计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20190117
56(6): 1192-1204, 2019

基于细粒度数据流架构的稀疏神经网络全连接层加速

向陶然[1,2] 叶笑春[1] 李文明[1] 冯煜晶[1,2] 谭 旭[1,2] 张 浩[1] 范东睿[1,2]
[1](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
[2](中国科学院大学 北京 100049)
(xiangtaoran@ict.ac.cn)

Accelerating Fully Connected Layers of Sparse Neural Networks with Fine-Grained Dataflow Architectures

Xiang Taoran[1,2], Ye Xiaochun[1], Li Wenming[1], Feng Yujing[1,2], Tan Xu[1,2], Zhang Hao[1], and Fan Dongrui[1,2]
[1](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Deep neural network (DNN) is a hot and state-of-the-art algorithm which is widely used in applications such as face recognition, intelligent monitoring, image recognition and text recognition. Because of its high computational complexity, many efficient hardware accelerators have been proposed to exploit high degree of parallel processing for DNN. However, the fully connected layers in DNN have a large number of weight parameters, which imposes high requirements on the bandwidth of the accelerator. In order to reduce the bandwidth pressure of the accelerator, some DNN compression algorithms are proposed. But accelerators which are implemented on FPGAs and ASICs usually sacrifice generality for higher performance and lower power consumption, making it difficult to accelerate sparse neural networks. Other accelerators, such as GPUs, are general enough, but they lead to higher power consumption. Fine-grained dataflow architectures, which break conventional Von Neumann architectures, show natural advantages in processing DNN-like algorithms with high computational efficiency and low power consumption. At the same time, it remains broadly applicable and adaptable. In this paper, we propose a scheme to accelerate the sparse DNN fully connected layers on a hardware accelerator based on fine-grained dataflow architecture. Compared with the original dense fully connected layers, the scheme reduces the peak bandwidth requirement of 2.44×～6.17×. In addition, the utilization of the computational resource of the fine-grained dataflow accelerator running the sparse fully-connected layers far exceeds the implementation by other hardware platforms, which is 43.15%, 34.57%, and 44.24% higher than the CPU, GPU, and mGPU, respectively.

Key words fine-grained dataflow; sparse neural network; general accelerator; data reuse; high parallel

收稿日期:2019-03-01;**修回日期**:2019-04-10

基金项目:**国家重点研发计划项目**(2018YFB1003501);**国家自然科学基金项目**(61732018,61872335,61802367);**中国科学院国际伙伴计划**(171111KYSB20170032);**计算机体系结构国家重点实验室创新项目**(CARCH3303,CARCH3407,CARCH3502,CARCH3505)
This work was supported by the National Key Research and Development Plan of China (2018YFB1003501), the National Natural Science Foundation of China (61732018, 61872335, 61802367), the International Partnership Program of Chinese Academy of Sciences (171111KYSB20170032), and the Innovation Project of the State Key Laboratory of Computer Architecture (CARCH3303, CARCH3407, CARCH3502, CARCH3505).

通信作者:**叶笑春**(yexiaochun@ict.ac.cn)

摘 要 深度神经网络(deep neural network, DNN)是目前最先进的图像识别算法,被广泛应用于人脸识别、图像识别、文字识别等领域.DNN具有极高的计算复杂性,为解决这个问题,近年来涌出了大量可以并行运算神经网络的硬件加速器.但是,DNN中的全连接层有大量的权重参数,对加速器的带宽提出了很高的要求.为了减轻加速器的带宽压力,一些DNN压缩算法被提出.然而基于FPGA和ASIC的DNN专用加速器,通常是通过牺牲硬件的灵活性获得更高的加速比和更低的能耗,很难实现稀疏神经网络的加速.而另一类基于CPU,GPU的CNN加速方案虽然较为灵活,但是带来很高的能耗.细粒度数据流体系结构打破了传统的控制流结构的限制,展示出了加速DNN的天然优势,它在提供高性能的运算能力的同时也保持了一定的灵活性.为此,提出了一种在基于细粒度数据流体系结构的硬件加速器上加速稀疏的DNN全连接层的方案.该方案相较于原有稠密的全连接层的计算减少了2.44×~6.17×的峰值带宽需求.此外细粒度数据流加速器在运行稀疏全连接层时的计算部件利用率远超过其他硬件平台对稀疏全连接层的实现,平均比CPU,GPU和mGPU分别高了43.15%,34.57%和44.24%.

关键词 细粒度数据流;稀疏神经网络;通用加速器;数据重用;高并行性

中图法分类号 TP387

深度神经网络近几年在飞速发展,它们在人脸识别、智能监控、图像识别、文字识别等领域有着非常出色的表现.特别是在2012年多伦多大学的Alex Krizhevsky团队凭借他们提出的深度神经网络分类模型AlexNet[1],获得了ImageNet挑战赛冠军.他们把分类误差记录从26%降到了15%.自此之后,大量公司和学者都投入了深度学习的研究中.

目前常用的DNN算法都具有大量的权重,其中全连接层(fully connected layers, FC layers)的权重比例非常高.例如,AlexNet有61×10^6的权重参数,而这其中全连接层的参数数量有59×10^6.VGG-16[2]有138×10^6的权重参数,其中全连接层的参数数量有124×10^6.为减少DNN算法的权重数量,文献[3]提出了一种使用剪枝(pruning)、权值量化与共享(weight quanti-zation and shared)和哈夫曼编码(Huffman coding)压缩权重的方法.如表1所示,经过剪枝后AlexNet和VGG-16在全连接层的权重数量大量减少.

Table 1 Compression Statistics for FC Layers in AlexNet and VGG-16 After Pruning

表1 剪枝后AlexNet与VGG-16中全连接层剩余的有效权重比率

DNN Model	Layer	# Weights	Weights Rate/%
AlexNet	FC6	38×10^6	9
	FC7	17×10^6	9
	FC8	4×10^6	25
VGG-16	FC6	103×10^6	4
	FC7	17×10^6	4
	FC8	4×10^6	23

虽然压缩DNN可以减少存储权重的空间,然而目前的DNN加速器并不能很好支持这样的算法执行.

近年来涌现出的许多DNN硬件加速器,例如DianNao[4],TPU[5],Eyeriss[6]等,充分利用了DNN的并行性和数据重用的特征,提出了低功耗高性能的DNN硬件加速方法.但是这些专用加速器只能运行稠密的DNN,甚至部分加速器不能支持DNN中一些常见的层,如激活层和局部响应归一化层.只有少量加速器,如EIE[7]和Cambricon-X[8],可以运行稀疏的神经网络.但是同样地在这些不灵活的硬件上很难实现一些新的算法.

细粒度数据流加速器已经在科学计算领域[9]和大数据领域[10]广泛应用,它在具有高性能的同时也保持着很高的通用性.本文构建的一个细粒度数据流加速器(fine-grained dataflow processing units, FDPU),它可以加速Stencil、FFT和矩阵乘等高性能应用[11-12],此外文献[13]还证明了DNN在细粒度数据流结构上的实现可以获得非常好的效率和能效比.如图1所示,FDPU(16tiles,8.192Tops)在运行AlexNet各层时,相较于GPU(NVIDIA Tesla K80,8.73TFLOPS)有1.48×~26×的加速.

为了更高效地在细粒度数据流加速器上实现DNN的加速,我们提出了一种在细粒度数据流加速器上加速稀疏的全连接层的方法.这种方法可以减少加速器对内存数据的访问量,并且基本不增加硬件设计,同时对加速器的性能影响也较小.

本文的贡献主要有3个方面:

1) 提出了一种适合细粒度数据流加速器的

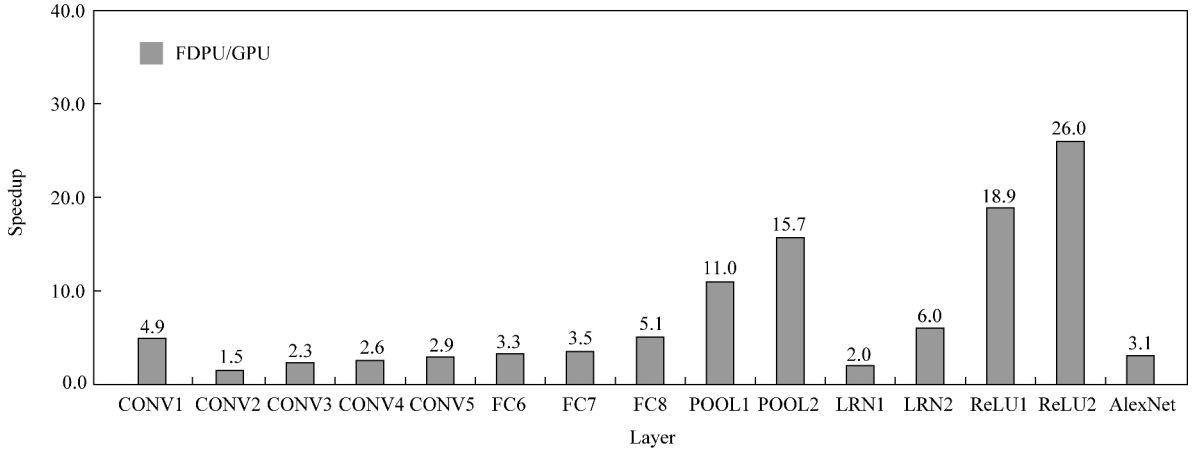


Fig. 1 Speedup of FDPU over GPU running layers of AlexNet

图 1 FDPU 相比于 GPU 在运行 AlexNet 各层时的加速比

压缩数据格式,并提出了读取该压缩数据格式的数据流图.

2) 分析了全连接层的特点,为稀疏的神经网络提出了在 FDPU 上的加速方案.

3) 对比了 FDPU 与 CPU,GPU 和专用数据流加速器的实验结果.从实验结果可知,本文提出的加速稀疏的全连接层的方法相较于原有稠密的全连接层运算减少了 2.44×～ 6.17×的峰值带宽需求.在输入图像的批次大小(batch size)为 64 时,FDPU 运行稀疏全连接层的计算部件利用率远超过其他硬件平台,平均比 CPU,GPU 和 mGPU 分别高了 43.15%,34.57%和 44.24%.

1 相关工作

1.1 通用细粒度数据流加速器

细粒度数据流架构由 Dennis[14] 提出,其结构与传统的控制流架构完全不同.细粒度数据流结构具有较好的指令并行性、数据复用性和低功耗等特点.细粒度数据流架构具有 3 种特性:

1) 数据流结构中的指令只要其操作数准备好了即可被执行,指令的执行不受程序计数器和指令窗口的限制,可以充分挖掘指令级并行和数据级并行;

2) 数据在执行单元(PE)间直接通信,避免了频繁的数据存取,减少访存开销;

3) 执行单元不需要如乱序执行、分支预测、深度流水等复杂的逻辑控制,简化了执行单元的设计.

TRIPS[15] 是由德克萨斯大学的 Burger 等人提出的细粒度数据流体系结构.TRIPS 可以同时支持 8 个程序块(frame)并行运行,用以掩盖指令之间传输操作数的延迟.程序块的指令被映射到 4×4 执行单元(PE)阵列上,每个 PE 执行被分配到的部分指令,并将执行结果传递给目的指令.程序块执行完成后,TRIPS 调度下一个程序块,将下一个程序块的指令映射到执行阵列上执行.WaveScalar[16] 是华盛顿大学的 Swanson 提出的基于簇(cluster)的可扩展数据流体系结构.每个簇包含 4 个域(domains),每个域包含 8 个 PE.数据流程序中的每条指令被映射到 PE 中.在程序执行期间,WaveScalar 不断替换无用指令并将新的未执行指令加载到 PE 中,就如同波浪一样一波一波地执行指令.2 种数据流体系结构都是通用处理器,无法充分利用 DNN 中的并行性来掩盖操作数传输的延迟,导致功能单元的利用率较低.

1.2 基于数据流思想的专用 DNN 加速器

相比于功耗较高的通用计算引擎,例如图形处理单元(GPU),越来越多的研究人员提出了在功耗和性能上更具优势的基于 ASIC[17-18] 或基于 FPGA[19-21] 的专用 DNN 加速器.其中基于脉动阵列的 DNN 加速器或者是简单的控制数据流入流出的 DNN 专用加速器都可以视为数据流思想延伸的产物.

纽约大学的 Farabet 等人在 2011 年提出了一款面向卷积神经网络的数据流加速器 NeuFlow[22],它是一个运行时可重配置的数据流结构.DianNao[4] 是由中国科学院计算技术研究所的陈云霁于 2014 年提出的深度学习专用加速器.它采用了基于分时复用的加速器设计结构,该结构由神经功能部件(neural functional unit, NFU)和片上存储构成.DianNao 可以说开启了专用 DNN 加速芯片研究的风潮.2016 年麻省理工大学的 Sze 提出了卷积神经

网络加速器 Eyeriss[6].Eyeriss 使用了一种最小化数据传输功耗开销的数据流模型,可以利用 DNN 具有的所有数据重用类型.TPU[5]是谷歌提出的机器学习专用芯片,它使用了 256×256 的脉动阵列加速矩阵乘和卷积,实现了非常高的吞吐量和极短的响应时间.FlexFlow[23]提出了基于多种并行类型互补的卷积实现,提高了计算资源的利用率.此外,它可以为不同的卷积层提供最优的并行混合方案.EIE[7]是 2016 年斯坦福大学提出的用于加速稀疏的全连接层和 RNN 等神经网络的专用加速器.Cambricon-X[8]提出了一款既可以加速稠密神经网络,也可以加速稀疏神经网络的硬件加速器.但是这些专用数据流加速器为了获得更高的性能牺牲了硬件的灵活性,有些加速器甚至不能支持 DNN 的全部层.相较于这些加速器,细粒度数据流体系结构可以提供更高的灵活性和更广的应用范围,针对不同应用自身的特点挖掘其并行性.如本文提出的架构可以用于数据中心[24]加速矩阵乘、FFT、Stencil 等高性能应用,同时也能用于加速神经网络等应用.

2 DNN 全连接层背景

DNN 在各个领域和各种应用中具有不同的形状和尺寸,比较著名的 DNN 模型有 AlexNet[1],VGG-16[2]等,这些模型独特的结构决定了它们能达到的精度和效率.DNN 是由一系列层(layer)构成的,主要有卷积层(convolutional layers, CONV)、池化层(pooling layers, POOL)、全连接层(fully connected layers, FC)、激活层(activation layers, ACT)和局部响应归一化层(local response normalization layers, LRN)等.DNN 的输入是一组需要被网络分析的信息,这些值可以是图像的像素、音频的采样幅度等数值表示.这些输入通过 DNN 的推断后,就会得到相应的分析结果.DNN 的精度还与它各个层中权重的值有关,可以通过训练对网络中的权值进行调整.

全连接层是 DNN 的重要组成部分,是 DNN 所有层中参数最多的层.全连接层通过使用滤波器权重从输入数据中提取特征.全连接层的参数如表 2 所示.

全连接层的输入是由从多张输入图像提取出的一组 2-D 输入特征图(input feature maps)构成.一张输入图像提取出的一组输入特征图称为一个通道(channel),每次输入网络的一组图片称为一个批次(batch).所以输入集是一个四维张量,其参数分别是一个 batch 内的图像个数 N、每张图像的输入特征图数量 C、输入特征图的行数 H、输入特征图的列数 W,本文用 $\boldsymbol{I}\in \mathrm{R}^{NCHW}$ 表示.全连接层的权重(filters)也是一个四维张量,其参数分别是每张图像的输入特征图数量 C、每张图像的输出特征图(output feature maps)数量 K、权重的行数 R、权重的列数 S,本文中用 $\boldsymbol{F}\in \mathrm{R}^{KCRS}$ 表示.在全连接层中,每一个结点都与上一层的所有结点相连.因此输入特征图与权重大小相同,即 $H=R$, $W=S$.多个 2-D filters 会组成一个 filter channel,分别与一个 channel 内对应的 2-D 输入特征图卷积,所得结果累加获得一个输出特征图.每个输入 channel 都会和 K 个 filter channel 卷积,从而获得输出特征图的四维张量,本文中用 $\boldsymbol{O}\in \mathrm{R}^{NKPQ}$ 表示(其中 P 是输出特征图的行数,Q 是输出特征图的列数,$P=Q=1$).

Table 2 Parameters of FC Layers

表 2 全连接层的参数

Parameter	Description
N	Number of images in batch
C	Number of ifmaps per image/number of filters per channel
H	Height of ifmaps
W	Width of ifmaps
K	Number of ofmaps per image/number of filter channels
R	Height of filters
S	Width of filters
P	Height of ofmaps
Q	Width of ofmaps

可以得到全连接层的计算为

$$O[n][k][p][q]=\sum_{c=0}^{C-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1}F[k][c][r][s]\times I[n][c][r][s], \quad (1)$$
$$0\leqslant n<N,\ 0\leqslant k<K,\ p=q=0.$$

从式(1)可以看出,全连接层可以看作是输入特征图与权重大小相同的特殊卷积层.根据全连接的运算特点,当 $N=1$ 时,通常将其转换为矩阵向量乘计算,其形式如图 2 所示.$N>1$ 时,则可以转化为矩阵乘.

$N=1$ 时,得到全连接层的公式为

$$O[k]=\sum_{m=0}^{CRS-1}F[k][m]\times I[m], \quad (2)$$
$$0\leqslant n<N,\ 0\leqslant k<K.$$

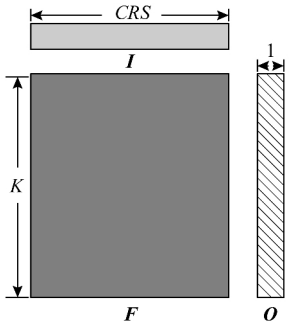


Fig. 2 Computation of FC layers when $N=1$

图 2 当 $N=1$ 时全连接层的计算形式

3 FDPU 结构

3.1 细粒度数据流体系结构概述

FDPU 是面向计算密集型的、具有简单的访存模式和数据重用特征应用的细粒度数据流加速芯片.FDPU 是基于 ASIC 实现的,它支持细粒度数据流指令集,通过最大化数据流指令级并行性提高其性能.FDPU 的总体结构如图 3 所示.FDPU 是由执行单元(processing element, PE)、数据缓存(data buffer, Dbuf)、指令缓存(command buffer, Cbuf)、一个微控制器(micro controller, MicC)和一个直接存储器存取(direct memory access, DMA)组成.PE 阵列呈二维结构排列.Dbuf 和 Cbuf 是用便签式存储(scratch pad memory, SPM)的存储访问形式实现的.Dbuf 分布在 PE 阵列的周围,Cbuf 则位于 PE 阵列的左侧.MicC 用于控制指令在 PE 上的执行.PE,Dbuf,Cbuf,MicC 通过 2-D mesh 网络相互通信.

1) PE.PE 的内部结构如图 3 的下半部分所示.每个 PE 包含可流水的执行单元、指令缓冲、操作数缓冲、指令发射控制器和本地寄存器单元.

2) Dbuf.用于存储数据,被所有 PE 共享.

3) Cbuf.用于存储要映射到 PE 中的指令和 PE 的本地寄存器值.

4) MicC.负责控制加速器的运行过程.执行开始时,CPU 会向 MicC 发送请求启动加速器.然后 MicC 会通过 Mesh 网络把 Cbuf 中存储的指令和本地寄存器的值送到 PE 内.PE 初始化完成后,MicC

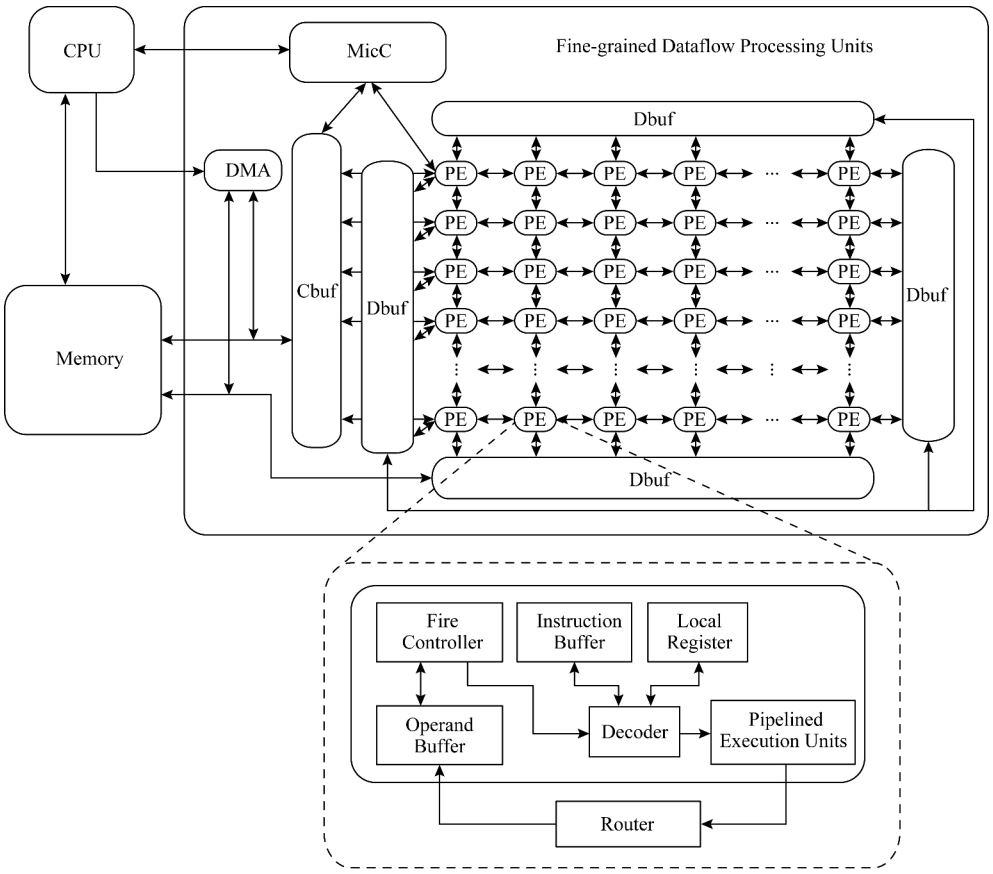


Fig. 3 FDPU architecture diagram

图 3 FDPU 体系结构框图

会向 PE 阵列流水地发送上下文启动指令和上下文 ID,并收集 PE 发来的上下文结束消息.当 PE 阵列将所有上下文都执行完后,MicC 会向 CPU 发送结束消息.

5) NoC.Router 之间采用静态 XY(X 方向优先)的确定性路由策略.

6) DMA.负责 Cbuf、Dbuf 和内存的数据交换.

3.2 并行模式

传统细粒度数据流架构(如 TRIPS[15]),将应用的程序划分为多个块,程序块之间必须串行执行.只有前一个程序块执行完后,才会载入和运行新的程序块,这造成计算部件的利用率不高.因此,FDPU 不只是简单地采用了数据流模式,还采用了循环流水[25]的优化方法提高计算部件的利用率.

1) 数据流模式

数据流模式是一种与传统控制流完全不同的计算模式.在传统控制流处理器中,指令按照程序计数器(program counter, PC)的顺序执行.但是,在数据流模式中,只要指令的操作数准备好了,那么这条指令即可被执行.

在数据流计算中,程序是以数据流图表示的,每条指令的执行结果直接传递到另外一条指令,指令与指令之间通过依赖边来建立依赖关系,从而形成数据流图.数据流图中的每个节点表示一条指令,每条边表示一条指令与另一条指令之间的依赖关系.如图 4(a)所示,把数送入数据流图,指令就会依据依赖关系依次被发射.例如当 Inst3 收到 Inst0 和 Inst1 输出的结果后,Inst3 就可以执行了.

在 PE 中,操作数缓冲中的每一个条目是被一条指令私有的,存储属于这条指令的所有操作数.指令发射控制器会监视指令所需的所有操作数是否都已就位.当指令满足发射条件,指令发射控制器会将指令和其操作数送入译码器中.译码完成后,opcode 和操作数被送入可流水的计算部件中.最终,指令的计算结果会通过 Mesh 网络传输到 Dbuf 中或依赖于这条指令的其他指令的数据缓冲条目中.

数据流图的指令会被映射到 PE 中,图 4(b)给出了图 4(a)中的一种可能的映射结果.文献[26]提出了一种在细粒度数据流体系结构中使用的指令映射算法——基于负载均衡(load balance centric, LBC)的指令映射算法.LBC 算法按照深度的优先顺序依次映射数据流图中的所有指令,对每条指令分别计算执行单元阵列中所有位置的代价,取最小代价的位置作为最佳映射位置.在本文中,为了获得尽可能更优的性能,数据流图的映射则是采用了手动映射的方法.

2) 循环流水模式

数据流程序可以在加速器上被多次执行,一个数据流程序的一次完整执行称为一个上下文.循环流水的优化方法面向具有可分块和并行性特征的应用,进一步利用上下文间的并行处理特征修改数据流结构中上下文切换逻辑,使每一个上下文的开始不需要等待上一个上下文的结束.这样循环流水化结构上的多个上下文以流水线的方式进入执行阵列,数据流图中的每一条指令在执行一次后会立刻接收到下一个上下文的操作数,使执行单元的利用率得到极大的提高.

如图 4(c)所示,多个上下文以流水线的方式流过整个数据图,结果以流的方式从数据流图中流出.Inst5 执行完上下文 1 的数据操作后,立刻接收到上下文 2 的操作数,可以再次执行运算.用户只需要配置 MicC,设置好需要计算的上下文数量.FDPU

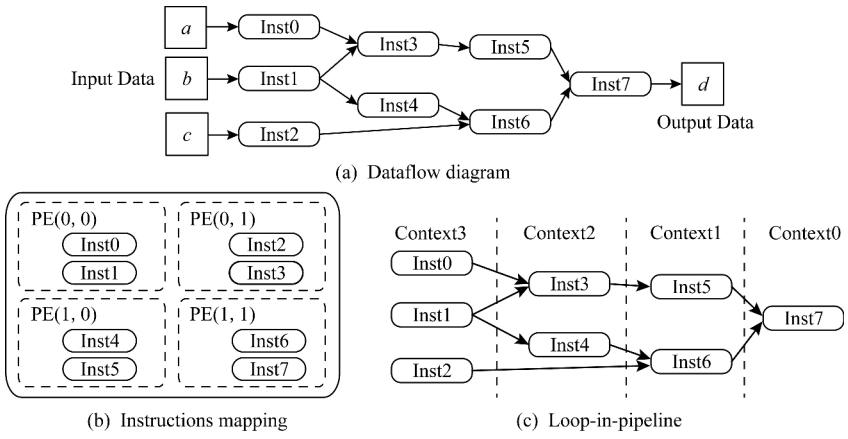


Fig. 4 Process of dataflow executing

图 4 数据流执行过程

开始计算后,MicC 将上下文流水送入执行阵列,并统计已经完成的上下文数量,确定当前计算是否完成.在上下文之间完全没有数据依赖时,所有上下文可以在加速器上充分流水;若上下文间有数据依赖,可通过 MicC 中的上下文控制逻辑控制上下文进入执行阵列的时间,维持有数据依赖的上下文之间的顺序.

3.3 数据流指令集

FDPU 的指令格式如图 5 所示.每个指令由指令码、指令依赖的源操作数个数、立即数、结果的目的地址组成.目的地址指向某个 PE 中的操作数缓冲的地址.FDPU 的指令集包含了基础的运算指令、访存指令和循环指令[27]等.表 3 列出了本文中用到的数据流指令和指令对应的功能.

Opcode	Oprand Number	Immediate	Destination Address3	Destination Address2	Destination Address1	Destination Address0

Fig. 5 Instruction format of FDPU

图 5 FDPU 的指令格式

Table 3 Descriptions of Dataflow Instructions

表 3 数据流指令描述

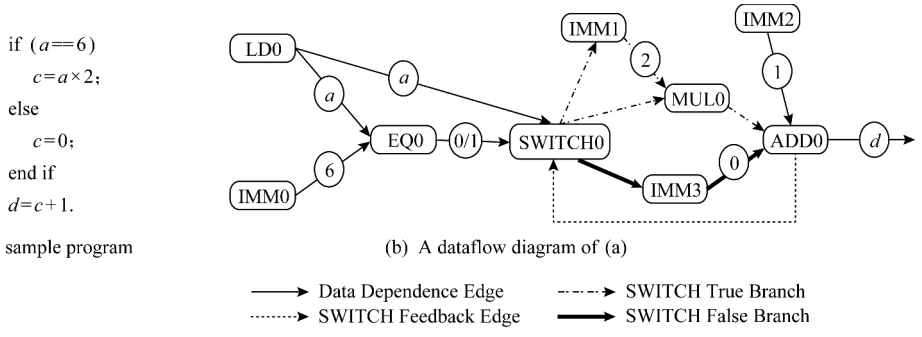
Inst	Type	Function Descriptions
LD	Mix	Load data
ST	Mix	Store data
COPY	Mix	Transmit data to multiple destination
IMM	Integer	Generate integer constant
ADD	Integer	Addition
SUB	Integer	Subtraction
MUL	Integer	Multiplication
EQ	Integer	When two operands are equal, return 1, else return 0
LSR	Integer	Right shift
AND	Integer	Bitwise AND
FIMM	Floating-point	Generate floating-point constant
FMUL	Floating-point	Multiplication
FMADD	Floating-point	Multiply-accumulate

4 基于稀疏的全连接层加速方案

4.1 细粒度数据流分支指令设计

为了增加数据流程序的控制能力、更好地支持稀疏的全连接层,本文设计了用于细粒度数据流结构的分支指令——SWITCH 指令.SWITCH 语句会根据上游给它的第 2 个操作数的值与 0 比大小的结果判断将数据发送给哪些下游指令.图 6 是一个使用 SWITCH 指令的数据流程序例子.图 6(a)是这个数据流程序实现功能的伪代码.如果 EQ 发送给 SWITCH0 的值大于 0,则 SWITCH0 会沿着点划线发送从 LD0 接收来的数据,接着会执行 IMM1,MUL0 指令;如果 EQ 发送给 SWITCH0 的值等于 0,则 SWITCH0 会沿着粗实线发送从 LD0 接收来数据,接着会执行 IMM3 指令.点划线和粗实线最终都会将数据传输到同一个指令的相同操作数位置,保证不论走哪个分支下游指令均可以发射.

可以看到图 6 中 ADD0 指令有一条虚线指向 SWITCH0,这条依赖边是为了保证在循环流水模式下,上下文流过 SWITCH0 的数据顺序与流过 ADD0 的数据顺序是一致的.如图 7(a)所示,在没有 ADD0 指向 SWITCH0 的数据依赖边时,当 2 个上下文的数据在不同的分支流动,后运行的上下文的数据有可能会先于前面的上下文到达后续的指令.这是因为指令执行(数据流指令的发射没有固定的



(a) A sample program (b) A dataflow diagram of (a)

Fig. 6 A dataflow diagram with SWITCH instruction

图 6 使用 SWITCH 指令的数据流图例子

顺序)和数据在片上网络传输的延迟是不确定的,这种情况会造成运算结果的错误.

在汇合指令 ADD0 增加一条指向 SWITCH0 的依赖边可以解决这样的问题,这条边被称为反馈依赖边.SWITCH 指令被增加了一个源操作数用于接收反馈信号,在执行初始状态下,这个操作数被初始化为已经收到;运行过程中,每次 SWITCH 执行后都需要等到分支后的某个汇合指令给它发送的操作数,才能进行下一次运算.如图 7(b)中,ADD0 担任了汇合指令的作用.汇合指令可以是任意种类的指令,但它必须是 SWITCH 指令的 2 个分支汇合后的数据流指令.这样的方式可以在不增加硬件开销的情况下保证 SWITCH 和其后的指令的数据流动顺序一致.

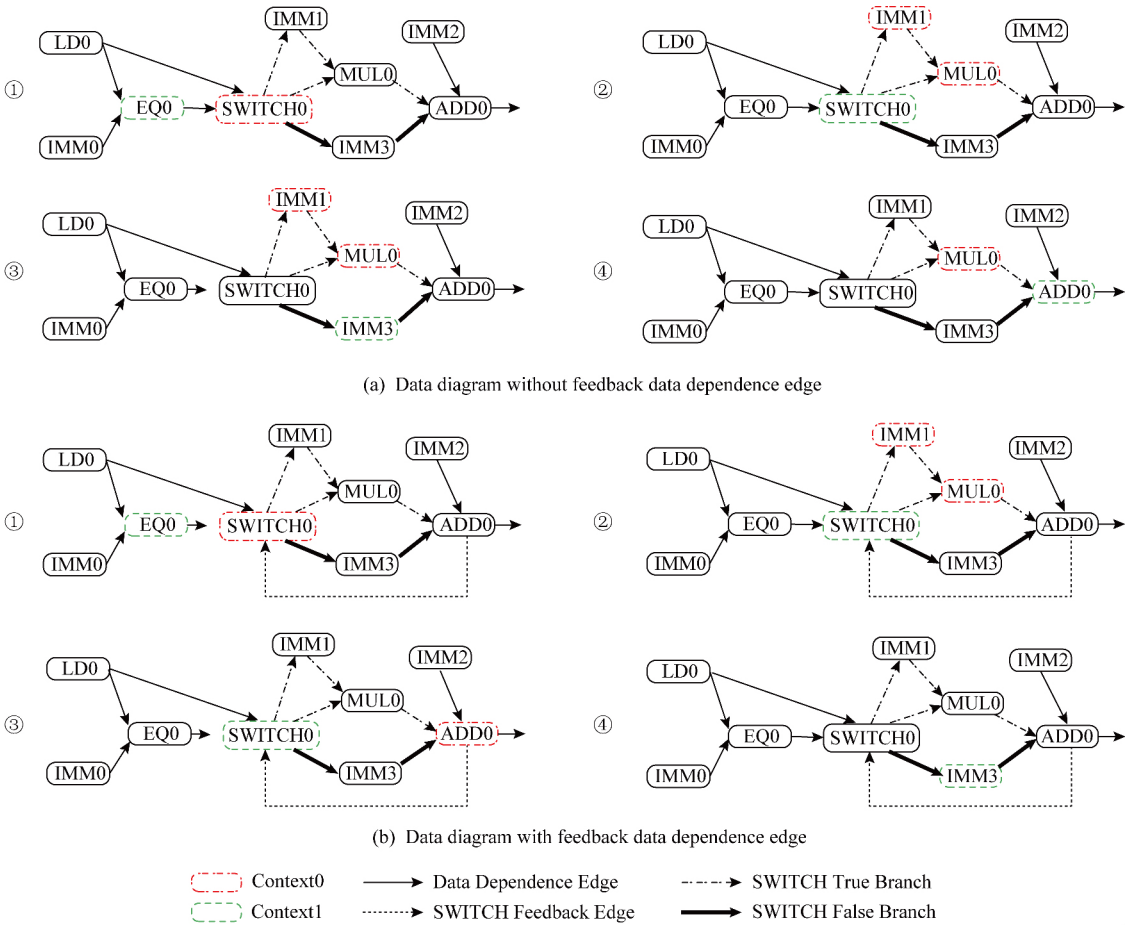


Fig. 7 Function description of the feedback data dependence edge of SWITCH

图 7 SWITCH 指令的反馈依赖边的功能说明

4.2 面向细粒度数据流的稀疏存储格式

常用的稀疏矩阵存储格式有很多,比如坐标格式(coordinate format, COO)、压缩稀疏行格式(compressed sparse row, CSR)等.但是由于数据流程序的控制性较弱,无法灵活地控制循环次数等,在这些格式很难用数据流程序高效解析.比如 COO,它用一个三元组表示,分别是(行号,列号,数值).读取到当前数据的行号与列号后,才能知道数据的位置,很难将数据划分为多个上下文并行处理.而 CSR 格式,虽然其格式可以方便地推算得到每行数据的数量和数据的起始位置,便于划分数据,但是由于每行数据的数量不确定,数据流图无法确定.对于细粒度数据流加速器,需要一种更具有“确定性”的稀疏矩阵存储格式.

本文提出了一种类似于 CSR 的存储格式,称为面向细粒度数据流的压缩稀疏行格式(fine-grained dataflow CSR, FD-CSR).FD-CSR 与 CSR 相同,也由 3 部分组成:数值(values)、列号(column indices)以及行偏移(row offsets).数值用于存储数组中所有的非零元素;列号的每一个比特标志了一行数据中每个元素是否为 0;行偏移则表示每行的第 1 个非零元素在 value 中的偏移位置.通过这样的设计,

可以使列号和行偏移都成为固定长度的数组,方便数据流程序进行读取.

如图 8 所示,一个任意长度的一维数组会被重新排列为每行 16 个元素的二维数组.这是因为每个列号的数据位宽是 16 bit,只能表示最多 16 个元素的有效位.列号的每一个比特代表其对应的数据是否为 0,如第 1 个列号为 6339,其二进制数值为 0001100011000011(最低位对应第 1 行的第 1 个数据,最高位对应第 1 行的最后一个数据).该数组压缩后的列号和行偏移的数据个数均为二维数组的行数,数值的数据个数则为数组中非零元素的个数.

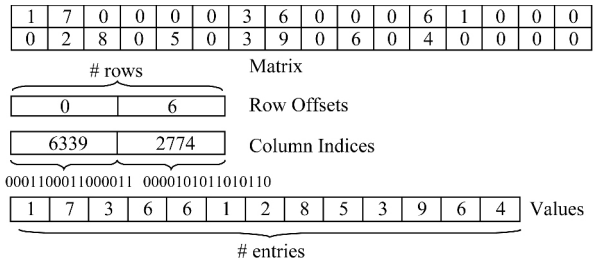


Fig. 8 Format of FD-CSR

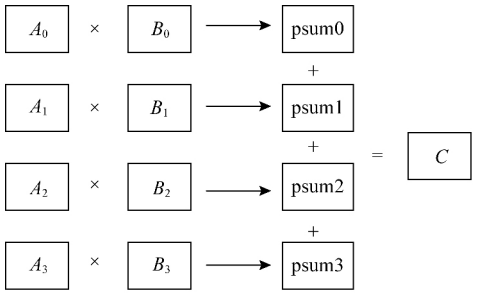
图 8 FD-CSR 稀疏矩阵存储格式

这样的稀疏矩阵存储格式的设计可以将列号和行偏移的个数固定.本文设计的数据流图会通过对列号和行偏移的分析,同时使用 SWITCH 语句,实现对不定个数的数值的解析访问.

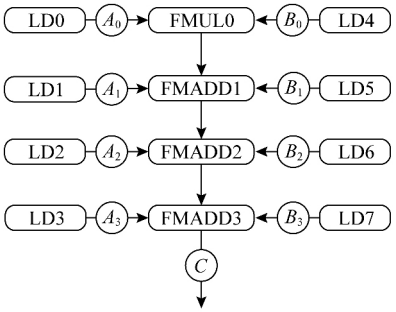
4.3 读取稀疏存储格式的数据流图设计

全连接层的计算通常被转化为矩阵向量乘.矩阵向量乘是由大量的向量点积组成的,其操作如图 9(a)所示,其中 A 代表权重的一行,B 代表输入特征图,C 代表输出特征图的一个元素.A 的每个元素与对应的 B 元素相乘,其结果进行累加,最终得到 C.在稠密的矩阵向量乘中,数据流图被写为图 9(b)的形式.在稀疏的全连接层中,权重使用了稀疏矩阵存储格式进行存储.为了读取稀疏的权重,本文提出了如图 9(c)所示的数据流图.

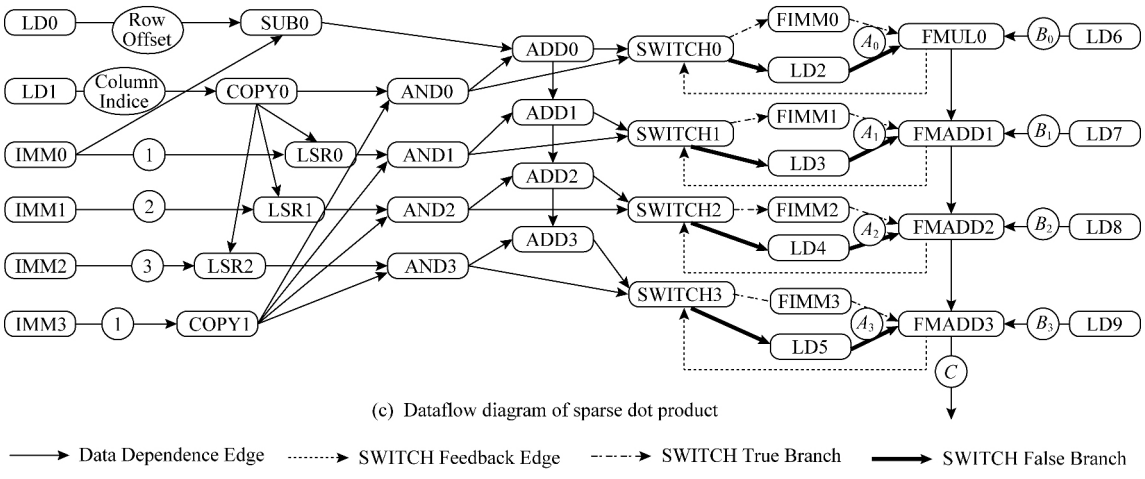
数据流图首先获取了本行的列号和行偏移.列号被共享给了多条指令,AND0 将列号与 0x1 按位与,获得了本行第 1 个数是否为非零数的结果.同样 AND1,AND2,AND3 分别将右移了 1 位、2 位、3 位的值与 0x1 按位与,分别获得了对应的数据是否为非零数的结果.行偏移首先被减一,然后与 AND0 输出的结果相加.如果 AND0 输出 1,那么 ADD0 算出



(a) Operation of dot product



(b) Dataflow diagram of dense dot product



(c) Dataflow diagram of sparse dot product

Fig. 9 Basic operation of FC layers

图 9 全连接层的基本操作

的就是本行第 1 个数在数值向量中的偏移.ADD0 的结果会传给下一个 ADD,只有 AND 输出的值为 1 时,即对应数据是非零数,ADD 的输出结果才是对应数据在数值向量中的地址偏移.AND 与 ADD 的结果均会被传给 SWITCH 指令.当 AND 输出 1 时,SWITCH 会把 ADD 算出的偏移传给 LD 指令,LD 指令从对应的存储地址取数并把获取到的数发送给 FMUL 或 FMADD.当 AND 输出 0 时,SWITCH 则会传输数据给 FIMM,FIMM 则将 0 值发送给下游.FMUL 指令和 FMADD 指令会从 2 条分支中的一条获得向量 A 中的一个元素,并与从 LD 指令传输来的向量 B 的元素相乘.FMADD 指令将乘积与上游送来的 C 的部分和进行累加,最终得到 C.

5 实验与结果

在本节中,我们使用本文提出的方法,在 FDPU 上加速运算了经过剪枝后的 AlexNet 和 VGG-16 中的稀疏全连接层.

5.1 度量标准

本文使用了计算部件利用率(computing resource utilization)来评估稀疏神经网络在不同峰值性能的硬件加速器的优化效果.

$$\text{计算部件利用率}=\frac{\text{测试程序性能}}{\text{结构的理论峰值性能}}\times 100\%. \tag{3}$$

5.2 实验平台

我们以中国科学院计算技术研究所自主研发的大规模并行模拟框架 SimICT[28] 为平台,实现了一个 C 语言的精确于时钟的模拟器.实验环境具体配置如表 1 所示.FDPU 模拟器的结构如图 3 所示,主要包含 ARM 处理器、DMA 控制器、内存和 FDPU 等模拟组件.

FDPU 加速器由 8×8 个 PE、32 个 Dbuf、8 个 Cbuf、1 个微控制器(MicC)和 1 个 DMA 控制器组成.所有 Dbuf 的大小均为 2 MB,每个 PE 内的操作数缓冲为 6 KB.浮点数据均使用 16-bit 表示.每个 PE 有 2 个 32-bit 的定点乘加器、用于计算 load/store 的地址索引以及 16-bit 的 SIMD4 的乘加器.FDPU 运行时的频率为 1 GHz,其峰值性能为 512 Gops.

我们用 verilog 对 FDPU 设计进行了实现和 rtl 级仿真,并对模拟器进行了时钟级的校准.然后用 45 nm 的工艺进行了综合,最终得到的面积约为 44 mm^2,功耗约为 3.27 W.

实验中的对比基准我们使用了 CPU(Intel Core i-7 5930k),GPU(NVIDIA GeForce GTX Titan X)和 Mobile GPU(NVIDIA Tegra K1).其性能数据均来自文献[7].

5.3 数据集

我们选择了 AlexNet 和 VGG-16 中稀疏程度差异较大的 3 个层进行性能对比,这 3 个层分别是 AlexNet-FC6,AlexNet-FC8 和 VGG16-FC7,它们的稀疏性分别为 9%,25%,4%.我们对比了运行稠密的全连接层和稀疏的全连接层的性能.其中稠密的全连接层来自 Caffe model zone,稀疏的全连接层使用了文献[3]中剪枝的算法生成.

5.4 实验和结果

图 10 展示了 FDPU 在加速稀疏全连接层与稠密全连接层所实现的计算部件利用率.我们可以看出,本文提出的稀疏全连接层的加速方法虽然相较于稠密的全连接层增加了大量的指令,但是仅有少量的性能损失.其中在 batch size 为 1 和 4 时,两者的性能差距最高为 1.4%;在 batch size 为 64 时,稀疏全连接层的计算部件利用率平均比稠密的低 11.7%.其主要原因在于 batch size 较小时,全连接层的数据复用程度很低,此时不论是稠密的还是

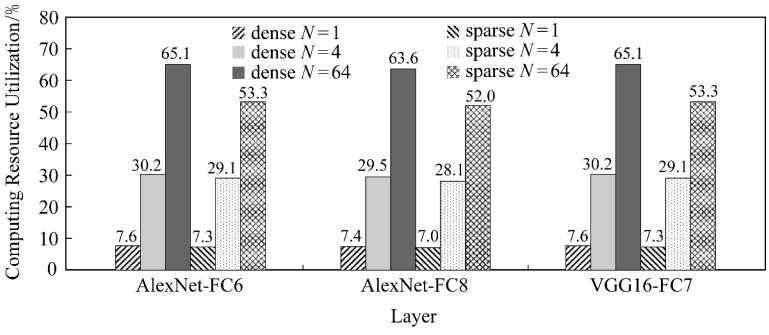


Fig. 10 Computing resource utilization of dense FC layers and sparse FC layers for FDPU

图 10 FDPU 运行稠密全连接层和稀疏全连接层时的计算部件利用率

稀疏的全连接层数据流图中计算指令的比例较低，访存和计算访存地址的指令比例较高.这使得因读取稀疏矩阵数据所增加的指令在访存指令的占比很低，导致稀疏与稠密的全连接层运行时计算部件的利用率相近.而 batch size 为 64 时，因数据在全连接层的复用程度增高，全连接层数据流图中计算指令的比例增加，使得因读取稀疏矩阵数据所增加的指令在访存指令的占比增高，导致了稀疏全连接层的性能较稠密的全连接层有较大的性能下降.

从图 10 中我们还可以发现本文提出的稀疏全连接层的加速方法在不同的稀疏比例的全连接层中计算部件利用率基本相等.这是因为我们提出的方案为了适应细粒度数据流结构，采取了比较稳定的数据流图设计.对于不同的全连接层，其数据流图的流动方式、执行的指令条数相差不大.对于每个权重，不论其是否为非零值，我们都会计算出其对应的地址索引.因此在不同的稀疏比例的全连接层中，FDPU 获得的计算部件利用率非常接近.

图 11 展示了 FDPU 运行稀疏全连接层相较于稠密全连接层对带宽需求的减少比.稀疏全连接层相较于稠密全连接层可以减少 2.44×～ 6.17×的峰值带宽需求.

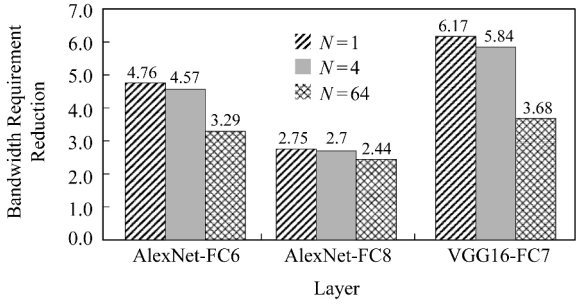


Fig. 11 Bandwidtion requirement reduction of sparse FC layers over dense FC layers

图 11 稀疏 FC 层相较于稠密 FC 层对带宽需求的减少比

表 4 列出了 FDPU 与 CPU，GPU 和 mGPU 在运行 batch size 为 64 的稀疏全连接层时的计算部件利用率对比.可以看到，FDPU 的计算部件利用率远超过其他硬件平台，平均比 CPU，GPU 和 mGPU 分别高了 43.15%，34.57%和 44.24%.

表 5 列出了 FDPU 与 CPU，GPU 和 mGPU 在运行稀疏全连接层时的峰值性能、面积效率和能效等的对比.其中 FDPU 的面积效率是 CPU 和 GPU 的 17.9 倍和 2.7 倍，能效是 CPU，GPU 和 mGPU 的 50 倍、9.7 倍和 7.9 倍.

Table 4 Computing Resource Utilization Comparison of FDPU and Other Platforms for Running Sparse FC Layers

表 4 FDPU 与其他平台运行稀疏 FC 层时的计算部件利用率对比 %

Platform	AlexNet-FC6	AlexNet-FC8	VGG16-FC7
CPU(Core i7-5930k)	7.92	2.99	18.16
GPU(Titan X)	17.32	7.66	29.84
mGPU(Tegra K1)	5.80	4.37	15.64
FDPU	53.26	52.01	53.26

Table 5 Area Efficiency and Energy Efficiency Comparison of CPU, GPU, mGPU, and FDPU

表 5 FDPU 与 CPU，GPU 和 mGPU 的面积效率和能效对比

Parameter	CPU(Core i7-5930k)	GPU (Titan X)	mGPU (Tegra K1)	FDPU
Technology/nm	22	28	28	45
Hardware Area/mm^2	356	601		44
Power/W	73	159	5.1	3.27
Peak Sparse M×V Throughput/Gops	122	1 375	54	273
Area Efficiency/(Gops/mm^2)	0.34	2.29		6.11
Energy Efficiency/(Gops·W^{-1})	1.67	8.65	10.59	83.49

6 关于卷积层的讨论

本文提出的方法主要利用了权重的稀疏性.卷积层在 FDPU 上实现是通过循环展开成矩阵乘的方式实现.因此，卷积层也可以利用这种方法减少存储空间和访存带宽需求.但是在卷积层中权重占输入数据的比重相较于全连接层较小，例如在 batch size 为 64 时，AlexNet 中的 5 个卷积层中权重分别占输入数据的 0.35%，9.43%，19.35%，19.35%和 13.79%.本文提出的方法对于卷积层的存储空间和带宽需求的改善很小，但是会造成一定的性能下降.因此在 FDPU 上目前卷积层还是用稠密的方式存储权重和计算.未来我们会进一步研究如何避免零值权重的计算，从而提高 FDPU 加速卷积层和全连接层的性能.

7 总 结

为了更高效地在细粒度数据流加速器上实现 DNN 的加速，我们提出了一种在细粒度数据流加速器上加速稀疏的全连接层的方法.这种方法可以

减少加速器对内存数据的访问量,并且基本不增加硬件设计,同时对加速器的性能影响也较小.本文提出的加速稀疏的全连接层的方法相较于原有稠密的全连接层运算减少了 2.44×～ 6.17×的峰值带宽需求.在 batch size 为 64 时,FDPU 运行稀疏全连接层的计算部件利用率远超过其他硬件平台,平均比 CPU,GPU 和 mGPU 分别高了 43.15%,34.57%和 44.24%.但是本文中提出的方法无法利用全连接层的稀疏性减少运行的指令条数.未来工作中,我们会继续研究稀疏全连接层的加速,目标是设计出可以利用权重的稀疏性减少计算的数据流图.

参 考 文 献

[1] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks [C] //Proc of NIPS 2012. Cambridge, MA: MIT Press, 2012: 1097-1105

[2] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition [J]. arXiv preprint, arXiv:1409.1556, 2014

[3] Han Song, Mao Huizi, Dally W J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding [J]. arXiv preprint, arXiv: 1510.00149, 2015

[4] Chen Yunji, Chen Tianshi, Du Zidong, et al. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning [C] //Proc of the 19th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2014: 269-284

[5] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit [C] //Proc of the 44th Annual Int Symp on Computer Architecture. New York: ACM, 2017: 1-17

[6] Chen Yu-Hsin, Emer J, Sze V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks [C] //Proc of the 43rd Annual Int Symp on Computer Architecture. New York: ACM, 2016: 367-379

[7] Han Song, Liu Xingyu, Mao Huizi, et al. EIE: Efficient inference engine on compressed deep neural network [C] //Proc of the 43rd Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2016: 243-254

[8] Zhang Shijin, Du Zidong, Zhang Lei, et al. Cambricon-X: An accelerator for sparse neural networks [C] //Proc of the 49th IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2016: No.20

[9] Verdoscia L, Vaccaro R, Giorgi R. A matrix multiplier case study for an evaluation of a configurable dataflow-machine [C] //Proc of the 12th ACM Int Conf on Computing Frontiers. New York: ACM, 2015: 1-6

[10] Milutinovic M, Salom J, Trifunovic N, et al. Guide to Dataflow Supercomputing [M]. Berlin: Springer, 2015

[11] Shen Xiaowe, Ye Xiaochun, Tan Xu, et al. An efficient network-on-chip router for dataflow architecture [J]. Journal of Computer Science and Technology, 2017, 32(1): 11-25

[12] Tan Xu, Shen Xiaowe, Ye Xiaochun, et al. A non-stop double buffering mechanism for dataflow architecture [J]. Journal of Computer Science and Technology, 2018, 33(1): 145-157

[13] Xiang Taoran, Feng Yujing, Ye Xiaochun, et al. Accelerating CNN algorithm with fine-grained dataflow architectures [C] //Proc of 2018 IEEE 20th Int Conf on High Performance Computing and Communications. Los Alamitos, CA: IEEE Computer Society, 2018: 243-251

[14] Dennis J B. First version of data flow procedure language [C] //Proc of Programming Symp, Proceedings Colloque Sur La Programmation. Berlin: Springer, 1974: 362-376

[15] Govindan M S S, Burger D. TRIPS: A distributed explicit data graph execution (EDGE) microprocessor [C] //Proc of 2007 IEEE Hot Chips 19 Symp (HCS). Piscataway, NJ: IEEE, 2013: 1-13

[16] Swanson S. The WaveScalar Architecture [M]. Saint Louis, Missouri: University of Washington, 2006

[17] Chen Guilin, Ma Sheng, Guo Yang. Survey on accelerating neural network with hardware [J]. Journal of Computer Research and Development, 2019, 56(2): 240-253 (in Chinese)
(陈桂林, 马胜, 郭阳. 硬件加速神经网络综述[J]. 计算机研究与发展, 2019, 56(2): 240-253)

[18] Ji Rongrong, Lin Shaohui, Chao Fei, et al. Deep neural network compression and acceleration: A review [J]. Journal of Computer Research and Development, 2018, 55(9): 1871-1888 (in Chinese)
(纪荣嵘, 林绍辉, 晁飞, 等. 深度神经网络压缩与加速综述[J]. 计算机研究与发展, 2018, 55(9): 1871-1888)

[19] Lu Liqiang, Liang Yun, Xiao Qingcheng, et al. Evaluating fast algorithms for convolutional neural networks on FPGAs [C] //Proc of 2017 IEEE 25th Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Los Alamitos, CA: IEEE Computer Society, 2017: 101-108

[20] Lu Liqiang, Liang Yun. SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs [C] //Proc of 2018 55th ACM/ESDA/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2018: 1-6

[21] Lu Liqiang, Liang Yun, Xiao Qingcheng, et al. Evaluating fast algorithms for convolutional neural networks on FPGAs [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019: 1-1

[22] Farabet C, Martini B, Corda B, et al. NeuFlow: A runtime reconfigurable dataflow processor for vision [C] //Proc of 2011 IEEE Computer Society Conf on Computer Vision and Pattern Recognition Workshops (CVPR Workshops 2011). Los Alamitos, CA: IEEE Computer Society, 2011: 109-116

[23] Lu Wenyan, Yan Guihai, Li Jiajun, et al. FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks [C] //Proc of 2017 IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2017: 553-564

[24] Fan Dongrui, Li Wenming, Ye Xiaochun, et al. SmarCo: An efficient many-core processor for high-throughput applications in datacenters [C] //Proc of 2018 IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2018: 596-607

[25] Shen Xiaowei, Ye Xiaochun, Tan Xu, et al. POSTER: An optimization of dataflow architectures for scientific applications [C] //Proc of the 2016 Int Conf on Parallel Architectures & Compilation Techniques (PACT). Piscataway, NJ: IEEE, 2016: 441-442

[26] Shen Xiaowei, Ye Xiaochun, Wang Da, et al. Optimizing dataflow architecture for scientific applications [J]. Chinese Journal of Computers, 2017, 40(9): 223-238 (in Chinese)
(申小伟, 叶笑春, 王达, 等. 一种面向科学计算的数据流优化方法[J]. 计算机学报, 2017, 40(9): 223-238)

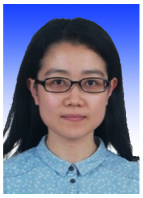
[27] Tan Xu, Ye Xiaochun, Shen Xiaowei, et al. A pipelining loop optimization method for dataflow architecture [J]. Journal of Computer Science and Technology, 2018, 33(1): 116-130

[28] Ye Xiaochun, Fan Dongrui, Sun Ninghui, et al. SimICT: A fast and flexible framework for performance and power evaluation of large-scale architecture [C] //Proc of IEEE Int Symp on Low Power Electronics & Design. Piscataway, NJ: IEEE, 2013: 273-278

Xiang Taoran, born in 1992. PhD candidate. Her main research interests include computer architecture, dataflow architecture.

Ye Xiaochun, born in 1981. PhD, associate professor. Member of CCF. His main research interests include algorithm paralleling and optimizing, software simulation, and architecture for high-performance computing.

Li Wenming, born in 1988. PhD, associate professor. His main research interests include high throughput computing architecture and software simulation.



Feng Yujing, born in 1984. PhD candidate. Her main research interests include computer architecture, heterogeneous system, dataflow architecture.

Tan Xu, born in 1991. PhD candidate. His main research interests include high throughput computing architecture and dataflow architecture.

Zhang Hao, born in 1981. PhD, associate professor. Member of CCF. Associate chief architect of the Godson-T many core processor. His main research interests include high throught CPU microarchitectures and application analysis.

Fan Dongrui, born in 1979. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include many-core processor design, high throughput processor design and low power micro-architecture.