。然后,你想要有像“if(evaluate_decision_tree(x,y)){corner_list[index++]＝(x,y);}”的一些东西。为了使这起作用,你再次需要原子计数器,因此是atomic_add(index,1)而不是index++,因为你具有更新index的多个线程。

[0387] 图6E示出多核处理器,其与图6A的多核处理器的不同之处在于包括用于便于在由多个核心执行的不同线程之间的同步的线程间通信(ITC)8000(当被应用在图6A的多核处理器上时)或者(如图6E所示的,当便于在由多个核心和串行处理器执行的不同线程之间的同步时。

[0388] ITC 8000可存储原子计数器和/或数据锁。

[0389] ITC 8000可以经由互连8030或以任何其它方式耦合到处理器核心606a-d和串行处理器616(并且尤其耦合到FIFO 698a-e)。

[0390] 每个原子计数器更新请求可以包括操作码(递增、递减或重置)、用于识别应当被更新的原子计数器的原子计数器标识符和用于识别请求部件(处理器核心或串行处理器)的请求标识符以及甚至线程。

[0391] 图6F示出处理器核心606a-d和串行处理器616的各种FIFO 698a-e。这些FIFO存储用于更新(递增、递减或重置)原子计数器的请求。

[0392] FIFO 698a-e由判别器8010读取,该判别器8010确定哪个请求要发送ITC 8000的原子计数器区域8020的输入缓冲器8012。如果在同一周期处接收到用于更新同一原子计数

器的两个原子计数器请求，则判别器8010选择原子计数器请求中的一个，并在将另一个原子计数器请求发送到输入缓冲器8012之前将所选择的请求发送到输入缓冲器8012。

[0393] 原子计数器区域8020包括存储器模块8002，其包括用于存储多个原子计数器值的多个条目8004a-n。

[0394] 存储器模块地址8004a-n位于解复用器8001和复用器8003之间。复用器8003之后是处理器8005，其能够使由复用器8003输出的原子计数器的值递增、递减和重置。由复用器8003输出的原子计数器的值还可以用作对核心(或串行处理器)的(经由输出端8008发送的)输出，该核心被请求更新或读取原子计数器。原子计数器的输出可能是更新之前(如附图中所示)或更新之后的值。

[0395] 所选择的值的原子计数器被用来控制复用器8003，以确定哪个原子计数器应被馈送给处理器8005。原子计数器更新请求的操作码被馈送到处理器8005并且控制由处理器8005执行的操作。所选择的值的原子计数器也用于控制解复用器8001，以确定在哪里存储经更新的原子计数器(存储器模块8002的哪个存储器条目)。请求者标识符被用于确定经由输出端8008输出并且可以通过互连8030传播的更新的原子计数器的命运。解复用器8001可以是行编码器。

[0396] 虽然图6F示出用于每周期更新多达一个原子计数器的单个处理器8005，但ITC 8000可以包括多个处理器8005和/或用于每周期更新多个不同原子计数器的任何其他电路。

[0397] 根据本发明的实施例，原子计数器表示不同的线程。通过确保只有单个部件可以一次更新某个原子计数器，提供了无冲突的线程分配过程。

[0398] 原子计数器可以指向指令存储器中的地址(其可以表示线程的开始)，可以指向数据单元，等等。

[0399] 使用用于存储原子计数器的专用单元(ITC)防止与访问存储在线程的执行期间处理的数据的存储器单元相关联的冲突和瓶颈。

[0400] 通过专用硬件实现原子计数器已经被发现比由通用处理器实现原子计数器快十倍。

[0401] 已经发现，在使用简单的单处理器硬件时，每周期更新单个原子计数器提供了足够的系统性能。

[0402] 原子计数器也可用于计算直方图。原子计数器可以存储直条的计数(频率)，和/或可以指向在分配给直条的数据单元的存储器区域内的数据单元的目的地。

[0403] 当在同一周期处接收到属于同一直条的多个数据单元时，可以使用原子计数器来更新计数和/或计算多个数据单元的目的地址而没有错误，使得计数器将被多个数据单元的数量更新，并且多个数据单元中的每一个接收不同的目的地址。

[0404] 因此，根据上文，本公开的一些例子针对一种多核处理器设备，其包括：多个处理核心，其配置成使用桶式线程处理来执行软件程序的多个指令线程；以及共享数据高速缓存，其配置成将数据发送到多个处理核心中的至少两个并从多个处理核心中的至少两个接收数据，共享数据高速缓存存储与在多个指令线程中的至少一个中的第一指令的执行相关联的数据。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，多个处理核心包括：多个线程寄存器，其中多个线程寄存器中的至少一个与多个指令线程中的

指令线程相关联;指令存储器,其中指令存储器存储与多个指令线程、一个或更多个算术逻辑单元以及桶式调度器相关联的一个或更多个指令,其中桶式调度器接收第一时钟信号、从指令存储器加载一个或更多个指令中的第一指令并且通过以下操作来执行第一指令:从第一线程寄存器检索数据,其中第一线程寄存器与和第一指令相关联的指令线程相关联,并且将接收到的数据发送到一个或更多个算术逻辑单元用于处理。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,桶式调度器接收第二时钟信号、从指令存储器加载一个或更多个指令中的第二指令并且通过以下操作来执行第二指令:从第二线程寄存器检索数据,其中第二线程寄存器与和第二指令相关联的指令线程相关联,并将接收到的数据发送到一个或更多个算术逻辑单元用于处理。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,多个线程寄存器中的每个线程寄存器被配置为加载与指令线程相关联的数据,线程寄存器与该指令线程相关联。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,多个处理核心同时执行多个指令线程。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,多个核心中的一个或更多个向共享数据高速缓存传输数据并从共享数据高速缓存接收数据,并且其中共享数据高速缓存被配置为存储从多个指令线程的执行产生的数据。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,该设备包括被配置为执行单线程化程序代码的串行处理器。

[0405] 本公开的一些例子针对处理软件程序的多个指令线程的方法,该方法包括:使用多个处理核心执行多个指令线程,其中多个处理核心使用桶式线程处理来执行多个指令线程,以及使用共享数据高速缓存向多个处理核心中的至少两个发送数据并从多个处理核心中的至少两个接收数据,其中共享数据高速缓存在多个指令线程中的至少一个中存储与第一指令的执行相关联的数据。除了上面公开的一个或更多个例子之外或作为其可选方案,在一些例子中,该方法包括:在多个处理核心的桶式调度器处接收第一时钟信号,从指令存储器加载第一指令,其中指令存储器存储与多个指令线程相关联的一个或更多个指令,通过以下步骤来执行第一指令:从第一线程寄存器检索数据,其中第一线程寄存器与和第一指令相关联的指令线程相关联,并且将接收到的数据发送到一个或更多个算术逻辑单元用于处理。除了上面公开的一个或更多个例子之外或作为其可选方案,在一些例子中,桶式调度器接收第二时钟信号,从指令存储器加载一个或更多个指令中的第二指令,并且通过以下操作来执行第二指令:从第二线程寄存器检索数据,其中第二线程寄存器与和第二指令相关联的指令线程相关联,并将接收到的数据发送到一个或更多个算术逻辑单元用于处理。除了上面公开的一个或更多个例子之外或作为其可选方案,在一些例子中,多个线程寄存器中的每个线程寄存器加载与指令线程相关联的数据,该线程寄存器与该指令线程相关联。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,多个处理核心同时执行多个指令线程。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,多个核心中的一个或更多个向共享数据高速缓存传输数据并从共享数据高速缓存接收数据,并且其中共享数据高速缓存被配置为存储从多个指令线程的执行产生的数据。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,该方法包括:使用串行处理器执行一个或更多个单线程化程序代码。

[0406] 本公开的一些例子针对非临时计算机可读存储介质,其上存储有用于处理软件程

序的多个指令线程的指令的集合，该指令在由计算设备执行时使计算设备：使用多个处理核心来执行多个指令线程，其中多个处理核心使用桶式线程处理来执行多个指令线程，并且使用共享数据高速缓存向多个处理核心中的至少两个发送数据和从多个处理核心中的至少两个接收数据，其中共享数据高速缓存存储与在多个指令线程中的至少一个中的第一指令的执行相关联的数据。除了以上公开的一个或更多个例子之外或作为其可选方案，使计算设备：在多个处理核心的桶式调度器处接收第一时钟信号，从指令存储器加载第一指令，其中指令存储器存储与多个指令线程相关联的一个或更多个指令，通过以下操作来执行第一指令：从第一线程寄存器检索数据，其中第一线程寄存器与和第一指令相关联的指令线程相关联，并且将接收到的数据发送到一个或更多个算术逻辑单元用于处理。除了上面公开的一个或更多个例子之外或作为其可选方案，桶式调度器接收第二时钟信号，从指令存储器加载一个或更多个指令中的第二指令，并且通过以下操作执行第二指令：从第二线程寄存器检索数据，其中第二线程寄存器与和第二指令相关联的指令线程相关联，并将接收到的数据发送到一个或更多个算术逻辑单元用于处理。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，多个线程寄存器中的每个线程寄存器加载与指令线程相关联的数据，该线程寄存器与该指令线程相关联。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，多个处理核心同时执行多个指令线程。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，多个核心中的一个或更多个向共享数据高速缓存传输数据并从共享数据高速缓存接收数据，并且其中共享数据高速缓存被配置为存储从多个指令线程的执行产生的数据。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，使计算设备使用串行处理器执行一个或更多个单线程化程序代码。

[0407] 使用指令链编码的多线程化核心

[0408] 使用上述处理器体系结构，在利用桶式线程处理的四核处理器中，处理器作为一个整体可以在单个时钟周期期间运行四条指令。换句话说，在一个时钟周期期间，处理器的每个核心都运行单个指令。增加在单个时钟周期期间运行的指令数量的一种方法是增加处理器内的核心的数量。但是，在每个处理器内添加更多的核心以便增加在单个时钟周期期间执行的指令的数量也可导致对于处理器的增加的功率消耗和较大的封装。

[0409] 在单个时钟周期期间增加处理吞吐量而不增加在每个处理器内的硬件的数量的一种方式是增加在单个时钟周期期间处理器的单个核心可以并行地运行的指令的数量。通过利用处理器所遇到的某些指令的共同或共享特性，单个核心可以在单个时钟周期期间执行多个指令。

[0410] 采取下面的伪代码例子(“例子2”)：

[0411] i.a＝OP2(OP1(b,c),d)

[0412] 上面的例子将在指令内嵌套指令，在本文被称为“指令链”。第一指令OP1(b,c)的结果被用作对指令OP2的输入。在常规处理器中，上述编码例子可以作为两个单独的指令来执行。在第一指令中，将对输入b和c执行操作1(OP1)。结果将接着被写到存储器。在第二指令中，可以对输入d执行操作2(OP2)，并将OP1的结果存储在存储器中。

[0413] 将操作1的结果写到外部存储器可能是浪费的，因为那个结果不被用户或程序需要，而是只需要作为第二操作的输入。换句话说，用户或程序不需要这个“中间结果”---由

OP2产生的输出。由于用户或程序不需要OP2的输出，因此减少实现链式指令所需的指令的数量可能是有用的。

[0414] 图7示出根据本公开的例子的示例性常规双发射处理器。图7的例子可以显示这种常规处理器可如何处理链式指令。图7的双发射处理器可以包括可以耦合到多个指令寄存器604a-b的程序存储器/指令存储器602。指令寄存器604a-b每个可以进一步耦合到寄存器文件606。

[0415] 两个独立的指令可以从程序存储器602被读取到指令寄存器604a和604b中。一个指令可以被加载到指令寄存器604a中，而另一个可以被加载到指令寄存器604b中。一旦指令寄存器604a和604b被加载，它们就可以将它们要执行的指令的操作数传递到寄存器文件606。

[0416] 使用指令寄存器604a作为例子，一旦来自程序存储器602的指令被加载到指令寄存器604a中，寄存器就可以将要操作的操作数传递给寄存器文件606，同时也将要执行的指令传递给执行单元608。

[0417] 在一个例子中，执行单元608可以包括一个或更多个算术逻辑单元、加载存储单元等，其可以执行由指令寄存器604a-b提供给它的指令。执行单元608可以具有对称或不对称的能力连同确定在执行单元608中的哪个单元将执行给定操作的逻辑。

[0418] 寄存器文件708可以将要被执行的操作数加载到执行单元608中。在图7的例子中，执行单元608可以具有四个读取端口(例如，对每个指令寄存器有两个读取端口)以在输入中读取指令的操作数，并且具有两个写入端口(对每个指令有一个写入端口)以将指令的结果写到寄存器文件606。

[0419] 在链式指令的例子中，指令寄存器604a和604b可以各自接收操作码。使用上面的例子2的命名法，第一指令寄存器604a可以接收OP1，而第二指令寄存器604b可以接收OP2。执行单元608可以接收两个单独的操作码，从每个指令寄存器有一个操作码。因此，在常规处理器中，链式指令可能需要总共六个寄存器文件端口(四个输入和两个输出)，并且程序存储器可以要求每周期读取两个指令。

[0420] 但是，如果链式指令可以被编码，使得它只使用一个操作码，并且任何未使用的结果(例如中间结果)都不被写到寄存器，则就寄存器文件端口和程序存储器带宽可以被保持而言，可以减少双发射处理器的总开销。

[0421] 图8示出根据本公开的例子的被配置为接受链式指令的示例性双发射核心处理器。图8中所示的处理器被配置为一次处理一个线程，并且因此仅具有一个指令寄存器704。

[0422] 采用以下示例操作(“例子3”)：

[0423] i.a＝b*c+d

[0424] 照惯例，上述等式可以在两个单独的指令中被编码。第一指令(即要执行的第一指令)可以是tmp＝b*c，其中“tmp”代表临时存储器或变量。第二指令可以是a＝tmp+d。在图7所示的配置中，这些指令将连续/串行地被操作。由于第二指令取决于第一指令的结果的事实，同时运行它们将是不可能的。相反，利用链接的处理器可以将与上述等式相关联的两条指令编码为一条指令。

[0425] 在图8的例子中，利用紧凑链式指令编码的处理器(本文中“紧凑链式指令编码”和“指令链接”将可互换地使用)可以将等式a＝b*c+d编码为单个指令。在第一周期中，指令寄

存器704可以包含上面的例子3的链式编码指令。与第一独立指令相关联的操作数(即,b*c)可以被发送到寄存器文件708,并且乘法指令可以从指令寄存器704被发送到执行单元712。与上述行动并行地,指令寄存器704可以将与第二从属指令相关联的操作数和指令(即,a、d和+)发送到链式指令寄存器706。链式指令寄存器706可以接收链式编码指令的从属部分,并且一旦链式指令的独立部分已经被执行就将该指令发送给适当的执行单元用于处理。仍然在第一周期内,执行单元712的单元之一计算b*c并将结果保存在中间寄存器710中。

[0426] 在下一个周期处,执行单元712的执行单元可以从中间寄存器710(其在上面的例子中包含b*c)以及从链式指令寄存器706(其在上面的例子中包含a、d和+)获取操作数并且在执行单元712中的一个处执行+指令。

[0427] 此外在同一周期期间,指令寄存器704可以加载新的指令,其然后被发送给执行单元712用于执行。因此,在一个周期中,两个操作(链式指令的从属部分和新指令)可以并行地发生。以这种方式,即使链式指令被呈现给执行单元712,但是在任何给定的周期处,两个操作仍然可以并行地发生,从而维持对于处理器的增加的吞吐量。

[0428] 如在上面的例子中所演示的,将链式操作编码为一个操作而不是两个单独的操作可以减少指令寄存器空间和存储器带宽的使用,而同时减少执行链式指令所需的读取和写入端口的数量。

[0429] 通过链式指令编码赋予处理器的某些优点可以在桶式线程化处理器中放大。例如,由于紧凑编码而导致的寄存器文件的输入端口和输出端口的减少可以在桶式线程化处理器中放大,其中具有附加读取端口的成本将乘以在桶式线程化处理器中的寄存器文件的数量。如上面关于图5A所讨论的,具有八个线程的桶式线程化处理器将具有八个寄存器文件。因此,输出端口和输入端口可通过使用链式指令编码而被节省的事实可以意味着在具有八个线程的桶式线程化处理器中,由一个寄存器文件引起的端口节省可以乘以由桶式线程化处理器利用的多个寄存器文件。

[0430] 图9示出根据本公开的例子的被配置用于桶式线程处理和接受链式指令的示例性双发射核心。图9的例子可以以与在图8中公开的例子实质上相同的方式操作。一个差异可以是,由于每个线程每N个周期只操作一次的事实,寄存器文件的附加输入端口可以被保持,其中N等于由桶式线程化处理器操纵的线程的数量(即,在图5A的例子中是八)。由于这个事实,在特定线程是空闲的周期期间,可以读入与链式操作(例如从属指令)相关联的操作数。

[0431] 转到图9并利用如在图7的例子中使用的示例指令a＝b*c+d(例子3),在第一周期处,包含执行a＝b*c+d的命令的链式指令可以从程序存储器802被加载到指令寄存器804。指令寄存器804然后可以将操作数b和c发送到寄存器文件806a,其可以对应于要由处理器执行的N个线程中的第一线程所使用的指令寄存器,并且将加法指令(+)发送到执行单元812。同时,指令寄存器804还可以将对于链式命令的操作数和指令(d和+)发送到链式指令寄存器808a。

[0432] 在周期1期间,b*c的乘法可以由执行单元812执行,并且结果可以被存储在中间寄存器810a中。中间寄存器810a可专用于存储第一线程的中间结果(例如,程序或用户不需要的结果)。以这种方式,每个线程可以具有它自己的中间寄存器(即,线程N可以具有如所示的中间寄存器810n)。

[0433] 在下一个周期处，对应于a、d的链式指令寄存器808a的内容可以被发送到寄存器文件806a，而+操作数可以被发送到执行单元812。如上面讨论的，到周期2开始的时间为止，b*c的结果可被存储在中间寄存器810a中。因此在周期2处，处理器可以开始处理链式命令的第二部分。

[0434] 同时，来自第二线程的新指令可以由处理器以与在周期1中b*c被处理的相同的方式来处理。因此在第二周期处，两个操作可以同时发生。可以通过将存储在中间寄存器810a中的b*c加到由寄存器文件806a从链式指令寄存器808a加载的操作数d来完成第一线程的链式操作。

[0435] 以这种方式，不是必须等待直到第一线程的回合再次出现(在周期N+1处)以执行链式指令的第二部分，链式指令的第二部分可以在与另一线程相关联的指令被执行的时间期间同时被执行。

[0436] 图9的处理器还可以包括复用器814。复用器814可以包含决定执行单元812的哪个单独的执行单元将在操作数上操作的逻辑。

[0437] 图10A和图10B示出当处理器正在处理链式指令时分别在第一时间段和第二时间段期间发生的操作。在图9中描述的处理器可以“时控(clocked)”，意味着时钟信号被馈送到一些或全部部件。时钟信号可以帮助使处理器的各个部件同步。当处理器的部件被时控时，由处理器执行的功能可以被描述为在时钟周期或时间段内发生。作为例子，第一时间段可以描述接收第一时钟信号和第二时钟信号的部件之间的时间段。第二时间段可以描述在第二时钟信号和第三时钟信号的接收之间的时间段，依此类推。

[0438] 图10A示出根据本公开的例子的在第一时钟周期期间的示例性链式指令处理流程。将关于在图9中提供的硬件图来描述图10A的处理流程。在步骤S902处，可以将链式指令从程序存储器802加载到指令寄存器804中。链式指令可以包含两个部分：独立指令部分和从属指令部分。独立指令部分可以指可以被执行而不必首先等待中间结果的链式指令的部分。在a＝b*c+d的例子中(例子3)，b*c可以指的是“独立部分”。从属指令部分可以指在它可以被执行之前必须等待中间结果产生的链式指令的部分。在上面的等式中，操作数a和d以及[+]操作都可以表示从属部分。应该注意，链式指令可以具有多于一个独立和从属指令部分。

[0439] 在步骤S904处，独立指令部分及其相关联的操作数可以被传输给执行单元812用于处理。独立指令部分的操作数可以从指令寄存器804传输到寄存器文件806a，而指令本身可以从指令寄存器804传输到执行单元812。执行单元812可以从寄存器文件806a中加载独立指令部分的适当操作数，以执行独立指令部分。

[0440] 在步骤S906处，从属指令部分被传输到链式指令寄存器808a，并在独立指令部分的执行发生时存储在那里。在步骤S908处，执行独立指令部分，并且在步骤S910处将结果存储在中间寄存器810a中。

[0441] 图10B示出根据例子的在第二时钟周期期间的示例性链式指令处理流程。在图10B的处理流程中，两个过程可以并行地发生：1)在步骤S912-S918(图10B的左侧)中的链式指令的从属部分的处理，以及2)在步骤S920-S926(图10B的右侧)中来自第二线程的指令的处理。如上面讨论的，在桶式线程化处理器中，单个处理器可以在每个时钟周期处理来自另一个线程的指令。在八线程化桶式处理器中，来自第一线程的指令可以在第一时钟周期处被

处理;来自第二线程的指令可以在第二时钟周期处被处理,依此类推。在第九个时钟周期处,第一线程可以再次“接收一个回合”,并执行线程中的下一个指令,或者如果先前的指令由于无论什么原因被停滞,则什么也不做。

[0442] 返回到图10B,在图10A中讨论的链式指令的从属部分的处理在步骤S912处开始。在步骤S912处,从属部分可以经由寄存器文件806a被传输到执行单元812,而与从属指令部分相关联的操作数可以被传输到寄存器文件806a。在步骤S914处,独立指令部分的结果可以从中间寄存器810a传输到执行单元812。在步骤S916处,可以在执行单元上执行从属指令部分,并且在步骤S918处,结果可以被存储在寄存器文件806a中。

[0443] 如以前讨论的,在桶式线程化处理器中,每个单独的线程可以在一个时钟周期期间接收一个回合。因此,在第二时钟周期中,第二线程的指令可以与第一线程的链式指令的从属指令部分的执行并行地被执行。

[0444] 在步骤S920处,可以将来自第二线程的指令加载到寄存器文件806b(未示出)。寄存器文件806b可以表示专用于源自第二线程的操作数的寄存器文件。在步骤S922处,与第二线程相关联的指令和操作数可以被传输到执行单元812用于执行,其可在步骤S924处发生。最后,在步骤S926处,第二线程的指令的结果可以存储在寄存器文件806b中。

[0445] 使来自第一线程的从属指令部分与来自第二线程的指令并行地被执行可以导致处理器的提高的吞吐量。链式指令可以在一个回合中被执行,而不是消耗线程的两个“回合”来执行链式指令。

[0446] 也可以在其中有多于两个指令部分和多个操作数的情况中应用指令链接,诸如下面的伪代码例子(“例子4”):

[0447] i.A=OP1(OP2(OP3(B,C)+d)+e)

[0448] 在上面的伪代码例子中,OP2取决于OP3的执行,而OP1取决于OP2的执行。图8的处理器例子也可以通过确保每个中间结果存储在中间寄存器中并且在与程序的另一个线程相关联的时钟周期期间操作来处理这些类型的链式操作。

[0449] 在一些例子中,操作数d(链式指令的第二部分)的读取可以在周期2而不是周期1期间发生。尽管这个例子可以消除对链式指令寄存器向寄存器文件发送操作数的需要,但是它可能不改变寄存器文件所需的读取端口的量,这是因为三个读取端口是寄存器文件执行所有必要的操作最大程度地所需的东西。与链式指令相关联的操作数是来自链式指令寄存器还是指令寄存器不改变寄存器文件所需的读取端口的数量。

[0450] 在一些例子中,可以通过采用隐式链接来改变被发送到寄存器文件806a的操作数的量。可以通过用链式指令的结果覆写链式指令的操作数来利用隐式链接。因此,使用上面的例子3,不是将操作编码为a=b*c+d,操作可替代地被编码为a=b*c+a。在本例中,输入操作数a在链式指令结果产生后被覆写。以这种方式,而不是必须将四个操作数(a、b、c、d)发送到寄存器文件,可以只发送三个操作数(a、b、c)。

[0451] 虽然具有核心间共享数据高速缓存的桶式线程化多核处理器可以在功率和速度方面产生更大的处理效率,如上面讨论的,但是它们可能对串行计算环境不是理想的。如果程序或代码具有可以被并行化的任务和必须被串行地处理的任务的混合,那么桶式线程化多核处理器可能在最佳效率和低于标准的效率的时期之间振荡。当可以并行地运行的线程正在操作时,可以实现桶式线程化处理器的优点。但是,当串行代码被操作时,处理器可能

变得相对低效。因此，包括单独的串行处理器可能是有利的，该串行处理器被优化以对必须串行地运行的代码进行操作。

[0452] 图10C示出了根据本公开的例子的被配置用于桶式线程处理和接受链式指令的示例性双发射核心。

[0453] 双发射核心包括除法器DIV 7026、第一ALU(AUA)7036、第二ALU(AUB)7038、加载存储单元7054、指令寄存器(IR)7004、链式指令寄存器(CIR)7006，线程寄存器TID 7010，包括寄存器文件RF0-n 7012.0-n的、中间寄存器ITM 7014.0-n的、第一寄存器Rsc 7016.0-n的和第二寄存器RFR 7018.0-n的多个集合、第一分配逻辑7020、第一和第二除法器输入缓冲器7022和7024、除法器输出缓冲器7028、第一和第二AUA输入缓冲器7032和7034、AUA输出缓冲器7037、第一和第二AUB输入缓冲器7042和7044、AUB输出缓冲器7046、输出总线7070、写回队列(WBQ)7050、控制器7060和线程间通信8000，其用于便于在由多个核心执行的不同线程之间和可选地也在多个核心与串行处理器之间的同步。

[0454] 指令由解码器7011解码并被发送到IR和/或CIR，除非它们不能被临时执行(例如除法器太忙)并且它们被发送到阴影(shadow)IR和/或阴影CIR。每线程可能有一个阴影IR和CIR。

[0455] 每个线程(在(n+1)个线程中的)与部件集合相关联，该部件集合包括中间寄存器、寄存器文件(也被称为线程寄存器文件)、它自己的第一寄存器和它自己的第二寄存器。

[0456] 控制器7060可以通过控制在(a)与不同线程相关联的部件集合(b)除法器DIV 7026、AUA 7036和AUB 7038中的任一个之间的信息流来实现桶式调度。

[0457] 根据本发明的实施例，AUA 7036被分配用于执行独立指令，而AUB 7038被分配用于执行链式(从属)指令。

[0458] AUA 7036和AUB 7038可以以流水线方式操作，在此期间，AUA 7036可以对每个线程一个接一个地执行独立指令，并且AUB 7038可以对同一个线程一个接一个地执行从属指令。

[0459] 假定(a)有八个线程((n+1)＝8)，(b)控制器7060执行八个线程的桶式调度，(c)与寄存器文件RF0 7012.0、ITM 7014.0、Rsc 7016.0和RFR 708.0相关联的第一线程的独立指令在第一周期期间被执行。与独立指令相关的两个操作数从RFR 7018.0被馈送到AUA 7036的第一和第二输入缓冲器7032和7034。

[0460] 在这些假设下：

[0461] AUA在第一周期期间执行独立指令。独立指令的执行的结果由AUA 7036输出，被馈送到AUA输出缓冲器7038，经由总线7070到WBQ 7050并最终存储在ITM 7014.0中(在第一周期完成之后)。

[0462] 核心7000下一次执行第一线程的任何指令将在第九周期处。

[0463] 在第一周期之后且在第九周期之前，对于从属指令的执行所需的另一操作数被发送到Rsc 7016.6。Rsc 7016.0和RFR 7018.0可以使用时分复用由RF0 7012.0的相同输出端口来馈送。

[0464] 在第九周期期间：(a)将ITM 7014.0的和Rsc 7016.0的内容(经由逻辑7020)馈送到AUB 7046的第一和第二输入缓冲器7042和7044；(b)AUB 7046执行从属指令。从属指令的执行的结果由AUB 7046输出，被馈送到AUB输出缓冲器7048，经由总线7070到WBQ 7050并最

终存储在RF0 7012.0中。

[0465] 在第九周期期间,AUA可以执行第一线程的新的独立命令。

[0466] 参考图10B,不是从第二线程取出指令(S920),该方法可以包括从第一线程取出下一个指令并且由AUA执行来自第一线程的该下一个指令(应该是独立指令),同时AUB(在同一周期期间)执行较老的从属指令。

[0467] 应该注意,一个或更多个线程的执行可能涉及计算除法操作,这可能涉及访问除法器DIV 7026。

[0468] 因为除法操作是冗长的(它可能例如持续数十个周期,例如30个周期),所以除法器可以包括多个独立的除法器模块,其可以彼此并行地操作并减少核心的总时延。

[0469] 图10C示出根据本发明的实施例的除法器7026、第一和第二除法器输入缓冲器7022和7024以及两个除法器输出缓冲器7028.1和7028.2。

[0470] DIV 7026包括第一和第二除法器7026.6和7026.9,但是可以包括多于两个除法器。

[0471] 第一和第二除法器输入缓冲器7022和7024之后是除法器FIF0 7026.0,其可以累加多对操作数。在图10C中,除法器FIF0 7026.0具有二的深度,但是除法器FIF0 7026.0可能更深。

[0472] 除法器FIF0 7026.0之后是解复用器7026.2,其用于将一对操作数(来自除法器FIF0 7026.0)分配到第一和第二除法器7026.6和7026.9中的一个,特别是分配到在第一和第二除法器7026.6和7026.9之前的输入缓冲器对(7026.3和7026.4、7026.7和7026.8)之一。

[0473] 第一除法器7026.5的输出被馈送到输出缓冲器7028.1并馈送到总线7070。第二除法器7026.9的输出被馈送到输出缓冲器7028.2并馈送到总线7070。

[0474] 除法器控制器7026.1例如通过向解复用器7026.2发送选择信号、通过控制操作数对穿过除法器FIF0的传播、通过触发第一和第二除法器7026.6和7026.9以执行除法操作等来控制DIV 7026的操作。

[0475] 除法器控制器7026.1可以从第一和第二除法器7026.6和7026.9接收状态指示(例如忙线信号和/或空闲信号)。

[0476] 当第一和第二除法器7026.6和7026.9都忙时,除法器控制器7026.1可以通知(图10C的)控制器7060不向除法器7026发送任何操作数,直到第一和第二除法器7026.6和7026.9中的一个或更多个不忙为止。除法器控制器7026.1可以通过任何其他手段(例如,通过将忙线信号发送到逻辑7020、到WBQ 7050、通过发送中断等)来阻止新操作数对到除法器7026的发送。

[0477] 除法器控制器7026.1可以指示解复用器7026.2将一对操作数发送到不忙的除法器,并且防止向已经繁忙的除法器发送新的一对操作数。

[0478] 因此,根据上文,本公开的一些例子针对一种多核处理器设备,其包括:多个处理核心,其被配置为使用桶式线程处理来执行软件程序的多个指令线程,其中多个处理核心被配置为通过下列操作来执行与多个指令线程的第一线程相关联的链式指令:在第一时间段期间将链式指令加载到第一指令存储器,其中链式指令包括独立指令部分和从属指令部分;在第一时间段期间将独立指令部分和与独立指令部分相关联的一个或更多个操作数传

输到一个或更多个执行单元;在第一时间段期间将从属指令部分传输到链式指令寄存器;在第一时间段期间由一个或更多个执行单元执行独立指令部分;在第一时间段期间将独立指令部分的结果写到第一中间寄存器;在第二时间段期间将从属指令部分、与从属指令部分相关联的一个或更多个操作数以及独立指令的结果传输到一个或更多个执行单元;以及在第二时间段期间执行从属指令部分。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,链式指令与多个线程中的第一线程相关联,并且其中设备在第二时间段期间与链式指令的从属指令部分的执行并行地处理与多个线程中的第二线程相关联的指令。除了上面公开的一个或更多个例子之外或作为其可选方案,在一些例子中,与从属指令部分相关联的一个或更多个操作数包括输入操作数,并且其中执行从属指令部分包括用从属指令部分的执行的结果覆写输入操作数。除了上面公开的一个或更多个例子之外或作为其可选方案,该设备还包括复用器,该复用器被配置为确定哪个执行单元将执行链式指令的从属指令部分,以及哪个执行单元将执行与第二线程相关联的指令。除了上面公开的一个或更多个例子之外或作为其可选方案,该设备还包括共享数据高速缓存,其被配置为向多个处理核心中的至少两个发送数据并从多个处理核心中的至少两个接收数据,该共享数据高速缓存存储与多个线程中的至少一个中的指令的执行相关联的数据,共享数据高速缓存和多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。除了以上公开的一个或庚多个例子之外或作为其可选方案,与从属指令部分相关联的一个或更多个操作数在第二时间段期间被存储在第一寄存器文件中,并且其中与第二线程相关联的指令包括在第二时间段期间存储在第二寄存器文件中的一个或更多个操作数。除了以上公开的一个或多个例子之外或作为其可选方案,该设备还包括寄存器文件、链式指令寄存器和与多个指令线程中的每个线程相关联的中间寄存器。

[0479] 此外或可选地,设备在第二时间段期间与链式指令的从属指令部分的执行并行地处理与第一线程相关联的另一个独立指令部分。此外或可选地,每个处理核心包括用于执行第一线程的独立指令部分的第一执行单元和用于执行第一线程的从属指令部分的第二执行单元。

[0480] 本公开的一些例子针对一种用于在被配置为执行多个指令线程的桶式线程化处理器中执行链式指令的方法。提供了一种方法,该方法包括:在第一时间段期间将链式指令加载到第一指令存储器,其中链式指令包括独立指令部分和从属指令部分;在第一时间段期间将独立指令部分和与独立指令部分相关联的一个或更多个操作数传输到一个或更多个执行单元;在第一时间段期间将从属指令部分传输到链式指令寄存器;在第一时间段期间由一个或更多个执行单元执行独立指令部分;在第一时间段期间将独立指令部分的结果写到第一中间寄存器;在第二时间段期间将从属指令部分、与从属指令部分相关联的一个或更多个操作数以及独立指令的结果传输到一个或更多个执行单元;以及在第二时间段期间执行从属指令部分。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,链式指令与多个线程中的第一线程相关联,并且其中该方法包括在第二时间段期间与链式指令的从属指令部分的执行并行地处理与多个线程中的第二线程相关联的指令。除了以上公开的一个或更多个例子之外或作为其可选方案,在一些例子中,与从属指令部分相关联的一个或更多个操作数包括输入操作数,并且其中执行从属指令部分包括用从属指令部分的执行的结果覆写输入操作数。除了以上公开的一个或更多个例子之外或作为其

可选方案，在一些例子中，该方法包括确定哪个执行单元将执行链式指令的从属指令部分和与第二线程相关联的指令。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，该方法包括使用共享数据高速缓存来向多个处理核心中的至少两个发送数据和从多个处理核心中的至少两个接收数据，共享数据高速缓存存储与在多个线程中的至少一个线程中的第一指令的执行相关联的数据，共享数据高速缓存和多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，在第二时间段期间，与从属指令相关联的一个或更多个操作数被存储在第一寄存器文件中，并且其中与第二线程相关联的指令部分包括在第二时间段期间存储在第二寄存器文件中的一个或更多个操作数。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，多个指令线程中的每个线程具有它自己的寄存器文件、链式指令寄存器以及与它相关联的中间寄存器。

[0481] 本公开的一些例子针对非临时计算机可读存储介质，其上存储有用于处理软件程序的链式指令的指令的集合，指令在由计算设备执行时使计算设备：在第一时间段期间将链式指令加载到第一指令存储器，其中链式指令包括独立指令部分和从属指令部分；在第一时间段期间将独立指令部分和与独立指令部分相关联的一个或更多个操作数传输到一个或更多个执行单元；在第一时间段期间将从属指令部分传输到链式指令寄存器；在第一时间段期间由一个或更多个执行单元执行独立指令部分；在第一时间段期间将独立指令部分的结果写到第一中间寄存器；在第二时间段期间将从属指令部分、与从属指令部分相关联的一个或更多个操作数以及独立指令的结果传输到一个或更多个执行单元；以及在第二时间段期间执行从属指令部分。除了以上公开的一个或更多个例子之外或作为其可选方案，链式指令与多个线程中的第一线程相关联，并且其中使计算设备在第二时间段期间与链式指令的从属指令部分的执行并行地处理与多个线程中的第二线程相关联的指令。除了以上公开的一个或更多个例子之外或作为其可选方案，与从属指令部分相关联的一个或更多个操作数包括输入操作数，并且其中执行从属指令部分包括用从属指令部分的执行的结果覆写输入操作数。除了以上公开的一个或更多个例子之外或作为其可选方案，使设备确定哪个执行单元将执行链式指令的从属指令部分和与第二线程相关联的指令。除了上面公开的一个或更多个例子之外或作为其可选方案，使设备使用共享数据高速缓存来向多个处理核心中的至少两个发送数据和从多个处理核心中的至少两个接收数据，共享数据高速缓存存储与在多个线程中的至少一个线程中的第一指令的执行相关联的数据，共享数据高速缓存和多个处理核心在不依赖于私有数据高速缓存的情况下进行交互。除了以上公开的一个或更多个例子之外或作为其可选方案，在第二时间段期间，与从属指令相关联的一个或更多个操作数被存储在第一寄存器文件中，并且其中与第二线程相关联的指令包括在第二时间段期间存储在第二寄存器文件中的一个或更多个操作数。除了以上公开的一个或更多个例子之外或作为其可选方案，多个指令线程中的每个线程具有它自己的寄存器文件、链式指令寄存器以及与它相关联的中间寄存器。

[0482] 用于多个处理器系统的高速缓存一致性方案

[0483] 多个加速器设备和高速缓存使用

[0484] 如上面在图1的上下文中所讨论的，SoC 102可以具有多于一个加速器，例如加速器112和114。硬件加速器112和114可以进一步包括多核处理器，例如分别在多核加速器

200、300和500中的多核处理器204、304和504。

[0485] 图11是示出对于其中诸如SoC 102的系统可以具有多个加速器1601a-n(每个都包括处理器1604a-n)的情况的高速缓存配置1600的广义图。注意,图11可以省略可被包括在加速器或使用高速缓存配置1600的其他设备中的某些部件。在图11中的公开内容并不意在进行限制。此外,应该注意,SoC 102仅是系统的类型的示例,1600可以为该系统提供高速缓存配置。处理器1604a-n可以是例如多核处理器,诸如多核处理器204、304或504。可选地,多个处理器1604a-n可以是单核处理器或单核处理器和多核处理器的某种组合。

[0486] 每个加速器1601a-n还可以包括与其各自的处理器1604a-n相关联的高速缓存1610a-n。尽管在图11中每个高速缓存1610a-n被表示为单个元件,但是实际上,每个高速缓存1610a-n可以包括用于处理器1604a-n中的每个单独核心的多个单独的高速缓存(例如,像图2的私有或“本地”高速缓存206a-d)。可选地,每个高速缓存1610a-n可以是在多个核心当中共享的单个高速缓存,诸如核心共享高速缓存514(图5A)。每个加速器1601a-n可以访问外部存储器1650,诸如主存储器或更高级的高速缓存。通常,每个加速器1601a-n将来自外部存储器1650的变量复制到高速缓存1610a-n,用于由处理器1604a-n相对快速地访问。

[0487] 在任一情况下,高速缓存1610a-n都可以保持与外部存储器中的地址对应的数据的本地副本。例如,本地高速缓存可以是具有相对有限的容量(例如4-600Mb)的L1或L2高速缓存,但是可以比外部存储器在少得多的时钟周期中做出响应。这样,相关联的处理器(例如1604a-n)可以比通过向外部存储器发送请求快得多地访问数据。如果处理器1604a-n请求当前未包含在高速缓存1610a-n中的数据,则它可以从外部存储器1650取出数据并将该数据保存到高速缓存1610a-n,使得下一个数据访问更快。由于高速缓存1610a-n可具有比外部存储器1650更有限的容量,因此高速缓存1610a-n可逐出一些数据以为新取出的数据腾出空间。如果要被逐出的数据被核心修改(变得“脏”),则修改的数据在逐出之前需要被写到外部存储器1650,使得修改高速缓存数据所做的工作不会丢失。这个过程将在下面更加详细地被描述。

[0488] 示例性高速缓存结构

[0489] 图12示出根据本文公开的某些例子的示例性高速缓存结构1700(或“高速缓存”)。示例性高速缓存结构1700可以表示例如图11中的高速缓存1610a-n中的任何一个的结构。在示例性高速缓存结构1700中,高速缓存的数据可以被保持在高速缓存行1701a-n中。对应于高速缓存的数据的元数据通常也被保持在示例性高速缓存结构1700中。这个元数据的性质可以变化,但是它通常包括地址数据(通常也被称为“地址位”或“标记”)1703a-1703n。这些地址数据1703a-1703n可以在外部或主存储器中指定它们对应的高速缓存行1701a-n的地址。元数据的其他例子可以包括指定数据相对于外部存储器或主存储器的读/写条件的有效位1703a-n。高速缓存结构1700还可以包括指定是否存在本地写入的脏位1707a-n。特别是,脏位1707a-n的使用将在下面更详细地被探索。

[0490] 高速缓存1700可以在高速缓存行粒度处管理数据。例如,具有64字节行的高速缓存1700可以在64字节的部分中取出数据。这即使在处理器请求少于64个字节的数据(例如,当核心只请求单个字节时)时也可发生。在高速缓存行粒度处管理数据可以是有效的,因为处理器可能访问接近它已经访问的数据的数据。例如,如果处理器已经请求了在地址X处的字节,那么与在X+n处的字节相比,更可能随后访问在外部存储器中的地址X+1和X+2处的字

节,其中n>>2。

[0491] 立即取出整个高速缓存行允许处理器在本地访问在所请求的数据的64字节内的邻近数据。如上面讨论的,由于数据被存储在更快的本地高速缓存1700中,这意味着处理器(例如,处理器1604a-n)然后更快地访问接近所请求的数据的任何数据。当外部存储器是动态随机存取存储器(DRAM)时,可以实现类似的效果,该动态随机存取存储器(DRAM)仅支持对在多个字节的“数据串(burst)”中的数据的访问。高速缓存行1701a-n大小可被调整为等于或超过DRAM数据串,提供对邻近数据的本地访问。另外,由于高速缓存1700通常存储与每个高速缓存行1701a-n相关联的元数据(例如,地址数据1703a-n、有效性数据1705a-n和/或指示行是否是脏的数据1607a-n),因此大小越大,元数据使用就越有效。例如,在1字节粒度而不是64字节粒度处操作的高速缓存1700必须为每个字节存储外部存储器地址1603a-n。由于高速缓存小于外部存储器,因此高速缓存存储器中的位置与外部存储器中的位置之间可能不存在一对一映射。类似的考虑因素适用于有效性数据1705a-n和脏位数据1707a-n。

[0492] 高速缓存一致性:综述

[0493] “高速缓存一致性”除了别的以外还涉及在实际可行的程度上维持在高速缓存(例如,图11的高速缓存1610a-n)和外部存储器之间的数据一致性的问题。当两个处理器使用来自外部存储器的相同数据的已更改或不同地更新的副本时,可能出现高速缓存一致性后果或问题。

[0494] 高速缓存一致性问题可以从它们出现的上下文方面来描述。当两个不同的处理器访问和/或修改相同的变量时,高速缓存数据的“真共享”产生。真共享可能是故意的或偶然的。相反,当两个不同的处理器访问和/或修改驻留在或存储在相同高速缓存行1601a-n中的两个不同变量时,出现“假共享”。下面将更详细地描述这些实例中的每一个。

[0495] 真共享

[0496] 图13A显示参与真共享的两个不同处理器的高速缓存行。高速缓存行1810是对于处理器1801(未示出)的本地高速缓存,而高速缓存行1820是对于处理器1802(未示出)的本地高速缓存。高速缓存行1810和1820分别在分段1815和1825中存储变量“X”的副本。保存在815和825中的X的副本最初是从外部存储器(诸如例如图11中的外部存储器1650)获得的。分段1815和1825可以是各种大小,但通常是多个字节的数据。

[0497] 当在处理器1801上运行的程序应该写到X并且在处理器1802上运行的程序需要X的这个被修改/写入的值而不是存储在它自己的高速缓存820或外部存储器中的X的值时,有意的真共享发生。如果处理器1802需要在处理器1801已经将那个修改的值写到外部或主存储器之前访问X的修改的副本,则可能出现潜在的问题。在这种情况下,X的修改值仅存在于不能由处理器1802直接访问的高速缓存行1810中(即在处理器1801的高速缓存中)的分段1815中。在处理器802自己的高速缓存中的X的值(即,在高速缓存1820的分段1825中的X的值)和在外部存储器中的值将相对于由处理器1801进行的最新修改不是最新的。相反,存储X的那些最新修改的唯一存储器可能是在高速缓存1810中的分段1815。因此,为了在处理器1801和1802之间共享X的修改值,需要实现用于传送X的修改值的机制和/或协议。

[0498] “意外共享”是另一个潜在的真共享问题。在意外共享中,在处理器1802上运行的程序可能不知道它需要或者应该使用已经由在处理器801上运行的程序所修改的X的版本。于是,在处理器1802上运行的程序可以访问X的过时副本,并且随后使用该过时值来进行计

算或确定。基于使用X的过时值的错误然后可被传播到与在处理器1802上运行的程序交互的其他程序和例程。

[0499] 在意外和有意真共享之间的差异在程序员的意图中。在有意真共享中，用于在两个处理器(例如，处理器1801和1802)上运行的过程的代码以处理器1801和1802通过变量(在上面的例子中的X)进行通信的需要在共享之前是已知的的方式来写入。在意外真共享中，程序员没有意识到同时或并行地运行的两个程序将试图访问相同的数据。在意外真共享的情况下，同时访问作为程序错误的结果而发生。

[0500] 在有意的和意外的真共享的上下文中，记录哪些数据已被写入以及哪些数据未被写入的一种方式是通过使用“脏”位1835和1845(图13A和13B)。注意，对“位”的提及并不意为进行限制。脏“位”可以在大小上比一位更大(在这种情况下，“脏位”可以更适当地被称为“指示符”)，并且可以储存除了本文描述的之外的信息。脏位通常存储两个值中的一个，第一个值指示数据在某个时间段(例如，自从脏位上次被重置以来的时间段)期间或在特定条件下被写入，以及另一个值指示相同数据在同一时间段中还没有被写入。在图13A和13B中所示的例子中，脏位1835和1845实际上是“脏行位”，意味着“变量”(其状态由脏位1835和1845指定)实际上分别是整个高速缓存行1810和1820。换言之，写到高速缓存行1810或1820的任何分段而不仅仅是1815或1825将设置相应的脏位1835或1845以指示写入发生。可选地，可以分配脏位以指定存储在分段1815和1825处的单独变量X的状态。在图13A和13B所示的例子中，使用脏位1835和1845，使得当它们被设置为“1”时，写入发生在相应的高速缓存行中。当它们被设置为“0”时，自从脏位1835和1845的上次重置以来没有发生写入。

[0501] 如图13A中的例子所示，在分段1815处将“新”值写到变量X将脏位1835设置为“1”。该设置指示对高速缓存行1810的写入已发生。相反，由于自从上次重置以来没有发生对高速缓存行1820的任何分段的写入(包括存储在分段1825中的变量X)，因此脏位1845保持被设置为“0”。在这种情况下，脏位1835和1845指示高速缓存行1810和1820中的数据不同步。但是，它们不指示哪个(或哪些)分段(即，代表变量X的分段1815和1825)不同步。

[0502] 假共享

[0503] 图13B示出当同一外部存储器高速缓存行的副本中的两个不同变量由处理器1801和1802写入时可能在例如图13A中讨论的配置中发生“假共享”的情况。具体地，当处理器1801修改变量X并且处理器1802修改存储在外部或主存储器中的相同高速缓存行中的另一变量Y时，“假共享”发生。这可能导致以下后果。

[0504] 如图13B所示，保持X和Y的本地高速缓存行1810和1820都是“脏的”，如分别由位1835和1845指示的。但是，它们由于不同的原因而是脏的。高速缓存行1810是脏的，因为它包含在1815处的X的新的或更新的值。另一方面，高速缓存行1820是脏的，因为它包含在1827处的Y的新的或更新的值。因此，高速缓存行1810包含在1817处的Y的旧的或过时的副本，并且高速缓存行1820包含在1825处的X的旧的或过时的副本。这意味着处理器1801和1802最终将高速缓存行1810和1820的内容写回到外部存储器。由于高速缓存行1810和1820都是外部存储器中的相同高速缓存行的本地副本，因此它们需要被写入到外部存储器中的那个相同高速缓存行。由于写入不能同时完成，因此其中一个写入必须覆写另一个写入。这导致丢失的工作和潜在的错误。如果处理器1801首先写回高速缓存行1810的其值，则处理器1802将最终用在1825中存储的旧X来覆写在1815中存储的新(或更新的)X。相反，如果处

理器1802首先将它的数据写回，则处理器1801将用存储在1817中的旧Y来覆写存储在1827中的新Y。

[0505] 这个覆写和因而产生的丢失的数据被称为“假共享”，因为变量X或Y均没有在处理器1801和1802之间被“真共享”。替代地，X只被处理器1801而不是1802访问，并且Y只被处理器1802而不是1801访问。这种“假共享”可能导致与在上述真共享的上下文中所述的问题类似的问题。

[0506] 解决在真或假共享中的数据一致性问题

[0507] 正如在上面的背景章节所讨论的，硬件和软件系统都可用来解决高速缓存一致性问题。这些系统(例如实现“全高速缓存一致性”的那些系统)常常在专用于在过程之间共享数据的硬件、软件或两者方面引入额外的复杂性。在本文描述的实施例中，用于解决假共享的附加硬件可以连同用于解决真共享的软件方案一起实现。该软件方案可以通过减少或消除并行过程的数据共享来降低与所共享的数据相关的问题的可能性。正因如此，本文描述的硬件和软件可能潜在地避免或减少即使在系统上运行时的在高速缓存一致性中的许多问题，而没有为了在“全高速缓存一致性”下完成数据共享而需要引入的额外的硬件和其他复杂性。

[0508] 用于解决假共享的硬件

[0509] 图14示出在本公开的上下文中解决假共享的一种方式。在图14中，高速缓存行1910对于处理器1901是本地的，而高速缓存行1920对于处理器1902是本地的。类似地，如在图13A和13B的上下文中，高速缓存行1910和1920都存储包含两个变量X和Y的相同外部存储器高速缓存行(未示出)的副本。注意，虽然图14仅示出对于每个高速缓存的单个高速缓存行1910/1920，但每个高速缓存可以包括多个高速缓存行。如图14所示，高速缓存行1910和1920都分别由位1935和1945标记为“脏”，因为两者都包含需要被写回到外部存储器的新修改或更新的数据。这是因为本地高速缓存1910和1920分别包括对于X和Y的数据的不同更新。具体地，高速缓存1910包括在分段1915处的对于X的新的或更新的值，而高速缓存1920包括在1927处的Y的新的或更新的值。高速缓存1910还包含在817处的Y的过时或未修改的副本，并且高速缓存1920包含在1925处的X的过时或未修改的副本。

[0510] 如图14所示，高速缓存1910和1920还各自包含分别在1950和1951处的位的附加阵列。共同地，这些被称为与它们各自的高速缓存行的各个字节相关联的“脏位”。在图14所示的例子中，对于在高速缓存1910/1920中的每个字节，在1950/1951中有一个脏位。这个特定的布置(每缓存字节有一个脏位)有助于跟踪在高速缓存中的每个位和因而每个变量的写入状态。然而，应该理解，其他布置是可能的，并且在高速缓存1910/1920中可能存在没有相关联的脏位的字节。

[0511] 每高速缓存行字节的脏位可用于监控高速缓存行字节的写入状态，如下所述。例如，当高速缓存行数据首先从外部存储器被取出到高速缓存行1910/1920中时，脏位1950/1951可以最初都被设置为“0”。随后，当在本地高速缓存行1910/1920中的一个字节被修改时，它的对应的脏位可以被设置为“1”。这个布置在图15中示出。

[0512] 图14中的例子示出对几个字节实现的这个布置。例如，对应于脏位1950a和1951a的字节还没有被处理器1910或1920修改。因此，它们仍然被设置为“0”。相反，对应于在高速缓存行1910中的变量X的四个字节中的每一个已由处理器1910更新或修改。因此，对应于这

些字节的四个脏位1950b-c全部被设置为“1”。由于处理器1920尚未修改或更新变量X,因此在高速缓存行1920中的对应脏位1951b-c仍然被设置为“0”。由于与在高速缓存行1920中的变量Y对应的四个字节中的每一个已经被处理器1920修改,因此与这些字节相对应的四个脏位1951f-i全部被设置为“1”。由于处理器1901没有修改Y,因此相应的脏位1950f-i仍然被设置为“0”。

[0513] 在逻辑上在一起的脏位可能使“脏行”位1935/1945变得冗余。这是因为当且仅当至少一个字节是脏的时,脏行位1935/1945指示高速缓存行1910/1920是脏的。尽管存在这种冗余,仍然维持脏行位1935/1945可能仍然是合乎需要的。由于它们是单个位,因此脏行位1935/1945通常可以比所有的脏位1950/1951一起更快和更容易地被读取。此外,检查脏行位1935/1945以确定在高速缓存1910/1920中的值是否已被更新不需要执行任何逻辑操作。因此,维持脏行位1935/1945可以允许在高速缓存1910/1920中的数据状态的有效评估,而不管任何冗余。脏行位1935/1945可以被维持(即明确被写入)或从脏位1950/1951计算。

[0514] 如图14所示,每个高速缓存行1910/1920还可以包括“有效”(或“有效性”)位1937/1947。这些有效位1937/1947对应于在图12的示例性高速缓存结构中所示的有效位1705a-n。

[0515] 如在图12的上下文中所讨论的,可以使用有效位来指定是否应使用来自外部存储器的数据来刷新相关联的高速缓存行。例如,将有效位1937设置为“0”指示在高速缓存行1910中的数据是无效的且需要被刷新。因此,在下一次从外部存储器读取数据期间,处理器1901将用存储在外部存储器的相应高速缓存行中的当前数据覆写高速缓存行1910。将有效位1937设置为“1”指示在高速缓存行1910中的数据是有效的(即,不需要用来自外部存储器的数据覆写高速缓存行1910)。以这种方式,每当启动或发起新过程时,都可以使用有效位1937来刷新相关的高速缓存行。注意,如在本文所讨论的,“启动”过程通常可以意味着远程地或在本地开始该过程的执行。一旦过程被启动,它就可通过例如将值复制到本地存储器来发起。例如,过程发起可以自动将所有有效位(例如1937和1947)设置为等于“0”,使得所有的高速缓存行将尽可能快地用存储在外部存储器中的值更新。这个过程被称为使高速缓存数据“无效”。在过程将数据写到外部存储器之后,过程也可能使高速缓存无效。如下面更详细讨论的,当高速缓存行1910/1920已经被写到外部存储器或主存储器时,可以在写回之后执行将有效位1937和1947设置为无效状态(例如,设置为“0”)。例如,在过程结束时当写回时使高速缓存1910/1920无效可以允许在下一个过程开始之前用来自外部存储器(例如,经验证的)的新数据来更新高速缓存1910/1920。也可以通过在高速缓存1910/1920的发起时以及在又一些其他情况下将有效位1937和1947设置为无效状态来使高速缓存1910/1920有效。

[0516] 处理器1901/1902可以使用脏位1950/1951来确定何时以及如何写到外部存储器以减少或消除假共享的问题。例如,处理器1901/1902可以生成单个存储器请求,以将它们的整个高速缓存行1950/1951写到外部存储器。当写入高速缓存行时,处理器1901/1902可以使用脏位1950/1951来只写那些脏的字节(例如,在图14所示的例子中具有值“1”)。在例子中,处理器1901将只写入存储在1915b-c中的X的新的或修改的值,并且将不改变在外部存储器的高速缓存行中的任何其他值。这意味着在其中处理器1902先前将Y的更新值(即,存储在1927中的更新值)写到外部存储器的情况下,处理器1901将不覆写Y的那个更新值。

如果写入以相反的顺序发生(即,首先处理器1901使用脏位1950b-c来写如X的更新值,后面是处理器1902使用脏位1951f-i来写如Y的更新值),则第二次写入将不覆写X的更新值。这是因为处理器1902将使用脏位1951f-i来仅将1927中的Y的新值写到外部存储器。它将不改变在外部存储器中的X的值,因为对应于在高速缓存行1920中的X的值的脏位(即,脏位1951b-c)被设置为“0”。因此,不管写入发生的顺序如何,都可以避免最新更新的数据的覆写。

[0517] 如上所述,使用脏位1950/1951来将高速缓存行1910/1920写到外部存储器可以使用标准总线协议来完成。例如,如上所述,可以做出对整个高速缓存行写入的单个存储器请求。在这种情况下,写入可以利用I/O一致性协议,诸如在“Maintaining I/O Data Coherence in Embedded Multicore Systems”(T.B.Berg,Micro,IEEE 29(3):第10-19页(2009年))中描述的软件I/O一致性协议。这种标准的高速缓存一致性协议提供了由高速缓存1910/1920用来请求和放弃共享数据的所有权的硬件机制。更具体地,“I/O一致性”涉及一种协议,其中第一处理器1901(如果1901是“I/O一致设备”)可以从第二处理器(例如,处理器1902,其中1902是提供对I/O一致设备的支持的CPU)读取数据。在I/O一致性协议中,处理器1901被保证从处理器1902获得共享数据的最新副本。处理器1901还可以执行写入,使得处理器1902可以检索共享数据的最新副本。如下面更详细讨论的,那些协议可以与本文描述的解决方案或实施例中的任一个结合来使用。

[0518] 根据本发明的实施例,每个核心被配置为(在一个或更多个周期期间)将待写到高速缓存行的内容以及相关联的有效位输出到外部存储器1650。外部存储器1650能够处理有效位并仅由更新的高速缓存行字节来更新高速缓存的行。例如,外部存储器可以被配置为字节使能的,并且能够只用所选择的字节更新高速缓存行。

[0519] 参卡图13B的例子,外部存储器1650可以在一个或更多个周期期间(同时和/或连续地)从处理器1801接收高速缓存行内容1810(包括修改的变量X 1815),并且从处理器1802接收高速缓存行内容1820(包括修改的变量Y 1817),二者都涉及相同的行,并且可以仅使用字节启用协议来相应地修改变量Y和X。

[0520] 在另一示例性方法中,可以每“脏字节”(即,对应于脏位的高速缓存数据的每字节)或每组相邻的脏字节生成存储器请求。这个方法需要对每个脏字节的单独写入。

[0521] 可选地,可以使用“字节使能位”来控制在高速缓存行1950/1951中的哪些数据将在单个写入中被写到外部存储器。将字节使能位设置为等于“1”指示与这些位对应的数据将被写到外部存储器。将字节使能位设置为“0”指示对应于这些位的数据将不被写入。因此,可以根据脏位1950/1951来设置字节使能位。更具体地,当相应的脏位1950/1951指示数据已被处理器1901/1902修改时,对应于单个存储器高速缓存行写入的字节使能位可被设置为“1”。对于在写入中的所有其他数据的字节使能位可以被设置为“0”。以这种方式设置字节使能位导致只有修改或更新的数据被写到外部存储器。

[0522] 这种类型的写入由上面讨论的I/O一致性协议支持。更具体地,通常通过在总线请求中使用字节使能位,I/O一致性支持少于完整高速缓存行的写入。换言之,如果处理器1901是“I/O一致设备”,则它如上所述自动使用字节使能位。

[0523] 单次写入方法至少因为它需要较少的写入操作来将多个更新的变量写到外部存储器而是相对有效的。例如,在其中高速缓存行1910具有两个更新的变量(每个变量在大小

上是1字节)(例如,变量A和B)的情况下,使用字节使能位将允许使用单次写入操作来将两个变量写到外部存储器。当外部存储器在数据串中写入(例如,DRAM)时,这个优点特别重要。在这种情况下,如果高速缓存行1910的大小和DRAM数据串的大小相等(例如,64字节),则写入A和B都将需要仅发出单个64字节数据串,其中字节使能位对应于变量,且B被设置为“1”。这个写入比可选方案(即,使用数据串来单独地写入A和B的每个脏字节)有效两倍。这是因为单独地写入A和B需要两个单字节数据串。但是,即使片上总线协议可支持单字节数据串,DRAM存储器也不支持。于是,2个单字节数据串将需要来自DRAM的两个数据串。对于具有64字节数据串大小的DRAM,这将需要两个64字节数据串来将A和B写到外部存储器。

[0524] 用于解决真共享的软件

[0525] 图11示出了根据本公开的例子的在过程之间的布置的图示。图11中的布置可以解决上面所述的意外和有意的真共享问题。如下面将更详细地描述的,在图11中所表示的系统的一个特征在于,所有过程都被编码,使得两个同时运行的过程不访问可写入(或“可变的”)变量。

[0526] 图16A和图16B示出由两个处理器执行的两个过程的非限制性例子,而这两个过程被设计(编程)为不同时访问同一存储器行的同一分段(例如相同字节)(甚至同一存储器行的内容的不同版本)。可以防止一个过程在任何预定的时间段期间访问同一存储器行的同一分段。该时间段可能很长(例如,数千乃至数万个周期)。应注意,这两个过程可能彼此并不相关,且一个过程不一定触发或控制另一个过程。

[0527] 真碰撞的预防和假碰撞的管理的组合提供了鲁棒的一致存储器管理方案。

[0528] 在图16A和图16B所示的例子中,两个处理器1901和1902分别运行两个不同的过程A和B。图16B示出在时间线的过程中过程A和B的持续时间。注意,尽管在图16A和图16B的时间线中仅示出两个过程A和B,但这并不意为进行限制。实际上,可以依次启动比在图16A和图16B中所示的多得多的过程。时间线在“0”处开始并显示六个阶段(11105-11130)。与处理器1901/1902相关联的高速缓存1910/1920的结构在图14中示出。

[0529] 如在图16B中指示的,过程A和B在如下意义上同时运行:过程A从整个时间线(0到11130)的持续时间运行并且过程B仅在时间线的一部分(11110到11125)期间运行。当过程A在处理器901上在0处被启动时,它的第一任务可以由“[正在]写入输入”(即,将它需要的变量从外部存储器(未示出)复制到处理器901的本地高速缓存1910)发起。例子可以包括在图14中表示的变量X和Y。过程A可以将对于高速缓存1910中的每行的所有有效位1937设置为等于“0”,使得从外部存储器的下一次读取将覆写高速缓存行1910。以这种方式设置有效位1937使得每个高速缓存行1910被假定为无效的可能是有利的。然后,新的输入可以从外部存储器中自由地读取,保证了高速缓存1910将不包含当新过程执行时由先前的过程写入的任何过时值。在将输入读取到高速缓存1910中并且过程A执行之后,过程A可以执行修改在高速缓存行1910中的变量(例如X和/或Y)的操作。当这发生时,过程A可以将变量的修改值写到本地高速缓存1910,将对应于新写入的变量的脏位1950设置为等于“1”。

[0530] 11105-11110启动过程B

[0531] 过程A可以在处理器1901上运行,使得1901是“消费者”处理器,而处理器1902是“生产者”处理器。在这种情况下,过程A可以在处理器1902上启动过程B,如在图16A中在11110处所示。过程B可以是例如花费相对长的时间来完成的计算过程。

[0532] 在过程B的启动(11110)处,在高速缓存1920中的有效位1947可能是无效的,例如等于“0”(即,指示在高速缓存1920中的数据需要用来自外部存储器的值来刷新)。以这种方式,可以从外部存储器自由地读取新的输入,保证了高速缓存1920在新过程执行时将不包含由先前的过程写入的任何过时值。例如,过程B可以读取变量X和Y(图8)。

[0533] 11110-11115过程B在没有共享变量的情况下执行

[0534] 在这个时段期间,过程B执行。当它这么做时,它可以将附加变量读入到它的高速缓存1920中和/或修改在高速缓存1920中的变量。例如,它可以修改变量X和Y。当这发生时,它将修改的变量存储在它的本地高速缓存1920中,将相应的脏位1951设置为等于“1”以指示修改的数据,如图14所示。当数据被修改时,过程B可进一步将有效位1947设置为“0”,指示高速缓存行1920不再是有效的。

[0535] 在过程B执行的时段期间,如图16A所示,过程A仍然可以执行,只要它的执行不需要可以被过程B访问的变量。在该例子中,如果过程B访问可变变量X和Y,则过程A将不被允许访问X和Y,直到过程B终止于11125处为止。而且,处理器1901还可以运行其他过程,假定那些过程也不访问被处理器B访问的变量。也就是说,在过程B正在执行的时候,在处理器1901上运行的其他过程将不被允许访问变量X和Y。在这个意义上,当过程B在处理器1902上执行时,处理器1901仍然自由地运行“独立工作”(例如,不依赖于可能被过程B修改的变量的工作)。将处理器1901限制为“独立”工作防止可变数据的共享。如果与在处理器1902上运行的过程B同时地在处理器1901上运行的过程不访问与过程B相同的可变数据或变量,则如上面在图8的上下文中所讨论的,可能不存在“真共享”问题。简而言之,当过程B正在执行时,没有过程可与过程B共享任何可变数据。

[0536] 11115-11125过程B通过共享所有可变变量来终止

[0537] 过程B的计算在11115处完成。此时,过程B可以发起“写回”或者用它对在处理器1902的本地高速缓存1920中的变量所做的任何改变来更新外部存储器的过程。写回可以如上面在图14的上下文中所描述的那样执行(例如,通过使用脏位1951来设置字节使能位,以便在一次写入中将所有修改的数据写到外部存储器)。写回通常在所有高速缓存行上反复。如在图16A中在115-11120处所示,该过程也可以将对于每个高速缓存行的所有的有效位1947设置为“0”(即,使每个高速缓存行无效)。无效和写回可以同时或单独地完成。

[0538] 如图16A所示,从11120到11125,过程B可以用信号表示它已经完成执行。过程B的发信号完成在写回之后允许其他过程访问由过程B使用的任何可变变量。因为过程B已经执行了写回,所以由过程B使用的变量的已更新值现在被存储在外部存储器中。在该示例情况下,过程B将变量X和Y写回到外部存储器,并且在接收到信号之后,过程A(或在处理器1901上运行的任何其他过程)将于是能够访问处理器B刚刚写到外部存储器的X和Y的新修改的副本。

[0539] 11130过程A读取输出

[0540] 在过程B已经终止并且发信号完成(11120-11125)之后,过程A(以及在系统上运行的任何其他过程)可以访问过程B刚刚写到外部存储器的变量。如图16A所示,过程A自由地从外部存储器读取由过程B生成的输出。那些输出可以包括被写到过程B的任何变量(例如,在例子中的变量X和Y)。

[0541] 模型与I/O一致性一致

[0542] 在上面且在图11的上下文中描述的真数据共享模型由I/O一致性协议支持。更具体地,I/O一致性支持仅在过程“边界”(即,过程启动1105-11110和终止11115-11125)处的共享数据。事实上,I/O一致性适应这种类型的数据共享,因为它通常为处理器1901(“I/O一致设备”)没有高速缓存的情况而设计。因为该协议假定对于处理器1901没有高速缓存,所以处理器1902不知道处理器1901的高速缓存的状态。因此,处理器1902向在处理器1901上运行的过程A提供输入的最新副本,并且在终止时将它的数据写出到外部存储器,好像它是无高速缓存的I/O设备(11115-11120)时所做的那样。

[0543] 结果的增加的一致性

[0544] 如上面讨论的,用于解决真共享的软件(图11)基本上禁止或减少在同时执行的过程当中的数据的共享。因此在执行期间数据共享被减少或消除,所以过程写到外部存储器中的共享变量的顺序更加一致。事实上,在严密或严格地应用这种限制性共享的情况下,写到外部存储器的过程的顺序可以被认为是固定的或不变的,导致更一致或甚至确定的结果。

[0545] 这通过例子最好地被示出。在图11示出的例子中,过程B将在11115处将它修改的任何变量写到/更新到外部存储器。也就是说,通过故意强加的约束,在过程A将更新它与过程B共享的任何数据或变量之前。换句话说,因为过程A不能访问由过程B正在使用的任何可变数据直到过程B通过将它的数据写到外部存储器(11115)和发信号完成(11125)来终止执行为止,过程B将总是在过程A写入数据之前写入任何相互共享的数据。这意味着在由图11所示的例子表示的某些实施例中,任何相互共享的数据(例如,变量X和Y)将总是首先由过程B且其次由过程A更新。只要过程A和B本身都是确定性的(即不产生随机数据),则由图11中的系统产生的结果也应该是确定性的。

[0546] 相反,通过并行过程来便于可变数据的共享的系统将导致共享数据/变量的非确定性的和有时不可预测的更新。这是因为这样的系统不能修正在其中过程更新外部存储器中的变量的顺序。一个例子是采用数据锁来防止一致性问题同时允许并行过程共享数据的系统。数据锁是在过程可以改变被锁保护的数据之前必须由过程检查的变量。当一次只有一个过程可检查锁时,一次只有一个过程可修改数据。因此,实现锁可以通过防止数据的同时访问/更新来减少一致性问题。然而,即使它们可能不会同时访问锁,两个过程(例如,图11中的过程A和B)在并行执行期间仍然可以访问锁和数据。在这种情况下,两个过程写到外部存储器的顺序可以由多种因素(包括输入到系统的数据的细节)决定。甚至输入数据的定序(例如,按照不同标准对输入数据进行排序)也可能潜在地改变两个过程写到外部存储器的顺序。由于某些结果可能取决于过程更新外部存储器中的数据的顺序,因此该系统可能具有结果或后果的相当大的可变性。

[0547] 图16C示出了根据本发明的实施例的数据锁8100a-j和数据锁请求器8200a-e。

[0548] 假定锁定数据锁的请求可以由属于图11的处理器1604a-n或属于图6A的处理器核心604a-d和串行处理器616的数据锁请求器8200a-e生成。假定有五个处理器。可提供任意数量的处理器/请求器。

[0549] 数据锁8100a-j可以属于ITC 8000的数据锁部分8020,但是可以位于其他地方。

[0550] 数据锁的数量可以是256,但这不一定是如此。可以提供在数据锁的数量与数据锁可以服务于的功能之间的折衷。例如,当使用数据锁来保护直方图位的内容时,每1024个直

条可以有256个数据锁,尽管可以提供在数据锁和直条的数量之间的任何关系。

[0551] 锁定数据锁的请求包括应该被锁定的数据锁的标识符和锁定命令。在图16C中,标识符被表示为C0.ReqLockID-C4.ReqLockID,并且锁操作码被表示为C0.lock-C4.lock。

[0552] 将数据锁解锁的请求包括应该被解锁的数据锁的标识符和解锁操作码。在图16C中,标识符被表示为C0.ReqLockID-C4.ReqLockID,而解锁操作码表示为C0.release-C4.release。

[0553] 数据锁8100a包括比较器8102(用于比较数据锁8100a的标识符-“Self-ID”与C0.ReqLockID-C4.ReqLockID中的每一个)和逻辑门(诸如,与门8104a-e和与门8105a-e),其形成不传播与数据锁8100a无关的数据锁请求(锁定或释放)的请求过滤器。

[0554] 用于将数据锁8100a解锁的请求被发送到与门8105a-e以与比较器8102的输出进行逻辑与,并将与门8105a-e的输出馈送到或门8010以生成指示对释放数据锁810a的一个或更多个请求的接受的释放(RELEASE)信号8108。

[0555] 与门8104a-e的输出信号连同掩码位(掩码8418的掩码[0]-掩码[5])一起被发送到与门8106a-e,其只将被掩码8418掩码的请求发送到第一优先级编码器8110。如在下面将被更详细解释的,掩码8418被超时地改变以提供动态仲裁方案。掩码可以允许被掩码的请求的处理以锁定数据锁8100a。

[0556] 第一优先级编码器8110仅接收锁定数据锁8100a的被掩码的请求,选择多达单个被掩码的请求,并且输出指示该选择的被掩码的授权向量8410(包括位授权_掩码的[0]-授权_掩码的[4])。假设一旦进行选择,就只设置掩码的授权向量8410的一个位。

[0557] 或门8114对掩码的授权向量8410的位应用了或操作以确定第一优先级编码器是否选择了单个掩码的请求。

[0558] 在第一优先级编码器8110仅在掩码的请求之间进行仲裁时,第二优先级编码器8112在所有请求之间仲裁,而不管掩码如何。

[0559] 第一和第二优先级编码器8110和8112可以应用任何仲裁方案。例如,第一和第二优先级编码器可以选择具有最低索引的请求。任何其他仲裁方案可能适用。第一和第二优先级编码器8110和8112可以应用相同的仲裁方案或者可以应用不同的仲裁方案。

[0560] 与门8104a-e的输出信号也被发送到第二优先级编码器8112。第二优先级编码器8112接收锁定数据锁8100a的请求,选择多达单个请求(不管掩码如何),并输出指示选择的掩码无知授权向量8412(包括位授权_无掩码[0]-授权_无掩码[4])。假设一旦进行选择,就只设置掩码无知授权向量的一位。

[0561] 被掩码的授权向量8410和掩码无知授予向量8412被馈送到第一复用器8118,并且这两个向量之间的选择由从或门8114输出的控制位确定。如果第一优先级编码器8110选择单个掩码的请求,则从第一复用器8118输出掩码的授权向量8410。如果第一优先级编码器8110没有选择任何掩码的请求,则从第一复用器输出掩码无知授权向量8412。

[0562] 数据锁8100a的值被存储在D型触发器8130中,并被表示为状态(STATE)8422。

[0563] 在状态8422和反向的释放8108之间应用与操作(门8116)以提供指示数据锁被锁定并且不存在将数据锁解锁的请求的“无人获胜”信号8414。

[0564] “无人获胜”信号8414被馈送到第二复用器8120,使得当数据锁被锁定并且不存在将数据锁解锁的请求时,第二复用器8120输出重置授权向量(例如,00000)。否则,第二复用

器8120将输出从第一复用器8118输出的信号,以提供指示锁定数据锁1800a的所选择的请求的授权向量。

[0565] 授权向量8420被输出到可以(基于授权向量的内容)通知哪个处理器成功锁定数据锁1800a的处理器。

[0566] 授权向量8420被发送到输出了输出信号8125的或门8124。输出信号8125和“无人获胜”信号8414被馈送到或门8128,其仅当输出信号8125和“无人获胜”信号8414都被重置时才重置触发器8130。

[0567] 授权向量8420也被馈送到掩码编码器8121,掩码编码器8121改变掩码8414并将更新的掩码馈送到缓冲器8122。缓冲器8122可以是FIFO。

[0568] 掩码编码器可以应用任何掩码改变方案。例如,掩码编码器可以生成识别所选择的请求的掩码,搁置掩码的对应的选择位和掩码的所有较低有效位(如果存在的话),并将在掩码中的所有较高有效位置位。

[0569] 这种硬件结构紧凑、快速且易于访问,并且允许每周期仅仅单个处理器来锁定单个数据锁。

[0570] 图16D示出根据本发明的实施例的数据锁更新请求的状态机8210。

[0571] 状态机由数据锁请求器8200a-e中的每一个执行。

[0572] 状态机8210由空闲(IDLE)状态8212启动。

[0573] 空闲8212之后可以是确定发送锁定数据锁请求(cmd_lock)的状态8218。

[0574] 空闲8212之后也可以是(如果不是确定发送锁定数据锁请求)检查解锁标准是否被满足,(a)数据锁请求器是否生成了数据锁解锁命令(cmd_unlock),以及(b)对于其数据锁由数据锁请求器锁定的线程(或过程)是否结束。可以以各种方式检测线程或过程的完成,例如检测为线程或过程分配的处理器的写入队列是否为空(store_but_empty[TID])。

[0575] 退出空闲还涉及向处理器发送停止从指令存储器取出指令的请求(set itu_WaitFor Lock),防止执行取决于数据锁的解锁的线程。

[0576] 如果满足解锁标准,则跳到状态8216,否则跳到等待直到满足解锁标准为止的状态8214。

[0577] 状态8216包括向ITU 8200发送将具有应该被解锁的数据锁的标识符的数据锁解锁的请求(例如,C0.ReqLockID和C0.release)。

[0578] 状态8216之后是状态空闲8212。从状态8216退出包括向处理器发送从指令存储器中重新开始取出指令的请求(set itu_WaitFor Lock),允许取决于数据锁的解锁的线程的执行。

[0579] 状态8218之后是检查(对锁定数据锁请求的)授权的步骤8220。如果请求已经被授权,则状态8220后面是阶段空闲8218。否则,状态8220之后是等待下一个时隙(被分配给被阻止执行直到锁定数据锁被授权为止的线程)的状态8222,并且步骤8222之后是状态8218(或状态8220)。

[0580] 状态机集成数据锁管理以及桶式调度,并且提供用于线程管理的紧凑、快速和低基板面(real estate)解决方案。

[0581] 增加的测试可靠性

[0582] 在图11的系统中的结果的增加的一致性可以便于调试。在这个系统中,如上面讨

论的,对外部存储器的更新的顺序由没有并发过程可访问可变数据的约束来固定。这意味着至少从它将数据更新到外部存储器的顺序来说遵守这个约束的任何代码都应该是确定性的。因此,调试这样的程序的一种方法是基于例如输入数据的定序或排序来测试不一致或不确定的结果。如果任何这样的输入产生不一致的结果,则1)系统中的一些过程违反约束反对访问可变数据的并发过程,2)一些过程本身不是确定性的,或者3)1)和2)都适用。

[0583] 而且,通过监控过程依赖性和数据到外部存储器的读/写,可以相对容易地构建解决潜在故障1)的调试工具。在图11中公开的模型中,过程A启动过程B意味着过程是相互依赖的。限制性共享模式只允许相互依赖的过程共享数据。具体地,过程A启动B意味着B可以读取通过A写入的存储器(1110)。这也意味着一旦过程B通过将它的数据写到外部存储器而终止执行(1125),过程A就可以读取通过过程B写入的存储器(1130)。因此,简单的调试工具可以被构建如下。该工具可以:a)存储写到特定变量的最后一个过程的ID,以及b)将该ID与自从上次写入以来从变量读取的过程的ID进行比较。如果ID比较指示变量由不依赖于上次写入数据的过程的过程读取,则它指示并发过程已共享可变数据。类似地,如果ID比较指示数据由不依赖于自从写入以来读取数据的任何过程的过程最后写入,那么它也指示并发过程共享可变数据。这两个实例在图11的上下文中讨论的状况下都将是程序错误。

[0584] 在不遵守并发过程无法访问可变数据的约束的系统中,这样的测试可能不能可靠地实现。这是因为这样的系统预期有至少在某种程度上取决于输入数据的顺序的结果。

[0585] 因此,根据上文,本公开的一些例子针对包括执行多个过程的多个处理器的设备,与多个处理器相关联的多个高速缓存被提供。该设备可以包括至少包括与第一处理器相关联的第一高速缓存的多个高速缓存。设备还可以包括由多个处理器访问并存储多个变量的存储器,其中多个处理器通过下列操作来执行多个过程中的至少一个:通过将多个变量的子集存储到第一高速缓存来在第一处理器上发起多个过程的第一过程;通过将子集的已更新副本写到存储器来终止第一过程;以及在第一过程的执行期间限制由除了第一过程以外的多个过程对子集的访问。多个过程可以包括第二过程,在第一处理器上的第一过程的执行由在第二处理器上执行的第二过程启动,终止执行包括第一过程使第一高速缓存无效并向第二过程用信号通知完成,并且在接收到该信号时,第二过程访问该子集。该设备可以包括指示符,该指示符提供至少第一状态和第二状态的指示,在第一状态中第一处理器未对在第一高速缓存中的数据的字节进行写入,以及在第二状态中第一处理器已经对在第一高速缓存中的数据的字节进行写入。指示符可以包括单个位。指示符可以被存储在第一高速缓存中。通过写入子集的已更新副本来终止第一过程的执行可以包括在第一状态期间生成对包括在第一高速缓存中的数据的字节的第一高速缓存的行的请求,将行写到存储器使得与在第一高速缓存中的高速缓存数据的字节对应的在存储器中的高速缓存数据的字节实质上是不变的,以及在第二状态期间将行写到存储器以覆写对应于在第一高速缓存中的数据的字节的在存储器中的数据的字节。多个处理器可以包括多个核心,并且多个核心共享多个高速缓存。I/O一致性协议可以使设备能够与另一设备通过接口连接。

[0586] 本公开的一些例子针对一种用于在具有多个处理器和多个高速缓存的设备中执行多个过程的方法,多个高速缓存包括至少第一高速缓存。参考图16E,方法8600包括将多个变量存储(8602)在由多个处理器访问的存储器中,通过将多个变量的子集存储到第一高速缓存来在第一处理器上发起(8604)多个过程的第一过程,通过将子集的已更新副本写到

存储器来终止(8606)第一过程,以及在第一过程的执行期间限制(8608)由除了第一过程以外的多个过程对子集的访问。这些过程被设计为使得在多个过程的执行期间,只有多达单个过程同时访问存储器子集(例如存储器行的一部分)。

[0587] 该方法可以包括在第二处理器上执行(8601)多个过程的第二过程,步骤8604可以包括由第二过程启动第一过程,步骤8606可以包括通过使第一高速缓存无效并且向该过程发信号通知完成来终止第一过程,并且在接收到该信号之时通过第二过程来访问(8612)该子集。该方法可以包括提供(8614)指示符,该指示符提供至少第一状态和第二状态的指示,在第一状态中第一处理器未对在第一高速缓存中的数据的字节进行写入,以及在第二状态中第一处理器已经对在第一高速缓存中的数据的字节进行写入。指示符可以包括单个位。通过写入子集的已更新副本来终止第一过程的执行的步骤8606可以包括在第一状态期间生成对包括在第一高速缓存中的数据的字节的第一高速缓存的行的请求,将行写到外部存储器使得与在第一高速缓存中的高速缓存数据的字节对应的在存储器中的高速缓存数据的字节实质上是不变的,并且在第二状态期间将行写到存储器以覆写对应于在第一高速缓存中的数据的字节的在存储器中的数据的字节。该方法可以包括在多个处理器的多个核心当中共享多个高速缓存。该方法可以包括使设备经由I/O一致性协议或者通过任何其他协议(诸如字节使能支持协议)来与另一个设备通过接口连接。该方法可以包括将具有内容的指示符输出到能够基于指示符来执行更新的存储器,从而管理假竞争。

[0588] 本公开的一些例子针对非临时计算机可读存储介质,其上存储有用于处理软件程序的指令的指令集,指令在由具有多个处理器和多个高速缓存存储器的设备执行时使设备将多个变量存储在由多个处理器访问的存储器中,通过将多个变量的子集存储到第一高速缓存来在第一处理器上启动多个过程中的第一过程,第一高速缓存是设备中的多个高速缓存中的一个,通过将子集的已更新副本写到存储器来终止第一过程,并且在第一过程的执行期间限制由除了第一过程以外的多个过程对子集的访问。指令可以使设备在第二处理器上执行多个过程中的第二过程,由第二过程启动第一过程,通过使第一高速缓存无效并且向第二过程用信号通知完成来终止第一过程,以及当接收到信号时通过第二过程来访问子集。指令可以使设备存储指示符,指示符提供至少第一状态和第二状态的指示,在第一状态中第一处理器未对在第一高速缓存中的数据的字节进行写入,以及在第二状态中第一处理器已经写到在第一高速缓存中的数据的字节。指示符可以包括单个位。指示符可以被存储在第一高速缓存中。通过写入子集的已更新副本来终止第一过程的执行可以包括在第一状态期间生成对包括在第一高速缓存中的数据的字节的第一高速缓存的行的请求,将行写到存储器使得与在第一高速缓存中的高速缓存数据的字节对应的在存储器中的高速缓存数据的字节实质上是不变的,并且在第二状态期间将行写到存储器以覆写与在第一高速缓存中的数据的字节对应的在外部存储器中的数据的字节。这些指令可以使设备在多个处理器的多个核心当中共享多个高速缓存。这些指令可以使设备经由I/O一致性协议与另一个设备通过接口连接。

[0589] 可以提供一种多核处理器设备,其可以包括被配置为使用桶式线程化处理来执行软件程序的多个线程的多个处理核心、一个或更多个数据高速缓存、一个或更多个指令存储器以及线程间通信模块,其可以包括原子计数器部分和用于在多个线程之间同步的数据锁部分。

[0590] 线程间通信模块可以属于或可不属于多个处理核心、一个或更多个数据高速缓存和一个或多个指令存储器。

[0591] 数据锁部分可以包括多个数据锁，其中每个数据锁可以包括用于对锁定数据锁的请求进行掩码的掩码、用于随着时间的过去而改变掩码的掩码编码器、用于选择选定掩码的请求以锁定数据锁的第一优先级编码器以及用于选择锁定数据锁而不考虑掩码的请求的第二优先级编码器。

[0592] 第一和第二优先级编码器可以应用相同的仲裁方案。

[0593] 第一和第二优先级编码器可以应用不同的仲裁方案。

[0594] 多个处理核心可以包括数据锁请求器；其中每个数据锁请求器被配置为执行状态机，其用于控制锁定数据锁的请求的发送，发送将数据锁解锁的请求以及用于选择性地阻止线程的执行。

[0595] 不同的数据锁是二进制变量，且只有当数据锁具有预定值时不同的线程才可以被执行。

[0596] 原子计数器部分可以包括可以包括用于存储多个原子计数器值的多个条目的存储器模块、用于按周期选择多达单个请求以用于更新给定原子计数器的判别器以及用于更新给定原子计数器的处理器。

[0597] 多个条目的数量可以小于数据锁部分的数据锁的数量。

[0598] 可以提供一种方法(见图48的方法6600)，其可以包括由多个处理核心使用桶式线程处理来执行(6210)软件程序的多个线程，其中由多个线程执行可以包括通过利用线程间通信模块的数据锁和原子计数器来在线程的执行之间同步(6220)。

[0599] 线程间通信模块可以属于或可以不属于多个处理核心、一个或更多个数据高速缓存和一个或更多个指令存储器。

[0600] 在线程的执行之间的同步可以包括管理(6622)多个数据锁，其中数据锁的管理可以包括通过掩码对用于锁定数据锁的请求进行掩蔽，随时间的过去由掩码编码器改变掩码，由第一优先级编码器选择选定的掩码的请求以锁定数据锁，并且由第二优先级编码器选择锁定数据锁的请求而不考虑掩码。

[0601] 该方法可以包括可以由第一优先级编码器和第二优先级编码器应用相同的仲裁方案。

[0602] 该方法可以包括可以由第一优先级编码器和第二优先级编码器应用不同的仲裁方案。

[0603] 多个处理核心可以包括数据锁请求器；其中方法可以包括由每个数据锁请求器执行(6630)用于控制锁定数据锁的请求的发送、发送将数据锁解锁的请求并用于选择性地阻止线程的执行的状态机。

[0604] 该方法可以包括在存储器模块的多个条目处存储(6640)多个原子计数器值，由判别器按周期选择多达单个请求以用于更新给定原子计数器，并且由处理器更新给定原子计数器。

[0605] 多个条目的数量可以小于数据锁部分的数据锁的数量。

[0606] 可以提供一种非临时计算机可读存储介质，其上存储有指令集，指令集用于由多个处理核心使用桶式线程处理来执行软件程序的多个线程，其中多个线程的执行可以包括

通过利用线程间通信模块的数据锁和原子计数器来在线程的执行之间同步。

[0607] 数据采集

[0608] 图17示出根据本公开的例子的示例性向量处理器。向量处理器2200可以包括存储数据的可寻址寄存器2202的阵列。向量处理器2200可以在多个输入寄存器2206a-2206c处从外部源接收数据，并将数据存储在可寻址寄存器2202内的特定存储器位置中。在一个例子中，输入寄存器可以输入命令参数(例如，命令的操作码)及其相关联的操作数。操作数可以包括数据或地址操作数，其可以参考阵列的至少一个寄存器。基于输入命令参数，逻辑/处理电路2204可以处理存储在可寻址寄存器中的数据。在美国专利号8,300,058中提供了对示例性向量处理器的完整讨论，该专利通过引用被全部并入本文。

[0609] 存储器组可以与上述系统100和2200结合来使用，以便提供所请求的数据。一种类型的存储器读取的一个例子称为“标量”读取。标量读取通常从单个存储器地址读取单个值。另一种类型的存储器读取是单个“向量”读取，其使用存储器中的向量的单个地址作为起始点来读取多个向量值。换句话说，向量读取将通过使用向量地址“A”以提取使用该地址的特定值来读取向量V(v0,v1,v2,...)的所有值。典型的向量读取依赖于在存储器中的接近的被读取的向量的数据。向量读取将能够通过使用下面的理解来加载向量V的值：如果对于向量的地址是A，那么分量v1位于存储器中的A+1处，分量v2位于存储器中的A+2处，分量A3位于存储器中的A+3处，等等。在上面讨论的八字节例子中，单个向量读取将使用单个访问来检索连续数据的八个字节，其中8个字节中的每一个对应于向量的不同分量。在单个向量读取中的数据最可能存储在单个存储器中。然而，在一些实现中向量读取到多个存储器也许是可能的。

[0610] 图18示出上述单个存储器访问“单个向量”读取的可选方案。具体地，图18示出可能能够“收集”从多个存储器组读取的数据的体系结构1300。收集在下面的意义上也是数据的向量读取：在一些实现中，它可以检索向量数据。再次选择8字节读取作为例子来说明图18所示的体系结构2300的实现。然而，应该理解，在本公开的上下文内，任何适当数量的字节的读取对于体系结构(如体系结构2300)都是可能的。

[0611] 如图18所示，体系结构2300可以用八个单独的存储器组2310a-h替代单个存储器组。体系结构2300还可以包括经由互连2320连接到存储器组2310a-h的多个存储器启动器2330a-h(在本例子中为八个存储器启动器)。每个启动器2330a-h可以经由互连2330将对应于所请求的数据的地址转发给存储器组2310a-h中的一个。互连2320可以将该地址路由到适当的组2310a-h，使得组2310a-h可以提供所请求的数据。通常，每个启动器2330a-h能够路由地址以访问每个组2310a-h中的数据。然而，在一些实施例中，启动器2330a-h可以被预先分配给特定的组或组2310a-h的子集。

[0612] 互连2320可以通过任何合适的体系结构(包括使用交叉开关、交换机、仲裁或其他互连硬件)来实现。互连2320可以进一步包括这些部件的多个实例以及这些和其他部件的各种布置。在某些实现中，互连2320处理当多个地址指向存储器组2310a-h中的同一个时可能发生的“竞争”可能是有利的。可以在本公开的上下文中使用的相对复杂的互连2320可以提供关于处理竞争的和在适应增加的可缩放性(例如，通过增加可用存储器组和/或启动器的数量)时的附加优点。然而，较不复杂的互连2320的设计也可以在较低功耗、复杂性和/或吞吐量方面提供优势。

[0613] 经由“收集”存储器操作在同一周期内访问八个存储器组2310a-h中的每一个可以呈现优于访问单个存储器组的优点，特别是经由标量或单个向量存储器读取。这个潜在的优势通过例子最好地被示出。每个组2310a-h可以具有例如每组一个字节的存取宽度。在这种情况下，如果操作需要数据的八个字节，每个字节存储在不同的组上，体系结构2300可以想得到地在同一个周期中提供数据的所有八个所需的字节。但是，这样做需要实现一种方案，以将所需字节的地址映射到对应于所需数据的组2310a-h，并且适当的地址由启动器2310a-h提供。许多这样的方案是可能的，包括通过仅迭代组2310a-h以顺序访问每个组。

[0614] 在一些实例中，对于数据的八个字节的单个数据请求可以映射到存储器中的不是连续数据块的一部分和/或不被接近地存储在存储器中的八个不同地址。作为例子，对于向量数据的8个字节的单个请求可以包括单字节向量分量，每个存储在不同的组2310a-h上。这可能在图像校正操作(例如鱼眼校正)时发生。在鱼眼校正中，受影响的数据(即需要被请求校正的数据)可能位于图像的周边周围。这样的数据具有存储在存储器中的不同位置处(例如，在具有不同地址的同一存储器组上或在完全不同的存储器组上)的相对高的可能性。在这种情况下，如上所述的单个向量读取可能不被优化用于处理这样的请求。例如，对单个存储器组的单个向量读取可能花费八个周期来读取这样的数据集，花费一个周期来从不同的地址中的每一个读取。

[0615] 图18中所示的体系结构2300然而有可能在一个周期中处理这样的请求。在2300中，每个组2310a-h可以每周期提供数据的一个字节。这导致每周期8个字节的吞吐量，即使存储在不同组中的数据在存储器中不是连续的或接近的。响应于一个请求而组合来自不同存储器位置的数据的这种类型的数据采集作为“收集”操作通常被认为与单个向量存储器读取不同。

[0616] 虽然本公开主要涉及其中指示符2330a-h(图18)中的每一个请求存储器的一个字节的示例性情况，但是其他情况是可能的。例如，每个启动器2330a-h可以请求存储器的两个字节。在这种情况下，启动器2330a-h需要一个字节，并且可以使用最低地址位来选择在响应数据中的正确字节。这在即使响应数据被返回给访问响应字内的不同字节的几个启动器也起作用。

[0617] 在一些实例中，对应于部件或单个数据请求中的其他数据的地址将映射到同一组。由于体系结构2300在该例子中每周期只能从每个组2310a-h提供一个字节，因此这个条件将导致较低的吞吐量。简而言之，在没有用于处理这种特定情况的机制的情况下，体系结构2300将需要两个周期来读取映射到同一组的两个不同地址的数据。例如，如果由启动器2330a-h生成的八个地址中的两个映射到同一组，那么该组将需要两个周期来提供数据的两个字节(即，对于由组中的两个地址中的每个表示的数据的两个字节有一个周期)。在这种情况下，吞吐量将被削减一半到每周期四个字节(在两个周期中的八个字节)的平均值。类似地，如果三个地址映射到同一个组，则需要三个周期来检索数据的所有八个字节。然后，吞吐量减少到八字节/三个周期＝2.67字节/周期。如果所有的地址映射到同一个组，那么用于实现多个组2310a-h将不会有效率增益，因为要获得数据的全部八个字节(即每个周期一个字节的吞吐量)需要8个周期。

[0618] 用于几乎相同的地址处理的已修改的实现

[0619] 如上面讨论的，访问在同一存储器组上的两个不同地址的收集操作可导致降级的

吞吐量。更具体地，在同一组上的不同地址必须在不同的周期中被访问，因而减少了每周期返回的存储器的平均数量。在两个或更多个周期期间访问在同一组上的两个或更多个不同地址的情况在本文中被称为“组竞争”。

[0620] 当被输入来收集的两个或更多个地址不严格地相同但指向在同一个存储器组上的同一个字时，可能出现相关的问题。在那种情况下，地址被认为是“几乎相同的”。除非收集操作具有用于处理几乎相同的地址的机制，否则它们通常被处理为在同一个组上的不同的地址。如上面讨论的，这种处理可能导致“组竞争”和吞吐量的相应降低。

[0621] 图19A示出包括经修改的互连的体系结构2400的实现，经修改的互连可代替互连2320(图18)来使用，以便将相同地址处理添加到收集操作。除了体系结构2400的各种部件之外，图19A还在图的中心处示出存储器组2410.0-2410.7。如图19A所示，存在8个组2410.0-2410.7。然而，数字8仅仅是示例性的，并且应当理解，在本公开的上下文中，体系结构2400可以根据需要被实现为适应多于八个组。体系结构2400也可以适应少于八个组，这取决于应用。

[0622] 假定每个“收集”数据读取命令包括待取出的八个字节的八个地址。地址的数量(八)和数据单元(字节)的长度只是非限制性的例子。互连2400包括元数据计算器24301，该元数据计算器24301根据预定规则生成元数据(例如指示符ac[][]2431)，使得仅多达单个存储器启动器同时访问同一行。元数据计算器24301可以每“收集”读取命令计算元数据。可选地，可以使用缓冲器来存储连续的收集读取命令，并且元数据计算器24301可以为一个收集读取命令接着一个收集读取指令计算元数据。元数据的计算可以包括比较数据单元的地址(或者更确切地，指向存储器行的地址的一部分)以找到几乎相同的地址，标记几乎相同的地址，且然后根据预定义的方案来生成元数据。

[0623] 如图19A所示，体系结构2400包括八个命令队列2420.0-2420.7。每个命令队列可以对应于存储器组2410.1-2410.7中的一个。一系列地址2430(A0…A7)被馈送到队列2420.0-2420.7中的每一个。除了地址2430之外，体系结构2400还可以使用指示符ac[][]2431，其可以用来帮助确定哪个组接收哪个地址，并且如在下一章节中更详细地讨论的，确定是否可以通过利用几乎相同的地址来避免竞争。

[0624] 每个队列2420.0-2420.7可以接收全部八个地址2430(A0…A7)，即使任何一个队列只能使用地址2430之一。这是因为在地址2430被提供给命令队列2420.0-2420.7时，与任何给定地址相对应的组可能是未知的。注意，在该例子中，地址的数量碰巧等于存储器组2420.0-2420.7的数量(即，八个)的事实仅仅是为了说明的目的，而不意为进行限制。地址2430的数量不必等于组2420.0-2420.7的数量。在本公开的某些实现中，地址2430的数量可以大于或小于组2420.0-2420.7的数量。

[0625] 如在后面的章节中更详细讨论的，每个命令队列2420.0-2420.7可以确定哪个地址(A0…A7)2430通过使用地址2430中的指示符ac[][]2431和/或“地址位”来排队(并且因此从它们的相关联的组收集哪些数据)。有许多用于使用指示符ac[][]2431或地址2430来确定哪个地址由哪个组2410.1-2410.7处理的方案。在本公开中对特定方案的讨论意在仅仅是示例性的。任何合适的方案在本公开的上下文内。

[0626] 用于将地址2430确定为队列的一个示例方案是针对每个命令队列2420.1-2420.7收集地址，对于该地址，ac[i][i]具有值“1”，其中“i”是队列2420.1-2420.7及其相关联的

组2410.1-2410.7的索引。另外或者作为本方案的可选方案，还可能将“地址位”或“组选择位”分配给Ai[]2430的某些值。在一个约定中，地址位1:3可以是组选择位，使得如果Ai[1:3]＝＝5，则具有索引5(即，2420.5)的命令队列2420将处理Ai。然而，应该理解，这些方案/约定仅仅是示例性的，并且可以在本公开的上下文中使用适用于该申请的任何其他约定/方案。

[0627] 虽然没有在图19A中明确地示出，流水线阶段可以在每个队列2420.0-2420.7的使用之前。对于每个队列2420.0-2420.7，流水线阶段可以确定哪个地址2430(A0...A7)对应于与队列相关联的组2410.0-2410.7。注意，流水线阶段可以是单个流水线阶段，或者本身可以包括几个流水线阶段。流水线阶段可以基于“地址位”或“组选择位”来执行，如上面讨论的。

[0628] 命令队列2420.0-2420.7可用于处理“组竞争”(即，映射到与队列2420.0-2420.7相关联的组2410.0-2410.7的多个不同地址)。在一些实施例中，队列2420.0-2420.7通过对不同地址排序来处理竞争，使得它们在不同的周期期间被处理，每周期一个地址。

[0629] 由于命令队列2420.0-2420.7对命令竞争的处理在吞吐量方面可能是昂贵的，因此避免将地址放置在命令队列2420.0-2420.7中可能是有利的，除非必要。例如，处理器中地址是不同的但指向在同一字内的数据(“几乎相同”的地址)而不使用命令队列2420.0-2420.7的情况可能是有利的。

[0630] 如图19A所示，该体系结构可以包括一系列复用器(“mux”或“muxes”)405a-h。复用器405a-h可以从组2401.0-2401.7接收所请求的数据。图19A示出从每个组2401.0-2401.7接收数据的复用器405a-h。虽然这是典型的，但是应该理解，该配置仅仅是示例性的，并且其他配置也符合本公开。例如，复用器405a-h中的任何一个可以仅从组2401.0-2401.7的子集接收数据。复用器405a-h还接收指示符矩阵ac[][]2431。复用器405a-h可以经由缓冲器440接收矩阵ac[][]2431或矩阵2431的元素，缓冲器440可以帮助选择要提供哪些数据元素。如果需要的话，缓冲器440还可以引入时延，以及在本文中未明确描述的其他功能。如图19A所示，每个复用器405a-h可以接收矩阵ac[][]2431的不同行。在与本公开一致的其他配置(未示出)中，每个复用器405a-h可以接收整个矩阵ac[][]2431。在后一种情况下，复用器405a-h可以选择要使用的矩阵ac[][]2431的部分，以便确定选择哪个数据。

[0631] 复用器405a-h可以使用矩阵ac[][]2431的元素来确定要提供哪个数据(即，来自哪个组2410.0-2410.7的数据)给与复用器405a-h相关联的字节结果寄存器2401a-h。用于确定向哪个寄存器2401a-h提供哪些数据的特定方法和约定可以改变，尽管在下面的章节中将讨论几个例子。应该理解，用于从本文讨论的ac[][]2431确定数据集的方法仅仅是示例性的，并且任何合适的方法都符合本公开。

[0632] 图19B是示出在图19A中的体系结构的示例性实现的流程图450。如图19B所示，在步骤2451中，地址(A0...A7)2430和指示符ac[][]2431被提供给每个命令队列2420.0-2420.7。在步骤2452处，命令队列2420.0-2420.7使用指示符ac[][]2431来选择适当的地址(A0...A7)2430，其对应于与队列2420.0-2420.7相关联的存储器组405.0-405.7。在这个阶段处，命令队列2410.0-2420.7也使用指示符ac[][]2431来确定所请求的地址是否与在它的队列中的A0...A7 2420地址中的另一个几乎相同。

[0633] 几乎相同的确定可以采取符合本公开的多种形式，如在以下章节中更详细讨论

的。继续步骤2452,如果所提供的地址与另一个地址(以前对该地址提供数据)几乎相同,则在步骤2453中,队列2420.0-2420.7将不将该地址转发到存储器组2410.0-2410.7用于检索。在步骤2453中,每个组2410.1-2410.7从其相应的队列2420.0-2420.7接收要检索的数据的地址。此外在步骤2453中,指示符矩阵ac[][]2431(或者矩阵的某个部分,例如由缓冲器440选择的部分)被转发到复用器405a-h。在步骤2454中,复用器404a-h中的每个接收来自组2410.1-2410.7中的每个的数据。在步骤455中,复用器404a-h使用指示符矩阵ac[][]2431来确定来自组2410.0-2410.7的哪些数据转发到与复用器相关联的字节结果寄存器2401a-h。当队列2420.0-2420.7中的每个为空时,流程图2450(以及收集操作)终止。

[0634] 执行图19B所示的方法可以在两个操作中完成。首先,两个或更多个几乎相同的地址可以被识别出并与指向同一个组的不是几乎相同的其他地址分开。当这完成时,可以使用在单个周期中对存储器组的单次读取来提供由几乎相同的地址所请求的数据。其次,当存储器组返回多字节字时,可以返回与几乎相同的地址中的每个对应的字节。以下章节将更详细地讨论第一步骤。

[0635] 区分几乎相同的地址

[0636] 区分几乎相同的地址的方法之一是应用流水线阶段(除了上面所述的流水线之外或与上面所述的流水线结合)来对不需要单独的存储器访问的地址进行排序。这样的地址可以是例如与在命令队列2420.0-2420.7中的某处的其他地址几乎相同的地址。可以通过创建指示每个地址2430(A0…A7)是否需要存储器地址的指示符(例如,位或其他指示符)来便于排序。指示符可以采取很多不同的形式,但是一种形式是位的二维阵列“ac[][]”2431。

[0637] 二维指示符ac[][]2431的第一个元素可用于指示给定地址是否需要数据读取。这在本文将被称为“访问”指示符。对于在系列2430中的每个地址A0...A7,访问指示符将是在ac[][]2431中的具有等于地址的索引的两个索引的元素。换句话说,对于A0的访问指示符是a[0][0],对于A1的访问指示符是a[1][1],对于A2的访问指示符是a[2][2],等等。应理解,这仅仅是用于说明的目的约定。其他约定是可能的并且在本公开的范围内。作为简写,我们在本文中通常将访问指示符称为“ac[x][x]”。

[0638] 在示例性约定中,可以指定ac[x][x],使得当ac[x][x]被设置为“1”时,与ac[x][x]相关联的地址2430需要单独的存储器读取。在这个配置中ac[x][x]将被设置为“1”的例子将包括其中ac[x][x]对应于没有几乎相同的地址的地址2430,或者其中ac[x][x]对应于作为一系列几乎相同的地址的最低索引地址的地址2430。如果与它相关联的地址与具有较低索引的在队列2420.0-2420.7中的另一地址2430几乎相同,则ac[x][x]可以被设置为“0”。在这种情况下,与ac[x][x]相关联的地址不需要单独的存储器读取,而是可以依赖于对于较低索引几乎相同地址的存储器读取。将在下面的描述中使用这个约定。然而,应该理解,该约定仅仅是说明性的,并且可以采用任何其他合适的约定(例如,当地址2430需要单独的存储器读取时,使用“1”或另一个指示符,等等)。

[0639] 在实现上述约定时,通过使用最低索引几乎相同地址,可以在几乎相同地址当中执行存储器读取。换句话说,如果有一系列地址2430(A0...A7)且地址A1和A3几乎相同,则在存储器读取中使用对于A1的地址来获得对于A1和A3二者的数据。符合上述约定,对于A1的a[x][x]然后被设置为“1”(需要单独的存储器读取)和对于A3设置为“0”(不需要单独的

存储器读取)。

[0640] 指示符ac[ ][ ]2431优选地包括指定在系列2430中每个地址和其他地址之间的关系的其他位。这个第二“关系”指示符在本文中将被称为“a[x][y]”。注意,在2430系列中的每个地址A0...A7将具有1个a[x][x]位和7个a[x][y]位,每个位a[x][y]位对应于该系列中的地址中的另一个地址。针对图20中的通用地址Ax示出这种约定。

[0641] 遵循示例性约定,可以使用关系指示符ac[y]来指示对于相关联的地址数据读取是否应该被传送到另一个地址。例如,如果对于在该系列地址2430(A0…A7)中的另一个地址,ac[x][y]被设置为“1”,则从与ac[x][1]相关联的地址Ax的读取中获得的数据应被提供给系列2430中的地址Ay。在那种情况下,由于不需要从Ay读取,因此对于Ay(a[y][y])的访问指示符可以被设置为“0”。换句话说,ac[x][y]指示应从Ax的数据读取接收数据的较高索引几乎相同地址。

[0642] 图21A-21C分别示出在所有三个地址A1、A3和A4都是几乎相同的示例性情况下对于单独地址A1、A3和A4的ac[x][y]2431矩阵的部分。如图21A所示,对于A1的a[x][x]指示符,最低索引几乎相同地址a[1][1]将被设置为“1”,使得数据从A1被读取。对于A3和A4的ac[x][x]指示符(a[3][3]和a[4][4])分别都将被设置为“0”,因为A3和A4是A1的较高索引几乎相同地址。这在图21B和图21C示出,图21B和21C分别代表对于A3和A4的ac[x][x]2431。

[0643] 在同一个例子中,对于A1的关系指示符ac[x][y]可以在A3和A4位置处被设置为“1”,以便指示从A1读取的数据也应该被传送到A3和A4。这在图21A中示意性地示出。也如图21A所示,在为地址“A3”和“A4”指定的位置处将ac[1][y]设置为“1”,因为与A1相关联的数据需要被传送以满足与A3和A4相关联的请求。这再次是因为在这个例子中,A3是A1的较高阶几乎相同地址。在图21A中,假设A3和A4是与在一系列地址2430(A0...A7)中的A1唯一几乎相同的地址,ac[1][y]的剩余值都具有值“0”。图21B和图21C分别示出对于A3和A4的相应指示符ac[x][y]2430,a[3][y]和a[4][y]。如图21B和图21C所示,对于A3和A4的指示符ac[x][y]都是“0”,因为二者都是与A1的较高索引几乎相同邻居。图21B和图21C还示出对于A3和A4的指示符ac[y]都是“0”,因为A1的较高索引几乎相同邻居(与A3和A4相关联的数据)不需要被提供给在一系列地址2430(A0...A7)中的任何其它地址。

[0644] 图22-24示出了在三个示例性的情况下的对于该一系列地址2430(A0...A7)的整个ac[ ][ ]2431矩阵。根据上述示例性约定,由于较低索引的地址实际上被访问,因此对于x>y,ac[x][y]被设置为“0”。由于在图22-24中的矩阵的左上部分中的每个元素都是零,因此矩阵的左上部分从这些图中的每一个中省略。

[0645] 对于其中在一系列地址2430(A0...A7)中的地址中没有一个与另一个几乎相同的示例性情况,图22示出了整个指示符矩阵(减去左上部分)ac[x][y]2431。换句话说,图22示出在下面的情况下的ac[x][y]2431,其中每个地址2430指向在存储器组2310a-2310h中的与由其他地址2430中的任一个指向的数据不在同一字中的数据。在那种情况下,指示符矩阵ac[x][y]2431的所有对角线元素(即,具有地址的x=y=索引的所有元素)被设置为“1”。这指示系列2430中的每个地址都被访问,并且对应于这些地址的数据将由复用器2905a-h提供给寄存器2401a-h。ac[x][y]中的所有非对角线元素都被设置为“0”,因为从访问地址2430中的每一个返回的数据仅被返回到该特定地址,而不是系列2430中的任何其他地址。换句话说,由于地址2430(A0...A7)中没有一个与任何其他地址几乎相同,因此响应于使用

该特定地址来请求数据的启动器，使用每个地址2430读取的数据将只被返回一次。对于每个地址2430的数据将不响应于不从那个地址特别请求数据的任何启动器而返回。为了反映这种情况，将ac[x][y]2431的所有非对角线元素设置为“0”。

[0646] 图23示出对于其中所有地址2420(A1...A7)与A0几乎相同的情况的ac[x][y]2431。由于A0是最低索引几乎相同地址，因此对于A0的a[x][x]被设置为“1”(即，a[0][0]＝1)。由于所有其他地址A1...A7是对A0的最高阶几乎相同地址，因此它们表示的数据请求可以通过单独访问A0来实现。相应地，对于其他地址A1…A7中的每一个，“访问”指示符元素(ac[x][x])都被设置为“0”。替代地，第一列(ac[0][y])(指示在经由A0访问的数据与其他地址之间的关系的列)都被设置为“1”。这表示应该使用来自A0访问的数据来为在系列2430中的其他地址A1...A7中的每一个提供数据。

[0647] 图24示出了对于第三个例子的ac[x][y]2431，其中A0-A3中的每一个分别与地址A4-A7几乎相同。在这种情况下，由于A0-A3是它们的几乎相同地址对的最低索引地址，因此A0-A3中的每一个在a[x][x]中都具有值“1”。换句话说：a[0][0]＝a[1][1]＝a[2][2]＝a[3][3]＝1。由于A4-A7是与具有较低索引地址(A0-A3)的几乎相同的较高索引地址，因此它们中的每一个在它的a[x][x]中具有值“0”(即，ac[4][4]＝ac[5][5]＝ac[6][6]＝ac[7][7]＝0)。由于来自A0的数据读取为A4所表示的数据读取提供足够的数据，因此A0相对于A4的关系指示符(即，a[0][4])被设置为“1”。再次，这指示由A4表示的数据请求将通过对A0执行的存储器读取而被满足，并且适当的数据将被路由以满足A4数据请求。类似地，对于它们各自的几乎相同的对A5、A6和A7，对于A1、A2和A3的关系指示符各自被设置为“1”(即，ac[1][5]＝ac[2][6]＝ac[3][7]＝1)。这指示来自A1、A2和A3的数据读取分别为由A5、A6和A7表示的数据读取提供足够的数据。如图24所示，ac[x][y]2431的所有其他元素被设置为“0”。

[0648] 上述约定/方案的实际编码可以以与本公开一致的各种方式来完成。现在将介绍对该方案编码的几个示例性方法。然而，应该理解，这些编码方法仅仅是采用上面讨论的一些通用概念、约定和方案的例子。应该理解，该代码的表示不是限制性的，并且本文公开的概念、约定和方案仅仅是示例性的。

[0649] 对上述约定和示例情况编码的一种方法是通过在伪代码中实现下面表达的下面的一般算法。一般算法用于基于一系列的n个地址An来计算ac[x][y]，其中n是任何整数(包括但不限于在上述例子中使用的“8”)。注意，在下面的伪代码中的符号“～＝”指示布尔表达式，其在它的操作数几乎相同时返回“1”，且否则返回“0”。在下面的公式中使用标识符“x”或“y”来表示在An或ac[x][y]中的索引的场合，应该理解，“x”指在ac矩阵中的列，而“y”指行，如图22-24所示。

[0650] 对于所有地址2430An(例如，在8字节例子中，n＝1...7)：

[0651] 1.设置ax[n][n]等于“1”(即，以指示必须访问An所请求的数据)，当且仅当对应于An的数据尚未被较低索引的地址(即，从n-1到0的索引x)访问时：

[0652] a.ac[n][n]＝乘积(!ac[x][n]，对于从n-1到0的x)

[0653] 2.(注意，在这个实现中，ac[0][0]总是为1，指示对应于A0的数据总是被访问。这是因为A0是最低索引地址，且因此不能与具有较低索引的另一个地址几乎相同。

[0654] 2.对于与在ac[n][y]中的An对应的所有行y，如果An与Ay几乎相同，则设置a[n][y]等于“1”(即，将与An相关联的数据返回给Ay的指示符)：

[0655] a.对于所有y>n：

[0656] i.ac[n][y]＝(ac[n][n])&&(An～＝Ay)

[0657] b.(注意，这个计算必须以特定的顺序被执行。具体地，必须在a[n][n]之后计算ac[n][y]，并且对于x<n的所有值必须在a[x][n]之后计算a[n][n])。

[0658] 根据上文，如果对于地址的访问元素(ac[x][x])是“0”，指示该元素未被访问，则将不会响应于使用任何其他地址做出的数据请求而提供与该访问元素对应的数据。因此，在这种情况下，如果ac[x][x]＝0，那么对于地址的a[x][y]＝0。

[0659] 图25A以实际代码示出上述一般算法的实现。如上所述，图25A中的代码对应于八个地址(A0...A7)2430及其对应的8x8指示符矩阵ac[][]2431的示例性情况。图25A中的代码被写入以符合上面所述的顺序要求。例如，在第一步骤中，将a[0][0]被设置为“1”，因为必须访问对应于地址A0的数据。如图25A所示，按列方式在列中计算a[x][y]2431(即，首先针对列ac[x＝0][y]，然后针对列ac[x＝1][y]，然后ac[x＝2][y]，...且最后针对列a[x＝7][y])。如上面所讨论的，在图25A示出的代码意为是示例性的而非限制性的。应该理解，该代码的变型在仍然实现本文公开的体系结构和方法的同时可以偏离图25A中所示的例子。

[0660] 图25A中的代码描述了设置ac[][]2431，使得它既可以用于确定要检索哪个数据也可以返回到寄存器2401a-h。具体地，一旦根据ac[][]2431(图19B，步骤2454)由组2310a-h返回读取数据，就必须将数据返回到正确的启动器(即，到对应于启动器的正确的字节结果寄存器2401a-h)(图19B，步骤2455)。为了完成这个，当且仅当必须将对应于Ai的读取数据返回给启动器j时，位ac[i]的列才可以使位ac[i][j]设置为1。使用图25A中的代码的一个可能的实现依赖于下面的事实：对于任何j，将存在至多一个实例，其中ac[i][j]是“1”。在示例性实现中，可以在由八个组返回的八个读取响应之间执行简单的逻辑“或”，其中对组i的读取响应与ac[i][j]进行逻辑“与”操作。在这种情况下，除非ac[i][j]＝＝0，否则读取响应i将输入或表达式(并返回到寄存器2401a-h)。在这种情况下，0将输入或表达式。

[0661] 因此，根据上文，本公开的一些例子针对包括一个或更多个存储器组、一个或更多个寄存器以及将一个或更多个存储器组与一个或更多个寄存器连接的连接器的设备。该设备还可以包括一个或更多个队列，该一个或更多个队列存储用于从一个或更多个存储器组访问数据的一组地址，该一组地址基于所选择的地址是否与在数据请求中的其他地址几乎相同来从数据请求中选择。该选择可以基于可包括具有对角元素的二维矩阵的指示符矩阵，对角元素指示在数据请求中的地址是最低索引近邻还是没有近邻。该选择可以基于对角元素来选择该一组地址。该选择可以包括与数据请求中的任何其他地址不几乎相同的地址、或者与数据请求中的一组地址几乎相同并且具有在该组中的地址的最低索引的地址中的至少一个。该选择可以包括确定请求中的地址的任一个是否对应于存储在一个或更多个存储器组中的同一个存储器组中的同一个字。连接器可以基于指示符矩阵向一个或更多个寄存器提供数据。基于指示符矩阵向一个或更多个寄存器提供数据可以包括：经由指示符矩阵确定与数据请求中的任何其它地址不几乎相同的或者与数据请求中的一组其它地址几乎相同并且具有该组中的地址的最低索引的在数据请求中的一个或更多个地址，以及将与所确定的一个或更多个地址相关联的数据提供给一个或更多个寄存器。该设备可以包括一个或更多个复用器，并且其中该确定由一个或更多个复用器执行。

[0662] 本公开的一些例子(见例如图25B)针对一种用于处理数据的方法8700,该方法包括:确定(8710)指示数据请求中的地址是否与在数据请求中的其他地址几乎相同的元数据(例如指示符矩阵);基于元数据来选择(8714)用于访问来自一个或更多个存储器组的数据的一组地址;将所选择的一组地址存储8718在一个或更多个队列中;以及基于一个或更多个队列来从存储器组检索8722对应于所选择的一组地址的数据。指示符矩阵可以包括具有对角元素的二维矩阵,该对角元素指示数据请求中的地址是最低索引近邻还是没有近邻,并且选择8714可以基于对角元素来选择该一组地址。选择8714可以包括选择与数据请求中的任何其它地址不几乎相同的地址或者与数据请求中的一组地址几乎相同并且具有该组中的地址的最低索引的地址中的至少一个。选择8714可以包括确定请求中的地址的任一个是否对应于存储在一个或更多个存储器组中的同一个中的同一个字。该方法可以包括基于指示符矩阵经由连接器来向一个或更多个寄存器提供8724数据。基于指示符矩阵向一个或更多个寄存器提供8724数据可以包括:经由指示符矩阵来确定8726与数据请求中的任何其它地址不几乎相同的或者与数据请求中的一组其它地址几乎相同并且具有该组中的地址的最低索引的在数据请求中的一个或更多个地址,并且将与所确定的一个或更多个地址相关联的数据提供8728给一个或更多个寄存器。该确定可以由一个或更多个复用器执行。根据本发明的一个实施例,每存储器组有一个队列,并且该方法包括将地址分配给所有队列,并基于元数据由队列选择要存储的地址。

[0663] 本公开的一些例子针对一种非临时计算机可读存储介质,其上存储有软件程序的指令集,该指令集在由计算设备执行时使计算设备确定指示数据请求中的地址是否与数据请求中的其他地址几乎相同的指示符矩阵,基于指示符矩阵来选择用于访问来自一个或更多个存储器组的数据的一组地址,将所选择的一组地址存储在一个或更多个队列中,以及基于一个或更多个队列来从存储器组检索相应于所选择的一组地址的数据。指示符矩阵可以包括具有对角元素的二维矩阵,该对角元素指示数据请求中的地址是最低索引近邻还是没有近邻,并且该选择可以基于对角元素来选择一组地址。该选择可以包括选择与数据请求中的任何其它地址不几乎相同的地址或者与数据请求中的一组地址几乎相同并且具有该组中的地址的最低索引的地址中的至少一个。该选择可以包括确定请求中的地址的任一个是否对应于存储在一个或更多个存储器组中的同一个存储器组中的同一个字。指令可以使计算设备基于指示符矩阵经由连接器来向一个或更多个寄存器提供数据。该提供可以包括经由指示符矩阵来确定与数据请求中的任何其它地址不几乎相同的或者与数据请求中的一组其它地址几乎相同并且具有该组中的地址的最低索引的在数据请求中的一个或更多个地址,并且将与所确定的一个或更多个地址相关联的数据提供给一个或更多个寄存器。该确定可以由一个或更多个复用器执行。

[0664] 前面的正文讨论了通过仅请求最低索引存储器启动器以取出存储器行来阻止对属于同一存储器行的几乎相同地址的多次访问,多次访问的阻止可以遵循用于选择哪个存储器启动器将取出存储器行的另一方案。这样的方案的非限制性例子可以涉及仅请求最高索引存储器启动器来取出存储器行。只要只有一个存储器启动器请求存储器行,并且请求近邻的任何其他存储器启动器没有访问存储器组,则可以使用任何其他预定的方案。

[0665] 该选择可以基于可包括具有对角元素的二维矩阵的指示符矩阵,对角元素指示数据请求中的地址是最低索引近邻还是没有近邻。

[0666] 地址生成器编程

[0667] 图26示出系统300,其是图17中的向量处理器200的示例性实现,其包括在本公开的上下文中的地址生成。系统300可以包括控制子系统3310,而处理子系统3380可以对应于图17中的逻辑/处理电路204。控制子系统3310可以将参数3320提供给处理子系统3380。处理子系统3380可以包括一个或更多个地址生成单元(AGU)3350a-n,如图26所示。另外,处理子系统3380可以包括常常以迭代或循环方式处理数据(诸如图像数据)的一个或更多个向量单元或数据处理单元(DPU)3330a-m。处理子系统3380可以进一步包括可以帮助处理子系统3380处理数据(诸如图像数据)的标量处理器,例如3370。

[0668] AGU 3350a-n可以通过DPU 3330a-m来帮助存储器访问,诸如在从存储器加载数据和将数据存储到存储器时。存储器可以是例如可寻址寄存器202(图17)、其他本地存储器、外部或远程存储器和/或主存储器。通常,使用用于输入和输出操作的DPU 3330a-m可以限制数据处理子系统3380的速度和/或效率,特别是当数据处理涉及迭代或循环计算时,迭代或循环计算涉及读和写大量数据。在这种情况下,如上面讨论的,访问存储器中的其各个位置中的数据可能需要大量的存储器地址的生成和使用。

[0669] 典型地,子系统3380基于由DPU 3330a-m处理的数据的类型来确定或计算这些地址。例如,对图像数据进行迭代通常以由第一地址定义或标识的在存储器中的位置开始,该第一地址由图像基指针确定。可以使用已知图像尺寸(例如宽度、高度和步幅)从基指针地址计算在图像数据中的其他地址。如果子系统3380完全或主要依靠DPU 3330a-m的带宽来处理数据并计算这些地址,则在DPU 3330a-m中的过程执行的吞吐量可能受到这些地址计算的限制。相反,地址计算可以被卸载到AGU 3350a-n,以便例如提高DPU 3330a-m的效率。在那种情况下,可以在输入向量命令(例如循环)之前基于由处理器3370(例如标量单元)写入的配置寄存器(未示出)通过AGU 3350a-n计算地址。在循环中,AGU 3350a-n可用于使指针或计数器递增和/或检查流程条件的结束。而且,调零计数器也可以由AGU 3350a-n处理。

[0670] 如图26所示,具有AGU 3350a-n的DPU 3330a-m通常通过接受来自控制子系统3310的以AGU参数(例如3320)形式的控制指令来执行代码。AGU 3350a-n可使用这些参数3320a-k来生成用于DPU 3330a-m的地址以写到外部或其他存储器以及从外部或其他存储器写入。这种方法对于处理某些类型的数据可能比其他类型更有效。例如,地址计算可能在处理大量数据时引起延迟,数据包括需要更密集的处理的部分,诸如跨越需要特殊和/或密集的分析的大量不同区域的图像数据扩展(例如,在下面更详细地讨论的图像“特征点”)。在这种情况下,根据AGU参数3320a计算地址的步骤可能是耗时的并导致在处理子系统3380的整体操作中的延迟。这在需要密集计算的数据的这些部分的位置不是事先已知的时尤其正确。

[0671] 在计算机视觉应用中可能需要大量地址计算用于处理的一类图像区域的一个例子是图像中的“特征点”的实例,需要计算上密集的评估/分析。“特征点”可以包括例如在所成像的对象的角附近的图像中的“角像素”。这样的角像素可能需要增加的分析来确定对象的空间范围和/或其他信息,例如对象的轨迹或其他对象到对象的相对接近度。这些特征点可能需要通过AGU 3350a-n的单独地址计算,并引起停转、延迟和较不有效的处理。将在下面更详细地讨论处理延误的方式。

[0672] 图27是示出通过示例性AGU 3450(诸如图26所示的AGU 3350a-n)的信息流的一般化示意图。尽管在图27中未示出,但示例性AGU 3450可以包括多于一个AGU 3350a-n。图4中

所示的控制子系统3410可以例如对应于图26中的控制系统3310。

[0673] 如图27所示,AGU参数(在本文中等同地被称为“控制参数”)3320可以由控制子系统3410提供给AGU 3450。控制参数3320可以包括图像数据,例如图像数据的阵列的基地址、图像的宽度(或步幅)以及图像的高度。这些参数可以一起指定要由DPU 3330a-m处理的图像(图26)。另外,AGU 3450可以从处理器3470接收命令3420。处理器3470可以对应于串行处理器3370(图26)并且是数据处理子系统3380的一部分。例如,AGU 3450可以还包括寄存器3450a1,其可以存储控制参数3320。如图27所示,AGU 3450可以基于控制参数3320来生成地址3430。地址通常指向与系统3400正在处理的数据相对应的在存储器中的位置。例如,它们可以对应于图像数据的或者可以被迭代地或者在计算循环中被处理的其他类型的数据的在存储器中的位置。

[0674] 图28示出与图26所示的系统300类似但具有向处理子系统3380提供控制参数3320a-k的集合的队列3520的系统3500。如图28所示并且如在下面更详细地探讨的,队列3520使系统3500能够将控制参数3320a-k顺序地馈送到AGU 3450。队列3520进一步使系统3500能够使用控制子系统3310来生成控制参数3320a-k,而AGU 3450使用控制参数3320a-k的不同集合来生成地址3430。例如,系统3500可以将控制参数集3320k提供给AGU 3450用于地址3430的生成。

[0675] 可以使用图29更详细地探讨这个过程,图29示出可以由图26和图28的系统处理的示例性图像帧3660。图29还示出在图像帧3660内的在存储器中的图像窗口3662。虽然在图29中仅示出一个图像窗口3662,但是应理解,可能有多个图像窗口(例如窗口664和666等),其可以一起覆盖整个图像帧3660。

[0676] 扫描图像窗口3662或图像窗口的一部分的过程可以通过接收可能在图像帧3660内的开始点(具有基地址)开始。开始点可以与图像帧3660的边界对齐或者不对齐(如图29所示)。

[0677] 图像窗口3662定义了在帧3660内的感兴趣区域。注意,在图29中的描绘不一定按比例绘制。图像窗口3662通常小于图像帧3660,并且通常存在多个图像窗口(例如3662a-n),其定义与同一图像帧3660不同的感兴趣区域。图29还示出各种尺寸:图像窗口3662的宽度和高度(宽度在Xmin和Xman之间被定界,高度在Ymin和Ymax之间被定界)。图像帧的宽度被表示为步幅。

[0678] 图像窗口3662的扫描还可以包括检索在图像窗口3662之外但是在图像帧3660之内的信息。例如,扫描可以跨越在x轴坐标Xstart和Xstart+(Xstep*Xcount)之间以及在y轴坐标Ystart和Ystart+(Ystep*Ycount)之间。Xstep是沿x轴的步长,Xcount定义沿x轴的步长的数量。Ystep是沿y轴的步长,Ycount定义沿y轴的步长的数量。

[0679] 可以垂直和水平地横越图像帧3660。例如,垂直横越图像可以以“ystep”的增量在“ystart”处开始,如图29所示。沿着垂直线的每个步长可以是由变量“ycount”计数的增量。类似地,图像的水平部分可以以如由“xcount”计数的“xstep”的增量从“xmin”开始被横越到“xmax”。

[0680] 仅作为例子,AGU 3450可以使用以下等式来将一个或更多个图像帧3660或窗口3662的图像坐标X,Y迭代地映射到存储器地址中:

[0681] 下一个地址=基址+X+Y*宽度 (等式1)

[0682] 在等式在图1中，“基址”是图像帧3660或图像窗口3662的第一部分的指针或存储器地址。对添加到“基址”的X和Y进行迭代生成表示图像帧3660或图像窗口3662的数据的在存储器中的所有地址。对于每个图像帧3660或窗口3662，“基址”是常数项。在存在多个图像帧3660或窗口3662的情况下，可以存在与多个帧或窗口中的每一个相关联的多个“基址”值。在任一种情况下，对X和Y进行迭代提供与图像帧或窗口相关的所有地址。

[0683] 迭代X和Y如下被完成。X被定义为X＝xstart+(xstep*xcount)。总之，X是与图像帧3660或窗口3662的图像坐标中的水平坐标有关的每行像素数。Y＝ystart+(ystep*ycount)。因此，Y是与图像帧3660或窗口3662的图像坐标中的垂直坐标有关的行的数量。X和Y可以根据存储具有某些假设(例如，每像素强度数据一个字节)的图像帧3660或窗口3662的存储器地址来交替地被定义。如图29所示的，“宽度”可以表示存储在图像帧3660或窗口3662中的行的存储器地址的数量或在存储连续行的相应的第一地址之间的差异。

[0684] 再次参考图28，控制参数集合3320k可以对应于图像帧3660的第一窗口3662。如上面讨论的，在图29的上下文中，可能存在与单个图像帧3660相关联的多个窗口(例如，3662、664、666等)。换句话说，第一窗口3662可以是对应于同一图像帧3660的一系列窗口中的一个窗口，系统3500需要对该图像帧3660执行图像处理。控制参数的集合3320a-k可以例如对应于多个窗口3662、664、666等中的每一个。在该例子中，控制参数3320k可以对应于第一窗口3662，而其他控制参数3320a-j可以对应于同一图像帧3660的后续窗口(664、666等)。

[0685] AGU 3450可以基于控制参数3320k来计算第一地址集合，例如3430k。在该计算期间，控制子系统3410可以生成控制参数3320j的第二集合。例如，控制参数3320j的第二集合可以例如用于图像帧3660的窗口624。如果控制参数3320a-k的生成比AGU 3450可基于控制参数3320a-k的任一个集合生成地址3430a-k更快，则队列3520可以用控制参数3320a-k的不同集合填充，因为在AGU 3450中的地址计算可计算地址。以这种方式使用队列3520可以向AGU 3450提供控制参数3320a-k的连续或几乎连续的供应。如将在下面更详细地探讨的，确保地址参数3320a-k到AGU 3450的几乎连续的供应可以使AGU 3450能够以高吞吐量连续地或几乎连续地操作，以便避免在由AGU 3450计算地址时的“瓶颈”。由于由AGU 3450计算地址可能是耗时的并且相对慢，确保AGU 3450在几乎连续的操作中运行可以允许处理子系统3380避免必须等待新地址3430并且因此以更高的吞吐量水平操作。下面将在图9-11中更详细地探讨这种关系。

[0686] 图30示出由数据处理子系统3380评估的指令的四阶段流水线的示意图。具体地，图30示出几个指令(包括具有四个阶段3710.1-3710.4的指令3710)的示例性流水线700执行。类似地，图30示出具有四个阶段3720.1-3720.4(未标注)的指令3720、具有四个阶段3730.1-3730.4(未标注)的指令3730等。如图30所示，可以每个周期$t_7^0-t_7^{10}$评估一个阶段。例如，指令3710的执行在t_7^0处以阶段3710.1(第一阶段)的评估开始，并在t_7^4处以阶段3710.4(第四阶段)的评估结束。

[0687] 当一条指令终止或每个周期通过阶段4被完全评估时，指令流水线的最大或“全”吞吐量被达到。如图30所示，在示例性流水线700中在t_7^4处达到全吞吐量。换句话说，在t_7^4处，对于不同的指令(即，指令3710的阶段4(3710.4)、指令3720的阶段3(3720.3)、指令3730的阶段2(3730.2)以及指令3740的阶段1(3740.1))评估四个阶段中的每个。因此，t_7^4是流水

线中的所有四个阶段都忙时的周期。如图30所示，全吞吐量一直持续到周期t_7^7。在t_7^7之后，指令3710.1-3770.1的所有第一阶段已被处理，且阶段1不再是全的。然后，流水线从周期$t_7^4-t_7^7$以全吞吐量操作。从$t_7^1-t_7^4$，当流水线尚未全时，该部分被称为执行的“头部”部分。从$t_7^8-t_7^{10}$，当流水线不再是全的时，该部分被称为流水线的“尾部”。通常，尽可能地保持流水线处于全或最大吞吐量是有利的，使得除了别的以外数据处理子系统3380也被最大限度地利用。

[0688] 图31示出由系统300对指令的典型流水线执行的控制子系统时间线。如图31所示，最初在t_8^0处，AGU 3450正在使用控制参数3320被编程，以便为DPU 3330a-m(图28)生成地址3430。在从$t_8^0-t_8^1$的时段期间，AGU 3450可以完全致力于生成地址3430。DPU 3330a-m(图28)可以是空闲的，等待接收地址3430以便开始它们的计算(例如，如上面在图6A至图7的上下文中所讨论的图像数据的处理)。DPU 3330a-m恰好在t_8^1之前从AGU 3450接收地址3430，并且DPU 3330a-m的阶段1开始评估指令3810的阶段1(3810.1)。在t_8^2处，AGU 3450对表示例如图像中的区域(诸如图29中所示的窗口3662)的数据的第一循环完成计算地址3430。在此之后，如图31所示，AGU 3450从$t_8^2-t_8^5$保持空闲，而指令3810-817中的每一个正由DPU 3330a-m处理。如图31所示，DPU 3330a-m流水线在时间t_8^3处达到全吞吐量。对于第一区域(例如，图像窗口3662)的全吞吐量持续直到t_8^4为止，其后“尾部”($t_8^4-t_8^6$)开始，因为不再有第一阶段要由DPU 3330a-m评估。在t_8^6后，评估了第一指令集(3810-817)的最后一个指令(817)的最后一个阶段(817.4)。在此之后，如图31所示，在继续进行任何进一步的数据处理之前，DPU 3330a-m必须等待来自AGU 3450的新地址3430。

[0689] 对于第二图像区域(“区域2”，例如图像窗口664)，周期再次在AGU 3450用对于第二区域的控制参数3320被编程时在t_8^5处开始。AGU 3450从$t_8^5-t_8^8$开始为该新的区域生成新的地址3430。相应地，新的地址在计算再次开始时在t_8^7之前被提供给DPU 3330a-m，这次在指令820上。再次，AGU 3450等待处理新的控制参数3320，这次对于第三区域，“区域3”(例如，图像窗口666)，直到t_8^8为止。这产生了其中DPU 3330a-m保持空闲的另一个时段，特别地在827.4的第四个步骤在等待AGU 3450生成新地址的同时在t_8^{11}处完成处理之后。类似地，对于DPU 3330a-m的新的空闲时段在DPU 3330a-m处理指令837.4的阶段4时在t_8^{13}之后开始。

[0690] 在图31所示的例子中，每当用新的控制参数3320对AGU 3450重新编程时，DPU 3330a-m经历空闲时段。即使当DPU 3330a-m没有完全空闲时，由AGU 3450地址生成引起的延迟也导致它们在小于全吞吐量下周期性地操作，特别地在图31中所示的时段$t_8^0-t_8^3$、$t_8^4-t_8^9$和$t_8^{10}-t_8^{12}$期间。虽然在执行开始时预期小于对于DPU流水线800的全吞吐量的时段(例如，在“头部”$t_8^0-t_8^3$期间)，但是在$t_8^4-t_8^9$和$t_8^{10}-t_8^{12}$处小于全吞吐量的时段由在AGU 3450计算地址3430时由可避免的延迟引起。特别是，如图28所示，队列3520的使用可以调解或减少少于全吞吐量的这些后面的时段。

[0691] 图32示出队列3520对DPU流水线900中的AGU 3450的应用。如在图31中的，存在流水线中的每个指令的四个阶段(阶段1-4)。应该理解，四个阶段的使用仅仅是示例性的。流水线900可以包括任何适当数量的阶段。图34A显示了从$t_9^0-t_9^2$的头部时段，因为当DPU 3330a-m正在等待AGU 3450生成地址3430时，AGU 3450使用控制参数3320被编程。如图32所示，全吞吐量在t_9^2处被达到并继续，直到在t_9^7处的尾部为止。

[0692] 如图28所示，队列3520可适应AGU参数3320a-k(或控制参数3320)的多个集合。例如，对于区域1的控制参数3320在头部时段$t_9^0-t_9^1$期间被提供给AGU 3450。在t_9^1处从区域1的控制参数3320处生成地址3450，此时队列3520将对于区域2的控制参数3320提供给AGU 3450。实质上同时，由AGU 3450基于对于第一图像区域的控制参数3320生成的地址3430被提供给DPU流水线900。这发起由DPU 3330a-m对指令910-913的评估，在t_9^2处达到了全吞吐量。如上所讨论的，当通过DPU流水线900达到全吞吐量时，AGU 3450已经对下一图像区域(即，区域2)处理控制参数3320。换句话说，控制子系统3310将继续推动队列3520中的控制参数3320(或AGU参数3320a-k)，直到队列3520变得充满控制参数3320的容量为止。可以使用各种容量的队列3520，如由示出具有可变容量k的队列3520的图28指示的。

[0693] 如图32所示，控制参数3320到t_9^3为止被提供给对于区域3的AGU 3450。在图32所示的例子中，队列3520在t_9^4处变满。这意味着对于几个区域的控制参数3320已被添加到队列3520，使得不再有控制参数3320可被添加。在这种情况下，即使没有新的控制参数被添加到队列3520，AGU 3450也可以继续处理队列3520中的控制参数3320以生成对于DPU 3330a-m的附加地址3450。当AGU 3450生成一组新的地址3430并准备好接受控制参数3320的新集合时，点在队列3520中打开，并且新的控制参数3320可以被添加。在图32中在t_9^5处指示队列3520的这个打开。在t_9^5处，更多的控制参数3320被添加到队列3520。或多或少地同时，AGU 3450可以用控制参数3320的新集合被编程，以便为另一个区域生成一组新的地址3430。在图32所示的简化例子中，对于四个图像区域的控制参数3320被提供给AGU 3450。应当理解，这仅仅是示例性的，并且可以在队列3520中处理任何合适数量的区域，其可以适应用于控制参数3320的集合的任何合适的容量。

[0694] 应该理解，上述过程将继续进行完全被占用的DPU流水线900，直到最后一个指令开始处理为止。这在图32中由指令914表示，指令914在t_9^7处开始处理。因此，t_9^7开始尾部时段，如上面更详细讨论的。在执行最后的过程之前，可以理解，DPU流水线900将以全吞吐量被占用(如图32中在t_9^2和t_9^7之间所示)，除非控制子系统3310不能足够快地向队列3520提供控制参数3320以与AGU 3450处理控制参数3320以产生地址3430的步调一致。在那种情况下，队列3520将变空，并且数据处理子系统3380将停止，直到控制子系统3310可以在队列3520中推送更多的控制参数3320为止。

[0695] 如图26和图28所示，处理子系统3380可以包括可选的组合元件或特征(“梳(comb)”)3360。组合元件3360可以用于解决当处理子系统3380需要访问“未对齐的数据”时可能出现的以下问题。注意，组合特征3360在图26和图28中被示具有多个部件3360a-m。这些多个部件3360a-m中的每一个可以如下所述单独地处理数据。路由部件3390可以提供数

据的不同集合来为此目的组合部件3360a-m。

[0696] 通常,硬件存储器模块每周期提供有限的固定数量的字节。换句话说,硬件存储器可以只以具有某个数量的字节的增量或分块来访问和提供存储器。典型的硬件存储器模块可以提供例如每周期仅8个字节。在这种情况下,存储器模块提供位于8(8字节)的倍数的地址处的字节。用于计算或处理的期望的或寻求的数据可以被存储在存储器中,使得数据不与这些“分块”对齐。这导致从硬件存储器获取的“数据的未对齐”,因为所需的数据可能存在于检索到的数据中,但不在方便的位置处。在这种情况下,未对齐的数据在它可被使用之前需要被检索和处理。组合元件3360可以用这个过程来帮助AGU 3350a-n。

[0697] 图33A和图33B示出对于对齐的(图33A)和未对齐的(图33B)的数据的8字节访问。在这两种情况下,期望数据的字节由“X”标识。理想地,每个数据检索或读取将被“对齐”,如图33A所示。在这种情况下,数据检索将向本地储存器提供期望数据的确切地8个字节。这在图33A中表示为示出数据的所有期望的8个字节(即,被标识为“X”的数据)。也就是说,在对齐的数据的情况下,单次读取将提供所有的期望数据。然而,常常如图33B所示,读取可能与数据存储未对齐一个“偏移量”。在这种情况下,通过检索/读取来提供期望数据的仅6字节部分(“X”)。另外,提供不需要的数据的两个字节(“Y”)来代替期望数据的其余字节(“X”)。在这种情况下,多次存储器访问是必需的,以提取所有的期望数据(“数据”,在左侧)。随后的处理则是必要的,以从所检索的数据集中提取期望数据。

[0698] 向量处理器上下文中的问题通过例子最好被示出。图像处理中的一个常见操作是获得导数。常常通过取在如下关于x=0定义的在两个不同的周期处的向量值中的差异来定义函数(f(x))的导数(f’(x)):

[0699] $$f'(x)=\frac{f(x+1)-f(x-1)}{2}$$ (等式2)

[0700] 其中“x+1”是较晚的时钟周期,而“x-1”是较早的周期。为了使用向量处理器(例如图2所示的处理器)来评估该表达式,处理子系统3380需要获得f(0+1)[或f(1,2,3,...,8)]和f(0-1)[或f(-1,0,...,6)]。这些数量都将与每周期提供8个字节的硬件存储器不对齐。

[0701] 图34A呈现示出组合元件3360如何操作以解决该问题的示意图。具体地,梳3360提供两个未对齐的8字节向量1101和1102以及偏移量1103以提供对齐的数据。在上面讨论的等式2的导数例子中,为了评估表达式,寻求8字节向量f(-1,0,...,6)。但是,由硬件提供的当前向量可能具有显著的偏移量。例如,存储器硬件可以取出先前向量1102和当前向量1101,如图34A所示。所寻求的向量可以被嵌在先前向量1102和当前向量1101中,使得所寻求的向量需要从表示先前向量1102和当前向量1101的串接的16字节向量中被提取。假定偏移量1103是已知的,组合函数提供所寻求的向量。

[0702] 如图34A所示,组合操作将先前向量1102与当前向量1101串接以形成具有期望数据1105的16字节向量。因此,操作comb(prev,curr,7)将通过1)首先串接1102和1101然后应用所请求的偏移量以提取正确数据来提供所寻求的向量f(-1,0,...,6)。

[0703] 在示例情况下,所寻求的向量是f(-1,0,...6),而当前向量1101可以是f(0,...,7)。先前的向量1102可以是f(-8,...,-1)。这意味着将当前向量1101与在偏移量为7的先前向量1102组合将提供所寻求的以后的向量(-1,0,...6)。换句话说,comb(prev,curr,7)。尽管图34A中的例子示出串接两个向量的梳操作,但它不限于此情况。更具体地,任何合适数

量的向量(例如,2、3或更多)可以被串接以提取数据。而且,使用“梳”串接所必需的向量的数量可取决于或涉及每周期由硬件存储器提供的存储器的量。例如,梳3360可以包括获得图34A所示的三个向量1102、1101和1104,并提取这些向量共有的数据集(未示出)。另外,使用图34A中所示的梳3360的修改版本,更多向量(例如,4、5或更多)的组合是可能的。可选地,来自多于两个向量的数据可以在多于两个向量上连续地多次使用梳3360来组合。

[0704] 在上面的例子中,偏移量“7”(例如,图34A中的偏移量1103)与被处理的数据没有特别的关系。但是,在图像处理的上下文中,偏移量可以并常常确实对应于正在被处理的图像的区域。例如,偏移量可以与对应于图像窗口3662(图29)的数据在存储器中的位置相关联。在这个意义上,每个图像窗口3662可以具有它自己的由AGU 3350a-n读取和理解的对应的偏移量,因为该窗口在存储器中具有特定位置。其他图像窗口(未示出)可以在存储器中具有与它们的位置相对应的它们自己的不同偏移量。在这个和其他的情况下,AGU 3350a-n可以基于由控制子系统3410提供给它们的控制参数3320来对偏移量1103编程。在一个例子中,AGU 3350a-n可以基于控制参数3320来计算偏移量1103。

[0705] 在迭代通过图像窗口3662继续进行的情况下,当AGU 3350a-n和数据处理子系统3380的其余部分通过图像窗口迭代时,图34A中的先前向量1102和当前向量1101将改变。然而,偏移量1103将可能保持相同,例如将是“循环不变的”或“迭代不变的”。如上面所讨论的,AGU 3350a-n可以利用这种循环不变偏移量1103来对梳3360编程。AGU 3350a-n计算对于新图像窗口3662和/或控制参3320的集合的地址。在涉及等式2的评估的上面的例子中,偏移量1103将是“7”。在其他例子中,其中图像窗口3662在8的倍数(如图33A和图33B所示的8字节存储器检索)加上偏移量(例如3)的地址处开始,AGU 3350a-n可以确定该偏移量并将它提供给梳3360。在这个变型中,每个AGU 3350a-n可能对梳3360编程。另一方面,根据需要,每次只有AGU 3350a-n的仅一个子集可对梳3360编程。

[0706] 如图32所示,使用队列3520来将控制参数3320排队可增加总数据处理子系统3380的吞吐量。尽管在图32中未明确示出,吞吐量的这个增加可以包括减少或最小化在计算输入到梳3360特征的参数时的拖延或延迟。例如,当控制参数3320在队列3520中排队时,如图28所示,这也可以以允许它最大化可能的吞吐量的方式提供输入参数梳3360。

[0707] 因此,根据上文,本公开的一些例子针对一种包括存储器和数据处理子系统的设备。数据处理子系统可以包括至少一个数据处理单元以及至少一个地址生成单元,其基于参数的第一集合来在存储器中生成第一组地址。该设备可以包括将参数的第一集合和参数的第二集合提供给数据处理子系统的控制子系统以及队列,该队列存储参数的第一集合和参数的第二集合并且在数据处理单元使用第一组地址来处理数据时向地址生成单元提供参数的第二集合。由数据处理单元处理的数据可以包括至少一个图像窗口。第一组地址中的至少一个可以对应于在存储器中存储至少一个图像窗口的第一部分的位置。地址生成单元可以通过使用图像参数对至少一个图像窗口进行迭代来生成第一组地址,图像参数包括在存储器中的存储图像窗口的第二部分的地址。数据处理子系统可以包括将图像参数提供给地址生成单元的标量处理器。数据处理单元可以包括至少一个向量处理器,并且由数据处理单元处理的数据包括向量数据。数据处理子系统可以以增量的方式从存储器中检索数据,并且数据处理单元可以使用偏移量来串接在检索到的数据中的未对齐的向量数据,该偏移量指示未对齐的向量数据关于存储器中的增量未对齐。地址生成单元可以计算由数据

处理单元在串接中使用的偏移量。地址生成单元可以通过对至少一个图像窗口的多个部分进行迭代来生成第一组地址，并且偏移量可以包括迭代不变参数。

[0708] 本公开的一些例子针对用于处理数据的方法。参考图34B，方法8800包括经由控制子系统将参数的第一集合和参数的第二集合提供(8810)到数据处理子系统，数据处理子系统包括至少一个数据处理单元和至少一个地址生成单元；将参数的第一集合和参数的第二集合存储(8812)在队列中；基于参数的第一集合经由地址生成单元来生成(8814)存储器中的第一组地址；以及当数据处理单元使用第一组地址处理数据时，经由队列将参数的第二集合提供(8816)到地址生成单元。由数据处理单元处理的数据可以包括至少一个图像窗口，第一组地址中的至少一个可以对应于在存储器中的存储至少一个图像窗口的第一部分的位置，以及经由地址生成单元生成第一组地址的步骤8814可以包括使用图像参数对至少一个图像窗口进行迭代(8818)，图像参数包括在存储器中的存储图像窗口的第二部分的地址。数据处理子系统可以包括将图像参数提供给地址生成单元的标量处理器。数据处理单元可以包括至少一个向量处理器，并且由数据处理单元处理的数据包括向量数据。该方法可以包括经由数据处理子系统以增量的方式从存储器中检索数据，并且经由数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据，该偏移量指示未对齐的向量数据在存储器中关于增量未对齐。该方法可以包括经由地址生成单元来计算由数据处理单元在串接中使用的偏移量。该方法可以包括经由地址生成单元通过对至少一个图像窗口的多个部分进行迭代来生成第一组地址，并且偏移量可以包括迭代不变参数。

[0709] 本公开的一些例子针对一种非临时计算机可读存储介质，其上存储有软件程序的指令集，指令集在由计算设备执行时使计算设备经由控制子系统将参数的第一集合和参数的第二集合提供到数据处理子系统，数据处理子系统包括至少一个数据处理单元和至少一个地址生成单元；将参数的第一集合和参数的第二集合存储在队列中；基于参数的第一集合经由地址生成单元来生成存储器中的第一组地址；以及当数据处理单元使用第一组地址处理数据时，经由队列将参数的第二集合提供到地址生成单元。由数据处理单元处理的数据可以包括至少一个图像窗口，第一组地址中的至少一个可以对应于在存储器中的存储至少一个图像窗口的第一部分的位置，以及该指令集可以使计算设备经由地址生成单元通过使用图像参数对至少一个图像窗口进行迭代来生成第一组地址，图像参数包括在存储器中的存储图像窗口的第二部分的地址。数据处理子系统可以包括将图像参数提供给地址生成单元的标量处理器。数据处理单元可以包括至少一个向量处理器，并且由数据处理单元处理的数据包括向量数据。该指令集可以使计算设备经由数据处理子系统以增量的方式从存储器中检索数据，并且经由数据处理单元使用偏移量来串接在检索到的数据中的未对齐的向量数据，该偏移量指示未对齐的向量数据在存储器中关于增量未对齐。指令集可以使计算设备经由地址生成单元来计算由数据处理单元在串接中使用的偏移量。指令集可以使计算设备经由地址生成单元通过对至少一个图像窗口的多个部分进行迭代来生成第一组地址，并且偏移量可以包括迭代不变参数。

[0710] 虽然已经参考附图充分描述了所公开的例子，但是应该注意，对于本领域的技术人员来说，各种变化和修改将变得明显。这样的改变和修改应被理解为被包括在由所附权利要求限定的所公开的例子的范围内用桶式分类的单线程向量化

[0711] 在下文中，术语“数据点”和“数据单元”以可互换的方式使用。

[0712] 图17的向量处理器3200的示例能力可以包括生成和存储直方图的能力。直方图可以包括数据结构或表格，其中属于特定类别的数据的出现频率被制成表。可能的数据值被分成几个类别或直条，并且对每个直条的出现频率被制成表。

[0713] 存储直方图可以包括将数据分类到直条内，对直条内的成员的数量计数，并然后确保属于特定直条的数据一起被存储在存储器中。图35示出根据本公开的例子的用于桶式分类的示例性方法。在图35中示出的方法可以由图17所示的向量处理器实现。

[0714] 在步骤S4302处，数据由向量处理器接收以被分类到直条中。数据可以经由输入寄存器206a-c从外部源被输入，或者在另一例子中可以已经存在于可寻址存储器寄存器202中。在图35所示的例子中，在被标记为x的示例数据上示出每个步骤以及相应的例子。在步骤4302中，所接收的数据被标记为“x”。

[0715] 在步骤S4304处，确定接收到的数据将被分类到的桶。该确定可以由处理电路208做出。在图像处理上下文中，数据将被存储在直方图中的哪个直条中的确定可取决于数据所表示的“色调(tone)”。在其他例子中，直条可以表示单独像素的亮度级。如在S4304中所示的且为了说明的目的，数据“x”将被存储于的桶由“A”表示。

[0716] 在步骤S4306处，可以更新向量处理器的直方图阵列。直方图阵列可以被存储在可寻址寄存器202内，并且表示已经被分类到特定直条中的成员的总数(即，数据点的数量)。在图35的例子中，在步骤S4306处，H[A]可以在确定数据“x”也将是直条A的一部分之前包含K个元素。可以更新对应于在直条“A”内的数据点的数量的直方图H[A]。在S4306的例子中，H[A]已经被更新以显示现在在直条内存在K+1个元素(因此加上数据“x”)。

[0717] 一旦直方图阵列被更新，基于包含在直方图阵列内的更新的值，就可以在步骤S4308处确定在可寻址寄存器202内的地址位置。如上面讨论的，属于特定数据点的每个数据点可以一起存储在可寻址寄存器202内。因此在图35的例子中，所有的数据点(包括x)都可以一起存储在存储器中。一旦在步骤S4304处将数据点分配给直方图直条，那个数据点就可以接着与也属于直方图直条的所有数据点存储在一起。为了便于在可寻址寄存器202内的数据点的这样的分类，可以利用直方图阵列来确定数据点将被存储在可寻址寄存器中的哪里。

[0718] 在S4308的例子中，可以使用直条A的直方图阵列(H[A])来确定数据点x将被存储在存储器中的哪里，因为x在步骤S4304处已经被确定为直条A的一部分。如在S4308中所示，所确定的地址可以被确定为K+1+Base，其中K+1表示如在关于步骤S4306的讨论中讨论的直方图的已更新值，并且基地址表示初始偏移值，其表示对于A直条的数据点的开始存储器位置。因此，在一个例子中，要被分类在A直条中的第一数据点可以被存储在Base+1存储器位置处。第二数据点(其在图35的例子中将使H[A]更新为2)可以被存储在存储器位置Base+2处。因此，使用图35的例子，由于“x”表示直方图直条A的第K+1个数据点，因此它在步骤S4310处可被存储在存储器位置K+1+Base处。

[0719] 图35的例子示出了串行化直方图分类过程的例子。每个数据点被分类到一个直条中，且然后基于直条确定一次一个地存储在存储器位置中。在图35中示出的过程也可以是向量化的，意味着可以同时对多个数据点分类，使得可以在一个操作中对数据的向量(即，具有多个数据点的单个数据对象)分类。

[0720] 向量化桶式分类算法可导致在它生成直方图所花费的时间中的加速。在要求实时

处理的应用和大量数据被处理的应用中，这种加速可以证明是有益的。不是必须等待每个单独的数据点必须一次一个地被分类，数据块可以同时被分类。在一个例子中，如果数据的向量包含四个数据点并且这四个数据点同时被分类到直条中，则与一次一个地对每个数据点分类的分类算法相对，将这些数据点分类所需的时间量可以大大减少。

[0721] 图36示出了根据本公开的例子的示例性向量化桶式分类算法。图4的算法与上面讨论的图35的例子类似，但不是一次对一个数据点进行操作，而是3个数据点的向量可以同时被分类。

[0722] 在步骤S4402处，可以接收数据的向量。在图36的例子中，数据的向量可以包含被标记为x、y和z的三个数据点。在步骤S4404处，可以以与关于图35的步骤S4304所讨论的方式实质上相同的方式给向量的每个数据点分配一个桶。在S4404的例子中，数据点x、y和z可以分别被分类到桶A、B和C中。

[0723] 一旦向量内的每个数据点与直方图内的直条匹配，该过程就可以移动到S4406，其中与所匹配的直条相关联的每个直方图阵列被更新，类似于图35的步骤S4306。在图36的例子中，在步骤S405处，每个直方图直条A、B、C可以具有分别与它关联的直方图阵列H[A]、H[B]和H[C]。在步骤S4406处，可以同时更新每个直方图阵列。类似于图35中的讨论，直方图阵列可以保持在特定直方图直条内包含的数据点的数量的计数。每当新数据点被添加到特定的直方图阵列时，就可以更新阵列。在步骤S4406处，当直条A、B和C正在接收新的数据点时，它们的每个相关联的直方图阵列可以递增1以说明新的数据点。因此，作为例子，H[A]可以从K更新到K+1，H[B]可以从L更新到L+1，H[C]可以从M更新到M+1。

[0724] 在步骤S4408处，可以以与在图35中在步骤S4308处所述的实质上相同的方式来确定每个数据点将被写到的地址。每个直方图直条可以具有相应的基地址，其中基于在直条内的每个数据点将被写到的地址。该地址可以通过将直条的当前直方图阵列值添加到直条的基地址来确定。在图36的例子中，在步骤S4408处，可以将数据点x写到地址K+1+BaseK，其中BaseK可以表示直条K的数据点的基地址。类似地，数据点y可以被写到存储器位置L+1+BaseL，其中BaseL可以表示直条L的数据点的基地址。最后，数据点z可以被写到存储器位置M+1+BaseM，其中BaseM可以表示直条M的数据点的基地址。不是如关于图35的步骤S4308所讨论的产生单个地址，该方法可以产生地址的向量，其中地址的向量的每个元素可以对应于在最初接收的数据的向量内包含的相应数据点的地址。在步骤s4410处，可以将数据写到在步骤S4408中确定的存储器位置。

[0725] 如果在向量内的每个数据点将被分类到不同的直条中，则关于图36描述的算法可以有效地操作。当向量内的每个数据点将被分类到不同的直条时，则与唯一地依赖于更新的直方图阵列的算法可以是有效的，因为更新的直方图阵列值可以被添加到直条的基存储器地址以生成数据点可以被存储于的准确地址。但是，如果向量包含将被分类到同一个直条中的两个数据点，则上述算法可产生错误。

[0726] 图37示出根据本公开的例子的另一示例性向量化直方图直条分类算法。图37的算法与上面讨论的图36的例子类似，但不是使向量内的每个数据点被分类到不同的直条中，而是同一向量内的多个数据点可被分类到同一直条中。

[0727] 在步骤S4502处，可以接收数据的向量。在图37的例子中，数据的向量可以包含标记为x、y和z的三个数据点。在步骤S4504处，可以以与关于图35的步骤S4304所讨论的实质

上相同的方式来给向量的每个数据点分配桶。在S4404的例子中，数据点x、y和z可以分别被分类到桶A、B和A内。与图36的例子相反，在图37的例子中，在同一向量内的两个数据点被确定为属于同一个桶(即，两个数据点x和z都被确定为属于桶A)。

[0728] 在步骤S4506处，以类似于图36的步骤S4406的方式，每个直条的直方图阵列被更新以反映所添加的数据点的添加。如所示，H[A]从K更新为K+2，以反映两个数据点被添加到直条A的事实。H[B]从K更新为K+1，以反映一个数据点正在被添加的事实。

[0729] 在步骤S4508处，类似于图36的步骤S4408，确定每个数据点将被写到的地址，并且可以部分地基于与每个数据点将被分类到的直条相关联的直方图阵列的值来生成地址的向量。如在步骤S4508所示的，由于算法使用直方图阵列值来生成地址，数据点x和z都被分配给地址K+2+BaseK，因为H[A]的值等于K+2。

[0730] 在步骤S4510处，可以将数据点存储于在步骤S4508处确定的地址位置中。然而，由于步骤S4508使用直方图阵列来确定地址，因此数据点x和z都可以被写到在K+2+BaseK处的同一地址位置。将两个数据点写到存储器中的同一位置可能导致错误和数据的丢失。

[0731] 在关于图35所讨论的串行化向量桶式分类算法中，当每个数据点一次一个地被分类时，直方图阵列可以逐点递增地被更新，使得在特定的直条内的每个数据点可以被写到唯一的存储器地址。然而，在向量化分类算法中，如图4和图37所讨论的，其中数据点被同时分类，可以更新直方图阵列以同时说明多个数据点。因此，唯一地依赖于更新的直方图阵列来生成地址的算法可能导致在被存储在同一直条中的数据点之间的存储器竞争，如图37的例子所示。因此，在向量化桶式分类算法中，算法可以被修改以解释其中在向量内的多个数据点将被分类到同一直条中的实例。

[0732] 图38示出了根据本公开的例子的能够处理竞争的示例性向量化桶式分类算法。步骤S4602和S4604与图37的它们的对应的S4502和S4504相同，且因此那些步骤的讨论可以如上参考。

[0733] 为了处理在单个向量内的多个数据点被分类到同一直条中的实例，在步骤S4606处，算法可以确定在数据点在步骤S4604处被分类到其中的桶之间是否存在任何竞争。竞争可以被定义为将被分类到同一个直条中的两个或更多数据点。如关于图37所讨论的，使用图36的向量化桶式分类例子在两个或更多个数据点之间存在竞争的情况下可能导致数据点被写到存储器中的错误位置上。

[0734] 一旦确定竞争存在，就可以在步骤4608处将向量分类算法串行化，以确保数据点根据它们已经被分类到其中的桶而被准确地存储在存储器中。

[0735] 如步骤S4608a-c处所示，可以使用参考图3A和图4讨论的向量桶式分类算法，但不是对数据的向量进行操作，该过程可以被串行化，使得在向量中的每个数据点可以被顺序地操作。因此在步骤S4608a处，与桶A相关联的直方图阵列可以从K被更新到K+1。在对H[A]的第一次更新之后，可以如上面关于图3A和图4所讨论的生成对于数据点x的地址。如在关于步骤S4608a的例子中所示的，地址可以由公式K+1+BaseK表示，其中K+1可以表示更新的直方图阵列值，并且BaseK可以表示直方图阵列将被存储于其中的初始存储器地址，如上面所讨论的。

[0736] 在步骤S4608b处，关于步骤S4608a描述的相同过程可以被应用于Y数据点，其中与桶B相关联的直方图阵列被更新，并且地址被生成用于存储在数据点Y的存储器中。

[0737] 在步骤S4608c处，关于步骤S4608a和S4608b描述的相同过程可以被应用于Z数据点，其中与桶A相关联的直方图阵列被更新(第二次)，并且地址被生成用于存储在数据点Z的存储器中。

[0738] 再次参考步骤S4608，如果没有在步骤S4606处检测到存储器竞争，则可以根据关于图36讨论的例子来向量化桶式分类。最后，在步骤S4608d处，可以将单独的数据点X、Y和Z写到它们相应的地址。在一个例子中，步骤S4608d可以顺序地被完成，意味着每个单独的数据点被顺序地写到存储器。在另一个例子中，数据点可以以与关于图35所讨论的实质上相同的方式被并行地写到存储器。

[0739] 如在上面图38的例子中所讨论的存储器竞争的情况下，通过部分地串行化向量桶式分类算法，与如关于图35讨论的被纯粹串行化的桶式分类算法相比，桶式分类算法的吞吐量仍然可以产生更快的吞吐量。当在特定向量内没有竞争时，吞吐量在其最快处，因为整个算法可以在向量上操作。然而，如果存在竞争，虽然吞吐量降低以处理竞争，但是通过仅串行化确保没有存储器错误所必需的算法的部分，当与纯粹序列化桶式分类算法相比时更高的吞吐量仍然被维持。

[0740] 因此，根据上文，本公开的一些例子针对一种设备，其包括：至少一个输入寄存器、至少一个可寻址寄存器以及至少一个处理单元，处理单元被配置为通过以下操作来生成直方图：经由输入寄存器接收数据向量，数据向量包括多个数据点；使多个数据点与直方图的多个直条匹配；确定在所匹配的多个数据点中是否存在竞争；如果在所匹配的多个数据点中存在竞争，则将数据向量的数据点的至少两个串行地分类到多个存储器位置中；以及如果在所匹配的多个数据点中不存在竞争，则将数据向量的数据点的至少两个并行地分类到多个存储位置中。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，如果在所匹配的多个数据点中存在竞争则将数据向量的数据点的两个或更多个顺序地分类到多个存储位置中包括：在第一时间段期间更新与直方图的至少一个直条相关联的直方图阵列；生成第一地址位置，第一地址位置对应于数据向量的第一数据点，第一地址位置基于在第一时间段期间更新的经更新的直方图阵列；基于所生成的第一地址位置来将第一数据点写到可寻址寄存器；在第二时间段期间更新直方图阵列；生成第二地址位置，第二地址位置对应于数据向量的第二数据点，第二地址位置基于在第二时间段期间更新的经更新的直方图阵列；以及基于所生成的第二地址位置来将第二数据点写到可寻址寄存器。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，如果在所匹配的多个数据点中不存在竞争则将数据向量的数据点的两个或更多个并行地分类包括：在第一时间段期间更新与直方图阵列的第一直条相关联的第一直方图阵列；在第一时间段期间更新与直方图阵列的第二直条相关联的第二直方图阵列；生成第一地址位置，第一地址位置对应于数据向量的第一数据点，第一地址位置基于经更新的第一直方图阵列；生成第二地址位置，第二地址位置对应于数据向量的第二数据点，第二地址位置基于经更新的第二直方图阵列；在第二时间段期间基于所生成的第一地址位置来将第一数据点写到可寻址寄存器；以及在第二时间段期间基于所生成的第二地址位置来将第二数据点写到可寻址寄存器。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，确定在所匹配的多个数据点内是否存在地址竞争包括确定多个数据点中的两个或更多个是否与直方图的多个直条的公共直条匹配。除了以上公开的一个或更多个例子之外或作为其可选方

案，在一些例子中，生成第一地址位置并生成第二地址位置包括将直方图阵列的值添加到与直方图的至少一个直条相关联的基地址。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，生成第一地址位置包括将经更新的第一直方图阵列的值添加到与直方图的第一直条相关联的基地址，并且其中生成第二地址位置包括将经更新的第二直方图阵列的值添加到与直方图的第二直条相关联的基地址。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，使多个数据点与直方图的多个直条匹配包括将多个数据点中的每个数据点关联到直方图的多个直条中的直条。

[0741] 本公开的一些例子针对一种方法，其包括：经由输入寄存器接收数据向量，数据向量包括多个数据点；使多个数据点与直方图的多个直条匹配；确定在所匹配的多个数据点中是否存在竞争；如果在匹配的多个数据点中存在竞争，则将数据向量的数据点的至少两个串行地分类到多个存储器位置中；以及如果在所匹配的多个数据点中不存在竞争，则将数据向量的数据点的至少两个并行地分类到多个存储器位置中。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，如果确定了在所匹配的多个数据点中存在竞争则将数据向量的两个或更多个数据点顺序地分类到多个存储器位置中包括：在第一时间段期间更新与直方图的至少一个直条相关联的直方图阵列；生成第一地址位置，第一地址位置对应于数据向量的第一数据点，第一地址位置基于在第一时间段期间更新的经更新的直方图阵列；基于所生成的第一地址位置来将第一数据点写到可寻址寄存器；在第二时间段期间更新直方图阵列；生成第二地址位置，第二地址位置对应于数据向量的第二数据点，第二地址位置基于在第二时间段期间更新的经更新的直方图阵列；以及基于所生成的第二地址位置来将第二数据点写到可寻址寄存器。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，如果在所匹配的多个数据点中不存在竞争则将数据向量的数据点的两个或更多个并行地分类包括：在第一时间段期间更新与直方图阵列的第一直条相关联的第一直方图阵列；在第一时间段期间更新与直方图阵列的第二直条相关联的第二直方图阵列；生成第一地址位置，第一地址位置对应于数据向量的第一数据点，第一地址位置基于经更新的第一直方图阵列；生成第二地址位置，第二地址位置对应于数据向量的第二数据点，第二地址位置基于经更新的第二直方图阵列；在第二时间段期间基于所生成的第一地址位置来将第一数据点写到可寻址寄存器；以及在第二时间段期间基于所生成的第二地址位置来将第二数据点写到可寻址寄存器。除了以上公开的一个或哥呢个多个例子之外或作为其可选方案，在一些例子中，确定在所匹配的多个数据点内是否存在地址竞争包括确定多个数据点中的两个或更多个是否与直方图的多个直条的公共直条匹配。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，生成第一地址位置并生成第二地址位置包括将直方图阵列的值添加到与直方图的至少一个直条相关联的基地址。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，生成第一地址位置包括将经更新的第一直方图阵列的值添加到与直方图的第一直条相关联的基地址，并且其中生成第二地址位置包括将经更新的第二直方图阵列的值添加到与直方图的第二直条相关联的基地址。除了上面公开的一个或更多个例子之外或作为其可选方案，在一些例子中，使多个数据点与直方图的多个直条匹配包括将多个数据点中的每个数据点关联到直方图的多个直条中的直条。

[0742] 本公开的一些例子针对一种非临时计算机可读存储介质，其上存储有用于处理软

件程序的链式指令的指令集，指令集在由计算设备执行时使计算设备：经由输入寄存器接收数据向量，数据向量包括多个数据点；使多个数据点与直方图的多个直条匹配；确定在所匹配的多个数据点中是否存在竞争；如果在所匹配的多个数据点中存在竞争，则将数据向量的数据点的至少两个串行地分类到多个存储器位置中；以及如果在所匹配的多个数据点中不存在竞争，则将数据向量的数据点的至少两个并行地分类到多个存储位置中。除了以上公开的一个或更多个例子之外或作为其可选方案，在一些例子中，如果确定了在所匹配的多个数据点中存在竞争则将数据向量的数据点的两个或更多个串行地分类到多个存储器位置中包括：在第一时间段期间更新与直方图的至少一个直条相关联的直方图阵列；生成第一地址位置，第一地址位置对应于数据向量的第一数据点，第一地址位置基于在第一时间段期间更新的经更新的直方图阵列；基于所生成的第一地址位置来将第一数据点写到可寻址寄存器；在第二时间段期间更新直方图阵列；生成第二地址位置，第二地址位置对应于数据向量的第二数据点，第二地址位置基于在第二时间段期间更新的经更新的直方图阵列；以及基于所生成的第二地址位置来将第二数据点写到可寻址寄存器。除了上面公开的一个或更多个例子之外或作为其可选方案，如果在所匹配的多个数据点中不存在竞争则将数据向量的数据点的两个或更多个并行地分类包括：在第一时间段期间更新与直方图阵列的第一直条相关联的第一直方图阵列；在第一时间段期间更新与直方图阵列的第二直条相关联的第二直方图阵列；生成第一地址位置，第一地址位置对应于数据向量的第一数据点，第一地址位置基于经更新的第一直方图阵列；生成第二地址位置，第二地址位置对应于数据向量的第二数据点，第二地址位置基于经更新的第二直方图阵列；在第二时间段期间基于所生成的第一地址位置来将第一数据点写到可寻址寄存器；以及在第二时间段期间基于所生成的第二地址位置来将第二数据点写到可寻址寄存器。除了以上公开的一个或更多个例子之外或作为其可选方案，确定在所匹配的多个数据点内是否存在地址竞争包括确定多个数据点中的两个或更多个是否与直方图的多个直条的公共直条匹配。除了以上公开的一个或更多个例子之外或作为其可选方案，生成第一地址位置并生成第二地址位置包括将直方图阵列的值添加到与直方图的至少一个直条相关联的基地址。除了上面公开的一个或更多个例子之外或作为其可选方案，生成第一地址位置包括将经更新的第一直方图阵列的值添加到与直方图的第一直条相关联的基地址，并且其中生成第二地址位置包括将经更新的第二直方图阵列的值添加到与直方图的第二直条相关联的基地址。除了上面公开的一个或更多个例子之外或作为其可选方案，使多个数据点与直方图的多个直条匹配包括将多个数据点中的每个数据点关联到直方图的多个直条中的直条。

[0743] 使用硬件存储器错误处理的组合电源管理

[0744] 说明书和附图提到“软错误”。应该注意，软错误仅仅是可以使用一个或更多个奇偶校验位来检测的错误的非限制性例子，并且应用不限于软错误。例如，检测到的错误可能是“硬”错误。

[0745] 说明书和附图提到从存储器中读取的字。字仅仅是存储器行的宽度和/或在单个读操作期间从存储器和/或原子读操作单元的读取的数据的量的非限制性例子。因此，从存储器中读取的数据的大小可能超过一个字。

[0746] 说明书和附图提到被写到存储器的字节。字节仅仅是存储器行的一部分的宽度和/或在单个写操作期间被写到存储器和/或原子写操作单元的数据的量的非限制性例子。

因此,写到存储器的数据的大小可以包括任意数量的位,并且可以超过一个字节,同时小于存储器行的相应宽度和/或在单个读操作期间从存储器和/或原子读操作单元读取的数据的量。

[0747] 说明书和附图提到奇偶校验位和奇偶校验。注意,奇偶校验仅仅是错误检测码的非限制性例子,并且任何类型的错误检测码和/或错误校正检测可以被应用。奇偶校验位可以被视为冗余位的非限制性例子。

[0748] 图39示出根据本公开的例子的示例性存储器设备。存储器设备55100可以包括存储器55102。存储器55102可以包括物理介质,其上可以存储数据。在一个例子中,存储器55102可以被实现为易失性存储器,例如同步动态随机存取存储器(SDRAM)模块或其他类型的动态随机存取存储器(DRAM)。存储器控制器55104可以连接到存储器55102。存储器控制器55104可以包含使存储器55102能够被读取和写入的逻辑,并且还可以包含允许存储器55102被刷新的逻辑。

[0749] 存储器控制器55104也可以与错误控制电路55106通过接口连接。错误控制电路55106可以包括当存储器控制器55104读取存储器时检查存储器55102以找到软错误所必需的逻辑。错误控制电路55106可以通过其在读周期期间检查存储器以找到软错误的一种方法是存储在读周期期间可以被检查到的奇偶校验位以确定软错误是否出现。此外或可选地,奇偶校验位可以被存储在存储器55102中。奇偶校验位可以与被奇偶校验位保护的数据(例如数据字节)存储在一起,且一个或更多个奇偶校验位可以存储在连续的存储器中,并且与数据字节相关的一个或更多个奇偶校验位可以在数据字节之前或之后。可选地,一个或更多个奇偶校验位可以存储在与数据字节分开的位置上或者在存储器控制器55104已知的任何布置中。奇偶校验位可以指插入或添加到一串二进制代码的末尾的位,其指示在串中的具有值1的位的数量是偶数还是奇数。

[0750] 作为例子,对于存储在存储器55102中的数据的每个字节,错误控制电路55106可以存储指示存储在字节中的1的数量是偶数还是奇数的奇偶校验位。可选地,奇偶校验位不被存储在错误控制电路55106中,而是被存储在错误控制电路之外,例如在存储器55102处。奇偶校验位可以在字节被写入时被置位。当读取来自存储器55102的数据的字节时,可以对照字节的所读取的值来检查奇偶校验位。如果奇偶校验位与所读取的数据不相关,则可以指示软错误。软错误可以指存储在字节中的位,该字节在其写到存储器中的时间与从存储器被读取的时间之间具有改变的值。

[0751] 照惯例,虽然存储器可以在逐字节基础上被写入,但是它可以从字中被读取。作为例子,存储器的字可能包含数据的32位或数据的4字节。在写周期期间,可以写到存储器的单个字节,并且错误控制电路55106可以确定对应于正被写到的特定字节的奇偶校验位。注意,存储器的字可以包括少于32位或多于32位。图1和图4的总线被标记为32位总线,其仅仅是总线宽度的一个例子。

[0752] 然而,在读周期期间,即使程序只请求读取数据的特定字节,也可以从中读取包含该字节的整个字。一旦整个字被读取,错误控制电路55106就基于所读取的数据计算一个或更多个新的奇偶校验位,并将一个或更多个新的奇偶校验位与之前计算并存储(在存储器55102中,在错误控制电路中或其他地方)的一个或更多个旧的奇偶校验位进行比较。可以对照整个字检查与整个字相关联的奇偶校验位,以确定任何软错误是否出现。如果与该字

相关联的奇偶校验位与在该字中找到的数据不匹配，则存储器控制器55104可以指示软错误已出现。

[0753] 当存储器55102掉电时，它可能无法刷新或存储它在掉电之前保存的数据。照惯例，当存储器从断开或较低功率状态被上电时，存储器可以被初始化，意味着数据被写到每个存储器位置以便给它初始状态。初始化存储器可以有助于确保在涉及存储器的任何读或写周期发生之前与存储器55102中的数据的每个字节相对应的奇偶校验位可以是正确的。

[0754] 图40示出根据本公开的例子的使用存储器初始化的示例性存储器设备功率门控方法。图40的方法5200可以以步骤5202开始，其中存储器55102掉电或被置于低功率状态中。低功率状态(也被称为降低模式)可以是一旦被进入则存储在存储器55102内的数据就被认为是无效的状态。在步骤5204处，存储器55102可以被加电并返回到完全功能(可以在正常模式中操作)。在步骤5206处，可以初始化存储器55102。初始化可以指将数据写到存储器的每个存储器位置的过程。写到存储器55102的数据可以是任意的，并且初始化存储器的过程可以允许存储器的每个字节的奇偶校验位在存储器的任何后续读取或写入之前被最初建立。

[0755] 一旦存储器在步骤5206处被初始化，利用存储器的程序就可以在步骤5208处对存储器的字节执行写命令。在步骤5210处，将数据的字节写到存储器55102的程序可以随后读取在步骤5208处被写入的数据的相同字节。然而，如上面讨论的，不是只读取先前被写入的数据的字节，存储器控制器55104可反而读取字节被存储于其中的数据的整个字。

[0756] 在步骤5212处，可以检查与待读取的数据的字节被存储于其中的数据的字相关联的奇偶校验位，以确定在步骤5208和步骤5210之间是否出现了任何软错误。即使在整个字中只有数据的一个字节被写入，但因为在步骤5206处存储器已被初始化，所以对应于整个字的奇偶校验位应该指示软错误是否出现。

[0757] 通过在上电后初始化存储器，可以确保用于检查软错误的奇偶校验位的准确性。然而，将数据写到每个存储器位置以便初始化存储器的过程可能是耗时的，并且通常可能花费许多时钟周期来使每一个存储器位置被初始化。

[0758] 在希望使用功率门控以最小化当设备空闲时的功率泄漏的设备中，在设备上的存储器可能被频繁地上电和掉电(例如每秒多次)。如果存储器在每次被上电时需要初始化，则由于在存储器的任何后续读取或写入可开始之前存储器必须等待相当多的时间量来初始化的事实，设备可以处理软件指令的速度可能大大降低。

[0759] 最小化初始化存储器所花费的时间的一种方式是完全消除存储器初始化的过程。图3A示出根据本公开的例子的没有存储器初始化的示例性存储器设备功率门控方法。

[0760] 图41示出根据本公开的例子的没有存储器初始化的示例性存储器设备功率门控方法。图41的方法300可以以与关于图40所描述的实质上相同的方式开始。在步骤5302处，存储器设备可以被掉电或者被置于低功率状态中，如上面关于功率门控操作所描述的。在掉电之后的后续时间段处，该方法可以移动到步骤5304，其中存储器设备被加电，然而与关于图40描述的方法相反，存储器在加电时没有被初始化。

[0761] 在步骤5306处，访问存储器的程序可以将数据的一个字节写到存储器设备。在稍后的时间段处，在步骤5308处，程序可以尝试读取在步骤5306处被写入的数据。但是，如上面所讨论的，不是从存储器中只读取数据的字节，存储器控制器可以读取待读取的数据的

字节被包含于其中的整个字。在步骤5310处，可以对照存储在字内的数据来检查在步骤5306处读取的字的奇偶校验位，以确定任何软错误是否出现。

[0762] 然而，由于存储器尚未被重新初始化，未被写入的字的字节可能具有由于设备先前掉电的事实而被破坏的数据。因此，对应于字内的数据的未写入字节的奇偶校验位可能不反映实际上存储在那些字节内的数据。因此，在步骤5310处，即使没有软错误出现于在步骤5306处被写入并从步骤5310处读取的数据的字节上，奇偶校验也可以产生软错误的指示。

[0763] 因此，在不涉及初始化过程的存储器加电过程中，虽然可以减少使存储器在加电时对程序变得可用所花费的时间，但是接收假的软错误指示的可能性可以显著增加。

[0764] 减轻在未被重新初始化的存储器中接收到假阳性软错误指示的可能性的一种方式是仅考虑与待读取的字的字节相关联的奇偶校验位而不是与该字相关联的所有奇偶校验位。以这种方式，虽然与未初始化的数据相关的奇偶校验位可能不反映在未初始化的字节中的数据的状态，但是它们可以被忽略，且因此不产生对软错误的假阳性指示。另外，由于待读取的字节在存储器设备上电之后先前被写入，因此与待读取的数据的字节相关联的奇偶校验位更可能准确地反映字节中的数据的状态，因为写入数据的字节的过程也重置奇偶校验位。

[0765] 因此，为了仅考虑与待读取的字的字节相关联的奇偶校验位，错误校正电路可以知道待读取的数据的特定字节，并且仅考虑与该字节相关联的奇偶校验位。这个知识可以通过使用读掩码被传递给错误校正电路。

[0766] 图41示出根据本公开的例子的具有读掩码功能的示例性存储器设备。在图41中描绘的存储器设备41可以包括存储器模块5402，存储器模块5402以与上面关于图39讨论的存储器55102实质上相同的方式操作。存储器控制器5404可以连接到存储器5402。存储器控制器5404可以包含使存储器5402能够被读取和写入的逻辑，并且还可以包含允许存储器5402被刷新的逻辑。

[0767] 存储器控制器5404也可以与错误控制电路5406通过接口连接。错误控制电路5406可以包括当由存储器控制器5404读取存储器时检查存储器5402以找到软错误所必需的逻辑。错误控制电路5406通过其可以在读周期期间检查存储器以找到软错误的一种方法是存储在读周期期间可以被检查的奇偶校验位以确定软错误是否出现。奇偶校验位可以指插入或添加到一串二进制代码的末尾的位，该位指示在串中的具有值1的位的数量是偶数还是奇数。

[0768] 作为例子，对于存储在存储器5402中的数据的每个字节，错误控制电路5406可以存储指示存储在字节中的1的数量是偶数还是奇数的奇偶校验位。奇偶校验位可以在字节被写入时被置位。当读取来自存储器5402的数据的字节时，可以对照字节的所读取的值来检查奇偶校验位。如果奇偶校验位与所读取的数据不相关，则可以指示软错误。软错误可以指存储在字节中的在其写到存储器中的时间与从存储器读取的时间之间具有改变的值的位。

[0769] 存储器控制器5404还可以与读掩码模块5408通过接口连接。在一个例子中，读掩码模块5408可以作为独立模块来操作，或者可以被包括在存储器5402内。为了说明性目的，读掩码模块5408被示为与存储器控制器5404分离，但是本公开不应该被解释为是限制性

的。

[0770] 读掩码模块5408可以确定在字内的数据的哪个特定字节正被读取，并将该信息提供给错误控制电路5406。利用该知识，错误控制电路可以只考虑与待从中读取的数据的字节相关联的奇偶校验位，而不是如上面讨论的被读取的数据的整个字的奇偶校验位。以这种方式，通过接收数据的哪个特定字节将被从中读取的知识，错误电路可以只考虑与待从中读取的数据的字节相关联的奇偶校验位，并且因此最小化生成软错误的假阳性指示的可能性。

[0771] 图43示出根据本公开的例子的没有存储器初始化和具有读掩码功能的示例性存储器设备功率门控方法。步骤5502、5504、5506和5508可以与图41的它们的对应步骤5302、5304、5306和5308实质上相同，且因此可以关于图41的讨论在上面找到哪些步骤的讨论。

[0772] 步骤5506、5508、5510和5512可以在对存储器加电的步骤5504之后对被写到存储器55102的每个数据字节被执行，可以在步骤5504之后在预定时段期间被执行，可以基于步骤5506、5508、5510和5512的先前迭代的成功或失败被执行，和/或可以基于方法5500的许多迭代的结果(主要是检测到的错误)被执行。可以根据整个存储器、根据存储器的多个存储器组(如果这存在的话)的存储器组、根据存储器的任何其他分段等来确定成功或失败。作为经验法则，更可靠的存储器可以减少经历方法5500的数据字节的数量。

[0773] 在步骤5510处，如上面讨论的存储器控制器或独立读掩码模块可以生成可以被传输到错误控制电路的读掩码。

[0774] 在步骤5512处，错误控制电路可利用读掩码以确定应使用该字的哪些奇偶校验位来确定在存储器模块中是否出现了软错误。使用指示字的哪个字节将被读取的读掩码，错误校正电路可以检查与待读取的数据的字节相关联的奇偶校验位，以确定是否出现软错误而不是查看与如上讨论的字相关联的所有奇偶校验位。使用读掩码来有效地掩码与待读取的数据的字节不相关联的奇偶校验位可以允许假阳性软错误的最小化。通过使用读掩码，存储器可以不需要在存储器设备的加电期间初始化，而同时最小化与对应于尚未被写入的数据的奇偶校验位相关联的假阳性软错误。

[0775] 图44示出根据本发明的实施例的方法5600。

[0776] 方法5600可以通过步骤5610开始，步骤5610在降低功率模式中操作存储器模块。降低功率模式可以是空闲模式、关闭模式或使存储器模块中的数据无效的任何模式。

[0777] 步骤5610之后可以是步骤5620，其在从在降低功率模式中操作存储器模块退出之后在正常功率模式中操作存储器模块。

[0778] 从在降低功率模式中操作存储器模块退出可以涉及向存储器模块供应电力或者将供应给存储器模块的电力增加到将使存储器模块能够存储有效信息的水平。

[0779] 从在降低功率模式中操作存储器模块退出后面不是初始化整个存储器模块。

[0780] 步骤5620之后可以是将第一大小的数据单元写到存储器模块的步骤5630。数据单元被存储在第二大小的存储器分段中。存储器分段可以存储数据单元和附加内容。附加内容可以是一个或更多个数据单元、冗余位和/或垃圾。

[0781] 第二尺寸超过第一尺寸。

[0782] 步骤5630之后可以是接收读命令以从存储器模块读取数据单元的步骤5640。读命令可以由存储器控制器接收。

[0783] 步骤5640之后可以是读取在存储器分段中存储的存储器分段内容的步骤5650。

[0784] 步骤5650之后可以是将存储器分段内容发送到错误控制电路的步骤5660。

[0785] 步骤5660之后可以是将读掩码传输到错误控制电路的步骤5670，其中读掩码指示与数据单元相关的一个或更多个冗余位。读掩码还可以指示忽略哪些冗余位(与存储器分段的其他部分有关)。

[0786] 步骤5670之后可以是由错误控制模块接收存储器分段内容和读掩码的步骤5680。

[0787] 步骤5680之后可以是步骤5690，其基于与数据单元有关的一个或更多个冗余位来检查数据单元中的一个或更多个错误，同时忽略与存储器分段内容的一个或更多个部分(其与数据单元不同)有关的冗余位。

[0788] 步骤5690之后可以是对步骤5690的结果做出响应的步骤56956。例如，如果在步骤5690中发现错误，则执行错误校正步骤，更新错误统计，确定未来的写操作是否将要求错误检测和/或错误校正，等等。

[0789] 步骤5690之后也可以是步骤5630。

[0790] 图45示出根据本发明的实施例的存储器模块5402、存储器控制器5404和各种数据结构。

[0791] 存储器控制器5404和存储器模块5402耦合到错误控制模块(未示出)。

[0792] 存储器模块5402可以包括许多存储器分段(例如存储器行、存储器行的一部分等)。

[0793] 存储器分段可以存储被称为存储器分段内容的内容。

[0794] 图45示出存储在存储器5402的三个存储器分段中的第一、第二和第三存储器分段内容(被表示为“内容”)5731、5731和5732。

[0795] 图45还示出可以存储在存储器控制器5404中的读掩码5408a、5408b和5408c。黑色部分可以表示第一值的位，而白色部分可以表示另一个值的位。

[0796] 第一存储器分段内容5731包括垃圾(无效位)5720、跟随有第一冗余位RB1 5711(与DU1相关联)的第一数据单元(DU1)5701、跟随有第二冗余位RB2 5712(与DU2相关联)的第二数据单元(DU2)5702、以及可能存储被认为保护整个存储器分段的内容的无效存储器分段内容RBC 5718的无效位。

[0797] 第二存储器分段内容5732包括垃圾(无效位)5720、跟随有第三冗余位RB3 5713(与DU3相关联)的第三数据单元(DU3)5703、跟随有第四冗余位RB4 5714(与DU4相关联)的第四数据单元(DU4)5704。

[0798] 第三存储器分段内容5733包括垃圾(无效位)5720、跟随有第六数据单元(DU6)5706的第五数据单元(DU5)5705。第五冗余位RB5 5715(与DU5相关联)和第六冗余位RB6 5716(与DU6相关联)被存储在存储器分段的末尾处。

[0799] 当正在读取第一或第四数据单元时，读掩码5408a可以用于取消掩码RB1 5711或RB4 5714，并且用于掩蔽存储在第一或第三存储器内容中的至少其他冗余位。

[0800] 当正在读取第一或第四数据单元时，读掩码5408b可以用于取消掩码RB1 5711或RB4 5714以及取消掩码DU1或DU4，并用于掩蔽存储在第一或第三存储器内容中的至少其他冗余位。

[0801] 当正在读取第五数据单元时，读掩码5408c可以用于取消掩码RB55715并且用于至

少掩码存储在第一或第三存储器内容中的其他冗余位。

[0802] 可以仅在可以存储冗余位的位置上应用读掩码。

[0803] 图46示出根据本发明的实施例的片上系统。

[0804] 片上系统6700包括互连和服务质量监控器6716、中央处理单元(mCPU)6712、用于管理与外围设备的通信的外围传输管理器6714、多个VMP处理器6720、PMA 6730、MPC 6740、存储器组6750、存储器组6752和介质接口6754、DDR存储器模块6780以及位于DDR存储器模块6780与互连和服务质量监控器6716之间的存储器接口6776和6774。

[0805] 多个VMP处理器6720、PMA 6730、MPC 6740以及可以属于片上系统(或者可以被制造在一个或更多个芯片中)的加速器的非限制性例子。不同类型的处理器可以提供在灵活性和效率之间的不同的折衷。在2016年2月9日提交的美国临时专利序列号62/293,147中说明了PMA 6730的非限制性例子。

[0806] MPC 6740可以包括使用如在本申请的多个附图(例如2、3、6、7、8、9、11、41和43)中所示的桶式线程处理的多个处理核心。

[0807] VMP 6720可以被配置为处理收集读命令,可以包括浮点处理器,可以包括如图28中所描述的FIFO。

[0808] 浮点

[0809] 由上面提到的核心、串行处理器和/或设备中的任一个执行的各种计算可能需要浮点计算。

[0810] 在生成下面列出的异常的同时执行浮点加法和/或减法运算可以极大地简化和减少与加法和/或减法运算相关的硬件。

[0811] 浮点数N等于由底B的E次幂乘以的尾数M:$N=1.M*B^E$。

[0812] 在数字的整数和浮点数表示之间的转换可以通过执行指数偏差来加快。指数偏差涉及操纵偏差,而不是使尾数乘以底的偏差次幂。浮点到整数的转换将包括由加法器将偏差加到指数,以及使尾数乘以底的所偏移的指数次幂,都可以在单个周期中完成。

[0813] 使用指数偏差允许在获得浮点值x和整数N(该整数包括偏差字段)作为它的2个输入的单个操作中从浮点数x产生整数y=convert_float_to_int(x*2^N)。在没有指数偏差的情况下,你需要2个操作:乘法运算z=x*2^N(其中2^N被表示为浮点数),以及然后y=convert_float_to_int(z)。

[0814] 应该注意,convert_float_to_int可以进行截尾、舍入、向上取整(ceil)或向下取整(floor),并且在具有指数偏差的所有这些情况下,你都节省乘法。

[0815] 假设底是2,尾数可以以第一设置位(一个点)开始,且然后是第二设置位。第一个设置位可以是默认位,并可在计算期间被忽略,在计算之后被加到输出尾数,并在计算之前从输入尾数中去除。

[0816] 在浮点表示中,非规格化数是具有太小的绝对值(具有低于预定指数值的指数)的数字。

[0817] 图47示出根据本发明的实施例的包括加法和减法单元6501和格式转换器6502的浮点处理器6500。

[0818] 为了解释的简单,浮点处理器还可以包括未示出的乘法器和/或除法器。

[0819] 浮点处理器6500可以被配置为支持以下指令:加法、减法、乘法、除法、比较、最小

值、最大值、绝对值、在整数和浮点数之间的转换、向下取整、向上取整和舍入到最接近的偶数。

[0820] 加法和减法单元6501接收两个操作数A 6503和B 6504。A由符号A、尾数A和指数A表示。B由符号B、尾数B和指数B表示。注意，字段的顺序可以不同于在这些附图中公开的。例如，字段的顺序可以是符号、指数和尾数。

[0821] 假定是尾数A和/或尾数B由默认设置位开始，那么这个默认设置位被从中去除或忽略。默认设置位可以不包括在尾数中，但可以在它之前。

[0822] 加法和减法单元6501包括异常块6518、用于检查A的指数是否超过B(以及如果是，则检查指数差是什么)的指数比较器(指数A>指数B)6506、初始尾数计算器6508、尾数格式化单元6510、指数选择器6512、符号选择器6516、指数修改器6514和复用器6540。

[0823] 指数比较器6506接收指数A和指数B并且确定指数A是否超过指数B并且输出被馈送到初始尾数计算器6508、指数选择器6512和符号选择器6516的输出信号6507。指数比较器8506块可以是指数差(ExpDiff，指数A-指数B的绝对值)。

[0824] 初始尾数计算器6508接收尾数A、尾数B、输出信号6507、符号A、符号B和加/减操作数。

[0825] 在加上或减去(根据加/减操作数)尾数A和尾数B之前，尾数应该是“对齐的”。例如且假设2的底，较小的指数的尾数移位ExpDiff。

[0826] 在移位之后，初始尾数计算器6508执行加法或减法运算以提供初始尾数结果6520。

[0827] 初始尾数结果6520可以以多个零位或溢出开始。尾数格式化单元6510必须找到“前导”1，并然后通过格式移位6524将尾数加法减法结果6520向右或向左(如果必要)移位，使得在该点之后的第一个位是设置位。

[0828] 指数选择器6512接收指数A和指数B、输出信号6507，并且选择指数A和指数B中的较大的指数作为指数结果6526。

[0829] 格式移位6524被馈送到指数修改器，因为尾数的移位(从而增加尾数)必须通过指数的减小来抵消。指数结果6526必须基于格式移位tShift来减小(向上或向下移动)以提供指数输出6528。

[0830] 符号选择器接收符号A、符号B、加/减操作数和输出信号6507并确定符号输出6532。

[0831] 异常块6518接收操作数A和B(或这些操作数的一部分)并生成以下异常。

[0832] a.如果任何输入变量是非标准化的或是溢出(正或负无穷大)，则声明“非数字”异常(NoN标志)并输出默认数字(诸如，规范的NoN值0xffffffff)。

[0833] b.如果输出数字的绝对值太大(例如，指数输出>0xfe)，则声明溢出异常(溢出标志)并输出默认数字(诸如，0xffffffff)。

[0834] c.如果输出数字太小(非规格化数)，则声明非标准化输出异常，设置零标志(或任何其他标志)并输出默认值(诸如零)。例如，当指数输出等于零且尾数输出不同于零时输出数字太小。

[0835] 应该注意，上面提到的异常中的任一个都可以通过设置或重置任何标志或通过输出任何其他指示符来指示。默认值可以有任何值。例如，默认值可以是零或任何其他预定数

字。可选地，默认数字可以是A和/或B的任何部分或由加法和减法单元6501计算的数字的任何部分。

[0836] 复用器6540由控制信号6537(来自异常块6518)控制，并且输出来自包括尾数输出6522、指数输出6528和符号输出6532的所计算的输出结果的异常块6518的默认值6535。

[0837] 通过声明非规格化数字和太大的输入数字为非数字，加法和减法单元6501被大大简化，因为它不包括管理非规格化数字的电路(诸如，用于找到前导1的复杂和/或冗长的电路)。

[0838] 根据本发明的实施例，当输入浮点数对的至少一个是非规格化数时生成异常，当输入浮点数

[0839] 格式转换器6502被配置为执行整数到浮点转换和浮点到整数转换。

[0840] 当执行整数到浮点转换时，格式转换器可以被配置成：

[0841] a.如果输入变量是正或负无穷大，那么它们被转换为“非数字”，并生成溢出或NoN异常(NoN标志)。输出默认数字(诸如，规范的NoN值0xffffffff)。

[0842] b.如果输入变量太小(非规格化数)，则声明下溢异常并输出默认值，诸如零。

[0843] 当执行整数到浮点转换时，格式转换器可以被配置成：

[0844] a.将无效的输入变量视为Int_max而不考虑指数偏差，并生成无效的输入异常。

[0845] b.无穷大成为INT_MAX，且负无穷大成为INT_MIN，以及当转换为整数异常时产生溢出。

[0846] c.非规格化数不触发异常。

[0847] 格式转换器使用指数偏差。整数可能伴随着偏差字段。当将整数转换为浮点时，偏差在转换后将被添加到浮点数的指数字段。当将浮点转换为偏差时从指数字段的值减小，且这个减小后面是将浮点数(使用减小的指数字段)转换为整数。

[0848] 指数偏差等于将该数字乘以2的指数偏差次幂(指数偏差默认值为零)。

[0849] 一个示例使用案例是将在0和1之间的数字转换为在0和2^8或2^16(即，8位或16位定点数)之间的整数。

[0850] 当将指数偏差加到输入浮点数的指数时出现下面的特殊案例：

[0851] a.指数溢出：如果指数的IEEE表示达到或超过255，则浮点数变为+inf或-inf。

[0852] b.指数下溢：如果指数的IEEE表示变为负数或零，则浮点数将变为非标准化的(但从不为零；注意，当明确表示的输入尾数位全为0时，零可通过简单实现(naive implementation)来产生)。

[0853] c.零输入：+0保持为+0，而-0保持-0(简单实现可产生非零数字。)

[0854] 可以提供一种浮点处理器，其可以包括加法和减法单元以及格式转换器，其中格式转换器被配置为使用指数偏差来将整数转换成浮点数，并且其中加法和减法单元可以包括可以被配置为当接收到作为非规格化数的输入操作数时生成无效异常的异常块。

[0855] 异常块可以被配置为将超过大小阈值的输入变量视为无效数字。

[0856] 浮点处理器可以被配置为通过应用加法或减法运算来计算结果；当结果是非规格化数时生成异常；并输出调到零位的数字和异常标志。

[0857] 浮点处理器可以包括：指数比较器，其可以被配置为比较两个输入操作数的指数；初始尾数计算器，其用于基于两个操作数的指数在两个输入操作数的尾数之间在对齐之后

加上或减去两个输入操作数的尾数以提供初始尾数结果;尾数格式化单元,其用于在初始尾数结果中找到前导设置位,并且用于当前导设置位不是初始尾数结果的最高有效位时移位初始尾数结果,使得前导设置位是结果的尾数的最高有效位。

[0858] 浮点处理器可以包括:指数选择器,其用于在两个输入操作数的指数之间进行选择以提供选定指数;指数修改器,其用于修改选定指数以便补偿初始尾数结果的移位以提供结果指数;以及用于确定结果符号的符号选择器。

[0859] 格式转换器可以被配置为当浮点数是非规格化数时将浮点数转换为整数而不生成异常。

[0860] 可以提供用于由浮点处理器执行浮点操作的方法(图49的方法6900),其中该方法可以包括由加法和减法单元加上或减去(6910)浮点格式的两个输入操作数;其中加上或减去可以包括:当两个输入操作数中的至少一个是非规格化数时生成(6920)无效数字异常;其中,非数字异常的生成可以包括输出默认值并设置异常标志;以及使用指数偏差来将整数转换(6930)为浮点数。

[0861] 步骤6910可以包括由异常块将超过大小阈值的输入变量视为无效数字。

[0862] 步骤6910可以包括通过应用加法或减法运算来计算结果;当结果是非规格化数时生成异常并输出调到零位的数字和异常标志。

[0863] 步骤6910可以包括由指数比较器比较两个输入操作数的指数;由初始尾数计算器基于两个操作数的指数在两个输入操作数的尾数之间在对齐之后加上或减去两个输入操作数的尾数以提供初始尾数结果;通过尾数格式化在初始尾数结果中找到前导设置位进行初始尾数结果的尾数格式化,以及当前导设置位不是初始尾数结果的最高有效位时移位初始尾数结果,使得前导设置位是输出尾数的最高有效位。

[0864] 步骤6910可以包括由指数选择器在两个输入操作数的指数之间进行选择以提供选定指数;由指数修改器修改选定指数以便补偿初始尾数结果的移位以提供输出指数;以及由符号选择器确定输出信号的符号。

[0865] 步骤6940可以包括:当浮点数是非规格化数时,由格式转换器将浮点数转换为整数而不生成异常。

[0866] 可以提供一种非临时计算机可读存储介质,其上存储有指令集,指令集用于由浮点处理器执行浮点操作,其包括通过加法和减法单元加上或减去浮点格式的两个输入操作数;其中加上或减去可以包括:当两个输入操作数中的至少一个是非规格化数时生成无效数字异常;其中,非数字异常的生成可以包括输出默认值并设置异常标志;以及使用指数偏差来将整数转换为浮点数。

[0867] 虽然已经参考附图充分描述了所公开的例子,但是应该注意,各种变化和修改将对本领域的技术人员变得明显。这样的变化和修改应被理解为被包括在如由所附权利要求限定的所公开的示例的范围内。

[0868] 可以提供在说明书中公开的任何方法的任何步骤的任何组合。

[0869] 可以提供权利要求中公开的任何方法的任何步骤的任何组合。

[0870] 可以提供在说明书中公开的任何设备、系统、处理器的任何部件、元件的任何组合。

[0871] 可以提供权利要求中公开的任何设备、系统、处理器的任何部件、元件的任何组

合。

[0872] 可以提供在说明书中公开的任何非临时计算机可读存储介质中存储的任何指令的任何步骤的任何组合。

[0873] 对术语“包括(comprise)”、“包括(comprises)”、“包括(comprising)”、“包括(including)”、“可包括(may include)”以及、“包括(includes)”中的任何一个的任何提及可以适用于术语“由...组成(consists)”、“由...组成(consisting)”和“基本上由...组成”。例如,描述步骤的任何方法都可以包括比附图中示出的步骤更多的步骤、仅仅在附图中所示的步骤或者实质上仅仅在附图中所示的步骤。同理适用于设备、处理器或系统的部件以及存储在任何非临时计算机可读存储介质中的指令。

[0874] 本发明也可以在用于在计算机系统上运行的计算机程序中实施,该计算机程序至少包括用于在可编程装置(例如计算机系统)上运行时执行根据本发明的方法的步骤、或者使得可编程装置能够执行根据本发明的设备或系统的功能的代码部分。计算机程序可以使存储系统将硬盘驱动器分配给硬盘驱动器组。

[0875] 计算机程序是例如特定应用程序和/或操作系统的指令列表。计算机程序可以例如包括以下各项中的一个或更多个:子例程、函数、过程、对象方法、对象实现、可执行应用、小应用程序、小服务程序、源代码、目标代码、共享库/动态加载库和/或被设计用于在计算机系统上执行的其它指令序列。

[0876] 计算机程序可以内部地存储于非临时计算机可读介质上。所有或一些的计算机程序可以设置在永久地、可移除地或远程地耦合到信息处理系统的计算机可读介质上。计算机可读介质可以包括例如且没有限制地任何数量的下列项:磁性存储介质,包括硬盘和磁带存储介质;光学存储介质,例如光盘介质(例如,CD-ROM、CD-R等)和数字视频盘存储介质;非易失性存储器存储介质,包括基于半导体的存储器单元,例如快闪存储器、EEPROM、EPROM、ROM;铁磁数字存储器;MRAM;易失性存储介质,包括寄存器、缓冲器或高速缓存、主存储器、RAM等。

[0877] 计算机过程通常包括执行(运行)程序或程序的一部分、当前程序值和状态信息以及由操作系统用以管理过程的执行的资源。操作系统(OS)是管理计算机资源共享、并为程序员提供用于访问这些资源的接口的软件。操作系统处理系统数据和用户输入,并通过分配和管理作为对系统的用户和程序的服务的任务和内部系统资源进行响应。

[0878] 计算机系统可以例如包括至少一个处理单元、相关联的存储器和多个输入/输出(I/O)设备。当执行计算机程序时,计算机系统根据计算机程序处理信息,并且经由I/O设备产生所得到的输出信息。

[0879] 在前述说明书中,已经参考本发明的实施例的具体示例描述了本发明。然而,显然,在不脱离如所附权利要求中阐述的本发明的更广泛的精神和范围的情况下,可以在其中进行各种修改和改变。

[0880] 此外,在说明书中和权利要求中的术语“前”、“后”、“顶部”、“底部”、“在…之上”、“在…之下”等(如果有的话)用于描述性目的,而不一定用于描述永久相对位置。应当理解,这样使用的术语在适当的情况下是可互换的,使得本文所描述的本发明的实施例例如能够在除了本文所示出或另外描述的那些方向之外的其它方向上操作。

[0881] 如在本文讨论的连接可以是适合于例如经由中间设备从相应的节点、单元或设备

传送信号或者将信号传送到相应的节点、单元或设备的任何类型的连接。因此，除非另外暗示或规定，否则连接可以例如是直接连接或间接连接。可以参考单个连接、多个连接、单向连接或双向连接来示出或描述连接。然而，不同的实施例可以改变连接的实现。例如，可以使用单独的单向连接而不是双向连接，反之亦然。而且，多个连接可以用连续地或以时间复用方式传送多个信号的单个连接代替。同样，携带多个信号的单个连接可以被分离成携带这些信号的子集的各种不同的连接。因此，用于传送信号的很多选择存在。

[0882] 虽然在例子中描述了电位的特定的导电类型或极性，但是应该认识到，电位的导电类型和极性可以反转。

[0883] 本文描述的每个信号可以被设计为正逻辑或负逻辑。在负逻辑信号的情况下，在逻辑真状态对应于逻辑电平零的情况下，信号是低电平有效的。在正逻辑信号的情况下，在逻辑真状态对应于逻辑电平一的情况下，信号是高电平有效的。注意，本文描述的任何信号可以被设计为负逻辑信号或正逻辑信号。因此，在可选的实施例中，被描述为正逻辑信号的那些信号可以被实现为负逻辑信号，以及被描述为负逻辑信号的那些信号可以被实现为正逻辑信号。

[0884] 此外，当提到信号、状态位或类似装置分别转化成它的逻辑真或逻辑假状态时，本文使用术语“断言”或“置位”和“否定”(或“解除断言”或“清除”)。如果逻辑真状态是逻辑电平一，则逻辑假状态是逻辑电平零。而且如果逻辑真状态是逻辑电平零，则逻辑假状态是逻辑电平一。

[0885] 本领域技术人员将认识到，逻辑块之间的边界仅仅是说明性的，并且可选实施例可以合并逻辑块或电路元件，或者对各种逻辑块或电路元件施加功能的替代分解。因此，应当理解，本文所描述的体系结构仅仅是示例性的，并且实际上可以实施实现相同功能的许多其它体系结构。

[0886] 实现相同功能的部件的任何布置被有效地“关联”，使得实现期望的功能。因此，本文组合以实现特定功能的任何两个部件可以被看作彼此“相关联”，使得实现期望的功能，而与体系结构或中间部件无关。同样，这样关联的任何两个部件也可以被视为彼此“可操作地连接”或“可操作地耦合”以实现期望的功能。

[0887] 此外，本领域技术人员将认识到上述操作之间的边界仅是说明性的。多个操作可以组合成单个操作，单个操作可以分布在附加操作中，并且操作可以在时间上至少部分地重叠地执行。此外，可选实施例可以包括特定操作的多个实例，并且在各种其它实施例中可以改变操作的顺序。

[0888] 还例如，在一个实施例中，所示示例可以被实施为位于单个集成电路上或在同一设备内的电路。另选地，该示例可实施为以合适方式彼此互连的任何数目的单独集成电路或单独设备。

[0889] 还例如，示例或其部分可以实施为物理电路的或者可转换成物理电路的逻辑表示的软或代码表示，例如以任何适当类型的硬件描述语言实施。

[0890] 此外，本发明不限于在非可编程硬件中实现的物理设备或单元，但也还可以应用在能够通过根据适当的程序代码进行操作来执行期望的设备功能的可编程设备或单元，诸如大型机、小型计算机、服务器、工作站、个人计算机、记事本、个人数字助理、电子游戏、汽车和其他嵌入式系统、蜂窝电话和在本申请中通常被表示为“计算机系统”的各种其他无线

设备中。

[0891] 然而，其它修改、变化和替代也是可能的。因此，说明书和附图被认为是说明性的而不是限制性的。

[0892] 在权利要求中，置于括号之间的任何附图标记不应被解释为限制权利要求。词“包括”不排除除了在权利要求中列出的那些之外的其他元件或步骤的存在。此外，如本文所使用的术语“一(a)”或“一(an)”被定义为一个或多于一个。此外，在权利要求中的引导性短语(例如“至少一个”和“一个或更多个”)的使用不应被解释为暗示由不定冠词“a”或“an”引入另一个权利要求要素将包含这样引入的权利要求要素的任何特定权利要求限制到仅包含一个这样的要素的发明，即使同一权利要求包括引导性短语“一个或更多个”或“至少一个”和不定冠词(例如“a”或“an”)。同理适用于定冠词的使用。除非另有说明，否则诸如“第一”和“第二”的术语用于任意地区分开这样的术语所描述的要素。因此，这些术语不一定旨在指示这样的要素的时间或其他优先级。某些度量在相互不同的权利要求中被叙述的不争事实并不指示这些度量的组合不能有利地被使用。

[0893] 虽然本文已经图示和描述了本发明的某些特征，但是本领域普通技术人员将想到许多修改、替换、改变和等同物。因此，应当理解，所附权利要求旨在覆盖落入本发明的真实精神内的所有这样的修改和改变。

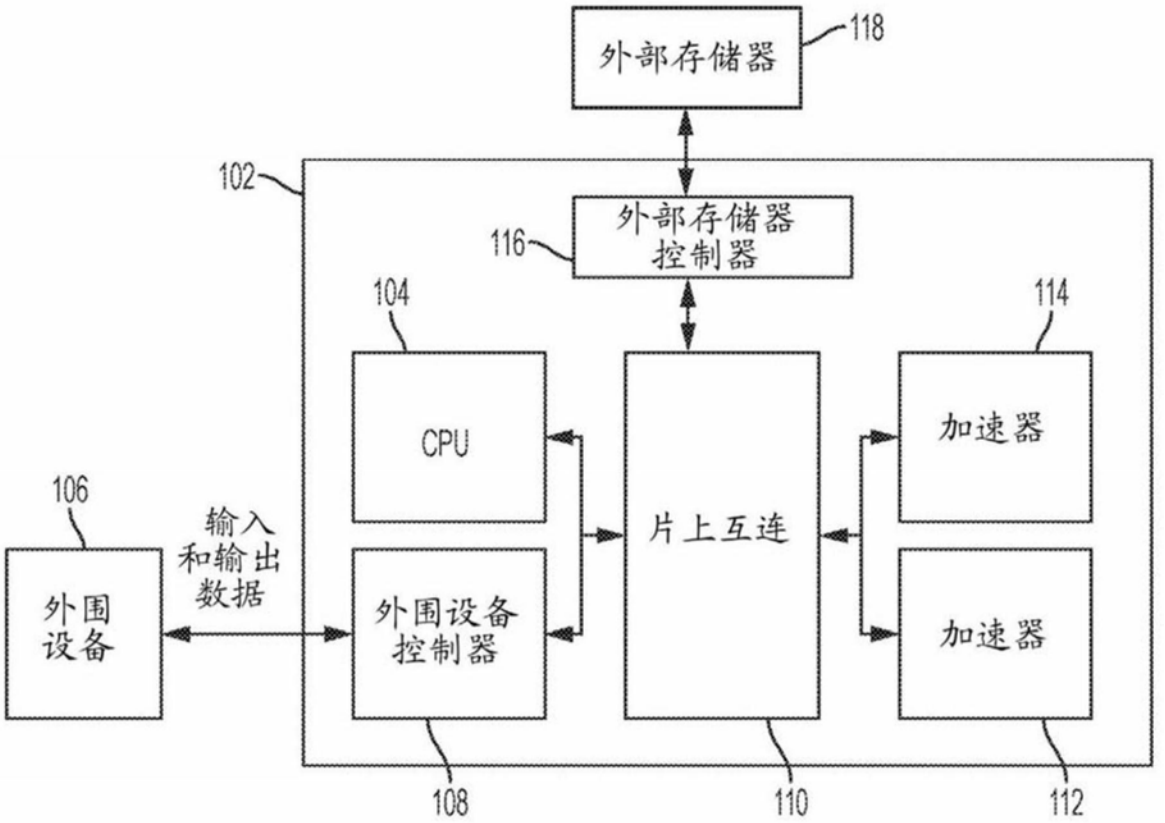

外部存储器
118
102
外部存储器
控制器
116
104
114
CPU
加速器
106
输入
和输出
数据
片上互连
外围
设备
外围设备
控制器
加速器
108
110
112

图1

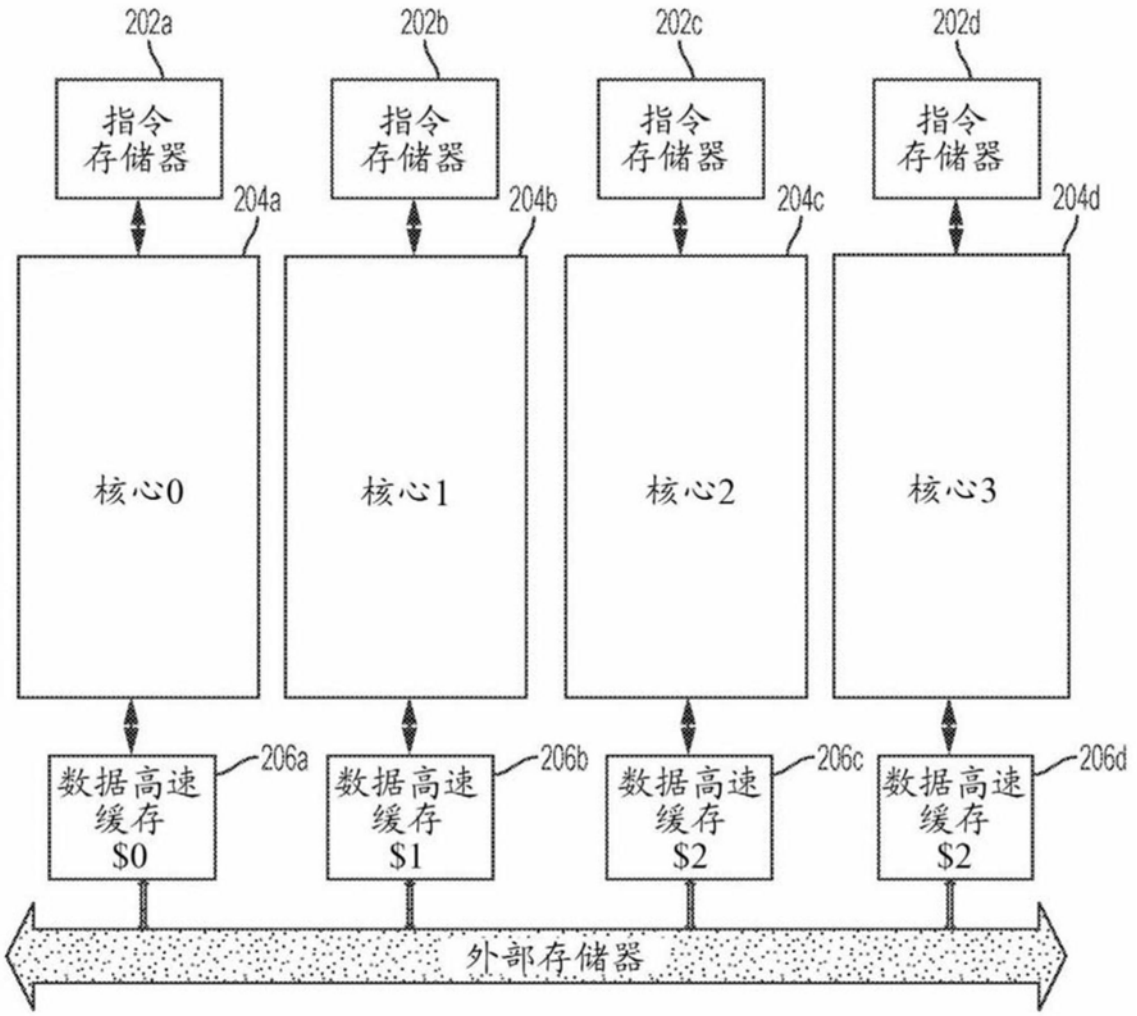

202a
202b
202c
202d
指令存储器
指令存储器
指令存储器
指令存储器
204a
204b
204c
204d
核心0
核心1
核心2
核心3
206a
206b
206c
206d
数据高速缓存 $0
数据高速缓存 $1
数据高速缓存 $2
数据高速缓存 $2
外部存储器

图2

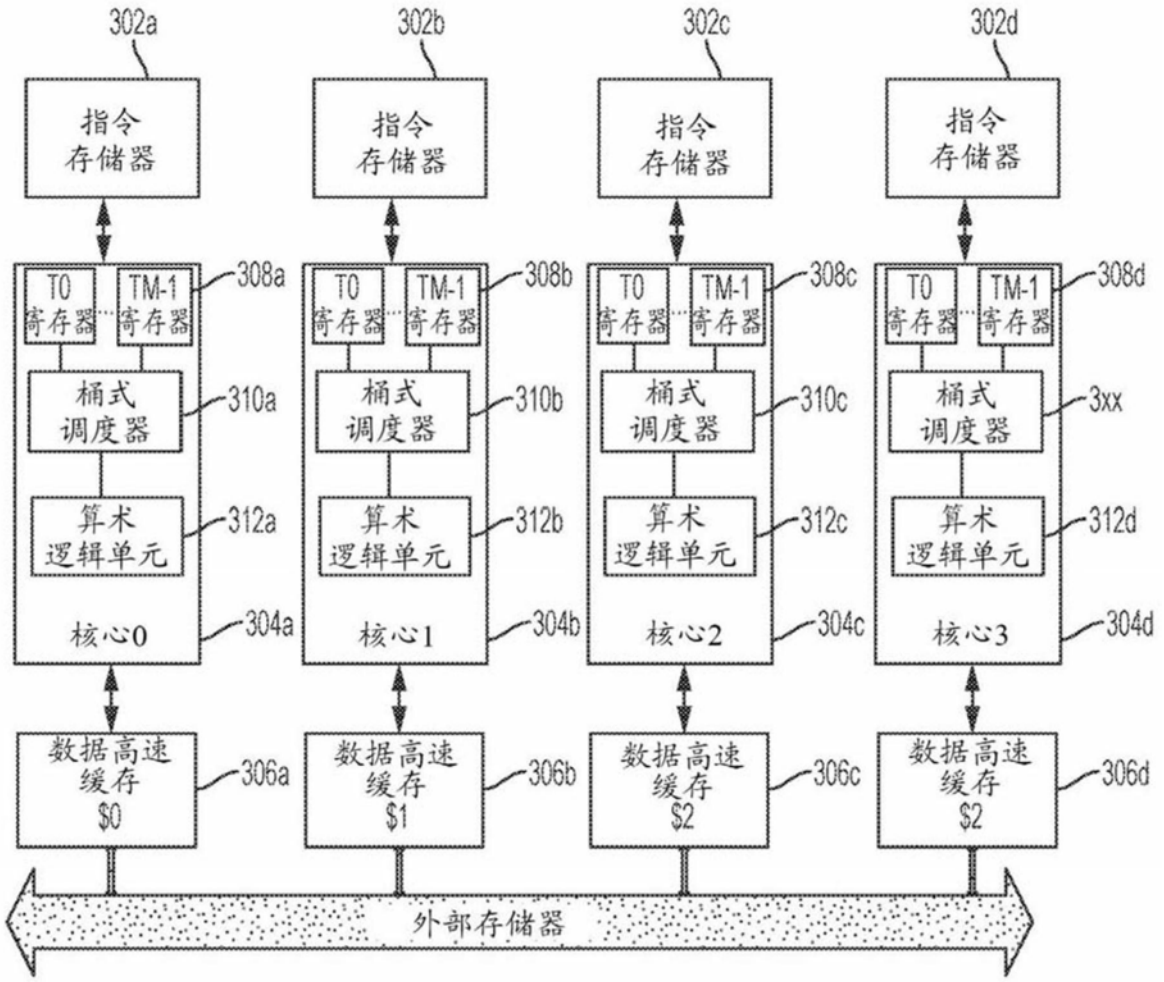

302a
302b
302c
302d
指令
存储器
T0
寄存器
TM-1
寄存器
308a
308b
308c
308d
桶式
调度器
310a
310b
310c
3xx
算术
逻辑单元
312a
312b
312c
312d
核心0
核心1
核心2
核心3
304a
304b
304c
304d
数据高速
缓存
$0
$1
$2
306a
306b
306c
306d
外部存储器

图3A

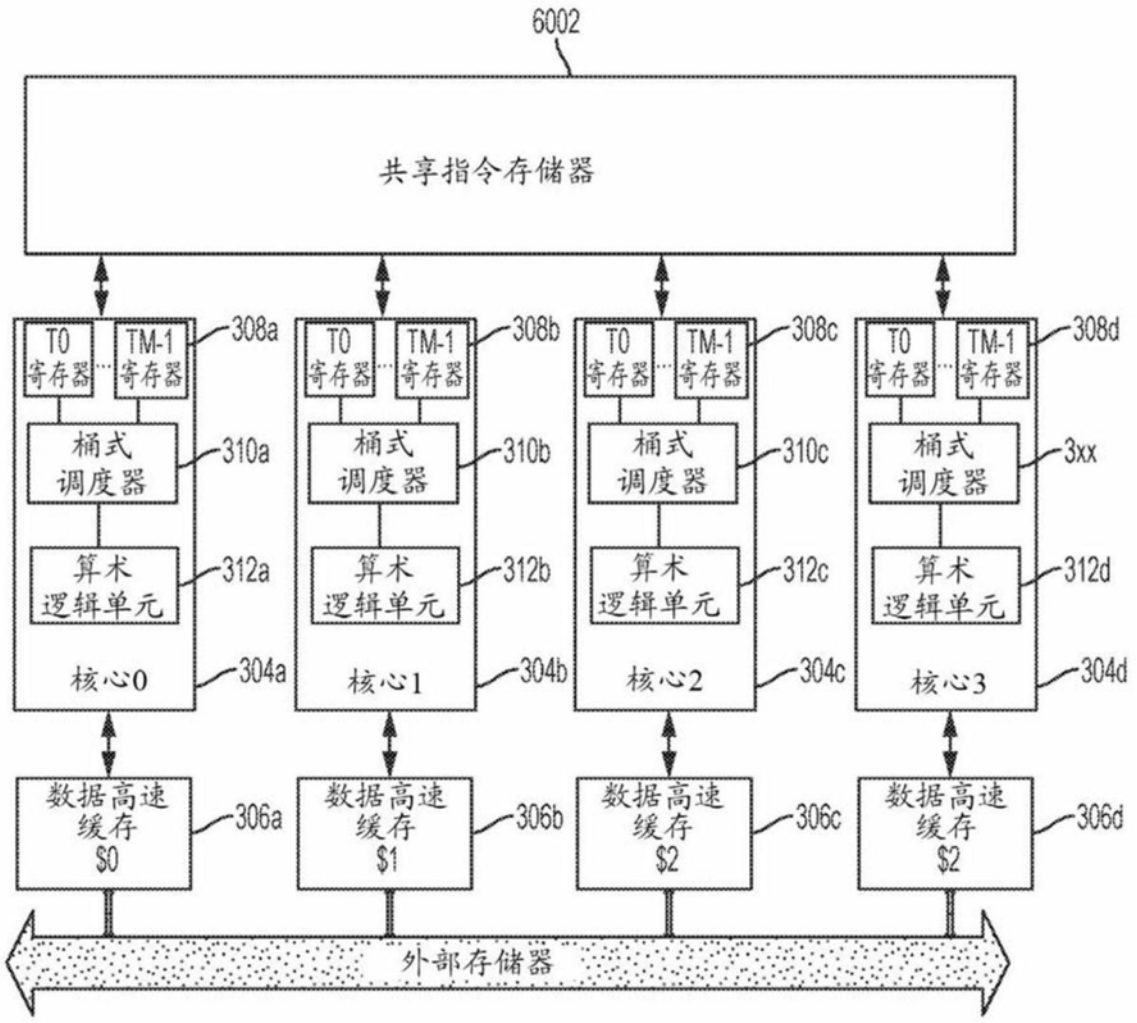

6002
共享指令存储器
T0 寄存器
TM-1 寄存器
308a
308b
308c
308d
桶式调度器
310a
310b
310c
3xx
算术逻辑单元
312a
312b
312c
312d
核心0
核心1
核心2
核心3
304a
304b
304c
304d
数据高速缓存 $0
数据高速缓存 $1
数据高速缓存 $2
数据高速缓存 $2
306a
306b
306c
306d
外部存储器

图3B

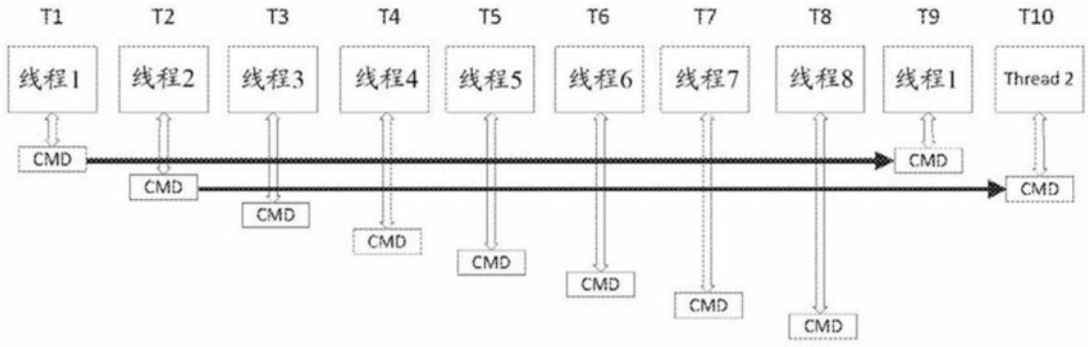

T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
线程1
线程2
线程3
线程4
线程5
线程6
线程7
线程8
线程1
Thread 2
CMD

图4

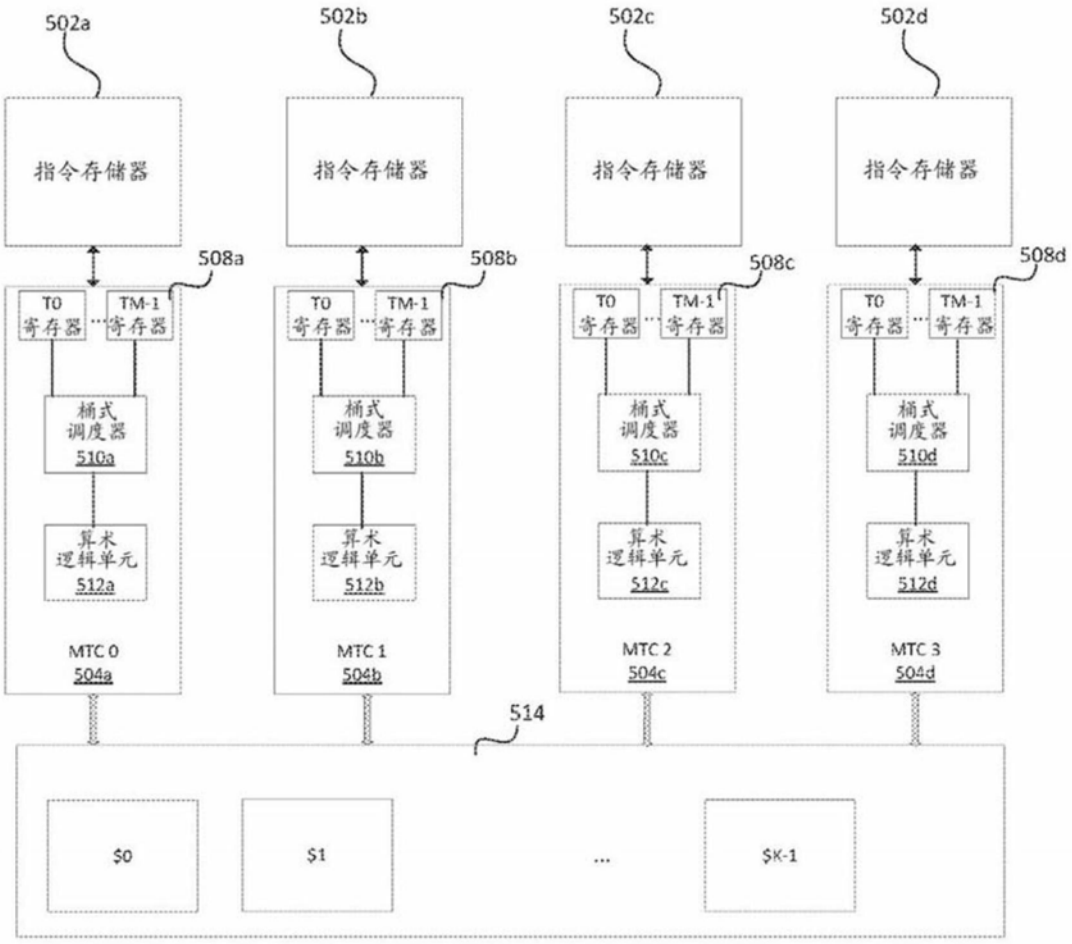

502a
502b
502c
502d
指令存储器
指令存储器
指令存储器
指令存储器
508a
508b
508c
508d
T0 寄存器
TM-1 寄存器
桶式调度器 510a
桶式调度器 510b
桶式调度器 510c
桶式调度器 510d
算术逻辑单元 512a
算术逻辑单元 512b
算术逻辑单元 512c
算术逻辑单元 512d
MTC 0 504a
MTC 1 504b
MTC 2 504c
MTC 3 504d
514
$0
$1
...
$K-1

图5A

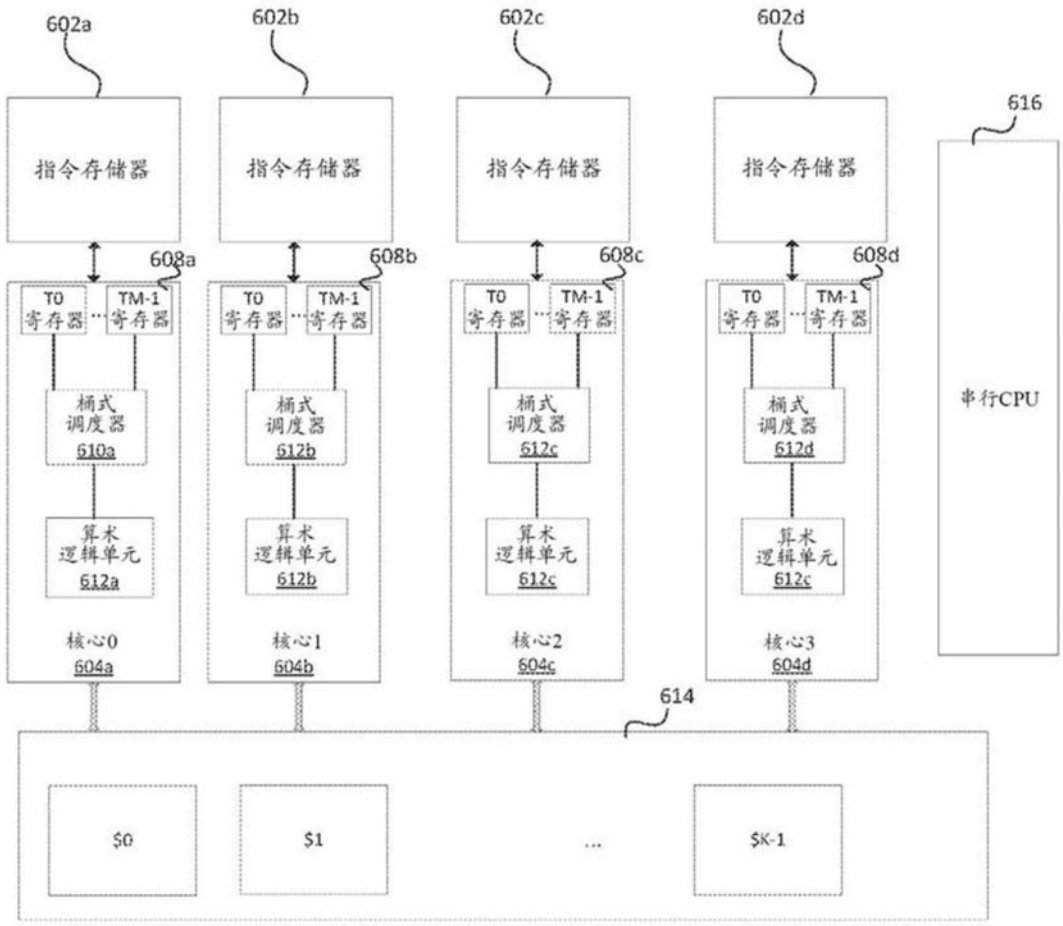

602a
602b
602c
602d
616
指令存储器
指令存储器
指令存储器
指令存储器
608a
608b
608c
608d
T0 寄存器
TM-1 寄存器
桶式 调度器 610a
桶式 调度器 612b
桶式 调度器 612c
桶式 调度器 612d
算术 逻辑单元 612a
算术 逻辑单元 612b
算术 逻辑单元 612c
算术 逻辑单元 612c
串行CPU
核心0 604a
核心1 604b
核心2 604c
核心3 604d
614
$0
$1
...
$K-1

图6A

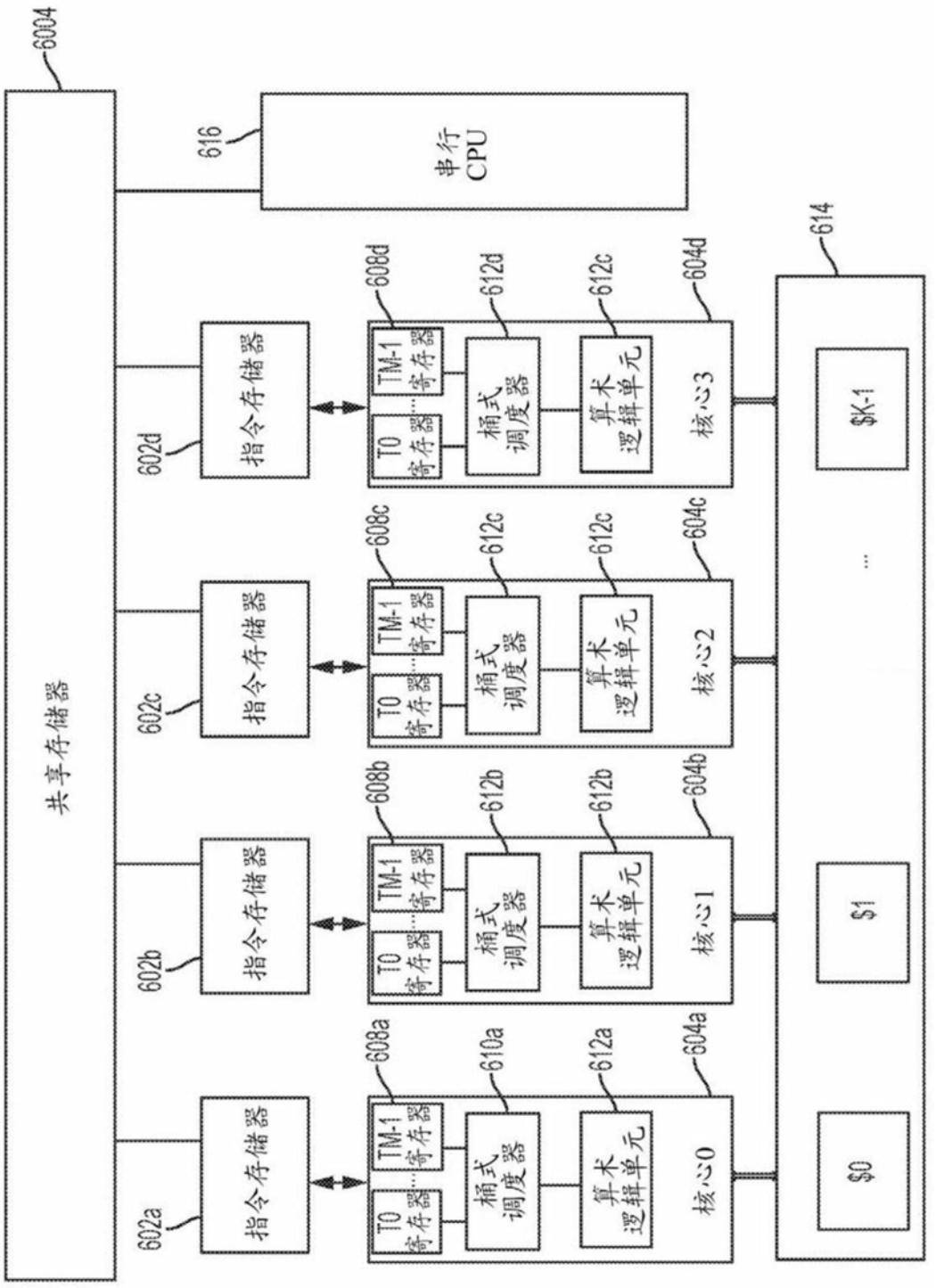

6004
共享存储器
616
串行CPU
614
602a
602b
602c
602d
指令存储器
608a
608b
608c
608d
TM-1寄存器
T0寄存器
610a
桶式调度器
612a
612b
612c
612d
算术逻辑单元
604a
604b
604c
604d
核心0
核心1
核心2
核心3
$0
$1
$K-1

图6B

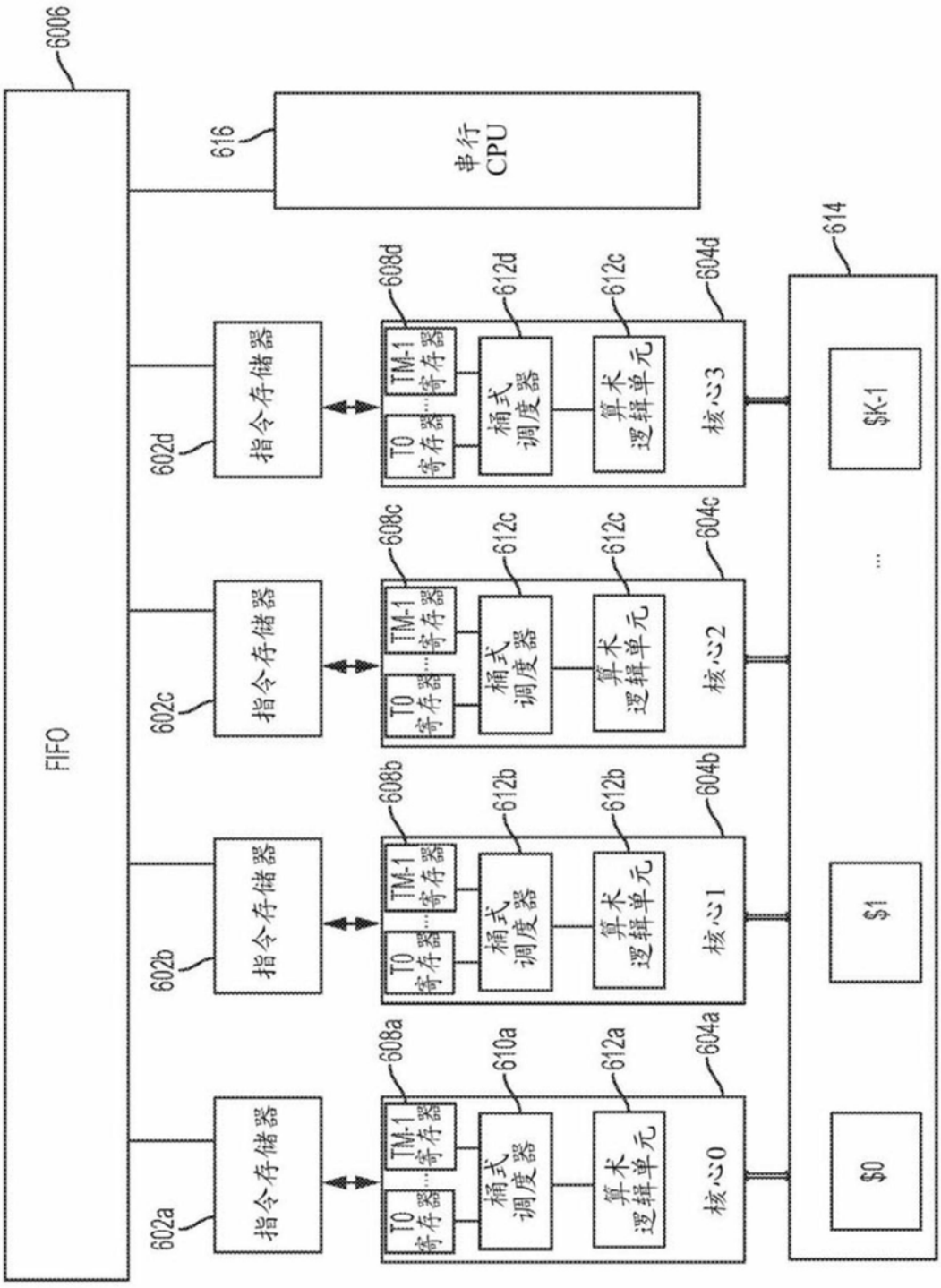

6006
FIFO
616
串行CPU
614
602a
602b
602c
602d
指令存储器
608a
608b
608c
608d
TM-1寄存器
TO寄存器
610a
桶式调度器
612a
612b
612c
612d
算术逻辑单元
604a
604b
604c
604d
核心0
核心1
核心2
核心3
$0
$1
$K-1

图6C

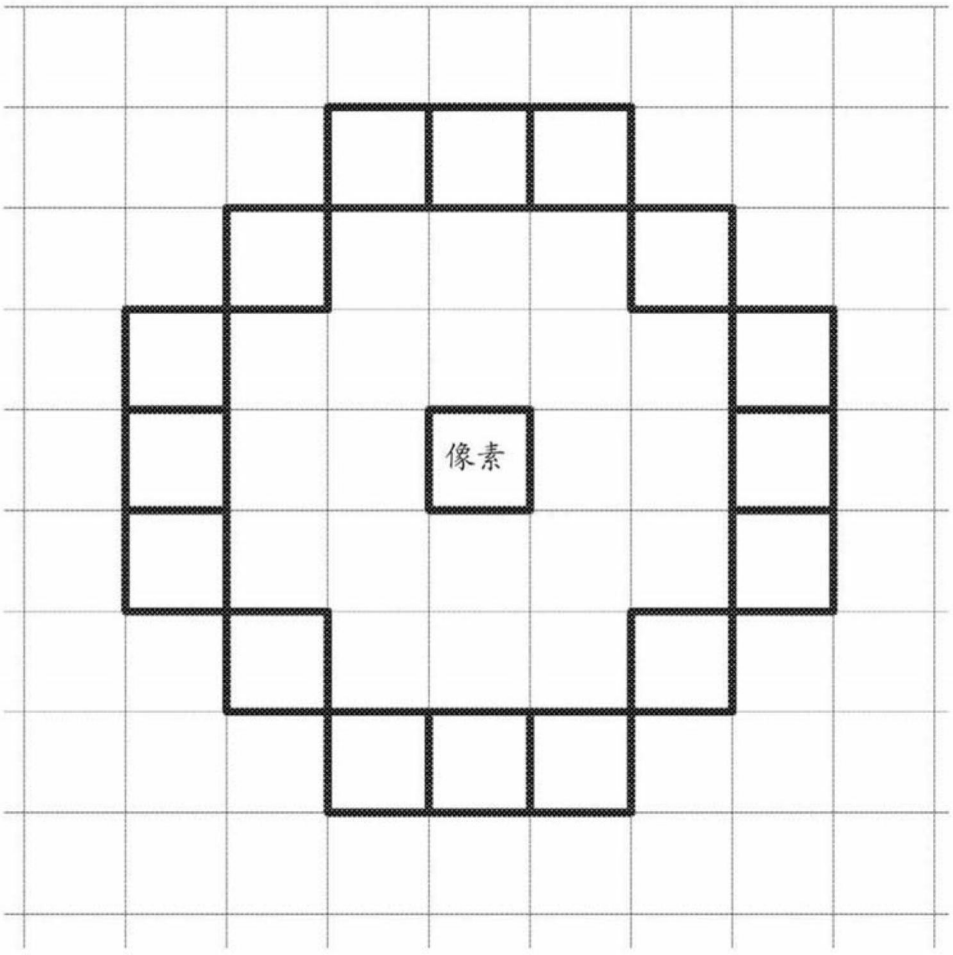


图6D

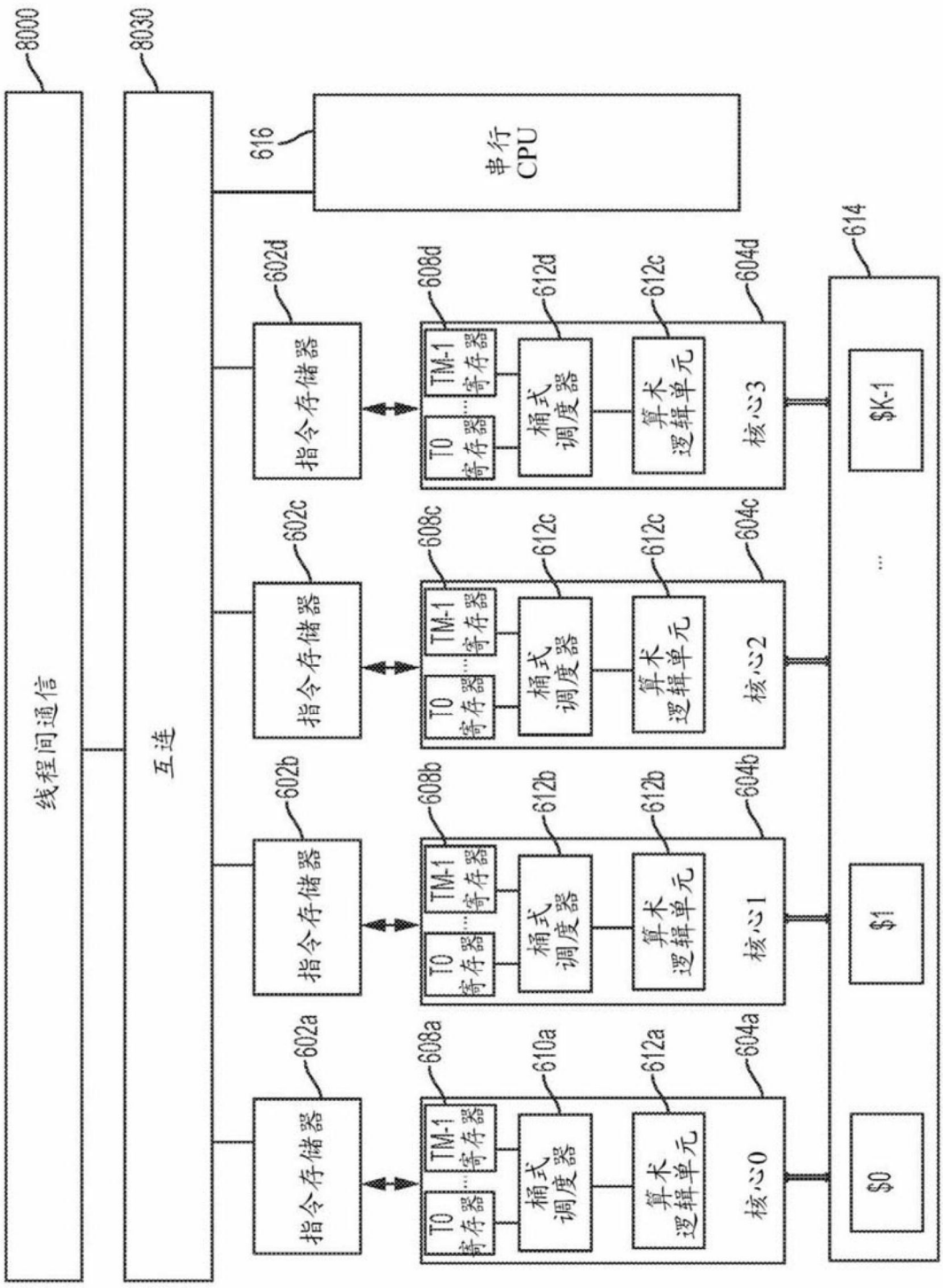

8000
8030
线程间通信
互连
616
串行CPU
602d
指令存储器
608d
TM-1寄存器
TO寄存器
桶式调度器
612d
算术逻辑单元
612c
604d
核心3
602c
指令存储器
608c
TM-1寄存器
TO寄存器
桶式调度器
612c
算术逻辑单元
612c
604c
核心2
602b
指令存储器
608b
TM-1寄存器
TO寄存器
桶式调度器
612b
算术逻辑单元
612b
604b
核心1
602a
指令存储器
608a
TM-1寄存器
TO寄存器
桶式调度器
610a
算术逻辑单元
612a
604a
核心0
614
$K-1
...
$1
$0

图6E

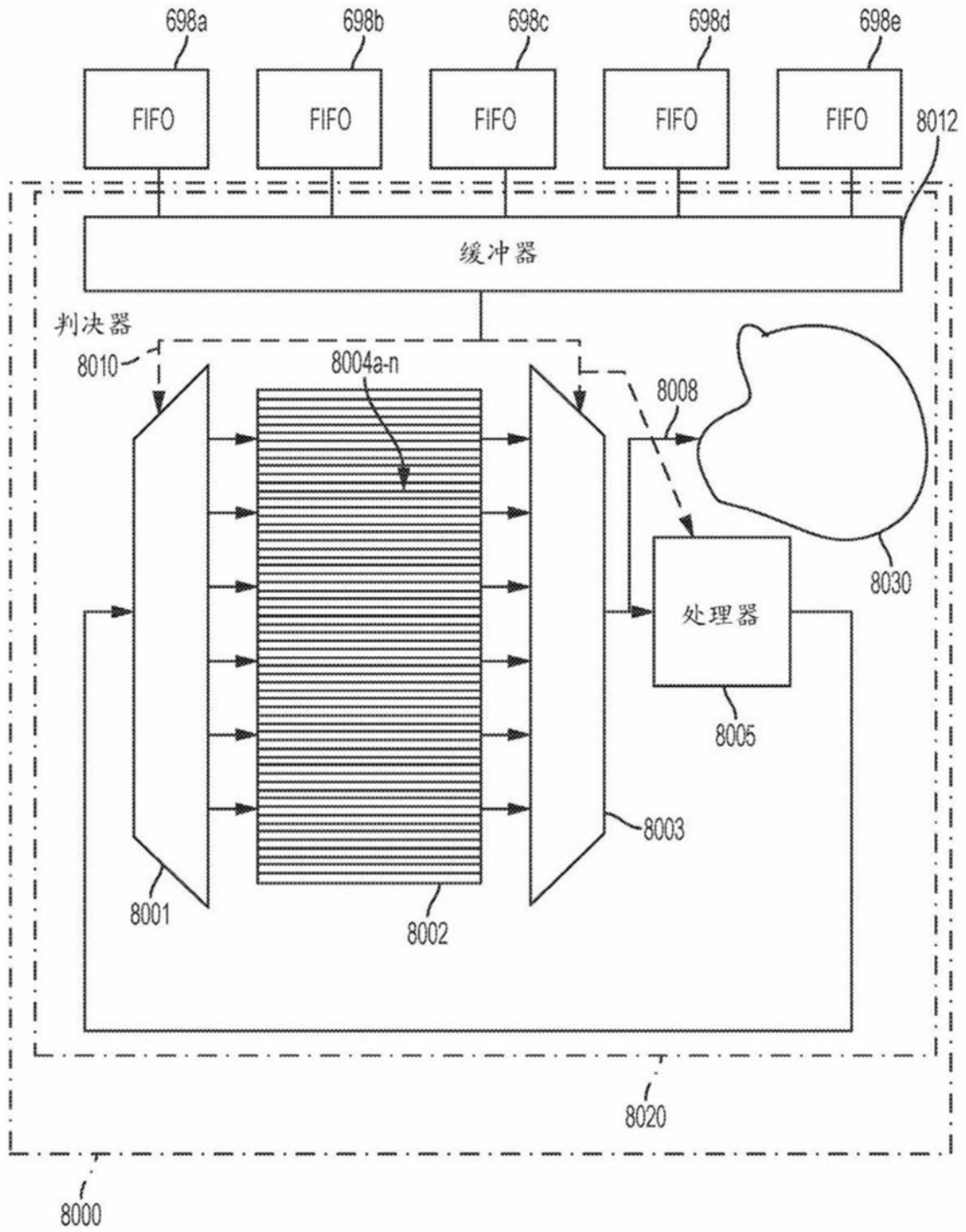

698a
698b
698c
698d
698e
FIFO
FIFO
FIFO
FIFO
FIFO
8012
缓冲器
判决器
8010
8004a-n
8008
8030
处理器
8005
8003
8001
8002
8020
8000

图6F

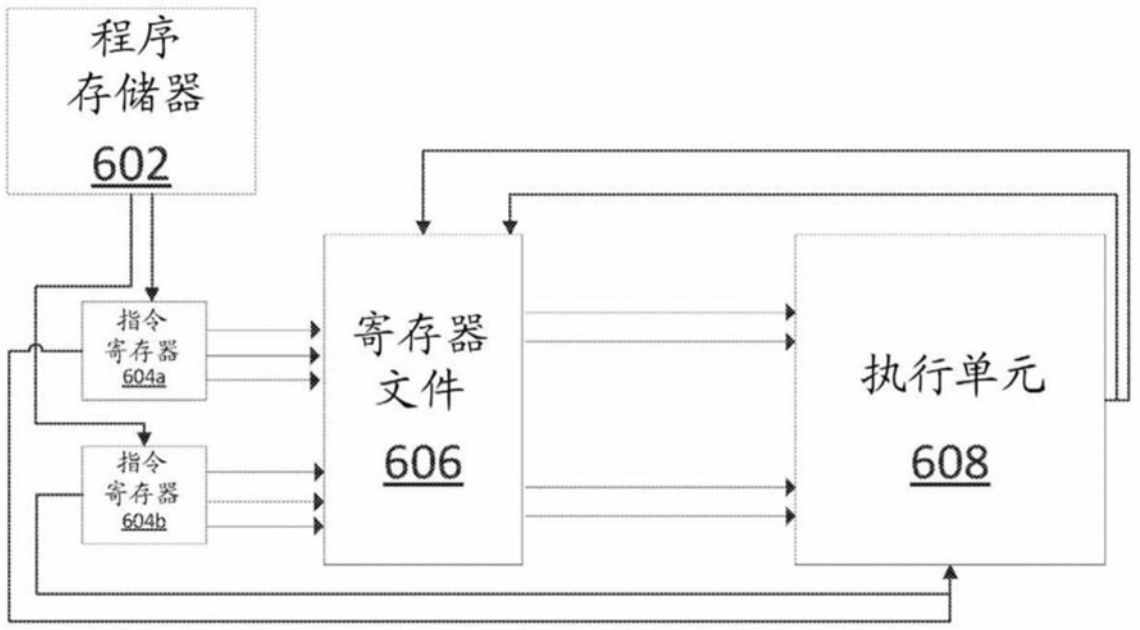


图7

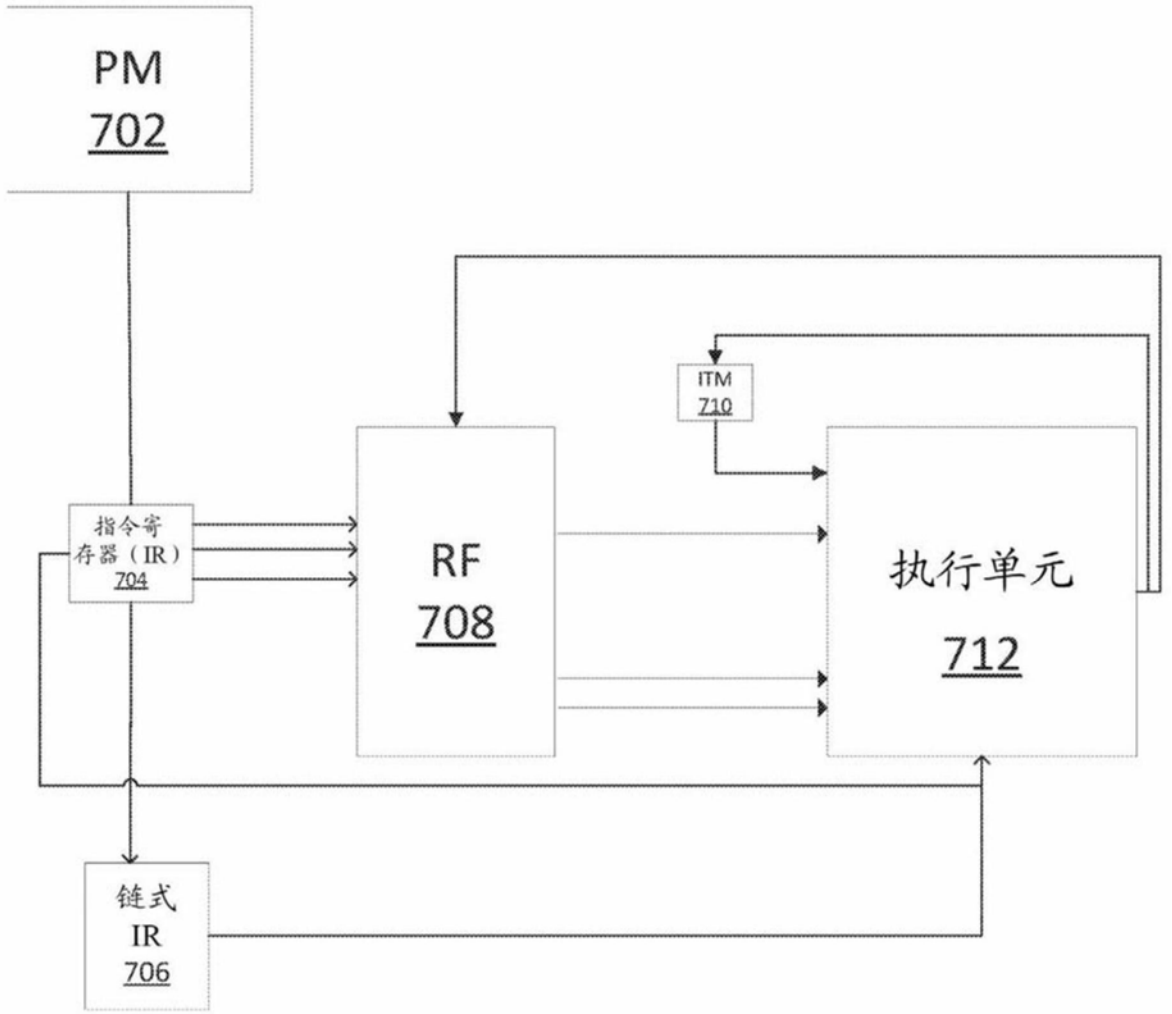


图8

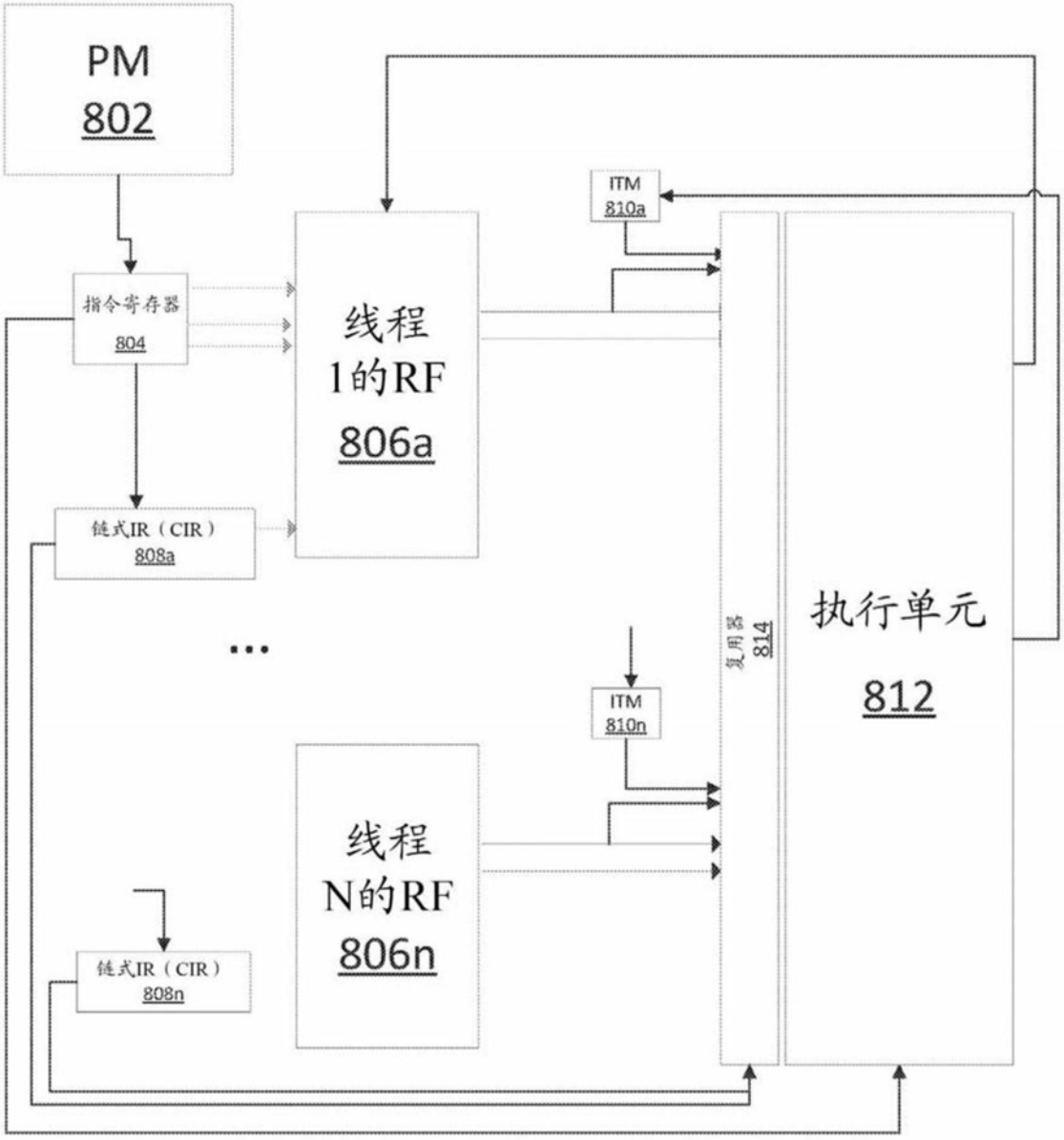

PM
802
指令寄存器
804
链式IR（CIR）
808a
线程
1的RF
806a
ITM
810a
复用器
814
执行单元
812
ITM
810n
线程
N的RF
806n
链式IR（CIR）
808n

图9

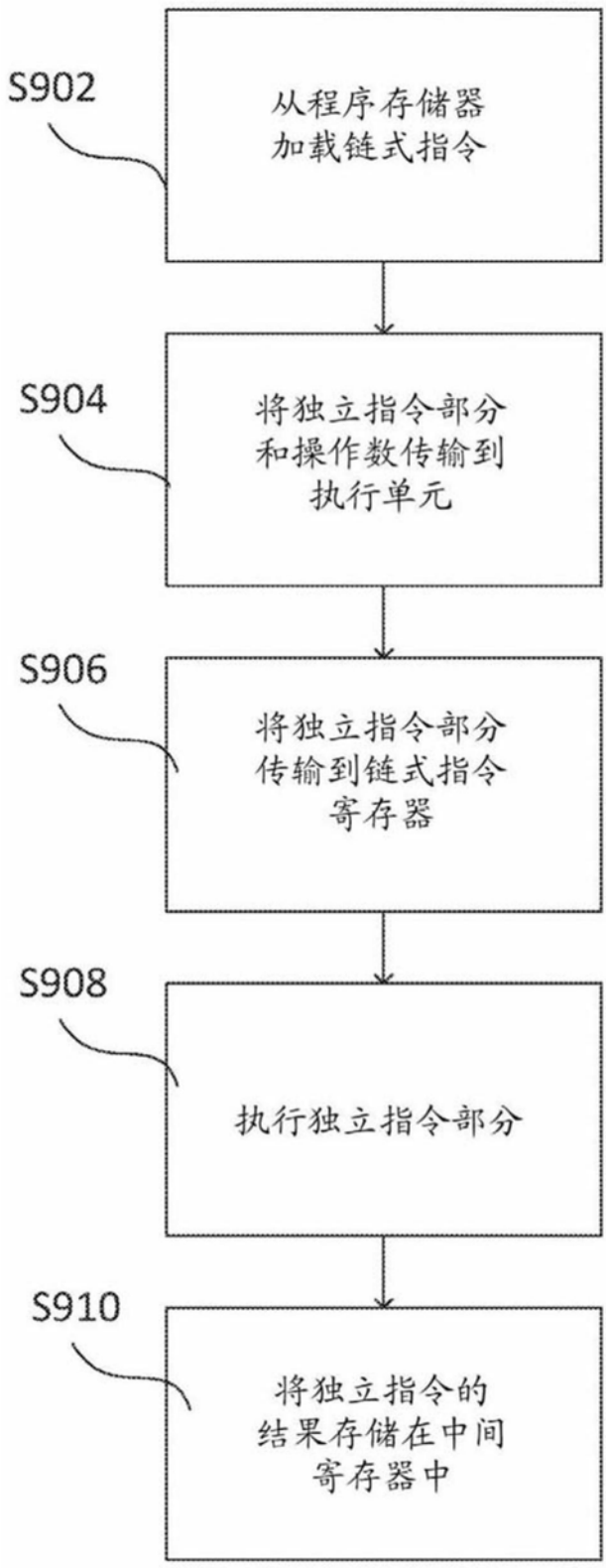

S902
从程序存储器
加载链式指令
S904
将独立指令部分
和操作数传输到
执行单元
S906
将独立指令部分
传输到链式指令
寄存器
S908
执行独立指令部分
S910
将独立指令的
结果存储在中间
寄存器中

图10A

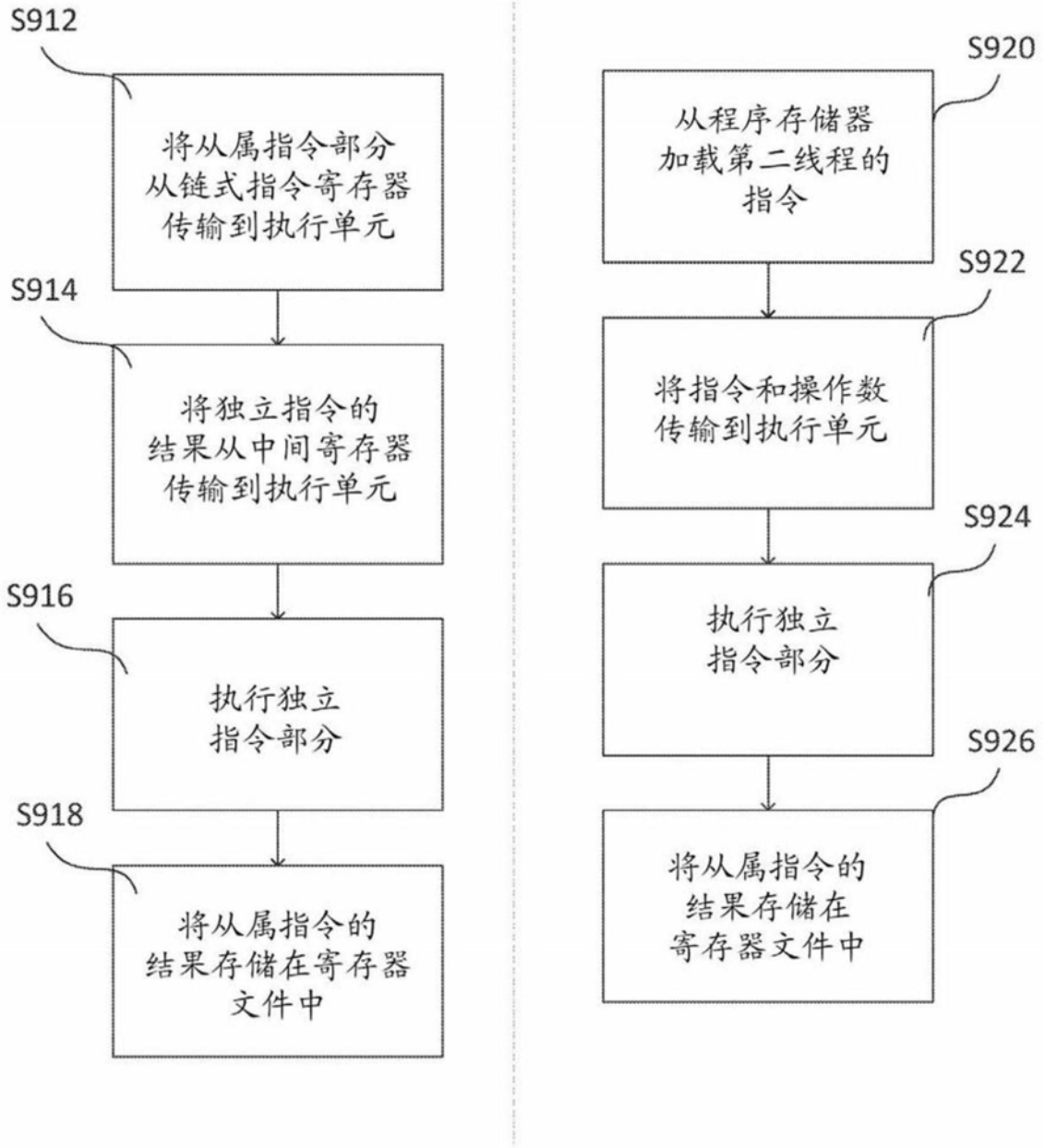

S912
将从属指令部分
从链式指令寄存器
传输到执行单元
S914
将独立指令的
结果从中间寄存器
传输到执行单元
S916
执行独立
指令部分
S918
将从属指令的
结果存储在寄存器
文件中
S920
从程序存储器
加载第二线程的
指令
S922
将指令和操作数
传输到执行单元
S924
执行独立
指令部分
S926
将从属指令的
结果存储在
寄存器文件中

图10B

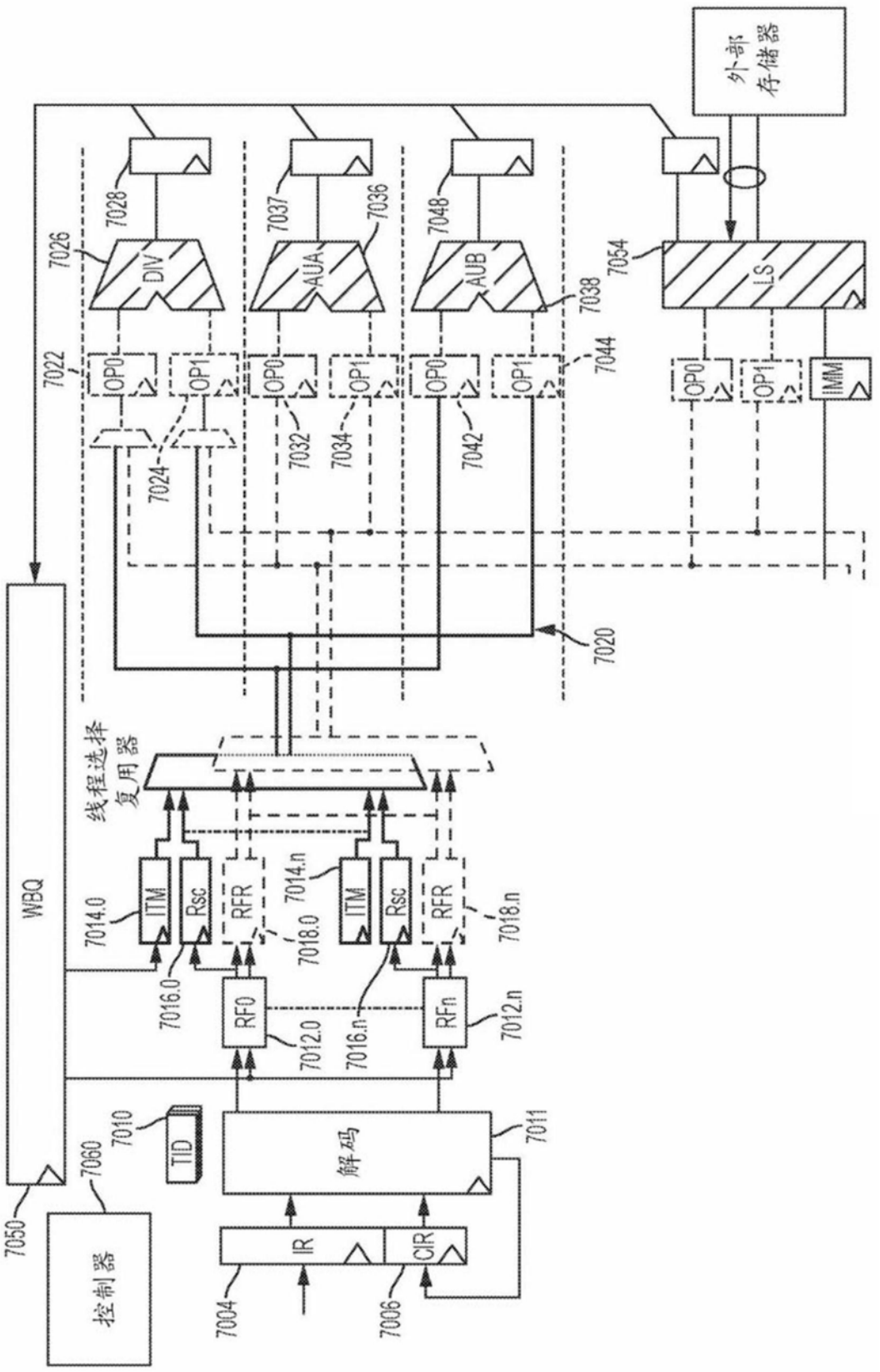

外部存储器
7028
7037
7036
7048
7026
DIV
AUA
AUB
7054
LS
7038
7022
OP0
OP1
7044
IMM
7024
7032
7034
7042
7020
线程选择复用器
WBQ
ITM
Rsc
RFR
7014.0
7014.n
7018.0
7018.n
7016.0
7016.n
RF0
RFn
7012.0
7012.n
7010
TID
解码
7011
7060
7050
控制器
IR
CIR
7004
7006

图10C

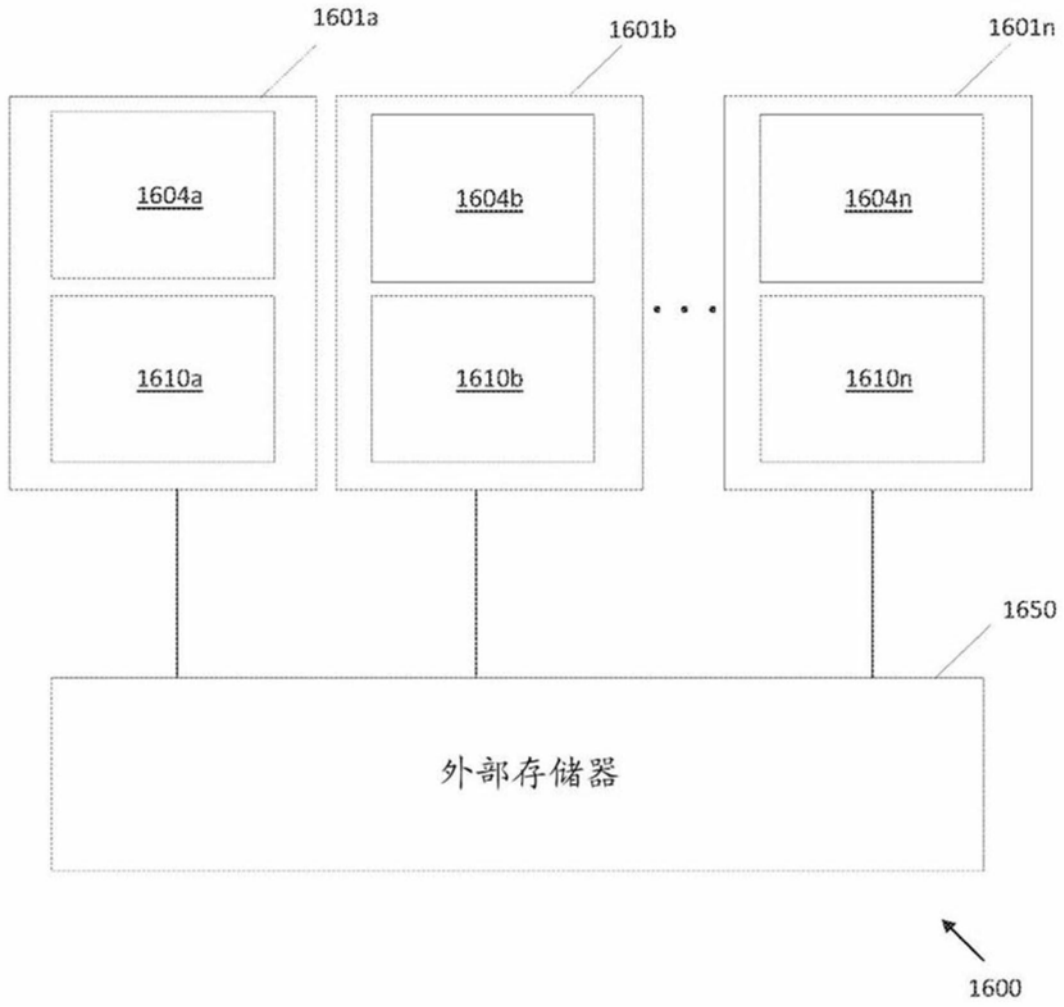

1601a
1601b
1601n
1604a
1604b
1604n
1610a
1610b
1610n
1650
外部存储器
1600

图11


有效
1705a
脏
1707a
地址（又称标记）
1703a
高速缓存行
1701a
有效
1705b
脏
1707b
地址（又称标记）
1703b
高速缓存行
1701b
有效
1705n
脏
1707n
地址（又称标记）
1703n
高速缓存行
1701n
元数据
高速缓存行数据
1700

图12


1835
1815
脏
=1
新X
1810
使用
处理器
来分类
1801
1845
1825
脏
=0
旧X
1820
使用
处理器
来分类
1802

图13A

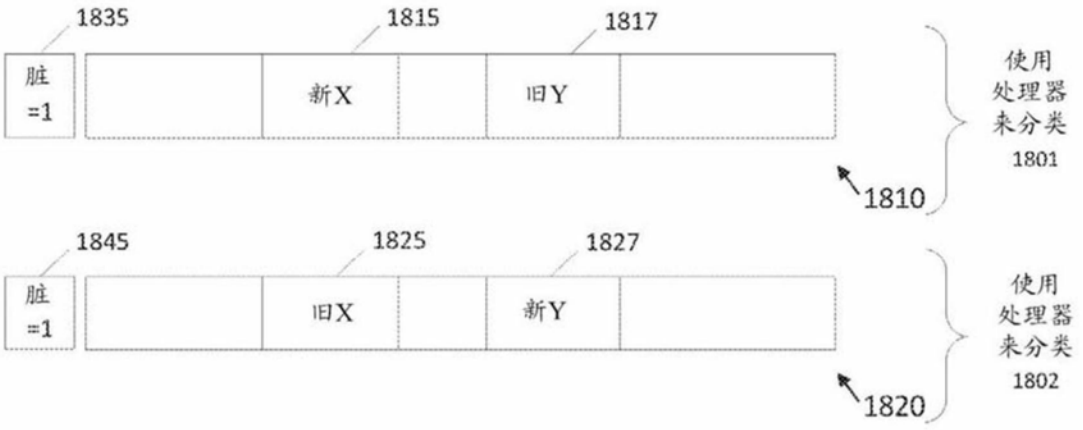

1835
1815
1817
脏
=1
新X
旧Y
使用
处理器
来分类
1801
1810
1845
1825
1827
脏
=1
旧X
新Y
使用
处理器
来分类
1802
1820

图13B

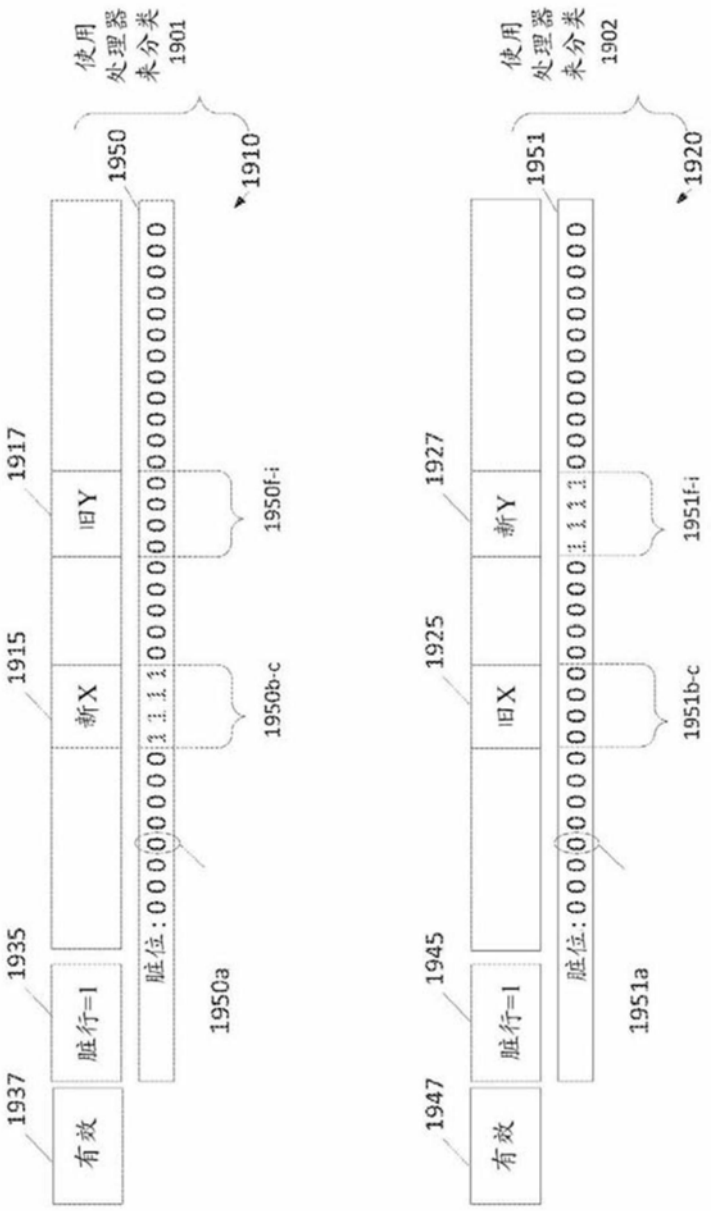


图14

脏位状态	脏/干净	含义
0	干净	数据不需要被写到外部存储器。
1	脏	数据已经在本地被更新。在外部存储器中的相应数据需要被重新写入/用本地变化来更新。

图15

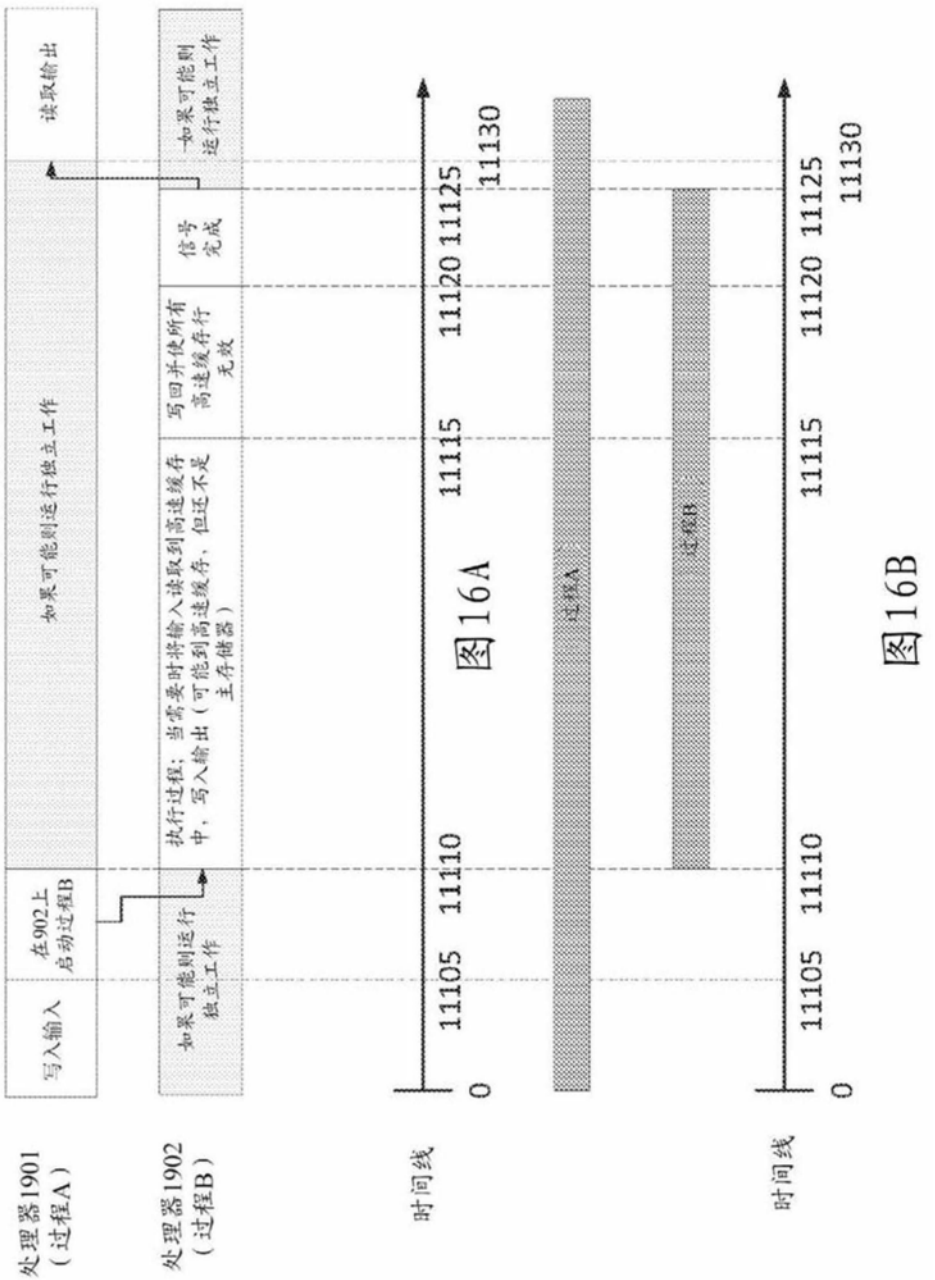

处理器1901
（过程A）
写入输入
在902上
启动过程B
如果可能则运行独立工作
读取输出
处理器1902
（过程B）
如果可能则运行
独立工作
执行过程：当需要时将输入读取到高速缓存
中，写入输出（可能到高速缓存，但还不是
主存储器）
写回并使所有
高速缓存行
无效
信号
完成
如果可能则
运行独立工作
时间线
0
11105
11110
11115
11120
11125
11130
过程A
过程B
时间线
0
11105
11110
11115
11120
11125
11130

图16A

图16B

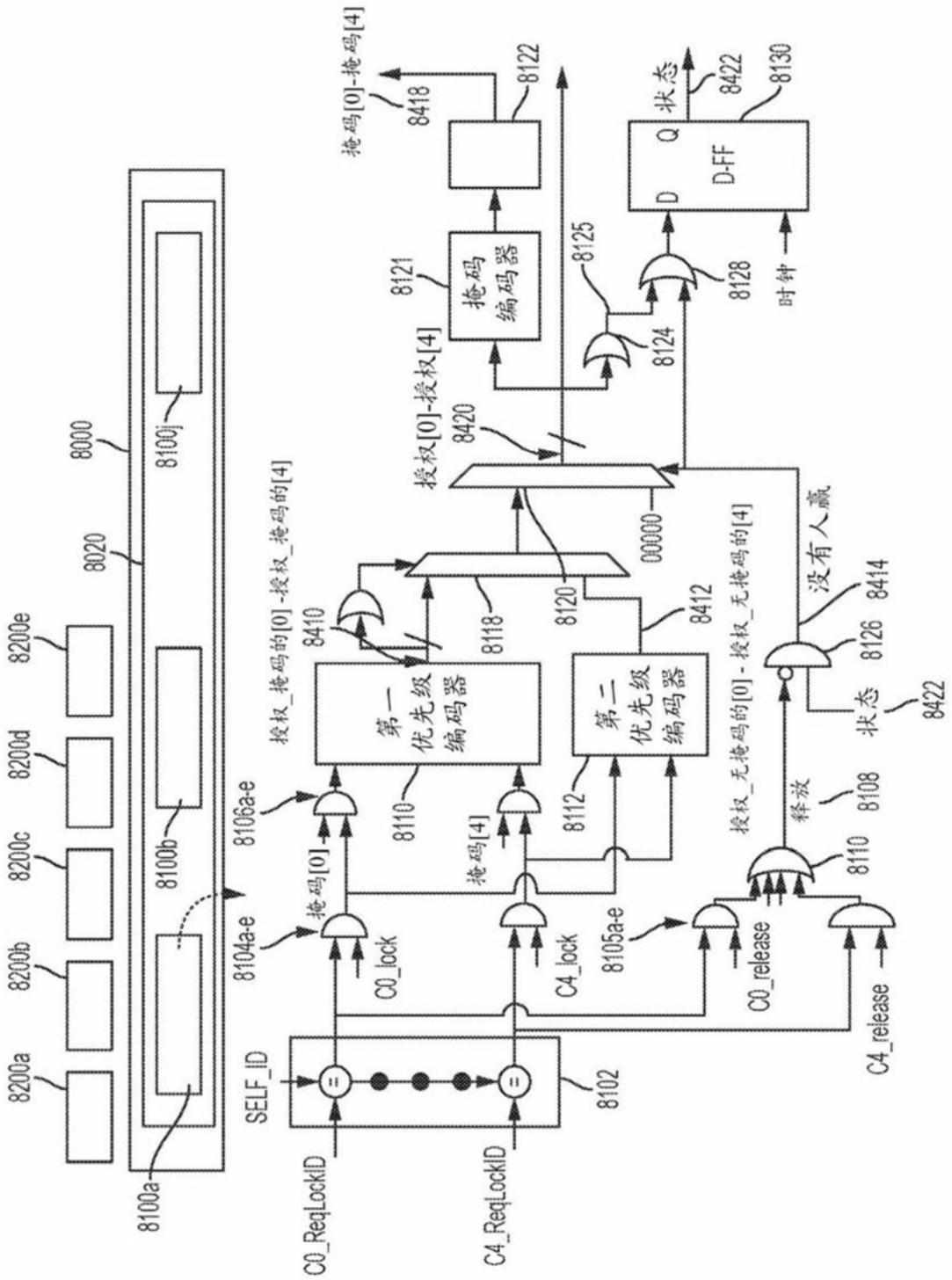

掩码[0]-掩码[4]
8418
8122
8121
掩码编码器
8125
8124
8128
D
Q
状态
D-FF
8422
8130
时钟
授权[0]-授权[4]
8420
8000
8020
8100j
8100b
8100a
8200a
8200b
8200c
8200d
8200e
授权_掩码的[0]-授权_掩码的[4]
8410
第一优先级编码器
8110
8118
8120
00000
8412
第二优先级编码器
8112
没有人赢
8414
8126
状态
8422
8106a-e
掩码[0]
C0_lock
8104a-e
掩码[4]
C4_lock
8105a-e
授权_无掩码的[0]-授权_无掩码的[4]
释放
8108
8110
C0_release
C4_release
SELF_ID
8102
C0_ReqLockID
C4_ReqLockID

图16C

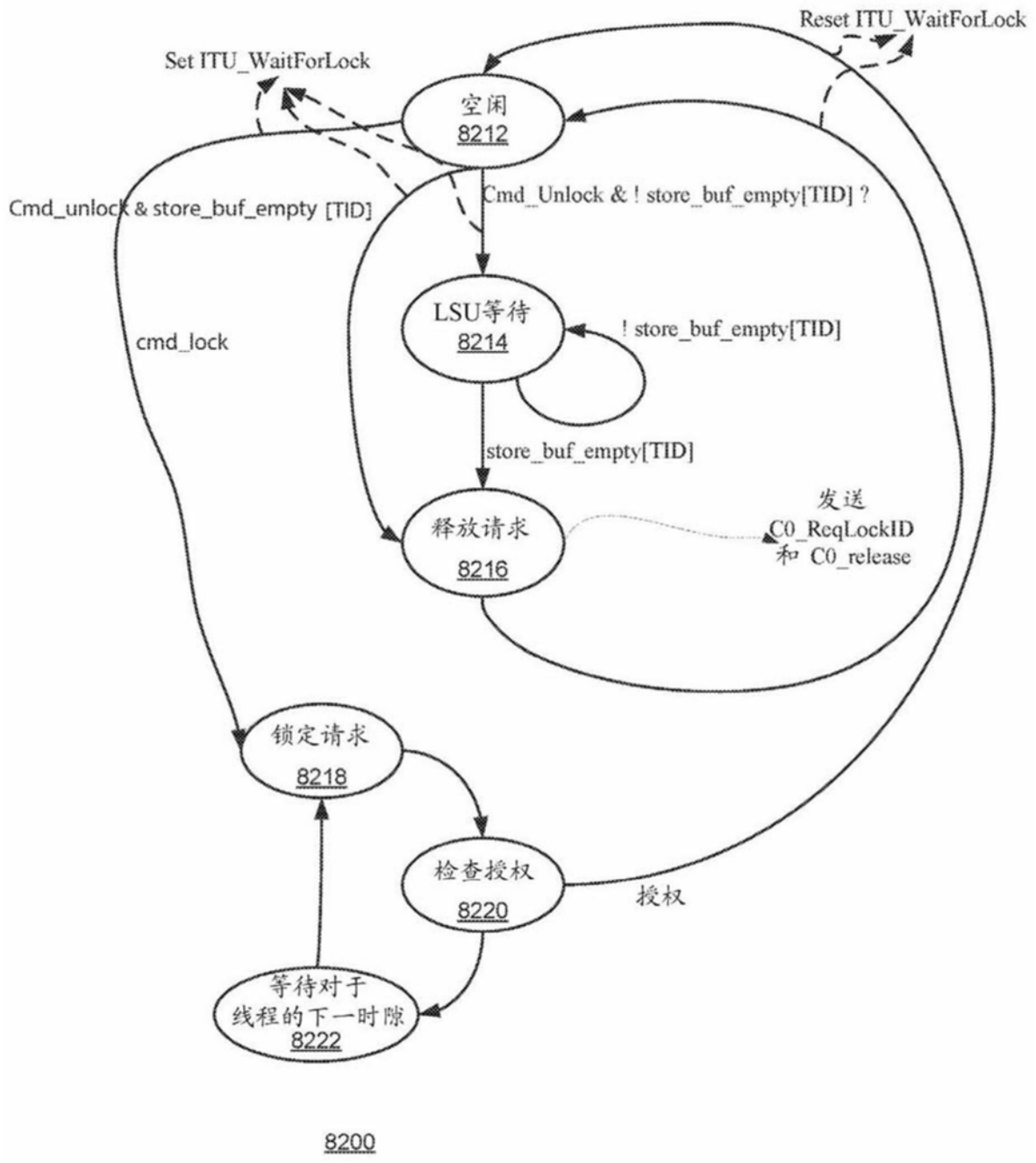

Reset ITU_WaitForLock
Set ITU_WaitForLock
空闲
8212
Cmd_unlock & store_buf_empty [TID]
Cmd_Unlock & ! store_buf_empty[TID] ?
LSU等待
8214
cmd_lock
! store_buf_empty[TID]
store_buf_empty[TID]
发送
C0_ReqLockID
和 C0_release
释放请求
8216
锁定请求
8218
检查授权
8220
授权
等待对于
线程的下一时隙
8222
8200

图16D

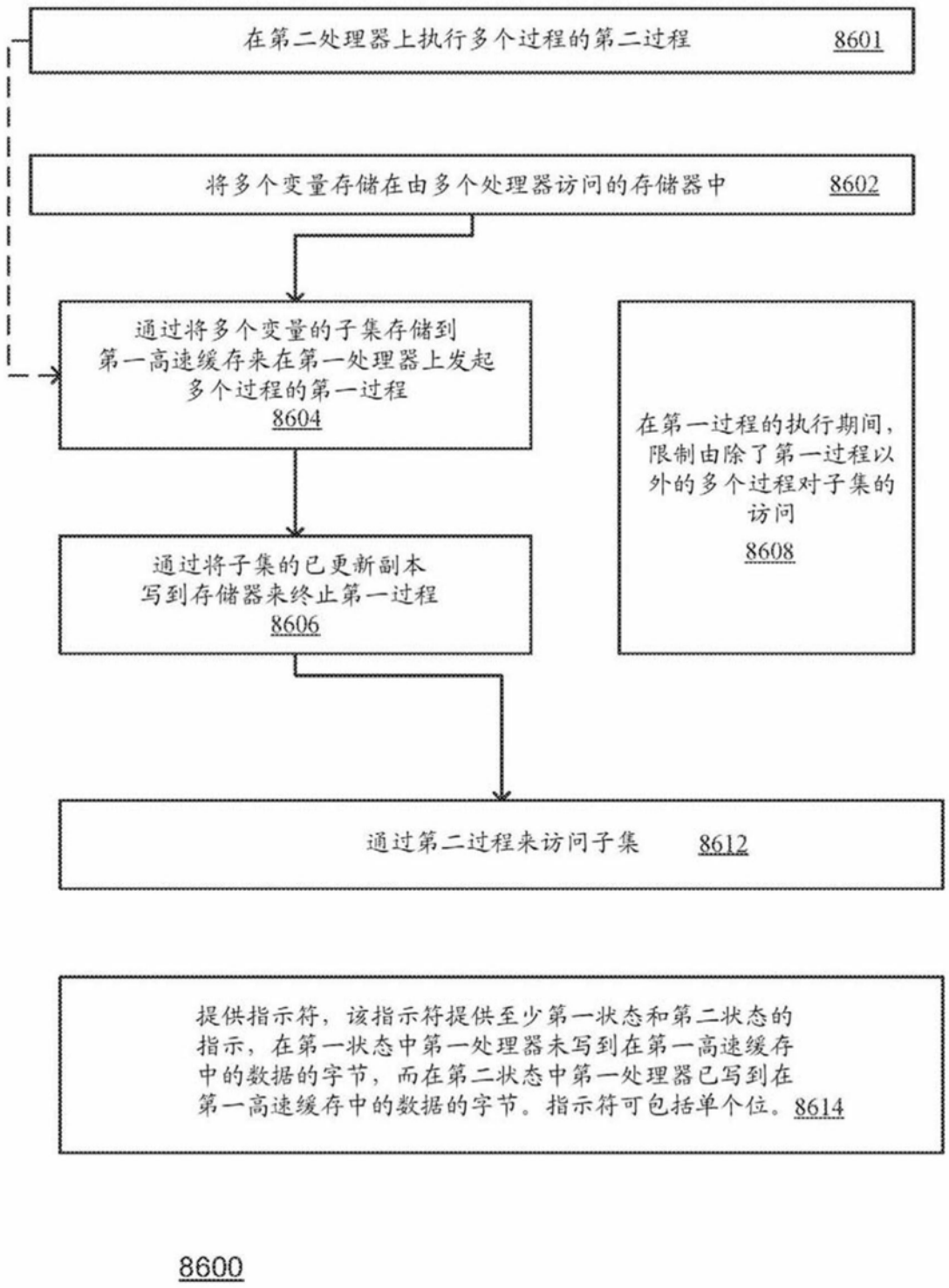

在第二处理器上执行多个过程的第二过程 8601
将多个变量存储在由多个处理器访问的存储器中 8602
通过将多个变量的子集存储到第一高速缓存来在第一处理器上发起多个过程的第一过程 8604
在第一过程的执行期间，限制由除了第一过程以外的多个过程对子集的访问 8608
通过将子集的已更新副本写到存储器来终止第一过程 8606
通过第二过程来访问子集 8612
提供指示符，该指示符提供至少第一状态和第二状态的指示，在第一状态中第一处理器未写到在第一高速缓存中的数据的字节，而在第二状态中第一处理器已写到在第一高速缓存中的数据的字节。指示符可包括单个位。8614
8600

图16E

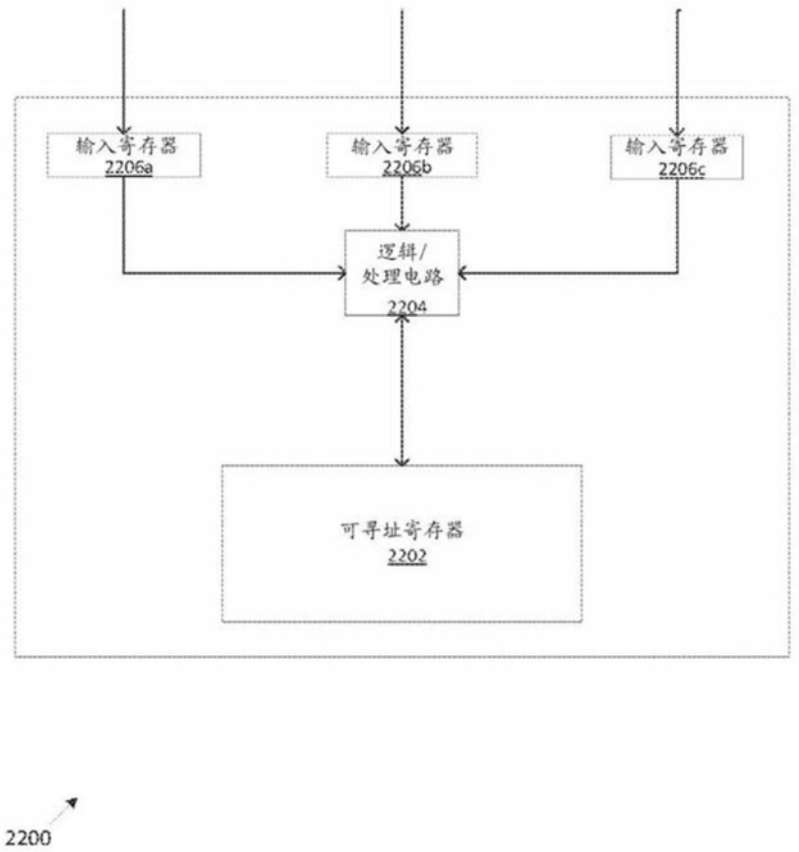

输入寄存器
2206a
输入寄存器
2206b
输入寄存器
2206c
逻辑/
处理电路
2204
可寻址寄存器
2202
2200

图17

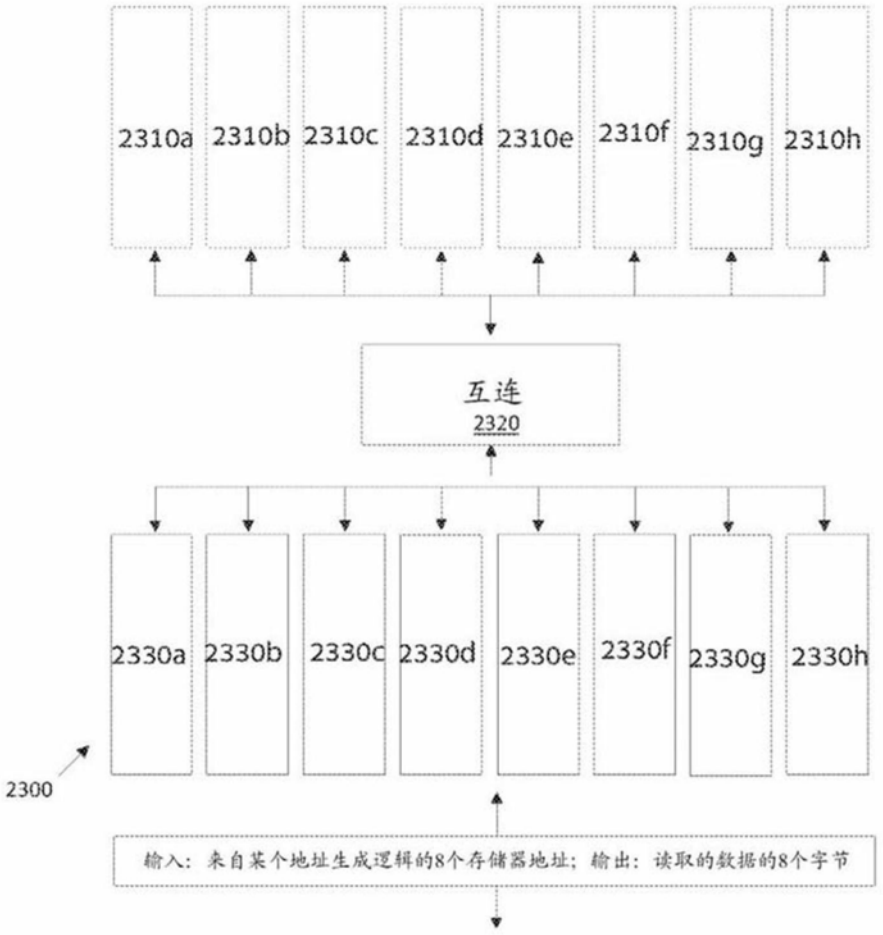

2310a
2310b
2310c
2310d
2310e
2310f
2310g
2310h
互连
2320
2330a
2330b
2330c
2330d
2330e
2330f
2330g
2330h
2300
输入：来自某个地址生成逻辑的8个存储器地址；输出：读取的数据的8个字节

图18

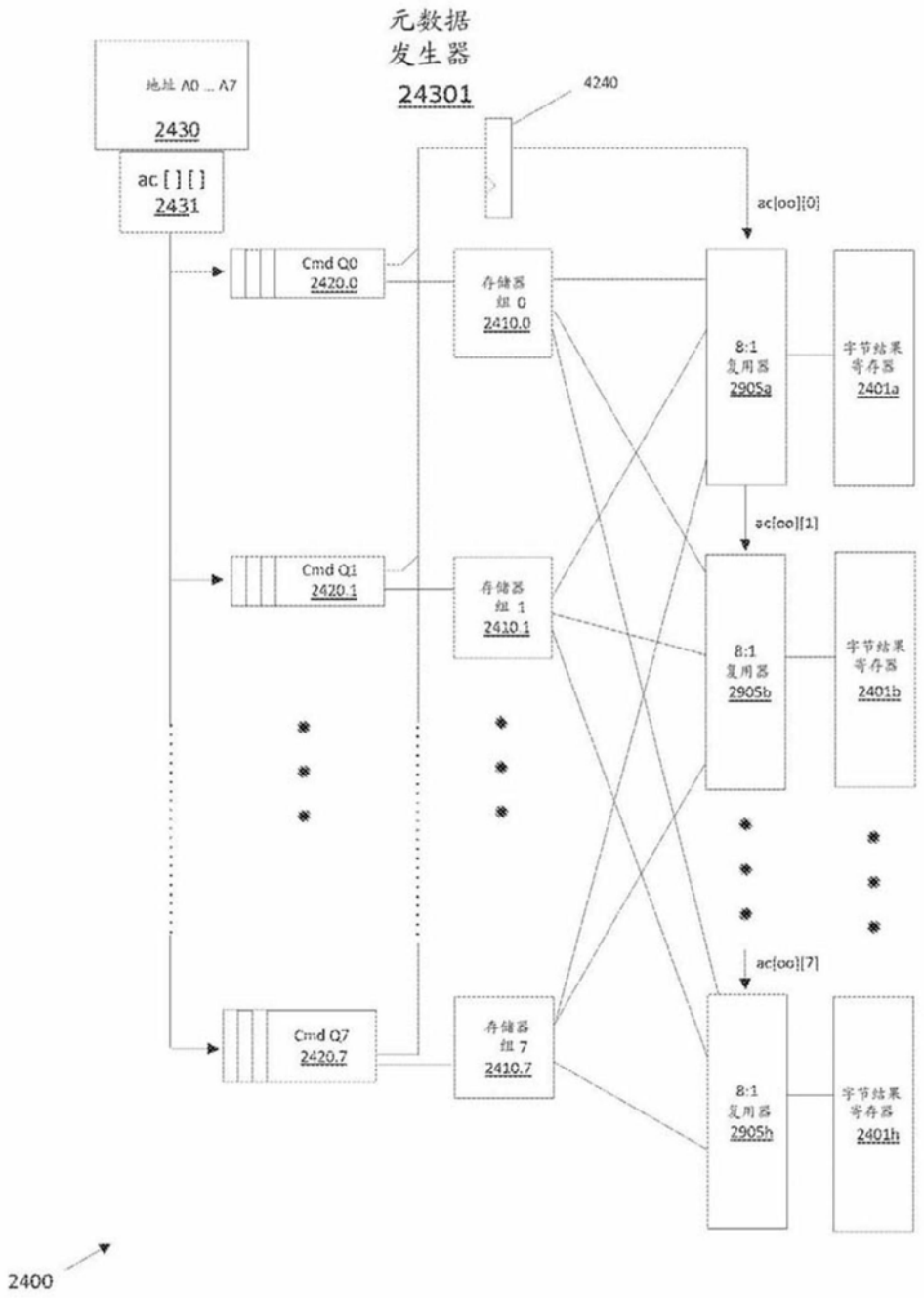

元数据
发生器
24301
地址 A0 ... A7
2430
ac [ ] [ ]
2431
4240
ac[oo][0]
Cmd Q0
2420.0
存储器
组 0
2410.0
8:1
复用器
2905a
字节结果
寄存器
2401a
ac[oo][1]
Cmd Q1
2420.1
存储器
组 1
2410.1
8:1
复用器
2905b
字节结果
寄存器
2401b
ac[oo][7]
Cmd Q7
2420.7
存储器
组 7
2410.7
8:1
复用器
2905h
字节结果
寄存器
2401h
2400

图19A

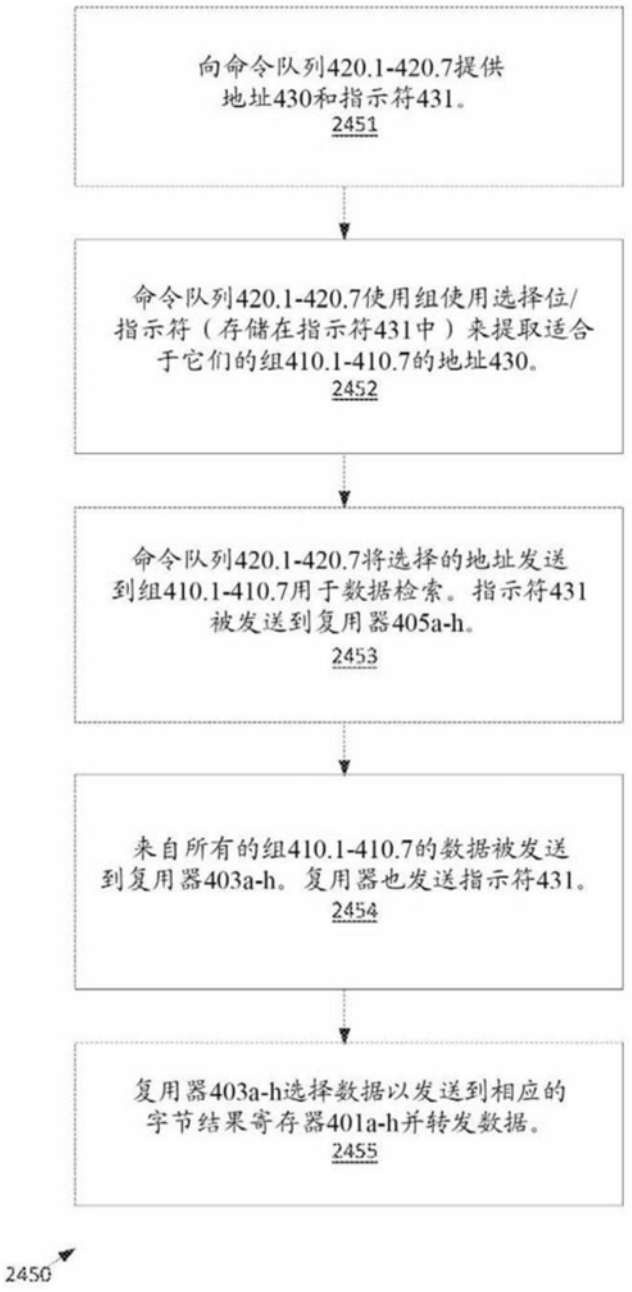

向命令队列420.1-420.7提供
地址430和指示符431。
2451
命令队列420.1-420.7使用组使用选择位/
指示符（存储在指示符431中）来提取适合
于它们的组410.1-410.7的地址430。
2452
命令队列420.1-420.7将选择的地址发送
到组410.1-410.7用于数据检索。指示符431
被发送到复用器405a-h。
2453
来自所有的组410.1-410.7的数据被发送
到复用器403a-h。复用器也发送指示符431。
2454
复用器403a-h选择数据以发送到相应的
字节结果寄存器401a-h并转发数据。
2455
2450

图19B

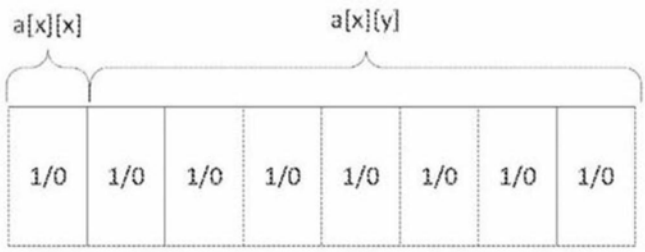

a[x][x]
a[x][y]
1/0
1/0
1/0
1/0
1/0
1/0
1/0
1/0

图20

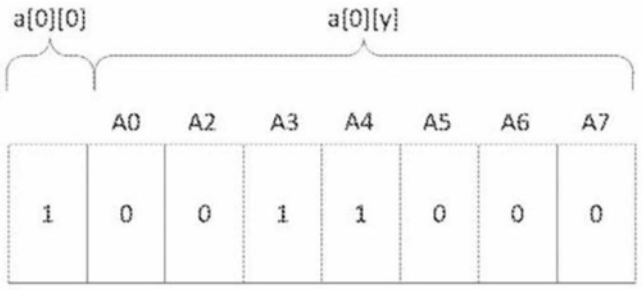


图21A

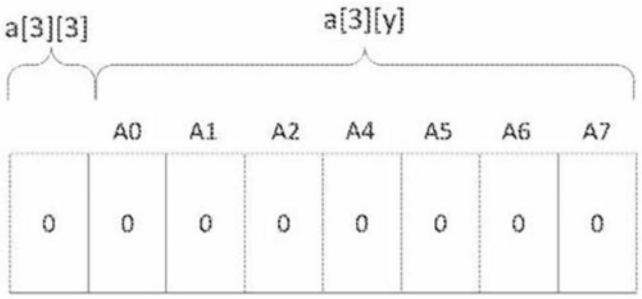


图21B

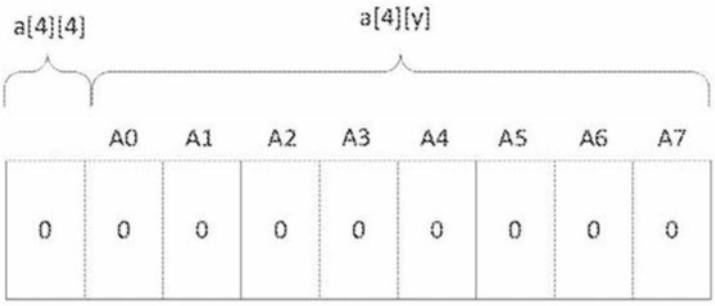


图21C

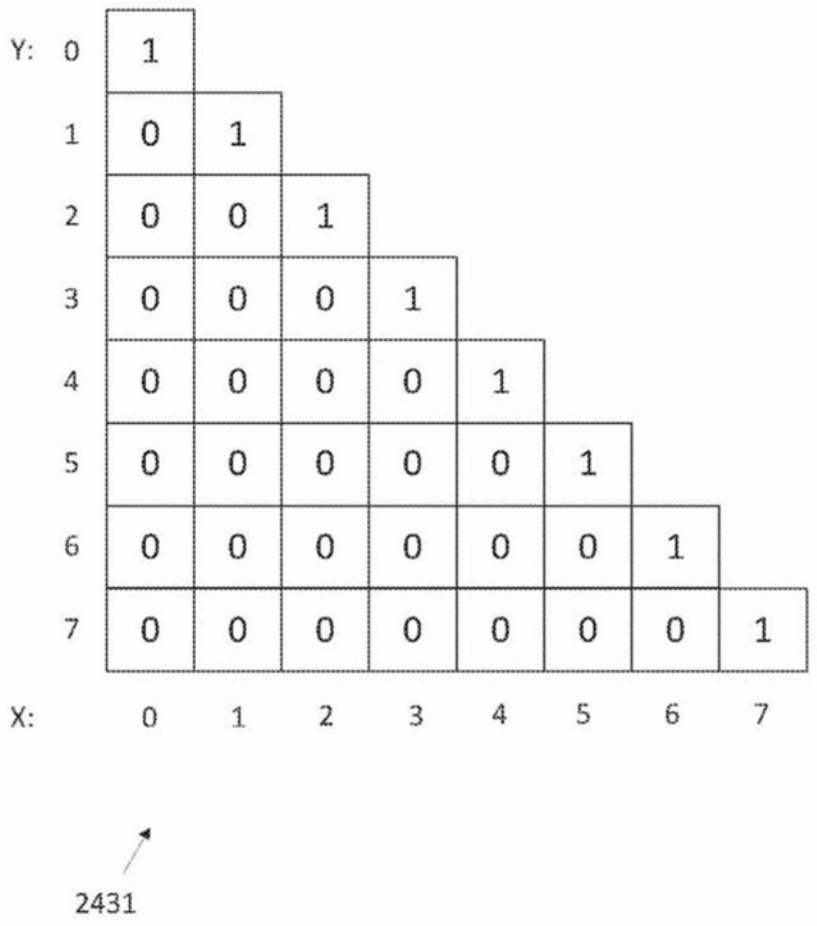


图22

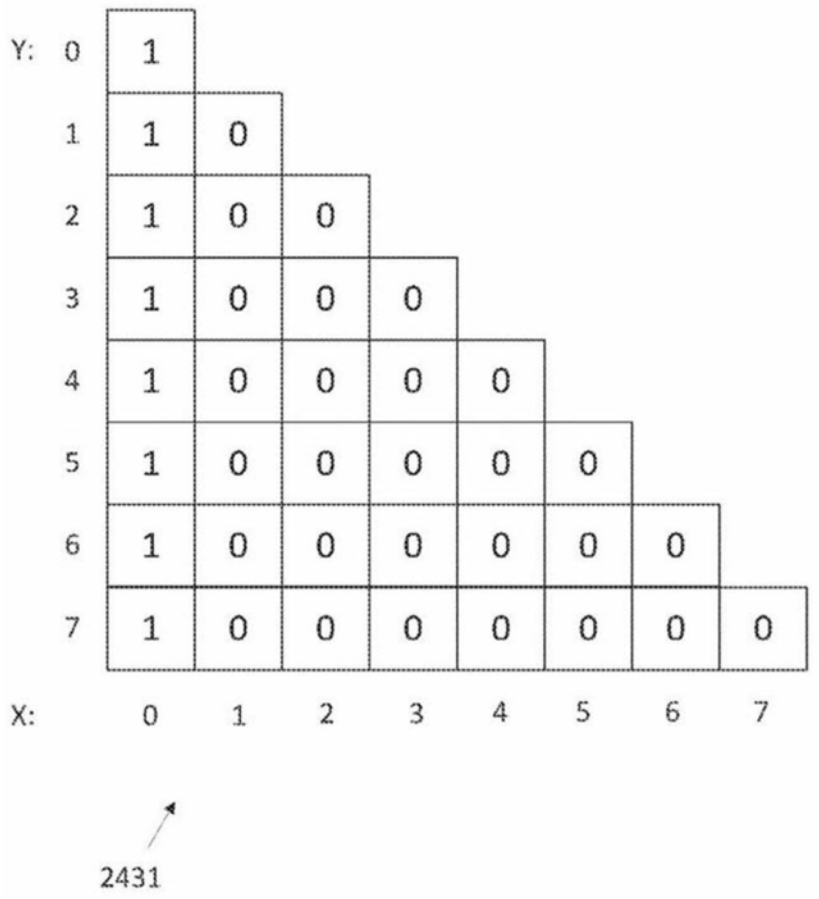


图23

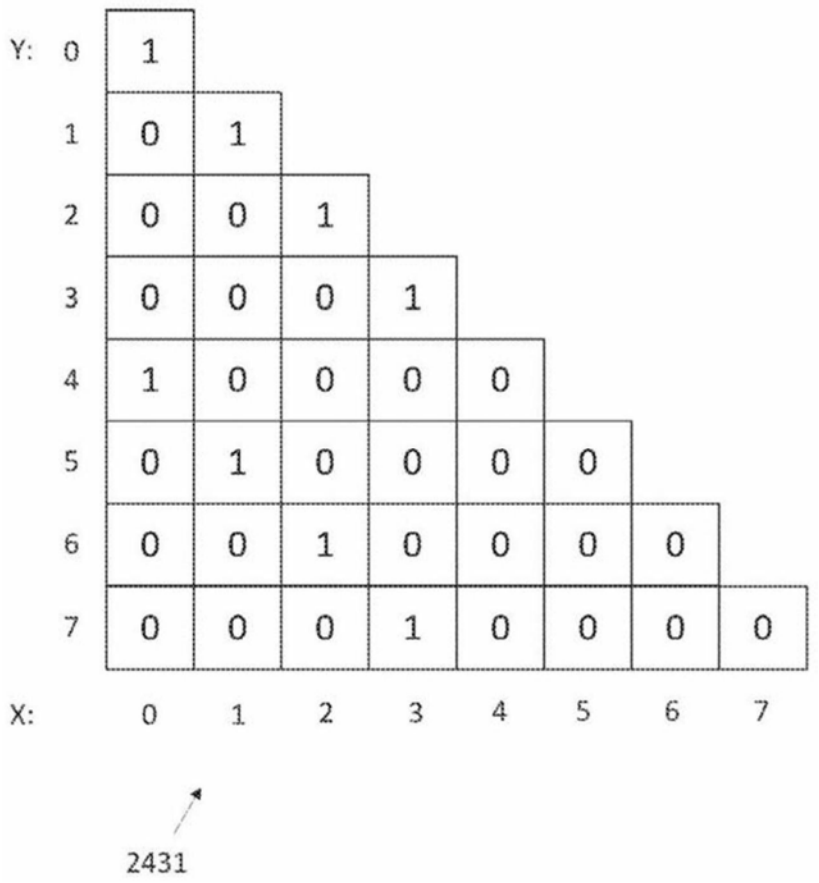

Y:
0 1
1 0 1
2 0 0 1
3 0 0 0 1
4 1 0 0 0 0
5 0 1 0 0 0 0
6 0 0 1 0 0 0 0
7 0 0 0 1 0 0 0 0
X: 0 1 2 3 4 5 6 7
2431

图24

```
ac[0][0] = 1;
ac[0][1] = (A[0]~=A[1]);
ac[0][2] = (A[0]~=A[2]);
ac[0][3] = (A[0]~=A[3]);
ac[0][4] = (A[0]~=A[4]);
ac[0][5] = (A[0]~=A[5]);
ac[0][6] = (A[0]~=A[6]);
ac[0][7] = (A[0]~=A[7]);

ac[1][1] = (!ac[0][1]);
ac[1][2] = ac[1][1] && (A[1]~=A[2]);
ac[1][3] = ac[1][1] && (A[1]~=A[3]);
ac[1][4] = ac[1][1] && (A[1]~=A[4]);
ac[1][5] = ac[1][1] && (A[1]~=A[5]);
ac[1][6] = ac[1][1] && (A[1]~=A[6]);
ac[1][7] = ac[1][1] && (A[1]~=A[7]);

ac[2][2] = (!ac[1][2] && !ac[0][2]);
ac[2][3] = ac[2][2] && (A[2]~=A[3]);
ac[2][4] = ac[2][2] && (A[2]~=A[4]);
ac[2][5] = ac[2][2] && (A[2]~=A[5]);
ac[2][6] = ac[2][2] && (A[2]~=A[6]);
ac[2][7] = ac[2][2] && (A[2]~=A[7]);

ac[3][3] = (!ac[2][3] && !ac[1][3] && !ac[0][3]);
ac[3][4] = ac[3][3] && (A[3]~=A[4]);
ac[3][5] = ac[3][3] && (A[3]~=A[5]);
ac[3][6] = ac[3][3] && (A[3]~=A[6]);
ac[3][7] = ac[3][3] && (A[3]~=A[7]);

ac[4][4] = (!ac[3][4] && !ac[2][4] && !ac[1][4] && !ac[0][4]);
ac[4][5] = ac[4][4] && (A[4]~=A[5]);
ac[4][6] = ac[4][4] && (A[4]~=A[6]);
ac[4][7] = ac[4][4] && (A[4]~=A[7]);

ac[5][5] = (!ac[4][5] && !ac[3][5] && !ac[2][5] && !ac[1][5] &&
!ac[0][5]);
ac[5][6] = ac[5][5] && (A[5]~=A[6]);
ac[5][7] = ac[5][5] && (A[5]~=A[7]);

ac[6][6] = (!ac[5][6] && !ac[4][6] && !ac[3][6] && !ac[2][6] &&
!ac[1][6] && !ac[0][6]);
ac[6][7] = ac[6][6] && (A[6]~=A[7]);

ac[7][7] = (!ac[6][7] && !ac[5][7] && !ac[4][7] && !ac[3][7] &&
!ac[2][7] && !ac[1][7] && !ac[0][7]);
```

图25A

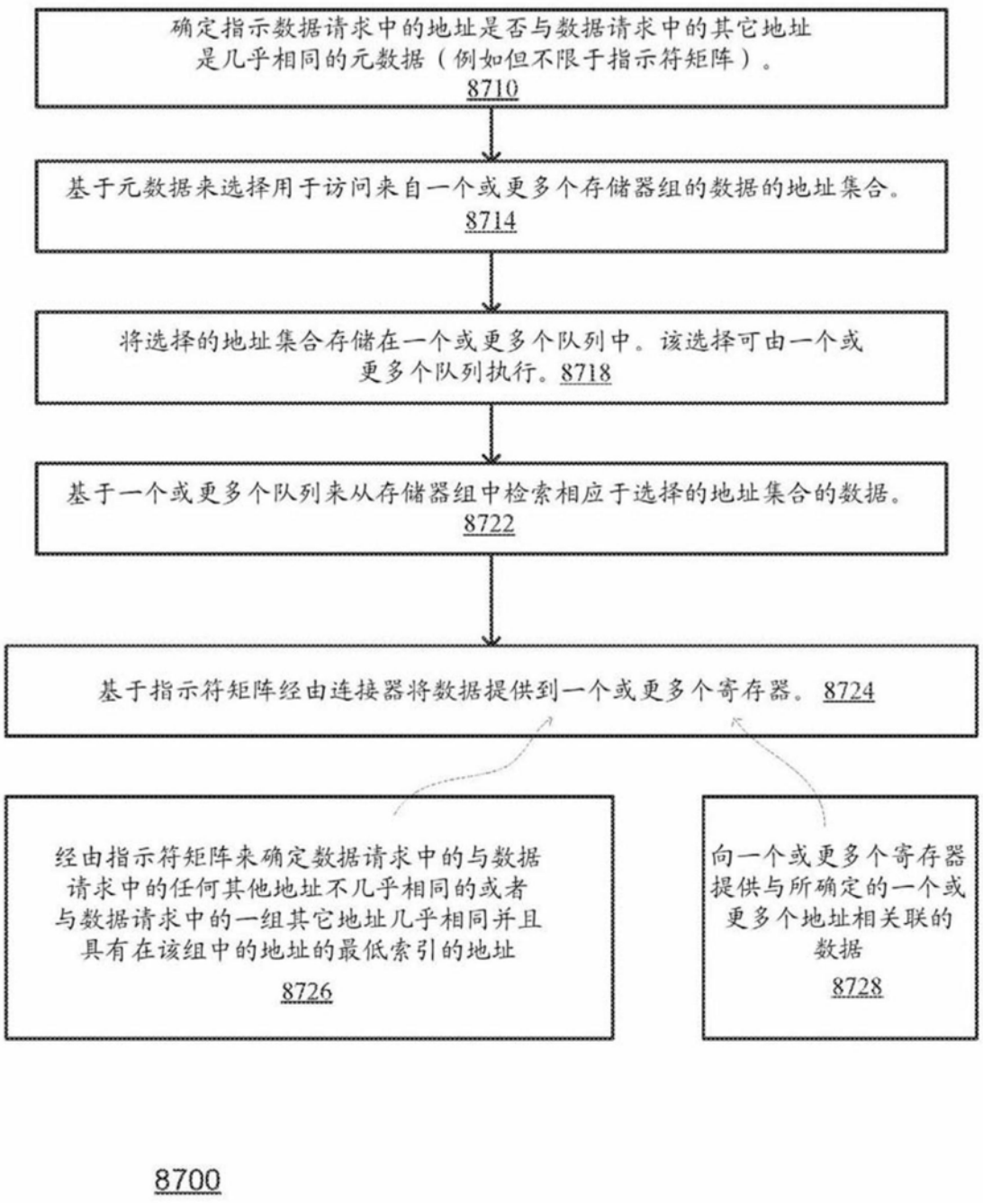

确定指示数据请求中的地址是否与数据请求中的其它地址是几乎相同的元数据（例如但不限于指示符矩阵）。
8710
基于元数据来选择用于访问来自一个或更多个存储器组的数据的地址集合。
8714
将选择的地址集合存储在一个或更多个队列中。该选择可由一个或更多个队列执行。8718
基于一个或更多个队列来从存储器组中检索相应于选择的地址集合的数据。
8722
基于指示符矩阵经由连接器将数据提供到一个或更多个寄存器。8724
经由指示符矩阵来确定数据请求中的与数据请求中的任何其他地址不几乎相同的或者与数据请求中的一组其它地址几乎相同并且具有在该组中的地址的最低索引的地址
8726
向一个或更多个寄存器提供与所确定的一个或更多个地址相关联的数据
8728
8700

图25B

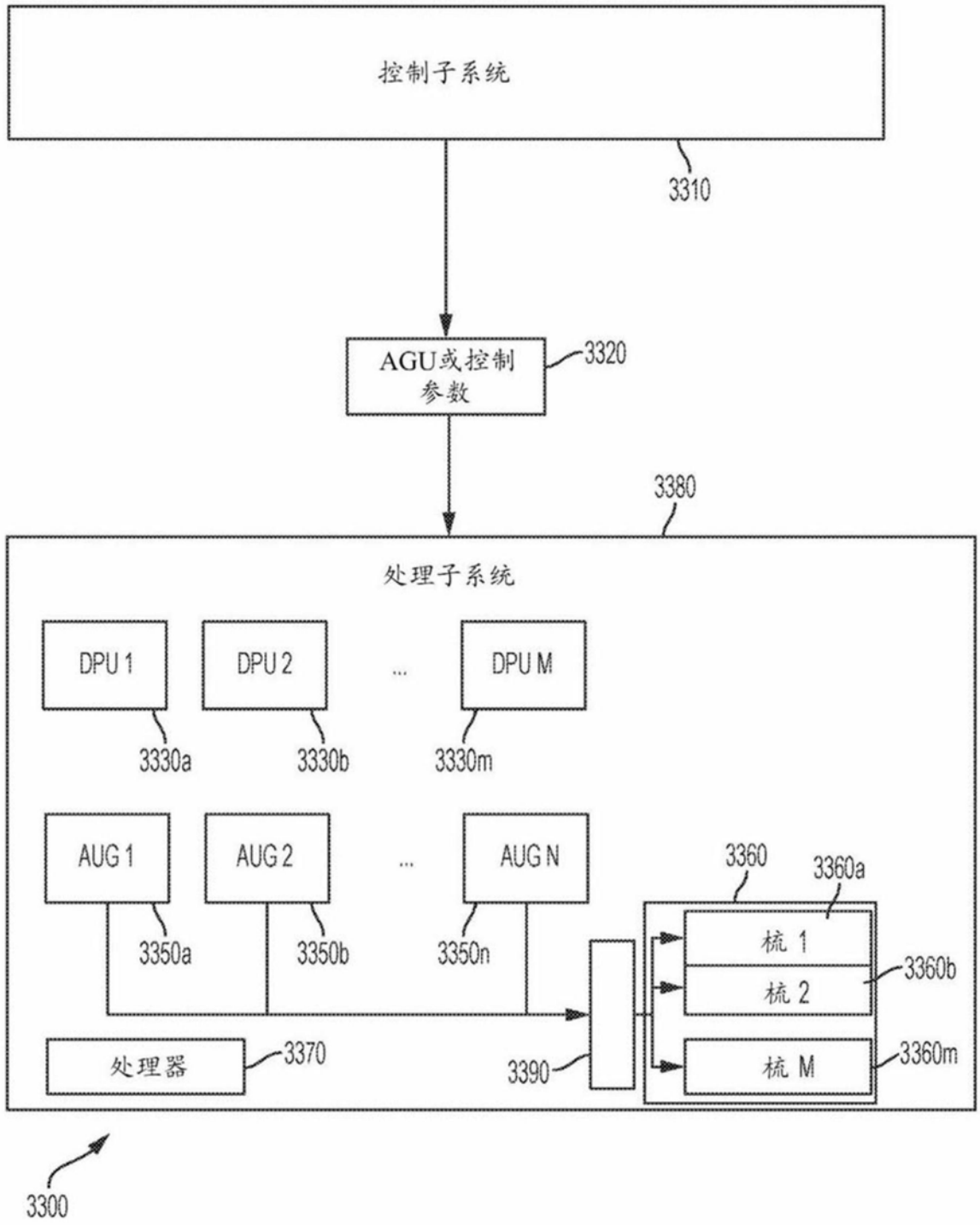

控制子系统
3310
AGU或控制参数
3320
3380
处理子系统
DPU 1
DPU 2
...
DPU M
3330a
3330b
3330m
AUG 1
AUG 2
...
AUG N
3350a
3350b
3350n
3360
3360a
梳 1
梳 2
梳 M
3360b
3360m
处理器
3370
3390
3300

图26

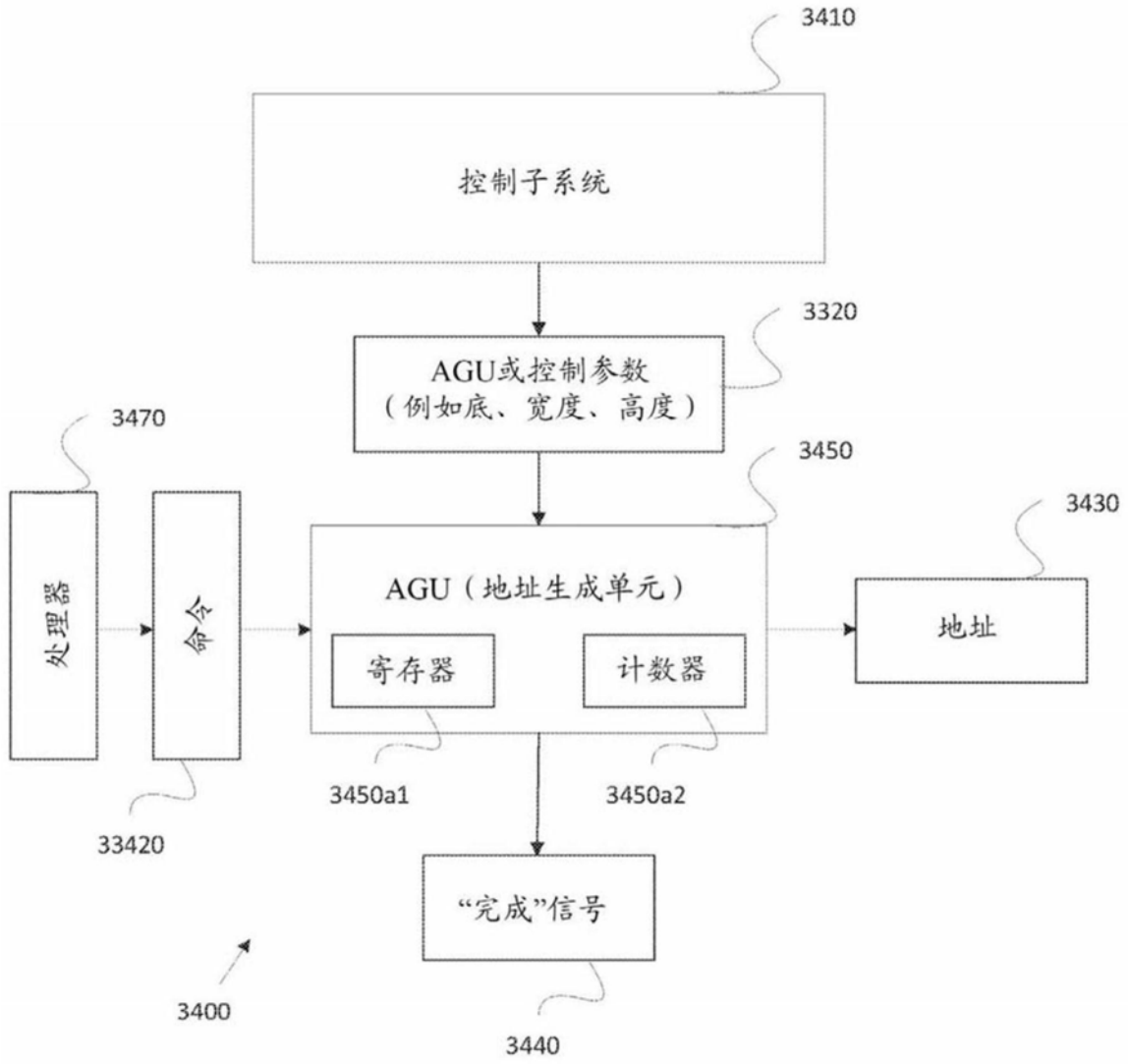

3410
控制子系统
3320
AGU或控制参数
（例如底、宽度、高度）
3470
3450
3430
处理器
命令
AGU（地址生成单元）
地址
寄存器
计数器
3450a1
3450a2
33420
“完成”信号
3400
3440

图27

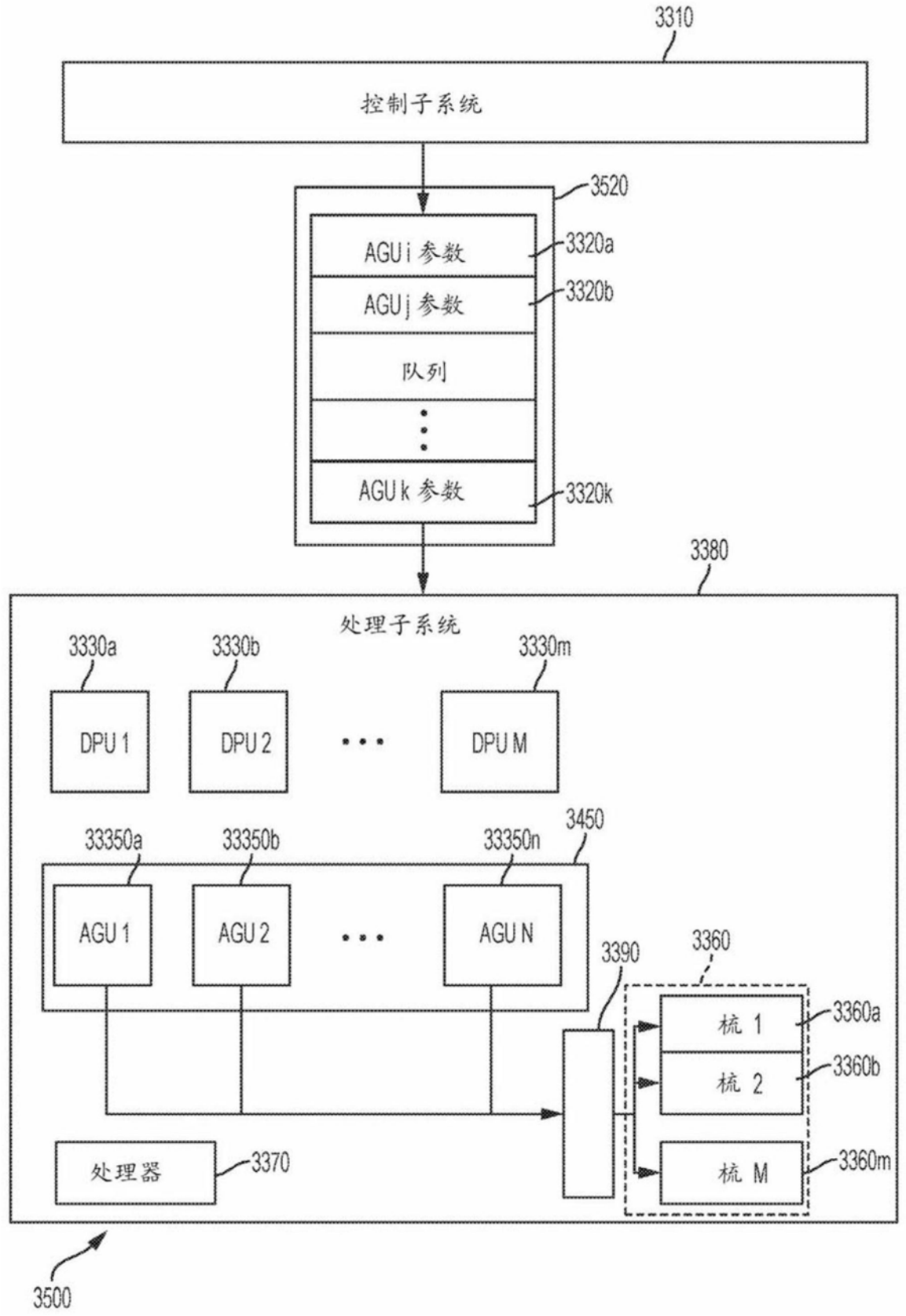

3310
控制子系统
3520
AGU i 参数
3320a
AGU j 参数
3320b
队列
AGU k 参数
3320k
3380
处理子系统
3330a
3330b
3330m
DPU 1
DPU 2
DPU M
3450
33350a
33350b
33350n
AGU 1
AGU 2
AGU N
3390
3360
梳 1
梳 2
梳 M
3360a
3360b
3360m
处理器
3370
3500

图28

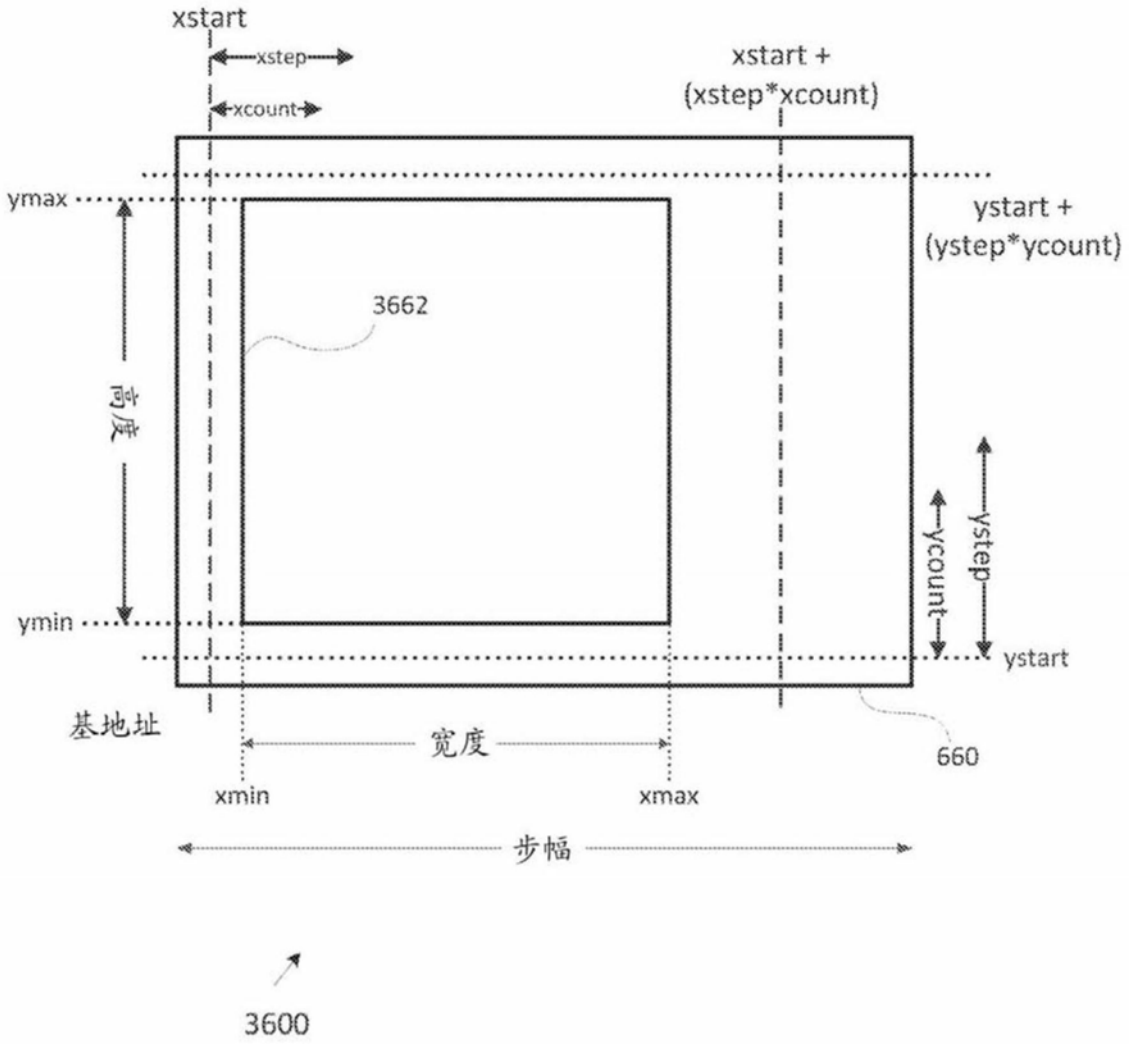

xstart
xstep
xcount
xstart +
(xstep*xcount)
ymax
ystart +
(ystep*ycount)
3662
高度
ycount
ystep
ymin
ystart
基地址
宽度
660
xmin
xmax
步幅
3600

图29

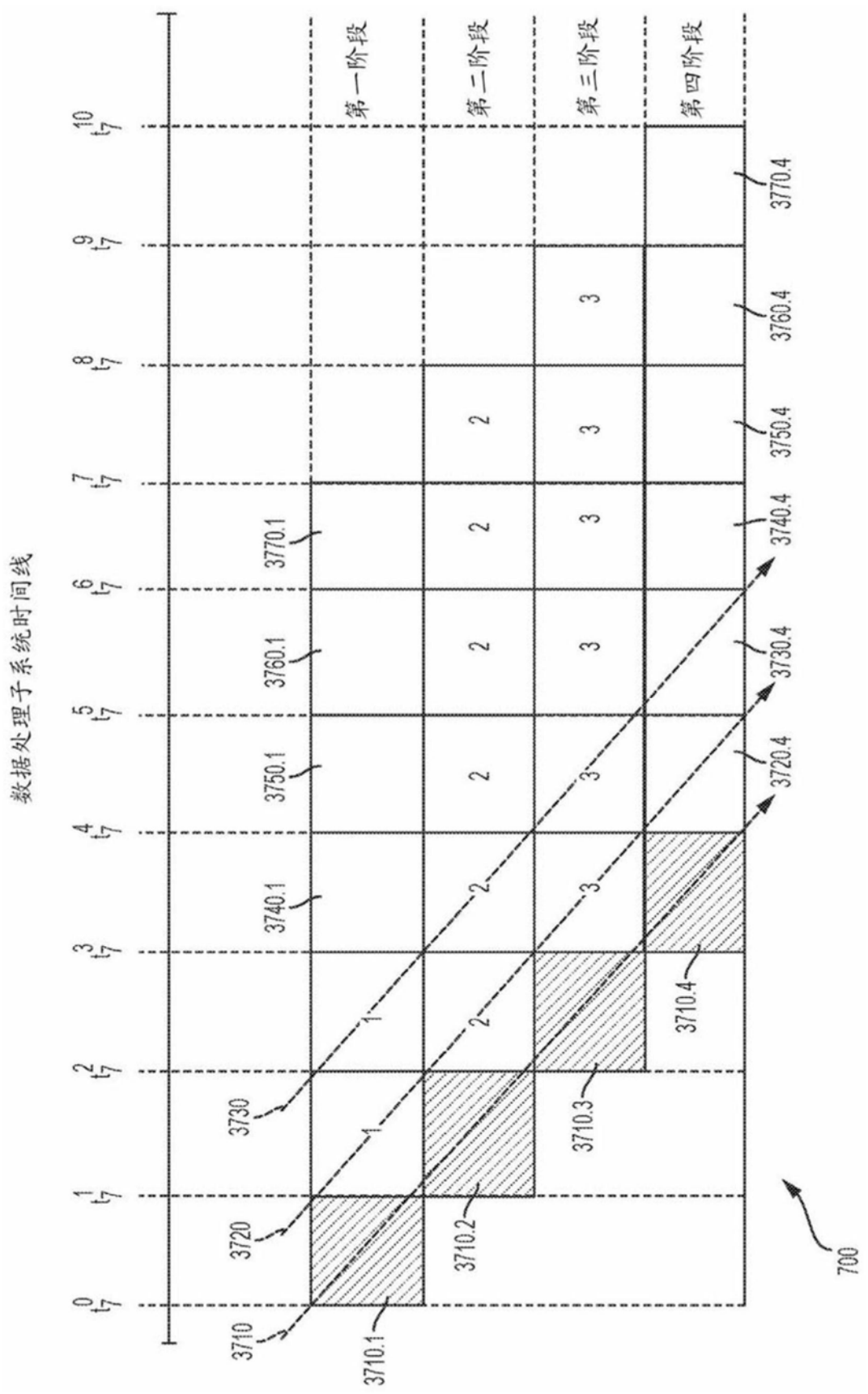

数据处理子系统时间线
第一阶段
第二阶段
第三阶段
第四阶段
t0
t1
t2
t3
t4
t5
t6
t7
t8
t9
t10
3710
3710.1
3720
3710.2
3730
3710.3
3740.1
3710.4
3750.1
3720.4
3760.1
3730.4
3770.1
3740.4
3750.4
3760.4
3770.4
700

图30

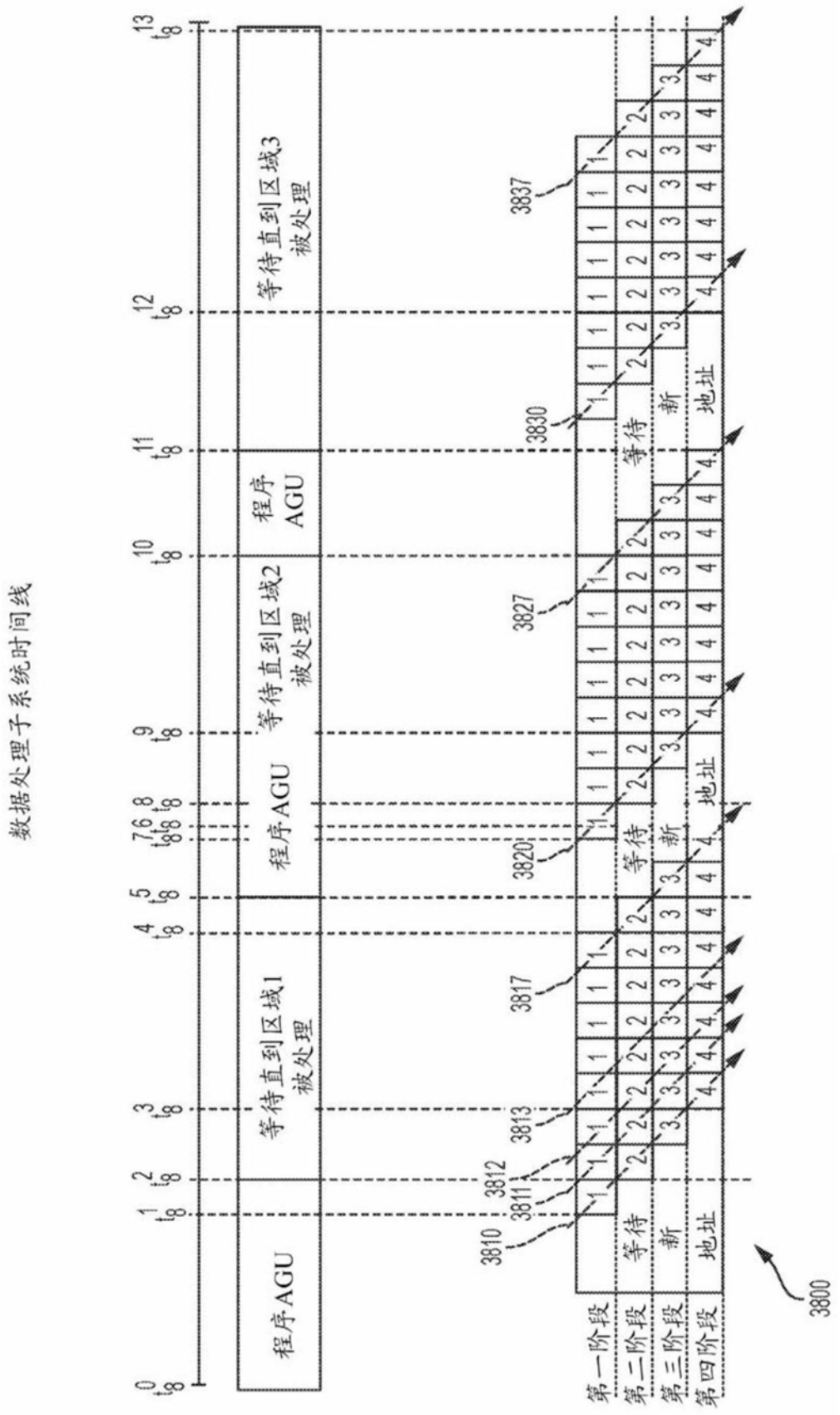

数据处理子系统时间线
程序AGU
等待直到区域1被处理
程序AGU
等待直到区域2被处理
程序AGU
等待直到区域3被处理
第一阶段
第二阶段
第三阶段
第四阶段
地址
等待
新
3800
3810
3811
3812
3813
3817
3820
3827
3830
3837

图31

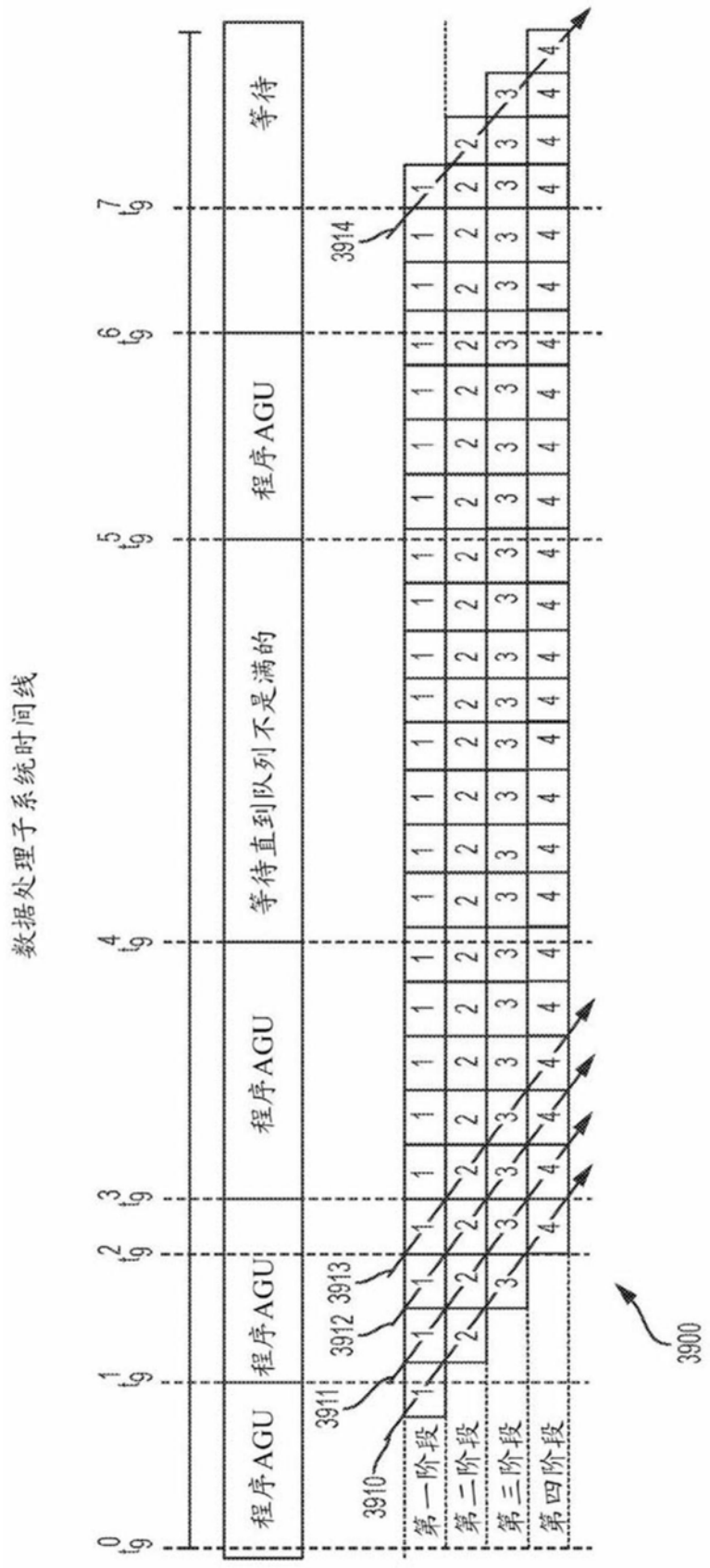

数据处理子系统时间线
程序AGU
程序AGU
等待直到队列不是满的
程序AGU
等待
第一阶段
第二阶段
第三阶段
第四阶段
3910
3911
3912
3913
3914
3900

图32

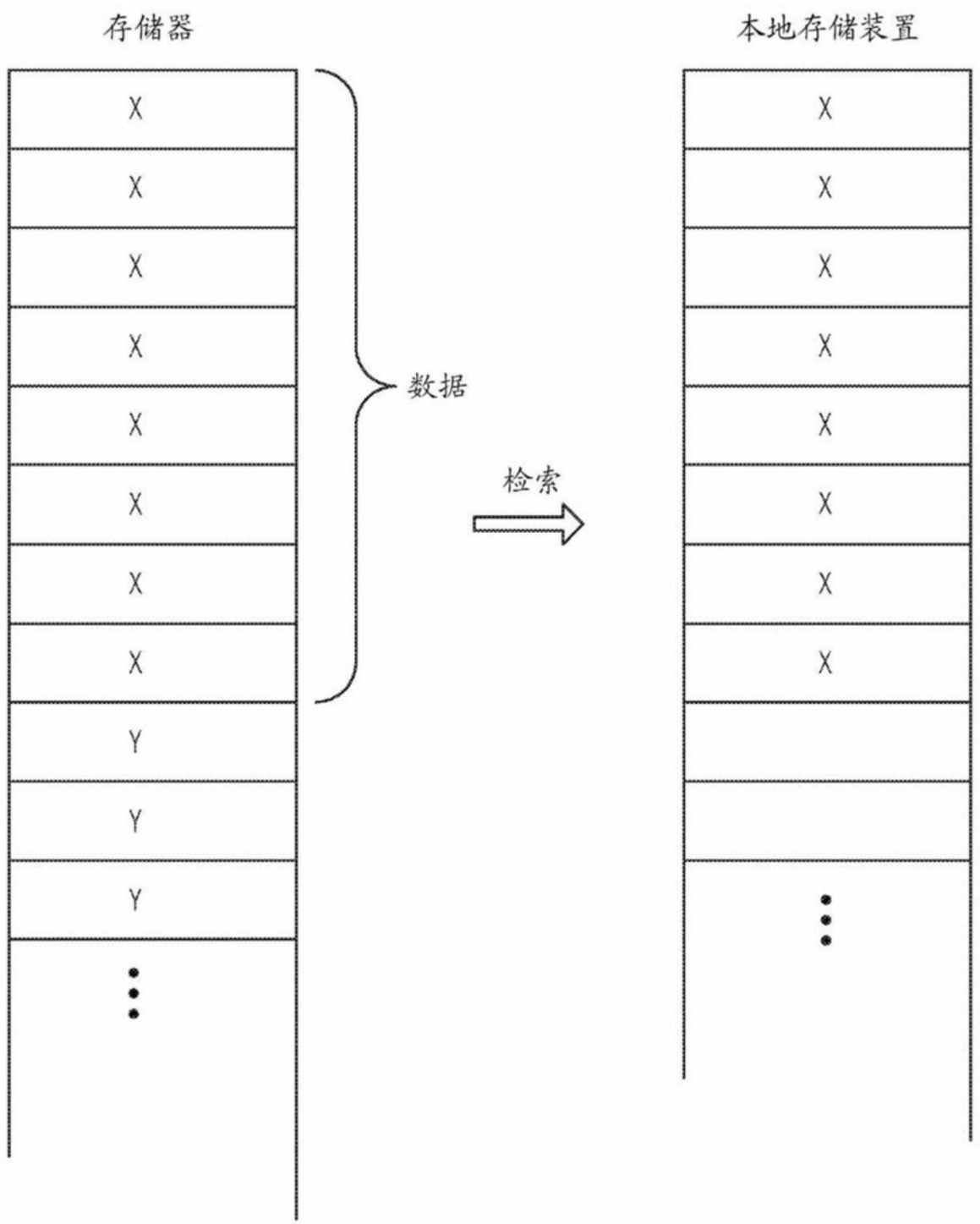


图33A

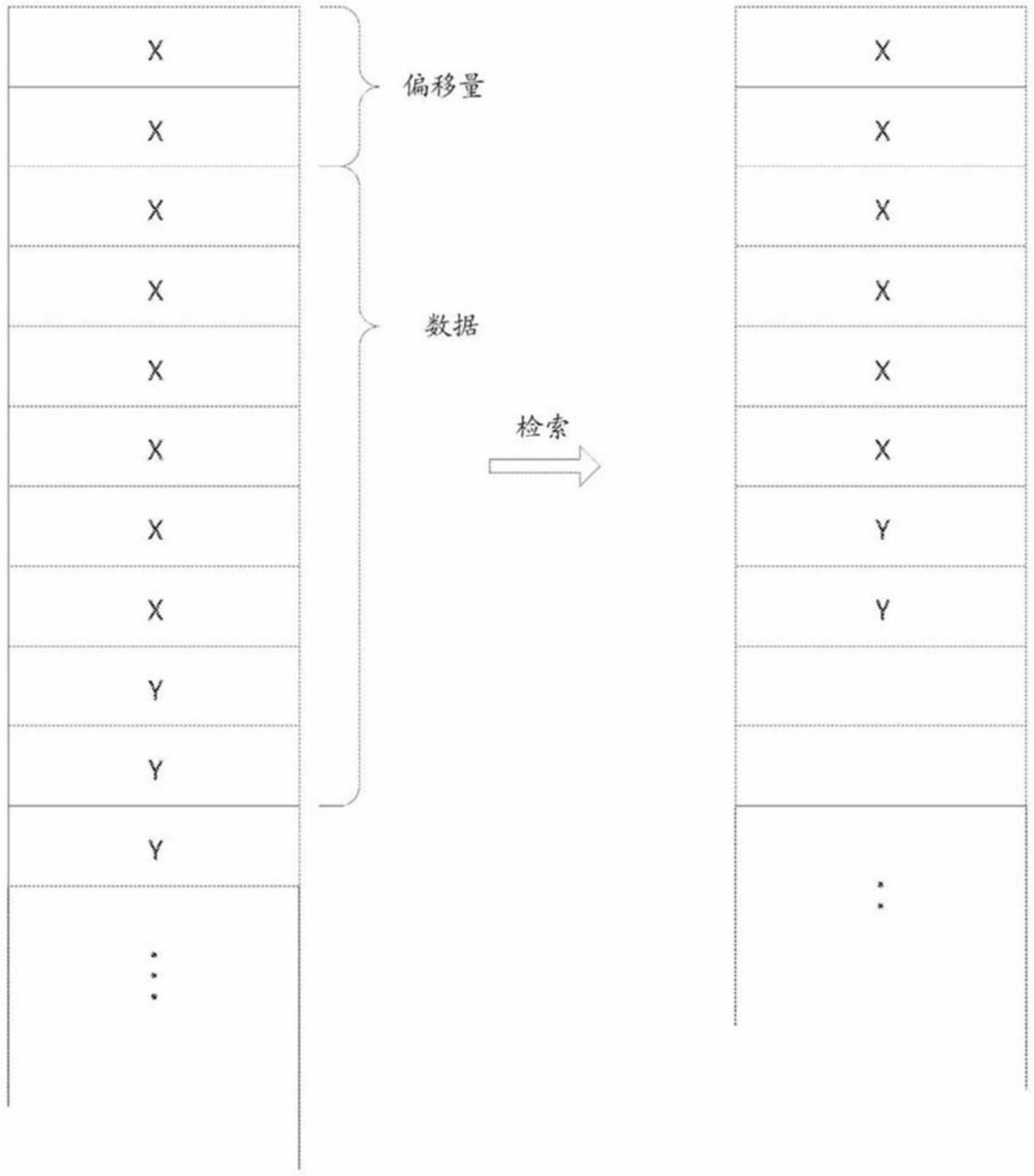


图33B

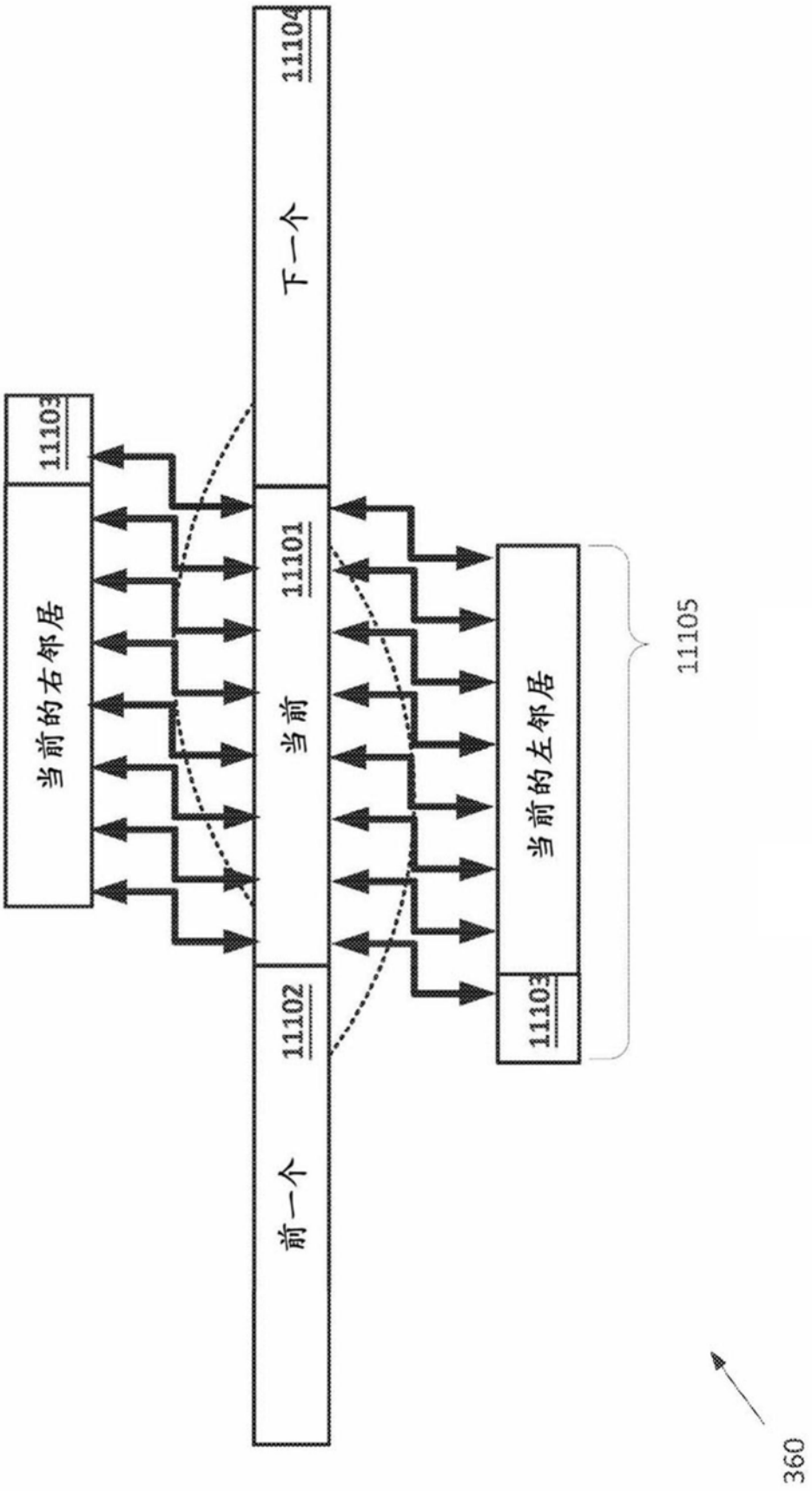

11104
下一个
11103
当前的右邻居
11101
当前
11102
前一个
当前的左邻居
11103
11105
360

图34A

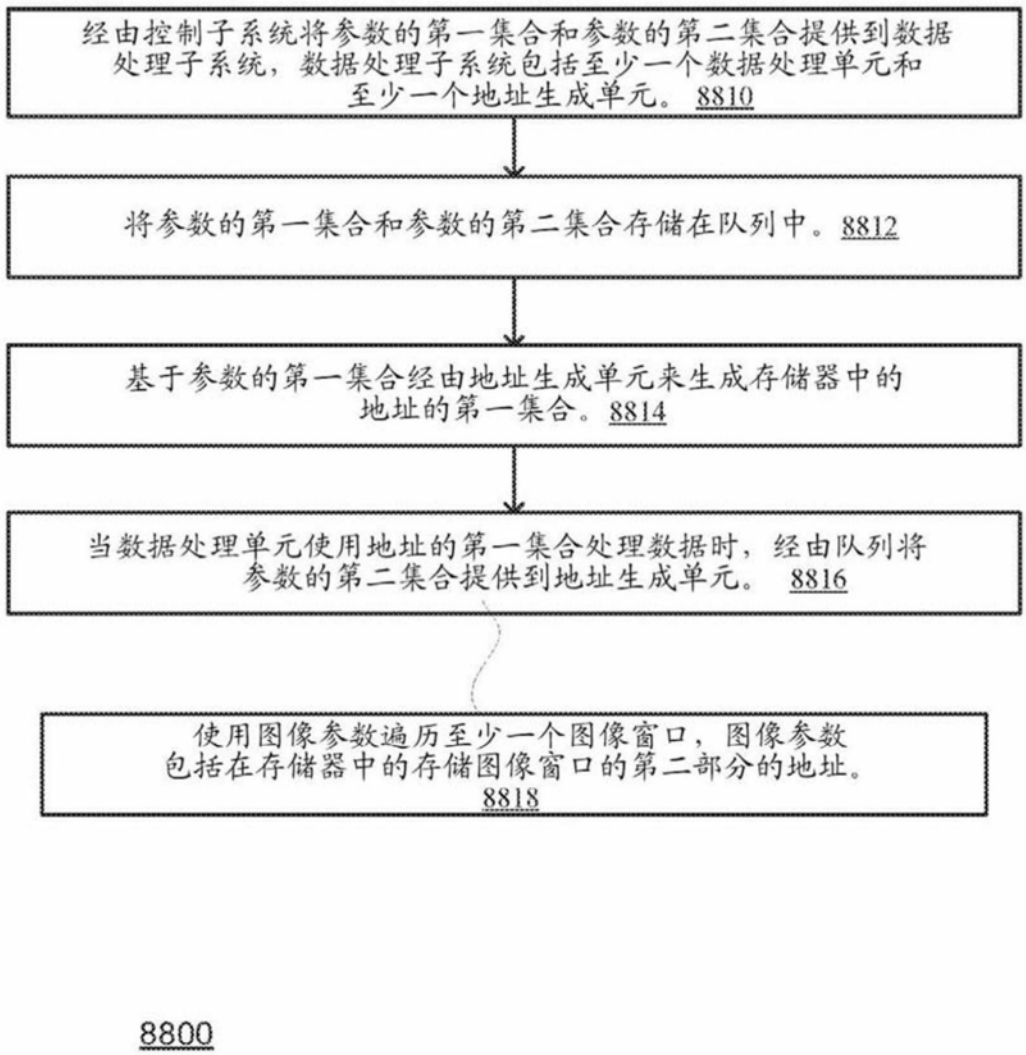

经由控制子系统将参数的第一集合和参数的第二集合提供到数据处理子系统，数据处理子系统包括至少一个数据处理单元和至少一个地址生成单元。 8810
将参数的第一集合和参数的第二集合存储在队列中。 8812
基于参数的第一集合经由地址生成单元来生成存储器中的地址的第一集合。 8814
当数据处理单元使用地址的第一集合处理数据时，经由队列将参数的第二集合提供到地址生成单元。 8816
使用图像参数遍历至少一个图像窗口，图像参数包括在存储器中的存储图像窗口的第二部分的地址。 8818
8800

图34B

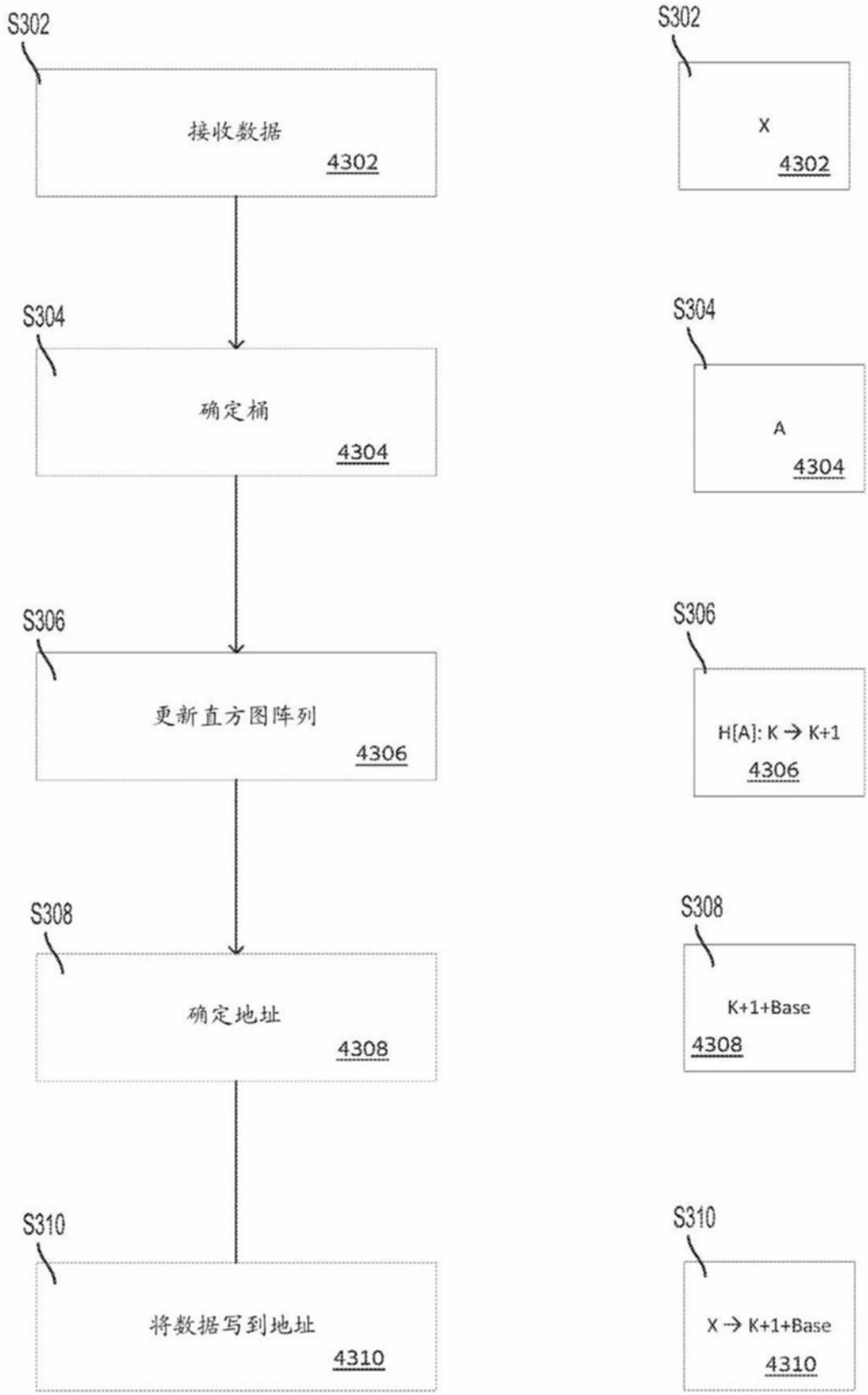

S302
接收数据
4302
S304
确定桶
4304
S306
更新直方图阵列
4306
S308
确定地址
4308
S310
将数据写到地址
4310
S302
X
4302
S304
A
4304
S306
H[A]: K → K+1
4306
S308
K+1+Base
4308
S310
X → K+1+Base
4310

图35

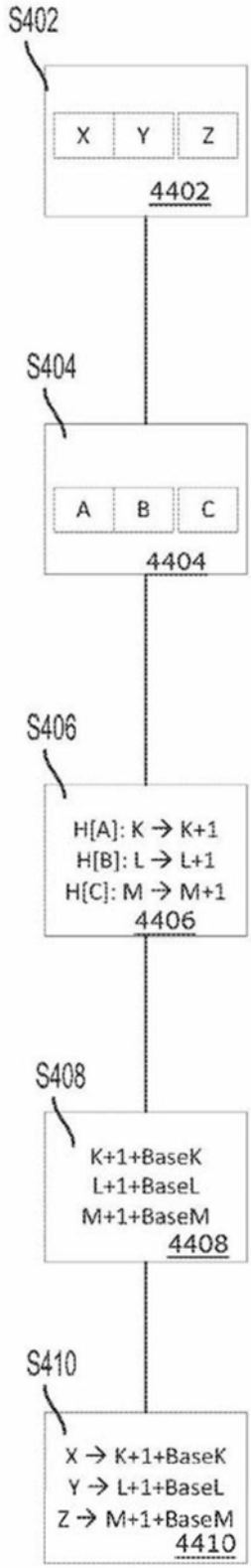

S402
X
Y
Z
4402
S404
A
B
C
4404
S406
H[A]: K → K+1
H[B]: L → L+1
H[C]: M → M+1
4406
S408
K+1+BaseK
L+1+BaseL
M+1+BaseM
4408
S410
X → K+1+BaseK
Y → L+1+BaseL
Z → M+1+BaseM
4410

图36

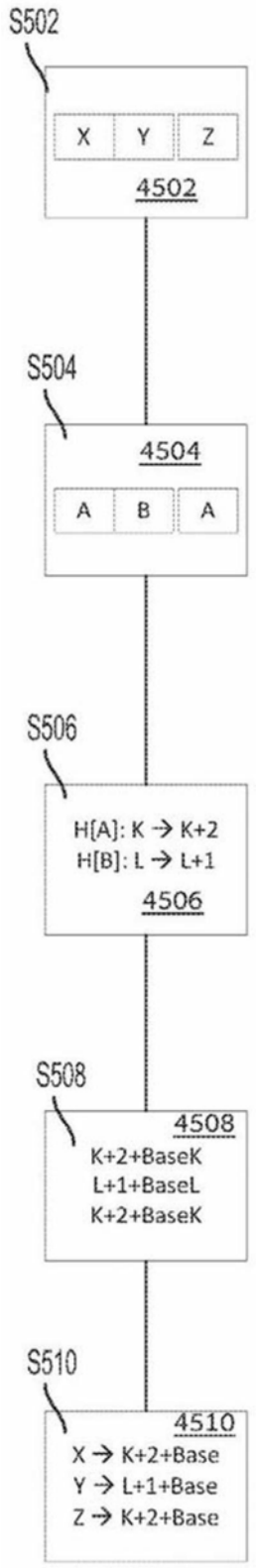

S502
X
Y
Z
4502
S504
4504
A
B
A
S506
H[A]: K → K+2
H[B]: L → L+1
4506
S508
4508
K+2+BaseK
L+1+BaseL
K+2+BaseK
S510
4510
X → K+2+Base
Y → L+1+Base
Z → K+2+Base

图37

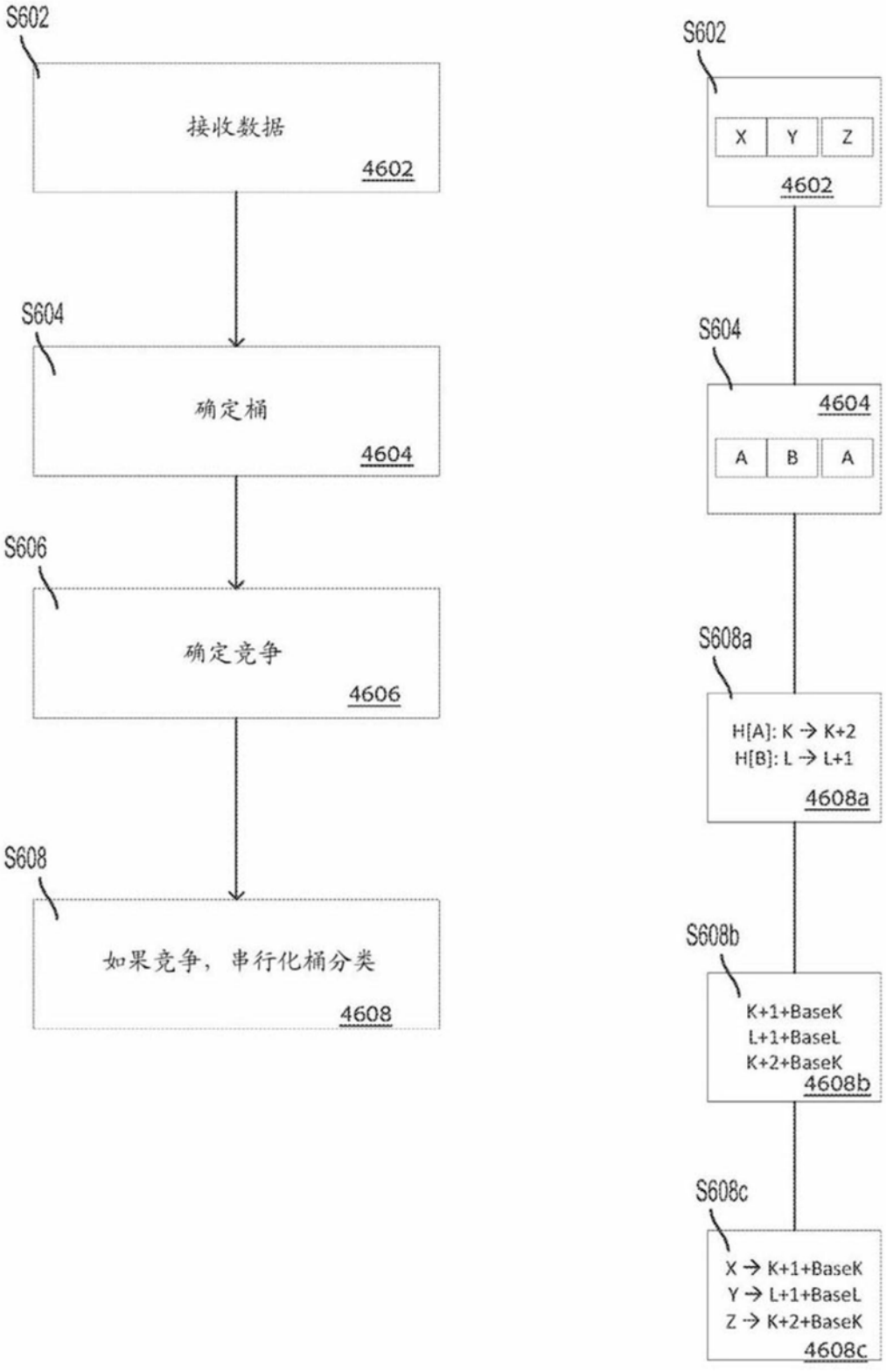

S602
接收数据
4602
S604
确定桶
4604
S606
确定竞争
4606
S608
如果竞争，串行化桶分类
4608
S602
X
Y
Z
4602
S604
4604
A
B
A
S608a
H[A]: K → K+2
H[B]: L → L+1
4608a
S608b
K+1+BaseK
L+1+BaseL
K+2+BaseK
4608b
S608c
X → K+1+BaseK
Y → L+1+BaseL
Z → K+2+BaseK
4608c

图38

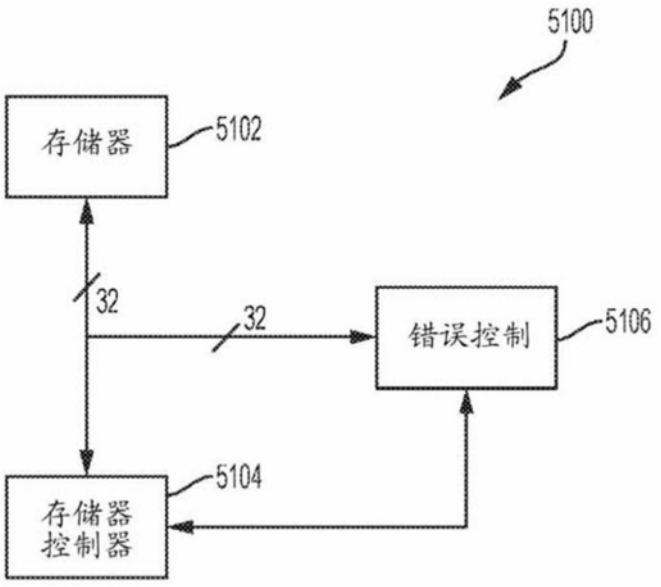

5100
存储器
5102
32
32
错误控制
5106
5104
存储器
控制器

图39

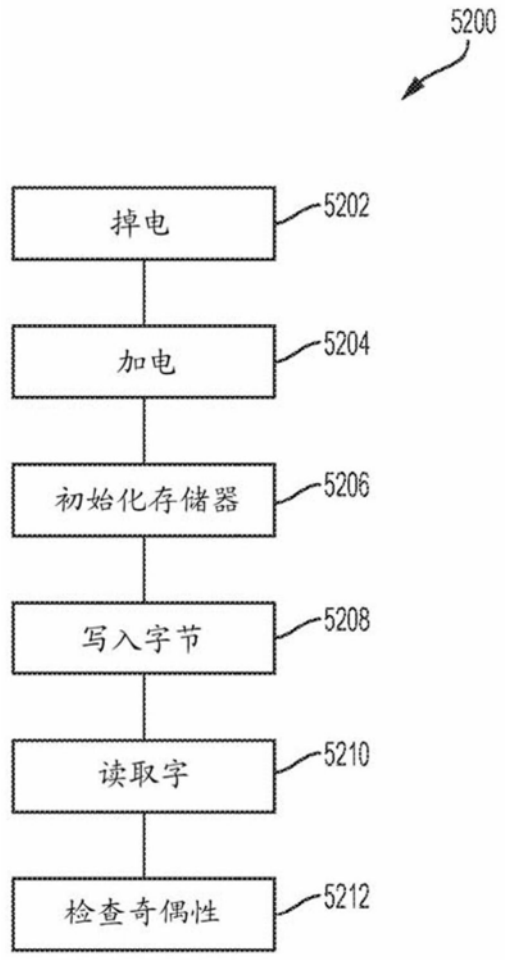

5200
掉电
5202
加电
5204
初始化存储器
5206
写入字节
5208
读取字
5210
检查奇偶性
5212

图40

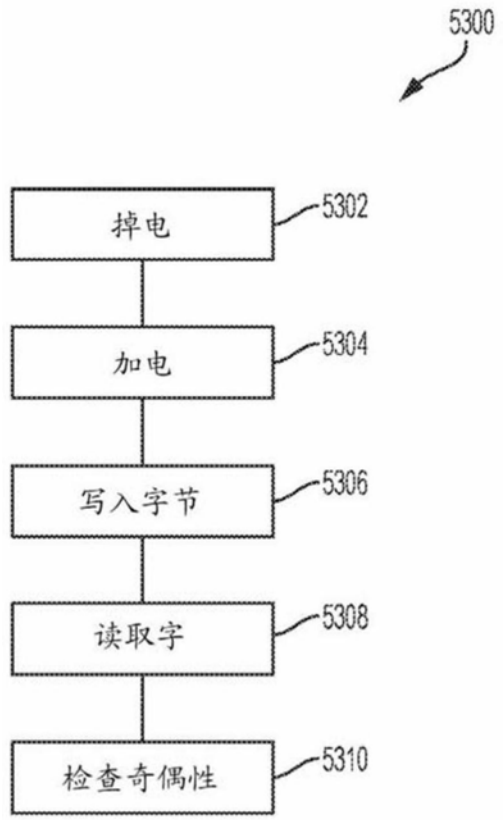

5300
掉电
5302
加电
5304
写入字节
5306
读取字
5308
检查奇偶性
5310

图41

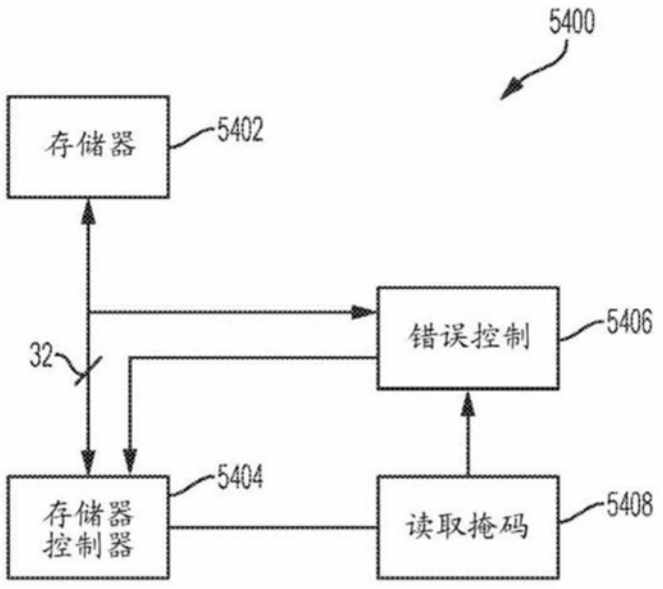

5400
存储器
5402
32
错误控制
5406
存储器
控制器
5404
读取掩码
5408

图42

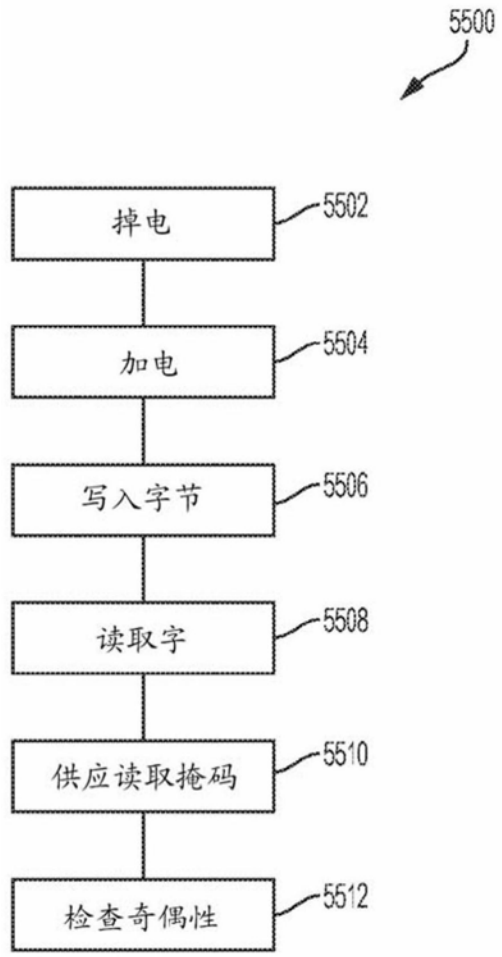

5500
掉电
5502
加电
5504
写入字节
5506
读取字
5508
供应读取掩码
5510
检查奇偶性
5512

图43

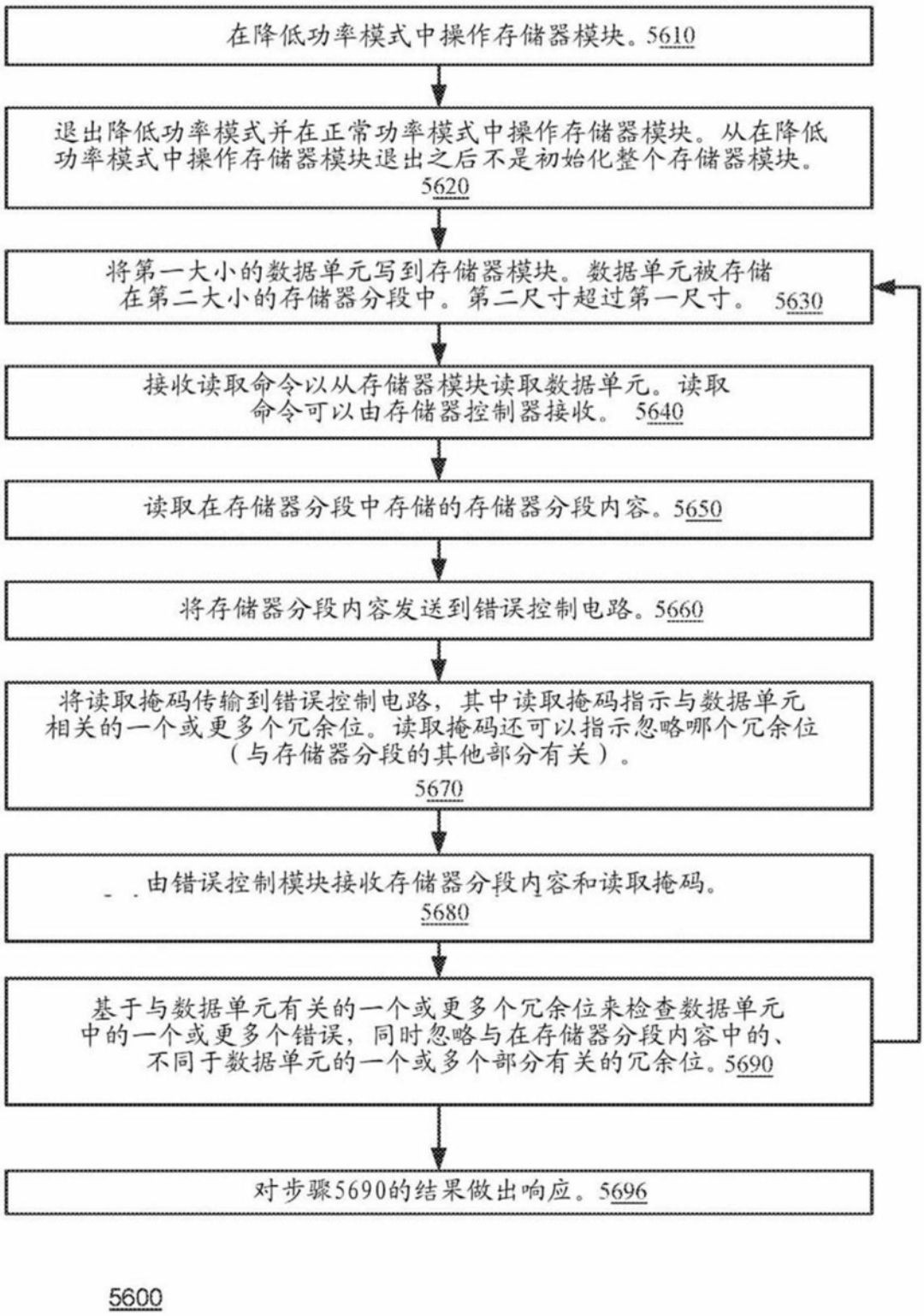

在降低功率模式中操作存储器模块。5610
退出降低功率模式并在正常功率模式中操作存储器模块。从在降低功率模式中操作存储器模块退出之后不是初始化整个存储器模块。5620
将第一大小的数据单元写到存储器模块。数据单元被存储在第二大小的存储器分段中。第二尺寸超过第一尺寸。5630
接收读取命令以从存储器模块读取数据单元。读取命令可以由存储器控制器接收。5640
读取在存储器分段中存储的存储器分段内容。5650
将存储器分段内容发送到错误控制电路。5660
将读取掩码传输到错误控制电路，其中读取掩码指示与数据单元相关的一个或更多个冗余位。读取掩码还可以指示忽略哪个冗余位（与存储器分段的其他部分有关）。5670
由错误控制模块接收存储器分段内容和读取掩码。5680
基于与数据单元有关的一个或更多个冗余位来检查数据单元中的一个或更多个错误，同时忽略与在存储器分段内容中的、不同于数据单元的一个或多个部分有关的冗余位。5690
对步骤5690的结果做出响应。5696
5600

图44

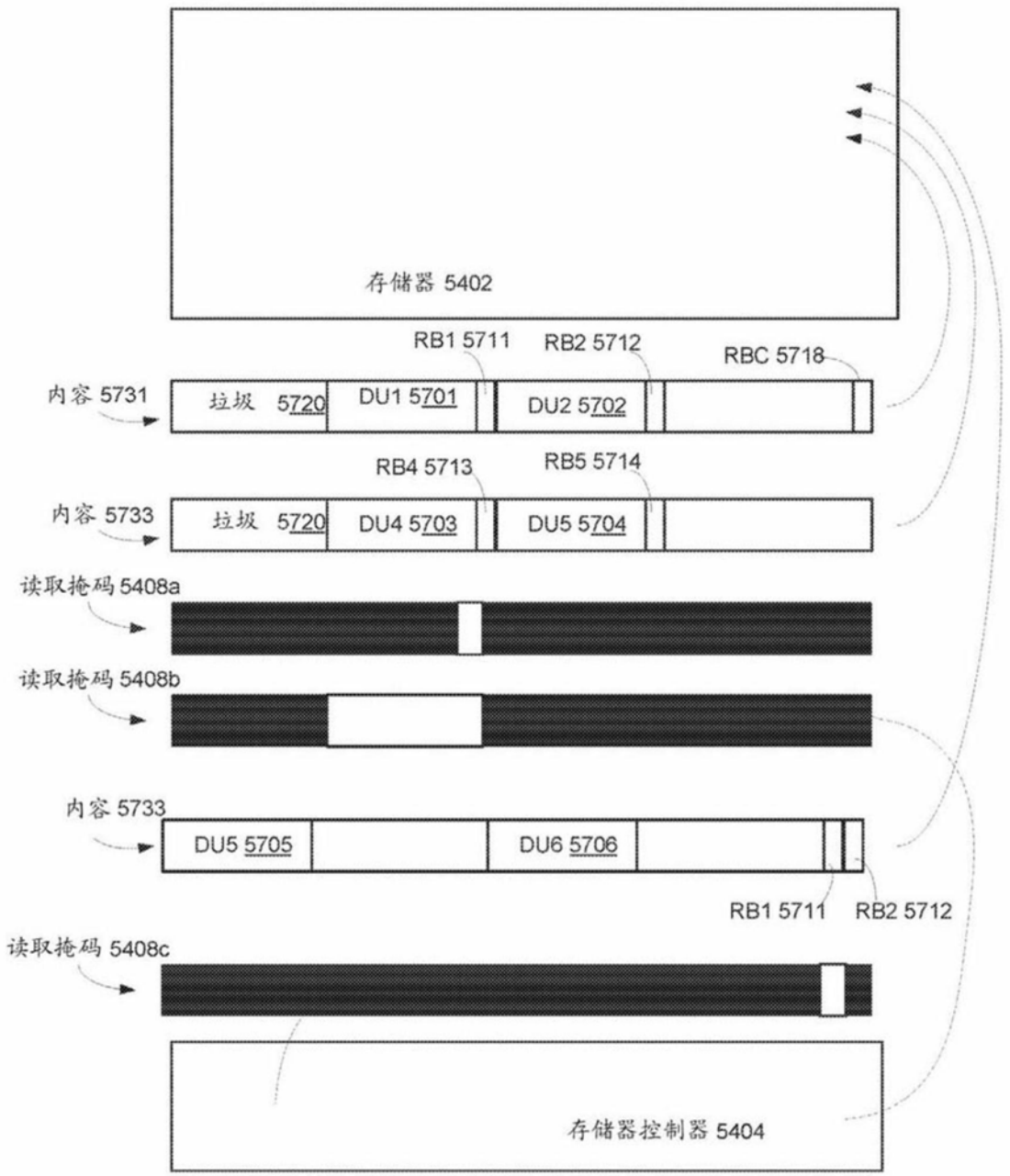

存储器 5402
RB1 5711
RB2 5712
RBC 5718
内容 5731
垃圾 5720
DU1 5701
DU2 5702
RB4 5713
RB5 5714
内容 5733
垃圾 5720
DU4 5703
DU5 5704
读取掩码 5408a
读取掩码 5408b
内容 5733
DU5 5705
DU6 5706
RB1 5711
RB2 5712
读取掩码 5408c
存储器控制器 5404

图45

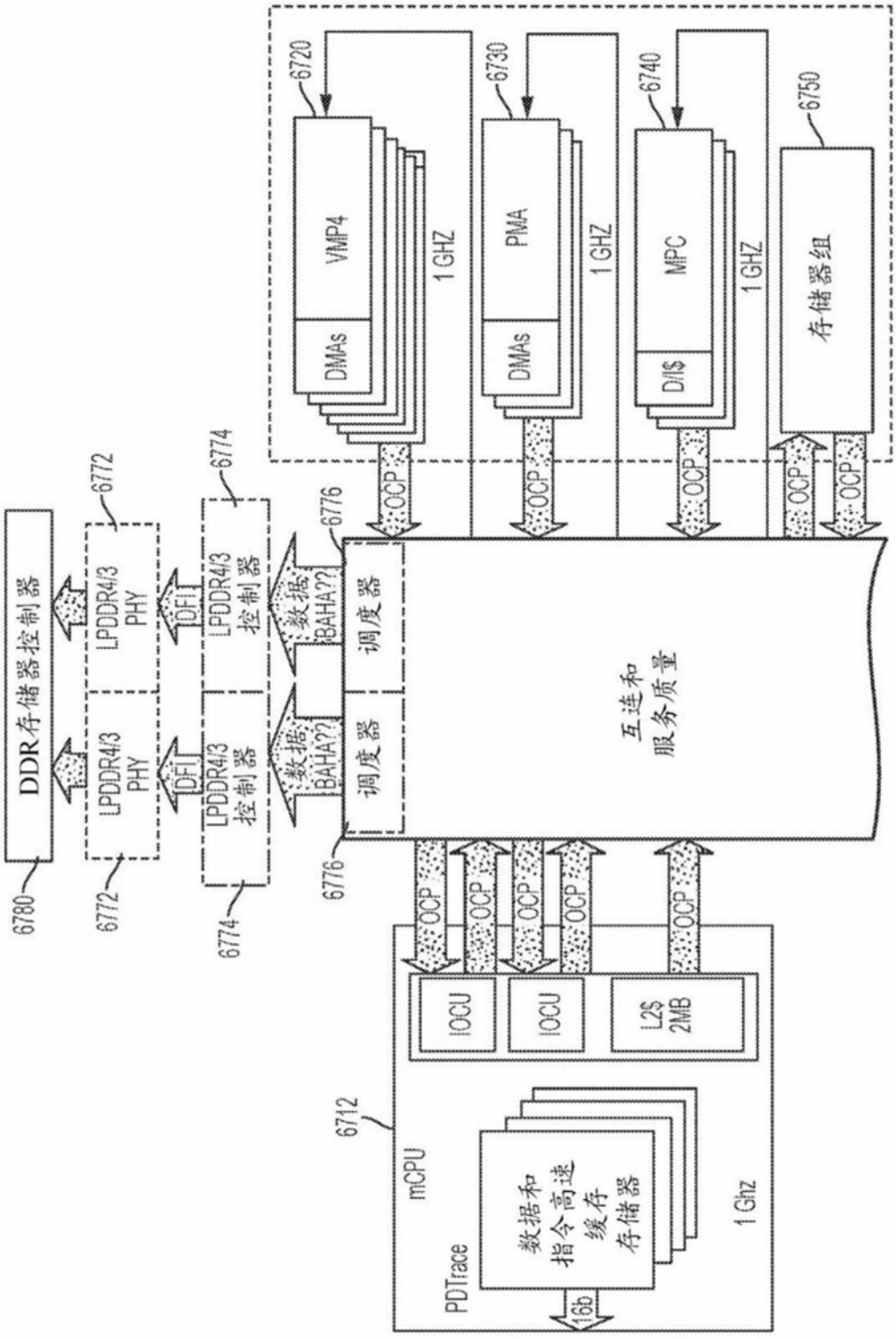

6720
VMP4
DMAs
1 GHZ
6730
PMA
DMAs
1 GHZ
6740
MPC
D/I$
1 GHZ
6750
存储器组
OCP
6772
6774
6776
6780
DDR存储器控制器
LPDDR4/3 PHY
DFI
LPDDR4/3 控制器
数据 BAHA??
调度器
互连和服务质量
IOCU
IOCU
L2$ 2MB
6712
mCPU
PDTrace
数据和指令高速缓存存储器
16b
1 Ghz

图46

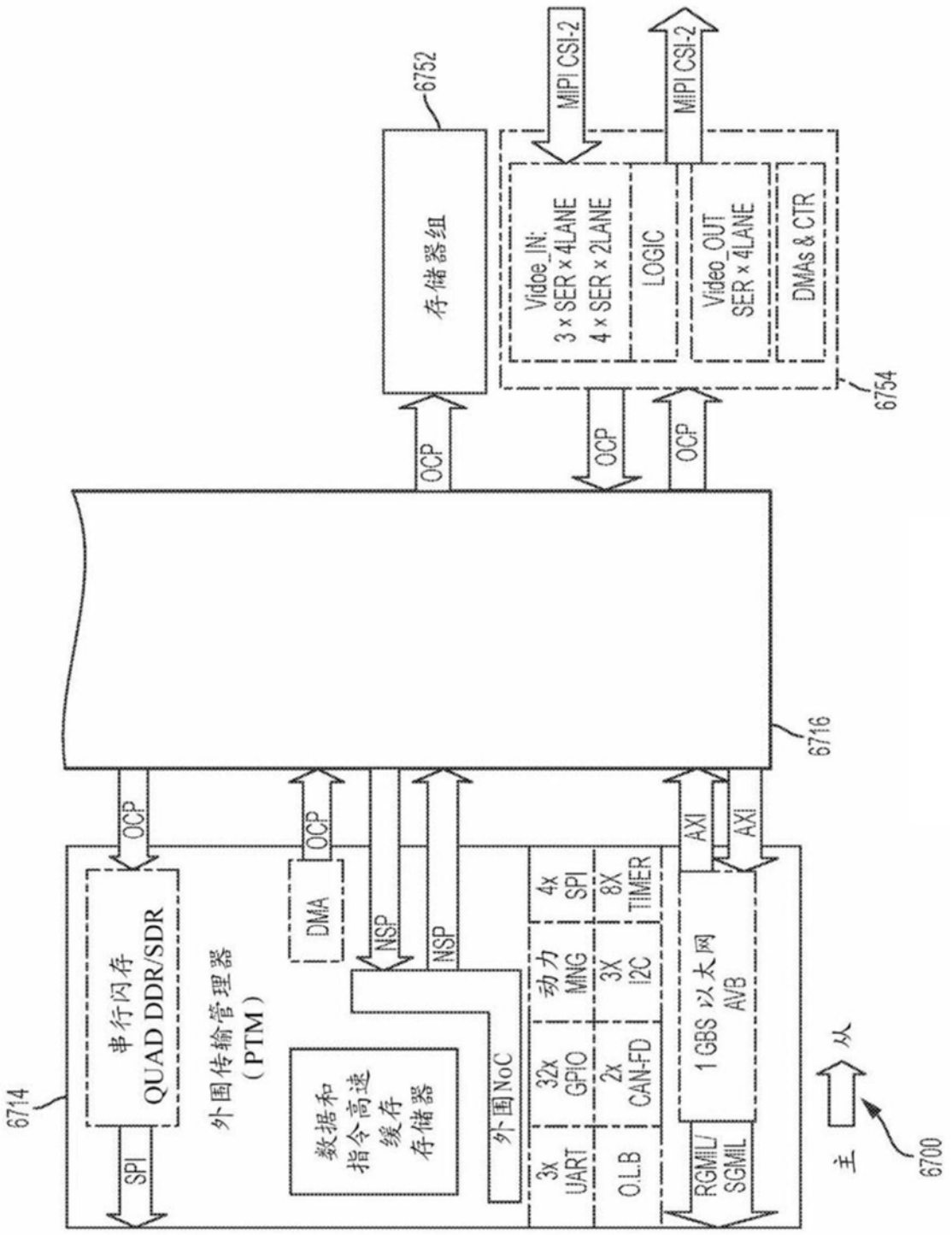

6752
存储器组
MIPI CSI-2
MIPI CSI-2
Vidoe_IN:
3 x SER x 4LANE
4 x SER x 2LANE
LOGIC
Video_OUT
SER x 4LANE
DMAs & CTR
6754
OCP
OCP
OCP
6716
串行闪存
QUAD DDR/SDR
外围传输管理器
(PTM)
DMA
NSP
NSP
AXI
AXI
4x
SPI
8X
TIMER
动力
MNG
3X
I2C
32x
GPIO
2x
CAN-FD
1 GBS 以太网
AVB
数据和
指令高速
缓存
存储器
外围 NoC
3x
UART
O.L.B
RGMII/
SGMIL
SPI
6714
主
从
6700

图46继续

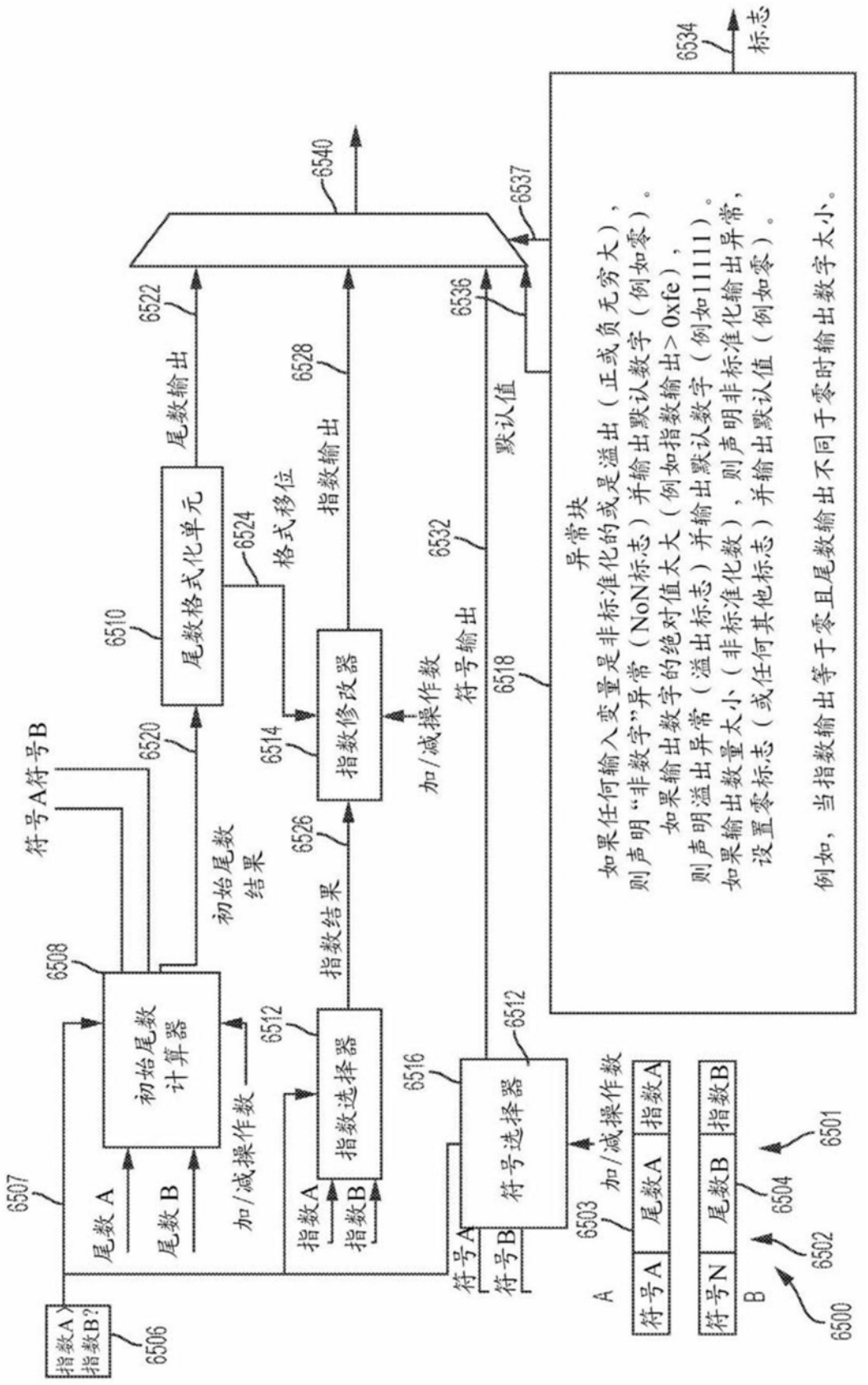

6500
6501
6502
6503
6504
6506
6507
6508
6510
6512
6514
6516
6518
6520
6522
6524
6526
6528
6532
6534
6536
6537
6540
指数A
指数B?
尾数 A
尾数 B
加/减操作数
初始尾数
计算器
初始尾数
结果
符号A符号B
指数选择器
指数结果
指数A
指数B
尾数格式化单元
格式移位
尾数输出
指数修改器
指数输出
加/减操作数
符号输出
符号选择器
符号A
符号B
默认值
异常块
如果任何输入变量是非标准化的或是溢出（正或负无穷大），
则声明“非数字”异常（NoN标志）并输出默认数字（例如零）。
如果输出数字的绝对值太大（例如指数输出>0xfe），
则声明溢出异常（溢出标志）并输出默认数字（例如111111）。
如果输出数量太小（非标准化数），则声明非标准化输出异常，
设置零标志（或任何其他标志）并输出默认值（例如零）。
例如，当指数输出等于零且尾数输出不同于零时输出数字太小。
标志
符号A
尾数A
指数A
符号N
尾数B
指数B
A
B

图47

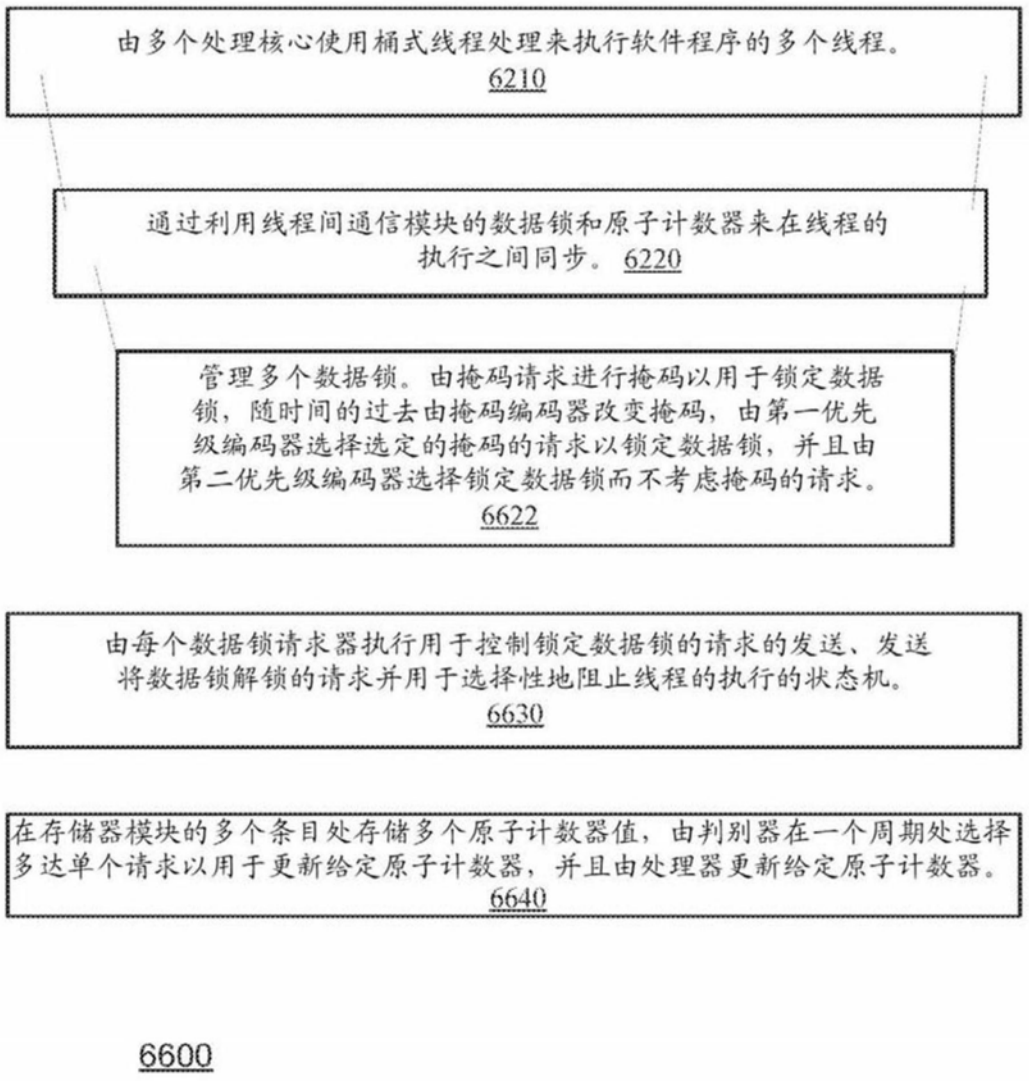

由多个处理核心使用桶式线程处理来执行软件程序的多个线程。
6210
通过利用线程间通信模块的数据锁和原子计数器来在线程的执行之间同步。 6220
管理多个数据锁。由掩码请求进行掩码以用于锁定数据锁，随时间的过去由掩码编码器改变掩码，由第一优先级编码器选择选定的掩码的请求以锁定数据锁，并且由第二优先级编码器选择锁定数据锁而不考虑掩码的请求。
6622
由每个数据锁请求器执行用于控制锁定数据锁的请求的发送、发送将数据锁解锁的请求并用于选择性地阻止线程的执行的状态机。
6630
在存储器模块的多个条目处存储多个原子计数器值，由判别器在一个周期处选择多达单个请求以用于更新给定原子计数器，并且由处理器更新给定原子计数器。
6640
6600

图48

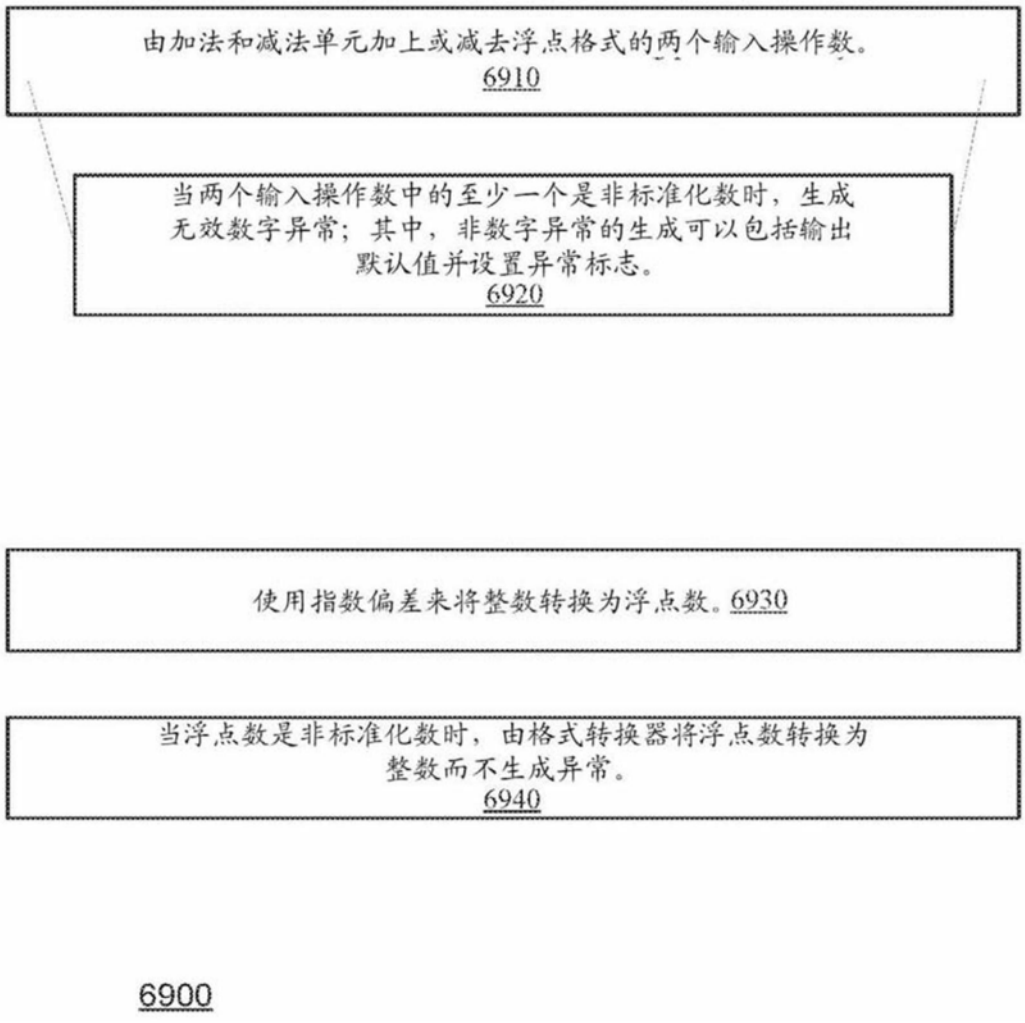

由加法和减法单元加上或减去浮点格式的两个输入操作数。
6910
当两个输入操作数中的至少一个是非标准化数时，生成无效数字异常；其中，非数字异常的生成可以包括输出默认值并设置异常标志。
6920
使用指数偏差来将整数转换为浮点数。6930
当浮点数是非标准化数时，由格式转换器将浮点数转换为整数而不生成异常。
6940
6900

图49