软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006071]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种神经网络指令集扩展与代码映射机制*

娄文启, 王 超, 宫 磊, 周学海

(中国科学技术大学 计算机科学与技术学院,安徽 合肥 230027)

通讯作者: 王超, E-mail: cswang@ustc.edu.cn

摘 要: 近年来,卷积神经网络(CNN)在图像识别和分类领域的高精度表现使其在机器学习领域受到了广泛关注.然而 CNN 的计算与访存密集特性给需要支持各种负载的通用处理器带来了巨大压力.因此,涌现了大量 CNN 专用硬件加速器.它们虽然提高了效率但缺乏灵活性.本文基于新兴的 RISC-V 架构设计了包含 10 条矩阵指令的专用指令集 RV-CNN.通过抽象典型 CNN 中的计算为指令,该指令集可灵活支持 CNN 推理过程并具有比通用 ISA 更高的代码密度.在此基础上,提出了代码至指令的映射机制.通过在 Xilinx ZC702 上使用该指令集构建不同网络模型发现,相比于 x86 处理器,RV-CNN 平均具有 141 倍的能效和 8.91 倍的代码密度;相比于 GPU,平均具有 1.25 倍的能效和 1.95 倍的代码密度.另外,相比于以往的 CNN 加速器,该设计在支持典型 CNN 模型的同时仍具有不错的能效.

关键词: 卷积神经网络;特定领域指令;RISC-V;代码映射;现场可编程门阵列

中图法分类号: TP311

Neural Network Instruction Set Extension and Code Mapping Mechanism

LOU Wen-Qi, WANG Chao, GONG Lei, ZHOU Xue-Hai

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: In recent years, due to the high-accuracy performance of Convolutional Neural Network (CNN) in character recognition and image classification, it has received widespread attention in the field of machine learning. Nevertheless, the compute-intensive and memory-intensive characteristic of CNN has posed huge challenges to the general-purpose processor, which needs to support various workloads. Therefore, a large number of CNN-specific hardware accelerators have emerged to improve efficiency. Whereas, although previous accelerators are significantly efficient, they usually lack flexibility. In this paper, we analyze classical CNN models and design a domain-specific instruction set of 10 matrix instructions, called RV-CNN, based on the promising RISC-V architecture. By abstracting CNN computation into instructions, our design can provide sufficient flexibility for CNN and possesses a higher code density than the general ISA. Based on this, a code-to-instruction mapping mechanism is proposed. By using the RV-CNN to build different CNN models on the Xilinx ZC702, it was found that compared to x86 processors, RV-CNN has an average of 141 times energy efficiency and 8.91 times the code density; compared to GPU, it has an average of 1.25 times energy efficiency and 1.95 times the code density. Besides, compared to previous CNN accelerators, the design supports typical CNN models while having good energy efficiency.

Key words: CNN; domain-specific instructions; RISC-V; code maping; FPGA

作为现代人工智能的重要分支,人工神经网络(artificial neural network,简称 ANN)近年来获得迅速发展.深度学习作为其在大数据时代背景下的延续和进化,通过增加网络模型的深度有效增强了传统 ANN 算法的数据特征提取能力.其中,卷积神经网络(CNN)作为深度学习中广泛采用的算法,被人们实现在如人脸识别[1]、目标检测[2]、语音识别[3]、自然语言理解[4]等应用中,并取得了令人瞩目的成果.得益于其在各类应用场景中的出色表现,CNN 已成为研究人员的关注重点,并广泛部署在数据中心以及边缘嵌入式设备中.

---

* 论文扩展于 Lou W, Wang C, Gong L, et al. RV-CNN: Flexible and Efficient Instruction Set for CNNs Based on RISC-V Processors. In: International Symposium on Advanced Parallel Processing Technologies. Cham: Springer, 2019. 3-14.

收稿时间: 2020-02-16; 修改时间: 2020-04-04; 采用时间: 2020-05-09; jos 在线出版时间: 2020-06-10

然而,CNN 出色的识别精度背后是不断加深、复杂的网络结构和巨大的计算、访存量.近年来,CNN 模型的参数量达到了百万级别,计算量达到千兆级别[5-7].CNN 密集的计算和访存也给通用处理器带来了巨大的压力.因此,近年来涌现了许多基于 FPGA[8-10]、GPU[5]和 ASIC[11,12]的加速器,并达到了优于 CPU 的性能和能效.在这三类平台中,相比于动辄消耗数十瓦甚至数百瓦的 GPU 平台,FPGA 和 ASIC 通常具有更低的功耗.尽管 GPU 在 CNN 模型的训练阶段具有独特的优势,但离线训练以进行在线预测的模式使得模型推理过程更加关键.而在推理过程中,FPGA、ASIC 的低功耗特性使其具有更广泛的应用领域,如电量受限的嵌入式平台.因此本文主要关注基于这两类平台的 CNN 加速器的相关工作.但经观察发现,先前的工作中此类加速器通常仅加速特定的网络结构或特定类型的层,模式相对固定、灵活性较低.

为解决该问题,陈云霁团队提出了 DianNao[11],一款针对不同机器学习应用的高吞吐量 ASIC 芯片并设计了超长指令字(very long instruction word,简称 VLIW)风格的指令,其支持 CNN 和多层感知机模型(multi-layer perceptrons,简称 MLPs).相继提出的 DaDianNao[13]则是在 DianNao 基础上的 SIMD 实现,不同之处在于用于计算的权值矩阵固化到本地的 eDRAM 上,减少了读取内存的次数.但两者中 VLIW 风格指令对计算过程的抽象较差,不了解底层硬件则难以使用.随后,该团队通过对 ANN 中的计算进行抽象,设计了专用指令集 Cambricon[14],其包含了标量、向量和矩阵等指令,支持多种神经网络且具有比传统 ISA 更高的代码密度和性能.然而,该指令集并不专用于 CNN,为了通用性牺牲了部分 CNN 特定的数据复用和指令中的并行计算.针对 CNN 应用,Luca 等人[15,16]提出了 PULP,一种可扩展的多核计算平台,并在其中增加了硬件卷积引擎以加速卷积操作.该平台中的单个核心均基于 RISC-V 开源架构,并扩展了点积和 packed-SIMD 等指令,可直接使用指令驱动加速单元,节省了操作系统层面的开销.但是该团队并未针对网络中的其它层设计相应指令.因此,一个高效、灵活且易于实现的 CNN 专用指令集仍是需要的.

本文通过研究典型的 CNN 模型的计算模式提出了一个小型且易于实现的 CNN 专用指令集,称为 RV-CNN[17],其包含 10 条矩阵指令,可以灵活地支持多种 CNN 结构的推理过程.随后介绍了由 CNN 模型描述文件到专用指令的映射过程.在实现方面,本文将指令集扩展进 RISC-V 架构处理器中,并对指令的实现进行了特定的优化.最终,通过典型的案例研究,本文从代码密度、性能和能效方面对该指令集及其实现进行对比评估.

本文第 1 节介绍专用指令集的设计偏好.第 2 节详细阐述专用指令的格式、功能以及代码的映射过程,并展示部分代码样例.第 3 节将指令集与典型专用指令集作定性比较.第 4 节介绍基于 RISC-V 核的指令集硬件实现.第 5 节展示实验步骤与实验结果.第 6 节总结并介绍下一步工作.

1 设计偏好

本节主要展示了为设计一个高效且易于实现的 CNN 专用指令集时的一些偏好,基于这些偏好,我们才能设计出具体的指令.

RISC-V 扩展.以往 CNN 硬件加速器通常以外设的方式工作,主机端通过驱动对加速器进行读写.考虑到大量数据在用户空间与内核空间的拷贝,此过程中操作系统层面的时间以及资源开销显然是不可避免的.然而,RISC-V 架构的出现给加速器的工作模式带来了更多选择.该指令集架构由基础指令集和其它可选指令集组成,具有开源性和指令可定制性,为用户提供了定制处理器微架构的可能[18]与设计专用指令的空间.因此,基于 RISC-V 架构设计专用指令控制加速单元的执行也更简洁、高效.基于以上两点分析,我们最终选择 RISC-V 作为目标 ISA,并在保持基本内核和每个标准扩展不变的前提下使用 CNN 专用指令对其进行扩展.最终专用指令可以配合 RV32 具有的标量和逻辑控制指令完成 CNN 的推理过程.

数据级并行.设计 CNN 专用指令集涉及很多因素,但其中涉及性能瓶颈部分才是应该关注的重点.考虑到 CNN 逐层堆叠的拓扑结构和不同层权重数据的独立性,设计矩阵指令以利用其操作中的数据级并行性而非挖掘其指令级并行性是更有效的.研究表明在 Intel Xeon 处理器核中用于计算的消耗仅占整个核能量消耗的 37%[19],其余的能量消耗为体系结构成本,并不是计算必须的.因此,在设计专用指令时,增加指令的粒度,将指令取指、译码和控制的开销平摊到多个元素的计算上,可以有效提升执行效率.此外,当处理涉及大量数据的计算

时,与传统的标量指令相比,矩阵指令可以显式地指定数据块之间的独立性,减少数据依赖检测逻辑的大小.并且,矩阵指令还具有较高的代码密度,因此我们这里主要关注数据级并行.

便签存储器(Scratchpad Memory).向量寄存器组通常出现在向量体系结构中,其中每个向量寄存器都包含了一个长度固定的向量,并且允许处理器一次操作向量中的所有元素.便签存储器是在片上用于存储临时计算数据的高速内部存储器,其具有直接寻址访问、代价低以及可变长度数据访问的特性.由于实现便签存储器的代价较低,因此通常部署较大尺寸的便签存储器并集成直接内存访问(DMA)控制器,以便进行快速数据传输.此外,考虑到密集、连续、可变长度的数据访问经常发生在 CNN 中,我们这里选择使用便签存储器去替代传统的向量寄存器组.

2 专用指令设计与映射

本节中,我们首先展示了专用指令集的构成,然后详细介绍了在第 1 节提出的设计偏好下专用指令的功能和格式.在此基础上,我们介绍了从基于深度学习框架的 CNN 模型描述文件到专用指令的映射过程,并列举了由专用指令实现的卷积层和池化层的代码.

2.1 专用指令扩展

RV-CNN 指令集的构成如表 1 所示,其包含了数据传输指令、逻辑指令和计算指令.配合部分基础的 RV-32I 指令集(此处不再描述),该指令集可以完成典型的 CNN 类计算.RV-CNN 指令集架构仍然和 RISC-V 架构保持一致,属于 load-store 架构,仅通过专用的指令进行数据传输.并且该指令集仍使用 RV-32 中的 32 个 32 位通用寄存器,用于存储标量值以及便签存储器的寄存器间接寻址.另外,我们设置了一个向量长度寄存器(vector-length register,简称 VLR)来指定运行时实际处理的向量长度.以下对指令进行详细介绍.

Table 1 An overview of RV-CNN

表 1 RV-CNN 指令集综述

指令类型	示例
数据传输指令	MLOAD / MSTORE
计算类指令	MMM / MMA / MMS / MMSA
逻辑类指令	MXPOOL / MNPOOL / APOOL / MACT

2.1.1 数据传输指令

为了灵活地支持矩阵运算,数据传输指令可以完成片外主存储器和片上便签存储器之间可变大小的数据块传输.图 1 展示了矩阵加载指令(MLOAD)的格式,其中 Reg0 指定片上目标地址.Reg1、Reg2 和 Reg3 分别指定矩阵的源地址、矩阵的大小和相邻元素的跨度.具体而言,该指令完成数据从主存向便签存储器的传输,其中指令的步幅字段可以指定相邻元素的跨度,以便避免了内存中昂贵的矩阵转置操作.相对应地,矩阵存储指令(MSTORE)完成便签存储器向主存储器方向的数据传输,其指令格式与 MLOAD 相似,不过经常忽略步幅字段以避免不连续的片外访存.

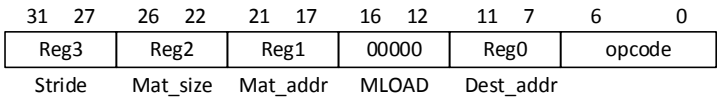

Fig.1 Matrix Load(MLOAD) instruction format

图 1 矩阵加载指令格式

2.1.2 矩阵计算指令

CNN 主要由卷积层、激励层、池化层和全连接层组成,其中大部分计算都集中在卷积层[20].在卷积层的计算中,卷积核在输入特征图上连续移动,并在重合区域执行点积以生成下一层的输入数据.在此过程中,同一卷积核在特征图的不同区域之间的计算是独立的,不同卷积核在特征图的相同区域的计算也是独立的.为了充分

利用卷积计算中的并行性,我们采用 Im2col(image to column)算法将 2-D 卷积运算转换为矩阵乘法运算(算法示意如图 2 所示).

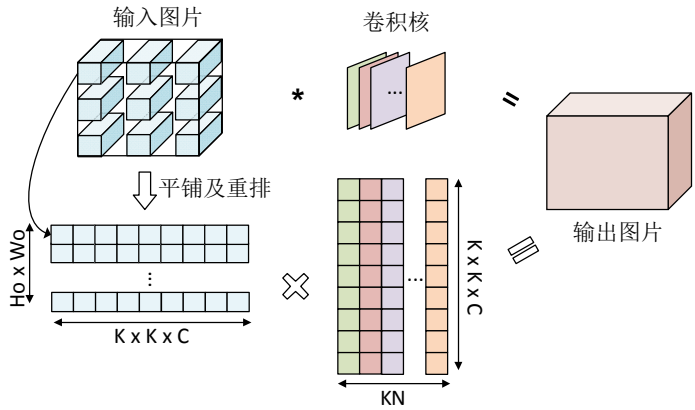

Fig.2 Matrix multiplication version of convolution

图 2 卷积运算的矩阵乘法版本

将 2-D 卷积映射到矩阵乘法操作后,可以很自然地使用 MMM(Matrix-Multiply-Matrix)指令执行该操作.其指令格式如图 3 所示,其中 Reg0 指定矩阵输出的便签存储器中的目的地址;16~12 位是指令的功能字段,指示矩阵乘运算.Reg1 和 Reg2 分别指定矩阵 1 和矩阵 2 的在便签存储器中的源地址.Reg3 中的四个字节分别代表矩阵的高(H)、宽(W)、卷积核大小(K)、卷积步长(S).考虑到实际执行中分片技术的使用,这里使用单个字节存储相应的信息是足够的.因此,卷积操作执行时的参数信息被打包成 32'b{H,W,K,S},并由 Reg3 指定.同时,由于数据分片载入,其产生的中间结果往往需要累加.这里不设置特定的矩阵加法指令而是设计了 MMS 指令.该指令在完成矩阵乘法计算后,将部分结果写入目标地址时与该地址原有的值累加后再存储,使得在完成累加的同时减少数据的重新载入.该指令的指令格式和各字段含义均与 MMM 指令一致,由功能字段指定该指令,故不再展示.此外,为了更大程度地利用数据局部性并减少对同一地址的并发读/写请求,我们选择采用专用的 MMM 指令执行矩阵乘法,而不是将其分解为更细粒度的指令(例如,矩阵向量乘和矢量点积).

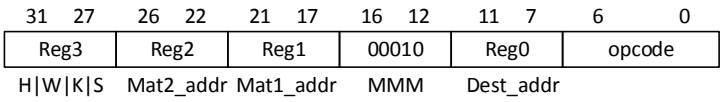

Fig.3 Matrix Multiply Matirx(MMM) instruction format

图 3 矩阵乘矩阵指令格式

全连接层通常在整个卷积神经网络的尾部以对之前各层学得的特征进行映射达到分类效果.全连接层的计算可以用矩阵向量乘法表示,而 MMM 指令在不同的参数下同样可以表示全连接层的计算,因此卷积层和全连接层可以复用相同的计算单元.

2.1.3 矩阵逻辑指令

融合(Fusion)[21]作为目前 DNN 加速器设计中常用的技术,其通过将部分层进行融合,从而以一次数据的加载、存储替代单独层的数据输入输出操作来最小化带宽限制.融合操作的优势以及目前 CNN 中激活层通常紧接在卷积层或全连接层之后的特点,使得设计相应的粗粒度指令将两者融合执行是非常适合的.而且激励层不改变输入张量的尺寸且激活函数逐元素进行激活操作,其需要的参数较少.因此,我们设计了 MMMA 指令,使矩阵相乘得到卷积层或全连接层的部分最终结果后可经过激活操作后输出,其指令格式与各字段含义均和 MMM 指令一致,由功能字段指定该指令.不过我们仍然保留了激活指令,以完成对输入的数据进行激活操作,其指令格式如图 4 所示,其指令中的 31~27 位用来决定激活函数的选择,如 ReLU()/sigmoid()/Tanh().

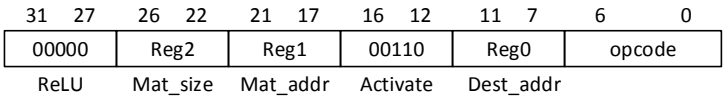

Fig.4 Matrix Activation (MACT) instruction format

图 4 矩阵激活指令格式

池化层通过降采样将输入数据的每个窗口子采样到单个池输出来减少输入图片的尺寸.实际上,卷积神经网络中相较于卷积层和全连接层,其余层包含了很少的计算且被数据访问时间限制.在某些 CNN 模型中,对池化层和相邻层采用融合技术同样是有效的.但是,不同于激活层在 CNN 中的位置相对固定且按元素操作,池化层仍具有一定的灵活度,如三个卷积层堆叠后池化.于是,我们这里仍将池化层当作单独的层处理.用于进行最大值池化的 MXPOOL 的指令格式如图 5 所示.其中 Reg0、Reg1、Reg2 分别表示输出数据的目标地址,输入数据的源地址和输入数据的长度.借鉴设计 MMM 指令的思想,观察到池化窗口通常为 2x2、3x3、5x5 等小尺寸而且通常采用分片技术处理输入数据,所以使用单个字节分别表示一次分片可处理的输入矩阵的高(H)、宽(W)、池化窗口大小(K)和步长(S)是足够的.这些必要信息进而被打包为 32 位值,由 Reg3 指定.

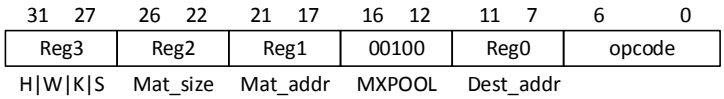

Fig.5 Matrix Maximum (MXPOOL) instruction format

图 5 矩阵池化指令格式

2.2 代码映射机制

RV-CNN 指令生成流程如图 6 所示.其中 CNN 模型描述文件可以是深度学习工程师熟悉的 Caffe、TensorFlow 或 Pytorch 等流行框架下的描述文件.由模型分析器对该描述文件进行解析以生成用于模型构建的参数信息和重排后的权重信息.在此基础上,应根据网络参数信息构建数据流图并提取算子,然后在不同的融合策略下将提取出的算子映射至指令池(RV-CNN 指令集)中的不同指令.由于我们设计的指令均为粗粒度指令,这里应提取粒度适合的算子才能映射至目标指令集.在专用指令提取后,应根据硬件的参数信息,如片上便签存储器的大小和硬件计算资源规模来决定分片大小.同时根据复用策略,如输入复用或权值复用等对指令进行编排以生成最终的代码.其中,应使用 RV32 基础指令集将参数信息加载至寄存器和完成循环控制.

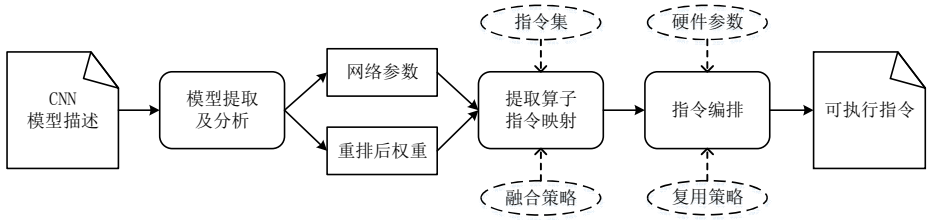

Fig.6 RV-CNN instructions generation process

图 6 RV-CNN 指令生成流程

2.3 代码示例

为了阐述提出的专用指令集的用法,我们列举了使用 RV-CNN 构建的 CNN 中两个具有代表性的部分,即卷积层和池化层.其中,卷积层的实现通过融合指令包含了激励层的操作.而全连接层的代码实现与卷积层代码类似,仅在配置参数上略有不同,故不作列举.

2.3.1 卷积层代码示例

RV-CNN 实现的卷积层代码如图 7 所示,其中左侧是在 Caffe 框架中编写的卷积层代码(用以示意),其完成对输入特征图(14x14x512),使用 512 组尺寸为 3x3、步幅为 1 的卷积核进行特征提取的过程.图中右侧则是使用专用指令完成相同功能的示意代码,其中假设硬件片上资源充足且数据排列顺序合适.

由于指令从寄存器中获取参数信息,因此我们首先需要向寄存器中加载必要的信息.这里首先加载立即数 0x0E0E0301 至$5 寄存器,根据第 2 节对指令格式的描述,该 32 位数据中由高至低的四个字节分别代表了输入数据的高度(14)、宽度(14)、卷积核尺寸(3)以及步幅(1).之后设置向量寄存器 VLR 为 16,代表一次处理的向量长度为 16.由于采用分片技术,这里设置循环计数器$11,$12 来完成深度方向和不同卷积核的遍历.随后,通过在寄存器$6,$8,$10 中加载数据在片外 DDR 中的实际地址,以便通过 MLOAD/MSTORE 完成数据传输.以上寄存器中的信息应由指令生成器或者用户根据网络模型以及硬件参数信息生成.在此基础上,开始实际计算过程.首先分别将数据从片外 0x30000 和 0x50000 处加载至由$1 和$2 指定的片上目的地址.数据加载完毕后,通过执行 MMM 指令完成计算.计算过程中的中间结果保存在片上便签存储器中.之后的计算通过使用 MMMS 指令完成计算与片上中间结果累加,以减少不必要的片外访存.最终,通过执行 MMMSA 指令完成部分最终结果的激活,并由 MSTORE 指令将结果传输至片外.

```
layer {
  type: "data"
  name: "data"
  top: "data"
  input_param: {
    shape: {
      dim: 512
      dim: 14
      dim: 14}
  }
}

layers {
 bottom: "data"
 top: "conv1"
 name: "conv1"
 type: CONVOLUTION
 convolution_param {
   num_output: 512
   pad: 1
    kernel_size: 3}
}

layers {
 bottom: "conv1"
 top: "conv2"
 name: "relu1"
 type: RELU
}
```

```
//$1:输入mat1地址, $2:输入mat2地址
//$3:临时变量地址, $4:输出尺寸
//$6:mat1地址, $7:mat1大小, $8:mat2地址,
//$9:mat2大小, $10:输出矩阵地址
//$11,$12:循环计数器

    LI      $5,   0x0E0E_0301 //H=14,W=14,k=3,s=1
    LI      $VLR,0x10         //设置vlr=16
    LI      $6,   0x30000
    LI      $8,   0x50000
    LI      $10,  0x70000
    LI      $11,0x1E          //设置计数器为30
    LI      $12,0x20
L0: MLOAD   $1,  $6,  $7      //加载分片权重
    MLOAD   $2,  $8,  $9      //加载分片激活
    MMM     $3,  $2,  $1, $5  //mat1 x mat2
    ADD     $8,  $8,  $9      // 更新mat2地址
L1: MLOAD   $2,  $8,  $9
    MMMS    $3,  $2,  $1,  $5   //mat2 x mat2 & accumulate
    ADD     $8,  $8,  $9
    SUB     $11, $11, #1
    BGE     $11, #0,  L1      // if(loop counter>0) goto label1
    MLOAD   $2,  $8,  $9
    MMSA    $3,  $2,  $1,  $5   //mat2 x mat2 & accumulate&relu
    MSTORE  $3,  $10,  $4     //结果送至地址($10)处
    SUB     $12,$12,  #1
    ADD     $6,  $6,  $7      //更新mat1 地址
    ADD     $10, $10, $4      //更新输出地址
    BGE     $12, #0, L0       // if(loop counter>0) goto label0
```

Fig.7 Convolutional layer code example implemented by RV-CNN

图 7 RV-CNN 实现的卷积层代码示例

2.3.2 池化层代码示例

RV-CNN 实现的池化层代码如图 8 所示,左侧仍以 Caffe 框架中编写池化层代码做功能示意,其表示对输入特征图(14x14x512),使用 2x2、步幅为 2 的池化窗口进行最大值采样.图右侧则是使用专用指令完成相同功能的示意代码,其中仍假设硬件片上资源充足且数据排列顺序合适.

由于池化层不包含权重数据且不在深度方向上累积,在输入图片大小合适的情况下,其 RV-CNN 实现的代码比卷积层代码简洁.在向相应寄存器中加载完参数后,利用 MLOAD 指令将待处理数据从片外 0x10000($6)处

加载至片上目的地址$1 处.输入数据加载完毕后,使用 MXPOOL 指令进行降采样并将结果存储在片上临时地址$5 处,采样结束后则由 MSTORE 将结果传输至片外地址 0x40000($7)处,该过程中输出并不会在片上累加.代码中的循环计数器是为了在输入数据深度方向上遍历,一次载入、池化、载出完成一次分片数据的采样,之后更新载入、载出地址以及计数器值.

```
layers {
  name: "pool"
  type: POOLING
  bottom: "data"
  top: "pool"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
//$1:输入mat 地址, $2:特征图尺寸
//$3:循环计数器, $5:临时变量地址
//$6:mat 地址, $7:输出mat地址, $8:输出大小

    LI      $4,    0x0E0E_0202  //H=14,W=14,k=2,s=2
    LI      $VLR,0x10           //设置 vlr=16
    LI      $3,    0x20         //设置循环计数器为32
    LI      $6,    0x10000
    LI      $7,    0x40000
L0: MLOAD   $1,    $6,  $2      //加载部分特征图
    MXPOOL  $5,    $1,  $2, $4  //降采样
    MSTORE  $5,    $7,  $8      //存储结果至地址($7)处
    ADD     $7,    $7,  $8
    ADD     $6,    $6,  $2      //更新输出地址
    SUB     $3,    $3,  #1
    BGE     $3,    #0,  L0      // if(loop counter>0) goto label0
```

Fig.8 Pooling layer code example implemented by RV-CNN

图 8 RV-CNN 实现的池化层代码示例

3 RV-CNN 和专用指令集对比

目前,在面向神经网络领域的专用指令集中,Cambricon 指令集被认为是最具代表性的指令集之一,而基于 RISC-V 架构的向量指令集扩展(RV-V)也被认为是加速神经网络计算的指令集.因此,本节将围绕指令集的适用范围、粗细粒度以及映射机制等方面,将 RV-CNN 与以上两种典型指令集进行定性分析和对比.

适用范围.RV-V 指令集的建立旨在利用应用中的数据级并行,其可广泛应用于科学计算,数据信号处理以及机器学习等领域.Cambricon 指令集则是面向神经网络领域,如 CNN、循环神经网络(RNN)以及长短期记忆网络(LSTM)等 10 余种网络模型而设计的指令集.相比之下,RV-CNN 指令集着重针对神经网络领域中的 CNN 而设计,其中涉及的运算类型较少.因此,前两者针对的领域更广泛,设计的难度也更大,这在指令集包含的指令类型和数目上也有所体现. RV-V 指令集已经包含了超过 60 条指令(草稿版本 0.8),Cambricon 指令集中包含了 47 条指令,而 RV-CNN 指令集仅包含了 10 条指令.

粗细粒度.RV-V 和 RV-CNN 指令集中分别包含了向量和矩阵指令,而 Cambricon 指令集中则包含了标量、向量以及矩阵三类不同的指令.Cambricon 之所以包含标量指令是因为其在概念上是一个完备的神经网络指令集,但其中用于加速计算的仍是向量和矩阵指令.因此,在指令粒度层面上,RV-CNN 指令集是三者中指令粒度最大的,Cambricon 指令集其次,由于 RV-V 指令集中均是向量指令,相比之下粒度最小.这也与指令集的适用范围相关,由于在设计 RV-V 指令集时针对的应用领域最为广泛,因此需要从多种计算操作中提取共性部分.考虑到硬件规模及能耗限制,这一过程往往需要结合算法特性,将不同的计算过程不断地向下拆分以寻求计算共性,提高指令集的表达能力,因此相较于前两者 RV-V 指令集的指令粒度最小.

代码映射机制.Cambricon 指令集的代码映射是基于框架的,其为流行的编程框架提供适配的机器学习高性能库与软件运行时支持,向上为框架提供丰富的算子和计算图方法以构造整个网络,向下通过调用内置驱动产生指令以控制硬件.RV-CNN 指令集的代码映射过程以基于框架的模型描述文件开始,不同于 Cambricon 对框架进行修改,RV-CNN 仅对框架下的模型描述文件进行分析,从而提取模型结构及权重信息,进而配合融合策略建立模型中算子和指令间的映射关系,之后根据复用策略进行指令编排,最终经汇编形成可执行文件.RV-V 指令集目前处于正在进行的状态,还未提供可使用的编译器来完成对代码的自动矢量化过程,仍需要用户手写汇编指令.但其中涉及到大量细粒度的向量指令,这给编程过程中寄存器分配以及指令编排增加了难度.

4 RV-CNN 的硬件实现

本节首先介绍了包含 RV-CNN 指令扩展的开源处理器核的整体结构,并详细介绍了指令的执行流程.随后讲述了与 RV-CNN 指令相对应的矩阵单元的组成及其子单元的功能,其中详细展示了矩阵乘法单元的结构,最后介绍了有关矩阵单元的优化细节.

4.1 整体架构

包含了 RV-CNN 扩展的 RISC-V 处理器核的主要功能部件以及简化的流水线结构如图 9 所示.可见,其包含了基本的五个流水线阶段:取指、译码、执行、访存和写回.其中矩阵计算单元处于流水线的执行阶段,用于完成矩阵指令的执行.在取指、译码阶段后,基础指令集中的指令将进入 ALU,然后进入下一个阶段.当译码阶段解析出当前指令是矩阵指令后,译码器从寄存器中获得相应信息并保存,待下个周期送入矩阵单元.矩阵单元根据接受的指令信息并检测相应功能部件的状态以确定是否执行.由于片上便签寄存器的地址空间用户可见,矩阵单元可以通过矩阵数据传输指令和内存进行交互,因此矩阵指令进入矩阵单元后不会经过访存和写回两个阶段,而其余指令不通过矩阵单元,其访存仍通过高速缓存(Cache).这样可以避免不必要的数据依赖性检测和硬件自动的数据换入换出.此外,由于矩阵单元包含了大量的计算单元,其内部有自己的流水线结构,译码器根据矩阵单元是否能接受矩阵指令信息来决定是否停止流水线.鉴于矩阵指令连续、大量的数据访问,我们在便签存储器外集成了 DMA 控制器,以便满足矩阵单元的数据访问需求.需要注意的是,矩阵单元完成计算和逻辑类指令所涉及的数据需要已经存在于片上,这需要程序的严格控制.

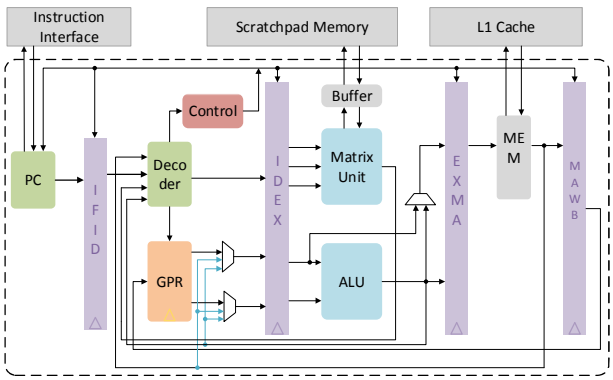

Fig.9 A simplified block diagram of processor core with RV-CNN extension

图 9 包含 RV-CNN 扩展的处理器核简化框图

4.2 矩阵单元

矩阵单元的整体结构如图 10 所示,其内部主要包含了输入输出单元、矩阵乘法单元、激活单元、池化单元以及内部控制器,其中橙色和灰色箭头分别表示控制流和数据流.矩阵单元接收传入的指令信息以及寄存器信息后存储到队列中.内部控制器(作为有限状态机)是矩阵单元的控制中心,其根据控制信息将唤醒子组件(如果可用)以完成相应任务.否则,它将生成一个反馈信号以指示相应的功能单元正忙.缓冲模块本质上是一个片上存储器,矩阵单元中的计算核心从中获取数据并将产生的结果写入.除融合指令会同时启动多个计算核心外,计算单元大致上和粗粒度指令一一对应.最后,输入输出模块则负责根据有效地址在矩阵单元与片上便签存储器之间进行数据传输.

由于矩阵乘法单元被卷积层和全连接层共用,其承担了大部分计算,因此该单元的实现对性能至关重要.这里我们采用了脉动阵列结构实现的矩阵乘法,其结构如图 10 右侧部分.脉动阵列是一种高效且简单的矩阵乘法实现方式,其通过二维网格将 MAC(Multiply-Accumulate)单元绑定在一起,除阵列的最外侧层(这里是最左侧和最上侧)的计算单元直接与片上缓冲相连以获取数据外,其余计算单元均从其邻居中获得输入.这种方式在 MAC 阵列规模变大时将显著减少片上缓冲的扇入扇出.同时,数据在 MAC 单元之间流动传递也增加了数据复

用.例如,MAC 阵列为 12x16 时,矩阵 B 中的元素按行在不同时刻由左至右在阵列中传递,数据输出时复用了 16 次;类似地,矩阵 A 中的元素按列在不同时刻由上至下在阵列中传递,输出时复用了 12 次.此过程中,配合流水线优化,每周期只需要向阵列输入 28 个数,则可以进行 192 个 MAC 操作.此外,短的局部互连也降低了布局布线的难度,因此我们这里选择使用脉动阵列作为矩阵乘法的实现方式.

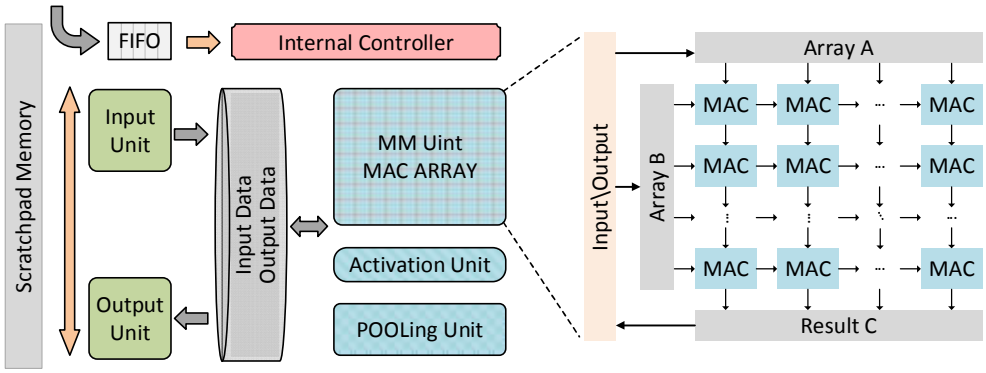

Fig.10 The block diagram of the matrix unit

图 10 矩阵单元的结构示意图

4.3 优化细节

数据复用.在处理卷积层时,对输入数据进行 Im2col 操作将在内存占用和带宽方面带来不菲的开销.例如,当卷积核步幅为 1 时,与原始输入数据相比,转换后的输入矩阵消耗的内存约为原始的 K*K 倍.为了保留 Im2col 操作带来的好处的同时减少额外的开销,我们实现了片上 Im2col 缓冲区,并根据地址在片上对切片数据进行数据重排和展开.设置 Im2col 缓冲区可以有效地增加数据复用,虽然未减少输入数据的访问次数,但使用开销较小的片上访问代替了开销大的片外访问,从而大大减少了该操作造成的额外的数据访问开销和外部带宽压力.

数据量化.传统上,不管是 CNN 的训练阶段还是预测阶段,其数据类型均采用 32 位单精度浮点数,这主要是因为它是现代 GPU 的标准数据格式.但研究发现,CNN 对有限的数值精度具有固有的鲁棒性,在预测阶段使用浮点计算并不是必须的.通过重训练和特定的微调手段,采用定点数进行预测造成的精度损失可以忽略不计(小于 1%)[22].对于许多 CNN,甚至 8 位位宽也可以达到足够的精度.之所以采取量化操作,是因为权重和激活的低位宽表示形式有助于避免昂贵的浮点计算,同时显着减少带宽需求和内存占用.DianNao[11]中显示,在台积电 65nm 工艺下,实现 32 位乘法器的面积和功耗要比 16 位乘法器的功耗高一个数量级.在这种权衡下,本文中矩阵单元的所有计算核心均采用 16 位定点数.

5 实验及结果

为了验证所提出的专用指令的有效性,我们在 FPGA 平台上构建了包含该指令集扩展的基于 RISC-V 架构的处理器核.在此基础上,使用 AlexNet、VGG16 两种不同规模的卷积神经网络进行评测,并将该原型系统与 Cambricon、CPU、GPU 和其它 FPGA 加速器进行对比分析.

5.1 实验方法

1) 原型系统的 FPGA 实现

我们使用 ZC702 开发板作为实验平台,该平台为嵌入式 FPGA 平台,其中包含了一块 XC7Z020 FPGA 芯片和 1GB DDR3 板载内存,可提供 4.2GB/s 的片外数据存取带宽.RISC-V 基础内核及矩阵单元的控制使用硬件描述语言 Verilog 完成,矩阵单元中的子计算单元设计则通过 Xilinx Vivado HLS 2017.4 高层次综合工具完成,完整的硬件工程通过 Xilinx Vivado 2017.4 集成开发环境进行综合和实现.

2) CPU 测试基准

我们使用 Caffe 深度学习框架(CPU-Only)在 CPU 平台上部署了两种目标网络模型.CPU 配置为 Intel i7-4790K,其中包含了 4 物理核心,最大线程数为 8,工作频率 4GHz,并配有 16GB DDR3 内存.

3) GPU 测试基准

GPU 版本的测试仍然通过 Caffe 深度学习框架(GPU-only)并通过 CuDNN5.1 加速库在 GPU 平台上部署两种目标网络模型.GPU 配置为 NVIDIA Tesla K40C,其最大可支持 2880 个硬件线程,工作频率 745-875MHz,并配有 12GB GDDR5 显存.

5.2 实验结果

在本小节中,我们首先报告了原型系统在 FPGA 平台上的资源消耗和功耗,然后围绕代码密度、性能及能效三个方面将该设计与 Cambricon、CPU、GPU 以及以往基于 FPGA 的加速器进行对比分析.

通过查看原型系统在 Vivado 工具中的部署报告,我们获得了其在目标平台的各项资源消耗以及 FPGA 的功耗,如表 2 所示.

Table 2 Hardware resource utilization deployed on Xilinx ZC702 platform

表 2 Xilinx ZC702 平台上部署的硬件资源利用率

Resource	DSP	BRAM	LUT	FF	Power(W)
Total amount	220	280	53200	106400	
Used	200	181	26177	30665	
Utilization	91%	65%	49%	29%	2.12

1. 与 Cambricon、CPU、GPU 平台的代码密度、性能与能效对比

代码密度.本文提出的专用指令不仅适用于加速 CNN 应用,而且还为具有类似计算模式的其它深度学习算法(如 MLPs)提供支持.通过使用 RV-CNN、C 和 CUDA-C 实现流行的 CNN 模型并测量其代码长度,我们对比了 RV-CNN 指令与基础 RV32、ARM、x86 和 GPU 的代码密度,结果如图 11 所示,其中以 RV-CNN 指令集实现的代码长度为基准.可见,相比于原本的 RV32(IMF)指令集,扩展 RV-CNN 后代码长度减少了 10.10 倍.相比与 GPU、x86 与 ARM 指令集,RV-CNN 的代码长度分别减少了 1.95 倍,8.91 倍和 10.97 倍.以 x86 指令的代码长度为基准,RV-CNN 的代码长度相比于 Cambricon 减少了 1.51 倍.

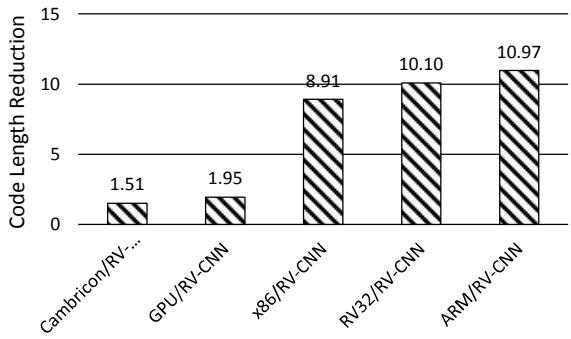

Fig.11 The reduction of code length against Cambricon, GPU, x86, RV32, and ARM

图 11 RV-CNN 相对于 Cambricon,GPU,x86,RV32 以及 ARM 的代码长度减少

性能及能效.图 12 展示了在两种神经网络(AlexNet 和 VGG16)的推理过程测试下,该设计与 CPU 和 GPU 的性能、能效对比,其中所有数值均为归一化到 CPU 的实验结果上.可见在 Xilinx ZC702 平台上,该设计在执行两种网络的性能上均是优于 CPU 的,分别达到 2.64 倍和 4.23 倍的加速效果.但由于该平台是嵌入式平台,用于计算的硬件资源 DSP(使用 91%)成为主要性能瓶颈,因此在执行两种网络的性能上均落后于 GPU.在能效方面,以每瓦特性能(GOPS/w)为基准,该设计在 AlexNet 和 VGG-16 的推理上,相较于 CPU 分别提升了 101.49 倍和 167.62 倍;相较于 GPU 分别提升了 1.06 倍和 1.40 倍.由于 Cambricon 加速器为 ASIC 设计,因此我们这里对比了性能密度(op/multiplier/cycle),即每周期乘法器完成的操作数.以 GPU 的测试数据为基准,RV-CNN 相比于

Cambricon 有 1.16 倍的提升.

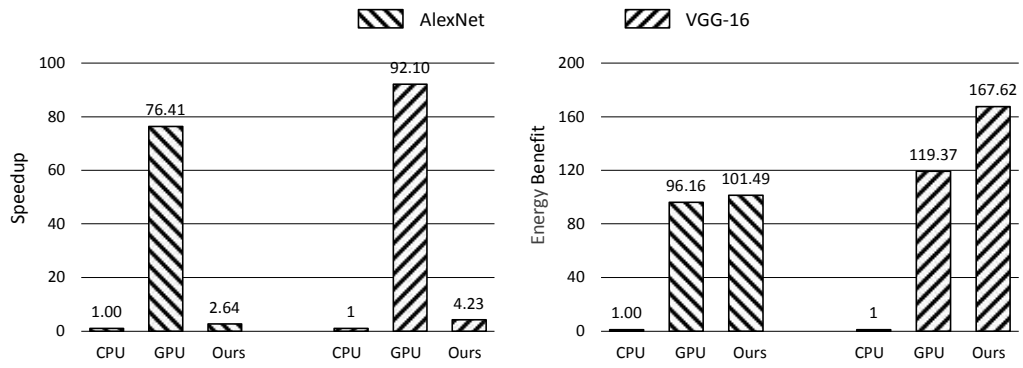

Fig.12 Performance and energy efficiency comparison between prototype system and CPU and GPU

图 12 原型系统与 CPU,GPU 的性能、能效对比

2. 与其他 FPGA 加速器的对比

表 3 列出了本设计与已有的典型 FPGA 加速器的结果对比.由于不同的工作采用了不同的量化策略和不同的硬件进行部署,因此很难选择一种有效且精确的比较方法.若以每秒千兆操作数(GOPS)作为性能评估标准,以前的工作可以实现比我们更好的性能.但是,更高的性能背后是更多的资源消耗,例如 DSP 和 LUT 资源,因而功耗也会相应增加.若以每瓦特的性能(GOPS/w)作为能效评估标准,与以往的加速器相比,我们的设计在保持灵活性下仍具有较高的能效.

Table 3 Comparison of the prototype system and previous FPGA-based accelerator deployment

表 3 原型系统与以往基于 FPGA 的加速器部署对比

	FPGA2015[23]	FPGA2016[24]	FCCM2017[25]	Ours
平台	Virtex7 VX485T	Stratix5 GSD8	Stratix5 GSMD5	Zynq XC7Z020
频率(MHz)	100	120	150	100
模型	AlexNet	VGG16	VGG16	AlexNet / VGG16
位宽	32-bit float	16-bit fixed	16-bit fixed	16-bit fixed
性能(GOPs)	61.62	117.8	364.36	21.77 / 35.95
功耗(w)	18.61	25.8	25	2.12
能效(GOPs/w)	3.31	4.57	14.57	10.27 / 16.96

6 结束语

CNN 在图像识别、目标检测领域的广泛应用使得其性能至关重要.本文通过分析典型 CNN 的计算模式,提出了一种高效且易于实现的专用指令集 RV-CNN,其包含了 10 条粗粒度的矩阵指令,可以灵活地为典型 CNN 模型推理过程提供支持.在此基础上,我们介绍了 CNN 模型描述文件到 RV-CNN 指令的映射过程.随后通过定性分析,从不同方面将 RV-CNN 与典型专用指令集进行比较.在指令实现方面,我们将该指令集扩展进了基于开源架构 RISC-V 的处理器核,并以相对紧耦合的方式将对应的矩阵单元嵌入经典的五级流水线中.最后,本设计在 Xilinx ZC702 平台对上进行综合实现,并以典型的神经网络进行测试.结果显示,相比与 Intel i7-4790K 处理器和 Tesla k40c GPU,该原型系统具有最高的能效和代码密度.此外,与先前的加速器相比,该原型系统在保持灵活性的同时也展现了不错的能效.

目前新型 CNN 网络层出不穷,还需考虑对其中诸如深度可分离卷积等操作进行指令优化以提高执行效率.此外,扩展指令的设计与实现应针对 RISC-V 特点做出协同优化.最后,根据模型及硬件信息执行的代码映射过程还未自动化,未来计划在以上方面做改进.

References:

[1] Wu F, Kong Y, Dong W, et al. Gradient-aware blind face inpainting for deep face verification. Neurocomputing. 2019, 331(FEB.28): 301-311.

[2] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. IEEE, 2016. 779-788. [doi:10.1109/CVPR.2016.91]

[3] Sainath T N, Mohamed A, Kingsbury B, et al. Deep convolutional neural networks for LVCSR. In: IEEE international conference on acoustics, speech, and signal processing. IEEE, 2013. 8614-8618. [doi:10.1109/ICASSP.2013.6639347]

[4] Collobert R, Weston J, Bottou L, et al. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research. 2011, 12(1): 2493-2537. [doi:10.1016/j.chemolab.2011.03.009]

[5] Krizhevsky A, Sutskever I, Hinton G. ImageNet Classification with Deep Convolutional Neural Networks. Advances in neural information processing systems. 2012, 25(2): 1097-1105. [doi:10.1145/3065386]

[6] Szegedy C, Liu W, Jia Y, et al. Going Deeper with Convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. IEEE, 2014. 1-9. [doi:10.1109/CVPR.2015.7298594]

[7] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. IEEE, 2016. 770-778. [doi:10.1109/CVPR.2016.90]

[8] Gong L, Wang C, Li X, et al. MALOC: A Fully Pipelined FPGA Accelerator for Convolutional Neural Networks With All Layers Mapped on Chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2018, 37(11): 2601-2612. [doi:10.1109/TCAD.2018.2857078]

[9] Wang C, Gong L, Yu Q, et al. DLAU: A Scalable Deep Learning Accelerator Unit on FPGA. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2017, 36(3): 513-517. [doi:10.1109/TCAD.2016.2587683]

[10] Wang C, Li X, Chen Y, et al. Service-Oriented Architecture on FPGA-Based MPSoC. IEEE Transactions on Parallel and Distributed Systems. 2017, 28(10): 2993-3006. [doi:10.1109/TPDS.2017.2701828]

[11] Chen T, Du Z, Sun N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In: architectural support for programming languages and operating systems. ACM, 2014. 269-284. [doi:10.1145/2541940.2541967]

[12] Moons B, Verhelst M. An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS. IEEE Journal of Solid-state Circuits. 2017, 52(4): 903-914. [doi:10.1109/JSSC.2016.2636225]

[13] Chen Y, Luo T, Liu S, et al. DaDianNao: A Machine-Learning Supercomputer. In: international symposium on microarchitecture. IEEE, 2014. 609-622. [doi:10.1109/MICRO.2014.58]

[14] Liu S, Du Z, Tao J, et al. Cambricon: an instruction set architecture for neural networks. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). IEEE, 2016. 393-405. [doi:10.1145/3007787.3001179]

[15] Conti F, Rossi D, Pullini A, et al. PULP: A Ultra-Low Power Parallel Accelerator for Energy-Efficient and Flexible Embedded Vision. Journal of Signal Processing Systems. 2015, 84(3): 339-354. [doi:10.1007/s11265-015-1070-9]

[16] M. Gautschi, Schiavone, and L. Benini, “A near-threshold risc-v core with dsp extensions for scalable iot endpoint devices,” IEEE Transactions on Very Large Scale Integration Systems, 2017, 25(10): 2700-2713.

[17] Lou W, Wang C, Gong L, et al. RV-CNN: Flexible and Efficient Instruction Set for CNNs Based on RISC-V Processors. In: International Symposium on Advanced Parallel Processing Technologies. Cham: Springer, 2019. 3-14.

[18] Bao Y, Wang S. Labeled von Neumann Architecture for Software-Defined Cloud. Journal of Computer Science and Technology. 2017, 32(2): 219-223. [doi:10.1007/s11390-017-1716-0]

[19] Cong J, Ghodrat M A, Gill M, et al. Accelerator-Rich Architectures: Opportunities and Progresses. In: design automation conference. IEEE, 2014. 1-6. [doi:10.1145/2593069.2596667]

[20] Lu L Q, Zheng S Z, Xiao Q C, et al. Accelerating convolutional neural networks on FPGAs (in Chinese). Sci Sin Inform, 2019, 49: 277-294. [doi: 10.1360/N112018-00291]

[21] Alwani M, Chen H, Ferdman M, et al. Fused-layer CNN accelerators. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) IEEE, 2016. 1-12. [doi:10.1109/micro.2016.7783725]

[22] Gysel P, Pimentel J, Motamedi M, et al. Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks. IEEE Transactions on Neural Networks. 2018, 29(11): 5784-5789.

[23] Zhang C, Li P, Sun G, et al. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2015. 161-170.

[24] Suda N, Chandra V, Dasika G, et al. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In: Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016. 16-25.

[25] Guan Y, Liang H, Xu N, et al. FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates. In: 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE, 2017. 152-159.

附中文参考文献:

[20]卢丽强,郑思泽,肖倾城,等.面向卷积神经网络的 FPGA 设计.中国科学,2019,49(03):37-54. [doi: 10.1360/N112018-00291]