



(10)授权公告号 CN 108564169 B

(21)申请号 201810324108.4

(22)申请日 2018.04.11

(65)同一申请的已公布的文献号

申请公布号 CN 108564169 A

(43)申请公布日 2018.09.21

(30) 优先权数据

62/484,353 2017.04.11 US

(73)专利权人 上海兆芯集成电路有限公司

地址 201203 上海市浦东新区张江高科技
园区金科路2537号301室

(72)发明人 道格拉斯·R·瑞德

G·葛兰·亨利 金·C·霍克

帕尔维兹·帕朗查尔

(74)专利代理机构 北京林达刘知识产权代理事
务所(普通合伙) 11277

代理人 刘新宇

(51) Int.Cl.

G06N 3/063(2006.01)

(56)对比文件

CN 106127302 A, 2016.11.16

EP 0490177 A3, 1993.07.14

US 5091864 A, 1992.02.25

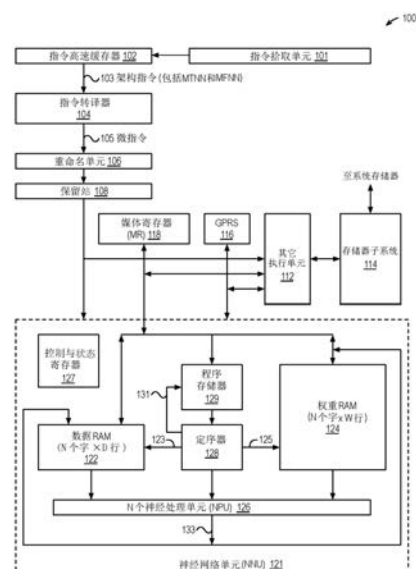
CN 106528047 A.2017.03.22

审查员 于淼

权利要求书4页 说明书107页 附图54页

硬件处理单元、神经网络单元和计算机可用
介质

本发明涉及一种硬件处理单元、神经网络单元和计算机可用介质。该硬件处理单元包括：累加器；乘加器，其用于在第一输入和第二输入上接收相应的第一因子和第二因子、并且在第三输入上接收加数，乘加器产生第一因子和第二因子的乘积与加数的总和、并将总和提供于乘加器的输出上；第一复用器，其接收第一操作数、正1和负1并选择其中之一以作为第一因子提供至乘加器；第二复用器，其接收第二操作数、正1和负1并选择其中之一以作为第二因子提供至乘加器；第三复用器，其接收第一操作数和第二操作数并选择其中之一以提供于第三复用器的输出上；以及第四复用器，其接收第三复用器的输出和总和、并选择其中之一以提供至累加器。



1. 一种硬件处理单元,包括:

累加器,其具有输入和输出;

乘加器,其具有输出以及第一输入、第二输入和第三输入,所述乘加器用于在第一输入和第二输入上接收相应的第一因子和第二因子、并且在第三输入上接收加数,所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数的总和、并将所述总和提供于所述乘加器的输出上;

第一复用器,其具有耦接至所述乘加器的第一输入的输入,并且所述第一复用器用于接收第一操作数、正1和负1并选择其中之一以作为所述第一因子提供至所述乘加器;

第二复用器,其具有耦接至所述乘加器的第二输入的输入,并且所述第二复用器用于接收第二操作数、正1和负1并选择其中之一以作为所述第二因子提供至所述乘加器;

第三复用器,其具有输出,所述第三复用器用于接收所述第一操作数和所述第二操作数并选择其中之一以提供于所述第三复用器的输出上;以及

第四复用器,其具有耦接至所述累加器的输入的输入,所述第四复用器用于接收所述第三复用器的输出和所述总和、并选择其中之一以提供至所述累加器。

2. 根据权利要求1所述的硬件处理单元,其中,还包括:

控制逻辑,用于响应于向所述累加器写入所述第一操作数和所述第二操作数的乘积与所述累加器的总和的第一命令而进行以下操作:

控制所述第一复用器选择所述第一操作数;

控制所述第二复用器选择所述第二操作数;以及

控制所述第四复用器选择所述总和,

其中,所述控制逻辑响应于使所述累加器包含所述第一操作数和所述累加器中的最大值的第二命令而进行以下操作:

控制所述第一复用器选择所述第一操作数;

控制所述第二复用器选择负1;

控制所述第三复用器选择所述第一操作数;以及

控制所述第四复用器在所述总和为负的情况下选择所述第一操作数,否则使所述累加器保留所述累加器的值。

3. 根据权利要求2所述的硬件处理单元,其中,还包括:

第五复用器,其具有耦接至所述乘加器的第三输入的输入,所述第五复用器用于接收所述第三复用器的输出和所述累加器的输出并选择其中之一以作为所述加数提供至所述乘加器,

其中,所述控制逻辑响应于向所述累加器写入所述第二操作数与所述第一操作数的差的第三命令而进行以下操作:

控制所述第一复用器选择负1;

控制所述第二复用器选择所述第二操作数;

控制所述第三复用器选择所述第一操作数;

控制所述第五复用器选择所述第三复用器的输出;以及

控制所述第四复用器选择所述总和。

4. 根据权利要求3所述的硬件处理单元,其中,

所述控制逻辑响应于向所述累加器写入所述第二操作数和所述累加器的总和的第四命令而进行以下操作：

控制所述第一复用器选择正1；

控制所述第二复用器选择所述第二操作数；

控制所述第五复用器选择所述累加器的输出；以及

控制所述第四复用器选择所述总和。

5. 根据权利要求3所述的硬件处理单元，其中，

所述第五复用器还接收零，并在零、所述第三复用器的输出和所述累加器的输出中选择其中之一以作为所述加数提供至所述乘加器，以及

其中，所述控制逻辑响应于向所述累加器写入所述第一操作数和所述第二操作数的乘积的第四命令而进行以下操作：

控制所述第一复用器选择所述第一操作数；

控制所述第二复用器选择所述第二操作数；

控制所述第五复用器选择所述零；以及

控制所述第四复用器选择所述总和。

6. 根据权利要求3所述的硬件处理单元，其中，

所述第一复用器还接收零，并在零、所述第一操作数、正1和负1中选择其中之一以作为所述第一因子提供至所述乘加器，

所述第五复用器还接收所述累加器的输出的二进制补码，并在所述累加器的输出的所述二进制补码、所述第三复用器的输出和所述累加器的输出中选择其中之一以作为所述加数提供至所述乘加器，以及

所述控制逻辑响应于使所述累加器包含所述累加器的绝对值的第四命令而进行以下操作：

控制所述第一复用器选择零；

控制所述第五复用器选择所述累加器的输出的所述二进制补码；以及

控制所述第四复用器在所述累加器为负的情况下选择所述总和，否则使所述累加器保留所述累加器的值。

7. 根据权利要求3所述的硬件处理单元，其中，

所述第一复用器还接收零，并在零、所述第一操作数、正1和负1中选择其中之一以作为所述第一因子提供至所述乘加器，

所述第五复用器还接收所述累加器的输出的二进制反码，并在所述累加器的输出的所述二进制反码、所述第三复用器的输出和所述累加器的输出中选择其中之一以作为所述加数提供至所述乘加器，

所述乘加器具有用于接收第二加数的第四输入，所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数和所述第二加数的总和，并将所述总和提供于所述乘加器的输出上，以及

所述控制逻辑响应于使所述累加器包含所述累加器的绝对值的第四命令而进行以下操作：

控制所述第一复用器选择零；

控制所述第五复用器选择所述累加器的输出的所述二进制反码；
在所述累加器为负的情况下提供正1作为所述第二加数，否则提供零；以及
控制所述第四复用器在所述累加器为负的情况下选择所述总和，否则使所述累加器保留所述累加器的值。

8. 根据权利要求3所述的硬件处理单元，其中，

所述第一复用器还接收零，并在零、所述第一操作数、正1和负1中选择其中之一以作为所述第一因子提供至所述乘加器，

所述第五复用器还接收所述累加器的输出的二进制反码，并在所述累加器的输出的所述二进制反码、所述第三复用器的输出和所述累加器的输出中选择其中之一以作为所述加数提供至所述乘加器，

所述乘加器具有用于接收第二加数的第四输入，所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数和所述第二加数的总和，并将所述总和提供于所述乘加器的输出上，以及

所述控制逻辑响应于使所述累加器加载所述累加器的二进制补码的第四命令而进行以下操作：

控制所述第一复用器选择零；

控制所述第五复用器选择所述累加器的输出的所述二进制反码；

提供正1作为所述第二加数；以及

控制所述第四复用器选择所述总和。

9. 根据权利要求2所述的硬件处理单元，其中，

所述第四复用器还接收符号值，并在所述符号值、所述第三复用器的输出和所述总和中选择其中之一以提供至所述累加器，以及

所述控制逻辑响应于基于所述累加器的符号使所述累加器包含第一预定值、第二预定值或第三预定值的第三命令而进行以下操作：

在所述累加器为零的情况下提供所述符号值作为所述第一预定值，在所述累加器为正整数的情况下提供所述符号值作为所述第二预定值，以及在所述累加器为负整数的情况下提供所述符号值作为所述第三预定值；以及

控制所述第四复用器选择所述符号值。

10. 根据权利要求2所述的硬件处理单元，其中，

所述第四复用器还接收饱和值；以及

所述控制逻辑响应于所述第一命令在检测到所述乘加器产生所述总和时出现溢位/欠位的情况下进行以下操作：

提供最大正/负值作为所述饱和值；以及

控制所述第四复用器选择所述饱和值以提供至所述累加器。

11. 根据权利要求1所述的硬件处理单元，其中，还包括：

第一减法器，用于从量化数据字中减去第一偏移以产生所述第一操作数；以及

第二减法器，用于从量化权重字中减去第二偏移以产生所述第二操作数。

12. 根据权利要求11所述的硬件处理单元，其中，所述量化数据字和所述量化权重字是8位无符号字，并且所述第一操作数和所述第二操作数是9位带符号字。

13. 根据权利要求12所述的硬件处理单元,其中,所述累加器保持23位带符号值。
14. 根据权利要求1所述的硬件处理单元,其中,所述第一操作数和所述第二操作数是9位带符号值,并且所述加数是23位带符号值。
15. 根据权利要求14所述的硬件处理单元,其中,所述总和是23位带符号值。

硬件处理单元、神经网络单元和计算机可用介质

技术领域

[0001] 本发明涉及一种硬件处理单元、神经网络单元和计算机可用介质。

背景技术

[0002] 最近,人工神经网络 (artificial neural network, ANN) 重新吸引了人们的兴趣,并且这种研究通常被称为深度学习、计算机学习和类似的术语。通用处理器计算能力的提高使得重新提起数十年前所消退的兴趣。ANN的最新应用包括语音识别和图像识别等。对于改进与ANN相关联的计算的性能和效率的需求正在增大。

发明内容

[0003] 一种硬件处理单元,包括:累加器,其具有输入和输出;乘加器,其具有输出以及第一输入、第二输入和第三输入,所述乘加器用于在第一输入和第二输入上接收相应的第一因子和第二因子、并且在第三输入上接收加数,所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数的总和、并将所述总和提供于所述乘加器的输出上;第一复用器,其具有耦接至所述乘加器的第一输入的输入,并且所述第一复用器用于接收第一操作数、正1和负1并选择其中之一以作为所述第一因子提供至所述乘加器;第二复用器,其具有耦接至所述乘加器的第二输入的输入,并且所述第二复用器用于接收第二操作数、正1和负1并选择其中之一以作为所述第二因子提供至所述乘加器;第三复用器,其具有输出,所述第三复用器用于接收所述第一操作数和所述第二操作数并选择其中之一以提供于所述第三复用器的输出上;以及第四复用器,其具有耦接至所述累加器的输入的输入,所述第四复用器用于接收所述第三复用器的输出和所述总和、并选择其中之一以提供至所述累加器。

[0004] 一种神经网络单元,包括:N个硬件处理单元的阵列,其中所述N个硬件处理单元各自包括:累加器,其具有输入和输出;乘加器,其具有输出以及第一输入、第二输入和第三输入,所述乘加器用于在第一输入和第二输入上接收相应的第一因子和第二因子、并且在第三输入上接收加数,所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数的总和、并将所述总和提供于所述乘加器的输出上;第一复用器,其具有耦接至所述乘加器的第一输入的输入,并且所述第一复用器用于接收第一操作数、正1和负1并选择其中之一以作为所述第一因子提供至所述乘加器;第二复用器,其具有耦接至所述乘加器的第二输入的输入,并且所述第二复用器用于接收第二操作数、正1和负1并选择其中之一以作为所述第二因子提供至所述乘加器;第三复用器,其具有输出,所述第三复用器用于接收所述第一操作数和所述第二操作数并选择其中之一以提供于所述第三复用器的输出上;以及第四复用器,其具有耦接至所述累加器的输入的输入,所述第四复用器用于接收所述第三复用器的输出和所述总和并选择其中之一以提供至所述累加器,其中,N至少为1024。

[0005] 一种非暂时性计算机可用介质,其包括计算机可用程序,所述计算机可用程序使得计算机用作根据本发明所述的硬件处理单元中的各部件。

[0006] 一种硬件处理单元,包括:累加器,其具有输入和输出;乘加器,其具有输出以及第

一输入、第二输入和第三输入,所述乘加器用于在第一输入和第二输入上接收相应的第一因子和第二因子、并且在第三输入上接收加数,所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数的总和、并将所述总和提供于所述乘加器的输出上以被选择性地加载至所述累加器中;第一复用器,其具有耦接至所述乘加器的第一输入的输出,并且所述第一复用器用于接收第一操作数、正1和负1并选择其中之一以作为所述第一因子提供至所述乘加器;第二复用器,其具有耦接至所述乘加器的第二输入的输出,并且所述第二复用器用于接收第二操作数、正1和负1并选择其中之一以作为所述第二因子提供至所述乘加器;第三复用器,其具有输出,所述第三复用器用于接收所述第一操作数和所述第二操作数并选择其中之一以提供于所述第三复用器的输出上;以及第四复用器,其具有耦接至所述乘加器的第三输入的输出,所述第四复用器用于接收所述第三复用器的输出和所述累加器的输出并选择其中之一以作为所述加数提供至所述乘加器。

[0007] 一种神经网络单元,包括:N个硬件处理单元的阵列,其中所述N个硬件处理单元各自包括:累加器,其具有输入和输出;乘加器,其具有输出以及第一输入、第二输入和第三输入,所述乘加器用于在第一输入和第二输入上接收相应的第一因子和第二因子、并且在第三输入上接收加数,所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数的总和、并将所述总和提供于所述乘加器的输出上以被选择性地加载至所述累加器中;第一复用器,其具有耦接至所述乘加器的第一输入的输出,并且所述第一复用器用于接收第一操作数、正1和负1并选择其中之一以作为所述第一因子提供至所述乘加器;第二复用器,其具有耦接至所述乘加器的第二输入的输出,并且所述第二复用器用于接收第二操作数、正1和负1并选择其中之一以作为所述第二因子提供至所述乘加器;第三复用器,其具有输出,所述第三复用器用于接收所述第一操作数和所述第二操作数并选择其中之一以提供于所述第三复用器的输出上;以及第四复用器,其具有耦接至所述乘加器的第三输入的输出,所述第四复用器用于接收所述第三复用器的输出和所述累加器的输出并选择其中之一以作为所述加数提供至所述乘加器,其中,N至少为1024。

[0008] 一种计算机程序产品,其被编码在至少一个非暂时性计算机可用介质中以与计算装置一起使用,所述计算机程序产品包括:计算机可用程序代码,其被实现在所述非暂时性计算机可用介质中,用于指定硬件处理单元,所述计算机可用程序代码包括:第一程序代码,用于指定累加器,所述累加器具有输入和输出;第二程序代码,用于指定乘加器,所述乘加器具有输出以及第一输入、第二输入和第三输入,所述乘加器用于在第一输入和第二输入上接收相应的第一因子和第二因子、并且在第三输入上接收加数,所述乘加器产生所述第一因子和所述第二因子的乘积与所述加数的总和、并将所述总和提供于所述乘加器的输出上以被选择性地加载至所述累加器中;第三程序代码,用于指定第一复用器,所述第一复用器具有耦接至所述乘加器的第一输入的输出,并且所述第一复用器用于接收第一操作数、正1和负1并选择其中之一以作为所述第一因子提供至所述乘加器;第四程序代码,用于指定第二复用器,所述第二复用器具有耦接至所述乘加器的第二输入的输出,并且所述第一复用器用于接收第二操作数、正1和负1并选择其中之一以作为所述第二因子提供至所述乘加器;第五程序代码,用于指定第三复用器,所述第三复用器具有输出,并且所述第一复用器用于接收所述第一操作数和所述第二操作数并选择其中之一以提供于所述第三复用器的输出上;以及第六程序代码,用于指定第四复用器,所述第四复用器具有耦接至所述乘

加器的第三输入的输出,并且所述第一复用器用于接收所述第三复用器的输出和所述累加器的输出并选择其中之一以作为所述加数提供至所述乘加器。

附图说明

[0009] 图1是示出包括神经网络单元(neural network unit, NNU)的处理器框图。

[0010] 图2是示出图1的NPU的框图。

[0011] 图3是示出图1的NNU的N个NPU的N个复用寄存器(mux-reg)的布置的实施例的框图,以说明N个复用寄存器作为针对从图1的数据RAM接收到的一行数据字的N字轮转器(N-word rotater)或循环移位器的操作。

[0012] 图4是示出用于储存在图1的NNU的程序存储器中并由该NNU执行的程序的表格。

[0013] 图5是示出NNU执行图4的程序的时序图。

[0014] 图6A是示出图1的NNU执行图4的程序的框图。

[0015] 图6B是示出图1的处理器进行架构程序的操作的流程图,所述架构程序使用NNU来执行典型地与人工神经网络的隐藏层的神经元相关联的(诸如图4的程序所执行的)乘法累加激活函数计算。

[0016] 图7是示出根据备选实施例的图1的NPU的框图。

[0017] 图8是示出根据备选实施例的图1的NPU的框图。

[0018] 图9是示出用于储存在图1的NNU的程序存储器中并由该NNU执行的程序的表格。

[0019] 图10是示出NNU执行图9的程序的时序图。

[0020] 图11是示出图1的NNU的实施例的框图。在图11的实施例中,神经元分成两部分即激活函数单元部分和ALU部分(此部分还包括移位寄存器部分),并且各激活函数单元部分由多个ALU部分共享。

[0021] 图12是示出图11的NNU执行图4的程序的时序图。

[0022] 图13是示出图11的NNU执行图4的程序的时序图。

[0023] 图14是示出向神经网络移动(MTNN)架构指令以及该架构指令相对于图1的NNU的部分的操作的框图。

[0024] 图15是示出从神经网络移动(MFNN)架构指令以及该架构指令相对于图1的NNU的部分的操作的框图。

[0025] 图16是示出图1的数据RAM的实施例的框图。

[0026] 图17是示出图1的权重RAM和缓存器的实施例的框图。

[0027] 图18是示出图1的可动态配置的NPU的框图。

[0028] 图19是示出根据图18的实施例的图1的NNU的N个NPU的2N个复用寄存器的布置的实施例的框图,以说明2N个复用寄存器作为针对从图1的数据RAM接收到的一行数据字的轮转器的操作。

[0029] 图20是示出用于储存在图1的NNU的程序存储器中并由该NNU执行的程序的表格,其中所述NNU具有根据图18的实施例的NPU。

[0030] 图21是示出NNU执行图20的程序的时序图,其中该NNU包括图18的在窄配置中操作的NPU。

[0031] 图22是示出图1的NNU的框图,其中该NNU包括图18的NPU以执行图20的程序。

- [0032] 图23是示出根据备选实施例的图1的可动态配置的NPU的框图。
- [0033] 图24是示出图1的NNU执行卷积运算所使用的数据结构示例的框图。
- [0034] 图25是示出图1的处理器执行架构程序的操作的流程图,所述架构程序将对卷积核执行卷积的NNU用于图24的数据阵列。
- [0035] 图26A是NNU程序的程序列表,其中该NNU程序利用图24的卷积核对数据矩阵执行卷积并将其写回权重RAM。
- [0036] 图26B是示出根据一个实施例的图1的NNU的控制寄存器的特定字段的框图。
- [0037] 图27是示出填有输入数据的图1的权重RAM的示例的框图,其中图1的NNU对所述输入数据进行池化(pooling)运算。
- [0038] 图28是NNU程序的程序列表,其中该NNU程序对图27的输入数据矩阵进行池化运算并将其写回权重RAM。
- [0039] 图29A是示出图1的控制寄存器的实施例的框图。
- [0040] 图29B是示出根据备选实施例的图1的控制寄存器的实施例的框图。
- [0041] 图29C是示出根据一个实施例的存储为两个部分的图29A的倒数的实施例的框图。
- [0042] 图30是更详细示出图2的AFU的实施例的框图。
- [0043] 图31是图30的AFU的操作的示例。
- [0044] 图32是图30的AFU的操作的第二示例。
- [0045] 图33是图30的AFU的操作的第三示例。
- [0046] 图34是示出图1的处理器以及图1的NNU的更详细的部分的框图。
- [0047] 图35是示出包括可变速率NNU的处理器的框图。
- [0048] 图36A是示出具有在一般模式中操作即以主时钟速率操作的NNU的处理器的操作示例的时序图。
- [0049] 图36B是示出具有在缓和模式中操作即以比主时钟速率小的速率操作的NNU的处理器操作示例的时序图。
- [0050] 图37是示出图35的处理器操作的流程图。
- [0051] 图38是更详细示出NNU的序列的框图。
- [0052] 图39是示出NNU的控制和状态寄存器的特定字段的框图。
- [0053] 图40是示出Elman RNN的示例的框图。
- [0054] 图41是示出在NNU执行与图40的Elman RNN相关联的计算时、该NNU的数据RAM和权重RAM内的数据布局的示例的框图。
- [0055] 图42是示出用于储存在NNU的程序存储器中并由该NNU执行以实现Elman RNN的程序并使用根据图41的布置的数据和权重的表格。
- [0056] 图43是示出Jordan RNN的示例的框图。
- [0057] 图44是示出在NNU执行与图43的Jordan RNN相关联的计算时、该NNU的数据RAM和权重RAM内的数据布局的示例的框图。
- [0058] 图45是示出用于储存在NNU的程序存储器中并由该NNU执行以实现Jordan RNN的程序并且使用根据图44的布置的数据和权重的表格。
- [0059] 图46是示出LSTM单元(cell)的实施例的框图。
- [0060] 图47是示出在NNU执行与图46的LSTM单元层相关联的计算时、该NNU的数据RAM和

权重RAM内的数据布局的示例的框图。

[0061] 图48是示出用于储存在NNU的程序存储器中并由该NNU执行以实现与LSTM单元层相关联的计算的程序并且使用根据图47的布置的数据和权重的表格。

[0062] 图49是示出在NPU组内具有输出缓存器屏蔽与反馈能力的NNU实施例的框图。

[0063] 图50是示出在图49的NNU执行与图46的LSTM单元层相关联的计算时、该NNU的数据RAM、权重RAM和输出缓存器内的数据布局的示例的框图。

[0064] 图51是示出用于储存在图49的NNU的程序存储器中并由该NNU执行以实现与LSTM单元层相关联的计算的程序并且使用根据图50的布置的数据和权重的表格。

[0065] 图52是示出在NPU组内具有输出缓存器屏蔽与反馈能力并且采用共享AFU的NNU实施例的框图。

[0066] 图53是示出根据备选实施例的在图49的NNU执行与图46的LSTM单元层相关联的计算时、该NNU的数据RAM、权重RAM和输出缓存器内的数据布局的示例的框图。

[0067] 图54是示出用于储存在图49的NNU的程序存储器中并由该NNU执行以实现与LSTM单元层相关联的计算的程序并且使用根据图53的布置的数据和权重的表格。

[0068] 图55是示出根据备选实施例的NPU的部分的框图。

[0069] 图56是示出在NNU执行与图43的Jordan RNN相关联的计算时、该NNU的数据RAM和权重RAM内的数据布局的示例、但采用图55的实施例所提供的益处的框图。

[0070] 图57是示出用于储存在NNU的程序存储器中并由该NNU执行以实现Jordan RNN的程序并且使用图56的布置的数据和权重的表格。

[0071] 图58是示出NNU的部分的实施例的框图。

[0072] 图59是示出NPU的实施例的框图。

[0073] 图60是示出NPU的备选实施例的框图。

[0074] 图61是示出NPU的备选实施例的框图。

[0075] 图62是示出处理器的框图。

[0076] 图63是更详细示出图62的环站 (ring stop) 的框图。

[0077] 图64是更详细示出图63的从接口的框图。

[0078] 图65是更详细示出图63的主接口的框图。

[0079] 图66是示出图63的环站以及NNU的环形总线耦接实施例的部分的框图。

[0080] 图67是示出图66的直接存储器访问控制器 (DMAC) 的框图。

[0081] 图68是示出图67的DMAC的块状态以及使用块状态的块状态机的框图。

[0082] 图69是示出图66的DMAC的框图。

[0083] 图70是示出图69的DMAC的块状态以及使用块状态的块状态机的框图。

[0084] 图71是示出基址寄存器和DMA控制字 (DCW) 的框图。

[0085] 图72是示出NNU的环形总线耦接实施例的框图。

具体实施方式

[0086] 具有架构神经网络单元的处理器的框图

[0087] 现参考图1, 示出一框图, 该框图示出包括神经网络单元 (NNU) 121 的处理器100。处理器100包括指令拾取单元101、指令高速缓存器102、指令转译器104、重命名单元106、保留

站108、媒体寄存器118、通用寄存器 (GPR) 116、NNU 121以外的执行单元112、以及存储器子系统114。

[0088] 处理器100是用作集成电路上的中央处理单元 (CPU) 的电子装置。处理器100接收数字数据作为输入,根据从存储器拾取的指令处理数据,并生成指令所规定的运算的结果作为输出。处理器100可用于台式计算机、移动计算机或平板计算机,并用于诸如计算、字编辑、多媒体显示和互联网浏览等的用途。处理器100还可设置于嵌入式系统内,以控制包括家用电器、移动电话、智能电话、车辆和工业控制装置等的各种各样的装置。CPU是通过对数据执行包括算术运算、逻辑运算和输入/输出运算的运算来执行计算机程序(也称为“计算机应用”或“应用”)指令的电子电路(即“硬件”)。集成电路 (IC) 是制作于小块半导体材料(通常为硅)上的一组电子电路。IC也称为芯片、微芯片或晶粒 (die)。

[0089] 指令拾取单元101控制将架构指令103从系统存储器(未示出)拾取至指令高速缓存器102。指令拾取单元101向指令高速缓存器102提供指定存储器地址的拾取地址,其中处理器100在所述存储器地址处将架构指令字节的高速缓存线拾取至指令高速缓存器102中。拾取地址是基于处理器100的指令指针(未示出)或程序计数器的当前值。通常,程序计数器依照指令大小循序递增,除非在指令串流中遇到诸如分支、调用或返回指令等的控制指令或者发生诸如中断、陷阱 (trap)、例外或错误等的例外条件,在上述情况下,利用诸如分支目标地址、返回地址或例外向量等的非循序地址来更新程序计数器。一般来说,程序计数器响应于执行单元112/121执行指令而进行更新。程序计数器也可响应于检测到例外条件(诸如指令转译器104遇到处理器100的指令集架构未定义的指令103等)而进行更新。

[0090] 指令高速缓存器102对从耦接至处理器100的系统存储器拾取到的架构指令103进行高速缓存。架构指令103包括以下将更详细说明的向神经网络移动 (MTNN) 指令和从神经网络移动 (MFNN) 指令。在一个实施例中,架构指令103是x86指令集架构 (ISA) 的指令,并附上MTNN指令与MFNN指令。在本发明的上下文中,x86ISA处理器作为在Intel® 80386®处理器执行相同的机器语言指令时在指令集架构层生成与Intel® 80386®处理器所生成的相同的结果的处理器。然而,其它实施例设想了其它指令集架构,诸如进阶精简指令集机器 (ARM)®、SUNSPARC®、或PowerPC®。指令高速缓存器102向指令转译器104提供架构指令103,指令转译器104将架构指令103转译为微指令105。

[0091] 微指令105被提供至重命名单元106、并且最终由执行单元112/121执行。微指令105实现架构指令。优选地,指令转译器104包括第一部分,其中该第一部分将频繁执行的和/或相对较不复杂的架构指令103转译为微指令105。指令转译器104还包括第二部分,其中该第二部分包括微码单元(未示出)。微码单元包括保持微码指令的微码存储器,其中所述微码指令实现架构指令集的复杂和/或不频繁使用的指令。微码单元还包括微定序器 (microsequencer),其中该微定序器将非架构微程序计数器 (micro-PC) 提供至微码存储器。优选地,微码指令经由微转译器(未示出)而被转译为微指令105。选择器根据微码单元当前是否具有控制权来从所述第一部分或所述第二部分选择微指令105以提供至重命名单元106。

[0092] 重命名单元106将架构指令103中所指定的架构寄存器重命名为处理器100的物理寄存器。优选地,处理器100包括重排序缓存器(未示出)。重命名单元106针对各微指令105

依照程序顺序来分配重排序缓存器中的条目。这样使得处理器100能够依照程序顺序引退(retire)微指令105及其相应的架构指令103。在一个实施例中,媒体寄存器118具有256位宽度,并且GPR 116具有64位宽度。在一个实施例中,媒体寄存器118是诸如高级向量扩充(AVX)寄存器等的x86媒体寄存器。

[0093] 在一个实施例中,重排序缓存器的各条目包括用于微指令105的结果的存储空间;此外,处理器100包括架构寄存器文件,其包括针对各架构寄存器(例如,媒体寄存器118、GPR 116以及其它架构寄存器)的物理寄存器。(优选地,例如,由于媒体寄存器118与GPR 116的大小不同,因此针对这两者,存在单独的寄存器文件。)对于微指令105中指定架构寄存器的各源操作数,重命名单元利用写入架构寄存器的旧有微指令105中的最新微指令的重排序缓存器索引来填充微指令105的源操作数字段。当执行单元112/121完成微指令105的执行时,执行单元112/121将结果写入微指令105的重排序缓存器条目。当微指令105引退时,引退单元(未示出)将所述结果从微指令的重排序缓存器条目写入与引退的微指令105所指定的架构目的地寄存器相关联的物理寄存器文件的寄存器。

[0094] 在另一实施例中,处理器100包括物理寄存器文件而不包括架构寄存器文件,其中该物理寄存器文件包括的物理寄存器的数量多于架构寄存器的数量,而且重排序缓存器条目不包括结果存储空间。(优选地,例如,由于媒体寄存器118与GPR 116的大小不同,因此针对这两者,存在单独的物理寄存器文件。)处理器100还包括指针表,其具有针对各架构寄存器的关联指针。对于微指令105中的指定架构寄存器的操作数,重命名单元利用指向物理寄存器文件中的空闲寄存器的指针来填充微指令105的目的地操作数字段。如果物理寄存器文件内不存在空闲寄存器,则重命名单元106暂时搁置流水线(pipeline)。对于微指令105的指定架构寄存器的各源操作数,重命名单元利用指向物理寄存器文件中被指派给写入架构寄存器的旧有微指令105中的最新微指令的寄存器的指针,来填充微指令105的源操作数字段。当执行单元112/121完成微指令105的执行时,执行单元112/121将结果写入物理寄存器文件中由微指令105的目的地操作数字段所指向的寄存器。当微指令105引退时,引退单元将微指令105的目的地操作数字段值复制到指针表中与该引退的微指令105所指定的架构目的地寄存器相关联的指针。

[0095] 保留站108保持微指令105,直到微指令准备好被发布至执行单元112/121以供执行为止。当微指令105的所有源操作数都可用并且执行单元112/121可用于执行微指令105时,准备好发布微指令105。执行单元112/121从重排序缓存器或前述第一实施例中的架构寄存器文件、或者从前述第二实施例中的物理寄存器文件接收寄存器源操作数。此外,执行单元112/121可经由结果转发总线(未示出)而直接从执行单元112/121接收寄存器源操作数。此外,执行单元112/121可以从保留站108接收微指令105所指定的即时操作数。如以下更详细所述,MTNN与MFNN架构指令103包括用于指定NNU 121所要执行的功能的即时操作数,其中该功能设置在MTNN与MFNN架构指令103转译成的一个或多个微指令105之一中。

[0096] 执行单元112包括一个或多个加载/储存单元(未示出),其从存储器子系统114加载数据并且将数据储存至存储器子系统114。优选地,存储器子系统114包括存储器管理单元(未示出),其可例如包括转译查找(lookaside)缓存器和表移动(tablewalk)单元、1级数据高速缓存器(和指令高速缓存器102)、2级统一高速缓存器、以及用于将处理器100与系统存储器接合的总线接口单元。在一个实施例中,图1的处理器100是作为多核处理器中的共

享末级高速缓存存储器的多个处理核心之一的处理核心的代表。执行单元112还可包括整数单元、媒体单元、浮点单元和分支单元。

[0097] NNU 121包括权重随机存取存储器(RAM) 124、数据RAM 122、N个神经处理单元(NPU) 126、程序存储器129、定序器128、以及控制和状态寄存器(CSRs) 127。NPU 126在概念上用作神经网络中的神经元。权重RAM 124、数据RAM 122与程序存储器129均可经由MTNN与MFNN架构指令103分别写入与读取。权重RAM 124排列为W行,每行N个权重字,并且数据RAM 122排列为D行,每行N个数据字。各个数据字与各个权重字均具有多个位,优选地为8位、9位、12位或16位。各个数据字用作网络中先前层的神经元的输出值(有时也称为激活值(activation)),并且各个权重字用作与进入网络当前层的神经元的连接相关联的权重。尽管在NNU 121的许多应用中、保持在权重RAM 124中的字或操作数实际上是与进入神经元的连接相关联的权重,但是应当理解,在NNU 121的其它应用中,保持在权重RAM 124中的字并非权重,但是因为这些字储存于权重RAM 124中,因此仍被称为“权重字”。例如,在NNU 121的某些应用中,例如图24至图26A的卷积示例或图27至图28的池化示例中,权重RAM 124可以保持非权重,诸如数据矩阵(例如图像像素数据)的元素等。同样地,尽管在NNU 121的许多应用中,保持在数据RAM 122中的字或操作数实际上是神经元的输出值或激活值,但是应当理解,在NNU 121的其它应用中,保持在数据RAM 122中的字并非如此,但是因为这些字储存于数据RAM 122中,因此仍然被称为“数据字”。例如,在NNU 121的某些应用中,例如图24至图26A的卷积示例中,数据RAM 122可以保持非神经元输出,诸如卷积核的元素等。

[0098] 在一个实施例中,NPU 126和定序器128包括组合逻辑、定序逻辑、状态机、或其组合。架构指令(例如,MFNN指令1500)将状态寄存器127的内容加载至GPR 116之一,以确定NNU 121的状态,例如确定为NNU 121已经完成了命令或已经完成了NNU 121从程序存储器129运行的程序的状态,或者确定为NNU 121自由接收新的命令或开始新的NNU程序的状态。

[0099] 有利地,NPU 126的数量可根据需要增加,并且权重RAM 124和数据RAM 122的大小可以相应地在宽度与深度上扩展。优选地,权重RAM 124较大,这是因为在典型的神经网络层中,存在与各个神经元相关联的许多连接因而存在许多权重。本文描述了与数据字和权重字的大小、权重RAM 124和数据RAM 122的大小、以及NPU 126的数量有关的各个实施例。在一个实施例中,具有64KB(8192位 $\times$ 64行)数据RAM 122、2MB(8192位 $\times$ 2048行)权重RAM 124以及512个NPU 126的NNU 121实现于台湾半导体制造有限公司(TSMC)的16纳米工艺中,占用面积约为3.3平方毫米。

[0100] 定序器128从程序存储器129拾取指令并执行,还包括产生地址和控制信号以提供给数据RAM 122、权重RAM 124和NPU 126。定序器128产生存储器地址123和读取命令以提供给数据RAM 122,从而在每行具有N个数据字的D行中选择其中之一并提供给N个NPU 126。定序器128还产生存储器地址125和读取命令以提供给权重RAM 124,从而在每行具有N个权重字的W行中选择其中之一以提供给N个NPU 126。定序器128产生以提供给NPU 126的地址123和125的顺序确定神经元之间的“连接”。定序器128还产生存储器地址123和写入命令以提供给数据RAM 122,从而在每行具有N个数据字的D行中选择其中之一以从N个NPU 126进行写入。定序器128还产生存储器地址125和写入命令以提供给权重RAM 124,从而在每行具有N个权重字的W行中选择其中之一以从N个NPU 126进行写入。定序器128还产生针对程序存储器129的存储器地址131以选择提供给定序器128的如以下所述等的NNU指令。存储器地址

131与程序计数器(未示出)相对应,其中定序器128通常通过程序存储器129的顺序位置使该程序计数器递增,除非定序器128遇到诸如循环指令(例如参见图26A)等的控制指令,在遇到控制指令的这种情况下,定序器128将程序计数器更新为该控制指令的目标地址。定序器128还产生针对NPU 126的控制信号,以指示NPU 126执行诸如初始化、算术/逻辑运算、转动和移位运算、激活函数、以及写回运算等的各种运算或功能,以下更详细地说明这样的范例(例如参见图34的微操作3418)。

[0101] N个NPU 126产生N个结果字133,其中结果字133可被写回权重RAM 124的行或写回数据RAM 122。优选地,权重RAM 124与数据RAM 122直接耦接至N个NPU 126。更具体地,权重RAM 124与数据RAM 122专用于NPU 126,而不由处理器100的其它执行单元112分享,并且这些NPU 126能够以持续方式(优选地,以流水线方式)在各时钟周期消耗来自权重RAM 124和数据RAM 122其中之一或两者的一行。在一个实施例中,数据RAM 122与权重RAM 124各自能够在各时钟周期向NPU 126提供8192位。如以下更详细所述,这8192位可以作为512个16位字或1024个8位字来消耗。

[0102] 有利地,可由NNU 121处理的数据集的大小不受限于权重RAM 124与数据RAM 122的大小,而仅受限于系统存储器的大小,这是因为可以使用MTNN与MFNN指令(例如,通过媒体寄存器118)将数据与权重在系统存储器与权重RAM 124以及数据RAM 122间移动。在一个实施例中,数据RAM 122是双端口的,使得能够在并行地从数据RAM 122读取或向数据RAM 122写入数据字时,将数据字写入至数据RAM 122。另外,包括高速缓存存储器的存储器子系统114的大型存储器阶层结构提供非常大的数据带宽以供系统存储器与NNU 121之间的传递。此外,优选地,存储器子系统114包括硬件数据预拾取器,其追踪存储器访问模式(诸如从系统存储器对神经数据和权重的加载等),并对高速缓存器阶层结构执行数据预拾取以促进向权重RAM 124与数据RAM 122的高带宽且低延迟的传递。

[0103] 虽然说明了被提供给各NPU 126的操作数其中之一是从权重存储器提供的并且被表示为权重(此用语常用于神经网络)的实施例,但应当理解,操作数可以是与能够通过所述设备提高速度的计算相关联的其它类型的数据。

[0104] 现参考图2,示出一框图,该框图示出图1的NPU 126。NPU 126操作以执行许多功能或运算。尤其是,有利地,NPU 126被配置为作为神经元或节点而在人工神经网络中操作以执行经典的乘法累加函数或运算。亦即一般而言,NPU 126(神经元)被配置为:(1)从与该NPU 126具有连接的各神经元(通常但没有必要一定从人工神经网络中的紧挨在前的一层)接收输入值;(2)将各输入值乘以与该连接相关联的相应权重值以产生乘积;(3)将所有乘积相加以产生总和;并且(4)对该总和执行激活函数以产生神经元的输出。然而,与如传统方式那样执行与所有连接输入相关联的所有乘法并接着将所有乘积相加在一起不同,有利地,各个神经元被配置为在给定的时钟周期内执行与连接输入其中之一相关联的权重乘法运算、然后将该乘积与同直到该点为止的在先时钟周期内所处理的连接输入相关联的乘积的累加值相加(累加)。假定存在至神经元的M个连接,则在对所有M个乘积进行累加后(耗费约M个时钟周期),该神经元对累加值执行激活函数以产生输出或结果。这具有如下的优点:与将关联于所有连接输入的所有乘积或者甚至乘积的子集相加的加法器相比,在神经元内需要更少的乘法器并且需要更小、更简单且更为快速的加法器电路(例如,2输入加法器)。因而这具有如下的优点:有利于在NNU 121内实现极大量(N个)的神经元(NPU 126),使得在

约M个时钟周期后, NNU 121已产生所有这些大量(N个)神经元的输出。最后, 由这样的神经元构成的NNU 121具备有效地针对大量不同的连接输入作为人工神经网络层而执行的优点。也就是说, 随着M针对不同层增加或减少, 产生神经元输出所需的时钟周期数相应地增加或减少, 并且资源(例如, 乘法器与累加器)得到充分利用; 而在更传统的设计中, 对于较小的M值, 某些乘法器和部分加法器没有被利用。因此, 本文所述的实施例关于至NNU 121的神经元的连接输入数具有灵活与高效的益处, 并且提供极高的性能。

[0105] NPU 126包括寄存器205、2输入复用寄存器(mux-reg) 208、算术逻辑单元(ALU) 204、累加器202、以及激活函数单元(AFU) 212。寄存器205从权重RAM 124接收权重字206并在后续时钟周期中提供其输出203。复用寄存器208选择输入207或211其中之一, 以储存在其寄存器中然后在后续时钟周期中提供于输出209上。一个输入207接收来自数据RAM 122的数据字。另一输入211接收相邻NPU 126的输出209。图2所示的NPU 126在图1的N个NPU 126中被标示为NPU J。也就是说, NPU J是N个NPU 126的代表性实例。优选地, NPU J的复用寄存器208的输入211接收NPU 126的实例J-1的复用寄存器208的输出209, 并且NPU J的复用寄存器208的输出209被提供给NPU 126的实例J+1的复用寄存器208的输入211。如此, 如下针对图3更详细所述, N个NPU 126的复用寄存器208作为N字轮转器或循环移位器整体操作。控制输入213控制这两个输入中的哪一个被复用寄存器208选择以储存于寄存器中并后续提供于输出209上。

[0106] ALU 204具有三个输入。一个输入从寄存器205接收权重字203。另一输入接收复用寄存器208的输出209。再一个输入接收累加器202的输出217。ALU 204对其输入执行算术和/或逻辑运算以产生提供于其输出上的结果。优选地, ALU 204所执行的算术和/或逻辑运算由储存于程序存储器129的指令指定。例如, 图4的乘法累加指令指定乘法累加运算, 即, 结果215是权重字203以及复用寄存器208的输出209的数据字的乘积与累加器202的值217的总和。可以指定的其它运算包括但不限于: 结果215是复用寄存器输出209的通过值; 结果215是权重字203的通过值; 结果215是零; 结果215是权重字203的通过值; 结果215是累加器202的值217与权重字203的总和; 结果215是累加器202的值217与复用寄存器的输出209的总和; 结果215是累加器202的值217与权重字203的最大值; 结果215是累加器202的值217与复用寄存器的输出209的最大值。

[0107] ALU 204将输出215提供至累加器202以储存在该累加器202中。ALU 204包括乘法器242, 其用于将权重字203与复用寄存器208的输出209的数据字相乘以产生一乘积246。在一个实施例中, 乘法器242将两个16位操作数相乘以产生32位结果。ALU 204还包括加法器244, 其用于将乘积246与累加器202的输出217相加以产生一总和, 该总和为在累加器202中累加以供储存于累加器202中的结果215。在一个实施例中, 加法器244将乘法器242的32位结果与累加器202的41位值217相加以产生41位结果。如此, 通过在多个时钟周期的过程中使用复用寄存器208的轮转器方面, NPU 126完成神经网络所需的针对神经元的乘积的相加。ALU 204还可以包括其它电路元件来执行如前所述的其它算术/逻辑运算。在一个实施例中, 第二加法器从复用寄存器208的输出209的数据字中减去权重字203以产生一差值, 然后加法器244将该差值与累加器202的输出217相加以产生总和215, 该结果即为累加器202内累加的结果。如此, 在多个时钟周期的过程中, NPU 126可以完成差值的相加。优选地, 如下更详细所述, 虽然权重字203与数据字209的大小相同(以位为单位), 但也可具有不同

的二进制小数点位置。优选地,如以下更详细所述,乘法器242与加法器244为整数乘法器与加法器,以有利地实现与浮点乘法器和加法器相比复杂度更低、更小、更快速且耗能更低的ALU 204。然而,应当理解,在其它实施例中,ALU 204执行浮点运算。

[0108] 虽然图2仅示出ALU 204内的乘法器242和加法器244,但是优选地,该ALU 204包括其它元件来执行上述的其它运算。例如,ALU 204优选包括用于将累加器202与数据/权重字进行比较的比较器(未示出)以及用于选择比较器所指示的两个值中较大者(最大值)以储存于累加器202中的复用器(未示出)。再例如,ALU 204优选包括选择逻辑(未示出),其用于使数据/权重字跳过乘法器242以使得加法器244能够将数据/权重字与累加器202的值217相加以产生用于储存于累加器202中的总和。这些附加运算在以下(例如针对图18至图29A)更详细地说明,并且可以用于执行例如卷积运算和池化运算。

[0109] AFU 212接收累加器202的输出217。AFU 212对累加器202的输出217执行激活函数以产生图1的结果133。一般而言,人工神经网络的中间层的神经元内的激活函数可用来优选地采用非线性的方式使乘积的累加和标准化。为了使累加和“标准化”,当前神经元的激活函数在连接至当前神经元的其它神经元预期作为输入而接收到的值的范围内产生结果值。(标准化结果有时称为“激活值”,如本文所述,激活值是当前节点的输出,而接收节点将该输出乘以与输出节点和接收节点之间的连接相关联的权重以产生一乘积,而该乘积与关联于至所述接收节点的其它输入连接的其它乘积累加。)例如,接收/所连接神经元预期接收0与1之间的值作为输入,在这种情况下,输出神经元可能需要将在0~1范围之外的累加和非线性地挤压和/或调整(例如向上移位以将负值转换为正值)为预期范围内的值。因此,AFU 212对累加器202的值217执行运算以使结果133进入已知范围内。所有N个NPU 126的结果133可被并行地写回数据RAM 122或权重RAM 124。优选地,AFU 212被配置为执行多个激活函数,并且例如来自控制寄存器127的输入选择这些激活函数之一以对累加器202的输出217执行。激活函数可以包括但不限于接跃函数(step function)、校正函数(rectify function)、S型函数(sigmoid function)、双曲正切(tanh)函数(hyperbolic tangent function)以及软加函数(softplus function)(也称为平滑校正函数)。软加函数是解析函数 $f(x) = \ln(1 + e^x)$,即1与 e^x 的总和的自然对数,其中“e”是欧拉(Euler)数,并且x是函数的输入217。优选地,如以下更详细所述,激活函数还可以包括通过累加器202的值217或其一部分的通过(pass-through)函数。在一个实施例中,AFU 212的电路在单个时钟周期内执行激活函数。在一个实施例中,AFU 212包括表格,所述表格接收累加值并针对某些激活函数(例如S型函数、双曲正切函数、软加函数等)输出与真正的激活函数将提供的值相近似的值。

[0110] 优选地,累加器202的宽度(以位为单位)大于AFU 212的输出133的宽度。例如,在一个实施例中,累加器的宽度为41位,以避免对(如以下例如针对图30更详细所述的)多达512个32位乘积的累加的精度损失,并且结果133的宽度为16位。在以下针对图8更详细地描述了其示例的一个实施例中,在后续时钟周期期间,“原始”累加器202的输出217值的不同部分通过AFU 212,并且被写回数据RAM 122或权重RAM 124。这样使得能够经由MFNN指令将原始累加器202的值加载回媒体寄存器118,如此处理器100的其它执行单元112上执行的指令可以执行AFU 212无法执行的诸如众所周知的软极大(softmax)激活函数等的复杂激活函数(也称为标准化指数函数)。在一个实施例中,处理器100的指令集架构包括执行指数函

数的指令,通常称为 e^x 或 $\exp(x)$,该指令可以用来加快处理器100的其它执行单元112对软极大激活函数的执行。

[0111] 在一个实施例中,NPU 126为流水线设计。例如,NPU 126可包括ALU 204的寄存器(诸如位于乘法器与加法器和/或ALU 204的其它电路之间的寄存器)以及保持AFU 212的输出寄存器。以下描述了NPU 126的其它实施例。

[0112] 现参考图3,示出一框图,该框图示出图1的NNU 121的N个NPU 126的N个复用寄存器208的布置的实施例,从而说明N个复用寄存器作为针对从图1的数据RAM 122接收到的一行数据字207的N字轮转器或循环移位器的操作。在图3的实施例中,N是512,使得NNU 121具有如图所示与512个NPU 126相对应的标示为0至511的512个复用寄存器208。每个复用寄存器208接收数据RAM 122的D行中的一行上的相应的数据字207。也就是说,复用寄存器0接收数据RAM 122的行中的数据字0,复用寄存器1接收数据RAM 122的行中的数据字1,复用寄存器2接收数据RAM 122的行中的数据字2,依此类推,复用寄存器511接收数据RAM 122的行中的数据字511。此外,复用寄存器1在另一输入211上接收复用寄存器0的输出209,复用寄存器2在另一输入211上接收复用寄存器1的输出209,复用寄存器3在另一输入211上接收复用寄存器2的输出209,依此类推,复用寄存器511在另一输入211上接收复用寄存器510的输出209,而复用寄存器0在另一输入211上接收复用寄存器511的输出209。每个复用寄存器208接收控制输入213,其用于控制是选择数据字207还是选择轮转输入211。如以下更详细所述,在一个操作模式中,在第一时间钟周期内,控制输入213控制各复用寄存器208选择数据字207以储存在寄存器中并后续提供给ALU 204;以及在后续的时钟周期(例如,如上所述的M-1个时钟周期)期间,控制输入213控制各复用寄存器208选择轮转输入211以储存在寄存器中并后续提供给ALU 204。

[0113] 尽管在图3(以及以下的图7和图19)所描述的实施例中,NPU 126被配置为将复用寄存器208/705的值向右轮转、即从NPU J到NPU J+1,但是设想了如下的实施例(诸如针对图24至图26的实施例等),其中NPU 126被配置为将复用寄存器208/705的值向左轮转,即从NPU J到NPU J-1。此外,设想了如下的实施例,其中NPU 126被配置为选择性地为复用寄存器208/705的值向左或向右轮转,例如由NNU指令指定。

[0114] 现参考图4,示出一表格,该表格示出用于储存在图1的NNU 121的程序存储器129中并由该NNU 121执行的程序。如上所述,示例性程序执行与人工神经网络的层相关联的计算。在图4的表格中,示出五行与三列。每一行对应于程序存储器129中的标示于第一行的地址。第二列指定指令,并且第三列表示与该指令相关联的时钟周期数。优选地,时钟周期数表示在流水线实施例中每指令时钟类型值中有效的时钟数,而非指令的延迟。如图所示,因为NNU 121的流水线的本质,因此各指令均有相关联的一个时钟周期,其中位于地址2的指令是一个例外,如以下更详细所述,由于该指令实际上自己重复511次,因而需要511个时钟。

[0115] 针对程序的各指令,所有的NPU 126并行处理该指令。也就是说,所有的N个NPU 126都在同一个(或多个)时钟周期中执行第一行中的指令,所有的N个NPU 126都在同一个(或多个)时钟周期中执行第二行中的指令,依此类推。然而,以下描述了其它实施例,其中有些指令是以部分并行且部分循序的方式执行的,例如,在例如针对图11的实施例那样NPU 126共享激活函数单元的实施例中,分别位于地址3与4的激活函数和输出指令即是以此方

式执行。图4的示例假定一层具有512个神经元(NPU 126),而每个神经元具有来自先前层的512个神经元的512个连接输入,总共有256K个连接。每个神经元从各连接输入接收16位数据值,并将该16位数据值乘以适当的16位权重值。

[0116] 位于地址0(尽管也可指定其它地址)的第一行指定初始化NPU指令。该初始化指令将累加器202的值清零。在一个实施例中,初始化指令也可指定向累加器202加载数据RAM 122或权重RAM 124的一行中由该指令指定地址的相应字。如以下针对图29A和图29B更详细所述,该初始化指令还将配置值加载到控制寄存器127中。例如,可以加载数据字207与权重字209的宽度,其中所述宽度可由ALU 204使用以确定电路所执行的运算的大小、并且可以影响储存于累加器202中的结果215。在一个实施例中,NPU 126包括用于在ALU 204的输出215储存于累加器202之前使该输出215饱和的电路,并且初始化指令将配置值加载到该电路中以影响饱和。在一个实施例中,还可以通过在ALU函数指令(例如,地址1处的乘法累加指令)或输出指令(诸如地址4处的写入AFU输出指令)中如此指定,来将累加器202清除为零值。

[0117] 位于地址1的第二行指定乘法累加指令,其中该乘法累加指令指示512个NPU 126从数据RAM 122的一行加载相应的数据字并从权重RAM 124的一行加载相应的权重字,并且对数据字输入207与权重字输入206执行第一乘法累加运算,该第一乘法累加运算是在以初始化累加器202为零值的状态下进行累加的。更具体地,该指令指示定序器128在控制输入213上产生值以选择数据字输入207。在图4的示例中,所指定的数据RAM 122的行是行17,并且所指定的权重RAM 124的行是行0,从而指示定序器128输出数据RAM地址123的值17并且输出权重RAM地址125的值0。因此,来自数据RAM 122的行17的512个数据字被提供至512个NPU 126的相应数据输入207,并且来自权重RAM 124的行0的512个权重字被提供至512个NPU 126的相应权重输入206。

[0118] 位于地址2的第三列指定计数为511的乘法累加轮转指令,该指令指示这512个NPU 126中的各NPU 126执行511次乘法累加运算。该指令向这512个NPU 126指示在511次乘法累加运算的每一次运算中输入ALU 204的数据字209是来自相邻NPU 126的轮转值211。也就是说,该指令指示定序器128在控制输入213上产生值以选择轮转值211。此外,该指令指示这512个NPU 126从权重RAM 124的“下一”行加载针对511次乘法累加运算的每一次运算的相应权重值。也就是说,该指令指示定序器128使权重RAM地址125相对于其在前一时钟周期中的值加1,在该示例中,指令的第一时钟周期是行1,下一个时钟周期是行2,再下一个时钟周期是行3,依此类推,第511个时钟周期是行511。针对这511个乘法累加运算中的每一个运算,将轮转输入211和权重字输入206的乘积与累加器202的先前值累加。这512个NPU 126在511个时钟周期内执行511次乘法累加运算,其中各NPU 126对来自数据RAM 122的行17的不同数据字即相邻NPU 126在先前周期执行运算的数据字以及在概念上为神经元的不同连接输入的与该数据字相关联的不同权重字执行乘法累加运算。在该示例中,假设各个NPU 126(神经元)的连接输入的数量为512,因此涉及512个数据字与512个权重字。一旦执行了对行2的乘法累加轮转指令的最后一次迭代,累加器202就包含全部512个连接输入的乘积的总和。在一个实施例中,NPU 126的指令集包括用于指示ALU 204执行由初始化NPU指令指定(诸如在图29A的ALU函数2926中指定)的ALU运算的“执行”指令,而非对于各类型的ALU运算(例如,如上所述的乘法累加、累加器与权重字的求最大等)具有单独的指令。

[0119] 位于地址3的第四行指定激活函数指令。激活函数指令指示AFU 212对于累加器202的值217执行指定的激活函数以产生结果133。以下更详细说明了根据一个实施例的激活函数。

[0120] 位于地址4的第五行指定写入AFU输出指令,其用于指示512个NPU 126将AFU 212的输出作为结果133写回至数据RAM 122的一行(在该示例中为行16)。也就是说,该指令指示定序器128输出值为16的数据RAM地址123以及写入命令(与在地址1处的乘法累加指令的情况下的读取命令相对)。优选地,在流水线的本质下,写入AFU输出指令的执行可与其它指令的执行重叠,使得写入AFU输出指令实际上在单个时钟周期内执行。

[0121] 优选地,每个NPU 126被配置为流水线,其中该流水线包括各种功能元件,例如复用寄存器208(以及图7的复用寄存器705)、ALU 204、累加器202、AFU 212、(图8的)复用器802、行缓存器1104以及(图11的)AFU 1112等,其中这些功能元件中的一些本身即可以是流水线式的。除了数据字207与权重字206外,流水线还从程序存储器129接收指令。这些指令沿着流水线流动并控制各种功能单元。在备选实施例中,程序内不包含激活函数指令。相反,初始化NPU指令指定要对累加器202的值217执行的激活函数,并且指出所指定的激活函数的值被保存在配置寄存器中,从而稍后在已经产生最后的累加器202值217后,也就是在地址2处的乘法累加轮转指令的最后一次迭代完成后,由流水线的AFU 212部分使用。优选地,为了节能目的,流水线的AFU 212部分是不活动的,直到写入AFU输出指令到达该AFU 212部分为止,此时AFU 212启动并对初始化指令所指定的累加器202的输出217执行激活函数。

[0122] 现参考图5,示出一时序图,该时序图示出NNU 121对图4的程序的执行。该时序图的每一行对应于第一行指出的连续时钟周期。其它各列对应于512个NPU 126中的不同的一个NPU 126并指示其运算。为了使说明简单且清楚,仅示出NPU 0、1和511的运算。

[0123] 在时钟0处,512个NPU 126中的每一个NPU 126执行图4的初始化指令,其中该初始化指令在图5中是通过将零值指派给累加器202来示出的。

[0124] 在时钟1处,512个NPU 126中的每一个NPU 126执行图4中地址1处的乘法累加指令。如图所示,NPU 0将数据RAM 122的行17的字0与权重RAM 124的行0的字0的乘积与累加器202的值(即零)累加;NPU 1将数据RAM 122的行17的字1与权重RAM 124的行0的字1的乘积与累加器202的值(即零)累加;依此类推,NPU 511将数据RAM 122的行17的字511与权重RAM 124的行0的字511的乘积与累加器202的值(即零)累加。

[0125] 在时钟2处,512个NPU 126中的每一个NPU 126执行图4中地址2处的乘法累加轮转指令的第一次迭代。如图所示,NPU 0将从NPU 511的复用寄存器208的输出209接收到的轮转数据字211(即从数据RAM 122接收到的数据字511)与权重RAM 124的行1的字0的乘积与累加器202的值累加;NPU 1将从NPU 0的复用寄存器208的输出209接收到的轮转数据字211(即从数据RAM 122接收到的数据字0)与权重RAM 124的行1的字1的乘积与累加器202的值累加;依此类推,NPU 511将从NPU 510的复用寄存器208的输出209接收到的轮转数据字211(即从数据RAM 122接收到的数据字510)与权重RAM 124的行1的字511的乘积与累加器202的值累加。

[0126] 在时钟3处,512个NPU 126中的每一个NPU 126执行图4中地址2处的乘法累加轮转指令的第二次迭代。如图所示,NPU 0将从NPU 511的复用寄存器208的输出209接收到的轮

转数据字211 (即从数据RAM 122接收到的数据字510) 与权重RAM 124的行2的字0的乘积与累加器202的值累加;NPU 1将从NPU 0的复用寄存器208的输出209接收到的轮转数据字211 (即从数据RAM 122接收到的数据字511) 与权重RAM 124的行2的字1的乘积与累加器202的值累加;依此类推,NPU 511将从NPU 510的复用寄存器208的输出209接收到的轮转数据字211 (即从数据RAM 122接收到的数据字509) 与权重RAM 124的行2的字511的乘积与累加器202的值累加。如图5的省略号所示,接下来的509个时钟周期各自依此持续进行,直到时钟512。

[0127] 在时钟512处,512个NPU 126中的每一个NPU 126执行图4中地址2处的乘法累加轮转指令的第511次迭代。如图所示,NPU 0将从NPU 511的复用寄存器208的输出209接收到的轮转数据字211 (即从数据RAM 122接收到的数据字1) 与权重RAM 124的行511的字0的乘积与累加器202的值累加;NPU 1将从NPU 0的复用寄存器208的输出209接收到的轮转数据字211 (即从数据RAM 122接收到的数据字2) 与权重RAM 124的行511的字1的乘积与累加器202的值累加;依此类推,NPU 511将从NPU 510的复用寄存器208的输出209接收到的轮转数据字211 (即从数据RAM 122接收到的数据字0) 与权重RAM 124的行511的字511的乘积与累加器202的值累加。在一个实施例中,需要多个时钟周期从数据RAM 122与权重RAM 124读取数据字与权重字以执行图4中地址1处的乘法累加指令;然而,数据RAM 122、权重RAM 124和NPU 126是流水线式的,使得一旦第一乘法累加运算开始 (例如,如图5的时钟1期间所示),就在连续的的时钟周期内开始后续的乘法累加运算 (例如,如时钟2-512期间所示)。优选地,响应于利用架构指令 (例如MTNN或MFNN指令,在后续针对图14和图15进行说明) 或架构指令转译成的微指令对于数据RAM 122和/或权重RAM 124的访问,NPU 126可以暂时搁置。

[0128] 在时钟513处,512个NPU 126中的每一个NPU 126的AFU 212执行图4中地址3处的激活函数指令。最后,在时钟514处,这512个NPU 126中的每一个NPU 126通过将结果133写回数据RAM 122的行16中的相应字,即将NPU 0的结果133写入数据RAM 122的字0,将NPU 1的结果133写入数据RAM 122的字1,依此类推,直到将NPU 511的结果133写入数据RAM 122的字511,来执行图4的地址4处的写入AFU输出指令。以上针对图5所述的运算也在图6A中以框图的形式示出。

[0129] 现参考图6A,示出一框图,该框图示出图1的NNU 121对图4的程序的执行。NNU 121包括512个NPU 126、接收地址输入123的数据RAM 122、以及接收地址输入125的权重RAM 124。尽管未示出,但在时钟0处,512个NPU 126执行初始化指令。如图所示,在时钟1处,行17的512个16位数据字被从数据RAM 122读出并提供至512个NPU 126。在时钟1至512处,行0至511的512个16位权重字分别被从权重RAM 124读出并提供至512个NPU 126。尽管未示出,但在时钟1处,512个NPU 126对加载的数据字和权重字执行相应的乘法累加运算。在时钟2至512处,512个NPU 126的复用寄存器208作为512个16位字的轮转器操作以将先前加载的数据RAM 122的行17的数据字轮转至相邻的NPU 126,并且NPU 126对轮转后的各数据字以及从权重RAM 124加载的各权重字执行乘法累加运算。尽管未示出,但在时钟513处,512个AFU 212执行激活指令。在时钟514处,512个NPU 126将相应的512个16位结果133写回数据RAM 122的行16。

[0130] 可以发现,产生结果字 (神经元输出) 并写回数据RAM 122或权重RAM 124所需的时钟数大致为神经网络的当前层所接收到的数据输入 (连接) 数量的平方根。例如,如果当前

层包括各自具有来自先前层的512个连接的512个神经元,则这些连接的总和是256K,并且产生当前层的结果所需的时钟数略超过512。因此,NNU 121为神经网络计算提供了极高的性能。

[0131] 现参考图6B,示出一流程图,该流程图示出图1的处理器100执行架构程序的操作,所述架构程序使用NNU 121来执行典型地与人工神经网络的隐藏层的神经元相关联的乘法累加激活函数计算(诸如图4的程序所执行的运算等)。图6B的示例假定四个隐藏层(通过方框602的NUM_LAYERS变量的初始化来标示)的计算,各个隐藏层具有512个神经元,各个神经元连接先前层全部的512个神经元(通过图4的程序)。然而,应当理解,这些层和神经元的数量是为了说明目的而选择的,并且NNU 121可用于针对不同数量的隐藏层、每一层中不同数量的神经元以及未全部连接的神经元执行相同的计算。在一个实施例中,对于这一层中不存在的神经元或者至神经元的不存在的连接,权重值可被设置为零。优选地,架构程序将第一组权重写入权重RAM 124并启动NNU 121,并且在NNU 121正执行与第一层相关联的计算时,此架构程序将第二组权重写入权重RAM 124,使得一旦NNU 121完成第一隐藏层的计算,NNU 121就可以开始第二层的计算。如此,架构程序往返于权重RAM 124的两个区域之间,以确保NNU 121被充分利用。流程始于方框602。

[0132] 在方框602处,如针对图6A所示和所述,处理器100(即运行在处理器100上的架构程序)将输入值写入数据RAM 122的当前神经元隐藏层,例如写入数据RAM 122的行17。可选地,这些值也可能已经在数据RAM 122的行17中作为NNU 121针对先前层(例如,卷积、池化或输入层)的运算结果133。此外,架构程序将变量N初始化为值1。变量N标示隐藏层中正由NNU 121处理的当前层。此外,架构程序将变量NUM_LAYERS初始化为值4,这是因为在本示例中存在四个隐藏层。流程进入方框604。

[0133] 在方框604处,如图6A所示,处理器100将层1的权重字写入权重RAM 124,例如写入行0至511。流程进入方框606。

[0134] 在方框606处,处理器100使用指定对程序存储器129进行写入的函数1432的MTNN指令1400,将(例如,图4的)乘法累加激活函数程序写入NNU 121的程序存储器129。处理器100随后利用指定开始执行程序的函数1432的MTNN指令1400来启动NNU程序。流程进入决策方框608。

[0135] 在决策方框608处,架构程序确定变量N的值是否小于NUM_LAYERS。如果是,则流程进入方框612;否则进入方框614。

[0136] 在方框612处,处理器100将层N+1的权重字写入权重RAM 124,例如写入行512至1023。因此,有利地,架构程序在NNU 121正执行当前层的隐藏层计算时将下一层的权重字写入权重RAM 124,使得一旦完成当前层的计算,即写入数据RAM 122后,NNU 121就可以立刻开始执行下一层的隐藏层计算。流程进入方框614。

[0137] 在方框614处,处理器100确定为(在层1的情况下,在方框606处开始的,在层2至4的情况下,则是在方框处618开始的)当前运行的NNU程序已经完成。优选地,处理器100通过执行MFNN指令1500读取NNU 121的状态寄存器127来对此进行确定。在备选实施例中,NNU 121产生一中断以表示其已经完成乘法累加激活函数层程序。流程进入决策方框616。

[0138] 在决策方框616处,架构程序确定变量N的值是否小于NUM_LAYERS。如果是,流程进入方框618;否则流程进入方框622。

[0139] 在方框618处,处理器100更新乘法累加激活函数程序,使得该处理器可以执行层N+1的隐藏层计算。更具体地,处理器100将图4的地址1处的乘法累加指令的数据RAM 122的行值更新为先前层的结果所写入的数据RAM 122行(例如,更新为行16),并且还更新输出行(例如,更新为行15)。处理器100随后开始更新后的NNU程序。可选地,图4的程序在地址4的输出指令中指定与地址1处的乘法累加指令所指定的行(即,从数据RAM 122读取的行)相同的行。在该实施例中,输入数据字的当前行被覆写(由于此行数据字已经被读入复用寄存器208并经由N字轮转器在这些NPU 126间进行轮转,因此只要这行数据字无需用于其它目的,这样的处理方式就是可接受的)。在这种情况下,在方框618处,无需更新NNU程序,而只需要重新开始NNU程序。流程进入方框622。

[0140] 在方框622处,处理器100从数据RAM 122读取层N的NNU程序的结果。然而,如果这些结果仅被用于下一层,则架构程序就无需从数据RAM 122读取这些结果,作为代替可将其保留在数据RAM 122中以用于下一个隐藏层计算。流程进入决策方框624。

[0141] 在决策方框624处,架构程序确定变量N的值是否小于NUM_LAYERS。如果是,则流程进入方框626;否则流程结束。

[0142] 在方框626处,架构程序使N加一。流程返回决策方框608。

[0143] 如可以从图6B的示例中确定,大致上每512个时钟周期,NPU 126(借助于图4的NNU程序的操作)对数据RAM 122执行一次读取和一次写入。此外,NPU 126大致每时钟周期对权重RAM 124进行读取以读取一行权重字。因此,权重RAM 124的全部带宽都被NNU 121执行隐藏层运算所用的混合方式消耗。此外,假定实施例包括写入和读取缓存器(诸如图17的缓存器1704等),与NPU 126进行读取并行地,处理器100写入权重RAM 124,使得缓存器1704大致每16个时钟周期对权重RAM 124执行一次写入以写入权重字。因此,在权重RAM 124的单一端口的实施例中(诸如针对图17所述的实施例等),大致每16个时钟周期,NPU 126就必须暂时搁置对权重RAM 124的读取,从而使得缓存器1704能够对权重RAM 124进行写入。然而,在权重RAM 124为双端口的实施例中,NPU 126无需搁置。

[0144] 现参考图7,示出一框图,该框图示出根据备选实施例的图1的NPU 126。图7的NPU 126在许多方面与图2的NPU 126类似。然而,图7的NPU 126另外包括第二输入复用寄存器705。该复用寄存器705选择输入206或711其中之一,以储存在寄存器中然后在后续时钟周期提供于输出203上。输入206从权重RAM 124接收权重字。另一输入711接收相邻NPU 126的第二复用寄存器705的输出203。优选地,NPU J的复用寄存器705的输入711接收NPU 126实例J-1的复用寄存器705的输出203,并且NPU J的输出被提供至NPU 126实例J+1的复用寄存器705的输入711。如此,与以上针对图3所述的方式相同,N个NPU 126的复用寄存器705整体操作为N字轮转器,但是针对的是权重字而非数据字。控制输入713控制这两个输入中的哪一个被复用寄存器705选择,以储存于寄存器中并在后续提供于输出203上。

[0145] 包括复用寄存器208和/或复用寄存器705(以及诸如图18和图23所示等的其它实施例中的复用寄存器)以实际上形成用于将从数据RAM 122和/或权重RAM 124接收到的一行数据/权重进行轮转的大型轮转器具有如下的优点:NNU 121不需要其他方面所需的在数据RAM 122和/或权重RAM 124之间的极大复用器来向适当的NNU 121提供必要的数据字/权重字。

[0146] 除激活函数结果外还写回累加器值

[0147] 在一些应用中,处理器100接收回(例如经由图15的MFNN指令接收至媒体寄存器118)原始的累加器202的值217是有用的,其中在其它执行单元112上执行的指令可以对这些累加器202的值217执行计算。例如,在一个实施例中,为了降低AFU 212的复杂度,AFU 212不被配置为执行软极大激活函数。因此,NNU 121可以将原始的累加器202的值217或其子集输出至数据RAM 122或权重RAM 124,而架构程序随后从数据RAM 122或权重RAM 124读取该原始的累加器202的值217或其子集并对原始值进行计算。然而,对原始的累加器202的值217的应用不限于软极大运算的执行,还可以设想其它应用。

[0148] 现参考图8,示出一框图,该框图示出根据备选实施例的图1的NPU 126。图8的NPU 126在许多方面与图2的NPU 126类似。然而,图8的NPU 126在AFU 212内包括复用器(mux)802,其中AFU 212具有控制输入803。累加器202的宽度(以位为单位)大于数据字的宽度。复用器802具有用于接收累加器202的输出217的数据字宽度部分的多个输入。在一个实施例中,累加器202的宽度为41位,而NPU 126被配置为输出16位的结果字133;因此,例如,复用器802(或图30的复用器3032和/或复用器3037)具有分别用于接收累加器202的输出217的位[15:0]、位[31:16]与位[47:32]的三个输入。优选地,并非由累加器202提供的输出位(例如位[47:41])被强制设定为零值位。

[0149] 响应于写入ACC指令(例如下述的图9的地址3至5处的写入ACC指令等),定序器128在控制输入803上产生值以控制复用器802选择累加器202的字(例如,16位)其中之一。优选地,复用器802还具有用于接收激活函数电路(例如,图30中的元件3022、3024、3026、3018、3014与3016)的输出的一个或多个输入,其中这些激活函数电路产生作为数据字的宽度的输出。响应于诸如图4的地址4处的写入AFU输出指令等的指令,定序器128在控制输入803上产生一个值以控制复用器802选择这些激活函数电路输出其中之一,而非累加器202的字其中之一。

[0150] 现参考图9,示出一表格,该表格示出用于储存于图1的NNU 121的程序存储器129中并由该NNU 121执行的程序。图9的示例性程序在许多方面与图4的程序类似。具体地,地址0至2处的指令是相同的。然而,图4的地址3与4处的指令在图9中被替换为写入ACC指令,其指示512个NPU 126将其累加器202的输出217作为结果133写回数据RAM 122的三行(在本示例中为行16至18)。也就是说,写入ACC指令指示定序器128在第一时钟周期内输出值为16的数据RAM地址123以及写入命令,在第二时钟周期内输出值为17的数据RAM地址123以及写入命令,并且在第三时钟周期内输出值为18的数据RAM地址123以及写入命令。优选地,写入ACC指令的执行可能与其它指令的执行重叠,使得写入ACC指令实际在三个时钟周期内执行,其中针对写入数据RAM 122的每一行为一个时钟周期。在一个实施例中,用户指定激活函数2934和(图29A的)控制寄存器127中的输出命令2956字段的值,以完成将累加器202的期望部分写入数据RAM 122或权重RAM 124。可选地,写入ACC指令可以可选地将累加器202的子集写回,而非将累加器202的全部内容写回。在一个实施例中,如以下针对图29至图31更详细所述,可以将标准型累加器202写回。

[0151] 现参考图10,示出一时序图,该时序图示出NNU 121对图9的程序的执行。图10的时序图与图5的时序图类似,并且时钟0至512是相同的。然而,在时钟513-515处,512个NPU 126中的每个NPU 126的AFU 212执行图9的地址3至5处的写入ACC指令其中之一。具体地,在时钟513处,512个NPU 126中的每一个NPU 126将累加器202的输出217的位[15:0]作为结果

133写回数据RAM 122的行16中的相应字;在时钟514处,512个NPU 126中的每一个NPU 126将累加器202的输出217的位[31:16]作为结果133写回数据RAM 122的行17中的相应字;而在时钟515处,512个NPU 126中的每一个NPU 126将累加器202的输出217的位[40:32]作为结果133写回数据RAM 122的行18中的相应字。优选地,位[47:41]被强制设定为零值。

[0152] 共享AFU

[0153] 现参考图11,示出一框图,该框图示出图1的NNU 121的实施例。在图11的实施例中,神经元分成两部分,即激活函数单元部分与ALU部分(ALU部分还包含移位寄存器部分),并且各个激活函数单元部分由多个ALU部分共享。在图11中,ALU部分是指NPU 126,而共享的激活函数单元部分是指AFU 1112。这与图2的实施例形成对比,例如,在图2的实施例中,各个神经元包含其自身的AFU 212。因此,例如,在一个实施例中,图11的实施例的NPU 126(ALU部分)包括图2的累加器202、ALU 204、复用寄存器208与寄存器205,但不包括AFU 212。在图11的实施例中,NNU 121作为示例而包括512个NPU 126;然而,设想了具有其它数量的NPU 126的其它实施例。在图11的示例中,这512个NPU 126被分组成64个组(在图11中被称为组0至63),并且每个组具有8个NPU 126。

[0154] NNU 121还包括行缓存器1104以及耦接在NPU 126和行缓存器1104之间的多个共享AFU 1112。行缓存器1104的宽度(以位为单位)与数据RAM 122或权重RAM 124的行相同,例如为512个字。针对每一个NPU 126组存在一个AFU 1112,即,每个AFU 1112具有相应的NPU 126组;因此,在图11的实施例中,存在与64个NPU 126组相对应的64个AFU 1112。组内的8个NPU 126中的各NPU 126共享相应的AFU 1112。设想了具有不同数量的AFU 1112以及每组中具有不同数量的NPU 126的其他实施例。例如,设想了组中的两个、四个或十六个NPU 126共享AFU 1112的其它实施例。

[0155] 共享AFU 1112的动机是为了缩减NNU 121的大小。大小缩减是以性能降低为代价而获得的。也就是说,例如,如下图12所展示那样,根据共享率可能需要更长的若干时钟来产生整个NPU 126阵列的结果133,在这种情况下,由于8:1的共享率,因而需要七个额外的时钟周期。然而,一般而言,与产生累加和所需的时钟数(例如,对于每个神经元具有512个连接的层,需要512个时钟)相比,前述额外的时钟数(例如7个)相对较少。因此,相对较小的性能影响(例如,增加百分之一的计算时间)对于NNU 121的大小缩减而言可以是一个合算的妥协。

[0156] 在一个实施例中,每一个NPU 126包括AFU 212,其中AFU 212用于执行相对简单的激活函数,从而使得这些简单的AFU 212能够相对较小并因此能被包含在每个NPU 126内;而共享的或复杂的AFU 1112执行相对复杂的激活函数,因此相对显著地大于简单的AFU 212。在这样的实施例中,只有在指定需要共享复杂AFU 1112的复杂激活函数的情况下才需要额外的时钟周期,而在指定由简单AFU 212配置执行的激活函数的情况下则不需要。

[0157] 现参考图12与图13,示出两个时序图,这些时序图示出图11的NNU 121对图4的程序的执行。图12的时序图与图5的时序图类似,并且时钟0至512相同。然而,在时钟513处,运算与图5的时序图中所述的运算不同,这是因为图11的NPU 126共享AFU 1112;即,组中的NPU 126共享与该组相关联的AFU 1112,并且图11示出共享。

[0158] 图13的时序图的每一行与第一列中所指示的连续时钟周期相对应。其它各列与64个AFU 1112中不同的AFU 1112相对应并指示其运算。为了简单清楚地进行图示,仅示出AFU

0、1和63的运算。图13的时钟周期与图12的时钟周期相对应,但以不同的方式示出NPU 126对AFU 1112的共享。如图13所示,在时钟0~512处,64个AFU 1112中的每一个AFU 1112都是处于不活动状态,而NPU 126执行初始化NPU指令、乘法累加指令以及乘法累加轮转指令。

[0159] 如图12和图13这两者所示,在时钟513处,AFU 0(与组0相关联的AFU 1112)开始对NPU 0(即组0中的第一个NPU 126)的累加器202的值217执行指定的激活函数,而AFU 0的输出将被储存至行缓存器1104的字0。同样在时钟513处,各AFU 1112都开始对相应的NPU 126组中的第一个NPU 126的累加器202执行指定的激活函数。因此,如图13所示,在时钟513处,AFU 0开始对NPU 0的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字0的结果;AFU 1开始对NPU 8的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字8的结果;依此类推,AFU 63开始对NPU 504的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字504的结果。

[0160] 如图所示,在时钟514处,AFU 0(与组0相关联的AFU 1112)开始对NPU 1(即组0中的第二个NPU 126)的累加器202的值217执行指定的激活函数,并且AFU 0的输出将被储存至行缓存器1104的字1。同样在时钟514处,各AFU 1112都开始对相应的NPU 126组中的第二个NPU 126的累加器202执行指定的激活函数。因此,如图13所示,在时钟514处,AFU 0开始对NPU 1的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字1的结果;AFU 1开始对NPU 9的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字9的结果;依此类推,AFU 63开始对NPU 505的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字505的结果。如图所示,这种模式持续直到时钟周期520为止,AFU 0(与组0相关联的AFU 1112)开始对NPU 7(即组0中的第八个(最后一个)NPU 126)的累加器202的值217执行指定的激活函数,并且AFU 0的输出将被储存至行缓存器1104的字7。同样在时钟520处,各AFU 1112都开始对相应的NPU 126组中的第八个NPU 126的累加器202执行所指定的激活函数。因此,如图13所示,在时钟520处,AFU 0开始对NPU 7的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字7的结果;AFU 1开始对NPU 15的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字15的结果;依此类推,AFU 63开始对NPU 511的累加器202执行指定的激活函数以产生将被储存至行缓存器1104的字511的结果。

[0161] 在时钟521处,一旦与512个NPU 126相关联的全部512个结果都已经产生并写入行缓存器1104,行缓存器1104就开始将其内容写入数据RAM 122或权重RAM 124。如此,64个NPU 126组中的每一个组内的AFU 1112都执行图4的地址3处的激活函数指令的一部分。

[0162] 如以下例如针对图29A至图33更详细所述,在ALU 204组之间共享AFU 1112的实施例(诸如图11中的实施例等)与整数ALU 204结合可以是特别有利的。

[0163] MTNN与MFNN架构指令

[0164] 现参考图14,示出一框图,该框图示出向神经网络移动(MTNN)架构指令1400以及该架构指令相对于图1的NNU 121的部分的操作。MTNN指令1400包括操作码(opcode)字段1402、src1字段1404、src2字段1406、gpr字段1408以及即时字段1412。MTNN指令1400为架构指令,即,该指令包含在处理器100的指令集架构内。优选地,指令集架构将操作码字段1402的预定值与MINN指令1400进行关联,来区分MTNN指令1400与指令集架构中的其它指令。MTNN指令1400的操作码1402可以包括或不包括诸如常见于x86架构中的前置码

(prefix)。

[0165] 即时字段1412提供用于向NNU 121的控制逻辑1434指定函数1432的值。优选地,函数1432被提供作为图1的微指令105的即时操作数。可以由NNU 121执行的函数1432包括但不限于写入数据RAM 122、写入权重RAM 124、写入程序存储器129、写入控制寄存器127、开始执行程序存储器129内的程序、暂停执行程序存储器129内的程序、完成执行程序存储器129内的程序的请求通知(例如中断)、以及重设NNU 121。优选地,NNU指令集包括其结果表示NNU程序已完成的指令。可选地,NNU指令集包括明确的产生中断的指令。优选地,对NNU 121进行重设包括除了数据RAM 122、权重RAM 124、程序存储器129的内容维持完整不动外,有效地强制NNU 121回到重设状态(例如,清空内部状态机并将其设定为空闲状态)。此外,诸如累加器202等的内部寄存器不会受到重设函数的影响,并且必须被明确地清空,例如使用图4的地址0处的初始化NPU指令来清空。在一个实施例中,函数1432可包括直接执行函数,其中第一源寄存器包含微操作(例如参见图34的微操作3418)。该直接执行函数指示NNU 121直接执行指定的微操作。如此,架构程序可以直接控制NNU 121来执行运算,而非将指令写入程序存储器129并于后续指示NNU 121执行程序存储器129内的指令或者借助于MTNN指令1400(或图15的MFNN指令1500)的执行。图14示出写入数据RAM 122的函数1432的示例。

[0166] gpr字段1408指定通用寄存器文件116内的一个GPR。在一个实施例中,每个GPR均为64位。如图所示,通用寄存器文件116将来自所选定的GPR的值提供至NNU 121,NNU 121将该值用作地址1422。地址1422选择函数1432中指定的存储器的行。在数据RAM 122或权重RAM 124的情况下,地址1422另外选择一数据块,该数据块的大小是选定行内的媒体寄存器(例如,512位)的位置的大小的两倍。优选地,该位置在512位边界上。在一个实施例中,复用器选择地址1422(或者在下述的MFNN指令1400的情况下的地址1422)或来自定序器128的地址123/125/131以提供至数据RAM 122/权重RAM 124/程序存储器129。在一个实施例中,如下更详细所述,数据RAM 122为双端口的,从而使得NPU 126能够与媒体寄存器118读取/写入数据RAM 122并行地对数据RAM 122进行读取/写入。在一个实施例中,为了类似的目的,权重RAM 124也是双端口的。

[0167] src1字段1404与src2字段1406各自指定媒体寄存器文件118中的媒体寄存器。在一个实施例中,每个媒体寄存器118均为256位。如图所示,媒体寄存器文件118将来自选定的媒体寄存器的串接数据(例如,512位)提供至数据RAM 122(或权重RAM 124或程序存储器129),以写入地址1422所指定的选定行1428并写入选定行1428中由地址1422指定的位置。有利地,通过执行一系列MTNN指令1400(以及下述的MFNN指令1500),处理器100上执行的架构程序可以填充数据RAM 122的行和权重RAM 124的行,并将诸如本文所述的(例如,图4和图9的)程序等的程序写入程序存储器129,以使NNU 121以非常快的速度对数据和权重执行运算,从而实现人工神经网络。在一个实施例中,架构程序直接控制NNU 121而非将程序写入程序存储器129。

[0168] 在一个实施例中,MTNN指令1400指定起始源寄存器和源寄存器的数量即Q,而非指定两个源寄存器(例如,1404和1406)。这种形式的MTNN指令1400指示处理器100将指定为起始源寄存器的媒体寄存器118以及接下来的Q-1个后续媒体寄存器118写入NNU 121,即写入所指定的数据RAM 122或权重RAM 124。优选地,指令转译器104将MTNN指令1400转译为写入所有Q个指定媒体寄存器118所需的数量的微指令。例如,在一个实施例中,当MTNN指令1400

将起始源寄存器指定为MR4并且Q为8的情况下,指令转译器104将MTNN指令1400转译为四个微指令,其中第一微指令写入MR4与MR5,第二微指令写入MR6与MR7,第三微指令写入MR8与MR9,而第四微指令写入MR10与MR11。在从媒体寄存器118至NNU 121的数据路径是1024位而非512位的备选实施例中,指令转译器104将MTNN指令1400转译为两个微指令,其中第一微指令写入MR4至MR7,第二微指令写入MR8至MR11。设想了如下的类似实施例,其中MFNN指令1500指定起始目的地寄存器以及目的地寄存器的数量,以使得每一个MFNN指令1500能够读取数据RAM 122或权重RAM 124的行中的大于单个媒体寄存器118的数据块。

[0169] 现参考图15,示出一框图,该框图示出从神经网络移动(MTNN)架构指令1500以及该架构指令相对于图1的NNU 121的部分的操作。MFNN指令1500包括操作码字段1502、dst字段1504、gpr字段1508以及即时字段1512。MFNN指令1500为架构指令,即该指令包含于处理器100的指令集架构内。优选地,该指令集架构将操作码字段1502的预定值与MFNN指令1500进行关联,以区分MFNN指令1500与指令集架构内的其它指令。MFNN指令1500的操作码1502可以包括或不包括诸如常见于x86架构中的前置码。

[0170] 即时字段1512提供用于向NNU 121的控制逻辑1434指定函数1532的值。优选地,函数1532被提供作为图1的微指令105的即时操作数。可以由NNU 121执行的函数1532包括但不限于读取数据RAM 122、读取权重RAM 124、读取程序存储器129、以及读取状态寄存器127。图15示出读取数据RAM 122的函数1532的示例。

[0171] gpr字段1508指定通用寄存器文件116内的一个GPR。如图所示,通用寄存器文件116将来自所选定的GPR的值提供至NNU 121,其中NNU 121将该值用作地址1522并以类似于图14的地址1422的方式操作,以选择函数1532中指定的存储器的行,并且在数据RAM 122或权重RAM 124的情况下,地址1522另外选择一数据块,该数据块大小是选定行内的媒体寄存器(例如,256位)的位置的大小。优选地,该位置在256位边界上。

[0172] dst字段1504指定媒体寄存器文件118中的媒体寄存器。如图所示,媒体寄存器文件118将数据(例如,256位)从数据RAM 122(或权重RAM 124或程序存储器129)接收至选定的媒体寄存器,此数据读取自地址1522所指定的选定行1528以及选定行1528中的地址1522所指定的位置。

[0173] NNU内部RAM端口配置

[0174] 现参考图16,示出一框图,该框图示出图1的数据RAM 122的实施例。数据RAM 122包括存储器阵列1606、读取端口1602以及写入端口1604。存储器阵列1606保持数据字,并且如上所述优选地布置成D行,每行N个字。在一个实施例中,存储器阵列1606包括64个水平布置的静态RAM单元的阵列(其中每个单元具有128位的宽度和64位的高度)以提供宽度为8192位并且具有64行的64KB数据RAM 122,并且数据RAM 122所占用的晶粒面积大致为0.2平方毫米。然而,设想了其它实施例。

[0175] 读取端口1602优选采用复用方式耦接至NPU 126和媒体寄存器118。(更精确地,媒体寄存器118可以经由结果总线耦接至读取端口1602,其中结果总线也可以将数据提供至重排序缓存器和/或至其它执行单元112的结果转发总线。)NPU 126与媒体寄存器118共享读取端口1602,以对数据RAM 122进行读取。写入端口1604也优选采用复用方式耦接至NPU 126以及媒体寄存器118。NPU 126与媒体寄存器118共享写入端口1604,以写入数据RAM 122。因此,有利地,媒体寄存器118可以在NPU 126正从数据RAM 122进行读取的同时并行地

写入数据RAM 122,或者NPU 126可以在媒体寄存器118正在从数据RAM 122进行读取的同时并行地写入数据RAM 122。这样可以有利地提供改进的性能。例如,NPU 126可以读取数据RAM 122(例如持续执行计算),同时媒体寄存器118可以将更多数据字写入数据RAM 122。再例如,NPU 126可以将计算结果写入数据RAM 122,同时媒体寄存器118从数据RAM 122读取计算结果。在一个实施例中,NPU 126可以将一行计算结果写入数据RAM 122,同时NPU 126还从数据RAM 122读取一行数据字。在一个实施例中,存储器阵列1606被配置成存储器区块(bank)。在NPU 126访问数据RAM 122时,所有的存储器区块都被激活以访问存储器阵列1606的整个行;而在媒体寄存器118访问数据RAM 122时,只有所指定的存储器区块会被激活。在一个实施例中,每个存储器区块的宽度为128位,而媒体寄存器118的宽度为256位,因此例如,每次媒体寄存器118访问时激活两个存储器区块。在一个实施例中,端口1602/1604其中之一为读取/写入端口。在一个实施例中,端口1602/1604两者都是读取/写入端口。

[0176] 如本文所述的NPU 126的轮转器能力的优点在于:与为了确保NPU 126被高度利用而要求在NPU 126执行计算的同时架构程序(经由媒体寄存器118)能够持续提供数据至数据RAM 122并且从数据RAM 122检索结果所需的存储器阵列相比,该轮转器能力有助于使数据RAM 122的存储器阵列1606的行显著减少,因而使该阵列相对小得多。

[0177] 内部RAM缓存器

[0178] 现参考图17,示出一框图,该框图示出图1的权重RAM 124与缓存器1704的实施例。权重RAM 124包括存储器阵列1706与端口1702。存储器阵列1706保持权重字,并且如上所述优选地布置成W行,每行具有N个字。在一个实施例中,存储器阵列1706包括128个水平布置的静态RAM单元的阵列(其中每个单元具有64位的宽度和2048位的高度)以提供宽度为8192位并且具有2048行的2MB权重RAM 124,并且权重RAM 124所占用的晶粒面积大致为2.4平方毫米。然而,设想了其它实施例。

[0179] 端口1702优选采用复用方式耦接至NPU 126和缓存器1704。NPU 126与缓存器1704经由该端口1702读取并写入权重RAM 124。缓存器1704还耦接至图1的媒体寄存器118,使得媒体寄存器118通过缓存器1704读取并写入权重RAM 124。因此,有利地,在NPU 126正在读取或写入权重RAM 124的同时,媒体寄存器118也可以并行地写入或读取缓存器1704(但是如果NPU 126当前正执行,优选地搁置NPU 126,以避免在缓存器1704访问权重RAM 124的同时访问该权重RAM 124)。这样可以有利地提升性能,特别是因为媒体寄存器118对于权重RAM 124的读取和写入相对于NPU 126对于权重RAM 124的读取和写入小得多。例如,在一个实施例中,NPU 126一次读取/写入8192位(一行),而媒体寄存器118的宽度为256位并且每个MTNN指令1400写入两个媒体寄存器118,即512位。因此,在架构程序执行十六个MTNN指令1400以填充缓存器1704的情况下,NPU 126和架构程序针对访问权重RAM 124而发生冲突的时间仅不到时间的约百分之六。在另一实施例中,指令转译器104将MTNN指令1400转译为两个微指令105,其中每个微指令105将单个数据寄存器118写入缓存器1704,在这种情况下,NPU 126和架构程序针对访问权重RAM 124而发生冲突的频率甚至更小。

[0180] 在包含缓存器1704的实施例中,利用架构程序写入权重RAM 124需要多个MTNN指令1400。一个或多个MTNN指令1400指定函数1432以写入缓存器1704中指定的数据块,随后MTNN指令1400指定函数1432以指示NNU 121将缓存器1704的内容写入权重RAM 124的指定行,其中数据块的大小是媒体寄存器118的位数的两倍、并且这些数据块自然地在缓存器

1704内对齐。在一个实施例中,在用于指定函数1432以对缓存器1704的指定数据块进行写入的各MTNN指令1400中,包含具有与缓存器1704的各数据块相对应的位的位掩码(bitmask)。来自两个指定源寄存器118的数据被写入缓存器1704中的设置了位掩码内的相应位的各数据块中。这对于权重RAM 124的行内的重复数据值而言可以是有用的。例如,为了将缓存器1704(以及后续的权重RAM 124的行)归零,程序设计者可以为源寄存器加载零值并且设置位掩码的所有位。此外,位掩码使得程序设计者能够仅写入缓存器1704中的选定数据块,从而保留其它数据块中的先前数据。

[0181] 在包含缓存器1704的一个实施例中,利用架构程序读取权重RAM 124需要多个MFNN指令1500。初始的MFNN指令1500指定函数1532以从权重RAM 124的指定行加载缓存器1704,随后一个或多个MFNN指令1500指定函数1532以将缓存器1704的指定数据块读取至目的地寄存器,其中数据块的大小是媒体寄存器118的位数,并且这些数据块自然地在缓存器1704内对齐。设想了如下的其它实施例,其中权重RAM 124包括多个缓存器1704,以通过增加NPU 126执行时架构程序的可访问数量来进一步减少NPU 126和架构程序间对访问权重RAM 124的竞争,这样可以增加在NPU 126无需访问权重RAM 124的时钟周期期间能够执行缓存器1704的访问的可能性。

[0182] 尽管图16描述了双端口数据RAM 122,但设想了权重RAM 124也是双端口的其它实施例。此外,尽管图17描述了缓存器用于权重RAM 124,但设想了数据RAM 122也具有与缓存器1704类似的关联缓存器的其它实施例。

[0183] 可动态配置的NPU

[0184] 现参考图18,示出一框图,该框图示出图1的可动态配置的NPU 126。图18的NPU 126在许多方面与图2的NPU 126类似。然而,图18的NPU 126可动态配置以在两个不同配置其中之一中操作。在第一个配置中,图18的NPU 126的操作类似于图2的NPU 126。也就是说,在第一个配置(在本文中称为“宽”配置或“单个”配置)中,NPU 126的ALU 204对单个宽数据字和单个宽权重字(例如,16位)执行运算以产生单个宽结果。相比之下,在第二个配置(在本文中称为“窄”配置或“双”配置)中,NPU 126对两个窄数据字和两个相应的窄权重字(例如,8位)执行运算以产生两个相应的窄结果。在一个实施例中,NPU 126的配置(宽的或窄的)由初始化NPU指令(例如,下述图20中的地址0的指令)进行。可选地,该配置也可以由MTNN指令实现,其中该MTNN指令的函数1432指定将NPU 126配置为所述配置(宽或窄)。优选地,由程序存储器129指令或确定配置(宽或窄)的MTNN指令填充配置寄存器。例如,配置寄存器的输出被提供给ALU 204、AFU 212以及产生复用寄存器控制信号213的逻辑。一般而言,图18的NPU 126的元件与图2中相同附图标记的元件执行类似的功能,并且为了理解图18,应当进行参照。然而,现将针对图18的实施例(包括与图2的不同处)进行说明。

[0185] 图18的NPU 126包括两个寄存器205A与205B、两个3输入复用寄存器208A与208B、ALU 204、两个累加器202A与202B、以及两个AFU 212A与212B。各寄存器205A/205B分别具有图2的寄存器205的宽度的一半(如8位)。各寄存器205A/205B从权重RAM 124接收相应的窄权重字206A/B206(例如8位)并在后续时钟周期内将其输出203A/203B提供至ALU 204的操作数选择逻辑1898。在NPU 126为宽配置的情况下,与图2的实施例的寄存器205的方式类似地,寄存器205A/205B实际上一同操作以接收来自权重RAM 124的宽权重字206A/206B(例如16位);以及在NPU 126为窄配置的情况下,寄存器205A/205B实际上独立地操作,从而各自

接收来自权重RAM 124的窄权重字206A/206B(例如8位),以使得NPU 126实际上为两个单独的窄NPU。然而,权重RAM 124的相同输出位都耦接并提供至寄存器205A/205B,而与NPU 126的配置无关。例如,NPU 0的寄存器205A接收字节0,NPU 0的寄存器205B接收字节1,NPU 1的寄存器205A接收字节2,NPU 1的寄存器205B接收字节3,依此类推NPU 511的寄存器205B接收字节1023。

[0186] 各复用寄存器208A/208B分别具有图2的寄存器208的宽度的一半(例如8位)。复用寄存器208A选择其输入207A、211A与1811A其中之一以储存在其寄存器中并在后续时钟周期内提供在输出209A上,并且复用寄存器208B选择其输入207B、211B与1811B其中之一以储存在其寄存器中并在后续时钟周期内在输出209B上提供至操作数选择逻辑1898。输入207A从数据RAM 122接收窄数据字(例如8位),并且输入207B从数据RAM 122接收窄数据字。在NPU 126为宽配置的情况下,与图2的实施例的复用寄存器208的方式类似地,复用寄存器208A/208B实际上一同操作以接收来自数据RAM 122的宽数据字207A/207B(例如16位);在NPU 126为窄配置的情况下,复用寄存器208A/208B实际上独立地操作,从而各自接收来自数据RAM 122的窄数据字207A/207B(例如8位),以使得NPU 126实际上为两个单独的窄NPU。然而,数据RAM 122的相同输出位都耦接并提供至复用寄存器208A/208B,而与NPU 126的配置无关。例如,NPU 0的复用寄存器208A接收字节0,NPU 0的复用寄存器208B接收字节1,NPU 1的复用寄存器208A接收字节2,NPU 1的复用寄存器208B接收字节3,依此类推NPU 511的复用寄存器208B接收字节1023。

[0187] 输入211A接收相邻NPU 126的复用寄存器208A的输出209A,并且输入211B接收相邻NPU 126的复用寄存器208B的输出209B。如图所示,输入1811A接收相邻NPU 126的复用寄存器208B的输出209B,并且输入1811B接收当前NPU 126的复用寄存器208A的输出209A。在图1所示的N个NPU 126中,图18所示的NPU 126被标示为NPU J。也就是说,NPU J是N个NPU的代表性实例。优选地,NPU J的复用寄存器208A的输入211A接收NPU 126实例J-1的复用寄存器208A的输出209A,并且NPU J的复用寄存器208A的输入1811A接收NPU 126实例J-1的复用寄存器208B的输出209B,并且NPU J的复用寄存器208A的输出209A被提供至NPU 126实例J+1的复用寄存器208A的输入211A以及NPU 126实例J的复用寄存器208B的输入211B这两者;并且NPU J的复用寄存器208B的输入211B接收NPU 126实例J-1的复用寄存器208B的输出209B,NPU J的复用寄存器208B的输入1811B接收NPU 126实例J的复用寄存器208A的输出209A,并且NPU J的复用寄存器208B的输出209B被提供至NPU 126实例J+1的复用寄存器208A的输入1811A以及NPU 126实例J+1的复用寄存器208B的输入211B这两者。

[0188] 控制输入213控制这三个输入中的哪一个输入被复用寄存器208A/208B选择以储存在各自的寄存器中并于后续提供在各自的输出209A/209B上。在NPU 126被(例如,如以下描述通过图20的地址1处的乘法累加指令)指示要从数据RAM 122加载一行的情况下,无论NPU 126是处于宽配置还是处于窄配置,控制输入213都控制各复用寄存器208A/208B从数据RAM 122的选定行的相应窄字中选择各自的窄数据字207A/207B(例如8位)。

[0189] 在NPU 126被(例如,如以下描述通过图20的地址2处的乘法累加轮转指令)指示为对先前接收的数据行的值进行轮转的情况下,如果NPU 126处于窄配置,则控制输入213控制各复用寄存器208A/208B选择相应的输入1811A/1811B。在这种情况下,复用寄存器208A/208B实际上独立操作,使得NPU 126实际上是两个单独的窄NPU。如此,如以下针对图19更详

细所述, N个NPU 126的复用寄存器208A和208B整体地作为2N个窄字的轮转器进行操作。

[0190] 在NPU 126被指示为对先前接收的数据行的值进行轮转的情况下, 如果NPU 126处于宽配置, 则控制输入213控制各复用寄存器208A/208B选择相应的输入211A/211B。在这种情况下, 复用寄存器208A/208B实际上整体地仿佛该NPU 126是单个宽NPU 126那样进行操作。如此, 与针对图3所述的方式类似地, N个NPU 126的复用寄存器208A和208B整体地作为N宽字的轮转器进行工作。

[0191] ALU 204包括操作数选择逻辑1898、宽乘法器242A、窄乘法器242B、宽2输入复用器1896A, 窄2输入复用器1896B, 宽加法器244A以及窄加法器244B。实际上, ALU 204包括操作数选择逻辑1898、宽ALU 204A (包括宽乘法器242A、宽复用器1896A和宽加法器244A) 以及窄ALU 204B (包括窄乘法器242B、窄复用器1896B和窄加法器244B)。优选地, 宽乘法器242A将两个宽字相乘, 并且与图2的乘法器242 (例如16位 $\times$ 16位乘法器) 类似。窄乘法器242B将两个窄字相乘 (例如产生16位结果的8位 $\times$ 8位乘法器)。当NPU 126为窄配置时, 借助于操作数选择逻辑1898, 宽乘法器242A实际上用作窄乘法器以使两个窄字相乘, 使得NPU 126实际上用作两个窄NPU。优选地, 宽加法器244A将宽复用器1896A的输出与宽累加器202A的输出217A相加以产生总和215A从而提供给宽累加器202A, 其类似于图2的加法器244。窄加法器244B将窄复用器1896B的输出与窄累加器202B的输出217B相加以产生总和215B从而提供给窄累加器202B。在一个实施例中, 窄累加器202B具有28位的宽度, 以避免在对多达1024个16位乘积进行累加时损失精度。当NPU 126为宽配置时, 窄乘法器242B、窄复用器1896B、窄加法器244B、窄累加器202B以及窄AFU 212B优选地不活动以降低能耗。

[0192] 如以下更详细所述, 操作数选择逻辑1898从209A、209B、203A与203B中选择操作数以提供至ALU 204的其它元件。优选地, 操作数选择逻辑1898还执行其它功能, 例如执行带符号值的数据字和权重字的符号扩展。例如, 如果NPU 126为窄配置, 则操作数选择逻辑1898在将窄数据字和权重字提供给宽乘法器242A之前, 将该窄数据字和权重字符号扩展至宽字的宽度。类似地, 如果ALU 204被指示为使窄数据/权重字通过 (经由宽复用器1896A而跳过宽乘法器242A), 则操作数选择逻辑1898在将窄数据/权重字提供给宽加法器244A之前, 将窄数据/权重字符号扩展至宽字的宽度。优选地, 图2的NPU 126的ALU 204中也存在执行符号扩展功能的逻辑。

[0193] 宽复用器1896A接收宽乘法器242A的输出以及来自操作数选择逻辑1898的操作数, 并从这些输入中选择其中之一以提供给宽加法器244A, 并且窄复用器1896B接收窄乘法器242B的输出以及来自操作数选择逻辑1898的操作数, 并选择这些输入其中之一以提供给窄加法器244B。

[0194] 操作数选择逻辑1898所提供的操作数取决于NPU 126的配置以及ALU 204基于NPU 126正执行的指令所指定的函数而执行的算术和/或逻辑运算。例如, 如果指令指示ALU 204执行乘法累加并且NPU 126为宽配置, 则操作数选择逻辑1898将作为输出209A和209B的串接的宽字提供至宽乘法器242A的一个输入、并将作为输出203A和203B的串接的宽字提供至另一输入, 而窄乘法器242B不活动, 使得NPU 126用作与图2的NPU 126类似的单个宽NPU126。而如果指令指示ALU 204执行乘法累加并且NPU 126处于窄配置, 则操作数选择逻辑1898将窄数据字209A的扩展后或扩宽后的版本提供至宽乘法器242A的一个输入、并将窄权重字203A的扩展后版本提供至另一输入; 此外, 操作数选择逻辑1898将窄数据字209B提

供至窄乘法器242B的一个输入并将窄权重字203B提供至另一输入。为将窄字扩展或扩宽，如果窄字带符号，则操作数选择逻辑1898对该窄字进行符号扩展；而如果窄字不带符号，则操作数选择逻辑1898为窄字填入值为零的高位。

[0195] 再例如，如果NPU 126处于宽配置并且指令指示ALU 204执行权重字的累加，则宽乘法器242A被跳过，并且操作数选择逻辑1898将输出203A和203B的串接提供至宽复用器1896A，以提供给宽加法器244A。而如果NPU 126为窄配置并且指令指示ALU 204执行权重字的累加，则宽乘法器242A被跳过并且操作数选择逻辑1898将输出203A的扩展后版本提供至宽复用器1896A，以提供给宽加法器244A；并且窄乘法器242B被跳过并且操作数选择逻辑1898将输出203B的扩展后版本提供至窄复用器1896B，以提供给窄加法器244B。

[0196] 再例如，如果NPU 126为宽配置并且指令指示ALU 204执行数据字的累加，则宽乘法器242A被跳过并且操作数选择逻辑1898将输出209A和209B的串接提供至宽复用器1896A，以提供给宽加法器244A。而如果NPU 126为窄配置并且指令指示ALU 204执行数据字的累加，则宽乘法器242A被跳过、并且操作数选择逻辑1898将输出209A的扩展后版本提供至宽复用器1896A，以提供给宽加法器244A；并且窄乘法器242B被跳过、并且操作数选择逻辑1898将输出209B的扩展后版本提供至窄复用器1896B，以提供给窄加法器244B。权重/数据字的累加可以有助于执行平均运算，其中这些平均运算用于诸如图像处理等的某些人工神经网络应用的池化层。

[0197] 优选地，NPU 126还包括：第二宽复用器（未示出），用于跳过宽加法器244A以便于在宽配置下利用宽数据/权重字或在窄配置下利用扩展后的窄数据/权重字来加载宽累加器202A；以及第二窄复用器（未示出），用于跳过窄加法器244B，以便于在窄配置下利用窄数据/权重字加载窄累加器202B。优选地，ALU 204还包括宽和窄的比较器/复用器组合（未示出），其中该比较器/复用器组合接收相应的累加器值217A/217B和相应的复用器1896A/1896B输出，以在累加器值217A/217B与数据/权重字209A/B/203A/B之间选择最大值，如以下例如针对图27和28更详细所述，这样的运算用于某些人工神经网络应用的池化层中。此外，操作数选择逻辑1898被配置为提供值为零的操作数（用于加零或用于清除累加器）并提供值为一的操作数（用于乘一）。

[0198] 窄AFU 212B接收窄累加器202B的输出217B并对其执行激活函数以产生窄结果133B，而宽AFU 212A接收宽累加器202A的输出217A并对其执行激活函数以产生宽结果133A。当NPU 126为窄配置时，宽AFU 212A相应地考虑宽累加器202A的输出217A并对其执行激活函数以产生窄结果（例如8位），这如以下例如针对图29A至图30更详细所述。

[0199] 从以上说明可以发现，有利地，单个NPU 126在为窄配置时实际上作为两个窄NPU而操作，因此对于较小的字提供大致为宽配置时的吞吐量的多达两倍的吞吐量。例如，假定神经网络层具有1024个神经元，而每个神经元从先前层接收1024个窄输入（并具有窄权重字），从而产生一百万个连接。与具有512个宽配置的NPU 126的NNU 121相比，具有512个窄配置的NPU 126的NNU 121能够在大致一半的时间内（约1026个时钟vs514个时钟）处理四倍的连接数（一百万个连接vs256K个连接），尽管处理的是窄字而非宽字。

[0200] 在一个实施例中，图18的可动态配置NPU 126包括与复用寄存器208A和208B类似的3输入复用寄存器以取代寄存器205A和205B，从而实现针对从权重RAM 124接收到的一行权重字的轮转器，这与针对图7的实施例所述的方式某种程度类似但采用针对图18所述的

可动态配置方式。

[0201] 现参考图19,示出一框图,该框图示出根据图18的实施例的图1的NNU 121的N个NPU 126的2N个复用寄存器208A/208B的布置的实施例、从而说明这2N个复用寄存器208A/208B作为针对从图1的数据RAM 122接收到的一行数据字207的轮转器的操作。在图19的实施例中,如图所示,N是512,使得NNU 121具有被标示为0至511的1024个复用寄存器208A/208B,其对应至512个NPU 126(实际上为1024个窄NPU)。NPU 126内的两个窄NPU标示为A和B,并且在每个复用寄存器208中,示出相应的窄NPU的指定。更具体地,为NPU 126 0的复用寄存器208A指定0-A,为NPU 126 0的复用寄存器208B指定0-B,为NPU 126 1的复用寄存器208A指定1-A,为NPU 126 1的复用寄存器208B指定1-B,为NPU 126 511的复用寄存器208A指定511-A,并且为NPU 126 511的复用寄存器208B指定511-B,这些值亦对应至以下所述的图21的窄NPU。

[0202] 每个复用寄存器208A接收数据RAM 122的D行的其中一行中的相应窄数据字207A,并且每个复用寄存器208B接收数据RAM 122的D行的其中一行中的相应窄数据字207B。也就是说,复用寄存器0A接收数据RAM 122行的窄数据字0,复用寄存器0B接收数据RAM 122行的窄数据字1,复用寄存器1A接收数据RAM 122行的窄数据字2,复用寄存器1B接收数据RAM 122行的窄数据字3,依此类推,复用寄存器511A接收数据RAM 122行的窄数据字1022,以及复用寄存器511B接收数据RAM 122行的窄数据字1023。此外,复用寄存器1A在其输入211A上接收复用寄存器0A的输出209A,复用寄存器1B在其输入211B上接收复用寄存器0B的输出209B,依此类推,复用寄存器511A在其输入211A上接收复用寄存器510A的输出209A,复用寄存器511B在其输入211B上接收复用寄存器510B的输出209B,并且复用寄存器0A在其输入211A上接收复用寄存器511A的输出209A,复用寄存器0B在其输入211B上接收复用寄存器511B的输出209B。每个复用寄存器208A/208B接收控制输入213,其中该控制输入213控制是要选择数据字207A/207B、还是选择轮转后输入211A/211B、亦或是选择轮转后输入1811A/1811B。最后,复用寄存器1A在其输入1811A上接收复用寄存器0B的输出209B,复用寄存器1B在其输入1811B上接收复用寄存器1A的输出209A,依此类推,复用寄存器511A在其输入1811A上接收复用寄存器510B的输出209B,复用寄存器511B在其输入1811B上接收复用寄存器511A的输出209A,并且复用寄存器0A在其输入1811A上接收复用寄存器511B的输出209B,复用寄存器0B在其输入1811B上接收复用寄存器0A的输出209A。每个复用寄存器208A/208B接收控制输入213,其中该控制输入213控制是要选择数据字207A/207B、还是选择轮转后输入211A/211B、亦或是选择轮转后输入1811A/1811B。如以下更详细所述,在一运算模式中,在第一时间钟周期中,控制输入213控制每个复用寄存器208A/208B选择数据字207A/207B以储存至寄存器并于后续提供至ALU 204;以及在后续时钟周期(例如上述的M-1时钟周期)中,控制输入213控制每个复用寄存器208A/208B选择轮转后输入1811A/1811B以储存至寄存器并于后续提供至ALU 204。

[0203] 现参考图20,示出一表格,该表格示出储存于图1的NNU 121的程序存储器129中并由该NNU 121执行的程序,其中该NNU 121具有根据图18的实施例的NPU 126。图20的示例性程序在许多方面与图4的程序类似。然而,以下将说明差异。位于地址0的初始化NPU指令指定NPU 126将为窄配置。此外,如图所示,位于地址2的乘法累加轮转指令将计数指定为1023,并需要1023个时钟周期。这是因为图20的示例假定一层实际上具有1024个窄(例如8

位) 神经元 (NPU), 每个窄神经元具有来自先前层的1024个神经元的1024个连接输入, 因此总共有1024K个连接。每个神经元从每个连接输入接收8位数据值, 并将该8位数据值乘以适当的8位权重值。

[0204] 现参考图21, 示出一时序图, 该时序图示出NNU 121执行图20的程序, 其中该NNU 121包括图18的在窄配置中操作的NPU 126。图21的时序图在许多方面与图5的时序图类似; 然而, 以下将说明差异。

[0205] 在图21的时序图中, NPU 126为窄配置, 这是因为位于地址0的初始化NPU指令将这些NPU初始化为窄配置。因此, 这512个NPU 126实际上作为1024个窄NPU (或神经元) 操作, 其中这1024个窄NPU在列内被指定为NPU 0-A和NPU 0-B (NPU 126 0的两个窄NPU)、NPU 1-A和NPU 1-B (NPU 126 1的两个窄NPU)、...、NPU 511-A与NPU 511-B (NPU 126 511的两个窄NPU)。为简单清楚地图示, 仅示出窄NPU 0-A、0-B与511-B的操作。由于地址2处的乘法累加轮转将计数指定为1023 (这需要1023个时钟周期) 这一事实, 因此图21的时序图的行包括多达1026个时钟周期。

[0206] 在时钟0处, 1024个NPU各自执行图4的初始化指令, 即图5所示将零值指派至累加器202的初始化指令。

[0207] 在时钟1处, 1024个窄NPU各自执行图20的地址1处的乘法累加指令。如图所示, 窄NPU 0-A将数据RAM 122的行17的窄字0和权重RAM 124的行0的窄字0的乘积与累加器202A的值 (即零) 累加; 窄NPU 0-B将数据RAM 122的行17的窄字1和权重RAM 124的行0的窄字1的乘积与累加器202B的值 (即零) 累加; 依此类推, 窄NPU 511-B将数据RAM 122的行17的窄字1023与权重RAM 124的行0的窄字1023的乘积与累加器202B的值 (即零) 累加。

[0208] 在时钟2处, 1024个窄NPU各自执行图20的地址2的乘法累加轮转指令的第一次迭代。如图所示, 窄NPU 0-A将从窄NPU 511-B的复用寄存器208B的输出209B所接收到的轮转后的窄数据字1811A (即从数据RAM 122所接收到的窄数据字1023) 和权重RAM 124的行1的窄字0的乘积与累加器202A的值217A累加; 窄NPU 0-B将从窄NPU 0-A的复用寄存器208A的输出209A所接收到的轮转后的窄数据字1811B (即从数据RAM 122所接收到的窄数据字0) 和权重RAM 124的行1的窄字1的乘积与累加器202B的值217B累加; 依此类推, 窄NPU 511-B将从窄NPU 511-A的复用寄存器208A的输出209A所接收到的轮转后的窄数据字1811B (即从数据RAM 122所接收到的窄数据字1022) 和权重RAM 124的行1的窄字1023的乘积与累加器202B的值217B累加。

[0209] 在时钟3处, 1024个窄NPU各自执行图20的地址2处的乘法累加轮转指令的第二次迭代。如图所示, 窄NPU 0-A将从窄NPU 511-B的复用寄存器208B的输出209B所接收到的轮转后的窄数据字1811A (即从数据RAM 122所接收到的窄数据字1022) 和权重RAM 124的行2的窄字0的乘积与累加器202A的值217A累加; 窄NPU 0-B将从窄NPU 0-A的复用寄存器208A的输出209A所接收到的轮转后的窄数据字1811B (即从数据RAM 122所接收到的窄数据字1023) 和权重RAM 124的行2的窄字1的乘积与累加器202B的值217B累加; 依此类推, 窄NPU 511-B将从窄NPU 511-A的复用寄存器208A的输出209A所接收到的轮转后的窄数据字1811B (即从数据RAM 122所接收到的窄数据字1021) 和权重RAM 124的行2的窄字1023的乘积与累加器202B的值217B累加。如图21的省略号所示, 在接下来的1021个时钟周期的每个时钟周期内依此持续进行, 直到时钟1024。

[0210] 在时钟1024处,1024个窄NPU各自执行图20的地址2处的乘法累加轮转指令的第1023次迭代。如图所示,窄NPU 0-A将从窄NPU 511-B的复用寄存器208B的输出209B所接收到的轮转后的窄数据字1811A(即从数据RAM 122所接收到的窄数据字1)和权重RAM 124的行1023的窄字0的乘积与累加器202A的值217A累加;窄NPU 0-B将从NPU 0-A的复用寄存器208A的输出209A所接收到的轮转后的窄数据字1811B(即从数据RAM 122所接收到的窄数据字2)和权重RAM 124的行1023的窄字1的乘积与累加器202B的值217B累加;依此类推,窄NPU 511-B将从NPU 511-A的复用寄存器208A的输出209A所接收到的轮转后的窄数据字1811B(即从数据RAM 122所接收到的窄数据字0)和权重RAM 124的行1023的窄字1023的乘积与累加器202B的值217B累加。

[0211] 在时钟1025处,1024个窄NPU各自的AFU 212A/212B执行图20的地址3处的激活函数指令。最后,在时钟1026处,这1024个窄NPU各自通过将其窄结果133A/133B写回数据RAM 122的行16的相应窄字,来执行图20的地址4处的写入AFU输出指令,即NPU 0-A的窄结果133A被写入数据RAM 122的窄字0,NPU 0-B的窄结果133B被写入数据RAM 122的窄字1,依此类推,NPU 511-B的窄结果133B被写入数据RAM 122的窄字1023。在图22中,还以框图的形式示出以上针对图21所述的操作。

[0212] 现参考图22,示出一框图,该框图示出图1的NNU 121,其中该NNU 121包括图18的NPU 126以执行图20的程序。NNU 121包括512个NPU 126即1024个窄NPU、接收其地址输入123的数据RAM 122、以及接收其地址输入125的权重RAM 124。尽管未示出,但在时钟0,这1024个窄NPU执行图20的初始化指令。如图所示,在时钟1处,行17的1024个8位数据字被从数据RAM 122读出并提供至1024个窄NPU。在时钟1至1024,行0至1023的1024个8位权重字分别被从权重RAM 124读出并提供至1024个窄NPU。虽然未示出,但在时钟1,1024个窄NPU对加载的数据字与权重字执行相应的乘法累加运算。在时钟2至1024,1024个窄NPU的复用寄存器208A/208B作为1024个8位字轮转器操作以将先前加载的数据RAM 122的行17的数据字轮转至相邻的窄NPU,并且窄NPU对相应的轮转后的数据字以及从权重RAM 124加载的相应窄权重字执行乘法累加运算。尽管未示出,但在时钟1025,1024个窄AFU 212A/212B执行激活指令。在时钟1026,1024个窄NPU将其相应的1024个8位结果133A/133B写回数据RAM 122的行16。

[0213] 可以发现,例如,图18的实施例相较于图2的实施例可具有优势,这是因为图18的实施例为程序设计者提供了灵活性,以在正被建模的特定应用需要一定精确度的情况下使用宽数据字和权重字(例如16位)进行计算,并且在该应用需要一定精确度的情况下使用窄数据字和权重字(例如8位)来进行计算。从一个角度来看,对于窄数据的应用而言,图18的实施例相较于图2的实施例,以额外的窄元件(例如复用寄存器208B、寄存器205B、窄ALU 204B、窄累加器202B、窄AFU 212B)作为代价,可提供两倍的吞吐量,这些额外的窄元件使NPU 126的面积增加约50%。

[0214] 三模式NPU

[0215] 现参考图23,示出一框图,该框图示出根据备选实施例的可动态配置的图1的NPU 126。图23的NPU 126不但可配置为宽配置与窄配置,还可配置于第三配置(在本文中称为“漏斗(funnel)”配置)。图23的NPU 126在许多方面与图18的NPU 126类似。然而,图18中的宽加法器244A在图23的NPU 126中由3输入宽加法器2344A所取代,其中3输入宽加法器

2344A接收作为窄复用器1896B的输出的扩展后版本的第三加数2399。用于操作具有图23的NPU 126的NNU 121的程序在许多方面与图20的程序类似。然而,地址0处的初始化NPU指令将这些NPU 126初始化为漏斗配置,而非窄配置。此外,地址2的乘法累加轮转指令的计数为511而非1023。

[0216] 在漏斗配置的情况下,NPU 126的操作与在以窄配置执行诸如图20的地址1处的乘法累加指令的情况下的操作在以下方面类似:NPU 126接收两个窄数据字207A/207B和两个窄权重字206A/206B;宽乘法器242A将数据字209A与权重字203A相乘以产生宽复用器1896A所选择的乘积246A;以及窄乘法器242B将数据字209B与权重字203B相乘以产生窄复用器1896B所选择的乘积246B。然而,宽加法器2344A将(宽复用器1896A所选择的)乘积246A和(宽复用器1896B所选择的)乘积246B/2399两者与宽累加器202A的值217A相加,而窄加法器244B与窄累加器202B不活动。此外,在以漏斗配置执行诸如图20的地址2处的乘法累加轮转指令时,控制输入213使复用寄存器208A/208B轮转两个窄字(例如16位),也就是说,复用寄存器208A/208B选择其相应的输入211A/211B,就如同宽配置一样。然而,宽乘法器242A将数据字209A与权重字203A相乘以产生宽复用器1896A所选择的乘积246A;窄乘法器242B将数据字209B与权重字203B相乘以产生窄复用器1896B所选择的乘积246B;以及宽加法器2344A将(宽复用器1896A所选择的)乘积246A和(宽复用器1896B所选择的)乘积246B/2399两者与宽累加器202A的值217A相加,而窄加法器244B与窄累加器202B如上所述不活动。最后,在以漏斗配置执行诸如图20的地址3处等的激活函数指令时,宽AFU 212A对所得到的总和215A执行激活函数以产生窄结果133A,而窄AFU 212B不活动。如此,只有标示为A的窄NPU产生窄结果133A,而标示为B的窄NPU所产生的窄结果133B是无效的。因此,写回的结果行(例如图20的地址4处的指令所指示的行16)包含空洞,这是因为只有窄结果133A是有效的,而窄结果133B是无效的。因此,与每个神经元在每个时钟周期内处理一个连接数据输入的图2和图18的实施例相对比,从概念上讲,每个时钟周期内,每个神经元(图23的NPU 126)处理两个连接数据输入,即将两个窄数据字乘以相应的权重并将这两个乘积累加。

[0217] 针对图23的实施例可以发现,所产生并写回数据RAM 122或权重RAM 124的结果字(神经元输出)的数量是所接收到的数据输入(连接)的数量的平方根的一半,并且写回的结果行具有空洞,即每隔一个窄字结果是无效的,更具体地,标示为B的窄NPU结果不具意义。因此,图23的实施例对于具有连续两层的神经网络特别有效,例如,第一层所具有的神经元数量为第二层的两倍(例如第一层所具有的1024个神经元完全连接至第二层的512个神经元)。此外,其它执行单元122(例如媒体单元,诸如x86AVX单元)在必要的情况下可对分散的(即具有空洞的)结果行执行压紧运算(pack operation)以使其紧密(即不具有空洞),以用于当NNU 121正执行与数据RAM 122和/或权重RAM 124的其它行相关联的其它计算时的后续计算。

[0218] 混合NNU运算:卷积能力与池化能力

[0219] 根据本文所述实施例的NNU 121的优点在于,该NNU 121能够并行地以类似于协处理器执行自己内部程序的方式操作、以及以类似于处理器的执行单元执行被发布给该执行单元的架构指令(或从架构指令转译来的微指令)的方式操作。架构指令具有由包括NNU 121的处理器所执行的架构程序。如此,NNU 121以混合方式操作,这种操作方式是有利的,因为它提供了维持对NNU 121的高利用率的能力。例如,图24至图26示出NNU 121执行卷积

运算的操作,在该操作中,NNU的利用率高,并且图27至图28示出NNU 121执行池化运算的操作,其中这些卷积运算和池化运算是卷积层、池化层以及诸如图像处理(例如边缘检测、锐化、模糊、辨识/分类)等的其它数字数据计算应用所需的。然而,NNU 121的混合运算不限于执行卷积或池化运算,而是混合特征也可用于执行其它运算,诸如以上针对图4至图13所述的传统神经网络乘法累加运算和激活函数运算等。也就是说,处理器100(更具体地,保留站108)向NNU 121发布MTNN指令1400与MFNN指令1500,其中响应于这些指令,NNU 121将数据写入存储器122/124/129并从由NNU 121进行写入的存储器122/124中读取结果,与此同时,响应于执行由处理器100(经由MTNN 1400指令)写入程序存储器129的程序,NNU 121对存储器122/124/129进行读取和写入。

[0220] 现参考图24,示出一框图,该框图示出图1的NNU 121使用以执行卷积运算的数据结构的示例。该框图包括卷积核2402、数据阵列2404、以及图1的数据RAM 122与权重RAM 124。优选地,(例如,图像像素的)数据阵列2404保持在附接至处理器100的系统存储器(未示出)中并通过处理器100执行MTNN指令1400而被加载至NNU 121的权重RAM 124。卷积运算是将第一矩阵与第二矩阵进行卷积的运算,其中第二矩阵在本文中被称为卷积核。如在本发明的上下文中所述,卷积核是系数的矩阵,其中这些系数也可称为权重、参数、元素或值。优选地,卷积核2402是处理器100正执行的架构程序的静态数据。

[0221] 数据阵列2404是数据值的二维阵列,并且各数据值(例如图像像素值)是数据RAM 122或权重RAM 124的字的的大小(例如16位或8位)。在该示例中,数据值为16位字,并且NNU 121被配置为512个宽配置NPU 126。此外,在实施例中,如以下更详细所述,NPU 126包括用于从权重RAM 124接收权重字206的复用寄存器(诸如图7的复用寄存器705等),以对从权重RAM 124接收到的一行数据值执行整体轮转器运算。在该示例中,数据阵列2404是2560列 $\times$ 1600行的像素阵列。如图所示,当架构程序将数据阵列2404与卷积核2402进行卷积时,该架构程序将数据阵列2402分为20个数据块,其中各数据块是 512×400 的数据矩阵2406。

[0222] 在示例中,卷积核2402是由系数、权重、参数、或元素构成的 3×3 矩阵。第一行系数被标示为C0,0、C0,1和C0,2;第二行系数被标示为C1,0、C1,1和C1,2;并且第三行系数被标示为C2,0、C2,1、和C2,2。例如,可用于执行边缘检测的卷积核具有以下系数:0,1,0,1,-4,1,0,1,0。再例如,可用于对图像进行高斯模糊的卷积核具有以下系数:1,2,1,2,4,2,1,2,1。在这种情况下,通常对最终累加值执行除法,其中,除数为卷积核2402的各元素的绝对值的总和(在该示例中为16)。又例如,除数是卷积核2402的元素的数量。又例如,除数是将卷积压缩回至期望值范围内的值,并且该除数根据卷积核2402的元素值、期望范围以及正被执行卷积运算的矩阵的输入值的范围来确定。

[0223] 如图24所示并且针对图25更详细所述,架构程序将卷积核2402的系数写入数据RAM 122。优选地,数据RAM 122的连续九行(卷积核2402内的元素数量)的每行的所有字以行为主要顺序被写入卷积核2402的不同元素。也就是说,如图所示,一行的每个字被写入有第一系数C0,0;下一行被写入有第二系数C0,1;下一行被写入有第三系数C0,2;再下一行被写入有第四系数C1,0;依此类推,第九行的每个字被写入有第九系数C2,2。为了对数据阵列2404的数据块的数据矩阵2406进行卷积,如以下特别是针对图26更详细所述,NPU 126按顺序重复读取数据RAM 122中保持卷积核2402的系数的九行。

[0224] 如图24所示并且针对图25更详细所述,架构程序将数据矩阵2406的值写入权重

RAM 124。在NNU程序执行卷积时,将结果矩阵写回权重RAM 124。优选地,如以下针对图25更详细所述,架构程序将第一数据矩阵2406写入权重RAM 124并启动NNU 121,并且当NNU 121在对第一数据矩阵2406与卷积核2402进行卷积时,架构程序将第二数据矩阵2406写入权重RAM 124,使得NNU 121一旦完成对第一数据矩阵2406的卷积,其即可开始对第二数据矩阵2406执行卷积。如此,架构程序往返于权重RAM 124的两个区域之间,以确保NNU 121被充分使用。因此,图24的示例示出第一数据矩阵2406A和第二数据矩阵2406B,其中第一数据矩阵2406A与占据权重RAM 124的行0至399行的第一数据块相对应,第二数据矩阵2406B与占据权重RAM 124的行500至899的第二数据块相对应。此外,如图所示,NNU 121将卷积的结果写回权重RAM 124的行900~1299和1300~1699,随后架构程序从权重RAM 124读出这些结果。保持在权重RAM 124中的数据矩阵2406的数据值标示为“D_{x,y}”,其中“x”是权重RAM 124的行数,“y”是权重RAM 124的字或列数。因此,例如,行399中的数据字511在图24中被标示为D399,511,该数据字由NPU 511的复用寄存器705接收。

[0225] 现参考图25,示出一流程图,该流程图示出图1的处理器100执行架构程序的操作,所述架构程序将用以对卷积核2402执行卷积的NNU 121用于图24的数据阵列2404。流程始于方框2502。

[0226] 在方框2502处,处理器100(即运行在处理器100上的架构程序)将图24的卷积核2402以针对图24所示和所述的方式写入数据RAM 122。此外,架构程序将变量N初始化为值1。变量N表示数据阵列2404中正由NNU 121处理的当前数据块。此外,架构程序将变量NUM_CHUNKS初始化为值20。流程进入方框2504。

[0227] 在方框2504处,如图24所示,处理器100将数据块1的数据矩阵2406写入权重RAM 124(例如数据块1的数据矩阵2406A)。流程进入方框2506。

[0228] 在方框2506处,处理器100使用用于指定对程序存储器129进行写入的函数1432的MTNN指令1400,将卷积程序写入NNU 121的程序存储器129。处理器100随后使用用于指定启动对程序的执行的函数1432的MTNN指令1400,来启动NNU卷积程序。以下针对图26A更详细地描述NNU卷积程序的示例。流程进入决策方框2508。

[0229] 在决策方框2508处,架构程序确定变量N的值是否小于NUM_CHUNKS。如果是,流程进入方框2512;否则进入方框2514。

[0230] 在方框2512处,如图24所示,处理器100将数据块N+1的数据矩阵2406写入权重RAM 124(例如数据块2的数据矩阵2406B)。因此,有利地,当NNU 121正在对当前数据块执行卷积时,架构程序将下一数据块的数据矩阵2406写入权重RAM 124,使得一旦完成当前数据块的卷积(即写入权重RAM 124),NNU 121就可以立即开始执行对下一个数据块的卷积。流程进入方框2514。

[0231] 在方框2514处,处理器100确定为(在数据块1的情况下从方框2506开始的,在数据块2~20的情况下从方框2518开始的)当前正在运行的NNU程序已经完成。优选地,处理器100通过执行MFNN指令1500以读取NNU 121的状态寄存器127来进行该确定。在备选实施例中,NNU 121产生中断,从而表示其已经完成卷积程序。流程进入决策方框2516。

[0232] 在决策方框2516处,架构程序确定变量N的值是否小于NUM_CHUNKS。如果是,流程进入方框2518;否则进入方框2522。

[0233] 在方框2518处,处理器100更新卷积程序使得该处理器可以对数据块N+1进行卷

积。更具体地,处理器100将权重RAM 124中在地址0处的初始化NPU指令的行值更新为数据矩阵2406的第一行(例如,更新为数据矩阵2406A的行0或数据矩阵2406B的行500),并且更新输出行(例如更新为行900或1300)。然后处理器100启动更新后的NNU卷积程序。流程进入方框2522。

[0234] 在方框2522处,处理器100从权重RAM 124读取数据块N的NNU卷积程序的结果。流程进入决策方框2524。

[0235] 在决策方框2524处,架构程序确定变量N的值是否小于NUM_CHUNKS。如果是,流程进入方框2526;否则流程结束。

[0236] 在方框2526处,架构程序使N的增加1。流程返回决策方框2508。

[0237] 现参考图26A,示出NNU程序的程序列表,其中该NNU程序利用图24的卷积核2402对数据矩阵2406执行卷积并将其写回权重RAM 124。该程序将通过地址1至9处的指令循环体来循环一定次数。地址0处的初始化NPU指令指定各NPU 126执行该循环体的次数,在图26A的示例中,与图24的数据矩阵2406内的行数相对应地,循环计数值为400,并且循环末端(地址10)处的循环指令使当前循环计数值递减,并且如果结果非零、则所述循环指令引起返回循环体的顶端(即返回地址1处的指令)的控制。初始化NPU指令也将累加器202清零。优选地,地址10处的循环指令也将累加器202清零。可选地,如上所述,地址1处的乘法累加指令可以指定将累加器202清零。

[0238] 对于程序的循环体的每一次执行,512个NPU 126并行地对 3×3 卷积核2402和数据矩阵2406的512个相应 3×3 子矩阵执行512个卷积。卷积是卷积核2402的元素与相应子矩阵内的对应元素的九个乘积的总和。在图26A的实施例中,512个相应 3×3 子矩阵各自的原点(中央元素)是图24的数据字 $D_{x+1,y+1}$,其中y(列编号)是NPU 126的编号,而x(行编号)是当前权重RAM 124中由图26A的程序的地址1处的乘法累加指令所读取的行编号(同样,该行编号由地址0处的初始化NPU指令进行初始化,在地址3与5的各乘法累加指令处递增,并且由地址9处的递减指令进行更新)。因此,针对程序的每一个循环,512个NPU 126计算512个卷积,并将512个卷积结果写回权重RAM 124的指定行。在本说明书中,为简单起见省略边缘处理(edge handling),但是应当注意,使用这些NPU 126的整体轮转特征将造成列中的两列从(例如,图像处理情况下的图像的)数据矩阵2406的一个垂直边缘到另一垂直边缘(例如从左侧边缘到右侧边缘或者反之亦然)产生环绕(wrapping)。现在针对循环体进行说明。

[0239] 地址1是用于指定数据RAM 122的行0并隐式地使用当前权重RAM 124的行的乘法累加指令,其中当前权重RAM 124的行优选保持在定序器128内(并由位于地址0的指令初始化为零以第一次通过循环体)。也就是说,位于地址1的指令使每个NPU 126从数据RAM 122的行0读取其相应字、从当前权重RAM 124的行读取其相应字,并对这两个字执行乘法累加运算。因此,例如,NPU 5将 $C_{0,0}$ 与 $D_{x,5}$ 相乘(其中“x”是当前权重RAM 124的行),将结果与累加器202的值217相加,并将总和写回累加器202。

[0240] 地址2是用于指定使数据RAM 122的行递增(即递增至行1)并且随后从数据RAM 122的递增后地址读取行的乘法累加指令。该指令还指定将每个NPU 126的复用寄存器705内的值轮转至相邻NPU 126,在这种情况下所述值是响应于地址1的指令而刚刚从权重RAM 124读取的一行数据矩阵2406值。在图24至26的实施例中,NPU 126被配置为将复用寄存器705的值向左轮转,即从NPU J轮转至NPU J-1,而非如以上针对图3、7和19所述从NPU J轮转

至NPU J+1。应当理解,在NPU 126被配置为向右轮转的实施例中,架构程序可以将卷积核2042的系数值按不同顺序写入数据RAM 122(例如绕着其中心列轮转)以实现相似卷积结果。此外,在需要时,架构程序可以对卷积核2402执行额外的预处理(例如转置(transposition))。此外,指令指定计数值2。因此,位于地址2的指令使每个NPU 126从数据RAM 122的行1读取其相应字,将轮转后的字接收至复用寄存器705,并对这两个字执行乘法累加运算。由于计数值为2,因此该指令还使每个NPU 126重复前述操作。也就是说,定序器128使数据RAM 122的行地址123增加(即增加至行2),并且每个NPU 126从数据RAM 122的行2读取其相应字,将轮转后的字接收至复用寄存器705,并且对这两个字执行乘法累加运算。因此,例如,假定当前权重RAM 124的行为27,在执行地址2处的指令后,NPU 5将C0,1和D27,6的乘积以及C0,2和D27,7的乘积累加至其累加器202。因此,在完成地址1和地址2处的指令后,C0,0和D27,5的乘积、C0,1和D27,6的乘积以及C0,2和D27,7的乘积将连同来自先前通过循环体的所有其它累加值一起累加至累加器202。

[0241] 地址3与4处的指令所执行的运算与地址1与2处的指令类似,然而借助于权重RAM 124的行递增指示符,这些指令对权重RAM 124的下一行执行运算,并且对数据RAM 122的接下来的三行(即行3至5)执行运算。也就是说,例如针对NPU 5,在完成地址1至4处的指令后,C0,0与D27,5的乘积、C0,1与D27,6的乘积、C0,2与D27,7的乘积、C1,0与D28,5的乘积、C1,1与D28,6的乘积以及C1,2与D28,7的乘积将连同来自先前通过循环体的所有其它累加值一起累加至累加器202。

[0242] 地址5和6处的指令所执行的运算与地址3与4处的指令类似,然而这些指令对权重RAM 124的下一行以及数据RAM 122的接下来的三行(即行6至8)执行运算。也就是说,例如针对NPU 5,在完成地址1至6的指令后,C0,0与D27,5的乘积、C0,1与D27,6的乘积、C0,2与D27,7的乘积、C1,0与D28,5的乘积、C1,1与D28,6的乘积、C1,2与D28,7、C2,0与D29,5的乘积、C2,1与D29,6的乘积、以及C2,2与D29,7的乘积将连同来自先前通过循环体的所有其它累加值一起累加至累加器202。也就是说,在完成地址1至6处的指令后、并且假定循环体开始时的权重RAM 124的行为27,则NPU 5例如将利用卷积核2402对以下 3×3 子矩阵进行卷积:

[0243] D27,5 D27,6 D27,7

[0244] D28,5 D28,6 D28,7

[0245] D29,5 D29,6 D29,7

[0246] 更一般地,在完成地址1到6处的指令后,512个NPU 126中的每个NPU 126都使用卷积核2402对下列 3×3 子矩阵进行了卷积:

[0247] Dr,n Dr,n+1 Dr,n+2

[0248] Dr+1,n Dr+1,n+1 Dr+1,n+2

[0249] Dr+2,n Dr+2,n+1 Dr+2,n+2

[0250] 其中r是循环体开始时权重RAM 124的行地址值,并且n是NPU 126的编号。

[0251] 地址7处的指令使累加器202的值217通过AFU 212。该通过函数使大小为从数据RAM 122和权重RAM 124读取的字的大小(以位为单位,在该示例中即16位)的字通过。优选地,如以下更详细所述,用户可指定输出格式,例如输出位中有多少位是小数位。可选地,指定除法激活函数,而非指定通过激活函数,其中诸如本文针对图29A和图30所述,所述除法

激活函数例如使用图30的“除法器”3014/3016其中之一将累加器202的值217除以一个除数。例如,在具有系数(诸如上述的高斯模糊核的十六分之一系数等)的卷积核2402的情况下,地址7处的激活函数指令可以指定除法激活函数(例如除以16),而非指定通过函数。可选地,架构程序可以在将卷积核2402系数写入数据RAM 122之前,对这些系数执行除以16的运算,并例如使用如下所述图29A的数据二进制小数点2922针对卷积核2402的值相应地调整二进制小数点的位置。

[0252] 地址8处的指令将AFU 212的输出写入权重RAM 124中由输出寄存器当前的值所指定的行,其中该当前值被地址0处的指令初始化并且借助于指令内的递增指示符在每次通过循环时递增。

[0253] 如根据图24至图26中具有 3×3 卷积核2402的示例可以确定,NPU 126大约每三个时钟周期读取权重RAM 124以读取数据矩阵2406的行,并且大约每12个时钟周期将卷积结果矩阵写入权重RAM 124。此外,假定包括诸如图17的缓存器1704等的写入和读取缓存器的实施例,与NPU 126进行读取与写入并行地,处理器100对权重RAM 124进行读取与写入,使得缓存器1704大约每16个时钟周期对权重RAM 124执行一次写入和一次读取,以分别写入数据矩阵2406以及读取卷积结果矩阵。因此,权重RAM 124的大约一半的带宽被NNU 121执行卷积核运算所用的混合方式消耗。尽管本示例包含 3×3 卷积核2402,但可以采用其它大小的卷积核,诸如 2×2 、 4×4 、 5×5 、 6×6 、 7×7 、 8×8 等矩阵,在这种情况下,NNU程序将改变。在卷积核较大的情况下,因为乘法累加指令的轮转版本的计数较大(例如图26A的程序的地址2、4与6处、以及较大卷积核所需的附加指令),因而NPU 126读取权重RAM 124的时间的百分比比较小,因此权重RAM 124的带宽被消耗的百分比也较小。

[0254] 可选地,架构程序将NNU程序配置为在不再需要输入数据矩阵2406的行之后对这些行进行覆写,而非将卷积结果写回权重RAM 124的不同行(例如行900~1299与1300~1699)。例如,在 3×3 卷积核的情况下,架构程序将数据矩阵2406写入权重RAM 124的行2~401,而非将数据矩阵2406写入行0~399,并且NPU程序被配置为从权重RAM 124的行0开始并且每通过一次循环体就递增地,将卷积结果写入权重RAM 124。如此,NNU程序只对不再需要的行进行覆写。例如,在第一次通过循环体之后(或更具体地,在执行地址1处的加载权重RAM 124的行0的指令之后),行0的数据可被覆写,但是行1~3的数据需要用于第二次通过循环体因而不会因为第一次通过循环体而被覆写;同样地,在第二次通过循环体之后,行1的数据可被覆写,但是行2~4的数据需要用于第三次通过循环体因而不会因为第二次通过循环体而被覆写;依此类推。在这样的实施例中,各个数据矩阵2406(数据块)的高度可以较大(例如,800行),从而得到较少的数据块。

[0255] 可选地,架构程序将NNU程序配置为将卷积结果写回数据RAM 122的卷积核2402上方(例如,行8上方)的行,而非将卷积结果写回权重RAM 124,并且当NNU 121(例如使用以下所述的图26B的数据RAM 122的最近写入的行2606的地址)写入结果时,架构程序从数据RAM 122读取这些结果。这种替代方案在权重RAM 124为单端口并且数据RAM 122为双端口的实施例中可以是有利的。

[0256] 从根据图24至图26A的实施例的NNU 121的操作可以发现,图26A的程序的每次执行需要约5000个时钟周期,因此,图24的整个 2560×1600 的数据阵列2404的卷积需要大约100000个时钟周期,明显少于以传统方式执行相同任务所需要的时钟周期数。

[0257] 现参考图26B,示出一框图,该框图示出根据一个实施例的图1的NNU 121的控制寄存器127的某些字段。状态寄存器127包括:字段2602,用于指示权重RAM 124中最近被NPU 126写入的行的地址;字段2606,用于指示数据RAM 122中最近被NPU 126写入的行的地址;字段2604,用于指示权重RAM 124中最近被NPU 126读取的行的地址;以及字段2608,用于指示数据RAM 122中最近被NPU 126读取的行的地址。这样使得执行于处理器100上的架构程序能够在其对数据RAM 122和/或权重RAM 124进行读取和/或写入时确定NNU 121的进度。利用这种能力,连同如上所述对输入数据矩阵进行覆写(或者如上所述将结果写入数据RAM 122)的选择,图24的数据阵列2404可以例如如下所述被处理为5个 512×1600 的数据块,而非20个 512×400 的数据块。处理器100从行2开始将第一 512×1600 数据块写入权重RAM 124,并开启NNU程序(该程序具有1600的循环计数、以及值为0的初始化权重RAM 124输出行)。当NNU 121执行NNU程序时,处理器100监测权重RAM 124的输出的位置/地址,从而(1)(使用MFNN指令1500)读取权重RAM 124中具有由NNU 121(从行0开始)写入的有效卷积结果的行、(2)一旦已经读取了有效卷积结果就将第二个 512×1600 数据矩阵2406(从行2开始)覆写于这些有效卷积结果,使得当NNU 121对于第一 512×1600 数据块完成NNU程序时,处理器100可以根据需要立即更新NNU程序并再次开启NNU程序以处理第二个 512×1600 数据块。该处理针对剩余的三个 512×1600 数据块再重复三次,以实现NNU 121的高利用率。

[0258] 有利地,在一个实施例中,如以下针对图29A、29B与30更详细所述,AFU 212具有对累加器202的值217高效地执行有效除法的能力。例如,使累加器202的值217除以16的激活函数NNU指令可用于上述的高斯模糊矩阵。

[0259] 尽管图24的示例中所使用的卷积核2402是应用于整个数据阵列2404的小型静态卷积核,但在其它实施例中,该卷积核可以是诸如常见于卷积神经网络的具有与数据阵列2404的不同数据值相关联的唯一权重的大型矩阵。当NNU 121以这种方式使用时,架构程序可以将数据矩阵与卷积核的位置互换,即将数据矩阵放置于数据RAM 122内并且将卷积核放置于权重RAM 124内,并且可通过对NNU程序的特定执行所处理的行数可以相对较少。

[0260] 现参考图27,示出一框图,该框图示出将图1中的权重RAM 124填充输入数据的示例,其中由图1的NNU 121对所述输入数据执行池化运算。人工神经网络的池化层所执行的池化运算通过取得输入矩阵的子区域或子矩阵并计算这些子矩阵的最大值或平均值来缩减输入数据矩阵(例如,图像或卷积后图像)的维度,并且这些最大值或平均值变为结果矩阵或池化矩阵。在图27和图28的示例中,池化运算计算各个子矩阵的最大值。池化运算对于例如执行对象分类或检测的人工神经网络特别有用。一般而言,池化运算有效地以所检查的子矩阵的元素数的因子来缩减输入矩阵的大小,并且特别地以子矩阵的相应维度的元素数来使输入矩阵在各个维度方向上缩减。在图27的示例中,输入数据是储存于权重RAM 124的行0至1599中的宽字(例如16位)的 512×1600 矩阵。在图27中,字以其所在的行列位置标示,例如,位于行0列0的字标示为D0,0;位于行0列1的字标示为D0,1;位于行0列2的字标示为D0,2;依此类推,位于行0列511的字标示为D0,511。同样,位于行1列0的字标示为D1,0;位于行1列1的字标示为D1,1;位于行1列2的字标示为D1,2;依此类推,位于行1列511的字标示为D1,511;依此类推,位于行1599列0的字标示为D1599,0;位于行1599列1的字标示为D1599,1;位于行1599列2的字标示为D1599,2;依此类推,位于行1599列511的字标示为D1599,511。

[0261] 现参考图28,示出NNU程序的程序列表,其中该NNU程序执行图27的输入数据矩阵

的池化运算并将其写回权重RAM 124。在图28的示例中,池化运算计算输入数据矩阵中各个 4×4 子矩阵的最大值。该程序使地址1至10处的指令的循环体循环多次。地址0处的初始化NPU指令指定每个NPU 126执行循环体的次数,例如在图28的示例中,循环体的循环计数值为400,并且循环末尾(地址11)处的循环指令使当前循环计数值递减,如果递减后的结果是非零值,则控制返回循环体的顶部(即返回地址1处的指令)。权重RAM 124内的输入数据矩阵实际上被NNU程序视为由四个相邻行构成的400个互斥组,即行0~3、行4~7、行8~11、依此类推,直到行1596~1599。由四个相邻行构成的每一个组包括128个 4×4 子矩阵,即由组的四行与四个相邻列行(即列0~3、列4~7、列8~11、依此类推直到列508~511)的交叉所形成的元素的 4×4 子矩阵。在512个NPU 126中,这512个NPU 126中每第四个NPU 126(即,128个NPU 126)对相应的 4×4 子矩阵执行池化运算,而其它四分之三的NPU 126则不使用。更具体地,NPU 0、4、8、依此类推直到NPU 508各自对其相应的 4×4 子矩阵执行池化运算,其中 4×4 子矩阵的最左侧列编号对应于NPU编号,而下方的行对应于当前权重RAM 124的行值,如以下更详细所述,该值被地址0处的初始化指令初始化为零并且在每次重复循环体时增加4。循环体的400次迭代与图27的输入数据矩阵中的 4×4 子矩阵组数(即输入数据矩阵的1600行除以4)相对应。初始化NPU指令也将累加器202清零。优选地,地址11处的循环指令也将累加器202清零。可选地,地址1的maxwacc指令指定将累加器202清零。

[0262] 对于程序的循环体的每次迭代,所使用的128个NPU 126并行地对输入数据矩阵的当前四行组中的128个相应的 4×4 子矩阵执行128个池化运算。更具体地,池化运算确定 4×4 子矩阵的16个元素中的最大值元素。在图28的实施例例中,对于所使用的128个NPU 126中的每个NPU y , 4×4 子矩阵的左下侧元素为图27的元素 $D_{x,y}$,其中 x 是循环体开始时当前权重RAM 124的行编号,由图28的程序的地址1处的maxwacc指令读取(此行编号也由地址0处的初始化NPU指令初始化,并在每次执行地址3、5和7处的maxwacc指令时递增)。因此,对于程序的每次循环,所使用的128个NPU 126将当前行组的相应128个 4×4 子矩阵的相应最大值元素写回权重RAM 124的指定行。以下针对该循环体进行说明。

[0263] 地址1处是maxwacc指令,用于隐式地使用当前权重RAM 124的行,该行优选保持在定序器128内(并由位于地址0处的指令初始化为零以用于第一次通过循环体)。地址1处的指令使每个NPU 126从权重RAM 124的当前行读取其相应字,将该字与累加器202的值217进行比较,并将这两个值的最大者储存于累加器202中。因此,例如,NPU 8确定累加器202的值217和数据字 $D_{x,8}$ (其中“ x ”是当前权重RAM 124的行)中的最大值,并将该最大值写回累加器202。

[0264] 地址2处是maxwacc指令,其用于指定将每个NPU 126的复用寄存器705内的值轮转至相邻NPU 126,所述值在这种情况下为响应于地址1处的指令而仅从权重RAM 124读取的一行输入数据矩阵值。在图27至图28的实施例例中,如以上针对图24至26所述,NPU 126被配置为将复用器705的值向左轮转,即从NPU J 轮转至NPU $J-1$ 。此外,指令指定计数值3。因此,地址2处的指令使每个NPU 126将轮转后的字接收至复用寄存器705并确定轮转后的字和累加器202的值217中的最大值,然后将该运算再重复两次。也就是说,每个NPU 126三次将轮转后的字接收至复用寄存器705中并确定轮转后的字和累加器202的值217中的最大值。因此,例如,假定循环体开始时的当前权重RAM 124的行为36,以NPU 8为例,在执行地址1与2处的指令后,NPU 8将在其累加器202中储存循环体开始时的累加器202以及四个权重RAM

124字D36,8、D36,9、D36,10和D36,11中的最大值。

[0265] 地址3与4处的maxwacc指令所执行的运算与地址1和2处的指令所执行的运算类似,然而利用权重RAM 124行递增指示符,地址3与4处的指令maxwacc对权重RAM 124的下一行执行运算。也就是说,假定循环体开始时的当前权重RAM 124的行是36,以NPU 8为例,在完成地址1至4处的指令之后,NPU 8将在其累加器202中储存循环体开始时的累加器202以及八个权重RAM 124的字D36,8、D36,9、D36,10、D36,11、D37,8、D37,9、D37,10与D37,11中的最大值。

[0266] 地址5至8处的maxwacc指令所执行的运算与地址3至4处的指令所执行的运算类似,然而地址5至8处的指令对权重RAM 124的接下来的两行执行运算。也就是说,假定循环体开始时的当前权重RAM 124列是36,以NPU 8为例,在完成地址1到8的指令后,NPU 8将在其累加器202中储存循环开始时的累加器202以及十六个权重RAM 124字D36,8、D36,9、D36,10、D36,11、D37,8、D37,9、D37,10、D37,11、D38,8、D38,9、D38,10、D38,11、D39,8、D39,9、D39,10与D39,11中的最大值。也就是说,假定循环体开始时的当前权重RAM124的行是36,以NPU 8为例,在完成地址1到8处的指令后,NPU 8将确定以下 4×4 子矩阵的最大值:

[0267] D36,8 D36,9 D36,10 D36,11

[0268] D37,8 D37,9 D37,10 D37,11

[0269] D38,8 D38,9 D38,10 D38,11

[0270] D39,8 D39,9 D39,10 D39,11

[0271] 更具体地,在完成地址1至8处的指令后,所用的128个NPU 126中的各NPU 126将确定以下 4×4 子矩阵的最大值:

[0272] Dr,n Dr,n+1 Dr,n+2 Dr,n+3

[0273] Dr+1,n Dr+1,n+1 Dr+1,n+2 Dr+1,n+3

[0274] Dr+2,n Dr+2,n+1 Dr+2,n+2 Dr+2,n+3

[0275] Dr+3,n Dr+3,n+1 Dr+3,n+2 Dr+3,n+3

[0276] 其中r是循环体开始时的权重RAM 124的行地址值,n是NPU 126的编号。

[0277] 地址9处的指令使累加器202的值217通过AFU 212。这种通过函数使得大小为从权重RAM 124读取的字的的大小(以位为单位,在该示例中即16位)的字通过。优选地,如以下更详细所述,用户可指定输出格式,例如输出位中有多少位是小数位。

[0278] 地址10处的指令将累加器202的值217写入权重RAM 124中由输出寄存器的当前值所指定的行,其中该当前值被地址0处的指令初始化,并借助于指令内的递增指示符在每次通过循环体时递增。更具体地,地址10处的指令将累加器202的宽字(例如16位)写入权重RAM 124。优选地,如以下针对图29A和图29B更详细所述,指令将16个位按照输出二进制小数点2916所指定的那样进行写入。

[0279] 可以发现,通过对循环体进行迭代而写入权重RAM 124的行包含具有无效数据的空洞。也就是说,结果133的宽字1至3、5至7、9至11、依此类推,直到宽字509至511都是无效或未使用的。在一个实施例中,AFU 212包括复用器,其中该复用器使得能够将结果压紧至行缓存器(诸如图11的行缓存器1104等)的相邻字中,以写回输出权重RAM 124行。优选地,激活函数指令指定每个空洞中的字数,并且空洞内的字数用于控制复用器的压紧结果。在一个实施例中,空洞数可指定为2~6的值,以对池化的 3×3 、 4×4 、 5×5 、 6×6 或 7×7 子矩阵

的输出进行压紧。可选地,执行于处理器100上的架构程序从权重RAM 124读取所产生的稀疏(即具有空洞)的结果行,并使用其它执行单元112(诸如使用例如x86SSE指令等的架构压紧指令的媒体单元等)执行压紧功能。有利地,以类似于前述的并行方式并利用NNU 121的混合本质,执行于处理器100上的架构程序可以读取状态寄存器127来监测权重RAM 124的最近写入行(例如图26B的字段2602)以读取所产生的稀疏行,将其压紧并写回权重RAM 124的同一行,使得准备好用作神经网络的下一层(诸如卷积层或传统神经网络层(即乘法累加层)等)的输入数据矩阵。此外,尽管本文所述的实施例对 4×4 子矩阵执行池化运算,但可以对图28的NNU程序进行修改,以对诸如 3×3 、 5×5 、 6×6 或 7×7 子矩阵等的其它大小的子矩阵执行池化运算。

[0280] 还可以发现,写入权重RAM 124的结果行数是输入数据矩阵的行数的四分之一。最后,在该示例中,并未使用数据RAM 122。然而,备选地,可以使用数据RAM 122,而不使用权重RAM 124,来执行池化运算。

[0281] 在图27和图28的示例中,池化运算计算子区域的最大值。然而,图28的程序可被修改为例如通过用sumwacc指令(将权重字与累加器202的值217相加)取代maxwacc指令并将地址9处的激活函数指令更改为将累加结果(优选经由如下所述的倒数相乘)除以各个子区域的元素数(在该示例中为16),来计算子区域的平均值。

[0282] 从根据图27和图28的实施例的NNU 121的运算可以发现,每一次执行图28的程序用约6000个时钟周期来对图27的整个 512×1600 数据矩阵执行池化运算,该时钟周期数可明显少于传统方式执行类似任务所需的时钟周期数。

[0283] 可选地,架构程序将NNU程序配置为将池化运算的结果写回数据RAM 122的行,而非将结果写回权重RAM 124,并且当NNU 121(例如使用图26B的数据RAM 122的最近写入的行2606的地址)将结果写入数据RAM 122时,架构程序从该数据RAM 122读取结果。这种替代方案在权重RAM 124为单端口并且数据RAM 122为双端口的实施例中可以是有利的。

[0284] 具有用户提供的二进制小数点的定点算术,全精度定点累加,用户指定的倒数值,累加器值的随机舍入,以及可选择激活/输出函数

[0285] 一般而言,由于在数字计算装置内执行算术运算的硬件单元对整数和浮点数分别执行算术运算、因而通常可分为“整数”单元与“浮点”单元。浮点数具有幅值(magnitude)(或尾数)和指数,通常还有符号。指数是基数(radix)点(通常为二进制小数点)相对于幅值的位置的指示。相较之下,整数不具有指数,而只具有幅值,通常还有符号。浮点单元的优点在于使程序设计者能够使用从非常大范围内的不同值中取得的数字来工作,而硬件则是在需要时负责调整数字的指数值,而不需程序设计者来进行调整。例如,假定两个浮点数 0.111×10^{29} 与 0.81×10^{31} 相乘。(虽然浮点单元通常工作于以2为基础的浮点数,但这里使用的是十进制小数或以10为基础的示例。)浮点单元自动负责将尾数相乘,将指数相加,随后将结果标准化回到值 $.8911 \times 10^{59}$ 。再例如,假定同样的两个浮点数相加。浮点单元在相加前自动负责将尾数的二进制小数点对齐以产生值为 $.81111 \times 10^{31}$ 的总和。

[0286] 然而,与浮点单元相关联的复杂性以及随之产生的大小、耗能、每指令的时钟周期的增加和/或周期时间的延长是众所周知的。实际上,出于这个原因,许多装置(例如,嵌入式处理器、微控制器以及相对低成本和/或低功率的微处理器)不包括浮点单元。从上述示例可以发现,一些复杂的浮点单元包括:用于执行与浮点加法和乘法/除法相关联的指数计

算的逻辑(即用以对操作数的指数执行加法/减法运算以产生浮点乘法/除法的结果指数值的加法器、用于确定操作数的指数相减以确定浮点加法的二进制小数点对齐移位量的减法器)、用于实现浮点加法中尾数的二进制小数点对齐的移位器、以及用于对浮点结果进行标准化处理的移位器。此外,流程进入方框单元通常需要执行浮点结果的舍入运算的逻辑、在整数格式与浮点格式间以及不同浮点精确度格式(例如扩增精度、双精度、单精度、半精度)间转换的逻辑、前导零与前导一的检测器、以及处理特殊浮点数(例如反常值、非数值和无穷值等)的逻辑。

[0287] 此外,存在如下的缺点:因为设计上需要被验证的数值空间的增加,浮点单元的正确度验证显著变得极其复杂,从而可能延长产品开发周期与上市时间。此外,如上所述,浮点算术意味着对计算所涉及的各浮点数的单独尾数字段和指数字段的储存和使用,这可能增加所需的存储空间的数量和/或在给定等量的存储空间来储存整数的情况下降低精确度。这些缺点中的许多都可以通过使用对整数执行算术运算的整数单元来避免。

[0288] 程序设计者经常编写处理小数即非整数的程序。这种程序可以在如下的处理器上执行,其中这些处理器不具有浮点单元,或者虽然具有浮点单元,但处理器的整数单元所执行的整数指令可以更快。为了利用与整数单元相关联的潜在性能优势,程序设计者对定点(fixed-point)数采用周知的定点算术。这样的程序包括执行于整数单元以处理整数或整数数据的指令。软件知道数据是小数,并且包含用于对整数数据执行运算以应对该数据实际上是小数这一事实的指令(例如,对齐移位)。基本上,定点软件手动执行浮点单元所执行的一些或全部功能。

[0289] 如本文中所使用的,“定点”数(或值或操作数或输入或输出)是一个数字,该数字的储存位被理解为包含表示该定点数的小数部分的位(这里被称为“小数位”)。定点数的储存位包含于存储器或寄存器内,例如存储器或寄存器内的8位或16位字。此外,定点数的储存位全部用来表示一个幅值,并且在某些情况下,其中一个位用来表示符号,但是定点数没有用来表示该数的指数的储存位。此外,在存储时指定该定点数的小数位的数量或二进制小数点位置,这与定点数的储存位不同,并且针对定点数所属的定点数集合(例如输入操作数的集合、累加值的集合或处理单元的阵列的输出结果的集合等)以共享或全局的方式指示小数位的数量或二进制小数点位置。

[0290] 有利地,在本文描述的实施例,ALU是整数单元,但是激活函数单元包含浮点算术硬件辅助或加速。这样使得ALU部分变得更小且更快,从而利于在给定的晶粒空间内使用更多的ALU。这意味着每单位晶粒空间有更多的神经元,从而在神经网络单元中特别有利。

[0291] 此外,有利地,与每个浮点数都需要指数储存位的浮点数相反,描述了如下的实施例,其中以针对全部数字集合的作为小数位的储存位的数量的指示来表示定点数,然而,该指示位于单个共享存储空间中,其全局地指示针对整个集合(例如一系列运算的输入的集合、一系列运算的累加值的集合、输出的集合)的所有数字中的小数位的数量。优选地,NNU的用户能够为数字集合指定小数储存位的数量。因此,应当理解,虽然在许多上下文中(例如一般数学中),术语“整数”是指带符号整数,即不具有小数部分的数字,但是在本上下文中,术语“整数”可以指具有小数部分的数字。此外,在本文的上下文中,术语“整数”旨在与浮点数进行区分,对于浮点数而言,其各自存储空间的位数的一部分用来表示浮点数的指数。类似地,整数算术运算(诸如整数单元执行的整数乘法或加法或比较等)假设操作数不

具有指数,因此,整数单元的整数元件(诸如整数乘法器、整数加法器、整数比较器等)不包含用来处理指数的逻辑,例如不需要为了加法或比较运算而移位尾数来使二进制小数点对齐,不需要为了乘法运算而将指数相加。

[0292] 此外,本文所述的实施例包括大型硬件整数累加器以在不损失精确度的情况下对一系列的大整数运算进行累加(例如,1000量级的乘法累加)。这样使得NNU能够避免处理浮点数,同时又能使累加值维持全精度,而不使其饱和或因为溢位而产生不准确的结果。如以下更详细所述,一旦这系列整数运算将结果累加至全精度累加器,定点硬件辅助就执行必要的缩放和饱和运算,以利用用户指定的累加值小数位的数量和输出值的小数位的期望数量的指示来将全精度累加值转换为输出值。

[0293] 如以下更详细所述,在将累加器值从全精度形式进行压缩以便用作激活函数的输入或者用于通过时,优选地,激活函数单元可以选择性地对该累加器值执行随机舍入。最后,按照神经网络的给定层的不同需求,NPU可以选择性地接受指示以应用不同的激活函数以及/或者输出许多不同形式的累加器值。

[0294] 现参考图29A,示出一框图,该框图示出图1的控制寄存器127的实施例。控制寄存器127可包括多个控制寄存器127。如图所示,控制寄存器127包括以下字段:配置2902、带符号数据2912、带符号权重2914、数据二进制小数点2922、权重二进制小数点2924、ALU函数2926、舍入控制2932、激活函数2934、倒数2942、移位量2944、输出RAM 2952、输出二进制小数点2954、以及输出命令2956。控制寄存器127值可以利用MTNN指令1400和NNU程序的指令(诸如初始化指令等)两者来写入。

[0295] 配置2902的值指定NNU 121是窄配置、宽配置还是漏斗配置,如上所述。配置2902意味着从数据RAM 122和权重RAM 124接收到的输入字的大小。在窄配置与漏斗配置中,输入字的大小是窄的(例如8位或9位),而在宽配置中,输入字的大小是宽的(例如12位或16位)。此外,配置2902意味着与输入字大小相同的输出结果133的大小。

[0296] 带符号数据值2912如果为真则表示从数据RAM 122接收到的数据字是带符号值,如果为假则表示这些数据字是不带符号值。带符号权重值2914如果为真则表示从权重RAM 124接收到的权重字是带符号值,如果为假则表示这些权重字是不带符号值。

[0297] 数据二进制小数点2922的值指示从数据RAM 122接收到的数据字的二进制小数点的位置。优选地,数据二进制小数点2922的值指示从二进制小数点位置的右侧起的位位置的数量。换言之,数据二进制小数点2922指示数据字的最低有效位中的多少位是小数位,即位于二进制小数点右侧。类似地,权重二进制小数点2924的值指示从权重RAM 124接收到的权重字的二进制小数点的位置。优选地,在ALU函数2926是乘法累加或输出累加器的情况下,NPU 126将保持在累加器202中的值的二进制小数点右侧的位数确定为数据二进制小数点2922和权重二进制小数点2924的总和。因此,例如,如果数据二进制小数点2922的值为5而权重二进制小数点2924的值为3,则累加器202内的值在二进制小数点右侧具有8个位。在ALU函数2926是和/最大累加器与数据/权重字、或者通过数据/权重字的情况下,NPU 126将保持在累加器202中的值的二进制小数点右侧的位数分别确定为数据二进制小数点2922/权重二进制小数点2924。在备选实施例中,如以下针对图29B所述,指定单个累加器二进制小数点2923,而非指定各自的数据二进制小数点2922和权重二进制小数点2924。

[0298] ALU函数2926指定由NPU 126的ALU 204执行的函数。如上所述,ALU 函数2926可以

包括但不限于:将数据字209与权重字203相乘并将乘积与累加器202累加;将累加器202与权重字203相加;将累加器202与数据字209相加;累加器202与数据字209中的最大值;累加器202与权重字203中的最大值;输出累加器202;通过数据字209;通过权重字203;输出零值。在一个实施例中,ALU函数2926由NNU初始化指令指定,并且由ALU 204响应于执行指令(未示出)而使用。在一个实施例中,ALU函数2926由个别的NNU指令(诸如上述的乘法累加和maxwacc指令等)指定。

[0299] 舍入控制2932指定(图30中)舍入器3004所使用的舍入的形式。在一个实施例中,可指定的舍入模式包括但不限于:不舍入、舍入至最近值、以及随机舍入。优选地,处理器100包括用于生成随机位3005的(图30的)随机位源3003,其中这些随机位3005被取样并且用来执行随机舍入以降低产生舍入偏置的可能性。在一个实施例中,在舍入位3005为1并且粘滞(sticky)位为零的情况下,如果取样的随机位3005为真,则NPU 126向上舍入,如果随机位3005为假,则NPU 126不向上舍入。在一个实施例中,随机位源3003基于对处理器100的随机电子特性(诸如半导体二极管或电阻的热噪声等)的取样来产生随机位3005,但是还设想了其它实施例。

[0300] 激活函数2934指定适用于累加器202的值217的函数以产生NPU 126的输出133。如上所述并且以下更详细所述,激活函数2934包括但不限于:S型函数;双曲正切函数;软加函数;校正函数;除以二的指定幂次方;乘以用户指定的倒数值以实现等效除法;通过整个累加器;以及如以下更详细所述的作为标准大小通过累加器。在一个实施例中,激活函数由NNU激活函数指令指定。可选地,激活函数由初始化指令指定,并响应于输出指令(例如图4中地址4处的写入AFU输出指令)而应用,在该实施例中,位于图4的地址3的激活函数指令归入输出指令。

[0301] 倒数2942值指定与累加器202的值217相乘以实现累加器202的值217的除法的值。也就是说,用户将倒数2942值指定为实际上期望的除数的倒数。这例如与如本文所述的卷积或池化运算相结合是有用的。优选地,如以下针对图29C更详细所述,用户将倒数2942值指定为两个部分。在一个实施例中,控制寄存器127包括一字段(未示出),其中该字段使得用户能够在多个内建除数值中指定一个进行除法,这些内建除数值的大小相当于常用的卷积核的大小,例如9、25、36或49。在这样的实施例中,AFU 212可以储存这些内建除数的倒数,以供与累加器202值217相乘。

[0302] 移位量2944指定AFU 212的移位器将累加器202的值217右移以实现除以二的幂次方的位数。这与大小为二的幂次方的卷积核的结合也可以是有用的。

[0303] 输出RAM 2952的值指定数据RAM 122和权重RAM 124中的哪一个要接收输出结果133。

[0304] 输出二进制小数点2954的值指示输出结果133的二进制小数点的位置。优选地,输出二进制小数点2954的值指示从输出结果133的二进制小数点位置的右侧起的位位置数量。换言之,输出二进制小数点2954指示输出结果133的最低有效位中的多少位是小数位,即位于二进制小数点右侧。AFU 212基于输出二进制小数点2954的值(在大部分的情况下,还基于数据二进制小数点2922的值、权重二进制小数点2924的值、激活函数2934的值和/或配置2902的值)来执行舍入、压缩、饱和与大小转换。

[0305] 输出命令2956控制输出结果133的各个方面。在一个实施例中,AFU 212利用标准

大小的概念,其中标准大小是配置2902所指定的宽度大小(以位为单位)的两倍。因此,例如,如果配置2902意指从数据RAM 122和权重RAM 124接收到的输入字的大小是8位,则标准大小是16位;在另一个示例中,如果配置2902意指从数据RAM 122与权重RAM 124接收到的输入字的大小是16位,则标准大小是32位。如本文所述,累加器202的大小较大(例如,窄累加器202B是28位,而宽累加器202A是41位)以维持中间计算(例如分别为1024个和512个NNU乘法累加指令)的全精度。如此,累加器202的值217大于(以位为单位)标准大小,并且AFU 212(例如以下针对图30所述的CCS 3008)针对激活函数2934(除了通过全累加器)的大部分值将累加器202的值217向下压缩至具有标准大小的值。输出命令2956的第一预定值指示AFU 212执行指定的激活函数2934以产生大小与原始输入字相同(即,标准大小的一半)的内部结果并将该内部结果作为输出结果133输出。输出命令2956的第二预定值指示AFU 212执行指定的激活函数2934以产生大小为原始输入字的两倍(即,标准大小)的内部结果并将该内部结果的下半部作为输出结果133输出;而输出命令2956的第三预定值指示AFU 212将标准大小的内部结果的上半部作为输出结果133输出。如以上针对图8至图10所述,输出命令2956的第四预定值指示AFU 212将累加器202的原始最低有效字(其宽度由配置2902指定)作为输出结果133输出;第五预定值指示AFU 212将累加器202的原始中间有效字作为输出结果133输出;以及第六预定值指示AFU 212将累加器202的原始最高有效字作为输出结果133输出。如上所述,输出全累加器202大小或标准大小的内部结果例如对于使处理器100的其它执行单元112能够执行诸如软极大激活函数等的激活函数而言可以是有利的。

[0306] 尽管图29A(以及图29B和图29C)的字段被描述为位于控制寄存器127中,但是在其它实施例中,一个或多个字段可以位于NNU 121的其它部分。优选地,许多字段可以包含在NNU指令本身中,并由定序器128进行解码以产生(图34的)用于控制ALU 204和/或AFU 212的微操作3416。此外,这些字段可以包含在储存于媒体寄存器118中的(图34的)微操作3414内,微操作3414控制ALU 204和/或AFU 212。在这样的实施例中,可以使初始化NNU指令的使用减至最少,并且在其它实施例中,去除初始化NNU指令。

[0307] 如上所述,NNU指令能够指定对存储器操作数(例如来自数据RAM 122和/或权重RAM 124的字)或轮转后的操作数(例如来自复用寄存器208/705)执行ALU运算。在一个实施例中,NNU指令还可以将操作数指定为激活函数的寄存器输出(例如图30的寄存器输出3038)。此外,如上所述,NNU指令能够指定使数据RAM 122或权重RAM 124的当前行地址递增。在一个实施例中,NNU指令可以指定被加入当前行的即时的带符号整数差值(delta value)以实现以除了一以外的值递增或递减。

[0308] 现参考图29B,示出一框图,该框图示出根据备选实施例的图1的控制寄存器127的实施例。图29B的控制寄存器127与图29A的控制寄存器127类似;然而,图29B的控制寄存器127包括累加器二进制小数点2923。累加器二进制小数点2923表示累加器202的二进制小数点位置。优选地,累加器二进制小数点2923的值表示从二进制小数点位置的右侧起的位位置数量。换言之,累加器二进制小数点2923表示累加器202的最低有效位中有多少位是小数位,即位于二进制小数点右侧。在该实施例中,累加器二进制小数点2923被明确指定,而非如以上针对图29A的实施例所述隐式地确定。

[0309] 现参考图29C,示出一框图,该框图示出根据一个实施例的以两个部分储存的图29A的倒数2942的实施例。第一部分2962是移位值,其表示用户想要与累加器202的值217相

乘的真实倒数值中被抑制的前导零的数量2962。前导零的数量是紧接在二进制小数点右侧的连续零的数量。第二部分2694是前导零被抑制的倒数2964的值,即将所有前导零移除后的真实倒数值。在一个实施例中,被抑制前导零的数量2962被存储为4位,而前导零被抑制的倒数2964的值则被存储为8位不带符号值。

[0310] 为了通过示例进行说明,假设用户期望将累加器202的值217乘以为49的倒数。以13个小数位表示的49的倒数的二进制表示是0.0000010100111(具有五个前导零)。在这种情况下,用户将被抑制前导零的数量2962填入值5,将前导零被抑制的倒数2964填入值10100111。在(图30的)倒数乘法器“除法器A”3014将累加器202的值217与前导零被抑制的倒数2964的值相乘之后,将所产生的乘积按照被抑制前导零的数量2962进行右移。这样的实施例可以有利地通过利用相对较少的位来表示倒数2942的值来实现高精度度。

[0311] 现参考图30,示出一框图,该框图更详细地示出图2的AFU 212的实施例。AFU 212包含:图1的控制寄存器127;正型转换器(positive form converter,PFC)和输出二进制小数点对齐器(output binary point aligner,OBPA)3002,其用于接收累加器202的值217;舍入器3004,其用于接收累加器202的值217以及OBPA 3002所移出的位数的指示;随机位源3003,如上所述,其用于产生随机位3005;第一复用器3006,其用于接收PFC与OBPA 3002的输出以及舍入器3004的输出;标准大小压缩器(compressor to canonical size,CCS)和饱和器3008,其用于接收第一复用器3006的输出;位选择器和饱和器3012,其用于接收CCS和饱和器3008的输出;校正器3018,其用于接收CCS和饱和器3008的输出;倒数乘法器3014,其用于接收CCS和饱和器3008的输出;向右移位器3016,其用于接收CCS和饱和器3008的输出;双曲正切(tanh)模块3022,其用于接收位选择器和饱和器3012的输出;S型模块3024,其用于接收位选择器和饱和器3012的输出;软加模块3026,其用于接收位选择器和饱和器3012的输出;第二复用器3032,其用于接收双曲正切模块3022、S型模块3024、软加模块3026、校正器3018、倒数乘法器3014、向右移位器3016的输出以及CCS和饱和器3008的通过标准大小的输出3028;符号恢复器3034,其用于接收第二复用器3032的输出;大小转换器和饱和器3036,其用于接收符号恢复器3034的输出;第三复用器3037,其用于接收大小转换器和饱和器3036的输出以及累加器的输出217;以及输出寄存器3038,其用于接收复用器3037的输出并且其输出是图1的结果133。

[0312] PFC和OBPA 3002接收累加器202的值217。优选地,如上所述,累加器202的值217是全精度值。也就是说,累加器202具有足够的储存位数来保持累加值,其中该累加值是由整数加法器244所生成的一系列由整数乘法器242产生的乘积的总和,而不舍弃乘法器242的各乘积或加法器的总和中的任何位,使得不会损失精度。优选地,累加器202至少具有足够的位数来保持NNU 121可被编程执行的乘积累加的最大数量。例如,参考图4的程序来进行说明,在宽配置下NNU 121可被编程执行的乘积累加的最大数量为512,而累加器202的位宽度为41。再例如,参考图20的程序来进行说明,在窄配置下NNU 121可被编程执行的乘积累加的最大数量为1024,而累加器202的位宽度为28。一般地,全精度累加器202具有至少Q个位,其中Q是M与 $\log_2 P$ 的总和,其中M是乘法器242的整数乘积的位宽度(例如,对于窄乘法器242而言为16位,或者对于宽乘法器242而言为32位),而P是可以累加到累加器202的整数乘积的最大容许数量。优选地,乘积累加的最大数量是经由NNU 121的程序设计者的编程规格所指定的。在一个实施例中,在假定从数据/权重RAM 122/124加载数据/权重字206/207的

行的一个先前的乘法累加指令(例如图4的地址1处的指令)的情况下,定序器128将乘法累加NNU指令(例如图4的地址2处的指令)的计数的最大值例如强制设置为511。

[0313] 有利地,通过包括具有足够大的位宽度以对所容许累加的最大数量的全精度值执行累加的累加器202,可简化NPU 126的ALU 204部分的设计。特别是,这样可以缓解对使用逻辑来使整数加法器244产生的总和饱和的需求,其中该逻辑将使小型累加器产生溢位,并且将需要持续追踪累加器的二进制小数点位置以确定是否发生了溢位从而知晓是否需要饱和。为了通过示例对具有非全精度累加器但具有饱和逻辑以处理非全精度累加器的溢位的设计的问题进行说明,假定存在以下情况。

[0314] (1) 数据字的值的范围介于0与1之间,并且所有储存位都用来储存小数位。权重字值的范围介于-8与+8之间,并且除了三个位以外的所有储存位都用来储存小数位。用于双曲正切激活函数的输入的累加值的范围介于-8与8之间,并且除了三个位以外的所有储存位都用来储存小数位。

[0315] (2) 累加器的位宽度为非全精度(例如,只有乘积的位宽度)。

[0316] (3) 假定累加器为全精度,则最终累加值将介于-8与8之间(例如,+4.2);然而,在序列中“点A”前的乘积更常见地倾向为正,而在点A后的乘积更常见地倾向为负。

[0317] 在这种情况下,可能获得不准确的结果(即,+4.2以外的结果)。这是因为在点A前的某些点处,当累加器值本应当是更大的值(例如,+8.2)时,累加器可能饱和到最大值+8,导致损失剩余的+0.2。累加器甚至可以使更多的乘积累加维持在饱和值,从而导致损失更多正值。因此,累加器的最终值可能为比累加器具有全精度位宽度的情况下的最终值(即小于+4.2)更小的值。

[0318] PFC 3002在累加器202的值217为负的情况下将其转换为正型,并产生附加位以指示原始值是正还是负,该附加位与所述值一起向下通过AFU 212的流水线。转换为正型简化了AFU 212的后续运算。例如,该操作使得只有正值才能被输入至双曲正切模块3022和S型模块3024,因而可以简化这些模块。此外,简化了舍入器3004和饱和器3008。

[0319] OBPA 3002向右移位或缩放正型值,以使其与控制寄存器127内指定的输出二进制小数点2954对齐。优选地,OBPA 3002将移位量计算为一差值,所述差值是从累加器202的值217的小数位数(例如,由累加器二进制小数点2923所指定、或是数据二进制小数点2922和权重二进制小数点2924的总和)减去的小数位数(例如,由输出二进制小数点2954所指定)。因此,例如,如果累加器202的二进制小数点2923为8(如上述实施例一样)并且输出二进制小数点2954为3,则OBPA 3002将该正型值右移5位以产生被提供至复用器3006和舍入器3004的结果。

[0320] 舍入器3004对累加器202的值217执行舍入。优选地,舍入器3004产生PFC和OBPA 3002所产生的正型值的舍入后版本,并将该舍入后版本提供至复用器3006。舍入器3004根据上述的舍入控制2932执行舍入,如本文上下文所述,该舍入控制2932可以包括使用随机位3005的随机舍入。复用器3006基于舍入控制2932(如本文所述可包含随机舍入)在其多个输入中选择其一(即来自PFC和OBPA 3002的正型值或来自舍入器3004的舍入后版本),并且将选择的值提供给CCS和饱和器3008。优选地,如果舍入控制2932指定不进行舍入,则复用器3006选择PFC和OBPA 3002的输出,否则选择舍入器3004的输出。在所设想的其它实施例中,AFU 212执行额外的舍入。例如,在一个实施例中,当位选择器3012对CCS和饱和器3008

的输出位进行压缩(如后述)时,位选择器3012基于丢失的低阶位进行舍入。又例如,在一个实施例中,对倒数乘法器3014的乘积(如后述)进行舍入。又例如,在一个实施例中,大小转换器3036在转换为适当的输出大小(如后述)时进行舍入,这可能涉及丢失舍入确定时的低阶位。

[0321] CCS 3008将复用器3006的输出值压缩至标准大小。因此,例如,如果NPU 126为窄配置或漏斗配置2902,则CCS 3008将28位的复用器3006的输出值压缩至16位;而如果NPU 126为宽配置2902,则CCS 3008将41位的复用器3006的输出值压缩至32位。然而,在压缩至标准大小前,如果压缩前值大于标准形式所能表达的最大值,则饱和器3008就会使压缩前值饱和至标准形式所能表达的最大值。例如,如果压缩前值中位于最高有效标准形式位左侧的任意位具有值1,则饱和器3008饱和至最大值(例如,饱和至全部为1)。

[0322] 优选地,双曲正切模块3022、S型模块3024和软加模块3026都包含查找表,例如可编程逻辑阵列(PLA)、只读存储器(ROM)、组合逻辑门等。在一个实施例中,为了简化并缩小这些模块3022/3024/3026的大小,向这些模块提供具有3.4形式的输入值,即三个整数位和四个小数位,也即输入值在二进制小数点右侧具有四个位并且在二进制小数点左侧具有三个位。选择这些值是因为在3.4形式的输入值范围(-8,+8)的极端处、输出值渐近地靠近其最小/最大值。然而,设想了例如采用4.3形式或2.5形式将二进制小数点放置在不同位置的其它实施例。位选择器3012在CCS和饱和器3008的输出中选择满足3.4形式标准的位,这涉及压缩处理,也就是会丢失某些位,这是因为标准形式具有较多的位数。然而,在选择/压缩CCS和饱和器3008的输出值之前,如果压缩前值大于3.4形式所能表达的最大值,则饱和器3012使压缩前值饱和至3.4形式所能表达的最大值。例如,如果压缩前值中位于3.4形式的最高有效位左侧的任意位具有值1,则饱和器3012饱和至最大值(例如,饱和至全部为1)。

[0323] 双曲正切模块3022、S型模块3024和软加模块3026对CCS和饱和器3008所输出的3.4形式值执行相应的激活函数(如上所述)以产生结果。优选地,双曲正切模块3022和S型模块3024的结果是0.7形式的7位结果,即零个整数位和七个小数位,也即输入值在二进制小数点右侧具有七个位。优选地,软加模块3026的结果是3.4形式的7位结果,即其形式与该模块3026的输入形式相同。优选地,双曲正切模块3022、S型模块3024和软加模块3026的输出被扩展至标准形式(例如在必要时加上前导零)并对齐以具有输出二进制小数点2954值所指定的二进制小数点。

[0324] 校正器3018产生CCS和饱和器3008的输出值的校正后版本。也就是说,如果CCS和饱和器3008的输出值(如前述其符号以流水线下移)为负,则校正器3018输出零值;否则,校正器3018将其输入值输出。优选地,校正器3018的输出为标准形式并具有输出二进制小数点2954值所指定的二进制小数点。

[0325] 倒数乘法器3014将CCS和饱和器3008的输出与倒数值2942中所指定的用户指定倒数值相乘,以产生其标准大小的乘积,其中该乘积实际上为CCS和饱和器3008的输出与作为倒数2942值的倒数的除数的商。优选地,倒数乘法器3014的输出为标准形式并具有由输出二进制小数点2954的值所指定的二进制小数点。

[0326] 向右移位器3016将CCS与饱和器3008的输出按移位量值2944中所指定的用户指定位数进行移位,以产生其标准大小的商。优选地,向右移位器3016的输出为标准形式并具有由输出二进制小数点2954的值所指定的二进制小数点。

[0327] 复用器3032选择激活函数2934的值所指定的适当输入,并将选择提供至符号恢复器3034,其中该符号恢复器3034在原始累加器202的值217为负值的情况下将复用器3032的正型输出转换为负型,例如转换为二补码形式。

[0328] 大小转换器3036基于以上针对图29A所述的输出命令2956的值,将符号恢复器3034的输出转换至适当的大小。优选地,符号恢复器3034的输出具有由输出二进制小数点2954的值指定的二进制小数点。优选地,对于输出命令2956的第一预定值而言,大小转换器3036舍弃符号恢复器3034输出的上半部分的位。此外,如果符号恢复器3034的输出为正并且超过配置2902指定的字大小所能表达的最大值、或者输出为负并且小于字大小所能表达的最小值,则饱和器3036将其输出分别饱和至该字大小所能表达的最大值/最小值。对于第二预定值和第三预定值,大小转换器3036使符号恢复器3034的输出通过。

[0329] 复用器3037基于输出命令2956选择大小转换器和饱和器3036的输出或累加器202的输出217以提供给输出寄存器3038。更具体地,对于输出命令2956的第一预定值和第二预定值,复用器3037选择大小转换器和饱和器3036的输出的低位字(其大小由配置2902指定)。对于第三预定值,复用器3037选择大小转换器与饱和器3036的输出的高位字。对于第四预定值,复用器3037选择原始的累加器202的值217的低位字;对于第五预定值,复用器3037选择原始的累加器202的值217的中间字;并且对于第六预定值,复用器3037选择原始的累加器202值217的高位字。如上所述,优选地,AFU 212在原始的累加器202的值217的高位字的高位填入零。

[0330] 现参考图31,示出图30的AFU 212的操作的示例。如图所示,配置2902被设置为NPU 126的窄配置。此外,带符号数据2912与带符号权重2914的值为真。此外,数据二进制小数点2922的值表示针对数据RAM 122的字的二进制小数点被定位为二进制小数点右侧有7位,并且NPU 126其中之一所接收的第一数据字的示例值被示为0.1001110。此外,权重二进制小数点2924的值表示针对权重RAM 124的字的二进制小数点被定位为二进制小数点右侧有3位,并且NPU 126其中之一所接收的第一权重字的示例值被示为000001.010。

[0331] 第一数据字和第一权重字的16位乘积(此乘积与累加器202的初始零值累加)被示为000000.1100001100。由于数据二进制小数点2912是7且权重二进制小数点2914是3,因此隐含的累加器202的二进制小数点被定为为二进制小数点右侧有10个位。在窄配置的情况下,在示例性实施例中,累加器202的宽度为28位。在示例中,示出在执行了所有ALU运算(例如图20中的全部1024个乘法累加)之后累加器202的值217为00000000000000000001.1101010100。

[0332] 输出二进制小数点2954的值表示输出的二进制小数点被定位为在二进制小数点的右侧有7个位。因此,在通过OBPA 3002和CCS 3008之后,对累加器202的值217进行缩放、舍入并压缩至标准形式的值,即000000001.1101011。在该示例中,输出二进制小数点位置表示7个小数位,并且累加器202的二进制小数点位置表示10个小数位。因此,OBPA 3002计算出差值3,并通过将累加器202的值217右移3位来对其进行缩放。这在图31中表示为累加器202的值217丧失3个最低有效位(二进制数100)。此外,在该示例中,舍入控制2932的值表示使用随机舍入,并且在该示例中假定取样随机位3005为真。因此,根据以上说明,最低有效位被向上舍入,这是因为累加器202的值217的舍入位(通过累加器202的值217的缩放而被移出的3个位中的最高有效位)为1,而粘滞位(通过累加器202的值217的缩放而被移出的

3个位中的2个最低有效位的布尔或运算结果)为0。

[0333] 在本示例中,激活函数2934表示要使用S型函数。因此,位选择器3012选择标准形式值的位以使得S型模块3024的输入具有三个整数位和四个小数位,如上所述,即如图所示的值001.1101。S型模块3024输出以标准形式设置的值,即所示的值000000000.1101110。

[0334] 该示例的输出命令2956指定第一预定值,即输出由配置2902所表示的字大小,在这种情况下为窄字(8位)。因此,大小转换器3036将标准S型输出值转换为具有隐含二进制小数点的8位量,所述二进制小数点被定位为在该二进制小数点右侧有7个位,从而如图所示产生输出值01101110。

[0335] 现参考图32,示出图30的AFU 212的操作的第二示例。图32的示例示出在激活函数2934表示使累加器202的值217以标准大小通过的情况下的AFU 212的操作。如图所示,配置2902被设置为NPU 126的窄配置。

[0336] 在该示例中,累加器202的宽度为28个位,并且累加器202的二进制小数点被定位为二进制小数点右侧有10个位(如上所述,这是因为根据一个实施例数据,二进制小数点2912和权重二进制小数点2914的总和为10,或者因为根据另一实施例,累加器二进制小数点2923明确被指定为具有值10)。在该示例中,图32示出在执行所有的ALU运算之后累加器202的值217,即000001100000011011.1101111010。

[0337] 在该示例中,输出二进制小数点2954的值表示输出的二进制小数点被定位为二进制小数点的右侧有4个位。因此,在通过OBPA 3002和CCS 3008之后,如图所示,累加器202的值217被饱和并压缩至标准形式值11111111111.1111,该值由复用器3032接收作为标准大小的通过值3028。

[0338] 在该示例中,示出两个输出命令2956。第一输出命令2956指定第二预定值,即输出标准形式大小的低位字。由于配置2902所指示的大小为窄字(8位)(意味着标准大小是16位),因此大小转换器3036选择标准大小通过值3028的低8位以产生如图所示的8位值11111111。第二输出命令2956指定第三预定值,即输出标准形式大小的高位字。因此,大小转换器3036选择标准大小通过值3028的高8位以产生如图所示的8位值11111111。

[0339] 现参考图33,示出图30的AFU 212的操作的第三示例。图33的示例示出在激活函数2934表示要使整个原始的累加器202的值217通过的情况下的AFU 212的操作。如图所示,配置2902被设置为NPU 126的宽配置(例如,16位输入字)。

[0340] 在该示例中,累加器202为41位宽,并且累加器202的二进制小数点被定位为在二进制小数点的右侧有8个位(如上所述,这是因为根据一个实施例,数据二进制小数点2912和权重二进制小数点2914的总和为8,或者因为根据另一实施例,累加器二进制小数点2923明确被指定为具有值8)。在该示例中,图33示出在执行了所有ALU运算之后累加器202的值217即001000000000000000001100000011011.11011110。

[0341] 在该示例中,示出三个输出命令2956。第一输出命令2956指定第四预定值,即输出原始累加器202值的低位字;第二个输出命令2956指定第五预定值,即输出原始累加器202值的中间字;以及第三个输出命令2956指定第六预定值,即输出原始累加器202值的高位字。由于配置2902所指示的大小为宽字(16位),因此图33示出,响应于第一输出命令2956,复用器3037选择16位值0001101111011110;响应于第二输出命令2956,复用器3037选择16位值00000000000011000;而响应于第三输出命令2956,复用器3037选择16位值

0000000001000000。

[0342] 如上所述,有利地,NNU 121对整数数据而非浮点数据执行运算。这有利于简化各NPU 126或至少ALU 204部分。例如,ALU 204无需包括浮点实现中为了将乘法器242的乘数的指数相加所需的加法器。类似地,ALU 204无需包括浮点实现中为了将加法器234的加数的二进制小数点对齐所需的移位器。本领域技术人员应当理解,浮点单元一般非常复杂;因此,这些仅仅是针对ALU 204的简化的范例,并且利用使得用户能够指定相关二进制小数点的硬件定点辅助可以通过即时整数实施例来实现其它简化。相较于浮点的实施例,ALU 204是整数单元这一事实可以有利地产生更小(且更快)的NPU 126,这进一步有利于将大型的NPU 126阵列整合到NNU 121内。AFU 212的部分可以基于(优选为用户指定的)累加值所需的小数位数量以及输出值所需的小数位数量,来处理累加器202的值217的缩放和饱和。有利地,如针对图11的实施例所述,任何额外复杂度以及伴随而来的大小的增加、AFU 212的定点硬件辅助内的功率和/或时间损耗,都可以通过在ALU 204部分间共享AFU 212的方式来进行分摊,例如这是因为采用共享的实施例可以减少AFU 1112的数量。

[0343] 有利地,本文所述的实施例享有许多与硬件整数算术单元的降低的复杂度相关联的益处(相较于使用浮点算术单元),同时仍提供针对小数(即具有二进制小数点的数)的算术运算。浮点算术的优点在于:针对单个值可能落在一个非常宽的值范围(该值范围实际上只受限指数范围的大小,而该大小可能非常大)内的任意位置的数据,提供算术运算。也就是说,每个浮点数具有其潜在唯一的指数值。然而,本文所述的实施例认识到并利用如下事实,即:存在某些应用,其中在这些应用中,输入数据高度平行,并且其值处于相对较窄的范围内,使得所有平行值的“指数”可以相同。因此,这些实施例使得用户能够一次针对所有的输入值和/或累加值指定二进制小数点位置。类似地,通过认识到并利用平行输出的类似范围特性,这些实施例使得用户能够一次针对所有的输出值指定二进制小数点位置。人工神经网络是这种应用的一个示例,但是本发明的实施例也可用于执行其它应用的计算。通过针对输入一次指定二进制小数点位置而非针对各单独的输入数进行该指定,相较于浮点实现,实施例可以更有效率地使用存储空间(例如,需要较少的存储器),以及/或者在使用类似数量的存储器的情况下提升精度,这是因为在浮点实现中用于指数的位可用来指定幅值的更大精度。

[0344] 此外有利地,这些实施例认识到在对大的一系列整数运算执行累加期间可能经历的潜在的精度丧失(例如,溢位或丧失较不重要的小数位),并且提供一个解决方法,主要是采用一个足够大的累加器来避免精度丧失的形式。

[0345] NNU微操作的直接执行

[0346] 现参考图34,示出一框图,该框图示出图1的处理器100以及NNU 121的部分细节。NNU 121包括NPU 126的流水线阶段3401。以阶段寄存器分隔开的流水线阶段3401包括实现如本文所述的NPU 126的运算的组合逻辑,诸如布尔逻辑门、复用器、加法器、乘法器、比较器等。流水线阶段3401从复用器3402接收微操作3418。微操作3418向下流动至流水线阶段3401并控制其组合逻辑。微操作3418是位的集合。优选地,微操作3418包括数据RAM 122的存储器地址123的位、权重RAM 124的存储器地址125的位、程序存储器129的存储器地址131的位、复用寄存器208/705的控制信号213/713的位、复用器802的控制信号803的位、以及(例如图29A至图29C的)控制寄存器217的许多字段等。在一个实施例中,微操作3418包括约

120个位。复用器3402从三个不同的源接收微操作,并选择其中一个作为提供给流水线阶段3401的微操作3418。

[0347] 复用器3402的一个微操作源是图1的定序器128。定序器128将从程序存储器129接收到的NNU指令解码,并且作为响应而产生提供至复用器3402的第一输入的微操作3416。

[0348] 复用器3402的第二个微操作源是从图1的保留站108接收微指令105以及从GPR 116和媒体寄存器118接收操作数的解码器3404。优选地,如上所述,微指令105是指令转译器104响应于对MTNN指令1400和MFNN指令1500的转译而产生的。微指令105可包括立即字段,该立即字段指定特定函数(由MTNN指令1400或MFNN指令1500所指定),诸如开始和停止执行程序存储器129内的程序、直接从媒体寄存器118执行微操作、或者如上所述读取/写入NNU的存储器等。解码器3404对微指令105进行解码并作为响应而产生提供至复用器3402的第二输入的微操作3412。优选地,响应于MTNN指令1400/MFNN指令1500的某些函数1432/1532,解码器3404无需产生沿流水线3401向下发送的例如写入控制寄存器127、开始执行程序存储器129内的程序、暂停执行程序存储器129内的程序、等待程序存储器129内的程序完成执行、从状态寄存器127读取以及重设NNU 121等的微操作3412。

[0349] 复用器3402的第三个微操作源是媒体寄存器118本身。优选地,如以上针对图14所述,MTNN指令1400可指定一函数以指示NNU 121直接执行从媒体寄存器118提供至复用器3402的第三输入的微操作3414。直接执行由架构媒体寄存器118提供的微操作3414可以特别有助于对NNU 121进行测试(例如内在自测试(BIST))和调试。

[0350] 优选地,解码器3404产生用于控制复用器3402选择的模式指示符3422。当MTNN指令1400指定一函数开始运行来自程序存储器129的程序时,解码器3404产生使复用器3402从定序器128选择微操作3416的模式指示符3422值,直到发生错误为止或直到解码器3404遇到指定函数停止运行来自程序存储器129的程序的MTNN指令1400为止。当MTNN指令1400指定用于指示NNU 121直接执行从媒体寄存器118提供的微操作3414的函数时,解码器3404产生使复用器3402从指定媒体寄存器118选择微操作3414的模式指示符3422值。否则,解码器3404产生使复用器3402选择从解码器3404选择微操作3412的模式指示符3422值。

[0351] 可变速率神经网络单元

[0352] 可能存在如下情形:NNU 121运行程序,然后进入空闲状态等待处理器100处理一些在其可以执行下一个程序前需要处理的事情。例如,假设在与针对图3至图6A所述的类似的情形,NNU 121对乘法累加激活函数程序(也可称为前馈神经网络层程序)连续运行两次或更多次。相较于NNU 121运行程序所花费的时间,处理器100明显花费更长的时间来将NNU程序下一次运行所用的512KB大小的权重值写入权重RAM 124。换言之,NNU 121可以在相对短的时间内运行程序,然后进入空闲状态,同时处理器100完成将接下来的权重值写入权重RAM 124以供下一次程序运行用。这种情形在以下更详细所述的图36A中可视地示出。在这种情形下,可以有利地使NNU 121以较慢速率运行并且花更长的时间来执行程序,以使NNU 121运行程序所需的能量消耗分散至较长的时间内,这可以倾向于使NNU 121乃至处理器100维持在较低温度。这种情形被称为缓和模式并且在以下更详细所述的图36B中可视地示出。

[0353] 现参考图35,示出一框图,该框图示出具有可变速率NNU 121的处理器100。该处理器100在许多方面与图1的处理器100类似,并且具有相同附图标记的元件是相同的。图35的

处理器100还包括耦接至处理器100的功能单元的时钟产生逻辑3502,这些功能单元即指令拾取单元101、指令高速缓存器102、指令转译器104、重命名单元106、保留站108、NNU 121、其它执行单元112、存储器子系统114、通用寄存器116以及媒体寄存器118。时钟产生逻辑3502包括诸如锁相环(PLL)等的时钟产生器,其产生具有主时钟速率或主时钟频率的时钟信号。例如,主时钟速率可以是1GHz、1.5GHz、2GHz等。时钟速率表示每秒的时钟信号的周期数,例如在高低状态之间的震荡数。优选地,时钟信号具有平衡的占空比,即周期的一半为高而另一半为低;可选地,时钟信号具有不平衡的占空比,其中时钟信号处在高状态的时间长于其处在低状态的时间,或者反之亦然。优选地,PLL能够被配置为产生具有多个时钟速率的主时钟信号。优选地,处理器100包括电源管理模块,其基于多种因素自动调整主时钟速率,这些因素包括动态检测到的处理器100的操作温度、利用率、以及来自系统软件(例如操作系统、BIOS)的指示期望性能和/或节能指标的命令。在一个实施例中,电源管理模块包括处理器100的微码。

[0354] 时钟产生逻辑3502还包括时钟分配网络或时钟树。时钟树将主时钟信号分配至处理器100的功能单元,即如图35所示将时钟信号3506-1分配至指令拾取单元101,将时钟信号3506-2分配至指令高速缓存器102,将时钟信号3506-10分配至指令转译器104,将时钟信号3506-9分配至重命名单元106,将时钟信号3506-8分配至保留站108,将时钟信号3506-7分配至NNU 121,将时钟信号3506-4分配至其它执行单元112,将时钟信号3506-3分配至存储器子系统114,将时钟信号3506-5分配至通用寄存器116,以及将时钟信号3506-6分配至媒体寄存器118,这些信号共同地称为时钟信号3506。时钟树包括节点或线,用于将主时钟信号3506传输至其相应的功能单元。此外,优选地,时钟产生逻辑3502包括时钟缓存器,时钟缓存器(特别是对于较远的节点)在需要提供较干净的时钟信号时重新产生主时钟信号、和/或提升主时钟信号的电压水平。此外,各个功能单元在需要时还可以包括其自身的子时钟树,子时钟树重新产生和/或提升其所接收到的相应主时钟信号3506。

[0355] NNU 121包括时钟降低逻辑3504,其接收缓和指示符3512、接收主时钟信号3506-7、并且作为响应而产生辅时钟信号。辅时钟信号具有如下的时钟速率,该时钟速率与主时钟速率相同,或者在缓和模式的情况下该时钟速率相对于主时钟速率降低了被编程到缓和指示符3512中的量,从而潜在地提供热量方面的益处。时钟降低逻辑3504在许多方面与时钟产生逻辑3502类似,时钟降低逻辑3504具有时钟分配网络或时钟树,所述时钟分配网络或时钟树将辅时钟信号分配至NNU 121的各个方框,如表示为将时钟信号3508-1分配至NPU 126的阵列,将时钟信号3508-2分配至定序器128、以及将时钟信号3508-3分配至接口逻辑3514,这些信号共同地或单独地称为辅时钟信号3508。优选地,如针对图34所示,NPU 126包括多个流水线阶段3401,流水线阶段包括用于从时钟降低逻辑3504接收辅时钟信号3508-1的流水线阶段寄存器。

[0356] NNU 121还包括用于接收主时钟信号3506-7和辅时钟信号3508-3的接口逻辑3514。接口逻辑3514耦接于处理器100前端的下部分(例如,保留站108、媒体寄存器118和通用寄存器116)和NNU 121的各个方框之间,这些方框为时钟降低逻辑3504、数据RAM 122、权重RAM 124、程序存储器129和定序器128。接口逻辑3514包括数据RAM缓存器3522、权重RAM缓存器3524、图34的解码器3404、以及缓和指示符3512。缓和指示符3512保持指定NPU 126的阵列将以多慢的速度执行NNU程序指令的值。优选地,缓和指示符3512指定除数值N,时钟

降低逻辑3504将主时钟信号3506-7除以该除数值以产生辅时钟信号3508,使得辅时钟信号的速率为 $1/N$ 。优选地, N 的值可被编程为多个不同预定值中的任一个,以使时钟降低逻辑3504产生具有多个不同速率的辅时钟信号3508,其中所述多个不同速率都小于主时钟速率。

[0357] 在一个实施例中,时钟降低逻辑3504包括时钟除法器电路,用以将主时钟信号3506-7除以缓和指示符3512的值。在一个实施例中,时钟降低逻辑3504包括时钟门(例如,与门),该时钟门利用使能信号来对主时钟信号3506-7进行门控,其中主时钟信号3506-7的每 N 个周期仅使能信号一次为真。例如,可以使用包含用于计数多达 N 的计数器的电路来产生使能信号。当伴随的逻辑检测到计数器的输出与 N 匹配时,逻辑在辅时钟信号3508上产生真值脉冲并重设计数器。优选地,缓和指示符3512的值可通过架构指令(诸如图14的MTNN指令1400等)进行编程。优选地,如针对图37更详细所述,仅在指示NNU 121开始运行NNU程序之前,运行于处理器100上的架构程序将缓和值编程到缓和指示符3512中。

[0358] 权重RAM缓存器3524耦接于权重RAM 124和媒体寄存器118之间,用于缓存它们之间的数据传输。优选地,权重RAM缓存器3524与图17的缓存器1704的一个或多个实施例类似。优选地,权重RAM缓存器3524中的从媒体寄存器118接收数据的部分以具有主时钟速率的主时钟信号3506-7作为时钟,而权重RAM缓存器3524中的从权重RAM 124接收数据的部分以具有辅时钟速率的辅时钟信号3508-3作为时钟,其中该辅时钟速率可以根据编程到缓和指示符3512中的值(即根据NNU 121是在缓和模式还是一般模式下操作)而相对于主时钟速率降低或不降低。在一个实施例中,如以上针对图17所述,权重RAM 124为单端口,并且可由媒体寄存器118经由权重RAM缓存器3524以及由图11的NPU 126或行缓存器1104两者以仲裁方式(arbitrated fashion)访问。在备选实施例中,如以上针对图16所述,权重RAM 124为双端口,并且各端口可由媒体寄存器118经由权重RAM缓存器3524以及NPU 126或行缓存器1104两者以并行方式访问。

[0359] 同样,数据RAM缓存器3522耦接于数据RAM 122和媒体寄存器118之间,用于缓存它们之间的数据传输。优选地,数据RAM缓存器3522与图17的缓存器1704的一个或多个实施例类似。优选地,数据RAM缓存器3522中的从媒体寄存器118接收数据的部分以具有主时钟速率的主时钟信号3506-7作为时钟,而数据RAM缓存器3522中的从数据RAM 122接收数据的部分以具有辅时钟速率的辅时钟信号3508-3作为时钟,其中该辅时钟速率可以根据编程到缓和指示符3512中的值(即根据NNU 121是在缓和模式还是一般模式下操作)而相对于主时钟速率降低或不降低。在一个实施例中,如以上针对图17所述,数据RAM 122为单端口,并且可由媒体寄存器118经由数据RAM缓存器3522以及由图11的NPU 126或行缓存器1104两者以仲裁方式访问。在备选实施例中,如以上针对图16所述,数据RAM 122为双端口,并且各端口可由经由数据RAM缓存器3522的媒体寄存器118以及NPU 126或行缓存器1104两者以并行方式访问。

[0360] 优选地,无论数据RAM 122和/或权重RAM 124是单端口还是双端口,接口逻辑3514都包括数据RAM缓存器3522和权重RAM缓存器3524以提供主时钟域和辅时钟域之间的同步。优选地,数据RAM 122、权重RAM 124和程序存储器129各自包括静态RAM(SRAM),其中该静态RAM包括相应的读取使能信号、写入使能信号和存储器选择信号。

[0361] 如上所述,NNU 121是处理器100的执行单元。执行单元是处理器的执行架构指令

被转译成的微指令(诸如图1中的架构指令103转移成的微指令105等)或执行架构指令103本身的功能单元。执行单元从处理器的通用寄存器(诸如GPR 116和媒体寄存器118等)接收操作数。执行单元响应于执行微指令或架构指令而产生可被写入通用寄存器的结果。架构指令103的示例分别为针对图14和图15所述的MTNN指令1400和MFNN指令1500。微指令实现架构指令。更具体地,执行单元对于架构指令被转译成的一个或多个微指令的整体执行对架构指令所指定的输入执行架构指令所指定的运算,以产生架构指令所定义的结果。

[0362] 现参考图36A,示出一时序图,该时序图示出具有在一般模式中即以主时钟速率操作的NNU 121的处理器100的操作示例。在时序图中,时间的进程是从左至右。处理器100正以主时钟速率运行架构程序。更具体地,处理器100的前端(例如,指令拾取单元101、指令高速缓存器102、指令转译器104、重命名单元106和保留站108)以主时钟速率拾取、解码且发布架构指令至NNU 121和其它执行单元112。

[0363] 最初,架构程序执行架构指令(例如,MTNN指令1400),处理器前端100将该架构指令发布至NNU 121以指示NNU 121开始运行其程序存储器129内的NNU程序。此前,架构程序执行架构指令以将用于指定主时钟速率的值写入缓和指示符3512,即使NNU处于一般模式。更具体地,被编程到缓和指示符3512中的值使时钟降低逻辑3504产生具有主时钟信号3506的主时钟速率的辅时钟信号3508。优选地,在这种情况下,时钟降低逻辑3504的时钟缓存器单纯提升主时钟信号3506。另外在此之前,架构程序执行架构指令以写入数据RAM 122和权重RAM 124并将NNU程序写入程序存储器129中。响应于开始NNU程序的MTNN指令1400,NNU 121开始以主时钟速率执行NNU程序,这是因为缓和指示符3512被编程为具有主速率值。在开始NNU 121运行后,架构程序继续以主时钟速率执行架构指令,包括并且主要以MTNN指令1400写入和/或读取数据RAM 122和权重RAM 124,以为NNU程序的下一次实例或调用或运行做准备。

[0364] 如图36A中的示例所示,相较于架构程序完成对数据RAM 122和权重RAM 124的写入/读取所花费的时间,NNU 121在明显更少的时间(例如四分之一的时间内)完成NNU程序的运行。例如,都以主时钟速率,NNU 121可能花费大约1000个时钟周期来运行NNU程序,而架构程序花费大约4000个时钟周期来运行。因此,NNU 121在剩下的时间(在本示例中为相当长的时间,例如,大约3000个主时钟速率周期)内处于空闲。如图36A中的示例所示,根据神经网络的大小和配置,该模式继续执行另一次,并可能继续多次。因为NNU 121可以是处理器100中相对大且晶体管密集的功能单元,因此NNU 121可能产生大量热,尤其是在以主时钟速率操作时。

[0365] 现参考图36B,示出一时序图,该时序图示出具有在缓和模式中即以比主时钟速率小的速率操作的NNU 121的处理器100的操作示例。图36B的时序图在许多方面与图36A的时序图相同,即处理器100以主时钟速率运行架构程序。并且在该示例中,假定图36B的架构程序和NNU程序与图36A的架构程序和NNU程序相同。然而,在开始NNU程序之前,架构程序执行MTNN指令1400,其中该MTNN指令1400将缓和指示符3512编程有令时钟降低逻辑3504产生具有小于主时钟速率的辅时钟速率的辅时钟信号3508的值。也就是说,架构程序使NNU 121处于图36B的缓和模式,而非图36A的一般模式。因此,NPU 126以辅时钟速率执行NNU程序,其中在缓和模式下,该辅时钟速率小于主时钟速率。在该示例中,假定缓和指示符3512被编程有利于将辅时钟速率指定为四分之一主时钟速率的值。作为结果,如比较图36A和图36B可

以看出, NNU 121在缓和模式下运行NNU程序所花费的时间是在一般模式下运行NNU程序所花费的时间的四倍长, 使得NNU 121处于空闲状态的时间量相对较短。因此, 图36B中NNU 121在大约是图36A中NNU 121在一般模式下运行程序所需的时间的四倍的时间段内消耗用于运行NNU程序的能量。因此, 图36B中NNU 121运行NNU程序所产生的热量的速率大约是图36A中的四分之一, 因而可具有本文所述的热量方面的益处。

[0366] 现参考图37, 示出一流程图, 该流程图示出图35的处理器100的操作。该流程图所示的操作在许多方面与以上针对图35、图36A和图36B所述的操作相同。流程始于方框3702。

[0367] 在方框3702处, 处理器100执行MTNN指令1400以将权重写入权重RAM 124并且将数据写入数据RAM 122。流程进入方框3704。

[0368] 在方框3704处, 处理器100执行MTNN指令1400以将指示符3512编程有指定低于主时钟速率的速率的值, 即使NNU 121处于缓和模式。流程进入方框3706。

[0369] 在方框3706处, 与图36B所呈现的方式相同, 处理器100执行MTNN指令1400以指示NNU 121开始运行NNU程序。流程进入方框3708。

[0370] 在方框3708处, NNU 121开始运行NNU程序。并行地, 处理器100执行MTNN指令1400以将新的权重写入权重RAM 124 (并且可能将新的数据写入数据RAM 122), 以及/或者执行MFNN指令1500以从数据RAM 122读取结果 (并且可能从权重RAM 124读取结果)。流程进入方框3712。

[0371] 在方框3712处, 处理器100执行MFNN指令1500 (例如读取状态寄存器127), 以检测NNU 121已结束运行其程序。假设架构程序选择良好的缓和指示符3512的值, 则如图36B所示, NNU 121运行NNU程序所花费的时间量与处理器100执行架构程序的访问权重RAM 124和/或数据RAM 122的部分所花费的时间大致相同。流程进入方框3714。

[0372] 在方框3714处, 处理器100执行MTNN指令1400以将缓和指示符3512编程有指定主时钟速率的值, 即使NNU 121处于一般模式。流程进入方框3716。

[0373] 在方框3716处, 与类似图36A所呈现的方式相同, 处理器100执行MTNN指令1400以指示NNU 121开始运行NNU程序。流程进入方框3718。

[0374] 在方框3718处, NNU 121开始以一般模式运行NNU程序。流程结束于方框3718。

[0375] 如上所述, 相对于NNU在一般模式下 (即以处理器的主时钟速率) 运行程序的时间, 在缓和模式下运行NNU程序可以使NNU运行程序的时间分散, 从而能够提供热量方面的益处。更具体地, 当NNU在缓和模式下运行程序时, 由于NNU以较慢的速率产生热量, 而这些热量被NNU (例如半导体装置、金属层和底层基材) 以及周围的封装体和冷却方案 (例如, 散热片、风扇) 耗散, 因此装置 (例如晶体管、电容、导线) 将有可能在较低的温度下工作。一般来说, 这还降低了处理器晶粒的其它部分内的装置温度。装置的较低工作温度 (特别是它们的接面温度) 可以具有减少漏电流的益处。此外, 由于单位时间内流过的电流量较少, 因此电感噪声和IR压降噪声也可以降低。另外, 较低温度对于处理器的MOSFET的负偏压温度不稳定性 (NBTI) 和正偏压温度不稳定性 (PBTI) 也有正面影响, 从而提升装置以及处理器部分的可靠度和/或寿命。较低温度还可以减轻处理器的金属层内的焦耳热和电迁移。

[0376] 关于NNU共享资源的架构程序与非架构程序之间的通信机制

[0377] 如上所述, 以图24至图28以及图35至37为例, 数据RAM 122和权重RAM 124是共享资源。NPU 126和处理器100的前端这两者共享数据RAM 122和权重RAM 124。更具体地, NPU

126与处理器100的前端(例如媒体寄存器118)对数据RAM 122和权重RAM 124进行读取与写入。换言之,运行于处理器100上的架构程序与运行于NNU 121上的NNU程序共享数据RAM 122和权重RAM 124,并且如上所述,在某些情形下,这需要对架构程序和NNU程序间的流程进行控制。该资源共享在某种程度上也适用于程序存储器129,这是因为架构程序对程序存储器129进行写入,而定序器128对程序存储器129进行读取。本文上下文所述的实施例提供用以控制架构程序和NNU程序之间对共享资源的访问流程的高性能解决方案。

[0378] 在本文所述的实施例中,NNU程序也称为非架构程序,NNU指令也称为非架构指令,而NNU指令集(在上文中也称为NPU指令集)也称为非架构指令集。非架构指令集不同于架构指令集。在处理器100包含用于将架构指令转译成微指令的指令转译器104的实施例中,非架构指令集也不同于微指令集。

[0379] 现参考图38,示出一框图,该框图更详细地示出NNU 121的定序器128。如上所述,定序器128将存储器地址131提供至程序存储器129,以选择被提供给定序器128的非架构指令。如图38所示,存储器地址131保持在定序器128的程序计数器3802内。定序器128通常以程序存储器129的顺序地址递增,除非定序器128遇到诸如循环或分支指令等的非架构指令,在这种情况下,定序器128将程序计数器3802更新为控制指令的目标地址,即更新为位于控制指令的目标处的非架构指令的地址。因此,保持在程序计数器3802的地址131指定当前被拾取以供NPU 126执行的非架构程序的非架构指令的程序存储器129中的地址。有利地,如以下针对图39所述,程序计数器3802的值可以由架构程序经由状态寄存器127的NNU程序计数器字段3912而获得。这使得架构程序能够基于非架构程序的进度来决定相对于数据RAM 122和/或权重RAM 124在哪里进行数据读取/写入。

[0380] 定序器128还包括循环计数器3804,该循环计数器3804与非架构循环指令相结合地使用,其中非架构循环指令诸如为图26A的地址10处的循环至1的指令以及图28的地址11处的循环至1的指令等。在图26A和图28的示例中,循环计数器3804加载地址0处的非架构初始化指令中所指定的值,例如值400。每一次定序器128遇到循环指令并跳跃至目标指令(例如,图26A的地址1处的乘法累加指令或图28的地址1处的maxwacc指令),定序器128就会使循环计数器3804递减。一旦循环计数器3804到达零,则定序器128进入下一顺序非架构指令。在备选实施例中,首次遭遇到循环指令时,循环计数器3804加载循环指令中所指定的循环计数值,以省去经由非架构初始化指令对循环计数器3804进行初始化的需求。因此,循环计数器3804的值指示出还要执行非架构程序的循环体的次数。有利地,如以下针对图39所述,可由架构程序经由状态寄存器127的循环计数3914字段来获得循环计数器3804的值。这使得架构程序能够基于非架构程序的进度来决定相对于数据RAM 122和/或权重RAM 124在哪里进行数据读取/写入。在一个实施例中,定序器128包括三个额外的循环计数器以适应非架构程序内的嵌套循环,并且另外三个循环计数器的值也可经由状态寄存器127读取。循环指令中具有一位表示这四个循环计数器中哪一个用于即时的循环指令。

[0381] 定序器128还包括迭代计数器3806。迭代计数器3806与非架构指令相结合地使用,其中这些非架构指令诸如为图4、图9、图20和图26A的地址2处的乘法累加指令以及图28中地址2处的maxwacc指令等,这些指令在下文中被称为“执行”指令。在以上示例中,各执行指令分别指定迭代计数511、511、1023、2以及3。当定序器128遇到指定非零迭代计数的执行指令时,定序器128将迭代计数器3806加载该指定值。此外,定序器128产生适当的微操作3418

以控制图34的NPU 126的流水线阶段3401内的逻辑以用于执行,并且使迭代计数器3806递减。如果迭代计数器3806大于零,则定序器128再次产生适当的微操作3418以控制NPU 126内的逻辑并使迭代计数器3806递减。定序器128持续以这种方式操作,直到迭代计数器3806达到零为止。因此,迭代计数器3806的值表示还要执行非架构执行指令内所指定的运算(例如,累加器和数据/权重字的乘法累加、取最大值、求和)的次数。有利地,如以下针对图39所述,迭代计数器3806的值可以由架构程序经由状态寄存器127的迭代计数3916字段获得。这样使得架构程序能够基于非架构程序的进度来决定相对于数据RAM 122和/或权重RAM 124在哪里进行数据读取/写入。

[0382] 现参考图39,示出一框图,该框图示出NNU 121的控制和状态寄存器127的某些字段。如以上针对图26B所示,这些字段包括NPU 126执行非架构程序而最近写入的权重RAM行的地址2602、NPU 126执行非架构程序而最近读取的权重RAM行的地址2604、NPU 126执行非架构程序而最近写入的数据RAM行的地址2606、以及NPU 126执行非架构程序而最近读取的数据RAM行的地址2608。此外,这些字段包括NNU程序计数器3912、循环计数3914、以及迭代计数3916。如上所述,架构程序可(例如通过MFNN指令1500)将状态寄存器127读取至媒体寄存器118和/或通用寄存器116,状态寄存器127包括NNU程序计数器3912字段值、循环计数3914字段值和迭代计数3916字段值。程序计数器3912的值反映图38的程序计数器3802的值。循环计数3914的值反映循环计数器3804的值。迭代计数3916的值反映迭代计数器3806的值。在一个实施例中,定序器128在每次修改程序计数器3802、循环计数器3804或迭代计数器3806时,都更新程序计数器3912字段值、循环计数3914字段值与迭代计数3916字段值,使得当架构程序读取这些字段值时,这些字段值是当前值。在另一实施例中,当NNU 121执行用于读取状态寄存器127的架构指令时,NNU 121单纯获得程序计数器3802的值、循环计数器3804的值以及迭代计数器3806的值并将这些值提供回架构指令(例如提供至媒体寄存器118或通用寄存器116)。

[0383] 从以上可以发现,图39的状态寄存器127的字段值可被表征为非架构程序在由NNU执行期间的进度的信息。以上已经描述了非架构程序进度的某些方面,诸如程序计数器3802的值、循环计数器3804的值、迭代计数器3806的值、最近写入/读取2602/2604的权重RAM 124地址125,以及最近写入/读取2606/2608的数据RAM 122地址123。执行于处理器100上的架构程序可以从状态寄存器127读取图39的非架构程序进度值,并例如通过诸如比较指令和分支指令等的架构指令使用这些信息来作出决策。例如,尤其是针对大型数据集和/或不同非架构指令的重叠执行实例,架构程序决定相对于数据RAM 122和/或权重RAM 124在哪些行进行数据/权重的写入/读取,以控制相对于数据RAM 122或权重RAM 124的数据流入和流出。本文上下文中描述了利用架构程序进行决策的示例。

[0384] 例如,如以上针对图26A所述,架构程序将非架构程序配置为将卷积的结果写回数据RAM 122中位于卷积核2402上方(例如行8上方)的行,而当NNU 121通过使用最近写入的数据RAM 122行的地址2606来写入结果时,架构程序从数据RAM 122读取这些结果。

[0385] 再例如,如以上针对图26B所述,架构程序利用来自图38的状态寄存器127的字段的信息来确定非架构程序对图24的数据阵列2404以5个 512×1600 的数据块执行卷积的进度。架构程序将 2560×1600 数据阵列2404的第一个 512×1600 数据块写入权重RAM 124并开始非架构程序,其中该非架构程序的循环计数为1600且初始化的权重RAM 124的输出行为

0。当NNU 121执行非架构程序时,架构程序读取状态寄存器127以确定最近写入的权重RAM 124的行2602,使得该架构程序可以读取由非架构程序写入的有效卷积结果,并且在架构程序已经读取所述有效卷积结果后利用下一个 512×1600 数据块覆写上述有效卷积结果,使得在NNU 121完成对第一个 512×1600 数据块的非架构程序时,处理器100可以按需要立即更新非架构程序并再次开始非架构程序以处理下一个 512×1600 数据块。

[0386] 又例如,假定架构程序使NNU 121执行一系列传统神经网络乘法累加激活函数运算,其中,权重储存于权重RAM 124中并且结果被写回数据RAM 122。在这种情况下,非架构程序一旦读取了权重RAM 124的行,就不会再读取。因此,架构程序可被配置为一旦当前权重已被非架构程序读取/使用,就开始用针对非架构程序的下一次执行实例(例如下一个神经网络层)的新权重来覆写权重RAM 124中的权重。在这种情况下,架构程序读取状态寄存器127以获得最近读取的权重RAM 2604行的地址2604,从而决定新的权重集可以写入权重RAM 124中的位置。

[0387] 又例如,假定架构程序知道非架构程序包括具有大迭代计数的执行指令,诸如图20的地址2处的非架构乘法累加指令等。在这种情况下,架构程序可能需要知道迭代计数3916,以知道大致上还需要多少个时钟周期才能完成非架构指令,使得该架构程序可以决定接下来要采取两个或多个动作中的哪个。例如,如果时间很长,则架构程序可以将控制权让与另一个架构程序,诸如操作系统等。同样,假定架构程序知道非架构程序包括具有相当大循环计数的循环体,诸如图28的非架构程序等。在这种情况下,架构程序可能需要知道循环计数3914,以知道大致上还需要多少个时钟周期才能完成非架构程序,使得该架构程序可以决定接下来要采取两个或多个动作中的哪个。

[0388] 又例如,假定架构程序使NNU 121执行与针对图27和图28所述的将要池化的数据储存在权重RAM 124中并且将结果写回权重RAM 124的池化运算类似的池化运算。然而,与图27和图28的示例不同,假设结果被写回权重RAM 124的顶部的400行,例如行1600~1999。在这种情况下,一旦非架构程序读取了权重RAM 124中进行池化的四行,其就不会再次读取。因此,架构程序可被配置为一旦当前四行数据都已被非架构程序读取/使用,就开始用新的数据覆写权重RAM 124中的数据(例如,利用针对非架构程序的下一次执行实例的权重进行覆写,以例如对池化后数据执行传统乘法累加激活函数运算)。在这种情况下,架构程序读取状态寄存器127以获得最近读取的权重RAM行的地址2604,从而决定新的权重集可以写入权重RAM 124中的位置。

[0389] 循环(recurrent)神经网络加速

[0390] 传统的前馈神经网络不包括针对网络的先前输入的存储器。前馈神经网络通常被用于执行随时间输入网络的多个输入彼此独立且输出也是如此的任务。相比之下,循环神经网络(RNN)通常有助于执行随时间输入网络的顺序至关重要的任务。(顺序通常被称为时间步骤。)因此,RNN包括存储器或内部状态的概念,其基于网络响应于该顺序中的先前输入所进行的计算来保持信息,并且RNN的输出取决于内部状态以及下一个时间步骤的输入。语音识别、语言建模、文本产生、语言翻译、图像描述产生以及某些形式的手写识别都是RNN能良好执行的任务的示例。

[0391] 三个周知的示例是Elman RNN、Jordan RNN和长短期记忆(LSTM)网络。Elman RNN包含承接节点(context node),其记忆当前时间步骤中RNN的隐藏层的状态,此状态被提供

为针对下一个时间步骤的隐藏层的输入。除了承接节点记忆RNN的输出层而非隐藏层的状态以外, Jordan RNN是相同的。LSTM网络包括LSTM单元的LSTM层。每个LSTM单元具有当前时间步骤的当前状态和当前输出、以及新的或下一个时间步骤的新状态和新输出。LSTM单元包括输入门和输出门以及遗忘门,其中遗忘门使得单元遗忘其所记忆的状态。这三种类型的RNN在下文中更有详细的描述。

[0392] 在本发明的上下文中,针对诸如Elman或Jordan RNN等的循环神经网络(RNN), NNU执行时间步骤,其中针对时间步骤的每个实例, NNU取得一组输入层节点值并执行使其传播通过RNN以产生输出层节点值以及隐藏层节点值和承接层节点值所需的计算。因此,输入层节点值与使用这些输入层节点值来计算隐藏层、输出层以及承接层节点值的时间步骤相关联;并且隐藏层、输出层以及承接层节点值与产生这些节点值的时间步骤相关联。输入层节点值是由RNN建模的系统的取样值,例如图像、语音取样、金融市场数据的快照。针对LSTM网络, NNU执行时间步骤,其中针对时间步骤的每个实例,该NNU取得一组记忆单元输入值并执行产生记忆单元输出值(以及单元状态以及输入门、遗忘门和输出门的值)所需的计算,这也可以理解为是通过LSTM层单元来传播单元输入值。因此,单元输入值与使用这些单元输入值来计算单元状态以及输入门、遗忘门和输出门的值的时间步骤相关联;并且单元状态以及输入门、遗忘门和输出门的值与产生这些值的时间步骤相关联。

[0393] 承接层节点值(也称为状态节点)是神经网络的状态,并且该状态是基于与先前时间步骤相关联的输入层节点值,而不仅是基于与当前时间步骤相关联的输入层节点值。NNU在时间步骤所执行的计算(例如,针对Elman或Jordan RNN的隐藏层节点值计算)是先前时间步骤产生的承接层节点值的函数。因此,时间步骤开始时的网络状态(承接节点值)影响该时间步骤期间所产生的输出层节点值。此外,时间步骤结束时的网络状态受到该时间步骤的输入节点值和该时间步骤开始时的网络状态这两者影响。同样,针对LSTM单元,单元状态值是基于与先前时间步骤相关联的记忆单元输入值,而不仅是基于与当前时间步骤相关联的记忆单元输入值。由于NNU在时间步骤所执行的计算(例如下一个单元状态)是先前时间步骤产生的单元状态值的函数,所以时间步骤开始时的网络状态(单元状态值)影响该时间步骤期间所产生的单元输出值,并且该时间步骤结束时的网络状态受到该时间步骤的单元输入值和先前网络状态这两者影响。

[0394] 现参考图40,示出一框图,该框图示出Elman RNN的示例。图40的Elman RNN包括:标示为D0、D1至Dn的输入层节点或神经元,共同称为(多个)输入层节点D并且个别地通称为输入层节点D;标示为Z0、Z1至Zn的隐藏层节点/神经元,共同称为(多个)隐藏层节点Z并且个别地通称为隐藏层节点Z;标示为Y0、Y1至Yn的输出层节点/神经元,共同称为(多个)输出层节点Y并且个别地通称为输出层节点Y;以及标示为C0、C1至Cn的承接层节点/神经元,共同称为(多个)承接层节点C并且个别地地通称为承接层节点C。在图40的示例性Elman RNN中,各隐藏层节点Z具有与各输入层节点D的输出连接的输入,并具有与各承接层节点C的输出连接的输入;各输出层节点Y具有与各隐藏层节点Z的输出连接的输入;而各承接层节点C具有与相应隐藏层节点Z的输出连接的输入。

[0395] 在许多方面, Elman RNN的操作与传统的前馈人工神经网络类似。也就是说,对于给定节点而言,存在与此节点的各个输入连接相关联的权重;节点在输入连接处接收到的值与其关联的权重相乘以产生乘积;此节点将与所有输入连接相关联的乘积相加以产生总

和(该总和可能还包括偏离项);通常,对该总和执行激活函数以产生此节点的输出值,该输出值有时称为节点的激活值。对于传统的前馈网络而言,数据总是沿着从输入层至输出层的一个方向流动。也就是说,输入层向隐藏层(通常有多个隐藏层)提供值,该隐藏层产生被提供至输出层的输出值,而该输出层产生可被捕获到的输出。

[0396] 然而,相较于传统的前馈网络,Elman RNN还包括一些反馈连接,即图40的从隐藏层节点Z至承接层节点C的连接。Elman RNN进行工作以使得:当输入层节点D在新的时间步骤中向隐藏层节点Z提供输入值时,承接节点C向隐藏层Z提供作为隐藏层节点Z响应于(相对于当前时间步骤的)先前输入的输出值的值。从这个意义上来说,Elman RNN的承接节点C是基于先前时间步骤的输入值的存储器。现在针对图41与图42来说明NNU 121执行与图40的Elman RNN相关联的计算的实施例的操作。

[0397] 为了本发明的目的,Elman RNN是至少包含输入节点层、隐藏节点层,输出节点层和承接节点层的循环神经网络。对于给定时间步骤,承接节点层储存隐藏节点层在先前时间步骤产生且由隐藏节点层反馈至承接节点层的结果。反馈至承接层的结果可以是激活函数的结果,或者可以是隐藏节点层在不执行激活函数的情况下所执行的累加的结果。

[0398] 现参考图41,示出一框图,该框图示出当NNU 121执行与图40的Elman RNN相关联的计算时,该NNU 121的数据RAM 122和权重RAM 124内的数据布局的示例。在图41的示例中,假定图40的Elman RNN具有512个输入节点D、512个隐藏节点Z、512个承接节点C、以及512个输出节点Y。此外,假定Elman RNN完全连接,即全部512个输入节点D都作为输入连接至各隐藏节点Z,全部512个承接节点C都作为输入连接至各隐藏节点Z,并且全部512个隐藏节点Z都作为输入连接至各输出节点Y。此外,NNU 121被配置为例如采用宽配置的512个NPU 126或神经元。最后,假定与从承接节点C至隐藏节点Z的连接相关联的权重的值均为1,因此无需储存这些为1的权重值。

[0399] 如图所示,权重RAM 124的低位的512行(行0至511)保持与输入节点D和隐藏节点Z之间的连接相关联的权重值。更具体地,如图所示,行0保持与从输入节点D0至隐藏节点Z的输入连接相关联的权重,即字0保持与输入节点D0和隐藏节点Z0之间的连接相关联的权重,字1保持与输入节点D0和隐藏节点Z1之间的连接相关联的权重,字2保持与输入节点D0和隐藏节点Z2之间的连接相关联的权重,依此类推,字511保持与输入节点D0和隐藏节点Z511之间的连接相关联的权重;行1保持与从输入节点D1至隐藏节点Z的输入连接相关联的权重,即字0保持与输入节点D1和隐藏节点Z0之间的连接相关联的权重,字1保持与输入节点D1和隐藏节点Z1之间的连接相关联的权重,字2保持与输入节点D1和隐藏节点Z2之间的连接相关联的权重,依此类推,字511保持与输入节点D1和隐藏节点Z511之间的连接相关联的权重;直至行511保持与从输入节点D511至隐藏节点Z的输入连接相关联的权重,即字0保持与输入节点D511和隐藏节点Z0之间的连接相关联的权重,字1保持与输入节点D511和隐藏节点Z1之间的连接相关联的权重,字2保持与输入节点D511和隐藏节点Z2之间的连接相关联的权重,依此类推,字511保持与输入节点D511和隐藏节点Z511之间的连接相关联的权重。这与以上针对图4至图6A所述的布局和使用类似。

[0400] 如图所示,权重RAM 124的接下来的512个行(行512至1023)以相同的方式保持与隐藏节点Z和输出节点Y之间的连接相关联的权重。

[0401] 数据RAM 122保持一系列时间步骤的Elman RNN节点值。更具体地,具有三行的三

元组 (triplet) 保持给定时间步骤的节点值。如图所示,在数据RAM 122具有64行的实施例中,该数据RAM 122可以保持20个不同时间步骤的节点值。在图41的示例中,行0至2保持时间步骤0的节点值,行3至5保持时间步骤1的节点值,依此类推,行57至59保持时间步骤19的节点值。三元组的第一行保持时间步骤的输入节点D的值。三元组的第二行保持时间步骤的隐藏节点Z的值。三元组的第三行保持时间步骤的输出节点Y的值。如图所示,数据RAM 122的各列保持其相应神经元或NPU 126的节点值。也就是说,如以下针对图42更详细所述,列0保持与节点D0、Z0与Y0相关联的节点值,其中这些节点值的计算由NPU 0执行;列1保持与节点D1、Z1与Y1相关联的节点值,其中这些节点值的计算由NPU 1执行;依此类推,列511保持与节点D511、Z511与Y511相关联的节点值,这些节点值的计算由NPU 511执行。

[0402] 如图41所示,与给定时间步骤相关联的三元组的第二行的隐藏节点Z的值是下一个时间步骤的承接节点C的值。也就是说,NPU 126在时间步骤内计算并写入的Z值成为该NPU 126在下一个时间步骤内计算Z值所使用的C值(该计算还使用该下一个时间步骤的输入节点D的值)。承接节点C的初始值(即,在时间步骤0计算行1中的Z值所使用的C值)假定为零。以下针对图42的非架构程序更详细地对此进行说明。

[0403] 优选地,输入节点D的值(图41的示例中的行0、3依此类推至行57的值)由处理器100上运行的架构程序经由MTNN指令1400写入/填入数据RAM 122,并且由NNU 121上运行的非架构程序(诸如图42的非架构程序等)读取/使用。相反,隐藏节点Z/输出节点Y的值(图41的示例中的行1和2、4和5依此类推至行58和59的值)由NNU 121上运行的非架构程序写入/填入数据RAM 122,并且由处理器100上运行的架构程序经由MFNN指令1500读取/使用。图41的示例假定架构程序执行以下步骤:(1)将20个不同时间步骤的输入节点D的值填入数据RAM 122(行0、3依此类推至行57);(2)开始图42的非架构程序;(3)检测非架构程序已经完成;(4)从数据RAM 122读出输出节点Y的值(行2、5依此类推至行59);以及(5)按需要重复步骤(1)至(4)若干次直到完成任务(例如完成用于对移动电话用户的话语进行识别的计算)。

[0404] 在备选方式中,架构程序执行以下步骤:(1)将单个时间步骤的输入节点D的值填入数据RAM 122(例如,行0);(2)开始非架构程序(图42的修改版本,其不需要循环并且访问数据RAM 122的行的单个三元组);(3)检测非架构程序已完成;(4)从数据RAM 122读出输出节点Y的值(例如,行2);以及(5)按需要重复步骤(1)至(4)若干次直到完成任务。优选这两种方式中的哪一种取决于RNN的输入值的取样方式。例如,如果任务容许在多个时间步骤(例如量级为20个时间步骤)对输入进行取样并执行计算,则可以优选第一种方式,因为该方式有可能带来更多的计算资源效率和/或更高的性能,然而,如果该任务只容许在单个时间步骤进行取样,则可能需要第二种方式。

[0405] 设想了与第二种方式类似的第三实施例,但是在该第三实施例中,不同于使用数据RAM 122的行的单个三元组,与第一种方式类似地,非架构程序使用行的多个三元组,即针对各时间步骤使用不同的三元组。在该第三实施例中,架构程序优选在步骤(2)之前包含如下的步骤,在该步骤中,架构程序在开始非架构程序之前对该非架构程序进行更新,例如将地址1处的指令内的数据RAM 122行更新为指向下一三元组。

[0406] 现参考图42,示出一表格,该表格示出储存于NNU 121的程序存储器129中并由NNU 121执行以实现Elman RNN的程序,并且使用根据图41的布置的数据与权重。以上详细说明了图42(以及图45、图48、图51、图54和图57)的非架构程序中的一些指令(例如乘法累加

(MULT-ACCUM) 指令、循环 (LOOP) 指令、初始化 (INITIALIZE) 指令), 并且除非另有说明, 以下描述中假定了这些描述。

[0407] 图42的示例性程序包含位于地址0至12处的13个非架构指令。地址0处的指令 (INITIALIZE NPU, LOOPCNT=20) 清除累加器202并且将循环计数器3804初始化为值20, 以使循环体 (地址4至11的指令) 执行20次。优选地, 初始化指令也会将NNU 121置为宽配置, 使得NNU 121被配置为512个NPU 126。根据以下说明可以发现, 在地址1至3以及地址7至11的指令执行期间, 512个NPU 126对应于并且用作512个隐藏层节点Z, 而在地址4至6的指令执行期间, 512个NPU 126对应于并且用作512个输出层节点Y。

[0408] 地址1至3处的指令在程序循环体之外并且只会执行一次。这些指令计算隐藏层节点Z的初始值并将其写入数据RAM 122的行1以供地址4至6处的指令的第一次执行实例使用, 从而计算出第一时间步骤 (时间步骤0) 的输出层节点Y。此外, 由地址1至3处的指令计算并写入数据RAM 122的行1的隐藏层节点Z的值变成承接层节点C的值以供地址7和8处的指令的第一次执行实例使用, 从而计算供第二时间步骤 (时间步骤1) 用的隐藏层节点Z的值。

[0409] 在地址1和2处的指令的执行期间, 512个NPU 126中的各NPU 126执行512次乘法运算以将数据RAM 122的行0的512个输入节点D的值乘以来自权重RAM 124的行0至511的NPU 126的相应列的权重, 从而产生累加至相应NPU 126的累加器202中的512个乘积。在地址3处的指令的执行期间, 这512个NPU 126的512个累加器202值通过并写入数据RAM 122的行1。也就是说, 地址3的输出指令将512个NPU 126中的各NPU 126的累加器202的值 (即初始隐藏层Z的值) 写入数据RAM 122的行1, 然后清除该累加器202。

[0410] 图42的非架构程序的地址1至2处的指令所执行的运算在许多方面与图4的非架构指令的地址1至2处的指令所执行的运算类似。更具体地, 地址1处的指令 (MULT-ACCUM DR ROW 0) 指示512个NPU 126中的各NPU 126将数据RAM 122的行0的相应字读入其复用寄存器208, 将权重RAM 124的行0的相应字读入其复用寄存器705, 将数据字和权重字相乘以产生乘积并将该乘积加入累加器202。地址2处的指令 (MULT-ACCUM ROTATE, WR ROW+1, COUNT=511) 指示512个NPU 126中的各NPU 126将来自相邻NPU 126的字轮转入其复用寄存器208 (利用由NNU 121的512个复用寄存器208的整体操作形成的512字轮转器, 其中地址1处的指令刚刚将数据RAM 122的行读入这些复用寄存器208), 将权重RAM 124的下一行的相应字读入其复用寄存器705, 将数据字和权重字相乘以产生乘积并将该乘积加入累加器202, 并且执行该运算511次。

[0411] 此外, 图42的地址3的单个非架构输出指令 (OUTPUT PASSTHRU, DR OUT ROW 1, CLR ACC) 将图4的地址3处的激活函数指令与地址4处的写入输出指令的操作合并 (尽管在图42的程序中使累加器202的值通过, 而在图4的程序中, 对累加器202的值执行激活函数)。也就是说, 在图42的程序中, 在输出指令中 (同样在地址6和11的输出指令中) 指定对累加器202的值执行的激活函数 (如果有的话), 而非如图4的程序那样在不同的非架构激活函数指令中指定。设想了图4 (以及图20、图26A和图28) 的非架构程序的备选实施例, 其中将 (例如, 图4的地址3和4的) 激活函数指令和写入输出指令的操作组合为如图42所示的单个非架构输出指令。图42的示例假定隐藏层 (Z) 的节点不对累加器的值执行激活函数。然而, 设想了其它实施例, 其中隐藏层 (Z) 对累加器的值执行激活函数, 在这种情况下, 地址3和11处的指令进行运算, 如S型、双曲正切、校正等。

[0412] 相较于地址1至3处的指令的单个执行实例,地址4至11处的指令位于程序循环体内并且被执行循环计数所指示的次数(例如20次)。地址7至11处的指令的前十九次执行实例计算隐藏层节点Z的值并将其写入数据RAM 122以供地址4至6处的指令的第二至二十次执行实例使用,从而计算剩余时间步骤(时间步骤1至19)的输出层节点Y。(地址7至11处的指令的最后/第二十次执行实例计算隐藏层节点Z的值并将其写入数据RAM 122的行61,但是不使用这些值。)

[0413] 在地址4和5处的指令(MULT-ACCUM DR ROW+1,WR ROW 512和MULT-ACCUMROTATE,WR ROW+1,COUNT=511)的第一次执行实例期间(时间步骤0),512个NPU 126中的各NPU 126执行512个乘法运算以将数据RAM 122的行1的512个隐藏节点Z的值(这些值由地址1至3的指令的单个执行实例产生并写入)乘以来自权重RAM 124的行512至1023的NPU 126的相应列的权重,以产生512个乘积,其中这512个乘积累加至相应NPU 126的累加器202中。在地址6处的指令(OUTPUT ACTIVATION FUNCTION,DR OUT ROW+1,CLR ACC)的第一次执行实例期间,对这512个累加值执行激活函数(例如S型、双曲正切、校正)以计算输出节点Y层的值,并且将结果写入数据RAM 122的行2。

[0414] 在地址4和5处的指令的第二次执行实例期间(时间步骤1),512个NPU 126中的各NPU 126执行512个乘法运算,以将数据RAM 122的行4中的512个隐藏节点Z的值(这些值由地址7至11的指令的第一次执行实例产生并写入)乘以来自权重RAM 124的行512至1023的NPU 126的相应列的权重,以产生512个乘积,其中这512个乘积累加至相应NPU 126的累加器202中,并且在地址6处的指令的第二次执行实例期间,对这512个累加值执行激活函数以计算输出节点Y层的值,其中这些值写入数据RAM 122的行5;在地址4和5处的指令的第三次执行实例期间(时间步骤2),512个NPU 126中的各NPU 126执行512个乘法运算以将数据RAM 122的行7中的512个隐藏节点Z的值(这些值通过地址7至11的指令的第二次执行实例产生并写入)乘以来自权重RAM 124的行512至1023的NPU 126的相应列的权重,以产生512个乘积,其中这512个乘积累加至相应NPU 126的累加器202中,并且在地址6的指令的第三次执行实例期间,对这512个累加值执行激活函数以计算输出节点Y层的值,并且将结果写入数据RAM 122的行8;依此类推,在地址4和5处的指令的第二十次执行实例期间(时间步骤19),512个NPU 126中的各NPU 126执行512个乘法运算以将数据RAM 122的行58的512个隐藏节点Z的值(这些值通过地址7至11的指令的第十九次执行实例产生并写入)乘以来自权重RAM 124的行512至1023的NPU 126的相应列的权重,以产生512个乘积,其中这512个乘积累加至相应NPU 126的累加器202中,并且在地址6处的指令的第二十次执行实例期间,对这512个累加值执行激活函数以计算输出节点Y层的值,并将结果写入数据RAM 122的行59。

[0415] 在地址7与8的指令的第一次执行实例期间,这512个NPU 126中的各NPU 126将数据RAM 122的行1的512个承接节点C的值累加至累加器202,这些值通过地址1至3的指令的单个执行实例产生。更具体地,地址7的指令(ADD_D_ACC DR ROW+0)指示这512个NPU 126各自将数据RAM 122的当前行(在第一次执行实例期间为行0)的相应字读入复用寄存器208,并将该字加入累加器202。地址8的指令(ADD_D_ACC ROTATE,COUNT=511)指示这512个NPU 126各自将来自相邻NPU 126的字轮转入复用寄存器208中(使用通过NNU 121的512个复用寄存器208的整体操作所形成的512个字的轮转器,其中地址7处的指令刚刚将数据RAM 122的行读入这些复用寄存器208),将该字加入累加器202,并且执行该运算511次。

[0416] 在地址7和8处的指令的第二次执行实例期间,512个NPU 126中的各NPU 126将数据RAM 122的行4的512个承接节点C的值累加至累加器202,其中这些值通过地址9至11的指令的第一次执行实例所产生并写入;在地址7和8处的指令的第三次执行实例期间,512个NPU 126中的各NPU 126将数据RAM 122的行7的512个承接节点C的值累加至累加器202,其中这些值通过地址9至11的指令的第二次执行实例所产生并写入;依此类推,在地址7和8处的指令的第二十次执行实例期间,512个NPU 126中的各NPU 126将数据RAM 122的行58的512个承接节点C的值累加至累加器202,其中这些值通过地址9至11的指令的第十九次执行实例所产生并写入。

[0417] 如上所述,图42的示例假定与从承接节点C至隐藏层节点Z的连接相关联的权重都具有为1的值。然而,在备选实施例中,在这些连接具有非零权重值的Elman RNN中,这些权重在图42的程序执行前放置于权重RAM 124内(例如行1024至1535中),并且地址7处的程序指令为MULT-ACCUM DR ROW+0、WR ROW 1024,而地址8处的程序指令为MULT-ACCUM ROTATE、WR ROW+1、COUNT=511。优选地,地址8处的指令不访问权重RAM 124,而是将通过地址7处的指令从权重RAM 124读入复用寄存器705的值轮转。在执行地址8处的指令的511个时钟周期期间不对权重RAM 124进行访问可以是有利的,因为其保留更多带宽供架构程序访问权重RAM 124使用。

[0418] 在地址9和10处的指令(MULT-ACCUM DR ROW+2、WR ROW 0和MULT-ACCUM ROTATE、WR ROW+1、COUNT=511)的第一次执行实例期间(时间步骤1),512个NPU 126中的各NPU 126执行512个乘法运算以将数据RAM 122的行3中的512个输入节点D的值乘以来自权重RAM 124的行0至511的NPU 126的相应列的权重,从而产生512个乘积,其中这512个乘积连同地址7和8处的指令所执行的对512个承接C节点值的累加一起累加至相应NPU 126的累加器202中以计算隐藏节点Z层的值,并且在地址11处的指令(OUTPUT PASSTHRU、DR OUT ROW+2、CLR ACC)的第一次执行期间,使512个NPU 126的512个累加器202的值通过并写入数据RAM 122的行4,并且累加器202被清零;在地址9和10处的指令的第二次执行实例期间(时间步骤2),512个NPU 126中的各NPU 126执行512个乘法运算以将数据RAM 122的行6的512个输入节点D的值乘以来自权重RAM 124的行0至511的NPU 126的相应列的权重,从而产生512个乘积,其中这512个乘积连同地址7和8处的指令所执行的对512个承接C节点值的累加一起累加至相应NPU 126的累加器202中,以计算隐藏节点Z层的值,并且在地址11处的指令的第二次执行期间,使这512个NPU 126的512个累加器202的值通过并写入数据RAM 122的行7,并且累加器202被清零;依此类推,在地址9和10处的指令的第十九次执行实例期间(时间步骤19),512个NPU 126中的各NPU 126执行512个乘法运算,以将数据RAM 122的行57的512个输入节点D的值乘以来自权重RAM 124的行0至511的NPU 126的相应列的权重,从而产生512个乘积,其中这512个乘积连同地址7和8处的指令所执行的对512个承接C节点值的累加一起累加至相应NPU 126的累加器202中,以计算隐藏节点Z层的值,并且在地址11处的指令的第十九次执行期间,使这512个NPU 126的512个累加器202的值通过并写入数据RAM 122的行58,并且累加器202被清零。如上所述,在地址9和10处的指令的第二十次执行实例期间所产生并被写入数据RAM 122的行61的隐藏节点Z层的值并不会被使用。

[0419] 地址12处的指令(LOOP 4)使循环计数器3804递减并且在新的循环计数器3804的值大于零的情况下循环回到地址4处的指令。

[0420] 现参考图43,示出一框图,该框图示出Jordan RNN的示例。图43的Jordan RNN在许多方面与图40的Elman RNN类似,即Jordan RNN包括输入层节点/神经元D、隐藏层节点/神经元Z、输出层节点/神经元Y、以及承接层节点/神经元C。然而,在图43的Jordan RNN中,承接层节点C的输入连接是从相应的输出层节点Y的输出反馈回来的,而非如图40的Elman RNN那样来自隐藏层节点Z的输出。

[0421] 为了本发明的目的,Jordan RNN是至少包含输入节点层、隐藏节点层、输出节点层以及承接节点层的循环神经网络。在给定时间步骤开始时,承接节点层包含输出节点层在前一个时间步骤产生并且由输出节点层反馈至承接节点层的结果。反馈至承接层的结果可以是激活函数的结果,或者其可以是输出节点层在不执行激活函数的情况下所执行的累加的结果。

[0422] 现参考图44,示出一框图,该框图示出当NNU 121执行与图43的Jordan RNN相关联的计算时,该NNU 121的数据RAM 122和权重RAM 124内的数据布局的示例。在图44的示例中,假定图43的Jordan RNN具有512个输入节点D、512个隐藏节点Z、512个承接节点C、以及512个输出节点Y。此外,假定该Jordan RNN为完全连接,即全部512个输入节点D都作为输入连接至各隐藏节点Z,全部512个承接节点C都作为输入连接至各隐藏节点Z,并且全部512个隐藏节点Z都作为输入连接至各输出节点Y。在图44的示例性Jordan RNN中,尽管对累加器202的值应用激活函数以产生输出层节点Y的值,但是假定使应用激活函数之前的累加器202值、而非实际的输出层节点Y值通过至承接层节点C。此外,NNU 121被配置为例如采用宽配置的512个NPU 126或神经元。最后,假定与从承接节点C至隐藏节点Z的连接相关联的权重的值均为1,因此无需储存这些为1的权重值。

[0423] 如图41的示例,权重RAM 124的低位的512行(行0至511)保持与输入节点D和隐藏节点Z之间的连接相关联的权重值,而权重RAM 124的接下来的512行(行512至1023)保持与隐藏节点Z和输出节点Y之间的连接相关联的权重值。

[0424] 与图41的示例类似,数据RAM 122保持一系列时间步骤的Jordan RNN节点值;然而,对于图44的示例而言,四行的四元组保持给定时间步骤的节点值。在数据RAM 122具有64行的实施例中,如图所示,数据RAM 122可以保持15个不同时间步骤的节点值。在图44的示例中,行0至3保持时间步骤0的节点值,行4至7保持时间步骤1的节点值,依此类推,行60至63保持时间步骤15的节点值。四元组中的第一行保持时间步骤的输入节点D的值。四元组中的第二行保持时间步骤的隐藏节点Z的值。四元组中的第三行保持时间步骤的承接节点C的值。四元组中的第四行保持时间步骤的输出节点Y的值。如图所示,数据RAM 122中的各列保持其相应的神经元或NPU 126的节点值。也就是说,如以下针对图44更详细所述,列0保持与节点D0、Z0、C0和Y0相关联的节点值,其计算由NPU 0执行;列1保持与节点D1、Z1、C1和Y1相关联的节点值,其计算由NPU 1执行;依此类推,列511保持与节点D511、Z511、C511和Y511相关联的节点值,其计算由NPU 511执行。

[0425] 图44所示的给定时间步骤的承接节点C的值在该时间步骤内产生并用作下一个时间步骤的输入。也就是说,NPU 126在该时间步骤内计算并写入的C值成为该NPU 126在下一个时间步骤计算Z值所使用的C值(该计算还使用该下一个时间步骤的输入节点D的值)。承接节点C的初始值(即,在时间步骤0计算行1中的Z值所使用的C值)假定为零。这在以下针对图45的非架构程序有更详细说明。

[0426] 如以上针对图41所述,优选地,输入节点D的值(图44的示例中的行0、4依此类推至行60的值)由运行于处理器100上的架构程序经由MTNN指令1400写入/填入数据RAM 122,并且由运行于NNU 121上的非架构程序(诸如图45的非架构程序等)读取/使用。相反,隐藏节点Z/承接节点C/输出节点Y的值(图44的示例中的行1/2/3、4/5/6依此类推至行60/61/62的值)由运行于NNU 121上的非架构程序写入/填入数据RAM 122,并且由运行于处理器100上的架构程序经由MFNN指令1500读取/使用。图44的示例假定架构程序执行以下步骤:(1)将15个不同时间步骤的输入节点D的值填入数据RAM 122(行0、4依此类推至行60);(2)开始图45的非架构程序;(3)检测非架构程序已完成;(4)从数据RAM 122读出输出节点Y的值(行3、7依此类推至行63);以及(5)按需要重复步骤(1)至(4)若干次以完成任务(例如用于执行对移动电话用户的话语进行识别的计算)。

[0427] 在备选方式中,架构程序执行以下步骤:(1)将单个时间步骤的输入节点D的值填入数据RAM 122(例如,行0);(2)开始非架构程序(图45中不需要循环并且访问数据RAM 122的单个四元组的修改版本);(3)检测非架构程序已完成;(4)从数据RAM 122读出输出节点Y的值(例如,行3);以及(5)按需要重复步骤(1)至(4)若干次以完成任务。优选这两种方式中的哪一种取决于RNN的输入值的取样方式。例如,如果任务容许在多个时间步骤(例如量级为15个时间步骤)对输入进行取样并执行计算,则可以优选第一种方式,因为该方式有可能带来更多计算资源效率和/或更高的性能,然而,如果该任务只容许在单个时间步骤进行取样,则可能需要第二种方式。

[0428] 设想了与第二种方式类似的第三实施例,但是在该第三实施例中,不同于使用数据RAM 122的行的单个四元组,与第一种方式类似,非架构程序使用行的多个四元组,即针对各时间步骤使用不同的四元组。在该第三实施例中,架构程序优选在步骤(2)之前包含如下的步骤,在该步骤中,架构程序在开始非架构程序之前对该非架构程序进行更新,例如将地址1处的指令内的数据RAM 122行更新为指向下一四元组。

[0429] 现参考图45,示出一表格,该表格示出储存于NNU 121的程序存储器129中并由NNU 121执行以实现Jordan RNN的程序、并且使用根据图44的布置的数据和权重。图45的非架构程序在许多方面与图42的非架构程序类似,但是二者的差异在下文中进行说明。

[0430] 图45的示例性程序包括位于地址0至13的14个非架构指令。地址0处的指令是初始化指令,该初始化指令清除累加器202并将循环计数器3804初始化至值15,以使循环体(地址4至12的指令)执行15次。优选地,初始化指令也将NNU 121置为宽配置,使得NNU 121被配置为512个NPU 126。可以发现,在地址1至3以及地址8至12的指令执行期间,这512个NPU 126对应于并且用作512个隐藏层节点Z,并且在地址4、5和7的指令执行期间,这512个NPU 126对应于并且用作512个输出层节点Y。

[0431] 地址1至5以及地址7处的指令与图42的地址1至6处的指令相同,并执行相同的函数。地址1至3处的指令计算隐藏层节点Z的初始值并将其写入数据RAM 122的行1以供地址4、5和7处的指令的第一次执行实例所使用,从而计算出第一时间步骤(时间步骤0)的输出层节点Y。

[0432] 在地址6处的输出指令的第一次执行实例期间,通过地址4和5处的指令累加的512个累加器202的值(这些值随后被地址7处的输出指令使用以计算并写入输出节点Y层的值)通过并写入数据RAM 122的行2,其中这些值是在第一时间步骤(时间步骤0)中产生并在第

二时间步骤(时间步骤1)期间使用的承接层节点C的值;在地址6处的输出指令的第二次执行实例期间,通过地址4和5处的指令累加的512个累加器202值(这些值随后被地址7处的输出指令使用以计算并写入输出节点Y层的值)通过并写入数据RAM 122的行6,其中这些值是在第二时间步骤(时间步骤1)中产生并在第三时间步骤(时间步骤2)期间使用的承接层节点C的值;依此类推,在地址6处的输出指令的第十五次执行实例期间,由地址4和5处的指令累加的512个累加器202值(这些值随后被地址7处的输出指令使用以计算并写入输出节点Y层的值)通过并写入数据RAM 122的行58,其中这些值是在第十五时间步骤(时间步骤14)中产生(并由地址8处的指令读取,但不会被使用)的承接层节点C的值。

[0433] 除一个差异外,地址8至12处的指令与图42的地址7至11处的指令相同并执行相同的功能。该差异即图45中地址8处的指令使数据RAM 122的行增加1(ADD_D_ACC DR ROW+1),而图42中地址7处的指令使数据RAM 122的行增加0(ADD_D_ACC DR ROW+0)。这是由于数据RAM 122内的数据布局的不同,具体地,图44中的布局包括四元组中针对承接层节点C值的单独行(例如,行2、6、10等),而图41中的布局不包括三元组中针对承接层节点C值的单独行,相反,承接层节点C的值与隐藏层节点Z的值共享行(例如,行1、4、7等)。地址8至12处的指令的十五次执行实例计算隐藏层节点Z的值并将其写入数据RAM 122(写入行5、9、13依此类推直到行57)以供地址4、5和7处的指令的第二至第十六次执行实例使用,从而计算第二至第十五时间步骤(时间步骤1至14)的输出层节点Y。(地址8至12处的指令的最后一次/第十五次执行计算隐藏层节点Z的值并将其写入数据RAM 122的行61,但是不使用这些值。)

[0434] 地址13处的循环指令使循环计数器3804递减并且在新的循环计数器3804的值大于零的情况下循环回到地址4处的指令。

[0435] 在备选实施例中,Jordan RNN被设计成使得承接节点C保持输出节点Y的激活函数值,即已经执行激活函数的累加值。在这种实施例中,因为输出节点Y的值与承接节点C的值相同,因此地址6处的非架构指令并不包含于非架构程序内。因此,可以消耗数据RAM 122的更少行。更精确地说,图44中保持承接节点C值的各行(例如,2、6、59)都不存在。此外,各时间步骤仅需要数据RAM 122的三行,使得可以适应20个时间步骤,而非15个,并且图45的非架构程序的指令的地址也会进行适当的修改。

[0436] LSTM单元

[0437] 循环神经网络中所使用的长短期记忆(LSTM)单元的概念是久为人知的。例如见“Long Short-Term Memory”(Sepp Hochreiter and Jürgen Schmidhuber, Neural Computation, 1997年11月15日, Vol. 9, No. 8, 1735-1780页);“Learning to Forget: Continual Prediction with LSTM”(Felix A. Gers, Jürgen Schmidhuber以及Fred Cummins, Neural Computation, 2000年10月, Vol. 12, No. 10, 2451-2471页);这两者都可以从MIT Press Journals获得。LSTM单元能够以各种形式构建。以下针对图46所述的LSTM单元4600是根据<http://deeplearning.net/tutorial/lstm.html>中标题为“LSTM Networks for Sentiment Analysis”上得到的教程所描述的LSTM单元来建模的。LSTM单元4600被提供为一般性地说明本文所述的NNU 121的实施例高效执行与LSTM相关联的计算的能力的方式。应当理解,NNU 121(包括针对图49所述的实施例)可以用于高效地执行与除图46所述的LSTM单元以外的其它LSTM单元相关联的计算。

[0438] 优选地,NNU 121可用于执行针对包括连接到其它层的LSTM单元层的循环神经网络

络的计算。例如,在LSTM教程中,网络包括用于接收LSTM层的LSTM单元的输出生(H)的均值池化层以及用于接收均值池化层的输出的逻辑回归层。

[0439] 现参考图46,示出一框图,该框图示出LSTM单元4600的实施例。

[0440] 如图所示,LSTM单元4600包括记忆单元输入(X)、记忆单元输出(H)、输入门(I)、输出门(O)、遗忘门(F)、单元状态(C)以及候选单元状态(C')。输入门(I)将记忆单元输入(X)门控至单元状态(C),而输出门(O)将单元状态(C)门控至记忆单元输出(H)。单元状态(C)被反馈为时间步骤的候选单元状态(C')。遗忘门(F)对候选单元状态(C')进行门控,其中该候选单元状态(C')被反馈并变成下一个时间步骤的单元状态(C)。

[0441] 在图46的实施例中,使用以下等式来计算上述各个值:

[0442] (1) $I = \text{SIGMOID}(W_i * X + U_i * H + B_i)$

[0443] (2) $F = \text{SIGMOID}(W_f * X + U_f * H + B_f)$

[0444] (3) $C' = \text{TANH}(W_c * X + U_c * H + B_c)$

[0445] (4) $C = I * C' + F * C$

[0446] (5) $O = \text{SIGMOID}(W_o * X + U_o * H + B_o)$

[0447] (6) $H = O * \text{TANH}(C)$

[0448] W_i 与 U_i 是与输入门(I)相关联的权重值,并且 B_i 是与输入门(I)相关联的偏置值。 W_f 与 U_f 是与遗忘门(F)相关联的权重值,并且 B_f 是与遗忘门(F)相关联的偏置值。 W_o 与 U_o 是与输出门(O)相关联的权重值,并且 B_o 是与输出门(O)相关联的偏置值。如图所示,等式(1)、(2)与(5)分别计算输入门(I)、遗忘门(F)和输出门(O)。等式(3)计算候选单元状态(C'),而等式(4)使用当前单元状态(C)、即使用当前时间步骤的单元状态(C)作为输入计算候选单元状态(C')。等式(6)计算单元输出(H)。设想了LSTM单元的其它实施例,其中这些实施例采用针对输入门、遗忘门、输出门、候选单元状态、单元状态以及单元输出的不同计算。

[0449] 为了说明本发明,LSTM单元包括记忆单元输入、记忆单元输出、单元状态、候选单元状态、输入门、输出门和遗忘门。针对各时间步骤,输入门、输出门、遗忘门和候选单元状态是当前时间步骤的记忆单元输入和先前时间步骤的记忆单元输出以及关联权重的函数。该时间步骤的单元状态是先前时间步骤的单元状态、候选单元状态、输入门和遗忘门的函数。从这个意义上说,单元状态被反馈并用于计算下一个时间步骤的单元状态。该时间步骤的记忆单元输出是该时间步骤计算出的单元状态和输出门的函数。LSTM网络是包括LSTM单元层的神经网络。

[0450] 现参考图47,示出一框图,该框图示出在NNU 121执行与图46的128个LSTM单元4600的层相关联的计算时、该NNU 121的数据RAM 122和权重RAM 124内的数据布局的示例。在图47的示例中,NNU 121被配置为例如采用宽配置的512个NPU 126或神经元,然而,只有128个NPU 126(例如,NPU 0至127)所产生的值会被使用,这是因为在该示例中,LSTM层中只有128个LSTM单元4600。

[0451] 如图所示,权重RAM 124保持NNU 121的相应NPU 0至127的权重值、偏置值和中间值。权重RAM 124的列0至127保持NNU 121的相应NPU 0至127的权重值、偏置值和中间值。行0至14各自保持上述等式(1)至(6)的以下相应值中的128个值以提供给NPU 0至127: W_i 、 U_i 、 B_i 、 W_f 、 U_f 、 B_f 、 W_c 、 U_c 、 B_c 、 C' 、 $\text{TANH}(C)$ 、 C 、 W_o 、 U_o 、 B_o 。优选地,权重值和偏置值- W_i 、 U_i 、 B_i 、 W_f 、 U_f 、 B_f 、 W_c 、 U_c 、 B_c 、 W_o 、 U_o 、 B_o (位于行0至8以及行12至14)-由运行于处理器100上的架构程序

经由MTNN指令1400写入/填入权重RAM 124,并由运行于NNU 121上的非架构程序(诸如图48的非架构程序等)读取/使用。优选地,如以下更详细所述,中间值 $-C'$ 、 $\text{TANH}(C)$ 、 C (位于行9至11)-由运行于NNU 121上的非架构程序写入/填入权重RAM 124并且还由其读取/使用。

[0452] 如图所示,数据RAM 122保持一系列时间步骤的输入(X)、输出(H)、输入门(I)、遗忘门(F)和输出门(O)的值。更具体地,五行的五元组保持给定时间步骤的X、H、I、F和O的值。如图所示,在数据RAM 122具有64行的实施例中,数据RAM 122可保持12个不同时间步骤的单元值。在图47的示例中,行0至4保持时间步骤0的单元值,行5至9保持时间步骤1的单元值,依此类推,行55至59保持时间步骤11的单元值。五元组中的第一行保持该时间步骤的X值。五元组中的第二行保持该时间步骤的H值。五元组中的第三行保持该时间步骤的I值。五元组中的第四行保持该时间步骤的F值。五元组中的第五行保持此时间步骤的O值。如图所示,数据RAM 122的各列保持其相应神经元或NPU 126的值。也就是说,如以下针对图48更详细所述,列0保持与LSTM单元0相关联的值,其计算由NPU 0执行;列1保持与LSTM单元1相关联的值,其计算由NPU 1执行;依此类推,列127保持与LSTM单元127相关联的值,其计算由NPU 127执行。

[0453] 优选地,(位于行0、5、9、依此类推至行55的)X值由运行于处理器100上的架构程序经由MTNN指令1400写入/填入数据RAM 122,并由运行于NNU 121上的非架构程序(诸如图48的非架构程序等)进行读取/使用。优选地,如以下更详细所述,(位于行2/3/4、7/8/9、12/13/14、依此类推至行57/58/59的)I值、F值与O值由运行于NNU 121上的非架构程序写入/填入数据RAM 122并且还由其进行读取/使用。优选地,(位于行1、6、10、依此类推至行56的)H值由运行于NNU 121上的非架构程序写入/填入数据RAM 122并且由其进行读取/使用,并由运行于处理器100上的架构程序经由MFNN指令1500进行读取。

[0454] 图47的示例假定架构程序执行以下步骤:(1)将12个不同的时间步骤的输入X的值填入数据RAM 122(行0,5,依此类推至行55);(2)开始图48的非架构程序;(3)检测非架构程序已经完成;(4)从数据RAM 122读出输出H的值(行1,6,依此类推至行59);以及(5)按需要重复步骤(1)至(4)若干次以完成任务(例如用于执行对移动电话用户的话语进行识别的计算)。

[0455] 在备选方式中,架构程序执行以下步骤:(1)将单个时间步骤的输入X的值填入数据RAM 122(例如,行0);(2)开始非架构程序(图48中不需要循环并且访问数据RAM 122的其中的单个五元组的修改版本);(3)检测非架构程序已完成;(4)从数据RAM 122读出输出H的值(例如,行1);以及(5)按需要重复步骤(1)至(4)若干次以完成任务。优选这两种方式中的哪一种取决于LSTM层的输入X值的取样方式。例如,如果任务容许在多个时间步骤(例如量级为12个时间步骤)对输入进行取样并执行计算,则可以优选第一种方式,因为该方式有可能带来更多计算资源效率和/或更高的性能,然而,如果该任务只容许在单个时间步骤执行取样,则可能需要第二种方式。

[0456] 设想了与第二种方式类似的第三实施例,但是在该第三实施例中,不同于使用数据RAM 122的行的单个五元组,与第一种方式类似地,非架构程序使用行的多个五元组,即针对各个时间步骤使用不同的五元组。在该第三实施例中,架构程序优选在步骤(2)之前包含如下的步骤,在该步骤中,架构程序在开始非架构程序之前对该非架构程序进行更新,例如将地址0处的指令内的数据RAM 122行更新为指向下一五元组。

[0457] 现参考图48,示出一表格,该表格示出储存于NNU 121的程序存储器129中并由NNU 121执行以实现与LSTM单元层相关联的计算的程序,并且使用根据图47的布置的数据和权重。图48的示例性程序包括位于地址0至23的24个非架构指令。地址0处的指令 (INITIALIZE NPU、CLR ACC、LOOPCNT=12、DR IN ROW=-1、DR OUT ROW=2) 清除累加器202并将循环计数器3804初始化为值12,以使循环体 (地址1至22的指令) 执行12次。该初始化指令还将数据RAM 122的待读取行 (例如,图26/39的寄存器2608) 初始化为值-1,该值将通过地址1处的指令的第一次执行实例而递增至零。该初始化指令还将数据RAM 122的待写入行 (例如,图26/39的寄存器2606) 初始化为行2。优选地,该初始化指令还将NNU 121置为宽配置,使得NNU 121被配置为512个NPU 126。从以下说明中可以发现,在地址0至23的指令执行期间,这512个NPU 126中的128个NPU 126对应于并用作128个LSTM单元4600。

[0458] 如图47所示,在地址1至4处的指令的第一次执行实例期间,这128个NPU 126 (即NPU 0至127) 中的各NPU 126针对第一时间步骤 (时间步骤0) 计算相应的LSTM单元4600的输入门 (I) 值并将I值写入数据RAM 122的行2的相应字;在地址1至4处的指令的第二次执行实例期间,这128个NPU 126中的各NPU 126针对第二时间步骤 (时间步骤1) 计算相应的LSTM单元4600的I值并将I值写入数据RAM 122的行7的相应字;依此类推,在地址1至4处的指令的第十二次执行实例期间,这128个NPU 126中的各NPU 126针对第十二时间步骤 (时间步骤11) 计算相应的LSTM单元4600的I值并将I值写入数据RAM 122的行57的相应字。

[0459] 更具体地,地址1处的乘法累加指令读取当前数据RAM 122行后面的包含与当前时间步骤相关联的单元输入 (X) 值的下一行 (在第一次执行实例期间为行0,在第二次执行实例期间为行5,依此类推,在第十二次执行实例期间为行55),读取权重RAM 124中包含 W_i 值的行0,并且将所读取的值相乘以产生第一乘积,其中该第一乘积累加至刚刚通过地址0处的初始化指令或地址22处的指令清除的累加器202中。接着,地址处2的乘法累加指令读取数据RAM 122中包含与当前时间步骤相关联的单元输出 (H) 值的下一行 (在第一次执行实例期间为行1,在第二次执行实例期间为行6,依此类推,在第十二次执行实例期间为行56),读取权重RAM 124中包含 U_i 值的行1,并且将所读取的值相乘以产生第二乘积,其中该第二乘积相加至累加器202中。通过地址2处的指令 (以及地址6、10和18处的指令) 从数据RAM 122读取的与当前时间步骤相关联的H值在先前时间步骤产生,并通过地址22处的输出指令写入数据RAM 122;然而,在地址2处的指令的第一次执行实例的情况下,数据RAM 122的行1中的H值以初始值写入。优选地,架构程序 (例如,使用MTNN指令1400) 在开始图48的非架构程序之前将初始H值写入数据RAM 122的行1;然而,设想了其它实施例,其中非架构程序包含用于将初始H值写入数据RAM 122的行1的初始指令。在一个实施例中,初始H值为零。接着,地址3处的将权重字相加至累加器的指令 (ADD_W_ACC WR ROW2) 读取权重RAM 124中包含 B_i 值的行2并将其相加至累加器202。最后,地址4处的输出指令 (OUTPUT SIGMOID、DR OUT ROW+0、CLR ACC) 对累加器202值执行S型激活函数并将结果写入当前数据RAM 122输出行 (针对第一次执行实例为行2,针对第二次执行实例为行7,依此类推,针对第十二次执行实例为行57) 并且清除累加器202。

[0460] 如图47所示,在地址5至8处的指令的第一次执行实例期间,128个NPU 126中的各NPU 126针对第一时间步骤 (时间步骤0) 计算相应LSTM单元4600的遗忘门 (F) 值,并将该F值写入数据RAM 122的行3的相应字;在地址5至8处的指令的第二次执行实例期间,128个NPU

126中的各NPU 126针对第二时间步骤(时间步骤1)计算相应LSTM单元4600的F值,并将该F值写入数据RAM 122的行8的相应字;依此类推,在地址5至8处的指令的第十二次执行实例期间,128个NPU 126中的各NPU 126针对第十二时间步骤(时间步骤11)计算相应LSTM单元4600的F值,并将该F值写入数据RAM 122的行58的相应字。地址5至8处的指令以与上述的地址1至4处的指令相同的方式计算F值,然而,地址5至7处的指令分别从权重RAM 124的行3、4和5读取 W_f 、 U_f 和 B_f 的值,以执行乘法和/或加法运算。

[0461] 在地址9至12处的指令的十二次执行实例期间,128个NPU 126中的各NPU 126针对相应的时间步骤计算相应的LSTM单元4600的候选单元状态(C')值并将 C' 值写入权重RAM 124的行9的相应字。地址9至12处的指令以与上述的地址1至4处的指令相同的方式计算 C' 值,然而,地址9至11处的指令分别从权重RAM 124的行6、7和8读取 W_c 、 U_c 和 B_c 的值,以执行乘法和/或加法运算。此外,地址12处的输出指令执行双曲正切激活函数而非(如地址4的输出指令所执行的)S型激活函数。

[0462] 更具体地,地址9处的乘法累加指令读取包含与当前时间步骤相关联的单元输入(X)值的当前数据RAM 122行(在第一次执行实例期间为行0,在第二次执行实例期间为行5,依此类推,在第十二次执行实例期间为行55),读取权重RAM 124中包含 W_c 值的行6,并且将所读取的值相乘以产生第一乘积,其中该第一乘积累加至刚刚通过地址8处的指令清除的累加器202中。接着,地址10处的乘法累加指令读取数据RAM 122中包含与当前时间步骤相关联的单元输出(H)值的下一行(在第一次执行实例期间为行1,在第二次执行实例期间为行6,依此类推,在第十二次执行实例期间为行56),读取权重RAM 124中包含 U_c 值的行7,并且将所读取的值相乘以产生第二乘积,其中该第二乘积相加至累加器202中。接着,地址11处的将权重字相加至累加器的指令读取权重RAM 124中包含 B_c 值的行8并将结果相加至累加器202。最后,地址12处的输出指令(OUTPUT TANH、WR OUT ROW 9、CLR ACC)对累加器202值执行双曲正切激活函数并将结果写入权重RAM 124的行9,并且清除累加器202。

[0463] 在地址13至16处的指令的十二次执行实例期间,128个NPU 126中的各NPU 126针对相应时间步骤计算相应LSTM单元4600的新的单元状态(C)值,并将新的 C 值写入权重RAM 124的行11的相应字,并且计算 $\tanh(C)$ 并将其写入权重RAM 124的行10的相应字。更具体地,地址13的乘法累加指令读取当前数据RAM 122行后面的包含与当前时间步骤相关联的输入门(I)值的下一行(在第一次执行实例期间为行2,在第二次执行实例期间为行7,依此类推,在第十二次执行实例期间为行57),读取权重RAM 124中包含候选单元状态(C')值(刚刚通过地址12处的指令写入)的行9,并且将所读取的值相乘以产生第一乘积,其中该第一乘积累加至刚刚通过地址12处的指令清除的累加器202中。接着,地址14处的乘法累加指令读取数据RAM 122中包含与当前时间步骤相关联的遗忘门(F)值的下一行(在第一次执行实例期间为行3,在第二次执行实例期间为行8,依此类推,在第十二次执行实例期间为行58),并读取权重RAM 124中包含先前时间步骤期间所计算出的当前单元状态(C)值(通过地址15处的指令的最近一次执行实例写入)的行11,并且将所读取的值相乘以产生第二乘积,其中该第二乘积被相加至累加器202。接着,地址15处的输出指令(OUTPUT PASSTHRU、WR OUT ROW 11)使累加器202的值通过并将这些值写入权重RAM 124的行11。应当理解,通过地址14处的指令从数据RAM 122的行11读取的 C 值是地址13至15处的指令的最近一次执行实例所产生并写入的 C 值。地址15处的输出指令不清除累加器202,使得其值可以被地址16处的指

令使用。最后,地址16处的输出指令(OUTPUT TANH、WR OUT ROW 10、CLR ACC)对累加器202的值执行双曲正切激活函数并将结果写入权重RAM 124的行10以供地址21处的计算单元输出(H)值的指令使用。地址16处的指令清除累加器202。

[0464] 如图47所示,在地址17至20处的指令的第一次执行实例期间,128个NPU 126中的各NPU 126针对第一时间步骤(时间步骤0)计算相应LSTM单元4600的输出门(O)值并将O值写入数据RAM 122的行4的相应字;在地址17至20处的指令的第二次执行实例期间,128个NPU 126中的各NPU 126针对第二时间步骤(时间步骤1)计算相应LSTM单元4600的O值并将O值写入数据RAM 122的行9的相应字;依此类推,在地址17至20处的指令的第十二次执行实例期间,128个NPU 126中的各NPU 126针对第十二时间步骤(时间步骤11)计算相应LSTM单元4600的O值并将O值写入数据RAM 122的行58的相应字。地址17至20处的指令以与上述的地址1至4处的指令相同的方式计算O值,然而,地址17至19处的指令分别从权重RAM 124的行12、13和14读取 W_o 、 U_o 和 B_o 的值,以执行乘法和/或加法运算。

[0465] 如图47所示,在地址21至22处的指令的第一次执行实例期间,128个NPU 126中的各NPU 126针对第一时间步骤(时间步骤0)计算相应LSTM单元4600的单元输出(H)值并将该H值写入数据RAM 122的行6的相应字;在地址21至22处的指令的第二次执行实例期间,128个NPU 126中的各NPU 126针对第二时间步骤(时间步骤1)计算相应LSTM单元4600的H值并将该H值写入数据RAM 122的行11的相应字;依此类推,在地址21至22处的指令的第十二次执行实例期间,128个NPU 126中的各NPU 126针对第十二时间步骤(时间步骤11)计算相应LSTM单元4600的H值并将该H值写入数据RAM 122的行60的相应字。

[0466] 更具体地,地址21处的乘法累加指令读取当前数据RAM 122行后面的包含与当前时间步骤相关联的输出门(O)值的接下来的第三行(在第一次执行实例期间为行4,在第二次执行实例期间为行9,依此类推,在第十二次执行实例期间为行59),读取权重RAM 124中包含tanh(C)值(由地址处16的指令写入)的行10,并且将所读取的值相乘以产生一乘积,其中该乘积累加至刚刚通过地址20处的指令清除的累加器202中。然后,地址22处的输出指令使累加器202的值通过并将其写入数据RAM 122的接下来第二个输出行11(在第一次执行实例期间为行6,在第二次执行实例期间为行11,依此类推,在第十二次执行实例期间为行61),并且清除累加器202。应当理解,通过地址22处的指令写入数据RAM 122的行(在第一次执行实例期间为行6,在第二次执行实例期间为行11,依此类推,在第十二次执行实例期间为行61)的H值是地址2,6,10和18处的指令的后续执行实例所消耗/读取的H值。然而,第十二次执行实例的写入行61的H值并未被地址2、6、10和18处的指令的执行实例所消耗/读取;反而优选地,该值被架构程序所消耗/读取。

[0467] 地址23的指令(LOOP 1)使循环计数器3804递减并且在新的循环计数器3804的值大于零的情况下循环回到地址1处的指令。

[0468] 现参考图49,示出一框图,该框图示出NPU组内具有输出缓存器屏蔽和反馈能力的NNU 121实施例。图49示出由四个NPU 126构成的单个NPU组4901。尽管图49示出单个NPU组4901,但是应当理解,NNU 121的各NPU 126都包含于NPU组4901内,使得存在N/J个NPU组4901,其中N是NPU 126的数量(例如,在宽配置下为512,或者在窄配置下为1024),而J是组4901内的NPU 126的数量(例如,在图49的实施例下为四个)。图49中将NPU组4901内的四个NPU 126称为NPU 0、NPU 1、NPU 2和NPU 3。

[0469] 图49的实施例中的各NPU 126与针对上述图7所述的NPU 126相同,并且附图标记相同的元件是相同的。然而,复用寄存器208被修改为包含四个额外的输入4905,复用寄存器705被修改为包含四个额外的输入4907,选择输入213被修改为从原始输入211和207以及额外的输入4905中进行选择以提供至输出209,并且选择输入713被修改为从原始的输入711和206以及额外的输入4907中进行选择以提供至输出203。

[0470] 如图所示,图11的行缓存器1104的一部分在图49中被称为输出缓存器1104。更具体地,示出输出缓存器1104的字0、1、2和3,其中这些字接收与NPU 0、1、2和3相关联的四个AFU 212的相应输出。输出缓存器1104中包含与NPU组4901相对应的N个字的字的部分被称为输出缓存器字组。在图49的实施例中,N为4。输出缓存器1104的四个字被反馈至复用寄存器208和705,并且作为四个额外输入4905由复用寄存器208接收以及作为四个额外输入4907由复用寄存器705接收。将输出缓存器字组反馈至其相应NPU组4901,这为非架构程序的算术指令提供了从与NPU组4901相关联的输出缓存器1104的字(即输出缓存器字组)中选择一个或两个字作为其输入的能力,其示例稍后参照图51的非架构程序(例如地址4、8、11、12和15处的指令)进行描述。也就是说,非架构指令内所指定的输出缓存器1104的字确定选择输入213/713上产生的值。这种能力有效地使得输出缓存器1104能够用作一类便笺式存储器(scratch pad memory),这可以使得非架构程序能够减少对数据RAM 122和/或权重RAM 124的写入次数以及后续从中读取的次数,例如减少中间产生和使用的值的数量。优选地,输出缓存器1104或行缓存器1104包括一维寄存器阵列,其中该一维寄存器阵列可被配置为储存1024个窄字或512个宽字。优选地,输出缓存器1104可以在单个时钟周期内被读取,并且在单个时钟周期内被写入。与可由架构程序和非架构程序两者访问的数据RAM 122和权重RAM 124不同,输出缓存器1104不可由架构程序访问,而只能由非架构程序访问。

[0471] 输出缓存器1104被修改为接收掩码输入4903。优选地,掩码输入4903包括与输出缓存器1104的四个字相对应的四个位,其中所述四个字与NPU组4901的四个NPU 126相关联。优选地,如果与输出缓存器1104的字相对应的掩码输入4903位为真,则输出缓存器1104的字维持其当前值;否则,利用AFU 212输出来更新输出缓存器1104的字。也就是说,如果与输出缓存器1104的字相对应的掩码输入4903位为假,则将AFU 212输出写入输出缓存器1104的字。这为非架构程序的输出指令提供了选择性地将AFU 212输出写入输出缓存器1104的某些字并且维持输出缓存器1104的其它字的当前值的能力,其示例后续参照图51的非架构程序的指令(例如地址6,10,13和14处的指令)进行描述。也就是说,非架构指令中所指定的输出缓存器1104的字确定掩码输入4903上所产生的值。

[0472] 为了简化说明,图49未示出(例如如图18、19和23的)复用寄存器208/705的输入1811。然而,设想了支持可动态配置的NPU 126和输出缓存器1104的反馈/屏蔽两者的实施例。优选地,在这种实施例中,输出缓存器字组相应地可动态配置。

[0473] 应当理解,尽管描述了NPU组4901内的NPU 126的数量为四的实施例,但还设想了该数量更大或更小的其它实施例。此外,在包括共享AFU 1112的实施例中,诸如图52所示等,在NPU组4901内的NPU 126的数量和AFU 212组内的NPU 126的数量之间可能存在协同关系。如针对图50和图51更详细所述,NPU组内的输出缓存器1104的屏蔽和反馈能力对于高效地执行与LSTM单元4600相关联的计算而言特别有益。

[0474] 现参考图50,示出一框图,该框图示出当NNU 121执行与图46的由128个LSTM单元

4600构成的层相关联的计算时、该NNU 121的数据RAM 122、权重RAM 124和输出缓存器1104内的数据布局的示例。在图50的示例中，NNU 121被配置为例如采用宽配置的512个NPU 126或神经元。如同图47和图48的示例，在图50和图51的示例中，LSTM层中仅存在128个LSTM单元4600。然而，在图50的示例中，全部512个NPU 126（例如，NPU 0至127）所产生的值都会被使用。有利地，在执行图51的非架构程序时，各NPU组4901整体作为LSTM单元4600进行操作。

[0475] 如图所示，数据RAM 122保持一系列时间步骤的单元输入(X)值和单元输出(H)值。更具体地，由两行构成的一对保持给定时间步骤的X值和H值。在数据RAM 122具有64行的实施例中，如图所示，数据RAM 122可以保持31个不同时间步骤的单元值。在图50的示例中，行2和3保持时间步骤0的值，行4和5保持时间步骤1的单元值，依此类推，行62和63保持时间步骤30的单元值。该对中的第一行保持时间步骤的X值，而该对中的第二行保持该时间步骤的H值。如图所示，数据RAM 122中与NPU组4901相对应的各四列组保持其相应LSTM单元4600的值。也就是说，如以下针对图51更详细所述，列0至3保持与LSTM单元0相关联的值，其计算由NPU 0~3即NPU组0执行；列4至7保持与LSTM单元1相关联的值，其计算由NPU 4~7即NPU组1执行；依此类推，列508至511保持与LSTM单元127相关联的值，其计算由NPU 508~511即NPU组127执行。如图所示，行1未使用，并且行0保持初始的单元输出(H)值，优选地由架构程序填入零值，但是也设想了非架构的初始指令填入行0的初始单元输出(H)值的实施例。

[0476] 优选地，(位于行2、4、6，依此类推至行62的)X值由运行于处理器100上的架构程序经由MTNN指令1400写入/填入数据RAM 122，并由运行于NNU 121上的非架构程序(诸如图50的非架构程序等)读取/使用。优选地，如以下更详细所述，(位于行3、5、7，依此类推至行63的)H值由运行于NNU 121上的非架构程序写入/填入数据RAM 122并且由其进行读取/使用。优选地，H值还由运行于处理器100上的架构程序经由MFNN指令1500进行读取。应当注意，图51的非架构程序假定在与NPU组4901相对应的各四列组(例如，列0~3、列4~7、列5~8，依此类推至列508~511)内，给定行中的四个X值(例如由架构程序)填入相同的值。同样，图51的非架构程序在给定行的与NPU组4901相对应的各四列组中，计算四个H值并写入相同值。

[0477] 如图所示，权重RAM 124保持NNU 121的NPU的权重值、偏置值和单元状态(C)值。在与NPU组4901相对应的各四列组中(例如，列0~3、列4~7、列5~8，依此类推至列508~511)：(1)索引除以4的余数等于3的列在行0、1、2和6中分别保持 W_c 、 U_c 、 B_c 和C的值；(2)索引除以4的余数等于2的列在行3、4和5中分别保持 W_o 、 U_o 和 B_o 的值；(3)索引除以4的余数等于1的列在行3、4和5中分别保持 W_f 、 U_f 和 B_f 的值；以及(4)索引除以4的余数等于0的列在行3、4和5中分别保持 W_i 、 U_i 和 B_i 的值。优选地，权重值和偏置值-(行0至5中的) W_i 、 U_i 、 B_i 、 W_f 、 U_f 、 B_f 、 W_c 、 U_c 、 B_c 、 W_o 、 U_o 、 B_o -由运行于处理器100上的架构程序经由MTNN指令1400写入/填入权重RAM 124中，并由运行于NNU 121的非架构程序(诸如图51的非架构程序等)进行读取/使用。优选地，如以下更详细所述，中间的C值由运行于NNU 121上的非架构程序写入/填入权重RAM 124并由其进行读取/使用。

[0478] 图50的示例假定架构程序执行以下步骤：(1)将31个不同时间步骤的输入X的值填入数据RAM 122(行2、4，依此类推至行62)；(2)开始图51的非架构程序；(3)检测非架构程序已经完成；(4)从数据RAM 122读出输出H的值(行3、5，依此类推至行63)；以及(5)按需要重复步骤(1)至(4)若干次以完成任务(例如用于执行对移动电话用户的话语进行识别的计算)。

[0479] 在备选方式中,架构程序执行以下步骤:(1)将单个时间步骤的输入X的值填入数据RAM 122(例如,行2);(2)开始非架构程序(图51的不需循环并且访问一对数据RAM 122行的修改版本);(3)检测非架构程序已经完成;(4)从数据RAM 122读出输出H的值(例如,行3);以及(5)按需要重复步骤(1)至(4)若干次以完成任务。优选这两种方式中的哪一种取决于LSTM层的输入X值的取样方式。例如,如果任务容许在多个时间步骤(例如量级为31个时间步骤)对输入进行取样并执行计算,则可以优选第一种方式,因为该方式有可能带来更多计算资源效率和/或更高的性能,然而,如果该任务只容许在单个时间步骤中执行取样,则可能需要第二种方式。

[0480] 设想了与第二种方式类似的第三实施例,但是在该第三实施中,不同于使用一对数据RAM 122行,与第一种方式类似,非架构程序使用多对行,即针对各时间步骤使用不同对。在该第三实施例中,架构程序优选在步骤(2)之前包含如下的步骤,在该步骤中,架构程序在开始非架构程序之前对该非架构程序进行更新,例如将地址处1的指令内的数据RAM 122的行更新为指向下一对。

[0481] 如图所示,在执行图51的非架构程序的不同地址处的指令之后,针对相应的NNU 121的NPU 0至511,输出缓存器1104保持单元输出(H)、候选单元状态(C')、输入门(I)、遗忘门(F)、输出门(O)、单元状态(C)和tanh(C)的中间值。在每个输出缓存器字组(例如,输出缓存器1104中与NPU组4901相对应的四字组,例如字0~3、4~7、5~8,依此类推至508~511)内,索引除以4的余数等于3的字被称为OUTBUF[3],索引除以4的余数等于2的字被称为OUTBUF[2],索引除以4的余数等于1的字被称为OUTBUF[1],而索引除以4的余数等于0的字被称为OUTBUF[0]。

[0482] 如图所示,在执行图51的非架构程序的地址2处的指令之后,对于各个NPU组4901而言,将相应LSTM单元4600的初始单元输出(H)值写入输出缓存器1104的全部四个字。在执行地址6处的指令之后,对于各个NPU组4901而言,将相应LSTM单元4600的候选单元状态(C')值写入OUTBUF[3],而输出缓存器1104的其它三个字则维持其先前值。在执行地址10处的指令之后,对于各个NPU组4901而言,将相应LSTM单元4600的输入门(I)值写入OUTBUF[3],将相应LSTM单元4600的遗忘门(F)值写入OUTBUF[1],将相应LSTM单元4600的输出门(O)值写入OUTBUF[2],而OUTBUF[3]则维持其先前值。在执行地址13处的指令之后,对于各个NPU组4901而言,将相应LSTM单元4600的新的单元状态(C)值写入OUTBUF[3](如以下针对图51更详细所述,将包括槽(slot)3中的C值的输出缓存器1104写入权重RAM 124的行6),而输出缓存器1104的其它三个字则维持其先前值。在执行地址14处的指令之后,对于各个NPU组4901而言,将相应LSTM单元4600的tanh(C)值写入OUTBUF[3],而输出缓存器1104的其它三个字则维持其先前值。在执行地址16处的指令之后,对于各个NPU组4901而言,将相应LSTM单元4600的新的单元输出(H)值写入输出缓存器1104的全部四个字。当程序在地址17处循环回到地址3时,该模式将从地址6至16(即排除地址2的执行,这是因为地址2在程序循环之外)再重复三十次。

[0483] 现参考图51,示出一表格,该表格示出储存于图49的NNU 121的程序存储器129中并由该NNU 121执行以实现与LSTM单元层相关联的计算的程序,并且使用根据图50的布置的数据和权重。图51的示例性程序包含地址0至17处的18个非架构指令。地址0的指令是初始化指令,该初始化指令清除累加器202并将循环计数器3804初始化至值31,以使循环体

(地址1至17的指令)执行31次。该初始化指令还使数据RAM 122的待写入行(例如,图26/39的寄存器2606)初始化为值1,该值通过地址16处的指令的第一次执行实例将递增至3。优选地,该初始化指令还将NNU 121置为宽配置,使得NNU 121被配置为512个NPU 126。从以下说明中可以发现,在地址0至17的指令执行期间,512个NPU 126构成的128个NPU组4901中的各NPU组对应于并用作128个LSTM单元4600中的一个LSTM单元。

[0484] 地址1和2处的指令在循环体之外并且只执行一次。这些指令产生初始单元输出(H)值(例如,0值)并将其写入输出缓存器1104的所有字。地址1处的指令从数据RAM 122的行0读取初始H值并将其放入通过地址0处的指令清除的累加器202中。如图50所示,地址2处的指令(OUTPUT PASSTHRU、NOP、CLR ACC)使累加器202的值通过至输出缓存器1104。地址2处的输出指令(以及图51的其它输出指令)中的标示“NOP”表示正输出的值只被写入输出缓存器1104,而不会被写入存储器,即不会被写入数据RAM 122或权重RAM 124。地址2处的指令也清除累加器202。

[0485] 地址3至17处的指令在循环体内,并且其执行次数为循环计数的值(例如,31)。

[0486] 地址3至6处的指令的每次执行实例都计算当前时间步骤的 $\tanh(C')$ 值并将其写入OUTBUF[3],其将由地址11处的指令使用。更具体地,地址3处的乘法累加指令从数据RAM 122的当前读取行(例如,2、4、6,依此类推至62)读取与时间步骤相关联的单元输入(X)值,从权重RAM 124的行0读取 W_c 的值,并将所读取的值相乘以产生一乘积,其中该乘积被相加到通过地址2处的指令清除的累加器202。

[0487] 地址4处的乘法累加指令(MULT-ACCUM OUTBUF[0]、WR ROW 1)从OUTBUF[0]读取H值(即NPU组4901的全部四个NPU 126),从权重RAM 124的行1读取 U_c 值,并将所读取的值相乘以产生第二乘积,其中该第二乘积被相加到累加器202。

[0488] 地址5处的将权重字相加至累加器的指令(ADD_W_ACC WR ROW 2)从权重RAM 124的行2读取 B_c 值并将相加至累加器202。

[0489] 地址6处的输出指令(OUTPUT TANH、NOP、MASK[0:2]、CLR ACC)对累加器202的值执行双曲正切激活函数,并且结果只被写入OUTBUF[3](即,只有NPU组4901中索引除以4的余数等于3的NPU 126对其结果进行写入),并且累加器202被清除。也就是说,如图50所示,地址6处的输出指令对OUTBUF[0]、OUTBUF[1]和OUTBUF[2]进行掩码(如MASK[0:2]命名所示)以使其维持当前值。此外,地址6处的输出指令不会写入存储器(如NOP命名所示)。

[0490] 地址7至10处的指令的每次执行实例计算当前时间步骤的输入门(I)值、遗忘门(F)值和输出门(O)值并将这些值分别写入OUTBUF[0]、OUTBUF[1]和OUTBUF[2],其将由地址11、12和15处的指令使用。更具体地,地址7处的乘法累加指令从数据RAM 122的当前读取行(例如,2、4、6,依此类推至62)读取与时间步骤相关联的单元输入(X)值,从权重RAM 124的行3读取 W_i 、 W_f 和 W_o 值,并将所读取的值相乘以产生乘积,其中该乘积被相加至通过地址6处的指令清除的累加器202。更具体地,在NPU组4901内,索引除以4的余数等于0的NPU 126计算X与 W_i 的乘积,索引除以4的余数等于1的NPU 126计算X与 W_f 的乘积,而索引除以4的余数等于2的NPU 126计算X与 W_o 的乘积。

[0491] 地址8处的乘法累加指令从OUTBUF[0]读取H值(即NPU组4901的全部四个NPU 126),从权重RAM 124的行4读取 U_i 、 U_f 与 U_o 值,并将所读取的值相乘以产生第二乘积,其中该第二乘积被相加至累加器202。更具体地,在NPU组4901内,索引除以4的余数等于0的NPU

126计算H与 U_i 的乘积,索引除以4的余数等于1的NPU 126计算H与 U_f 的乘积,而索引除以4的余数等于2的NPU 126计算H与 U_o 的乘积。

[0492] 地址9处的将权重字相加至累加器的指令从权重RAM 124的行5读取 B_i 、 B_f 和 B_o 的值并将这些值相加至累加器202。更具体地,在NPU组4901中,索引除以4的余数等于0的NPU 126将 B_i 值相加,索引除以4的余数等于1的NPU 126将 B_f 值相加,并且索引除以4的余数等于2的NPU 126将 B_o 值相加。

[0493] 地址10处的输出指令(OUTPUT SIGMOID、NOP、MASK[3]、CLR ACC)对累加器202的值执行S型激活函数并将计算出的I值、F值和O值分别写入OUTBUF[0]、OUTBUF[1]和OUTBUF[2],并且清除累加器202,而不写入存储器。也就是说,如图50所示,地址10处的输出指令对OUTBUF[3]进行掩码(如MASK[3]命名所示)以使OUTBUF[3]维持其当前值(即 C')。

[0494] 地址11至13处的指令的每次执行实例都计算当前时间步骤产生的新的单元状态(C)值并将该值写入权重RAM 124的行6(更具体地,写入行6中与NPU组4901相对应的四列内的索引除以4的余数等于3的字)以供下一个时间步骤中使用(即由地址12处的指令在下次循环迭代期间使用)。此外,地址14处的指令的每次执行实例都将 $\tanh(C)$ 值写入OUTBUF[3],其将由地址15处的指令使用。

[0495] 更具体地,地址11处的乘法累加指令(MULT-ACCUM OUTBUF[0]、OUTBUF[3])从OUTBUF[0]读取输入门(I)值,从OUTBUF[3]读取候选单元状态(C')值,并将所读取的值相乘以产生第一乘积,其中该第一乘积被相加至通过地址10处的指令清除的累加器202中。更具体地,NPU组4901内的四个NPU 126中的各NPU 126计算I值与 C' 值的第一乘积。

[0496] 地址12处的乘法累加指令(MULT-ACCUM OUTBUF[1]、WR ROW 6)指示NPU 126从OUTBUF[1]读取遗忘门(F)值,从权重RAM 124的行6读取相应的字,并将其相乘以产生第二乘积,其中该第二乘积与累加器202内由地址11处的指令产生的第一乘积相加。更具体地,在NPU组4901中的索引除以4的余数等于3的NPU 126的情况下,从行6读取的字是先前时间步骤中计算出的当前单元状态(C)的值,使得第一乘积和第二乘积的总和是新的单元状态(C)。然而,对于NPU组4901中的其它三个NPU 126而言,从行6读取的字是无关的值,这是因为这些字所产生的累加值将不被使用,即不被地址13和14处的指令放入输出缓存器1104而将会被地址14处的指令清除。也就是说,只有NPU组4901中索引除以4的余数等于3的NPU 126所产生得到的新的单元状态(C)值将会被使用,即被地址13和14处的各指令使用。在地址12处的指令的第二至三十一执行实例的情况下,从权重RAM 124的行6读取的C值是循环体的前次迭代期间由地址13处的指令写入的。然而,对于地址12处的指令的第一次执行实例,由架构程序在开始图51的非架构程序之前、或者由非架构程序的修改版本将初始值写入行6中的C值。

[0497] 如图50所示,地址13处的输出指令(OUTPUT PASSTHRU、WR ROW 6、MASK[0:2])使累加器202值、即计算出的C值仅通过至OUTBUF[3](即,只有NPU组4901中索引除以4的余数等于3的NPU 126将计算出的C值写入输出缓存器1104),而权重RAM 124的行6利用更新后的输出缓存器1104写入。也就是说,地址13处的输出指令对OUTBUF[0]、OUTBUF[1]和OUTBUF[2]进行掩码以使OUTBUF[0]、OUTBUF[1]和OUTBUF[2]维持其当前值(即I、F和O)。如上所述,只有与NPU组4901相对应的各四列组内的行6中索引除以4的余数等于3的字中的C值会被使用,即被地址12处的指令使用;因此,如图50所示,非架构程序不会理会权重RAM 124的行6

中位于列0~2、列4-6、依此类推至列508-510中的值(即,I值、F值和O值)。

[0498] 地址14处的输出指令(OUTPUT TANH、NOP、MASK[0:2]、CLR ACC)对累加器202值执行双曲正切激活函数,将计算出来的 $\tanh(C)$ 值写入OUTBUF[3],并清除累加器202,而不写入存储器。如图50所示,如地址13处的输出指令那样,地址14处的输出指令对OUTBUF[0]、OUTBUF[1]和OUTBUF[2]进行掩码,以使OUTBUF[0]、OUTBUF[1]和OUTBUF[2]维持其当前值。

[0499] 地址15至16处的指令的每次执行实例都计算当前时间步骤所产生的单元输出(H)值并将该值写入数据RAM 122的当前输出行后面的接下来的第二行,以供架构程序将来读取以及在下一时间步骤中使用(即在下次循环迭代期间由地址3和7处的指令使用)。更具体地,地址15处的乘法累加指令从OUTBUF[2]读取输出门(O)值,从OUTBUF[3]读取 $\tanh(C)$ 值,并将这两个值相乘以产生乘积,其中该乘积被相加至通过地址14处的指令清除的累加器202中。更具体地,NPU组4901内的四个NPU 126中的各NPU 126计算值O与 $\tanh(C)$ 的乘积。

[0500] 如图50所示,地址16处的输出指令使累加器202的值通过,并在第一次执行实例期间将计算出的H值写入行3,在第二次执行实例期间将计算出的H值写入行5,依此类推,在第三十一次执行实例期间将计算出的H值写入行63,以供地址4和8的处指令后续使用。此外,如图50所示,这些计算出的H值被放入输出缓存器1104中,以供地址4和8处的指令后续使用。地址16处的输出指令还清除累加器202。在一个实施例中,LSTM单元4600被设计成使得地址16处的输出指令(以及/或者图48的地址22处的输出指令)具有激活函数(例如S型或双曲正切函数等),而非使累加器202的值通过。

[0501] 地址17处的循环指令使循环计数器3804递减并且在新的循环计数器3804值大于零的情况下循环回到地址3处的指令。

[0502] 可以发现,图51的非架构程序的循环体中的指令数与图48的非架构程序的循环体中的指令数相比大致减少34%,这得益于图49的NNU 121实施例中的输出缓存器1104的反馈和掩码能力。此外,图51的非架构程序的数据RAM 122中的存储器布局所适应的时间步骤数大致为图48的非架构程序的数据RAM 122中的存储器布局所适应的时间步骤数的三倍,这也得益于图49的NNU 121实施例中的输出缓存器1104的反馈与掩码能力。根据利用NNU 121执行LSTM单元层计算的特地架构程序应用,这些改进可以是有帮助的,特别是在LSTM层中的LSTM单元4600的数量少于或等于128的应用中。

[0503] 在图47至图51的实施例中,假定权重值和偏置值在各个时间步骤维持相同。然而,设想了如下的其它实施例,其中权重值和偏置值随时间步骤而改变,在这种情况下,并非如图47和图50所示将权重RAM 124填充有单个权重值和偏置值组,而是针对各时间步骤将权重RAM 124填充有不同的权重值和偏置值组,并且图48和图51的非架构程序的权重RAM 124地址相应地被修改。

[0504] 一般而言,在前述图47至图51的实施例中,权重值、偏置值和中间值(例如,C、C')储存于权重RAM 124中,而输入值和输出值(例如,X、H)储存于数据RAM 122中。这对于数据RAM 122为双端口且权重RAM 124为单端口的实施例而言可以是有利的,这是因为从非架构程序和架构程序至数据RAM 122会有更多的流量。然而,因为权重RAM 124较大,因此设想了如下的实施例,其中在这些实施例中,将非架构程序和架构程序写成互换储存值的存储器(即,数据RAM 122和权重RAM 124)。也就是说,W值、U值、B值、C'值、 $\tanh(C)$ 值和C值储存于数据RAM 122中,而X值、H值、I值、F值和O值储存于权重RAM 124中(图47的修改实施例);以

及W值、U值、B值和C值储存于数据RAM 122中,而X值和H值储存于权重RAM 124中(图50的修改实施例)。对于这些实施例,由于权重RAM 124较大,因此可以在一个批次中一起处理较多数量的时间步骤。这对于使用NNU 121执行计算的某些架构程序应用而言可以是有利的,其中所述计算受益于较多数量的时间步骤,并且单端口存储器(例如,权重RAM 124)为其提供足够带宽。

[0505] 现参考图52,示出一框图,该框图示出在NPU组内具有输出缓存器掩码和反馈能力并且采用共享AFU 1112的NNU 121的实施例。图52的NNU 121在许多方面与图49的NNU 121类似,并且具有相同附图标记的元件是相同的。然而,图49的四个AFU 212由单个共享AFU 1112所取代,其中该单个共享AFU 1112接收四个累加器202的四个输出217并产生至OUTBUF[0]、OUTBUF[1]、OUTBUF[2]和OUTBUF[3]的四个输出。针对共享AFU 1112的运算,图52的NNU 121以与上文中针对图49至图51所述的相同的方式、且与上文中针对图11至图13所述的方式相同的方式进行操作。

[0506] 现参考图53,示出一框图,该框图示出根据备选实施例的在图49的NNU 121的执行与图46的具有128个LSTM单元4600的层相关联的计算时、该NNU 121的数据RAM 122、权重RAM 124和输出缓存器1104内的数据布局的示例。图53的示例在许多方面与图50的示例类似。然而,在图53中,Wi值、Wf值和Wo值位于行0中(而非如图50的行3);Ui值、Uf值和Uo值位于行1中(而非如图50的行4);Bi值、Bf值和Bo值位于行2中(而非如图50的行5);C值位于行3中(而非如图50位于行6)。另外,图53的输出缓存器1104的内容与图50相同,然而,如以下更详细所述,由于图54与图51的非架构程序的差异,因此在7处(而非图50中的10处)的指令执行之后第三行的内容(即I值、F值、O值和C'值)出现在输出缓存器1104中;在10处(而非图50中的13处)的指令执行之后第四行的内容(即I值、F值、O值和C值)出现在输出缓存器1104中;在11处(而非图50中的14处)的指令执行之后第五行的内容(即I值、F值、O值与tanh(C)值)出现在输出缓存器1104中;以及在13处(而非图50中的16处的)的指令执行之后第六行的内容(即H值)出现在输出缓存器1104。

[0507] 现参考图54,示出一表格,该表格示出储存于图49的NNU 121的程序存储器129中并由该NNU 121执行以实现与LSTM单元层相关联的计算的程序,并且使用根据图53的布置的数据和权重。图54的示例性程序在许多方面与图51的程序类似。更具体地,在图54和图51中,地址0至5处的指令相同;图54的地址7和8处的指令与图51的地址10和11处的指令相同;并且图54的地址10至14处的指令与图51的地址13至17处的指令相同。

[0508] 然而,图54中地址6处的指令并不会清除累加器202(而图51的地址6处的指令会清除累加器202)。此外,地址7至9处的指令并不存在于图54的非架构程序内。最后,除了图54的地址9处的指令读取权重RAM 124的行3而图51的地址12处的指令读取权重RAM的行6以外,图54的地址9处的指令与图51的地址12处的指令相同。

[0509] 由于图54的非架构程序与图51的非架构程序之间的差异,图53的布局使用的权重RAM 124的行数减少三个,并且在程序循环内所包括的指令数也减少三个。实际上,图54的非架构程序的循环体大小基本上只有图48的非架构程序的循环体大小的一半,并且大致只有图51的非架构程序的循环体大小的80%。

[0510] 现参考图55,示出一框图,该框图示出根据备选实施例的NPU 126的部分。更具体地,对于图49的多个NPU 126中的单个NPU 126而言,示出复用寄存器208及其关联的输入

207、211和4905、以及复用寄存器705及其关联的输入206、711和4907。除了图49的输入外，NPU 126的复用寄存器208和复用寄存器705各自接收index_within_group(组内的索引)输入5599。index_within_group输入5599指示特定NPU 126在其NPU组4901内的索引。因此，例如，在各NPU组4901具有四个NPU 126的实施例中，在各NPU组4901内，NPU 126之一在其index_within_group输入5599中接收值零，NPU 126之一在其index_within_group输入5599中接收值一，NPU 126之一在其index_within_group输入5599中接收值二，并且NPU 126之一在其index_within_group输入5599中接收值三。换句话说，NPU 126所接收到的index_within_group输入5599的值是该NPU 126在NNU 121内的索引除以J的余数，其中J是NPU组4901内的NPU 126的数量。因此，例如，NPU 73在其index_within_group输入5599中接收值一，NPU 353在其index_within_group输入5599中接收值三，而NPU 6在其index_within_group输入5599中接收值二。

[0511] 此外，在控制输入213指定预定值(这里被称为“SELF”)的情况下，复用寄存器208选择与index_within_group输入5599的值相对应的输出缓存器1104的输入4905。因此，有利地，在非架构指令以SELF的值指定接收来自输出缓存器1104的数据(在图57的地址2和7的指令中标示为OUTBUF[SELF])的情况下，各个NPU 126的复用寄存器208从输出缓存器1104接收相应字。因此，例如，在NNU 121执行图57的地址2和7的非架构指令的情况下，NPU 73的复用寄存器208在四个输入4905中选择第二个(索引1)以接收来自输出缓存器1104的字73，NPU 353的复用寄存器208在四个输入4905中选择第四个(索引3)以接收来自输出缓存器1104的字353，而NPU 6的复用寄存器208在四个输入4905中选择第三个(索引2)以接收来自输出缓存器1104的字6。尽管未用于图57的非架构程序，但是非架构指令可以指定利用SELF的值(OUTBUF[SELF])接收来自输出缓存器1104的数据，以使控制输入713指定预定值，从而使各个NPU 126的复用寄存器705从输出缓存器1104接收其相应字。

[0512] 现参考图56，示出一框图，该框图示出在NNU 121执行与图43的Jordan RNN相关联的计算但采用图55的实施例所提供的益处时、该NNU 121的数据RAM 122和权重RAM 124内的数据布局的示例。权重RAM 124内的权重的布局与图44相同。数据RAM 122内的值的布局与图44类似，除了各时间步骤具有用于保持输入层节点D值和输出层节点Y值的关联的一对行、而非如图44的行的四元组。也就是说，隐藏层Z值和承接层C值不写入数据RAM 122。而是如针对图57的非架构程序更详细所述，将输出缓存器1104用作隐藏层Z值和承接层C值的便笺式存储器。有利地，OUTBUF[SELF]输出缓存器1104的反馈特征潜在使得非架构程序能够更快(由于将对数据RAM 122的两次写入和两次读取替换为对输出缓存器1104的两次写入和两次读取)，并使得各时间步骤能够使用较少的数据RAM 122空间，如图所示，这使得数据RAM 122能够保持大约两倍于图44和图45的实施例的时间步骤(特别地为32个时间步骤)。

[0513] 现参考图57，示出一表格，该表格示出储存于NNU 121的程序存储器129中并由该NNU 121执行以实现Jordan RNN的程序，并且使用根据图56的布置的数据和权重。图57的非架构程序在许多方面与图45的非架构程序类似，其差异如下所述。

[0514] 图57的示例性程序包括位于地址0至11的12个非架构指令。地址0处的初始化指令清除累加器202并将循环计数器3804初始化为值32，以使循环体(地址2至11的指令)执行32次。地址1处的输出指令将(通过地址0的指令所清除的)累加器202的零值放入输出缓存器1104。可以发现，512个NPU 126在地址2至6的指令的执行期间对应于并用作512个隐藏层节

点Z,而在地址7至10的指令的执行期间对应于并用作512个输出层节点Y。也就是说,地址2至6处的指令的32次执行实例计算32个相应时间步骤的隐藏层节点Z的值,并将该值放入输出缓存器1104中,供地址7至9处的指令的相应32次执行实例使用,从而计算这32个相应时间步骤的输出层节点Y并将其写入数据RAM 122,并且供地址10处的指令的相应32次执行实例使用以将32个相应时间步骤的承接层节点C放入输出缓存器1104中。(放入输出缓存器1104中的第32个时间步骤的承接层节点C并不会被使用。)

[0515] 在地址2和3处的指令(ADD_D_ACC OUTBUF[SELF]和ADD_D_ACC ROTATE,COUNT=511)的第一次执行实例期间,512个NPU 126中的各NPU 126将输出缓存器1104的512个承接节点C值累加至累加器202中,其中这些承接节点C值通过执行地址0至1的指令所产生并写入。在地址2和3处的指令的第二次和后续的执行实例期间,512个NPU 126中的各NPU 126将输出缓存器1104的512个承接节点C值累加至累加器202中,其中这些承接节点C值通过执行地址7至8和10的指令所产生并写入。更具体地,如上所述,地址2处的指令指示各NPU 126的复用寄存器208选择其相应的输出缓存器1104字,并将该字相加至累加器202;地址3处的指令指示NPU 126在512个NPU 126中的由512个NPU 126的相连接的复用寄存器208整体形成的512字轮转器内对承接节点C值进行轮转,以使各NPU 126能够将512个承接节点C值累加至累加器202。地址3处的指令并不会清除累加器202,这使得地址4和5处的指令能够将(与其相应权重相乘的)输入层节点D值与通过执行地址2和3处的指令所累加的承接层节点C值进行累加。

[0516] 在地址4和5处的指令(MULT-ACCUM DR ROW+2,WR ROW 0和MULT-ACCUM ROTATE,WRROW+1,COUNT=511)的各执行实例期间,512个NPU 126中的各NPU 126执行512次乘法运算,以将数据RAM 122中与当前时间步骤相关联的行(例如,对于时间步骤0而言为行0,对于时间步骤1而言为行2,依此类推,对于时间步骤31而言为行62)中的512个输入节点D值乘以来自权重RAM 124的行0至511中的NPU 126的相应列的权重,由此产生512个乘积,其中这512个乘积连同地址2与3处的指令所执行的对512个承接C节点值的累加一起累加至相应NPU 126的累加器202以计算隐藏节点Z层值。

[0517] 在地址6处的指令(OUTPUT PASSTHRU,NOP,CLR ACC)的各次执行期间,512个NPU 126的512个累加器202值通过并被写入输出缓存器1104的相应字,并且累加器202被清除。

[0518] 在地址7和8处的指令(MULT-ACCUM OUTBUF[SELF],WR ROW 512与MULT-ACCUM ROTATE,WR ROW+1,COUNT=511)的各执行实例期间,512个NPU 126中的各个NPU 126执行512次乘法运算,以将输出缓存器1104中的512个隐藏节点Z值(由地址2至6处的指令的相应执行实例所产生并写入)乘以来自权重RAM 124的行512至1023的NPU 126的相应列的权重,从而产生512个乘积,其中这512个乘积被累加至相应NPU 126的累加器202中。

[0519] 在地址9处的指令(OUTPUT ACTIVATION FUNCTION,DR OUT ROW+2)的各次执行实例期间,对512个累加值执行激活函数(例如,双曲正切函数、S型函数、校正函数)以计算输出节点Y层值,其被写入数据RAM 122中与当前时间步骤相关联的行(例如,对于时间步骤0而言为行1,对于时间步骤1而言为行3,依此类推,对于时间步骤31而言为行63)。地址9处的输出指令不清除累加器202。

[0520] 在地址10处的指令(OUTPUT PASSTHRU,NOP,CLR ACC)的各次执行实例期间,通过地址7和8处的指令所累加的512个值被放入输出缓存器1104中以供地址2和3处的指令的下

一次执行实例使用,并且累加器202被清除。

[0521] 地址11处的循环指令使循环计数器3804递减、并且在新的循环计数器3804值大于零的情况下循环回到地址处2的指令。

[0522] 如针对图44所述,在利用图57的非架构程序所执行的示例性Jordan RNN中,尽管对累加器202值应用激活函数以产生输出层节点Y值,但是假定应用激活函数之前的累加器202值通过承接层节点C、而非实际的输出层节点Y值。然而,对于将激活函数应用于累加器202值以产生承接层节点C的Jordan RNN而言,地址10处的指令将从图57的非架构程序中移除。尽管描述了Elman或Jordan RNN包括单个隐藏节点层的实施例(例如,图40和42),但是应当理解,处理器100和NNU 121的实施例被配置为以本文所述的方式类似的方式,高效地执行与包括多个隐藏层的RNN相关联的计算。

[0523] 如以上针对图2所述,有利地,各个NPU 126被配置为作为人工神经网络内的神经元进行操作,而NNU 121的所有NPU 126以大量并行的方式进行操作以高效地计算网络层的神经元输出值。NNU进行操作(特别是通过使用NPU复用寄存器整体构成的轮转器进行操作)的并行方式可能是与计算神经元层输出值的传统方式直觉上相反的方式。更具体地,传统方式通常涉及执行与单个神经元或相对小的神经元子集相关联的计算(例如,使用并行算术单元来执行乘法与加法),然后继续执行与层中的下一个神经元相关联的计算,依此类推以连续方式继续执行,直到针对该层中的所有神经元进行了计算为止。与之相对,在各个时钟周期,NNU 121的所有NPU 126(神经元)并行地执行与所有神经元输出的产生相关联的计算的小子集(例如,单个乘法与累加)。有利地,在大约M个时钟周期结束时(其中M是当前层内连接的节点数),NNU 121已经计算出所有神经元的输出。对于许多人工神经网络配置,由于存在大量NPU 126,因此NNU 121能够在M个时钟周期结束时计算整层的所有神经元的神经元输出值。从本文的说明中可以发现,这种计算效率对于所有类别的人工神经网络计算都有用,其中这些人工神经网络包含但不限于前馈与循环神经网络,诸如Elman、Jordan和LSTM网络等。最后,虽然描述了NNU 121被配置为512个NPU 126(例如采用宽字配置)以执行循环神经网络计算的实施例,但是也设想了NNU 121被配置为1024个NPU 126(例如采用窄字配置)以执行循环神经网络计算的其它实施例,并且如上所述设想了具有512或1024以外的不同数量的NPU 126的NNU 121的实施例。

[0524] 神经处理单元

[0525] 现参考图58,示出一框图,该框图示出NNU 121的一部分的实施例。NNU 121包括移动单元5802、移动寄存器5804、数据复用寄存器208、权重复用寄存器705、NPU 126、复用器5806、输出单元5808、以及输出寄存器1104。数据复用寄存器208和复用寄存器208与上述相同,但是被修改为额外地接收来自移动寄存器5804以及来自额外的相邻NPU 126的输入。在一个实施例中,除如上所述来自J+1的输出209以外,数据复用寄存器208还在输入211上接收来自NPU J-1和J-4的输出209;同样,除如上所述来自J+1的输出203以外,权重复用寄存器705还在输入711上接收来自NPU J-1和J-4的输出203。输出寄存器1104与上述被称为行缓存器1104和输出缓存器1104的缓存器相同。输出单元5808在许多方面与上述的激活函数单元212/1112相同,其可以包括激活函数(例如,S型函数、双曲正切函数、校正函数、软加函数);然而,这些输出单元5808优选还包括用于对累加器202的值进行再量化的再量化单元,其实施例如下所述。NPU 126在许多方面与上述相同;然而将在图58之后的图中对NPU 126

的方面进行更详细的说明。如上所述,设想了不同实施例,其中数据字宽度和权重字宽度可以具有各种大小(例如,8位、9位、12位或16位),并且多个字大小可以被给定实施例支持(例如,8位和16位)。然而,针对下图示出代表性实施例,其中保持在存储器122/124、移动寄存器5804、复用寄存器208/705以及输出寄存器1104中的数据字宽度和权重字宽度是8位字,即字节。

[0526] 图58示出NNU 121的截面。例如,所示的NPU 126是(诸如上述等的)NPU 126的阵列的代表。代表性NPU 126指的是N个NPU 126中的NPU[J]126,其中J在0和N-1之间。如上所述,N是较大的数,并且优选为2的幂次方。如上所述,N可以是512、1024或2048。在一个实施例中,N是4096。由于阵列中的大量NPU 126,因此有利的是,各NPU 126尽可能小,以将NNU 121的大小保持在期望限度内和/或容纳更多NPU 126以增加NNU 121所进行的神经网络相关计算的加速性。以下针对图59至图61详细说明了有助于得到相对较小的大小、同时仍提供在神经网络计算方面有用的许多功能的NPU 126的实施例。

[0527] 此外,尽管移动单元5802和移动寄存器5804各自的宽度为N个字节,但是仅示出移动寄存器5804的一部分。具体地,示出移动寄存器5804中的输出5824向NPU[J]126的复用寄存器208/705提供字节的部分,该部分表示为移动寄存器[J]5804。此外,尽管移动单元5802的输出5822(向存储器122/124和移动寄存器5804)提供N个字节,但是仅字节J被提供用于加载到移动寄存器[J]5804中,该移动寄存器[J]5804随后在输出5824上将字节J提供至数据复用寄存器208和权重复用寄存器705。

[0528] 此外,尽管NNU 121包括多个输出单元5808,但图58中仅示出单个输出单元5808,即对NPU组内的NPU[J]126以及多个NPU 126(诸如以上针对图11和52所述的NPU 126等)的累加器输出217执行运算的输出单元5808。输出单元5808被称为输出单元[J/4],这是因为与图52的实施例相同,在图58的实施例中,各输出单元5808由四个NPU 126的组所共享。同样,尽管NNU 121包括多个复用器5806,但图58中仅示出单个复用器5806,即接收其NPU组内的NPU[J]126以及多个NPU 126的累加器输出217的复用器5806。同样,复用器5806指的是复用器[J/4],这是因为该复用器5806在四个累加器202输出217中选择其一以提供至输出单元[J/4]5808。

[0529] 最后,尽管输出寄存器1104的宽度为N个字节,但在图58中仅示出单个4字节段(表示为输出寄存器[J/4]1104),其中该4字节段从包括NPU[J]126的NPU组内的四个NPU 126接收输出单元[J/4]5808所产生的四个量化字节。输出寄存器1104的输出133中的所有N个字节被提供至移动单元5802,但是在图58中仅示出输出寄存器[J/4]1104的四字节段中的四个字节。此外,如以上针对图49和图52更详细所述,输出寄存器[J/4]1104的四字节段中的四个字节被作为输入提供至复用寄存器208/705。

[0530] 尽管在图58中复用寄存器208/705被示出为与NPU 126相区分,但存在与各NPU 126相关联的一对相应的复用寄存器208/705,并且如以上例如针对图2、图7、图49和图52所述,这些复用寄存器208/705可被认为是NPU 126的一部分。

[0531] 移动单元5802的输出5822被耦接至移动寄存器5804、数据RAM 122和权重RAM 124,其各自可以被输出5822写入。移动单元5802的输出5822、移动寄存器5804、数据RAM 122和权重RAM 124的宽度都是N个字节(例如,N是4906)。移动单元5802从如下的五个不同源接收N个量化字节并且选择其中之一作为其输入:数据RAM 122、权重RAM 124、移动寄存

器5804、输出寄存器1104以及即时值。优选地,移动单元5802包括互连以能够对其输入执行运算从而产生其输出5822的多个复用器,现将对这些运算进行说明。

[0532] 移动单元5802对其输入执行的运算包括:使输入通过至输出;使输入轮转指定量;以及提取并且压紧输入的指定字节。在从程序存储器129拾取的MOVE指令中指定运算。在一个实施例中,可指定的轮转量为8、16、32和64字节。在一个实施例中,轮转方向向左,但是也设想了轮转方向向右或任意方向的其它实施例。在一个实施例中,提取和压紧运算是在预定大小的输入块内执行的。块大小由MOVE指令所指定。在一个实施例中,预定块大小为16、32和64字节,并且这些块位于指定块大小的对齐边界上。因此,例如,当MOVE指令指定块大小为32时,移动单元5802提取输入的N个字节的各32字节块(例如,如果N为4096,则存在128个块)内的指定字节,并且将其压紧在相应的32字节块内(优选在块的一端)。在一个实施例中,NNU 121还包括与移动寄存器5804相关联的N位掩码寄存器(未示出)。指定加载掩码寄存器操作的MOVE指令可以指定数据RAM 122或权重RAM 124的行作为其源。响应于MOVE指令指定加载掩码寄存器的操作,移动单元5802从RAM的行的N个字中的各字中提取位0,并将N个位储存到N位掩码寄存器的相应位中。在执行用于写入移动寄存器5804的后续MOVE指令期间,位掩码中的位用作移动寄存器5804的相应字节的写入启用/禁用。在备选实施例中,64位掩码由INITIALIZE指令指定,该指令用于在执行用以指定提取和压紧功能的MOVE指令之前加载到掩码寄存器中;响应于MOVE指令,移动单元5802提取由储存在掩码寄存器中的64位掩码所指定的(例如,128个块中的)每个块内的字节。在备选实施例中,用于指定提取和压紧运算的MOVE指令还指定步幅和偏移;响应于MOVE指令,移动单元5802从由偏移指定的字节开始在各块内每N个字节进行一次提取,其中N是步幅,并且将所提取的字节压缩在一起。例如,如果MOVE指令指定步幅为3且偏移为2,则移动单元5802在每个块内从字节2开始每三个字节进行一次提取。

[0533] 现参考图59,示出一框图,该框图示出NPU 126的实施例。NPU 126包括控制逻辑5999、用于保持数据量化偏移5942的寄存器、用于保持权重量化偏移5944的寄存器、D-减法器5952、W-减法器5954、表示为复用器1D 5902的第一复用器、表示为复用器1W 5904的第二复用器、表示为复用器2 5906的第三复用器、表示为复用器3 5908的第四复用器、表示为复用器4 5912的第五复用器、乘加器5918以及累加器202。

[0534] D-减法器5952从接收自图58的复用寄存器208的量化数据字209中减去数据量化偏移5942,以产生被提供为复用器1D 5902的输入的第一差5972或第一操作数5972。W-减法器5954从接收自图58的权重复用寄存器705的量化权重字203中减去权重量化偏移5944,以产生被提供为复用器1W 5904的输入的第二差5974或第二操作数5974。优选地,数据量化偏移5942和权重量化偏移5944可经由架构储存指令(例如,图14的MTNN指令或者NNU 121是外围装置的实施例(诸如针对图62所述的环形总线耦接实施例等)中的存储器储存指令)或者通过储存于程序存储器129中并且由定序器128拾取的指令(例如,INITIALIZE指令)进行编程。优选地,数据字209、权重字203、数据量化偏移5942和权重量化偏移5944是8位无符号值,并且第一操作数5972和第二操作数5974是9位带符号值。

[0535] 除第一操作数5972外,复用器1D 5902还接收正1值(例如,带符号的9位值)和负1值(例如,带符号的9位值)。在控制逻辑5999的控制下,复用器1D 5902在其三个输入中选择其一,以作为带符号的9位第一因子5982提供给乘加器5918的第一输入。除第二操作数5974

外,复用器1W 5904还接收正1值(例如,带符号的9位值)和负1值(例如,带符号的9位值)。在控制逻辑5999的控制下,复用器1W 5904在其三个输入中选择其一,以作为带符号的9位第二因子5984提供给乘加器5918的第二输入。

[0536] 复用器2 5906接收第一操作数5972和第二操作数5974,对这两者进行符号扩展(优选扩展至23位),并且在控制逻辑5999的控制下选择其中之一以作为带符号的23位第三操作数5986提供给复用器3 5908的输入。

[0537] 除接收第三操作数5986以外,复用器3 5908还接收零(例如,带符号的23位值)和累加器202的输出217(其也是带符号的23位值)。在控制逻辑5999的控制下,复用器3 5908在其三个输入中选择其一,以作为带符号的23位加数5988提供给乘加器5918的第三输入。加数5988还被提供至控制逻辑5999。

[0538] 乘加器5918产生第一因子5982和第二因子5984的乘积与加数5988的带符号23位总和5976,并将该总和5976作为输入提供给复用器4 5912。总和5976还被提供至控制逻辑5999。乘加器是具有至少三个输入操作数和一个输出的硬件逻辑电路。乘加器产生输入操作数中的两个输入操作数的乘积与另一个输入操作数的总和。乘加器将该总和提供于其输出上。乘加器可以具有额外的输入操作数,在这种情况下,乘加器在其输出上提供两个输入操作数的乘积与其他输入操作数的总和。例如,在乘加器具有四个输入操作数的情况下,该乘加器在其输出上提供乘积与另两个输入操作数的总和。

[0539] 优选地,乘加器5918包括用于产生带符号17位乘积的带符号9位 $\times$ 带符号9位的乘法器部分。优选地,乘加器5918包括用于将带符号17位乘积和作为累加器202的宽度的字(在一个实施例中为带符号23位值)相加的加法器部分。有利地,具有相对于传统乘加器(例如,16位 $\times$ 16位乘法器和32位+32位加法器)而言较小的乘加器,这使得每个NPU 126能够相对较小,这使得阵列中的NPU 126的数量更大。这可以显著地提高NNU 121所执行的神经网络计算的性能。优选地,乘加器5918例如执行由图1和图7的乘法器242和加法器244的组合所执行的功能。

[0540] 在一个实施例中,图59的乘加器5918是逻辑合成工具(例如,加利福尼亚州山景城的Synopsys公司或加利福尼亚州圣何塞的Cadence设计系统公司所开发的众所周知的合成工具)根据诸如以下的Verilog语句所合成的硬件逻辑:

[0541] `assign D = (A*B) + C;`

[0542] 其中A是复用器1D 5902的输出(第一因子5982),B是复用器1W 5904的输出(第二因子5984),C是复用器3 5908的输出(加数5988),以及D是乘加器5918的输出(总和5976)。优选地,A和B被声明为带符号的9位逻辑值,并且C和D被声明为带符号的23位逻辑值。

[0543] 除了总和5976以外,复用器4 5912还在第二输入上接收来自复用器25906的第三操作数5986,并且在第三输入上接收来自控制逻辑5999的饱和值5996。在控制逻辑5999的控制下,复用器4 5912在其输入中选择其一,以提供给累加器202的输入。在一个实施例中,累加器202是寄存器,并且复用器4 5912还在第四输入上接收累加器202的输出217,这使得累加器202能够通过将当前值写回其本身来保留其当前值。在备选实施例中,不利用复用器4 5912的输出更新累加器202,而仅对累加器202进行时钟门控(即,累加器202不进行更新),在这种情况下,期望保留当前累加器202的值。如以下更详细所述,这种示例是在累加器202是求最大函数的较大值或求最小函数的较小值的情况下发生的。由于累加器202不切

换其大部分门,因此可以节省电力。在该实施例中,无需从累加器202到复用器4 5912的反馈路径。

[0544] 除接收加数5988、总和5976和累加器202的输出217以外,控制逻辑5999还接收函数5994。函数5994指定NPU 126被命令执行的运算或函数。优选地,例如如以上针对图34所述,函数5994是被提供给NPU 126的微操作3418的一部分。控制逻辑5999提供适当的饱和值5996,并且控制复用器1D 5902、复用器1W 5904、复用器2 5906、复用器3 5908和复用器4 5912完成如现将针对下表1进行说明的指定运算/函数。

[0545]

函数	操作码名称	<u>1D</u>	<u>1W</u>	<u>2</u>	<u>3</u>	<u>4</u>
ACC=D*W+ACC	D_TIMES_W_ACC	D	W	X	ACC	如果OVF/UDF, 则为SV, 否则为SUM
ACC=D*W	D_TIMES_W	D	W	X	0	SUM

[0546]

ACC=D+ACC	D_PLUS_ACC	D	+1	X	ACC	如果OVF/UDF, 则为SV, 否则为SUM
ACC=D	D_ONLY	D	+1	X	0	SUM
ACC=W+ACC	W_PLUS_ACC	+1	W	X	ACC	如果OVF/UDF, 则为SV, 否则为SUM
ACC=W	W_ONLY	+1	W	X	0	SUM
ACC=ACC-D	ACC_MINUS_D	D	-1	X	ACC	如果OVF/UDF, 则为SV, 否则为SUM
ACC=-D	MINUS_D	D	-1	X	0	SUM
ACC=ACC-W	ACC_MINUS_W	-1	W	X	ACC	如果OVF/UDF, 则为SV, 否则为SUM
ACC=-W	MINUS_W	-1	W	X	0	SUM
ACC=MAX(D,ACC)	D_MAX_ACC	D	-1	D	ACC	如果新符号为负, 则为M2, 否则为ACC
ACC=MAX(W,ACC)	W_MAX_ACC	-1	W	W	ACC	如果新符号为负, 则为M2, 否则为ACC
ACC=MIN(D,ACC)	D_MIN_ACC	D	-1	D	ACC	如果新符号为正, 则为M2, 否则为ACC
ACC=MIN(W,ACC)	W_MIN_ACC	-1	W	W	ACC	如果新符号为正, 则为M2, 否则为ACC
ACC=D+W	D_PLUS_W	+1	W	D	M2	如果OVF/UDF, 则为SV, 否则为SUM
ACC=D-W	D_MINUS_W	-1	W	D	M2	如果OVF/UDF, 则为SV, 否则为SUM
ACC=W-D	W_MINUS_D	D	-1	W	M2	如果OVF/UDF, 则为SV, 否则为SUM
ACC=0	ACC_EQ_0	D	-1	D	0	SUM

[0547] 表1

[0548] 在表1中, 函数 (FUNCTION) 列指定NPU 126响应于指令或命令所执行的数学函数或运算, 其中所述指令或命令指定操作码名称 (OPCODE NAME) 列中所表示的函数5994。列1D、列1W、列2、列3和列4分别与复用器1D 5902、复用器1W 5904、复用器2 5906、复用器3 5908和复用器4 5912相对应。这些列中的值指定控制逻辑5999响应于用于指定表1的给定行中所列出的函数/运算的指令/命令来控制给定复用器选择哪个输入。在表1中:

[0549] 1. MAX (X, Y) 是指两个指定输入中的最大值;

[0550] 2. MIN (X, Y) 是指两个指定输入中的最小值;

- [0551] 3.ACC是指累加器202的输出217；
- [0552] 4.D是指D-减法器5952的第一操作数5972输出；
- [0553] 5.W是指W-减法器5954的第二操作数5974输出；
- [0554] 6.+1是指复用器1D 5902或复用器1W 5902的正1输入；
- [0555] 7.-1是指复用器1D 5902或复用器1W 5902的负1输入；
- [0556] 8.X是指无关条件；
- [0557] 9.M2是指复用器2 5906(可以为D或W)的第三操作数5986的输出；
- [0558] 10.SV是指饱和值5996；
- [0559] 11.SUM是指乘加器5918的总和5976的输出；
- [0560] 12.新符号是指总和5976的符号；以及
- [0561] 13.OVF/UDF是指控制逻辑5999所检测到的溢位/欠位条件。
- [0562] 因此,例如,在从程序存储器129拾取并且由NPU 126执行的指令指定D_TIMES_W_ACC函数的情况下,如表1所述:
- [0563] 1.复用器1D 5902选择第一操作数5972；
- [0564] 2.复用器1W 5904选择第二操作数5974；
- [0565] 3.复用器2 5906是无关的；
- [0566] 4.复用器3 5908选择累加器202的输出217；以及
- [0567] 5.在控制逻辑5999检测到溢位/欠位的情况下,复用器4 5912选择饱和值5996,否则选择总和5976。
- [0568] 再例如,在从程序存储器129拾取并且由NPU 126执行的指令指定D_MAX_ACC函数的情况下,如表1所述:
- [0569] 1.复用器1D 5902选择第一操作数5972；
- [0570] 2.复用器1W 5904选择负一；
- [0571] 3.复用器2 5906选择第一操作数5972；
- [0572] 4.复用器3 5908选择累加器202的输出217；以及
- [0573] 5.在总和5976的符号为负的情况下(例如,在总和5976的高位是二进制一的情况下),复用器4 5912选择第三操作数5986(在这种情况下为复用器25906所选择的第一操作数5972),否则控制逻辑5999使得保留累加器202的当前值。
- [0574] 又例如,在从程序存储器129拾取并且由NPU 126执行的指令指定D_PLUS_W函数的情况下,如表1所述:
- [0575] 1.复用器1D 5902选择正一；
- [0576] 2.复用器1W 5904选择第二操作数5974；
- [0577] 3.复用器2 5906选择第一操作数5972；
- [0578] 4.复用器3 5908选择复用器2 5906所输出的第三操作数5986；以及
- [0579] 5.在控制逻辑5999检测到溢位/欠位的情况下,复用器4 5912选择饱和值5996,否则选择总和5976。
- [0580] W_MAX_ACC函数例如用于图28的最大池化运算(这里被称为MAXWACC)。D_PLUS_ACC函数例如用于图42、图45、图51、图54和图57中所述的循环神经网络计算(这里被称为ADD_D_ACC)。

[0581] 优选地,控制逻辑5999包括溢位/欠位逻辑,其中通过识别为正由乘加器5918相加的两个值的大小不同来有利地对所述溢位/欠位逻辑进行简化。例如,加数5988优选为23位,而第一因子5982和第二因子5984的乘积具有较少的位,例如,17位。图59的实施例中的溢位/欠位逻辑仅检查加数5988的最高两位和总和5976的符号。更具体地,如果加数5988的最高两位是二进制01、并且总和5976的符号为负,则控制逻辑5999检测到溢位;而如果加数5988的最高两位是二进制10、并且总和5976的符号为正,则控制逻辑5999检测到欠位。

[0582] 如果控制逻辑5999检测到溢位,则输出可表示的最大正数作为饱和值5996;如果控制逻辑5999检测到欠位,则输出可表示的最大负数作为饱和值5996。

[0583] 在一个实施例中,NPU 126包括额外的阶段寄存器(未示出)。例如,复用器1D 5902、复用器1W 5904和复用器2 5906可以是复用寄存器。

[0584] 现参考图60,示出一框图,该框图示出NPU 126的备选实施例。图60的NPU 126在许多方面与图59的NPU 126类似。然而,图60的NPU 126还包括取反器6004,复用器1D 5902还接收零值,并且控制逻辑5999提供符号值6002作为复用器4 5912的额外输入。取反器6004接收累加器输出217并且输出其二进制补码,即累加器输出217的算术负值,也称为累加器202的取反值。符号值6002是表示输入(例如,累加器202)为正、为负还是为零的三个不同预定值其中之一。在一个实施例中,在累加器202为零的情况下符号值6002为零,在累加器202为正的情况下为符号值6002为二进制01,并且在累加器202为负的情况下符号值6002为二进制11。取反器6004、复用器1D 5902的零输入以及符号值6002使得图60的NPU 126能够支持在输入至控制逻辑5999的函数5994上指定的至少三个额外函数/运算,即累加器202的绝对值、取反和符号。下表2描述了控制逻辑5999控制复用器5902/5904/5906/5908/5912进行所述三种额外函数的运算。

[0585]	函数	操作码名称	1D	1W	2	3	4
	ACC=ABS(ACC)	ACC_ABS	Z	X	X	N	如果旧符号为负,则为SUM,
[0586]							否则为ACC
	ACC=NEG(ACC)	ACC_NEG	Z	X	X	N	SUM
	ACC=SIGN(ACC)	ACC_SIGN	X	X	X	X	SIGN_VALUE

[0587] 表2

[0588] 在表2中:

[0589] 1.ABS(X)是指输入的绝对值;

[0590] 2.NEG(X)是指输入的算术负值;

[0591] 3.SIGN(X)表示输入为正、为负还是为零;

[0592] 4.Z是指复用器1D 5902的零输入;

[0593] 5.N是指取反器6004的输出6006(累加器202的算术负值);

[0594] 6.SIGN_VALUE是指符号值6002;以及

[0595] 7.旧符号是指加数5988(由于复用器3 5908的选择因而在ABS(ACC)的情况下为累加器202的输出217)的符号。

[0596] 因此,在从程序存储器129拾取并且由NPU 126执行的指令指定ABS_ACC函数的情况下,如表2所述:

[0597] 1.复用器1D 5902选择零输入；

[0598] 2.复用器3 5908选择取反器6004的输出6006(累加器202的算术负值)；以及

[0599] 3.在控制逻辑5999检测到累加器202的旧值/当前值为负的情况下，复用器4 5912选择总和5976，否则控制逻辑5999使得保留累加器202的当前值。

[0600] 另外，在从程序存储器129拾取并且由NPU 126执行的指令指定ABS_NEG函数的情况下，如表2所述：

[0601] 1.复用器1D 5902选择零输入；

[0602] 2.复用器3 5908选择取反器6004的输出6006(累加器202的算术负值)；以及

[0603] 3.复用器4 5912选择总和5976。

[0604] 最后，在从程序存储器129拾取并且由NPU 126执行的指令指定ABS_SIGN函数的情况下，如表2所述：

[0605] 1.在累加器202为零的情况下，控制逻辑5999输出为零的符号值6002，在累加器202为正的情况下控制逻辑5999输出为二进制01的符号值6002，并且在累加器202为负的情况下控制逻辑5999输出为二进制11的符号值6002；以及

[0606] 2.复用器4 5912选择符号值6002。

[0607] 根据上文可以发现，在函数5996指定ACC_ABS或ACC_NEG的情况下，由于复用器1D 5902选择零输入，因此第一因子5982将为零，进而第一因子5982和第二因子5984的乘积将为零，因而乘加器5918将零与加数5988相加。这导致总和5976即为加数5988的值，当函数5996在累加器202为负的情况下指定ACC_NEG或指定ACC_ABS时，该值为累加器202的值的取反值6006；而当函数在累加器202为正的情况下指定ACC_ABS时，控制逻辑5999使得保留累加器202的当前值。在备选实施例中，复用器1W 5904接收零值而非复用器1D 5902，来实现相同的目的。

[0608] 现参考图61，示出一框图，该框图示出NPU 126的备选实施例。在图61的备选实施例中，图60的取反器6004由倒置器6104所取代，其中倒置器6104的输出被提供至复用器3 5908。倒置器6104的输出6106是累加器输出217的二进制反码，即累加器输出217的各个位的倒置，也称为累加器202的倒置值。此外，控制逻辑5999将乘加器5918的附加的一位加数6102提供至该乘加器5918的最低有效位。在操作码是ACC_NEG的情况下或者在操作码是ACC_ABS并且累加器输出217为负的情况下，该一位加数6102是二进制一；否则该一位加数6102是二进制零。在操作码是ACC_ABS并且累加器输出217为负的情况下，利用乘加器5918将为二进制一的加数6102与累加器202的逐位倒置值6106(以及与零乘积)相加，以产生作为累加器输出217的二进制补码的总和5976。与图60的实施例相比，图61的实施例可以具有NPU 126较小的优点。

[0609] 环形总线耦接的神经网络单元；从属接口和多个主接口；从属程序和神经网络程序均可编程的DMA控制器；多个微操作源

[0610] 以上描述了NNU 121是处理器100的执行单元的实施例。现将描述如下的实施例，其中NNU 121与多核处理器的多个传统处理核心一起位于环形总线上，以作为神经网络加速器进行操作，其中该神经网络加速器由其它核心共享，从而代表所述核心以比这些处理核心所能执行的更快的方式执行神经网络相关计算。在许多方面中，NNU 121像外围装置那样进行操作，其中运行在核心上的程序可以控制NNU 121执行神经网络相关计算。优选地，

多核处理器和NNU 121被制造于单个集成电路上。由于NNU 121的大小可能相当大,特别是针对NPU 126的数量和存储器122/124的大小很大(例如,具有4096个字节宽的数据RAM 122和权重RAM 124的4096个NPU 126)的实施例,因此这种实施例可以提供如下的优点,即不会使各核心的大小以NNU 121的大小增加,而是存在比核心少的NNU 121,并且这些核心共享NNU 121,这使得集成电路可以更小,尽管是以潜在较低的性能作为交换。

[0611] 现参考图62,示出一框图,该框图示出处理器100。处理器100包括多个环站4004,其中这多个环站4004以双向方式彼此连接以形成环形总线4024。图40的实施例包括表示为4004-0、4004-1、4004-2、4004-3、4004-M、4004-D和4004-U的七个环站。处理器100包括分别被称为核心复合体0 4012-0、核心复合体1 4012-1、核心复合体2 4012-2和核心复合体3 4012-3的四个核心复合体4012,其中这四个核心复合体4012各自包括用于将核心复合体4012耦接至环形总线4024的四个环站4004-0、4004-1、4004-2和4004-3。处理器100还包括非核心部分4016,其包括用于将非核心4016耦接至环形总线4024的环站4004-U。最后,处理器100包括通过环站4004-D而耦接至环形总线4024的动态随机存取存储器(DRAM)控制器4018。最后,处理器100包括通过环站4004-M而耦接至环形总线4024的NNU 121。在美国非临时申请15366027、15366053和15366057(以下称为“Dual Use NNU Memory Array Applications(双用NNU存储器阵列应用)”,各自提交于2016年12月1日并且通过引用而全文并入于此)中所述的一个实施例中,如其中所述,NNU 121包括存储器阵列,该存储器阵列可被用作NNU 121的NPU 126的阵列所使用的存储器(例如,图1的权重RAM 124)或者用作核心复合体4012所共享的高速缓存存储器,例如用作牺牲高速缓存器(victim cache)或用作末级高速缓存器(LLC)片。尽管图40的示例包括四个核心复合体4012,但还设想了具有不同数量的核心复合体4012的其它实施例。例如,在一个实施例中,处理器100包括八个核心复合体4012。

[0612] 非核心4016包括用于控制处理器100对外围装置可以耦接的系统总线4022的访问的总线控制器4014,诸如视频控制器、盘控制器、外围总线控制器(例如,PCI-E)等。在一个实施例中,系统总线4022是众所周知的V4总线。非核心4016还可以包括其它功能单元,诸如电源管理单元和私有RAM等(例如,核心4002的微码所使用的非架构存储器)。在备选实施例中,DRAM控制器4018耦接至系统总线,并且NNU 121经由环形总线4024、总线控制器4014和DRAM控制器4018来访问系统存储器。

[0613] DRAM控制器4018控制作为系统存储器的DRAM(例如,异步DRAM或同步DRAM(SDRAM),诸如双倍数据速率同步DRAM、直接Rambus DRAM或降低延迟的DRAM等)。核心复合体4012、非核心4016和NNU 121经由环形总线4024来访问系统存储器。更具体地,NNU 121从系统存储器将神经网络的权重和数据读取至数据RAM 122和权重RAM 124中,并经由环形总线4024将来自数据RAM 122和权重RAM 124的神经网络结果写入系统存储器。此外,在作为牺牲高速缓存器进行操作时,存储器阵列(例如,数据RAM 122或权重RAM 124)在高速缓存器控制逻辑的控制下将高速缓存器行驱逐至系统存储器。此外,在作为LLC片进行操作时,存储器阵列和高速缓存器控制逻辑从系统存储器填充高速缓存器行,并将高速缓存器行写回并驱逐至系统存储器。

[0614] 四个核心复合体4012包括各自的LLC片4012-0、4012-1、4012-2和4012-3,其中各LLC片耦接至环站4004并且一般地个别地称为LLC片4006或者共同称为(多个)LLC片4006。

各核心4002包括高速缓存器存储器,诸如耦接至环站4004的2级(L2)高速缓存器4008等。各核心4002还可以包括1级高速缓存器(未示出)。在一个实施例中,核心4002是x86指令集架构(ISA)核心,但是设想了核心4002是另一ISA(例如,ARM、SPARC、MIPS等)核心的其它实施例。

[0615] 如图40所示,LLC片4006-0、4006-1、4006-2和4006-3整体形成核心复合体4012所共享的处理器100的LLC 4005。各LLC片4006包括存储器阵列和高速缓存器控制逻辑。如在上文中通过引用而并入的双用NNU存储器阵列应用中所述,模式指示符可以被设置为使得NNU 121的存储器阵列用作LLC 4005的附加(例如,第五或第九)片4006-4。在一个实施例中,各LLC片4006包括2MB的存储器阵列,但是设想了具有不同大小的其它实施例。此外,设想了存储器阵列的大小和LLC片4006的大小不同的实施例。优选地,LLC 4005包含L2高速缓存器4008以及高速缓存器阶层结构中的任何其它高速缓存器(例如,L1高速缓存器)。

[0616] 环形总线4024或环4024是促进包括DRAM控制器4018、非核心4016以及LLC片4006在内的相干组件之间的通信的可扩展双向互连。环4024包括两个单向环,这两个单向环各自还包括五个子环:请求(Request),用于传输包括加载的大多数类型的请求包;监听(Snoop),用于传输监听请求包;确认(Acknowledge),用于传输响应包;数据(Data),用于传输数据包以及包括写入的特定请求项;以及信用(Credit),用于在远程队列中发射并获得信用。附接至环4024的各节点经由环站4004而连接,其中该环站4004包含用于在环4024上发送和接收包的队列,例如如针对图62至图65更详细所述的队列。队列是代表远程队列中所要接收的附接组件在环4024上发起请求的出口队列、或从环4024接收要转发至附接组件的请求的入口队列。在出口队列在环上发起请求之前,其首先从远程目的地入口队列获得信用环上的信用。这确保了远程入口队列具有可用于在请求到达时处理该请求的资源。在出口队列希望在环4024上发送事务包时,该出口队列仅可以在没有预先占有最终以远程节点为目的地的传入包的情况下发送事务包。在传入包从任意方向到达环站4004时,询问包的目的地ID以确定该环站4004是否是所述包的最终目的地。如果该目的地ID不等于环站4004的节点ID,则包在后续时钟内继续前进至下一环站4004。否则,包在同一时钟内离开环4024,以供包的事务类型所涉及的任一入口队列所消耗。

[0617] 一般而言,LLC 4005包括N个LLC片4006,其中N个片4006中的各片4006负责对通过散列(hash)算法或简称为散列所确定的处理器100的物理地址空间的不同的约1/N进行高速缓存。散列是将物理地址作为输入、并选择负责对该物理地址进行高速缓存的适当LLC片的函数。在必须从核心4002或监听代理向LLC 4005进行请求的情况下,该请求必须被发送至负责对该请求的物理地址进行高速缓存的适当LLC片4006。适当LLC片4006是通过对请求的物理地址应用散列而确定的。

[0618] 散列算法是满射函数(surjective function),其中该满射函数的域是物理地址的集合或其子集,并且该满射函数的范围是当前包括的LLC片4006的数量。更具体地,该范围是LLC片4006的索引(例如,在八个LLC片4006的情况下为0至7)的集合。该函数可以通过检查物理地址位的适当子集来进行计算。例如,在具有八个LLC片4006的系统中,散列算法的输出可以简单地是 $PA[10:8]$,即物理地址位中的三个位,也即位8至位10。在LLC片4006的数量为8的另一实施例中,散列的输出是其它地址位(例如,作为 $\{PA[17], PA[14], PA[12] \sim PA[10] \sim PA[9]\}$ 所产生的三位)的逻辑函数。

[0619] 在任何LLC 4005高速缓存完成之前,该LLC 4005的所有请求器都必须具有相同的散列算法。由于散列规定了操作期间对地址进行高速缓存的位置以及将要发送监听的位置,因此散列仅通过所有核心4002、LLC片4006和监听代理之间的协调来改变。如双用NNU存储器阵列应用中所述,更新散列算法基本上包括:(1)使所有核心4002同步以防止新的可高速缓存访问;(2)执行LLC 4005中当前所包括的所有LLC片4006的回写无效,这导致修改后的高速缓存器行被写回到系统存储器并且所有的高速缓存器行都无效(如下所述,回写无效可以是选择性回写无效,其中只有地址被新散列算法散列至与旧散列算法不同的片的那些高速缓存器行被驱逐,即无效,并且如果被修改,则在无效之前被写回);(3)将散列更新消息广播给各个核心4002和监听源,这命令各个核心4002和监听源改变为新的散列(如下所述,从包含性(inclusive)散列变为独占散列,或者反之亦然);(4)对用于控制对存储器阵列的访问的选择逻辑的模式输入进行更新;以及(5)用新的散列算法恢复执行。

[0620] 当LLC片4006的数量N为8即2的幂次方时,上述的散列算法是有用的,并且这些算法可被修改为容易地适应2的其它幂次方,例如,针对4个片修改为PA[9:8]或者针对16个片修改为PA[11:8]。然而,根据LLC 4005中是否包含NNU LLC片4006-4(以及根据核心复合体4012的数量),N可以是或者可以不是2的幂次方。因此,如双用NNU存储器阵列应用中所述,当NNU 121存储器阵列具有双重用途时,可以使用至少两个不同的散列。

[0621] 在备选实施例中,NNU 121和DRAM控制器4018均耦接至单个环站4004。单个环站4004包括使得NNU 121和DRAM控制器4018能够在彼此之间传输请求和数据而非经由环形总线4024来传输请求和数据的接口。这可以是有利的,因为这可以减少环形总线4024上的流量,并且提供NNU 121和系统存储器之间的高传输性能。

[0622] 优选地,处理器100在单个集成电路或芯片上制造而成。因此,可以以非常高的可持续速率在系统存储器和/或LLC 4005与NNU 121之间实现数据传输,这对于神经网络应用、特别是权重和/或数据的量相对较大的神经网络应用而言可以是非常有利的。亦即,尽管并非像图1的实施例那样是核心4002的执行单元,但NNU 121紧密地耦接至核心4002,相较于例如耦接至诸如PCIe总线等的外围总线的神经网络单元,这可以提供显著的存储器性能优势。

[0623] 现参考图63,示出一框图,该框图更详细地示出图62的环站4004-N。环站4004-N包括从属接口6301、被称为主接口0的第一主接口6302-0、以及被称为主接口1的第二主接口6302-1。主接口0 6302-0和主接口1 6302-1一般地被个别地称为主接口6302或者共同称为(多个)主接口6302。环站4004-N还包括耦接至在环形总线4024的第一单向环4024-0上分别提供传出请求(REQ)、数据(DATA)和确认(ACK)的各缓存器6352、6354和6356的三个仲裁器6362、6364和6366;这三个仲裁器6362、6364和6366在第一单向环4024-0上分别接收传入请求(REQ)、数据(DATA)和确认(ACK)。环站4004-N包括耦接至在环形总线4024的第二单向环4024-1上分别提供传出请求(REQ)、数据(DATA)和确认(ACK)的各附加缓存器6332、6334和6336的三个附加仲裁器6342、6344和6346;这三个仲裁器6342、6344和6346在第二单向环4024-1上分别接收传入请求(REQ)、数据(DATA)和确认(ACK)。以上描述了环形总线4024的各单向环的请求子环、数据子环和确认子环。监听子环和信用子环未示出,但是从属接口6301和主接口6302也耦接至监听子环和信用子环。

[0624] 从属接口6301包括加载队列6312和储存队列6314;主接口0 6302-0包括加载队列

6322和储存队列6324;以及主接口1 6302-1包括加载队列6332和储存队列6334。从属接口6301的加载队列6312接收来自环形总线4024的单向环4024-0和4024-1两者的请求并对这些请求进行排队,并且向环形总线4024的相应仲裁器6364和6344中的各仲裁器提供队列化的数据。从属接口6301的储存队列6314接收来自环形总线4024的两个方向的数据并对这些数据进行排队,并且向环形总线4024的相应仲裁器6366和6346中的各仲裁器提供确认。主接口0 6302-0的加载队列6322接收来自第二单向环4024-1的数据,并向第一单向环4024-0的仲裁器6362提供队列化的请求。主接口0 6302-0的储存队列6324接收来自第二单向环4024-1的确认,并向第一单向环4024-0的仲裁器6364提供队列化的数据。主接口1 6302-1加载队列6332接收来自第一单向环4024-0的数据,并向第二单向环4024-1的仲裁器6342提供队列化的请求。主接口1 6302-1的储存队列6334接收来自第一单向环4024-0的确认,并向第二单向环4024-1的仲裁器6344提供队列化的数据。从属接口6301的加载队列6312将队列化的请求提供至NNU 121,并接收来自NNU 121的数据。从属接口6301的储存队列6314将队列化的请求和数据提供至NNU 121,并接收来自NNU 121的确认。第一主接口0 6302-0的加载队列6322接收来自NNU 121的请求并对这些请求进行排队,并且向NNU 121提供数据。第一主接口0 6302-0的储存队列6324接收来自NNU 121的请求和数据并对这些请求和数据进行排队,并且向NNU 121提供确认。第二主接口1 6302-1的加载队列6332接收来自NNU 121的请求并对这些请求进行排队,并且向NNU 121提供数据。第二主接口1 6302-2的储存队列6334接收来自NNU 121的请求和数据并对这些请求和数据进行排队,并且向NNU 121提供确认。

[0625] 通常,从属接口6301接收核心4002所作出的从NNU 121加载数据的请求(由加载队列6312接收)并且接收核心4002所作出的将数据储存至NNU 121的请求(由储存队列6314接收),但是从属接口6301还可以从其它环形总线4024代理接收这样的请求。例如,经由从属接口6301,核心4002可以:相对于控制/状态寄存器127写入控制数据并且读取状态数据;将指令写入程序存储器129;相对于数据RAM 122和权重RAM 124写入/读取数据/权重;以及将控制字写入总线控制器存储器6636以对NNU 121的DMA控制器6602(参见图66)进行编程。更具体地,在NNU 121位于环形总线4024上而非作为核心4002的执行单元的实施例,核心4002可以对控制/状态寄存器127进行写入以指示NNU 121执行与针对图14的MTNN指令1400所述的类似的操作,并且可以从控制/状态寄存器127进行读取以指示NNU 121执行与针对图15的MFNN指令1500所述的类似的操作。操作的列表包括但不限于:开始程序存储器129中的程序的执行、暂停程序存储器129中的程序的执行、对程序存储器129中的程序的执行完成的请求通知(例如,中断)、对NNU 121进行重设、写入DMA基址寄存器、以及写入选通(strobe)地址以使得相对于数据/权重RAM 122/124写入或读取行缓存器。此外,从属接口6301可以在NNU 121的请求下向每个核心4002产生中断(例如,PCI中断)。优选地,定序器128指示从属接口6301例如响应于对从程序存储器129拾取的指令进行解码而产生中断。可选地,DMAC 6602可指示从属接口6301例如响应于完成DMA操作(例如,在将作为神经网络层计算结果的数据字从数据RAM 122写入系统存储器之后)而产生中断。在一个实施例中,中断包括向量,诸如8位x86中断向量等。优选地,由DMAC 6602从总线控制存储器6636读取的控制字中的标志指定DMAC 6602是否指示从属接口6301在DMA操作完成时产生中断。

[0626] 通常,NNU 121经由主接口6302产生将数据写入系统存储器的请求(由储存队列

6324/6334接收),并且经由主接口6302产生(例如,经由DRAM控制器4018)从系统存储器读取数据的请求(由加载队列6322/6332接收),但主接口6302还可以接收来自NNU 121的相对于其它环形总线4024代理读取/写入数据的请求。例如,经由主接口6302,NNU 121可以将数据/权重从系统存储器传输到数据RAM 122和权重RAM 124,并且可以将数据从数据RAM 122和权重RAM 124传输到系统存储器。

[0627] 优选地,经由环形总线4024可访问的NNU 121的各种实体(诸如数据RAM 122、权重RAM 124、程序存储器129、总线控制存储器6636和控制/状态寄存器127等)被存储器映射到系统存储器空间内。在一个实施例中,可访问的NNU 121实体经由周知的外围组件互连(PCI)配置协议的PCI配置寄存器来进行存储器映射。

[0628] 具有针对环站4004-N的两个主接口6302的优点在于,其使得NNU 121能够相对于系统存储器(经由DRAM控制器4018)和各种L3片4006两者同时进行发送和/或接收,或者备选地能够以具有单个主接口的实施例的两倍带宽相对于系统存储器并行地进行发送和/或接收。

[0629] 在一个实施例中,数据RAM 122是64KB,其被布置为每行4KB的16行,因此需要4个位来指定其行地址;权重RAM 124为8MB,其被布置为每行4KB的2K行,因此需要11个位来指定其行地址;程序存储器129为8KB,其被布置为每行64位的1K行,因此需要10个位来指定其行地址;总线控制存储器6636是1KB,其被布置为每行64位的128行,因此需要7个位来指定其行地址;队列6312/6314/6322/6324/6332/6334中的各队列包括16个条目,因此需要4个位来指定条目的索引。另外,环形总线4024的单向环4024的数据子环的宽度为64字节。因此,64字节的部分在本文中被称为块、数据块、数据的块等(“数据”一般可用于指代数据和权重这两者)。因此,数据RAM 122或权重RAM 124的行尽管在块水平上不可寻址,但各自被细分为64个块;此外,(图66的)数据/权重写入缓存器6612/6622和(图66的)数据/权重读取缓存器6614/6624各自也被细分为各自具有64字节的64个块,并且在块水平上可寻址;因此,需要6个位来指定行/缓存器内的块的地址。以下描述为了便于说明而假定这些大小;然而,设想了各种大小不同的其它实施例。

[0630] 现参考图64,示出一框图,该框图更详细地示出图63的从属接口6301。从属接口6301包括耦接至图63的环形总线4024的加载队列6312、储存队列6314、仲裁器6342、6344、6346、6362、6364和6366、以及缓存器6332、6334、6336、6352、6354和6356。图64还包括产生对仲裁器6362的请求的其它请求器6472(例如,主接口0 6302-0)以及产生对仲裁器6342的请求的其它请求器6474(例如,主接口1 6302-1)。

[0631] 从属加载队列6312包括耦接至请求仲裁器6416和数据仲裁器6414的条目6412的队列。在所示的实施例中,队列包括16个条目6412。每个条目6412包括用于地址、源标识符、方向、事务标识符以及与请求相关联的数据块的存储。地址指定NNU 121内的加载所请求的数据以返回到请求方环形总线4024代理(例如,核心4002)的位置。地址可以指定控制/状态寄存器127、或数据RAM 122或权重RAM 124内的块位置。在地址指定数据RAM 122/权重RAM 124内的块位置的情况下,高位指定数据RAM 122/权重RAM 124的行,而低位(例如,6个位)指定所指定的行内的块。优选地,低位用于控制数据/权重读取缓存复用器6615/6625(参见图66)以选择数据/权重读取缓存器6614/6624内的适当块(参见图66)。源标识符指定请求方环形总线4024代理。方向指定要在两个单向环4024-0或4024-1中的哪一个单向环上将数

据发送回至请求方代理。事务标识符由请求方代理指定,并由环站4004-N连同所请求的数据一起返回给请求方代理。

[0632] 每个条目6412也具有相关联的状态。有限状态机(FSM)更新状态。在一个实施例中,FSM按照如下进行操作。当加载队列6312检测到环形总线4024上以其为目的地的加载请求时,该加载队列6312分配可用条目6412并填充所分配的条目6412,并且FSM将所分配的条目6412的状态更新为请求方NNU。请求仲裁器6416在请求方NNU条目6412之间进行仲裁。当所分配的条目6412赢得仲裁并作为请求发送至NNU 121时,FSM将条目6412标记为待处理NNU数据。当NNU 121利用该请求的数据进行响应时,加载队列6312将数据加载到条目6412中并将条目6412标记为请求方数据环。数据仲裁器6414在请求方数据环条目6412之间进行仲裁。当条目6412赢得仲裁并且数据在环形总线4024上被发送至请求数据的环形总线4024代理时,FSM将条目6412标记为可用并且在其信用环上发出信用。

[0633] 从属储存队列6314包括耦接至请求仲裁器6426和确认仲裁器6424的条目6422的队列。在所示的实施例中,队列包括16个条目6422。每个条目6422包括用于地址、源标识符、以及与请求相关联的数据的存储。地址指定NNU 121内的由请求方环形总线4024代理(例如,核心4002)提供的要存储至的位置。地址可以指定控制/状态寄存器127、数据RAM 122或权重RAM 124内的块位置、程序存储器129内的位置、或者总线控制存储器6636内的位置。在地址指定数据RAM 122/权重RAM 124内的块位置的情况下,高位指定数据RAM 122/权重RAM 124的行,而低位(例如,6个位)指定所指定的行内的块。优选地,低位用于控制数据/权重解复用器6611/6621以选择数据/权重写入缓存器6612/6622内的适当块进行写入(参见图66)。源标识符指定请求方环形总线4024代理。

[0634] 每个条目6422也具有相关联的状态。有限状态机(FSM)更新状态。在一个实施例中,FSM按照如下进行操作。当储存队列6314检测到环形总线4024上以其为目的地的储存请求时,该储存队列6314分配可用条目6422并填充所分配的条目6422,并且FSM将所分配的条目6422的状态更新为请求方NNU。请求仲裁器6426在请求方NNU条目6422之间进行仲裁。当条目6422赢得仲裁并与条目6422的数据一起被发送至NNU 121时,FSM将条目6422标记为待处理NNU确认。当NNU 121利用确认进行响应时,储存FSM将条目6422标记为请求方确认环。确认仲裁器6424在请求方确认环条目6422之间进行仲裁。当条目6422赢得仲裁并且在确认环上将确认被发送至请求储存数据的环形总线4024代理时,FSM将条目6422标记为可用并且在其信用环上发出信用。储存队列6314还从NNU 121接收wr_busy信号,其中该wr_busy信号指示储存队列6314不从NNU 121进行请求,直到wr_busy信号不再有效为止。

[0635] 现参考图65,示出一框图,该框图更详细地示出图63的主接口0 6302-0。尽管图65示出主接口0 6302-0,但该主接口0 6302-0还代表了图63的主接口16302-1的细节,因此将被一般地称为主接口6302。主接口6302包括耦接至图63的环形总线4024的加载队列6322、储存队列6324、仲裁器6362、6364和6366、以及缓存器6352、6354和6356。图65还示出产生针对仲裁器6366的确认请求的其它确认请求器6576(例如,从属接口6301)。

[0636] 主接口6302还包括仲裁器6534(图63中未示出),其中该仲裁器6534从加载队列6322以及从其它请求器6572(例如,NNU 121和DRAM控制器4018共享环站4004-N的实施例中的DRAM控制器4018)接收请求,并将赢得仲裁请求呈现给图63的仲裁器6362。主接口6302还包括缓存器6544,其中该缓存器6544从环形总线4024接收与加载队列6312的条目6512相关

联的数据,并将其提供给NNU 121。主接口6302还包括仲裁器6554(图63中未示出),其中该仲裁器6554从储存队列6324以及从其它请求器6574(例如,NNU 121和DRAM控制器4018共享环站4004-N的实施例中的DRAM控制器4018)接收数据,并将赢得仲裁数据呈现给图63的仲裁器6364。主接口6302还包括缓存器6564,其中该缓存器6564从环形总线4024接收与储存队列6314的条目6522相关联的确认,并将其提供给NNU 121。

[0637] 加载队列6322包括耦接至仲裁器6514的条目6512的队列。在所示的实施例中,队列包括16个条目6512。每个条目6512包括用于地址和目的地标识符的存储。地址指定环形总线4024地址空间中(例如,系统存储器位置)的地址(在一个实施例中为46位)。目的地标识符指定将从中加载数据的环形总线4024代理(例如,系统存储器)。

[0638] 加载队列6322从NNU 121(例如,从DMAC 6602)接收主加载请求,以将来自环形总线4024代理(例如,系统存储器)的数据加载到数据RAM 122或权重RAM 124中。主加载请求指定目的地标识符、环形总线地址以及要使用的加载队列6322条目6512的索引。当加载队列6322从NNU 121接收到主加载请求时,加载队列6322填充索引后的条目6512,并且FSM将条目6512状态更新为请求方信用。当加载队列6322从信用环获得信用以向目的地环形总线4024代理(例如,系统存储器)发送对数据的请求时,FSM将状态更新为请求方请求环。仲裁器6514在请求方请求环条目6512之间进行仲裁(并且仲裁器6534在加载队列6322和其它请求器6572之间进行仲裁)。当条目6512被授予请求环时,在请求环上向目的地环形总线4024代理(例如,系统存储器)发送请求,并且FSM将状态更新为待处理数据环。当环形总线4024利用(例如来自系统存储器的)数据进行响应时,该数据被接收在缓存器6544中。并被提供给NNU 121(例如,提供给数据RAM 122、权重RAM 124、程序存储器129或总线控制存储器6636),并且FSM将条目6512状态更新为可用。优选地,条目6512的索引被包括在数据包内,以使得加载队列6322能够确定与数据包相关联的条目6512。优选地,加载队列6322将条目6512索引连同数据一起提供给NNU 121,以使得NNU 121能够确定数据与哪个条目6512相关并且使得NNU 121能够重新使用条目6512。

[0639] 主储存队列6324包括耦接至仲裁器6524的条目6522的队列。在所示的实施例中,队列包括16个条目6522。每个条目6522包括用于地址、目的地标识符、用于保持要储存的数据所用的数据字段以及相干标志的存储。地址指定环形总线4024地址空间(例如,系统存储器位置)中的地址。目的地标识符指定数据将被储存至其中的环形总线4024代理(例如,系统存储器)。相干标志与数据一起被发送至目的地代理。如果设置了相干标志,则该相干标志指示DRAM控制器4018监听LLC 4005并且使LLC 4005中的副本(如果其存在的话)无效。否则,DRAM控制器4018在不监听LLC 4005的情况下将数据写入系统存储器。

[0640] 储存队列6324从NNU 121(例如,从DMAC 6602)接收主储存请求,以将数据从数据RAM 122或权重RAM 124储存至环形总线4024代理(例如,系统存储器)。主储存请求指定目的地标识符、环形总线地址、要使用的储存队列6324条目6522的索引、以及要储存的数据。当储存队列6324从NNU 121接收到主储存请求时,该储存队列6324填充所分配的条目6522,并且FSM将条目6522状态更新为请求方信用。当储存队列6324从信用环获得信用以向目的地环形总线4024代理(例如,系统存储器)发送数据时,FSM将状态更新为请求方数据环。仲裁器6524在请求方数据环条目6522之间进行仲裁(并且仲裁器6554在储存队列6324和其它请求器6574之间进行仲裁)。当条目6522被授予数据环时,在数据环上向目的地环形总线

4024代理(例如,系统存储器)发送数据,并且FSM将状态更新为待处理确认环。当环形总线4024利用对数据的(例如,来自系统存储器的)确认进行响应时,在缓存器6564中接收到该确认。然后,储存队列6324向NNU 121提供确认,以向该NNU 121通知已经执行了储存,并且FSM将条目6522状态更新为可用。优选地,储存队列6324不必仲裁以向NNU 121提供确认(例如,如图66的实施例中那样,对于每个储存队列6324存在DMAC 6602)。然而,在储存队列6324必须仲裁以提供确认的实施例中,当环形总线4024利用确认进行响应时,FSM将条目6522状态更新为请求方NNU完成,并且一旦条目6522赢得仲裁并且提供对NNU 121的确认,FSM就将条目6522状态更新为可用。优选地,条目6522的索引被包括在从环形总线4024接收到的确认包内,这使得储存队列6324能够确定与确认包相关联的条目6522。储存队列6324将条目6522索引连同确认一起提供给NNU 121,以使得NNU 121能够确定数据与哪个条目6512相关联并且使得NNU 121能够重新使用条目6522。

[0641] 现参考图66,示出一框图,该框图示出图63的环站4004-N以及NNU 121的环形总线耦接实施例的一部分。示出环站4004-N的从属接口6301、主接口0 6302-0和主接口1 6302-1。图66的NNU 121的环形总线耦接实施例包括上文中详细描述的数据RAM 122、权重RAM 124、程序存储器129、定序器128、控制/状态寄存器127的实施例。NNU 121的环形总线耦接实施例在许多方面与上述的执行单元实施例类似,并且为了简洁起见,这些方面将不被重新描述。NNU 121的环形总线耦接实施例还包括图58中所述的元件,例如,移动单元5802、移动寄存器5804、复用寄存器208/705、NPU 126、复用器5806,输出单元5808、以及输出寄存器1104。NNU 121还包括第一直接存储器访问控制器(DMAC0)6602-0、第二直接存储器访问控制器(DMAC1)6602-1、总线控制存储器6636、数据解复用器6611、数据写入缓存器6612、数据RAM复用器6613、数据读取缓存器6614、数据读取缓存复用器6615、权重解复用器6621、权重写入缓存器6622、权重RAM复用器6623、权重读取缓存器6624、权重读取缓存复用器6625、从属复用器6691、主0复用器6693、以及主1复用器6692。在一个实施例中,数据解复用器6611、数据写入缓存器6612、数据读取缓存器6614、数据读取缓存复用器6615、权重解复用器6621、权重写入缓存器6622、权重读取缓存器6624和权重读取缓存复用器6625中的三个各自分别与环形总线4024的从属接口6301、主接口0 6302-0和主接口1 6302-1相关联。在一个实施例中,以数据解复用器6611、数据写入缓存器6612、数据读取缓存器6614、数据读取缓存复用器6615、权重解复用器6621、权重写入缓存器6622、权重读取缓存器6624和权重读取缓存复用器6625中的三个为一对,这三个各自分别与环形总线4024的从属接口6301、主接口0 6302-0和主接口1 6302-1相关联,从而支持双重缓存方式的数据传输。

[0642] 数据解复用器6611分别被耦接以接收来自从属接口6301、主接口06302-0和主接口1 6302-1的数据块。数据解复用器6611还分别耦接至数据写入缓存器6612,数据写入缓存器6612耦接至数据RAM复用器6613,数据RAM复用器6613耦接至数据RAM 122,数据RAM 122耦接至数据读取缓存器6614,数据读取缓存器6614分别耦接至数据读取缓存复用器6615,数据读取缓存复用器6615耦接至从属复用器6691、主0复用器6693和主1复用器6692。从属复用器6691耦接至从属接口6301,主0复用器6693耦接至主接口06302-0,并且主1复用器6692耦接至主接口1 6302-1。权重解复用器6621还分别耦接至权重写入缓存器6622,权重写入缓存器6622耦接至权重RAM复用器6623,权重RAM复用器6623耦接至权重RAM 124,权重RAM 124耦接至权重读取缓存器6624,权重读取缓存器6624分别耦接至权重读取缓存复

用器6625,权重读取缓存复用器6625耦接至从属复用器6691、主0复用器6693和主1复用器6692。数据RAM复用器6613和权重RAM复用器6623还耦接至输出寄存器1104和移动寄存器5804。数据RAM 122和权重RAM 124还分别耦接至NPU 126的移动单元5802和数据复用寄存器208以及权重复用器寄存器705。控制/状态寄存器127耦接至从属接口6301。总线控制存储器6636耦接至从属接口6301、定序器128、DMAC0 6602-0和DMAC1 6602-1。程序存储器129耦接至从属接口6301和定序器128。定序器128耦接至程序存储器129、总线控制存储器6636、NPU 126、移动单元5802和输出单元5808。DMAC0 6602-0还耦接至主接口0 6302-0,并且DMAC1 6602-1还耦接至主接口1 6302-1。

[0643] 数据写入缓存器6612、数据读取缓存器6614、权重写入缓存器6622和权重读取缓存器6624是数据RAM 122和权重RAM 124的宽度,即NPU 126阵列的宽度,这里通常被称为N。因此,例如,在一个实施例中,存在4096个NPU 126,并且数据写入缓存器6612、数据读取缓存器6614、权重写入缓存器6622和权重读取缓存器6624的宽度为4096字节,但是设想了N是除4096以外的值的其它实施例。数据RAM 122和权重RAM 124一次被写入整个N字行。输出寄存器1104、移动寄存器5804和数据写入缓存器6612经由数据RAM复用器6613写入数据RAM 122,其中该数据RAM复用器6613选择其中之一以将一行字写入数据RAM 122。输出寄存器1104、移动寄存器5804和权重写入缓存器6622经由权重RAM复用器6623写入权重RAM 124,其中该权重RAM复用器6623选择其中之一以向权重RAM 124写入一行字。控制逻辑(未示出)控制数据RAM复用器6613在数据写入缓存器6612、移动寄存器5804和输出寄存器1104之间进行仲裁以存取数据RAM 122,并且控制权重RAM复用器6623在权重写入缓存器6622、移动寄存器5804和输出寄存器1104之间进行仲裁以访问权重RAM 124。数据RAM 122和权重RAM 124还一次读取整个N字行。NPU 126、移动单元5802和数据读取缓存器6614从数据RAM 122读取一行字。NPU 126、移动单元5802和权重读取缓存器6624从权重RAM 124读取一行字。控制逻辑还控制NPU126(数据复用器寄存器208和权重复用器寄存器705)、移动单元5802和数据读取缓存器6614以确定它们中的哪一个(如果有的话)读取由数据RAM 122输出的一行字。在一个实施例中,针对图34所述的微操作3418可以包括控制数据RAM复用器6613、权重RAM复用器662、NPU 126、移动单元5802、移动寄存器5804、输出寄存器1104、数据读取缓存器6614和权重读取缓存器6624的控制逻辑信号中的至少一些。

[0644] 数据写入缓存器6612、数据读取缓存器6614、权重写入缓存器6622和权重读取缓存器6624可在块大小对齐的块中寻址。优选地,数据写入缓存器6612、数据读取缓存器6614、权重写入缓存器6622和权重读取缓存器6624的块大小与环形总线4024数据子环的宽度相匹配。这使得环形总线4024适于对数据/权重RAM 122/124进行如下的读取/写入。通常,环形总线4024对数据写入缓存器6612的每个块执行块大小的写入,并且一旦数据写入缓存器6612的所有块都被填充,数据写入缓存器6612就将其N字内容写入数据RAM 122的整行。同样,环形总线4024对权重写入缓存器6622的每个块执行块大小的写入,并且一旦权重写入缓存器6622的所有块都被填充,权重写入缓存器6622就将其N字内容写入权重RAM 124的整行。相反,从数据RAM 122将N字行读取至数据读取缓存器6614中;然后环形总线4024从数据读取缓存器6614的每个块执行块大小的读取。同样,从权重RAM 124将N字行读取至权重读取缓存器6624中;然后环形总线4024从权重读取缓存器6624的每个块执行块大小的读取。尽管数据RAM 122和权重RAM 124在图66中表现为双端口存储器,但它们优选为单端口

存储器,使得单个数据RAM 122端口被数据RAM复用器6613和数据读取缓存器6614所共享,并且单个权重RAM 124端口被权重RAM复用器6623和权重读取缓存器6624所共享。因此,整行读取/写入布置的优点在于,其通过具有单个端口使得数据RAM 122和权重RAM 124更小(在一个实施例中,权重RAM 124为8MB并且数据RAM 122为64KB),而环形总线4024相对于数据RAM 122和权重RAM 124的写入和读取消耗的带宽比写入单独块时消耗的带宽更少,因此为NPU 126、输出寄存器1104、移动寄存器5804和移动单元5802解放出更多的带宽,以进行N个字宽的行的访问。

[0645] 控制/状态寄存器127被提供至从属接口6301。从属复用器6691接收与从属接口6301相关联的数据读取缓存复用器6615的输出、以及与从属接口6301相关联的权重读取缓存复用器6625的输出,并选择其中之一以提供给从属接口6301。以这种方式,从属加载队列6312接收用于对由从属接口6301对控制/状态寄存器127、数据RAM 122或权重RAM 124作出的加载请求进行响应的数据。主0复用器6693接收与主接口0 6302-0相关联的数据读取缓存复用器6615的输出、以及与主接口0 6302-0相关联的权重读取缓存复用器6625的输出,并选择其中之一以提供给主接口0 6302-0。以这种方式,主接口0 6302-0接收用于对由主接口0 6302-0储存队列6324作出的储存请求进行响应的数据。主1复用器6692接收与主接口1 6302-1相关联的数据读取缓存复用器6615的输出、以及与主接口1 6302-1相关联的权重读取缓存复用器6625的输出,并选择其中之一以提供给主接口1 6302-1。以这种方式,主接口1 6302-1接收用于对由主接口1 6302-1储存队列6324作出的储存请求进行响应的数据。如果从属接口6301加载队列6312请求从数据RAM122进行读取,则从属复用器6691选择与从属接口6301相关联的数据读取缓存复用器6615的输出;而如果从属接口6301加载队列6312请求从权重RAM 124进行读取,则从属复用器6691选择与从属接口6301相关联的权重读取缓存复用器6625的输出。同样,如果主接口0 6302-0储存队列请求从数据RAM 122读取数据,则主0复用器6693选择与主接口0 6302-0相关联的数据读取缓存复用器6615的输出;而如果主接口0 6302-0储存队列请求从权重RAM 124读取数据,则主0复用器6693选择与主接口0 6302-0相关联的权重读取缓存复用器6625的输出。最后,如果主接口1 6302-1储存队列请求从数据RAM 122读取数据,则主1复用器6692选择与主接口1 6302-1相关联的数据读取缓存复用器6615的输出;而如果主接口1 6302-1储存队列请求从权重RAM 124读取数据,则主1复用器6692选择与主接口1 6302-1相关联的权重读取缓存复用器6625的输出。因此,环形总线4024代理(例如,核心4002)可以经由从属接口6301加载队列6312从控制/状态寄存器127、数据RAM 122或权重RAM 124进行读取。此外,环形总线4024代理(例如,核心4002)可以经由从属接口6301储存队列6314向控制/状态寄存器127、数据RAM 122、权重RAM 124、程序存储器129或总线控制存储器6636进行写入。更具体地,核心4002可以将程序(例如,执行完全连接、卷积、池化、LSTM或其它循环神经网络层计算的程序)写入程序存储器129,然后写入控制/状态寄存器127以开始程序。此外,核心4002可以将控制字写入总线控制存储器6636,以使DMAC 6602在数据RAM 122或权重RAM 124与环形总线4024代理(例如,系统存储器或LLC 4005)之间执行DMA操作。定序器128还可以将控制字写入总线控制存储器6636,以使DMAC 6602在数据RAM 122或权重RAM 124与环形总线4024代理之间执行DMA操作。最后,如以下更详细所述,DMAC 6602可以执行DMA操作以执行环形总线4024代理(例如,系统存储器或LLC 4005)与数据/权重RAM 122/124之间的传输。

[0646] 从属接口6301、主接口0 6302-0和主接口1 6302-1彼此耦接以向其各自的数据解复用器6611和权重解复用器6621提供数据块。仲裁逻辑(未示出)在输出寄存器1104、移动寄存器5804和从属接口6301、主接口0 6302-0和主接口16302-1、数据写入缓存器6612之间为了对数据RAM 122的访问而进行仲裁,并且在输出寄存器1104、移动寄存器5804和从属接口6301、主接口0 6302-0和主接口1 6302-1、权重写入缓存器6622之间为了对权重RAM 124的访问而进行仲裁。在一个实施例中,写入缓存器6612/6622优先于输出寄存器1104和移动寄存器5804,并且从属接口6301优先于主接口6302。在一个实施例中,每个数据解复用器6611具有耦接至各自的数据写入缓存器6612的64个块的64个输出(每个输出优选为64字节)。数据解复用器6611在耦接至数据写入缓存器6612的适当块的输出上提供所接收到的块。同样,每个权重解复用器6621具有耦接至各自的权重写入缓存器6622的64个块的64个输出(每个输出优选为64字节)。权重解复用器6621在耦接至权重写入缓存器6622的适当块的输出上提供所接收到的块。

[0647] 当从属储存队列6314向其数据/权重解复用器6611/6621提供数据块时,该从属储存队列6314还向数据/权重解复用器6611/6621提供要进行写入的数据/权重写入缓存器6612/6622的适当块的地址作为控制输入。块地址是保持在条目6422中的地址的低六位,其由生成从属存储事务的环形总线4024代理(例如核心4002)指定。相反,当加载储存队列6312从其数据/权重读取缓存复用器6615/6625请求数据块时,该加载储存队列6312还向数据/权重读取缓存复用器6615/6625提供要读取的数据/权重读取缓存器6614/6624的适当块的地址作为控制输入。块地址是保持于条目6412中的地址的低六位,其中该条目6412由产生从属加载事务的环形总线4024代理(例如,核心4002)指定。优选地,核心4002可以经由(例如,到预定环形总线4024地址的)从属接口6301来执行从属储存事务,以使NNU 121将数据/权重写入缓存器6612/6622的内容写入数据/重量RAM 122/124;相反,核心4002可以经由(例如,到预定环形总线4024地址的)从属接口6301来执行从属储存事务,以使NNU 121将数据/权重RAM 122/124的行读取到数据/权重读取缓存器6614/6624中。

[0648] 当主接口6302加载队列6322/6332向其数据/权重解复用器6611/6621提供数据块时,该主接口6302加载队列6322/6332还将条目6512的索引提供至向加载队列6322/6332发出加载请求的相应DMAC 6602。为了将整个4KB的数据从系统存储器传输至数据/权重RAM122/124的列,DMAC 6602必须产生64个主加载请求至加载队列6322/6332。DMAC 6602逻辑上将64个主加载请求分成四组,每组包含十六个请求。DMAC 6602将组内的16个请求传输至加载队列6322/6322的相应16个条目6512。DMAC 6602维持与每个条目6512索引相关联的状态。该状态指示四个组中的当前正使用条目来加载数据块的那一组。因此,如以下更详细描述,当DMAC 6602从加载队列6322/6322接收条目6512索引时,DMAC 6602的逻辑通过将组号与索引串联来构造块地址,并将构造的块地址作为控制输入提供给数据/权重解复用器6611/6621。

[0649] 相反,当主接口6302储存队列6324/6334从其数据/权重缓存复用器6615/6625请求数据块时,该主接口6302加载队列6324/6334还将条目6522的索引提供至向储存队列6322/6332发出储存请求的相应DMAC 6602。为了将整个4KB的数据从数据/权重RAM122/124的行传输至系统存储器,DMAC 6602必须产生64个主储存请求至储存队列6324/6334。DMAC 6602逻辑上将64个储存请求分成四组,每组包含十六个请求。DMAC 6602向储存队列6324/

6334的相应16个条目6522进行组内的十六个请求。DMAC 6602维持与每个条目6522索引相关联的状态。该状态指示四个组中当前正使用条目来储存数据块的那一组。因此,如以下更详细所述,当DMAC 6602从储存队列6324/6334接收条目6522索引时,DMAC 6602的逻辑通过将组号与索引串联来构造块地址,并将构造的块地址作为控制输入提供给数据/权重读取缓存复用器6615/6625,。

[0650] 现参考图67,示出一框图,该框图示出图66的DMAC 6602。DMAC 6602耦接至图66的环站4004-N。更具体地,图67示出DMAC 6602的执行主加载操作(即,从环形总线4024代理(例如,从系统存储器或LLC 4005)至NNU 121(例如,至数据/权重RAM 122/124)的数据传输)的部分。作为说明性示例,将针对图67和图68来描述主加载操作,其中DMAC 6602将整行数据从系统存储器传输至权重RAM 124。在该示例中,传输是4KB,并且块大小是64字节,使得DMAC 6602执行从系统存储器到写入缓存器6622的64个块传输,然后使写入缓存器6622将其内容写入权重RAM 124。然而,应当理解,当从LLC 4005和/或向数据RAM 122传输数据时,DMAC 6602执行类似的操作。此外,应当理解,DMAC 6602-0和DMAC 6602-1都可以执行类似的操作,使得事务可以在环形总线4024的两个方向4204-0和4204-1上执行。行大小的主加载操作可以是NNU 121所请求的较大的DMA操作的一部分。例如,定序器128从程序存储器129拾取并执行的程序可以将控制字写入总线控制存储器6636,该总线控制存储器6636请求将500行从系统存储器传输至权重RAM 124。在这种情况下,DMAC 6602将这里描述的4KB的主加载操作执行500次。优选地,DMAC 6602包括从总线控制存储器6636接收控制字的另一部分,并且向图67中所述的DMAC 6602的该部分进行4KB大小的主加载请求。DMAC 6602的该另一部分还向程序通知较大的DMA操作已完成。

[0651] DMAC 6602接收请求主加载操作的nnuload_req 6712信号。nnuload_reqaddr 6714信号指定要加载的4KB的数据的环形总线4024地址。优选地,地址在4KB边界上对齐,并且字节数隐含为4KB。nnuload_ramrow 6715信号指定数据要被加载至的权重RAM 124行。

[0652] 响应于请求,DMAC 6602设置到环站4004-N(即到主接口6302加载队列6322/6332)的busload_req 6722、busload_reqidx 6724和busload_reqaddr 6726信号,以如以上针对图65所述地请求来自系统存储器的主加载事务。busload_reqidx 6724指定要用于执行主加载事务的条目6512的索引。busload_reqaddr 6726指定要读取数据的系统存储器位置的环形总线4024地址。DMAC 6602进行64个这样的64字节主加载事务请求,各请求针对被请求行的64个块中的不同的一个块。如上所述,在N(即数据RAM 122、权重RAM 124和NPU 126阵列的宽度)不同于4096以及/或者块大小不同于64的实施例中,DMAC 6602必须执行的主加载事务的数量可能会有所不同。图68更详细地描述了各个块请求的处理。

[0653] 环站4004-N利用针对各数据块的busload_datavalid 6732、busload_dataidx 6734和busload_data 6736信号进行响应。busload_dataidx 6734指定用于执行主加载事务并且与busload_data 6736信号上提供的数据块相关联的条目6512的索引。应当理解,如以下针对图68更详细所述,环形总线4024可以按照与加载队列6322/6332请求各个数据块的顺序不同的顺序返回这些数据块。

[0654] 作为响应,DMAC 6602发送nnuload_blkdatavalid 6742、nnuload_blkaddr 6744和nnuload_blkdata 6746信号。nnuload_blkdata 6746信号提供由环站4004-N返回的数据块。特别地以响应于nnuload_blkdatavalid 6742的设置,nnuload_blkaddr 6744指定权重

写入缓存器6622内的数据块的地址,并且用于控制权重解复用器6621和权重写入缓存器6622以在其中写入正确的数据块。

[0655] 一旦全部64个数据块都已从系统存储器返回并被写入权重写入缓存器6622中,即一旦权重写入缓存器6622满了,DMAC 6602就声明nnuload_ramwrite 6748以将权重写入缓存器6622的内容写入由nnuload_ramrow 6715指定的权重RAM 124的行。

[0656] 现在参考图68,示出一框图,该框图示出图67的DMAC 6602的块状态6802以及使用这些块状态6802的块状态机6804。如以下更详细所述,每个块状态6802指定数据块的状态,即:待处理(P)、被请求(Q)或就绪(R)。块状态6802被示出为4行16列的阵列,总共64个块状态6802对应于图67中所述的主加载操作的64个数据块,这些块状态被编号为0至63,对应于权重写入缓存器6622(或数据写入缓存器6612)内的数据块的地址。阵列的16列各自与加载队列6322/6332的16个条目6512(即,条目索引0至15)其中之一相关联。阵列的每行与不同组的16个数据块相对应。更具体地,组0包括数据块0至15,组1包括数据块16至31,组2包括数据块32至47,而组3包括数据块48至63。优选地,块状态6802被保持在DMAC 6602的触发器或其它状态存储中。

[0657] 在重设时,全部64个块状态6802置于R状态。当DMAC 6602(例如,经由nnuload_req 6712信号)接收到新的主加载请求时,状态机6804将所有块状态6802转换为P状态,以指示从系统存储器加载相应数据块的请求为待处理。然后状态机6804如下所述在64个待处理块请求中进行仲裁。状态机6804授予数据块向环站4004-N发送其加载请求的许可,并在如下情况下将数据块的块状态6802转换为Q状态:(1)当前块处于P状态,(2)具有较小块地址和相同加载队列索引的所有块(即,位于其列中当前块上方的所有块,也即位于具有较小组编号的组内的所有块)处于R状态(即,它们已经从环形总线4024接收了数据),以及(3)对于具有比当前块更低的加载队列索引的任何待处理块(即,当前块的列的左侧列内的任何P状态块),其中具有与当前块相同的加载队列索引但具有较小的块地址的块(即,位于其列中的当前块上方的块,也即具有较小组编号的组中的块)处于Q状态。有利地,这提供了对加载队列6322/6332条目6512的高效使用,并且可以以高度利用的方式实现对系统存储器的访问。当从环形总线4024接收回数据块并将其写入加权写入缓存器6622时,状态机6804将其块状态6802转换为R。一旦所有64个数据块的块状态6802都处于R状态,DMAC 6602就设置nnuload_ramwrite 6748。

[0658] 现参考图69,示出一框图,该框图示出图66的DMAC 6602。DMAC 6602耦接至图66的环站4004-N。更具体地,图69示出DMAC 6602的执行主储存操作(即,从NNU 121(例如从数据/权重RAM 122/124)至环形总线4024代理(例如至系统存储器或LLC 4005)的数据传输)的部分。作为说明性示例,将针对图69和图70来描述主储存操作,其中DMAC 6602将整行数据从数据RAM 122传输至系统存储器。在该示例中,传输为4KB,并且块大小为64字节,使得在DMAC 6602使一行数据从数据RAM 122读取至读取缓存器6614中之后,DMAC 6602执行从读取缓存器6622向系统存储器的64个块的传输。然而,应当理解,当向LLC 4005和/或从数据RAM 124传输数据时,DMAC 6602执行类似的操作。此外,应当理解,DMAC 6602-0和DMAC 6602-1这两者都可以执行类似的操作,使得事务可以在环形总线4024的两个方向4204-0和4204-1上执行。行大小的主储存操作可以是NNU 121所请求的较大DMA操作的一部分。例如,定序器128从程序存储器129拾取并执行的程序可以将控制字写入总线控制存储器6636,该

总线控制存储器6636请求将500行从数据RAM 122传输至系统存储器。在这种情况下,DMAC 6602将执行这里描述的500个4KB的主储存操作。优选地,DMAC 6602包括从总线控制存储器6636接收控制字的另一部分,并且向图69中所述的DMAC 6602的该部分进行4KB大小的主储存请求。DMAC 6602的该另一部分还向程序通知较大的DMA操作已完成。

[0659] DMAC 6602接收请求主储存操作的nnustore_req 6912信号。nnustore_reqaddr 6914信号指定要储存的4KB数据的环形总线4024地址,并且nnustore_ramrow 6915信号指定来自数据RAM 122中要写入系统存储器的一行数据。优选地,地址在4KB边界上对齐,并且字节数隐含为4KB。

[0660] 响应于请求,DMAC 6602设置nnustore_ramread6916信号以使由nnustore_ramrow6915信号指定的数据RAM 122的行被读入数据读取缓存器6614中。可选地,DMAC 6602的另一部分在向主储存操作部分进行请求之前使数据RAM122的行被读入读取缓存器6614。

[0661] 在将行读入数据读取缓存器6614之后,DMAC 6602设置nnustore_blkaddr6919以指定读取缓存器6614内的数据块的地址,并且在nnustore_blkdata 6919信号上接收指定的数据块。更具体地,如针对图70更详细所述,DMAC 6602在nnustore_blkaddr 6919上设置64个不同的块地址以从读取缓存器6614读取所有64个数据块,从而执行主储存操作。nnustore_blkaddr 6918用于控制数据读取缓存复用器6615以从其读取正确的数据块。

[0662] 对于从读取缓存器6614接收到的各数据块,DMAC 6602设置到环站4004-N(即,到主接口6302加载队列6322/6332)的busstore_req 6922、busstore_reqidx 6924、busstore_reqaddr 6926和busstore_reqdata 6928信号以如以上针对图65所述地请求至系统存储器的主储存事务。busstore_reqidx 6924指定要用于执行主储存事务的条目6522的索引。busstore_reqaddr 6926指定要写入数据的系统存储器位置的环形总线4024地址(提供于busstore_reqdata 6928上)。也就是说,DMAC 6602进行64个这样的64字节主储存事务请求,各自针对从数据RAM 122读取的行的64个块中的不同一个块。如上所述,在N(即数据RAM 122、权重RAM 124和NPU 126阵列的宽度)不同于4096以及/或者块大小不同于64的实施例中,DMAC 6602必须执行的主储存事务的数量可能会有所不同。图70更详细地描述了各个块请求的处理。

[0663] 环站4004-N利用针对各写入数据块的busstore_datadone 6932和busstore_dataidx 6934信号进行响应。busstore_dataidx 6934指定用于执行主储存事务并且与从环形总线4024接收到的针对先前提供于busstore_reqdata 6928信号上的数据块的确认相关联的条目6522的索引。应当理解,如以下针对图70更详细所述,环形总线4024可以按照与储存队列6324/6334发送各个数据块的顺序不同的顺序返回这些数据块。

[0664] 一旦从系统存储器返回了所有64个数据块的确认,DMAC 6602就设置nnustore_datadone 6942以表示主储存操作已经完成。

[0665] 现在参考图70,示出一框图,该框图示出图69的DMAC 6602的块状态7002以及使用这些块状态7002的块状态机7004。如以下更详细所述,每个块状态7002指定数据块的状态,即:待处理(P)、被请求(Q)或就绪(R)。块状态7002被示出为4行16列的阵列,总共64个块状态7002对应于图69中所述的主储存操作的64个数据块,这些块状态被编号为0至63,对应于数据读取缓存器6614(或权重读取缓存器6624)内的数据块的地址。阵列的16列各自与储存

队列6324/6334的16个条目6522(即,条目索引0至15)其中之一相关联。阵列的每行与不同组的16个数据块相对应。更具体地,组0包括数据块0至15,组1包括数据块16至31,组2包括数据块32至47,而组3包括数据块48至63。优选地,块状态7002被保持在DMAC 6602的触发器或其它状态存储中。

[0666] 在重设时,全部64个块状态7002置于R状态。当DMAC 6602(例如,经由nnustore_req 6912信号)接收到新的主储存请求时,状态机7004将所有块状态7002转换为P状态,以指示将相应数据块储存至系统存储器的请求为待处理。然后状态机7004如下所述在64个待处理块请求中进行仲裁。状态机7004授予数据块向环站4004-N发送其储存请求的许可,并在如下情况下将数据块的块状态7002转换为Q状态:(1)当前块处于P状态,(2)具有较小块地址和相同储存队列索引的所有块(即,位于其列中当前块上方的所有块,也即位于具有较小组编号的组内的所有块)处于R状态(即,它们已经向环形总线4024发送了数据),以及(3)对于具有比当前块更低的储存队列索引的任何待处理块(即,当前块的列的左侧列内的任何P状态块),其中具有与当前块相同的储存队列索引但具有较小的块地址的块(即,位于其列中的当前块上方的块,也即位于具有较小组编号的组中的块)处于Q状态。有利地,这提供了对储存队列6324/6334条目6522的高效使用,并且可以以高度利用的方式实现对系统存储器的访问。当从环形总线4024接收回确认时,状态机7004将其块状态7002转换为R。一旦所有64个数据块的块状态7002都处于R状态,DMAC 6602就设置nnustore_datadone 6942。

[0667] 现参考图71,示出一框图,该框图示出基址寄存器7198和DMA控制字(DCW)7104。在一个实施例中,如图71所示,NNU 121包括与主接口0 6302-0加载队列6322、主接口0 6302-0储存队列6324、主接口1 6302-1加载队列6332以及主接口1 6302-1储存队列6334相关联的四个基址寄存器7198。DMAC 6602使用基址寄存器7198来构造环形总线4024地址。优选地,每个基址寄存器7198是22位。优选地,NNU 121的装置驱动器分配系统存储器的四个区域,其中这四个区域各自的大小为16MB且为16MB对齐,并且执行经由从属接口6301的储存以将四个系统存储器区域的基址(例如,46位环形总线4024地址的高22个位)写入四个基址寄存器7198的相应基址寄存器7198中。当加载/储存队列6322/6324/6332/6334在环形总线4024上产生事务时,其通过将适当的基址寄存器7198的内容放置为高22个位来构造环形总线4024地址。低6个位为零,这是因为访问是以64字节块为单位进行的。当DMAC 6602向主接口6302发出加载/储存请求时,中间18位由DMAC 6602提供。在上述的数据解复用器6611、数据写入缓存器6612、数据读取缓存器6614、数据读取缓存复用器6615、权重解复用器6621、权重写入缓存器6622、权重写入缓存器6624以及权重读取缓存复用器6625各自成对分别与环形总线4024的主接口06302-0和主接口1 6302-1相关联以支持双重缓存方式的数据传输的备选实施例中,存在一对基址寄存器7198,其与主接口0 6302-0加载队列6322、主接口0 6302-0储存队列6324,主接口1 6302-1加载队列6332和主接口1 6302-1储存队列6334中的各队列相关联。

[0668] 图71还示出DCW 7104。在一个实施例中,DCW 7104包括环形总线地址7112;数据/权重RAM 122/124地址7114;NNU存储器空间指示符7116;方向指示符7118;计数7122;相干指示符7124;中断标志7126;以及等待标记7128。如上所述,核心4002可以执行从属储存操作以将DCW 7104写入总线控制存储器6636,以使DMAC 6602执行DMA操作;并且程序存储器129中的程序可以执行将DCW 7104写入总线控制存储器6636的指令,以使DMAC 6602执行

DMA操作。

[0669] 环形总线地址7112指定要传输的数据的环形总线4024地址空间中的位置(例如,系统存储器地址)。数据/权重RAM 122/124地址7114指定要读取或写入的数据/权重RAM 122/124中的行。NNU存储器空间指示符7116指定数据RAM 122或权重RAM 124是否是DMA操作的目标/源。方向指示符7118指示DMA操作是从数据/权重RAM 122/124到环形总线4024还是从环形总线4024到数据/权重RAM 122/124。计数7122指定要传输的数据/权重RAM 122/124的行数。相干指示符7124指定LLC 4005是否应监听环形总线4024地址。如果环站4004-N对LLC 4005中发现的环状总线4024地址执行主加载事务,则将LLC 4005中的相应数据返回给NNU 121;否则,从系统存储器返回数据,但不将该数据放入LLC 4005中。中断标志7126指定在完成DMA操作时从属接口6301是否将向核心4002发送中断。

[0670] 等待标记7128指定与由DCW 7104指定的DMA操作相关联的值。程序中在用于将DCW 7104写入总线控制存储器6636的指令之后的指令可以指定相同的标记值,在这种情况下,定序器128将使后续指令等待执行,直到与等待标记值相关联的DMA操作完成为止。在一个实施例中,在等待与等待标记值相关联的DMA操作完成的同时,NNU 121的未使用部分被置于较低功率模式。例如,NPU 126可以移除其时钟,直到DMAC 6602指示DMA操作完成为止。

[0671] 在一个实施例中,DCW 7104还包括将DCW 7104链接到总线控制存储器6636中的另一DCW7104的链接字段,从而使编程者能够有效地创建DMA操作程序。

[0672] 现参考图72,示出一框图,该框图示出NNU 121的环形总线耦接实施例。图72在一些方面与图34相同,并且附图标记相同的元件是相同的。如同图34,图72示出NNU 121从多个源接收微操作以提供其流水线的的能力。然而,在图72的实施例中,NNU 121如图62中那样经由环形总线4024耦接至核心4002,现将描述差异。

[0673] 在图72的实施例中,复用器3402从五个不同的源接收微操作。复用器3402将选择的微操作3418提供给NPU 126流水线级3401、数据RAM 122和权重RAM 124、移动单元5802和输出单元5808,以对其进行控制,如上所述。如针对图34所述,第一个源是产生微操作3416的定序器128。第二个源是图34的解码器3404的修改后版本,用于从核心4002所储存的从属接口6301储存队列6314接收储存请求的数据块。如以上针对图34所述,数据块可以包括与从MTNN指令1400或MFNN指令1500转译来的微指令类似的信息。解码器3404对数据块进行解码并作为响应而产生微操作3412。一个示例是响应于从从属接口6301储存队列6314接收到的将数据写入数据/权重RAM 122/124的请求、或者响应于从从属接口6301加载队列6312接收到的从数据/权重RAM 122/124读取数据的请求而产生的微操作3412。第三个源是来自核心4002所储存的从属接口6301储存队列6314的储存请求的直接数据块,其中核心4002包括NNU 121直接执行的微操作3414,如以上针对图34所述。优选地,核心4002储存至环形总线4024地址空间中的不同存储器映射地址,以使得解码器3404能够区分第二微操作源和第三微操作源。第四个源是由DMAC 6602产生的微操作7217。第五个源是空运算微操作7219,其中响应于该空运算微操作7219,NNU 121保持其状态。

[0674] 在一个实施例中,五个源具有解码器3404所执行的优先方案,其中直接微操作3414具有最高优先级;由解码器3404响应于从属接口6301的从属储存操作而产生的微操作3412具有第二高优先级;由DMAC 6602产生的微操作7217具有次高优先级;由定序器128产生的微操作3416具有次高的优先级;并且空运算微操作是默认的(即最低优先级),复用器

3402在没有其它源请求时所选择的源。根据一个实施例,当DMAC 6602或从属接口6301需要存取数据RAM 122或权重RAM 124时,其优先于运行在定序器128上的程序,并且解码器3404使定序器128暂停直到DMAC 6602和从属接口6301已完成其存取为止。

[0675] 虽然这里描述了本发明的各种实施例,但这些实施例是通过示例而非限制的方式呈现。对于相关计算机领域的技术人员将是明显的,可以在不偏离本发明的范围的情况下进行形式和细节方面的各种改变。例如,软件可以例如支持本发明所述的设备和方法的功能、制造、建模、仿真、描述以及/或者测试等。这可以使用一般编程语言(例如,C、C++)、包含Verilog HDL、VHDL等的硬件描述语言(HDL),或其它可用程序来实现。这样的软件可以设置于任何已知的计算机可用介质,诸如磁带、半导体、磁盘、光盘(例如,CD-ROM、DVD-ROM等)、网络、有线或其它通信介质等。这里描述的设备和方法的实施例可被包含于诸如处理器核心(例如,采用HDL体现或指定)等的半导体知识产权核心中,并且通过集成电路的制作而被转换为硬件。此外,这里描述的设备和方法还可体现为硬件和软件的组合。因此,本发明不应以这里描述的任何典型实施例来进行限制,而应仅根据以下权利要求和其等同项来进行限定。具体地,本发明可以实现在可用于通用计算机的处理器装置内。最后,本领域技术人员应当理解,他们可以容易地使用所公开的概念和具体实施例作为基础,来设计并修改其它结构以在不偏离所附权利要求所限定的本发明的范围的情况下实现本发明的相同目的。

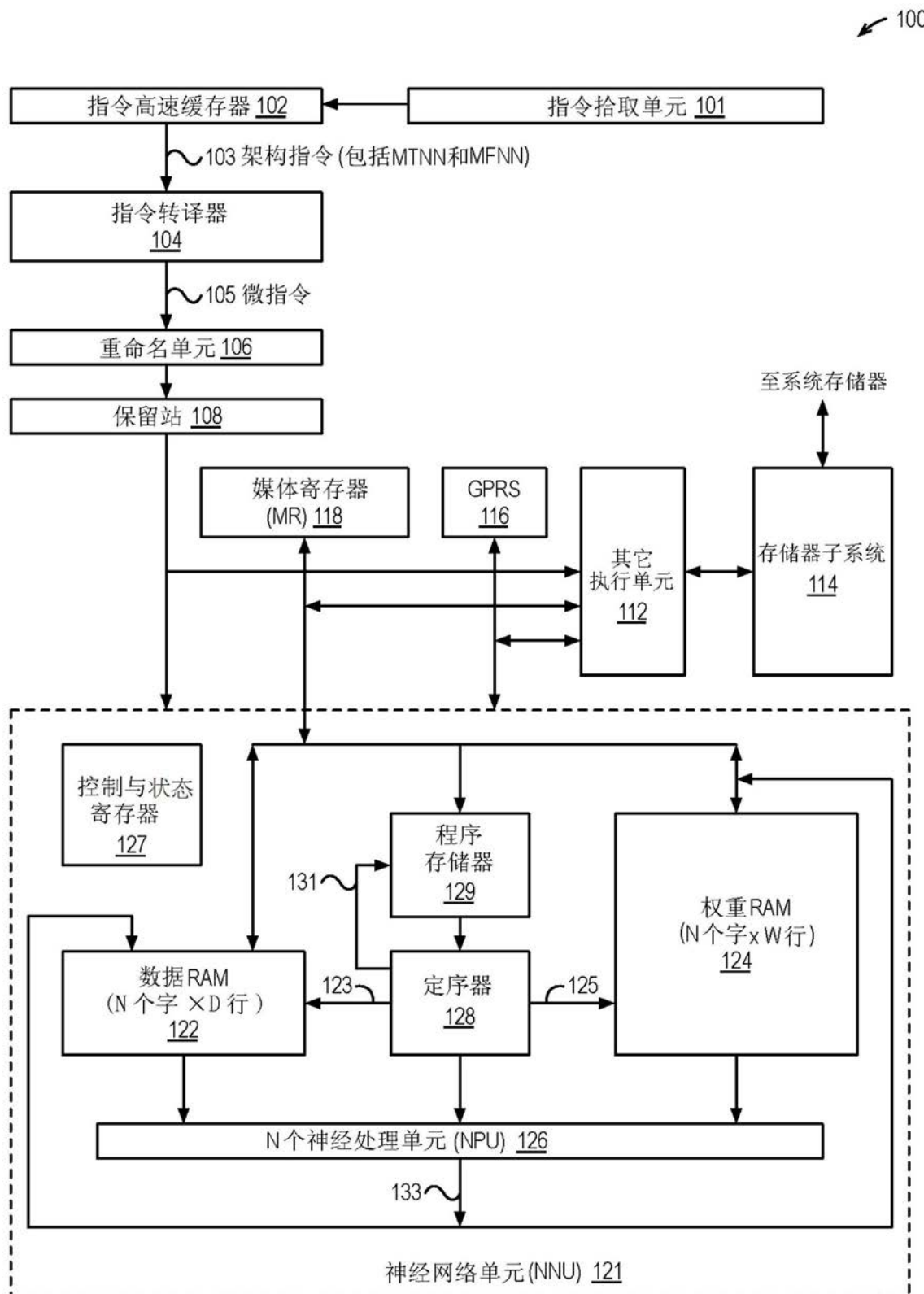


图1

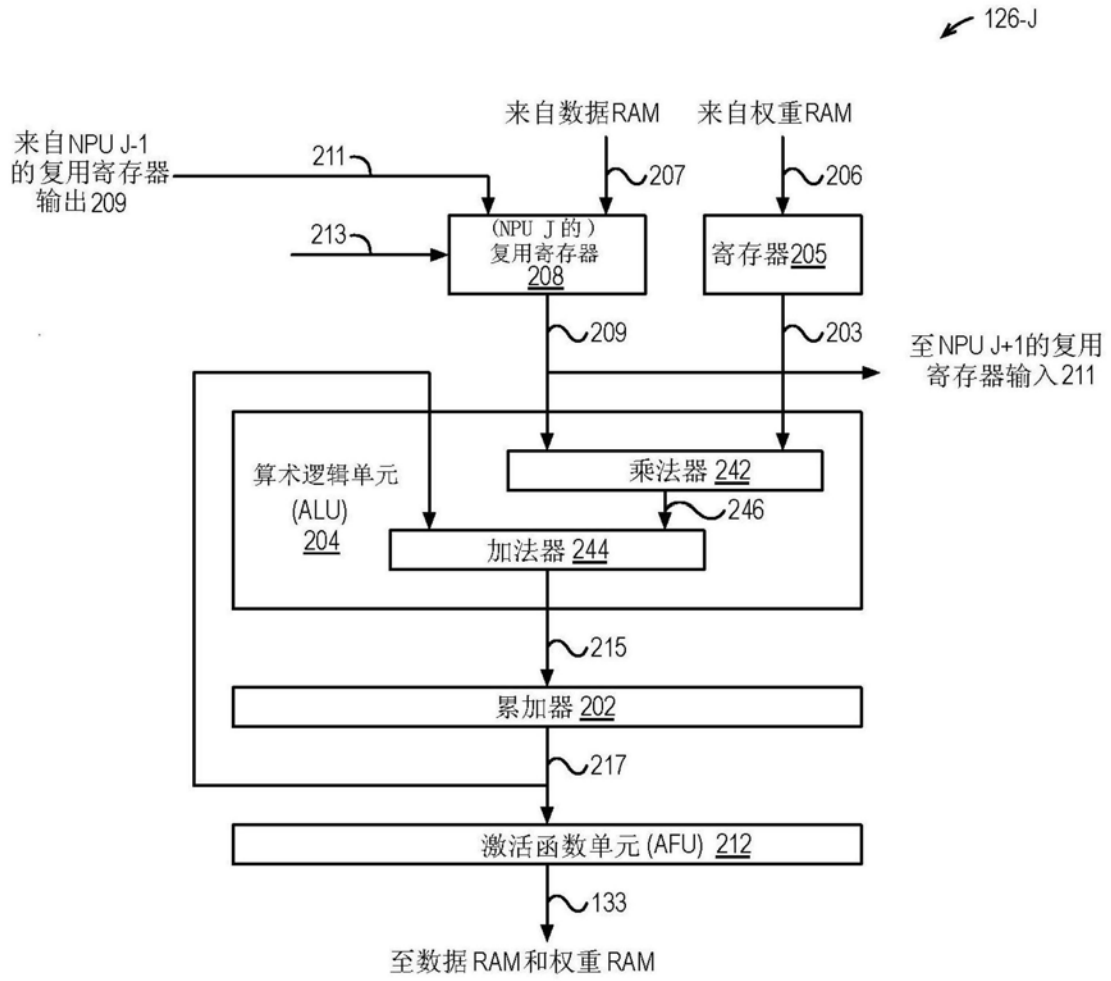


图2

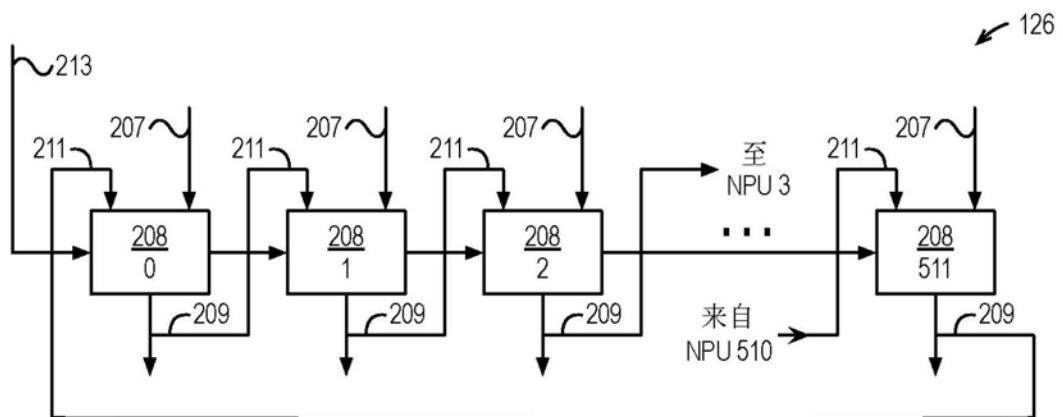


图3

地址	指令	时钟
0	INITIALIZE NPU	1
1	MULT-ACCUM DR ROW 17, WR ROW 0	1
2	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511	511
3	ACTIVATION FUNCTION	1
4	WRITE AFU OUTPUT TO DR ROW 16	1

图4

CLK	NPU 0	NPU 1	...	NPU 511
0	ACC = 0	ACC = 0		ACC = 0
1	ACC += DR:17,0 * WR:0,0	ACC += DR:17,1 * WR:0,1		ACC += DR:17,511 * WR:0,511
2	ACC += MREG:511 * WR:1,0	ACC += MREG:0 * WR:1,1		ACC += MREG:510 * WR:1,511
3	ACC += MREG:511 * WR:2,0	ACC += MREG:0 * WR:2,1		ACC += MREG:510 * WR:2,511
			...	
512	ACC += MREG:511 * WR:511,0	ACC += MREG:0 * WR:511,1		ACC += MREG:510 * WR:511,511
513	AFU (ACC)	AFU (ACC)		AFU (ACC)
514	WRITE AFU OUTP TO DR:16,0	WRITE AFU OUTP TO DR:16,1		WRITE AFU OUTP TO DR:16,511

图5

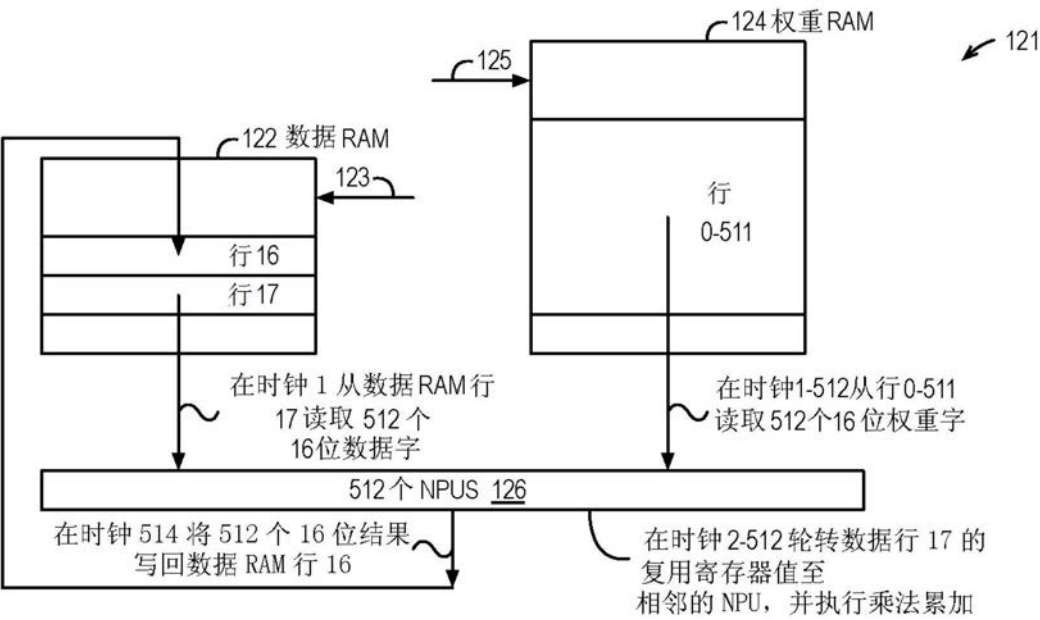


图6A

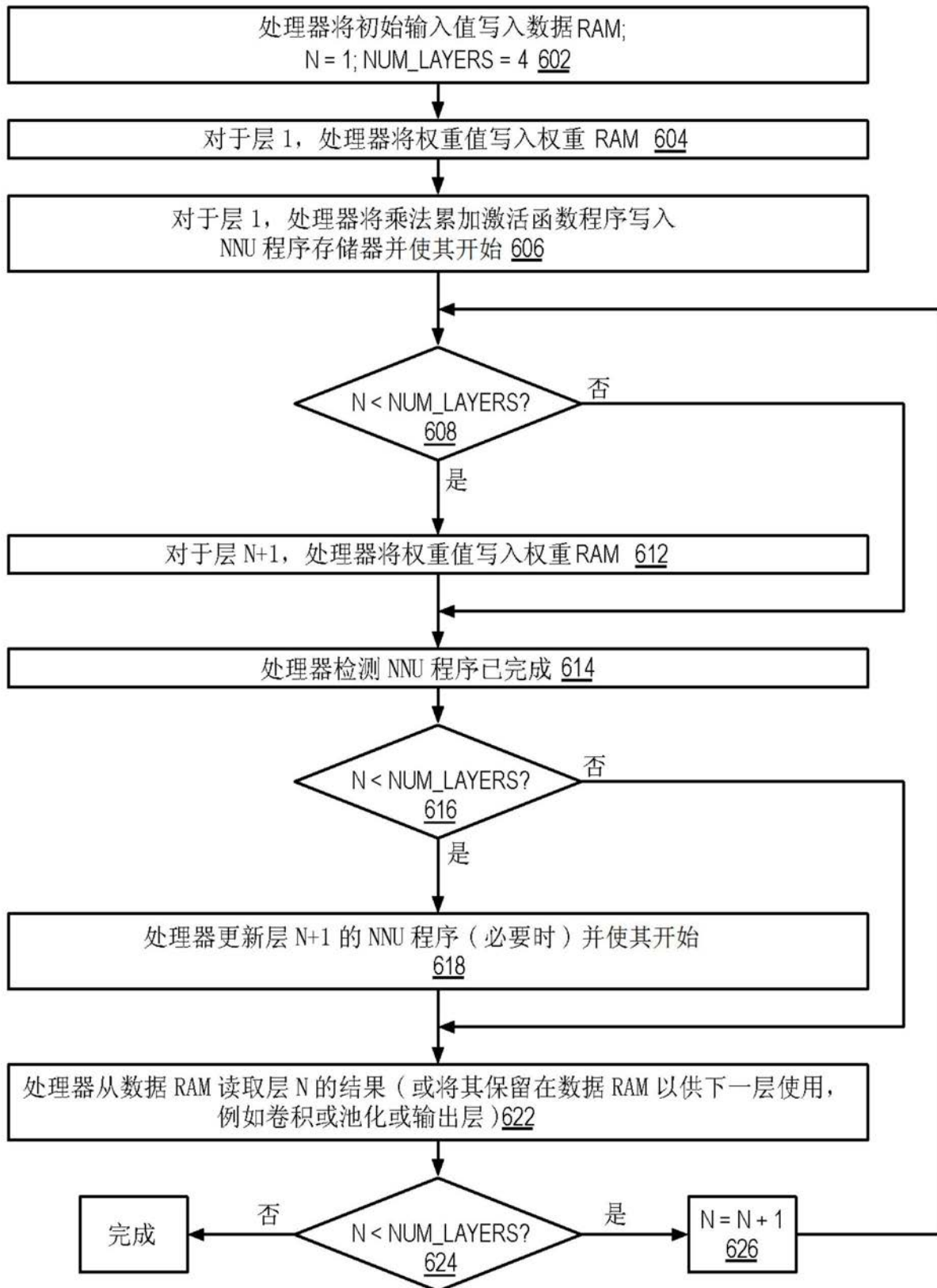


图6B

126-J

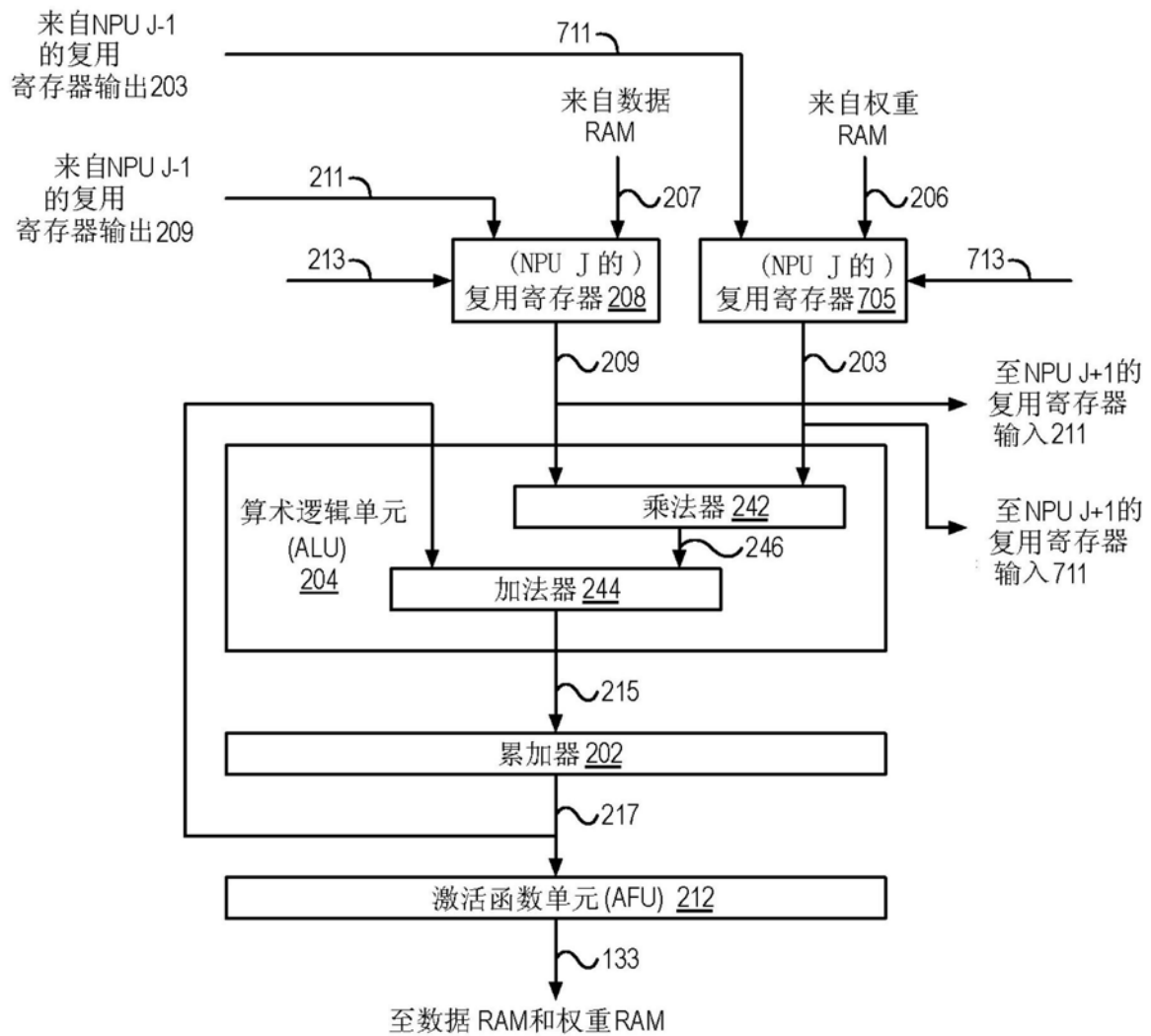


图7

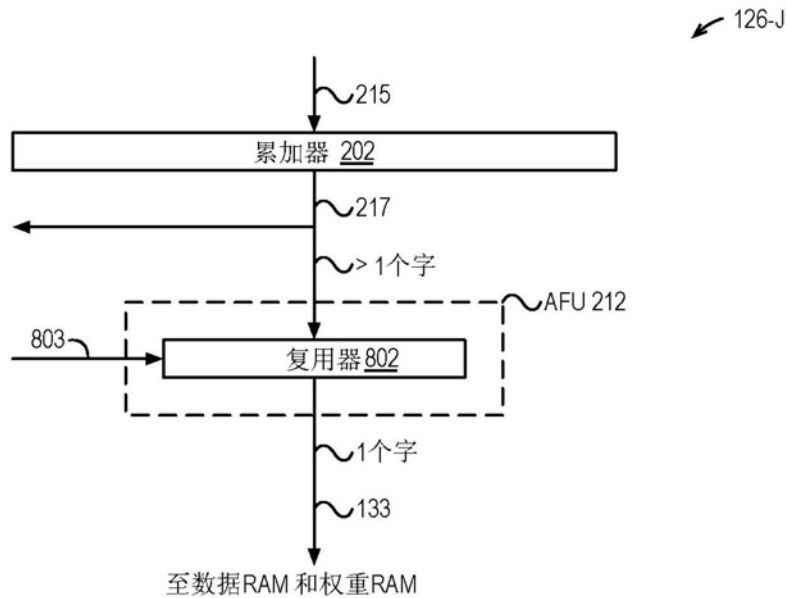


图8

地址	指令	时钟
0	INITIALIZE NPU	1
1	MULT-ACCUM DR ROW 17, WR ROW 0	1
2	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511	511
3	WRITE ACC TO DR ROW 16	1
4	WRITE ACC TO DR ROW 17	1
5	WRITE ACC TO DR ROW 18	1

图9

CLK	NPU 0	NPU 1	...	NPU 511
0	ACC = 0	ACC = 0		ACC = 0
1	ACC += DR:17,0 * WR:0,0	ACC += DR:17,1 * WR:0,1		ACC += DR:17,511 * WR:0,511
2	ACC += MREG:511 * WR:1,0	ACC += MREG:0 * WR:1,1		ACC += MREG:510 * WR:1,511
3	ACC += MREG:511 * WR:2,0	ACC += MREG:0 * WR:2,1		ACC += MREG:510 * WR:2,511
		...		
512	ACC += MREG:511 * WR:511,0	ACC += MREG:0 * WR:511,1		ACC += MREG:510 * WR:511,511
513	WRITE ACC[15:00] TO DATA RAM ROW 16			
514	WRITE ACC[31:16] TO DATA RAM ROW 17			
515	WRITE ACC[47:32] TO DATA RAM ROW 18			

图10

121

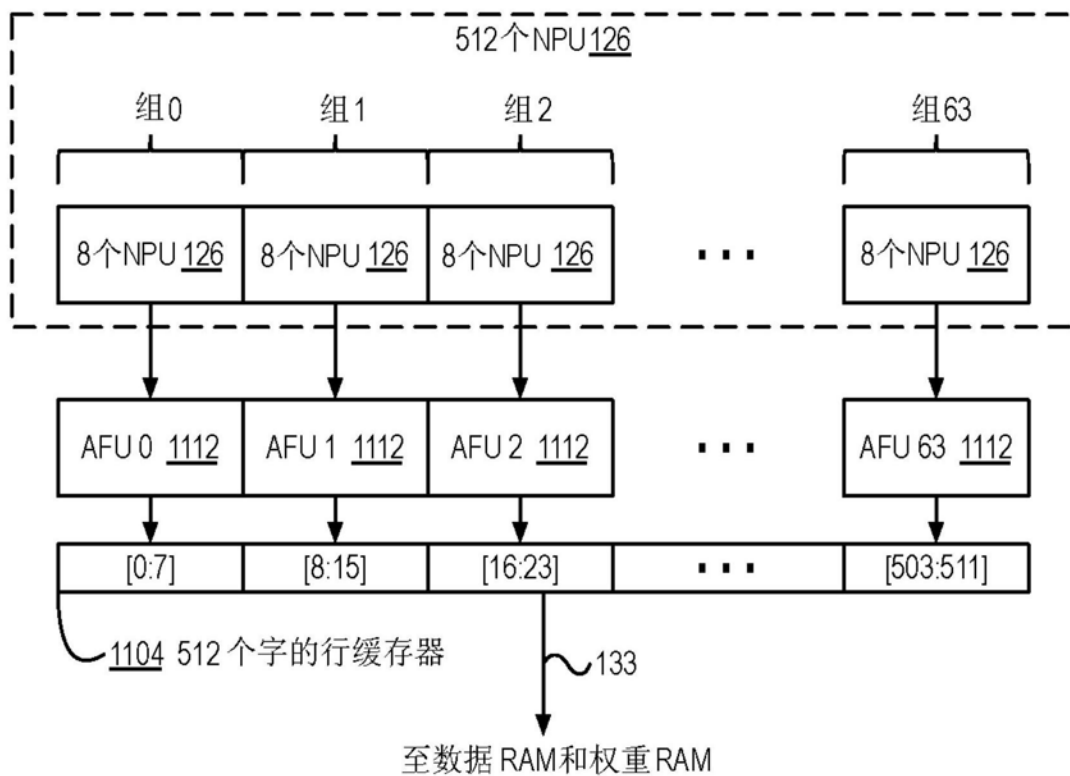


图11

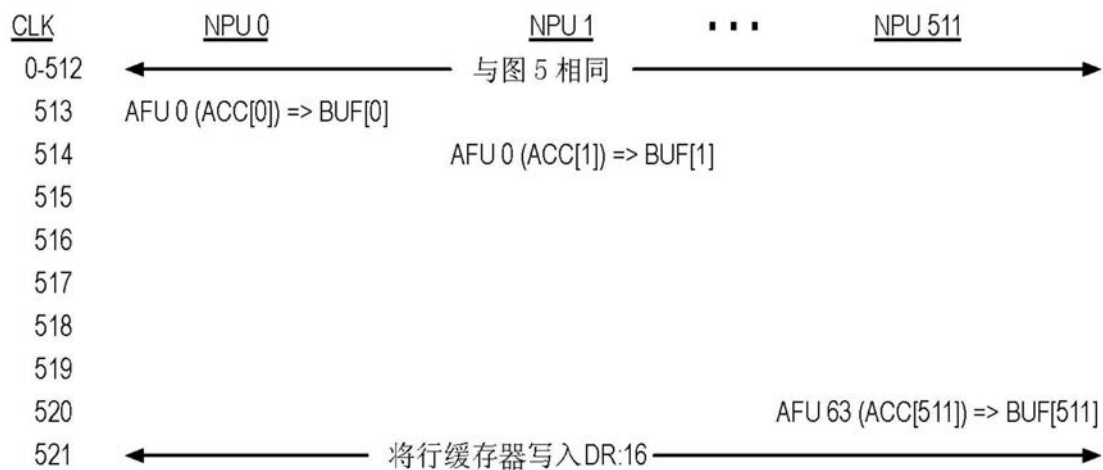


图12

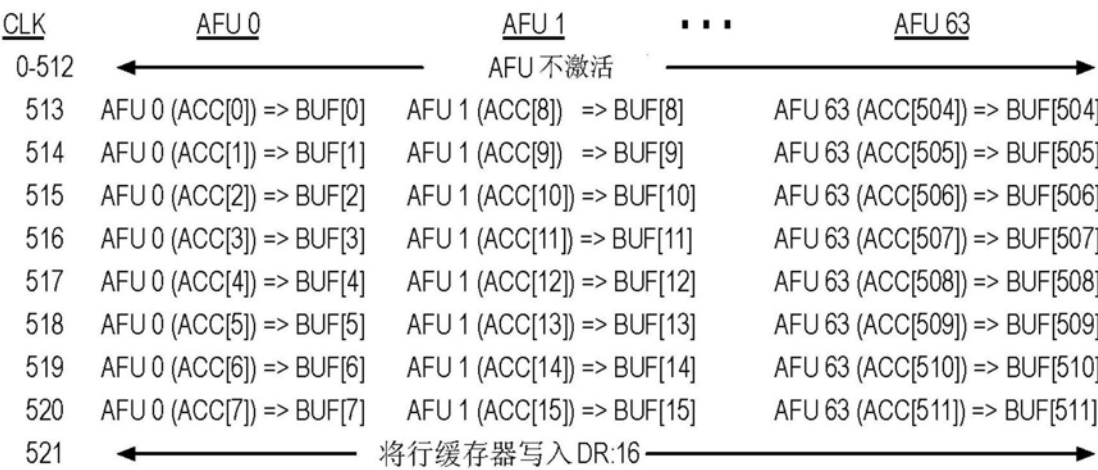


图13

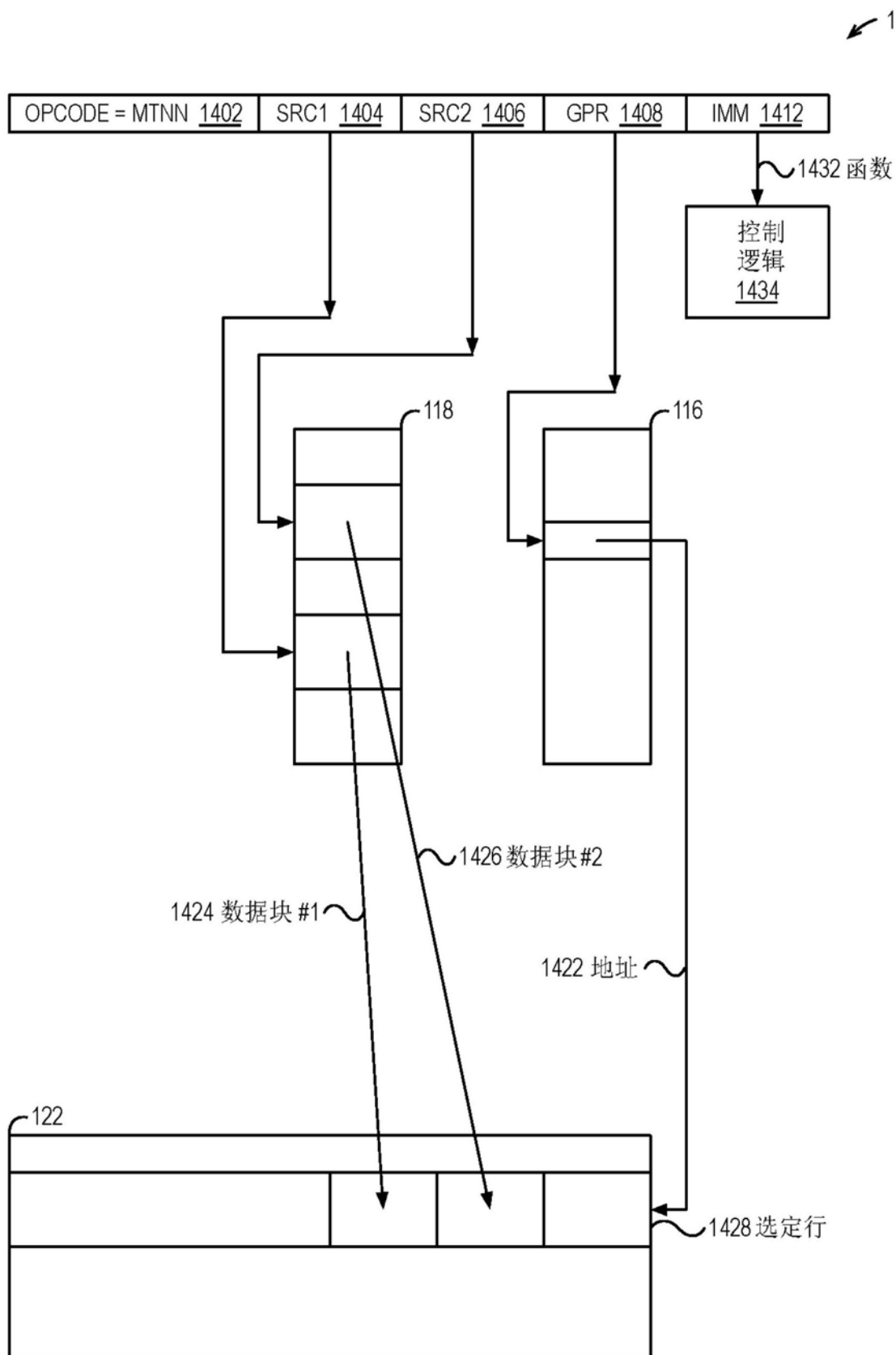


图14

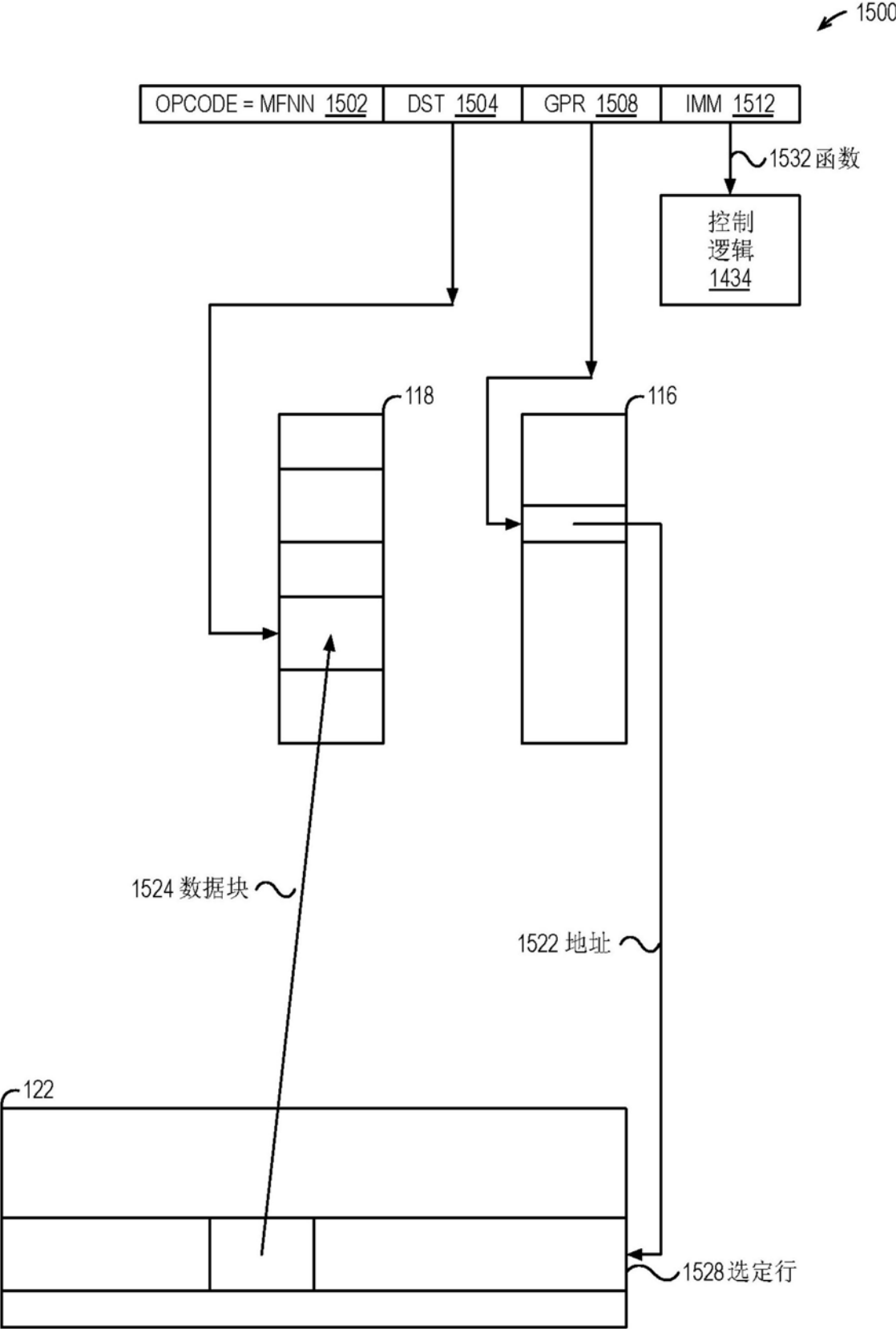


图15

↖ 122

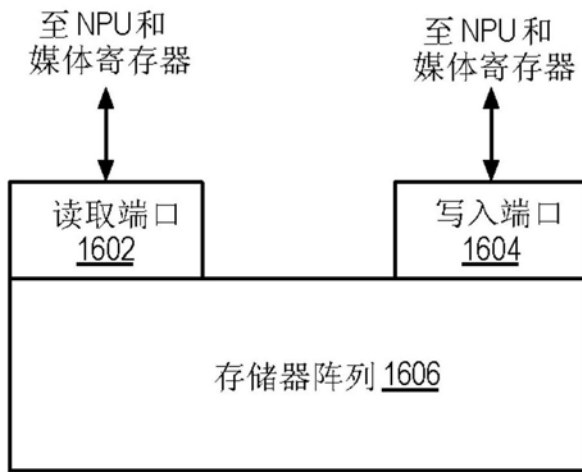


图16

↖ 124

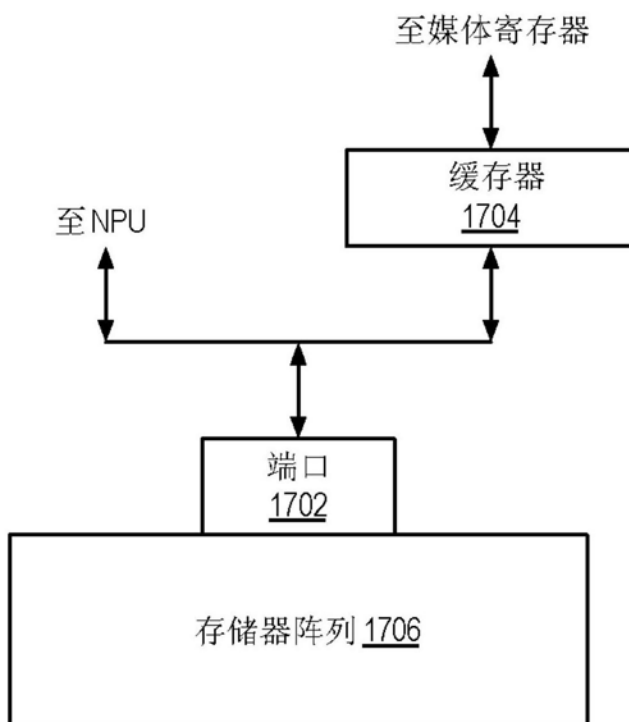


图17

126-J

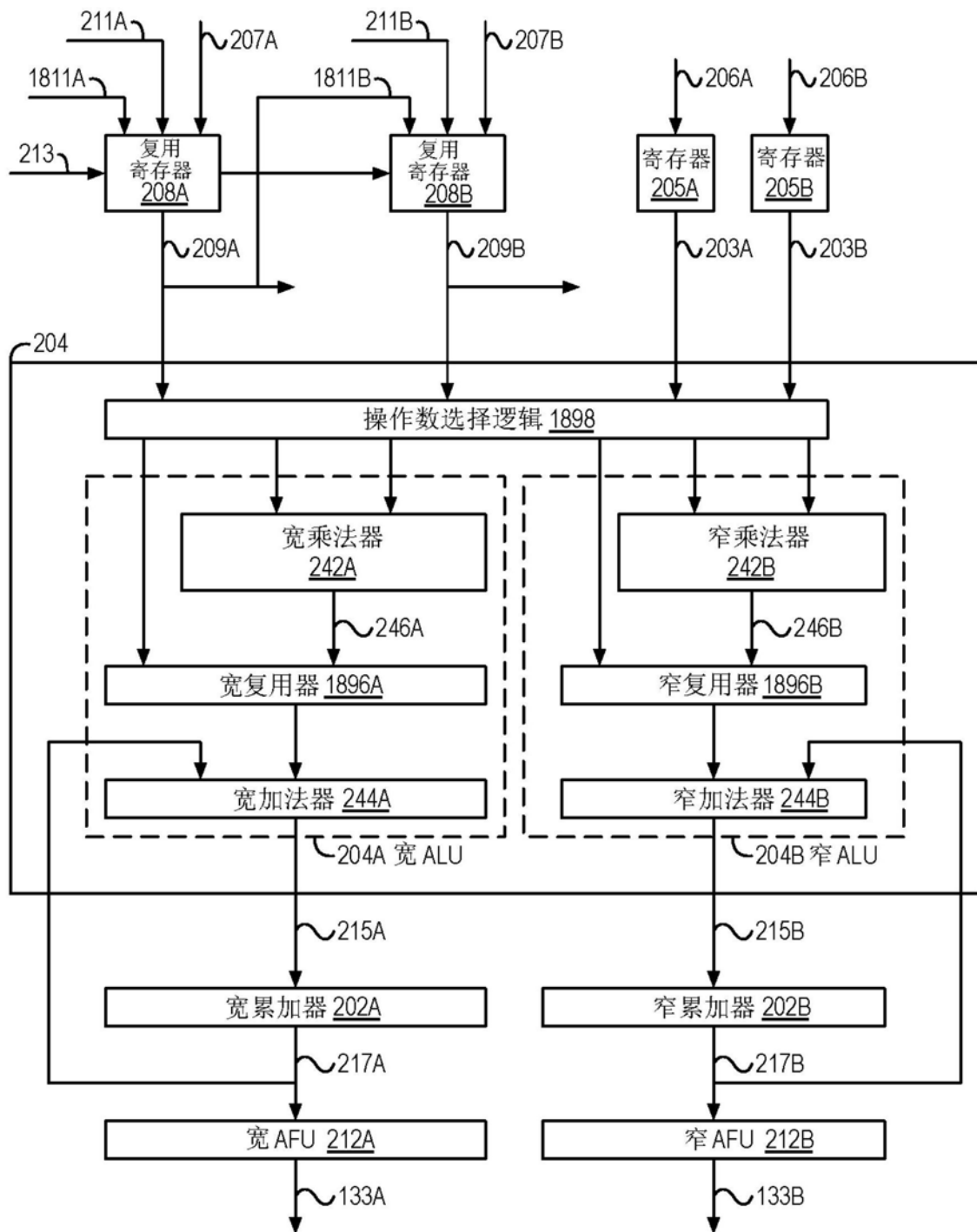


图18

126

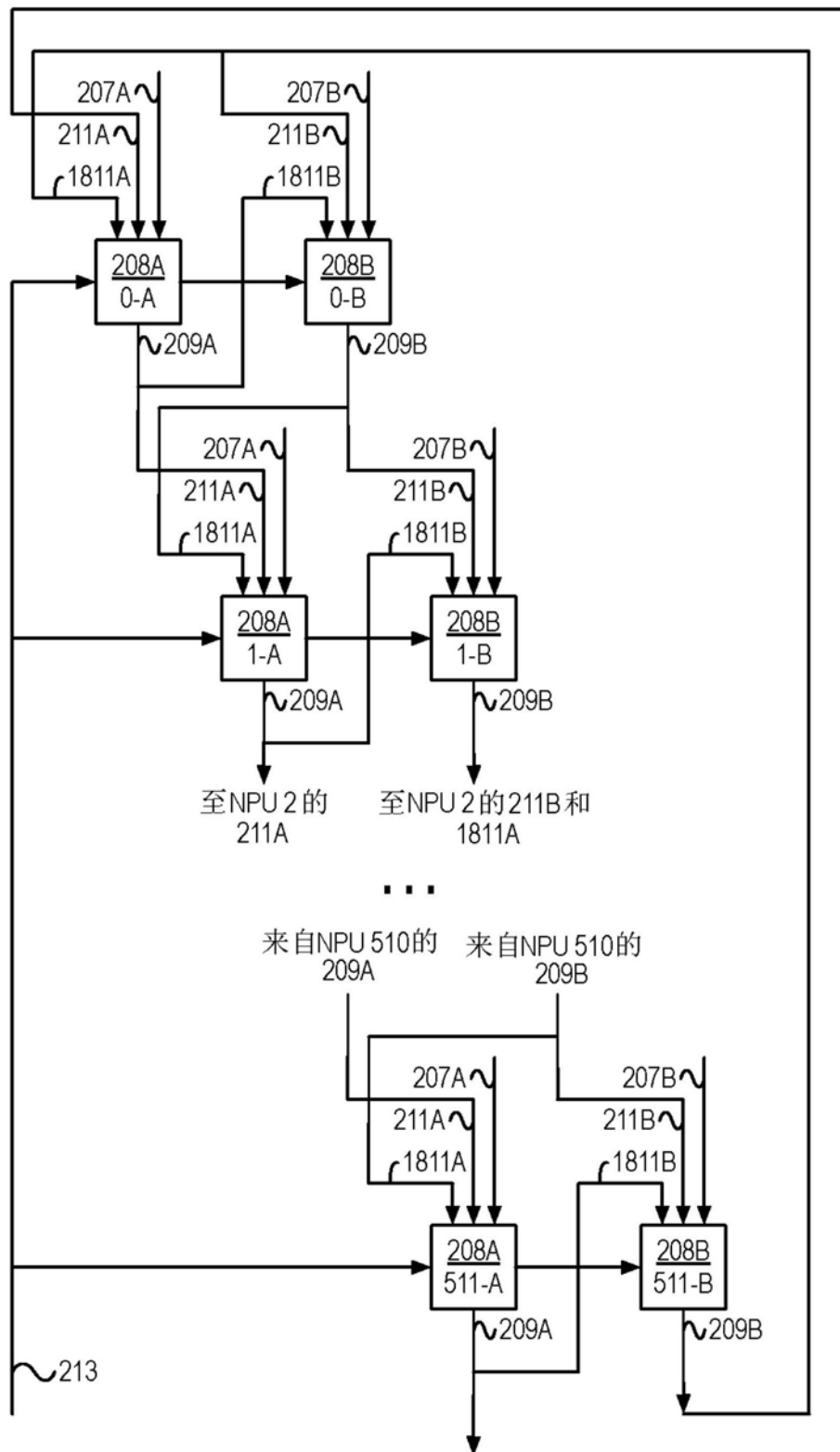


图19

地址	指令	时钟
0	INITIALIZE NPU NARROW	1
1	MULT-ACCUM DR ROW 17, WR ROW 0	1
2	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 1023	1023
3	ACTIVATION FUNCTION	1
4	WRITE AFU OUTPUT TO DR ROW 16	1

图20

CLK	NPU 0-A	NPU 0-B	...	NPU 511-B
0	ACC = 0	ACC = 0		ACC = 0
1	ACC += DR:17,0 * WR:0,0	ACC += DR:17,1 * WR:0,1		ACC += DR:17,1023 * WR:0,1023
2	ACC += MREGB:511 * WR:1,0	ACC += MREGA:0 * WR:1,1		ACC += MREGA:511 * WR:1,1023
3	ACC += MREGB:511 * WR:2,0	ACC += MREGA:0 * WR:2,1		ACC += MREGA:511 * WR:2,1023
			...	
1024	ACC += MREGB:511 * WR:1023,0	ACC += MREGA:0 * WR:1023,1		ACC += MREGA:511 * WR:1023,1023
1025	AFU (ACC)	AFU (ACC)		AFU (ACC)
1026	WRITE AFU OUTP TO DR:16,0	WRITE AFU OUTP TO DR:16,1		WRITE AFU OUTP TO DR:16,1023

图21

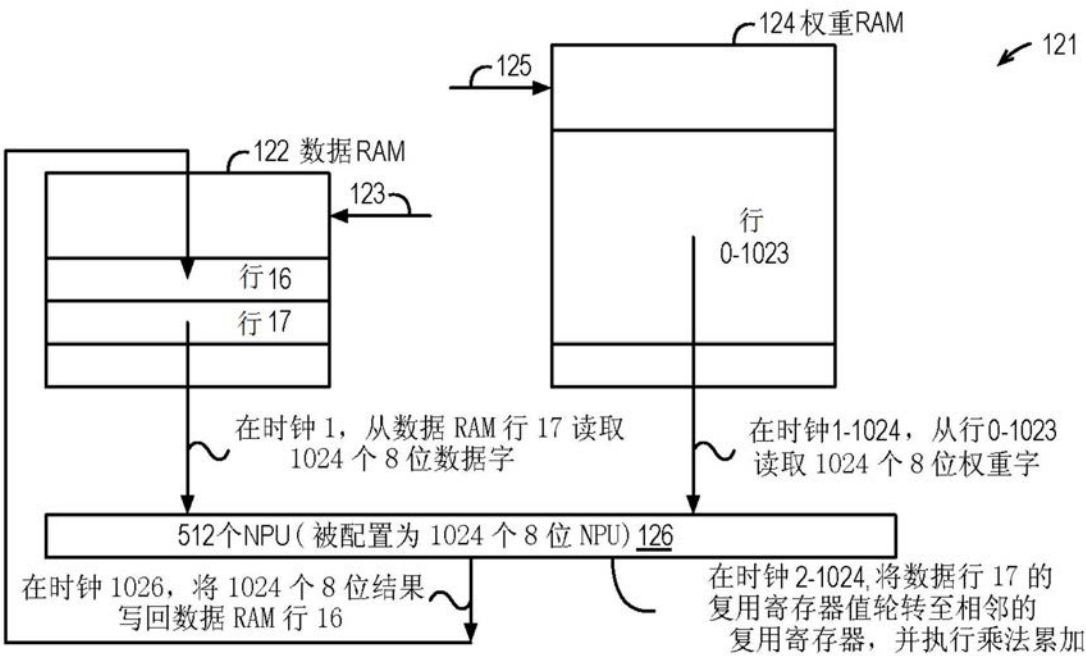


图22

126-J

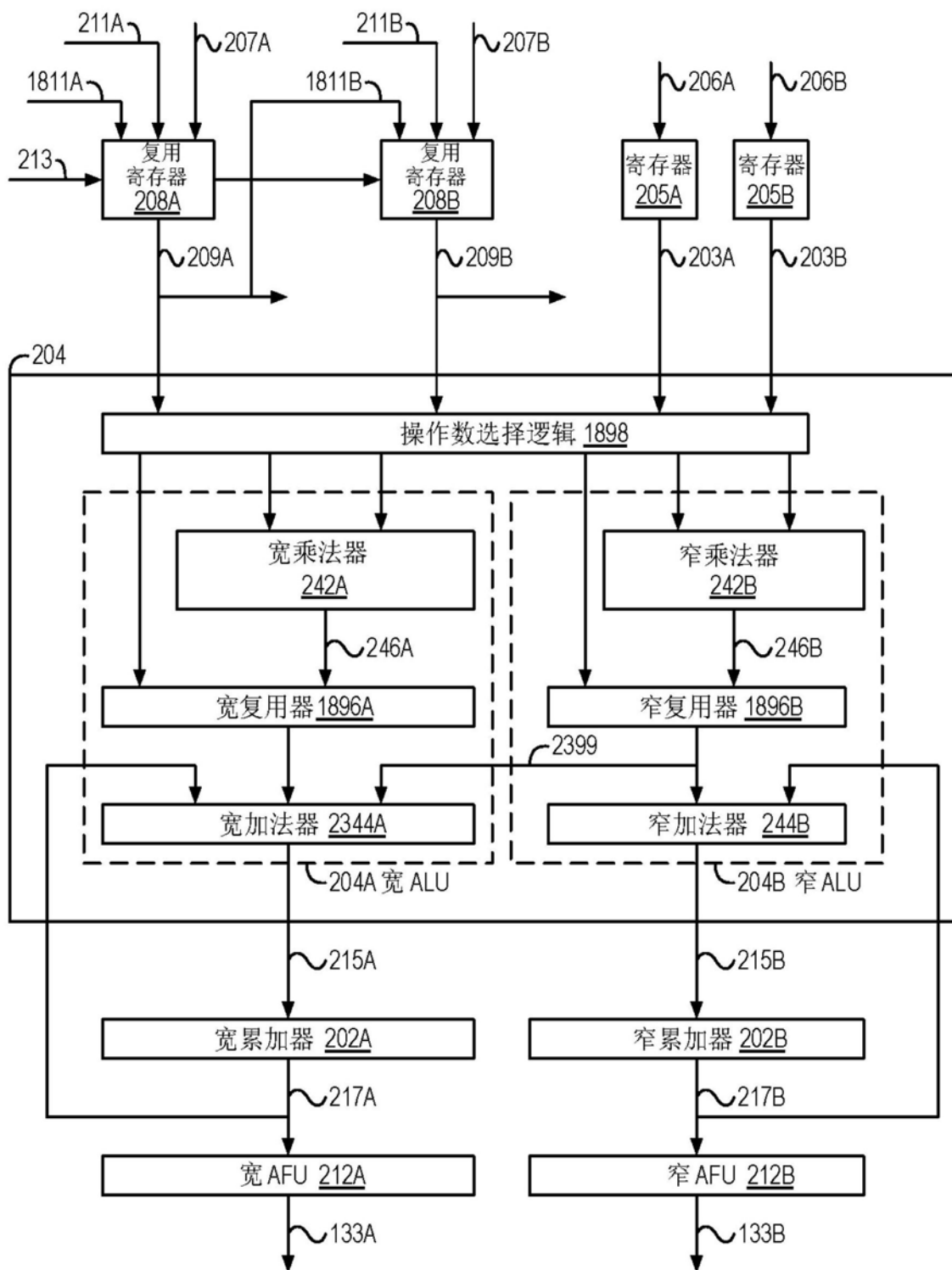


图23

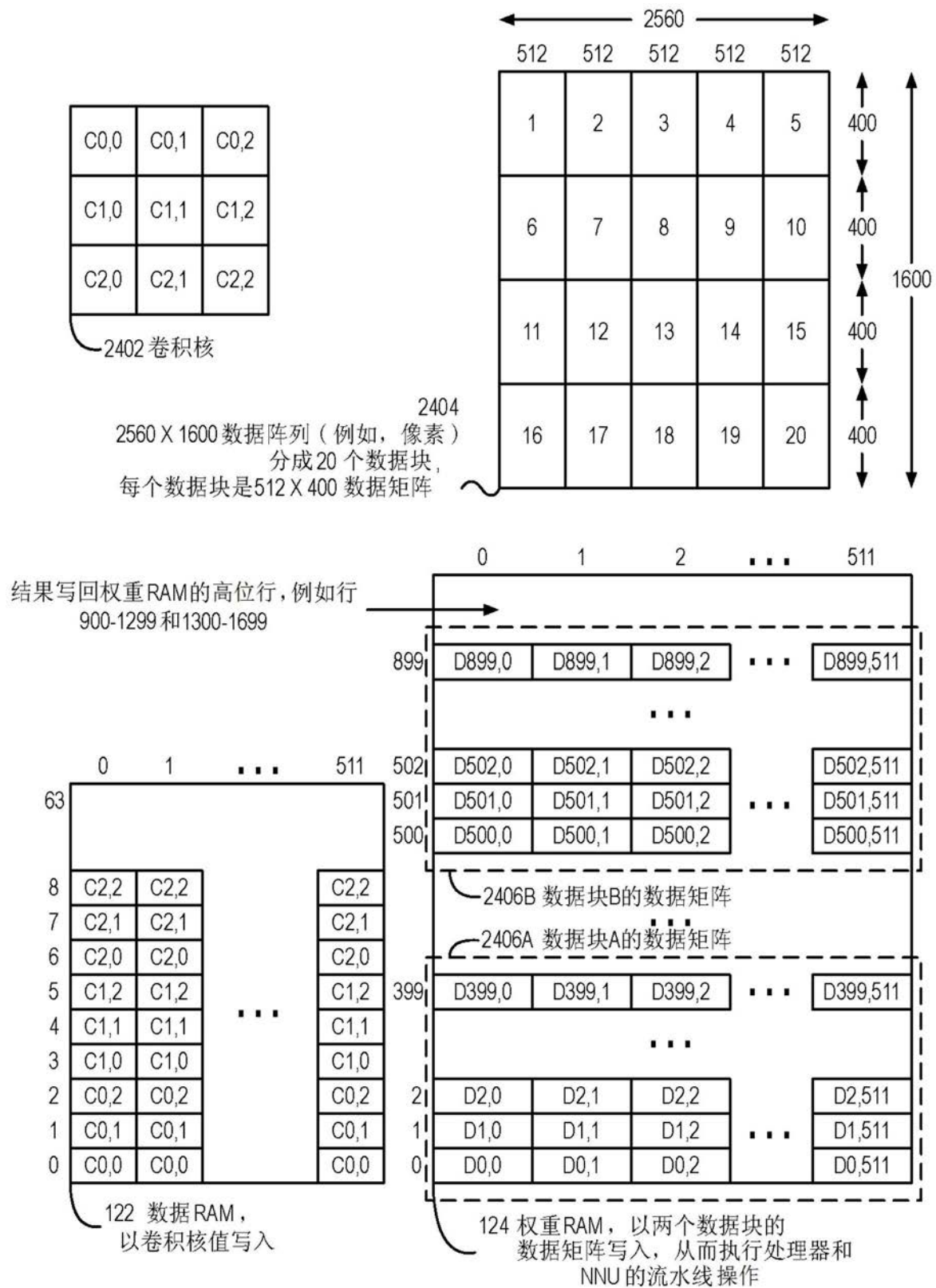


图24

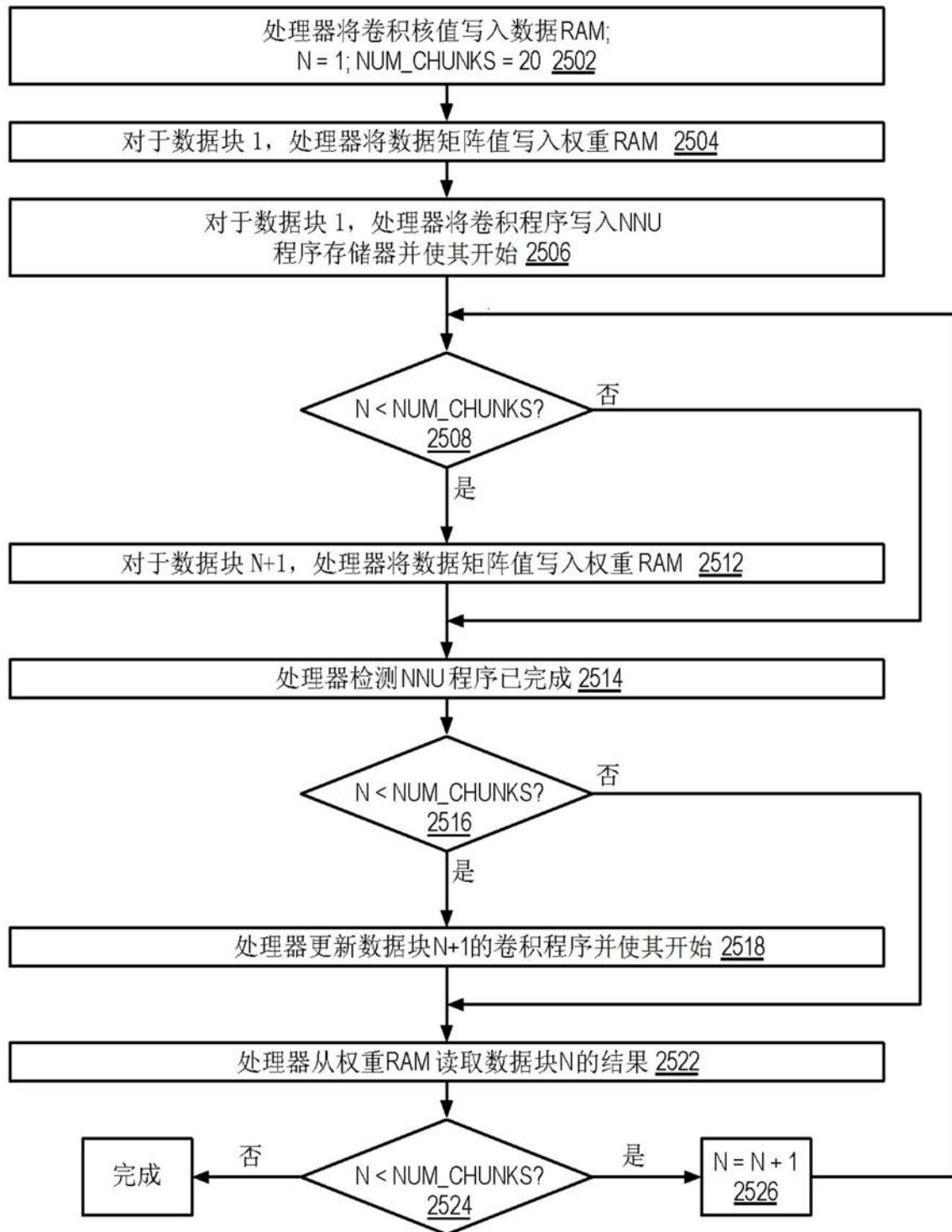


图25

地址	指令	时钟
0	INITIALIZE NPU, WR ROW = 0, OUTROW = WR 900, LOOPCNT = 400	1
1	MULT-ACCUM DR ROW 0	1
2	MULT-ACCUM DR ROW +1, W ROTATE, COUNT = 2	2
3	MULT-ACCUM DR ROW +1, WR ROW +1	1
4	MULT-ACCUM DR ROW +1, W ROTATE, COUNT = 2	2
5	MULT-ACCUM DR ROW +1, WR ROW +1	1
6	MULT-ACCUM DR ROW +1, W ROTATE, COUNT = 2	2
7	ACTIVATION FUNCTION, PASS THROUGH	1
8	WRITE AFU OUTPUT TO OUTROW +1	1
9	DECREMENT WR ROW	1
10	LOOP TO 1	1

图26A

127

最近写入的权重RAM 行的地址	<u>2602</u>
最近读取的权重RAM 行的地址	<u>2604</u>
最近写入的数据RAM 行的地址	<u>2606</u>
最近读取的数据RAM 行的地址	<u>2608</u>

图26B

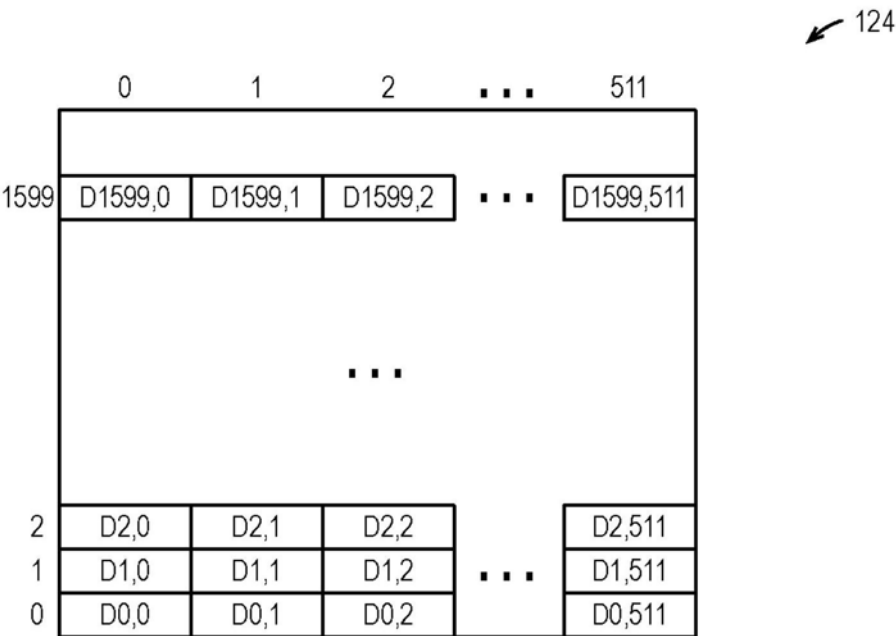


图27

地址	指令	时钟
0	INITIALIZE NPU, WR ROW = 0, OUTROW = WR 0, LOOPCNT = 400	1
1	MAXWACC	1
2	MAXWACC W ROTATE, COUNT = 3	2
3	MAXWACC WR ROW +1	1
4	MAXWACC W ROTATE, COUNT = 3	2
5	MAXWACC WR ROW +1	1
6	MAXWACC W ROTATE, COUNT = 3	2
7	MAXWACC WR ROW +1	1
8	MAXWACC W ROTATE, COUNT = 3	2
9	ACTIVATION FUNCTION, PASS THROUGH	1
10	WRITE AFU OUTPUT TO OUTROW +1	1
11	LOOP TO 1	1

* 只有 4 的倍数的 NPU（即，0、4、8 等）产生有效结果，而输出行的数量是输入行的数量的四分之一，并且数据 RAM 并未被使用

图28

127

配置（窄、宽、漏斗型） 2902
带符号数据 2912
带符号权重 2914
数据二进制小数点 2922
权重二进制小数点 2924
ALU 函数 2926
舍入控制 2932
激活函数 2934
倒数 2942
移位量 2944
输出 RAM 2952
输出二进制小数点 2954
输出命令 2956

图29A

127

累加器二进制小数点 2923

图29B

2942

被抑制的前导零的数量 2962
前导零被抑制倒数 2964

图29C

212

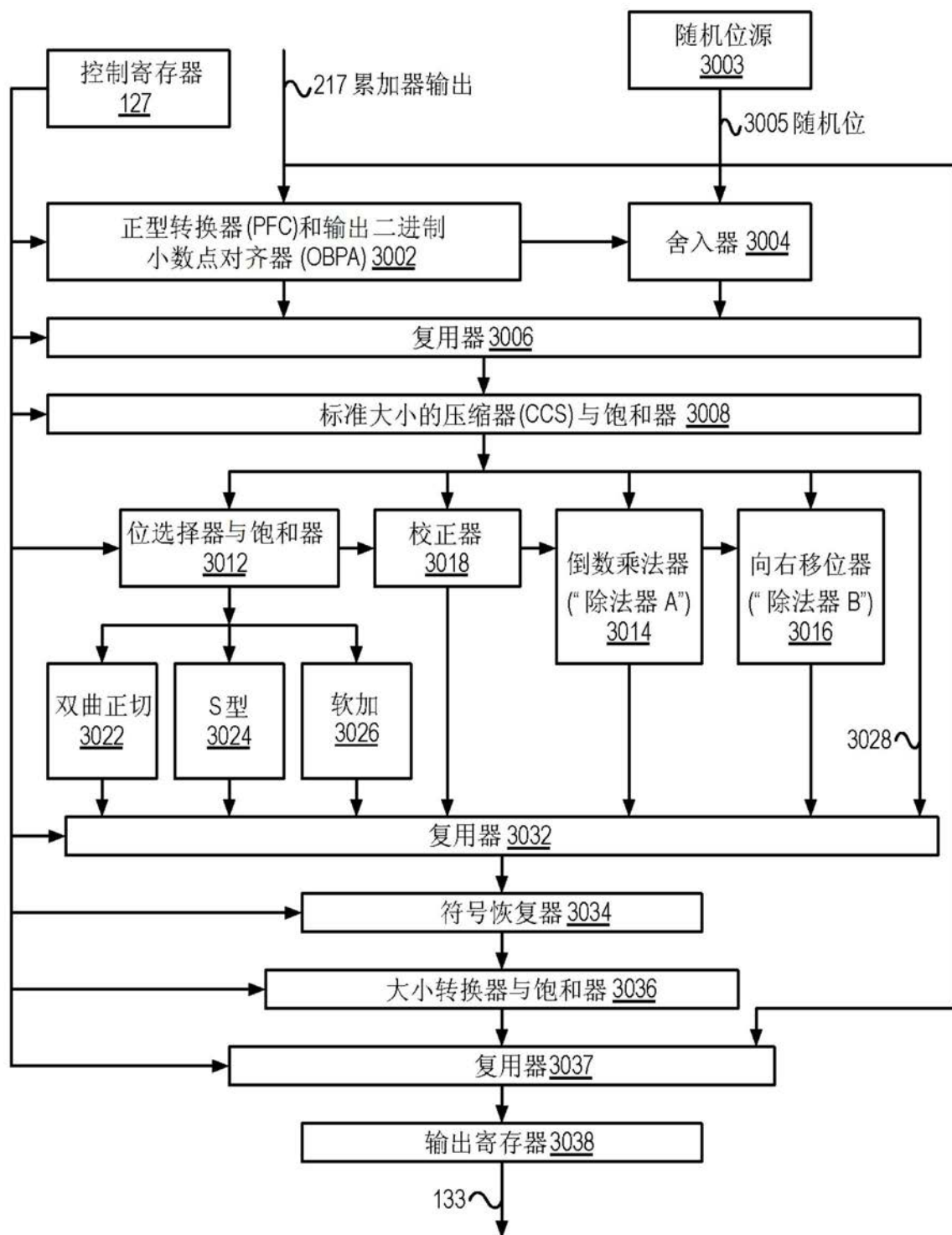


图30

配置 = 窄 (8 位), 带符号数据 = 真, 带符号权重 = 真
 数据二进制小数点 = 7 (第一数据字) 0.1001110
 权重二进制小数点 = 3 (第一权重字) 00001.010
 隐含乘法格式 (第一个乘法运算后的值) 000000.1100001100
 所有乘法累加运算后的累加器 00000000000000000001.1101010100
 输出二进制小数点 = 7, 舍入并压缩至标准形式 000000001.1101011
 舍入控制 = 随机, 并且随机位为真, 因此向上舍入 ↗
 激活函数 = S 型, 因此压缩至 3.4 形式 001.1101
 标准形式的 S 型输出 000000000000.1101110
 输出命令 = 窄字大小 01101110

图31

配置 = 窄 (8 位)
 所有 ALU 运算后的累加器 (累加器二进制小数点 = 10) 000001100000011011.1101111010
 输出二进制小数点 = 4, 饱和并压缩至标准形式 111111111111.1111
 激活函数 = 通过标准
 输出命令 = 标准形式大小, 低位字 11111111
 输出命令 = 标准形式大小, 高位字 11111111

图32

配置 = 宽 (16 位)
 所有 ALU 运算后的累加器 (累加器二进制小数点 = 8) 00100000000000000001100000011011.11011110
 激活函数 = 通过完全原始的累加器
 输出命令 = 低位字 0001101111011110
 输出命令 = 中间字 0000000000011000
 输出命令 = 高位字 0000000001000000

图33

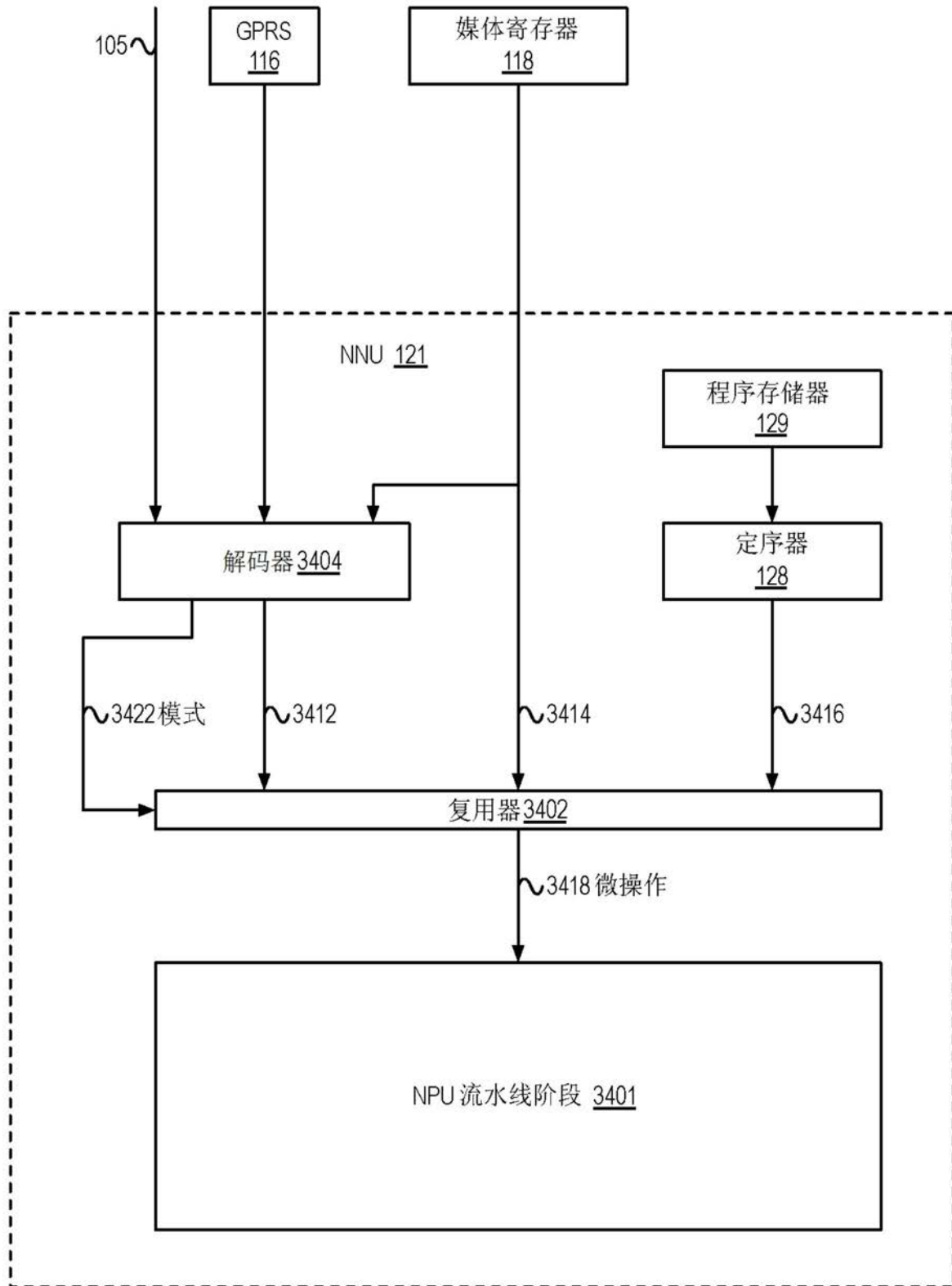


图34

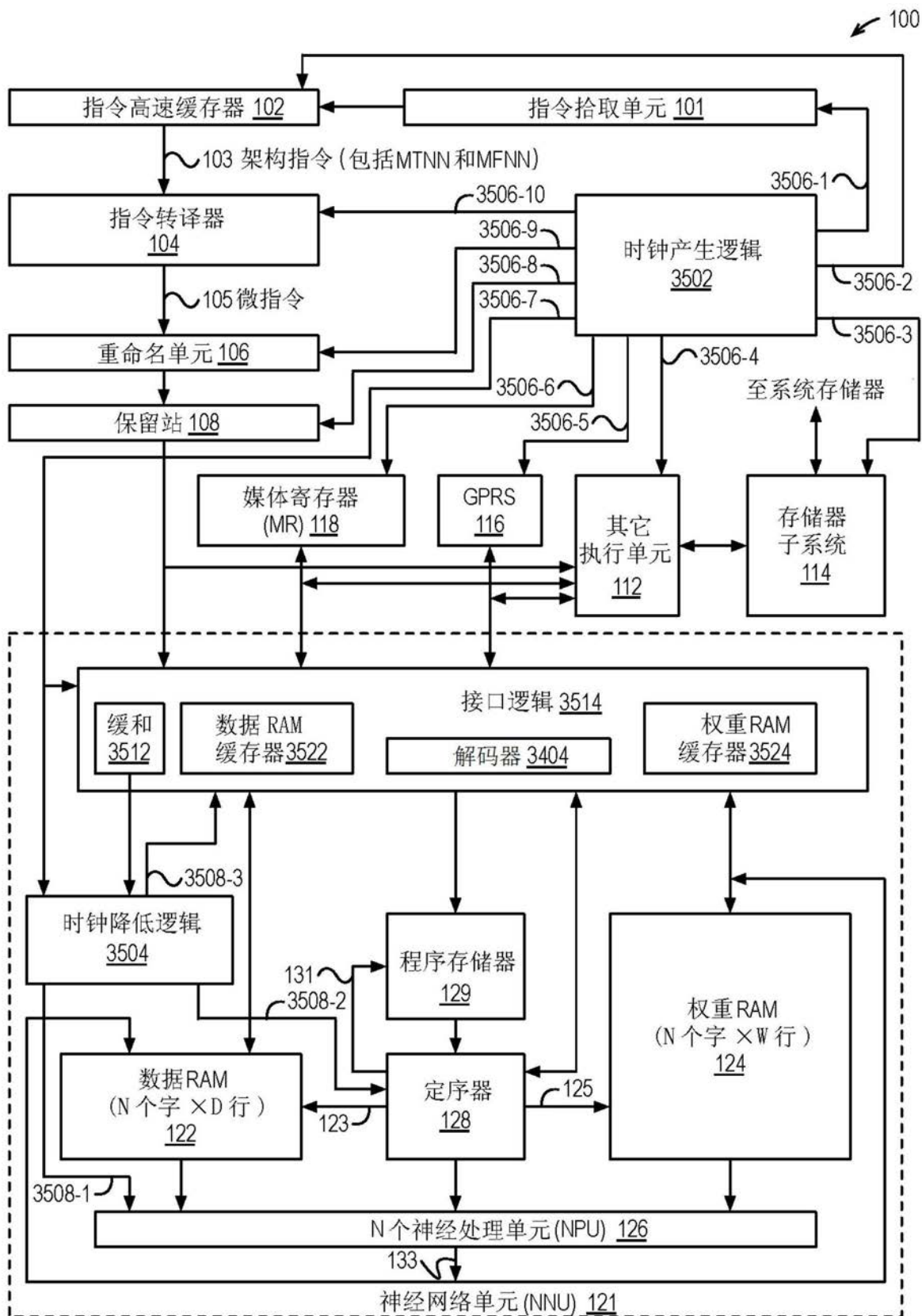


图35

一般模式

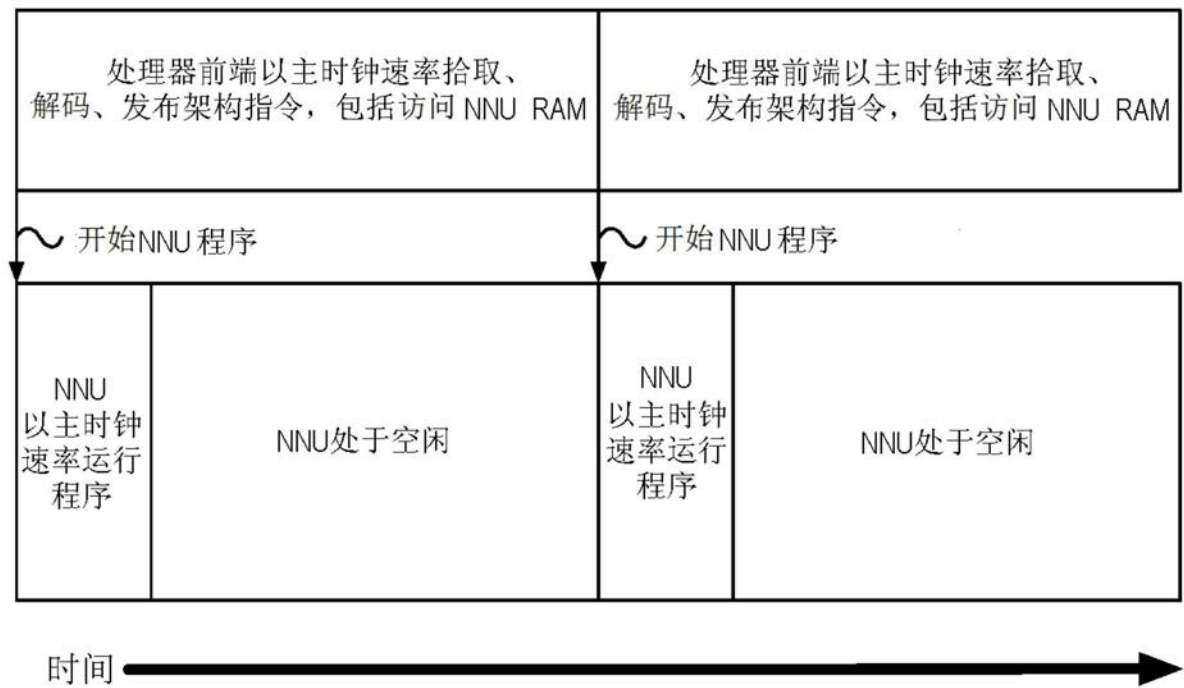


图36A

缓和模式



图36B

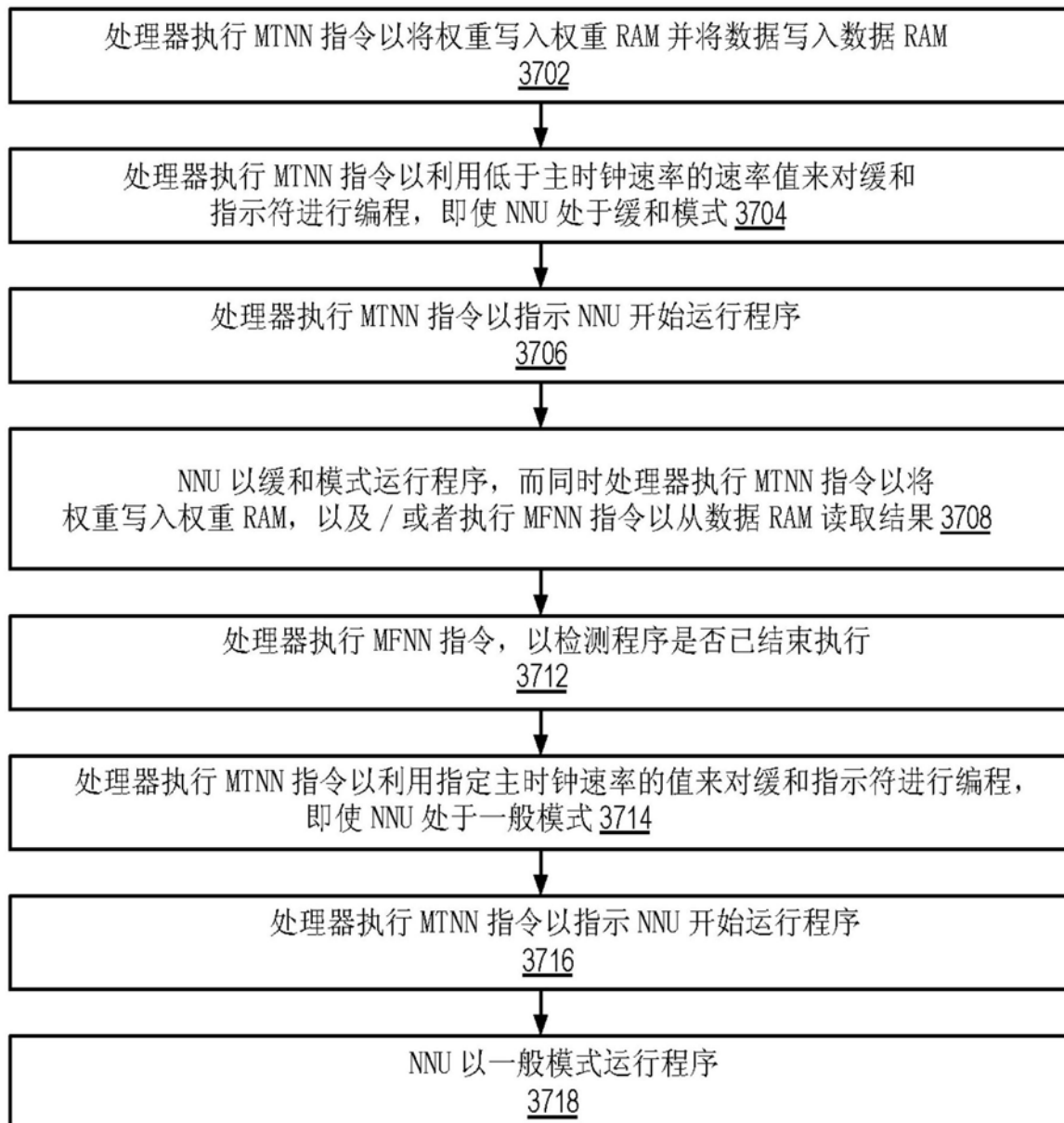


图37

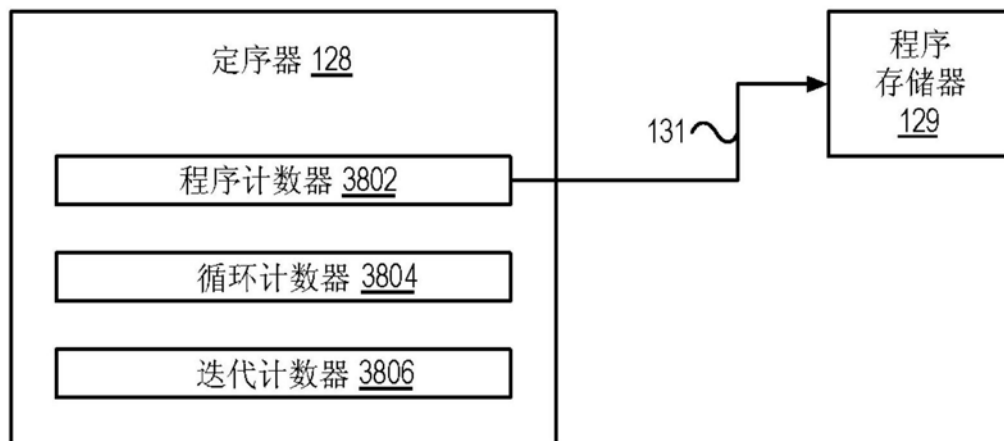


图38

127

权重RAM的最近写入行的地址 <u>2602</u>
权重RAM的最近读取行的地址 <u>2604</u>
数据RAM的最近写入行的地址 <u>2606</u>
数据RAM的最近读取行的地址 <u>2608</u>
NNU程序计数器 <u>3912</u>
循环计数 <u>3914</u>
迭代计数 <u>3916</u>

图39

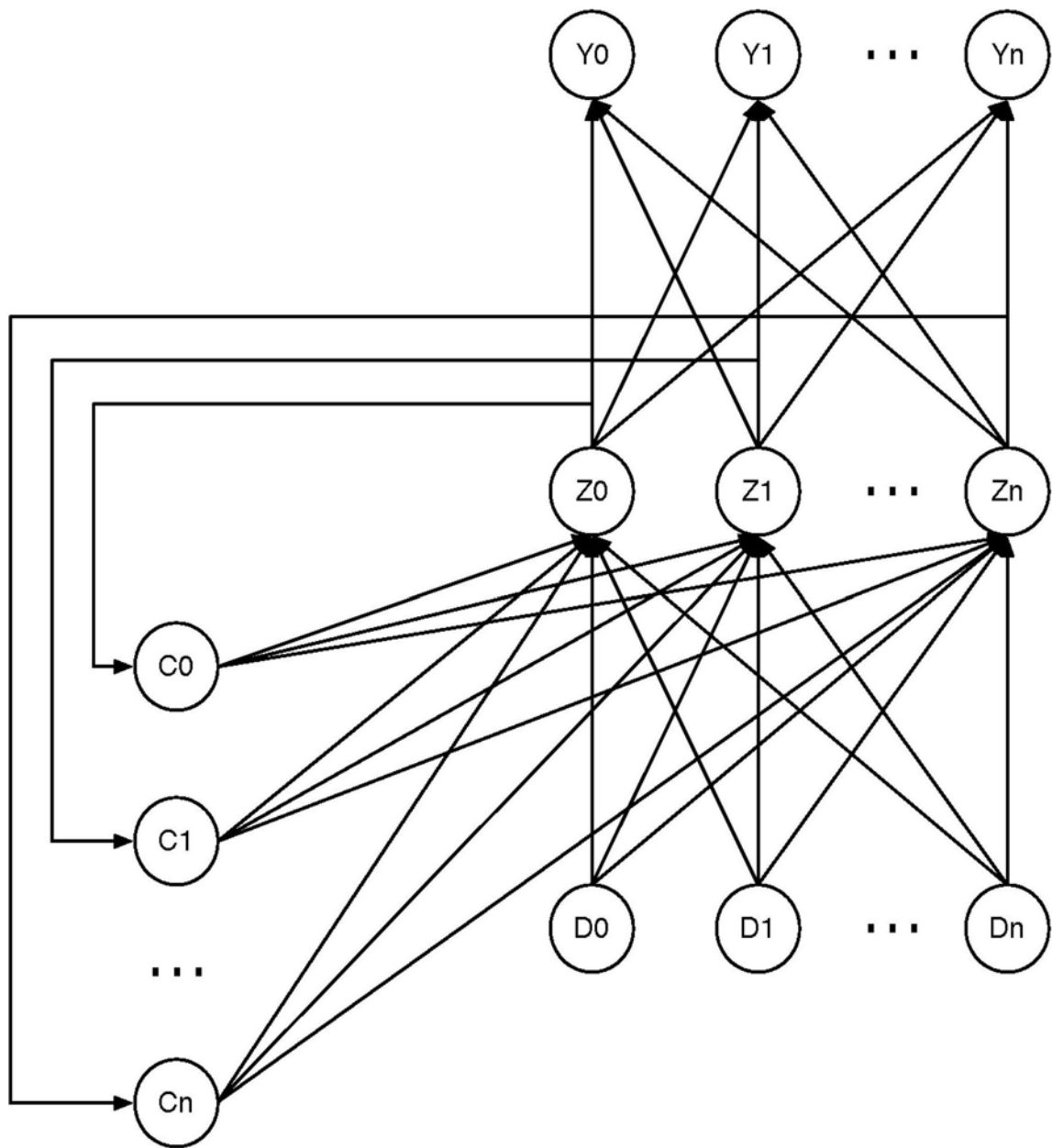
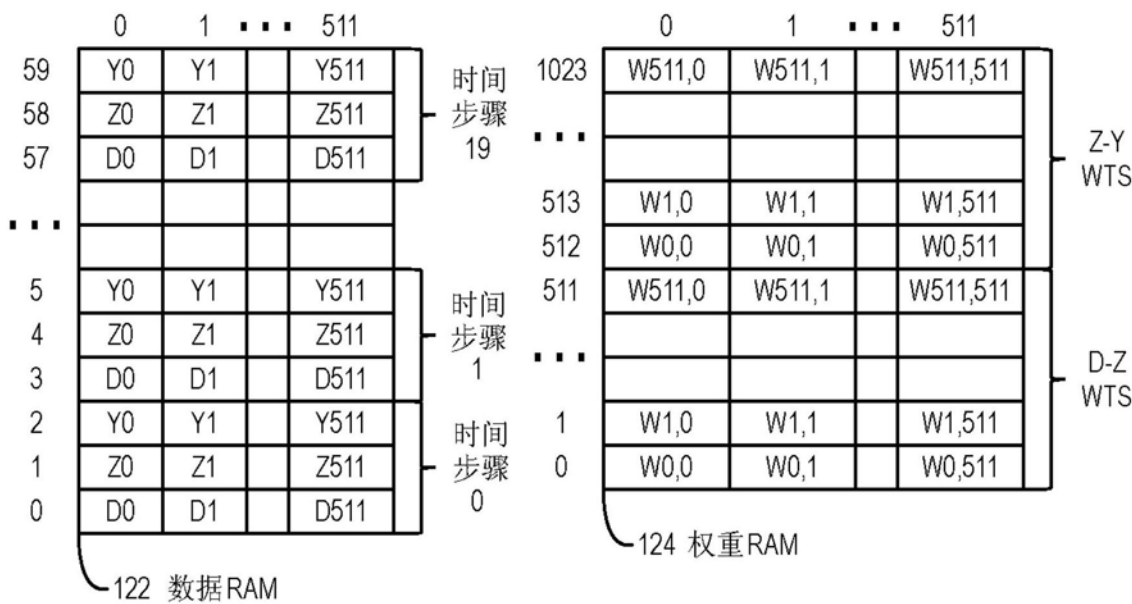


图40



* 某时间步骤的 Z 值为下一时间步骤的 C 值

图41

地址	指令
0	INITIALIZE NPU, LOOPCNT = 20 // 20 个时间步骤：假定初始 C 节点为零
1	MULT-ACCUM DR ROW 0, WR ROW 0 // 计算初始 Z 层
2	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
3	OUTPUT PASSTHRU, DR OUT ROW 1, CLR ACC // 将初始 Z 层输出至 DRAM
4	MULT-ACCUM DR ROW +1, WR ROW 512 // 计算 Y 层
5	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
6	OUTPUT ACTIVATION FUNCTION, DR OUT ROW +1, CLR ACC // 将 Y 层输出至 DRAM
7	ADD_D_ACC DR ROW +0 // 累加承接节点：假定权重为一
8	ADD_D_ACC ROTATE, COUNT = 511 // 不清除 ACC
9	MULT-ACCUM DR ROW +2, WR ROW 0 // 累加 D 以计算 Z 层
10	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
11	OUTPUT PASSTHRU, DR OUT ROW +2, CLR ACC // 将 Z 层输出至 DRAM
12	LOOP 4

图42

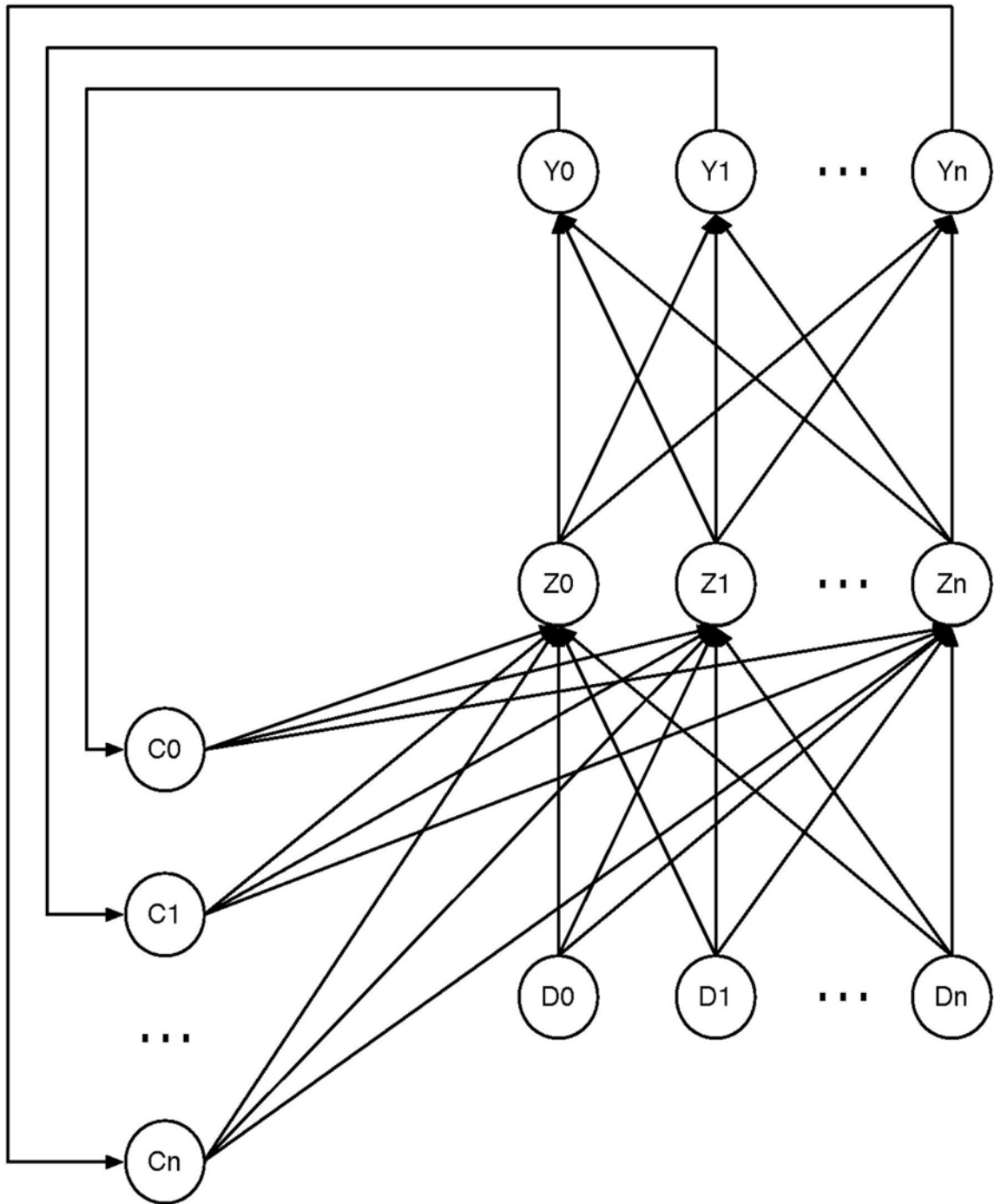


图43

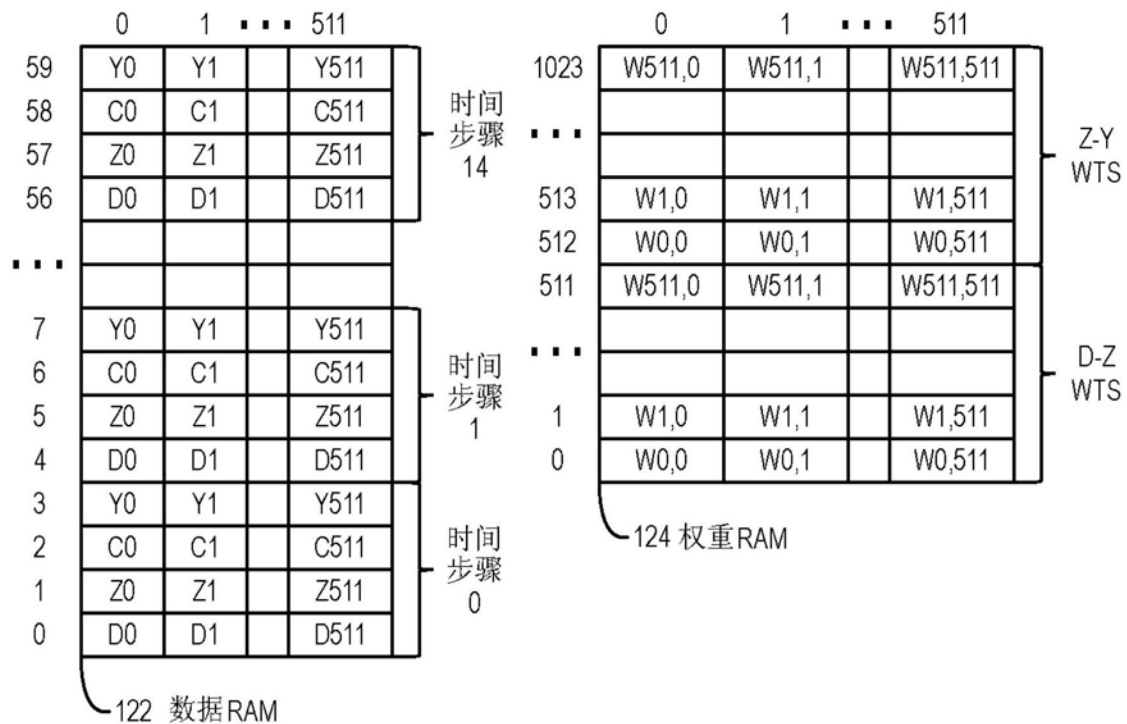
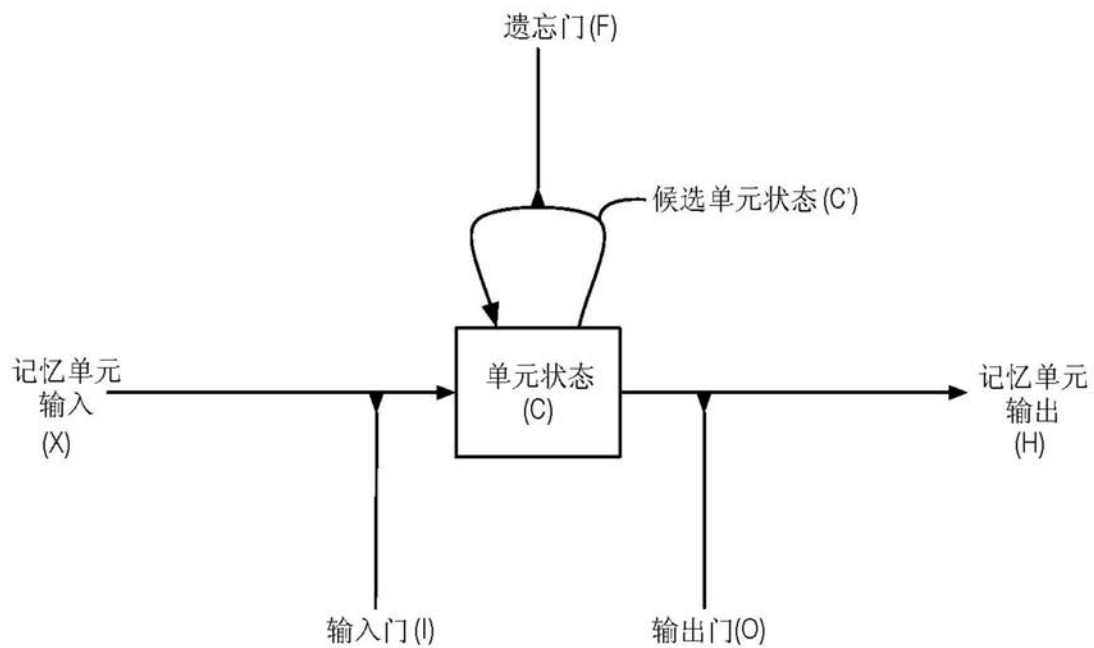


图44

地址	指令
0	INITIALIZE NPU, LOOPCNT = 15 //15 个时间步骤：假定初始 C 节点为零
1	MULT-ACCUM DR ROW 0, WR ROW 0 // 计算初始 Z 层
2	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
3	OUTPUT PASSTHRU, DR OUT ROW 1, CLR ACC // 将初始 Z 层输出至 DRAM
4	MULT-ACCUM DR ROW +1, WR ROW 512 // 计算 Y 层
5	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
6	OUTPUT PASSTHRU, DR OUT ROW +1 // 将 C 层输出至 DRAM；不清除 ACC
7	OUTPUT ACTIVATION FUNCTION, DR OUT ROW +1, CLR ACC //将 Y 层输出至 DRAM
8	ADD_D_ACC DR ROW +1 // 累加承接节点：假定权重为一
9	ADD_D_ACC ROTATE, COUNT = 511
10	MULT-ACCUM DR ROW +2, WR ROW 0 //累加 D 以计算 Z 层
11	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
12	OUTPUT PASSTHRU, DR OUT ROW +2, CLR ACC // 将 Z 层输出至 DRAM
13	LOOP 4

图45

4600 LSTM单元



- (1) $I = \text{SIGMOID}(W_i * X + U_i * H + B_i)$
- (2) $F = \text{SIGMOID}(W_f * X + U_f * H + B_f)$
- (3) $C' = \text{TANH}(W_c * X + U_c * H + B_c)$
- (4) $C = I * C' + F * C$ (即, 使用当前单元状态作为输入计算新的单元状态)
- (5) $O = \text{SIGMOID}(W_o * X + U_o * H + B_o)$
- (6) $H = O * \text{TANH}(C)$

W_i 和 U_i 是与输入门相关联的权重

W_f 和 U_f 是与遗忘门相关联的权重

W_o 和 U_o 是与输出门相关联的权重

B_i , B_f 和 B_o 是与相应的输入门、遗忘门和输出门相关联的偏置输入

图46

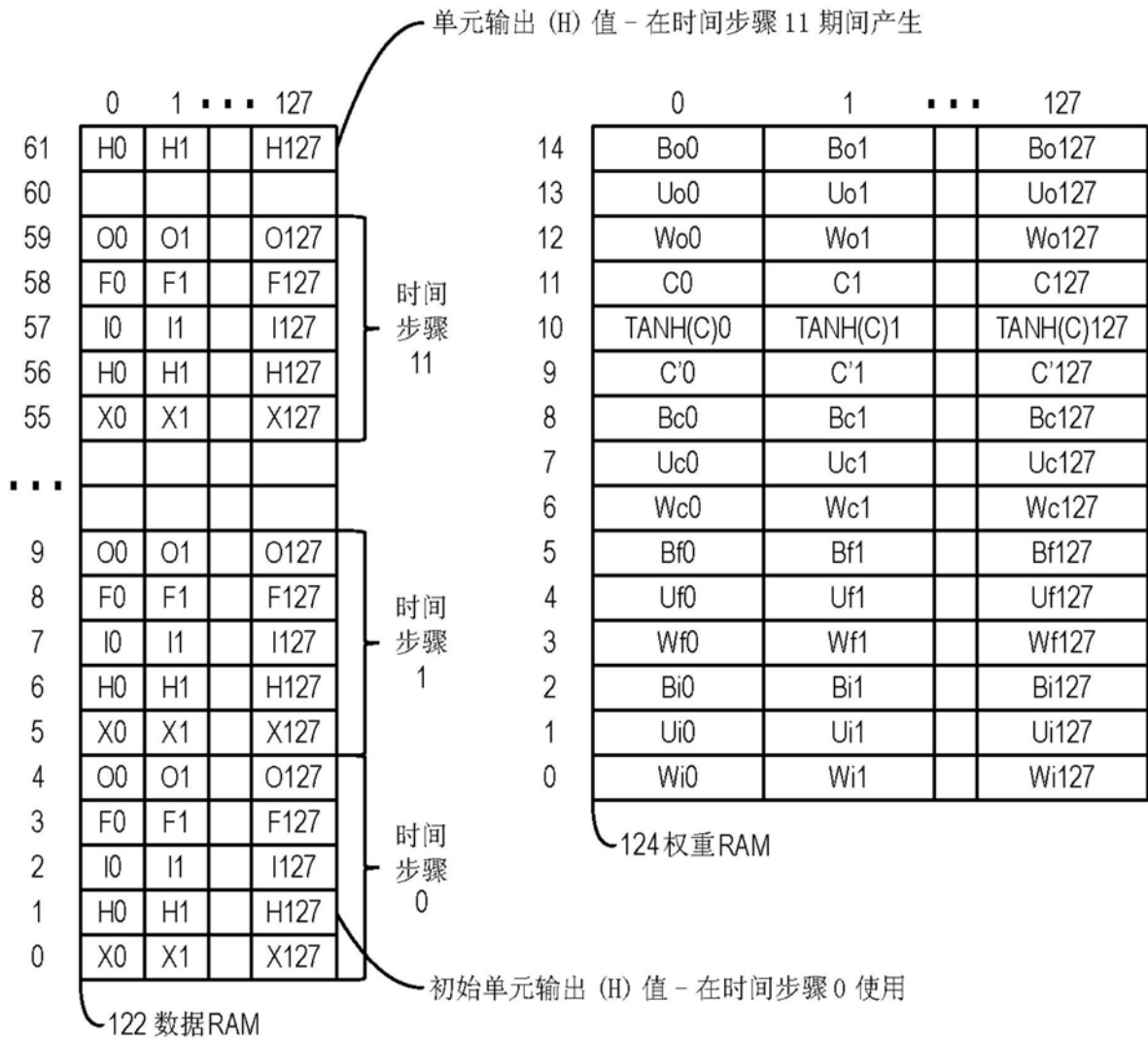


图47

地址	指令
0	INITIALIZE NPU, CLR ACC, LOOPCNT = 12, DR IN ROW = -1, DR OUT ROW = 2 //12个时间步骤
1	MULT-ACCUM DR ROW +1, WR ROW 0 //计算并写入输入门:
2	MULT-ACCUM DR ROW +1, WR ROW 1 // $I = \text{SIGMOID}(W_i * X + U_i * H + B_i)$
3	ADD_W_ACC WR ROW 2
4	OUTPUT SIGMOID, DR OUT ROW +0, CLR ACC
5	MULT-ACCUM DR ROW -1, WR ROW 3 //计算并写入遗忘门:
6	MULT-ACCUM DR ROW +1, WR ROW 4 // $F = \text{SIGMOID}(W_f * X + U_f * H + B_f)$
7	ADD_W_ACC WR ROW 5
8	OUTPUT SIGMOID, DR OUT ROW +1, CLR ACC
9	MULT-ACCUM DR ROW -1, WR ROW 6 //计算并写入候选:
10	MULT-ACCUM DR ROW +1, WR ROW 7 // $C' = \text{TANH}(W_c * X + U_c * H + B_c)$
11	ADD_W_ACC WR ROW 8
12	OUTPUT TANH, WR OUT ROW 9, CLR ACC
13	MULT-ACCUM DR ROW +1, WR ROW 9 //计算并写入新的单元状态:
14	MULT-ACCUM DR ROW +1, WR ROW 11 // $C = I * C' + F * C$
15	OUTPUT PASSTHRU, WR OUT ROW 11 //输出 C 以供下一时间步骤使用
16	OUTPUT TANH, WR OUT ROW 10, CLR ACC //输出 TANH (C) 以供 H 计算
17	MULT-ACCUM DR ROW -3, WR ROW 12 //计算并写入输出门:
18	MULT-ACCUM DR ROW +1, WR ROW 13 // $O = \text{SIGMOID}(W_o * X + U_o * H + B_o)$
19	ADD_W_ACC WR ROW 14
20	OUTPUT SIGMOID, DR OUT ROW +1, CLR ACC
21	MULT-ACCUM DR ROW +3, WR ROW 10 //计算并写入单元输出:
22	OUTPUT PASSTHRU, DR OUT ROW +2, CLR ACC // $H = O * \text{TANH}(C)$
23	LOOP 1

图48

↙ 121

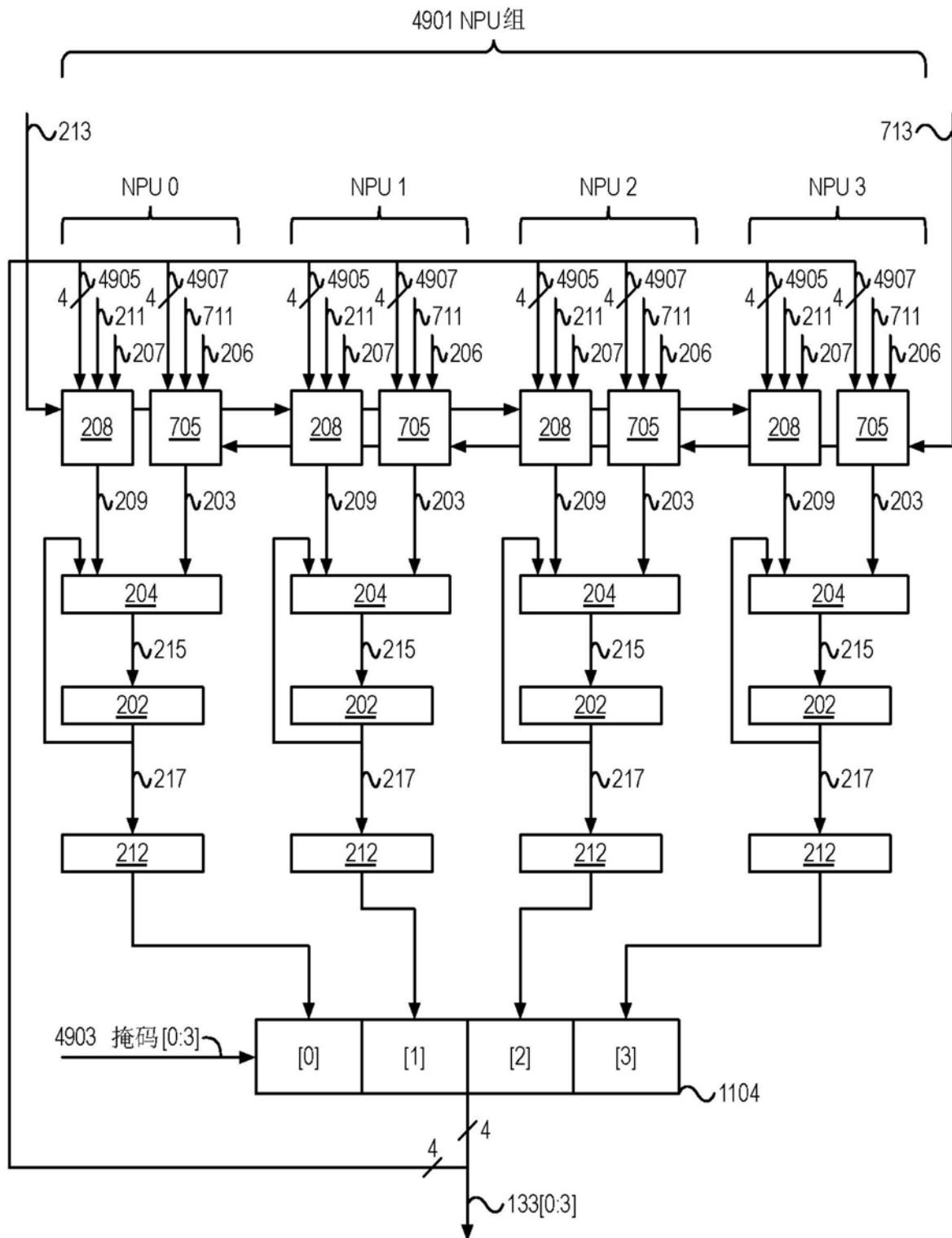


图49

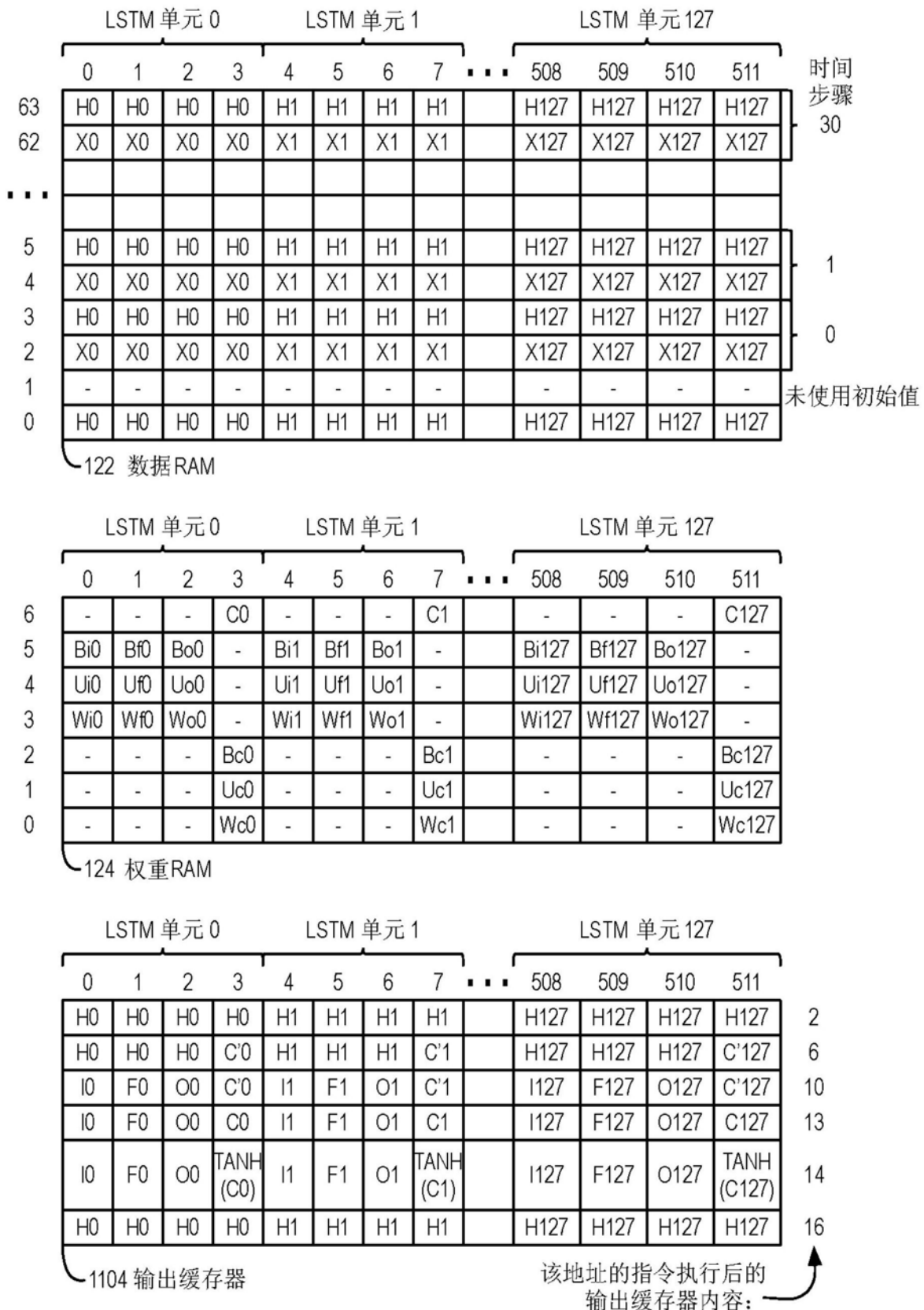


图50

地址	指令
0	INITIALIZE NPU, CLR ACC, LOOPCNT = 31, DR OUT ROW = 1 // 31个时间步骤
1	ADD_D_ACC DR ROW 0 // 将初始输出放置于累加器
2	OUTPUT PASSTHRU, NOP, CLR ACC // 将初始输出放置于输出缓存器
3	MULT-ACCUM DR ROW +2, WR ROW 0 // $C' = W_c * X$
4	MULT-ACCUM OUTBUF[0], WR ROW 1 // $C' = W_c * X + U_c * H$
5	ADD_W_ACC WR ROW 2 // $C' = W_c * X + U_c * H + B_c$
6	OUTPUT TANH, NOP, MASK[0:2], CLR ACC // 只将 C' 输出至输出缓存器
7	MULT-ACCUM DR ROW +0, WR ROW 3 // $I = W_i * X$; $F = W_f * X$; $O = W_o * X$
8	MULT-ACCUM OUTBUF[0], WR ROW 4 // $I = W_i * X + U_i * H$; $F = W_f * X + U_f * H$; $O = W_o * X + U_o * H$
9	ADD_W_ACC WR ROW 5 // $I = W_i * X + U_i * H + B_i$; $F = W_f * X + U_f * H + B_f$; $O = W_o * X + U_o * H + B_o$
10	OUTPUT SIGMOID, NOP, MASK[3], CLR ACC // 只将 I, F, O 输出至输出缓存器
11	MULT-ACCUM OUTBUF[0], OUTBUF[3] // $C = I * C'$
12	MULT-ACCUM OUTBUF[1], WR ROW 6 // $C = I * C' + F * C$
13	OUTPUT PASSTHRU, WR OUT ROW 6, MASK[0:2] // 输出 C (供下一个时间步骤使用); 不清除 ACC
14	OUTPUT TANH, NOP, MASK[0:2], CLR ACC // 只将 TANH (C) 输出至输出缓存器
15	MULT-ACCUM OUTBUF[2], OUTBUF[3] // $H = O * \text{TANH}(C)$
16	OUTPUT PASSTHRU, DR OUT ROW +2, CLR ACC
17	LOOP 3

图51

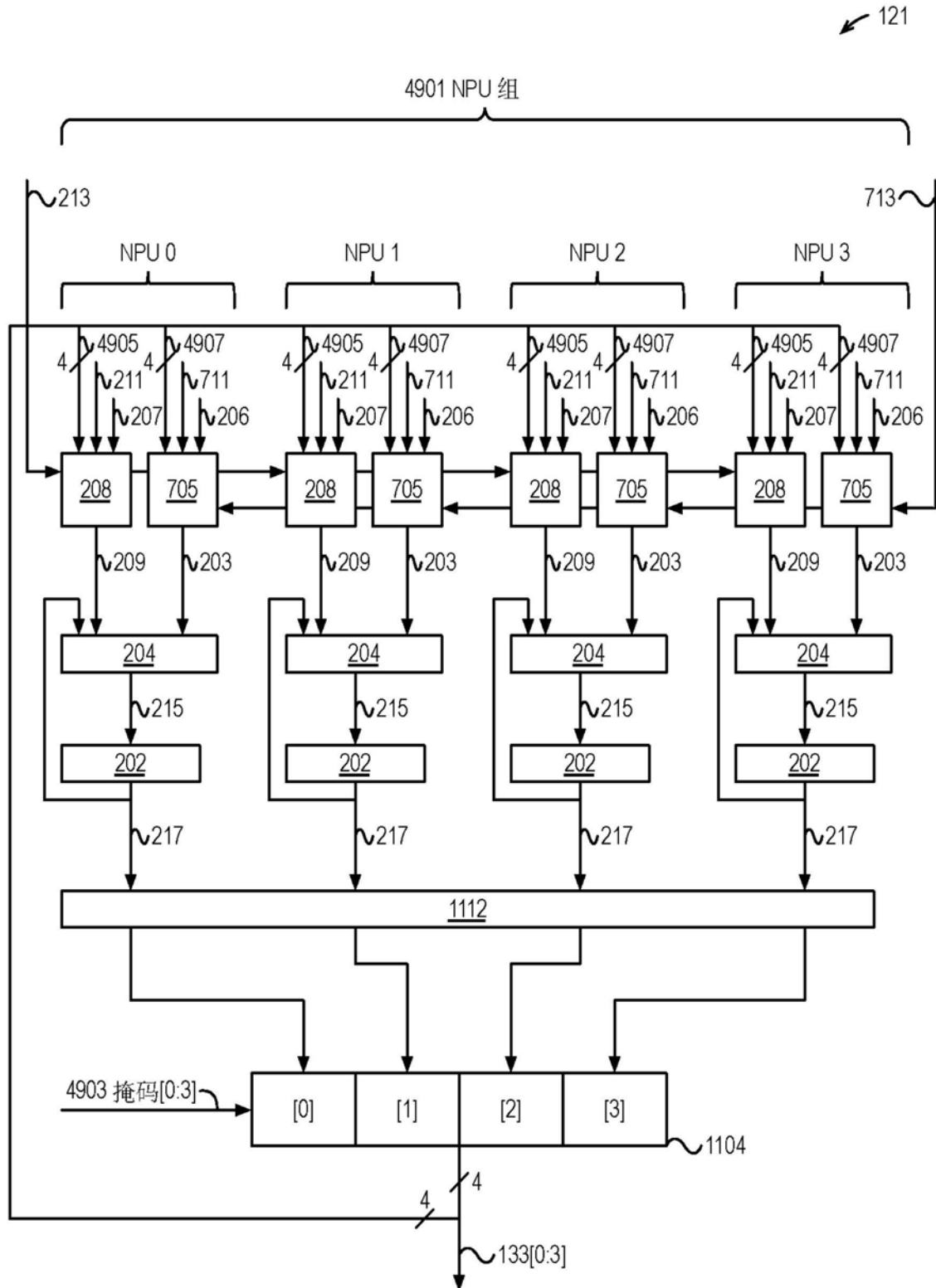


图52

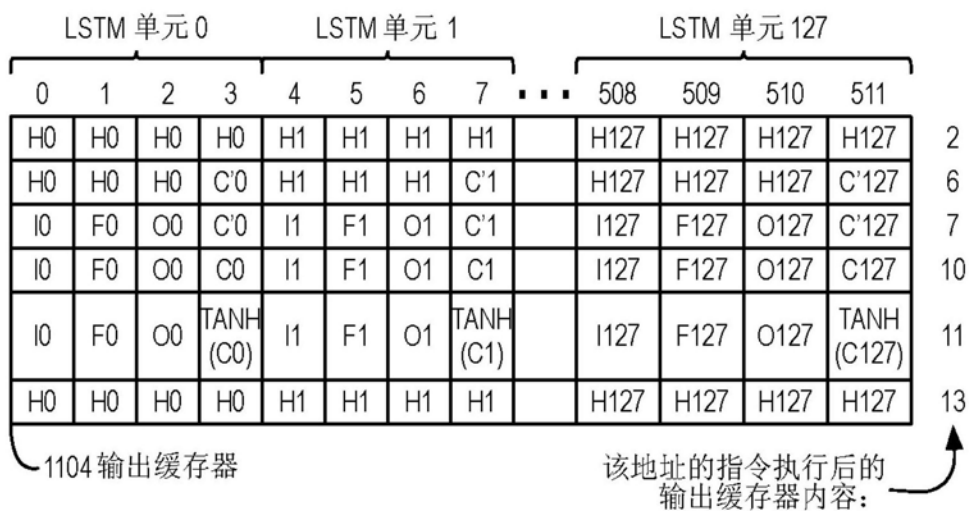
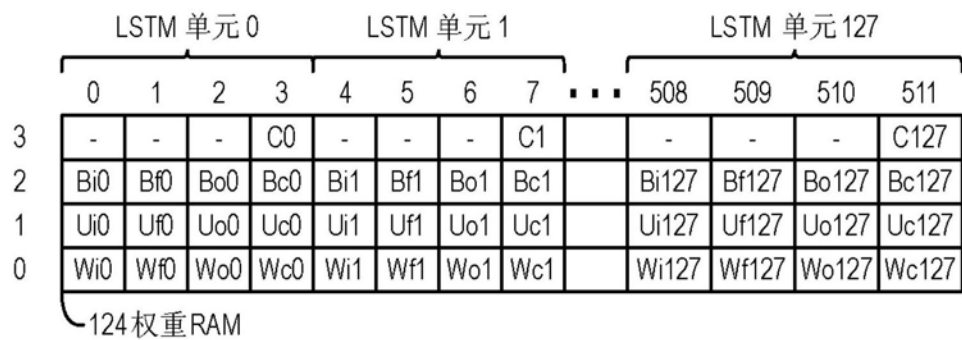
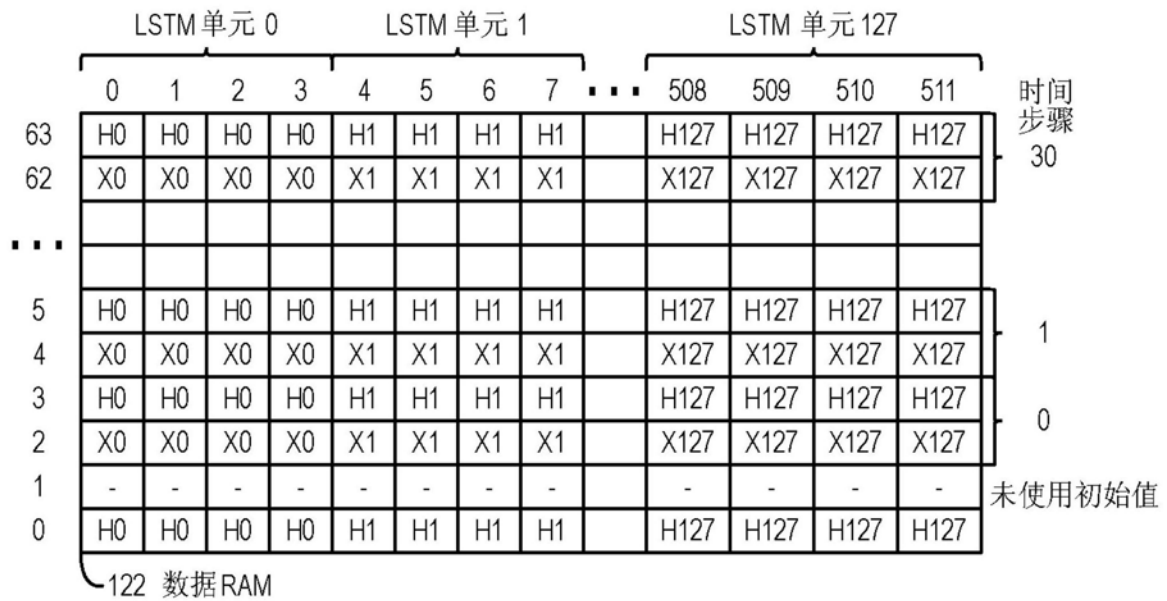


图53

地址	指令
0	INITIALIZE NPU, CLR ACC, LOOPCNT = 31, DR OUT ROW = 1 // 31个时间步骤
1	ADD_D_ACC DR ROW 0 // 将初始输出放置于累加器
2	OUTPUT PASSTHRU, NOP, CLR ACC // 将初始输出放置于输出缓存器
3	MULT-ACCUM DR ROW +2, WR ROW 0 // $I=Wi*X$; $F=Wf*X$; $O=Wo*X$; $C'=Wc*X$
4	MULT-ACCUM OUTBUF[0], WR ROW 1 // $I=Wi*X+Ui*H$; $F=Wf*X+Uf*H$; $O=Wo*X+Uo*H$; $C'=Wc*X+Uc*H$
5	ADD_W_ACC WR ROW 2 // $I=Wi*X+Ui*H+Bi$; $F=Wf*X+Uf*H+Bf$; $O=Wo*X+Uo*H+Bo$; $C'=Wc*X+Uc*H+Bc$
6	OUTPUT TANH, NOP, MASK[0:2] // 只将 C' 输出至输出缓存器; 不清除 ACC
7	OUTPUT SIGMOID, NOP, MASK[3], CLR ACC // 只将 I, F, O 输出至输出缓存器
8	MULT-ACCUM OUTBUF[0], OUTBUF[3] // $C = I * C'$
9	MULT-ACCUM OUTBUF[1], WR ROW 3 // $C = I * C' + F * C$ (即, 当前单元状态)
10	OUTPUT PASSTHRU, WR OUT ROW 3, MASK[0:2] // 输出 C (供下一个时间步骤使用); 不清除 ACC
11	OUTPUT TANH, NOP, MASK[0:2], CLR ACC // 只将 $TANH(C)$ 输出至输出缓存器
12	MULT-ACCUM OUTBUF[2], OUTBUF[3] // $H = O * TANH(C)$
13	OUTPUT PASSTHRU, DR OUT ROW +2, CLR ACC
14	LOOP 3

图54

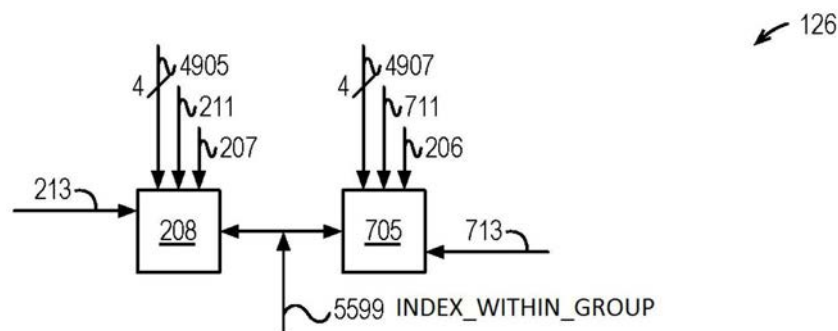


图55

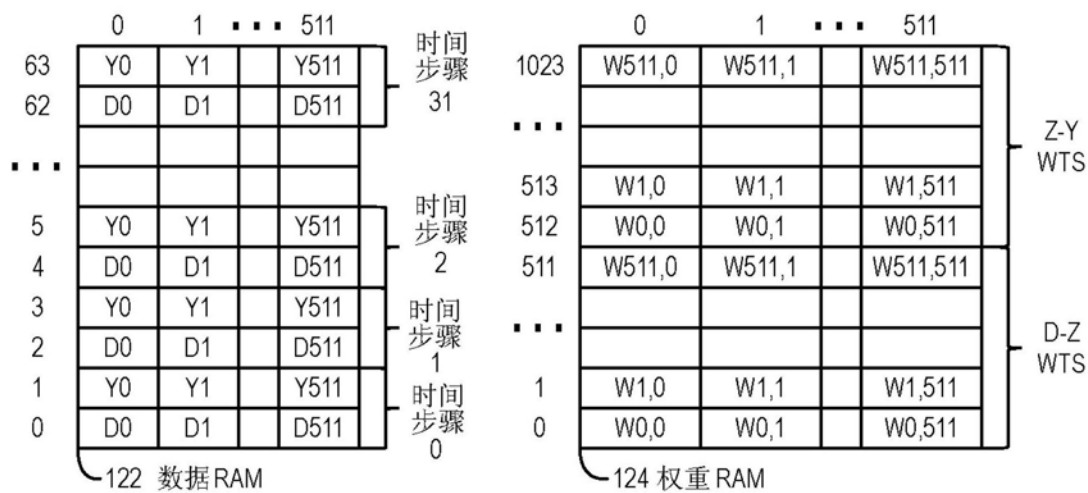


图56

地址	指令
0	INITIALIZE NPU, LOOPCNT = 32, DR ROW = -2, DR OUT ROW = -1, CLR_ACC // 32个时间步骤
1	OUTPUT PASSTHRU, NOP, CLR ACC //将初始 C 值（零）放置于输出缓存器
2	ADD_D_ACC OUTBUF[SELF] //累加输出缓存器内的 C 值
3	ADD_D_ACC ROTATE, COUNT = 511 //不清除 ACC
4	MULT-ACCUM DR ROW +2, WR ROW 0 //计算 Z; 将 D 累加至累加后的 C
5	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
6	OUTPUT PASSTHRU, NOP, CLR ACC //将 Z 放置于输出缓存器
7	MULT-ACCUM OUTBUF[SELF], WR ROW 512 //计算 Y 和 C
8	MULT-ACCUM ROTATE, WR ROW +1, COUNT = 511
9	OUTPUT ACTIVATION FUNCTION, DR OUT ROW +2 //将 Y 写入数据 RAM
10	OUTPUT PASSTHRU, NOP, CLR ACC //将新的 C 值放置于输出缓存器
11	LOOP 2 //将 Z 层输出至 DRAM

图57

↙ 121

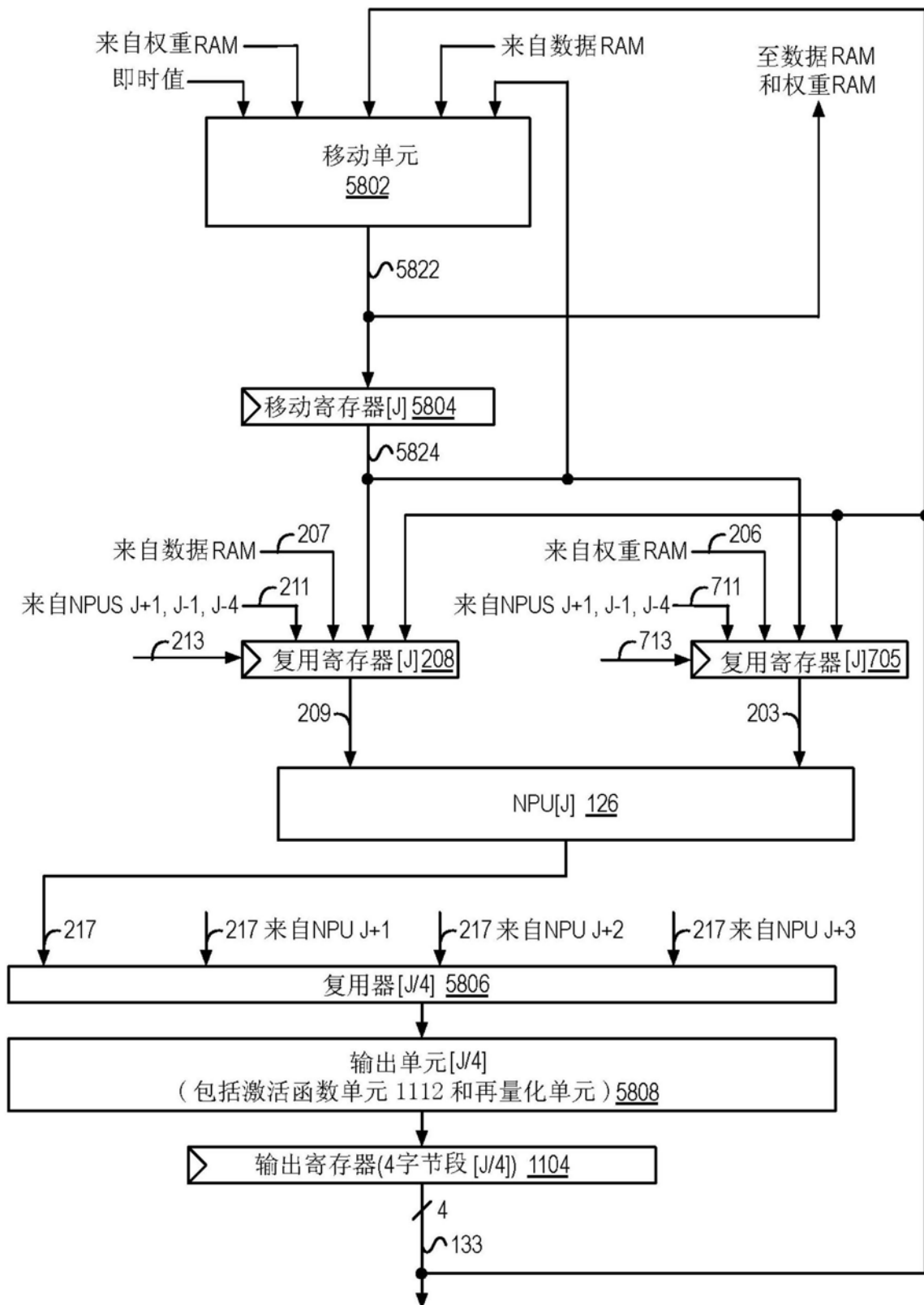


图58

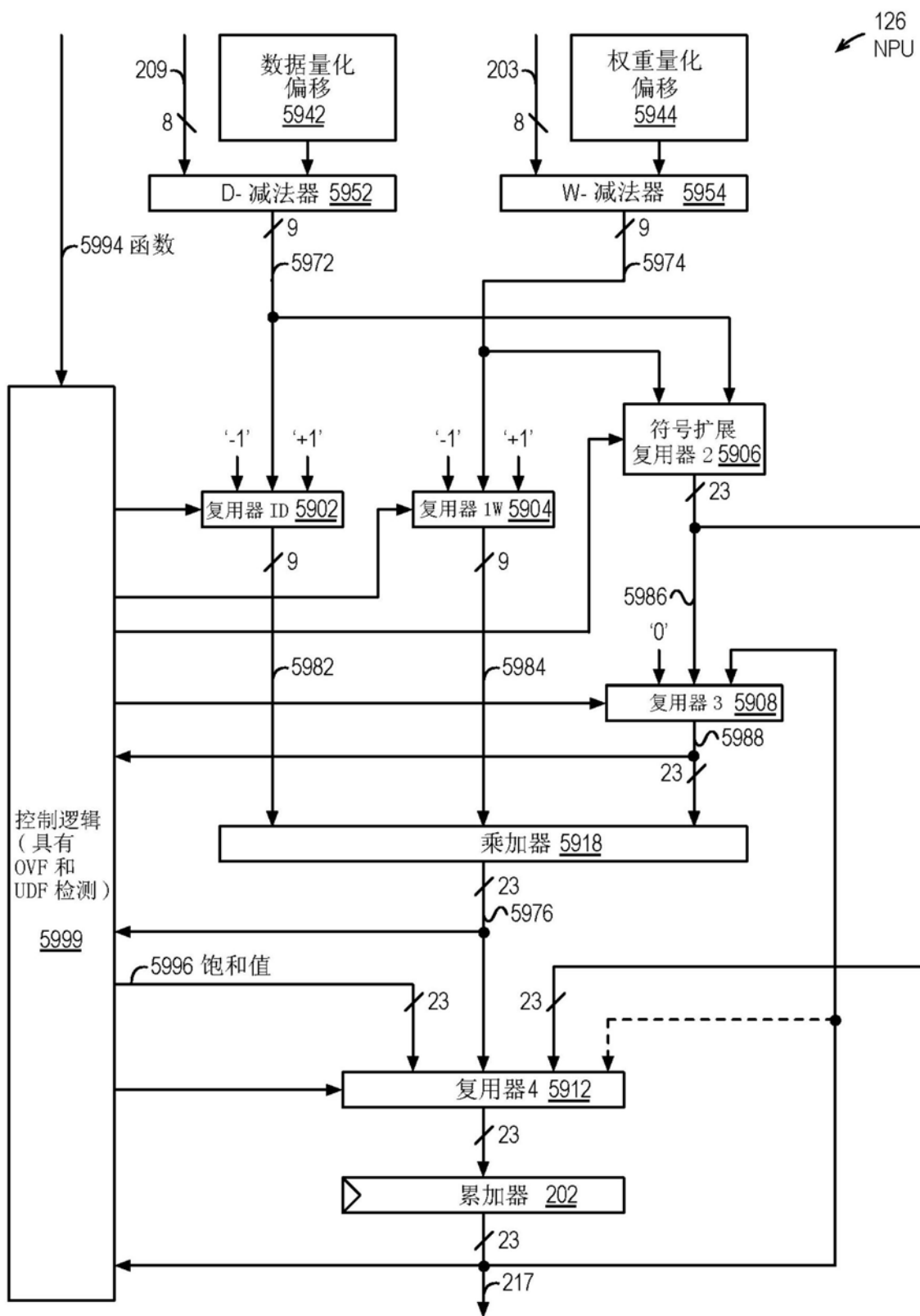


图59

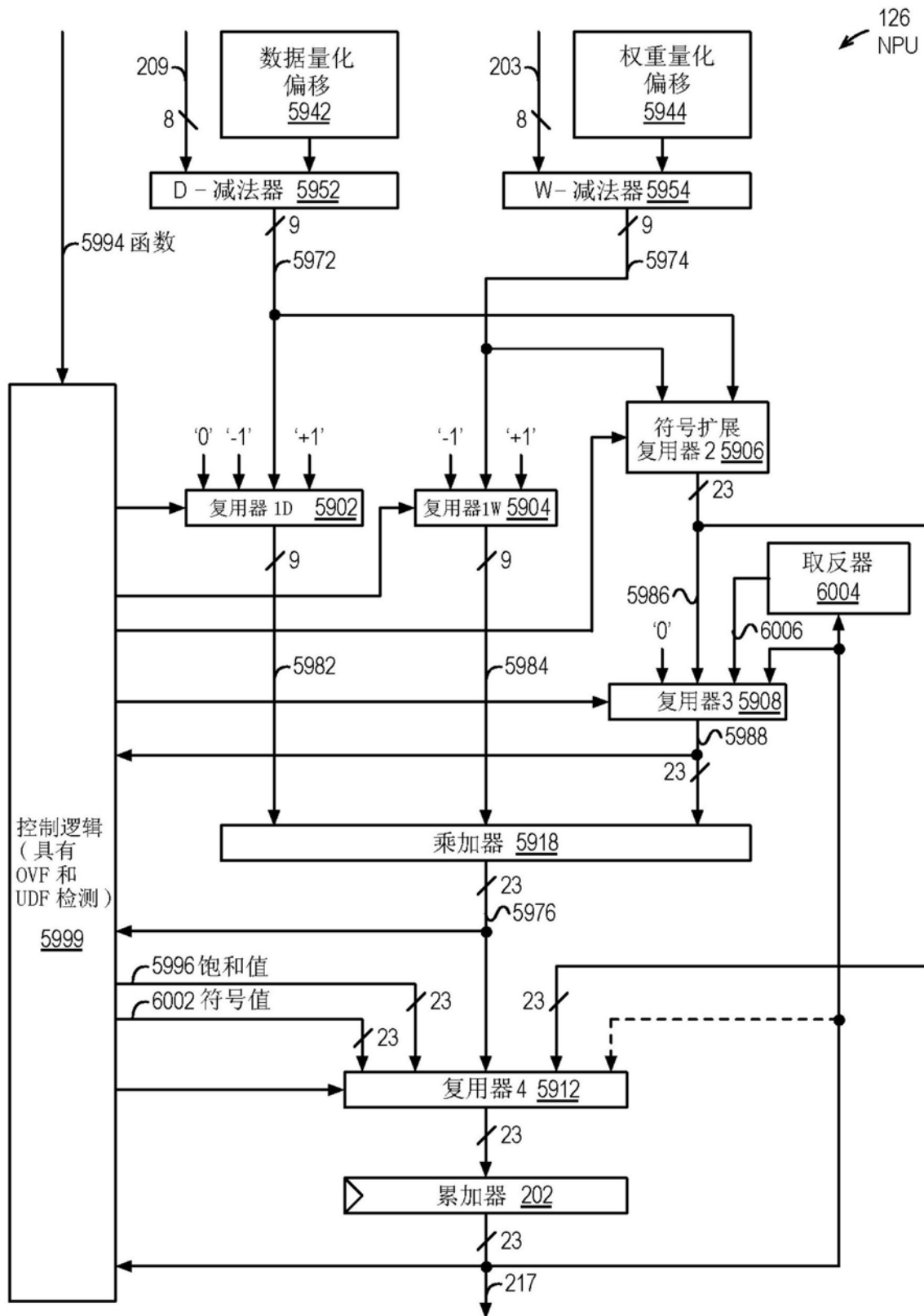


图60

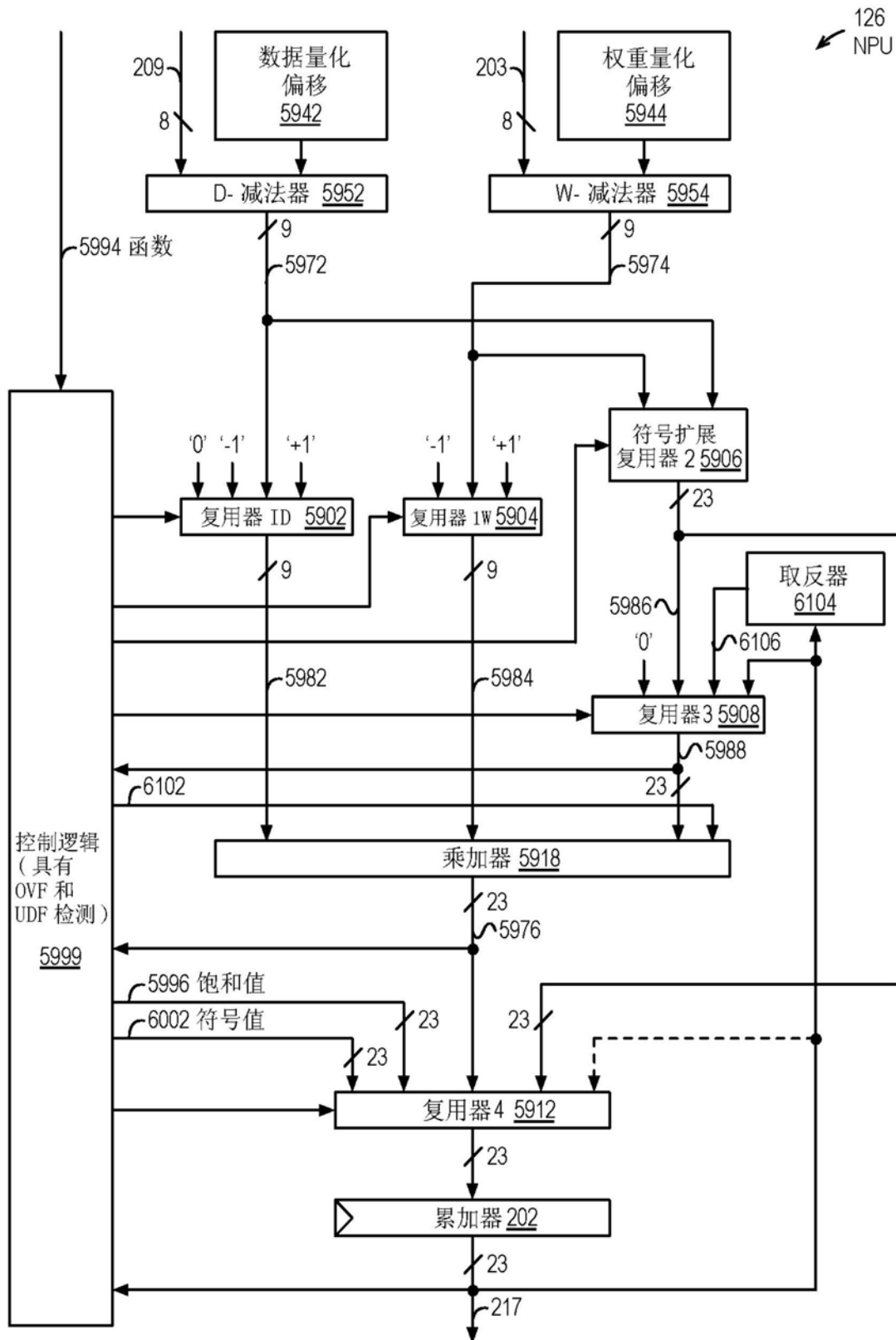


图61

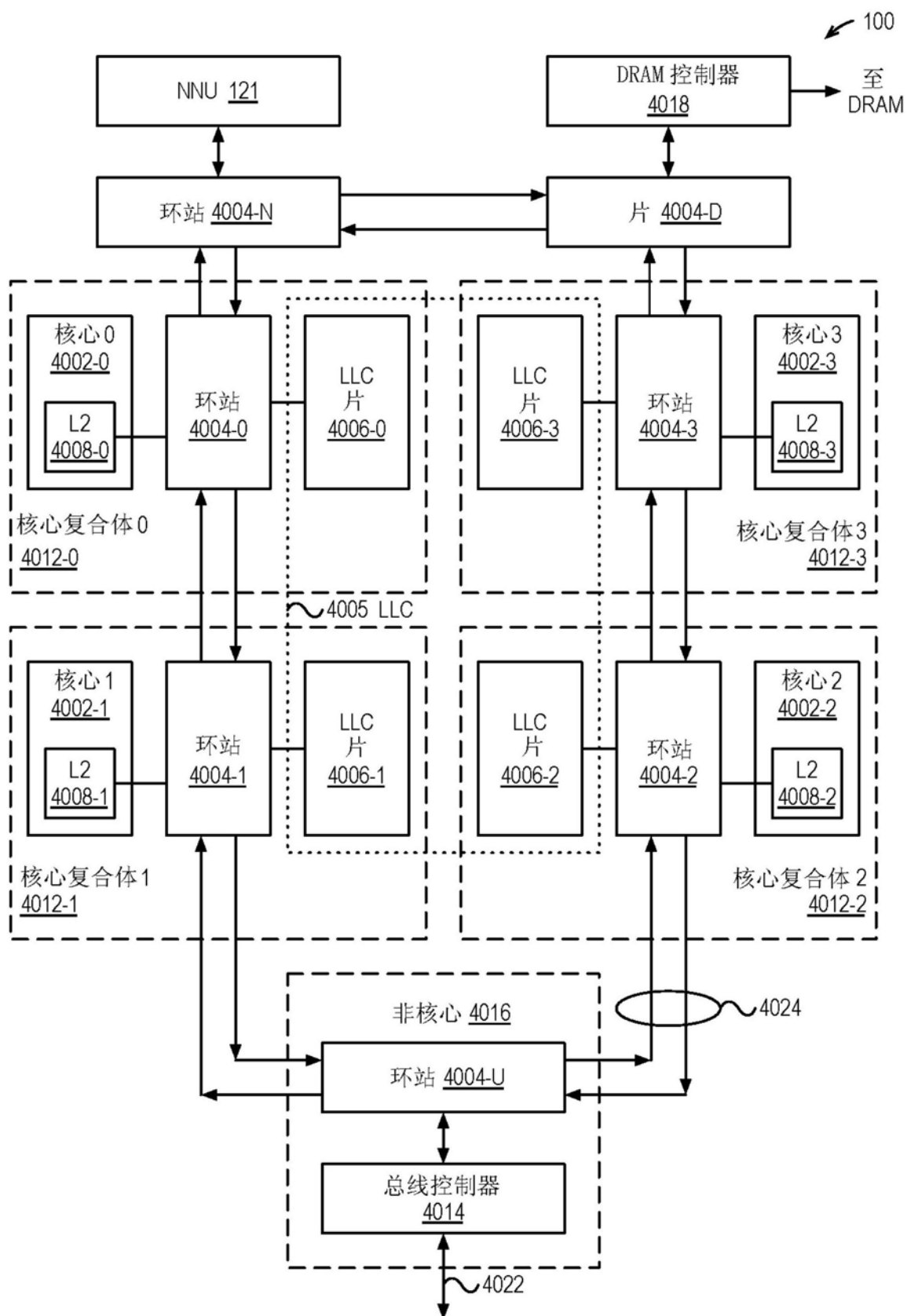


图62

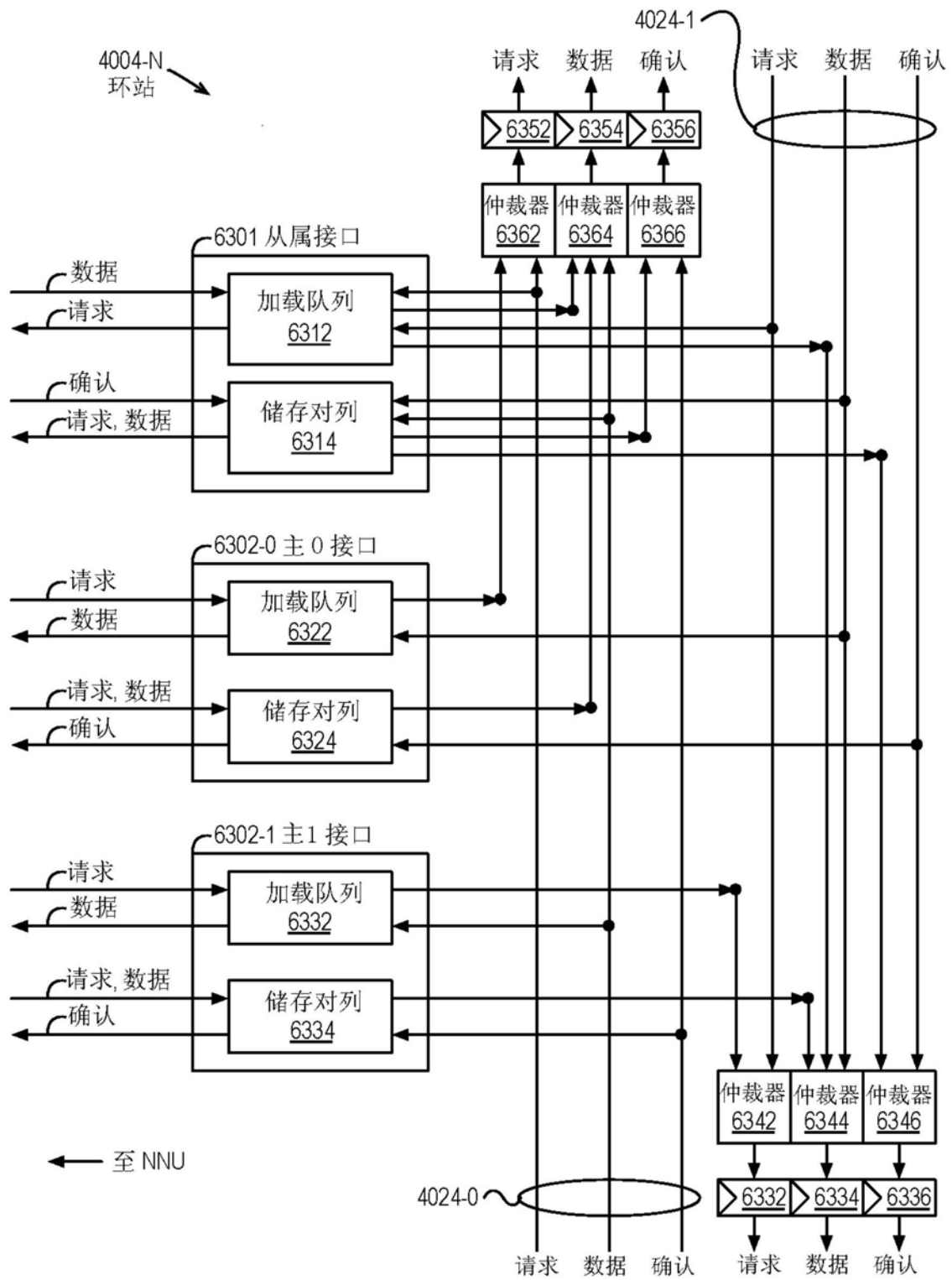


图63

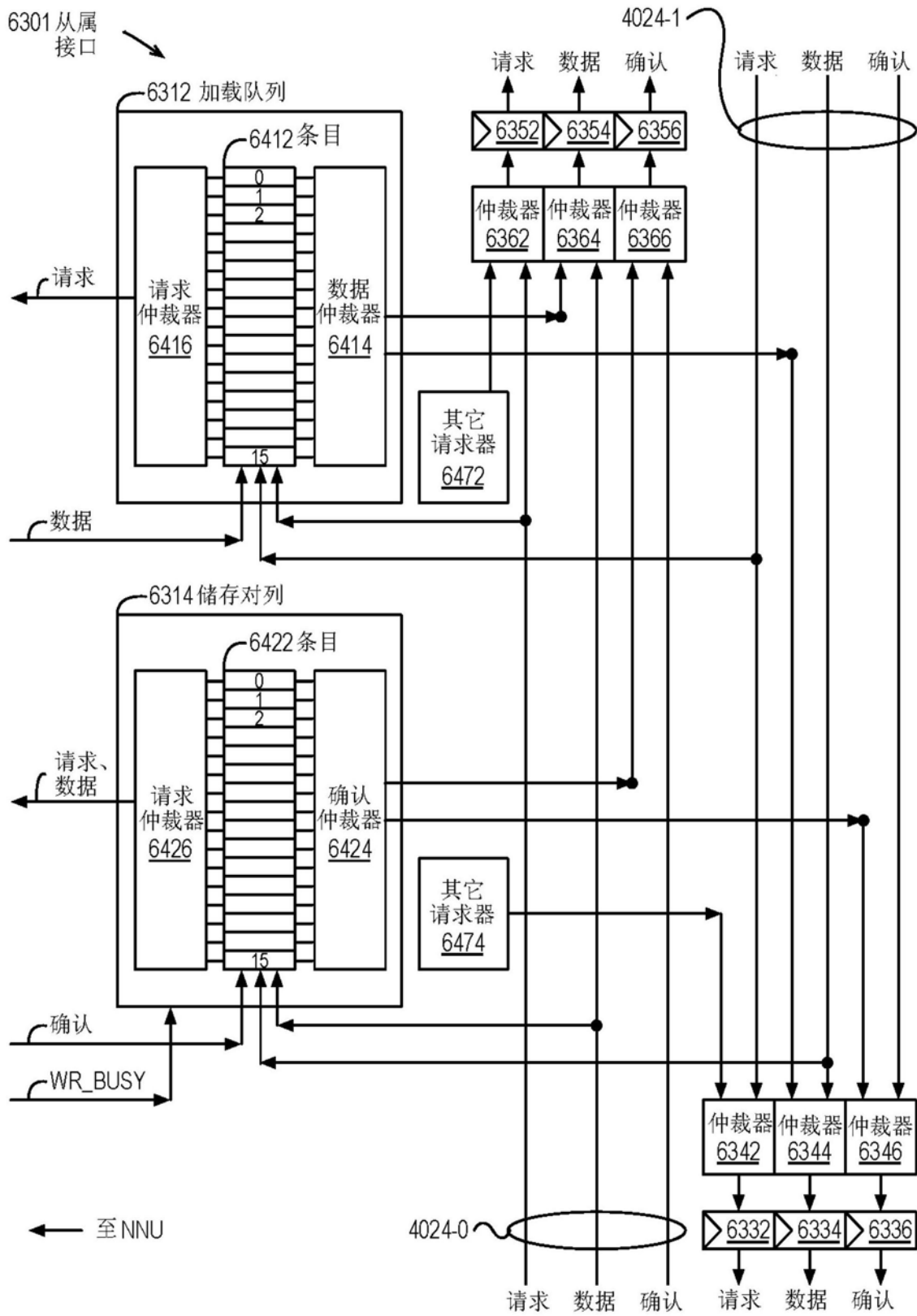


图64

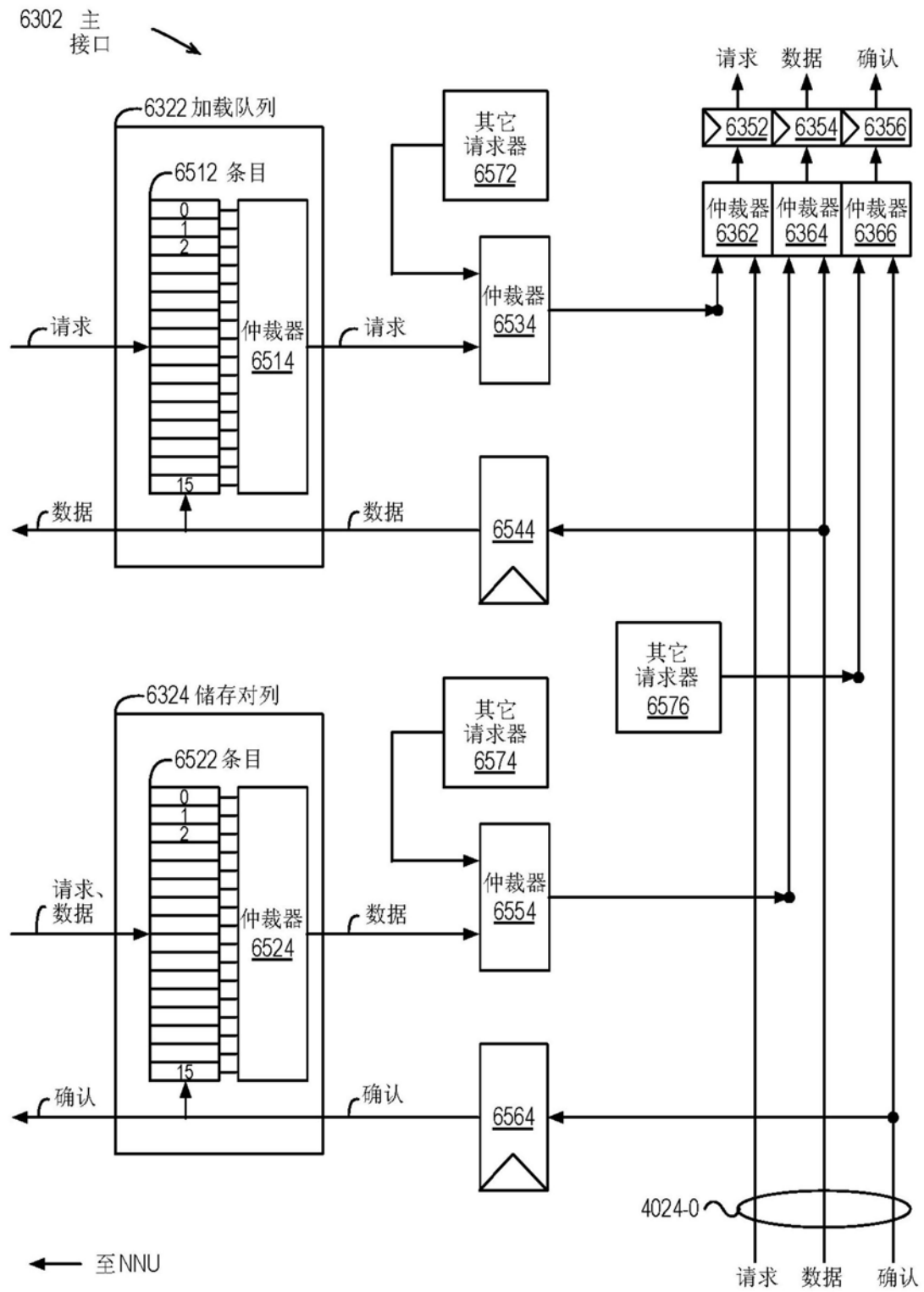


图65

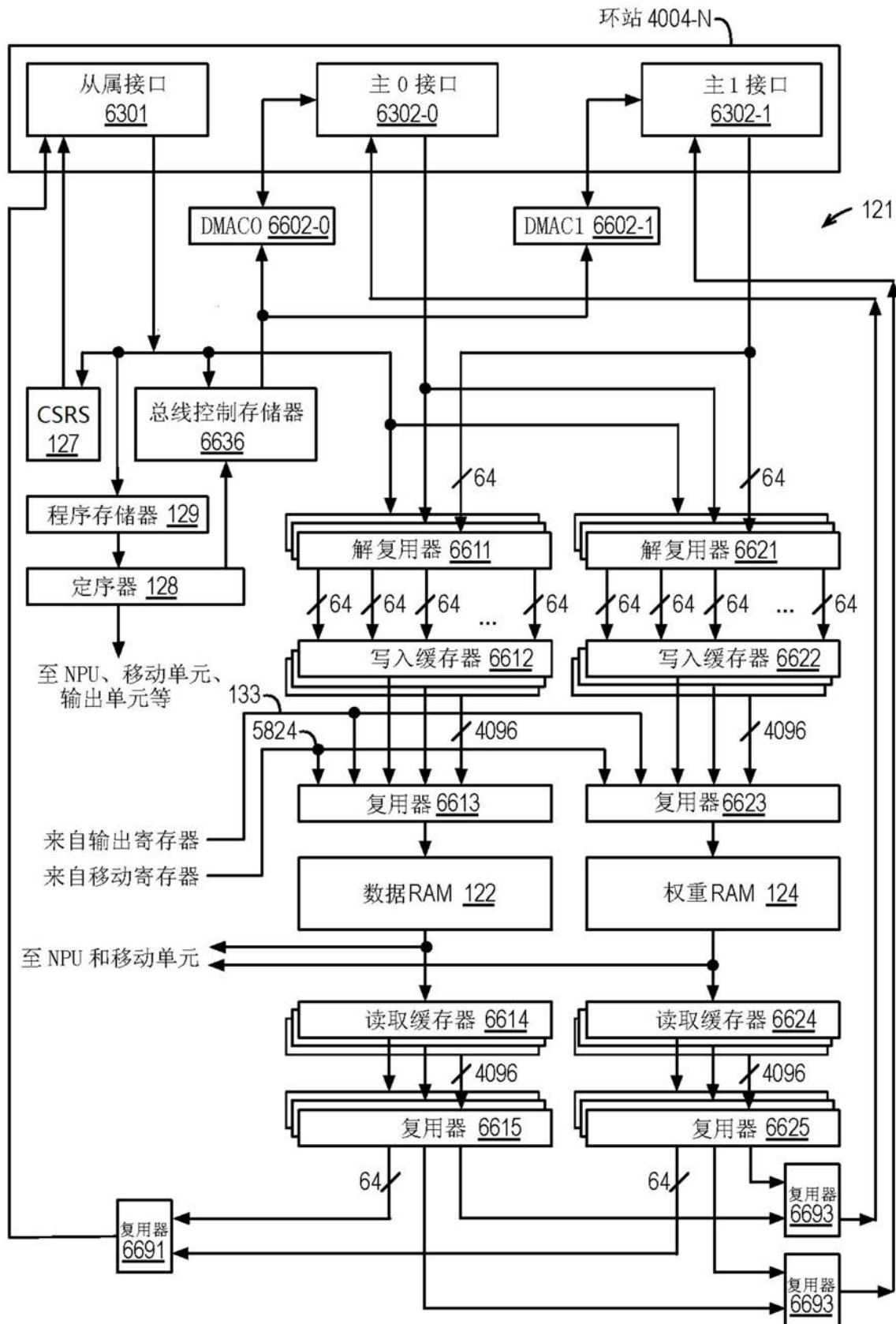


图66

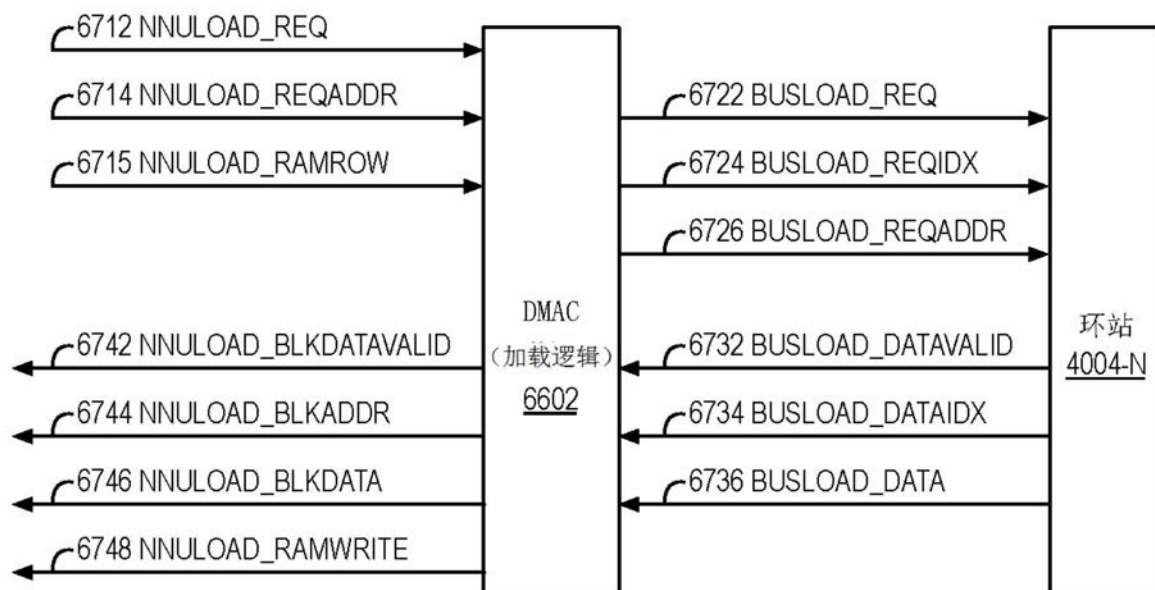


图67

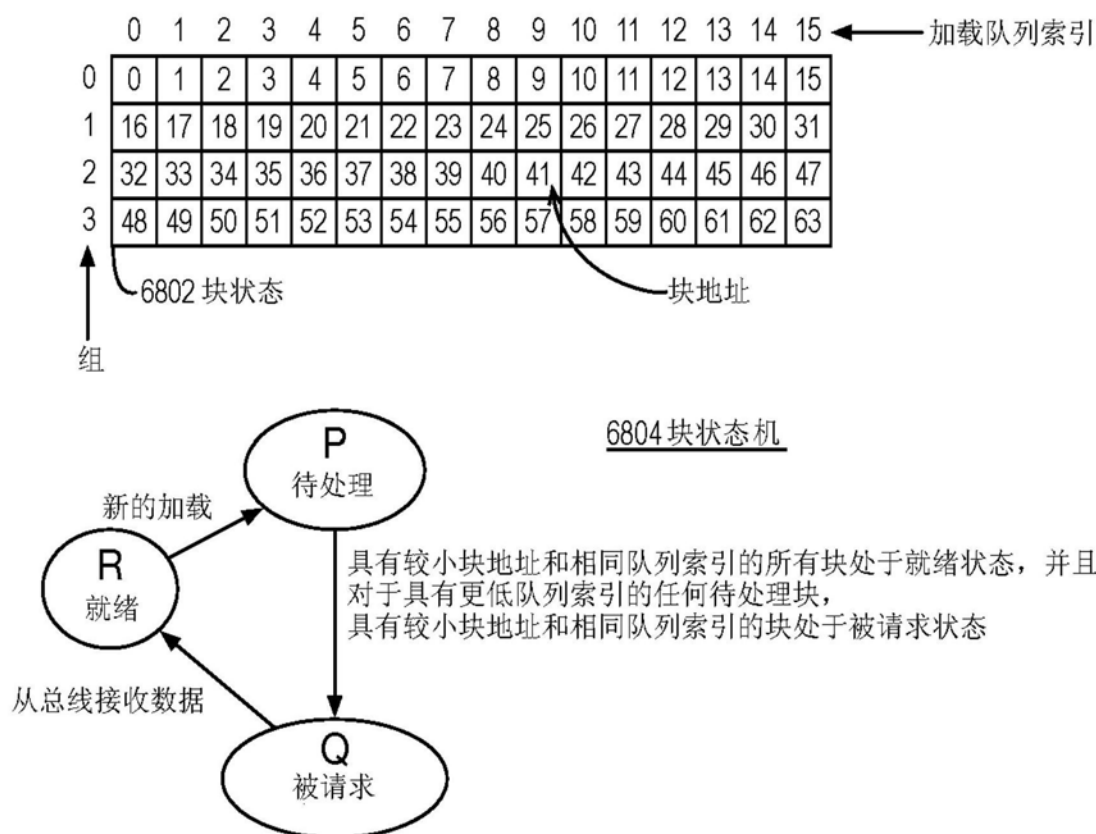


图68

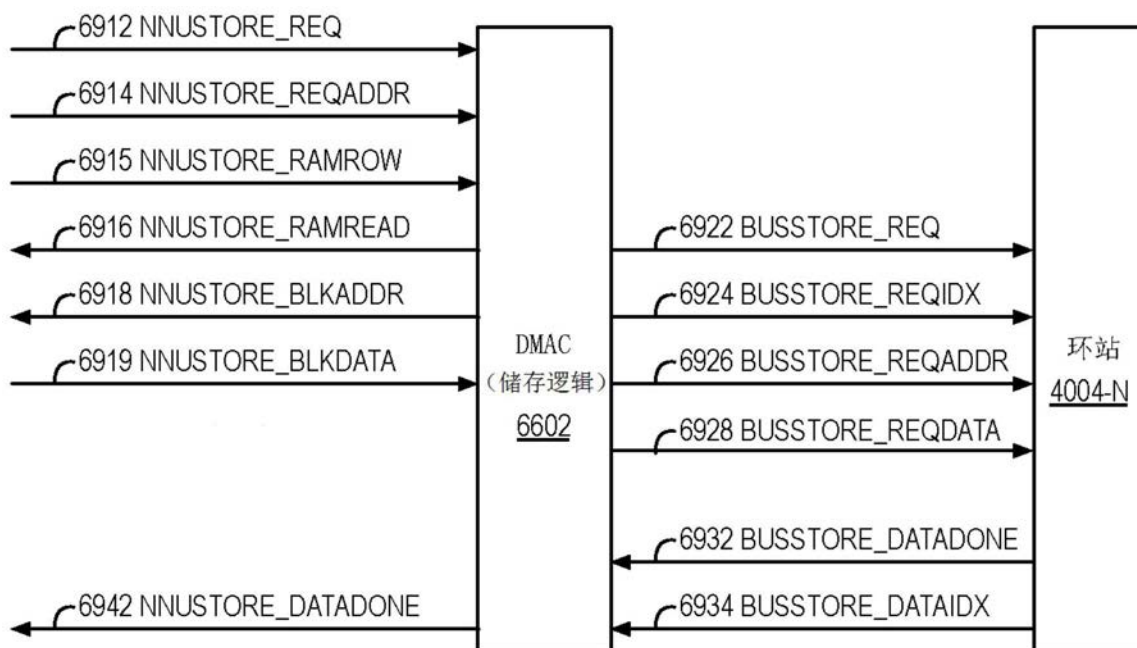


图69

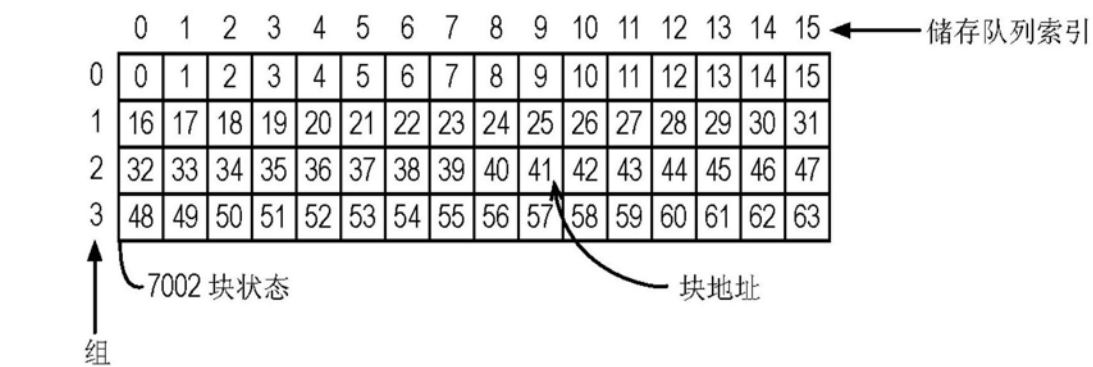


图70

基址寄存器 0 <u>7198-0</u>	基址寄存器 1 <u>7198-1</u>	基址寄存器 2 <u>7198-2</u>	基址寄存器 3 <u>7198-3</u>
--------------------------	--------------------------	--------------------------	--------------------------

7104 DMA 控制字 (DCW)

环形总线 地址 <u>7112</u>	RAM 地址 <u>7114</u>	存储器 空间 <u>7116</u>	方向 <u>7118</u>	计数 <u>7122</u>	相干 <u>7124</u>	中断 <u>7126</u>	等待标记 <u>7128</u>
---------------------------	--------------------------	--------------------------	-------------------	-------------------	-------------------	-------------------	---------------------

图71

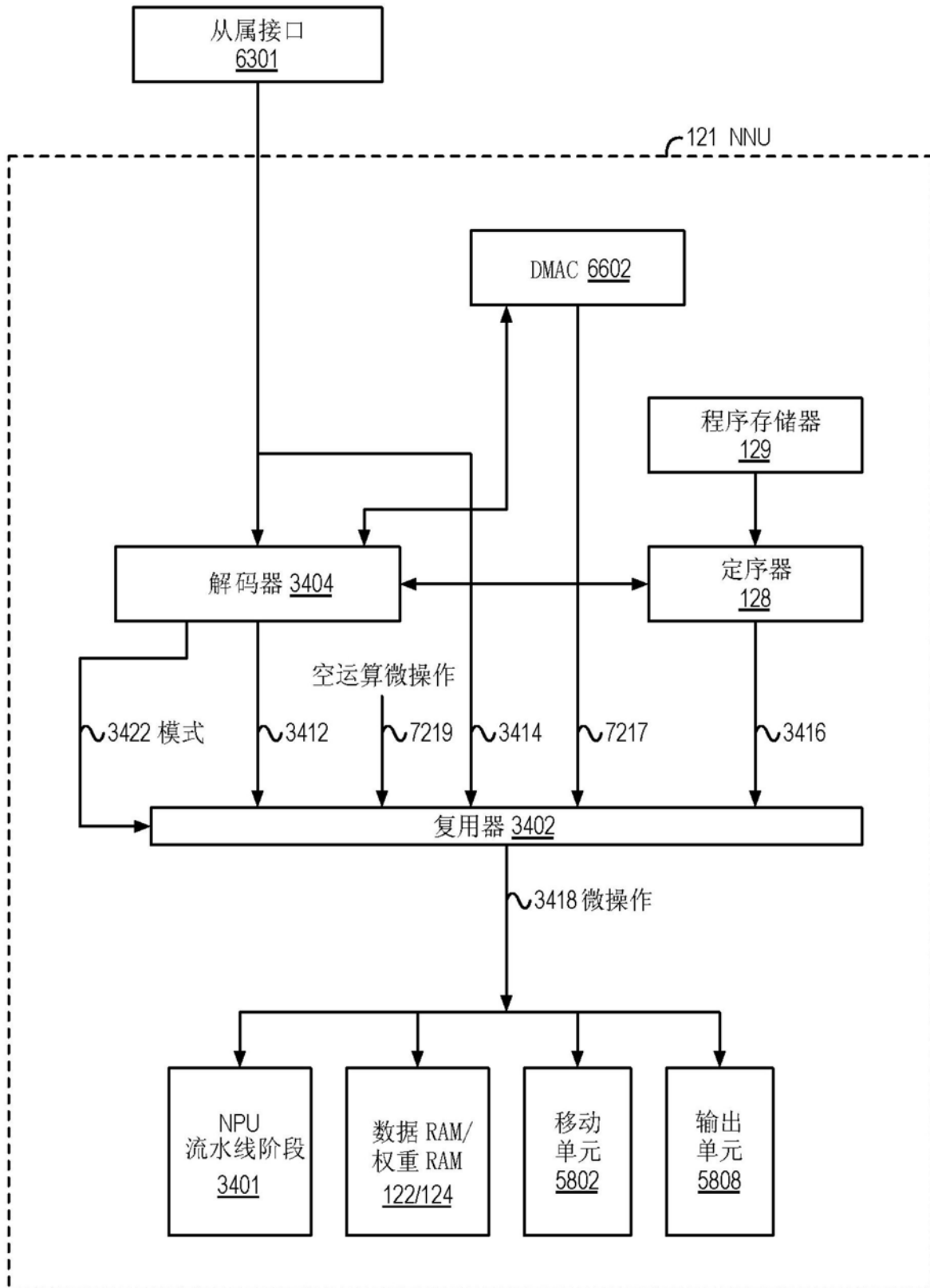


图72