高技术通讯 2023 年 第33 卷 第6 期:568-580

doi:10.3772/j.issn.1002-0470.2023.06.002

面向训练的卷积神经网络加速器设计①

杨 灿② 王重熙 章隆兵③

(*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)

(**中国科学院计算技术研究所 北京 100190)

(***中国科学院大学 北京 100049)

摘 要 随着深度神经网络的广泛应用,对神经网络模型的训练速度需求也不断增长,各类面向训练的加速器应运而生。然而,在训练过程的各阶段,同一个层展现出了巨大的计算差异,计算差异性使得单一数据流结构的加速器在某些阶段的处理上达不到最高的效率。而图形处理器(GPU)等通用性设计通常不能充分地利用各阶段操作的特性使得利用率较低。为了解决这个问题,本文针对卷积神经网络(CNN)训练不同阶段的操作,分别提出了高效的执行方案,设计了一个统一的加速器处理单元硬件结构,能够将所有阶段的执行方案高效地映射到其上运行,并以这个统一的处理单元为基础实现了一个高效的支持训练的卷积神经网络加速器。实验结果显示,基于4 个常用的卷积神经网络模型,卷积层训练的前向过程、反向过程的运算资源利用率分别达到了 77.6%、67.3%,相比于现有主流的利用 Tensor 核心加速深度学习任务的 GPU,运算资源利用率提高了 45.1% 和 41.7%。

关键词 神经网络;训练;加速器;卷积神经网络(CNN)

0 引 言

近年来,随着神经网络在图像、语音、自然语言处理等领域的广泛应用,对神经网络模型训练速度的需求也不断提升,各类针对神经网络模型训练过程的加速器应运而生。Google 推出了 TPUv2、TPUv3[1],支持多类深度学习应用的训练。NVIDIA 的 V100 图形处理器(graphics processing unit,GPU)采用了专门的 Tensor 核心来加速深度学习任务,最新的 A100 GPU 进一步升级了 Tensor 核心,大幅缩短了训练的时间。华为的 Ascend[2] 采用了 3D Cube 单元来实现对卷积以及矩阵乘法的加速。TensorDash[3] 利用训练过程中各类操作数的稀疏性来提高训练的效率。RaPiD[4] 实现了支持超低精度训练的加速器,提高了训练的性能。

相比于推理,神经网络的训练不仅包含了前向计算的过程,还包含了误差反向传播和权值梯度计算的过程,同一个层在训练各不同阶段需要进行的计算有很大的差别。为了处理计算差异性,一种方式是采用通用结构的加速器,如 GPU,将训练不同阶段的操作都化为矩阵乘法来进行。但是,化成矩阵乘法可能损失原操作的特征,例如一些数据复用机会等,使得不能针对具体的操作实现更高效的处理;另外,为了通用性,加速器通常会采用复杂的调度方案,进一步影响加速器的性能。

另一类加速器是采用数据流的空间结构加速器,数据流一般针对操作的特性进行了优化,能够实现更高的性能和能效。文献[5]提出了一个权值固定的数据流,实现了对权值的复用。文献[6]采用

① 中国科学院战略性先导科技专项(XDC05020100)资助项目。

② 女,1992 年生,博士生;研究方向:计算机系统结构,加速器设计;E-mail:798383855@qq.com。

③ 通信作者,E-mail:lbzhang@ict.ac.cn。

(收稿日期:2022-01-19)

了输出值固定的数据流,能够减少计算过程中产生的部分结果与各级内存的交互。文献[7]提出了行固定的数据流,能够实现对输入值、权值和输出值的多重复用,最小化数据移动的能耗。文献[5-7]都是单一的数据流,文献[8,9]指出不同操作的最佳数据流可能是不相同的。因此,一个数据流不能完全高效地支持不同形状、不同阶段层的加速。为了解决这个问题,文献[10]提出了一个灵活数据流结构,能同时利用层内多种并行性提高加速器的资源利用率,但只考虑了神经网络的推理。文献[11]针对生成对抗网络训练过程中不同类型的卷积,设计了跳过0的输出值固定、跳过0的权值固定2种数据流,分别使用单独的硬件资源实现。文献[9]针对多深度神经网络(deep neural network, DNN)工作负载采用了多数据流加速器来提供高度的灵活性和利用率。文献[9,11]为了有计算差异的操作都能得到高效的处理,采用了专门的硬件资源来实现每一个子数据流的结构。但是,在卷积神经网络(convolutional neural network, CNN)模型的训练过程中,由于批归一化(batch normalization, BN)[12]等技术的应用,通常找不到多个不同的工作负载同时处理,这使得一部分子数据流硬件资源得不到高效的使用。

针对以上问题,本文设计实现了一个面向CNN训练的加速器,以一个采用一套运算资源和存储资源统一的处理单元硬件结构为基础,实现对CNN训练各阶段操作的高效执行。首先,根据卷积层在训练各阶段操作的要求,分别提出高效的执行方案,能够跳过所有阶段在垂直方向上的冗余计算,并提供多类数据复用机会。然后,设计了一个统一的加速器处理单元硬件结构,能够利用各类数据复用机会减少对各级内存的访问,并且跳过水平方向的冗余计算,使得卷积层在训练所有阶段的执行方案都能高效地映射到该统一的处理单元硬件结构上执行;并用这个处理单元作为基础,设计实现了一个面向CNN训练的加速器。实验结果显示,本文提出的加速器关于卷积层训练的前向过程、反向过程的运算资源利用率分别达到了77.6%、67.3%,比现有主流的GPU的运算资源利用率高了45.1%、41.7%。

1　训　练

本节主要介绍了DNN的训练过程,特别是卷积层训练过程中的各阶段的具体操作。

1.1　训练过程

一个DNN的训练主要包含了前向计算、反向计算2个阶段。前向传播(forward propagate, FP)过程按照DNN网络的顺序,将mini-batch个样本的输入特征图输入到网络中进行计算,得到最终的输出值以后,计算出误差值,然后开始反向计算过程。反向计算过程包含2个部分,一是将前向过程计算出来的误差值根据链式法则逆着网络按层传播,称为反向传播(backward propagate, BP)过程;二是利用每层的误差值与输入特征图值计算出该层的权值梯度,称为权值梯度产生(weight gradient generate, WG)过程。最后,利用算好的权值梯度,根据训练算法中的更新权值的方法,对权值进行更新,得到新的权值后,再开始下一批样本的训练。分别用ifmap和ofmap表示FP过程中每层的输入激励数据、输出结果数据;用ierr和oerr表示BP过程中每层的输入误差数据和输出结果数据。

1.2　卷积层

在图像分类、逐像素语义分割、目标检测、实例分割、人体关键点检测、视频分类等任务中的常用DNN模型里,卷积层占据了最主要的运行时间。

表1中给出了文中用到的卷积层关于输入数据、卷积核、输出数据、滑动步长的记号。

表1　卷积层中各记号代表的含义

记号	含义
IH、IW、IC	ifmap/oerr的高度、宽度、通道数
OH、OW、OC	ofmap/ierr的高度、宽度、通道数
KH、KW	卷积核的高度、宽度
S	卷积层的滑动步长

1.2.1　FP过程

卷积层的FP过程是一个3D跨步卷积操作,如图1(a)所示。这个卷积操作使用$IH \times IW \times IC$的ifmap作为输入数据,使用OC个$KH \times KW \times IC$的卷

积核组成权值,采用该卷积层的滑动步长 S 作为 FP 卷积操作的滑动步长。ifmap 分别与每个卷积核做 3D 卷积,各得到一个 $OH \times OW$ 的 ofmap 通道,共计算出 $OH \times OW \times OC$ 的 ofmap 作为输出结果。

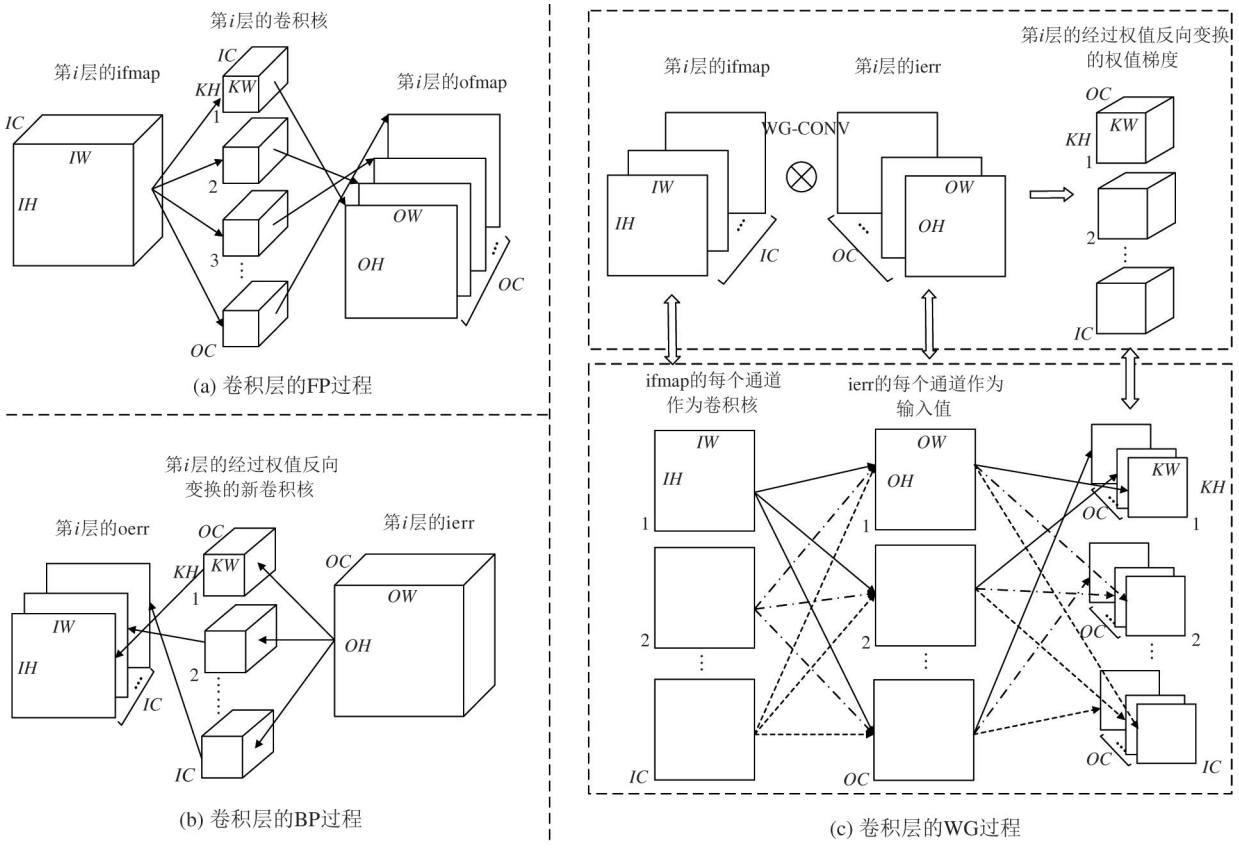


图 1　卷积层的 FP、BP、WG 过程

1.2.2　BP 过程

卷积层的 BP 过程是一个 3D 转置卷积操作,如图 1(b)所示。使用 $OH \times OW \times OC$ 的 ierr 作为原输入数据,按照图 1(a)中所示的方法对 ierr 的每个通道进行扩展——在 ierr 相邻行间插入 $S-1$ 行的 0,在 ierr 相邻列间插入 $S-1$ 列的 0,再按规定进行 padding 操作后得到该卷积操作真正的输入数据 ierr _ ext。将 FP 过程中使用的 OC 个卷积核进行“权值反向变换”操作——每个通道先旋转 180 °,再将同一个通道拼接在一起,形成新的 IC 个卷积核,构成该卷积操作的权值。采用 1 作为该卷积操作的滑动步长,进行 3D 卷积操作,算出 $IW \times IH \times IC$ 的 oerr 作为输出结果。

1.2.3　WG 过程

卷积层的 WG 过程如图 1(c)所示,与 BP 相同,使用 $OH \times OW \times OC$ 的 ierr 作为原输入数据,并对 ierr 进行同样的扩展得到真正的输入数据 ierr _ ext。使用该层的 ifmap 的第 i 个通道作为卷积核,与 ierr _ ext 的所有通道分别做步长为 1 的 2D 卷积操作,得到经过权值反向变换操作的第 i 个卷积核的权值梯度,i 取值为 $1 \sim IC$。

1.3　卷积层训练过程的差异性

首先,FP 和 BP 都是 3D 卷积,但 WG 是 2D 卷积,不包含多个输入通道的结果累加得到一个输出通道结果的过程。

其次,自从 VGG[13] 中提出可以用多个 3×3 的卷积核构成的卷积层来代替较大卷积核的卷积层后,后续的 CNN 设计中,卷积层多采用宽度、高度较小的卷积核。文献[14]提出的神经架构搜索算法使用了 3×3 和 5×5 的卷积核作为基础组成构件。考察 VGG-19、ResNet-34/50、MobileNet V2/V3、GoogleNet、MNASNet 等 10 个常用的 CNN 模型中各

层的卷积核大小,发现卷积核的大小常为1、3、5,其中超过5的层数少于5%,因此,FP和BP使用的卷积核的宽度和高度都较小。而在一般的训练过程[13,15-16]中,通常先对图像进行预处理,将图像大小调整为224×224后再开始计算,从前至后的卷积层的宽度和高度逐渐减小,从224缩减为7,由于WG使用ifmap作为卷积核,于是,其卷积核的宽度和高度的取值范围较大,取值从几到几百不等。

2　卷积层各阶段的执行方案

将一行输入值与一行卷积核值做卷积算得一行输出值的部分结果的操作称为“卷积行单位操作”。本节以卷积行单位操作为基本单位,省略对多个输入通道的描述,分别提出卷积层的FP、BP、WG 3个阶段的执行方案。

2.1　FP的执行方案

FP阶段是跨步卷积。图2展示了滑动步长S为1的卷积层的FP执行方案。将输入值行、卷积核行、输出值行都分成1类,执行分为1步。在这步中,将输入值行组成新的输入数据图input_n,处理的卷积核类共3行,采用一个高度为输出值行数(这里取14)的计算框在input_n上从上至下按步长1滑动2次,得到3个计算框。这3个计算框和3行卷积核值一一对应。对于每对计算框与卷积核行,用该卷积核行与计算框中的所有输入值行做卷积行单位操作,计算出输出值的部分结果。然后将关于3个计算框的3个部分结果按位置累加起来,得到最终14行输出值。

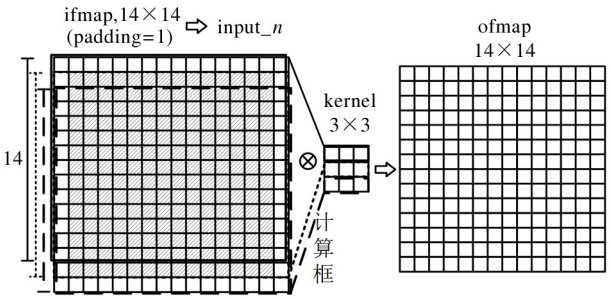


图2　步长S=1的卷积层的FP的执行方案

图3展示了步长S为2的卷积层的FP执行方案。将输入值行、卷积核行按照行号模2分别分成2类,输出值行分成1类。执行共分为2步,每步都完成一类输入值行与一类卷积核行的相关计算,算出7行的部分结果,其中每步内的执行方式与图2中的FP执行方案完全一致。最后,再将2个步骤中的7行部分结果按位置累加起来,得到最终的输出值行。

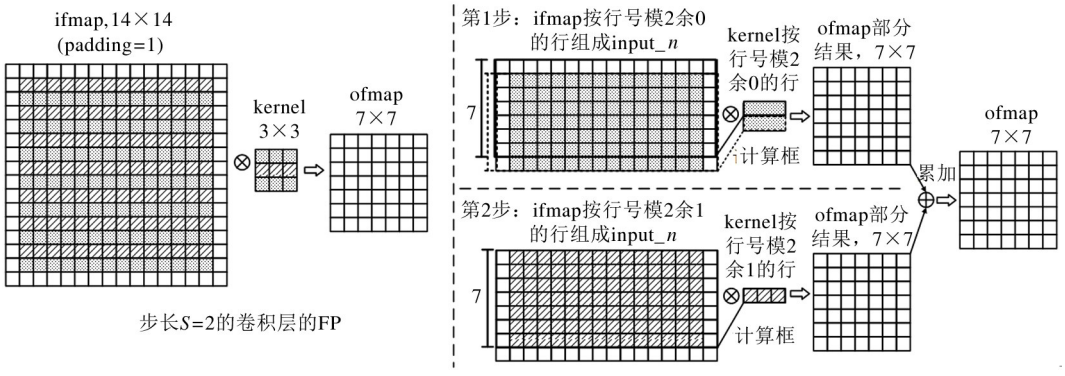
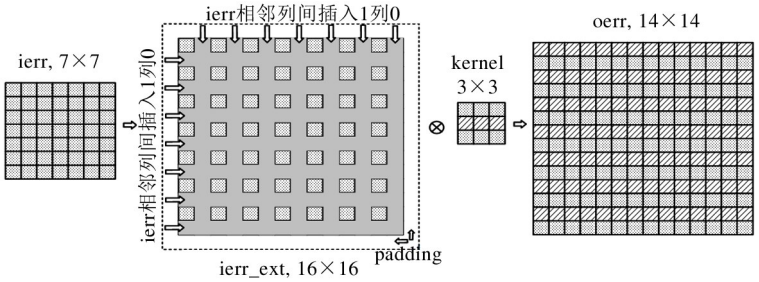


图3　步长S=2的卷积层的FP执行方案

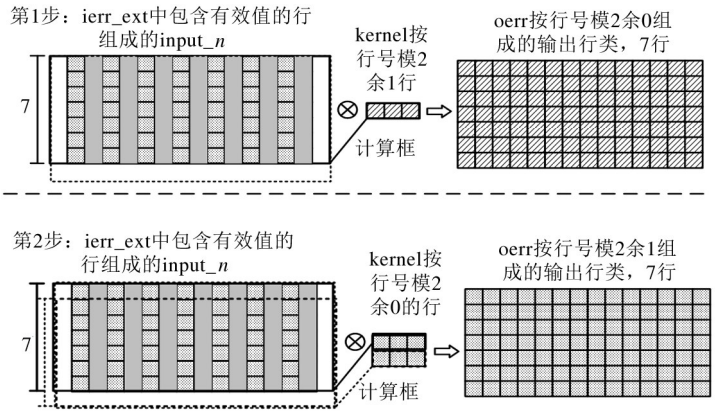
2.2　BP的执行方案

图4(b)展示了步长S为2的卷积层的BP执行方案。将卷积核行、输出值行按照行号模2的结果分别分成2类,将ierr_ext中包含原输入数据(ierr)的行组成新输入数据图input_n。执行分成2步,每步都完成input_n与一类卷积核行的相关计算,得到一类的输出值行,其中每步内的执行方式与图2中的FP执行方案完全一致。

FP和BP的每个步骤都进行input_n与一类卷积核行的计算,得到一类输出值行。其中,每步具体的执行方式与图2的FP执行方案一致。



（a）步长 $S=2$ 的卷积层的 BP



（b）步长 $S=2$ 的卷积层的 BP 执行方案

图 4　步长 $S=2$ 的卷积层的 BP 执行方案

2.3　WG 的执行方案

图 5(b)展示了步长为 2 的卷积层的 WG 执行方案。将卷积核行、输出值行按照行号模 2 的结果分别分成 2 类，将 ierr _ ext 中包含原输入数据(ierr)的行组成新输入数据图 input _ n。执行分为 2 步，每步都完成 input _ n 与 1 类卷积核行的相关计算，得到 1 类的输出值行。和 FP、BP 的执行方式不同，第 1 步使用一个高度为该类卷积核行数(这里取 7)的计算框，在 input _ n 上从上至下按照步长 1 滑动 1 次，得到 2 个计算框，与 2 行输出值一一对应。对于每对计算框与输出值行，用卷积核类的每一个行分别与计算框的每一个输入行做卷积行单位操作，并将结果按位置累加，得到该对的那个输出值行结果。第 2 步的执行方式和第 1 步完全一致，采用高度为 7 的计算框，得到 1 对计算框与输出值行，计算出该输出值行的结果。

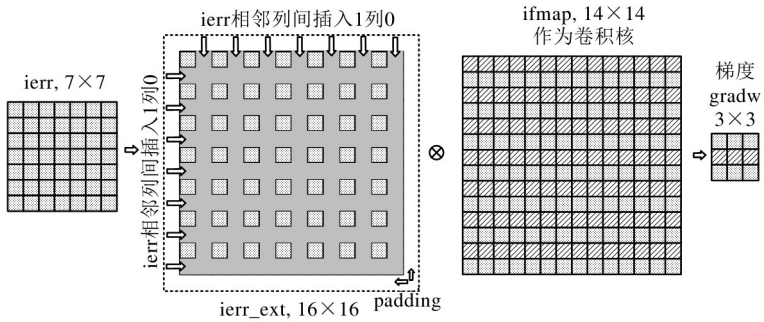
2.4　执行方案的优点

首先，该方案跳过了各个阶段垂直方向的冗余计算。执行 FP 时，在垂直方向上只进行跨步 S 后的卷积计算，相比于步长为 1 的卷积，减少了 $\frac{S-1}{S}$ 的计算量；执行 BP 和 WG 时，仅使用包含原输入数据(ierr)的行组成新输入数据图进行计算，消除了关于 ierr 相邻行间插入的 $S-1$ 行 0 的所有计算。

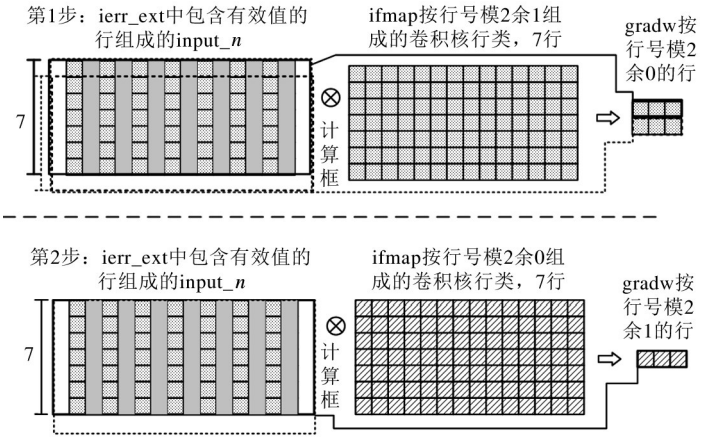
另外，该方案提供了多类数据复用机会。一个步骤内的相邻 2 个计算框中共用多行输入值，仅有 1 行不同，提供了对输入值的复用机会；FP 和 BP 的步骤中每对计算框与卷积核行的计算，提供了对卷积核行的复用机会；WG 的步骤中每对计算框与输出值行相关的计算，提供了对输出值的复用机会。

3　加速器结构

基于上述执行方案，本文提出了一个统一的加速器处理单元硬件结构，能够将 FP、BP 和 WG 的执行方案高效地映射到该结构上执行，并利用这个处理单元设计实现了一个支持训练的加速器。



(a) 步长 $S=2$ 的卷积层的 BP



(b) 步长 $S=2$ 的卷积层的 BP 执行方案

图 5　步长 $S=2$ 的卷积层的 WG 执行方案

3.1　加速器的硬件结构

加速器整体结构如图 6 所示,计算部分由 M 行 N 列的处理单元(processing unit, PU)组成,每列中

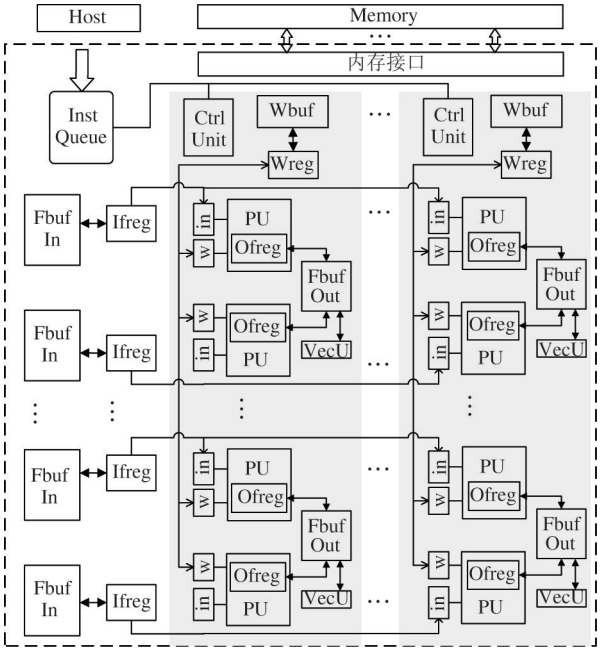


图 6　加速器的整体结构

2 个 PU 共享一块片上缓存 Fbuf _ Out,主要用来存储输出值以及 WG 的卷积核值。每个 PU 行对应一块片上缓存 Fbuf _ In 和输入寄存器堆 Ifreg,给该行中每个 PU 提供输入值。每个 PU 列对应一块片上缓存区 Wbuf 和卷积核寄存器堆 Wreg,给该列中每个 PU 提供卷积核值。另外,每块 Fbuf _ Out 都连接了一个向量单元(vector unit, VU),用来执行其他的向量操作。有一个指令队列 Inst _ Queue,用来从 Host 处接收指令并转换成控制信号控制每个 PU 列的执行。

统一的处理单元硬件结构如图 7 所示,主要包含了运算模块 dotp _ 4 ×2 和输出值寄存器堆 ofreg。dotp _ 4 ×2 由 4 行 2 列的 dotp 组成,dotp 是一个点积累加单元,其结构如图 8(a)所示。dotp 共有 3 个输入,其中 2 个是向量输入 $\boldsymbol{A}=(a_0, a_1, a_2, a_3)$ 和 $\boldsymbol{B}=(b_0, b_1, b_2, b_3)$,$a_i$ 和 b_i 都是 16 比特的浮点数(fp16);另外一个是标量 c,为 32 比特的浮点数(fp32)。dotp 完成点积累加操作 $out=\sum_{i=0}^{3} a_i \times b_i + c$,输出一个 fp32 的结果。ofreg 共有 8 项,每项是

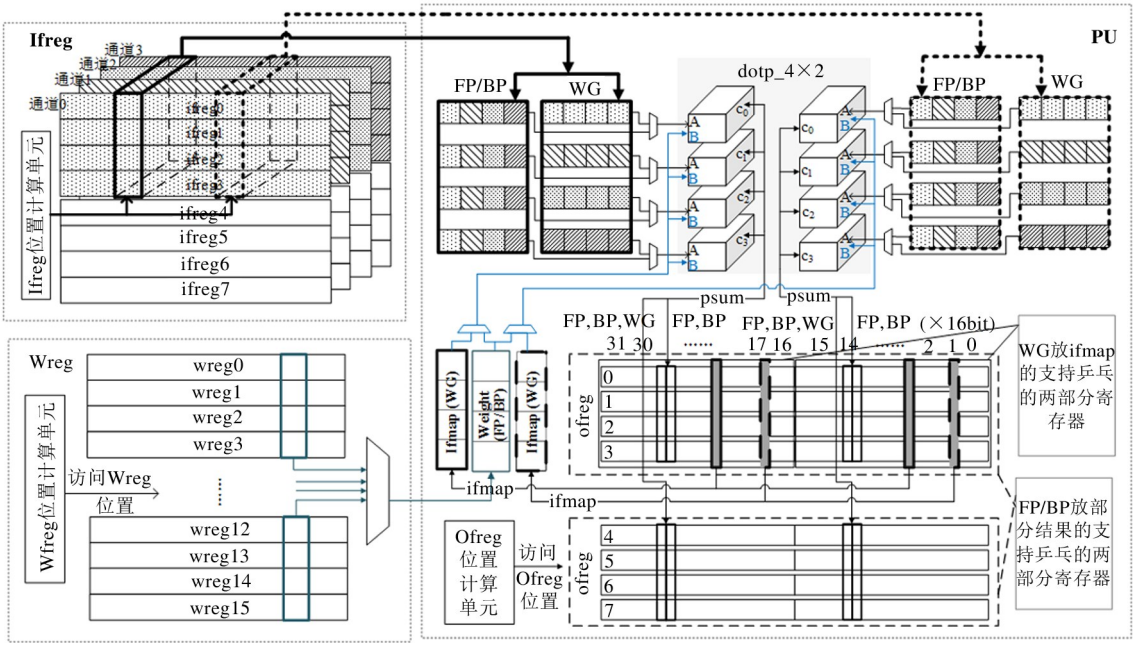
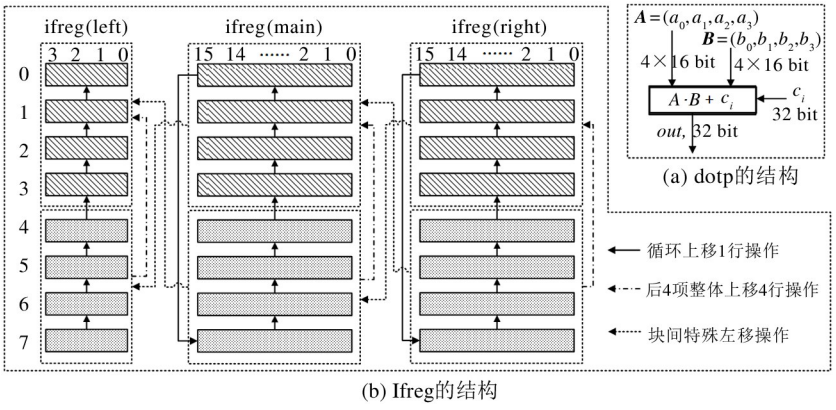


图 7　统一的加速器处理单元硬件结构



(a) dotp的结构

(b) Ifreg的结构

图 8　dotp 与 Ifreg 的结构

32 × 16 比特,用来存储计算过程中产生的部分结果,以及在 WG 过程中给 dotp _ 4 × 2 提供源操作数。

图 7 中还给出了给运算模块提供操作数的卷积核寄存器堆 Wreg 和输入值寄存器堆 Ifreg。Wreg 共有 16 项,每项是 22 × 32 比特,在 FP、BP 过程中用来放置卷积核值;在 WG 过程中用来放置权值梯度 gradw,每项可以平分成 2 份支持乒乓操作。Ifreg 结构如图 8(b)中所示。Ifreg 共有 8 项,每项都分成 3 块,ifreg(left)是 4 × 16 比特,ifreg(main)和 ifreg(right)都是 16 × 16 比特。每块中各项 ifreg 支持循环上移 1 行操作和后 4 项整体上移 4 行操作。相邻块间支持块间特殊左移操作——将右边块中的 8 个项依次串起来,形成一个环,根据指示信息分别选择第 4、5、6、7 项或第 0 项作为“头”,形成一个新的 8 项的串,左移到相邻块的 8 项中。

3.2　卷积层的 FP 和 BP 的映射方式

3.2.1　计算划分与执行

将 FP、BP 一个步骤内的计算框与输出值类都按照 4 行划分,如图 9 所示。以 4 行的输出值块的计算为单位,映射到多个 PU 上并行执行或者单个 PU 上依次执行。记该步中处理的卷积核类的行数为 *KN*。对于每个 4 行的输出值块,使用对应的 4 行的计算子框从 input _ *n* 的相应位置处从上至下按步

长 1 滑动 $KN-1$ 次,形成 KN 个计算子框,与 KN 行卷积核一一对应。将 KN 对计算子框与卷积核行的计算依次映射到同一个 PU 中执行。其中,每对计算子框与卷积核行的计算包含了 4 个卷积行单位操作。执行时,将这 4 个卷积行单位操作同时分配给 PU 中的 4 行 dotp 并行处理,计算出 4 行部分结果,暂存在 ofreg 的前 4 项或后 4 项中。

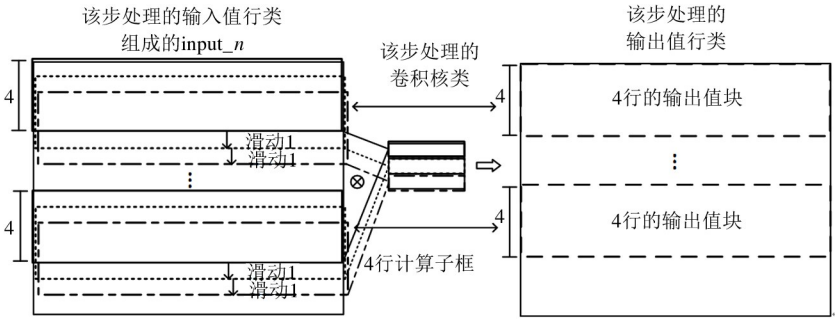


图 9　FP、BP 的计算框按 4 行划分示意图

为了减少 ofreg 与 Fbuf _ Out 的交互次数,采用了输出固定的计算顺序,将 4 行输出值按 16 列分块依次计算。如图 7 所示,将 8 项 ofreg 分成 2 份乒乓使用,4 项用来存储正在计算的 4×16 的输出值块的部分结果;另外 4 项存储上次计算好的输出值块,用来进行一些后续操作,例如格式转换、将结果写回到 Fbuf _ Out 中等。这样,就使得关于不同输出值块的计算能够持续进行。

FP 和 BP 都是 3D 卷积,采用 dotp 来实现每次 4 个输入通道的累加过程,以便减少 psum 产生的次数,进而减少对 ofreg 的访问。为了支持 dotp 的计算,如图 7 所示,处理单元 PU 对应了 4 份 Ifreg,分别存储来自 4 个通道的输入值行。16 项 wreg 中分别存储来自 16 个输入通道的卷积核行。计算时,分别从 Ifreg 前 4 项同一位置读出来自 4 份 Ifreg 的输入值,把 4 项中的值分别送到 4 行 dotp 中作为输入 $\boldsymbol{A}$;从 4 项 wreg 的同一位置读出 4 个输入通道的卷积核值,广播到每行的 dotp 中作为输入 $\boldsymbol{B}$;从 4 项 ofreg 的同一位置读出 4 个累积部分结果,分别送到 4 行 dotp 中作为输入 c,执行点积累加操作,得到 4 个新部分结果,分别写回到 4 项 ofreg 的相同位置中去。

3.2.2　复用权值与输入值的执行方式

(1)权值复用

一对计算子框与卷积核行的计算过程中,4 行 dotp 同时处理 4 个卷积行单位操作,此时将卷积核行中的 1 个权值广播到 4 行 dotp 中参与计算,实现了对权值的复用。

(2)输入值行间复用

相邻 2 个计算子框中有 3 行输入值是相同的。于是,在使用 ifreg 前 4 项中的数据进行一对计算子框与卷积核行的计算的同时,将下一个计算子框中新增的那行输入值读到 ifreg4 中。待前 4 项的计算完成以后,利用 ifreg 的循环上移 1 行操作,将下一个计算子框中的输入值移动到 ifreg 前 4 项中参与计算,实现了输入值的行间复用。

(3)输入值块间复用

执行 FP 时,计算 O 列输出值需要 $(O-1)\times S+KW$ 列的输入值。如图 10 所示,一个 $KW=5$、$S=2$ 的卷积层,计算出 16 列输出值需要 35 列的输入值。将输入值按 16 列分块,于是 35 列输入值分布在第 1 ~ 4 块中。以图 10 为例,计算时,首先将 4 行第 1、2、3 块输入值分别读到 ifreg(left)、ifreg(main)、ifreg(right)中,计算出 0 ~ 7 列输出值的结果。在此过程中,随着 ifreg 的循环上移 1 行的操作,被移出的行存到了 ifreg 的第 4 ~ 7 项中。待前 8 列输出值计算完成后,使用移出的行数作为指示,进行块间特殊左移操作,将第 2、3 块输入值分别移动到 ifreg(left)和 ifreg(main)中,仅需新读入第 4 块输入值到 ifreg(right)中,即可完成 8 ~ 15 列输出值的计算。于是,实现了对第 2、3 块输入值的块间复用。

(4)输入值无行间、块间复用

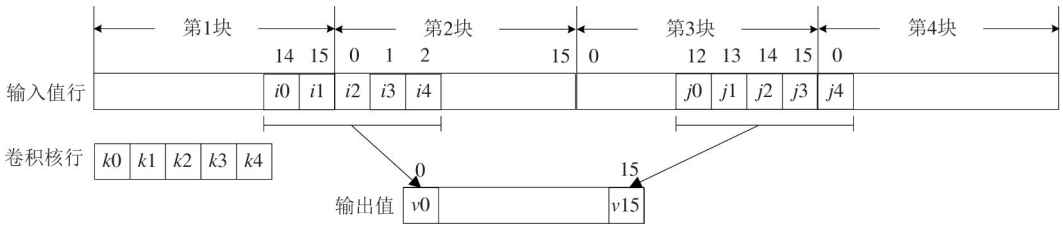


图 10　16 列输出值对应的输入值列

当下次的计算所需的输入值与 ifreg 中已读好的值不存在行间复用或块间复用时，使用 ifreg 前 4 项中的值进行计算的同时，从 Fbuf _ In 中预取下一次计算需要的 4 行输入值到 ifreg 后 4 项中。待当前计算完成后，使用 ifreg 后 4 项整体上移 4 行操作，将新的 4 行输入值移动到 ifreg 前 4 项中，即可开始新一次的计算。

执行时，通过对权值和输入值的复用，减少了从 Fbuf _ In 和 Wbuf 中读数据的次数；另一方面，在输入值无复用机会时，采用预取输入值的方式，能够减少给运算单元提供源操作数时被阻塞的机会，从而保证了运算单元的性能。

3.2.3　卷积行单位操作的执行

如图 7 所示，处理单元中使用 2 个 dotp 来执行同一个卷积行单位操作。计算过程中，采用 3 个灵活的位置计算单元——ifreg 位置计算单元、wreg 位置计算单元、ofreg 位置计算单元来得到 dotp 中源操作数在 ifreg、wreg、ofreg 的位置和 dotp 的结果在 ofreg 的位置。执行 FP 时，利用灵活的索引计算单元，只进行跨步 S 后的计算；执行 BP 时，不执行输入值列间插入的 $S-1$ 列 0 的相关计算。

当输出值按 16 列划分时，最后一块输出值的长度可能小于 16。于是，将最后一块输出值平分给 2 个 dotp 执行；同样地，利用 3 个位置计算单元得到源操作数和目标结果的位置。当计算出最后一块输出值的所有结果后，提前结束。

于是，使用 3 个灵活的位置计算单元，能够跳过 FP 和 BP 水平方向上由不同类型的卷积带来的冗余计算，并实现对任意列数的输出值块的高效支持。

3.3　卷积层的 WG 的映射方式

WG 的卷积核较大，在一项 wreg 中可能放不下一整行；WG 输出值取值较小，通常不足够按 4 × 16 的块划分；WG 是 2D 卷积，没有将多个输入通道结果累加的过程，按照 FP、BP 在 dotp 上的执行方式效率很低。因此，FP、BP 的执行方式对于 WG 是不适合的，需要提出针对 WG 的专用映射方式。

3.3.1　计算划分与执行

将 WG 一个步骤内计算框和卷积核类都按照 4 行划分，如图 11 所示，一个 4 行的计算子框与一块 4 行的卷积核一一对应。记该步中处理的输出值类的行数为 OM。对于每个 4 行的卷积核块，使用对应的计算子框从 input _ n 的相应位置处从上至下按步长 1 滑动 $OM-1$ 次，形成 OM 个计算子框，分别与该 4 行卷积核块进行计算，得到 OM 行输出值。以 4 行卷积核块与 OM 个计算子框的所有运算为单位，映射到多个 PU 上并行执行或者单个 PU 上依次

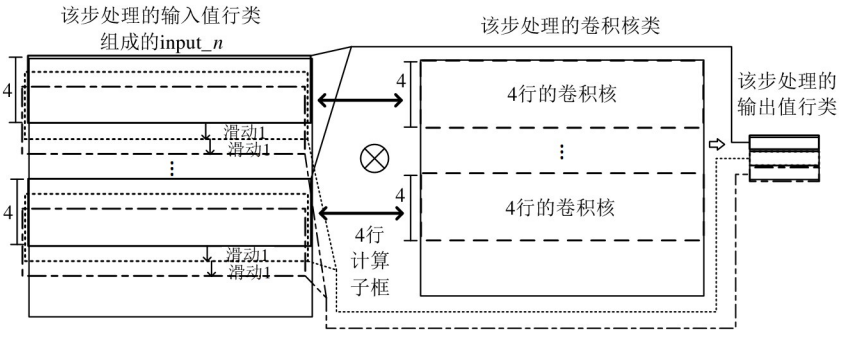


图 11　WG 的计算框按 4 行划分示意图

执行。在该单位运算内部,将 4 行卷积核块按 $16 \times S$ 列划分成 $4 \times 16S$ 的子块,以每个 $4 \times 16S$ 的子块与 OM 个计算子框中相应部分输入值的运算为“子单位运算”,将子单位运算分配到同一个 PU 中的一行 dotp 中依次执行。其中,每次处理 1 个计算子框与 $4 \times 16S$ 的子块相关的计算,由于每次处理的计算包含了 4 个卷积行单位操作并按位置累加部分结果,于是直接利用 dotp 实现 4 行部分结果的累加操作。

同一个 PU 中的 4 行 dotp 分别同时执行 4 个通道的 ierr 与同一通道的卷积核的上述运算。与 FP、BP 相同,将 4 个通道的 ierr 存到 4 份 Ifreg 中;将一个通道的 ifmap 读入到 ofreg 的前 4 项中,为 PU 提供源操作数。

如图 7 所示,分别从 4 份 Ifreg 的前 4 项中的同一位置读出 4 个值,分别送到 4 行 dotp 中作为输入 $\boldsymbol{A}$;从 ofreg 前 4 项中的同一位置读出 4 个 ifmap 值广播到 4 行 dotp 中作为输入 $\boldsymbol{B}$;从 ofreg 后 4 项的同一位置读出 4 个累积的部分结果分别送到 4 行 dotp 中作为输入 c,执行点积累加操作,得到 4 个新部分结果,写回到后 4 项 ofreg 的相同位置中去。

当一个步骤中计算 1 个通道的权值梯度总数不超过 8 时,每行 dotp 产生不超过 16 个的部分结果,能够在 ofreg 的后 4 项中放得下。因此,采用输出固定的计算顺序,如图 7 所示,将 4 行 dotp 产生的部分结果存到 ofreg 的后 4 项中。当 1 个通道的权值梯度总数超过 8 时,将 ofreg 后 4 项分成 2 份乒乓使用,一份用来存当前正在计算的输出值的部分结果;另一份负责已算好的累积结果的存取。采用这 2 种方式,对于最常见的 KW 为 1 或 3 的情况,都能够实现高效的执行。

3.3.2　复用权值与输入值的执行方式

(1)权值复用

在使用一行 dotp 执行一个子单位运算的过程中,由于 ifmap 在一项 ofreg 中能放下 32 个,于是当 S 不超过 2 时,4 项 ofreg 能够放下 $4 \times 16S$ 的所有卷积核,实现了该过程中权值在同一输入通道内的复用。另外,ofreg 中的卷积核同时与 4 个输入通道的 ierr 进行计算,实现了权值在多个输入通道间的复用。

(2)输入值行间复用

在一个子单位运算的执行过程中,相邻 2 个计算子框中有 3 行输入值是相同的。于是,和 FP、BP 中相同,只需将下一个计算子框中新增的那行输入值读到 ifreg4 中,复用 3 行相同的输入值。

(3)输入值块间复用

在单位运算内部,当一个子单位运算完成后,开始下一个子单位运算时,和 FP、BP 中相同,仅需使用 ifreg 块间特殊左移操作,将 ifreg(main)和 ifreg(right)中的 16 列输入值分别移动到 ifreg(left)和 ifreg(main)中,再读入新的 16 列输入值到 ifreg(right)中,即可开始计算,实现了输入值的块间复用。

(4)输入值无行间、块间复用时

和 FP、BP 中相同,在执行计算的同时,预取下一次计算需要的 4 行输入值到 ifreg 的后 4 项中。

总结起来,执行 WG 时,同样通过对权值和输入值的复用以及预取输入值的方式,减少了供数被阻塞的情况,保证了 WG 执行的性能。

3.3.3　跳过水平方向上的冗余操作

和 BP 时相似,利用 3 个位置计算单元,直接跳过 ierr 相邻列间插入的 0 的计算,并且,计算子框按 16 列划分后最后一块的列数小于 16 时,能够执行完有效输入列的相关运算后提前结束。于是,跳过了 WG 水平方向上的冗余计算,并且实现了对任意列数的输入值块的高效支持。

4　实验与结果

本节介绍了实验环境、实验数据,将加速器的性能与 GPU 进行了对比。

4.1　实验方法

利用 Verilog 语言在 RTL 级实现了本文提出的加速器。使用 Synopsys Design Compiler 在 ST 28 nm 工艺下进行综合。使用 Synopsys VCS 工具对加速器设计进行模拟和验证,测量其性能。

实验中对比数据使用了 NVIDIA GeForce RTX 2080 Ti 执行相同模型的训练过程,采用了 Pytorch

Profiler 来测量 GPU Kernel 的运行时间。该 GPU 的 Boost 时钟频率是 1635 MHz,有 544 个 Tensor 核心,峰值性能为 56.9 Tensor TFLOPS,内存带宽为 616 GB/s。

4.2　加速器配置

实验中加速器采用 16 行 32 列的 PU 组成运算阵列,共 16 384 个乘加单元。Fbuf _ Out 共 256 块,每块 64 kB。Fbuf _ In 共 16 块,每块 512 kB。Wbuf 共 32 块,每块 36 kB。加速器的频率为 800 MHz。峰值性能是 26.2 TFLOPS。关于 Wbuf 的访存带宽为 25.6 GB/s,关于 Fbuf _ In 和 Fbuf _ Out 的访存带宽为 205 GB/s。该配置下乘加单元数、片上缓存区大小、带宽都与主流训练加速器相近。

4.3　Benchmark

为了评估本文提出的加速器性能,采用了 3 类常见的处理图像分类任务的 CNN 模型——VGG[13]、ResNet[15]和 Wide ResNet[16]作为测试模型。具体地,本文选取了热门机器学习框架 Pytorch 中专门面向计算机视觉任务的 torchvision 包中提供的 4 个模型——VGG19、ResNet18、ResNet50 和 Wide ResNet50 作为测试用例。使用 CIFAR-10 数据集进行训练,统一将数据集中原始图片尺寸规则化为 3 ×224 ×224 后再进行训练。训练过程中 mini-batch 设定为 32。

4.4　实验和结果

使用运算资源利用率(computing resource utilization)[17-18]来度量加速器的性能。运算资源利用率 = 测试程序实际性能/加速器理论峰值性能 × 100%。其中,实际性能、峰值性能都用 FLOPS 度量。图 12 展示了本文加速器和 GPU 关于 4 个网络模型中卷积层的 FP、BP、WG 阶段的运算资源利用率以及整体的运算资源利用率。如图 12 所示,本文加速器关于 FP、BP、WG 整体阶段的平均利用率分别是 77.6%、67.3%、70.4%,分别比 GPU 的利用率高了 45.1%、41.7%、42.9%。其中,利用率最高的是 VGG19,其 FP、BP、WG 整体的利用率分别达到了 97.1%、90.2%、92.4%。

图13选取了ResNet50中19个不同的卷积层,

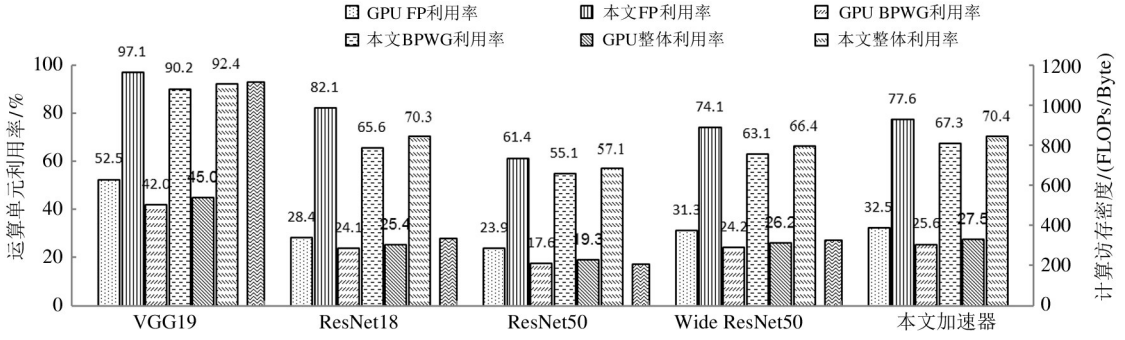


图 12　运算资源利用率比较

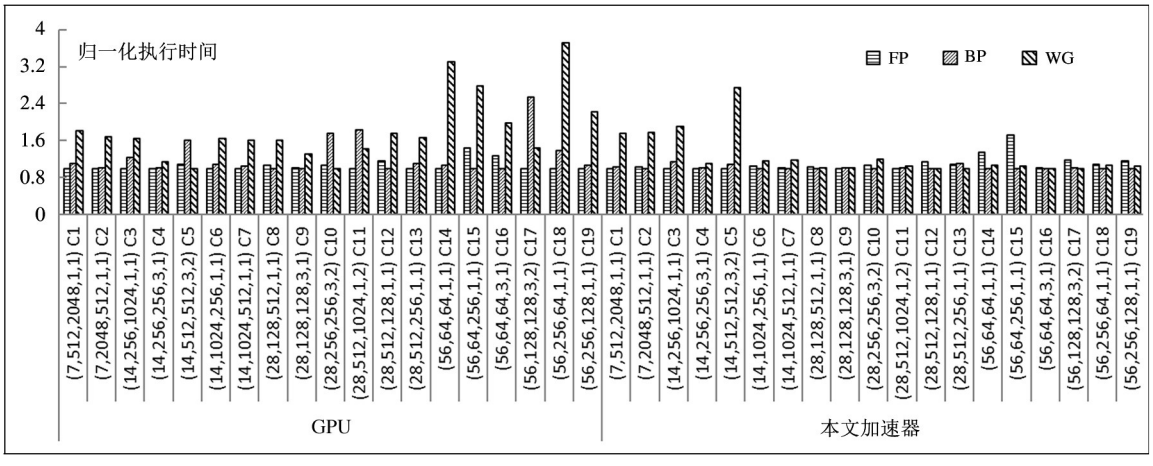


图 13　GPU 和本文加速器执行 FP、BP、WG 的时间

分别给出了 GPU 和本文加速器执行 FP、BP、WG 的时间,每层各阶段的时间都按该层 FP、BP、WG 中执行时间的最小值进行归一化。图中层序号后的 5 个值依次表示该卷积层的(IW, IC, OC, KW, S)。从图 13 中 GPU 的执行时间可以看出,除了 C4、C5、C10,其他层的 WG 执行时间都明显地更长;C5、C10、C11、C17 这 4 个步长为 2 的卷积层的 BP 执行时间是 FP 的 1.5 ~ 2.5 倍,这说明训练不同阶段的操作的计算差异性对 GPU 的执行效率产生了很大的影响。从图 13 中本文加速器执行时间可以看出,除了 C1、C2、C3、C5 的 WG 以及 C14、C15 的 FP,其他层的 FP、BP、WG 执行时间都比较接近,说明本文提出的由统一处理单元组成的加速器对训练不同阶段的操作都实现了有效的支持。

分析发现,C1、C2、C3、C5 的 WG 执行时间更长是由于 WG 阶段使用了 fp32 的梯度,而 FP、BP 中使用的权值是 fp16。因此,WG 比 FP、BP 要求更大的关于 Wbuf 的访存带宽,于是,加速器中配置的 25.6 GB/s的关于 Wbuf 的带宽对 WG 阶段带来了更大的性能限制。增加 Wbuf 的带宽就能够消除其对性能的影响。在受到访存带宽限制的情况下,C1、C2、C3、C5 的 WG 阶段的运算资源利用率分别是 24%、24%、41% 和 26%,而相应的 GPU 的运算资源利用率分别是 12%、12%、14% 和 19%,说明本文加速器仍旧实现了更高的利用率。

另外,BP 和 WG 用到了同一个输入值 ierr,FP 与之相比,关于 Fbuf _ In 和 Fbuf _ Out 的访存带宽需求更高,这使得 C14、C15 的 FP 执行出现了更多的阻塞,导致了 FP 时间更长的现象。

表 2 给出了加速器各组成部分的面积与功耗。加速器整体面积为 159 mm^2,功耗为 96 W。其中,PU 阵列的面积占比为 33%,功耗占比为 59%;片上 3 类缓存区的面积占比为 58%,功耗占比为 29%。

表 2　加速器的面积和功耗

加速器	面积/mm^2	占比/%	功耗/W	占比/%
整体	158.93	100.00	96.41	100.00
PU	52.48	33.02	56.58	58.68
Ifreg	4.78	3.01	4.52	4.69
Wreg	5.36	3.37	3.03	3.14
Fbuf _ Out	58.67	36.92	17.82	18.48
Fbuf _ In	29.33	18.46	8.91	9.24
Wbuf	4.13	2.60	1.25	1.30
Inst _ Queue	0.04	0.02	0.03	0.03
Ctrl Unit	0.40	0.25	0.30	0.31
VU	3.72	2.34	3.97	4.12

5　结　论

本文针对神经网络训练过程中前向传播、反向误差传播、梯度计算的各阶段中的不同操作,分别设计出高效的执行方案,跳过了在垂直方向上的冗余计算,提出了一个统一的加速器处理单元硬件结构。利用这个处理单元设计实现了一个面向卷积神经网络训练的加速器,能够将训练各阶段的执行方案都高效地映射到加速器上执行,并且能够利用多类数据复用机会减少对各级内存的访问以及跳过水平方向上的冗余计算。实验结果显示,基于 4 个常用的卷积神经网络模型,卷积层训练的前向过程、反向过程以及整体的运算资源利用率分别达到了 77.6%、67.3% 和 70.4%,比现有主流的 GPU 高了 45.1%、41.7% 和 42.9%。

虽然目前加速器已经达到了相当高的利用率,但是和 FP 阶段相比,BP、WG 的利用率还有提升的空间,后续工作将进一步提升 BP、WG 的性能。

参考文献

[1] NORRIE T, PATIL N, YOON D H, et al. Google's training chips revealed: TPUv2 and TPUv3[C]//2020 IEEE Hot Chips 32 Symposium. Palo Alto: IEEE, 2020: 1-70.

[2] LIAO H, TU J, XIA J, et al. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing: industry track paper[C]//2021 IEEE International Symposium on High-Performance Computer Architecture. Seoul: IEEE, 2021: 789-801.

[3] MAHMOUD M, EDO I, ZADEH A H, et al. Tensordash: exploiting sparsity to accelerate deep neural network training[C]//2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture. Athens: IEEE, 2020: 781-795.

[4] VENKATARAMANI S, SRINIVASAN V, WANG W, et al. RaPiD: AI accelerator for ultra-low precision training and inference[C]//2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture. Valencia: IEEE, 2021: 153-166.

[5] SANKARADAS M, JAKKULA V, CADAMBI S, et al. A

massively parallel coprocessor for convolutional neural networks[C]// The 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors. Boston: IEEE, 2009: 53-60.
[6] DU Z, FASTHUBER R, CHEN T, et al. ShiDianNao: shifting vision processing closer to the sensor[C]// Proceedings of the 42nd Annual International Symposium on Computer Architecture. Portland: IEEE, 2015: 92-104.
[7] CHEN Y H, EMER J, SZE V. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks[C]//2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture. Seoul: IEEE, 2016: 367-379.
[8] LU L Q, GUAN N Q, et al. TENET: a framework for modeling tensor dataflow based on relation-centric notation[C]// 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture. Valencia: IEEE, 2021: 720-733.
[9] KWON H, LAI L, PELLAUER M, et al. Heterogeneous dataflow accelerators for multi-DNN workloads[C]// 2021 IEEE International Symposium on High-Performance Computer Architecture. Seoul: IEEE, 2021: 71-83.
[10] LU W, YAN G, LI J, et al. Flexflow: a flexible dataflow accelerator architecture for convolutional neural networks[C]//2017 IEEE International Symposium on High Performance Computer Architecture. Austin: IEEE, 2017: 553-564.
[11] SONG M, ZHANG J, CHEN H, et al. Towards efficient microarchitectural design for accelerating unsupervised GAN-based deep learning[C]//2018 IEEE International Symposium on High Performance Computer Architecture. Vienna: IEEE 2018: 66-77.
[12] IOFFE S, SZEGEDY C. Batch normalization: accelerating deep network training by reducing internal covariate shift[C]// International Conference on Machine Learning. Lille: ICML, 2015: 448-456.
[13] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[C]//2015 International Conference on Learning Representations. San Diego: ICLR, 2015:1-14.
[14] TAN M, CHEN B, PANG R, et al. Mnasnet: platform-aware neural architecture search for mobile[C]// Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. Long Beach: IEEE, 2019: 2820-2828.
[15] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016: 770-778.
[16] ZAGORUYKO S, KOMODAKIS N. Wide residual networks[EB/OL]. (2016-03-23)[2022-01-20]. https://arxiv.org/pdf/1605.07146.pdf.
[17] VENKATARAMANI S, RANJAN A, BANERJEE S, et al. Scaledeep: a scalable compute architecture for learning and evaluating deep networks[C]// Proceedings of the 44th Annual International Symposium on Computer Architecture. Toronto: IEEE, 2017: 13-26.
[18] 向陶然, 叶笑春, 李文明,等. 基于细粒度数据流架构的稀疏神经网络全连接层加速[J]. 计算机研究与发展, 2019, 56(6): 1192-1024.

An accelerator for convolutional neural network training

YANG Can, WANG Chongxi, ZHANG Longbing
(* State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
(** Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
(*** University of Chinese Academy of Sciences, Beijing 100049)

Abstract

As deep neural network is widely used in many fields, the demand for speed of training neural network models is also increasing. As a result, a variety of high performance accelerators for training appears. But, the operation of the same layer shows great computation heterogeneity in different training phases, which makes the accelerator with single dataflow unable to achieve ideal efficiency when processing some training phases. General processors like graphics processing unit (GPU) are usually under low utilization because the characteristics of the operation are not fully utilized. In order to solve this problem, this paper proposes efficient execution schemes for different operations in each training phase, designs a processing unit with unified hardware structure, which can efficiently map execution schemes of each phase to it, and builds a high-performance accelerator for convolutional neural network (CNN) training using the processing unit with unified hardware structure. The experimental results show that, based on 4 commonly used convolutional neural network models, the computing resource utilization in forward propagation and backward propagation training phases reaches 77.6% and 67.3% respectively, which is 45.1% and 41.7% higher than the state-of-the-art Tensor core-based GPU.

Key words: neural network, training, accelerator, convolutional neural network(CNN)