# 基于Winograd算法的目标检测加速器设计与优化

李 斌[1,2],齐延荣[1,2],周清雷[1,2]
(1. 郑州大学计算机与人工智能学院,河南郑州 450001; 2. 郑州大学信息工程学院,河南郑州 450001)

**摘 要:** 卷积神经网络(Convolutional Neural Networks,CNN)已被广泛应用于图像处理领域. 基于CNN的目标检测模型,如YOLO,已被证明在许多应用中是最先进的. CNN对计算能力和内存带宽要求极高,通常需要部署到专用硬件平台,FPGA因其高性能、低功耗和可重配置性成为CNN的有效硬件加速器. 以往的基于FPGA的目标检测加速器主要采用传统卷积算法,然而,传统卷积算法的高运算复杂度限制了加速器的性能. 基于此,本文设计了一种基于Winograd算法的目标检测加速器. 考虑到各模块间的联系,采用模块融合策略融合卷积层和池化层模块,降低数据移动次数,减少片外存储器访问次数,提高加速器整体性能. 以YOLO2模型为例,对数据访问模式、池化内核、参数重排序、数据通路优化进行分析设计,并部署在U280板卡上. 实验结果表明,量化后mAP降低了0.96%,性能达249.65 GOP/s,是Xilinx官网所给数据的4.4倍.

**关键词:** 目标检测; FPGA; Winograd算法; 模块融合; YOLO2
**中图分类号:** TP391.41;TP183 **文献标识码:** A **文章编号:** 0372-2112(2022)10-2387-11
**电子学报URL:**http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20201371

# Design and Optimization of Target Detection Accelerator Based on Winograd Algorithm

LI Bin, QI Yan-rong, ZHOU Qing-lei
(1. *School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, Henan* 450001, *China*;
2. *School of Information Engineering, Zhengzhou University, Zhengzhou, Henan* 450001, *China*)

**Abstract:** Convolutional neural network(CNN) has been widely used in the field of image processing. CNN-based target detection models, such as YOLO, have proven to be the most advanced in many applications. CNN has extremely high requirements for computing power and memory bandwidth, and usually needs to be deployed on a dedicated hardware platform. FPGA has become an effective hardware accelerator for CNN due to its high performance, low power consumption and reconfigurability. In the past, FPGA-based target detection accelerators mainly used traditional convolution algorithms. However, the high computational complexity of traditional convolution algorithms limited the accelerator's performance. Based on this, this paper designs a target detection accelerator based on Winograd algorithm. Taking into account the connection between the modules, the module fusion strategy is adopted to fuse the convolutional layer and the pooling layer modules to reduce the number of data movement, reduce the number of off-chip memory accesses, and improve the overall performance of the accelerator. Take the YOLO2 model as an example, analyze and design the data access mode, pooled kernel, parameter reordering, and data path optimization, and deploy it on the U280 board. The experimental results show that mAP is reduced by 0.96% after quantification, and the performance reaches 249.65GOP/s, which is 4.4 times the data given by Xilinx official website.

**Key words:** target detection; FPGA; Winograd algorithm; module integration; YOLO2

## 1 引言

近年来,卷积神经网络(Convolutional Neural Networks,CNN)在计算机视觉领域取得了巨大成功. CNN模型的时间复杂度,即模型的运算次数可用FLOPs衡量,单个卷积层的时间复杂度为$O(M^2\times K^2\times C_{in}\times C_{out})$. 其中,$M$表示输出特征图的宽度,$K$表示卷积核的宽度,$C_{in}$表示输入通道数,$C_{out}$表示输出通道数. CNN模型整体的时间复杂度是各卷积层复杂度的累加. 随着应用需求

收稿日期:2020-12-01;修回日期:2021-08-17;责任编辑:王天慧

的增加和应用场景的复杂性，CNN 网络的层次不断加深，深度学习模型的计算复杂度也不断增加．目标检测是深度学习中最具挑战性的任务．目标检测算法中，最具代表性的网络 YOLO[1,2]算法比其他算法具有更快的性能．

CNN 识别精度的提高是以巨大的计算复杂度为代价的．CPU 很难提供 CNN 所要求的大规模并行性，面对如此巨大的计算压力，硬件加速器（如 GPU，FPGA，ASIC）被用来加速 CNN．在这些设计中，FPGA 可重构的特点使其能很好地适应当前深度学习日新月异的演变速度，因此已有许多基于 FPGA 的卷积神经网络加速器的研究[3]．在 FPGA 框架设计方面，人们已经对各优化手段做出充分研究，例如如何做并行计算，如何做数据定点化工作，如何降低带宽需求等等．CNN 卷积层包含大量乘法运算，乘法运算需消耗大量时间．FPGA 所包含的 DSP 资源主要用于卷积的乘法运算．减少乘法次数，能够提高 DSP 利用率．Winograd 算法利用元素间的结构相似性，将输出特征图中元素组合在一起．与普通矩阵乘法相比，使用 Winograd 算法能够减少乘法次数，进而达到加速 CNN 的目的[4~8]．研究证明 Winograd 算法适合为具备小型滤波器的 CNN 推导高效算法．YOLO2 模型中滤波器尺寸为 3×3，极少数为 1×1，显然，在 YOLO2 中引入 Winograd 算法可以有效降低算法复杂度，从而在 FPGA 平台上实现比传统卷积算法更好的加速性能．

CNN 通常涉及多个层，其中一层的输出特征映射为下一层的输入特征映射．卷积运算通常占 CNN 总运算量的 90% 以上，现有 CNN 的 FPGA 实现主要基于传统卷积算法，将 Winograd 算法用于目标检测 CNN 加速器的设计工作较少．使用传统卷积算法的 CNN 的 FPGA 实现方案已取得较高的效率，如果提高算法本身效率，更高的效率是可能的．基于此，本文详细分析了 YOLO2 推理阶段的调度流程．对文献[4,7]中存在的问题，提出解决方案：调整循环执行顺序，最大化复用缓冲区内数据，重排序参数数据，使得所需数据连续存储在存储器．基于文献[5,9,10]已有工作，做出改进：融合模块，降低数据传输延时．本文主要贡献如下：

（1）对部署到 FPGA 的 YOLO2 模型进行优化设计时，采用 Winograd 算法代替传统卷积算法，提高计算性能．

（2）Winograd 卷积采用行缓冲结构最大限度地实现数据复用，降低外部 DRAM 访存次数．

（3）深入分析卷积循环的优化策略，采用一种以最小化数据移动和内存访问为目标的循环执行顺序．

（4）重排输入特征映射参数、权重参数、输出特征参数，使所需数据参数连续存储，增大传输数据的突发长度．

## 2 背景

### 2.1 目标检测

本设计选用的 YOLO2 网络模型如图 1 所示，主要包括卷积层和池化层．YOLO2 采用改进的 Darknet-19 框架，通过移除最后一个卷积层来修改网络以进行检测，可以同时预测边界框的位置和类别．YOLO2 网络中第 0~22 层是 Darknet-19 网络，后面从第 23 层开始，是添加的检测网络．YOLO2 由大量卷积层构成，因卷积层计算密集，将该模型部署到嵌入式终端，运行时间较长．然而，FPGA 具备高效并行计算能力，将 YOLO2 部署到 FPGA 上能够缩短运行时间．

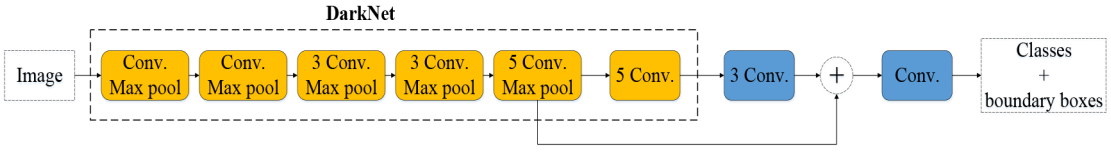



图 1 YOLO2 网络结构

### 2.2 Winograd 卷积算法

Winograd 算法最早于 1980 年由 Shmuel Winograd 提出，主要用来减少 FIR 滤波器的计算量．Winograd 卷积通过对 input tile 和 filter 的转换来减少乘法数量，乘法的减少意味着较少的循环迭代，从而提高性能．为简单起见，用 $F(m,r)$ 表示具有 $m$ 个输出的 r-tap FIR 滤波器．

在 Winograd 算法中，对于一维卷积，输出为 $m$，卷积核长为 $r$，则其计算过程如式（1）所示，对应的乘法数量为 $\mu(F(m,r))=m+r-1$．将一维卷积扩展到二维，如果输出维度为 $m\times n$，卷积核维度为 $r\times s$，则计算过程如式（2）所示，其对应的乘法数量为 $\mu(F(m\times n,r\times s))=\mu(F(m,r))\times\mu(F(n,s))=(m+r-1)\times(n+s-1)$

$$\boldsymbol{Y}=\boldsymbol{A}^{\mathrm{T}}\left[(\boldsymbol{G}g)\odot(\boldsymbol{B}^{\mathrm{T}}d)\right] \tag{1}$$

$$\boldsymbol{Y}=\boldsymbol{A}^{\mathrm{T}}\left[\left[\boldsymbol{G}g\boldsymbol{G}^{\mathrm{T}}\right]\odot\left[\boldsymbol{B}^{\mathrm{T}}d\boldsymbol{B}\right]\right]\boldsymbol{A} \tag{2}$$

其中，$\odot$ 为 element-wise multiplication（Hadamard product）对应位置相乘；$g$ 为卷积核；$d$ 为输入信号；$\boldsymbol{G}$ 为 Filter transform 矩阵，尺寸为 $(m+r-1)\times r$；$\boldsymbol{B}^{\mathrm{T}}$ 为 Input transform 矩阵，尺寸为 $(m+r-1)\times(m+r-1)$；$\boldsymbol{A}^{\mathrm{T}}$ 为 Output transform 矩阵，尺寸为 $m\times(m+r-1)$．

对于 $F(4,3)$，假定输入为 $\boldsymbol{x}=(x_0,x_1,x_2,x_3,x_4,x_5)$，卷积核为 $\boldsymbol{w}=(w_0,w_1,w_2)$，输出为 $\boldsymbol{y}=(y_0,y_1,y_2)$．对于

Winograd算法,首先对输入$\boldsymbol{x}$、卷积核$\boldsymbol{w}$进行适当变换,如式(3)和式(4)所示;其次,通过计算$\boldsymbol{x}'$和$\boldsymbol{w}'$的点积,得到中间变换结果,如式(5)所示;最后,Winograd算法给出的输出数据如式(6)所示.

$$\boldsymbol{x}'=\begin{pmatrix}4x_0-5x_2+x_4,\ -4x_1-4x_2+x_3+x_4,\\ 4x_1-4x_2-x_3+x_4,\ -2x_1-x_2+2x_3+x_4,\\ 2x_1-x_2-2x_3+x_4,4x_1-5x_3+x_5\end{pmatrix} \tag{3}$$

$$\boldsymbol{w}'=\begin{pmatrix}\frac{w_0}{4},\ -\frac{w_0+w_1+w_2}{6},\\ -\frac{w_0-w_1+w_2}{6},\ \frac{w_0+2w_1+4w_2}{24},\\ \frac{w_0-2w_1+4w_2}{24},w_2\end{pmatrix} \tag{4}$$

$$\begin{cases}f_1=\dfrac{\left(4x_0-5x_2+x_4\right)w_0}{4}\\ f_2=-\dfrac{\left(-4x_1-4x_2+x_3+x_4\right)\left(w_0+w_1+w_2\right)}{6}\\ f_3=-\dfrac{\left(4x_1-4x_2-x_3+x_4\right)\left(w_0-w_1+w_2\right)}{6}\\ f_4=\dfrac{\left(-2x_1-x_2+2x_3+x_4\right)\left(w_0+2w_1+4w_2\right)}{24}\\ f_5=\dfrac{\left(2x_1-x_2-2x_3+x_4\right)\left(w_0-2w_1+4w_2\right)}{24}\\ f_6=\left(4x_1-5x_3+x_5\right)w_2\end{cases} \tag{5}$$

$$\begin{aligned}\boldsymbol{y}&=\begin{bmatrix}y_0\\y_1\\y_2\\y_3\end{bmatrix}=\begin{bmatrix}x_0&x_1&x_2\\x_1&x_2&x_3\\x_2&x_3&x_4\\x_3&x_4&x_5\end{bmatrix}\begin{bmatrix}w_0\\w_1\\w_2\end{bmatrix}\\&=\begin{bmatrix}f_1+f_2+f_3+f_4+f_5\\f_2-f_3+2f_4-2f_5\\f_2+f_3+4f_4+4f_5\\f_2-f_3+8f_4-8f_5+f_6\end{bmatrix}\end{aligned} \tag{6}$$

在变换过程中,由于卷积核$w$在某阶段是固定的参数,卷积核的变换可以提前计算,并得到1个6×1的向量;由于输入$\boldsymbol{x}$是变化的,每次计算FIR时都需要对输入$\boldsymbol{x}$做变换,得到1个6×1向量,这个过程需要16次加法.$\boldsymbol{x}'$和$\boldsymbol{w}'$进行点积时,共计6次乘法. 为得到最终结果,还需14次加法,即Winograd卷积只需6次乘法和30次加法.

如果直接计算$F(4,3)$,则如式(7)所示. 其中,$(x_0)$,$(x_1,x_2)$、$(x_1,x_2,x_3)$、$(x_2,x_3,x_4)$和$x_3,x_4,x_5$为4个输入序列且它们是连续的,$w_0,w_1,w_2$为卷积核的3个参数,这一过程总共需要12次乘法和8次加法.

FPGA含有大量加法器资源,加法的增多对算法效率影响小,乘法需消耗DSP资源,布线时间长,FPGA包含的DSP资源有限,故减少乘法可以提升算法效率.

$$\begin{aligned}F(4,3)&=\begin{bmatrix}x_0&x_1&x_2\\x_1&x_2&x_3\\x_2&x_3&x_4\\x_3&x_4&x_5\end{bmatrix}\begin{bmatrix}w_0\\w_1\\w_2\end{bmatrix}\\&=\begin{bmatrix}x_0w_0+x_1w_1+x_2w_2\\x_1w_0+x_2w_1+x_3w_2\\x_2w_0+x_3w_1+x_4w_2\\x_3w_0+x_4w_1+x_5w_2\end{bmatrix}\end{aligned} \tag{7}$$

综上所述,对于$F(4,3)$,Winograd算法需要6次乘法,普通卷积需要12次乘法. Winograd算法是以额外空间为代价来减少乘法次数,达到加速目的的.

### 2.3　相关工作

YOLO是目标检测的代表性网络,学者们对YOLO模型的加速器进行了大量研究,并取得了卓越成果. Bao等人[4]提出一种基于Winograd算法的目标检测模型YOLO的加速器设计方法,采用Winograd算法降低模型计算复杂度,采用缓冲流水线优化存储器,量化CNN模型参数,降低资源消耗,在检测精度损失小于3%的情况下,该设计方案不仅优化了FPGA资源利用率,而且将功耗降低到了2.7 W. Huang等人[5]提出了一种基于Winograd算法的加速器架构,该架构既适用于卷积层也适用于全连接层;同时,还建立了分析模型,用来评估性能和资源利用率;使用高级综合工具实现设计,使用VGG16从资源利用率、性能和效率3个方面评估了不同Tile大小的加速器,综合考虑,$F(4\times4,3\times3)$的设计取得了较好的性能;在VUS440平台上,VGG16平均性能达到943 GOPS. Shi等人[7]针对存储器技术问题,提出一种高效的递归内存访问布局的设计方法,使用XCVU095板卡评估该方案,该方案具有很高的计算资源利用率,吞吐量高达921.6 GOP/s. Nguyen等[9]提出一种高性能硬件高效的流式实时目标检测体系结构,通过量化网络和优化数据路径来消除中间数据的片外访问;通过对流的批处理,在不增加硬件成本的前提下,实现了1.877个TOPS的吞吐量,优于以往的大多数设计. Lian等人[10]提出了一种基于块FP算法的CNN加速器,以减少CNN模型的硬件成本和数据通信量,该方法在不同CNN模型上的准确率损失不到0.12%,无需再训练;并利用三级并行卷积引擎、乒-乓存储器访问模式和优化的片内缓冲方案,将所提架构部署在Xilinx VC709评估板,实现了760.83 GOP/s的吞吐量和82.88 GOP/(s·W)的功率效率,明显优于以前的架构.

文献[4]虽采用缓冲流水线优化存储器优化内存,充分利用每个缓冲区的优势,保证数据的交互和传输. 但却没有考虑缓冲区内数据的重用性,增加了数据访

存次数. 文献[5]着重探讨在FPGA上使用Winograd算法加速CNN的可能性,并采用PE的并行设计提高计算模块吞吐量. 文献[7]结合稀疏Winograd卷积、小规模脉动阵列簇和可定制的内存布局设计,在FPGA上部署了一个加速器,其虽具有较好性能,但未考虑存储器技术的进步,合理优化存储器可进一步提升性能. 文献[9]对YOLO参数进行了重新训练和量化,采用了二进制权重和低位激活操作,降低片外访存.

## 3 基于FPGA的设计

### 3.1 加速器体系架构

本设计的加速器总体硬件架构如图2所示,软件结构可分为2部分:主机端(host)和设备端(device). CPU能够控制host和device之间的所有接口,利用CPU调度将YOLO2网络参数的特征图输入到DDR内存缓冲区,并通过总线与外围操作系统电路进行交互. CPU采用AXI总线读取加速电路中的运行结果,并在host端执行图像预处理和显示. 在device端,外部DDR内存中的数据被缓存到片上RAM中,YOLO2加速器的读内存(memRead)、卷积(Conv)、激活(ReLU)、池化(Pool)和写内存(memWrite)电路在FPGA中布局布线,构成整个硬件加速器的数据通路.

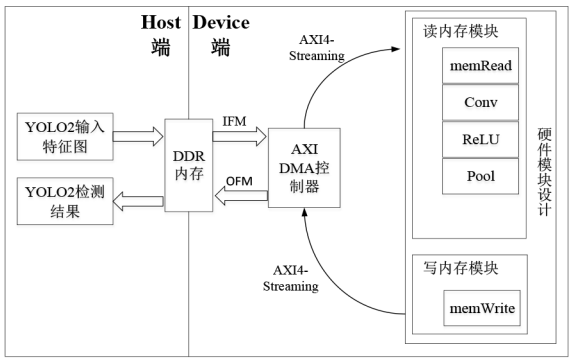



图2 基于FPGA的YOLO2加速器体系结构图

### 3.2 FPGA顶层架构

用于目标检测的标准CNN由一个或多个卷积层、池化层构成,建议一个功能一个模块,如图3所示,这些模块通过管道(pipes)连接.

其中,MemRD模块从全局内存中读取输入特征数据、权重数据;Conv模块实现卷积乘加运算;Pool模块在Conv的输出数据流上执行下采样;MemWR模块向全局内存写入输出特征数据. 级联内核模块形成一个深层的计算管道,可以实现一系列基本的CNN操作,而无需将层间数据存储回全局内存中.

### 3.3 卷积运算

卷积是CNN算法中的主要运算,涉及输入特征图

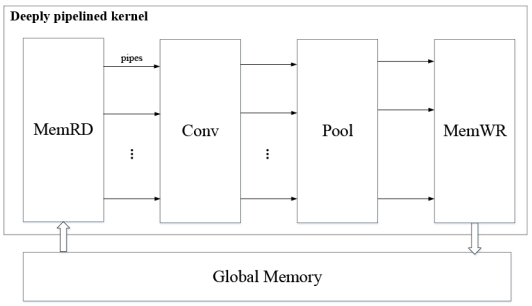



图3 FPGA顶层架构

和卷积核(权重)的三维乘累加(Multiply Accumulate, MAC)运算. 卷积由4层循环组成,算法1为卷积计算算法. 图4展示了Winograd卷积运算流程. 其中,Nkx和Nky表示卷积核尺寸,tile_x和tile_y分别表示一维、二维分块尺寸,Nif和Nof分别表示输入特征图通道数、输出特征图通道数,Nix和Niy分别表示输入特征图一维、二维像素数,Nox和Noy分别表示输出特征图一维、二维像素数,$S$表示步长,$L$表示卷积层索引.

**算法1 卷积计算算法**

**Input:** Nkx,Nky,Nif,Nof,Nix,Niy,Nox,Noy,$S$,$L$

**Output:** pixelL(no;$x$,$y$)

**Step:**

```
For no=0 to Nof
  For y=0 to Noy
    For x=0 to Nox
      For ni=0 to Nif
        For ky=0 to Nky
          For kx=0 to Nkx
            pixelL(no;x,y) ←pixelL(no;x,y)+
                pixelL-1(ni;x+kx,y+ky)*weightL-1(ni,no;kx,ky)
            pixelL(no;x,y)=pixelL(no;x,y)+bias(no)
```

### 3.4 调度流程

FPGA片上内存存储资源有限,无法完全存储深度CNN算法的全部数据. 因此,使用更密集的外部DRAM来存储所有层的权重和中间像素结果是合理的.

依据CNN模型层的定义进行分层. 由于每层卷积运算所需卷积核规模较大,并且考虑FPGA资源,对每层进行分组划分,各组能够并行执行,复用输入特征数据(组数,即并行通道数). 由于各组所含特征图尺寸较大,很难将所有数据一次计算完成,根据卷积核长宽对特征图分块,每块尺寸与卷积核一致,块内数据可以一次计算(块,即与卷积核尺寸相同的输入特征图). 本文研究内容中,分组大小为8,分块大小为8.

卷积神经网络模型层与层之间串行执行,前一层输出作为当前层的输入,如图5($a$)所示. 利用卷积计算的数据局部性对输入特征图和权重特征图进行分组划分,复用片上缓存数据,减少访存次数. 如图5($b$)所示,

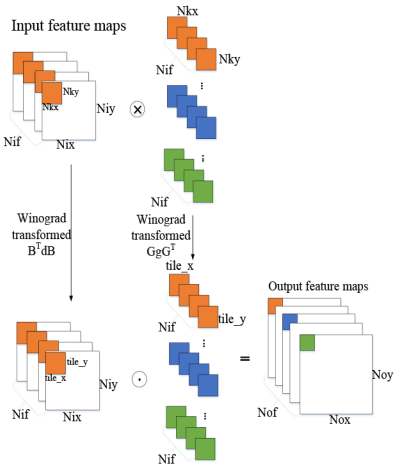



图4　卷积运算及其参数

每次读取、写入多组数据,缓解FPGA资源不足的问题.如图5(*c*)所示,每次读取、写入多块数据,块数据复用结束后,清空块内缓存数据,读取下一分块数据,使得访存不至于成为加速器设计的瓶颈.对读取到的分块数据,根据图5(*d*)的控制流处理:根据层类别,将片上缓存中的块数据分发给Conv和Pool模块进行处理,将处理后的数据写入输出缓冲区中,将得到的最终结果写入片外DRAM缓存.

## 4　优化策略

### 4.1　融合优化

节点融合是指将多个节点融合为一个高效节点.节点融合优化是窥孔优化技术的一种,通过对融合前后的节点进行代价评价,根据代价决定是否进行节点融合优化.节点融合过程如图6所示,其中,虚线圆圈为待融合节点,箭头为节点间的数据依赖关系.经过节点融合优化,将操作节点OP2和OP4融合为节点OP5,同时维持了节点OP2和OP4的依赖关系.

借鉴于节点融合思想,考虑到各模块之间的联系,将卷积模块(conv)、池化模块(pool)融合到内存读模块(memRD)中,并维持原有模块的依赖关系.融合优化实施方法为:将memRD模块生成的输入、权重、偏置数据暂存到数组变量in,weight,bias中(片上RAM),Conv模块从in,weight,bias中直接取出数据,做卷积运算,并将卷积结果暂存到数组conv中,Pool模块取出conv存储的数据,对提取到的特征降采样.经模块融合,将操作模块memRD,Conv,Pool融合为模块memRD′,如图7所示.其中,虚线圆圈为待融合模块,箭头为模块间的数据依赖关系.

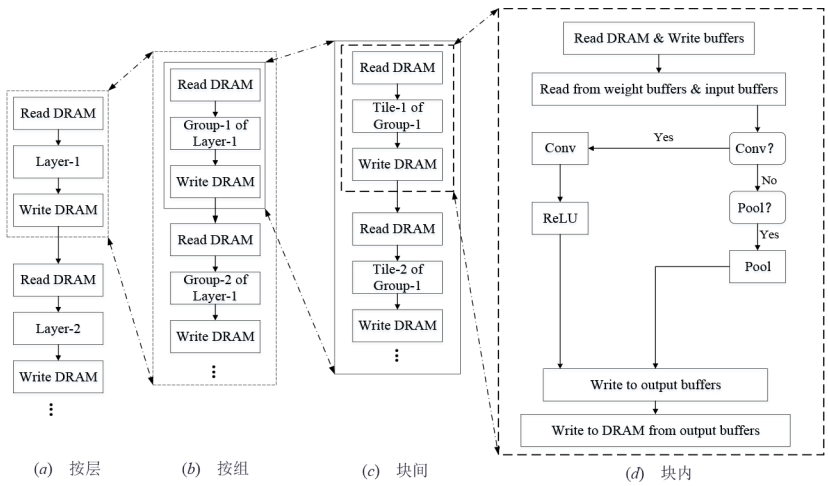
模块融合后,FPGA顶层架构如图8所示.以其中一层为例,假设模块融合前传输时延为TransmissionTime,则其计算式如式(8)所示,模块融合后传输时延为TransmissionTime′,计算式如式(9)所示.其中,$data_{in}$表示输入数据参数量,$data_{weight}$表示权重数据参数量,$data_{bias}$表示偏置参数量,$data_{conv}$表示卷积输出参数量,$data_{pool}$表示池化输出参数量,bandwidth表示传输带宽.模块融合前各模块通过通道传输数据,模块融合后,将存储到通道中的数据转存到变量中,需要时不必从通道中获取,减少了数据移动次数(即减少片外存储器访问次数),降低了数据传输延迟[11,12].

$$\text{TransmissionTime}=\frac{data_{in}+data_{weight}+data_{bias}}{bandwidth}\times 2+\frac{data_{conv}+data_{pool}}{bandwidth} \tag{8}$$

$$\text{TransmissionTime}'=\frac{data_{in}+data_{weight}+data_{bias}}{bandwidth}+\frac{data_{pool}}{bandwidth} \tag{9}$$

### 4.2　数据访问模式

为有效地映射和执行卷积循环,主要采用3种优化




(*a*) 按层　(*b*) 按组　(*c*) 块间　(*d*) 块内

图5　调度流程图

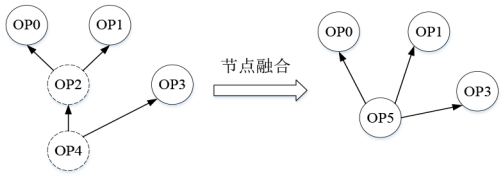



图6 节点融合

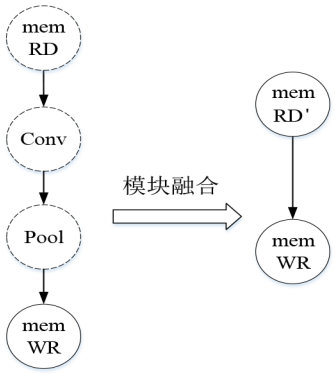



图7 模块融合

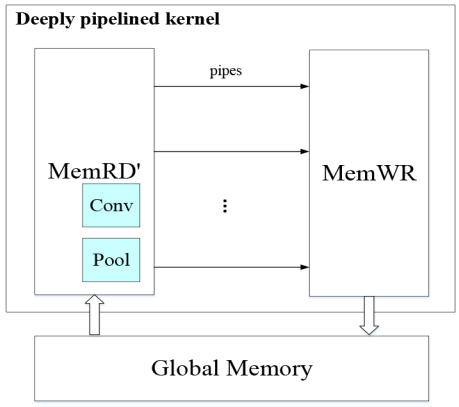



图8 模块融合后的FPGA顶层架构

技术:循环展开、循环平铺和循环交换,定制加速器的计算和通信模式. 以Loop-2为例,循环展开操作如图9所示. 展开不同的卷积循环将指导不同卷积计算的并行化,这会影响到有关数据重用机会和内存访问模式的最佳PE阵列体系结构. 循环平铺用于将整个数据分为多个块,这些块可以放入片上缓冲区,其决定了片上缓冲区所需的容量,执行块的数据从外部存储器中读取并存储在片上缓冲区中. 循环交换决定了4个卷积循环的计算顺序.

本文采取的卷积运算策略如图10所示,循环计算顺序为Loop-1→Loop-3→Loop-2→Loop-4. Loop-2和Loop-4作为最内部的2个循环,可以提高权重的复用率,降低访存次数. Loop-1和Loop-3作为最外部循环,一次能够输出一个像素,可以缓解片上存储资源的压力. Nkx和Nky平铺因子分别设置为6和1(假设Tkx=6,Tky=1),Nif展开因子为16(即$M$=16),Nox和Noy展开

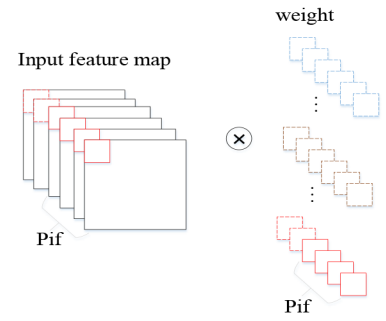



图9 循环展开

因子分别为4和1(假设Pox=4,Poy=1),Nof展开因子为32(即$N$=32). Nkx和Nky对应Loop-1的平铺因子,即一个Winograd卷积操作;Nif对应Loop-2的展开因子,即并行输入通道数;Nox和Noy对应Loop-3的展开因子,即卷积核复用;Nof对应Loop-4的展开因子,即并行输出通道数.

首先,输入滑动块沿输入图像的宽度滑动,这一过程被称为行过程(a row pass). 每次将输入滑动块与$N$个权重块卷积,以生成$N$个临时输出值. 这些权重块可重复用于行过程. 这些计算被并行处理并保存在行缓冲区中,从而创建临时输出通道

然后,输入滑动块沿通道方向滑动,提取新的$N$个权重块并将其与滑动块进行卷积. 卷积输出与来自行缓冲器的相应值累加,然后保存在行缓冲器中. 此操作重复$N_i$=Nif/$M$次,直到计算出所有Nif个输入信道,此时行缓冲区中的值是最终输出通道.

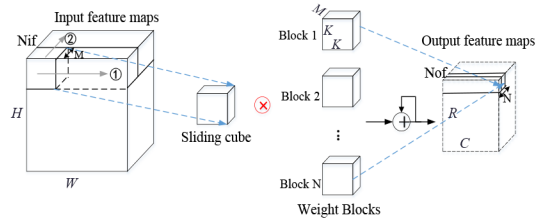



图10 卷积运算策略

### 4.3 Winograd卷积

以其中一个输入块为例,图11详细绘制了采用Winograd算法的卷积层计算结构图. Winograd卷积一次生成一块输出特征映射,具体地说,给定一个$n\times n$的输入块,Winograd卷积将生成一个$m\times m$的输出块,为得到下一个$m\times m$输出块,只需将输入块卷积窗口步长设为$m$.

Winograd卷积对内存带宽要求很高,由于相邻块在水平和垂直方向具有共享的输入特征数据,利用这一特性设置行缓冲区. 在行缓冲区,横向滑动输入块,最大限度地实现数据复用,降低访存次数. 行缓冲区(line buffer)的每一行在所有通道中存储相同的行,每个输入行缓冲区包含$M\times W$个元素,PE从行缓冲区中获

取数据. 最后,采用双缓冲区重叠数据传输和计算.

为重用数据,将 $n+m$ 条输入行缓冲区存储在片上存储器,并将其作为一个循环缓冲区执行循环操作. 最初,计算引擎直接从输入行缓冲区读取前 $n$ 行数据,而其下一个 $m$ 行数据将从外部 DDR 读取. 由于相邻 tile 之间的跨距 stride=$m$,计算引擎将跳过 $m$ 行并处理行缓冲区中的下一个 $n$ 行数据,从外部 DDR 中读取的数据覆盖跳过的 $m$ 行数据,这个过程被循环执行.

将二维卷积沿高度方向划分为多个一维卷积,先计算其中一个一维卷积,再计算其他一维卷积,循环执行,直到多个一维卷积执行完毕. 最后,综合一维卷积运算结果,合并为二维卷积运算结果.

其中,$M$ 表示输入特征图展开因子,$W$ 表示输入特征图的宽度,$H$ 表示输入特征图的高度,$N$ 表示 filter 个数(输出特征图展开因子),$C$ 表示输出特征图的宽度,$R$ 表示输入特征图的高度.

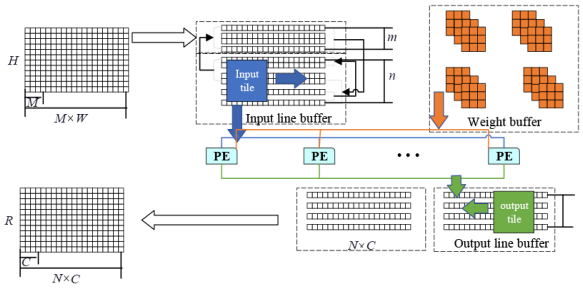



图 11　Winograd 卷积设计

## 4.4　池化内核优化

为池化内核提出了一种基于行缓冲区的硬件结构,以 2×2 池化核为例,其内部逻辑如图 12 所示. 池化计算原理如下:

(1)内核先从通道中逐行读取特征图的数据,然后将读取的数据保存在线性缓冲区中,本设计中使用 1 个线性缓冲区(line_buf_0);

(2)借助行缓冲区求出列最大值,并将列最大值存储在池化寄存器 pool_reg 中,然后通过池化寄存器移位计算出行最大值,即一个池化窗口最大值(pool_final).

该池化操作通过采用行列同时执行的方式,节省了计算时间,提升了池化操作速度.

## 4.5　片上存储单元设计

数据传输是加速器优化设计的主要约束. 所有输入特征数据、权重数据都存储在外部存储器中. 因 CNN 输入数据参数量大,FPGA 片上存储资源有限,加速器很难同时从外部存储器获取所有数据. 由于存储带宽有限,将输入特征数据、权重数据写入片上缓存时存在访存时延. 为降低访存时延对系统优化的影响,采用乒-乓缓冲区将数据写入和计算重叠,减少数据的传输时延[13],如图 13 所示. 缓冲区 1 被写入数据时,从缓冲区 2 获取计算数据;缓冲区 2 被写入数据时,从缓冲区 1 获取计算数据.

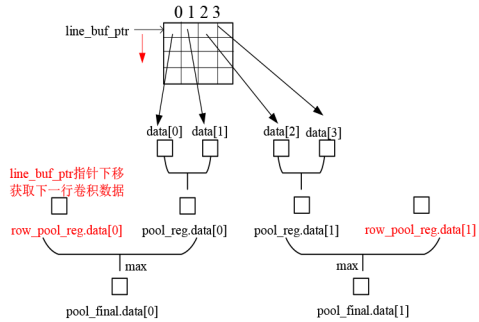



图 12　池化操作

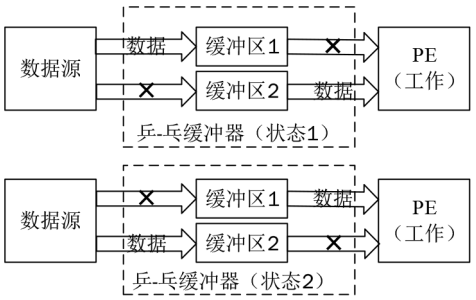



图 13　“乒-乓”缓冲

## 4.6　参数重排序

为减少从外部存储器读取输入特征图、权重参数的次数,对输入特征图、权重参数重新排序,使得卷积计算所需的参数在内存中连续存储,增大传输长度,有效提高带宽利用率. 以权重参数为例,根据分块(tlie)、分组($M$)、多通道并行($N$)对权重参数重排序,重排序后存储方式如图 14 所示. 最终的输出数据需存储到外部 DDR 中,若输出数据无序,则会增加访存次数,输出数据重排序后存储方式如图 15 所示.

权重数据重排序流程:①先存储分块数据(每块 tile_size 个数据);②块数据存储完成后,存储分组中的其他块数据;③由于多通道并行执行,分组数据存取完成后,存储其他通道中所需数据.

输出数据重排序流程:①先存储分组数据,每组一个数据,共存储 $M$ 个数据;②存储分组维度上的其他数据,共存储 Nof 个数据(Nof 为第三维维度总数).

权重数据、输出数据均以行优先方式进行存储,重复上述排序流程,直至所有数据均已存储完毕. 重排序后,一次读取多个连续存储的数据,增大传输数据长度,降低传输时延,减少访问内存次数,提高带宽利用率.

## 4.7　数据通路优化

### 4.7.1　片上数据传输

为减少部分和的数目和移动次数,尽可能展开

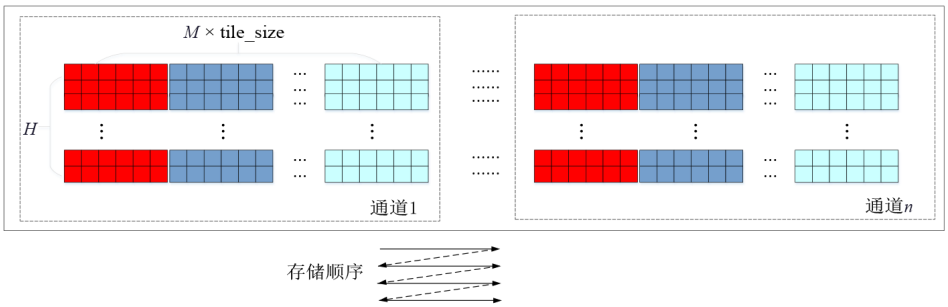



图 14 weight 重排序存储

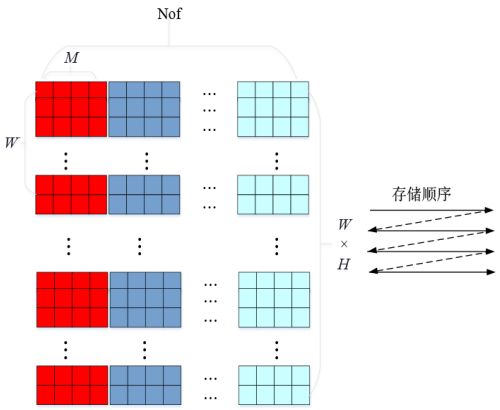



图 15 输出重排序存储

Loop-1 和 Loop-2. 串行计算 Loop-1 和 Loop-2，并确保 Loop-1 和 Loop-2 所需的数据得到缓存，通过计算 Loop-1 和 Loop-2 可以得到 $m\times m\times$Nof 个部分和. 这些部分和以最小的数据移动量保留在本地寄存器中，不会增加缓冲区访问和存储开销，因此，降低了片上缓冲区访问次数，减少了片上数据传输量.

#### 4.7.2 片外数据传输

当首先计算 Loop-1 和 Loop-2 以减少部分和时，其中一个卷积层的输入像素和权重都没有完全缓存，需多次访问 DRAM 来完成该层的卷积计算. 为降低 DRAM 的访问次数，即减少片外传输量，为每层的输入像素和权重分配足够容量大的缓冲区，将该层所需的数据全部存储.

Loop-4 的展开可以重用输入像素，降低 DRAM 输入像素缓冲区的访问次数. Loop-3 的展开可以重用权重，减少了 DRAM 权重缓存区的访问次数. 从 DRAM 取出的数据在被移出缓冲区之前得到了充分利用，在得到最终输出数据后，将数据写回 DRAM 的输出缓冲区，每个输出像素只写入 DRAM 一次，减少了片外数据传输量.

由于 DRAM 访问代价比块内存访问的延迟和 energy 都要高，故减少外部内存访问次数对提高整体性能和能效非常重要.

## 5 实验结果分析

### 5.1 软硬件实验环境

为验证设计方法的有效性，对所采取的加速方案进行板上测试. 选用的 FPGA 开发工具为 Xilinx 公司的 Vitis IDE 软件，选用的开发板为 U280，硬件开发环境为 Vivado 2019.2. CPU 平台为 Intel(R) Core(TM) i5-8500，频率为 3 GHz. 其中，高性能 U280 开发板具有 PCIe Gen3x16 连接性的 PCI Express 的 Xilinx DMA 子系统，拥有 8GB HBM2+32GB DDR4 内存，并支持第四代 PCIe 和 CCIX 互联标准.

### 5.2 性能分析

本文使用公开的 COCO 数据集，测试图片尺寸为 554×554×3，对 YOLO2 的 1D 模式 Winograd 算法 $F(4,3)$ 进行评估. 表 1 对比了 Winograd 卷积与普通卷积所需乘法次数. 依据该模型架构所采用的 YOLO2 网络参数，分别计算出不采用分块的卷积所需乘法数、采用分块的卷积所需乘法数以及采用分块的 Winograd 卷积所需乘法数，表 2 仅列出了部分层所需乘法数. 由表 2 知，采用分块、Winograd 卷积策略可以降低乘法数[18].

**表 1 采用 Winograd 算法降低复杂度**

| $F(4,3)$ | 乘法数 |
|---|---|
| Winograd algorithm | 6 |
| Standard algorithm | 12 |
| Arithmetic complexity reduction | 2x |

由于位宽限制，本文采用如下设置：通道 Lane_num =8，各通道内并行因子 Vec_size=8，表 3 为融合前和融合后各模块理论计算时间，理论计算时间的计算式如式(10)、式(11)和式(12)所示. 由表 3 可知，融合后比融合前所需时间明显减少. 通过模块融合，省去了卷积层、池化层的通道数据传输量，降低了数据传输时间，减少了整个模型的运行时间.

$$\text{TotalTime}=\sum_{i=1}^{M}\left(\begin{matrix}\text{ComputeTime}_i+\\ \text{TransmissionTime}_i\end{matrix}\right) \tag{10}$$

**表2　部分Conv层乘法数对比**

| | Input Feature | Output Feature | Kernel Size | Winograd algorithm 乘法数 | Standard algorithm 乘法数 | Non-blocking algorithm 乘法数 |
|---|---|---|---|---|---|---|
| Conv1 | 3 | 544×544 | 3×3 | 21307392 | 42614784 | 127844352 |
| Conv2 | 16 | 272×272 | 3×3 | 56819712 | 113639424 | 340918272 |
| Conv5 | 32 | 136×136 | 3×3 | 56819712 | 113639424 | 340918272 |
| Conv6 | 64 | 68×68 | 3×3 | 56819712 | 113639424 | 340918272 |
| Conv7 | 128 | 68×68 | 1×1 | 56819712 | 113639424 | 340918272 |
| Conv8 | 64 | 68×68 | 3×3 | 56819712 | 113639424 | 340918272 |
| Conv11 | 128 | 34×34 | 3×3 | 60162048 | 120324096 | 340918272 |
| Conv12 | 256 | 34×34 | 1×1 | 60162048 | 120324096 | 37879808 |
| Conv13 | 128 | 34×34 | 3×3 | 60162048 | 120324096 | 340918272 |
| Conv16 | 256 | 17×17 | 3×3 | 66846720 | 133693440 | 340918272 |
| Conv17 | 512 | 17×17 | 1×1 | 66846720 | 133693440 | 37879808 |
| Conv21 | 256 | 17×17 | 3×3 | 33423360 | 66846720 | 170459136 |
| Conv22 | 256 | 17×17 | 1×1 | 4177920 | 8355840 | 2367488 |
| Total | | | | 1125113856 | 2250227712 | 4448509952 |

$$\text{TransmissionTime}=\frac{\text{data}_t}{\text{bandwidth}} \tag{11}$$

$$\text{ComputeTime}=\frac{\text{data}_c}{f} \tag{12}$$

其中,TotalTime表示总运行时间,ComputeTime$_i$表示第$i$层所需的计算时延,TransmissionTime$_i$表示第$i$层数据传输时延,$M$表示CNN模型层数,data$_t$表示传输数据量,bandwidth表示带宽,data$_c$表示计算数据量,$f$表示频率.

**表3　融合前后时间对比**

| | 融合前 | | | | 融合后 | |
|---|---|---|---|---|---|---|
| 模块 | memRead | Conv | maxPool | memWrite | memRead | memWrite |
| 时间/s | 0.016 37 | 0.047 72 | 0.032 243 46 | 0.013 35 | 0.020 12 | 0.013 35 |
| 总时间/s | 0.109 683 46 | | | | 0.033 47 | |

本设计经综合、布局布线后,各模块资源占用情况如表4所示. 因memRead模块融合了conv模块和maxPool模块,故其占用资源较多. 表5为资源利用率情况,从表5可以看出,该板卡资源利用率均比较低,系统内仍有足够资源可以使用,可进一步优化程序或解除带宽限制以提升资源利用率. 以YOLO2模型为例,数据采用8位整数,实现了提议的高效能目标检测加速器,mAP降低了0.96%. 表6对比了不同平台上YOLO模型的性能,由表6知,本方案具有低延时,同文献[4,16,17]和Xilinx官网所给数据相比,该方案具有较高的性

**表4　各模块资源占用**

| 模块名 | FF | LUT | DSP | BRAM |
|---|---|---|---|---|
| memRead | 135 576 | 221 478 | 23 | 218 |
| memWrite | 20 220 | 31 079 | 27 | 3 |

**表5　资源耗费**

| 资源种类 | $N_{\text{已用}}$/个 | $N_{\text{可用}}$/个 | 利用率/% |
|---|---|---|---|
| FF | 155 796 | 2 607 360 | 5.98 |
| LUT | 252 557 | 1 303 680 | 19.40 |
| DSP | 50 | 9 024 | 0.55 |
| BRAM | 221 | 2 016 | 10.96 |

**表6　与其他文献的对比**

| | 文献[4] | 文献[14] | 文献[15] | 文献[16] | 文献[17] | Xilinx官网 | 本设计 |
|---|---|---|---|---|---|---|---|
| CNN模型 | Yolo | Yolo2 | Yolo2 | Yolo1 | Tiny-Yolo2 | Yolo2 | Yolo2 |
| FPGA板卡 | Pynq-z2 | ZCU102 | PYNQ-Z1 Zynq 7z020 | ZC706 | Cyclone V PCIe | U280 | U280 |
| 时钟频率/MHz | 125 | 300 | 200 | 200 | 117 | 300 | 400 |
| 权重精度 | fixed-16 | fixed-16 | - | fixed-32 | fixed-16 | fixed-8 | fixed-8 |
| 时间/ms | 124 | 288 | 611 | 74.39 | 339 | 66.20 | 27.04 |
| 性能/(GOP/s) | 24.17 | 102.2 | 48.23 | 18.82 | 21.6 | 56.65 | 249.65 |

能. 本文采用Winograd卷积降低乘法运算次数以提高计算时间,并使用模块融合优化策略降低传输时延,而Xilinx官网提供的实验数据仅执行了8位定点量化. Xilinx官网实验平台与本文采用的硬件平台一样,具有可比性,更能体现设计方案的优势. 对比Xilinx官网数据,本方案具有低延迟、高性能.

## 6 结论

本文提出了一种基于Winograd算法的高效能目标检测加速器,采用了模块融合策略、Winograd卷积算法,大幅降低算法复杂度,改善FPGA的CNN性能. 该算法采用缓冲区数组,减少片上数据访问次数,通过具有足够大的片上缓冲区和适当的循环计算顺序,使得每个像素和权重只需从DRAM传输一次,就可以实现最少的DRAM访问次数. 此外,该算法通过模块融合,降低了片外数据传输,有效提高了程序运行时间.

在识别准确率损失较小的情况下,可考虑对CNN模型剪枝、降低权重精度,进一步压缩模型大小,还可以考虑如何解除内存限制,采用更高的并行度.

### 参考文献


[1] REDMON J , FARHADI A . YOLO9000: Better, faster, stronger[C]// IEEE Conference on Computer Vision & Pattern Recognition. Honolulu: IEEE, 2017: 6517-6525.

[2] NAKAHARA H, YONEKAWA H, FUJII T, et al. A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA[C]//The 2018 ACM/SIGDA International Symposium. Monterey: ACM, 2018: 31-40.

[3] GUO K, ZENG S, YU J, et al. [DL] A survey of FPGA-based neural network inference accelerators[J]. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2019, 12(1): 1-26.

[4] BAO C, XIE T, FENG W, et al. A power-efficient optimizing framework fpga accelerator based on winograd for yolo [J]. IEEE Access, 2020, 8: 94307-94317.

[5] HUANG Y , SHEN J , WANG Z , et al. A high-efficiency fpga-based accelerator for convolutional neural networks using winograd algorithm[J]. Journal of Physics Conference Series, 2018, 1026: 012019.

[6] YANG A, LI Y, SHU H, et al. An opencl-based FPGA accelerator for compressed YOLOv2[C]//2019 International Conference on Field-Programmable Technology(ICFPT). Tianjin: IEEE, 2019: 235-238.

[7] SHI F, LI H, GAO Y, et al. Sparse Winograd Convolutional Neural Networks on Small-Scale Systolic Arrays [EB/OL]. (2018-10-03)[2020-12-01]. https://arxiv.org/abs/1810.01973.

[8] 武铮,安虹,金旭,等.基于Intel平台的Winograd快速卷积算法研究与优化[J]. 计算机研究与发展,2019,56(4): 825-835.
WU Z, AN H, JIN X, et al. Research and optimization of Winograd fast convolution algorithm based on Intel platform[J]. Computer Research and Development, 2019, 56 (4) ): 825-835. (in Chinese)

[9] 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.

[10] LIAN X, LIU Z, SONG Z, et al. High-performance FPGA-based CNN accelerator with block-floating-point arithmetic[J]. IEEE Transactions on Very Large Scale Integration(VLSI) Systems, 2019, 27(8): 1874-1885.

[11] XIAO Q, LIANG Y, LU L, et al. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs[C]//Design Automation Conference. Austin: IEEE, 2017: 1-6.

[12] ALWANI M, CHEN H, FERDMAN M, et al. Fused-layer CNN accelerators[C]//2016 49th Annual IEEE/ACM International Symposium on Microarchitecture(MICRO). Taipei: IEEE, 2016: 1-12.

[13] BI F, YANG J. Target detection system design and FPGA implementation based on YOLO v2 algorithm[C]//2019 3rd International Conference on Imaging, Signal Processing and Communication(ICISPC). Singapore: IEEE, 2019: 10-14.

[14] ZHANG S, CAO J, ZHANG Q, et al. An FPGA-based reconfigurable CNN accelerator for YOLO[C]//2020 IEEE 3rd International Conference on Electronics Technology (ICET). Chengdu: IEEE, 2020: 74-78.

[15] LU T Y, CHIN H H, WU H I, et al. A very Compact Embedded CNN Processor Design Based on Logarithmic Computing[EB/OL]. (2020-10-13) [2020-12-01]. https://arxiv.org/abs/2010.11686.

[16] ZHAO R, NIU X, WU Y, et al. Optimizing CNN-based object detection algorithms on embedded FPGA platforms [C]//International Symposium on Applied Reconfigurable Computing. Rennes: Springer, 2017: 255-267.

[17] WAI Y J, MOHD YUSSOF Z BIN, SALIM S I BIN, et al. Fixed point implementation of Tiny-Yolo-v2 using OpenCL on FPGA[J]. International Journal of Advanced Computer Science and Applications, 2018, 9(10): 506-512.

[18] 齐延荣. 基于FPGA的深度学习图像识别加速与优化研究[D]. 郑州: 郑州大学, 2021.
QI Y R. Research on Acceleration and Optimization of Deep Learning Image Recognition Based on FPGA[D]. Zhengzhou: Zhengzhou University, 2021.

## 作者简介

**李　斌**　男,1986年生,河南郑州人.主要研究方向为信息安全和高性能计算.
E-mail: iebinli@zzu.edu.cn

**齐延荣(通讯作者)**　女,1995年生,河南濮阳人.主要研究方向为图像处理和高性能计算.
E-mail: 2297149111@qq.com

**周清雷**　男,1962年9月生,河南郑州人.教授、博士生导师、中国计算机学会高级会员.主要研究方向为自动机理论、信息安全和计算复杂性理论方面.