ambricon：神经网络的指令集体系结构

**摘要**—神经网络（NN）是一系列新兴机器学习和模式重建应用的模型。神经网络技术通常运用在通用处理器（如CPU和GPGPU）上，这些处理器通常不会节能，因为他们投入了过多的硬件资源来灵活地支持各种工作负载。因此，最近已经提出用于神经网络的专用硬件加速器来提高能量效率。然而，这样的加速器是为一小组共享类似计算模式的NN技术而设计的，它们采用直接对应于NN的高级功能块（例如层）的复杂和信息性指令（控制信号），甚至NN作为一个整体。尽管对于有限类似的NN技术简单易行，但指令集缺乏灵活性，使得这种加速器设计不能以足够的灵活性和效率支持各种不同的NN技术。

在这篇论文中，我们提出了一种新的用于NN加速器的领域专用指令集架构（ISA），称为Cambricon，它是一种集标量、向量、矩阵、逻辑、数据传输和控制指令于一体的加载-存储架构。对现有的NN技术进行综合分析，我们对总共十种不同的具有代表的NN技术的评估表明，Cambricon在广泛的NN技术上表现出强大的描述能力，并且提供比诸如x86，MIPS和GPGPU的通用ISA更高的代码密度。与最新的神经网络加速器设计DaDianNao[5]（只能容纳3种神经网络技术）相比，我们采用台积电65纳米技术实现的基于Cambricon的加速器原型的延迟/功耗/面积开销只有微不足道，涵盖了10个不同的NN基准。

**一、导言**

人工神经网络（简称神经网络）是最初受到神经科学启发的机器学习技术的一个大家族，在过去的十年中一直在向更深更大的结构发展。虽然计算成本很高，但 [22],[25],[26],[27]所示的深度学习神经网络技术已经成为了广泛应用的先进技术（如模式识别[ 8]和网络搜索[17]），有些甚至在特定的任务上取得了人性化的表现，比如ImageNet识别[23]和Atari 2600视频游戏[33]。

传统上，NN技术是在由CPU和GPGPU组成的通用平台上执行的，这些平台通常不具备能源效率，因为这两种处理器投入了过多的硬件资源来灵活地支持各种工作负载[7]、[10]、[45 ]。硬件加速器已经被定制为神经网络，最近已经被作为节能的替代方法进行了研究[3]、[5]、[11]、[29]、[32]。这些加速器通常采用直接指定高级功能块（例如层类型：卷积/池/分类器）或甚至整个NN的高级和信息性指令（控制信号），而不是低级计算操作（例如点积），并且它们的解码器可以针对每条指令进行完全优化。

虽然对于一小部分类似的NN技术（因此是一个小的指令集）而言简单易行，但是这种加速器的指令译码器的设计/验证复杂性和面积/功率开销将很容易变得不可接受地大，虽然对于一小部分类似的NN技术（因此是一个小指令集）而言，简单易用，但这种加速器的指令译码器的设计/验证复杂性和面积/功率开销很容易变得难以接受地大，灵活地支持各种不同的NN技术的需求导致指令集的显着扩展。因此，这种加速器的设计只能有效地支持NN技术的一个小子集，它们共享非常相似的计算模式和数据局部性，但是不能处理现有NN技术之间的显着差异。例如，最先进的神经网络加速器DaDianNao[5]可以有效地支持多层感知器（MLPs）[50]，但是不能容纳神经元彼此完全连接的Boltzmann机器[BM][39]。因此，ISA设计仍然是一个根本性的尚未解决的挑战，极大地限制了现有NN加速器的灵活性和效率。

在本文中，我们研究了NN加速器的ISA设计，受RISC ISA设计原则[37]的成功启发：（a）首先，分解描述高级神经网络功能块的复杂和信息指令（例如层）转换成对应于低级计算操

陈云基（cyj@ict.ac.cn）是本文负责通讯方面的作者。

作（例如点积）的较短指令，使得加速器具有更广泛的应用范围，因为用户现在可以使用低级操作来组装新的高级功能块，这些功能块在新的NN技术中是不可或缺的；（b）其次，简单和简短的指令大大降低了指令译码器的设计/验证复杂度和功率/面积。

本研究的结果是一种新颖的用于NN加速器的ISA，称为Cambricon。Cambricon是一个加载-存储体系结构，其指令全部为64位，并且包含64个用于标量的32位通用寄存器（GPR），主要用于控制和寻址。为了支持密集的、连续的、可变长度的向量/矩阵数据访问（这是NN技术中常见的），可忽略不计的面积/功耗，Cambricon不使用任何向量寄存器文件，而是将数据保存在片上暂存器内存中，对程序员/编译器是可见的。没有必要在片上存储器中实现多个端口（如寄存器文件中那样），因为同时访问不同的存储区以地址的低阶位进行分解足以支持NN技术（第四节）。与性能受限于寄存器文件的有限宽度的SIMD不同，Cambricon有效地支持更大和可变的数据宽度，因为片上暂存器存储器组可以容易地制造得比寄存器文件更宽。

我们评估Cambricon总共十种代表性但不同的NN技术（MLP [2]、CNN [28]、RNN [15]、LSTM [15]、Autoencoder [49]、稀疏自动编码器[49]、BM [39]、RBM [39]、SOM [48]、HNN [36]），并观察到Cambricon比通用ISAs，如MIPS（13.38倍）、x86（9.86倍）和GPGPU（6.41倍）提供更高的代码密度。与最新的最先进的NN加速器设计DaDianNao[5]（其仅能适应3种NN技术）相比，我们基于Cambricon的加速器原型在TSMC 65nm技术中实现的延迟、功耗、 和面积管理费用（分别为4.5％/ 4.4％/ 1.6％），涵盖10个不同的NN基准。

我们在这项工作中的主要贡献如下：1）我们提出了一种新颖的轻量级ISA，对NN技术具有强大的描述能力;；2）我们对现有神经网络技术的计算模式进行全面研究； 3）我们通过采用台积电65纳米技术的第一款基于Cambricon的加速器来评估Cambricon的有效性。

本文的其余部分安排如下。第2部分简要讨论了Cambricon所遵循的一些设计准则，并对Cambricon进行了概述。第三部分介绍了Cambricon的计算和逻辑指令。第四部分介绍了一个Cambricon加速器的原型。第五部分根据经验对Cambricon进行评估，并与其他ISA进行比较。第六部分讨论了Cambricon扩展技术的可能性。第七节介绍相关工作。第八节总结全文。

**二、建议的ISA概述**

在本节中，我们首先描述我们提出的ISA的设计指南，然后简要概述ISA。

**A.设计指南**

为了为神经网络设计一个简洁、灵活和高效的ISA，我们根据它们的计算操作和内存访问模式分析各种神经网络技术，基于此我们在做出具体设计决策之前提出了一些设计准则。

**•数据级并行性。**我们观察到，在大多数NN技术中，神经元和突触数据被组织为层，然后以均匀/对称的方式进行操纵。在适应这些操作时，由矢量/矩阵指令启用的数据级并行性可能比传统标量指令的指令级并行性更高效，并且对应于更高的代码密度。 因此，Cambricon的重点将是数据级并行。

**•定制的矢量/矩阵指令。**尽管有许多线性代数库（如BLAS库[9]）成功地涵盖了广泛的科学计算应用，但对于NN技术，在这些代数库中定义的基本操作并不一定是有效和高效的选择甚至是多余的）。更重要的是，传统的线性代数库没有涵盖NN技术的许多常见操作。例如，BLAS库不支持向量的元素级指数计算，也不支持在突触初始化，退出[8]和受限玻尔兹曼机（RBM）[39]中生成随机向量。因此，我们必须全面地为现有的NN技术定制一组小而有代表性的矢量/矩阵指令集，而不是简单地从现有的线性代数库中重新实现矢量/矩阵运算。

•**使用片上Scratchpad存储器**。我们观察到NN技术通常需要对矢量/矩阵数据进行密集的、连续的和可变长度的访问，因此使用固定宽度的耗能矢量寄存器文件不再是最具成本效益的选择。在我们的设计中，我们用片上暂存器替换矢量寄存器文件，为每个数据访问提供灵活的宽度。这通常是神经网络中数据级并行性的高效选择，因为神经网络中的突触数据通常很大而且很少被重用，从而降低了矢量寄存器文件带来的性能增益。

**B.Cambricon概述**。我们按照II-A部分提出的指导方针设计Cambricon，并在表I中提供了Cambricon的概述。Cambricon是一个只允许用加载/存储指令访问主存储器的加载存储架构。 Cambricon包含用于标量的64个32位通用寄存器（GPR），可用于片上暂存器存储器的寄存器间接寻址以及暂时保持标量数据。

表1. Cambricon指令概述



**指令类型。**Cambricon包含四种类型的指令：计算、逻辑、控制和数据传输指令。尽管不同指令的有效位数可能不同，但指令长度对于存储器对齐和加载/存储/解码逻辑的设计简单性固定为64位。在本节中，我们只提供控制和数据传输指令的简要介绍，因为它们与其相应的MIPS指令类似，尽管它们已经被调整为适合NN技术。然而，对于计算指令（包括矩阵，向量和标量指令）和逻辑指令，细节将在下一节（第三节）中提供。

**控制指令。**控制指令。Cambricon有两个控制指令、跳转和条件分支，如图1所示。跳转指令通过立即数或GPR值指定偏移量，并将累加到程序计数器（PC）。除了偏移之外，条件分支指令指定预测器（存储在GPR中），并且通过预测器与零之间的比较确定分支目标（PC + {偏移}或PC + 1）。

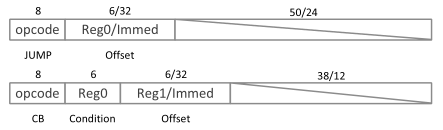


图1. 头部：跳转指令。底部：条件分支（CB）指令。

**数据传输指令。** Cambricon中的数据传输指令支持可变的数据大小，以便灵活地支持矩阵和向量计算/逻辑指令（参见第III节有关这些指令）。具体来说，这些指令可以从/到主存储器的可变长度数据块（由数据传输指令中的数据宽度操作数指定）加载/存储到/从片上暂存器存储器中，或者将数据 片暂存器和标量GPRs。图2举例说明了Vector LOAD（VLOAD）指令，它可以从主存储器加载一个Vsize大小的矢量到矢量暂存器存储器，其中主存储器中的源地址是保存在GPR中的基地址与即时编号之和的地址中。 Vector STORE（VSTORE），Matrix LOAD（MLOAD）和Matrix STORE（MSTORE）指令的格式类似于VLOAD。

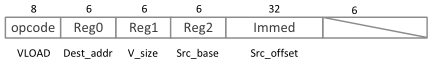


图2.矢量加载（VLOAD）指令。

**片上Scratchpad存储器。** Cambricon不使用任何向量寄存器文件，而是直接将数据保存在片上暂存器内存中，这对程序员/编译器是可见的。换句话说，Cambricon中的片内暂存器存储器的作用与传统ISA中的向量寄存器文件的作用相似，向量操作数的大小不再受固定宽度向量寄存器文件的限制。因此，矢量/矩阵大小在Cambricon指令中是可变的，唯一显著的限制是在相同指令中的矢量/矩阵操作数不能超过暂存器存储器的容量。如果超过，则编译器会将长向量/矩阵分解为短片段/块，并生成多条指令来处理它们。

就像32x512b向量寄存器已经被烧入Intel AVX-512 [18]一样，片上的容量矢量和矩阵指令的存储器必须在Cambricon中修复。更具体地说，Cambricon将存储器容量固定为向量指令64KB，矩阵指令768KB。然而，Cambricon没有对暂存器存储器的银行数量施加特别的限制，为微架构级的实现留下了很大的自由度。

**三、计算/逻辑指令**

在神经网络中，大多数算术运算（例如，加法、乘法和激活函数）可以被汇总为矢量运算[10]、[45]，根据我们的定量观察，比率可以高达99.992％最先进的卷积神经网络（GoogLeNet）赢得2014 ImageNet竞赛（ILSVRC14）[43]。同时我们还发现，GoogLeNet中的向量运算（如点积运算）中的99.791％可以进一步聚合为矩阵运算（如向量 - 矩阵乘法）。简而言之，神经网络可以自然分解为标量、矢量和矩阵运算，而ISA设计必须有效地利用潜在的数据级并行性和数据局部性。

**A.矩阵说明**

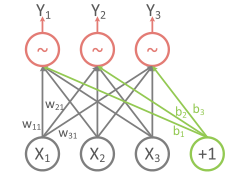


图3.神经网络的典型操作

我们对现有的神经网络技术进行了全面的综述，为Cambricon设计了六条矩阵指令。在这里，我们以一个众所周知的具有代表性的神经网络（MLP）[50]为例，说明它是如何被矩阵指令支持的。从技术上讲，MLP通常具有多个层次，每个层次根据已知的一些神经元（即输入神经元）计算一些神经元（即输出神经元）的值。更具体地说，图3中的输出神经元（i = 1,2,3）可以被计算为，其中是第j个输入神经元，（i = 1，2，3）是第i个输出神经元和第j个输入神经元之间的权重，（i = 1,2,3）是第i个输出神经元的偏差，f是激活函数。输出神经元可以计算为一个向量y =（，，）：

y = f（W x + b）， (1)

其中x =（，，）和b =（，，）分别是输入神经元和偏差的向量，W =（）是权重矩阵，f是激活函数的元素版本（参见第三-B节）。

方程式（1）中的关键一步是计算W x，它将由Cambricon中的矩阵多向量（MMV）指令执行。我们在图4中说明这个指令，其中Reg0指定向量输出（Voutaddr）的基址暂存器存储器地址; Reg1指定矢量输出（Voutsize）的大小； Reg2、Reg3和Reg4分别指定矩阵输入的基地址（Minaddr）、矢量输入的基地址（Vinaddr）和矢量输入的大小（Vinsize，注意它是可变的）。 MMV指令可以支持任意比例的矩阵向量乘法，只要所有的输入和输出数据都可以同时保存在暂存器中。我们选择使用专用MMV指令来计算W x，而不是将其分解为多个向量点积，因为后一种方法需要额外的工作量（例如，显式同步，对同一地址的并发读/写请求）来重用输入向量x在M的不同行向量中，效率较低。



图4.矩阵多向量（MMV）指令。

与前馈情况不同，MMV指令不再为神经网络的后向训练过程提供有效的支持。更具体地说，众所周知的后向传播（BP）算法的关键步骤是计算梯度向量[20]，其可以被表示为乘以矩阵的向量。如果我们用MMV指令来实现它，我们需要一个额外的指令来实现matrix转置，这在数据移动中是相当昂贵的。为了避免这种情况，Cambricon提供了一个直接适用于后向训练过程的矢量多矩阵（VMM）指令。除了操作码之外，VMM指令与MMV指令具有相同的字段。

另外，在训练NN时，权重矩阵W通常需要用W = W +ηW递增更新，其中η是学习率，W是估计为两个向量的外积。 Cambricon提供一个外部产品（OP）指令（输出是矩阵），矩阵多标量（MMS）指令和矩阵加矩阵（MAM）指令来协同执行权重更新。此外，Cambricon还提供了矩阵减矩阵（Matrix-Subtract-Matrix，MSM）指令来支持限制玻尔兹曼机器（RBM）中的权重更新[39]。

**B.矢量指令**

使用方程以图1为例，可以观察到前面小节中定义的矩阵指令仍然不足以执行所有的计算。我们仍然需要将W x的向量输出和偏移向量b相加，然后执行元素方式激活到W x + b。

尽管Cambricon直接提供了向量加法向量（Vector-Add-Vector，VAV）指令，它需要多个指令来支持元素激活。在不失一般性的情况下，我们以广泛使用的sigmoid 396激活为例，。对输入向量（比如说a）的每个元素执行的元素方式的sigmoid激活可以分解为3个连续的步骤，分别由3条指令支持：

1.计算输入失量a中的每个元素（，i = 1，...，n）的指数。Cambricon提供了矢量指数（VEXP）指令，用于指向矢量元素的指数。

2.将常量1添加到向量的每个元素（，...，）。 Cambricon提供了一个Vector-Add- Scalar（VAS）指令，其中标量可以是一个立即的或由GPR指定的。

3.对于每个向量索引i = 1、…、n，将除以1 +。 Cambricon提供了一个Vector-Div- Vector（VDV）指令，用于向量之间的元素分割。

然而，S形并不是现有的NN所使用的唯一激活函数。为了实现各种激活函数的元素版本，Cambricon提供了一系列矢量算术指令，如矢量多矢量（VMV），矢量子矢量（VSV）和矢量对数（VLOG）。在设计硬件加速器时，与不同超越函数（如对数函数，三角函数和反三角函数）有关的指令可以高效地重用相同的功能块（包括加法，移位和查表操作），使用CORDIC技术[24]。此外，还有一些激活功能（例如max（0，a）和| a |）部分依赖于逻辑运算（比如比较），我们将在第三节中介绍相关的Cambricon指令（例如矢量比较指令）

此外，随机矢量生成是许多神经网络技术（例如丢失[8]和随机采样[39]）中常见的重要操作，但在为科学设计的传统线性代数库中不被视为是必需的计算（例如，BLAS库不包括这个操作）。 Cambricon提供了一个专用指令（Random-Vector，RV），用于生成一个随机数向量，服从均匀分布[0,1]。给定均匀的随机向量，借助矢量算术指令和矢量比较指令，我们可以使用Ziggurat算法[31]，进一步产生服从其他分布（例如，高斯分布）的随机矢量。

**C.逻辑指令**

最先进的NN技术利用一些包含比较或其他逻辑运算的操作。最大池操作就是这样一种操作（参见图5a中的一个例子），该操作寻找在合并窗口内的神经元之间具有最大输出的神经元，并且针对不同输入特征映射中的相应合并窗口重复该操作（见图5b）。 Cambricon支持使用矢量大于合并（VGTM）指令的最大池操作，参见图6.VGTM指令通过比较输入向量（Vout）的每个元素，通过比较输入向量的相应元素0（ Vin0）和输入向量-1（Vin1），即Vout [i] =（Vin0 [i]> Vin1 [i]）→Vin0 [i]：Vin1 [i]。我们在第三部分E节中展示了最大池操作的Cambricon代码，该代码将同一输入向量中所有输入要素图的相同位置上的神经元聚合，迭代执行VGTM并获得最终结果（也可参见图5c的一个例证）。

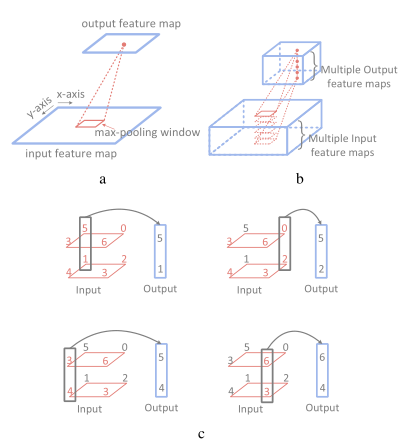


图5.最大池操作。

除矢量计算指令外，Cam-bricon还提供了矢量大于（VGT），矢量平均指令（VE），矢量与/或非指令（VAND / VOR / VNOT），标量比较和标量 逻辑指令来处理分支条件，即计算前述的条件分支（CB）指令的预测器。

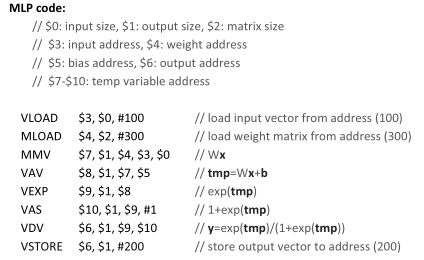


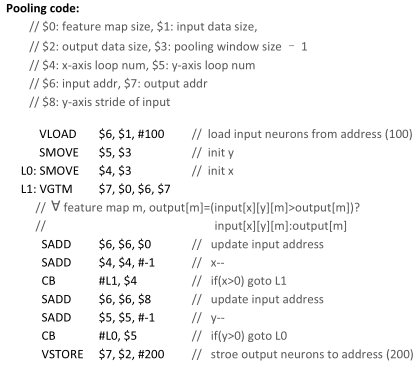
图6.矢量大于合并（VGTM）指令。

**D.标量指令**

尽管我们发现在Cambricon中只有0.008％的GoogLeNet算术运算不能被矩阵和矢量指令所支持，但也有神经元不可缺少的标量运算，如基本算术运算和标量超越函数。我们在表1中总结了它们，这些表格已被正式定义为Cambricon的标量指令。

**E.代码示例**

为了说明我们提出的指令集的用法，我们实现了三个简单而有代表性的神经网络组件，一个MLP前馈层[50]，一个池层[22]和一个玻尔兹曼机器（BM）层[39] bricon说明。为了简洁起见，我们省略了所有三层的标量加载/存储指令，并且仅显示用于汇聚层的单个汇集窗口（具有多个输入和输出特征映射）的程序片段。我们在图7中说明具体的Cambricon程序片段，我们观察到Cambricon的代码密度显着高于x86和MIPS（参见第五部分的综合评估）。



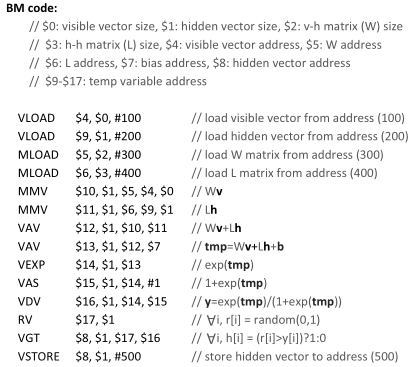
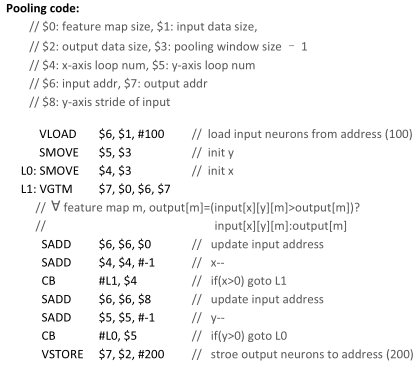


图7. MLP，池和BM的Cambricon程序片段。

**四、原型加速器**

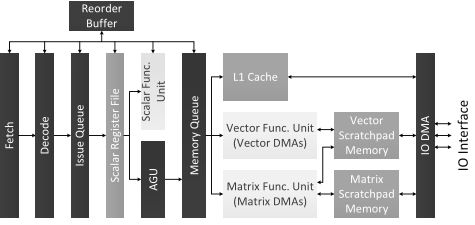


图8.基于Cambricon的原型加速器。

在本节中，我们将介绍一个Cambricon的原型加速器。我们在图8中说明了这个设计，它包含七个主要的指令流水线阶段：读取，解码，发行，寄存器读取，执行，回写和提交。因为我们发现这些经典技术已经足以反映出灵活性（第三部分-B1节），简洁性（第五部分-B2节）和效率（第五部分-B3节）的ISA。我们并没有试图探索新兴的技术（如3D叠加[51]和非易失性存储器[47]，[46] ）在我们的原型设计中，但留下了未来工作的探索，因为我们认为有前途的ISA必须易于实施，不应与新兴技术紧密结合。

如图8所示，在提取和解码阶段之后，将指令注入到有序发布队列中。从标量寄存器文件中成功读取操作数（标量数据或向量/矩阵数据的地址/大小）后，将根据指令类型将指令发送到不同的单元。控制指令和标量计算/逻辑指令将被发送到标量功能单元直接执行。在写回标量寄存器文件之后，只要已经成为最老的尚未执行的指令，就可以从重排缓冲器1提交这样的指令。

数据传输指令，矢量/矩阵计算指令和矢量逻辑指令可以访问一级缓存或暂存器存储器将被发送到地址生成单元（AGU）。这样的指令需要在有序存储器队列中等待，以便利用存储器队列中较早的指令来解决潜在的存储器依赖性2。之后，标量数据传输指令的加载/存储请求将被发送到L1缓存，矢量的数据传输/计算/逻辑指令将被发送到矢量功能单元，矩阵的数据传输/计算指令将被发送到矩阵功能单元。执行之后，这样的指令可以从内存队列中退出，然后只要它已经成为最老的未提交执行指令就从重新排序缓冲区提交

加速器实现了向量和矩阵的功能单元。矢量单元包含32个16位加法器，32个16位乘法器，并配有64KB暂存器。矩阵单元包含1024个乘法器和1024个加法器，它们被分成32个独立的计算块，以避免长距离数据移动时过度的电线拥塞和功耗。每个计算块都配有一个单独的24KB暂存器。32个计算块通过一个h-tree总线连接起来，用来向每个块广播输入值，并从每个块中收集输出值。

一个值得注意的Cambricon功能是它不使用任何向量寄存器文件，但保持片内暂存器内存的数据。为了高效地访问暂存器，原型加速器的矢量/矩阵功能单元集成了三个DMA，每个DMA对应一个指令的一个向量/矩阵输入/输出。另外，暂存器内存配有一个IO DMA。但是，每个暂存器存储器本身仅为每个存储库提供一个端口，但可能需要处理最多四个并发读/写请求。我们为暂存器设计了一个特定的结构来解决这个问题（见图9）。具体而言，我们

根据地址的低位两位将存储器分解为四个存储区，并通过交叉开关将它们与四个读/写端口连接，以确保不会同时访问存储区。得益于专用硬件支持，Cambricon不需要昂贵的多

端口向量寄存器文件，并且可以灵活有效地使用片上暂存器内存来支持不同的数据宽度。

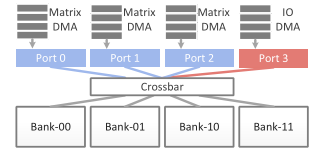


图9.矩阵暂存器内存的结构

**五、实验评估**

在本节中，我们首先描述评估方法，然后给出实验结果。

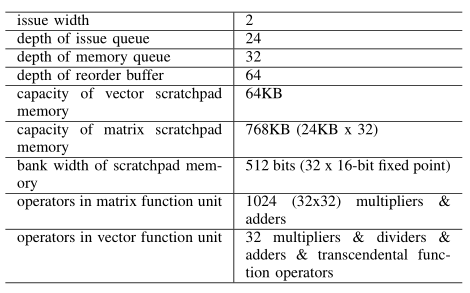
**A.方法论**

设计评估。我们用Synopsys Design Compiler使用TSMC 65nm GP标准VT库合成Cam- bricon的原型加速器（Cambricon-ACC，请参阅第IV节），将合成设计放置在Synopsys ICC编译器中，使用Synopsys VCS进行仿真和验证，根据模拟的数值变化转储（VCD）文件估算Synopsys Prime-Time PX的功耗。我们正计划推出一款MPW原型加速器，65纳米制程的面积预算为60平方毫米，目标工作频率为1 Ghz。因此，我们采用适度的功能单位大小和暂存记忆容量，以适应面积预算。表2显示了设计参数的细节。

1即使按顺序发布指令，我们也需要一个重排序缓冲区，因为不同指令的执行阶段可能需要大量不同的周期数。

2我们说如果两条指令访问 一个重叠的内存区域，则它们依赖于内存，并且至少有一条指令需要写入内存区域。

表2.我们的原型加速器的参数



基线。我们将Cambricon-ACC与三条基线进行比较。前两个是基于通用CPU和GPU的，最后一个是最先进的NN硬件加速器：

**•CPU。** CPU基准是支持256位SIMD的x86-CPU（Intel Xeon E5-2620,2.10GHz，64GB内存）。我们使用Intel MKL库[19]为CPU基线实现向量和矩阵原语，GCC v4.7.2使用选项“-O2 -lm -march = native”编译所有基准，以启用SIMD指令。

**•GPU。**GPU基准是一个现代化的GPU卡（在28nm工艺中，NVIDI-A K40M，12GB GDDR5,4.29TFlops峰值）。我们使用NVIDIA的cuBLAS库[35]实现了所有的基准测试（见下面），这是一个用于GPU的最先进的线性代数库。

**•NN加速器。**基准加速器是先进的神经网络加速器DaDianNao，与GPU相比能够显着提高能效[5]。我们重新实现了65nm工艺的DaDianNao架构，但是由于我们没有65nm eDRAM库，所以我们用SRAM来替换所有的eDRAM。此外，我们重新调整了DaDianNao的大小，使其具有与我们的设计相当的算术运算符和片上SRAM容量，这使得我们在前一段中提到的我们的地区预算（<60平方毫米）下的两个加速器的公平比较成为可能。DaDianNao的重新实施的版本有一个中央瓷砖和总共32个叶片。中央瓷砖有64KB SRAM，32个16位加法器和32个16位乘法器;每个叶片有24KB SRAM，32个16位加法器和32个16位乘法器。换句话说，在重新实现的DaDianNao中，加法器和乘法器的总数以及总的SRAM容量都与我们的原型加速器相同。尽管我们在两个加速器中都不得不放弃eDRAM，但这仍然是一个合理的实验设置，因为加速器的灵活性主要取决于它的ISA，而不是它整合的具体设备。从这个意义上说，即使我们诉诸大型eDRAM来移除主内存访问并提高两个加速器的性能，从Cambricon获得的灵活性仍将保留在那里。

**基准。**我们以10个有代表性的NN技术作为我们的基准，见表3。每个基准都被手动翻译成汇编器，以便在Cambricon-ACC和DaDianNao上执行。我们用Synopsys VCS.B评估他们的周期性能。

**B.实验结果**

我们将Cambricon和Cambricon-ACC与性能和能量等指标进行比较。我们还提供了原型加速器的详细布局特性。

**1）灵活性：**鉴于通用ISA（例如，x86，MIPS和GPU-ISA）提供了明显的灵活性，这里我们将讨论局限于NN加速器的ISA。DaDianNao [5]和DianNao [3]是两个独特的NN加速器，它们具有明确的ISA（其他的通常是硬连线的）。他们共享类似的ISA，我们的讨论以具有更好性能和多核扩展性的DaDianNao为例。具体而言，该加速器的ISA仅包含与四种流行的神经网络类型（完全连接分类器层，卷积层，合并层和本地响应规范化层）相对应的四条512位VLIW指令，使其成为一种相当 NN域不完整的ISA。在表3中列出的10个代表性基准网络中，DaDianNao ISA仅能够表示MLP，CNN和RBM，但未能实现其余7个基准（RNN，LSTM，AutoEncoder，稀疏自动编码器，BM，SOM和HNN）。一个解释DaDianNao在7个代表网络上失败的观察井，它们不能被归类为四种层次的聚合（因此DaDianNao指令的聚合）。相比之下，Cambricon定义了总共43个64位标量/控制/向量/矩阵指令，并且足够灵活地表达所有10个网络。

**2）代码密度：**只有当ISA足够灵活时，代码密度才是有意义的ISA度量涵盖目标领域的广泛应用。因此，我们只将Cambricon的代码密度与GPU，MIPS和x86进行比较，分别用Cambricon，CUDA-C和C实现10个基准。我们手工编写Cambricon程序;使用nvcc编译CUDA-C程序，并在删除初始化和系统调用指令后计算生成的ptx文件的长度;我们分别使用x86和MIPS编译器编译C程序（使用-O2选项）。然后我们计算两种汇编器的长度。我们在图10中说明了Cambricon在其他ISA上的代码长度的减少。平均而言，Cambricon的代码长度分别比GPU，x86和MIPS短约6.41倍，9.86倍和13.38倍。这些观察结果并不令人意外，因为Cambricon将许多标量操作汇总为向量指令，并将矢量操作进一步汇总为矩阵指令，这大大减少了代码长度。

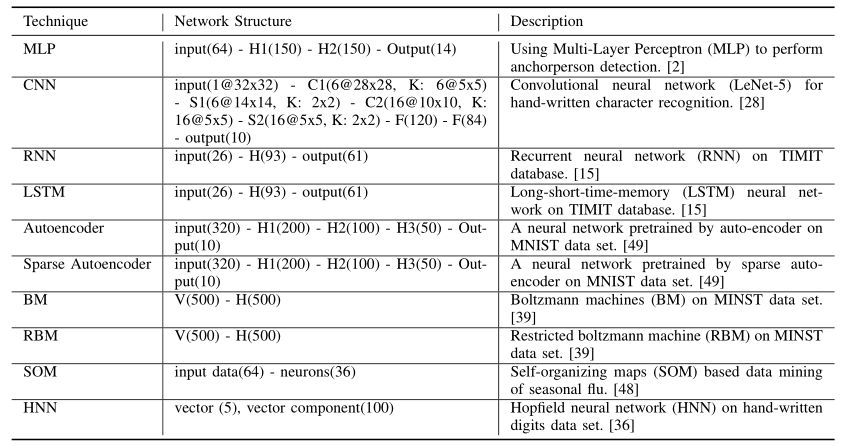
特别地，在MLP上，Cambricon可以将代码密度提高13.62倍，22.62倍，对于GPU，x86和MIPS分别为32.92倍。主要原因是MLP的Cambricon代码中的标量指令非常少。但是，在CNN上，Cambricon的GPU，x86和MIPS的码长分别只有1.09x，5.90x和8.27x。这是因为CNN的主体是一个深层嵌套的循环，需要许多单独的标量操作来操作循环变量。因此，将标量运算集成到向量运算中的优点在代码密度方面有小的增益。

另外，我们收集10个基准中的Cambricon指令类型的百分比分解。平均38.0％的指令是数据传输指令，4.8％的指令是控制指令，12.6％的指令是矩阵指令，33.8％的指令是矢量指令，10.9％的指令是标量指令。这一观察结果清楚地表明矢量/矩阵指令在神经网络技术中起着至关重要的作用，因此这些指令的高效实现对基于Cambricon的加速器的性能至关重要。

3）性能：我们比较了Cambricon-ACC与表3中列出的所有10个基准测试中的x86-CPU和GPU。图12说明了Cambricon-ACC对x86-CPU，GPU和DaDianNao的加速。Cambricon-ACC的平均速度比x86-CPU和GPU分别快91.72倍和3.09倍。这并不奇怪，因为Cambricon-ACC集成了专门针对NN技术而优化的专用功能单元和暂存器存储器。

另一方面，由于ISA的不完整和受限制，DaDianNao只能容纳10个基准点中的3个（即MLP，CNN和RBM），因此其灵活性明显比Cambricon-ACC差。与此同时，Cambricon-ACC更好的灵活性不会导致显著的性能损失。我们比较了Cambricon-ACC和DaDianNao在DaDianNao可以支持的三个基准上，并且观察到Cambricon-ACC平均只比DaDianNao慢了4.5％。Cambricon-ACC在DaDianNao上性能损失较小的原因是，Cambricon将DaDianNao的复杂高层功能指令（例如卷积层指令）分解为较短和较低级别的计算指令（如MMV和点产品），这可能在指令之间引入额外的管道泡沫。然而，由Cambri-con提供的高代码密度，额外气泡的数量适中，因此相应的性能损失可以忽略不计。

表3.基准（H表示隐藏层，C表示卷积层，K表示内核，P表示分层，F表示分层，V表示可见层）。



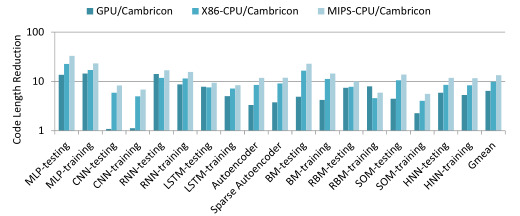


图10. 针对GPU，x86-CPU和MIPS-CPU的代码长度缩短

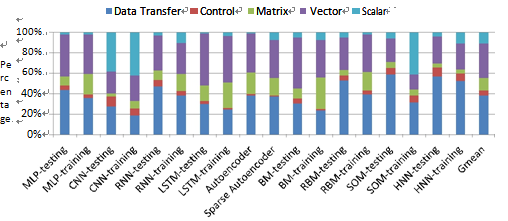


图11.指令类型在所有基准测试中的百分比

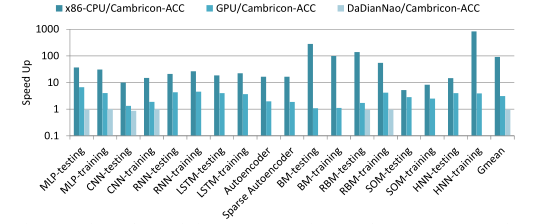


图12. Cambricon-ACC对x86-CPU，GPU和DaDianNao的加速

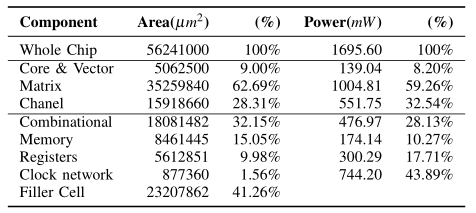
**4）能源消耗：**我们还比较了Cambricon-ACC，GPU和大电量的能耗，可以估算为耗电量（单位：瓦特）和执行时间（单位：秒）。NVPROF报告GPU的功耗，并根据模拟值变化转储（VCD）文件使用Synopsys Prime-Tame PX估算DaDianNao和Cambricon-ACC的功耗。由于缺乏硬件支持来估计CPU的实际功耗，因此我们没有与CPU基准进行能量比较。然而，最近有报道指出，在神经网络应用中，SIMD-CPU比GPU（NVIDIA K20M）的能量效率要低一个数量级，这对我们的实验有很好的补充。

如图13 ，GPU和大电号的能耗分别Cambricon-ACC的130.53倍和0.916倍，DaDianNao的能源平均超过3个基准，因为它只能满足10个基准中的3个。与Cambricon-ACC相比，GPU的功耗要高得多，因为GPU花费过多的硬件资源来灵活地支持各种工作负载。另一方面，Cambricon-ACC的能耗仅略高于DaDianNao，因为两个加速器集成了相同尺寸的功能单元和片上存储器，并且工作在相同的频率。Cambricon-ACC消耗的额外能量主要来自指令流水线逻辑，存储器队列以及向量超越功能单元。相比之下，DaDianNao使用低精度但轻量级的查找表，而不是使用先验的功能单位.

**5）芯片布局**：我们在图14中显示了Cambricon-ACC的布局，列出了表IV中的面积和功率故障。Cambricon-ACC的总面积为56.24平方毫米，比DaDianNao（55.34平方毫米，重新实施的版本）大1.6％左右。组合逻辑（主要是矢量和矩阵功能单元）占Cambricon-ACC面积的32.15％，片上存储器（主要是矢量和矩阵暂存器）消耗约15.05％的面积。矩阵部分（包括矩阵功能单元和矩阵暂存器）占Cambricon-ACC面积的62.69％，而核心部分（包括指令流水线逻辑，标量函数单元，存储器队列等）和向量部分（包括向量函数单元和矢量暂存器内存）只占面积的9.00％。剩余的28.31％面积被通道部分消耗，包括连接核心与向量部分和矩阵部分的导线，以及将矩阵部分的不同块连接在一起的导线。

我们还使用Synopsys PrimePower估算原型设计的功耗。峰值功耗为1.695 W（低于100％触发率），仅为K40M GPU的一个百分比。更具体地说，核心和矢量部分和矩阵部分分别消耗8.20％和59.26％的功率。此外，通道部分的数据移动消耗了32.54％的功率，这比核心和矢量部分的功率高几倍。可以预期，如果我们不将矩阵部分划分成多个块，那么信道部分的功耗可以高得多。

表4.采用台积电65纳米技术实现的Cambricon-ACC（1 GHz）布局特性



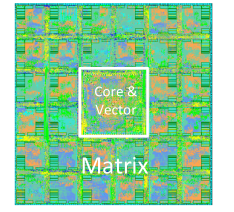


图14.采用台积电65nm技术实现的Cambricon-ACC的布局

六、扩展技术的潜力

尽管Cambricon是为现有的神经网络技术而设计的，但它也可以支持未来的神经网络技术，甚至是一些经典的统计技术，只要它们可以在Cambricon中分解为标量/电子/矩阵指令。这里我们以逻辑回归[21]为例，并说明Cambricon如何支持它。 从技术上讲，逻辑回归包含两个阶段，即训练阶段和预测阶段。训练阶段采用类似于MLP技术的训练阶段的梯度下降算法，这可以由Cambricon支持。在预测阶段，输出可以计算为（其中x =（， ... ）T是输入向量，总是等于1，θ=（，...）T是模型参数）。我们可以利用点积指令，标量基本算术指令和Cambricon的标量指数指令来执行逻辑回归的预测阶段。另外，给定一批n个不同的输入向量，Cambricon协同工作的MMV指令，vec-tor基本算术指令和向量指数指令并行计算n个输入的低预测相位。

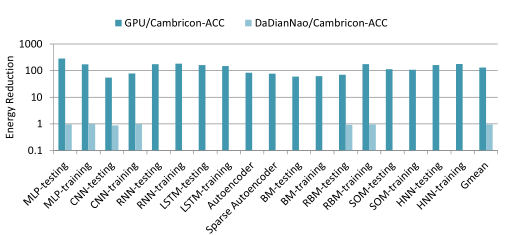


图13. Cambricon-ACC在GPU和DaDianNao上的能量减少

七、相关工作

在本节中，我们总结了神经网络技术和神经网络加速器设计的前期工作。

**神经网络。**现有的NN技术已经在其网络拓扑和学习算法中表现出显着的多样性。例如，深度信任网络（DBN）[41]由一系列层组成，每个层完全连接到相邻的层。相比之下，卷积神经网络（CNN）[25]使用卷积/合并窗口来指定神经元之间的连接，因此连接密度远低于DBN。有趣的是，DBN和CNN的连接密度均低于玻尔兹曼机器（BM）[39]，它们将所有的神经元彼此完全连接起来。针对不同神经网络的学习算法也可能存在差异，如用于训练多层感知器（MLP）的反向传播算法[50]，用于训练受限玻尔兹曼机器（RBM）的吉布斯采样算法[ 39]，以及用于训练自组织映射（SOM）的非监督学习算法[34]。

简而言之，尽管采用高级，复杂和信息性的指令可能是支持一小组类似NN技术的加速器的可行选择，但显着的多样性和大量现有NN技术使得构建单个加速器不可行，它使用大量的高级指令来涵盖范围广泛的NN。此外，没有一定程度的通用性，即使现有的成功加速器设计可能很容易因为神经网络技术的发展而变得可用。

**NN加速器。**神经网络技术的计算量非常大，传统上是在CPU和GPGPU组成的通用平台上执行的，这些通用平台对于NN技术通常不是节能的[3]，因为他们投入了过多的硬件资源来灵活地支持各种工作负载。在过去的十年中，已经有许多定制化为神经网络的硬件加速器，在FPGA上实现[13]，[38]，[40]，[42]或ASIC [3]，[12]，[14]， [44]。Farabet等人提出了一种名为Neuflow的加速度体系结构[12]，用于CNN的前馈路径。Maashri等人实现了另一个NN加速器，它在交换结构周围安排了几个定制的加速器[30]。Esmaeilzadeh等人提出了用于多层感知器（MLP）的SIMD类架构（NnSP）[10]。Chakradhar等人将CNN映射到可重构电路[1]。Chi等人提出了PRIME [6]，这是一种新颖的内存进程内体系结构，在基于ReRAM的主内存中实现了可重构的NN加速器。Hashmi等人提出了Aivo框架来表征它们特定的皮层网络模型和学习算法，它们可以为通用CPU和GPU而不是硬件加速器产生其网络模型的执行代码[16]。上述设计是针对一种特定的NN技术（例如，MLP或CNN）而定制的，其应用范围是有限的。陈等人提出了一个小型的神经网络加速器，称为DianNao，它的指令直接对应于CNN中的不同层类型[3]。DaDianNao采用了类似的指令集，但通过保持所有的网络参数在芯片上，达到了更高的性能和能源效率，这是加速器体系结构而不是ISA的创新[5]。所以DaDianNao的应用范围仍然受到ISA的限制，与DianNao的情况相似。刘等人设计了可以容纳7种经典机器学习技术的PuDianNao加速器，其控制模块只提供7种不同的操作码（每种操作码对应一种特定的机器学习技术）[29]。因此，PuDianNAO只允许对七种机器学习技术进行微小的改动。总而言之，指令集缺乏灵活性，阻碍了以前的加速器灵活有效地支持各种不同的NN技术。

**比较。**与先前的工作相比，我们将传统的高级和复杂指令描述成与较低级别的计算操作相对应的更短的指令（例如标量/向量/矩阵运算），这允许硬件加速器具有更广泛的应用范围。而且，简单和简短的指令可以减少加速器的设计和验证的复杂性。

**八、结论和未来的工作**

在本文中，我们提出了一种叫做Cambricon的神经网络的新颖的ISA，它允许NN加速器灵活地支持各种不同的NN技术。我们将Cambricon与十个不同但具有代表性的神经网络的x86和MIPS进行比较，并观察到Cambricon的代码密度明显高于x86和MIPS。我们在台积电65纳米技术中采用基于Cambricon的原型加速器，面积为56.24平方毫米，功耗仅为1.695瓦。得益于Cambricon，这个原型加速器可以容纳所有十个基准NN，而最先进的NN加速器DaDianNao只能支持其中的三个。即使在对三个基准神经网络进行测试的时候，我们的原型加速器仍然可以实现与最先进的加速器相媲美的性能/能效，而且开销可以忽略不计。我们未来的工作包括原型加速器的最终芯片封装，将Cambricon集成到通用处理器的尝试，以及对Cambricon进行深入研究以支持更广泛的应用的方案。

**解决方案**

这项工作得到中国国家自然科学基金（资助号：61133004,61303158,61432016,61472396, 61473275,61522211,61532016,61521092,61502446），中国973计划（资助号：2015CB358800），中国科学院重大研究计划（资助号：XDA06010403，XDB02040009），中国科学院国际合作重点项目（Grant 171111KYS-B20130002）和10000人才计划。 Xie部分由美国国家科学基金会1461698,1500848和1533933支持。

**参考文献**

[1] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A Dynamically Configurable Coprocessor for Convolutional Neural Networks. In Proceedings of the 37th Annual International Symposium on Computer Architecture, 2010.

[2] Yun-Fan Chang, P. Lin, Shao-Hua Cheng, Kai-Hsuan Chan, Yi-Chong Zeng, Chia-Wei Liao, Wen-Tsung Chang, Yu-Chiang Wang, and Yu Tsao. Robust anchorperson detection based on audio streams using a hybrid I-vector and DNN system. In Proceedings of the 2014 Annual Summit and Conference on Asia-Pacific Signal and Information Processing Association, 2014.

[3] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Te-

mam. DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. In

Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014.

[4] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. A High-Throughput Neural Network Accelerator. IEEE Micro, 2015.

[5] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DaDianNao: A Machine-Learning Supercomputer. In

Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014.

[6] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In Proceedings of the 43rd International Symposium on Computer Architecture (ISCA), 2016.

[7] A. Coates, B. Huval, T. Wang, D. J. Wu, and A. Y. Ng. Deep learning with cots hpc systems. In Proceedings of the 30th International Conference on Machine Learning, 2013.

[8] G.E. Dahl, T.N. Sainath, and G.E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013.

[9] V. Eijkhout. Introduction to High Performance Scientific Computing. In www.lulu.com, 2011.

[10] H. Esmaeilzadeh, P. Saeedi, B.N. Araabi, C. Lucas, and Sied Mehdi Fakhraie. Neural network stream processing core (NnSP) for em-bedded systems. In Proceedings of the 2006 IEEE International Symposium on Circuits and Systems, 2006.

[11] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural Acceleration for General-Purpose Approximate Programs. In

Proceedings of the 2012 IEEE/ACM International Symposium on Microarchitecture, 2012.

[12] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. NeuFlow: A runtime reconfigurable dataflow processor for vision. In Proceedings of the 2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2011.

[13] C. Farabet, C. Poulet, J.Y. Han, and Y. LeCun. CNP: An FPGA-based processor for Convolutional Networks. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, 2009.

[14] V. Gokhale, Jonghoon Jin, A. Dundar, B. Martini, and E. Culurciello.

A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks. In IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014.

[15] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM networks. In Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, 2005.

[16] Atif Hashmi, Andrew Nere, James Jamal Thomas, and Mikko Lipasti. A Case for Neuromorphic ISAs. In Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems, 2011.

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Management, 2013.

[18] INTEL. AVX-512. https://software.intel.com/en-us/blogs/2013/avx-512-instructions.

[19] INTEL. MKL. https://software.intel.com/en-us/intel-mkl.

[20] Pineda Fernando J. Generalization of back-propagation to recurrent neural networks. Phys. Rev. Lett., 1987.

[21] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An Introduction to Statistical Learning. 2013.

[22] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In Proceedings of the 12th IEEE International Conference on Computer Vision, 2009.

[23] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In arXiv:1502.01852, 2015.

[24] V. Kantabutra. On hardware for computing exponential and trigono-metric functions. Computers, IEEE Transactions on, 1996.

[25] Alex Krizhevsky, Sutskever Ilya, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In Ad-vances in Neural Information Processing Systems 25. 2012.

[26] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. In Proceedings of the 24th International Conference on Machine Learning, 2007.

[27] Q.V. Le. Building high-level features using large scale unsupervised learning. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998.

[29] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. PuDianNao: A Polyvalent Machine Learning Accelerator. In Pro-ceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, 2015.

[30] Maashri, A.A. and DeBole, M. and Cotter, M. and Chandramoorthy, N. and Yang Xiao and Narayanan, V. and Chakrabarti, C. Accelerat-ing neuromorphic vision algorithms for recognition. In Proceedings of the 49th ACM/EDAC/IEEE Design Automation Conference, 2012.

[31] G Marsaglia and W W. Tsang. The ziggurat method for generating random variables. Journal of statistical software, 2000.

[32] Paul A Merolla, John V Arthur, Rodrigo Alvarez-icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D Flickner, William P Risk, Rajit Manohar, and Dharmendra S Modha. A million spiling-neuron interated circuit with a scalable communication network and interface. Science, 2014.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, An-dreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. In Nature, 2015.

[34] M.A. Motter. Control of the NASA Langley 16-foot transonic tunnel with the self-organizing map. In Proceedings of the 1999 American Control Conference, 1999.

[35] NVIDIA. CUBLAS. https://developer.nvidia.com/cublas.

[36] C.S. Oliveira and E. Del Hernandez. Forms of adapting patterns to Hopfield neural networks with larger number of nodes and higher storage capacity. In Proceedings of the 2004 IEEE International Joint Conference on Neural Networks, 2004.

[37] David A. Patterson and Carlo H. Sequin. RISC I: A Reduced Instruction Set VLSI Computer. In Proceedings of the 8th Annual Symposium on Computer Architecture, 1981.

[38] M. Peemen, A.A.A. Setio, B. Mesman, and H. Corporaal. Memory-centric accelerator design for Convolutional Neural Networks. In

Proceedings of the 31st IEEE International Conference on Computer Design, 2013.

[39] R Salakhutdinov and G Hinton. An Efficient Learning Procedure for Deep Boltzmann Machines. Neural Computation, 2012.

[40] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Dur-danovic, E. Cosatto, and H.P. Graf. A Massively Parallel Copro-cessor for Convolutional Neural Networks. In Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009.

[41] R. Sarikaya, G.E. Hinton, and A. Deoras. Application of Deep Belief Networks for Natural Language Understanding. Audio, Speech, and Language Processing, IEEE/ACM Transactions on, 2014.

[42] P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale Convolutional Networks. In Proceedings of the 2011 International Joint Conference on Neural Networks, 2011.

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In arX-iv:1409.4842, 2014.

[44] O. Temam. A defect-tolerant accelerator for emerging high-performance applications. In Proceedings of the 39th Annual In-ternational Symposium on Computer Architecture, 2012.

[45] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on CPUs. In In Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011, 2011.

[46] Yu Wang, Tianqi Tang, Lixue Xia, Boxun Li, Peng Gu, Huazhong Yang, Hai Li, and Yuan Xie. Energy Efficient RRAM Spiking Neural Network for Real Time Classification. In Proceedings of the 25th Edition on Great Lakes Symposium on VLSI, 2015.

[47] Cong Xu, Dimin Niu, Naveen Muralimanohar, Rajeev Balasubra-monian, Tao Zhang, Shimeng Yu, and Yuan Xie. Overcoming the Challenges of Cross-Point Resistive Memory Architectures. In Pro-ceedings of the 21st International Symposium on High Performance Computer Architecture, 2015.

[48] Tao Xu, Jieping Zhou, Jianhua Gong, Wenyi Sun, Liqun Fang, and Yanli Li. Improved SOM based data mining of seasonal flu in mainland China. In Proceedings of the 2012 Eighth International Conference on Natural Computation, 2012.

[49] Xian-Hua Zeng, Si-Wei Luo, and Jiao Wang. Auto-Associative Neural Network System for Recognition. In Proceedings of the 2007 International Conference on Machine Learning and Cybernetics, 2007.

[50] Zhengyou Zhang, M. Lyons, M. Schuster, and S. Akamatsu. Com-parison between geometry-based and Gabor-wavelets-based facial expression recognition using multi-layer perceptron. In Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition, 1998.

[51] Jishen Zhao, Guangyu Sun, Gabriel H. Loh, and Yuan Xie. Optimiz-ing GPU energy efficiency with 3D die-stacking graphics memory and reconfigurable memory interface. ACM Transactions on Archi-tecture and Code Optimization, 2013.