(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 110337635 B
(45)授权公告日 2023.09.19

(21)申请号 201780086894.2
(22)申请日 2017.07.01
(65)同一申请的已公布的文献号
申请公布号 CN 110337635 A
(43)申请公布日 2019.10.15
(30)优先权数据
62/473,732 2017.03.20 US
(85)PCT国际申请进入国家阶段日
2019.08.20
(86)PCT国际申请的申请数据
PCT/US2017/040534 2017.07.01
(87)PCT国际申请的公布数据
WO2018/174925 EN 2018.09.27
(73)专利权人 英特尔公司
地址 美国加利福尼亚州
(72)发明人 R·凡伦天 D·鲍姆 Z·斯波伯
J·考博尔
E·乌尔德-阿迈德-瓦尔
B·L·托尔 M·J·查尼
M·阿德尔曼 B·泽维
A·海内克 S·卢巴诺维奇
(74)专利代理机构 上海专利商标事务所有限公司 31100
专利代理师 李炜 黄嵩泉
(51)Int.Cl.
G06F 9/30(2006.01)
(56)对比文件
US 2005193050 A1,2005.09.01
US 6393554 B1,2002.05.21
CN 1707426 A,2005.12.14
US 2004111587 A1,2004.06.10
CN 102360344 A,2012.02.22
CN 104137055 A,2014.11.05
US 2006101245 A1,2006.05.11
US 2008071851 A1,2008.03.20
Miao Hu.Dot-product engine for neuromorphic computing:programming 1T1M crossbar to accelerate matrix-vector multiplication.《DAC 16:Proceedings of the 53rd Annual Design Automation Conference》.2016,1-6.
审查员 姚晓斌

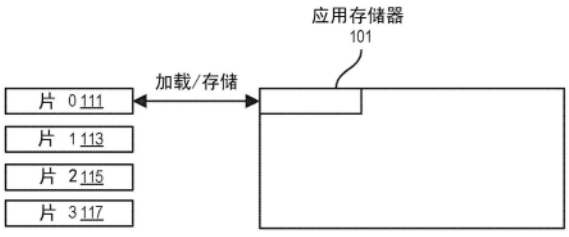
权利要求书2页 说明书33页 附图45页

(54)发明名称

用于点积操作的系统、方法和装置

(57)摘要

本文中详述的实施例涉及矩阵操作。例如，详述了对矩阵(片)点积操作的指令支持的实施例。示例性指令包括:计算有符号字的点积,并利用饱和累加在双字中;计算字节的点积,并利用饱和累加到双字中,其中,输入字节可以是有符号或无符号的,并且双字累加具有输出饱和;等等。



CN 110337635 B

1.一种处理器,包括:

解码电路,用于对指令解码,所述指令具有用于第一源矩阵操作数、第二源矩阵操作数和目的地矩阵操作数的字段,其中,所述第一源矩阵操作数、所述第二源矩阵操作数和所述目的地矩阵操作数中的每一个都包括数据元素的多个矩阵;以及

执行电路,用于执行经解码的指令以:

通过对来自所述第一源矩阵操作数和所述第二源矩阵操作数的元素执行点积操作来计算结果,以及

将所述结果累加到所述目的地矩阵操作数的数据元素位置中。

2.如权利要求1所述的处理器,其中,所述第一源矩阵操作数和所述第二源矩阵操作数的元素是有符号字元素,并且其中,所述目的地矩阵操作数的元素是有符号双字。

3.如权利要求1所述的处理器,其中,所述第一源矩阵操作数和所述第二源矩阵操作数的元素是有符号字尺寸的元素,并且其中,所述目的地矩阵操作数的元素是有符号四字尺寸的元素。

4.如权利要求1所述的处理器,其中,所述第一源矩阵操作数和所述第二源矩阵操作数的元素是字节尺寸的元素,并且其中,所述目的地矩阵操作数的元素是双字尺寸的元素。

5.如权利要求1所述的处理器,其中,所述第一源矩阵操作数的元素是字节尺寸的元素且所述第二源矩阵操作数的元素是4位尺寸的元素,并且其中,所述目的地矩阵的元素是双字尺寸的元素。

6.如权利要求1所述的处理器,其中,所述第一源矩阵操作数的元素和所述第二源矩阵操作数的元素是4位尺寸的元素,并且其中,所述目的地矩阵的元素是双字尺寸的元素。

7.如权利要求1所述的处理器,其中,所述结果是利用饱和计算出的。

8.如权利要求1所述的处理器,其中,所述执行电路包括多个融合乘法加法器。

9.如权利要求1所述的处理器,其中,所述指令指示所述第一源矩阵操作数和所述第二源矩阵操作数中的至少一者包含无符号数据值。

10.如权利要求1所述的处理器,其中,当所述第一源矩阵操作数具有的列数与所述第二源矩阵操作数具有的行数不同时,错误被生成。

11.如权利要求1所述的处理器,其中,当所述目的地矩阵操作数的行数与所述第一源矩阵操作数的行数不同时,错误被生成。

12.如权利要求1所述的处理器,其中,当所述目的地矩阵操作数的列数与所述第二源矩阵操作数的列数不同时,错误被生成。

13.一种用于计算机处理器的方法,包括:

对指令解码,所述指令具有用于第一源矩阵操作数、第二源矩阵操作数和目的地矩阵操作数的字段,其中,所述第一源矩阵操作数、所述第二源矩阵操作数和所述目的地矩阵操作数中的每一个都包括数据元素的多个矩阵;以及

执行经解码的指令以:

通过对来自所述第一源矩阵操作数和所述第二源矩阵操作数的元素执行点积操作来计算结果,以及

将所述结果累加到所述目的地矩阵操作数的数据元素位置中。

14.如权利要求13所述的方法,其中,所述第一源矩阵操作数和所述第二源矩阵操作数

的元素是有符号字元素，并且其中，所述目的地矩阵操作数的元素是有符号双字。

15.如权利要求13所述的方法，其中，所述第一源矩阵操作数和所述第二源矩阵操作数的元素是有符号字尺寸的元素，并且其中，所述目的地矩阵操作数的元素是有符号四字尺寸的元素。

16.如权利要求13所述的方法，其中，所述第一源矩阵操作数和所述第二源矩阵操作数的元素是字节尺寸的元素，并且其中，所述目的地矩阵操作数的元素是双字尺寸的元素。

17.如权利要求13所述的方法，其中，所述第一源矩阵操作数的元素是字节尺寸的元素且所述第二源矩阵操作数的元素是4位尺寸的元素，并且其中，所述目的地矩阵的元素是双字尺寸的元素。

18.如权利要求13所述的方法，其中，所述第一源矩阵操作数的元素和所述第二源矩阵操作数的元素是4位尺寸的元素，并且其中，所述目的地矩阵的元素是双字尺寸的元素。

19.如权利要求13所述的方法，其中，所述结果是利用饱和计算出的。

20.如权利要求13所述的方法，其中，所述执行电路包括多个融合乘法加法器。

21.如权利要求13所述的方法，其中，所述指令指示所述第一源矩阵操作数和所述第二源矩阵操作数中的至少一者包含无符号数据值。

22.如权利要求13所述的方法，其中，当所述第一源矩阵操作数具有的列数与所述第二源矩阵操作数具有的行数不同时，错误被生成。

23.如权利要求13所述的方法，其中，当所述目的地矩阵操作数的行数与所述第一源矩阵操作数的行数不同时，错误被生成。

24.如权利要求13所述的方法，其中，当所述目的地矩阵操作数的列数与所述第二源矩阵操作数的列数不同时，错误被生成。

25.一种机器可读介质，包括代码，所述代码当被执行时使机器执行如权利要求13-24中的任一项所述的方法。

用于点积操作的系统、方法和装置

技术领域

[0001] 本发明的领域总体上涉及计算机处理器架构,更具体地涉及矩阵操纵。

背景技术

[0002] 在诸如机器学习和其他批量数据处理之类的许多计算任务中,矩阵正变得日益重要。

附图说明

[0003] 在所附附图中以示例方式而非限制方式说明本发明,在附图中,类似的附图标记指示类似的要素,其中:

[0004] 图1图示经配置的片(tile)的实施例;

[0005] 图2图示矩阵存储的若干示例;

[0006] 图3图示利用矩阵(片)操作加速器的系统的实施例;

[0007] 图4和图5示出如何使用矩阵操作加速器来共享存储器的不同实施例;

[0008] 图6图示使用片的矩阵乘法累加操作(“TMMA”)的实施例;

[0009] 图7图示链式融合乘法累加指令的迭代的执行的子集的实施例;

[0010] 图8图示链式融合乘法累加指令的迭代的执行的子集的实施例;

[0011] 图9图示链式融合乘法累加指令的迭代的执行的子集的实施例;

[0012] 图10图示链式融合乘法累加指令的迭代的执行的子集的实施例;

[0013] 图11图示根据实施例的尺寸为2的幂的SIMD实现方式,其中,累加器使用比至乘法器的输入的尺寸大的输入尺寸;

[0014] 图12图示利用矩阵操作电路的系统的实施例;

[0015] 图13图示处理器核流水线的实施例,该处理器核流水线支持使用片的矩阵操作;

[0016] 图14图示处理器核流水线的实施例,该处理器核流水线支持使用片的矩阵操作;

[0017] 图15图示按行为主格式和列为主格式表达的矩阵的示例;

[0018] 图16图示矩阵(片)的使用的示例;

[0019] 图17图示矩阵(片)的使用的方法的实施例;

[0020] 图18图示TILECONFIG指令的示例性执行;

[0021] 图19(A)-图19(D)图示(多个)寄存器的示例;

[0022] 图20图示将被支持的矩阵(片)的描述的实施例;

[0023] 图21图示由处理器执行以处理TILECONFIG指令的方法的实施例;

[0024] 图22图示使用存储器寻址来执行TILECONFIG指令的更详细的描述;

[0025] 图23图示TILECONFIG指令的执行的示例性伪代码;

[0026] 图24图示TILEDOTPRODUCT指令的示例性执行;

[0027] 图25图示由处理器执行以处理矩阵(片)点积指令的方法的实施例;

[0028] 图26图示与由处理器执行以执行TILEDOTPRODUCT指令的示例方法有关的附加细

节；

[0029] 图27A-图27G图示用于执行TILEDOTPRODUCT操作的示例方法；

[0030] 图28(A)-图28(C)图示示例性指令格式；

[0031] 图29是根据本发明的一个实施例的寄存器架构的框图；

[0032] 图30A-图30B图示有序流水线和有序核；

[0033] 图31A-图31B图示更具体的示例性有序核架构的框图，该核将是芯片中的若干逻辑块(包括相同类型和/或不同类型的其他核)中的一个逻辑块；

[0034] 图32是根据本发明的实施例的可具有多于一个的核、可具有集成存储器控制器、并且可具有集成图形器件的处理器的框图；

[0035] 图33-图36是示例性计算机架构的框图；并且

[0036] 图37是根据本发明的实施例的对照使用软件指令转换器将源指令集中的二进制指令转换成目标指令集中的二进制指令的框图。

具体实施方式

[0037] 在以下描述中，陈述了众多特定细节。然而，应当理解，可在没有这些特定细节的情况下实践本发明的实施例。在其他实例中，未详细示出公知的电路、结构和技术，以免使对本描述的理解模糊。

[0038] 说明书中对“一个实施例”、“实施例”、“示例实施例”等的引用表明所描述的实施例可以包括特定的特征、结构或特性，但是每个实施例不一定都包括该特定的特征、结构或特性。此外，此类短语不一定是指同一个实施例。此外，当结合实施例描述特定的特征、结构或特性时，认为结合无论是否被明确描述的其他实施例而影响此类特征、结构或特性是在本领域技术人员的知识范围之内的。

[0039] 在许多主流处理器中，处置矩阵是困难的和/或指令密集性任务。例如，可将矩阵的多行置入多个紧缩数据(例如，SIMD或向量)寄存器中，随后可单独地对矩阵的多行进行操作。例如，取决于数据尺寸，将两个8x2矩阵相加可能要求加载或聚集到四个紧缩数据寄存器中。随后，执行对与来自每个矩阵的第一行对应的紧缩数据寄存器的第一加法，并执行对与来自每个矩阵的第二行对应的紧缩数据寄存器的第二加法。随后，将所得到的紧缩数据寄存器往回分散到存储器。虽然对于小型矩阵，这种场景可能是可接受的，但是对于较大的矩阵，这通常是不可接受的。

[0040] I.高层级讨论

[0041] 本文中描述的是用于在诸如中央处理单元(CPU)、图形处理单元(GPU)和加速器之类的计算机硬件中支持矩阵操作的机制。矩阵操作利用表示存储器的一个或多个紧缩区域(诸如，寄存器)的2维(2-D)数据结构。贯穿本说明书，这些2-D数据结构被称为片。注意，矩阵可以比片小(使用少于片的全部)，或可利用多个片(矩阵大于任一片的尺寸)。贯穿本说明书，使用矩阵(片)语言来指示使用影响矩阵的片来执行的操作，那个矩阵是否大于任一片通常是不相关的。

[0042] 每个片可由不同的操作来作用，这些操作诸如本文中详述的那些操作，包括但不限于：矩阵(片)乘法、片加法、片减法、片对角线、片归零、片转置、片点积、片广播、片行广播、片列广播、片乘法、片乘法和累加、片移动，等等。此外，在未来可以与这些操作一起使用

或为了支持非数值应用而使用对诸如使用缩放和/或偏置的操作器的支持，非数值应用例如，OpenCL“本地存储器”、数据压缩/解压缩，等等。

[0043] 存储(诸如，(非易失性和易失性的)存储器、寄存器、高速缓存等)的多个部分被布置为具有不同横向尺度和纵向尺度的片。例如，片可具有横向尺度4(例如，矩阵的四行)和纵向尺度8(例如，矩阵的8列)。典型地，横向尺度是相关的元素尺寸(例如，2位、4位、8位、16位、32位、64位、128位等)。可支持多种数据类型(单精度浮点、双精度浮点、整数等)。

[0044] A.经配置的片的示例性使用

[0045] 图1图示经配置的片的实施例。如图所示，存在从应用存储器101加载的四个片111、113、115和117。在该示例中，片T0 111和T1 113具有带有4元素字节(例如，单精度数据)的M行和N列。片T2 115和片T3 117具有带有8元素字节(例如，双精度数据)的M行和N/2列。由于双精度操作数的宽度是单精度操作数的两倍，因此该配置与用于提供片选项的调色板一致，将至少16*N*M字节的总存储提供给至少4个名称。取决于所使用的指令编码方案，可用的片的数量有所不同。

[0046] 在一些实施例中，片参数是可定义的。例如，“调色板”用于提供片选项。示例性选项包括但不限于：片名称的数量、存储的行中的字节数、片中的行数和列数，等等。例如，片的最大“高度”(行数)可定义为：

[0047] 片最大行＝所构造的存储/(调色板名称的数量*每行的字节数)

[0048] 由此，可写入应用，使得名称的固定使用将能够利用跨实现方式的不同存储尺寸。

[0049] 使用片配置(“TILECONFIG”)指令完成对片的配置，其中，在所选择的调色板中定义特定的片使用。该声明包括要使用的片名称的数量、每个名称(片)的所请求的行数和列数，并且在一些实施例中包括每个片的所请求的数据类型。在一些实施例中，在TILECONFIG指令的执行期间执行一致性校验，以确定其匹配调色板条目的限制。

[0050] B.示例性片存储类型

[0051] 图2图示矩阵存储的若干示例。在(A)中，片被存储在存储器中。如图所示，每“行”由四个紧缩数据元素组成。为了达到下一“行”，使用跨步值。注意，行可被连续地存储在存储器中。当片存储不映射底层存储器阵列行宽度时，跨步式存储器访问允许对一行到随后对下一行的访问。

[0052] 从存储器加载片以及向存储器存储片典型地是从应用存储器到紧缩的数据行的跨步式访问。示例性TILELOAD和TILESTORE指令或作为加载操作指令中的TILE(片)操作数的对应用存储器的其他指令引用在一些实施例中是可重新开始的，以针对每条指令处置(高达)2*行的页错误、未掩码的浮点异常和/或中断。

[0053] 在(B)中，矩阵存储在由多个寄存器组成的片中，这些寄存器诸如，紧缩数据寄存器(单指令多数据(SIMD)或向量寄存器)。在该示例中，片被叠加在三个物理寄存器上。典型地，使用连续的寄存器，然而，情况不必是这样。

[0054] 在(C)中，矩阵被存储在可由在片操作中使用的融合乘法累加(FMA)电路访问的非寄存器存储中的片中。该存储可在FMA内部，或邻近FMA。此外，在一些实施例中，如下文所讨论，该存储可用于数据元素，而不是整行或片。

[0055] 经由CPUID报告TMMA架构的所支持的参数。在一些实施例中，信息列表包括最大高度和最大SIMD尺度。配置TMMA架构要求指定每个片的尺度、每个片的元素尺寸以及调色板

标识符。通过执行TILECONFIG指令来完成该配置。

[0056] TILECONFIG指令的成功执行启用后续的TILE操作器。TILERELEASEALL指令清除片配置，并禁用TILE操作(直到下一TILECONFIG指令执行)。在一些实施例中，在使用片的上下文切换中使用XSAVE、XSTORE等。在一些实施例中，在XSAVE中使用2个XCR0位，一个用于TILECONFIG元数据，一个位与实际的片有效载荷数据对应。

[0057] TILECONFIG不仅配置片使用，还设置状态变量，该状态变量指示在片经配置的情况下程序在代码区域中。实现方式可枚举对可与片区域一起使用的其他指令的限制，诸如，没有对现有寄存器组的使用，等等。

[0058] 退出片区域典型地利用TILERELEASEALL指令来完成。该指令不取参数并迅速使所有片无效(指示数据不再需要任何保存或恢复)，并且清除与处于片区域中对应的内部状态。

[0059] 在一些实施例中，片操作将使超出由片配置指定的尺度的任何行和任何列归零。例如，随着每一行被写入，片操作将使超出所配置的列数(将元素的尺寸考虑在内)的数据归零。例如，对于64字节的行以及配置有10行和12列的片，写入FP32元素的操作将以12*4字节向前10行中的每一行写入输出/结果数据，并且使每一行中的其余的4*4字节归零。片操作还对前10个经配置的行之后的任何行完全归零。当使用具有64字节的行的1K的片时，将会有16行，因此，在该示例中，最后6行也将被归零。

[0060] 在一些实施例中，当加载数据时，上下文恢复(例如，XRSTOR)强制使超出片的所配置的行的数据将被维持为零。如果没有有效配置，则所有行被归零。对片数据的XRSTOR能够加载超出那些所配置的列的列中的无用信息。XRSTOR对超出所配置的列数进行清除不应当是可能的，因为不存在与片配置相关联的元素宽度。

[0061] 当将整个TILE存储区写入存储器时，上下文保存(例如，XSAVE)暴露整个TILE存储区。如果XRSTOR将无用数据加载到片的最右边部分中，则将由XSAVE保存那个数据。对于超出为每个片指定的数量的行，XSAVE将写入零。

[0062] 在一些实施例中，片指令是可重新开始的。访问存储器的操作允许在页错误之后重新开始。凭借受控制和/或状态寄存器控制的对异常的掩码，处理浮点操作的计算指令也允许未掩码的浮点异常。

[0063] 为了支持在这些事件之后重新开始指令，指令将信息存储在下文详述的起始寄存器中。

[0064] II.矩阵(片)操作系统

[0065] A.示例性硬件支持

[0066] 图3图示利用矩阵(片)操作加速器的系统的实施例。在该图示中，主机处理器/处理系统301将命令311(例如，矩阵操纵操作，诸如，算术或矩阵操纵操作、或加载和存储操作)传递至矩阵操作加速器307。然而，这以这种方式示出，仅用于讨论的目的。如稍后所详述，该加速器307可以是处理核的部分。典型地，作为片操纵操作器指令的命令311将片称为寄存器-寄存器(“reg-reg”)或寄存器-存储器(“reg-mem”)格式。诸如TILESTORE、TILELOAD、TILECONFIG等的其他命令不对片执行数据操作。命令可以是供加速器307处置的经解码的指令(例如，微操作)或宏指令。

[0067] 在该示例中，一致性存储器接口303耦合至主机处理器/处理系统301和矩阵操作

加速器405，使得它们能够共享存储器。图4和图5示出如何使用矩阵操作加速器来共享存储器的不同实施例。如图4中所示，主机处理器401和矩阵操作加速器电路405共享同一存储器403。图5图示其中主机处理器501和矩阵操作加速器505不共享存储器，但可访问彼此的存储器的实施例。例如，处理器501可访问片存储器507，并照常利用其主机存储器503。类似地，矩阵操作加速器505可访问主机存储器503，但更典型地使用其自身的存储器507。注意，这些存储器可以是不同类型的。

[0068] 矩阵操作加速器307包括耦合至数据缓冲器305的多个FMA 309(在一些实现方式中，这些缓冲器305中的一个或多个被存储在如图所示的网格的FMA中)。数据缓冲器305对(例如，使用片加载或片存储指令)从存储器加载的片和/或向存储器存储的片进行缓冲。数据缓冲器可以是例如多个寄存器。典型地，这些FMA被布置为能够读取和写入片的链式FMA 309的网格。在该示例中，矩阵操作加速器307用于使用片T0、T1和T2来执行矩阵乘法操作。片中的至少一个片被容纳在FMA网格309中。在一些实施例中，操作中的所有片都被存储在FMA网格309中。在其他实施例中，仅子集被存储在FMA网格309中。如图所示，T1被容纳，而T0和T2不被容纳。注意，A、B和C是指这些片的矩阵，这些矩阵可以占据或可以不占据片的整个空间。

[0069] 图6图示使用片的矩阵乘法累加操作(“TMMA”)的实施例。

[0070] 矩阵(片A 601)中的行数与串联的(链式)FMA的数量匹配，这些串联的FMA包括计算的等待时间。实现方式可自由地在更小高度的网格上再循环，但是计算保持相同。

[0071] 源/目的地向量来自N行的片(片C 605)，并且FMA的网格611执行N个向量-矩阵操作，从而导致执行片的矩阵乘法的完整指令。片B 603是另一向量源，并将“广播”项提供给每一级中的FMA。

[0072] 在操作中，在一些实施例中，(存储在片B 603中的)矩阵B的元素跨FMA的矩形网格散布。(存储在片A 601中的)矩阵B使其行的元素被转置，以与FMA的矩形网格的列尺度匹配。在网格中的每个FMA处，A和B的元素被相乘，并被加到(来自附图中上方的)传入的被加数，并且传出和被传递至FMA的下一行(或最终输出)。

[0073] 单个步骤的等待时间与K(矩阵B的行高)成比例，并且从属的TMMA典型地(在单片中或跨片)具有足够的源-目的地行以隐藏该等待时间。实现方式还可跨时间步长分割SIMD(紧缩数据元素)尺度M(矩阵A的行高)，但是这仅改变K乘以的常数。当程序指定比由TMACC枚举的最大值小的K时，实现方式利用“掩码”或“早出”来自由地实现此。

[0074] 整个TMMA的等待时间与N*K成比例。重复率与N成比例。每条TMMA指令的MAC的数量为N*K*M。

[0075] 图7图示链式融合乘法累加指令的迭代的执行的子集的实施例。具体而言，图7图示目的地的一个紧缩数据元素位置的迭代的执行电路。在该实施例中，链式融合乘法累加对多个有符号源进行操作，其中累加器2倍于输入数据尺寸。

[0076] 第一有符号源(源1 701)和第二有符号源(源2 703)各自都具有四个紧缩数据元素。这些紧缩数据元素中的每一个都存储诸如浮点数据之类的有符号数据。第三有符号源(源3 709)具有两个紧缩数据元素，其中的每一个都存储有符号数据。第一有符号源701的尺寸和第二有符号源703的尺寸是第三有符号源(初始值或先前结果)709的尺寸的一半。例如，第一有符号源701和第二有符号源703可具有32位的紧缩数据元素(例如，单精度浮点)，

而第三有符号源709可具有64位的紧缩数据元素(例如,双精度浮点)。

[0077] 在该图示中,仅示出第一有符号源701和第二有符号源703的最高有效的两个紧缩数据元素位置以及第三有符号源709的最高有效的紧缩数据元素位置。当然,还将处理其他紧缩数据元素位置。

[0078] 如图所示,成对地处理紧缩数据元素。例如,使用乘法器电路705将第一有符号源701和第二有符号源703的最高有效的紧缩数据元素位置的数据相乘,并且使用乘法器电路707将来自第一有符号源701和第二有符号源703的次高有效的紧缩数据元素位置的数据相乘。在一些实施例中,这些乘法器电路705和707重新用于其他紧缩数据元素位置。在其他实施例中,使用附加的乘法器电路,使得并行地处理紧缩数据元素。在一些上下文中,使用尺寸为有符号第三源709的尺寸的通道来完成并行执行。使用加法电路711将这些乘法中的每个乘法的结果相加。

[0079] (使用不同的加法器713或同一加法器711)将这些乘法结果的加法的结果加到来自有符号源3 709的最高有效紧缩数据元素位置的数据。

[0080] 最终,第二加法的结果被存储到有符号目的地715中与来自有符号第三源709的所使用的紧缩数据元素位置对应的紧缩数据元素位置中,或者如果有下一迭代,则该第二加法的结果被继续传递到该下一迭代。在一些实施例中,将写掩码应用于此存储,使得如果对应的写掩码(位)被置位,则存储发生,如果对应的写掩码(位)未被置位,则存储不发生。

[0081] 图8图示链式融合乘法累加指令的迭代的执行的子集的实施例。具体而言,图8图示目的地的一个紧缩数据元素位置的迭代的执行电路。在该实施例中,链式融合乘法累加正对有符号源进行操作,其中,累加器2倍于输入数据的尺寸。

[0082] 第一有符号源(源1 801)和第二有符号源(源2 803)各自都具有四个紧缩数据元素。这些紧缩数据元素中的每一个都存储诸如整数数据之类的有符号数据。第三有符号源(源3 809)具有两个紧缩数据元素,其中的每一个都存储有符号数据。第一有符号源801的尺寸和第二有符号源803的尺寸是第三有符号源809的尺寸的一半。例如,第一有符号源801和第二有符号源803可具有32位的紧缩数据元素(例如,单精度浮点),而第三有符号源809可具有64位的紧缩数据元素(例如,双精度浮点)。

[0083] 在该图示中,仅示出第一有符号源801和第二有符号源803的最高有效的两个紧缩数据元素位置以及第三有符号源809的最高有效的紧缩数据元素位置。当然,还将处理其他紧缩数据元素位置。

[0084] 如图所示,成对地处理紧缩数据元素。例如,使用乘法器电路805将第一有符号源801和第二有符号源803的最高有效的紧缩数据元素位置的数据相乘,并且使用乘法器电路807将来自第一有符号源801和第二有符号源803的次高有效的紧缩数据元素位置的数据相乘。在一些实施例中,这些乘法器电路805和807重新用于其他紧缩数据元素位置。在其他实施例中,使用附加的乘法器电路,使得并行地处理紧缩数据元素。在一些上下文中,使用尺寸为有符号第三源(初始值或先前迭代结果)809的尺寸的通道来完成并行执行。使用加法/饱和电路813将多个乘法中的每个乘法的结果加到有符号第三源809。

[0085] 当加法导致过大的值时,加法/饱和(累加器)电路813保留操作数的符号。具体而言,对于多路加法与向目的地或下一迭代的写入之间的无限精度结果,饱和评估发生。当累加器813是浮点且输入项是整数时,乘积的和以及浮点累加器输入值被转换为无限精度值

(数百位的定点数),执行乘法结果与第三输入的加法,并执行向实际累加器类型的单次舍入。

[0086] 无符号饱和意味着输出值被限于那个元素宽度的最大无符号数(全1)。有符号饱和意味着值被限于处于那个元素宽度的最小负数与最大正数之间的范围中(例如,对于字节,范围为从-128(=-2^7)到127(=2^7-1))。

[0087] 加法和饱和校验的结果被存储到有符号结果815中与来自有符号第三源809的所使用的紧缩数据元素位置对应的紧缩数据元素位置中,或者如果有下一迭代,则该结果被继续传递到该下一迭代。在一些实施例中,将写掩码应用于此存储,使得如果对应的写掩码(位)被置位,则存储发生,如果对应的写掩码(位)未被置位,则存储不发生。

[0088] 图9图示链式融合乘法累加指令的迭代的执行的子集的实施例。具体而言,图9图示目的地的一个紧缩数据元素位置的迭代的执行电路。在该实施例中,链式融合乘法累加正对有符号源和无符号源进行操作,其中,累加器4倍于输入数据的尺寸。

[0089] 第一有符号源(源1 901)和第二无符号源(源2 903)各自都具有四个紧缩数据元素。这些紧缩数据元素中的每一个都具有诸如浮点数据或整数数据之类的数据。第三有符号源(初始值或结果915)具有存储有符号数据的紧缩数据元素。第一源901的尺寸和第二源903的尺寸是第三有符号源915的尺寸的四分之一。例如,第一源901和第二源903可具有16位的紧缩数据元素(例如,字),而第三有符号源915可具有64位的紧缩数据元素(例如,双精度浮点或64位整数)。

[0090] 在该图示中,仅示出第一源901和第二源903的最高有效的四个紧缩数据元素位置以及第三有符号源915的最高有效的紧缩数据元素位置。当然,如果还有任何其他紧缩数据元素位置,则还将处理这些紧缩数据元素位置。

[0091] 如图所示,按四元组处理紧缩数据元素。例如,使用乘法器电路907将第一源901和第二源903的最高有效的紧缩数据元素位置的数据相乘,使用乘法器电路905将来自第一源901和第二源903的次高有效的紧缩数据元素位置的数据相乘,使用乘法器电路909将来自第一源901和第二源903的第三高有效的紧缩数据元素位置的数据相乘,并且使用乘法器电路911将来自第一源901和第二源903的最低有效的紧缩数据元素位置的数据相乘。在一些实施例中,在乘法之前,对第一源901的有符号紧缩数据元素进行符号扩展,并且对第二源903的无符号紧缩数据元素进行零扩展。

[0092] 在一些实施例中,这些乘法器电路905-911重新用于其他紧缩数据元素位置。在其他实施例中,使用附加的乘法器电路,使得并行地处理紧缩数据元素。在一些上下文中,使用尺寸为有符号第三源915的尺寸的通道来完成并行执行。使用加法电路911将这些乘法中的每个乘法的结果相加。

[0093] (使用不同的加法器913或同一加法器911)将这些乘法结果的加法的结果加到来自有符号源3 915的最高有效紧缩数据元素位置的数据。

[0094] 最终,第二加法的结果919被存储到有符号目的地中与来自有符号第三源915的所使用的紧缩数据元素位置对应的紧缩数据元素位置中,或者被传递到下一迭代。在一些实施例中,将写掩码应用于此存储,使得如果对应的写掩码(位)被置位,则存储发生,如果对应的写掩码(位)未被置位,则存储不发生。

[0095] 图10图示链式融合乘法累加指令的迭代的执行的子集的实施例。具体而言,图10

图示目的地的一个紧缩数据元素位置的迭代的执行电路。在该实施例中，链式融合乘法累加正对有符号源和无符号源进行操作，其中，累加器4倍于输入数据的尺寸。

[0096] 第一有符号源(源1 1001)和第二无符号源(源2 1003)各自都具有四个紧缩数据元素。这些紧缩数据元素中的每一个都存储诸如浮点数据或整数数据之类的数据。第三有符号源(初始值或先前结果1015)具有存储有符号数据的紧缩数据元素。第一源1001的尺寸和第二源1003的尺寸是第三有符号源1015的尺寸的四分之一。例如，第一源1001和第二源1003可具有16位的紧缩数据元素(例如，字)，而第三有符号源1015可具有64位的紧缩数据元素(例如，双精度浮点或64位整数)。

[0097] 在该图示中，仅示出第一源1001和第二源1003的最高有效的四个紧缩数据元素位置以及第三有符号源1015的最高有效的紧缩数据元素位置。当然，如果还有任何其他紧缩数据元素位置，则还将处理这些紧缩数据元素位置。

[0098] 如图所示，按四元组处理紧缩数据元素。例如，使用乘法器电路1007将第一源1001和第二源1003的最高有效的紧缩数据元素位置的数据相乘，使用乘法器电路1005将来自第一源1001和第二源1003的次高有效的紧缩数据元素位置的数据相乘，使用乘法器电路1009将来自第一源1001和第二源1003的第三高有效的紧缩数据元素位置的数据相乘，并且使用乘法器电路1011将来自第一源1001和第二源1003的最低有效的紧缩数据元素位置的数据相乘。在一些实施例中，在乘法之前，对第一源1001的有符号紧缩数据元素进行符号扩展，并且对第二源1003的无符号紧缩数据元素进行零扩展。

[0099] 在一些实施例中，这些乘法器电路1005-1011重新用于其他紧缩数据元素位置。在其他实施例中，使用附加的乘法器电路，使得并行地处理紧缩数据元素。在一些上下文中，使用尺寸为有符号第三源1015的尺寸的通道来完成并行执行。使用加法/饱和电路1013将这些乘法结果的加法的结果加到来自有符号源3 1015的最高有效紧缩数据元素位置的数据。

[0100] 当加法导致对于有符号饱和过大或过小的值时，加法/饱和(累加器)电路1013保留操作数的符号。具体而言，对于多路加法与向目的地的写入之间的无限精度结果，饱和评估发生。当累加器1013是浮点且输入项是整数时，乘积的和以及浮点累加器输入值被转换为无限精度值(数百位的定点数)，执行乘法结果与第三输入的加法，并执行向实际累加器类型的单次舍入。

[0101] 加法和饱和校验的结果1019被存储到有符号目的地中与来自有符号第三源1015的所使用的紧缩数据元素位置对应的紧缩数据元素位置中，或者被传递到下一迭代。在一些实施例中，将写掩码应用于此存储，使得如果对应的写掩码(位)被置位，则存储发生，如果对应的写掩码(位)未被置位，则存储不发生。

[0102] 图11图示根据实施例的尺寸为2的幂的SIMD实现方式，其中，累加器使用比至乘法器的输入的尺寸大的输入尺寸。注意，(至乘法器的)源和累加器值可以是有符号值或无符号值。对于具有2X输入尺寸的累加器(换言之，累加器输入值的尺寸是源的紧缩数据元素的尺寸的2倍)，表1101图示不同的配置。对于字节尺寸的源，累加器使用尺寸为16位的字或半精度浮点(HPFP)值。对于字尺寸的源，累加器使用尺寸为32位的32位整数或单精度浮点(SPFP)值。对于SPFP或32位整数尺寸的源，累加器使用尺寸为64位的64位整数或双精度浮点(DPFP)值。

[0103] 对于具有4X输入尺寸的累加器(换言之,累加器输入值的尺寸是源的紧缩数据元素的尺寸的4倍),表1103图示不同的配置。对于字节尺寸的源,累加器使用尺寸为32位的32位整数或单精度浮点(SPFP)值。在一些实施例中,对于字尺寸的源,累加器使用尺寸为64位的64位整数或双精度浮点(DPFP)值。

[0104] 对于具有8X输入尺寸的累加器(换言之,累加器输入值的尺寸是源的紧缩数据元素的尺寸的8倍),表1105图示配置。对于字节尺寸的源,累加器使用64位整数。

[0105] 如之前所提示,矩阵操作电路可被包括在核中,或可作为外部加速器。图12图示利用矩阵操作电路的系统的实施例。在该图示中,多个实体与环形互连1245耦合。

[0106] 多个核1201、1203、1205和1207提供非基于片的指令支持。在一些实施例中,矩阵操作电路设于核1203中,在其他实施例中,矩阵操作电路1211和1213是在环形互连1245上可访问的。

[0107] 此外,提供一个或多个存储器控制器1223-1225,以代表核和/或矩阵操作电路来与存储器1233和1231通信。

[0108] 图13图示处理器核流水线的实施例,该处理器核流水线支持使用片的矩阵操作。分支预测和解码电路1303执行对来自存储在指令存储1301中的指令的分支预测、对这些指令的解码和/或分支预测和解码两者。例如,本文中详述的指令可存储在指令存储中。在一些实现方式中,分开的电路用于分支预测,并且在一些实施例中,至少一些指令被解码为一个或多个微操作、微代码进入点、微指令、其他指令或使用微代码1305的其他控制信号。分支预测和解码电路1303可使用各种不同的机制来实现。合适机制的示例包括但不限于查找表、硬件实现、可编程逻辑阵列(PLA)、微代码只读存储器(ROM)等。

[0109] 分支预测和解码电路1303耦合至重命名/分配器电路1307,在一些实施例中,该重命名/分配器电路1307耦合至调度器电路1309。在一些实施例中,这些电路通过执行以下步骤中的一个或多个来提供寄存器重命名、寄存器分配和/或调度功能:1)将逻辑操作数值重命名为物理操作数值(例如,一些实施例中的寄存器别名表);2)将状态位和标志分配给经解码的指令;以及3)(例如,在一些实施例中,使用预留站)调度经解码的指令供在指令池外部的执行电路上执行。

[0110] 调度器电路1309表示任何数量的不同调度器,包括预留站、中央指令窗等。(多个)调度器单元调度器电路1309耦合至(多个)物理寄存器堆1315或包括(多个)物理寄存器堆1315。(多个)物理寄存器堆1315中的每一个表示一个或多个物理寄存器堆,其中不同的物理寄存器堆存储一种或多种不同的数据类型,诸如,标量整数、标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点、状态(例如,作为要执行的下一指令的地址的指令指针)、片,等等。在一个实施例中,(多个)物理寄存器堆1315包括向量寄存器电路、写掩码寄存器电路和标量寄存器电路。这些寄存器电路可提供架构向量寄存器、向量掩码寄存器和通用寄存器。(多个)物理寄存器堆1315被引退电路1317覆盖,以图示可实现寄存器重命名和乱序执行的各种方式(诸如,使用(多个)重排序缓冲器和(多个)引退寄存器堆、使用(多个)未来文件(future file)、(多个)历史缓冲器、和(多个)引退寄存器堆、使用寄存器映射和寄存器池,等等)。引退电路1317和(多个)物理寄存器堆1315耦合至(多个)执行电路1311。

[0111] 尽管在乱序执行的上下文中描述了寄存器重命名,但应当理解,可以在有序架构中使用寄存器重命名。虽然处理器的所图示的实施例也可包括分开的指令和数据高速缓存

单元以及共享的L2高速缓存单元，但替代实施例也可具有用于指令和数据两者的单个内部高速缓存，诸如例如，第一级(L1)内部高速缓存、或多个级别的内部高速缓存。在一些实施例中，该系统可包括内部高速缓存和在核和/或处理器外部的外部高速缓存的组合。替代地，所有高速缓存都可在核和/或处理器的外部。

[0112] 执行电路1311包括一个或多个执行电路1321、1323和1327的集合以及一个或多个存储器访问电路1325的集合。执行电路1321、1323和1327执行各种操作(例如，移位、加法、减法、乘法)并对各种数据类型(例如，标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点)执行。尽管一些实施例可以包括专用于特定功能或功能集的多个执行单元，但其他实施例可包括仅一个执行单元或全部执行所有功能的多个执行单元。标量电路1321执行标量操作，向量/SIMD电路1323执行向量/SIMD操作，并且矩阵操作电路1327执行本文中详述的矩阵(片)操作。

[0113] 存储器访问单元的集合1364耦合到存储器单元1370，该存储器单元1370包括数据TLB单元1372，该数据TLB单元1372耦合到数据高速缓存单元1374，该数据高速缓存单元1374耦合到第二级(L2)高速缓存单元1376。在一个示例性实施例中，存储器访问单元1364可包括加载单元、存储地址单元和存储数据单元，其中的每一个均耦合到存储器单元1370中的数据TLB单元1372。指令高速缓存单元1334进一步耦合到存储器单元1370中的第2级(L2)高速缓存单元1376。L2高速缓存单元1376耦合到一个或多个其他级别的高速缓存，并最终耦合到主存储器。

[0114] 作为示例，示例性寄存器重命名的、乱序发布/执行核架构可以如下所述地实现流水线：1)指令取出电路执行取出和长度解码级；2)分支和解码电路1303执行解码级；3)重命名/分配器电路1307执行分配级和重命名级；4)调度器电路1309执行调度级；5)(耦合至或被包括在调度器电路1307和重命名/分配电路1307和存储器单元中的)(多个)物理寄存器堆执行寄存器读取/存储器读取级；执行电路1311执行执行级；6)存储器单元和(多个)物理寄存器堆单元执行写回/存储器写入级；7)各个单元可涉及异常处置级；以及8)引退单元和(多个)物理寄存器堆单元执行提交级。

[0115] 核可支持一个或多个指令集(例如，x86指令集(具有与较新版本一起添加的一些扩展)；加利福尼亚州桑尼维尔市的MIPS技术公司的MIPS指令集；加利福尼亚州桑尼维尔市的ARM控股公司的ARM指令集(具有诸如NEON等任选附加扩展))，其中包括本文中描述的(多条)指令。在一个实施例中，核1390包括用于支持紧缩数据指令集扩展(例如，AVX1、AVX2)的逻辑，由此允许许多多媒体应用所使用的操作利用紧缩数据来执行。

[0116] 应当理解，核可支持多线程化(执行两个或更多个并行的操作或线程的集合)，并且可以按各种方式来完成该多线程化，此各种方式包括时分多线程化、同步多线程化(其中单个物理核为物理核正在同步多线程化的各线程中的每一个线程提供逻辑核)、或其组合(例如，时分取出和解码以及此后诸如用Intel®超线程化技术来同步多线程化)。

[0117] 图14图示处理器核流水线的实施例，该处理器核流水线支持使用片的矩阵操作。分支预测和解码电路1403执行对来自存储在指令存储1401中的指令的分支预测、对这些指令的解码和/或分支预测和解码两者。例如，本文中详述的指令可存储在指令存储中。在一些实现方式中，分开的电路用于分支预测，并且在一些实施例中，至少一些指令被解码为一个或多个微操作、微代码进入点、微指令、其他指令或使用微代码1405的其他控制信号。分

支预测和解码电路1403可使用各种不同的机制来实现。合适机制的示例包括但不限于查找表、硬件实现、可编程逻辑阵列(PLA)、微代码只读存储器(ROM)等。

[0118] 分支预测和解码电路1403耦合至重命名/分配器电路1407，在一些实施例中，该重命名/分配器电路1407耦合至调度器电路1409。在一些实施例中，这些电路通过执行以下一个或多个步骤来提供寄存器重命名、寄存器分配和/或调度功能：1)将逻辑操作数值重命名为物理操作数值(例如，一些实施例中的寄存器别名表)；2)将状态位和标志分配给经解码的指令；以及3)(例如，在一些实施例中，使用预留站)调度经解码的指令以供在指令池外部的执行电路上执行。

[0119] 调度器电路1409表示任何数量的不同调度器，包括预留站、中央指令窗等。(多个)调度器单元调度器电路1409耦合至(多个)物理寄存器堆1415或包括(多个)物理寄存器堆1415。(多个)物理寄存器堆1415中的每一个表示一个或多个物理寄存器堆，其中不同的物理寄存器堆存储一种或多种不同的数据类型，诸如，标量整数、标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点、状态(例如，作为要执行的下一指令的地址的指令指针)、片，等等。在一个实施例中，(多个)物理寄存器堆1415包括向量寄存器电路、写掩码寄存器电路和标量寄存器电路。这些寄存器电路可提供架构向量寄存器、向量掩码寄存器和通用寄存器。(多个)物理寄存器堆1415被引退电路1417覆盖，以图示可实现寄存器重命名和乱序执行的各种方式(诸如，使用(多个)重排序缓冲器和(多个)引退寄存器堆、使用(多个)未来文件(future file)、(多个)历史缓冲器、和(多个)引退寄存器堆、使用寄存器映射和寄存器池，等等)。引退电路1417和(多个)物理寄存器堆1415耦合至(多个)执行电路1411。

[0120] 尽管在乱序执行的上下文中描述了寄存器重命名，但应当理解，可以在有序架构中使用寄存器重命名。虽然处理器的所图示的实施例也可包括分开的指令和数据高速缓存单元以及共享的L2高速缓存单元，但替代实施例也可具有用于指令和数据两者的单个内部高速缓存，诸如例如，第一级(L1)内部高速缓存、或多个级别的内部高速缓存。在一些实施例中，该系统可包括内部高速缓存和在核和/或处理器外部的外部高速缓存的组合。替代地，所有高速缓存都可在核和/或处理器的外部。

[0121] 执行电路1411包括一个或多个执行电路1427的集合以及一个或多个存储器访问电路1425的集合。执行电路1427执行本文中详述的矩阵(片)操作。

[0122] 存储器访问单元的集合1464耦合到存储器单元1470，该存储器单元1470包括数据TLB单元1472，该数据TLB单元1472耦合到数据高速缓存单元1474，该数据高速缓存单元1474耦合到第二级(L2)高速缓存单元1476。在一个示例性实施例中，存储器访问单元1464可包括加载单元、存储地址单元和存储数据单元，其中的每一个均耦合到存储器单元1470中的数据TLB单元1472。指令高速缓存单元1434进一步耦合到存储器单元1470中的第二级(L2)高速缓存单元1476。L2高速缓存单元1476耦合到一个或多个其他级别的高速缓存，并最终耦合到主存储器。

[0123] 作为示例，示例性寄存器重命名的、乱序发布/执行核架构可以如下所述地实现流水线：1)指令取出电路执行取出和长度解码级；2)分支和解码电路1403执行解码级；3)重命名/分配器电路1407执行分配级和重命名级；4)调度器电路1409执行调度级；5)(耦合至或被包括在调度器电路1407和重命名/分配电路1407和存储器单元中的)(多个)物理寄存器堆执行寄存器读取/存储器读取级；执行电路1411执行执行级；6)存储器单元和(多个)物理

寄存器堆单元执行写回/存储器写入级;7)各个单元可涉及异常处置级;以及8)引退单元和(多个)物理寄存器堆单元执行提交级。

[0124] 核可支持一个或多个指令集(例如,x86指令集(具有与较新版本一起添加的一些扩展);加利福尼亚州桑尼维尔市的MIPS技术公司的MIPS指令集;加利福尼亚州桑尼维尔市的ARM控股公司的ARM指令集(具有诸如NEON等任选附加扩展)),包括本文中描述的(多条)指令。在一个实施例中,核1490包括用于支持紧缩数据指令集扩展(例如,AVX1、AVX2)的逻辑,由此允许许多多媒体应用所使用的操作利用紧缩数据来执行。

[0125] 应当理解,核可支持多线程化(执行两个或更多个并行的操作或线程的集合),并且可以按各种方式来完成该多线程化,此各种方式包括时分多线程化、同步多线程化(其中单个物理核为物理核正在同步多线程化的各线程中的每一个线程提供逻辑核)、或其组合(例如,时分取出和解码以及此后诸如用Intel®超线程化技术来同步多线程化)。

[0126] B.布局

[0127] 贯穿本说明书,使用行为主的数据布局来表达数据。列为主的用户应当根据项的定向来变换这些项。图15图示按行为主格式和列为主格式表达的矩阵的示例。如图所示,矩阵A是2x3矩阵。当该矩阵按行为主的格式存储时,行的数据元素是连续的。当该矩阵按列为主的格式存储时,列的数据元素是连续的。$A^T*B^T=(BA)^T$是矩阵的公知属性,其中,上标T表示转置。按行为主的数据那样来读取列为主的数据导致看起来像转置矩阵的矩阵。

[0128] 在一些实施例中,在硬件中利用行为主的语义,并且列为主的数据将交换操作数顺序并使结果是矩阵的转置,但是对于从存储器的后续列为主的读取,其是正确的非转置矩阵。

[0129] 例如,如果具有两个要相乘的列为主的矩阵:

[0130] a b g i k ag+bh ai+bj ak+bl

[0131] c d*h j l= cg+dh ci+dj ck+dl

[0132] e f eg+fh ei+fj ek+fl

[0133] (3x2)(2x3) (3x3)

[0134] 输入矩阵将按如下方式被存储在线性存储器中(列为主):

[0135] a c e b d f

[0136] 以及

[0137] g h i j k l。

[0138] 以尺度2x3和3x2将那些矩阵读取为行为主的,则它们将表现为:

[0139] a c e和g h

[0140] b d f i j

[0141] k l

[0142] 交换顺序和矩阵乘法:

[0143] g h a c e ag+bh cg+dh eg+fh

[0144] i j * b d f= ai+bj ci+dj ei+fj

[0145] k l ak+bl ck+dl ek+fl

[0146] 转置矩阵移出,并且随后可按行为主的顺序被存储:

[0147] ag+bh cg+dh eg+fh ai+bj ci+dj ei+fj ak+bl ck+dl ek+fl

[0148] 并且在后续的列为主的计算中被使用,其是正确的未转置矩阵:

[0149] ag+bh ai+bj ak+bl

[0150] cg+dh ci+dj ck+dl

[0151] eg+fh ei+fj ek+fl

[0152] III.示例性使用

[0153] 图16图示矩阵(片)的使用的示例。在该示例中,矩阵C 1601包括两个片,矩阵A 1603包括一个片,并且矩阵B 1605包括两个片。该图示出用于计算矩阵乘法的算法的内循环的示例。在该示例中,来自矩阵C 1601的两个结果片tmm0和tmm1用于将中间结果累加。当来自矩阵A 1603的一个片(tmm2)乘以来自矩阵B 1605的两个片时,这个片被重复使用2次。指针用于加载来自箭头所指示方向的新A片和两个新B片。未示出的外循环调整用于C片的指针。

[0154] 如图所示的示例性代码包括片配置指令的使用,并且被执行以配置片使用,加载片,用于处理片的循环,将片存储到存储器,并释放片使用。

[0155] 图17图示矩阵(片)的使用的实施例。在1701处,配置片使用。例如,执行TILECONFIG指令以配置片使用,包括设置每个片的行数和列数。典型地,在1703处,从存储器加载至少一个矩阵(片)。

[0156] IV.示例性指令

[0157] A.片配置

[0158] 如上文所讨论,片使用典型地需要在使用前进行配置。例如,可能不需要完全使用所有的行和列。不配置这些行和列在一些实施例中不仅节省了功率,而且可使用配置来判定操作是否将生成错误。例如,如果M和L不相同,则(NxM)*(L*N)形式的矩阵乘法典型地将不起作用。

[0159] 本文中详述的是矩阵(片)配置("TILECONFIG")指令及其执行的实施例。在使用利用片的矩阵之前,在一些实施例中,将配置片支持。例如,配置每个片有多少行和多少列、将使用的片,等等。TILECONFIG指令是对计算机自身的改进,因为它提供对配置计算机以使用(作为处理器核的部分的、或作为外部设备的)矩阵加速器的支持。具体而言,TILECONFIG指令的执行使得配置从存储器被检取,并被应用于矩阵加速器内的矩阵(片)设置。

[0160] i.示例性执行

[0161] 图18图示TILECONFIG指令的示例性执行。TILECONFIG指令格式包括用于操作码和存储器地址的字段。

[0162] 如所图示,TILECONFIG指令将地址用作指向存储器1801位置的指针,该存储器1801位置包含将支持的矩阵(片)的描述1803。

[0163] 处理器/核1805的执行电路1811通过以下步骤来执行TILECONFIG:经由存储器控制器1815从存储器1801检取描述1803;在片配置1817中配置用于调色板的片(设置行数和列数);以及标记矩阵支持处于使用中。具体而言,指令执行资源1811配置成按设置片配置1817所指定来使用片。指令执行资源还可包括用于指示片使用的机器专用寄存器或配置寄存器。

[0164] 片配置1817被设置成经由TILECONFIG指令的执行、如由片描述1803所指示地指示针对每个片的参数。所设置的参数是每个片的行数和列数。还设置附加的值,诸如,使用中

(in-use)值和开始值。片配置1817利用一个或多个寄存器1819来存储片使用和配置信息。

[0165] ii.示例性片存储

[0166] 图19(A)-图19(D)图示(多个)寄存器1919的示例。图19(A)图示多个寄存器1919。如图所示,每个片(TMM0 1901...TMMN 1903)具有分开的寄存器,其中每个寄存器存储那个特定片的行尺寸和列尺寸。StartK和StartM被存储在分开的寄存器1911和1913中。对一个或多个状态寄存器1915置位(例如,TILES_CONFIGURED=1)以指示片经配置以供使用。

[0167] 图19(B)图示多个寄存器1919。如图所示,每个片具有用于其行和其列的分开的寄存器。例如,TMM0行配置1921、TMM0列配置1923、StartK和StartM被存储在分开的寄存器1911和1913中。对一个或多个状态寄存器1915置位(例如,TILES_CONFIGURED=1)以指示片经配置以供使用。

[0168] 图19(C)图示单个寄存器1919。如图所示,该寄存器将片配置(每片的行和列)1931、StartK 1911和StartM 1913存储在作为紧缩数据寄存器的单个寄存器中。对一个或多个状态寄存器1915置位(例如,TILES_CONFIGURED=1)以指示片经配置以供使用。

[0169] 图19(D)图示多个寄存器1919。如图所示,单个寄存器存储片配置(每片的行和列)1931。StartK和StartM被存储在分开的寄存器1911和1913中。对一个或多个状态寄存器1915置位(例如,TILES_CONFIGURED=1)以指示片经配置以供使用。

[0170] 构想了其他组合,诸如,将开始寄存器组合到单个寄存器中,在该单个寄存器中,这些开始寄存器被分开显示,等等。

[0171] iii.示例性存储的矩阵(片)描述

[0172] 图20图示将被支持的矩阵(片)的描述的实施例。在该示例中,每个字段为字节。在字节[0]中,存储调色板ID 2001。调色板ID用于对调色板表1813进行索引,该调色板表1813如由配置所定义、根据调色板ID来存储片中的字节数以及与该ID相关联的片的每行的字节。字节1-7是预留的,并且典型地为零。

[0173] 字节8-9存储用于“startM”寄存器2003的值,并且字节10-11存储用于“startK”寄存器2005的值。为了支持在这些事件后重新开始指令,这些指令将信息存储在这些寄存器中。startM指示应当被用于重新开始的行。startK指示针对相关操作的内积中的位置。不需要行(列)中的该位置。像逐元素加法/减法/乘法这样的二维操作仅使用startM。三维操作使用来自startM和startK两者的值。典型地,仅需要startM的操作在写入startM时将使startK归零。

[0174] 在不重新开始被中断的片指令的任何时刻,在一些实施例中,使startM值和startK值归零是软件的职责。例如,未掩码的浮点异常处置程序可决定在软件中完成操作,并且将程序计数器值改变为另一指令,通常是下一指令。在这种情况下,在恢复程序之前,软件异常处置程序必须使由操作系统呈现给该软件异常处置程序的异常帧中的startM值和startK值归零。操作系统后续将重新加载那些值。

[0175] 字节16-17存储片0的行数2013和列数2015,字节18-19存储片1的行数和列数,以此类推。换言之,每一2字节组指定片的行数和列数。如果2字节的组不用于指定片参数,则它们应当具有值零。为比实现限制或调色板限制更多的片指定片参数导致错误。未配置的片用0行0列被设置为INIT(初始)状态。

[0176] 最终,存储器中的配置典型地以诸如用于若干连续字节的全零之类的结尾描述结

束。

[0177] iv.(多个)示例性格式

[0178] 用于TILECONFIG指令的格式的实施例是TILECONFIG地址。在一些实施例中，TILECONFIG是指令的操作码助记符。地址是指向存储器中的矩阵(片)描述的指针。在一些实施例中，地址字段是R/M值(诸如，2446)。

[0179] 在实施例中，指令的编码包括比例-索引-基址(SIB)型存储器寻址操作数，其间接地标识存储器中的多个被索引的目的地位置(例如，字段2450)。在一个实施例中，SIB型存储器操作数可包括标识基址寄存器的编码。基址寄存器的内容可表示存储器中的基址，存储器中的特定目的地位置的地址通过该基址来计算。例如，基址可以是用于扩展向量指令的潜在的目的地位置的块中的第一位置的地址。在一个实施例中，SIB型存储器操作数可包括标识索引寄存器的编码。索引寄存器的每个元素可指定索引或偏移值，该索引或偏移值能用于通过基址来计算潜在的目的地位置的块内的相应目的地位置的地址。在一个实施例中，SIB型存储器操作数可包括指定比例因子的编码，该比例因子在计算相应目的地地址时将应用于每个索引值。例如，如果比例因子值4被编码在SIB型存储器操作数中，则从索引寄存器的元素获得的每个索引值可乘以4，且随后加到基址，以计算目的地地址。

[0180] 在一个实施例中vm32{x,y,z}形式的SIB型存储器操作数可标识使用SIB型存储器寻址指定的存储器操作数的向量数组。在该示例中，使用共同的基址寄存器、常数比例因子和包含各自都为32位的索引值的各个元素的向量索引寄存器来指定存储器地址的数组。向量索引寄存器可以是128位的寄存器(例如，XMM)寄存器(vm32x)、256位的(例如，YMM)寄存器(vm32y)或512位的(例如，ZMM)寄存器(vm32z)。在另一实施例中，vm64{x,y,z}形式的SIB型存储器操作数可标识使用SIB型存储器寻址指定的存储器操作数的向量数组。在该示例中，使用共同的基址寄存器、常数比例因子和包含各自都为64位的索引值的各个元素的向量索引寄存器来指定存储器地址的数组。向量索引寄存器可以是128位的寄存器(例如，XMM)寄存器(vm64x)、256位的(例如，YMM)寄存器(vm64y)或512位的(例如，ZMM)寄存器(vm64z)。

[0181] v.(多种)示例性执行方法

[0182] 图21图示由处理器执行以处理TILECONFIG指令的方法的实施例。

[0183] 在2101处，取出指令。例如，取出TILECONFIG指令。TILECONFIG指令的实施例包括用于操作码以及存储器地址操作数的字段。

[0184] 在2103处，对取出的指令解码。例如，由诸如本文中详述的解码电路对取出的TILECONFIG指令进行解码。

[0185] 在2105处，检取在存储器地址操作数的存储器地址处发现的描述，并且(根据需要)调度经解码的指令。

[0186] 在2107处，由诸如本文中所详述的执行电路(硬件)执行经解码的指令。对于TILECONFIG指令，执行将使执行电路在片配置中配置片的使用(设置行数和列数)并标记矩阵(片)支持处于使用中(活跃)。例如，配置一个或多个寄存器1819。片支持使用(例如，“TILES_CONFIGURED(片_经配置)”)典型地通过设置状态、控制或机器专用寄存器中的位来指示。具体而言，指令执行资源1811配置成按经检取的配置所指定来使用片。

[0187] 在一些实施例中，在2109处，提交或引退指令。

[0188] 图22图示使用存储器寻址来执行TILECONFIG指令的更详细的描述。典型地，这在该描述已从存储器检取之后由诸如上文详述的执行电路之类的执行电路来执行。虽然未图示，但是在一些实施例中，首先执行检查以判定片是否被支持。通常通过CPUID检查来发现支持。

[0189] 在2201处，作出调色板ID是否被支持的判定。例如，CPUID是否声明该ID被支持?如果否，则在2203处，通用保护错误发生。

[0190] 在2205处，读取第一片特定分组。例如，读取片0(T0)的行数和列数。

[0191] 在2207处，作出所读取的分组是否有效的判定。例如，如果行数或列数中的一者(非两者)被设置为0，则分组不是有效的，并且在2203处，配置停止且不认为片处于使用中。例如当行或列中的一者(非两者)为零时，无效组发生。此外，当行数的值大于所支持的行的最大值时(该最大值通过将调色板ID的片字节尺寸除以调色板表中所发现的针对该调色板ID的每行的字节数而发现)，错误发生。另一潜在的错误是在存在比所支持的名称更多的名称时。

[0192] 如果所读取的分组有效，则在2211处，将与所读取的分组相关联的片配置成使用由片配置中的分组所指定的行数和列数。片中的元素的尺寸由针对调色板ID的调色板表条目设置。

[0193] 在2213处，作出经检取的配置的所有片是否都已被配置的判定。例如，所有可能的片名称是否都已被处理?在一些实施例中，当针对特定片的行和列两者都为0时，则所有片都已被处理。

[0194] 当不是所有片都已被配置时，在2215处，递增片编号，使得将评估配置中的下一片。

[0195] 在2217处，读取经递增的片的分组。例如，读取片1(T1)的行数和列数。在2207处，作出所读取的分组是否有效的判定，以此类推。

[0196] 当所有片都已被配置时，则在2209处，指令完成。例如通过设置寄存器中的使用中指示符，片将被标记为处于用于矩阵操作的使用中。

[0197] vi.示例性伪代码

[0198] 图23图示TILECONFIG指令的执行的示例性伪代码。

[0199] B.片点积

[0200] 本文中详述的是矩阵(片)点积(“TILEDOTPRODUCT”)指令及其执行的实施例。TILEDOTPRODUCT指令是对计算机自身的改进，因为它提供对利用单条指令执行涉及数据值的两个矩阵(片)的点积操作的支持。具体而言，TILEDOTPRODUCT指令的执行导致执行对来自数据值的两个源矩阵(片)的元素的点积操作并将结果累加到目的地矩阵(片)的对应数据元素位置中。源矩阵(片)中的数据元素的尺寸取决于指令和片支持而有所不同。源矩阵(片)中所包含的数据元素的示例性尺寸包括但不限于4位、8位、16位、32位、64位、128位、256位，等等。在一些实施例中，目的地矩阵(片)的、不具有在源矩阵(片)中的对应元素的行和列的元素被归零。

[0201] i.示例性执行

[0202] 图24图示TILEDOTPRODUCT指令的示例性执行。TILEDOTPRODUCT指令格式包括用于操作码(例如，在图中示出为“TDP”)、目的地累加器操作数(例如，在图中示出为

“DESTINATION MATRIX(TILE)”(“目的地矩阵(片)”))以及两个源操作数(例如,在图中示出为“FIRST SOURCE MATRIX(TILE)”(“第一源矩阵(片)”)和“SECOND SOURCE MATRIX(TILE)”(“第二源矩阵(片)”))的字段。在实施例中,目的地累加器操作数用于将从对第一和第二源矩阵(片)操作数的元素执行点积操作而得到的数据进行累加。在图24中示出示例目的地矩阵(片)操作数2401,其初始地存储双字尺寸的数据元素的矩阵。

[0203] 两个源矩阵(片)操作数字段分别表示第一源矩阵(片)操作数2403和第二源矩阵(片)操作数2405。如先前所详述,矩阵(片)可存储在寄存器的集合中、存储在存储器中的位置中(例如,作为跨步式行)、或存储在可由执行电路访问的存储中。

[0204] 在图24中,目的地矩阵(片)累加器操作数2401、第一源矩阵(片)操作数2403和第二源矩阵(片)操作数2405中的每一个都包括2x2的数据元素矩阵。图24中的矩阵的尺度仅用于说明性目的;一般而言,TILEDOTPRODUCT指令可对其中同第一矩阵(片)操作数相关联的列数与第二矩阵(片)操作数的行数相同的任何两个源矩阵(片)操作数操作(也就是说,如图24中所示,其中,第一矩阵(片)操作数的尺度为M行x K列,并且第二矩阵(片)操作数的尺度为K行x N列)。该示例中的目的地矩阵(片)累加器操作数具有M行x N列,使得目的地矩阵(片)中的行数与第一矩阵(片)操作数中的行数相同,并且目的地矩阵(片)中的列数与第二矩阵(片)操作数中的列数相同。

[0205] 如图所示,执行电路2407使用融合乘法加法器(FMA)2409的网格,通过以下方式来执行经解码的TILEDOTPRODUCT指令:对两个源矩阵(片)操作数2403和2405的元素执行点积操作,并且将结果累加到目的地矩阵(片)累加器操作数2401的对应的数据元素位置中。

[0206] 参照示例目的地矩阵(片)累加器操作数2401和源矩阵(片)操作数2403和2405,执行电路2407使用第一源矩阵(片)操作数2403的第一行和第二源矩阵(片)操作数2405的第一列生成点积值,并将结果累加在目的地矩阵(片)操作数2401的[0,0]数据元素位置中。在图24中,例如,目的地矩阵(片)操作数2401的[0,0]数据元素位置将初始存储的值W与使用第一源矩阵(片)操作数2403的第一行(元素[A,B]和[C,D])与第二源矩阵(片)操作数2405的第一列(元素[I,J]和[M,N])所计算的点积值累加,即,W+DP([A,B],[I,J])+DP([C,D],[M,N])。

[0207] 执行电路2407进一步使用第一源矩阵(片)操作数2403的第一行与第二源矩阵(片)操作数2405的第二列计算点积值,并将结果累加在目的地矩阵(片)操作数2401的[0,1]数据元素位置中。执行电路2407进一步使用第一源矩阵(片)操作数2403的第二行与第二矩阵(片)操作数2405的第一列生成点积值,并将结果累加在目的地矩阵(片)操作数2401的[1,0]数据元素位置中。执行电路2407进一步使用第一源矩阵(片)操作数2403的第二行与第二源矩阵(片)操作数2405的第二列生成点积值,并将结果累加在目的地矩阵(片)操作数2401的[1,1]数据元素位置中。

[0208] ii.(多个)示例性格式

[0209] 用于TILEDOTPRODUCT指令的格式的一个实施例是TDPWSSDS TMM1,TMM2,TMM3。在一些实施例中,TDPWSSDS是该指令的操作码助记符,其中,该助记符的“TDP”部分指示TILEDOTPRODUCT操作,并且该助记符的“WSSDS”部分指示该指令计算包括有符号字尺寸的元素的源矩阵(片)操作数的点积,并且利用饱和将结果累加到包括双字尺寸的元素的目的地矩阵(片)中。在该指令格式以及下文中的指令格式中,TMM1是用于目的地矩阵(片)操作

数的字段。TMM2和TMM3是用于矩阵(片)源操作数的字段。在一些实施例中,TMM3字段是R/M值(诸如,2846),TMM1字段是REG 2844,并且数据元素尺寸在2865中发现。

[0210] 用于TILEDOTPRODUCT指令的格式的另一实施例是TDPWSSQS TMM1,TMM2,TMM3。在一些实施例中,TDPWSSQS是该指令的操作码助记符,其中,该助记符的"WSSQS"部分指示该指令计算包括有符号字尺寸的元素的源矩阵(片)操作数的点积,并且利用饱和将结果累加到包括四字尺寸的元素的目的地矩阵(片)中。

[0211] 用于TILEDOTPRODUCT指令的格式的另一实施例是TDPB[SS/UU/US/SU]DS TMM1,TMM2,TMM3。在一些实施例中,TDPB[SS/UU/US/SU]DS是该指令的操作码助记符,其中,该助记符的"B"和"D"部分指示该指令计算包括字节尺寸的元素的源矩阵(片)操作数的点积,并且利用饱和将结果累加到包括双字尺寸的元素的目的地矩阵(片)累加器操作数中。

[0212] 在实施例中,上文指令中的以及类似地下文指令中的助记符的[SS/UU/US/SU]部分指示两个源矩阵(片)操作数中的每个源矩阵(片)操作数包括有符号数据值还是无符号数据值。[SS/UU/US/SU]助记符部分的第一个字母对应于第一源矩阵(片)操作数,并且第二个字母对应于第二源矩阵(片)操作数。例如,"SS"指示两个源矩阵(片)操作数都是有符号的,"UU"指示两个源矩阵(片)操作数都是无符号的,"US"指示第一源矩阵(片)操作数是无符号的且第二源矩阵(片)操作数是有符号的,并且"SU"指示第一源矩阵(片)操作数是有符号的且第二源矩阵(片)操作数是无符号的。如果第一源矩阵(片)操作数或第二源矩阵(片)中的任一者是有符号的,则目的地矩阵(片)操作数是有符号的;否则,当两个源矩阵(片)操作数都是无符号的时,目的地矩阵(片)是无符号的。如果源矩阵(片)操作数中的任一者是有符号的,则结果输出饱和是有符号饱和;否则,结果输出饱和是无符号饱和。

[0213] 用于TILEDOTPRODUCT指令的格式的另一实施例是TDP8B[SS/UU/US/SU]4BITDS TMM1,TMM2,TMM3。在一些实施例中,TDP8B[SS/UU/US/SU]4BITDS是该指令的操作码助记符,其中,"8B"和"4BITD"标识符指示该指令计算双字源矩阵(片)操作数(一个操作数包含字节尺寸的元素,并且另一操作数包含4位(半字节)尺寸的元素)的点积,并且将结果累加到包括双字尺寸的元素的目的地矩阵(片)中。

[0214] 用于TILEDOTPRODUCT指令的格式的另一实施例是TDP4BIT[S,U][S,U]DS TMM1,TMM2,TMM3。在一些实施例中,TDP4BIT[S,U][S,U]DS是该指令的操作码助记符,其中,该助记符的"4BIT"和"D"部分指示该指令计算双字源矩阵(片)操作数(每个源矩阵(片)操作数包括4位(半字节)尺寸的元素)的点积,并且将结果累加到包括双字尺寸的元素的目的地矩阵(片)中。

[0215] 在实施例中,指令的编码包括间接地标识存储器中的多个经索引的目的地位置的比例-索引-基址(SIB)型存储器寻址操作数。在一个实施例中,SIB型存储器操作数可包括标识基址寄存器的编码。基址寄存器的内容可表示存储器中的基址,存储器中的特定目的地位置的地址通过该基址来计算。例如,基址可以是用于扩展向量指令的潜在的目的地位置的块中的第一位置的地址。在一个实施例中,SIB型存储器操作数可包括标识索引寄存器的编码。索引寄存器的每个元素可指定索引或偏移值,该索引或偏移值能用于通过基址来计算潜在的目的地位置的块内的相应目的地位置的地址。在一个实施例中,SIB型存储器操作数可包括指定在计算相应目的地地址时要应用于每个索引值的比例因子的编码。例如,如果比例因子值4被编码在SIB型存储器操作数中,则从索引寄存器的元素获得的每个索引

值可乘以4,且随后加到基址,以计算目的地地址。

[0216] 在一个实施例中vm32{x,y,z}形式的SIB型存储器操作数可标识使用SIB型存储器寻址指定的存储器操作数的向量数组。在该示例中,使用共同的基址寄存器、恒定的比例因子和包含各自都为32位的索引值的各个元素的向量索引寄存器来指定存储器地址的阵列。向量索引寄存器可以是128位的寄存器(例如,XMM)寄存器(vm32x)、256位的(例如,YMM)寄存器(vm32y)或512位的(例如,ZMM)寄存器(vm32z)。在另一实施例中vm64{x,y,z}形式的SIB型存储器操作数可标识使用SIB型存储器寻址指定的存储器操作数的向量数组。在该示例中,使用共同的基址寄存器、恒定的比例因子和包含各自都为64位的索引值的各个元素的向量索引寄存器来指定存储器地址的阵列。向量索引寄存器可以是128位的寄存器(例如,XMM)寄存器(vm64x)、256位的(例如,YMM)寄存器(vm64y)或512位的(例如,ZMM)寄存器(vm64z)。

[0217] iii.(多种)示例性执行方法

[0218] 图25图示由处理器执行以处理矩阵(片)点积指令的方法的实施例。

[0219] 在2501处,取出指令。例如,取出TILEDOTPRODUCT指令。该TILEDOTPRODUCT指令包括用于操作码、第一源矩阵(片)操作数、第二源矩阵(片)操作数和目的地矩阵(片)操作数的字段。在一些实施例中,该指令进一步包括用于写掩码的字段。在一些实施例中,从指令高速缓存取出该指令。源操作数和目的地操作数由紧缩数据组成。TILEDOTPRODUCT指令的操作码指示将对源矩阵(片)操作数执行点积操作。在一些实施例中,操作码进一步指示第一源矩阵(片)操作数和第二源矩阵(片)操作数中的每一个由有符号值还是无符号值组成。在一些实施例中,操作码进一步指示包括第一源矩阵(片)操作数、第二源矩阵(片)操作数和目的地矩阵(片)操作数中的每一个的矩阵(片)数据值的尺寸(例如,指定数量的位、字节、四字、双字,等等)。

[0220] 在2503处,对取出的指令解码。例如,由诸如本文中所详述的解码电路对取出的TILEDOTPRODUCT指令解码。

[0221] 在2505处,检取与经解码的指令的源矩阵(片)操作数相关联的数据值,并且(根据需要)调度经解码的指令。例如,当源矩阵(片)操作数中的一个或多个是存储器操作数时,检取来自所指示的存储器位置的数据。

[0222] 在2507处,由诸如本文中所详述的执行电路(硬件)执行经解码的指令。对于TILEDOTPRODUCT指令,执行使执行电路对源数据执行点积操作。在一些实施例中,经解码的矩阵点积指令的执行使执行电路:通过对来自第一源矩阵操作数和第二源矩阵操作数的元素执行点积操作来计算结果;并且将结果累加到目的地矩阵操作数的元素中。

[0223] 在一些实施例中,当以下一种或多种情况为真时,产生错误:和第一源矩阵操作数相关联的列数与和第二源矩阵操作数相关联的行数不同;和目的地矩阵(片)操作数相关联的行数与和第一源矩阵(片)操作数相关联的行数不同;以及和目的地矩阵(片)操作数相关联的列数与和第二源矩阵(片)操作数相关联的列数不同。

[0224] 在一些实施例中,在2509处,提交或引退指令。

[0225] 图26图示与由处理器执行以执行TILEDOTPRODUCT指令的示例方法有关的附加细节,其中,该指令具有用于第一源矩阵(片)操作数、第二源矩阵(片)操作数和目的地矩阵(片)累加器操作数的字段。

[0226] 在2601处，执行电路用值0设置第一计数器。在2602处，判定第一计数器是否小于目的地矩阵(片)操作数的所配置的行的数量。如果第一计数器不小于目的地矩阵(片)操作数的所配置的行的数量，则该过程结束。

[0227] 在2603处，如果第一计数器小于目的地矩阵(片)操作数的所配置的行的数量，则用值0设置第二计数器。在2604处，判定第二计数器是否小于第一源矩阵(片)操作数的所配置的列的数量。如果不是，则在2612处，使第一计数器递增，并且该过程返回到2602。

[0228] 在2605处，如果第二计数器小于第一源矩阵(片)操作数的所配置的列的数量，则将来自目的地矩阵(片)操作数的、由第一计数器标识的行写入临时位置。例如，如果第一计数器当前设置为0，则行[第一计数器]标识目的地矩阵(片)操作数的第一行。类似地，如果第一计数器当前设置为1，则行[第一计数器]标识目的地矩阵(片)累加器操作数的第二行，以此类推。在2606处，用值0设置第三计数器。

[0229] 在2607处，判定第三计数器是否小于目的地矩阵操作数的所配置的列的数量。如果第三计数器不小于目的地矩阵(片)累加器操作数的所配置的列的数量，则在2610处，将存储在临时位置处的数据值写入目的地矩阵(片)操作数的、由第一计数器标识的行(即，目的地矩阵(片)操作数的行[第一计数器])。在2611处，使第二计数器递增，并且该过程返回到2604。

[0230] 如果第三计数器小于目的地矩阵(片)累加器操作数的所配置的列的数量，则在2608处，执行电路执行涉及第一源矩阵(片)操作数在位置行[第一计数器,第二计数器]处的数据元素和第二源矩阵(片)操作数在位置行[第二计数器,第三计数器]处的数据元素的点积操作，并将结果累加在临时位置的元素位置[第三计数器]处。参考图24，假定第一计数器、第二计数器和第三计数器中的每一个当前都设置为0，则在2608处执行涉及第一源矩阵(片)操作数2402在位置行[0,0]处的数据元素(元素值[A,B])和第二源矩阵(片)操作数2403在位置行[0,0]处的数据元素(元素值[I,J])的点积操作，并将结果累加在(当前存储来自目的地矩阵(片)累加器操作数2401的行[0,0]的值W的)临时位置的元素位置[0]处。

[0231] 在2609处，使第三计数器递增，并且该过程返回到2607。图26中所描述的过程的结果是执行对来自第一源矩阵(片)操作数和第二源矩阵(片)操作数的元素的点积操作，并将结果累加到目的地矩阵(片)累加器操作数的元素中，如在图24的目的地矩阵(片)操作数2401中所图示。

[0232] 图27A-图27G图示用于执行如上所述的TILEDOTPRODUCT操作的示例方法。例如，2701和2703中示出的步骤图示用于执行TILEDOTPRODUCT操作的示例过程，该TILEDOTPRODUCT操作涉及有符号字尺寸的元素的源矩阵利用饱和被累加到双字尺寸的元素中(例如，基于示例格式TDPWSSDS TMM1,TMM2,TMM3的指令)。具体而言，2701图示用于对双字自变量执行乘加操作的示例助手过程DPWSS(c,x,y)。如在所伴随的过程2703中所示，在2701中图示的乘加操作被用作对源矩阵(片)操作数的行和列执行的点积计算的部分。

[0233] 2703的步骤指示：如果以下任一项为真，则生成错误：矩阵(片)架构当前不被配置；第一源矩阵(片)中的列数与第二源矩阵(片)中的行数不同；目的地矩阵(片)中的行数与第一源矩阵(片)中的行数不同；目的地矩阵(片)中的列数与第二源矩阵(片)中的列数不同；或者第二源矩阵(片)中的行数或列数超出所配置的限值。

[0234] 2703的过程进一步继续进行以迭代通过目的地矩阵(片)和源矩阵(片)的行和列。

具体而言,示例过程通过对来自第一源矩阵(片)操作数(“tsrc1”)和第二源矩阵(片)操作数(”tsrc2”)的元素执行点积操作来计算结果,并将结果累加到目的地矩阵(片)累加器操作数(“tsrcdest”)的元素中。

[0235] 图27B-图27C中示出的示例过程图示了被执行以实现其他示例TILEDOTPRODUCT指令格式的过程。例如,图27B-图27C中的2705和2707中示出的过程图示了对四字自变量执行乘加操作的示例助手函数DPWSSQ(c,x,y)。2707中示出的示例图示了用于执行TILEDOTPRODUCT操作的过程,该TILEDOTPRODUCT操作涉及有符号字尺寸的元素的矩阵累加到双字尺寸的元素中(例如,如上文所描述,基于具有格式TDPWSSQS TMM1,TMM2,TMM3的指令)。

[0236] 图27D中的2709中示出的过程图示了用于对双字自变量执行乘加操作的示例助手函数DPBD(c,x,y)。2711中示出的示例图示了用于执行TILEDOTPRODUCT操作的过程,该TILEDOTPRODUCT操作涉及字节尺寸的元素的矩阵累加到双字尺寸的元素中(例如,如上文所描述,基于具有格式TDPB[SS,UU,US,SU]DS TMM1,TMM2,TMM3的指令)。

[0237] 图27E-图27F中的2713中示出的过程图示了用于执行TILEDOTPRODUCT操作的过程,该TILEDOTPRODUCT操作涉及双字源矩阵(片)操作数,其中一个操作数包含字节尺寸的元素且另一操作数包含4位(半字节)尺寸的元素,并且结果被累加到包含双字尺寸的元素的目的地矩阵(片)中(例如,如上文所描述,基于具有格式TDP8B[SS,UU,US,SU]4BITDS TMM1,TMM2,TMM3的指令)。

[0238] 图27G中的2715中示出的过程图示了用于执行TILEDOTPRODUCT操作的过程,该TILEDOTPRODUCT操作涉及双字源矩阵(片)操作数,其中每个操作数包含4位(半字节)尺寸的元素,并且结果被累加到包含双字尺寸的元素的目的地矩阵(片)中(例如,如上文所描述,基于具有格式TDP4BIT[SS,UU,US,SU]DS TMM1,TMM2,TMM3的指令)。

[0239] iv.示例

[0240] 示例1一种处理器,包括:解码电路,用于对指令解码,该指令具有用于第一源矩阵操作数、第二源矩阵操作数和目的地矩阵操作数的字段;以及执行电路,用于执行经解码的指令以:通过对来自第一源矩阵操作数和第二源矩阵操作数的元素执行点积操作来计算结果;以及将结果累加到目的地矩阵操作数的数据元素位置中。

[0241] 示例2示例1的处理器,其中,第一源矩阵操作数和第二源矩阵操作数的元素是有符号字元素,并且其中,目的地矩阵操作数的元素是有符号双字。

[0242] 示例3示例1的处理器,其中,第一源矩阵操作数和第二源矩阵操作数的元素是有符号字尺寸的元素,并且其中,目的地矩阵操作数的元素是有符号四字尺寸的元素。

[0243] 示例4示例1的处理器,其中,第一源矩阵操作数和第二源矩阵操作数的元素是字节尺寸的元素,并且其中,目的地矩阵操作数的元素是双字尺寸的元素。

[0244] 示例5示例1的处理器,其中,第一源矩阵操作数的元素是字节尺寸的元素且第二源矩阵操作数的元素是4位尺寸的元素,并且其中,目的地矩阵的元素是双字尺寸的元素。

[0245] 示例6示例1的处理器,其中,第一源矩阵操作数的元素和第二源矩阵操作数的元素是4位尺寸的元素,并且其中,目的地矩阵的元素是双字尺寸的元素。

[0246] 示例7示例1-6中的任一项的处理器,其中,结果是利用饱和计算出的。

[0247] 示例8示例1-7中的任一项的处理器,其中,执行电路包括多个融合乘法加法器。

[0248] 示例9示例1-8中的任一项的处理器，其中，指令指示第一源矩阵操作数和第二源矩阵操作数中的至少一者包含无符号数据值。

[0249] 示例10示例1-9中的任一项的处理器，其中，当第一源矩阵操作数具有的列数与第二源矩阵操作数具有的行数不同时，错误被生成。

[0250] 示例11示例1-10中的任一项的处理器，其中，当目的地矩阵操作数的行数与第一源矩阵操作数的行数不同时，错误被生成。

[0251] 示例12示例1-11中的任一项的处理器，其中，当目的地矩阵操作数的列数与第二源矩阵操作数的列数不同时，错误被生成。

[0252] 示例13一种方法，包括：对指令解码，该指令具有用于第一源矩阵操作数、第二源矩阵操作数和目的地矩阵操作数的字段；以及执行经解码的指令以：通过对来自第一源矩阵操作数和第二源矩阵操作数的元素执行点积操作来计算结果；以及将结果累加到目的地矩阵操作数的数据元素位置中。

[0253] 示例14示例13的方法，其中，第一源矩阵操作数和第二源矩阵操作数的元素是有符号字元素，并且其中，目的地矩阵操作数的元素是有符号双字。

[0254] 示例15示例13的方法，其中，第一源矩阵操作数和第二源矩阵操作数的元素是有符号字尺寸的元素，并且其中，目的地矩阵操作数的元素是有符号四字尺寸的元素。

[0255] 示例16示例13的方法，其中，第一源矩阵操作数和第二源矩阵操作数的元素是字节尺寸的元素，并且其中，目的地矩阵操作数的元素是双字尺寸的元素。

[0256] 示例17示例13的方法，其中，第一源矩阵操作数的元素是字节尺寸的元素且第二源矩阵操作数的元素是4位尺寸的元素，并且其中，目的地矩阵的元素是双字尺寸的元素。

[0257] 示例18示例13的方法，其中，第一源矩阵操作数的元素和第二源矩阵操作数的元素是4位尺寸的元素，并且其中，目的地矩阵的元素是双字尺寸的元素。

[0258] 示例19示例13-18中的任一项的方法，其中，结果是利用饱和计算出的。

[0259] 示例20示例13-19中的任一项的方法，其中，执行电路包括多个融合乘法加法器。

[0260] 示例21示例13-20中的任一项的方法，其中，指令指示第一源矩阵操作数和第二源矩阵操作数中的至少一者包含无符号数据值。

[0261] 示例22示例13-21中的任一项的方法，其中，当第一源矩阵操作数具有的列数与第二源矩阵操作数具有的行数不同时，错误被生成。

[0262] 示例23示例13-22中的任一项的方法，其中，当目的地矩阵操作数的行数与第一源矩阵操作数的行数不同时，错误被生成。

[0263] 示例24示例13-23中的任一项的方法，其中，当目的地矩阵操作数的列数与第二源矩阵操作数的列数不同时，错误被生成。

[0264] 示例25提供一种非暂态机器可读介质，存储有指令，该指令当由处理器执行时使该处理器执行方法，该方法包括：对指令解码，该指令具有用于第一紧缩数据源操作数、第二紧缩数据源操作数和紧缩数据目的地操作数的字段；以及执行经解码的指令以：通过对来自第一源矩阵操作数和第二源矩阵操作数的元素执行点积操作来计算结果；以及将结果累加到目的地矩阵操作数的数据元素位置中。

[0265] 示例26示例25的非暂态机器可读介质，其中，第一源矩阵操作数和第二源矩阵操作数的元素是有符号字元素，并且其中，目的地矩阵操作数的元素是有符号双字。

[0266] 示例27示例25的非暂态机器可读介质，其中，第一源矩阵操作数和第二源矩阵操作数的元素是有符号字尺寸的元素，并且其中，目的地矩阵操作数的元素是有符号四字尺寸的元素。

[0267] 示例28示例25的非暂态机器可读介质，其中，第一源矩阵操作数和第二源矩阵操作数的元素是字节尺寸的元素，并且其中，目的地矩阵操作数的元素是双字尺寸的元素。

[0268] 示例29示例25的非暂态机器可读介质，其中，第一源矩阵操作数的元素是字节尺寸的元素且第二源矩阵操作数的元素是4位尺寸的元素，并且其中，目的地矩阵的元素是双字尺寸的元素。

[0269] 示例30示例25的非暂态机器可读介质，其中，第一源矩阵操作数的元素和第二源矩阵操作数的元素是4位尺寸的元素，并且其中，目的地矩阵的元素是双字尺寸的元素。

[0270] 示例31示例25-30中的任一项的非暂态机器可读介质，其中，结果是利用饱和计算出的。

[0271] 示例32示例25-31中的任一项的非暂态机器可读介质，其中，执行电路包括多个融合乘法加法器。

[0272] 示例33示例25-32中的任一项的非暂态机器可读介质，其中，指令指示第一源矩阵操作数和第二源矩阵操作数中的至少一者包含无符号数据值。

[0273] 示例34示例25-33中的任一项的非暂态机器可读介质，其中，当第一源矩阵操作数具有的列数与第二源矩阵操作数具有的行数不同时，错误被生成。

[0274] 示例35示例25-34中的任一项的非暂态机器可读介质，其中，当目的地矩阵操作数的行数与第一源矩阵操作数的行数不同时，错误被生成。

[0275] 示例36示例25-35中的任一项的非暂态机器可读介质，其中，当目的地矩阵操作数的列数与第二源矩阵操作数的列数不同时，错误被生成。

[0276] 示例37提供一种系统，该系统包括：处理器；以及耦合到该处理器的加速器，该加速器包括：解码电路，用于对指令解码，该指令具有用于第一源矩阵操作数、第二源矩阵操作数和目的地矩阵操作数的字段；以及执行电路，用于执行经解码的指令以：通过对来自第一源矩阵操作数和第二源矩阵操作数的元素执行点积操作来计算结果；以及将结果累加到目的地矩阵操作数的数据元素位置中。

[0277] 示例38示例37的系统，其中，第一源矩阵操作数和第二源矩阵操作数的元素是有符号字元素，并且其中，目的地矩阵操作数的元素是有符号双字。

[0278] 示例39示例37的系统，其中，第一源矩阵操作数和第二源矩阵操作数的元素是有符号字尺寸的元素，并且其中，目的地矩阵操作数的元素是有符号四字尺寸的元素。

[0279] 示例40示例37的系统，其中，第一源矩阵操作数和第二源矩阵操作数的元素是字节尺寸的元素，并且其中，目的地矩阵操作数的元素是双字尺寸的元素。

[0280] 示例41示例37的系统，其中，第一源矩阵操作数的元素是字节尺寸的元素且第二源矩阵操作数的元素是4位尺寸的元素，并且其中，目的地矩阵的元素是双字尺寸的元素。

[0281] 示例42示例37的系统，其中，第一源矩阵操作数的元素和第二源矩阵操作数的元素是4位尺寸的元素，并且其中，目的地矩阵的元素是双字尺寸的元素。

[0282] 示例43示例37-42中的任一项的系统，其中，结果是利用饱和计算出的。

[0283] 示例44示例37-43中的任一项的系统，其中，执行电路包括多个融合乘法加法器。

[0284] 示例45示例37-44中的任一项的系统，其中，指令指示第一源矩阵操作数和第二源矩阵操作数中的至少一者包含无符号数据值。

[0285] 示例46示例37-45中的任一项的系统，其中，当第一源矩阵操作数具有的列数与第二源矩阵操作数具有的行数不同时，错误被生成。

[0286] 示例47示例37-46中的任一项的系统，其中，当目的地矩阵操作数的行数与第一源矩阵操作数的行数不同时，错误被生成。

[0287] 示例48示例37-47中的任一项的系统，其中，当目的地矩阵操作数的列数与第二源矩阵操作数的列数不同时，错误被生成。

[0288] V.详细的示例性系统、处理器和仿真

[0289] 本文中详述的是用于执行上文描述的指令的硬件、软件等的示例。例如，下文所描述的内容详述了指令执行的多个方面，包括诸如取出、解码、调度、执行、引退等之类的各种流水线级。

[0290] 指令集包括一种或多种指令格式。给定的指令格式定义各种字段(位的数量、位的位置)以指定将要执行的操作(操作码)以及将对其执行该操作的(多个)操作数等等。通过指令模板(或子格式)的定义来进一步分解一些指令格式。例如，可将给定指令格式的指令模板定义为具有该指令格式的字段(所包括的字段通常按照相同顺序，但是至少一些字段具有不同的位的位置，因为较少的字段被包括)的不同子集，和/或定义为具有以不同方式进行解释的给定字段。由此，ISA的每一条指令使用给定的指令格式(并且如果经定义，则按照该指令格式的指令模板中的给定的一个指令模板)来表达，并包括用于指定操作和操作数的字段。例如，示例性ADD(加法)指令具有特定的操作码和指令格式，该特定的指令格式包括用于指定该操作码的操作码字段和用于选择操作数(源1/目的地以及源2)的操作数字段；并且该ADD指令在指令流中出现将使得在操作数字段中具有选择特定操作数的特定的内容。

[0291] A.示例性指令格式

[0292] 本文中所描述的(多条)指令的实施例能以不同的格式体现。另外，在下文中详述示例性系统、架构和流水线。(多条)指令的实施例可在此类系统、架构和流水线上执行，但是不限于详述的那些系统、架构和流水线。

[0293] VEX指令格式

[0294] VEX编码允许指令具有多于两个操作数，并且允许SIMD向量寄存器长于128位。VEX前缀的使用提供了三操作数(或者更多操作数)句法。例如，先前的两操作数指令执行诸如A＝A+B之类的覆写源操作数的操作。VEX前缀的使用使操作数能执行诸如A＝B+C之类的非破坏性操作。

[0295] 图28A图示出示例性指令格式，包括VEX前缀2802、实操作码字段2830、Mod R/M字节2840、SIB字节2850、位移字段2862以及IMM8 2872。图28B图示出来自图28A的哪些字段构成完整操作码字段2874和基础操作字段2841。图28C图示出来自图28A的哪些字段构成寄存器索引字段2844。

[0296] VEX前缀(字节0-2)2802以三字节的形式进行编码。第一字节是格式字段2890(VEX字节0，位[7:0])，该格式字段2890包含显式的C4字节值(用于区分C4指令格式的唯一值)。第二－第三字节(VEX字节1-2)包括提供专用能力的数个位字段。具体地，REX字段2805(VEX

字节1,位[7-5])由VEX.R位字段(VEX字节1,位[7]-R)、VEX.X位字段(VEX字节1,位[6]-X)以及VEX.B位字段(VEX字节1,位[5]-B)组成。这些指令的其他字段对如在本领域中已知的寄存器索引的较低的三个位(rrr、xxx以及bbb)进行编码,以使得可通过增加VEX.R、VEX.X以及VEX.B来形成Rrrr、Xxxx以及Bbbb。操作码映射字段2815(VEX字节1,位[4:0]-mmmmm)包括用于对隐含的前导操作码字节进行编码的内容。W字段2864(VEX字节2,位[7]-W)——由记号VEX.W表示,并且提供取决于该指令的不同功能。VEX.vvvv 2820(VEX字节2,位[6:3]-vvvv)的作用可包括如下:1)VEX.vvvv对以反转(1补码)的形式被指定的第一源寄存器操作数进行编码,并且对具有两个或更多个源操作数的指令有效;2)VEX.vvvv对针对某些向量位移以1补码的形式被指定的目的地寄存器操作数进行编码;或者3)VEX.vvvv不对任何操作数进行编码,该字段被保留并且应当包含1111b。如果VEX.L 2868尺寸字段(VEX字节2,位[2]-L)=0,则它指示128位向量;如果VEX.L=1,则它指示256位向量。前缀编码字段2825(VEX字节2,位[1:0]-pp)提供用于基础操作字段2841的附加位。

[0297] 实操作码字段2830(字节3)还被称为操作码字节。操作码的一部分在该字段中被指定。

[0298] MOD R/M字段2840(字节4)包括MOD字段2842(位[7-6])、Reg字段2844(位[5-3])和R/M字段2846(位[2-0])。Reg字段2844的作用可包括如下:对目的地寄存器操作数或源寄存器操作数(Rrrr的rrr)进行编码;或者被视为操作码扩展,并且不用于对任何指令操作数进行编码。R/M字段2846的作用可包括如下:对引用存储器地址的指令操作数进行编码;或者对目的地寄存器操作数或源寄存器操作数进行编码。

[0299] 比例、索引、基址(SIB)——比例字段2850(字节5)的内容包括SS2852(位[7-6]),其用于存储器地址生成。先前已经针对寄存器索引Xxxx和Bbbb提及了SIB.xxx 2854(位[5-3])和SIB.bbb 2856(位[2-0])的内容。

[0300] 位移字段2862和立即数字段(IMM8)2872包含数据。

[0301] B.示例性寄存器架构

[0302] 图29是根据本发明的一个实施例的寄存器架构2900的框图。在所图示的实施例中,有32个512位宽的向量寄存器2910;这些寄存器被引用为zmm0到zmm31。较低的32个zmm寄存器的较低阶256个位覆盖(overlay)在寄存器ymm0-15上。较低的32个zmm寄存器的较低阶128个位(ymm寄存器的较低阶128个位)覆盖在寄存器xmm0-15上。

[0303] 通用寄存器2925——在所示出的实施例中,有十六个64位通用寄存器,这些寄存器与现有的x86寻址模式一起使用以对存储器操作数寻址。这些寄存器通过名称RAX、RBX、RCX、RDX、RBP、RSI、RDI、RSP以及R8到R15来引用。

[0304] 标量浮点栈寄存器堆(x87栈)2945,在其上面重叠了MMX紧缩整数平坦寄存器堆2950——在所图示的实施例中,x87栈是用于使用x87指令集扩展来对32/64/80位浮点数据执行标量浮点操作的八元素栈;而使用MMX寄存器来对64位紧缩整数数据执行操作,以及为在MMX与XMM寄存器之间执行的一些操作保存操作数。

[0305] 在一些实施例中,使用物理寄存器上方的叠加结构来支持片2920。例如,取决于实现方式,片可以利用16个1024位的寄存器、32个512位的寄存器等等。

[0306] 本发明的替代实施例可以使用更宽的或更窄的寄存器。另外,本发明的替代实施例可以使用更多、更少或不同的寄存器堆和寄存器。

[0307] 示例性核架构、处理器和计算机架构

[0308] 处理器核能以不同方式、出于不同的目的、在不同的处理器中实现。例如，此类核的实现可以包括：1)旨在用于通用计算的通用有序核；2)旨在用于通用计算的高性能通用乱序核；3)旨在主要用于图形和/或科学(吞吐量)计算的专用核。不同处理器的实现可包括：1)CPU，其包括旨在用于通用计算的一个或多个通用有序核和/或旨在用于通用计算的一个或多个通用乱序核；以及2)协处理器，其包括旨在主要用于图形和/或科学(吞吐量)的一个或多个专用核。此类不同的处理器导致不同的计算机系统架构，这些计算机系统架构可包括：1)在与CPU分开的芯片上的协处理器；2)在与CPU相同的封装中但在分开的管芯上的协处理器；3)与CPU在相同管芯上的协处理器(在该情况下，此类协处理器有时被称为专用逻辑或被称为专用核，该专用逻辑诸如，集成图形和/或科学(吞吐量)逻辑)；以及4)芯片上系统，其可以将所描述的CPU(有时被称为(多个)应用核或(多个)应用处理器)、以上描述的协处理器和附加功能包括在同一管芯上。接着描述示例性核架构，随后描述示例性处理器和计算机架构。本文中详细描述了包括示例性核、处理器等的电路(单元)。

[0309] C.示例性核架构

[0310] 有序和乱序核框图

[0311] 图30A是图示根据本发明的各实施例的示例性有序流水线和示例性的寄存器重命名的乱序发布/执行流水线的框图。图30B是示出根据本发明的各实施例的要包括在处理器中的有序架构核的示例性实施例和示例性的寄存器重命名的乱序发布/执行架构核的框图。图30A-图30B中的实线框图示有序流水线和有序核，而虚线框的任选增加图示寄存器重命名的、乱序发布/执行流水线和核。考虑到有序方面是乱序方面的子集，将描述乱序方面。

[0312] 在图30A中，处理器流水线3000包括取出级3002、长度解码级3004、解码级3006、分配级3008、重命名级3010、调度(也被称为分派或发布)级3012、寄存器读取/存储器读取级3014、执行级3016、写回/存储器写入级3018、异常处置级3022和提交级3024。

[0313] 图30B示出处理器核3090，该处理器核3090包括前端单元3030，该前端单元3030耦合到执行引擎单元3050，并且前端单元3030和执行引擎单元3050两者都耦合到存储器单元3070。核3090可以是精简指令集计算(RISC)核、复杂指令集计算(CISC)核、超长指令字(VLIW)核、或混合或替代的核类型。作为又一选项，核3090可以是专用核，诸如例如，网络或通信核、压缩引擎、协处理器核、通用计算图形处理单元(GPGPU)核、图形核，等等。

[0314] 前端单元3030包括分支预测单元3032，该分支预测单元3032耦合到指令高速缓存单元3034，该指令高速缓存单元3034耦合到指令转换后备缓冲器(TLB)3036，该指令转换后备缓冲器3036耦合到指令取出单元3038，该指令取出单元3038耦合到解码单元3040。解码单元3040(或解码器)可对指令解码，并且生成从原始指令解码出的、或以其他方式反映原始指令的、或从原始指令导出的一个或多个微操作、微代码进入点、微指令、其他指令、或其他控制信号作为输出。解码单元3040可使用各种不同的机制来实现。合适机制的示例包括但不限于，查找表、硬件实现、可编程逻辑阵列(PLA)、微代码只读存储器(ROM)等。在一个实施例中，核3090包括存储用于某些宏指令的微代码的微代码ROM或其他介质(例如，在解码单元3040中，或以其他方式在前端单元3030内)。解码单元3040耦合到执行引擎单元3050中的重命名/分配器单元3052。

[0315] 执行引擎单元3050包括重命名/分配器单元3052，该重命名/分配器单元3052耦合

到引退单元3054和一个或多个调度器单元的集合3056。(多个)调度器单元3056表示任何数量的不同调度器,包括预留站、中央指令窗等。(多个)调度器单元3056耦合到(多个)物理寄存器堆单元3058。(多个)物理寄存器堆单元3058中的每一个物理寄存器堆单元表示一个或多个物理寄存器堆,其中不同的物理寄存器堆存储一种或多种不同的数据类型,诸如,标量整数、标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点,状态(例如,作为要执行的下一条指令的地址的指令指针)等等。在一个实施例中,(多个)物理寄存器堆单元3058包括向量寄存器单元和标量寄存器单元。这些寄存器单元可以提供架构向量寄存器、向量掩码寄存器和通用寄存器。(多个)物理寄存器堆单元3058被引退单元3054覆盖,以图示可实现寄存器重命名和乱序执行的各种方式(例如,使用(多个)重排序缓冲器和(多个)引退寄存器堆;使用(多个)未来文件、(多个)历史缓冲器、(多个)引退寄存器堆;使用寄存器映射和寄存器池,等等)。引退单元3054和(多个)物理寄存器堆单元3058耦合到(多个)执行集群3060。(多个)执行集群3060包括一个或多个执行单元的集合3062以及一个或多个存储器访问单元的集合3064。执行单元3062可执行各种操作(例如,移位、加法、减法、乘法)并可对各种数据类型(例如,标量浮点、紧缩整数、紧缩浮点、向量整数、向量浮点)执行。尽管一些实施例可以包括专用于特定功能或功能集合的多个执行单元,但是其他实施例可包括仅一个执行单元或全都执行所有功能的多个执行单元。(多个)调度器单元3056、(多个)物理寄存器堆单元3058和(多个)执行集群3060示出为可能有多个,因为某些实施例为某些类型的数据/操作创建分开的流水线(例如,标量整数流水线、标量浮点/紧缩整数/紧缩浮点/向量整数/向量浮点流水线,和/或各自具有其自身的调度器单元、(多个)物理寄存器堆单元和/或执行集群的存储器访问流水线——并且在分开的存储器访问流水线的情况下,实现其中仅该流水线的执行集群具有(多个)存储器访问单元3064的某些实施例)。还应当理解,在使用分开的流水线的情况下,这些流水线中的一个或多个可以是乱序发布/执行,并且其余流水线可以是有序的。

[0316] 存储器访问单元的集合3064耦合到存储器单元3070,该存储器单元3070包括数据TLB单元3072,该数据TLB单元3072耦合到数据高速缓存单元3074,该数据高速缓存单元3074耦合到第二级(L2)高速缓存单元3076。在一个示例性实施例中,存储器访问单元3064可包括加载单元、存储地址单元和存储数据单元,其中的每一个均耦合到存储器单元3070中的数据TLB单元3072。指令高速缓存单元3034还耦合到存储器单元3070中的第二级(L2)高速缓存单元3076。L2高速缓存单元3076耦合到一个或多个其他级别的高速缓存,并最终耦合到主存储器。

[0317] 作为示例,示例性寄存器重命名的乱序发布/执行核架构可如下所述地实现流水线3000:1)指令取出3038执行取出级3002和长度解码级3004;2)解码单元3040执行解码级3006;3)重命名/分配器单元3052执行分配级3008和重命名级3010;4)(多个)调度器单元3056执行调度级3012;5)(多个)物理寄存器堆单元3058和存储器单元3070执行寄存器读取/存储器读取级3014;执行集群3060执行执行级3016;6)存储器单元3070和(多个)物理寄存器堆单元3058执行写回/存储器写入级3018;7)各单元可牵涉到异常处置级3022;以及8)引退单元3054和(多个)物理寄存器堆单元3058执行提交级3024。

[0318] 核3090可支持一个或多个指令集(例如,x86指令集(具有已与较新版本一起添加的一些扩展);加利福尼亚州桑尼维尔市的MIPS技术公司的MIPS指令集;加利福尼亚州桑尼

维尔市的ARM控股公司的ARM指令集(具有诸如NEON的任选的附加扩展)),其中包括本文中描述的(多条)指令。在一个实施例中,核3090包括用于支持紧缩数据指令集扩展(例如,AVX1、AVX2)的逻辑,由此允许使用紧缩数据来执行由许多多媒体应用使用的操作。

[0319] 应当理解,核可支持多线程化(执行两个或更多个并行的操作或线程的集合),并且可以按各种方式来完成该多线程化,各种方式包括时分多线程化、同时多线程化(其中单个物理核为物理核正在同时多线程化的线程中的每一个线程提供逻辑核)、或其组合(例如,时分取出和解码以及此后的诸如英特尔®超线程化技术中的同时多线程化)。

[0320] 尽管在乱序执行的上下文中描述了寄存器重命名,但应当理解,可以在有序架构中使用寄存器重命名。尽管所图示的处理器的实施例还包括分开的指令和数据高速缓存单元3034/3074以及共享的L2高速缓存单元3076,但是替代实施例可以具有用于指令和数据两者的单个内部高速缓存,诸如例如,第一级(L1)内部高速缓存或多个级别的内部高速缓存。在一些实施例中,该系统可包括内部高速缓存和在核和/或处理器外部的外部高速缓存的组合。或者,所有高速缓存都可以在核和/或处理器的外部。

[0321] 具体的示例性有序核架构

[0322] 图31A-图31B图示更具体的示例性有序核架构的框图,该核将是芯片中的若干逻辑块(包括相同类型和/或不同类型的其他核)中的一个逻辑块。取决于应用,逻辑块通过高带宽互连网络(例如,环形网络)与一些固定的功能逻辑、存储器I/O接口和其他必要的I/O逻辑进行通信。

[0323] 图31A是根据本发明的实施例的单个处理器核以及它至管芯上互连网络3102的连接及其第二级(L2)高速缓存的本地子集3104的框图。在一个实施例中,指令解码器3100支持具有紧缩数据指令集扩展的x86指令集。L1高速缓存3106允许对进入标量和向量单元中的、对高速缓存存储器的低等待时间访问。尽管在一个实施例中(为了简化设计),标量单元3108和向量单元3110使用分开的寄存器集合(分别为标量寄存器3112和向量寄存器3114),并且在这些寄存器之间传输的数据被写入到存储器,并随后从第一级(L1)高速缓存3106读回,但是本发明的替代实施例可以使用不同的方法(例如,使用单个寄存器集合或包括允许数据在这两个寄存器堆之间传输而无需被写入和读回的通信路径)。

[0324] L2高速缓存的本地子集3104是全局L2高速缓存的一部分,该全局L2高速缓存被划分成多个分开的本地子集,每个处理器核一个本地子集。每个处理器核具有到其自身的L2高速缓存的本地子集3104的直接访问路径。由处理器核读取的数据被存储在其L2高速缓存子集3104中,并且可以与其他处理器核访问其自身的本地L2高速缓存子集并行地被快速访问。由处理器核写入的数据被存储在其自身的L2高速缓存子集3104中,并在必要的情况下从其他子集转储清除。环形网络确保共享数据的一致性。环形网络是双向的,以允许诸如处理器核、L2高速缓存和其他逻辑块之类的代理在芯片内彼此通信。在一些实施例中,每个环形数据路径为每个方向1024位宽。

[0325] 图31B是根据本发明的实施例的图31A中的处理器核的一部分的展开图。图31B包括L1高速缓存3104的L1数据高速缓存3106A部分,以及关于向量单元3110和向量寄存器3114的更多细节。具体地,向量单元3110是32宽向量处理单元(VPU)(见16宽ALU 3128),该单元执行整数、单精度浮点以及双精度浮点指令中的一个或多个。该VPU通过混合单元3120支持对寄存器输入的混合,通过数值转换单元3122A-B支持数值转换,并且通过复制单元

3124支持对存储器输入的复制。

[0326] 具有集成存储器控制器和图形器件的处理器

[0327] 图32是根据本发明的实施例的可具有多于一个的核、可具有集成存储器控制器、以及可具有集成图形器件的处理器3200的框图。图32中的实线框图示具有单个核3202A、系统代理3210、一个或多个总线控制器单元的集合3216的处理器3200，而虚线框的任选增加图示具有多个核3202A-N、系统代理单元3210中的一个或多个集成存储器控制器单元的集合3214以及专用逻辑3208的替代处理器3200。

[0328] 因此，处理器3200的不同实现可包括：1)CPU，其中专用逻辑3208是集成图形和/或科学(吞吐量)逻辑(其可包括一个或多个核)，并且核3202A-N是一个或多个通用核(例如，通用有序核、通用乱序核、这两者的组合)；2)协处理器，其中核3202A-N是旨在主要用于图形和/或科学(吞吐量)的大量专用核；以及3)协处理器，其中核3202A-N是大量通用有序核。因此，处理器3200可以是通用处理器、协处理器或专用处理器，诸如例如，网络或通信处理器、压缩引擎、图形处理器、GPGPU(通用图形处理单元)、高吞吐量的集成众核(MIC)协处理器(包括30个或更多核)、嵌入式处理器，等等。该处理器可以被实现在一个或多个芯片上。处理器3200可以是一个或多个基板的一部分，和/或可使用多种工艺技术(诸如例如，BiCMOS、CMOS、或NMOS)中的任何技术被实现在一个或多个基板上。

[0329] 存储器层次结构包括核3204A-N内的一个或多个级别的高速缓存、一个或多个共享高速缓存单元的集合3206、以及耦合到集成存储器控制器单元的集合3214的外部存储器(未示出)。共享高速缓存单元的集合3206可包括一个或多个中间级别的高速缓存，诸如，第二级(L2)、第三级(L3)、第四级(L4)或其他级别的高速缓存、末级高速缓存(LLC)和/或以上各项的组合。虽然在一个实施例中，基于环的互连单元3212将集成图形逻辑3208、共享高速缓存单元的集合3206以及系统代理单元3210/(多个)集成存储器控制器单元3214互连，但是替代实施例可使用任何数量的公知技术来互连此类单元。在一个实施例中，在一个或多个高速缓存单元3206与核3202A-N之间维持一致性。

[0330] 在一些实施例中，核3202A-N中的一个或多个能够实现多线程化。系统代理3210包括协调和操作核3202A-N的那些部件。系统代理单元3210可包括例如功率控制单元(PCU)和显示单元。PCU可以是对核3202A-N以及集成图形逻辑3208的功率状态进行调节所需的逻辑和部件，或可包括这些逻辑和部件。显示单元用于驱动一个或多个外部连接的显示器。

[0331] 核3202A-N在架构指令集方面可以是同构的或异构的；即，核3202A-N中的两个或更多个核可能能够执行相同的指令集，而其他核可能能够执行该指令集的仅仅子集或不同的指令集。

[0332] D.示例性计算机架构

[0333] 图33-图36是示例性计算机架构的框图。本领域中已知的对膝上型设备、台式机、手持PC、个人数字助理、工程工作站、服务器、网络设备、网络集线器、交换机、嵌入式处理器、数字信号处理器(DSP)、图形设备、视频游戏设备、机顶盒、微控制器、蜂窝电话、便携式媒体播放器、手持设备以及各种其他电子设备的其他系统设计和配置也是合适的。一般地，能够包含如本文中所公开的处理器和/或其他执行逻辑的各种各样的系统或电子设备一般都是合适的。

[0334] 现在参考图33，所示出的是根据本发明一个实施例的系统3300的框图。系统3300

可以包括一个或多个处理器3310、3315，这些处理器耦合到控制器中枢3320。在一个实施例中，控制器中枢3320包括图形存储器控制器中枢(GMCH)3390和输入/输出中枢(IOH)3350(其可以在分开的芯片上)；GMCH 3390包括存储器和图形控制器，存储器3340和协处理器3345耦合到该存储器和图形控制器；IOH 3350将输入/输出(I/O)设备3360耦合到GMCH 3390。或者，存储器和图形控制器中的一个或这两者被集成在(如本文中所描述的)处理器内，存储器3340和协处理器3345直接耦合到处理器3310，并且控制器中枢3320与IOH 3350处于单个芯片中。

[0335] 附加的处理器3315的任选性在图33中通过虚线来表示。每一处理器3310、3315可包括本文中描述的处理核中的一个或多个，并且可以是处理器3200的某一版本。

[0336] 存储器3340可以是例如动态随机存取存储器(DRAM)、相变存储器(PCM)或这两者的组合。对于至少一个实施例，控制器中枢3320经由诸如前端总线(FSB)之类的多分支总线、点对点接口、或者类似的连接3395来与(多个)处理器3310、3315进行通信。

[0337] 在一个实施例中，协处理器3345是专用处理器，诸如例如，高吞吐量MIC处理器、网络或通信处理器、压缩引擎、图形处理器、GPGPU、嵌入式处理器，等等。在一个实施例中，控制器中枢3320可以包括集成图形加速器。

[0338] 在物理资源3310、3315之间可以存在包括架构、微架构、热、功耗特性等一系列品质度量方面的各种差异。

[0339] 在一个实施例中，处理器3310执行控制一般类型的数据处理操作的指令。嵌入在这些指令内的可以是协处理器指令。处理器3310将这些协处理器指令识别为具有应当由附连的协处理器3345执行的类型。因此，处理器3310在协处理器总线或者其他互连上将这些协处理器指令(或者表示协处理器指令的控制信号)发布到协处理器3345。(多个)协处理器3345接受并执行所接收的协处理器指令。

[0340] 现在参见图34，所示出的是根据本发明的实施例的第一更具体的示例性系统3400的框图。如图34中所示，多处理器系统3400是点对点互连系统，并且包括经由点对点互连3450耦合的第一处理器3470和第二处理器3480。处理器3470和3480中的每一个都可以是处理器3200的某一版本。在本发明的一个实施例中，处理器3470和3480分别是处理器3310和3315，而协处理器3438是协处理器3345。在另一实施例中，处理器3470和3480分别是处理器3310和协处理器3345。

[0341] 处理器3470和3480示出为分别包括集成存储器控制器(IMC)单元3472和3482。处理器3470还包括作为其总线控制器单元的一部分的点对点(P-P)接口3476和3478；类似地，第二处理器3480包括P-P接口3486和3488。处理器3470、3480可以经由使用点对点(P-P)接口电路3478、3488的P-P接口3450来交换信息。如图34中所示，IMC 3472和3482将处理器耦合到相应的存储器，即存储器3432和存储器3434，这些存储器可以是本地附连到相应处理器的主存储器的部分。

[0342] 处理器3470、3480可各自经由使用点对点接口电路3476、3494、3486、3498的各个P-P接口3452、3454来与芯片组3490交换信息。芯片组3490可以任选地经由高性能接口3492来与协处理器3438交换信息。在一个实施例中，协处理器3438是专用处理器，诸如例如，高吞吐量MIC处理器、网络或通信处理器、压缩引擎、图形处理器、GPGPU、嵌入式处理器，等等。

[0343] 共享高速缓存(未示出)可被包括在任一处理器中，或在这两个处理器的外部但经

由P-P互连与这些处理器连接，使得如果处理器被置于低功率模式，则任一个或这两个处理器的本地高速缓存信息可被存储在共享高速缓存中。

[0344] 芯片组3490可以经由接口3496耦合到第一总线3416。在一个实施例中，第一总线3416可以是外围部件互连(PCI)总线或诸如PCI快速总线或另一I/O互连总线之类的总线，但是本发明的范围不限于此。

[0345] 如图34中所示，各种I/O设备3414可连同总线桥3418一起耦合到第一总线3416，该总线桥3418将第一总线3416耦合到第二总线3420。在一个实施例中，诸如协处理器、高吞吐量MIC处理器、GPGPU、加速器(诸如例如，图形加速器或数字信号处理(DSP)单元)、现场可编程门阵列或任何其他处理器的一个或多个附加处理器3415耦合到第一总线3416。在一个实施例中，第二总线3420可以是低引脚数(LPC)总线。在一个实施例中，各种设备可耦合到第二总线3420，这些设备包括例如键盘和/或鼠标3422、通信设备3427以及存储单元3428，该存储单元3428诸如可包括指令/代码和数据3430的盘驱动器或者其他大容量存储设备。此外，音频I/O 3424可以被耦合到第二总线3420。注意，其他架构是可能的。例如，代替图34的点对点架构，系统可以实现多分支总线或其他此类架构。

[0346] 现在参考图35，示出的是根据本发明的实施例的第二更具体的示例性系统3500的框图。图34和35中的类似元件使用类似的附图标记，并且从图35中省略了图34的某些方面以避免混淆图35的其他方面。

[0347] 图35图示处理器3470、3480可分别包括集成存储器和I/O控制逻辑(“CL”)3572和3582。因此，CL 3572、3582包括集成存储器控制器单元，并包括I/O控制逻辑。图35图示不仅存储器3432、3434耦合到CL 3572、3582，而且I/O设备3514也耦合到控制逻辑3572、3582。传统I/O设备3515被耦合到芯片组3490。

[0348] 现在参考图36，示出的是根据本发明的实施例的SoC 3600的框图。图32中的类似要素使用类似的附图标记。另外，虚线框是更先进的SoC上的任选的特征。在图36中，(多个)互连单元3602被耦合到：应用处理器3610，其包括一个或多个核的集合3202A-N、高速缓存单元3204A-N以及(多个)共享高速缓存单元3206；系统代理单元3210；(多个)总线控制器单元3216；(多个)集成存储器控制器单元3214；一个或多个协处理器的集合3620，其可包括集成图形逻辑、图像处理器、音频处理器和视频处理器；静态随机存取存储器(SRAM)单元3630；直接存储器访问(DMA)单元3632；以及用于耦合到一个或多个外部显示器的显示单元3640。在一个实施例中，(多个)协处理器3620包括专用处理器，诸如例如，网络或通信处理器、压缩引擎、GPGPU、高吞吐量MIC处理器、或嵌入式处理器，等等。

[0349] 本文公开的机制的各实施例可以被实现在硬件、软件、固件或此类实现方式的组合中。本发明的实施例可实现为在可编程系统上执行的计算机程序或程序代码，该可编程系统包括至少一个处理器、存储系统(包括易失性和非易失性存储器和/或存储元件)、至少一个输入设备以及至少一个输出设备。

[0350] 可将程序代码(诸如，图34中图示的代码3430)应用于输入指令，以执行本文中描述的功能并生成输出信息。可以按已知方式将输出信息应用于一个或多个输出设备。为了本申请的目的，处理系统包括具有处理器的任何系统，该处理器诸如例如，数字信号处理器(DSP)、微控制器、专用集成电路(ASIC)或微处理器。

[0351] 程序代码可以用高级的面向过程的编程语言或面向对象的编程语言来实现，以便

与处理系统通信。如果需要，也可用汇编语言或机器语言来实现程序代码。事实上，本文中描述的机制不限于任何特定的编程语言的范围。在任何情况下，该语言可以是编译语言或解释语言。

[0352] 至少一个实施例的一个或多个方面可以由存储在机器可读介质上的表示性指令来实现，该指令表示处理器中的各种逻辑，该指令在被机器读取时使得该机器制造用于执行本文中所述的技术的逻辑。被称为“IP核”的此类表示可以被存储在有形的机器可读介质上，并可被供应给各个客户或生产设施以加载到实际制造该逻辑或处理器的制造机器中。

[0353] 此类机器可读存储介质可以包括但不限于通过机器或设备制造或形成的制品的非暂态、有形布置，其包括存储介质，诸如硬盘；任何其他类型的盘，包括软盘、光盘、紧致盘只读存储器(CD-ROM)、可重写紧致盘(CD-RW)以及磁光盘；半导体器件，诸如，只读存储器(ROM)、诸如动态随机存取存储器(DRAM)和静态随机存取存储器(SRAM)的随机存取存储器(RAM)、可擦除可编程只读存储器(EPROM)、闪存、电可擦除可编程只读存储器(EEPROM)；相变存储器(PCM)；磁卡或光卡；或适于存储电子指令的任何其他类型的介质。

[0354] 因此，本发明的实施例还包括非暂态的有形机器可读介质，该介质包含指令或包含设计数据，诸如硬件描述语言(HDL)，它定义本文中描述的结构、电路、装置、处理器和/或系统特征。这些实施例也被称为程序产品。

[0355] E.仿真(包括二进制变换、代码变形等)

[0356] 在一些情况下，指令转换器可用于将指令从源指令集转换至目标指令集。例如，指令转换器可以将指令变换(例如，使用静态二进制变换、包括动态编译的动态二进制变换)、变形、仿真或以其他方式转换成要由核处理的一条或多条其他指令。指令转换器可以用软件、硬件、固件、或其组合来实现。指令转换器可以在处理器上、在处理器外、或者部分在处理器上且部分在处理器外。

[0357] 图37是根据本发明的实施例的对照使用软件指令转换器将源指令集中的二进制指令转换成目标指令集中的二进制指令的框图。在所图示的实施例中，指令转换器是软件指令转换器，但替代地，该指令转换器可以用软件、固件、硬件或其各种组合来实现。图37示出可使用第一编译器3704来编译高级语言3702形式的程序，以生成可由具有至少一个第一指令集核的处理器3716原生执行的第一二进制代码(例如，x86)3706。在一些实施例中，具有至少一个第一指令集核的处理器3716表示通过兼容地执行或以其他方式处理以下各项来执行与具有至少一个x86指令集核的英特尔处理器基本相同的功能的任何处理器：1)英特尔x86指令集核的指令集的实质部分，或2)目标为在具有至少一个x86指令集核的英特尔处理器上运行以便取得与具有至少一个x86指令集核的英特尔处理器基本相同的结果的应用或其他软件的目标代码版本。第一编译器3704表示可操作用于生成第一指令集中的二进制代码3706(例如，目标代码)的编译器，该二进制代码可通过或不通过附加的链接处理在具有至少一个第一指令集核的处理器3716上执行。类似地，图37示出可以使用替代的指令集编译器3708来编译高级语言3702形式的程序，以生成可以由不具有至少一个第一指令集核的处理器3714(例如，具有执行加利福尼亚州桑尼维尔市的MIPS技术公司的MIPS指令集、和/或执行加利福尼亚州桑尼维尔市的ARM控股公司的ARM指令集的核的处理器)原生执行的替代的指令集二进制代码3710。指令转换器3712用于将第一二进制代码3706转换成可以由不具有第一指令集核的处理器3714原生执行的代码。该转换后的代码不大可能与替代的

指令集二进制代码3710相同，因为能够这样做的指令转换器难以制造；然而，转换后的代码将完成一般操作，并且由来自替代指令集的指令构成。因此，指令转换器3712通过仿真、模拟或任何其他过程来表示允许不具有第一指令集处理器或核的处理器或其他电子设备执行第一二进制代码3706的软件、固件、硬件或其组合。

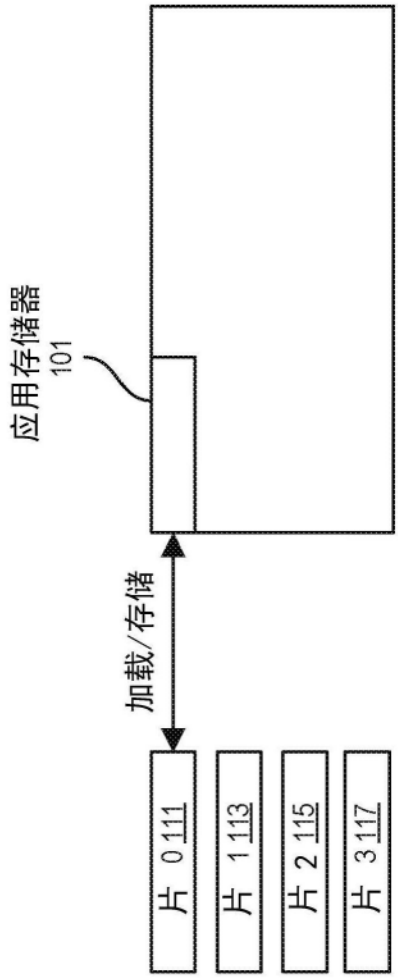


图1

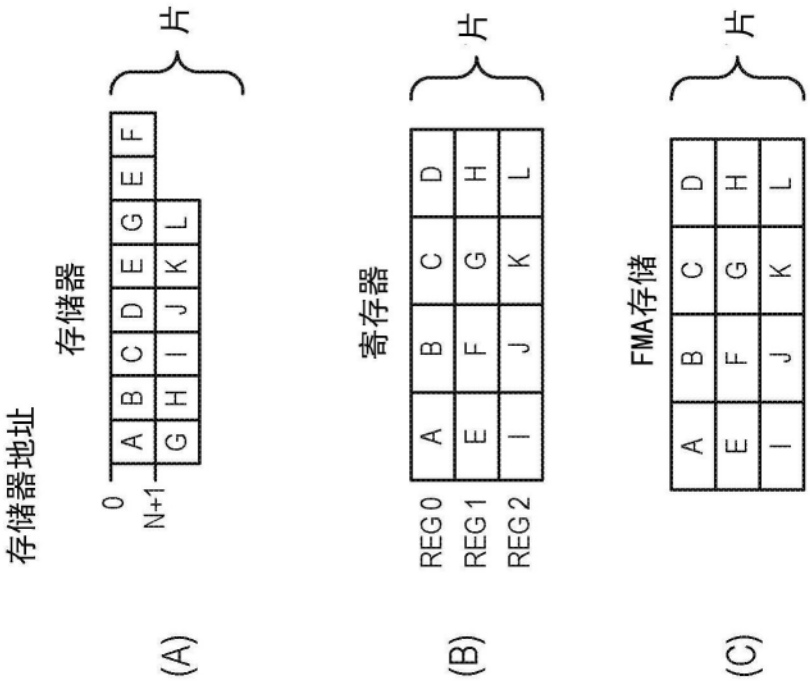

存储器地址
存储器
0
N+1
A B C D E F
G H I J K L
片
(A)
寄存器
REG 0
REG 1
REG 2
A B C D
E F G H
I J K L
片
(B)
FMA存储
A B C D
E F G H
I J K L
片
(C)

图2

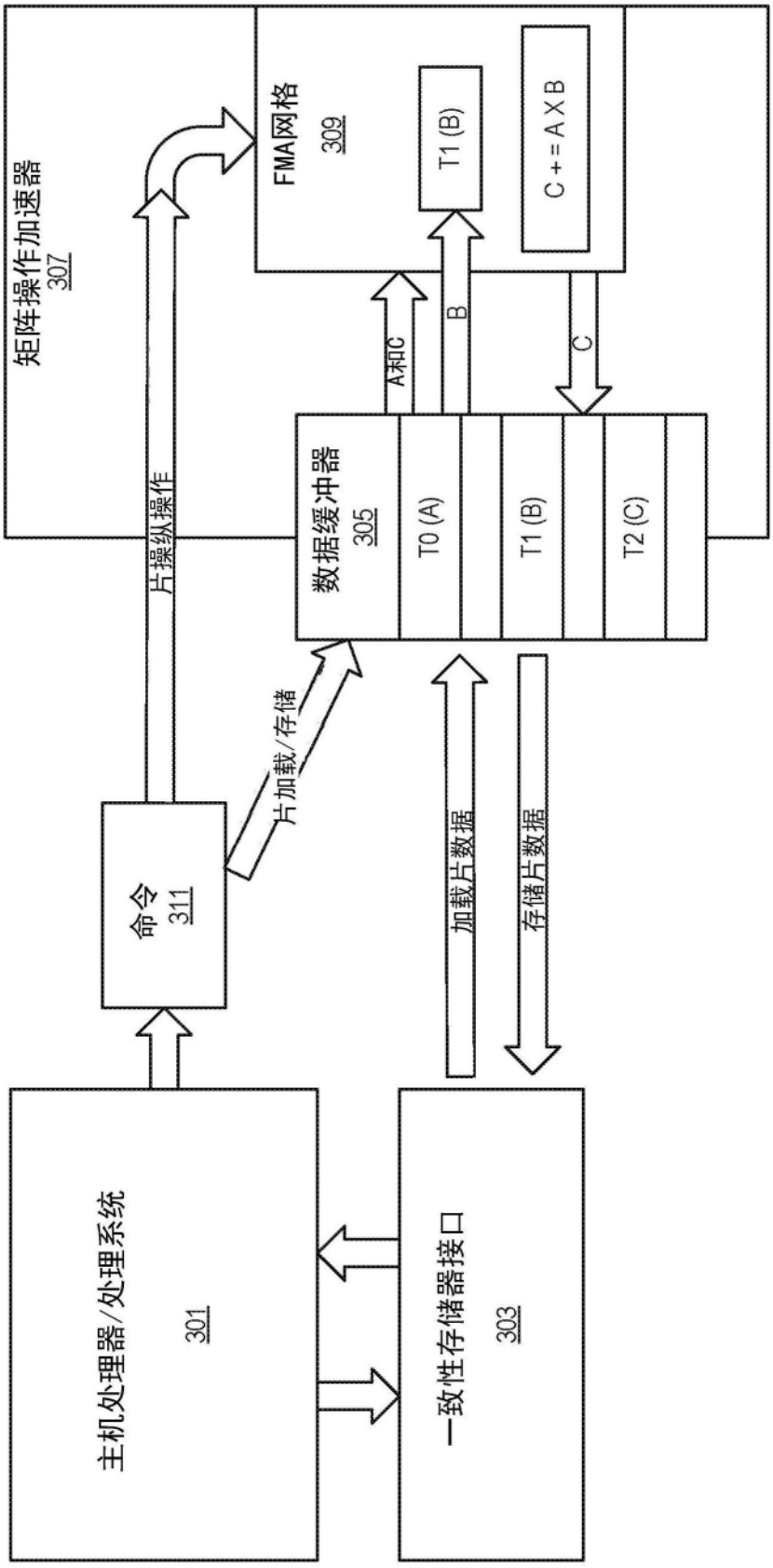

矩阵操作加速器
307
FMA网格
309
T1 (B)
C += A X B
A和C
B
C
片操纵操作
数据缓冲器
305
T0 (A)
T1 (B)
T2 (C)
片加载/存储
命令
311
加载片数据
存储片数据
主机处理器/处理系统
301
一致性存储器接口
303

图3

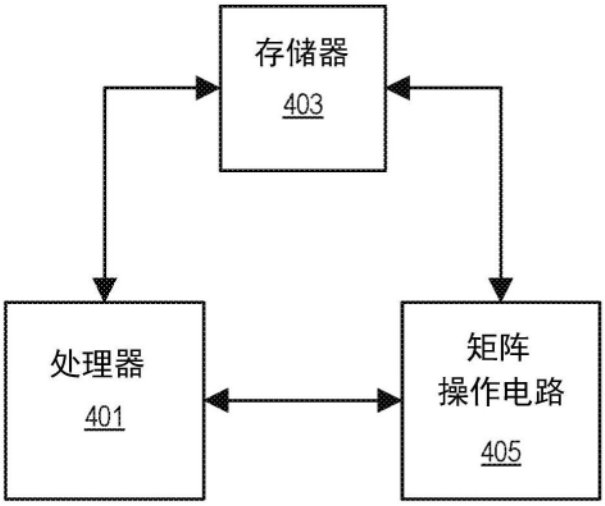

存储器
403
处理器
401
矩阵
操作电路
405

图4

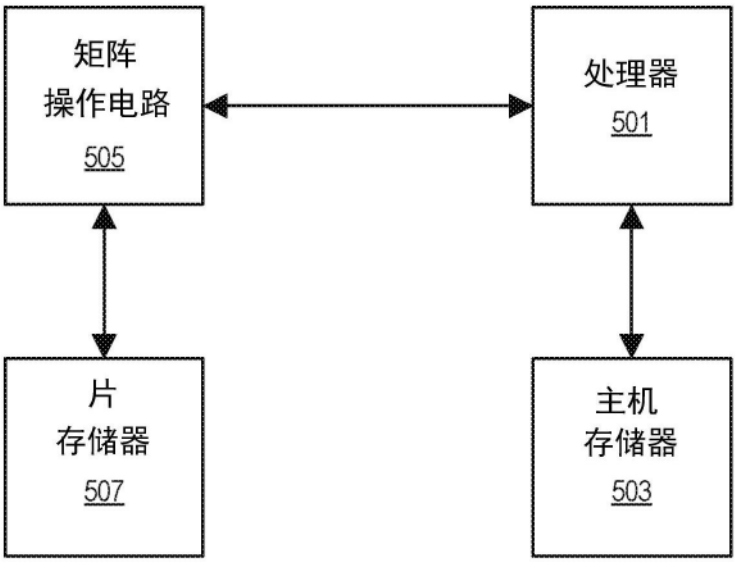

矩阵
操作电路
505
处理器
501
片
存储器
507
主机
存储器
503

图5

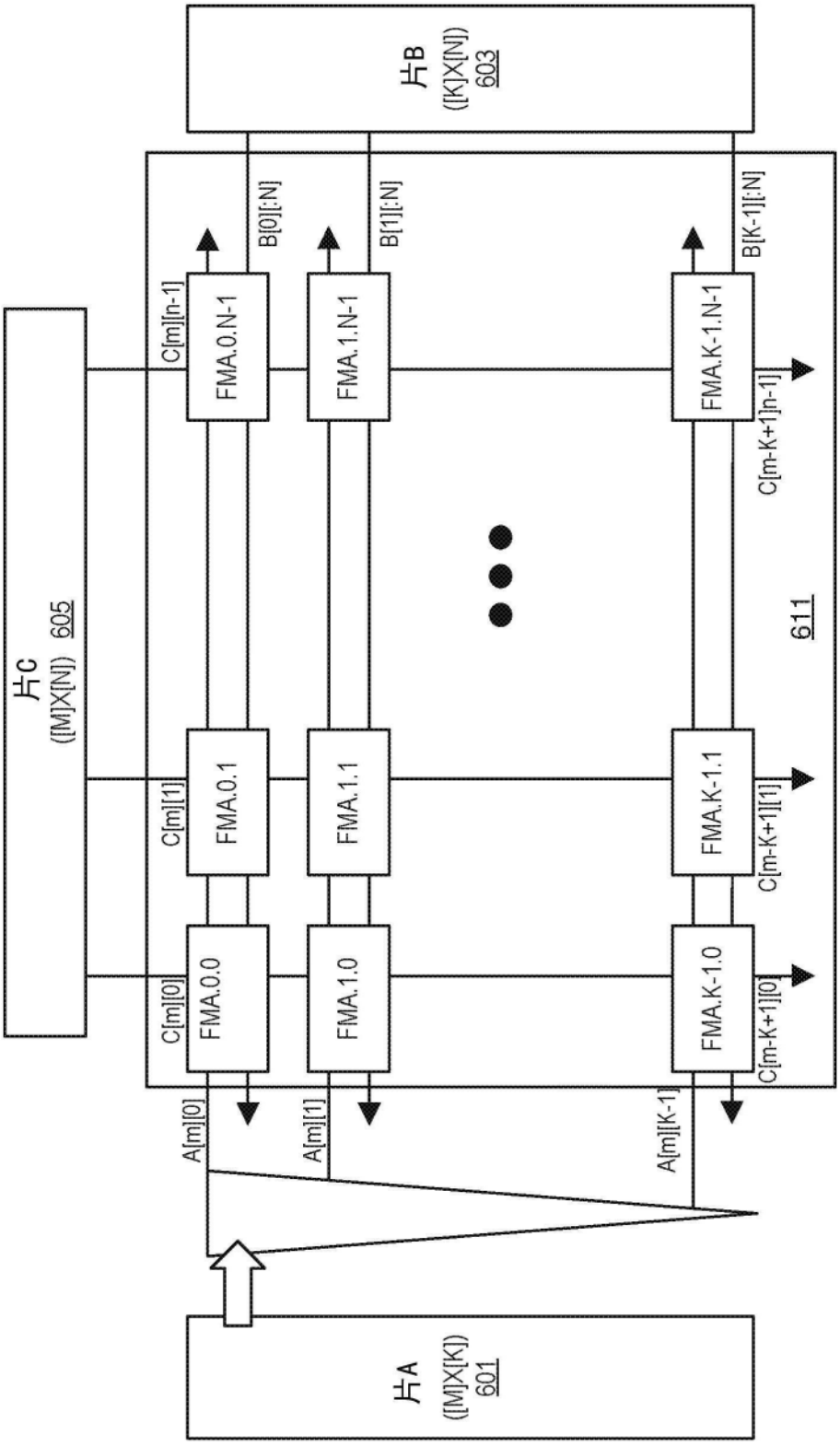

片B
([K]X[N])
603
B[0][:N]
B[1][:N]
B[K-1][:N]
C[m][n-1]
FMA.0.N-1
FMA.1.N-1
FMA.K-1.N-1
C[m-K+1][n-1]
片C
([M]X[N])
605
611
C[m][1]
FMA.0.1
FMA.1.1
FMA.K-1.1
C[m-K+1][1]
C[m][0]
FMA.0.0
FMA.1.0
FMA.K-1.0
C[m-K+1][0]
A[m][0]
A[m][1]
A[m][K-1]
片A
([M]X[K])
601

图6

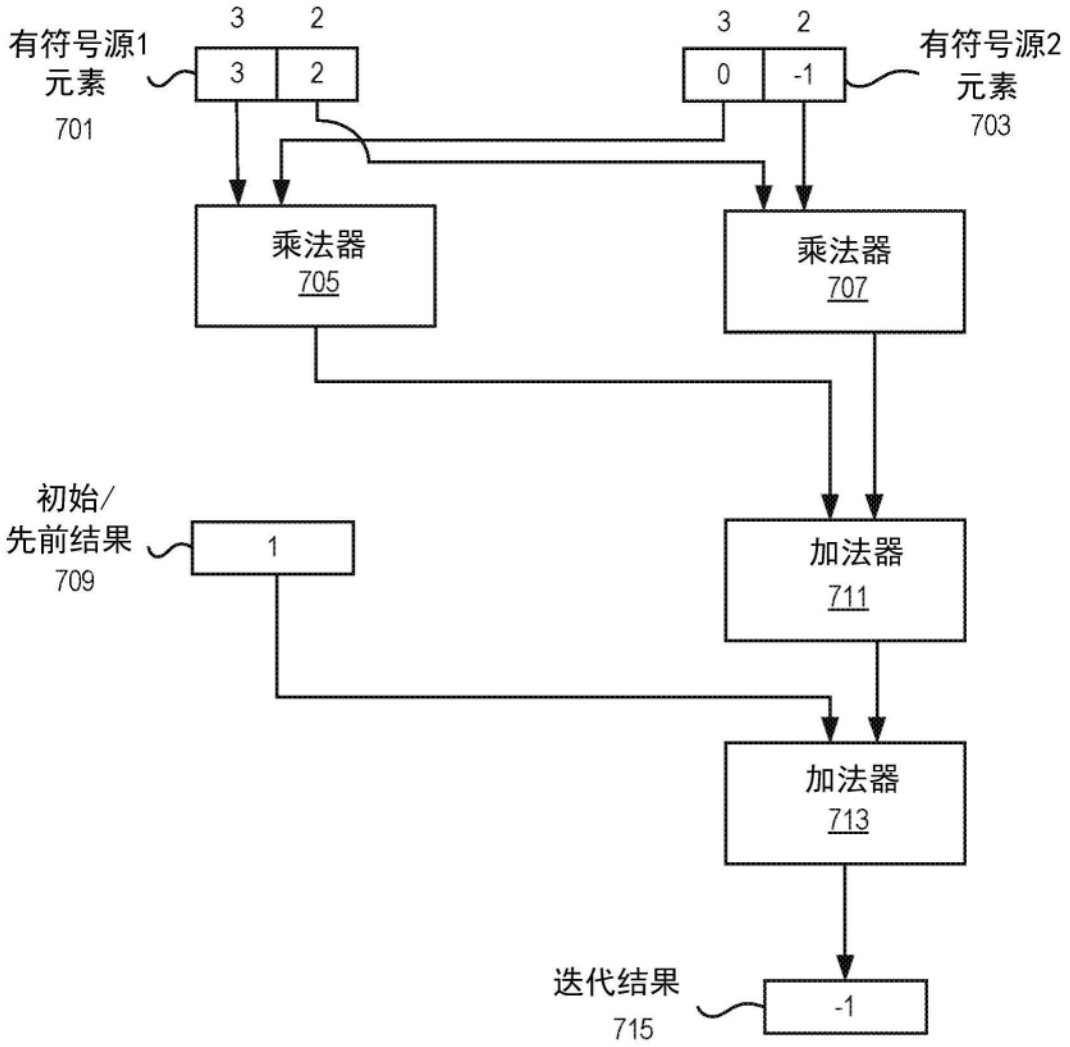

有符号源1
元素
701
3
2
3
2
有符号源2
元素
703
3
2
0
-1
乘法器
705
乘法器
707
初始/
先前结果
709
1
加法器
711
加法器
713
迭代结果
715
-1

图7

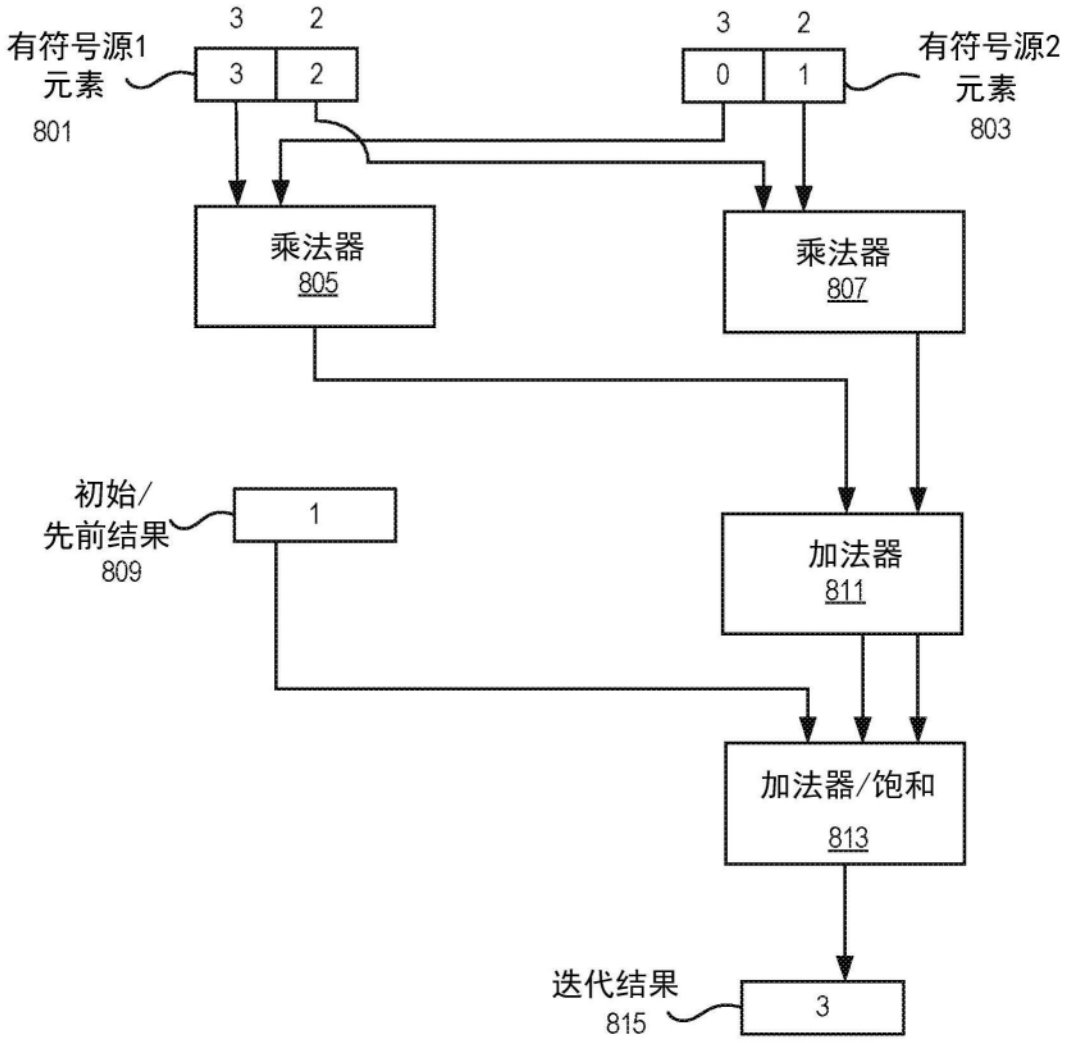

有符号源1
元素
801
3
2
3
2
有符号源2
元素
803
3
2
0
1
乘法器
805
乘法器
807
初始/
先前结果
809
1
加法器
811
加法器/饱和
813
迭代结果
815
3

图8

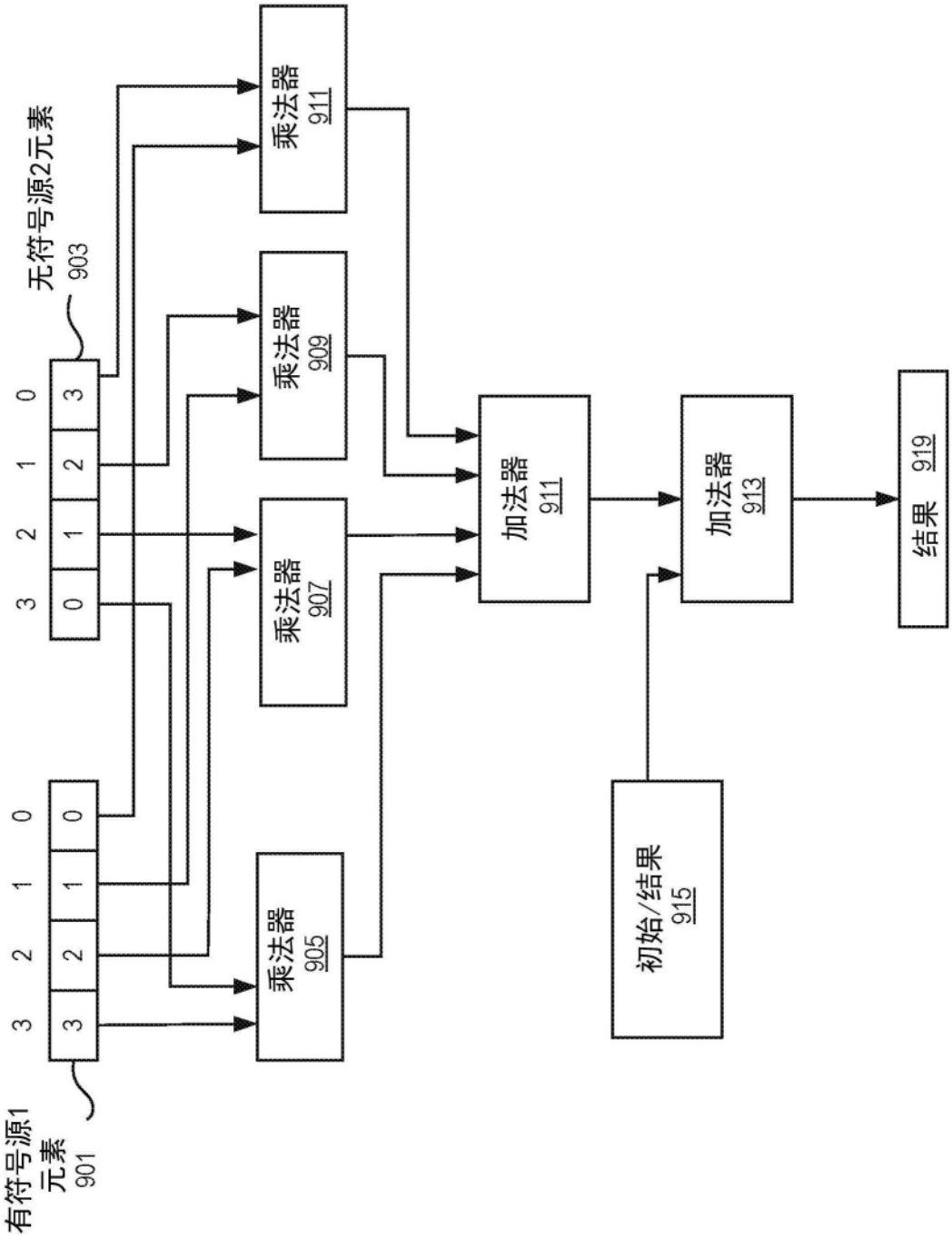

有符号源1元素 901
无符号源2元素 903
乘法器 905
乘法器 907
乘法器 909
乘法器 911
加法器 911
加法器 913
初始/结果 915
结果 919

图9

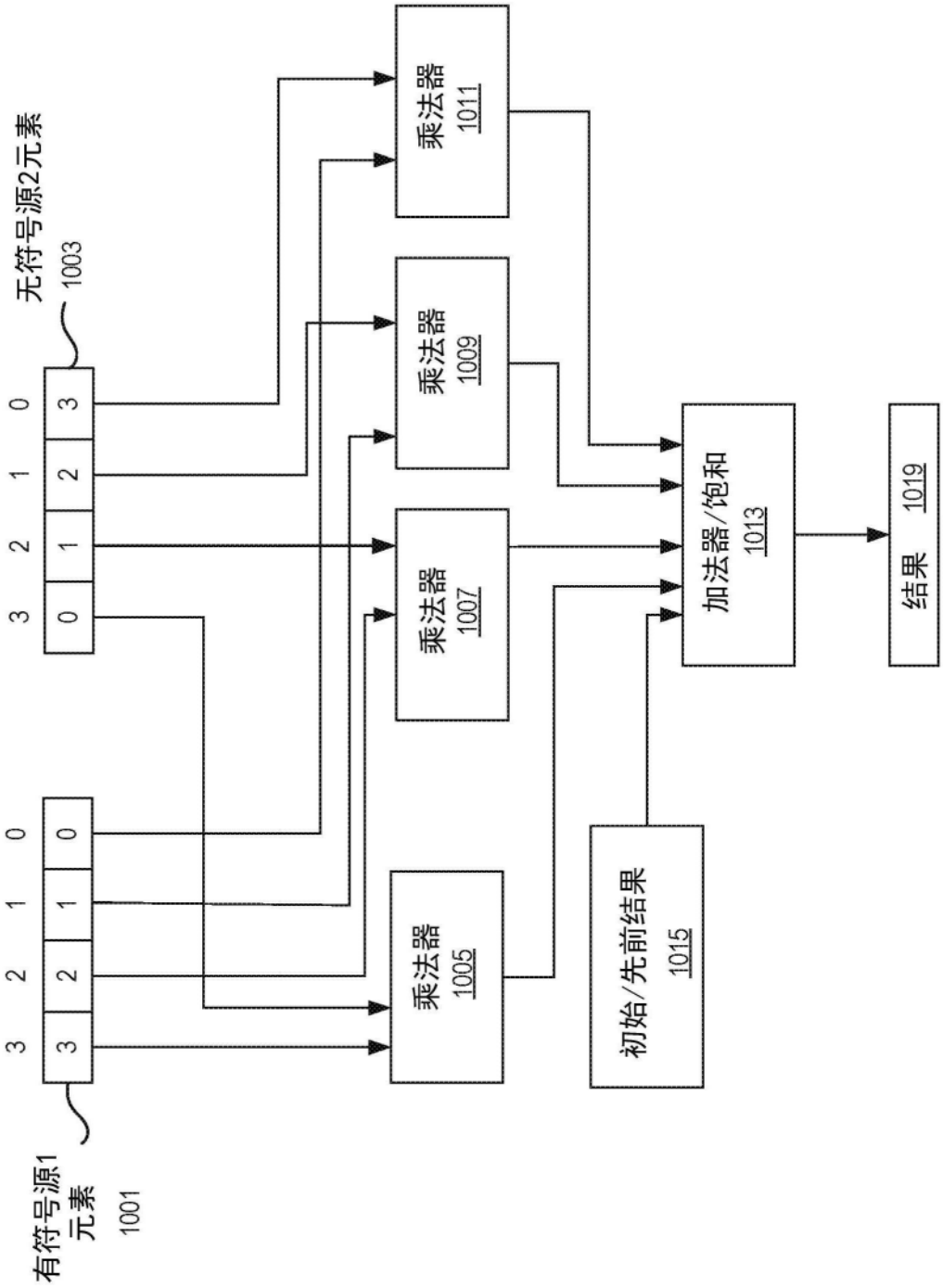

无符号源2元素
1003
3 2 1 0
0 1 2 3
有符号源1
元素
1001
3 2 1 0
3 2 1 0
乘法器
1005
乘法器
1007
乘法器
1009
乘法器
1011
初始/先前结果
1015
加法器/饱和
1013
结果 1019

图10

累加器2X输入尺寸 1101

源	位	累加器	位
字节	8	字/HPFP	16
字	16	整数32/SPFP	32
SPFP/整数32	32	整数64/DPFP	64

累加器4X输入尺寸 1103

源	位	累加器	位
字节	8	整数32/SPFP	32
字	16	整数64/DPFP	64

累加器8X输入尺寸 1105

源	位	累加器	位
字节	8	整数64/DPFP	64

图11

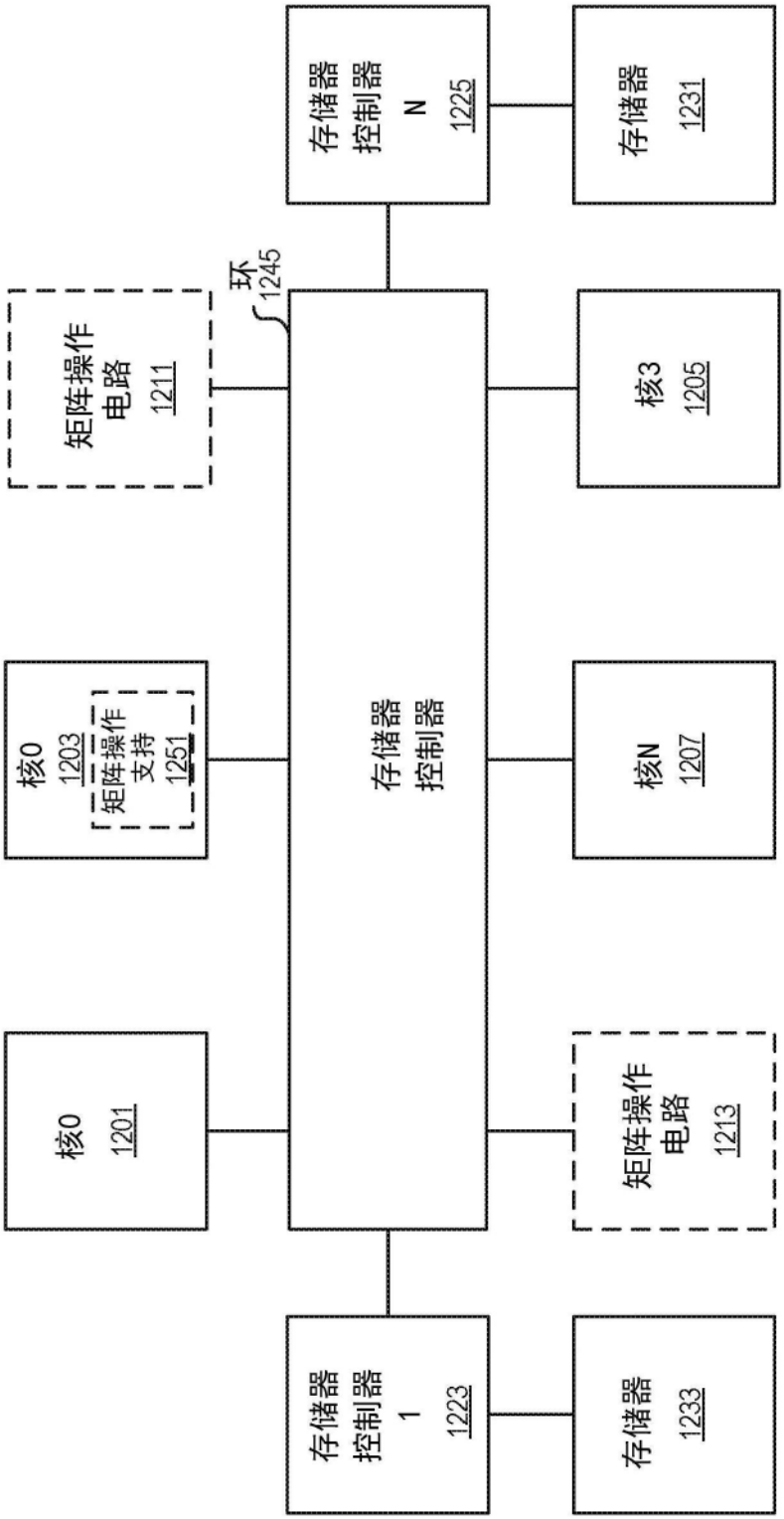

存储器控制器 N 1225
存储器 1231
环 1245
矩阵操作电路 1211
核3 1205
核0 1203
矩阵操作支持 1251
存储器控制器
核N 1207
核0 1201
矩阵操作电路 1213
存储器控制器 1 1223
存储器 1233

图12

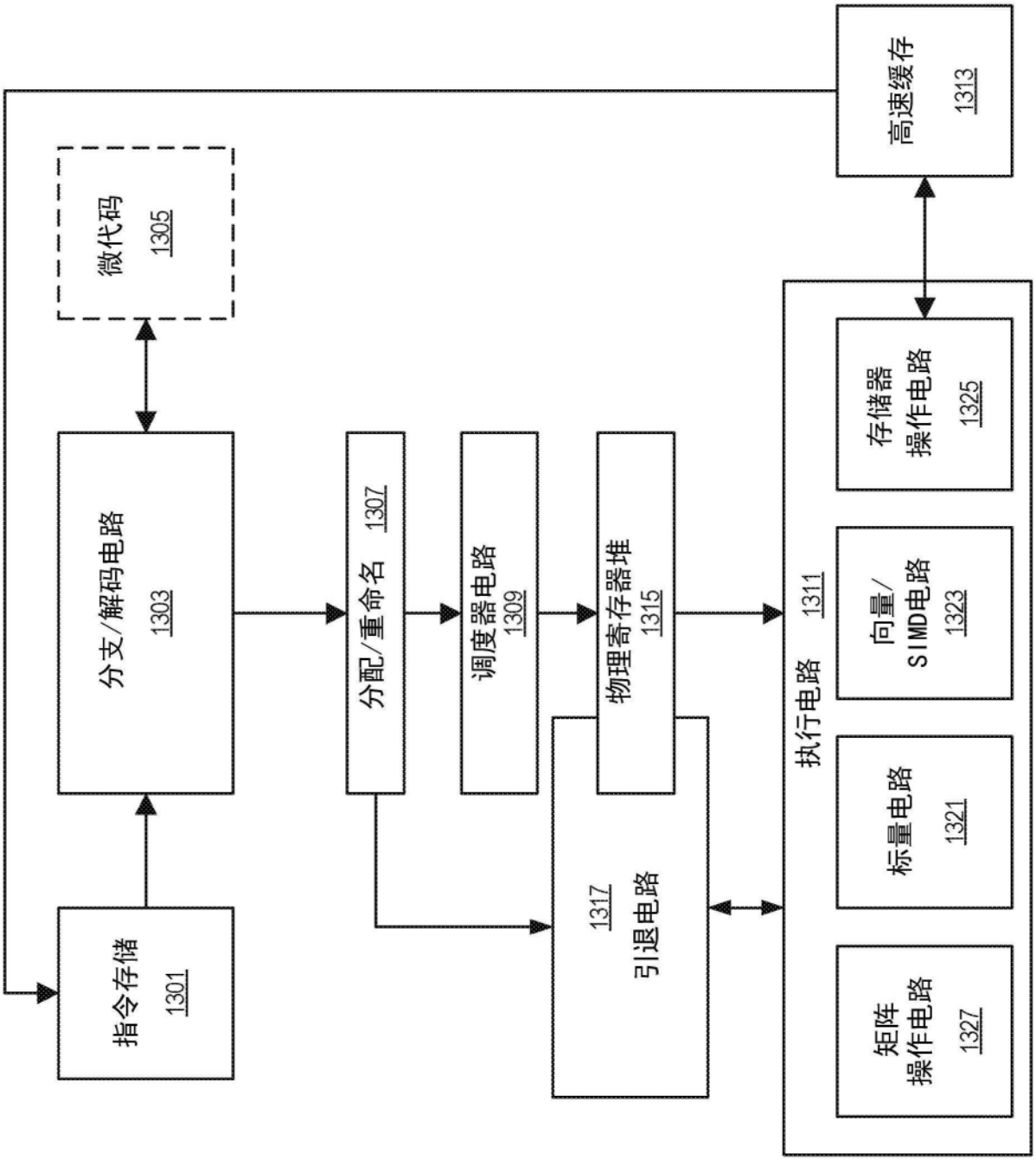

高速缓存 1313
微代码 1305
分支/解码电路 1303
分配/重命名 1307
调度器电路 1309
物理寄存器堆 1315
执行电路 1311
存储器操作电路 1325
向量/SIMD电路 1323
标量电路 1321
矩阵操作电路 1327
指令存储 1301
1317 引退电路

图13

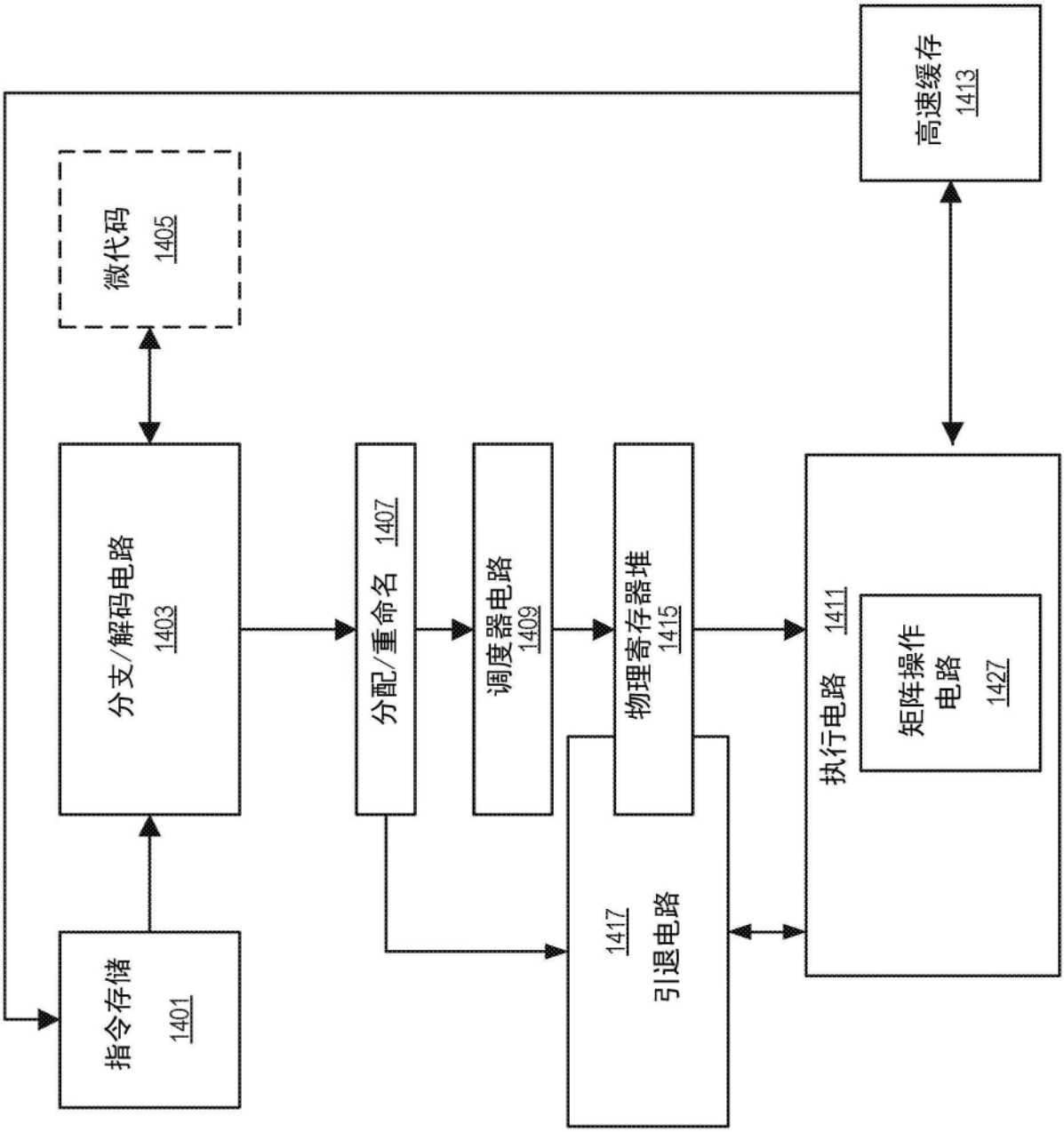

高速缓存
1413
微代码
1405
分支/解码电路
1403
分配/重命名
1407
调度器电路
1409
物理寄存器堆
1415
执行电路
1411
矩阵操作
电路
1427
指令存储
1401
1417
引退电路

图14

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}$$

地址	值
0	A_{11}
1	A_{12}
2	A_{13}
3	A_{21}
4	A_{22}
5	A_{23}

行为主

地址	值
0	A_{11}
1	A_{21}
2	A_{12}
3	A_{22}
4	A_{13}
5	A_{23}

列为主

图15

```
TILECONFIG [RAX]
//假定一些外循环驱动高速缓存分片（未示出）
{
TILELOAD TMM0, RSI+RDI  //SRCDST，RSI指向C，RDI具有
TILELOAD TMM1, RSI+RDI+N //C的第二片，在SIMD尺度N中展开
MOV KK, 0
LOOP:
TILELOAD TMM2, R8+R9 //SRC2是A的跨步式加载，重新用于2条TMMA指令。
TILELOAD TMM3, R10+R11//SRC1是B的跨步式加载
TMMAPS TMM0, TMM2, TMM3 //更新C的左片
TILELOAD TMM3, R10+R11+N //SRC1加载有来自下一最右片的B
TMMAPS TMM1, TMM2, TMM3 //更新C的右片
ADD R8, K //用循环外部已知的常数更新指针
ADD R10, K*R11
ADD KK, K
CMP KK, LIMIT
JNE LOOP
TILESTORE RSI+RDI, TMM0 //更新存储器中的C矩阵
TILESTORE RSI+RDI+M, TMM1
} //外循环的结尾
TILERELEASE //将片返回至初始状态
```

图16

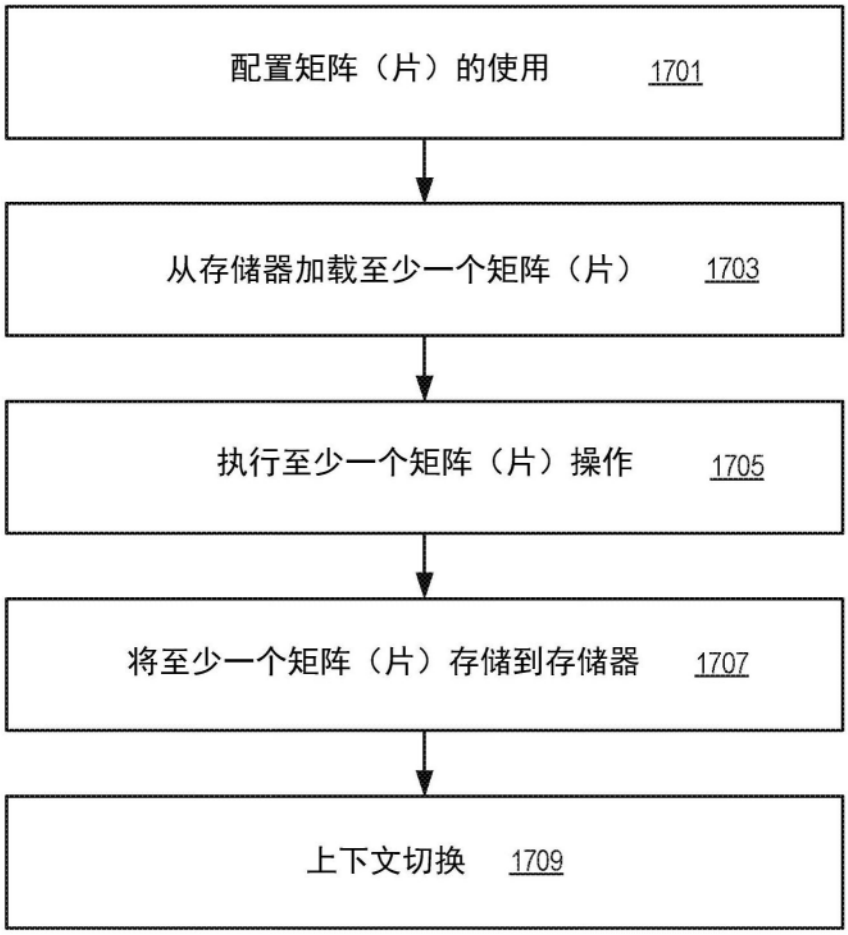

配置矩阵（片）的使用 1701
从存储器加载至少一个矩阵（片） 1703
执行至少一个矩阵（片）操作 1705
将至少一个矩阵（片）存储到存储器 1707
上下文切换 1709

图17

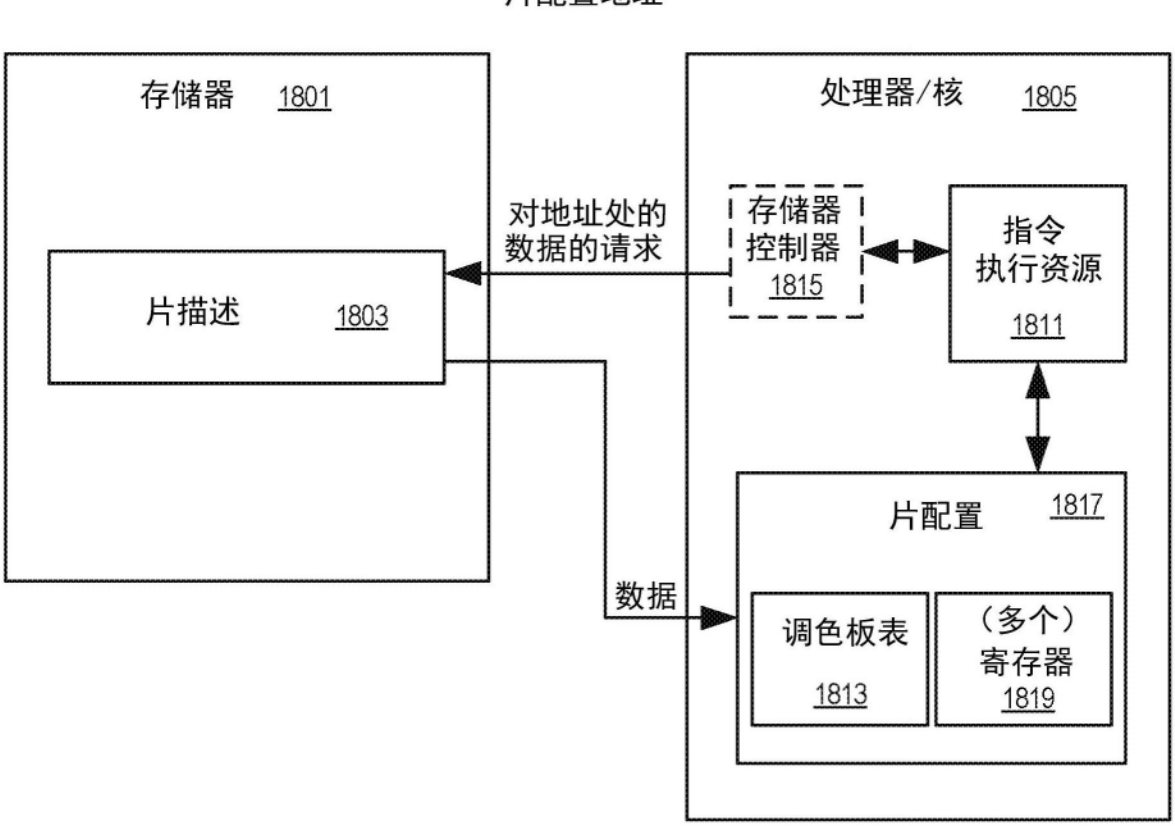

片配置地址
存储器 1801
片描述 1803
对地址处的数据的请求
处理器/核 1805
存储器控制器 1815
指令执行资源 1811
片配置 1817
数据
调色板表 1813
（多个）寄存器 1819

图18

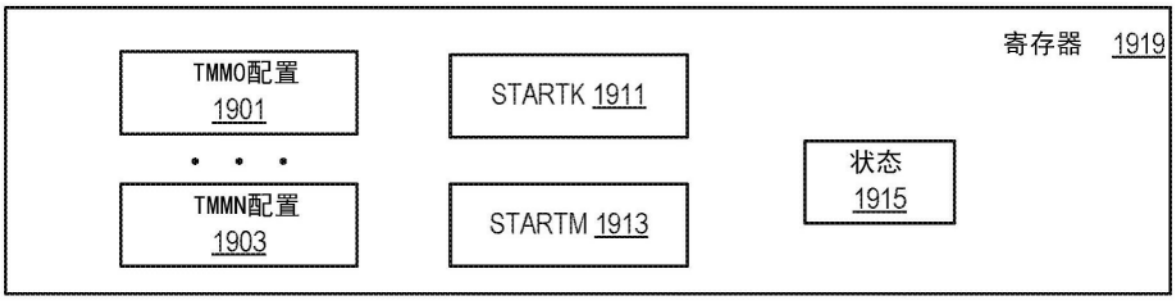

寄存器 1919
TMM0配置 1901
TMMN配置 1903
STARTK 1911
STARTM 1913
状态 1915

图19(A)

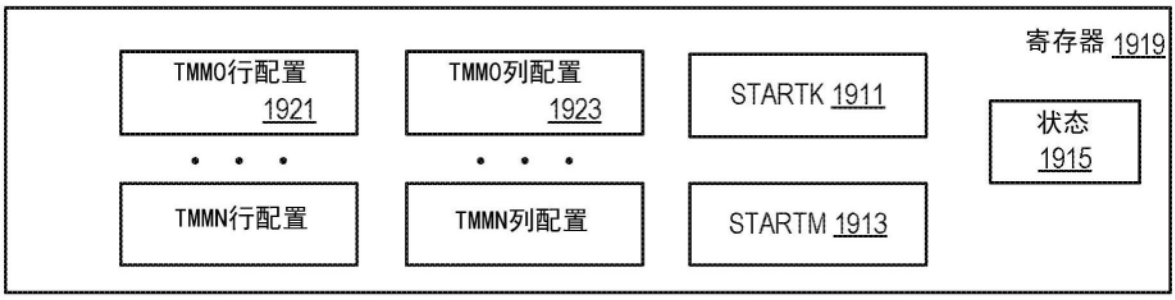


图19(B)

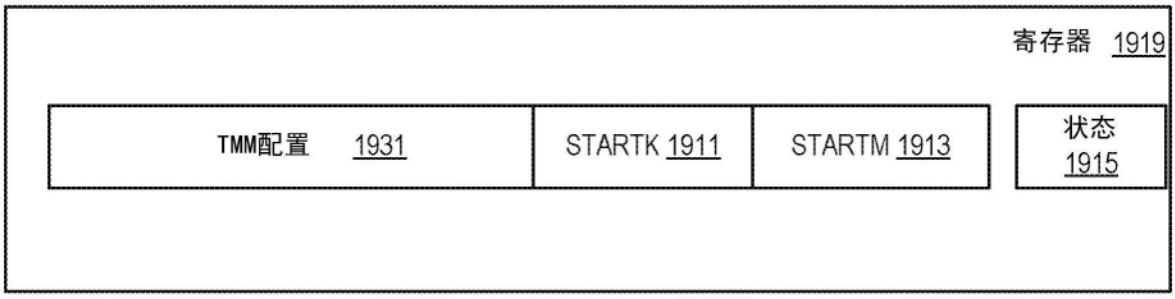


图19(C)

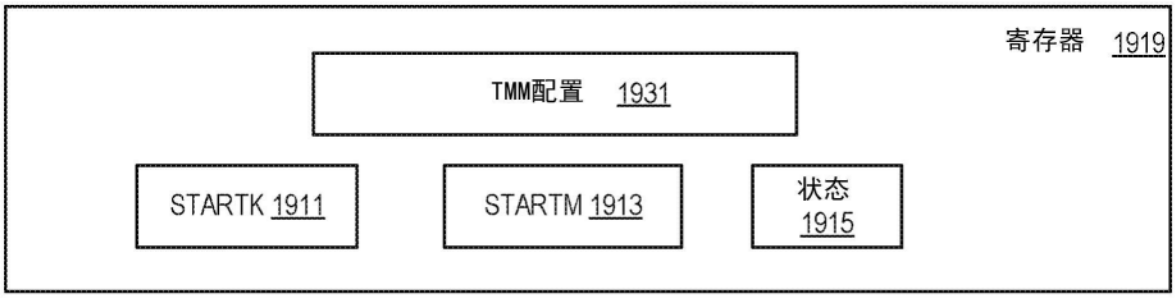


图19(D)

调色板ID 2001	0
0	0
0	0
0	0
STARTM 2003	
STARTK 2005	
0	0
TMM0行 2013	TMM0列 2015
TMM1行	TMM1列
● ● ●	
TMM15行	TMM15列
0	

图20

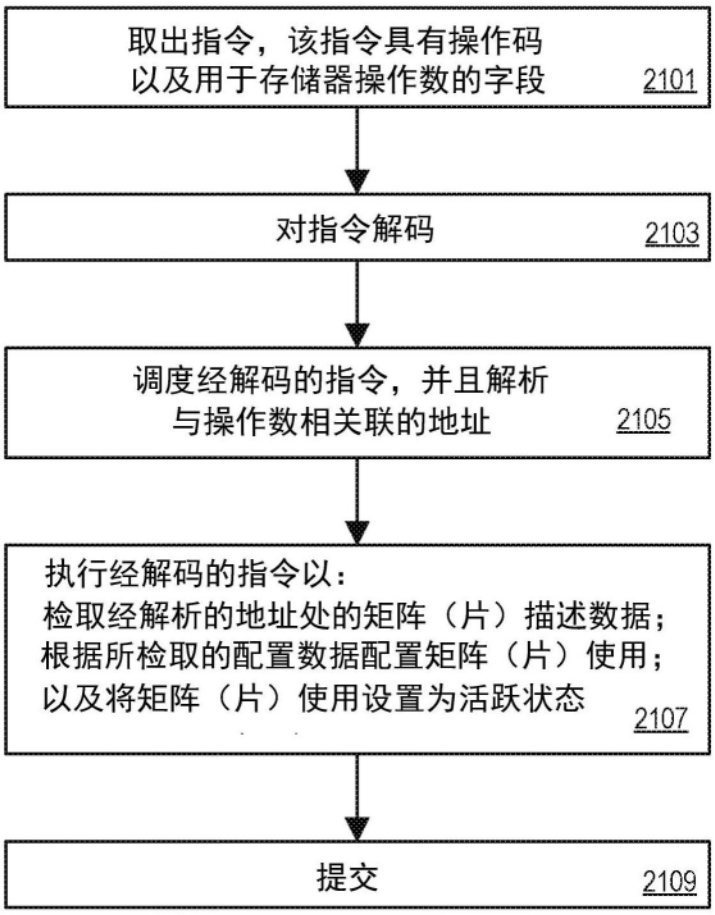

取出指令，该指令具有操作码
以及用于存储器操作数的字段
2101
对指令解码
2103
调度经解码的指令，并且解析
与操作数相关联的地址
2105
执行经解码的指令以：
检取经解析的地址处的矩阵（片）描述数据；
根据所检取的配置数据配置矩阵（片）使用；
以及将矩阵（片）使用设置为活跃状态
2107
提交
2109

图21

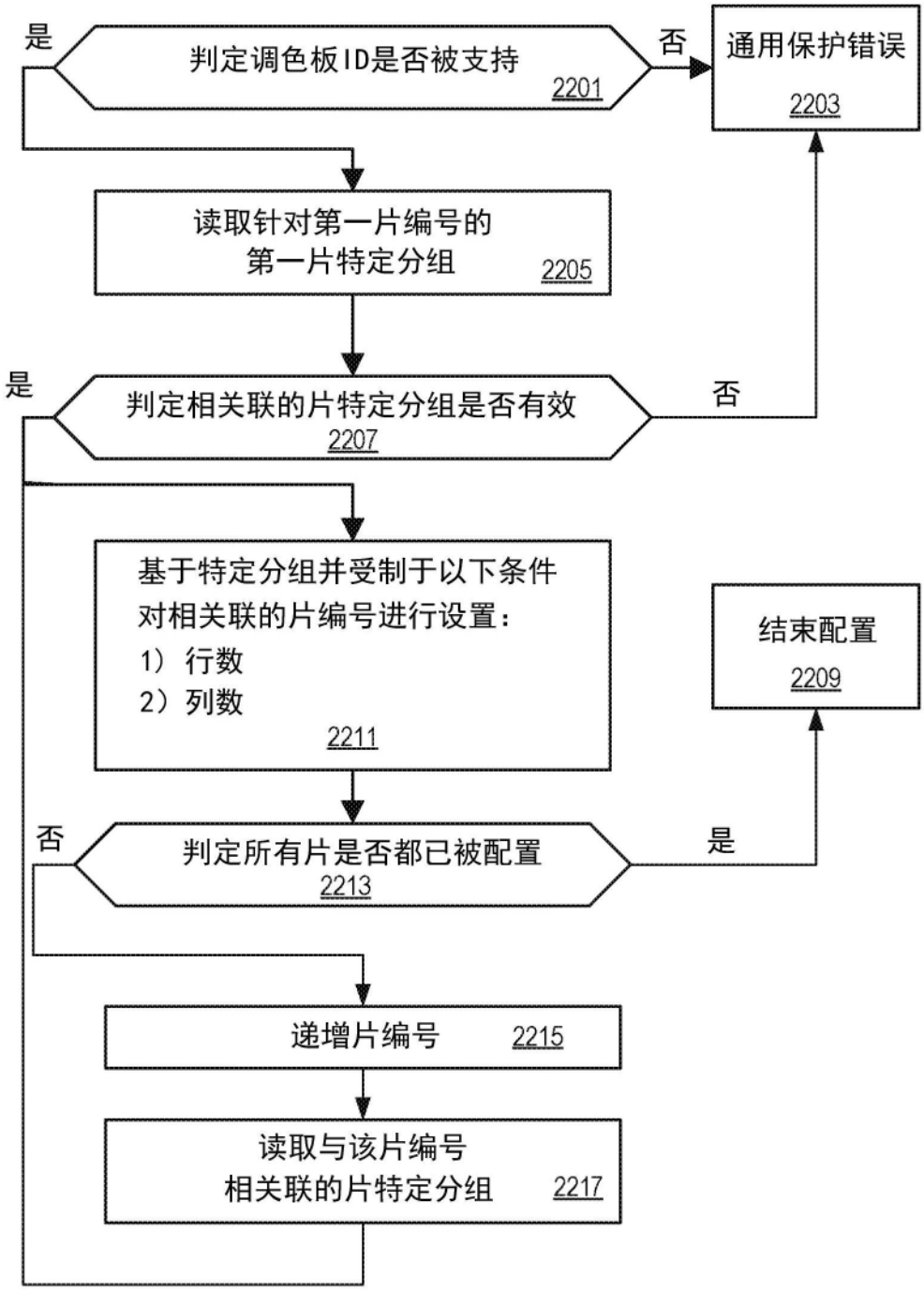

是
判定调色板ID是否被支持
2201
否
通用保护错误
2203
读取针对第一片编号的
第一片特定分组
2205
是
判定相关联的片特定分组是否有效
2207
否
基于特定分组并受制于以下条件
对相关联的片编号进行设置：
1）行数
2）列数
2211
结束配置
2209
否
判定所有片是否都已被配置
2213
是
递增片编号
2215
读取与该片编号
相关联的片特定分组
2217

图22

```
//存储器有效载荷的格式，每个字段是字节
//0：调色板_ID
//1-7：预留的（必须为零）
// 8-9: STARTM(16B)
// 10-11: STARTK(16B)
//12-15：预留的（必须为零）
//16-17：片0行片0列
//18-19：片1行片1列
//20-21：片2行片2列
// ...
//46-47：片15行片15列
//48-63：16B预留的（必须为零）
TILECONFIG MEM
PALETTE_ID := MEM.BYTE[0]
#GP IF PALETTE_ID IS AN UNSUPPORTED PALETTE //来自CPUID
#GP IF MEM.BYTE[1..7 ISNONZERO
STARTM := MEM.WORD[4] //字节8，9
STARTK := MEM.WORD[5] //字节10，11
#GP IF MEM.BYTE[12..15] ISNONZERO

MAX_NAMES := IMPL.MAX_TILE_BYTES /PALETTE_TABLE[I].TILE_BYTES
MAX_NUM_ROWS := PALETTE_TABLE[PALETTE_ID].TILE_BYTES/
PALETTE_TABLE[PALETTE_ID].BYTES_PER_ROW
P := 16
FOR N IN 0 ...MAX_NAMES-1:
T[N].ROWS :=MEM.BYTE[P++]
T[N].COLS :=MEM.BYTE[P++]
#GP IF T[N].ROWS ==0
#GP IF T[N].COLS ==0
#GP IF T[N].ROWS >MAX_NUM_ROWS
//所有指令基于它们的元素宽度来检查列限值，因此在此我们不将限制强加给列


WHILE P < 64: // 确认64B块中的剩余字节为零
#GP IF MEM.BYTE[P] !=0
P := P +1
FOR N IN 0 ...MAX_NAMES-1:
TILEZERO TILE[N]
TILES_CONFIGURED := 1
```

图23

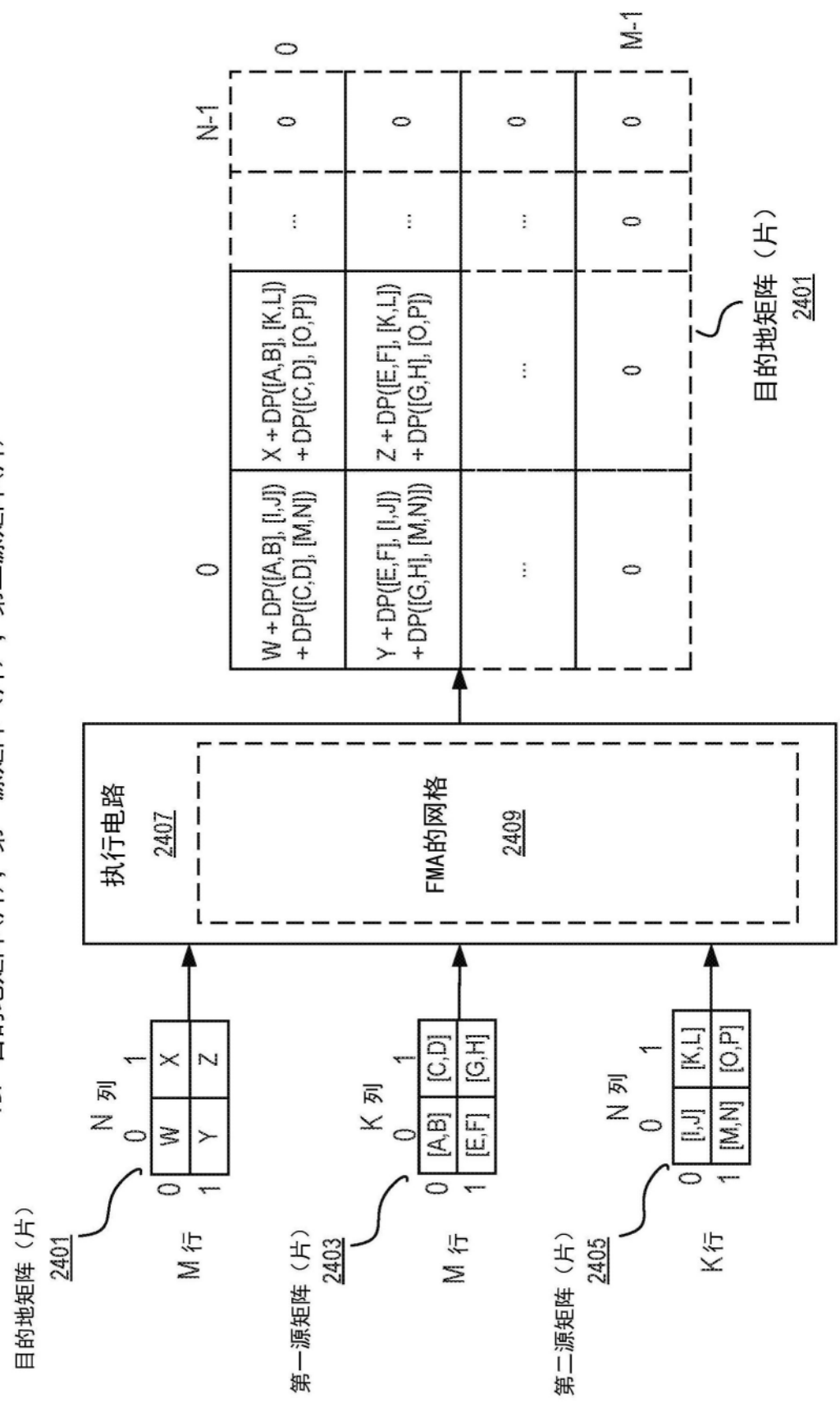

TDP 目的地矩阵(片)，第一源矩阵（片），第二源矩阵(片)
目的地矩阵（片）
2401
N 列
M 行
W
X
Y
Z
第一源矩阵（片）
2403
K 列
M 行
[A,B]
[C,D]
[E,F]
[G,H]
第二源矩阵（片）
2405
N 列
K行
[I,J]
[K,L]
[M,N]
[O,P]
执行电路
2407
FMA的网格
2409
W + DP([A,B], [I,J]) + DP([C,D], [M,N])
X + DP([A,B], [K,L]) + DP([C,D], [O,P])
Y + DP([E,F], [I,J]) + DP([G,H], [M,N]))
Z + DP([E,F], [K,L]) + DP([G,H], [O,P])
目的地矩阵（片）
2401

图24

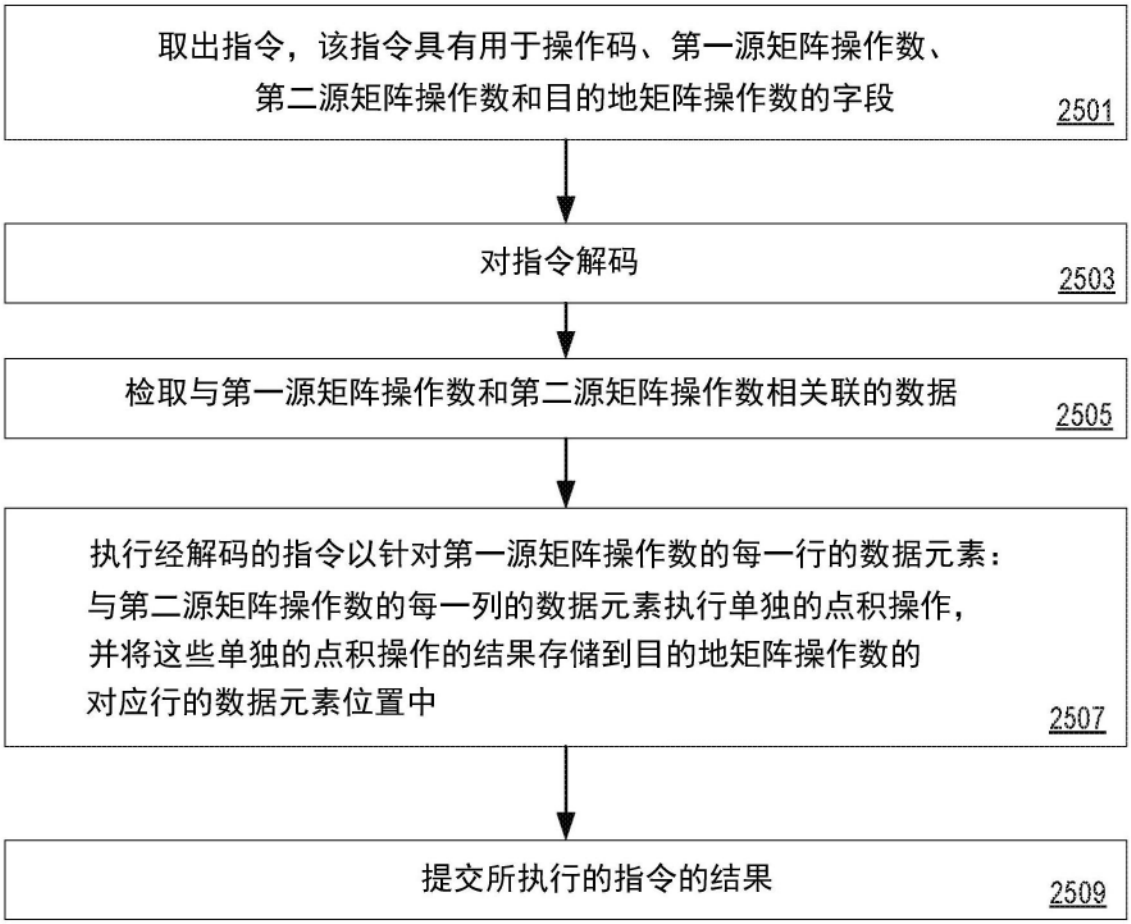

取出指令，该指令具有用于操作码、第一源矩阵操作数、
第二源矩阵操作数和目的地矩阵操作数的字段
2501
对指令解码
2503
检取与第一源矩阵操作数和第二源矩阵操作数相关联的数据
2505
执行经解码的指令以针对第一源矩阵操作数的每一行的数据元素：
与第二源矩阵操作数的每一列的数据元素执行单独的点积操作，
并将这些单独的点积操作的结果存储到目的地矩阵操作数的
对应行的数据元素位置中
2507
提交所执行的指令的结果
2509

图25

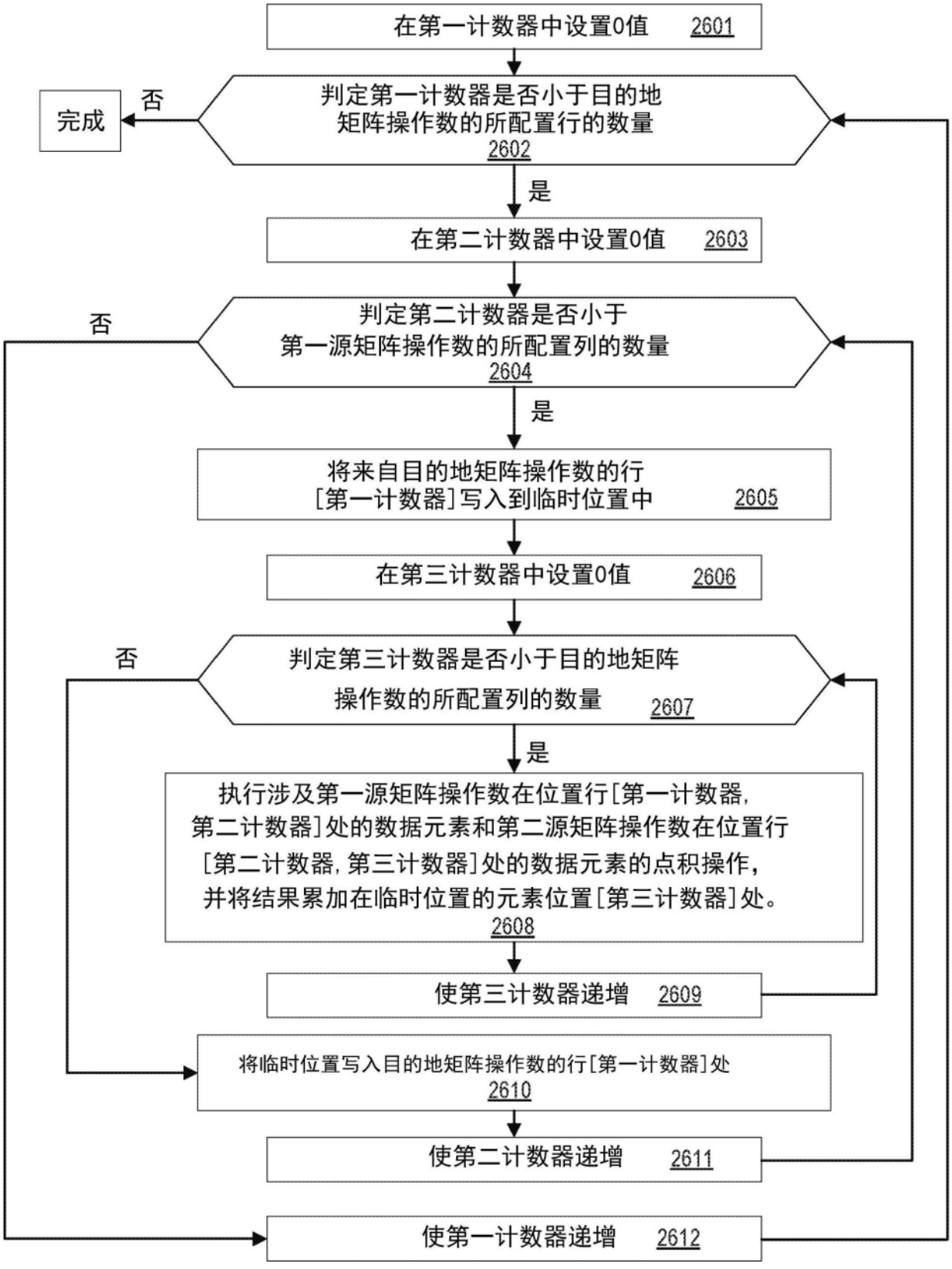

在第一计数器中设置0值 2601
判定第一计数器是否小于目的地矩阵操作数的所配置行的数量 2602
否
完成
是
在第二计数器中设置0值 2603
判定第二计数器是否小于第一源矩阵操作数的所配置列的数量 2604
否
是
将来自目的地矩阵操作数的行[第一计数器]写入到临时位置中 2605
在第三计数器中设置0值 2606
判定第三计数器是否小于目的地矩阵操作数的所配置列的数量 2607
否
是
执行涉及第一源矩阵操作数在位置行[第一计数器,第二计数器]处的数据元素和第二源矩阵操作数在位置行[第二计数器,第三计数器]处的数据元素的点积操作,并将结果累加在临时位置的元素位置[第三计数器]处。 2608
使第三计数器递增 2609
将临时位置写入目的地矩阵操作数的行[第一计数器]处 2610
使第二计数器递增 2611
使第一计数器递增 2612

图26

2701 TDPWSSDS点积指令助手函数

```
define DPWSS(c,x,y): //自变量是双字
     p1dword := SIGN_EXTEND(x.word[0]) * SIGN_EXTEND(y.word[0])
     p2dword := SIGN_EXTEND(x.word[1]) * SIGN_EXTEND(y.word[1])

     c := SIGNED_DWORD_SATURATE(c + p1dword + p2dword)
```

2703 TDPWSSDS点积操作

```
TDPWSSDS tsrcdest, tsrc1, tsrc2
// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)
#GP if TILES_CONFIGURED == 0
#GP if tsrc1.cols != tsrc2.rows
#GP if tsrcdest.rows != tsrc1.rows
#GP if tsrcdest.cols != tsrc2.cols

#GP if tsrc2.rows > impl.tmul_maxk
#GP if tsrc2.cols * 4 > impl.tmul_maxn
#GP if tsrcdest.cols * 4 > impl.tmul_maxn

for m in 0 ... tsrcdest.rows-1:
     for k in 0 ... tsrc1.cols-1:
          tmp := tsrcdest.row[m]
          for n in 0 ... tsrcdest.cols-1:
               DPWSS(tmp.dword[n],
                    tsrc1.row[m].dword[k],
                    tsrc2.row[k].dword[n])
          write_row_and_zero(tsrcdest, m, tmp, 4 * tsrcdest.cols)

zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tileconfig_start()
```

图27A

2705 TDPWSSDS点积指令助手函数

```
define DPWSSQ(c,x,y): //c是四字。x和y是包含4个int16的四字
      p1dword := SIGN_EXTEND(x.word[0]) * SIGN_EXTEND(y.word[0])
      p2dword := SIGN_EXTEND(x.word[1]) * SIGN_EXTEND(y.word[1])
      p3dword := SIGN_EXTEND(x.word[2]) * SIGN_EXTEND(y.word[2])
      p4dword := SIGN_EXTEND(x.word[3]) * SIGN_EXTEND(y.word[3])

      c := SIGNED_QWORD_SATURATE(c + p1dword + p2dword + p3dword + p4dword)
```

2707 TDPWSSDS点积操作

```
TDPWSSQS tsrcdest, tsrc1, tsrc2
// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)
// 片A是由tsrc1指示的片寄存器的偶/奇对。
// tiles[.]符号指示表示所有片寄存器的数组。


#GP if TILES_CONFIGURED == 0
 //reg_id_tsrc1是用于tsrc1操作数的编码中的寄存器字段。
t1_left := (reg_id_tsrc1 & ~1)
t1_right := (reg_id_tsrc1 & ~1) + 1

startK := tileconfig.startK
startM := tileconfig.startM

#GP if tsrcdest.rows != tiles[t1_left].rows
#GP if tsrcdest.rows != tiles[t1_right].rows
#GP if tsrcdest.cols != tsrc2.cols

#GP if startM >= tsrcdest.rows
#GP if startK >= tsrc2.rows

#GP if tsrc2.rows        > impl.tmul_maxk
#GP if tsrc2.cols * 8    > impl.tmul_maxn
#GP if tsrcdest.cols * 8 > impl.tmul_maxn

if tsrc2.rows > impl.tmul_maxk / 2:
      lim_left := impl.tmul_maxk/2
      lim_right := tsrc2.rows - lim_left
      #GP if tiles[t1_left].cols != lim_left
      #GP if tiles[t1_right].cols != lim_right
else:
      lim_left := tsrc2.rows
      lim_right := 0 // 如果不需要则忽略t1_right!
      #GP if tiles[t1_left].cols != lim_left
```

（在图27C继续）

图27B

2707 TDPWSSDS点积操作（续）

```
while startM < tsrcdest.rows:
      while startK < tsrc2.rows:
            tempC := tsrcdest.row[startM]

            tk := startK % (impl.tmul_maxk/2)
            if startK < impl.tmul_maxk/2:
                  tempA := tiles[t1_left].row[m].qword[tk]
            else:
                  tempA := tiles[t1_right].row[m].qword[tk]

            for n := 0 ... tsrcdest.cols-1: //SIMD循环
                  DPWSSQ(tempC.qword[n],
                        tempA,
                        tsrc2.row[startK].qword[n])
            write_row_and_zero(tsrcdest, startM, tempC, 8 * tsrcdest.cols)
            startK := startK + 1
      startK := 0
      startM := startM + 1

zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tileconfig_start()
```

图27C

2709 TDPB[SS,UU,US,SU]DS点积操作助手函数

```
define DPBD(c,x,y): //自变量是双字
      if *UU version*:
            saturation_fn := UNSIGNED_SATURATION
      else:
            saturation_fn := SIGNED_SATURATION

      if *x operand is signed*:
            extend_src1 := SIGN_EXTEND
      else:
            extend_src1 := ZERO_EXTEND

      if *y operand is signed*:
            extend_src2 := SIGN_EXTEND
      else:
            extend_src2 := ZERO_EXTEND

      p0word := extend_src1(x.byte[0]) * extend_src2(y.byte[0])
      p1word := extend_src1(x.byte[1]) * extend_src2(y.byte[1])
      p2word := extend_src1(x.byte[2]) * extend_src2(y.byte[2])
      p3word := extend_src1(x.byte[3]) * extend_src2(y.byte[3])

      c := saturation_fn(c + p0word + p1word + p2word + p3word)
```

2711 TDPB[SS,UU,US,SU]DS点积操作

```
TDPB[SS,SU,US,SS]DS tsrcdest, tsrc1, tsrc2
// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)
#GP if TILES_CONFIGURED == 0
#GP if tsrc1.cols != tsrc2.rows
#GP if tsrcdest.rows != tsrc1.rows
#GP if tsrcdest.cols != tsrc2.cols

#GP if tsrc2.rows > impl.tmul_maxk
#GP if tsrc2.cols * 4 > impl.tmul_maxn
#GP if tsrcdest.cols * 4 > impl.tmul_maxn

for m in 0 ... tsrcdest.rows-1:
      for k in 0 ... tsrc1.cols-1:
            tmp := tsrcdest.row[m]
            for n in 0 ... tsrcdest.cols-1:
                  DPBD(tmp.dword[n],
                        tsrc1.row[m].dword[k],
                        tsrc2.row[k].dword[n])
            write_row_and_zero(tsrcdest, m, tmp, 4 * tsrcdest.cols)

zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tileconfig_start()
```

图27D

2713 TDP8B[SS,UU,US,SU]4BITDS点积操作

```
TDP8B[SS,SU,US,UU]4BITDS tsrcdest, tsrc1, tsrc2
 //tsrdest (m×n):=tilepair(m×k, 字节)*tsrc2 (k×n, 4b值)
// 片对是由tsrc1指示的片寄存器的偶/奇对。
// tiles[.]符号指示表示所有片寄存器的数组。

#GP if TILES_CONFIGURED == 0
//reg_id_tsrc1是用于tsrc1操作数的编码中的寄存器字段。
t1_left := (reg_id_tsrc1 & ~1)
t1_right := (reg_id_tsrc1 & ~1) + 1

#GP if tsrcdest.rows != tiles[t1_left].rows
#GP if tsrcdest.rows != tiles[t1_right].rows
#GP if tsrcdest.cols != tsrc2.cols

#GP if tsrc2.rows        > impl.tmul_maxk
#GP if tsrc2.cols * 4    > impl.tmul_maxn
#GP if tsrcdest.cols * 4 > impl.tmul_maxn

if tsrc2.rows > impl.tmul_maxk / 2:
      lim_left := impl.tmul_maxk/2
      lim_right := tsrc2.rows - lim_left
      #GP if tiles[t1_left].cols != lim_left
      #GP if tiles[t1_right].cols != lim_right
else:
      lim_left := tsrc2.rows
      lim_right := 0 //如果不需要则忽略t1_right!
      #GP if tiles[t1_left].cols != lim_left

if *tsrc2 operand is signed*:
      extend_src2 := SIGN_EXTEND
else:
      extend_src2 := ZERO_EXTEND

if *tsrc1 operand is signed*:
      extend_src1 := SIGN_EXTEND
else:
      extend_src1 := ZERO_EXTEND
```

（在图27F继续）

图27E

2713 TDP8B[SS,UU,US,SU]4BITDS点积操作（续）

```
if *tsrc1 is unsigned and tsrc2 is unsigned*:
      saturation_fn := UNSIGNED_SATURATION
else:
      saturation_fn := SIGNED_SATURATION

for m in 0..tsrcdest.rows-1:
      for k in 0..tsrc2.rows-1:
            tk := k % (impl.tmul_maxk/2)
            if k < impl.tmul_maxk/2:
                  tq := tiles[t1_left].row[m].qword[tk]
            else:
                  tq := tiles[t1_right].row[m].qword[tk]
            for n in 0..tsrcdest.cols-1:
                  for b in 0..7: //4位数组索引和tq字节索引
                        x := tsrc2[k].dword[n].nibble[b] //提取4位
                        tprod.dword[b] := extend_src2(x) * extend_src1(tq.byte[b])
                  //9路和
                  tsrcdest.row[m].dword[n] := saturation_fn(tsrcdst.row[m].dword[n] +
                                              tprod.dword[0] +
                                              tprod.dword[1] +
                                              tprod.dword[2] +
                                              ...
                                              tprod.dword[7] )
zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tileconfig_start()
```

图27F

2715 TDP4BIT[S,U][S,U]DS点积操作

```
TDP4BIT[SS,SU,US,UU]DS tsrcdest, tsrc1, tsrc2
//tsrdest (m×n)+=tsrc1(m×k, 字节) op tsrc2 (k×n, 4b值)

#GP if TILES_CONFIGURED == 0
#GP if tsrcdest.cols != tsrc2.cols
#GP if tsrc2.rows > impl.tmul_maxk
#GP if tsrc2.cols * 4 > impl.tmul_maxn
#GP if tsrcdest.cols * 4 > impl.tmul_maxn

startK := tileconfig.startK
startM := tileconfig.startM
#GP if startM >= tsrcdest.rows
#GP if startK >= tsrc2.rows

if *src2 operand is signed*:
      extend_src2 := SIGN_EXTEND
else:
      extend_src2 := ZERO_EXTEND

if *tmem operand is signed*:
      extend_mem := SIGN_EXTEND
else:
      extend_mem := ZERO_EXTEND

if *src1 is unsigned and src2 is unsigned*:
      saturation_fn := UNSIGNED_SATURATION
else:
      saturation_fn := SIGNED_SATURATION

while startM < tsrcdest.rows:
      while startK < tsrc2.rows:
            td := tsrc1.row[startM].dword[startK]
            for n in 0..tsrcdest.cols-1:
                  for b in 0..7:
                        x := tsrc2.row[startK].dword[n].nibble[b]
                        tprod.dword[b] := extend_src2(x) * extend_mem(td.nibble[b])
                  //9路和
                  tsrcdest.row[startM].dword[n] :=
                              saturation_fn(tsrcdst.row[startM].dword[n] +
                              tprod.dword[0] +
                              tprod.dword[1] +
                              tprod.dword[2] +
                              ...
                              tprod.dword[7])
      startK := 0
      startM := startM + 1
zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tileconfig_start()
```

图27G

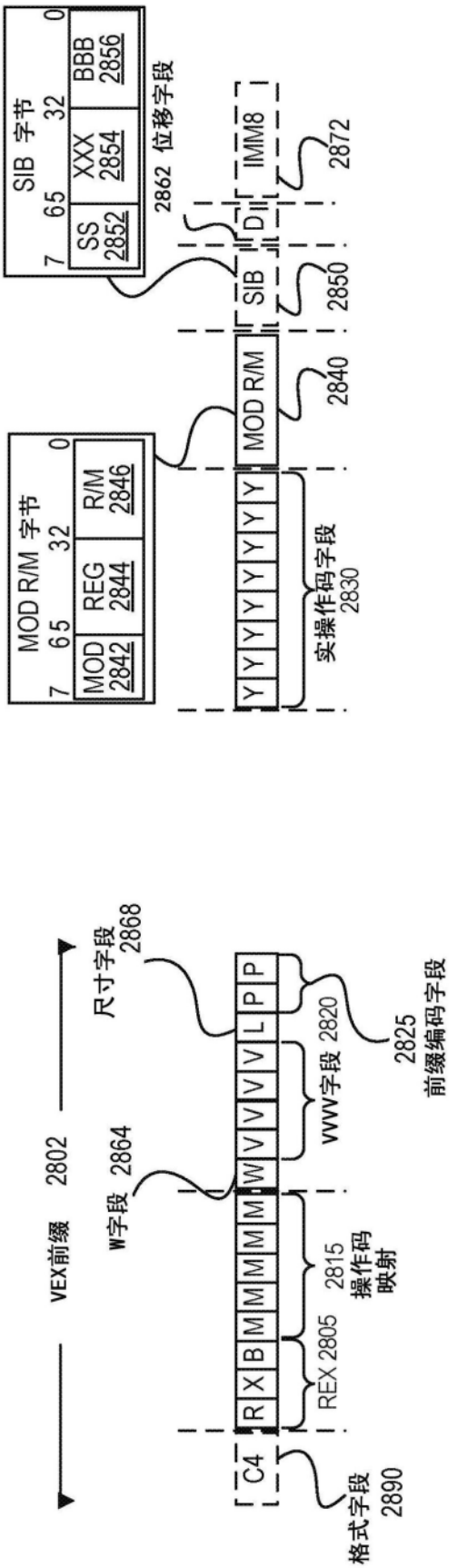

SIB 字节
7 6 5 3 2 0
SS 2852
XXX 2854
BBB 2856
2862 位移字段
MOD R/M 字节
MOD 2842
REG 2844
R/M 2846
MOD R/M
SIB
DISP
IMM8
2872
2850
2840
实操作码字段
2830
Y Y Y Y Y Y Y Y
VEX前缀 2802
尺寸字段 2868
W字段 2864
VVV字段 2820
2825
前缀编码字段
C4
R X B M M M M W V V V V L P P
REX 2805
2815
操作码映射
格式字段
2890

图28A

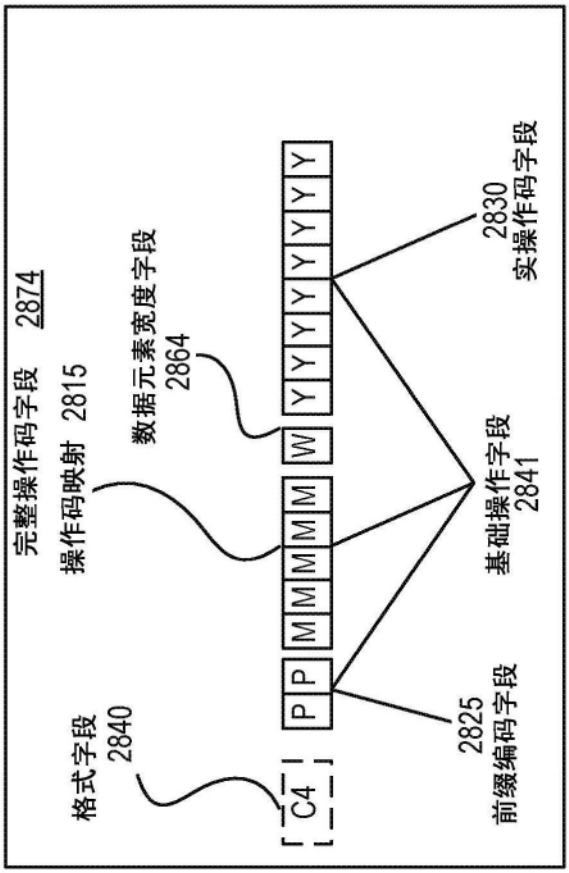

完整操作码字段 2874
操作码映射 2815
数据元素宽度字段 2864
格式字段 2840
C4
P P M M M M W Y Y Y Y Y Y Y
2825 前缀编码字段
基础操作字段 2841
2830 实操作码字段

图28B

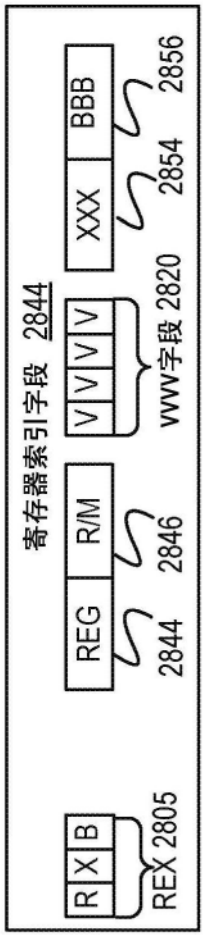

寄存器索引字段 2844
R X B
REX 2805
REG
R/M
2844
2846
V V V V
vvv字段 2820
XXX
BBB
2854
2856

图28C

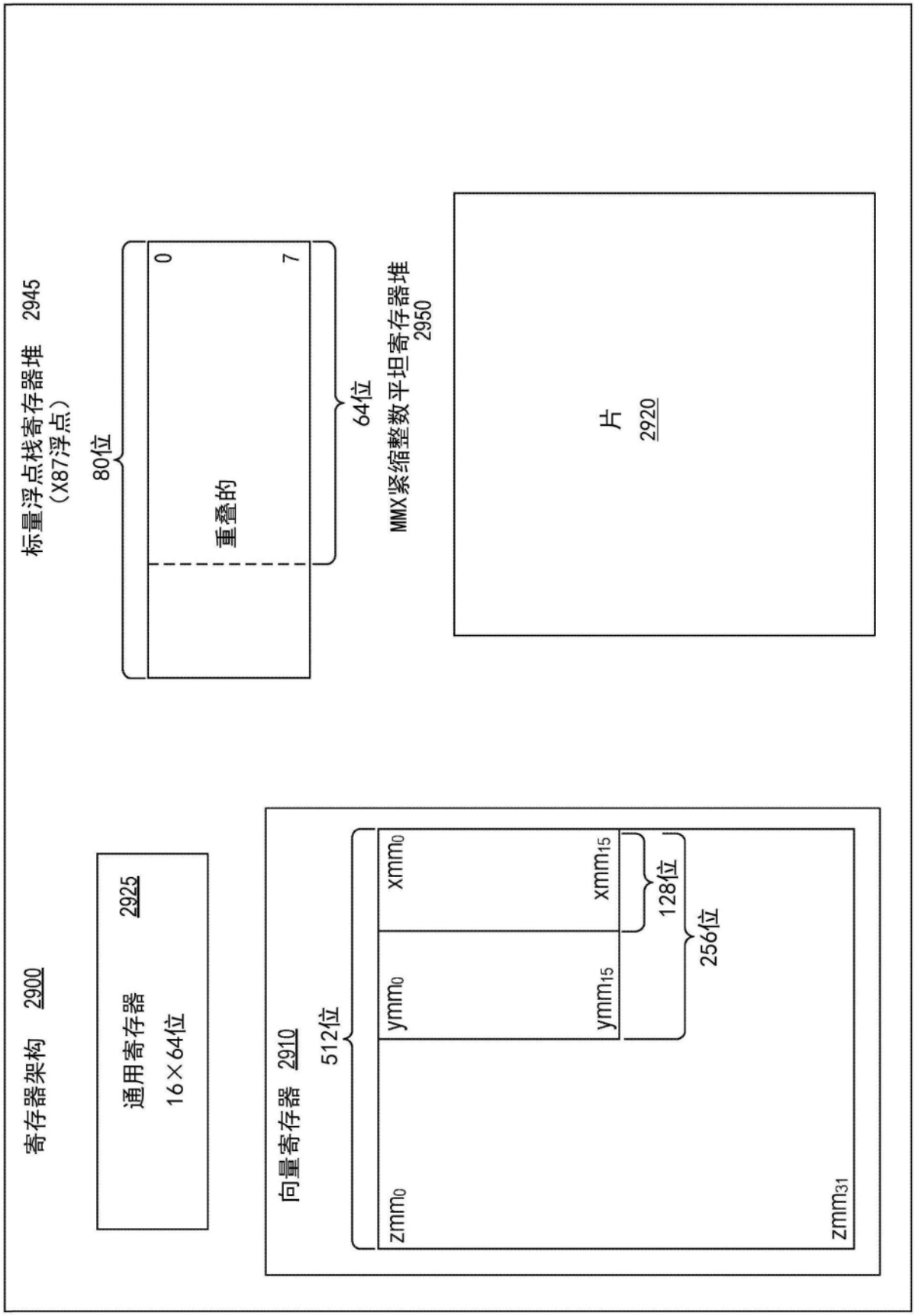

寄存器架构 2900
通用寄存器 2925
16×64位
向量寄存器 2910
512位
zmm0
ymm0
xmm0
ymm15
xmm15
128位
256位
zmm31
标量浮点栈寄存器堆 2945
(X87浮点)
80位
0
7
重叠的
64位
MMX紧缩整数平坦寄存器堆
2950
片
2920

图29

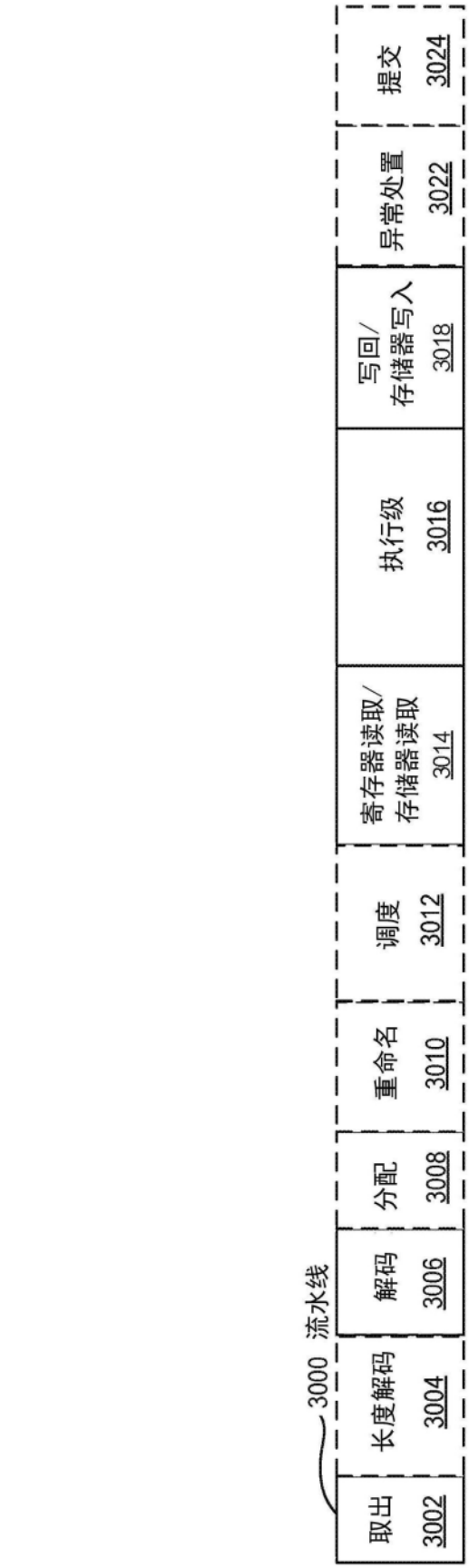

3000 流水线
取出 3002
长度解码 3004
解码 3006
分配 3008
重命名 3010
调度 3012
寄存器读取/存储器读取 3014
执行级 3016
写回/存储器写入 3018
异常处置 3022
提交 3024

图30A

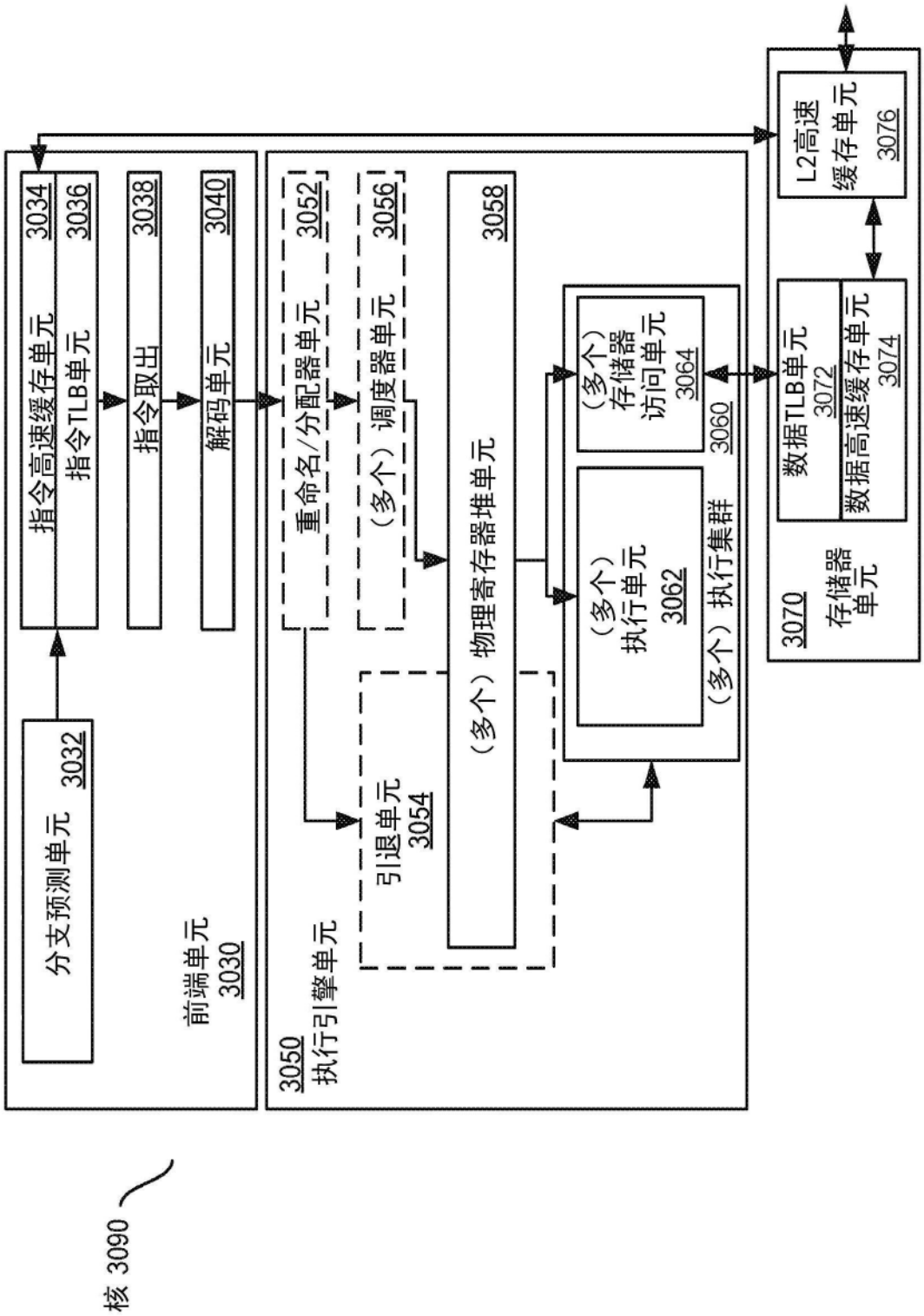

核 3090
前端单元 3030
分支预测单元 3032
指令高速缓存单元 3034
指令TLB单元 3036
指令取出 3038
解码单元 3040
3050 执行引擎单元
重命名/分配器单元 3052
引退单元 3054
(多个) 调度器单元 3056
(多个) 物理寄存器堆单元 3058
(多个) 执行单元 3062
(多个) 存储器访问单元 3064
(多个) 执行集群 3060
3070 存储器单元
数据TLB单元 3072
数据高速缓存单元 3074
L2高速缓存单元 3076

图30B

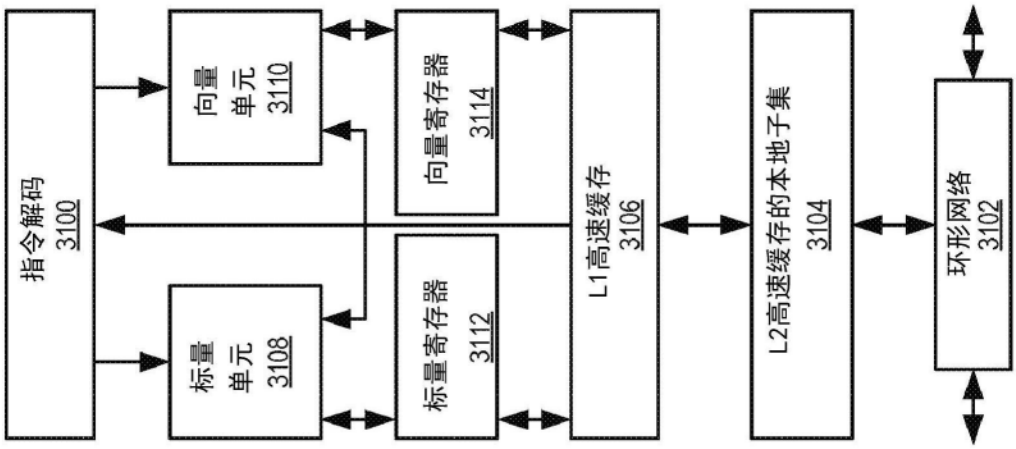

指令解码
3100
向量
单元
3110
标量
单元
3108
向量寄存器
3114
标量寄存器
3112
L1高速缓存
3106
L2高速缓存的本地子集
3104
环形网络
3102

图31A

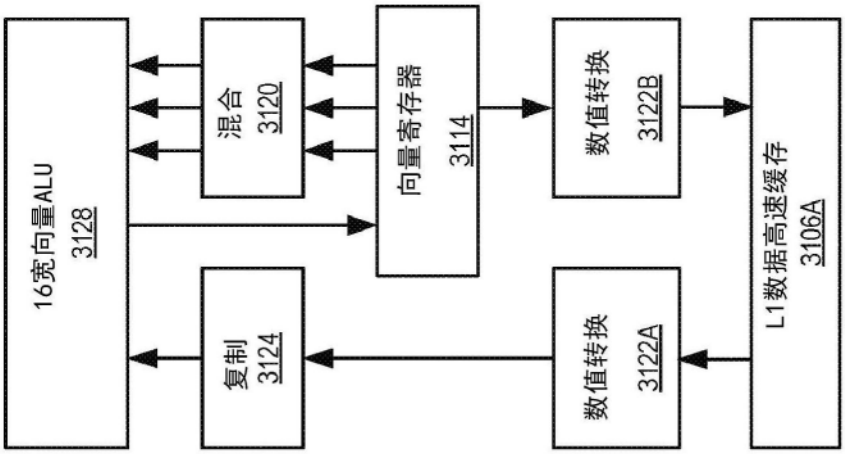

16宽向量ALU
3128
混合
3120
向量寄存器
3114
数值转换
3122B
L1数据高速缓存
3106A
复制
3124
数值转换
3122A

图31B

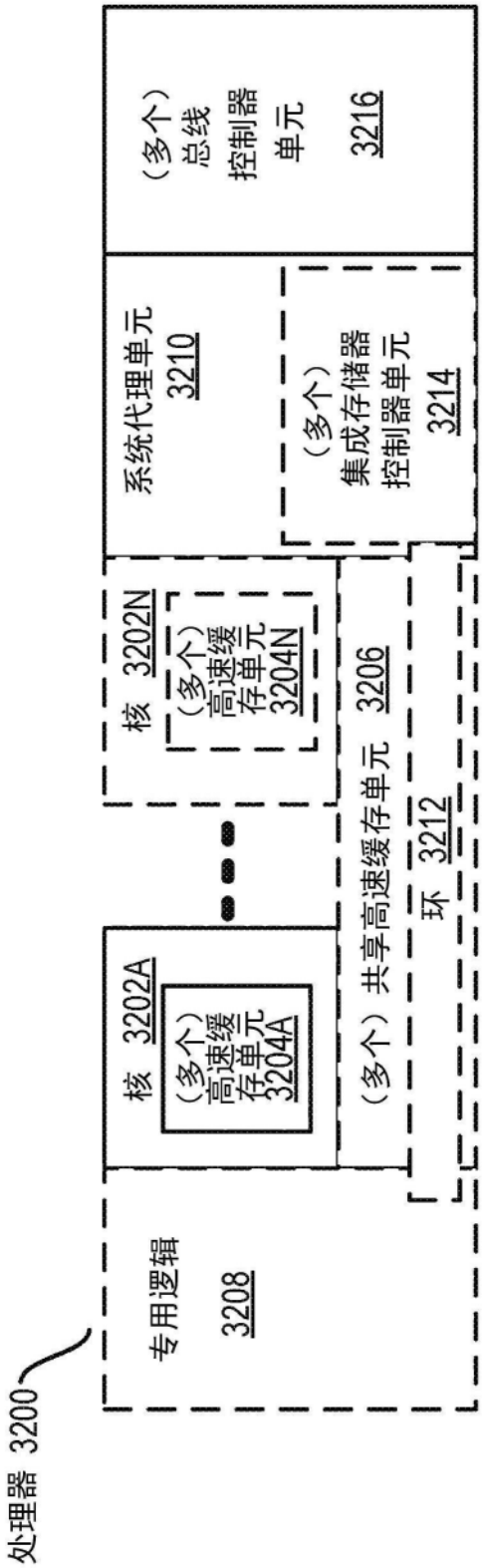

处理器 3200
专用逻辑
3208
核 3202A
(多个)
高速缓
存单元
3204A
核 3202N
(多个)
高速缓
存单元
3204N
(多个)共享高速缓存单元 3206
环 3212
系统代理单元
3210
(多个)
集成存储器
控制器单元
3214
(多个)
总线
控制器
单元
3216

图32

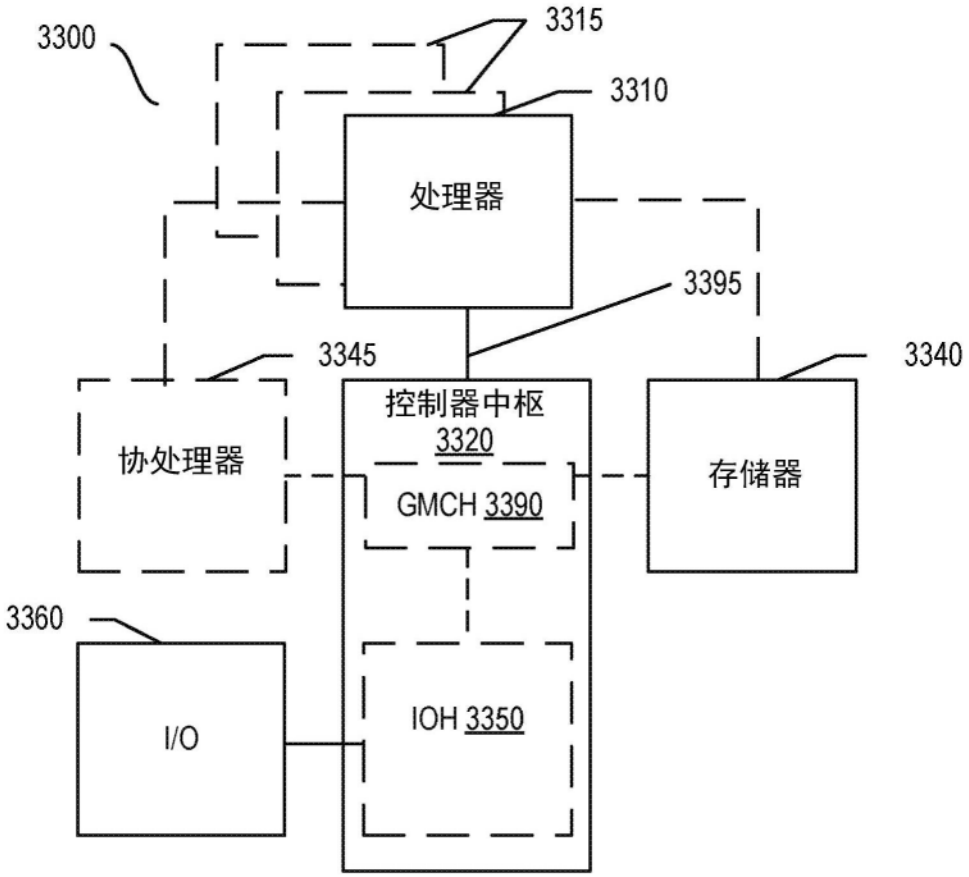

3300
3315
3310
处理器
3395
3345
3340
控制器中枢
3320
协处理器
GMCH 3390
存储器
3360
I/O
IOH 3350

图33

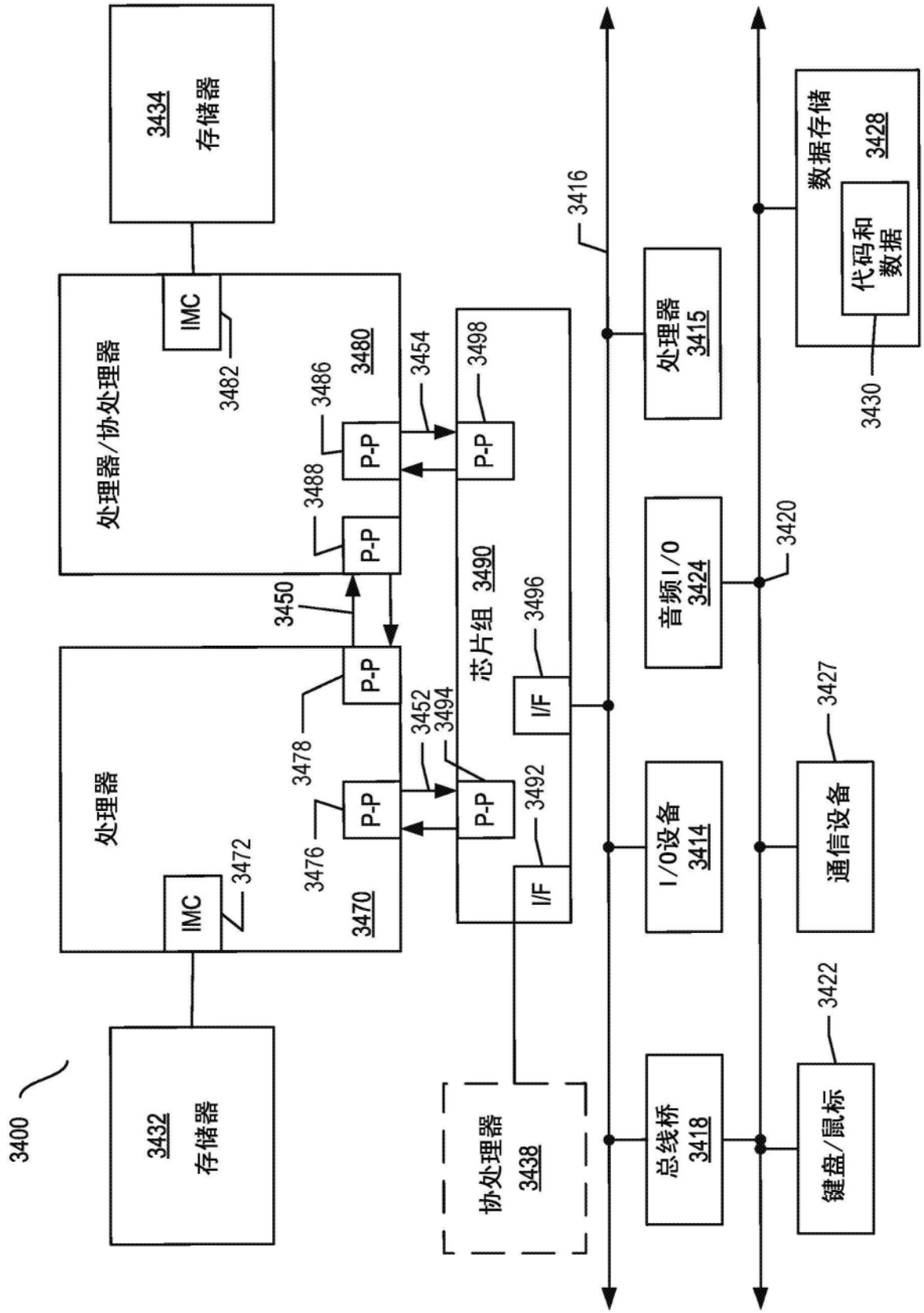

3400
3432
存储器
3434
存储器
处理器
IMC
3472
处理器/协处理器
IMC
3482
3470
3476
3478
3480
3486
3488
3450
3452
3454
3494
3498
芯片组 3490
P-P
I/F
3496
3492
协处理器
3438
3416
处理器
3415
音频I/O
3424
3420
I/O设备
3414
3427
通信设备
总线桥
3418
3422
键盘/鼠标
数据存储
3428
代码和
数据
3430

图34

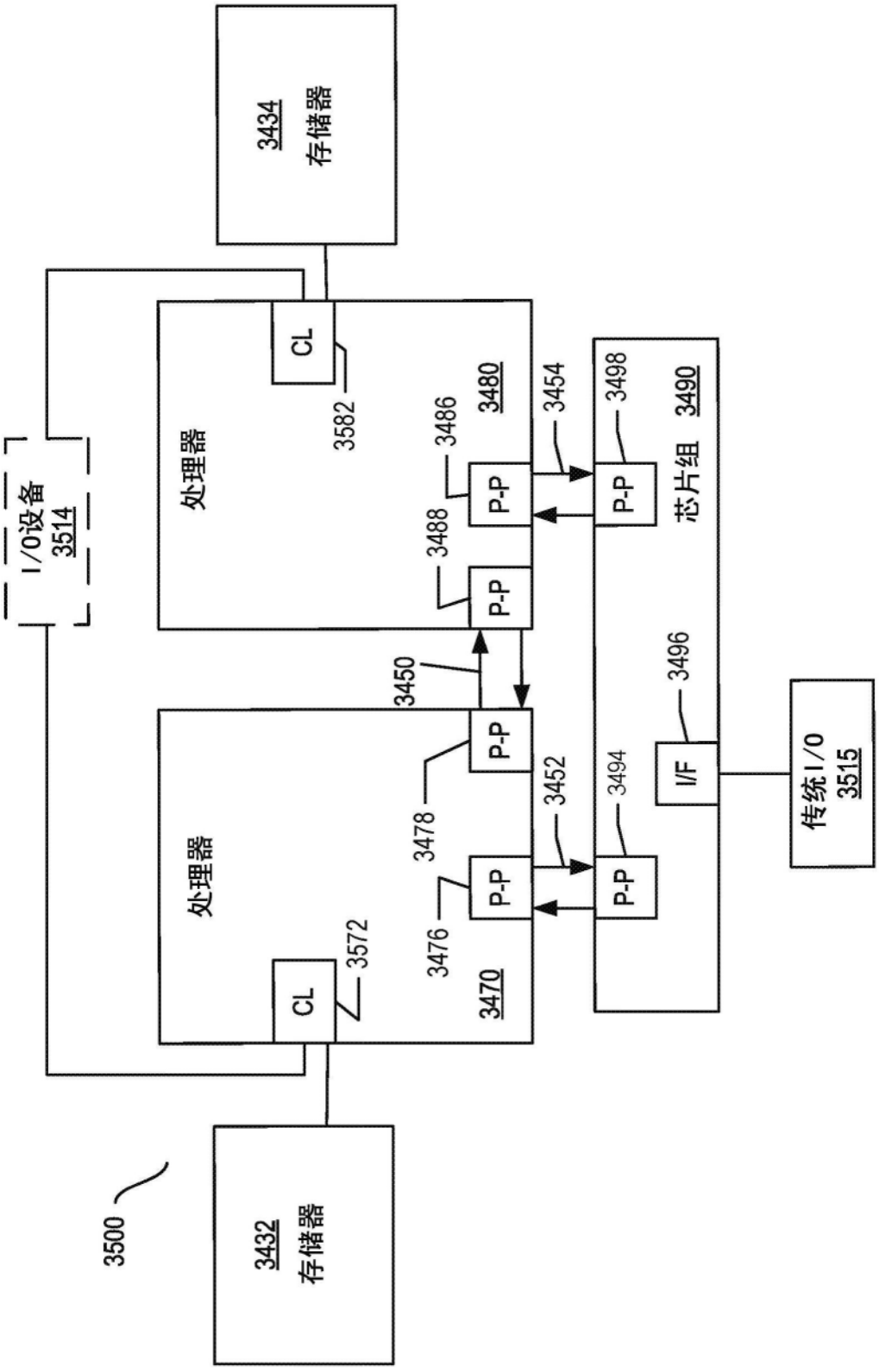

3434
存储器
CL
3582
处理器
3480
3486
3488
P-P
P-P
3454
3498
P-P
芯片组 3490
I/O设备
3514
3450
处理器
3478
3476
3470
P-P
P-P
3452
3494
P-P
I/F
3496
传统I/O
3515
CL
3572
3500
3432
存储器

图35

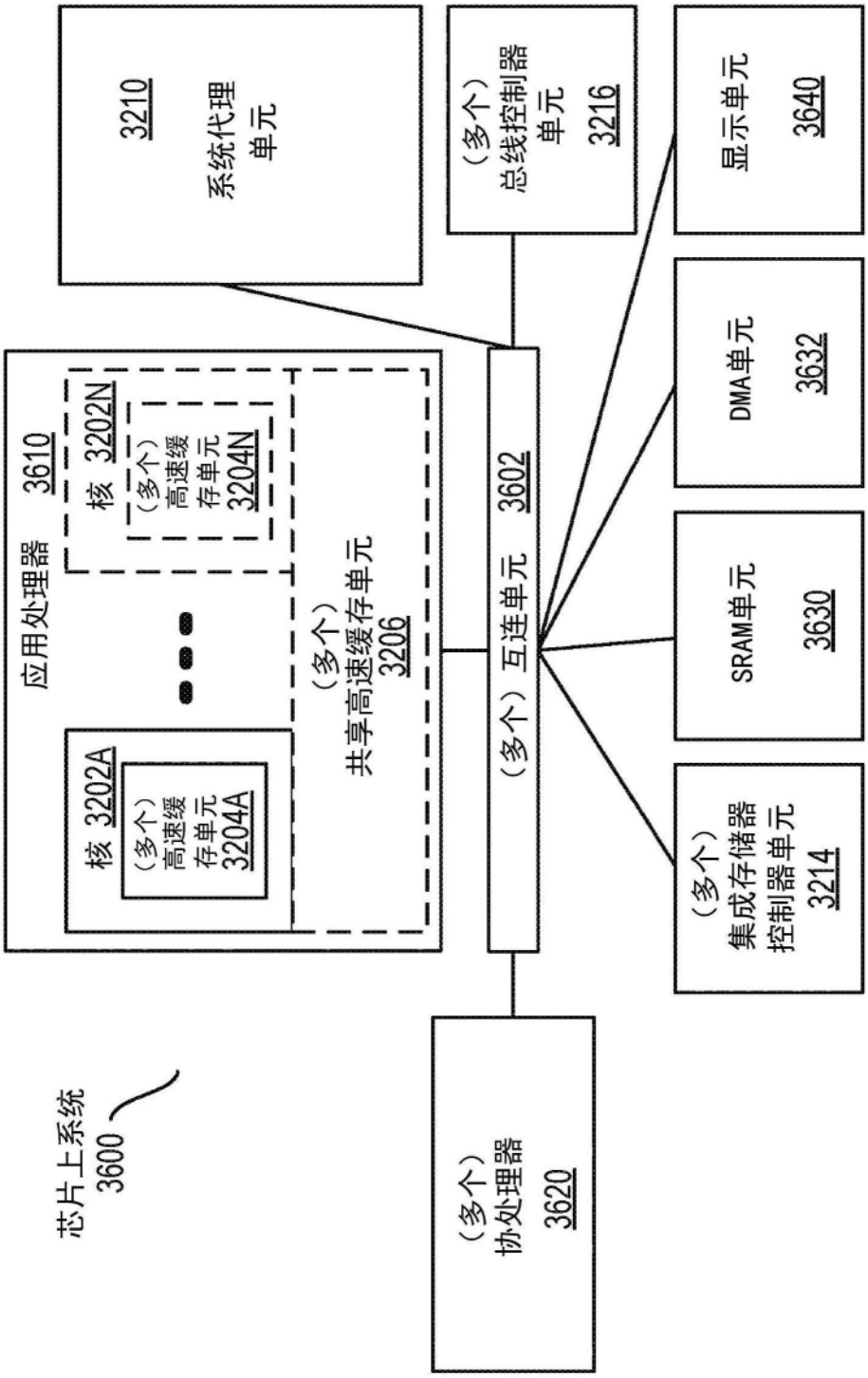

芯片上系统
3600
应用处理器 3610
核 3202A
（多个）高速缓存单元 3204A
核 3202N
（多个）高速缓存单元 3204N
（多个）共享高速缓存单元 3206
3210
系统代理单元
（多个）互连单元 3602
（多个）总线控制器单元 3216
显示单元 3640
DMA单元 3632
SRAM单元 3630
（多个）集成存储器控制器单元 3214
（多个）协处理器 3620

图36

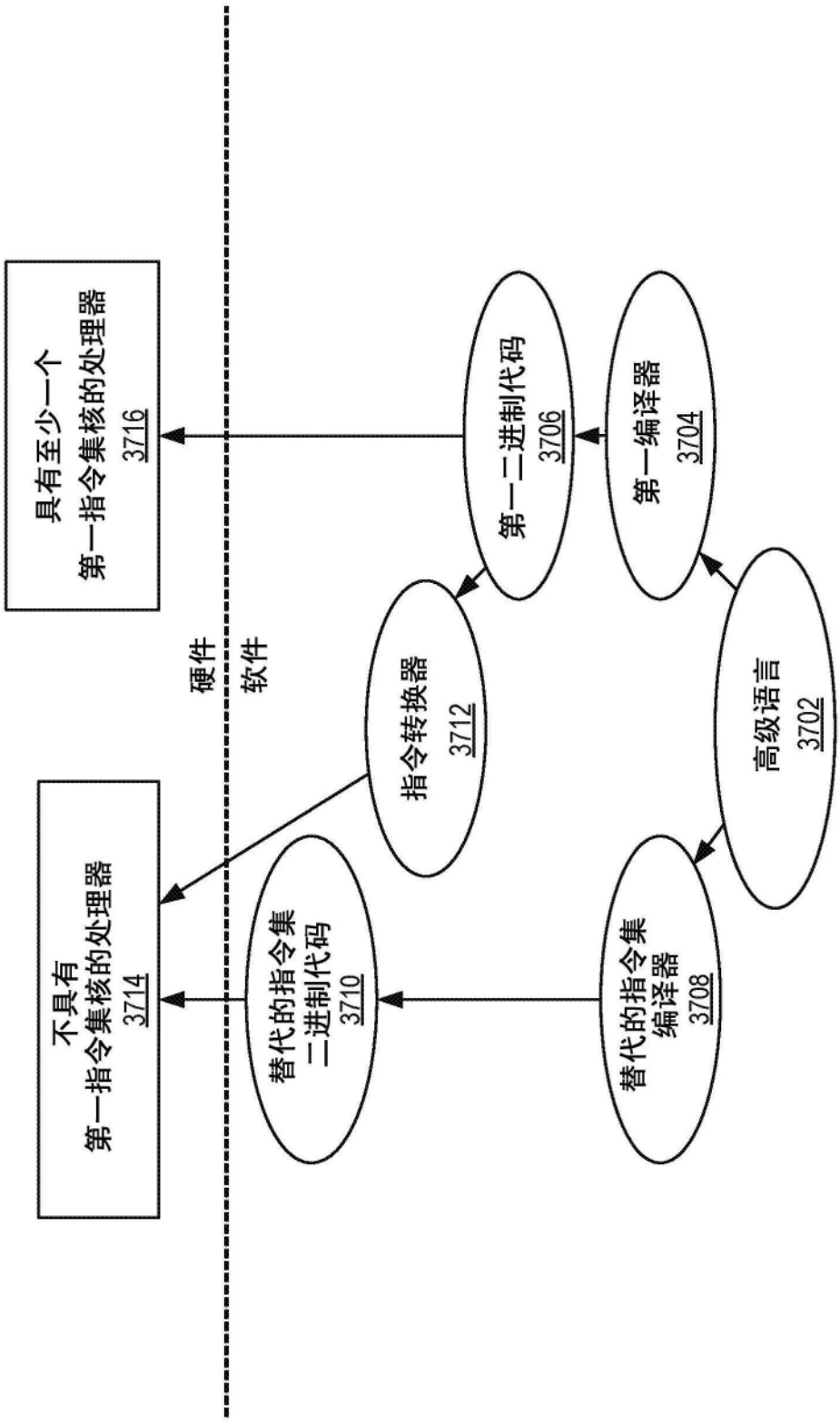

具有至少一个
第一指令集核的处理器
3716
第一二进制代码
3706
第一编译器
3704
硬件
软件
指令转换器
3712
高级语言
3702
不具有
第一指令集核的处理器
3714
替代的指令集
二进制代码
3710
替代的指令集
编译器
3708

图37