

引用格式: 谢帅, 蒋力, 叶瑶瑶. 针对实时目标检测的多维度并行 FPGA 加速器设计[J]. 微电子学与计算机, 2021, 38(8): 13-19. [XIE S, JIANG L, YE Y Y. Multidimensional parallel FPGA accelerator design for real-time object detection[J]. Microelectronics & Computer, 2021, 38(8): 13-19.]

针对实时目标检测的多维度并行 FPGA 加速器设计

谢 帅¹, 蒋 力², 叶瑶瑶¹

(1 上海交通大学 微纳电子学系, 上海 200240;

2 上海交通大学 计算机科学与工程系, 上海 200240)

摘 要: 目标检测任务对于检测任务精度和实时性都有很高要求, YOLOv3-tiny 网络在这两点有很好的表现. 但是其复杂的网络结构, 使得实际应用需要从软件和硬件方面都进行针对性的优化. 为了达到实时要求, 综合使用三种优化技术: 在软件层面, 通过融合批归一层降低计算量, 低位宽增大资源利用率; 设计多维度并行 FPGA 计算核心匹配多个卷积层, 提高整体吞吐率; 细粒度层间流水和 pingpong 缓存设计, 降低数据传输时间. 在 ZCU104 型号的 FPGA 上, 实现了 418×418 图片的 21ms 检测延时, 超过同类加速器设计, 并在 DSP 效率上有 2.86 倍或者 8.81 倍的提升.

关键词: YOLOv3-tiny; FPGA 加速器; 多维度并行; 低延时; 高 DSP 效率

中图分类号: TP183

文献标识码: A

文章编号: 1000-7180(2021)08-0013-07

Multidimensional parallel FPGA accelerator design for real-time object detection

XIE Shuai¹, JIANG Li², YE Yaoyao¹

(1 Department of Micro/Nano Electronics, Shanghai Jiao Tong University, Shanghai 200240, China;

2 Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: The YOLOv3-tiny network performs well in both accuracy and real-time for object detection. However, its complex network structure makes practical applications require targeted optimization from both software and hardware aspects. In order to meet the real-time requirements, three optimization techniques are used comprehensively. At the software level, the amount of computation is reduced through the fusion of batch normalization layer, while the low bit width to increase resource utilization. The multi-dimensional parallel FPGA computation cores are designed to match multiple convolutional layers to improve the overall throughput. Fine-grained inter-layer flow and pingpong buffer design to reduce the data transfer time. With the ZCU104 model FPGA, it achieves a detection latency of 21ms for 418 x 418 images, which exceeds similar accelerator designs and improves the DSP efficiency by 2.86 times or 8.81 times.

Key words: YOLOv3 tiny; FPGA accelerator; multidimensional parallel; low delay; high DSP efficiency

1 引言

近些年以来, 卷积神经网络 (Convolutional Neural Network, CNN) 被广泛应用在许多与图像

相关的机器学习任务中. 在目标检测领域, 一些神经网络模型已经取得了较好的检测效果. 但是, 其复杂的网络结构以及巨大的计算量和通信量, 伴随着实际场景下的实时性要求, 制约着神经模型的应用普

及,传统的计算架构(如 CPU)远不能适应这种算法计算. GPU 作为一种较为普遍的神经网络计算平台,也存在着能耗较高,利用率不足的缺点,通常只会用在服务器端.而在广大的推理应用场景下,采用合适的硬件平台进行加速十分必要.

现场可编程门阵列(Field Programmable Gate Array, FPGA)由于其能耗和灵活的可编程性的优势,日益成为流行的加速器平台.利用 FPGA 来进行神经网络的加速实现,现有的研究主要分为两方面.一方面是从网络模型出发,可以通过裁枝算法降低网络的计算量和通信量,也可以如文献[1-3],通过对数据进行量化,提高硬件计算资源的利用效率.另一方面是从硬件着手,设计高效的加速器架构.文献[4]设定了基本的加速器架构,即设计输入通道以及输出通道进行并行计算,因为加速器架构相对简单,可以用基本的公式作为分析模型,来预估周期.然后遍历设计参数空间,根据 roofline 模型^[5]来选择吞吐率和带宽较为匹配的设计方案,但是从效果来看,其设计的吞吐率并不是很高.文献[6]和文献[7]都采用了列式排布的缓存方式,有效地降低了片上缓存的需求,提高了硬件资源的利用效率,但是存在无法处理具有分支结构的网络的问题.在其两者的实现中,都是针对 YOLOv2^[8]这一模型进行设计. YOLOv2 的模型恰好是线性的结构,而检测精度远远超过 YOLOv2 的 YOLOv3^[9]具有多分支的网络结构,该加速器架构就不合适了.文献[10]从循环顺序入手,探索加速器的设计方案,但是他们的探索主要集中在最小化数据传输开销上面,而不是最大化 CNN 加速器的性能上.

本文针对 YOLOv3 tiny 模型,采用了软件和硬件联合加速的方案,利用层融合、量化降低网络模型大小,设计多个维度的并行度;同时充分利用卷积公式的可复用性,设计创新的计算核心,可以同时复用 3×3 以及 1×1 的卷积核;最后通过合理的缓存设计,巧妙地降低了片上 buffer 的需求,提高了硬件资源利用率.本文提出的加速器基于 ZCU104,对于 8 bit 计算,可以达到 263.8 GOP/s,吞吐率达到每秒处理 47.6 张图片,能够满足高性能的实时计算要求.

2 目标检测网络以及优化

2.1 目标检测网络

目标检测任务,是指检测数字图像中物体的位置、类别以及置信度等属性.因此,目标检测对于一

些同时需要定位以及分类的场景就十分具有吸引力,例如自动驾驶、军事侦察等.当 AlexNet^[11]在 ImageNet 夺冠之后,利用卷积神经网络进行目标检测的研究涌现.其中,主要有两类方法,一种 multi-stage 的网络,这种方法通过先生成候选框,然后对候选框进行分类回归预测,例如 RCNN 和它的扩展 FastRCNN 和 FasterRCNN.另外一种是一-stage 的网络,这种方法是直接生成坐标回归,例如 YOLO, SSD, RetinaNet.通常而言, multi-stage 的网络性能较好,准确度较高,但是相对耗时,而 one-stage 的网络由于省略了候选框的步骤,因此速度较快,在一些实时性要求较高的领域得到了普遍应用.

但是 one-stage 的网络由于直接进行目标定位和分类预测两个任务,进行检测的候选框是固定的,使得容易出现检测到背景框或者忽略小尺寸目标的情况存在,从而降低网络精度.针对这一问题, YOLOv2 通过批归一化(Batch Normalization)^[12]、高分辨率分类器、多尺度训练等技术,在检测速度和检测精度上达到了较好的平衡. YOLOv3 在 YOLOv2 的基础上,通过逻辑回归、多尺度预测坐标来进一步提升精度.为了达到高精度下的实时性要求,选取了 YOLOv3 的精简版本 YOLOv3-tiny 模型进行任务的执行. YOLOv3-tiny 的网络模型更小,意味着计算量和数据存储量的需求降低,更加有利于硬件的设计实现.

2.2 网络的压缩

很多深度卷积网络模型,在训练的时候往往会经过 BN 层,来避免梯度消失的现象,缩短训练时间,同时也能提高模型的泛化能力.但是当用于实际任务中,网络参数不需要改变, BN 层的效果可以由 1×1 的卷积进行卷积替换,我们通常会进一步地将 BN 层合并到前面的卷积层中,进而降低运算量,并且不会降低推理的精度.

假设训练过程中,某个 batch 有 n 个样本,其特征分别为 $x_1, x_2, \dots, x_n$, 归一化公式如下所示:

$$x_i = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta = \frac{\gamma x_i}{\sqrt{\sigma^2 + \epsilon}} + \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \epsilon}} \quad (1)$$

式中, μ 和 σ^2 分别为这个 batch 的均值和方差; γ 和 β 分别是缩放系数和平移系数; ϵ 为防止分母为 0 而设置的一个极小值. 公式(1)的等式的最右边,实际是一个 $W * x + b$ 的形式. 由于 $\mu, \sigma, \gamma, \beta, \epsilon$ 在推理过程中,都是常数,因此上一层的特征图经过 BN 层之后,实际也就是一个 1×1 的卷积操作. 而这个

1×1 的卷积操作可以进一步的融合进前一层的 Conv 层,并且没有精度损失.

文献[13]的实验结果表明 8 bit 定点化基本不会影响模型的精度,并且可以大幅度提升速度. 尽管将数据位宽降低并不会降低模型的计算次数,但是一方面,位宽的下降可以降低 FPGA 片上存储空间的消耗,另一方面,也可以提高单个 DSP 进行的操作个数,从而提升算力. 因此,在本文的实现方案中,将数据位宽定为 8 比特.

3 加速器架构设计

3.1 系统架构设计

本文提出了一种 CPU+FPGA 的异构架构来执行图像任务的加速. CPU 部分负责任务的启动、外部存储的管理以及网络执行后的数据处理. 实际的图像任务在异构系统中进行执行分为三部分,如下:

1)数据准备. 在这一阶段,CPU 负责将预先存放在其他存储介质的数据或者通过相机采集到的数据载入到 DDR 中. 这些数据包括输入的图片数据以及权值数据. 当网络中需要累加偏置数据时,还需要存放偏置.

2)FPGA 加速. 当数据准备完毕,CPU host 会启动 FPGA 上面的加速任务. FPGA 通过 DMA 将 DDR 中的数据读取到相应的 buffer 中去,然后开始 CNN 的加速计算.

3)结果处理. FPGA 完成 CNN 的所有操作之后,将网络计算出的结果写回 DDR. CPU host 通过读取 DDR 相应位置的数据得到 FPGA 加速之后的结果,通过根据任务或者网络对这些结果进行进一步的处理. 例如将图片中检测到的物体的位置显示出来.

图 1 显示了整个系统中 FPGA 部分的架构. 这部分由 FPGA 的 PL 实现,主要包含 PE array、Buffer、Control Logic、以及 DMA. PE array 完成 CNN 中的大部分计算任务,包括 CONV layers、Pooling layers、以及 FC layers. Buffer 分为 activation buffer,即 Buffer0 和 Buffer1,和 Weight Buffer. Bias 由于数据量较小,在图中省略了. Control Logic 负责每个卷积层如何读取片上 Buffer 的数据到 PE 阵列中进行加速,控制 PE array 加速然后将数据写到输出 Buffer 中去. DMA 将位于外部存储中的数据传送到 PL 一侧的片上 Buffer 中去.

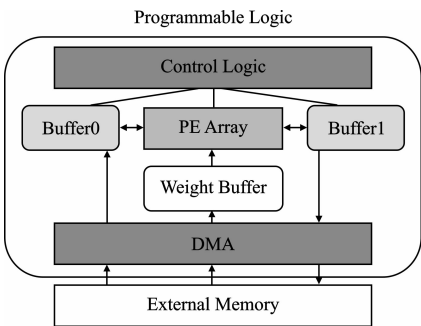


图 1 系统架构设置

3.2 计算核心设计

从图 2 中我们可以看到计算核心架构的三个部分:卷积模块、累加模块、池化模块.

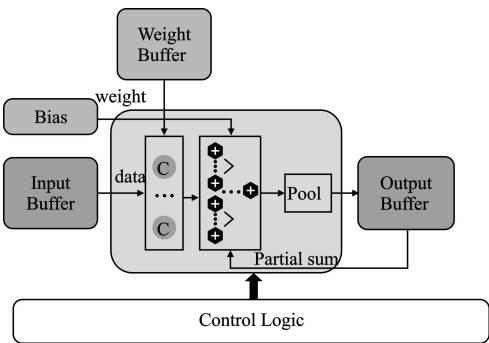


图 2 计算核心设计

卷积模块是最核心的部分,负责卷积计算,如何设计将很大程度上影响加速器的整体性能. 实际任务中采用的 YOLOv3 tiny 模型结构如图 3 所示. 为了提高算力,输入通道并行度和输出通道并行度设为 8,卷积核 9 个 weight 都并行计算,输出 neuron 列方向并行度为 2,意味着可以同时输出列方向上 2 个 output neuron. 整体所需要的 DSP 个数为 576.

目前最新的一些网络的卷积核日益小型化,这个网络中主要的卷积核类型是 3×3,因此我们针对 3×3 卷积核设计的模块结构如图 4 所示. 输出 neuron 的列方向并行度设为 2,同时可以匹配下一层的 MaxPooling 层的步长,使得层间细粒度流水设计简单. 由于对于每个输入输出通道,计算都类似. 下面以一张输入特征图卷积一个 weight window 为例讲解卷积过程. 由于需要同时输出两个 output neuron,我们需要得到两个 Data window 进行卷积. 因为后一个 Data window 的前两列和当前计算的 Data block 的后两列是重叠的,因此设计了一个四列的 Data block 来存放当前卷积所需要两个 Data window 的数据. 同时这个 Data block 可以缓存后面两个 Data window 所需要的数据,不过后两个 Data window 还需要两列新的数据,这可以从两个行缓存和

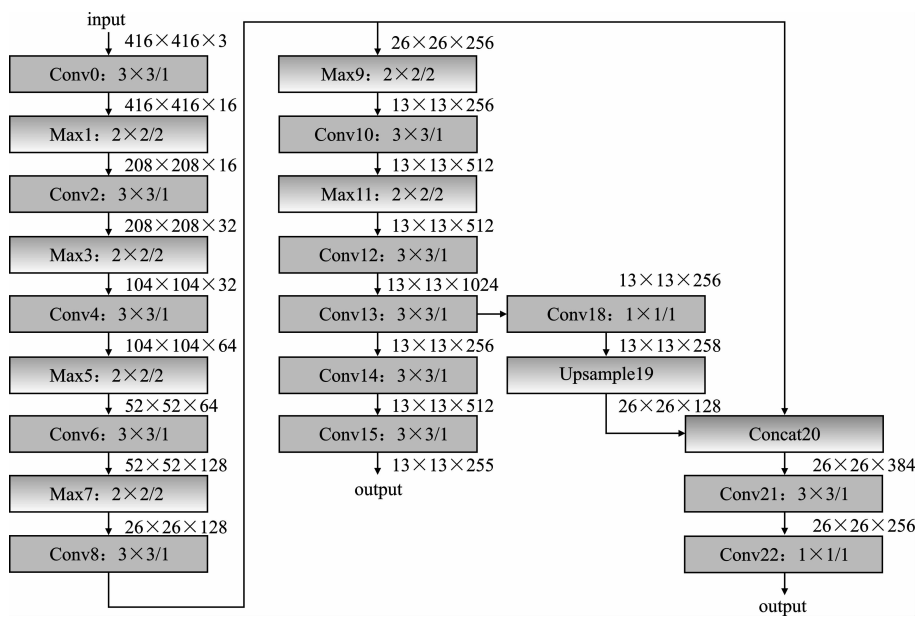


图 3 YOLOv3-tiny 模型

上一层的特征图中读取. 计算过程如下: 首先将当前 Data block 的左边两列数据右移 2 个单位, 如图 4 中 shift 箭头线所示, 1、4、7 和 2、5、8 位置的数据分别覆盖 3、6、9 和 a、b、c 位置的数据. 然后将行缓存里的 2×2 个数据以及当前从 Input Buffer 中读取的数据 2 个数据组成两列, 送到 Data block, 更新 Data block 中 2、5、8 和 1、4、7 位置的数据, 如图 4 中 load 箭头线所示. 最后, 将 input data 和行缓存的第二行数据分别更新行缓存的第二行数据和第一行数据, 如图 4 中 update 箭头线所示.

这样通过两个行缓存以及 Data block, 我们可以

充分利用卷积公式的可复用性, 当输入 2 个 input data 时, 我们可以完成 2×9 个计算量. 2 表示两个 window, 9 表示一个 window 里面的九个乘法. 当 Weight window 和 Data block 的数据经过乘法器阵列完成乘法之后, 进入加法树进行进一步计算. 累加模块将乘法器阵列得到的乘法结果进行累加, 同时如果需要加上偏置数据, 偏置数据也会作为输入传入累加模块. 当需要多次累加才能得到最终的输出时, 来自 Output Buffer 的 partial sum 也会作为输入进行累加. 如果当前卷积层为 1×1 的类型, 那么直接读取 2 个 weight 和 2 个 input data 进行卷积操作即可.

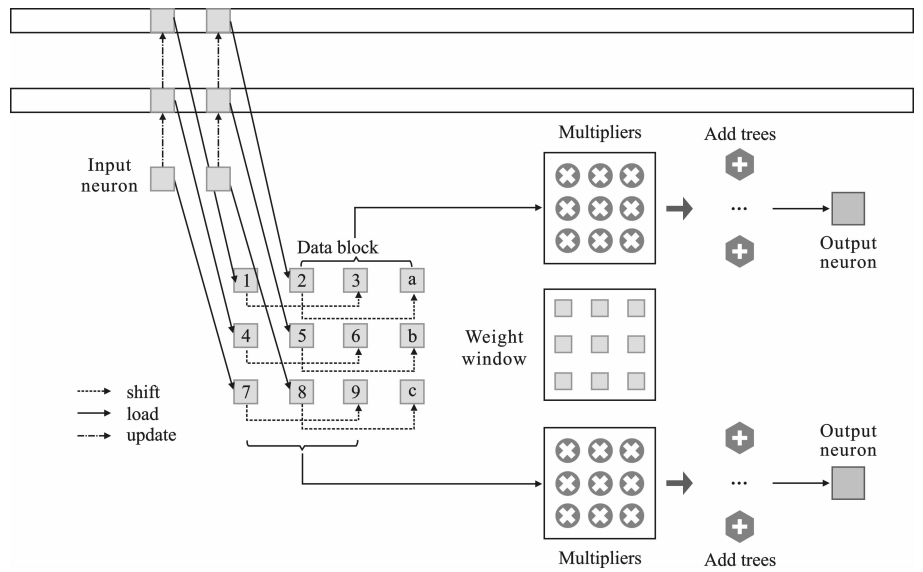


图 4 卷积模块设计

可以看到,这种多维度并行的卷积模块架构可以充分利用卷积公式的可复用性. 根据文献[14],卷积公式主要有三种类型的数据并行方式. 此卷积模块针对三种类型数据都进行了数据并行:

神经元并行. 通过 Data block 缓存 4 列输入 neuron,可以支持同时计算 2 个输出神经元值.

特征图并行. 由于设置了输入通道和输出通道的并行度都为 8,相当于同时计算 8 个输入特征图和 8 个输出特征图的数据.

突触并行. 卷积核的 9 个 weight 都进行了并行计算.

这样,通过三种方式的并行计算,可以适应不同类型的卷积层,从而提高整体的吞吐率.

3.3 缓存设计与细粒度层间流水

为了尽可能降低数据传输的时间以及降低 DDR 访问带来的功耗,如图 1 所示,在本文的架构中,Data Buffer 设定为能够存放下一层的数据. 这意味着 Buffer0 的大小只需要大于网络层中存放到 Buffer0 的最大值即可,Buffer1 同理.

在实现中,通过两方面的优化技术达到了这一点. 首先,将卷积层和池化层进行细粒度的流水. 即采用和卷积模块类似的方式,将经过卷积模块的两个数据地送入行缓存和一个 window,通过 FindMax 函数得到池化层之后的最终的 output data,如图 5 所示. 这样由于经过池化层,特征图的大小降低为原来的四分之一,Output Buffer 所需要的空间也降低为四分之一. 另外,UPsample19 层也可以和

CONV18 层进行细粒度的层间流水,只不过 UPsample19 层进行流水是为了减少运算周期,而不是减少缓存的大小.

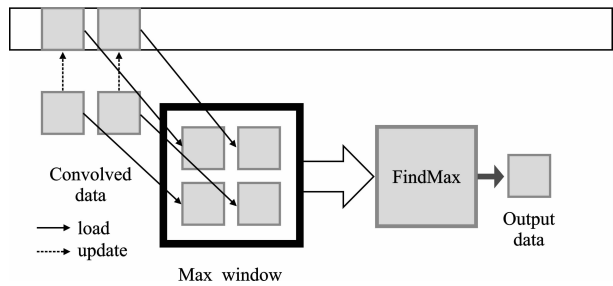


图 5 细粒度层间流水设计

其次,因为 YOLOv3-tiny 模型中,特征图的尺寸随着层数增大以 2 的平方倍率降低,因为行列方向上都减小为 2 倍,而特征图的通道数却只随着层数增大以 2 的倍率降低. 这样浅层的结果所需要的空间是最大的,那么针对 YOLOv3-tiny 网络多输出分支的卷积层,输入 Buffer 的一个分支的结果可以放在输出 Buffer,另一个分支的数据可以放在输入 Buffer 的高地址位. 如 Conv8 层和 Conv13 层的输出. 表 1 展示了各层所需要的缓存空间的大小. 缓存大小一行以输出缓存最小的 Conv18 层为基本单位, $13 \times 13 \times 128 = 21632$. 其他层的输出缓存需求都是其倍数. 其中,未标出地址的 Buffer 都是存放在最低的地址. 如表 1 所示,BF1 最大需要 32×21632 个 8bit 缓存,BF0 最大需要 16×21632 个 8bit 缓存,对应 516 个 BRAM18K.

表 1 各层输出缓存大小

网络层	C0-M1	C2-M3	C4-M5	C6-M7	C8-M9	C10-M11	C12
输出分支 1	BF1	BF0	BF1	BF0	BF1	BF0	BF1
缓存大小	32	16	8	4	2	4	8
输出分支 2	/	/	/	/	BF0[17;24]	/	/
缓存大小	/	/	/	/	8	/	/
网络层	C13	C14	C15	C18-U19	Concat	C21	C22
输出分支 1	BF0	BF1	BF0[1;2]	BF0[13;16]	BF0[13;24]	BF1	BF0[3;10]
缓存大小	2	4	2	4	12	8	8
输出分支 2	BF1[31;32]	/	/	/	/	/	/
缓存大小	2	/	/	/	/	/	/

对于 Weight 数据,如图 6 所示,通过 pingpong 的方式读取,可以减少 weight 读取的周期. 每个 tile 读取的个数由并行度决定,即 $8 \times 8 \times 9 = 576$ 个,依次为输入、输出通道和卷积核的并行度. 由于需要 576 个数据同时读取参与计算,故只能用 LUTRAM 实现,不消耗 BRAM.

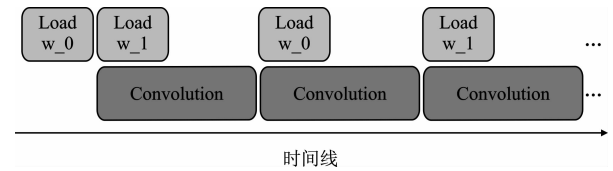


图 6 Weight 读取过程

4 实验结果与分析

4.1 实验环境

为了验证本文提到的高吞吐量加速器架构设计方案,实验在 Xilinx ZCU104 型号的 FPGA 板子上进行了 YOLOv3-tiny 模型的计算. 工作频率为 200 MHz,布局布线由 Vivado 工具完成.

4.2 实验结果与分析

本部分主要对比了不同设计方案下的性能指标以及 DSP 的利用率. 性能指标为单位时间硬件进行的计算次数,一般用 GOPs/s 表示,即每秒进行多少个 G 的操作,表示硬件的算力. 一般这个指标有理论性能和实际性能之分. 理论性能 P_{peak} 由片上的并行计算资源决定,可由公式(2)表示. 其中, PF 表示硬件可支持的并行度, β 表示每个单元每个周期可以支持的计算个数,在通常的加速器设计中,这个值为 2,一次 MAC 代表两次操作. 实际性能 P_{actual} 可由总的操作次数除以总的时间,也即总的操作次数和频率的乘积再除以总的运行周期数得到,如公式(3)所示. 这个实际的算力,通常跟网络结构,硬件设计效率,数据位宽都有关系. 为了更好地评估加速器的设计效率,即 DSP 的利用率,参考文献[6],引入 $DSP_{efficiency}$ 来评估不同 FPGA 加速的方案的设计效率,如公式(4)所示. 该值越大越好,说明越充分利用了实际硬件的计算潜力.

$$P_{peak} = \beta \times PF \times f \tag{2}$$

$$P_{actual} = \frac{OPs}{t} \tag{3}$$

$$DSP_{efficiency} = \frac{P_{actual}}{P_{peak}} \tag{4}$$

表 2 性能对比

	文献[15]	文献[16]	文献[6]	本文
硬件平台	VC707	Zedboard	ZC-706	ZCU104
频率/MHz	200	100	200	200
网络类型	YOLOv3-tiny	YOLOv3-tiny	YOLOv2-tiny	YOLOv3-tiny
位宽/bits	18	16	16	8
DSP	2304/3600	160/220	680/900	576/1728
BRAM	141/2060	185/280	185/545	516/624
延时/ms	—	532	59	21
实际算力/GOPs/s	460.8	10.5	234.0	263.8
DSP 效率/%	20.0	6.5	86.0	57.2

如表 2 所示,本设计的实际算力为 263.8 GOPs/s,这个值可由网络模型的操作次数除以延时计算可得. FPGA 处理延时为 21 ms,相当于每秒处

理约 47.6 张图片,完全能够满足实时性的要求,并且超过其他文献的结果. 本文的 DSP 效率可由公式(4)计算, P_{actual} 为 263.8 GOPs/s, P_{peak} 计算公式为 $2 \times 1152 \times 200M = 460.8$ GOPs/s. 2 表示 β , 1152 为卷积公式所有维度并行度的乘积,即 $2 \times 8 \times 8 \times 9$. 200 M 为 FPGA 运行的频率. P_{actual} 和 P_{peak} 相除即为 DSP 效率 57.2%. 文献[15]和文献[16]的 DSP 效率计算过程同理,因此本文在 DSP 效率上有 2.86 倍或 8.81 倍的提升. 本设计的 DSP 效率不如文献[6],这是因为本文的网络是 YOLOv3-tiny 模型,具有多分支的网络结构,而文献[6]使用的是相对简单的 YOLOv2-tiny 模型. 本设计的 DSP 效率未能达到最高的原因主要是,输出 neuron 列方向并行度设为 2,而不同层的特征图尺寸并不总是 2 的倍数.

5 结束语

根据实验结果,本文提出的多维度并行加速器可以很好地适应 YOLOv3-tiny 复杂的网络结构,在 DSP 利用率方面,相比于其他针对 YOLOv3-tiny 的设计方案有了 2.86 倍或 8.81 倍的提升. 加速核心可以应用到 3×3 卷积核和 1×1 卷积核类型网络,使得硬件的算力和兼容性同时得到提升. 通过层融合和量化位宽的技术,使得网络模型的参数量大大降低,尽可能降低了对于 FPGA 片上缓存容量的需求. 在延时指标上,达到了 21 ms,也超过了同类型其他加速器设计,完全可以满足实时性检测的要求.

参考文献:

[1] WANG H, ZHANG Q M, WANG Y H, et al. Structured probabilistic pruning for convolutional neural network acceleration[C]//BMVC. 2018, 3: 7.

[2] NGUYEN D T, KIM H, LEE H J, et al. An approximate memory architecture for a reduction of refresh power consumption in deep learning applications[C]//2018 IEEE International Symposium on Circuits and Systems (ISCAS). Florence, Italy: IEEE, 2018: 1-5. DOI: 10.1109/ISCAS.2018.8351021.

[3] NGUYEN D T, NGUYEN T N, KIM H, et al. A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019, 27(8): 1861-1873. DOI: 10.1109/TVLSI.2019.2905242.

[4] ZHANG C, LI P, SUN G Y, et al. Optimizing FPGA-based accelerator design for deep convolutional

- neural networks[C]//Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, California, USA: ACM, 2015: 161-170. DOI: 10.1145/2684746.2689060.
- [5] WILLIAMS S, WATERMAN A, PATTERSON D. Roofline: an insightful visual performance model for multicore architectures[J]. Communications of the ACM, 2009, 52(4): 65-76. DOI: 10.1145/1498765.1498785.
- [6] ZHANG X F, WANG J S, ZHU C, et al. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs[C]//2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). San Diego, CA, USA: IEEE, 2018: 1-8. DOI: 10.1145/3240765.3240801.
- [7] 李岑, 贺光辉. 用于实时目标检测的 FPGA 神经网络加速器设计[J]. 微电子学与计算机, 2020, 37(7): 6-11. DOI: 10.19304/j.cnki.issn1000-7180.2020.07.002.
- LI C, HE G H. Design of FPGA-based neural network accelerator for real-time objective detection[J]. Microelectronics & Computer, 2020, 37(7): 6-11. DOI: 10.19304/j.cnki.issn1000-7180.2020.07.002.
- [8] REDMON J, FARHADI A. YOLO9000: better, faster, stronger[C]//Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition. Honolulu, HI, USA: IEEE, 2017: 6517-6525. DOI: 10.1109/CVPR.2017.690.
- [9] REDMON J, FARHADI A. YOLOv3: An incremental improvement[J]. arXiv e-prints, 2018.
- [10] PEEMEN M, MESMAN B, CORPORAAL H. Inter-tile reuse optimization applied to bandwidth constrained embedded accelerators[C]//Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). Grenoble, France: IEEE, 2015: 169-174. DOI: 10.7873/date.2015.1033.
- [11] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90. DOI: 10.1145/3065386.
- [12] IOFFE S, SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//International conference on machine learning. PMLR, 2015: 448-456.
- [13] VANHOUCKE V, SENIOR A, MAO M Z. Improving the speed of neural networks on CPUs[C]//Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011. 2011.
- [14] LU W Y, YANG G H, LI J J, et al. FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks[C]//2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). Austin, TX, USA: IEEE, 2017: 553-564. DOI: 10.1109/HPCA.2017.29.
- [15] AHMAD A, PASHA M A, RAZA G J. Accelerating tiny YOLOv3 using FPGA-Based hardware/software co-design[C]//2020 IEEE International Symposium on Circuits and Systems (ISCAS). Seville, Spain: IEEE, 2020: 1-5. DOI: 10.1109/ISCAS45731.2020.9180843.
- [16] YU Z W, BOUGANIS C S. A parameterisable FPGA-Tailored architecture for YOLOv3-Tiny[C]//International Symposium on Applied Reconfigurable Computing. Toledo, Spain: Springer, 2020: 330-344. DOI: 10.1007/978-3-030-44534-8_25.

作者简介:

谢帅 男, (1997-), 硕士研究生. 研究方向为加速器、集成电路设计等. E-mail: xieshuai@sjtu.edu.cn

蒋力 男, (1984-), 博士, 副教授. 研究方向包括计算机体系结构、电子设计自动化、可靠性等.

叶瑶瑶 女, (1987-), 博士, 讲师. 研究方向包括计算机体系结构、电子设计自动化、SoC 等.