面向灵活并行度的稀疏卷积神经网络加速器

袁海英,曾智勇,成君鹏
(北京工业大学信息学部,北京 100124)

摘 要: 大规模卷积神经网络计算复杂度高且资源开销大,这极大提高了深度学习算法的硬件部署成本. 在模型推理过程中充分利用层间稀疏激活的信息冗余,以较低资源开销和几乎无损的网络精度降低推理时延和功耗提供高效的加速器解决方案. 针对稀疏卷积神经加速器中控制粒度过大导致运算模块利用率过低问题,本文提出基于FPGA具有灵活并行度的稀疏卷积神经网络加速器架构. 基于运算簇思想对卷积运算模块实现灵活调度,根据卷积层结构在线调整输入通道和输出激活的并行度;根据输出激活并行运算的数据一致性设计了一种输入数据的并行传播方式. 本文在Xilinx VC709目标设备上实现了提出的加速器硬件架构,它包含1 024个乘累加单元,提供409.6 GOP/s理论峰值算力;实际运算速度在VGG-16模型中达到325.8 GOP/s,等效于稀疏激活优化前加速器的794.63 GOP/s,运算性能达到baseline模型4.6倍以上.

关键词: FPGA; 卷积神经网络; 硬件加速; 稀疏感知; 并行计算
中图分类号: TN47 **文献标识码:** A **文章编号:** 0372-2112(2022)08-1811-08
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20211514

A Sparsity-Aware Convolutional Neural Network Accelerator with Flexible Parallelism

YUAN Hai-ying, ZENG Zhi-yong, CHENG Jun-peng
(*Faculty of Information Technology, Beijing University of Technology, Beijing* 100124, *China*)

Abstract: Convolutional neural network involves in high computational complexity and excessive hardware resources, which greatly increases hardware deployment cost of deep learning algorithm. It is a promising scheme to make full use of the information redundancy of sparsity activation between layers can reduce the inference delay and power consumption with low resource overhead and almost lossless network accuracy. To solve low utilization problem of operation module caused by coarse-grained control in sparse convolution neural network accelerator, a sparsity-aware accelerator with flexible parallelism based on FPGA is designed. Convolution operation module is flexibly scheduled based on operation clustering idea, and the parallelism of input channel and output activation is adjusted online. In addition, a parallel propagation mode of input data is designed according to the data consistency during output activated parallel operation. The proposed hardware architecture is implemented on Xilinx VC709. It contains up to 1 024 multiplication and accumulation units and provides 409.6 GOP/s peak computing power, and the operation speed is up to 325.8 GOP/ s in VGG-16 model, which is equivalent to 794.63 GOP/s of accelerator without sparse activation optimization. Its performance is 4.6 times more than that of baseline model.

Key words: field programmable gate array; convolutional neural network; hardware accelerator; sparse perception; parallel computation

1 引言

随着深度学习技术不断突破[1],卷积神经网络(Convolutional Neural Network, CNN)成为图像识别[2]、视频跟踪、目标检测和信息处理等人工智能应用领域的主流算法之一[3]. 网络性能提升涉及到庞大的参数量与计算量,由此带来过高的资源开销和计算开销. 为了高效加速CNN模型运算过程,基于FPGA的加速器以其高能量利用率和可重构性而备受业界关注[4,5]. OPU[6]和Caffine[7]等采用的矩阵式运算单元以并行度高与数据复用能力强等优势成功应用于许多加速器中. 通过合理利用激活的稀疏性,在不降低运算精度情况下加速卷积运算过程[8,9]. 随着CNN模型的深入研

收稿日期:2021-11-12;修回日期:2022-06-08;责任编辑:李勇锋

究，许多工作意识到在卷积模型中各个卷积层的尺寸相差过大是限制加速器性能提升的原因之一[10]. 为此，本文提出了一种具有灵活并行度的稀疏CNN加速器(Sparsity-aware Accelerator with Flexible Parallelism，SAFP)，主要贡献概括为：

1. 基于FPGA设计了一种具有三个并行度参数的稀疏CNN加速器，其中两个可以在线配置，保证所有卷积层的高效运算.

2. 应用了一种output-based的并行任务划分方法和稀疏数据流，最大限度地利用了二维矩阵结构的乘累加阵列.

3. 针对输出激活的并行运算，设计了一种优化的片上数据并行传输方法，降低加速器对传输带宽的依赖.

研究表明：本文提出的加速器减少了网络碎片化影响，提升了CNN中卷积层运算效率. 加速器归一化性能达到794.63 GOP/s，远超过未进行稀疏化处理的最高运算性能.

2 研究背景

CNN通过多层卷积运算从多个抽象级别上提取图片特征[11,12]. 考虑到CNN网络的运算量主要来自卷积层[13]，因此对卷积层运算进行加速可以有效提高推理速度[14]. 卷积运算过程如图1所示.

将Chin张大小为h_in×w_in输入图像与Chout个卷积核进行卷积运算，输入特征图中卷积核的滑窗每次移动都会计算出一个输出激活像素，将输入图像映射为Chout张大小为h_out×w_out输出图像.

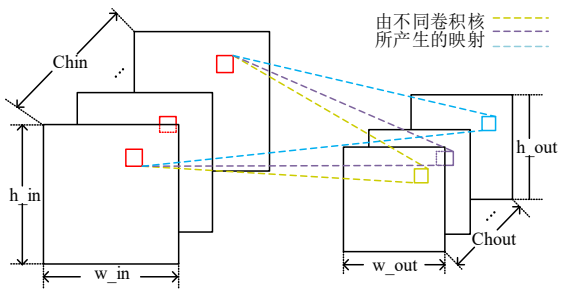


图1 卷积映射示意图

层与层之间通常采用非线性函数来避免相邻两层卷积运算融合成一次运算[15]. ReLu函数的非线性处理会带来大量0输出. 业界对其进行了一系列优化. Cnvlutin[9]可以跳过0激活所参与的运算周期来减少运算负载. 然而，这种结构在应对CNN不同卷积层运算时出现较明显的碎片化问题，从而导致具有较高并行运算资源的Cnvlutin在应对较小卷积层时运算效率低下.

本文提出的SAFP架构将一个完整的运算结构切分成多个运行独立输出激活的运算簇(cluster)，上述操作相当于增加了一个并行维度. 此外，深入探究了加速器不同并行维度之间的数据关系，由此减少卷积运算对片外通信带宽的依赖.

3 多任务划分的数据流

CNN中的数据流决定了卷积运算在硬件结构上的平铺展开方式，本文提出基于多任务划分的数据流，大大降低运算性能对卷积层尺寸的敏感性.

3.1 运算任务划分

为了实现加速器的通用性，对于卷积网络的不同层通常采用相同的数据流动方式. 这限制了运算效率提升，当输入通道较小时，会有大量输入通道的运算模块闲置. 如果在输入通道较小的卷积层中调用输入通道并行度上的运算模块去计算不同的输出激活，则能保证运算效率. 对输出激活并行运算实际上是将一个特征图卷积运算拆分成多个并行运行任务. 图2所示的w_in×h_in输入特征图被T_p个任务映射为w_out×h_out/T_p的输出特征图. 设k为卷积核大小，stride为跨步，pad为padding大小，则每一个任务依赖尺寸为w_in×((h_out/T_p-1)×stride+k-2×pad)的输入特征图，其中w_in×(k-stride)的特征图为两个任务所共享.

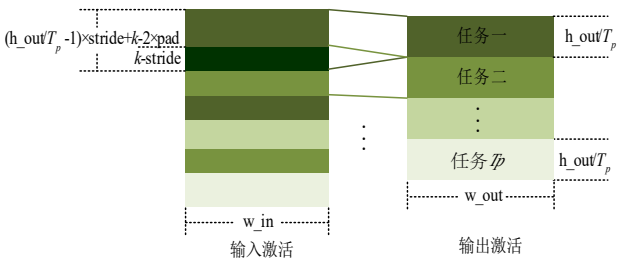


图2 任务划分

受限于硬件资源，无法使每个计算单元都拥有运行独立任务的能力. 本文将独立运行一个任务的数个运算单元(Computing Unit，CU)的集合定义为cluster，CU实现所有输出通道的并行度T_n和1个输入通道的并行度. 整个加速器具有T_m个CU. 每个cluster私有一个输出缓存用来隔离不同的任务输出. 定义Dtm=ceil(ceil(Chin/T_n)/T_m)，其中ceil()为向上取整函数.

当Dtm等于1时，T_p为1；当Dtm不等于1时，任务并行度T_p的配置算法描述在图3中. 采用该算法求出T_p值为2的指数，这种不连续的可配置参数极大地简化了加速器的控制逻辑结构.

3.2 多任务数据流

算法1实现了多任务并行运算的数据流，将运算平铺到由T_m个CU组成的T_m×T_n乘累加(Multiplication and Accumulation，MAC)阵列上. 第2层循环将整个运算分成T_p个并行任务. 由于并行任务在输出特征图上进行

```
1 for Tp=Tm/2-1 to 0 do
2 |  // Dtm = ceil(chin/Tn)/Tm
3 |  if Dtm << Tp <= Tm then
4 |  |  Tp ← 1 << Tp
5 |  |  break
6 |  end
7 end
```

图3　计算任务并行度T_p的伪代码段

划分,每个运算区域中T_m/T_p个CU运算一个任务的不同输入通道,这些运算之间互不干扰. 由于输出通道的运算相对独立,其运算只对$k*k$大小的激活和权重区域存在数据依赖,故该运算被平铺到CU中T_n个MAC上. 判断语句的外层循环中的数据流以stripe=T_n为单位流动,在这种大粒度控制情况下,激活-权重的链接不受到非结构化稀疏的影响. 判断语句生成非0激活相对于stripe的位置指针off,利用该指针寻址到对应权重位置,从而允许数据流跳过0激活.

算法1　多任务并行运算的数据流

Input: input action ia, and weight w
Output: output action oa
for $k=0$ **to** ceil$\left(\text{Chout}/T_n\right)-1$ **do**
for $p=0$ **to** T_p-1 **do**
for out = h_out × w_out × p **to** h_out × w_out$(p+1)$ **do**
sum$\left[T_n\right]\left[T_m+1\right]\leftarrow 0$
for slice = 0 **to** T_m/T_p-1 **do**
for c,r,s = dtm × slice, 0, 0 **to**
$(\text{Dtm}+1)\times\text{slice}-1, Ky-1, Kx-1$ **do**
//Dtm = ceil$\left(\text{ceil}\left(\text{Chin}/T_n\right)/T_m\right)$
h = out/h_out;
w = out/h_out;
for off = 0 **to** stripe = T_n-1 **do**
if
ia$\left[h\times s-\text{pad}+r\right]\left[w\times s-\text{pad}+s\right]\left[c\times T_m+\text{off}\right]!\ =0$
then
for $kk=0$ **to** T_n-1 **do**
sub oa$[kk]$ =ia$[h\times s-\text{pad}+r][w\times s-\text{pad}+s][c\times T_m+\text{off}]\times w[r][s]$
$[c\times T_m+\text{off}][k\times T_n+kk]$
end
end
end
for $kk=0$ **to** T_n-1 **do**
sum[kk][slice] = sum[kk][slice] + sub oa[kk]
end
end
end
end
for $kk=0$ **to** T_n-1 **do**
for $m=0$ **to** T_m-1 **do**
sum$[kk][T_m]$ = sum$[kk][T_m]$ + sum$[kk][m]$
end
oa$[h][w][k\times T_n+kk]$ = sum$[kk][T_m]$
sum$[kk][T_m]=0$
end
end

多任务划分仅影响到每个子任务的数据起点,故每个CU执行相同的数据流形式.

4　加速器硬件架构

图4为加速器架构顶层视图. 在运算时由CPU向配置寄存器写入加速器的运行参数. AXI总线将读命令发生器的请求发送到片外存储器. 数据分配器接收来自AXI总线的数据并将其分配给不同cluster进行卷积运算. Cluster的输出进入到加法树中. 加法树的输出进入数据路由模块,它根据T_p值设置成不同的输出缓存连接方式. 最后由一个仲裁器将不同的输出缓存写回片外缓存中. 为了使多任务的运算输出重新耦合成单任务的运算输入,仲裁器根据不同输出激活所属cluster的物理位置来配置输出地址.

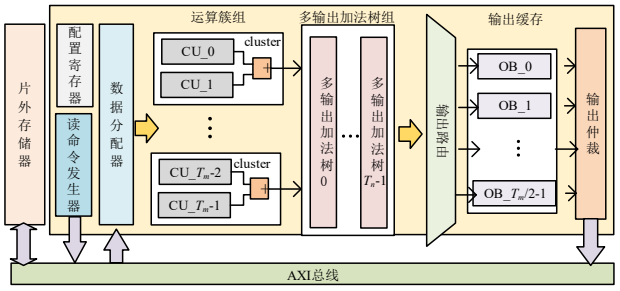


图4　加速器架构顶层视图

4.1　CU结构中的卷积运算

作为卷积运算的最小单元,图5所示CU结构完成稀疏激活的感知和运算. 稀疏激活感知器(Sparse Perception,SP)从激活缓存(Activation Buffer,AB)内提取T_n个激活中的非零值及其对应stripe的偏移值,缓存在非0缓存nz_buf中. nz_buf中的偏移值送入权重缓存(Weight Buffer,WB)进行权重寻址,得到T_n个输出通道的权重,将其与对应非0激活在T_n个MAC中进行乘累加运算.

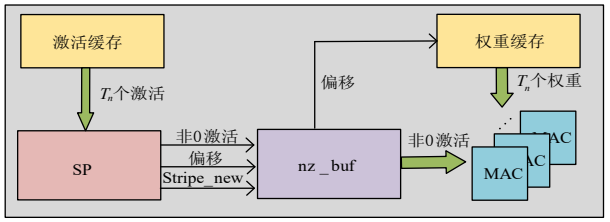


图5　CU结构

4.1.1　片上数据传输

在运行多任务时,数据的并行传输可以减少重复传输,故有必要分析片上缓存数据的一致性. 以T_p=4、T_m=8为例的片上数据存储方式如图6所示. 该例共有4个任务,每个任务被分配到T_m/T_p=2个CU中. 每个CU运行不同输入通道的卷积运算,相邻的两个CU运行一

个任务. 由于数据流根据输出特征图的高度对任务进行划分,计算相邻任务输出特征图所涉及的激活数据具有 w_in×(k-stride)个激活的数据重叠,如图6(a)中深蓝色所示. 由于多任务运行在CNN的同一个卷积层上,因此不同任务使用相同的权重数据,如图6(b)所示.

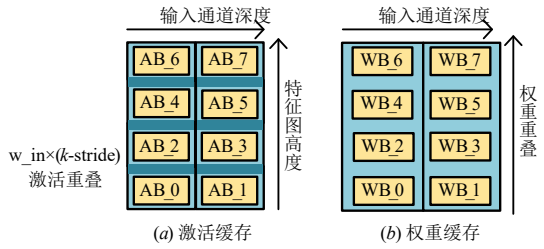


图6 片上缓存数据存储方式

根据数据重叠情况进行并行传输可以减少片外存储器的访问. 根据T_p值在数据分配器中使用一组T_m位的队列对数据一致性进行标记,队列中每一位对应着一个CU的缓存,仅对1比特位所对应的缓存数据进行并行传输. 权重传输的起始队列值为$\frac{2^{T_m}-1}{2^{\frac{T_m}{T_p}}-1}$. 在需要传输不同输入通道的数据时仅需将队列左移一位以选择不同的缓存. 特殊地,当T_p=1时起始队列值为"00000001",这意味着没有权重进行并行传输.

由于激活数据一致性较低,故大多数时候采用顺序传输. 不同任务之间存在w_in×(k-stride)的激活重叠,这体现在不同cluster中传输这些重叠的激活数据时仍然采用并行传输形式. 在传输非重叠数据时,激活传输的队列值为"00000001";在传输重叠数据时,传输队列值为$\frac{2^{\frac{2T_m}{T_p}}-1}{2^{\frac{T_m}{T_p}}-1}$. 在传输不同任务所对应的激活数据时,将队列左移$T_m/T_p$位;在传输同一任务的不同输入通道所对应的激活数据时将队列左移1位. 这种数据传输方式既节省了片上传输时间,又减少了片外数据传输的带宽依赖.

4.1.2 稀疏激活运算

稀疏神经网络加速器在推理过程中将训练时构造的激活-权重密集连接模式打乱,在CU中重新构造激活-权重的稀疏连接模式. 在运算时,激活和权重均以stripe=T_n为粒度进行寻址. 激活每次寻址T_n个输入通道上的数据,将这些数据输入SP中,提取出非0值及其在stripe中的偏移值offset. 当一个stripe中的数据全为0时,SP仍输出一个0以便对该stripe进行占位. 偏移值用来进行权重的细粒度寻址,在权重stripe位置为c情况下,其细粒度寻址位置为c×T_n+offset,该地址索引到T_n个输出通道的权重上. 一个非零激活和T_n个权重进入T_n个MAC所组成的运算行中,实现T_n个输出通道的并行运算.

4.2 多输出加法树

由于多任务需要输出多个输出激活结果,并且一个任务只耦合部分cluster输出,因此需要设计一种多输出的加法树结构. 多输出加法树组中具有T_n个加法树,每个加法树具有T_m个输入,输入为每个CU中的一个MAC,加法器的输出连接到输出路由逻辑. 根据输出激活的并行度,多输出加法树上各个加法器的输出配置为单个任务的最终输出,也可以配置为下一级加法树的输入. 设加法树中最远离输入端的一级加法器为第0级,则将第$\log_2 T_p$级的所有加法器配置为输出的加法器,并将上一级的加法器全部关闭. 加法树中同一级加法器相隔离的特性保证了多任务运行互不干扰.

图7中,同一矩形所对应的cluster属于同一运算路径,相同运算路径中的数据经过加法树相互耦合. 每条运算路径对应一个(组)的输出缓存,根据运算路径不同,由输出数据路由模块将加法树输出的数据路由到对应输出缓存中.

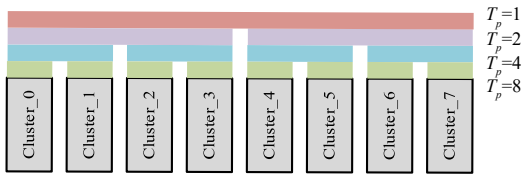


图7 不同T_p情况下的运算路径

4.3 输出缓存匹配

为了尽可能地提高输出激活并行度,给每个cluster设置一个输出缓存(Output Buffer,OB). 每个缓存由深度为256的FIFO构成,由此匹配AXI总线的突发长度,充分利用传输带宽. 然而,由于输出激活并行度的可配置性,这些缓存无法与每个任务一一对应,此时需要设计数据路由逻辑,为每个任务分配均衡的输出缓存. 输出数据路由模块负责将输出数据引导到对应输出缓存中,具有8个cluster的硬件结构如图8所示.

输出数据路由模块由8个数据选择器组成,支持最多8任务的数据路由. AL_a_b代表来自第a级加法树的第b个加法器的输出. 偶数标号的数据选择器通常为不同任务的分界点,故它们比奇数标号的数据选择器拥有更多数据输入端. 除了MUX_0,其它MUX均有来自上一级输出缓存的输入端口. 由此链接成一个容量较大的输出缓存. 当T_p=2时,8个输出缓存最终仅有OB_3和OB_7两个数据出口,这样每个任务具有4×256的输出缓存,如图9所示.

当激活的稀疏度较低时,各个输出缓存的数据存

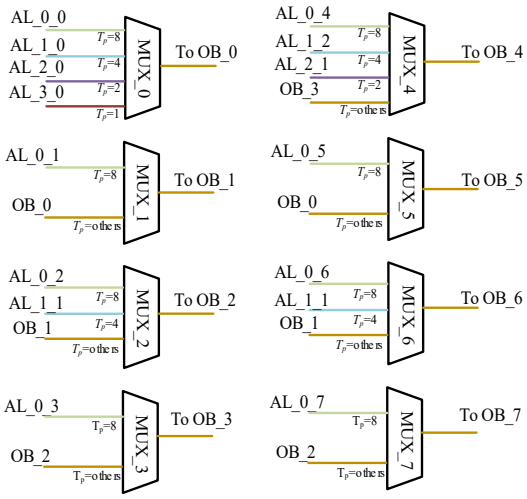


图 8　输出数据路由模块

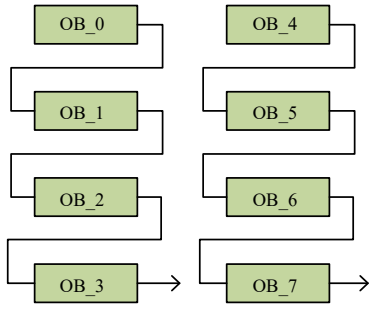


图 9　T_p=2 时输出缓存的链接关系

入速度相仿,由此造成大量输出缓存争抢输出带宽. 采用图 9 的输出缓存链接方式可以有效地扩充输出缓存深度,减缓运算对输出带宽的压力.

输出仲裁器将数据通过 AXI 总线写入片外存储器时,需要将多个任务的输出结果重新耦合成一个完整的输出结果. 其实现方式是在外部存储器上为每个任务划分大小为 h_out×w_out×Chout/T_p 地址区域并前后紧密相接,这样下一层的卷积运算可以将上一层的输出结果看作一个完整的特征图输入.

5　实验与分析

实验评估工程建立在 Xilinx VC709 的 FPGA 平台上,采用 Xilinx Vivado(v2019.1)工具进行逻辑综合并实现了 RTL 代码. 加速器所有性能仿真均使用实现时序仿真来估计运算性能;在运行时序仿真时使用 saif 文件记录下加速器大部分节点的翻转率信息,将该信息导入 Vivado 的功耗评估工具,以尽可能准确地评估功耗性能.

5.1　资源消耗报告

该加速器采用模块化设计. 各个 cluster 结构相同且 cluster 数量可定义为宏,根据 FPGA 的实际硬件资源进行灵活配置. 该加速器运行在 0.2 GHz 工作频率上,部署参数设置为 T_m=64,T_n=16,1 024 个乘法器,其理论运算速度高达 2×0.2 GHz×64×16=409.6 GOP/s. FPGA 硬件实现后的资源利用率统计在表 1 中,BRAM 用于构造激活、权重和输出缓存,DSP 用于实现 MAC 阵列.

表 1　资源消耗

类型	LUT	FF	BRAM	DSP
使用量	322 745	537 191	768	1 089
利用率	74.5%	62.00%	52.00%	30.25%

考虑到过度利用 BRAM 会增加布线难度,故加速器仅使用 52% 的 BRAM 资源. 每个 MAC 均采用 DSP 实现,其中 1 024 个 DSP 用于运算,其余 DSP 用于寻址.

5.2　综合性能评估

为了综合评估多任务策略对该加速器性能的提升效果,建立一个与其总体架构相同的 baseline 模型. 该模型仅运行在单任务模式下,其余行为均与该加速器一样,以模拟未使用多任务策略的加速器性能.

AlexNet 是一个碎片化问题十分严重的 CNN 模型,它在很少卷积层中跨越了过大的输入通道数量. SAFP 在各个卷积层的运行时间占比如图 10 所示;每个卷积层运行时间较为平均,这意味着 SAFP 对于运算碎片化并不敏感. 作为对比,在 baseline 中,仅第 1 层的运行时间就超过了 90%.

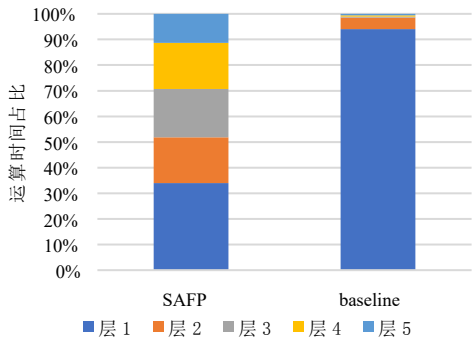


图 10　在 AlexNet 中各个卷积层的运算时间占比

为了反映稀疏加速器的实际加速能力,需要根据公式 $\text{performance}_{\text{normalizationt}} = \dfrac{\text{performance}_{\text{real}}}{\text{density}}$ 将稀疏加速器的实际性能转化为归一化性能.

VGG-16 各个卷积层在 SAFP 和 baseline 模型上的运算性能对比如图 11 所示. 得益于高效的并行度拓展策略,SAFP 的平均归一化运算性能达到 794.63 GOP/s,是 baseline 的 4.6 倍. 值得注意的是,其第 1 层的运算速度甚至达到 baseline 的 30 倍以上.

VGG-16 的计算规模不足以将卷积运算的输入通道平铺在所有计算单元上,图 12 的 baseline 模型中各个

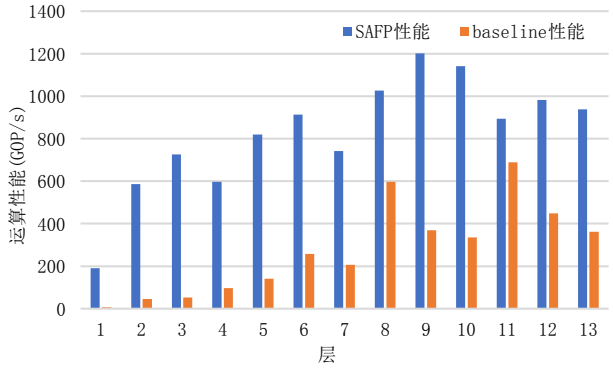


图11 在VGG-16中不同层的运算性能

卷积层的最高利用率均不超过50%,这导致超过一半的运算资源被闲置. 就VGG-16模型而言,无论baseline的运算资源如何增加,其运算性能均不会有所提升. 应用了多任务的SAFP除了第1层,其余层的运算资源利用率均达到100%.

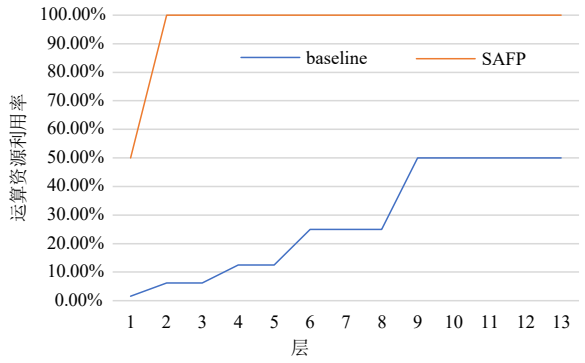


图12 在VGG-16中不同层的资源利用率

然而,任务量T_p提升在增加算力的同时也增加了运算模块对片外带宽的依赖. 针对运算任务进行划分时,需要考虑到计算与访存之间的关系. 计算访存的屋顶(Roofline)模型如图13所示,用于分析VGG-16各卷积层计算与访存之间的关系,以定制化地确定图3算法中的T_p值.

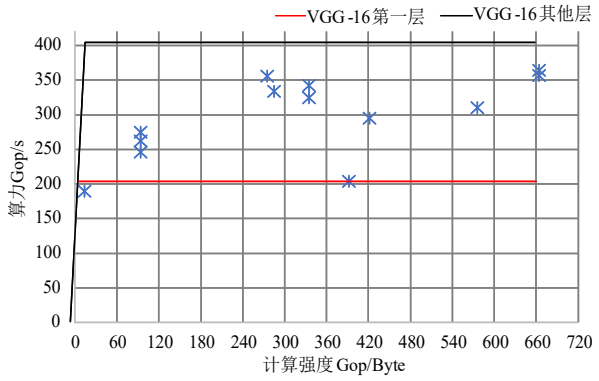


图13 本文加速器的Roofline模型

其中,黑色折线为本加速器Roofline模型的“屋顶”和“屋檐”,折线拐点的左右两边分别为带宽瓶颈区和计算瓶颈区. 可见,VGG-16绝大部分卷积层的运算结果均落到了计算瓶颈区域,仅有第一个卷积层(具有32个任务并行度)的运算结果落到了带宽瓶颈区. 图13显示该层运算只利用到了一半运算资源,因此运算第一个卷积层的Roofline模型的“屋顶”实际应为红线部分,此时,第一个卷积层的运算速度十分接近算力上限,故带宽对加速器性能的影响极为有限. 此外,任务并行度T_p的计算方法不会带来加速器的额外带宽压力.

5.3 性能分析

表2对比了本文加速器SAFP与部分FPGA加速器的运算性能和功耗. 根据式(1),SAFP模型归一化性能达到794.63 GOP/s,远超过许多没有针对稀疏激活优化的设计[10]. LACS[16]没有实现较大的并行度,这直接限制了加速器的性能发挥;SAFP实现了其四倍的并行度,因此SAFP性能比其高出很多. 这也证明了SAFP可以在具有较大逻辑资源的FPGA上取得较好的加速效果. 文献[17]对于不同尺寸的卷积核运算需要进行分割或补全的操作,这一操作会减少加速器的运算效率,因此SAFP以更少的DSP资源实现了更高的运算性能. OMNI[18]必需将权值分为几组并强制每个组具有相同

表2 加速器性能对比

Design	文献[10]	LACS[16]	文献[17]	OMNI[18]	本文	
Precision	Fixed-16	Fixed-8	Fixed-16	Fixed-16	Fixed-16	
Density	—	—	23.5%	12%	41%	39%
Device	Arria-10GX 1150	XC7Z030	ZCU102	ZCU102	VC709	
CNN Module	VGG-16	VGG-16	VGG-16	VGG-16	VGG-16	VGG-19
Frequency(MHz)	150	250	200	200	200	
DSP Utilization	1 518	256	1 144	—	1 089	
Power(W)	—	—	23.6	23.6	17.4	
Performance (GOP/s)	645.25	127.5	309.0	127.1	325.8	332.7

数量的非0数据,大幅度降低了加速器的稀疏敏感度与灵活性,并且该加速器需要额外的软件支持来进行权重的模块化剪枝. 基于上述分析,在VGG-16测试集中,SAFP加速器实现了更高的运算性能.

虽然SAFP将运行多任务所需的重叠数据进行广播传输,由于网络层数越深,通道数越多,导致任务数量变少,其重叠数据大幅减少,由此削弱了广播传输的优势.

6 结论

针对稀疏卷积神经网络加速器的配置灵活性较低问题,本文提出了一种可在线配置并行度的硬件架构,它执行了一个面向单运算多任务并行的稀疏激活感知数据流. 基于该数据流设计了一个具有多个cluster的硬件加速器,cluster内部的稀疏感知模块可以高效地过滤0激活;每个cluster根据输入通道数量被分配到不同任务的运算上,从而令加速器的运算更趋于平衡;深入挖掘不同cluster之间的数据一致性,采用数据分配器并行传输具有一致性的数据,减少了加速器对片外传输带宽的严重依赖. 该加速器架构在FPGA平台上硬件实现并进行充分验证和综合评估,实验结果表明其运算性能高达325.8 GOP/s,平均功耗为17.4 W,达到比大多稀疏感知加速器更高的综合性能.

参考文献

[1] BAI L, LYU Y, HUANG X. RoadNet-RT: High throughput CNN architecture and SoC design for real-time road segmentation[J]. IEEE Transactions on Circuits and Systems I, 2021, 68(2): 704-714.

[2] KRIZHEVSKY A, SUTSKEVER I, HINTON G. ImageNet classification with deep convolutional neural networks [J]. Advances in Neural Information Processing Systems, 2012, 25(2): 1097-1105.

[3] HE K M, ZHANG X Y, REN S Q, et al. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification[C]//IEEE International Conference on Computer Vision. Santiago: IEEE, 2015: 1026-1034.

[4] 刘杰, 葛一凡, 田明, 马力强. 基于ZYNQ的可重构卷积神经网络加速器[J]. 电子学报, 2021, 49(4): 729-735.
LIU Jie, GE Yi-fan, TIAN Ming, MA Li-qiang. Reconfigurable convolutional network accelerator based on ZYNQ [J]. Acta Electronica Sinica, 2021, 49(4): 729-735. (in Chinese)

[5] LIANG S, YIN S, LIU L, et al. Acoarse-grained reconfigurable architecture for compute-intensive mapreduce acceleration[J]. IEEE Computer Architecture Letters, 2016, 15 (2): 69-72.

[6] YU Y, WU C, ZHAO T, et al.OPU: An FPGA-based overlay processor for convolutional neural networks[J]. IEEE Transactions on Very Large-Scale Integration(VLSI) Systems, 2020, 28 (1): 35-47.

[7] ZHANG C, ZHENMAN F, PEIPEI Z, et al. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks[C]//IEEE/ACM International Conference on Computer-Aided Design(ICCAD). Austin: IEEE, 2016: 1-8.

[8] GUO J, YIN S, OUYANG P, et al. Bit-width based resource partitioning for CNN acceleration on FPGA[C]// IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines(FCCM). Napa: IEEE, 2017: 31-31.

[9] ALBERICIO J, JUDD P, HETHERINGTON T, et al. Cnvlutin: Ineffectual-neuron-freedeep neural network computing[C]//IEEE 43th International Symposium on Computer Architecture. Seoul: IEEE, 2016: 1-13.

[10] MA Y, CAOY, VRUDHULA S, et al. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks[C]//ACM/Sigda International Symposium on Field-programmable Gate Arrays. Monterey: ACM, 2017:45-54.

[11] LEE H, GROSSE R, RANGANATH R, et al. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations[C]//26th International Conference on Machine Learning. Montreal: ACM, 2009: 609-616.

[12] KARPATHY A, TODERICI G, SHETTY S, et al. Large-scale video classification with convolutional neural networks[C]//Computer Vision & Pattern Recognition. Columbus: IEEE, 2014: 1725-1732.

[13] YU D, DENG L. Deep learning and its applications to signal and information processing[J]. IEEE Signal Processing Magazine,2011, 28 (1): 145-154.

[14] CONG J, XIAO B. Minimizing computation in convolutional neural networks[C]//International Conference on Artificial Neural Networks. Hamburg: Springer, Cham, 2014: 281-290.

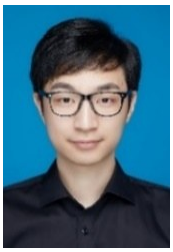
[15] LI Y, MA S, GUO Y, et al. Configurable CNN accelerator based on tiling dataflow[C]//2018 IEEE 9th International Conference on Software Engineering and Service Science(ICSESS). Beijing: IEEE, 2018: 309-313.

[16] SHANG J W, QIAN L, ZHANG Z, et al.LACS: A high-computational-efficiency accelerator for CNNs[J]. IEEE

Access, 2020, 8: 6045-6059.
[17] ZHU C, HUANG K, YANG S, et al. Anefficient hardware accelerator for structured sparse convolutional neural networks on FPGAs[J]. IEEE Transactions on Very Large-Scale Integration(VLSI) Systems, 2020, 28 (9): 1953-1965.
[18] LIANG Y, LU L Q, XIE J M. OMNI: A framework for integrating hardware and software optimizations for sparse CNNs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021, 40 (8): 1648-1661.

作者简介

袁海英 女,博士,1976年出生于四川阆中,现为北京工业大学信息学部副教授,主要研究方向为面向人工智应用的高能效计算系统、微弱信号检测与智能信息处理、电子系统容错与通信总线技术.
E-mail: yhycn@126.com



曾智勇 男,1997年出生于北京. 现为北京工业大学信息学部硕士研究生. 主要研究方向为基于FPGA的卷积神经网络加速器.
E-mail: m_x_zy@126.com

成君鹏 男,1995年出生于江苏盐城. 现为北京工业大学信息学部硕士研究生. 主要研究方向为轻量级卷积神经网络.
E-mail: chengjp@emails.bjut.edu.cn