(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 114579929 B
(45)授权公告日 2023.08.08

(21)申请号 202210247720.2
(22)申请日 2022.03.14
(65)同一申请的已公布的文献号
申请公布号 CN 114579929 A
(43)申请公布日 2022.06.03
(73)专利权人 海飞科(南京)信息技术有限公司
地址 210000 江苏省南京市建邺区贤坤路1号科创中心2楼220-578号
(72)发明人 杨经纬 葛建明 李甲 桑永奇 谢钢锋 姚飞 仇小钢
(74)专利代理机构 北京市金杜律师事务所 11256
专利代理师 张宁
(51)Int.Cl.
G06F 17/16(2006.01)
G06F 7/523(2006.01)
(56)对比文件
CN 112559163 A,2021.03.26
CN 114090956 A,2022.02.25
CN 113836049 A,2021.12.24
US 2007271325 A1,2007.11.22
US 2020201642 A1,2020.06.25
US 2020104126 A1,2020.04.02
US 2019266217 A1,2019.08.29
US 11086968 B1,2021.08.10
Shaden Smith等.SPLATT:efficient and parallel sparse tensor-matrix multiplication.《2015 ieee international parallel and distributed processing symposium》.2015,全文.
审查员 李慧然

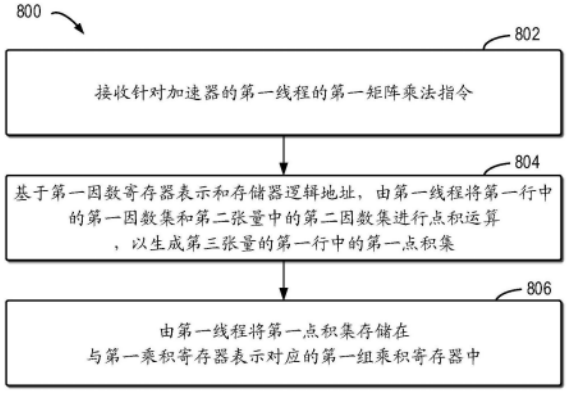
权利要求书5页 说明书20页 附图6页

(54)发明名称

加速器执行的方法和电子设备

(57)摘要

本文描述了一种提供了一种由加速器执行的方法和电子设备。该方法包括:接收针对加速器的第一线程的第一矩阵乘法指令;第一线程集基于针对第二张量的存储器逻辑地址将第二张量中的第二因数集广播至第二线程集;第二线程集中的第一线程基于所述第一因数寄存器表示将第一因数集和所述第二因数集进行点积运算,以生成第三张量的第一行中的第一点积集。通过将矩阵分解,并且按行分配线程,这样多个线程可以并行处理矩阵张量的多个行,从而加快矩阵乘法的处理效率。此外,由于编程人员在编程时知晓矩阵张量的行列结构以及加速器中的线程状况,因此可以灵活使用线程来并行处理矩阵乘法,从而提高编程的灵活性。



CN 114579929 B

1.一种由加速器执行的方法,包括:

接收针对加速器的第一线程集的第一张量乘法指令,所述第一张量乘法指令包括针对所述第一线程集的第一线程指示、针对第一张量的第一因数寄存器表示、针对第二张量的存储器逻辑地址、以及针对第三张量的第一乘积寄存器表示;

所述第一线程集基于针对所述第二张量的存储器逻辑地址将所述第二张量中的第二因数集广播至第二线程集,所述第二线程集不同于所述第一线程集;

所述第二线程集中的第一线程基于所述第一因数寄存器表示将所述第一张量中的第一行中的第一因数集和所述第二因数集进行点积运算,以生成所述第三张量的第一行中的第一点积集;以及

由所述第一线程将所述第一点积集累加到与第一乘积寄存器表示对应的第一组乘积寄存器中。

2.根据权利要求1所述的方法,还包括:

响应于接收到所述第二因数集,所述第二线程集中的第二线程基于所述第一因数寄存器表示将所述第一张量的第二行中的第三因数集和所述第二因数集进行点积运算,以生成所述第三张量的第二行中的第二点积集;以及

由所述第二线程将所述第二点积集累加到与第一乘积寄存器表示对应的第二组乘积寄存器中;

提供针对第一线程集各线程的执行条件,对于不满足执行条件的线程,其访存操作被视作超出张量地址范围而被忽略。

3.根据权利要求1所述的方法,其中所述第一张量乘法指令还包括第一合并计算模式指示;

生成所述第三张量的第一行中的第一点积集包括:

基于所述第一合并计算模式指示和所述第一因数寄存器表示,由所述第一线程将所述第一行中的第一因数集和所述第二因数集进行点积运算,以生成所述第三张量的第一行中的第一点积集。

4.根据权利要求3所述的方法,还包括:

基于所述第一合并计算模式指示和所述第一因数寄存器表示,由所述第一线程集中的第三线程将所述第一因数集和所述第二张量的第四因数集进行点积运算,以生成所述第三张量的第一行中的第三点积集,所述第四因数集不同于所述第二因数集,所述第三点积集不同于所述第一点积集;以及

由所述第三线程将所述第三点积集累加到与所述第一乘积寄存器表示对应的第三组乘积寄存器中。

5.根据权利要求1所述的方法,其中所述第一张量乘法指令还包括转置指示;

生成所述第三张量的第一行中的第一点积集包括:

基于所述转置指示和所述第一因数寄存器表示,由所述第一线程将所述第一行中的第一因数集和所述第二张量中的第二因数集进行点积运算,以生成所述第三张量的第一行中的第一点积集。

6.根据权利要求5所述的方法,其中基于所述转置指示和所述第一因数寄存器表示,由所述第一线程将所述第一行中的第一因数集和所述第二张量中的第二因数集进行点积运

算以生成所述第三张量的第一行中的第一点积集包括：

基于所述转置指示和所述存储器逻辑地址，将所述第二张量中的多个行的因数加载至高速缓存；

按列从所述多个行的因数中选择因数以形成所述第二因数集；以及

基于所述第一因数寄存器表示，由所述第一线程将所述第一行中的第一因数集和所述第二因数集进行点积运算以生成所述第三张量的第一行中的第一点积集。

7.根据权利要求1-6中任一项所述的方法，其中所述第一线程集将与所述存储器逻辑地址对应的所述第二因数集以广播的形式并行提供给所述第二线程集中的全部线程中的计算单元，而不提供至所述全部线程中的寄存器。

8.根据权利要求7所述的方法，其中所述存储器逻辑地址包括段基准数据和偏移数据，所述段基准数据表示所述第二张量的起始地址，所述偏移数据表示所述第二张量在多个维度中的各维上的偏移量。

9.根据权利要求1或3所述的方法，其中所述第一张量乘法指令还包括第二合并计算模式指示；

生成所述第三张量的第一行中的第一点积集包括：

基于所述第二合并计算模式指示和所述第一因数寄存器表示，由所述第一线程将所述第一行中的第一因数集和所述第二张量中的第二因数集进行点积运算，以生成所述第三张量的第一行中的第一点积集。

10.根据权利要求9所述的方法，还包括：

基于所述第二合并计算模式指示和所述第一因数寄存器表示，由所述第二线程集中的第四线程将所述第一行中的第五因数集和所述第二张量的第六因数集进行点积运算，以生成所述第三张量的所述第一行中的第四点积集，所述第五因数集不同于所述第一因数集，所述第六因数集不同于所述第二因数集，所述第四点积集不同于所述第一点积集；以及

由所述第四线程将所述第四点积集累加至与所述第一乘积寄存器表示对应的所述第一组乘积寄存器。

11.根据权利要求1所述的方法，其中

所述第一乘积寄存器表示对应于一个或多个乘积寄存器，所述一个或多个乘积寄存器的数目与合并计算模式以及所述第二张量的列数相关，不同线程的乘积寄存器构成结果张量，每个线程的乘积寄存器包括结果张量每行的部分或全部；以及

所述结果张量的行数与第一张量的行数相同，所述结果张量的列数与第二张量的列数相同。

12.根据权利要求11所述的方法，其中

所述第二线程集中的线程内的乘积寄存器的数目是可变的，所述乘积寄存器的数目取决于所述第一张量乘法指令的执行条件，所述执行条件确定对所述第二张量中的列的访问；以及

如果所述第二张量中的第一列未被访问，则所述第二张量中的第一列不参与矩阵乘法计算。

13.根据权利要求1所述的方法，其中

所述第一张量乘法指令在一次完整的执行过程中被多次发射，其中所述第一张量乘法

指令第一次以存储指令的方式被发射，以用于获取所述第二张量中的列数据或者行数据；以及

响应于获取到所述第二张量中的列数据或者行数据，并且所述第一张量的数据已被存储在所述第一因数寄存器中，第一张量乘法指令以数学计算指令的方式被二次或比二次更多的次数发射，以用于执行所述第三张量的行内的各列结果的计算。

14.根据权利要求13所述的方法，其中

在进行二次或比二次更多的次数发射之前，检查所述第一因数寄存器的对应的令牌状态；

如果所述令牌状态表示所述第一张量的数据已被存储在所述第一因数寄存器中，则以数学计算指令方式发射，否则阻塞发射队列，直至所述第一张量的数据已被存储在所述第一因数寄存器中。

15.根据权利要求11所述的方法，还包括：

基于所述第一乘积寄存器表示，确定针对所述第三张量的乘积寄存器使用范围是否超出单个线程内的寄存器文件的范围；以及

如果确定针对所述第三张量的乘积寄存器使用范围超出单个线程内的寄存器文件的范围，则忽略超出寄存器文件范围的计算操作或访存操作并且报错。

16.一种电子设备，包括：

流处理器；

页表装置，耦合至所述流处理器；

存储器；

处理引擎单元，耦合至所述流处理器、所述存储器和所述页表装置，被配置为执行权利要求1-15中任一项所述的方法。

17.一种加速器，包括：

接收单元，被配置为接收针对加速器的第一线程集的第一张量乘法指令，所述第一张量乘法指令包括针对所述第一线程集的第一线程指示、针对第一张量的第一因数寄存器表示、针对第二张量的存储器逻辑地址、以及针对第三张量的第一乘积寄存器表示；

广播单元，被配置为由所述第一线程集基于针对所述第二张量的存储器逻辑地址将所述第二张量中的第二因数集广播至第二线程集，所述第二线程集不同于所述第一线程集；

生成单元，被配置为由所述第二线程集中的第一线程基于所述第一因数寄存器表示将所述第一张量中的第一行中的第一因数集和所述第二因数集进行点积运算，以生成所述第三张量的第一行中的第一点积集；以及

存储单元，被配置为由所述第一线程将所述第一点积集累加到与第一乘积寄存器表示对应的第一组乘积寄存器中。

18.根据权利要求17所述的加速器，其中

所述生成单元被进一步配置为：响应于接收到所述第二因数集，所述第二线程集中的第二线程基于所述第一因数寄存器表示将所述第一张量的第二行中的第三因数集和所述第二因数集进行点积运算，以生成所述第三张量的第二行中的第二点积集；以及

所述存储单元被进一步配置为：由所述第二线程将所述第二点积集累加到与第一乘积寄存器表示对应的第二组乘积寄存器中。

19.根据权利要求18所述的加速器，其中所述第一张量乘法指令还包括第一合并计算模式指示；

所述生成单元被进一步配置为：

基于所述第一合并计算模式指示和所述第一因数寄存器表示，由所述第一线程将所述第一行中的第一因数集和所述第二因数集进行点积运算，以生成所述第三张量的第一行中的第一点积集。

20.根据权利要求19所述的加速器，其中

所述生成单元被进一步配置为：基于所述第一合并计算模式指示和所述第一因数寄存器表示，由所述第一线程集中的第三线程将所述第一因数集和所述第二张量的第四因数集进行点积运算，以生成所述第三张量的第一行中的第三点积集，所述第四因数集不同于所述第二因数集，所述第三点积集不同于所述第一点积集；以及

所述存储单元被进一步配置为：由所述第三线程将所述第三点积集累加到与所述第一乘积寄存器表示对应的第三组乘积寄存器中。

21.根据权利要求17所述的加速器，其中所述第一张量乘法指令还包括转置指示；

所述生成单元被进一步配置为：

基于所述转置指示和所述第一因数寄存器表示，由所述第一线程将所述第一行中的第一因数集和所述第二张量中的第二因数集进行点积运算，以生成所述第三张量的第一行中的第一点积集。

22.根据权利要求21所述的加速器，其中所述生成单元被进一步配置为：

基于所述转置指示和所述存储器逻辑地址，将所述第二张量中的多个行的因数加载至高速缓存；

按列从所述多个行的因数中选择因数以形成所述第二因数集；以及

基于所述第一因数寄存器表示，由所述第一线程将所述第一行中的第一因数集和所述第二因数集进行点积运算以生成所述第三张量的第一行中的第一点积集。

23.根据权利要求17或19所述的加速器，其中所述第一张量乘法指令还包括第二合并计算模式指示；

所述生成单元被进一步配置为：

基于所述第二合并计算模式指示和所述第一因数寄存器表示，由所述第一线程将所述第一行中的第一因数集和所述第二张量中的第二因数集进行点积运算，以生成所述第三张量的第一行中的第一点积集。

24.根据权利要求23所述的加速器，其中所述生成单元被进一步配置为：

基于所述第二合并计算模式指示和所述第一因数寄存器表示，由所述第二线程集中的第四线程将所述第一行中的第五因数集和所述第二张量的第六因数集进行点积运算，以生成所述第三张量的所述第一行中的第四点积集，所述第五因数集不同于所述第一因数集，所述第六因数集不同于所述第二因数集，所述第四点积集不同于所述第一点积集；以及

由所述第四线程将所述第四点积集累加至与所述第一乘积寄存器表示对应的所述第一组乘积寄存器。

25.根据权利要求17所述的加速器，其中

所述第一乘积寄存器表示对应于一个或多个乘积寄存器，所述一个或多个乘积寄存器

的数目与合并计算模式以及所述第二张量的列数相关，不同线程的乘积寄存器构成结果张量，每个线程的乘积寄存器包括结果张量每行的部分或全部；以及

所述结果张量的行数与第一张量的行数相同，所述结果张量的列数与第二张量的列数相同。

26.根据权利要求25所述的加速器，其中

所述第二线程集中的线程内的乘积寄存器的数目是可变的，所述乘积寄存器的数目取决于所述第一张量乘法指令的执行条件，所述执行条件确定对所述第二张量中的列的访问；以及

如果所述第二张量中的第一列未被访问，则所述第二张量中的第一列不参与矩阵乘法计算。

27.根据权利要求17所述的加速器，其中

所述第一张量乘法指令在一次完整的执行过程中会被多次发射，其中所述第一张量乘法指令第一次以存储指令的方式被发射，以用于获取所述第二张量中的列数据或者行数据；以及

响应于获取到所述第二张量中的列数据或者行数据，并且所述第一张量的数据已被存储在所述第一因数寄存器中，所述第一张量乘法指令以数学计算指令的方式被二次或比二次更多的次数发射，以用于执行所述第三张量的行内的各列结果的计算。

28.根据权利要求27所述的加速器，还包括检查单元，被配置为在进行二次或比二次更多的次数发射之前，检查所述第一因数寄存器的对应的令牌状态；

如果所述令牌状态表示所述第一张量的数据已被存储在所述第一因数寄存器中，则以数学计算指令方式发射，否则阻塞发射队列，直至所述第一张量的数据已被存储在所述第一因数寄存器中。

29.根据权利要求25所述的加速器，还包括越界检查单元，被配置为

基于所述第一乘积寄存器表示，确定针对所述第三张量的乘积寄存器使用范围是否超出单个线程内的寄存器文件的范围；以及

如果确定针对所述第三张量的乘积寄存器使用范围超出单个线程内的寄存器文件的范围，则忽略超出寄存器文件范围的计算操作或访存操作并且报错。

30.根据权利要求17-22中任一项所述的加速器，其中所述第一线程集将与所述存储器逻辑地址对应的所述第二因数集以广播的形式并行提供给所述第二线程集中的全部线程中的计算单元，而不提供至所述全部线程中的寄存器。

加速器执行的方法和电子设备

技术领域

[0001] 本公开的实施例一般地涉及电子领域，更具体而言涉及一种由加速器执行的方法和加速器。

背景技术

[0002] 诸如图形处理器(GPU)之类的并行高性能多线程多核处理系统处理数据的速度比过去快得多。这些处理系统可以将复杂的计算分解为较小的任务，并且由多核并行处理以增加处理效率并且减少处理时间。

[0003] 在一些情形下，诸如GPU之类的多核处理器对具有大量相同或相似形式的数据的张量的处理尤为有利。张量数据在计算机领域通常表示一维或多维数组的数据，例如图像数据就是一种常规的二维张量数据，其可以由二维数组表示。再例如，彩色图像是一种三维数组数据，除了包括宽和高的二维像素阵列之外，彩色图像还包括红绿蓝(RGB)通道维度。对诸如二维数组之类的张量进行处理例如可以包括矩阵乘法。基于GPU之类内部加速器的常规矩阵乘法对于程序编程人员通常不可获知，因此编程人员通常不了解硬件执行矩阵乘法的过程，因而也无法针对硬件对矩阵乘法的计算进行优化，这导致了程序的执行效率以及张量处理的效率通常较低。

发明内容

[0004] 本公开的实施例提供了一种用于由加速器执行的方法和电子设备。

[0005] 在第一方面，提供了一种由加速器执行的方法。该方法包括：接收针对加速器的第一线程集的第一张量乘法指令，第一张量乘法指令包括针对第一线程集的第一线程指示、针对第一张量的第一因数寄存器表示、针对第二张量的存储器逻辑地址、以及针对第三张量的第一乘积寄存器表示；第一线程集基于针对第二张量的存储器逻辑地址将第二张量中的第二因数集广播至第二线程集，第二线程集不同于第一线程集；第二线程集中的第一线程基于第一因数寄存器表示将第一张量中的第一行中的第一因数集和第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集；以及由第一线程将第一点积集累加到与第一乘积寄存器表示对应的第一组乘积寄存器中。通过将矩阵分解，并且按行分配线程，这样多个线程可以并行处理矩阵张量的多个行，从而加快矩阵乘法的处理速度。此外，由于编程人员在编程时知晓矩阵张量的行列结构以及加速器中的线程状况，因此可以灵活使用线程来并行处理矩阵乘法，从而提高编程的灵活性。

[0006] 在一种可能的实现方式中，第一因数集包括第一张量的第一行中的至少一部分因数数据。第二因数集包括第二张量中的至少一部分的因数数据。第一点积集包括第三张量的第一行中的至少一部分乘积数据。

[0007] 在一种可能的实现方式中，每个线程包括第一组寄存器和第二组寄存器，其中第一组寄存器用于存储第一因数矩阵的一行中数据的至少一部分，第二组寄存器用于存储乘积矩阵中的一行的数据。第二因数矩阵的一列中的数据可以存储于片上存储器、一级高速

缓存或片外存储器。这样，在矩阵乘法执行过程中，第一线程的执行单元可以仅从第一组寄存器读取第一因数矩阵的一行中的数据一次，并且在后续针对第二因数矩阵各列的点积运算过程中重复使用。此外，第二因数矩阵的一列中的数据可以被并行广播至多个(例如与第一因数矩阵的行相同数目或其一半数目)线程中的执行单元，并且重复使用。以此方式，可以减少数据在不同存储装置之间的传送，从而减少矩阵乘法计算过程中因数据传输引起的时间延迟。

[0008] 在一种可能的实现方式中，该方法还包括响应于接收到第二因数集，第二线程集中的第二线程基于第一因数寄存器表示将第一张量的第二行中的第三因数集和第二因数集进行点积运算，以生成第三张量的第二行中的第二点积集；以及由第二线程将第二点积集累加到与第一乘积寄存器表示对应的第二组乘积寄存器中。

[0009] 在一种可能的实现方式中，第一张量乘法指令还包括第一合并计算模式指示。生成第三张量的第一行中的第一点积集包括：基于第一合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0010] 在一种可能的实现方式中，该方法还包括：基于第一合并计算模式指示和第一因数寄存器表示，由第一线程集中的第三线程将第一因数集和第二张量的第四因数集进行点积运算，以生成第三张量的第一行中的第三点积集，第四因数集不同于第二因数集，第三点积集不同于第一点积集；以及由第三线程将第三点积集累加到与第一乘积寄存器表示对应的第三组乘积寄存器中。

[0011] 在一种可能的实现方式中，第一张量乘法指令还包括第二合并计算模式指示。生成第三张量的第一行中的第一点积集包括：基于第二合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0012] 在一种可能的实现方式中，该方法还包括：基于第二合并计算模式指示和第一因数寄存器表示，由第二线程集中的第四线程将第五因数集和第二张量的第六因数集进行点积运算，以生成第三张量的第一行中的第四点积集，第五因数集不同于第一因数集，第六因数集不同于第二因数集，第四点积集不同于第一点积集；以及由第四线程将第四点积集累加至与第一乘积寄存器表示对应的第一组乘积寄存器。

[0013] 在一种可能的实现方式中，第一张量乘法指令还包括转置指示。生成第三张量的第一行中的第一点积集包括：基于转置指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0014] 在一种可能的实现方式中，基于转置指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算以生成第三张量的第一行中的第一点积集包括：基于转置指示和存储器逻辑地址，将第二张量中的多个行的因数加载至高速缓存；按列从多个行的因数中选择因数以形成第二因数集；以及基于第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二因数集进行点积运算以生成第三张量的第一行中的第一点积集。

[0015] 在一种可能的实现方式中，多个行中未被选择的多个因数被保留在高速缓存中，

直至上述多个未被选择的因数被选择用于进行矩阵乘法的计算。

[0016] 在一种可能的实现方式中，第一线程集将与存储器逻辑地址对应的第二因数集以广播的形式并行提供给第二线程集中的全部线程中的计算单元，而不提供至全部线程中的寄存器。

[0017] 在一种可能的实现方式中，存储器逻辑地址包括段基准数据和偏移数据，段基准数据表示在第二张量中的起始地址，偏移数据表示在第二张量多个维度中的各维上的偏移量。

[0018] 在一种可能的实现方式中，第一乘积寄存器表示对应于一个或多个乘积寄存器，一个或多个乘积寄存器的数目与合并计算模式以及第二张量的列数相关，不同线程的乘积寄存器构成结果张量，每个线程的乘积寄存器包括结果张量每行的部分或全部；以及结果张量的行数与第一张量的行数相同，结果张量的列数与第二张量的列数相同。

[0019] 在一种可能的实现方式中，第二线程集中的线程内的乘积寄存器的数目是可变的，乘积寄存器的数目取决于第一张量乘法指令的执行条件，执行条件确定对第二张量中的列的访问；以及如果第二张量中的第一列未被访问，则第二张量中的第一列不参与矩阵乘法计算。

[0020] 在一种可能的实现方式中，第一张量乘法指令被多次发射，其中第一张量乘法指令第一次以存储指令的方式被发射，以用于获取第二张量中的列数据或者行数据；以及响应于获取到第二张量中的列数据或者行数据，并且第一张量的数据已被存储在第一因数寄存器中，第一张量乘法指令以数学计算指令的方式被二次或多次发射，以用于执行第三张量的行内的各列结果的计算。

[0021] 在一种可能的实现方式中，在进行二次或多次发射之前，检查第一因数寄存器的对应的令牌状态；如果令牌状态表示第一张量的数据已被存储在第一因数寄存器中，则以数学计算指令方式发射，否则阻塞发射队列，直至第一张量的数据已被存储在第一因数寄存器中。

[0022] 在一种可能的实现方式中，基于第一乘积寄存器表示，确定针对第三张量的乘积寄存器使用范围是否超出单个线程内的寄存器文件的范围；以及如果确定针对第三张量的乘积寄存器使用范围超出单个线程内的寄存器文件的范围，则忽略超出寄存器文件范围的计算操作或访存操作并且报错。

[0023] 根据本公开的第二方面，提供一种电子设备。电子设备包括：流处理器；页表装置，耦合至流处理器；存储器；处理引擎单元，耦合至流处理器、存储器和页表装置，被配置为执行根据第一方面的方法。

[0024] 根据本公开的第三方面，提供一种电子设备。该电子设备包括：接收单元，被配置为接收针对加速器的第一线程集的第一张量乘法指令，第一张量乘法指令包括针对第一线程集的第一线程指示、针对第一张量的第一因数寄存器表示、针对第二张量的存储器逻辑地址、以及针对第三张量的第一乘积寄存器表示；广播单元，被配置为由第一线程集基于针对第二张量的存储器逻辑地址将第二张量中的第二因数集广播至第二线程集，第二线程集不同于第一线程集；生成单元，被配置为由第二线程集中的第一线程基于第一因数寄存器表示将第一张量中的第一行中的第一因数集和第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集；以及存储单元，被配置为由第一线程将第一点积集累加到与第

一乘积寄存器表示对应的第一组乘积寄存器中。通过将矩阵分解，并且按行分配线程，这样多个线程可以并行处理矩阵张量的多个行，从而加快矩阵乘法的处理效率。此外，由于编程人员在编程时知晓矩阵张量的行列结构以及加速器中的线程状况，因此可以灵活使用线程来并行处理矩阵乘法，从而提高编程的灵活性。

[0025] 在一种可能的实现方式中，每个线程包括第一组寄存器和第二组寄存器，其中第一组寄存器用于存储第一因数矩阵的一行中数据的至少一部分，第二组寄存器用于存储乘积矩阵中的一行的数据。第二因数矩阵的一列中的数据可以来自于片上存储器、一级高速缓存或片外存储器。这样，在矩阵乘法执行过程中，第一线程的执行单元可以仅从第一组寄存器读取第一因数矩阵的一行中的数据一次，并且在后续点积运算过程中重复使用。此外，第二因数矩阵的一列中的数据可以被并行广播至多个(例如与第一因数矩阵的行相同数目或其一半数目)线程中的执行单元，并且重复使用。以此方式，可以减少数据在不同存储装置之间的传送，从而减少矩阵乘法计算过程中因数据传输引起的时间。

[0026] 在一种可能的实现方式中，生成单元被进一步配置为响应于接收到第二因数集，第二线程集中的第二线程基于第一因数寄存器表示将第一张量的第二行中的第三因数集和第二因数集进行点积运算，以生成第三张量的第二行中的第二点积集。存储单元908被进一步配置为由第二线程将第二点积集累加到与第一乘积寄存器表示对应的第二组乘积寄存器中。

[0027] 在一种可能的实现方式中，第一张量乘法指令还包括第一合并计算模式指示。生成单元还被配置为：基于第一合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0028] 在一种可能的实现方式中，生成单元还被配置为基于第一合并计算模式指示和第一因数寄存器表示，由第一线程集中的第三线程将第一因数集和第二张量的第四因数集进行点积运算，以生成第三张量的第一行中的第三点积集，第四因数集不同于第二因数集，第三点积集不同于第一点积集。存储单元还被配置为由第三线程将第三点积集累加到与第一乘积寄存器表示对应的第三组乘积寄存器中。

[0029] 在一种可能的实现方式中，第一张量乘法指令还包括第二合并计算模式指示。生成单元还被配置为基于第二合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0030] 在一种可能的实现方式中，生成单元还被配置为基于第二合并计算模式指示和第一因数寄存器表示，由第二线程集中的第四线程将第五因数集和第二张量的第六因数集进行点积运算，以生成第三张量的第一行中的第四点积集，第五因数集不同于第一因数集，第六因数集不同于第二因数集，第四点积集不同于第一点积集。存储单元还被配置为由第四线程将第四点积集累加至与第一乘积寄存器表示对应的第一组乘积寄存器。

[0031] 在一种可能的实现方式中，第一张量乘法指令还包括转置指示。生成单元还被配置为：基于转置指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0032] 在一种可能的实现方式中，生成单元还被配置为：基于转置指示和存储器逻辑地

址，将第二张量中的多个行的因数加载至高速缓存；按列从多个行的因数中选择因数以形成第二因数集；以及基于第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二因数集进行点积运算以生成第三张量的第一行中的第一点积集。在一种可能的实现方式中，多个行中未被选择的多个因数被保留在一级高速缓存中，直至上述多个未被选择的因数被选择用于进行矩阵乘法的计算。

[0033] 在一种可能的实现方式中，第一线程集将与存储器逻辑地址对应的第二因数集由以广播的形式并行提供给第二线程集中的全部线程。

[0034] 在一种可能的实现方式中，存储器逻辑地址包括段基准数据和偏移数据，段基准数据表示第二张量的起始地址，偏移数据表示第二张量在多个维度中的各维上的偏移量。

[0035] 在一种可能的实现方式中，第一乘积寄存器表示对应于一个或多个乘积寄存器，一个或多个乘积寄存器的数目与合并计算模式以及第二张量的列数相关，不同线程的乘积寄存器构成结果张量，每个线程的乘积寄存器包括结果张量每行的部分或全部；以及结果张量的行数与第一张量的行数相同，结果张量的列数与第二张量的列数相同。

[0036] 在一种可能的实现方式中，第二线程集中的线程内的乘积寄存器的数目是可变的，乘积寄存器的数目取决于第一张量乘法指令的执行条件，执行条件确定对第二张量中的列的访问；以及如果第二张量中的第一列未被访问，则第二张量中的第一列不参与矩阵乘法计算。

[0037] 在一种可能的实现方式中，第一张量乘法指令被多次发射，其中第一张量乘法指令第一次以存储指令的方式被发射，以用于获取第二张量中的列数据或者行数据；以及响应于获取到第二张量中的列数据或者行数据，并且第一张量的数据已被存储在第一因数寄存器中，第一张量乘法指令以数学计算指令的方式被二次或多次发射，以用于执行第三张量的行内的各列结果的计算。

[0038] 在一种可能的实现方式中，加速器还包括检查单元，检查单元被配置为在进行二次或多次发射之前，检查第一因数寄存器的对应的令牌状态；如果令牌状态表示第一张量的数据已被存储在第一因数寄存器中，则以数学计算指令方式发射，否则阻塞发射队列，直至第一张量的数据已被存储在第一因数寄存器中。

[0039] 在一种可能的实现方式中，加速器还包括越界检查单元。越界检查单元被配置为基于第一乘积寄存器表示，确定针对第三张量的乘积寄存器使用范围是否超出单个线程内的寄存器文件的范围；以及如果确定针对第三张量的乘积寄存器使用范围超出单个线程内的寄存器文件的范围，则忽略超出寄存器文件范围的计算操作或访存操作并且报错。

[0040] 在一种可能的实现方式中，第一线程集将与存储器逻辑地址对应的第二因数集以广播的形式并行提供给第二线程集中的全部线程中的计算单元，而不提供至全部线程中的寄存器。

[0041] 根据本公开的实施例的方法和电子设备，编程人员可以从矩阵角度考虑线程任务分配，这样可以使用一个或多个线程来计算第一因数矩阵的一行与第二因数矩阵的点积，并且将相应结果累加到相同线程内的乘积寄存器，从而增加针对矩阵乘法的编程灵活性并且提高矩阵乘法的执行效率。

附图说明

[0042] 通过结合附图对本公开示例性实施例进行更详细的描述，本公开的上述以及其他目的、特征和优势将变得更加明显，其中，在本公开示例性实施例中，相同的参考标号通常代表相同部件。

[0043] 图1示出了本公开的多个实施例能够在其中实现的示例环境的示意图；

[0044] 图2示出了根据本公开的一个实施例的芯片示意框图；

[0045] 图3示出了根据本公开的一个实施例的三维张量示意框图；

[0046] 图4示出了根据本公开的一个实施例的图像数据的页分配示意图；

[0047] 图5示出了根据本公开的一个实施例的矩阵乘法的示意图；

[0048] 图6示出了根据本公开的一个实施例的矩阵乘法的一部分的示意图；

[0049] 图7示出了根据本公开的另一实施例的矩阵乘法的一部分的示意图；

[0050] 图8示出了根据本公开的一个实施例的由加速器执行的方法的示意流程图；以及

[0051] 图9示出了根据本公开的一个实施例的电子设备的示意框图。

具体实施方式

[0052] 下面将参照附图更详细地描述本公开的优选实施例。虽然附图中示出了本公开的优选实施例，然而应该理解，本公开可以以各种形式实现而不应被这里阐述的实施例限制。相反，提供这些实施例是为了使本公开更加透彻和完整，并且能够将本公开的范围完整地传达给本领域的技术人员。

[0053] 在本文中使用的术语“包括”及其变形表示开放性包括，即“包括但不限于”。除非特别申明，术语“或”表示“和/或”。术语“基于”表示“至少部分地基于”。术语“一个示例实施例”和“一个实施例”表示“至少一个示例实施例”。术语“另一实施例”表示“至少一个另外的实施例”。术语“第一”、“第二”等等可以指代不同的或相同的对象。下文还可能包括其他明确的和隐含的定义。

[0054] 如前文所提及的，基于GPU之类的内部硬件加速器的常规矩阵乘法对于程序编程人员通常不可获知，因此编程人员通常不了解硬件执行矩阵乘法的过程，因而也无法针对硬件对矩阵乘法的计算进行优化，这导致了程序的执行效率以及张量处理的效率通常较低。

[0055] 在本公开的一些实施例中，编程人员可以从矩阵的行列结构角度考虑线程任务分配，这样可以使用一个或多个线程来计算第一因数矩阵的一行与第二因数矩阵的点积，并且将相应结果查出在相同线程内的乘积寄存器，从而增加针对矩阵乘法的编程灵活性并且提高矩阵乘法的执行效率。

[0056] 图1示出了本公开的多个实施例能够在其中实现的示例环境100的示意图。示例环境100例如可以是诸如计算机之类的具有计算能力的电子设备。在一个实施例中，示例环境100例如包括中央处理器(CPU)20、系统存储器10、北桥/存储器桥30、加速器40、设备存储器50和南桥/输入输出(IO)桥60。系统存储器10例如可以是诸如动态随机存取存储器(DRAM)之类的易失性存储器。北桥/存储器桥30例如集成了内存控制器、PCIe控制器等，其负责CPU 20和高速接口之间的数据交换以及桥接CPU 20和南桥/IO桥60。南桥/IO桥60用于计算机的低速接口，例如串行高级技术接口(SATA)控制器等。加速器40例如可以包括诸如图形处理

器(GPU)和/或人工智能(AI)加速器等用于对图形、视频等数据进行加速处理的装置或芯片。在一个实施例中,加速器40可以是GPU。在另一实施例中,加速器40可以是AI芯片。设备存储器50例如可以是诸如DRAM之类的位于加速器40外部的易失性存储器。在本公开中,设备存储器50也被称为片外存储器,即,位于加速器40的芯片外部的存储器。相对而言,加速器40的芯片内部也具有易失性存储器,例如一级(L1)高速缓存(cache)以及可选的二级(L2)高速缓存。这将在下文结合本公开的一些实施例具体描述。虽然在图1中示出了本公开的多个实施例能够在其中实现的一种示例环境100,但是本公开不限于此。本公开的一些实施例也可以在诸如ARM架构和RISC-V架构之类的具有诸如GPU之类的加速器的一些应用环境中使用。

[0057] 图2示出了根据本公开的一个实施例的加速器200的示意框图。加速器200例如可以是图1中加速器40的芯片的一种具体实现方式。加速器200例如是诸如GPU之类的加速器芯片。在一个实施例中,加速器200包括流处理器(SP)210、页表装置220、处理引擎(PE)单元230、直接存储器访问(DMA)控制器240、L1高速缓存(cache)260和L2高速缓存250。

[0058] 加速器200由诸如CPU 20之类的主机设备控制,并且接收来自CPU 20的指令。SP 210对来自CPU 20的指令进行分析,并且将经分析的操作指派给PE单元230、页表装置220和DMA控制器240进行处理。页表装置220用于管理加速器200的片上虚拟存储。在本公开中,L2高速缓存250和诸如图1中的设备存储器50之类的片外存储器构成虚拟存储系统。页表装置220由SP 210、PE单元230和DMA控制器240共同维护。

[0059] PE单元230包括多个处理引擎(processing engine,PE)PE_1、PE_2……PE_N,其中N表示大于1的整数。PE单元230中的每个PE可以是单指令多线程(SIMT)装置。在PE中,每个线程可以具有自己的寄存器堆(register file),并且每个PE的所有线程还共享一个统一寄存器堆(uniform register file)。多个PE可以并行地执行相同或不同的处理工作,可以并行地进行下文所述的地址转换和存储器中目标数据的访问,从而减少处理时间。可以理解,多个PE处理的目标元素并不相同,并且目标元素所在的段、页、缓存行和元素的属性、尺寸、维度排序等可以有所不同,如下文具体描述。

[0060] 在一个实施例中,目标元素的逻辑地址可以表示为seg:RF:imm,其中seg表示段基址寄存器,RF表示偏移寄存器,imm表示偏移立即数。从张量角度而言,逻辑地址可以包括目标元素在第一段张量中各维上的基准数据和偏移数据。偏移数据表示目标元素在第一段的多个维度中的各维上的偏移量,段基准数据为段起始点的地址。

[0061] 在一个实施例中,第一段包括至少一个页,加速器200可以至少根据目标元素页的各维的尺寸,将逻辑地址转换为线性地址。线性地址包括目标元素页的一维页标识和目标元素在目标元素页内的一维偏移值。具体而言,加速器200可以根据第一段内各维上页的页尺寸得到目标元素在各维上所处的页序号偏移,由此获得目标元素所处的页的一维标识。例如,目标元素位于图3中的张量的最上层,通过上述方式可以确定目标元素的页标识为P[1]。

[0062] 此外,加速器还可以得到目标元素在该页内部各维上的相对偏移量,并以此为基础,确定目标元素相对于页的起始位置的一维线性偏移量。页的一维标识以及页内的一维线性偏移量共同构成目标元素的线性地址。

[0063] 加速器200根据针对目标元素页的页表项,将线性地址转换为物理地址,页表项包

括至少一个页中的每个页的页物理地址。具体而言，在一个实施例中，加速器200在获取目标元素的页标识之后，可以根据页标识查找页表装置220中对应的项，获取页的物理地址。该物理地址加上目标元素在目标元素页的一维线性偏移量即为目标元素的物理地址。该物理地址可以表示片外的设备存储器50或片上的存储器，例如L2高速缓存250上的目标元素的存储地址。备选地，目标元素页的页表项也可以存储相对于其它页的物理地址，并且基于目标元素页相对于其它页的偏移、其它页的物理地址和一维线性偏移量来获得目标元素的物理地址。

[0064] 除了物理地址之外，页表项还可以包括其它属性，例如状态，用于表示页是否加载完毕，即是否可用。本公开对此不进行限制。虽然在此示出了地址的二级转换，但是本公开不限于此。备选地，也可以经过更多级转换。例如，分级计算页偏移、缓存行偏移、元素偏移，并且依次与物理地址相加以得到目标元素的最终的物理地址。

[0065] 在一个实施例中，加速器200将多个页中的第一页从片外的存储器移入片上的存储器，并且建立与第一页对应的第一页表项，第一页表项存储第一页在存储器中的物理地址。如果将多个页中的第一页从存储器移入片外存储器，则加速器200可以删除与第一页对应的第一页表项。

[0066] 加速器将第一段S1中的目标元素的逻辑地址转换为在片上虚拟存储器中的物理地址。片上虚拟存储器可以包括片上的L2高速缓存250和片外的设备存储器50。逻辑地址包括第一段在张量中的段基准数据和偏移数据，段基准数据和偏移数据分别表示目标元素在第一段的多个维度中的各维上的基址和偏移量。

[0067] 每个线程可以在自己的寄存器堆与存储器子系统之间做线程级的数据交换。每个线程有自己的算数逻辑执行单元并使用自己的存储地址，其采用典型的寄存器存取架构(load-store architecture)。每个执行单元包括一个支持多种数据类型的浮点/定点单元以及一个算数逻辑单元。

[0068] 大多数的指令执行算数和逻辑运算，例如，浮点和定点数的加、减、乘、除，或者逻辑与、或、非等。操作数来自于寄存器。存储器读写指令可以提供寄存器与片上/片外存储器之间的数据交换。一般地，PE中所有的执行单元可以同步地执行相同指令。通过使用谓词(predicate)寄存器，可以屏蔽部分执行单元，从而实现分支指令的功能。

[0069] 在一个实施例中，图2的加速器200可以例如执行如下操作：1)组建页表项内容和初始状态；2)将诸如图1中的设备存储器50之类的片外存储器上的数据搬运至片上存储器，例如L2高速缓存250；3)启动和执行程序；4)定义各个段并对张量以及存储的属性进行描述；

[0070] 5)在程序执行完成时，将执行结果的数据写入至片外存储器。

[0071] 可以理解，在公开的实施例中，加速器200所处理的数据主要针对多维张量。例如，在一个实施例中，张量可以是四维张量，其具有四个维度D1、D2、D3和D4，并且张量在各维上的尺寸可以不同。在另一些实施例中，张量可以是一维、二维、三维或更多维张量，本公开对此不进行限制。

[0072] 此外，在本公开的实施例中，张量内部可以支持诸如uint8、int8、bfloat16、float16、uint16、int16、float32、int32、uint32以及其他自定义元素类型，本公开对此也不进行限制。对于张量的寻址而言，其以元素为基本单位。例如，如果元素类型为int8，则元

素以字节为单位。再例如，如果元素类型为int16，则寻址基本单位为双字节，依此类推。

[0073] 在一些情形中，张量所包含的数据量可能较大，而L2高速缓存250的容量有限，因此无法将张量整体加载至片上的L2高速缓存250。在本公开的一些实施例中，为了便于张量的并行处理，可以将张量划分为至少一个段。在张量仅包括一个段的情形下，张量即为段。而在张量包括多个段的情形下，段为张量的一部分。CPU 20可以通过指令指定段的各个部分由哪个PE进行处理。

[0074] 图3示出了根据本公开的一个实施例的三维张量300的示意框图。三维张量300具有三个维度D1、D2和D3，并且包括第一段S1、第二段S2和第三段S3。CPU 20可以指定段S1的张量元素由PE_1、PE_2、PE_3、PE_4、PE_5、PE_6、PE_7和PE_8处理。此外，CPU20还指定了第二段S2的张量元素由PE_1-PE_4处理。在本公开的实施例中，每个段所具有的尺寸可以不同，因此编程人员可以基于设计需要灵活配置段。实际上，页的划分可以在任意一个或多个维上实施，并且各维上划分的页数是相互独立的。

[0075] 在一个实施例中，可以将张量数据存储于片上的高速存储器，例如L2高速缓存250。但由于片上的高速存储器的容量较少，因此在张量规模较大时，编程人员可以将张量划分为多个段，每个段描述张量一部分。核心程序(kernel)可以分多次启动，每次由DMA控制器240提前将张量的一个段由片外存储搬运到片内存储，并供kernel操作使用。在多次启动kernel后，张量包含的所有段均被处理，整个运行过程结束。当片上的高速存储器足以容纳kernel所要访问的所有张量时，一个张量仅需要一个段描述即可，kernel也只需要启动一次。

[0076] 进一步地，在本公开的一些实施例中，在一个段内，还可以设置至少一个页以进一步细分张量。例如，在第一段S1中，具有4个页P[1]、P[2]、P[3]和P[4]。第二段S2仅具有一个页。在本公开的实施例中，每个段所具有的页的数目可以不同，因此编程人员可以基于设计需要灵活配置段内页的尺寸。例如，将页配置为适于整体存入L2高速缓存250。

[0077] 如上所述，当对张量寻址时，最小的寻址单元是以元素为单元。一个页通常可以包括多个元素。目标元素所在的页在本文中被称为“目标元素页”。在本公开的一些实施例中，页可以包括多个缓存行。目标元素页可以位于L2高速缓存250中时，如果PE经由L1高速缓存260读取目标元素，则L2高速缓存250需要将L2高速缓存250中的包括目标元素在内的一小部分的物理地址连续的数据整体传输至L1高速缓存260。这一小部分数据也被称为缓存行(cache line)数据，而这种缓存机制基于空间邻近性原理。PE从L1高速缓存260读取数据仅需几个时钟周期，而L1高速缓存260从L2高速缓存250读取数据可能需要几十个甚至上百个时钟周期。因此，期望减少L1高速缓存260从L2高速缓存250读取数据的次数。虽然在此以“缓存行”来描述从L2高速缓存250到L1高速缓存260的最小传输数据单位，但在本公开中，这部分数据可以并不必然按行或列排列，一个“缓存行”里面的数据分布在多个维上，且各维上分布的数据尺寸不限于1。PE对一个段内的数据进行并行处理，PE的分配在数据的逻辑地址空间展开，独立于段的物理存储结构，具体如下文描述。

[0078] 在图3中，第一页P[1]中的第一组缓存行被指定由PE_1处理，第二组缓存行被指定由PE_2处理。虽然在此以顺序示出了张量由多个PE依序处理，但是可以理解张量数据的处理独立于PE的顺序，本公开对此不进行限制。例如图3中的PE_2表示部分的张量数据可以由PE_M处理，其中M表示不大于N的任意整数。

[0079] 图4示出了根据本公开的一个实施例的图像数据400的页分配示意图。图像数据是典型的二维张量。在一个实施例中，图像数据400例如为8*8像素。换言之，图像数据400在第一维D1具有8个像素，并且在第二维D2也具有8个像素。因此，图像数据400具有像素P00、P01……P77。在图4的实施例中，图像数据400仅具有一个段，但是按两个维度分为4个页P[1]、P[2]、P[3]和P[4]。4个页可以按第二维D2划分以分配给PE_1和PE_2处理，也可以按第一维D1划分以分配给PE_1和PE_2处理。此外，还可以按对角线划分。本公开对此不进行限制。

[0080] 图5示出了根据本公开的一个实施例的矩阵乘法500的示意图。张量通常可以包括一个或多个维度。二维张量可以被认为是矩阵。在一些情形下，可能需要对两个二维矩阵进行矩阵乘法以获得乘积矩阵。在本公开中，对于矩阵乘法C＝A×B，矩阵C表示乘积矩阵，矩阵A表示第一因数矩阵，并且矩阵B表示第二因数矩阵。在图5中，第一因数矩阵A 502与第二因数矩阵B 504相乘，可以获得乘积矩阵C 506。在本公开中，“点积运算”可以包括对应矩阵元素的乘法操作以及可选的乘积相加操作。具体而言，第一因数矩阵502可以是m×k矩阵，第二因数矩阵504可以是k×n矩阵，其中m，k和n均表示正整数。根据矩阵乘法的规则，乘积矩阵因此是m×n矩阵。由此可见，第一因数矩阵502包括m行和k列，第二因数矩阵504包括k行和n列，并且乘积矩阵因此包括m行和n列。

[0081] 在进行矩阵乘法时，可以将第一行A[1][1]…A[1][k]与B[1][1]…B[k][1]进行点积运算以得到C[1][1]。具体而言，C[1][1]可以通过下面的式子(1)表示：

[0082] C[1][1]＝A[1][1]×B[1][1]+ A[1][2]×B[2][1]…+ A[1][k]×B[k][1] (1)

[0083] 类似地，可以进行点积运算以得到C[m][1]和C[m][n]。C[m][1]和C[m][n]可以通过下面的式子(2)和(3)表示：

[0084] C[m][1]＝A[m][1]×B[1][1]+ A[m][2]×B[2][1]…+ A[m][k]×B[k][1] (2)

[0085] C[m][n]＝A[m][1]×B[1][n]+ A[m][2]×B[2][n]…+ A[m][k]×B[k][n] (3)

[0086] 可以看出，矩阵C包括了m×n个矩阵元素，并且每个矩阵元素是由k个乘积结果相加而成。在本公开中，针对上述的乘积矩阵C＝A×B，乘积结果表示矩阵A的一个矩阵元素和矩阵B中的一个矩阵元素相乘的结果，而点积结果表示矩阵A的多个矩阵元素和矩阵B中的相应多个矩阵元素分别相乘并且多个乘积结果相加得到的结果。

[0087] 图6示出了根据本公开的一个实施例的矩阵乘法600的示意图。在一个实施例中，乘积矩阵C 602可以包括m行和n列，并且每行对应于一个线程。每个线程包括n个寄存器，以用于存储每行的n个点积结果。在PE执行时，m个线程可以并行执行以提高执行效率。在具体执行过程中，可以首先将与矩阵C对应的所有寄存器初始化为0。以C[1][1]为例，如上面的式子(1)所示，C[1][1]的计算包括k次乘法计算以及k-1次加法运算(实际上相当于k次累加，因为矩阵元素被初始化为0，第一个乘积元素相当于与0进行累加)。然后依次计算，例如第一个线程首先计算矩阵元素C[1][1]的第一个乘积结果A[1][1]×B[1][1]，第二个线程并行地首先计算矩阵元素C[2][1]的第一个乘积结果A[2][1]×B[1][1]，以此类推。即，m个线程都先计算各自对应的矩阵C的一个行的第一矩阵元素的第一个乘积结果。可以理解，此时既未得到乘积矩阵C 602的第一列的完整结果，也未开展乘积矩阵C 602的各行除第一列外的其他列的计算。

[0088] 第一个线程然后计算矩阵第二列元素C[1][2]的第一个乘积结果A[1][1]×B[1]

[2],第二个线程并行地计算矩阵元素C[2][2]的第一个乘积结果A[2][1]×B[1][2],以此类推。即,m个线程计算各自对应的矩阵C的一个行的第二矩阵元素的第一个乘积结果。此时未得到乘积矩阵C 602的第一、二列的完整结果,也未开展乘积矩阵C 602的各行除第一、二列外的其他列的计算。M个线程并行计算到第n轮后,得到乘积矩阵C602的各行所有列矩阵元素的第一个乘积结果。第一个线程然后计算矩阵元素C[1][1]的第二个乘积结果A[1][2]×B[2][1]并且将其与第一个乘积A[1][1]×B[1][1]相加,第二个线程并行地首先计算矩阵元素C[2][1]的第二个乘积结果A[2][2]×B[2][1]并且将其与第一个乘积A[2][1]×B[1][1]相加,以此类推,M个线程并行计算到第n轮后,矩阵C 602的所有列被计算完成。即,m个线程都先计算各自对应的矩阵C的一个行的各元素的第二个乘积与第一乘积相加的结果。

[0089] 以此类推,直至计算完成各个矩阵元素的第k个乘积结果,并且将其分别与前k-1个乘积结果之和相加,以获得最终的矩阵C 604。换言之,在计算过程中,对于矩阵C 604实际上包括k轮计算。每一轮都计算矩阵C的各个矩阵元素的一部分,并且将计算结果与先前轮次的计算结果累加在相应寄存器中。如图6所示,矩阵C 602的每个矩阵元素具有相同颜色图案,这表明每个矩阵元素都被计算了相同数目的轮次的乘积累加。矩阵C 604的每个矩阵元素则是经过k轮累加之后获得的最终结果,因此每个矩阵元素的颜色相比于矩阵C 602的颜色更深。

[0090] 虽然在图6的实施例中,每次仅计算一个乘积结果并且将其与先前的结果累加在寄存器中,但是这仅是示意而非对本公开的范围进行限制。在另一些实施例中,可以将每轮计算多个乘积结果并且进行累加。例如,可以将k维分为s段,每次计算s段内的乘积结果的累加。例如,在s=k/2的情形下,针对C[1][1],第一轮计算可以计算A[1][1]×B[1][1]+A[1][2]×B[2][1]。在执行s轮之后,可以获得C[1][1]的完整值。这样,可以基于PE单元的计算资源的分配情况,更为灵活地使用这些计算资源,从而给编程人员赋予更大的编程灵活性。

[0091] 图7示出了根据本公开的另一实施例的矩阵乘法700的示意图。与图6不同,在图7中,多个线程可以并行先计算完成矩阵C的各个矩阵元素的所有乘积结果的累加,然后按列计算矩阵C的下一列的矩阵元素。如图7所示,矩阵C 702的第一列的矩阵元素具有比第n列更深的颜色,这表明第一列的矩阵元素都被计算了相同数目的轮次的乘积累加,而最后一列的矩阵元素此时并未被计算,例如仍为初始值0。矩阵C 704的每个矩阵元素则是经过k轮累加之后获得的最终结果,其中矩阵C 704的第一列的矩阵元素的颜色与矩阵C 702的第一列的矩阵元素的颜色相同,这表明矩阵C 702的第一列首先被计算完成,然后才进行下一轮的计算。与图6的实施例相似,也可以将k维分为s段,每次计算s段内的乘积结果的累加。

[0092] 虽然在图6和图7的实施例中,乘积矩阵C的每个行都使用一个线程执行矩阵计算而得到,但是这仅是示意,而非对本公开的范围进行限制。在线程数目显著大于矩阵行的数目时,例如线程数目为乘积矩阵行数数目的2倍、3倍或更多倍以上时,可以针对乘积矩阵C的每个行,使用2个、3个或更多个线程去计算得到乘积矩阵C,具体如下文所述。

[0093] 由于乘积矩阵C的一行可以由一个或多个线程去执行矩阵计算而得到,因此编程人员可以根据矩阵乘法中的第一因数矩阵A、第二因数矩阵B以及由此得到的乘积矩阵C的行数目和列数目而灵活分配线程。具体而言,在一些实施例中,可以在张量乘法指令中,给

各个线程分配相应的第一因数矩阵A、第二因数矩阵B和乘积矩阵C的相关信息来将矩阵乘法的一部分任务，以灵活和高效地利用PE单元中的计算资源。上面在图5-图7中描述了矩阵乘法的一般性概念，下面将结合图8来具体描述矩阵乘法的一些实施例。

[0094] 图8示出了根据本公开的一个实施例的由加速器执行的方法800的示意流程图。方法800用于执行如上结合图5-图7所示的矩阵乘法。在802，接收针对加速器的第一线程集的第一张量乘法指令，所述第一张量乘法指令包括针对所述第一线程集的第一线程指示、针对第一张量的第一因数寄存器表示、针对第二张量的存储器逻辑地址、以及针对第三张量的第一乘积寄存器表示。在一个实施例中，电子设备可以有两个线程集，其中第一线程集用于将矩阵B的数据广播至第二线程集中的线程的计算单元。例如，第一线程集将与所述存储器逻辑地址对应的所述第二因数集由以广播的形式并行提供给第二线程集中的全部线程或部分线程。换言之，第一线程集被配置用于广播矩阵B的数据，而第一线程集被配置为响应于接收到矩阵A的数据来执行A×B。第二线程集中的每个线程包括第一组寄存器和第二组寄存器，其中第一组寄存器用于存储第一因数矩阵的一行中数据的至少一部分，第二组寄存器用于存储乘积矩阵中的一行的数据。

[0095] 第一张量乘法指令的一个示意性示例例如是@p1,mm.R0,ur4:rf290:0x00,R256，其中@p1表示与第一线程相关联的保护谓词操作数。@p1例如可以是第一线程的布尔谓词变量。如果谓词值为假，则不执行该条指令的数据加载操作。如果谓词值为真，则以ur4:rf290:0x00正常访问片上存储器，例如L1高速缓存260、L2高速缓存250或经由DMA 240控制的DDR(Double Data Rate)存储器之类的动态随机存取存储器(dynamic random access memory,DRAM)，并第一线程集将得到的数据内容广播至第二线程集中的所有线程。换言之，可以提供针对各线程的执行条件，对于不满足执行条件的线程，其访存被视作超出张量地址范围而被忽略，或放弃第二线程集相应线程要执行的张量乘法操作。R0表示用于存储乘积矩阵C中的一行的各个乘积元素的第二组寄存器中的起始寄存器，例如寄存器R0-R255用于存储乘积矩阵C中的一行的各个乘积元素。ur4:rf290:0x00则表示第二因数矩阵的逻辑地址，例如是前面所述的目标元素的逻辑地址seg:RF:imm的一个具体示例。R256表示第一组寄存器中的起始寄存器，第一组寄存器用于存储第一因数矩阵中的一行中的、在一轮点积(矩阵A和矩阵B中的相应元素的乘法和累加)运算中所涉及的相关矩阵元素。在一个实施例中，第一组寄存器和第二组寄存器都位于相同的线程内，这样可以减少在计算过程中数据的传输的功耗和时间。

[0096] 可以理解，第一乘积寄存器表示可以对应于一个或多个乘积寄存器。一个或多个乘积寄存器的数目与合并计算模式以及所述第二张量的列数相关，如下文详述。不同线程的乘积寄存器构成结果张量，结果张量的行数与第一张量的行数相同，结果张量的列数与第二张量的列数相同。例如，256个线程可以构成具有256行的结果张量。每个线程的乘积寄存器文件包括结果张量每行的部分或全部。例如，每个线程的乘积寄存器文件可以对应于结果张量的一行。在合并计算模式下，每个线程的乘积寄存器可以对应于结果张量的一行的一部分。

[0097] 此外，可以理解，第二线程集中的线程内的乘积寄存器的数目是可变的。乘积寄存器的数目取决于第一张量乘法指令的执行条件。执行条件确定对所述第二张量中的列的访问。例如，在一些情形下，第二线程集中的线程内的全部乘积寄存器可以仅一部分被使用。

在另一些情形下，第二线程集中的线程内的全部乘积寄存器的另一部分或全部被使用。如果第二张量中的第一列未被访问，则第二张量中的第一列不参与矩阵乘法计算。

[0098] 在一个具体实现方式中，第一张量乘法指令可以被发射两次或者更多次。在第一次发射中，第一张量指令被发射到存储系统，矩阵乘法(matrix multiply,mm)指令可以从加速器200的高速缓存或指令段中被取出，并且被送至加速器200的流水线单元，经过译码之后作为常规存取指令发射，其存取地址为诸如ur4:rf290:0x00之类的seg:RF:imm。换言之，第一张量乘法指令第一次以存储指令的方式被发射，以用于获取所述第二张量中的列数据或者行数据。

[0099] 响应于获取到第二张量中的列数据或者行数据，并且第一张量的数据已被存储在第一因数寄存器中，第一张量乘法指令以数学计算指令的方式被二次或多次发射，以用于执行所述第三张量的行内的各列结果的计算。

[0100] 加速器200可以读取矩阵C和矩阵A对应的数据块寄存器，例如R0-R255以及R256-R257，然后读取第一次发射过程中得到的第二因数矩阵B的数据块并且执行点积运算，并且将临时计算结果写入相应的寄存器，例如R0-R255中一个寄存器。这样，在矩阵乘法执行过程中，第一线程的执行单元可以仅从第一组寄存器读取第一因数矩阵的一行中的数据一次，并且在后续点积运算过程中重复使用。可以理解，在一些情形下，针对第三张量的乘积寄存器使用范围可能超出单个线程内的寄存器文件的范围。例如，数据块寄存器R0-R255不足以存储第三张量中的一行乘积数据，例如第三张量的一行乘积数据需要300个数据寄存器来存储。在一个实施例中，加速器200可以基于第一乘积寄存器表示，确定针对第三张量的乘积寄存器使用范围是否超出单个线程内的寄存器文件的范围。如果确定针对第三张量的乘积寄存器使用范围超出单个线程内的寄存器文件的范围，则忽略超出寄存器文件范围的计算操作或访存操作并且报错。

[0101] 在一些实施例中，在进行再次发射之前，需要检查第一因数寄存器的就绪状态，具体检查其对应的令牌(token)状态，如果token状态表征第一因数就绪，则以数学计算指令方式发射，否则阻塞发射队列，直至第一因数寄存器就绪。具体而言，在进行二次或多次发射之前，加速器200可以检查第一因数寄存器的对应的令牌状态。如果令牌状态表示所述第一张量的数据已被存储在所述第一因数寄存器中，则以数学计算指令方式发射，否则阻塞发射队列，直至所述第一张量的数据已被存储在所述第一因数寄存器中。

[0102] 由于每个线程在执行并行的mm计算都涉及基本上相同的第二因数矩阵B的矩阵元素数据块，因此第二因数矩阵B的每一段数据块都被广播到所有线程去并行执行。在一个实施例中，可以n个步骤来完成一段数据的计算任务。计算从第二因数矩阵B和乘积矩阵C的第0列开始，每次向后移动一列，直至所有列循环完毕。每个线程都可以为mm指令指定独立的列地址，每个列取回来的数据都广播到所有的线程做计算。

[0103] 在一个实施例中，第二因数矩阵B的一列中的数据可以来自于L1 cache、L2 cache或片外存储器。这样，第二因数矩阵的一列中的数据可以被并行广播至多个(例如与第一因数矩阵的行相同数目或其一半数目)线程中的执行单元，并且重复使用。以此方式，可以减少数据在不同存储装置之间的传送，从而减少矩阵乘法计算过程中因数据传输引起的时间。

[0104] 在804，第一线程集基于针对第二张量的存储器逻辑地址将第二张量中的第二因

数集广播至第二线程集，如上所述。

[0105] 在806，第二线程集中的第一线程基于第一因数寄存器表示将第一张量中的第一行中的第一因数集和第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。点积预算可以包括乘法运算和加法运算。第一因数寄存器表示例如是R256，并且存储器逻辑地址是诸如ur4:rf290:0x00之类的seg:RF:imm。在一些实施例中，第二线程集中的每个线程内的寄存器的个数是可变的，具体由张量乘法指令的执行条件控制，该条件控制对第二张量中各列的访问，如果某列未被访问，则该列不参与矩阵乘法计算，因此与该列对应的乘积寄存器不存在。

[0106] 可以理解，在本公开的实施例中，矩阵乘法并非一次完成，而是基于寄存器的大小，第一因数矩阵A、第二因数矩阵B和乘积矩阵C中的矩阵元素的类型以及加速器200中的计算单元的计算能力等因素综合考虑而多次执行来完成。换言之，单个线程内的第一因数寄存器集包括第一张量单行内数据的至少部分数据，第一因数寄存器集包括一个或多个寄存器，其具体数目可以由单轮张量乘法指令支持的数据长度决定，诸如可以是2个寄存器，每个寄存器包括一个或者多个数据元素；例如对于int8数据类型，2个寄存器包括8个数据元素。参与张量乘法的线程与第一张量的行数成正比关系。例如第一张量的行数可以是256，参与张量乘法的线程的数目也可以是256。

[0107] 在一个实施例中，第一张量乘法指令可以进一步例如是@p1,mm8.sa.ub.R0,ur4:rf290:0x00,R256。其与@p1,mm.R0,ur4:rf290:0x00,R256相同或相似之处在此不再赘述，参见上面相关说明。mm8表示矩阵乘法所涉及的元素的数据类型是8位，sa表示与寄存器R256相关联的第一因数矩阵A中的元素数据是有符号型的int8，ub表示与逻辑地址ur4:rf290:0x00相关联的第二因数矩阵B中的元素数据是无符号型的uint8。可以理解，第一因数矩阵A、第二因数矩阵B和乘积矩阵C中的矩阵元素的类型也可以是其它数据类型，本公开对此不进行限制。

[0108] 由于矩阵乘法涉及多个矩阵元素的多次点积运算，因此在一些实施例中可以分段进行多次运算，并且将多次点积运算的结果累加以获得最终的mm结果。在一个实施例中，例如基于@p1,mm8.sa.sb.R0,ur4:rf290:0x00,R256，可以确定第一因数寄存器表示R256和存储器逻辑地址ur4:rf290:0x00。对于第二线程集中的第一线程而言，第一因数寄存器表示R256和存储器逻辑地址ur4:rf290:0x00例如可以对应于第一组寄存器中的第一寄存器和矩阵B的张量段的基准点的数据块。第一寄存器中存储器了第一因数集，例如A[1][1]，并且矩阵B的张量段的基准点的数据块表示矩阵B的张量段的基准点的数据块，例如B[1][1]。在进行矩阵乘法之后，可以得到乘积矩阵C的第三张量的第一行中的第一点积集A[1][1]×B[1][1]。在另一实施例中，第一因数集可以包括A[1][1]和A[1][2]，第二因数集可以包括B[1][1]和B[2][1]，因此第一点积集可以包括A[1][1]×B[1][1]+A[1][2]×B[2][1]。在又一实施例中，第一因数集可以是A[1][1]、A[1][2]和A[1][3]，第二因数集可以包括B[1][1]、B[2][1]和B[3][1]，因此第一点积集可以包括A[1][1]×B[1][1]+A[1][2]×B[2][1]+A[1][3]×B[3][1]。本公开对于第一因数集、第二因数集和第一点积集的范围不进行限制，该范围可以由编程人员在对矩阵乘法进行编程时基于矩阵元素的数据类型、寄存器容量等因素而灵活配置，例如通过设置张量乘法指令中的数据类型来自动配置。

[0109] 虽然在此以乘积矩阵C中的单个乘积元素C[1][1]的示例进行描述，但是可以理

解,这仅是示意而非对本公开的范围进行限制。在一些实施例中,单个线程可以对乘积矩阵C中的一行中的多个乘积元素进行并行计算。例如第二线程集中的第一线程可以并行计算C[1][1]-C[1][8]的各自的第一点积集A[1][1]×B[1][1]、A[1][1]×B[1][2]、A[1][1]×B[1][3]、A[1][1]×B[1][4]、A[1][1]×B[1][5]、A[1][1]×B[1][6]、A[1][1]×B[1][7]和A[1][1]×B[1][8]。在另一实施例中,第一线程也可以并行计算C[1][1]-C[1][8]的各自的第一点积集A[1][1]×B[1][1]+A[1][2]×B[2][1]、A[1][1]×B[1][2]+A[1][2]×B[2][2]、A[1][1]×B[1][3]+A[1][2]×B[2][3]、A[1][1]×B[1][4]+A[1][2]×B[2][4]、A[1][1]×B[1][5]+A[1][2]×B[2][5]、A[1][1]×B[1][6]+A[1][2]×B[2][6]、A[1][1]×B[1][7]+A[1][2]×B[2][7]和A[1][1]×B[1][8]+A[1][2]×B[2][8]。

[0110] 在808,由第二线程集中的第一线程将第一点积集累加到与第一乘积寄存器表示对应的第一组乘积寄存器中。例如,第一线程可以将上述计算的点积结果累加到对应的第一组乘积寄存器中,例如R0-R7寄存器。与上面类似,第一组乘积寄存器所包括的寄存器的范围可以由mm指令灵活配置。通过将矩阵分解,并且按行分配线程,这样多个线程可以并行处理矩阵张量的多个行,从而加快矩阵乘法的处理效率。此外,由于编程人员在编程时知晓矩阵张量的行列结构以及加速器中的线程状况,因此可以灵活使用线程来并行处理矩阵乘法,从而提高编程的灵活性。

[0111] 在一些实施例中,方法800还包括响应于接收到第二因数集,第二线程集中的第二线程基于第一因数寄存器表示将第一张量的第二行中的第三因数集和第二因数集进行点积运算,以生成第三张量的第二行中的第二点积集;以及由第二线程将第二点积集累加到与第一乘积寄存器表示对应的第二组乘积寄存器中。可以理解,虽然第二线程集中的第一线程和第二线程具有相同的第一张量乘法指令,例如,在一个实施例中,第一张量乘法指令可以表示为@p1,mm8.sa.sb.R0,ur4:rf290:0x00,R256,但是由于可以使用诸如加载指令的其它一些指令将第一张量的第一行数据加载至第一线程,并且将第二行数据加载至第二线程,因此第一线程和第二线程基于已加载的第一张量的数据便可以正确执行点积运算。

[0112] 与第二线程集中的第一线程相同或相似,第二线程集中的第二线程也包括第一组寄存器,例如R256-R257用于存储第一因数矩阵的第二行中的第三因数集,并且还包括第二组寄存器,例如R0-R255以用于存储第三张量的第二行的第二点积集。第一线程和第二线程实际上分别针对第一因数矩阵A的第一行和第二行并且分别针对第一乘积矩阵C的第一行和第二行而进行并行的mm计算,因此通过并行计算,可以大大节省计算时间。此外,由于每个线程和每个矩阵行存在固定的对应关系,因此也可以避免多个线程依据忙碌程度动态分配矩阵乘法计算任务(例如,一个线程可以计算两个矩阵行,而另一线程仅计算一个矩阵行的一部分)所造成的开销。

[0113] 在一些情形下,例如当线程数目远大于乘积矩阵C中的行的数目时,可能会造成部分线程闲置。例如,当PE单元包括64个线程,而乘积矩阵C中的行的数目仅为16时,如果每行仍仅分配一个线程,则会有48个线程被闲置。在此情形下,可以通过在张量乘法指令中,设置第一合并计算模式指示或第二合并计算模式指示来将多个线程(例如第二线程集中的第一线程和第三线程)用于乘积矩阵C中的一行的计算。

[0114] 例如,在一个实施例中,第一张量乘法指令还包括第一合并计算模式指示,例如KA2。KA2表示两个线程参与一个矩阵行的计算。在另一些实施例中,第一合并计算模式指示

可以包括KA1、KA3、KA4等其它指示，区别仅在于KA之后跟随的数字。KA1则表示单个线程参与一个矩阵行的计算，KA3表示三个线程参与一个矩阵行的计算，以此类推。在一些实施例中，在没有第一合并计算模式指示的情形下，可以默认为单个线程执行一个矩阵行的计算。在第一合并计算模式指示为KA2的情形下，第一线程和第三线程接收到的第一张量乘法指令的示意性示例例如可以是@p1,mm8.KA2.sa.sb.R0,ur4:rf290:0x00,R256。可以理解，KA1-KA4仅使用用于表示第一合并计算模式指示的一种实现方式，可以使用其它字符或是其它表示方式来表示第一合并计算模式指示。

[0115] 可以看出，通过增加第一合并计算模式指示KA2，第二线程集中的第一线程和第三线程共同计算乘积矩阵C中的同一行中的乘积元素。例如，第一线程用于计算第一组乘积元素C[1][1]-C[1][127]，而第三线程用于计算第二组乘积元素C[1][128]-C[1][256]，或者第一线程用于计算第一组乘积元素C[1][1]，C[1][3]，C[1][5]…C[1][255]，而第三线程用于计算第二组乘积元素C[1][2]，C[1][4]，C[1][6]…C[1][256]。

[0116] 在此情形下，基于第一合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集，并且将第一点积集累加到第一线程的第二组寄存器中。第三线程基于第一合并计算模式指示和第一因数寄存器表示，将第一因数集和第二张量的第四因数集进行点积运算，以生成第三张量的第一行中的第三点积集，第四因数集不同于第二因数集，第三点积集不同于第一点积集。第三线程还进一步将第三点积集累加到与第一乘积寄存器表示对应的第三组乘积寄存器中，该第三组乘积寄存器位于第三线程中。可以理解，第一合并计算模式指示计算模式指示可以与上面针对图8的实施例结合使用，因此针对图8描述的各个方面在此不再赘述。

[0117] 在另一个实施例中，第一张量乘法指令还包括第二合并计算模式指示，例如KB2。KB2表示两个线程共同参与乘积矩阵中的每个乘积元素的计算。在另一些实施例中，第二合并计算模式指示可以包括KB1、KB3、KB4等其它指示，区别仅在于KB之后跟随的数字。KB1表示单个线程参与参与乘积矩阵中的每个乘积元素的计算，KB3表示三个线程共同参与乘积矩阵中的每个乘积元素的计算，以此类推。在一些实施例中，在没有第二合并计算模式指示的情形下，可以默认为单个线程执行一个矩阵行的计算。在第二合并计算模式指示为KB2的情形下，第二线程集中的第一线程和第四线程所接收到的第一张量乘法指令的示意性示例例如可以是@p1,mm8.KB2.sa.sb.R0,ur4:rf290:0x00,R256。可以理解，KB1-KB4仅使用用于表示第二合并计算模式指示的一种实现方式，可以使用其它字符或是其它表示方式来表示第二合并计算模式指示。

[0118] 可以看出，通过增加第二合并计算模式指示KB2，第二线程集中的第一线程和第四线程共同参与乘积矩阵中的每个乘积元素的计算。具体而言，例如针对点积A[1][1]×B[1][1]+A[1][2]×B[2][1]，第一线程可以计算A[1][1]×B[1][1]，第四线程可以与第一线程并行地计算A[1][2]×B[2][1]，第一线程和第四线程随后加和。例如，第四线程将的乘积发送至第一线程，第一线程执行加法操作以得到点积结果。第一线程将点积结果累加乘积寄存器。对于A[1][1]×B[1][2]+A[1][2]×B[2][2]、A[1][1]×B[1][3]+A[1][2]×B[2][3]、A[1][1]×B[1][4]+A[1][2]×B[2][4]、A[1][1]×B[1][5]+A[1][2]×B[2][5]、A[1][1]×B[1][6]+A[1][2]×B[2][6]、A[1][1]×B[1][7]+A[1][2]×B[2][7]和A[1][1]×B[1][8]+

A[1][2]×B[2][8]，第一线程和第四线程可以类似地操作，以获得第一点积集。在另一个实施例中，可以默认第一线程将乘积发送至第四线程，而由第四线程执行乘积的加法并且将点积结果累加到第四线程的乘积寄存器。

[0119] 再例如，针对点积A[1][1]×B[1][1]+A[1][2]×B[2][1]+A[1][3]×B[3][1]+A[1][4]×B[4][1]，第一线程可以计算A[1][1]×B[1][1]+A[1][2]×B[2][1]，第二线程集中的第四线程可以与第一线程并行地计算A[1][3]×B[3][1]+A[1][4]×B[4][1]，第一线程随后进行加和处理。例如，第四线程将点积发送至第一线程，并且第一线程执行加法操作以得到点积结果。第一线程随后将点积结果累加到乘积寄存器。在另一个实施例中，可以默认第一线程将点积发送至第四线程，而由第四线程执行点积的加法并且将点积结果累加到第四线程的乘积寄存器。

[0120] 在此情形下，第一线程基于第二合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集，并且将第一点积集累加到第一线程的第二组寄存器中。第四线程基于第二合并计算模式指示和第一因数寄存器表示，将第四线程将第一行中的第五因数集和第二张量的第六因数集进行点积运算，以生成第三张量的第一行中的第四点积集，第五因数集不同于第一因数集，第六因数集不同于第二因数集，第四点积集不同于第一点积集。第一线程还进一步将第四点积集累加到与第一乘积寄存器表示对应的第三组乘积寄存器，该第三组乘积寄存器位于第一线程中。可以理解，第二合并计算模式指示计算模式指示可以与上面针对图8的实施例结合使用，因此针对图8描述的各个方面在此不再赘述。

[0121] 此外，在一些情形下，例如线程数目远大于乘积矩阵的行数目时，可以将第一合并计算指示与第二合并计算指示结合使用。即，乘积矩阵不仅每行可以分为不同的部分使用不同的线程组来计算，每行内的每个点积元素也可以由不同的线程来计算。例如，针对C[1][1]-C[1][8]，C[1][1]-C[1][4]可以由第一组线程来计算，而C[1][5]-C[1][8]可以由第二组线程来计算。进一步地，针对每个点积元素，例如C[1][1]＝A[1][1]×B[1][1]+A[1][2]×B[2][1]+A[1][3]×B[3][1]+A[1][4]×B[4][1]+A[1][5]×B[5][1]+A[1][6]×B[6][1]+A[1][7]×B[7][1]+A[1][8]×B[8][1]，其中第一组线程中的第一线程计算A[1][1]×B[1][1]+A[1][2]×B[2][1]+A[1][3]×B[3][1]+A[1][4]×B[4][1]，而第一组线程中的第二线程计算A[1][5]×B[5][1]+A[1][6]×B[6][1]+A[1][7]×B[7][1]+A[1][8]×B[8][1]，以此类推。

[0122] 在矩阵乘法的计算过程中，第二因数矩阵通常按列与第一因数矩阵的行元素进行点积运算。然而，在一些情形下，被存储在诸如DDR之类的存储器中的第二因数矩阵在物理上通常是按行存储。因此，当线程从存储器中读取第二因数矩阵的元素，例如B[1][1]，基于空间邻近性原理，其通常也将该元素在物理上相近的一些元素一次读取到L1高速缓存中，例如B[1][2]、B[1][3]、B[1][4]与B[1][1]被一同读取到L1高速缓存。然而，在做矩阵乘法的过程中，实际上一个线程可能需要的相同列的元素，例如B[1][1]和B[2][1]。这时，则又需要消耗若干个时钟周期从存储器读取B[2][1]以及在本次计算过程中不需要的B[2][2]、B[2][3]和B[2][4]到L1高速缓存中。在常规情形下，B[1][2]、B[1][3]、B[1][4]、B[2][2]、B[2][3]和B[2][4]通常会因L1高速缓存的动态刷新规则而被丢弃。在后续矩阵计算过程中，当需要B[1][2]、B[1][3]、B[1][4]、B[2][2]、B[2][3]或B[2][4]时，线程重新从存储器读取

相应数据到L1高速缓存中。由此可见，这样的多次重复读取极大地浪费了从存储器到L1高速缓存传输数据的时间。

[0123] 在本公开的一些实施例中，针对例如第二因数矩阵B的矩阵元素按行存储的情形，在张量乘法指令中进一步设置了转置指示。在一个实施例中，第一张量乘法指令还包括转置指示。第一张量乘法指令的一个进一步的示意性示例是@p1，mm8.KA1.T1.sa.sb R0,ur4:rf290:0x00,R256，其中T1表示第二因数矩阵B需要被转置。在另一些实施例中，在张量乘法指令不包括转置指示时，可以默认第二因数矩阵B无需被转置。在又一些实施例中，可以在张量乘法指令中使用T0表示第二因数矩阵B无需被转置。

[0124] 第二线程集中的第一线程因此可以基于转置指示和第一因数寄存器表示，将第一张量第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。具体而言，第一线程集基于转置指示和存储器逻辑地址，将第二张量中的多个行的因数加载至高速缓存。例如，第一线程集可以将B[1][1]-B[1][4]、B[2][1]-B[2][4]、B[3][1]-B[3][4]和B[4][1]-B[4][4]均加载进入L1高速缓存。第一线程集继而按列从多个行的因数中选择因数，例如选择B[1][1]、B[2][1]、B[3][1]和B[4][1]，以形成第二因数集并且将其广播至第二线程集。第二线程集继而基于第一因数寄存器表示将第一行中的第一因数集和第二因数集进行点积运算以生成第三张量的第一行中的第一点积集。注意，此时由于存在转置指示T1，因此B[1][2]-B[1][4]、B[2][2]-B[2][4]、B[3][2]-B[3][4]和B[4][2]-B[4][4]被直接保留在高速缓存中，而不被动态刷新掉。这样，第二线程集中的第一线程在执行后续矩阵计算过程中，无需再次从存储器读取B[1][2]-B[1][4]、B[2][2]-B[2][4]、B[3][2]-B[3][4]和B[4][2]-B[4][4]，由此可以大大节省时间。

[0125] 虽然在此以B[1][1]-B[1][4]、B[2][1]-B[2][4]、B[3][1]-B[3][4]和B[4][1]-B[4][4]为例来说明转置指示，但是可以理解这仅是示意。可以用于转置的第二因数矩阵B的范围可以变化，例如，在第二因数矩阵B的行数为诸如256行之类的其它行数目时，可以将全部行的cache line都加载进高速缓存，并且直至cache line中的数据都已经用于矩阵乘法计算之后再从高速缓存释放。这样，可以大大节省反复从存储器读取数据到L1高速缓存所需的时间。

[0126] 在上文中，主要以二维张量的形式来描述根据本公开的实施例的矩阵乘法的原理和示例。但是可以理解，本公开不限于二维张量的形式的矩阵乘法计算，而是可以包括一维张量或更多维张量的乘法或卷积的计算。对于一维张量而言，相当于二维张量中的一个维度为1，因此在此不再赘述。

[0127] 对于三维或更高维度的矩阵计算而言，可以将第一因数矩阵A和第二因数矩阵B中的除k维之外的其它维度进行降维分解以获得等效二维矩阵，k维通常不被分解是因为为了进行矩阵乘法，第一因数矩阵A中的k列和第二因数矩阵B中的k行的数目需要相等。

[0128] 在一个实施例中，假设第一因数张量A是m×x×k的三维张量，并且第二因数张量B是k×n×y×z的四维张量，其中k、m、n、x、y和z均表示正整数。可以将第一因数张量A转换为(m×x,k)形式的二维张量。即，在x维上切割，并且将切割后的x个m×k的二维张量按行拼接，以获得二维等效矩阵A’。在此情形下，可以使用m×x个线程来并行计算。此外，类似地，可以将第二因数张量切割为y×z个k×n的二维矩阵，并且依次按列拼接，以获得二维等效矩阵B’。可以理解，虽然在此以三维张量和四维张量的乘法(卷积)为例来说明矩阵降维，但

是这仅是示意而非对本公开的范围进行限制。其它多维mm的降维可以类似处理，在此不再赘述。降维之后的mm，可以参见前面针对图8的关于mm的具体描述，在此同样不再赘述。

[0129] 图9示出了根据本公开的一个实施例的电子设备900的示意框图。电子设备900可以用于执行图8所示的方法800，因此关于图8描述的各个方面可以选择性适用于电子设备900。电子设备900包括接收单元902、广播单元903、生成单元904和存储单元906。

[0130] 接收单元902被配置为接收针对加速器的第一线程集的第一张量乘法指令，第一张量乘法指令包括针对第一线程集的第一线程指示、针对第一张量的第一因数寄存器表示、针对第二张量的存储器逻辑地址、以及针对第三张量的第一乘积寄存器表示。广播单元903被配置为由第一线程集基于针对第二张量的存储器逻辑地址将第二张量中的第二因数集广播至第二线程集，第二线程集不同于第一线程集。生成单元904被配置为由第二线程集中的第一线程基于第一因数寄存器表示将第一张量中的第一行中的第一因数集和第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。存储单元906被配置为由第一线程将第一点积集累加到与第一乘积寄存器表示对应的第一组乘积寄存器中。通过将矩阵分解，并且按行分配线程，这样多个线程可以并行处理矩阵张量的多个行，从而加快矩阵乘法的处理效率。此外，由于编程人员在编程时知晓矩阵张量的行列结构以及加速器中的线程状况，因此可以灵活使用线程来并行处理矩阵乘法，从而提高编程的灵活性。

[0131] 在一个实施例中，每个线程包括第一组寄存器和第二组寄存器，其中第一组寄存器用于存储第一因数矩阵的一行中数据的至少一部分，第二组寄存器用于存储乘积矩阵中的一行的数据。第二因数矩阵的一列中的数据可以来自于片上存储器、一级高速缓存或片外存储器。这样，在矩阵乘法执行过程中，第一线程的执行单元可以仅从第一组寄存器读取第一因数矩阵的一行中的数据一次，并且在后续点积运算过程中重复使用。此外，第二因数矩阵的一列中的数据可以被并行广播至多个(例如与第一因数矩阵的行相同数目或其一半数目)线程中的执行单元，并且重复使用。以此方式，可以减少数据在不同存储装置之间的传送，从而减少矩阵乘法计算过程中因数据传输引起的时间。

[0132] 在一个实施例中，生成单元904被进一步配置为响应于接收到第二因数集，第二线程集中的第二线程基于第一因数寄存器表示将第一张量的第二行中的第三因数集和第二因数集进行点积运算，以生成第三张量的第二行中的第二点积集。存储单元908被进一步配置为由第二线程将第二点积集累加到与第一乘积寄存器表示对应的第二组乘积寄存器中。

[0133] 在一个实施例中，第一张量乘法指令还包括第一合并计算模式指示。生成单元904还被配置为：基于第一合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0134] 在一个实施例中，生成单元904还被配置为基于第一合并计算模式指示和第一因数寄存器表示，由第一线程集中的第三线程将第一因数集和第二张量的第四因数集进行点积运算，以生成第三张量的第一行中的第三点积集，第四因数集不同于第二因数集，第三点积集不同于第一点积集。存储单元906还被配置为由第三线程将第三点积集累加到与第一乘积寄存器表示对应的第三组乘积寄存器中。

[0135] 在一个实施例中，第一张量乘法指令还包括第二合并计算模式指示。生成单元904还被配置为基于第二合并计算模式指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一

点积集。

[0136] 在一个实施例中，生成单元904还被配置为基于第二合并计算模式指示和第一因数寄存器表示，由第二线程集中的第四线程将第五因数集和第二张量的第六因数集进行点积运算，以生成第三张量的第一行中的第四点积集，第五因数集不同于第一因数集，第六因数集不同于第二因数集，第四点积集不同于第一点积集。存储单元906还被配置为由第四线程将第四点积集累加至与第一乘积寄存器表示对应的第一组乘积寄存器。

[0137] 在一个实施例中，第一张量乘法指令还包括转置指示。生成单元904还被配置为：基于转置指示和第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集。

[0138] 在一个实施例中，生成单元904还被配置为：基于转置指示和存储器逻辑地址，将第二张量中的多个行的因数加载至高速缓存；按列从多个行的因数中选择因数以形成第二因数集；以及基于第一因数寄存器表示，由第一线程将第一行中的第一因数集和第二因数集进行点积运算以生成第三张量的第一行中的第一点积集。在一个实施例中，多个行中未被选择的多个因数被保留在一级高速缓存中，直至上述多个未被选择的因数被选择用于进行矩阵乘法的计算。

[0139] 在一个实施例中，第一线程集将与存储器逻辑地址对应的第二因数集由以广播的形式并行提供给第二线程集中的全部线程。

[0140] 在一个实施例中，存储器逻辑地址包括段基准数据和偏移数据，段基准数据表示第二张量的起始地址，偏移数据表示第二张量在多个维度中的各维上的偏移量。

[0141] 此外，虽然采用特定次序描绘了各操作，但是这应当理解为要求这样操作以所示出的特定次序或以顺序次序执行，或者要求所有图示的操作应被执行以取得期望的结果。在一定环境下，多任务和并行处理可能是有利的。同样地，虽然在上面论述中包含了若干具体实现细节，但是这些不应当被解释为对本公开的范围的限制。在单独的实施例的上下文中描述的某些特征还可以组合地实现在单个实现中。相反地，在单个实现的上下文中描述的各种特征也可以单独地或以任何合适的子组合的方式实现在多个实现中。

[0142] 尽管已经采用特定于结构特征和/或方法逻辑动作的语言描述了本主题，但是应当理解所附权利要求书中所限定的主题未必局限于上面描述的特定特征或动作。相反，上面所描述的特定特征和动作仅仅是实现权利要求书的示例形式。

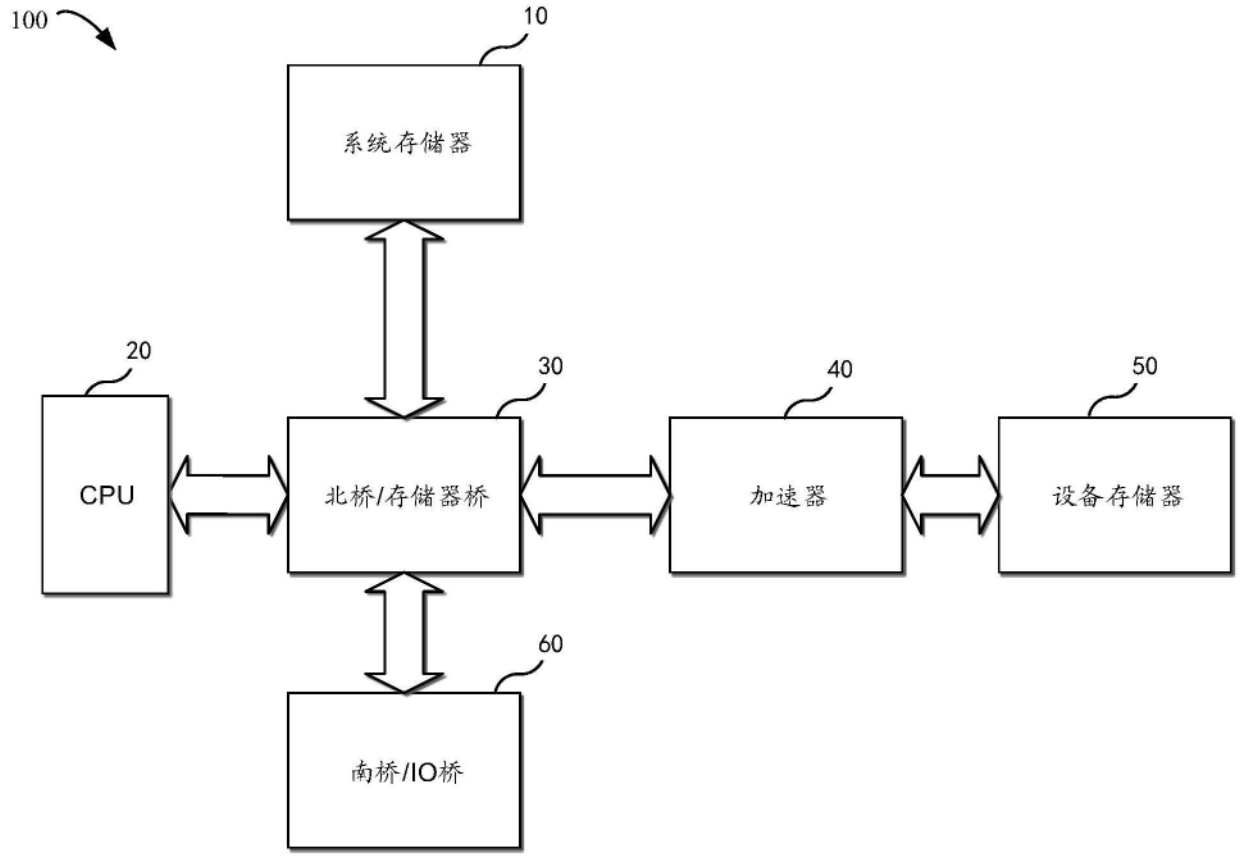


图1

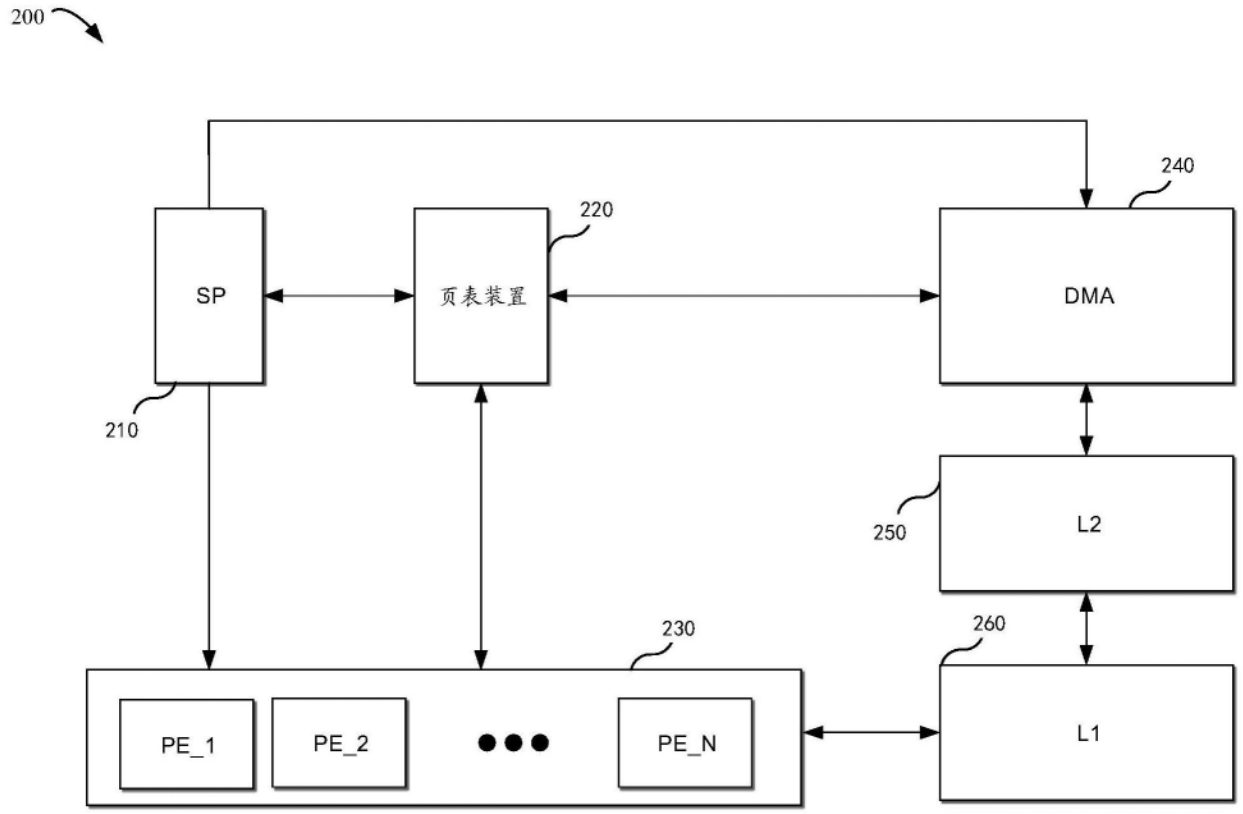


图2

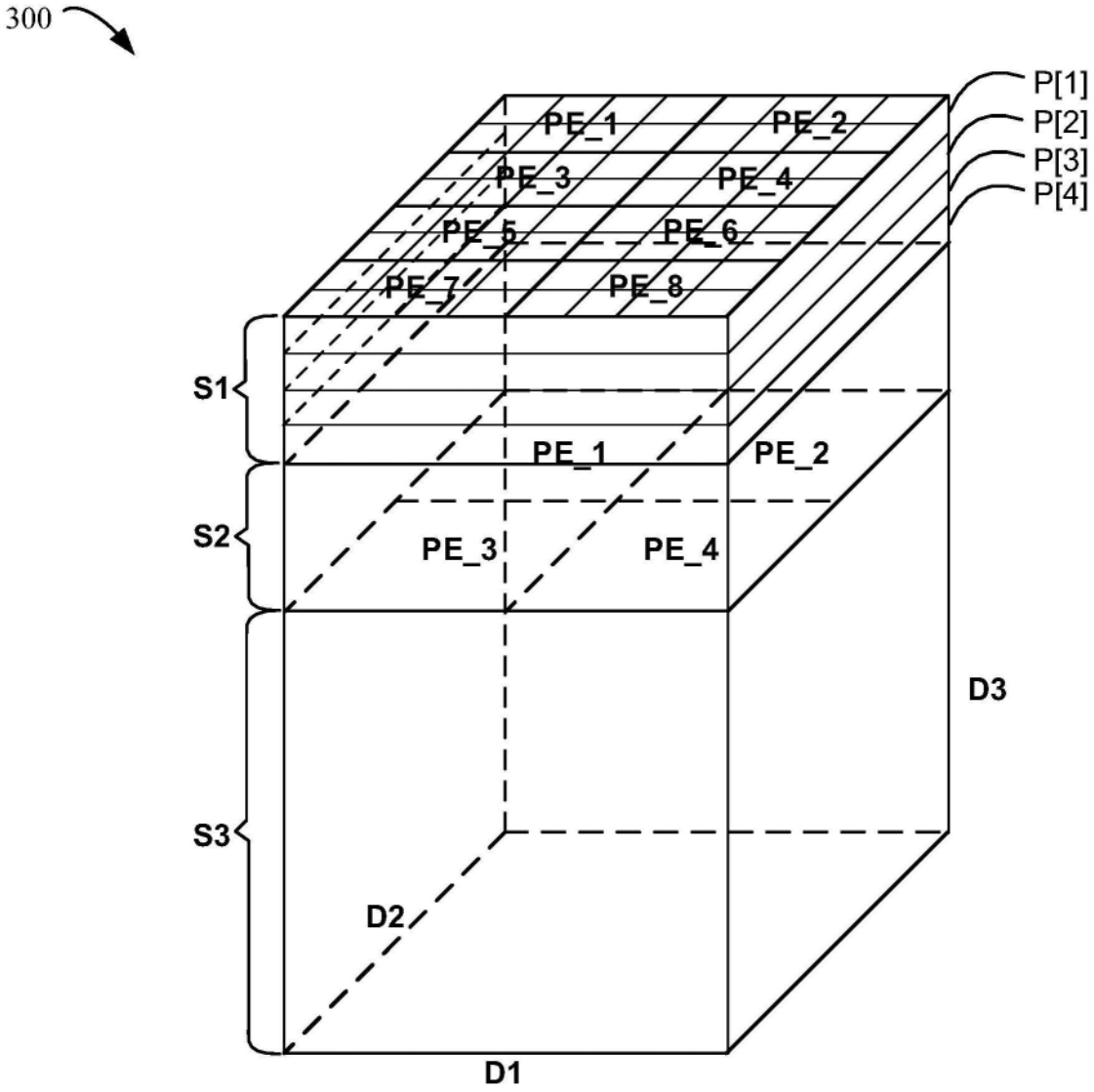


图3

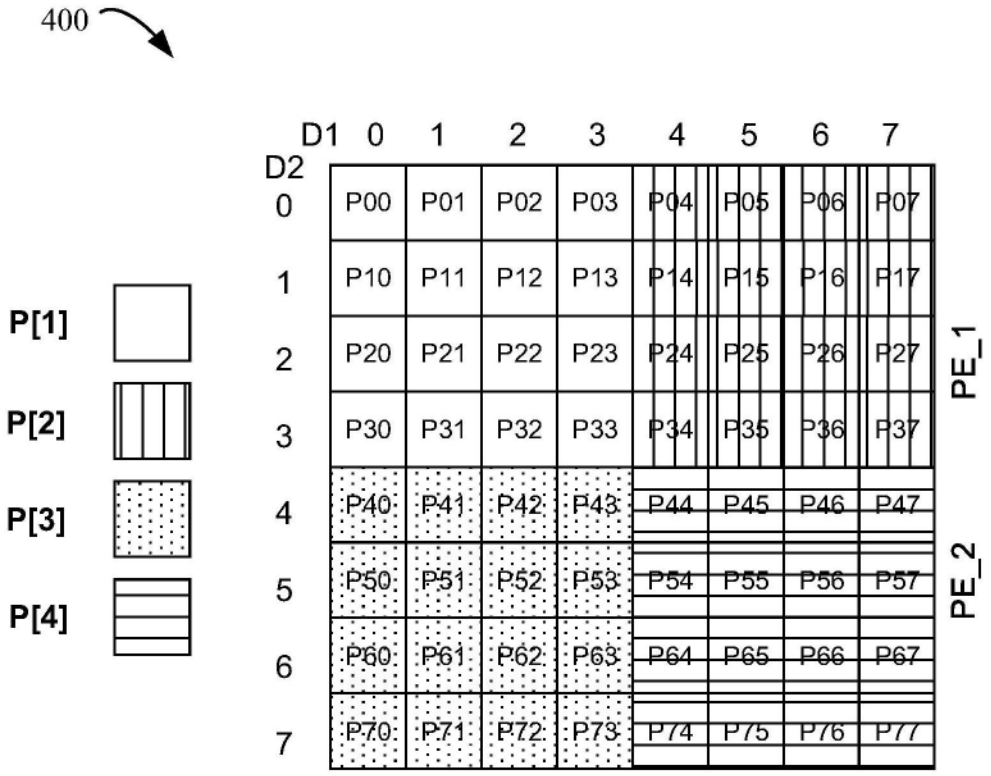

400
P[1]
P[2]
P[3]
P[4]
D1 0 1 2 3 4 5 6 7
D2
0 P00 P01 P02 P03 P04 P05 P06 P07
1 P10 P11 P12 P13 P14 P15 P16 P17
2 P20 P21 P22 P23 P24 P25 P26 P27
3 P30 P31 P32 P33 P34 P35 P36 P37
4 P40 P41 P42 P43 P44 P45 P46 P47
5 P50 P51 P52 P53 P54 P55 P56 P57
6 P60 P61 P62 P63 P64 P65 P66 P67
7 P70 P71 P72 P73 P74 P75 P76 P77
PE_1
PE_2

图4

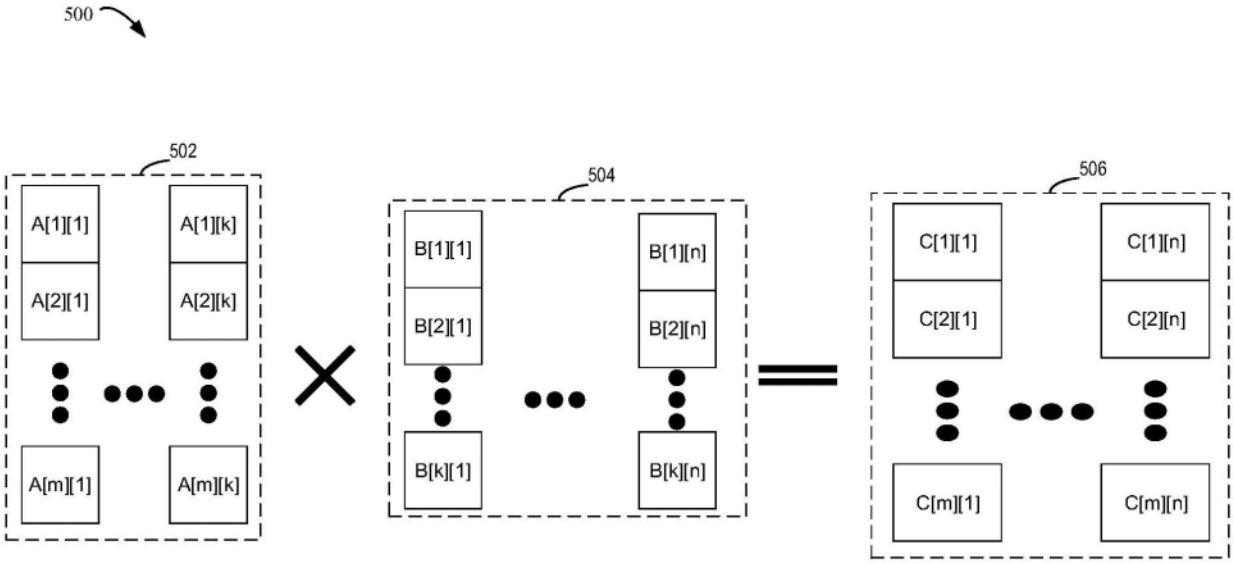

500
502
A[1][1]
A[1][k]
A[2][1]
A[2][k]
A[m][1]
A[m][k]
504
B[1][1]
B[1][n]
B[2][1]
B[2][n]
B[k][1]
B[k][n]
506
C[1][1]
C[1][n]
C[2][1]
C[2][n]
C[m][1]
C[m][n]

图5

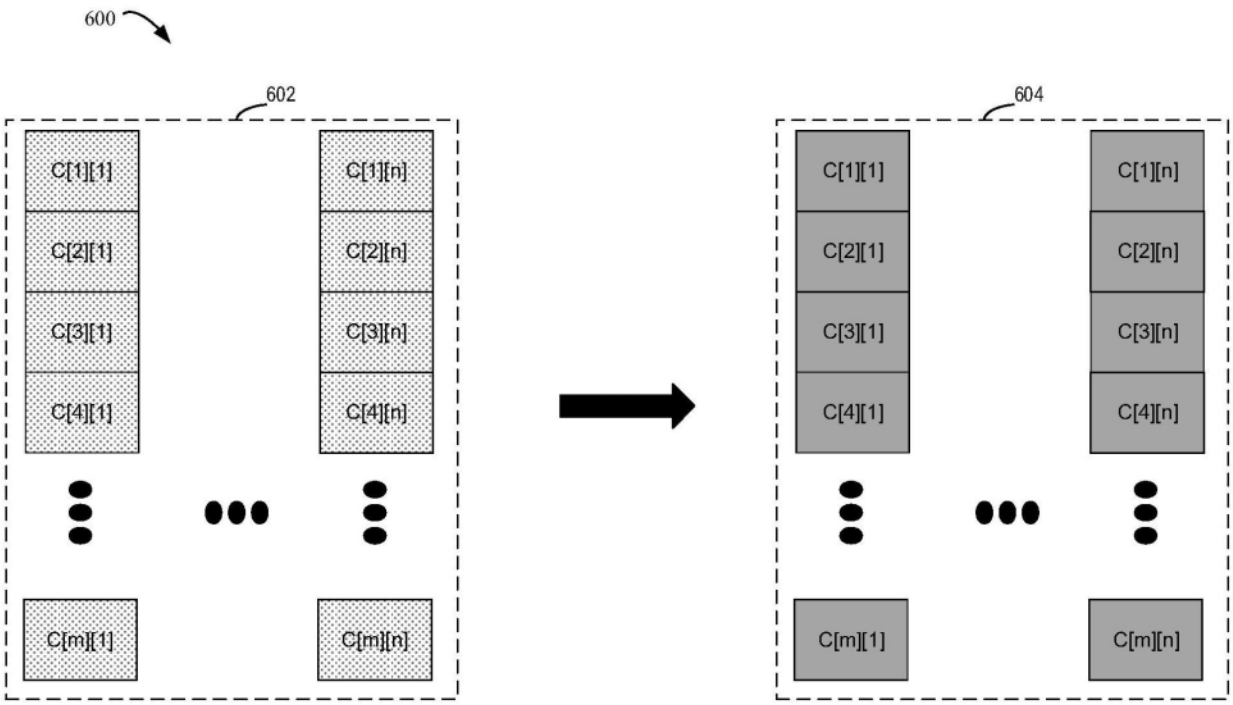

600
602
C[1][1]
C[2][1]
C[3][1]
C[4][1]
C[1][n]
C[2][n]
C[3][n]
C[4][n]
C[m][1]
C[m][n]
604
C[1][1]
C[2][1]
C[3][1]
C[4][1]
C[1][n]
C[2][n]
C[3][n]
C[4][n]
C[m][1]
C[m][n]

图6

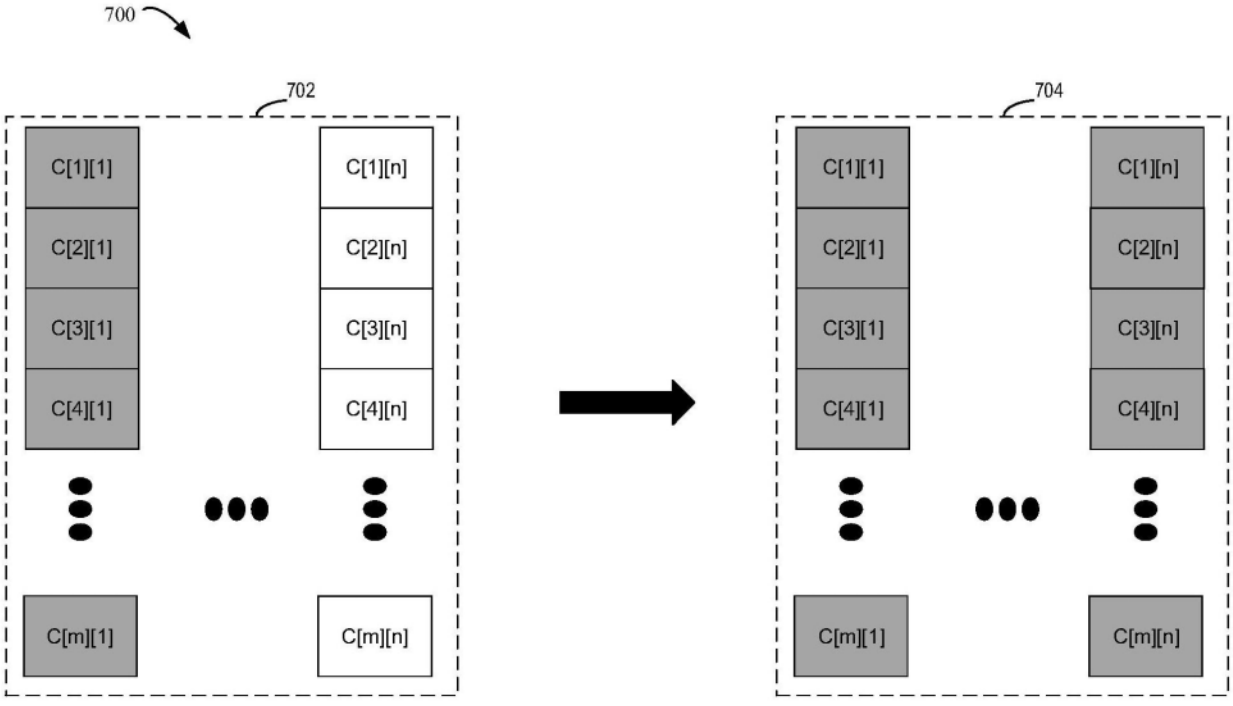

700
702
C[1][1]
C[2][1]
C[3][1]
C[4][1]
C[1][n]
C[2][n]
C[3][n]
C[4][n]
C[m][1]
C[m][n]
704
C[1][1]
C[2][1]
C[3][1]
C[4][1]
C[1][n]
C[2][n]
C[3][n]
C[4][n]
C[m][1]
C[m][n]

图7

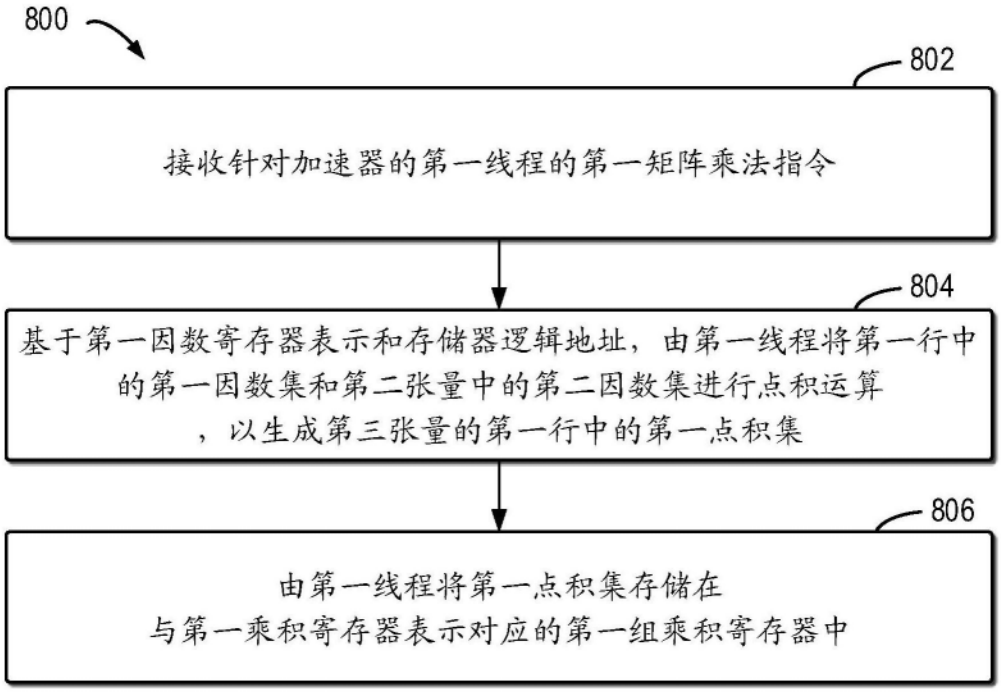

800
802
接收针对加速器的第一线程的第一矩阵乘法指令
804
基于第一因数寄存器表示和存储器逻辑地址，由第一线程将第一行中的第一因数集和第二张量中的第二因数集进行点积运算，以生成第三张量的第一行中的第一点积集
806
由第一线程将第一点积集存储在与第一乘积寄存器表示对应的第一组乘积寄存器中

图8

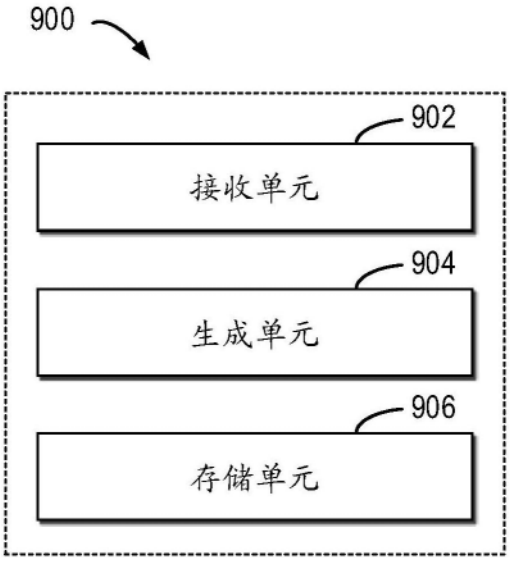

900
902
接收单元
904
生成单元
906
存储单元

图9