

引用格式:郭谦,贺光辉. 基于 FPGA 的卷积神经网络硬件加速器设计空间探索研究[J]. 微电子学与计算机, 2020, 37(8): 66-71. [GUO Q, HE G H. Research on design space exploration of FPGA-based convolutional neural network hardware accelerator [J]. Microelectronics & Computer, 2020, 37(8): 66-71.]

基于 FPGA 的卷积神经网络硬件加速器 设计空间探索研究

郭 谦, 贺光辉

(上海交通大学 电子信息与电气工程学院, 上海 200240)

摘 要: 为了解决基于 FPGA 的卷积神经网络硬件加速器资源分配的问题, 提出一种基于细粒度流水线架构的设计空间探索方法. 为了提高吞吐率, 该方法主要使用了三种技术: 1) 通过对 DSP 进行多阶段分配, 实现各级流水线平衡; 2) 利用可调节的中间值缓存, 协调 BRAM 和 DDR 带宽资源; 3) 利用深度可分解卷积替换部分卷积层, 减少网络整体计算量. 为了验证提出的设计空间探索方法, 在 ZC-706FPGA 上实现了 YOLO2-tiny 网络, 结果表明与同类设计相比, 本设计的吞吐率与能效比高, 整体延时低.

关键词: 卷积神经网络硬件加速器; 设计空间探索; 细粒度流水线

中图分类号: TP183

文献标识码: A

文章编号: 1000—7180(2020)08—0066—06

Research on design space exploration of FPGA-based convolutional neural network hardware accelerator

GUO Qian, HE Guang-hui

(School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: In order to solve the problem of allocating resources for FPGA-based convolutional neural network hardware accelerator, a design space exploration method based on fine-grained pipeline architecture is proposed. In order to improve throughput, the method mainly uses three technologies: 1) achieve multi-stage allocation of DSP to balance the pipeline; 2) introduce adjustable intermediate result buffer to coordinate BRAM and DDR bandwidth resources; 3) utilize the depth decomposable convolution to replace part of the convolutional layers to reduce the overall computation of the network. In order to verify the proposed design space exploration method, the YOLO2-tiny network is implemented on the ZC-706 FPGA. Compared with the similar design, the results show that the throughput and energy efficiency of the design are relatively higher, and the overall latency is shorter.

Key words: CNN hardware accelerator; design space exploration; fine-grained pipeline

1 引言

近年来, 卷积神经网络(Convolutional Neural Network, CNN)在图像处理领域的性能不断提升^[1], 计算复杂度也在不断增加, 针对 CNN 的硬件加速器设计已成为重要的研究方向. 相比于图像处

理单元与专用集成电路, 现场可编程门阵列(Field-Programmable Gate Array, FPGA)功耗较低且具有可重构性, 被广泛应用于神经网络加速器的实现.

CNN 结构复杂多样, 不同层所需的计算量差距较大, 故针对不同的网络, 如何合理地为一层分配 FPGA 片上的计算、存储与带宽资源, 成为制约硬件

加速器整体性能的关键因素. 文献[2]针对设计中计算吞吐率与带宽不匹配的问题,提出一种基于 roof-line 模型的方法在多个设计方案中探索最优解,但该设计的整体吞吐率较低;文献[3]通过建立分析模型实现片上资源的负载均衡,使得吞吐率最大化.但是该方法需按时分复用的方式逐层进行计算,增加了整体计算延时. 文献[4]基于流式架构进行设计空间探索,通过为每级流水线合理分配资源实现流水线平衡. 但该设计无法对原始网络进行优化,不能进一步减少流水线计算时间. 针对这些问题,本文基于细粒度流水线架构,提出了一种设计空间探索方法,对 DSP 进行多阶段分配来平衡流水线;使用可调整的中间值缓存协调 BRAM 和 DDR 带宽;利用深度可分解卷积降低计算复杂度. 通过在 ZC-706 FPGA 上实现 YOLO2-tiny 网络并与同类设计相比,本设计的吞吐率与能效比高,整体延时低.

2 流式架构加速器设计及网络结构优化

2.1 细粒度流水线

流式架构是一种常见的神经网络加速器实现方式,通过将整个神经网络映射到 FPGA 上,以层级流水线的形式对各个卷积层和全连接层进行并行计算,从而增加整个网络的吞吐率.

如图 1(a)所示,在传统流水线中,下一层的卷积计算必须在上一层的计算完全结束后才可以进行,增加了整个网络的计算延时. 同时,这种计算模式需要对每一层的中间值进行存储,通常 BRAM 资源无法满足,必须将其存储到 DDR 中,增加了带宽需求和功耗. 为了解决该问题,本文采用了细粒度流水线的设计,使用中间值缓存,按列存储每一层的部分结果,一旦存满下一层一次卷积计算所需的输入数据,下一层卷积计算即可开始进行,从而缩短了整个网络的计算延时,具体如图 1(b)所示. 同时,中间值缓存只需存储少量的计算结果,可以用 BRAM 直接实现,无需再访问 DDR,减少了带宽需求与计算功耗.

2.2 中间值缓存与权重复用

在细粒度流水线设计中,卷积层计算的硬件架构如图 2 所示. 每一层卷积计算时从该层的中间值缓存中获取上一层的计算结果,从权重缓存中获取所需的权重,通过卷积计算单元计算完毕后,将输出结果按列存储在下一层的中间值缓存中.

该架构中的权重缓存利用卷积计算具有参数共享的特性,通过权重复用减少对 DDR 的访问,具体的复用次数可以通过中间值缓存进行调整. 图 3

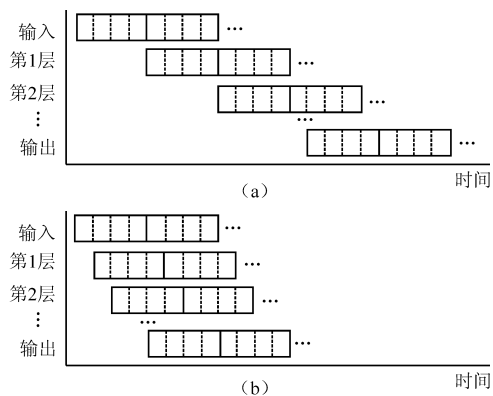


图 1 传统流水线与细粒度流水线示意图

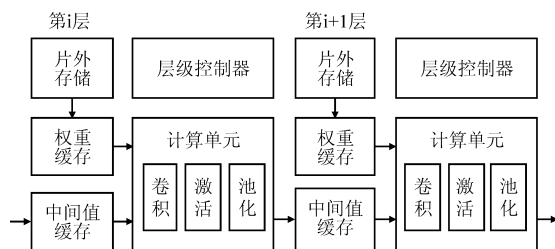


图 2 卷积层计算架构

为中间值缓存的示意图,其中空白列 Col_i^{use} 为前一层已经计算出的结果,阴影列 Col_i^{fill} 为前一层正在计算的结果. 以卷积核大小为 3×3 ,步长为 1 为例,当空白列的数目为 3 时,当前层的卷积计算即可开始进行. 当完成一次卷积计算后,卷积核先向下进行滑动,直到滑动到输入特征图的最底部,此时权重复用次数为输出特征图的高度 H_{out} . 若中间值缓存中空白列的数目为 4,则卷积核可以继续向右进行 1 次滑动,此时权重复用次数为 $2 \times H_{out}$. Col_i^{use} 和 Col_i^{fill} 的数量与列方向上权重复用次数 $Reuse_i$ 的关系为公式 (1) 与 (2) 所示:

$$Col_i^{use} = Kw_i + (Reuse_i - 1) \times Stride_i \quad (1)$$

$$Col_i^{fill} = Reuse_i \times Stride_i + Reuse_{i-1} - 1 \quad (2)$$

式中, Kw_i 为卷积核宽度, $Stride_i$ 为步长. 中间值缓存通过增加缓存列数提高权重复用次数,对 BRAM 资源和带宽资源进行协调,从而适应不同规模的卷积计算.

2.3 深度可分解卷积

在卷积神经网络中,卷积层通常占据整个神经网络 90% 以上的计算量,成为影响硬件加速器性能的关键. 为了减少卷积层的计算量,在 MobileNet 等网络中使用深度可分解卷积对正常卷积进行了替换,使得在保持网络准确度略有损失的基础上极大地降低了网络的计算量[5].

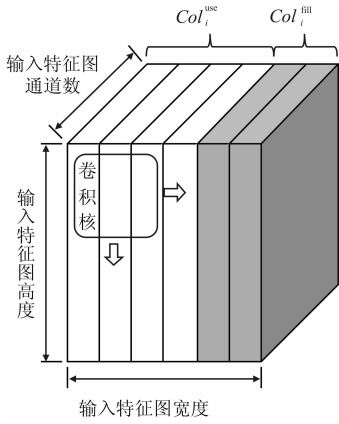
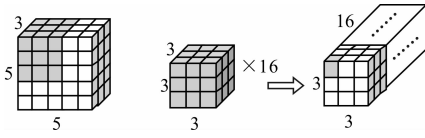
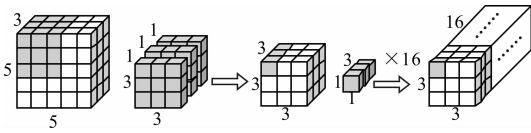


图 3 中间值缓存示意图



(a) 正常卷积计算模式



(b) 深度可分解卷积计算模式

图 4 正常卷积与深度可分解卷积计算示意图

图 4(a) 为正常卷积的计算流程, 通过卷积核在输入特征图上的滑动得到输出特征图上的值, 每得到一个输出均需要将卷积核中每个值与输入特征图中对应的元素相乘, 并将所有结果进行累加。图 4(b) 为深度可分解卷积的计算流程, 该操作将正常卷积分解为深度卷积和逐点卷积两部分, 其中深度卷积是对输入特征图的每个通道单独进行卷积计算, 逐点卷积是对深度卷积的结果做卷积核大小为 1×1 的正常卷积。假设输入特征图大小为 D_F 、通道数为 M , 卷积核大小为 D_K , 输出特征图大小为 D_P 、通道数为 N , 则深度可分解卷积的计算量 C_2 与正常卷积的计算量 C_1 的比值为:

$$\frac{C_2}{C_1} = \frac{D_P^2 \times D_K^2 \times M + D_P^2 \times M \times N}{D_P^2 \times D_K^2 \times M \times N} = \frac{1}{N} + \frac{1}{D_K^2} \quad (3)$$

3 设计空间探索

3.1 模型建立

设计空间探索是通过建立数学模型, 寻找满足约束条件下的目标函数最优解。在本文中, 片上

DSP、BRAM 以及 DDR 带宽资源是整个设计空间探索的约束条件, 加速器的吞吐率为优化目标。在流式架构中, 由于整个神经网络按照卷积层与全连接层划分为多级流水线, 整体性能受计算时间最长的层级限制, 因此如何通过资源分配对各级流水线进行平衡成为提高吞吐率的关键。同时, 为了减少网络整体计算量, 可以将 CNN 中部分层替换为深度可分解卷积, 通过一定的精度损失来对模型进行轻量化。因此, 本文的设计空间探索问题建模如下:

输入: CNN 模型每层计算量 C_i , FPGA 片上计算资源 DSP_{total} , 存储资源 $BRAM_{total}$, 带宽资源 BW_{total} , 精度损失阈值 T 。

输出: 每一层卷积计算的通道并行度 PC_i 与核并行度 PK_i , 中间值缓存的读宽度 $widthrd_i$ 、读深度 $depthrd_i$ 以及写宽度 $widthwr_i$ 。

$$\text{优化目标: } \min \left(\max \frac{C_i}{DSP_i} \right)$$

约束条件:

- (1) $\sum_{i=1}^n DSP_i \leq DSP_{total}$
- (2) $\sum_{i=1}^n BRAM_i \leq BRAM_{total}$
- (3) $\sum_{i=1}^n BW_i \leq BW_{total}$
- (4) $L \leq T$

式中, DSP_i 、 $BRAM_i$ 与 BW_i 为每级流水线分配的资源数量, L 为轻量化后模型的精度损失。

3.2 设计空间探索方法

设计空间探索方法的整体框架如图 5 所示, 通过对 DSP 资源的多阶段分配来实现流水线平衡。第一阶段分配时, 主要根据各级流水线的计算量 C_i 对 DSP_{total} 进行分配; 第二阶段分配时, 首先通过 DSP_i 分配卷积计算所需的并行度因子 PC_i 与 PK_i , 再对 $BRAM_{total}$ 和 BW_{total} 进行分配, 若分配成功, 则记录此时的 $\max \frac{C_i}{DSP_i}$; 否则, 选择 $\frac{C_i}{DSP_i}$ 最小的层级, 将其 DSP_i 减半, 重新进行该步骤; 第三阶段分配时, 若 $\max \frac{C_i}{DSP_i}$ 对应的层级为卷积层, 则尝试将其替换为深度可分解卷积, 并重新对 DSP_{total} 、 $BRAM_{total}$ 和 BW_{total} 进行分配。若网络的精度损失 L 小于阈值 T 且 $\max \frac{C_i}{DSP_i}$ 减少, 则执行替换操作并重复进行该步骤直至替换失败, 输出最终探索到的资源分配策略。下面针对框架中分配 DSP、分配并行度以及分配

RAM 和带宽三个步骤进行详细说明.

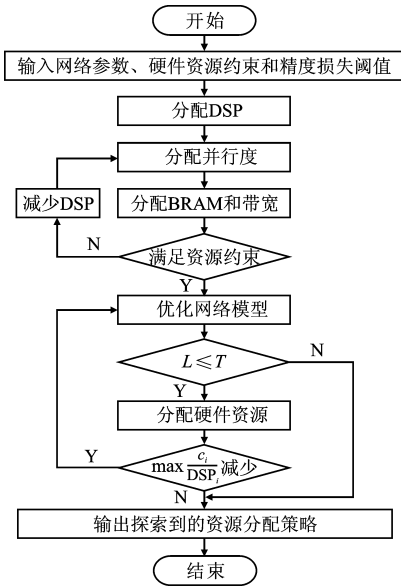


图 5 设计空间探索方法框架图

3.2.1 DSP 分配策略

为了平衡流水线,首先根据每层的计算量 C_i 对 DSP_{total} 进行分配使得各级的计算周期 $\frac{C_i}{DSP_i}$ 尽可能相等,并且将 DSP_i 调整为 2 的指数,以方便硬件实现.其次,若整个网络所需的 DSP 数量小于 DSP_{total} ,可以选择 $\frac{C_i}{DSP_i}$ 最大的层,若将其 DSP_i 翻倍后整个网络所需的 DSP 数量仍小于 DSP_{total} ,则再进行翻倍,从而缩短整体计算延时.具体流程如算法 1 所示.

算法 1: DSP 分配算法

输入:

CNN 每层的计算量 C_i ,总体计算量 C_{total}

DSP 资源总数 DSP_{total}

输出:

每级所需的 DSP 数量 DSP_i

1: 分配 DSP 使得每级计算所需周期 $\frac{C_i}{DSP_i}$ 相等

$$DSP_i = \frac{C_i}{C_{total}} \times DSP_{total}$$

2: 调整 DSP_i 为 2 的指数

$$DSP_i = 2^{\lceil \log_2 DSP_i \rceil}$$

3: while $\sum_{i=1}^n DSP_i \leq DSP_{total}$ do

4: 选择 $\frac{C_i}{DSP_i}$ 最大的层级 $i_1, i_2, \dots, i_m$

5: if $\sum_{i=1}^n DSP_i + \sum_{j=1}^m DSP_{i_j} \leq DSP_{total}$

6: for $j = 1; j \leq m; j++$ do

7: $DSP_{i_j} = 2 \times DSP_{i_j}$

8: end for

9: else

10: break

11: end if

12: end while

3.2.2 并行度分配策略

在卷积计算过程中,通道并行度 PC_i 决定了每个周期从中间值缓存中读取的输入数据数量,核并行度 PK_i 决定了每个周期从权重缓存中读取的卷积核数量.在一个周期中, PC_i 个输入数据与 PK_i 组卷积核进行乘加计算,其中每组卷积核具有 PC_i 个数据,所需的 DSP 数目 DSP_i 与 PC_i 和 PK_i 的关系为:

$$DSP_i = PC_i \times PK_i \quad (4)$$

为了在一个周期内从权重缓存中读出 DSP_i 个数据,权重缓存的读宽度需设为 $DSP_i \times DW$,其中 DW 为数据位宽,通常为 8 位或 16 位.由于在 Xilinx FPGA 中,18 K 大小的 BRAM 可配置的最大位宽为 36 位,为了满足读宽度需求,需要将多个 BRAM 进行拼接.由于在拼接起来的权重缓存中,相邻 BRAM 间无法相互利用空闲位,故权重缓存所需的最少 BRAM 数量 $BRAM_i^{wgt}$ 为 $\left\lceil \frac{DSP_i \times DW}{32} \right\rceil$.同时,若权重缓存可以存储相应层的所有权重,则不需设置为乒乓缓存,可以存储的卷积核数目 K_i 为输出通道数 C_{out_i} ;否则,权重缓存中仅有一半容量可以用来存储当前的权重, K_i 为 $\left\lceil \frac{BRAM_i^{wgt} \times 32 \times 256}{Kernel_i \times DW} \right\rceil$,其中 $Kernel_i$ 为卷积核的大小.通过 K_i 即可得到 PK_i 与 PC_i ,具体流程如算法 2 所示.

算法 2: 并行度分配算法

输入:

卷积层分配的 DSP 数 DSP_i

卷积核大小 $Kernel_i$,输出通道数 C_{out_i}

数据位宽 DW

输出:

卷积计算的通道并行度 PC_i 与核并行度 PK_i

1: 计算权重缓存所需的最少 BRAM 数

$$BRAM_i^{wgt} = \left\lceil \frac{DSP_i \times DW}{32} \right\rceil$$

2: 计算权重缓存一半容量可存储的卷积核数目

$$K_i = \left\lceil \frac{BRAM_i^{wgt} \times 32 \times 256}{Kernel_i \times DW} \right\rceil$$

3: 不需要乒乓缓存

4: if $2 \times K_i \geq C_{out_i}^{out}$

5: $K_i = C_{out_i}^{out}$

6: 需要乒乓缓存

7: else

```

8:    $K_i = K_i$ 
9: end if
10:  $PK_i = 2^{\lceil \log_2 K_i \rceil}$ 
11:  $PC_i = \frac{DSP_i}{PK_i}$ 

```

3.2.3 BRAM 和带宽分配策略

在 CNN 中,全连接层是访存密集型运算,需要占据大量带宽,故在带宽分配时需要先满足全连接层的需求.在分配卷积层带宽时,先不考虑中间值缓存在列方向上的复用,计算所需的带宽 BW_i^{conv} ,若所需带宽总和大于 BW_{total} ,则逐渐增加中间值缓存在列方向上的权重复用次数 $Reuse_i$,直到满足带宽约束条件或者超过可用的 BRAM 总量 $BRAM_{\text{total}}^b$.

具体流程如算法 3 所示.

算法 3:BRAM 与带宽分配算法

输入:

卷积层分配的通道并行度 PC_i
 每级分配的 DSP 数 DSP_i
 中间值缓存可用的 BRAM 总数 $BRAM_{\text{total}}^b$
 带宽资源总数 BW_{total}
 数据位宽 DW ,工作频率 f

输出:

中间值缓存读宽度 $width_i^{\text{rd}}$ 、读深度 $depth_i^{\text{rd}}$ 以及写宽度 $width_i^{\text{wr}}$

```

1: 计算一个 DSP 所需的带宽
    $BW_{\text{DSP}} = f \times DW$ 
2: 分配全连接层所需的带宽
    $BW_{fc_i} = DSP_i \times BW_{\text{DSP}}$ 
3: 计算卷积层可以使用的总带宽
    $BW_{\text{total}}^{\text{conv}} = BW_{\text{total}} - BW_{fc_i}^{\text{c}}$ 
4: 初始化每一层卷积的中间值缓存
    $Reuse_i = 1, width_i^{\text{rd}} = PC_i \times DW,$ 
    $depth_i^{\text{rd}} = \frac{H_i^{\text{in}} \times C_i^{\text{in}} \times (Kw_i + Stride_i)}{PC_i},$ 
    $width_i^{\text{wr}} = PK_{i-1} \times DW$ 
5: 初始化每一层卷积所需的带宽
    $BW_i^{\text{conv}} = \frac{DSP_i \times BW_{\text{DSP}}}{H_i^{\text{out}} \times Reuse_i}$ 
6: while  $\sum_{i=1}^n BW_i^{\text{conv}} \leq BW_{\text{total}}^{\text{conv}}$  do
7:   选择所需带宽最大的层  $j$ ,增加第  $j$  层与第  $j+1$  层中间值缓存的读深度:
8:    $depth_j^{\text{rd}} += \frac{H_j^{\text{in}} \times C_j^{\text{in}} \times Stride_j \times 2}{PC_j},$ 
    $depth_{j+1}^{\text{rd}} += \frac{H_{j+1}^{\text{in}} \times C_{j+1}^{\text{in}}}{PC_{j+1}}$ 
9:   if  $\sum_{i=1}^n f(width_i^{\text{rd}}, depth_i^{\text{rd}}) \leq BRAM_{\text{total}}^b$ 
10:     $Reuse_j = Reuse_j + 1$ 

```

```

11:    $BW_j^{\text{conv}} = BW_j^{\text{conv}} \times \frac{Reuse_j - 1}{Reuse_j}$ 
12: else
13:   break
14: end if
15: end while

```

4 实验结果与分析

4.1 实验环境

为评估本文中设计空间探索方法的性能,选用 Cifar10、AlexNet、VGG-16 以及 YOLO2-tiny 网络进行测试,其中 Cifar10 为含有 7 层卷积层的全卷积网络.同时,为了公平地与文献[4]中的实验结果进行比较,将 YOLO2-tiny 的第 7 层与第 8 层卷积的卷积核数量更改为 512.本文所选用的硬件平台包括 Xilinx ZC-706 FPGA 以及一块 1GB DDR3 SODIM 存储器,片上 DSP 总数为 900, BRAM 总数为 545.

4.2 实验结果与分析

本设计在硬件平台的约束下,探索到的资源分配结果如表 1 所示,其中工作频率设置为 200 MHz,数据位宽为 8 bit.可以看出四种网络的 DSP 使用率均在 67% 以上,带宽使用率均在 85% 以上,从而有效地利用了板上资源.为了衡量流水线是否得到有效平衡,文献[4]使用了 DSP 效率来进行评估,其定义为:

$$DSP_{\text{efficiency}} = \frac{Throughput}{\beta \times DSP_{\text{used}} \times f} \quad (5)$$

式中,分子表示加速器实际达到的算力,分母是理论上所使用 DSP 的算力总和. β 表示一个周期内每个 DSP 可以执行的操作数,对于 16 位定点数, β 为 2;对于 8 位定点数, β 为 4.从表 1 中可以看出,除了 AlexNet 以外,其余 3 种网络的 DSP 效率均在 93% 以上,验证了该设计空间探索方法对流水线平衡的有效性.

表 1 设计空间探索结果

网络	DSP 使用量	BRAM 使用量	带宽使用率 (%)	流水线计算时间 (ms)	DSP 效率 (%)
Cifar10	744	97	85.71	0.02	93.79
AlexNet	753	127	98.70	2.95	79.92
VGG-16	647	268	86.60	8.01	95.40
YOLO2-tiny	610	189	99.66	22.12	98.58

利用设计空间探索的结果,可以配置硬件模板的参数.为验证本设计总体的性能,选用 YOLO2-ti-

ny 网络进行硬件实现,结果如表 2 所示.

表 2 与同类设计比较结果

	本文	文献[4]	文献[6]	文献[7]
硬件平台	ZC-706	ZC-706	Cyclone V PCIe	VC-707
工作频率(MHz)	200	200	117	200
数据位宽(bit)	8	16	16	8
DSP 使用量	610/900	680/900	122/300	1024/3600
BRAM 使用量	245/545	333/545	—	—
整体延时(ms)	30	59	—	65
吞吐率(GOPs)	424	234.0	21.6	461.5
能效比(GOPs/w)	47	—	—	35.3
DSP 效率(%)	86.9	86.0	75.7	56.3

可以看出在相同的资源约束下,与其它针对 YOLO2-tiny 网络的加速器相比,本文中提出的设计方法整体延时低,吞吐率与能效比高,优于同类设计.同时,与表 1 中结果相比,DSP 使用数目相同;BRAM 由于在权重缓存与中间值缓存之外仍有使用,故略高于探索结果;整体延时由于需要考虑细粒度流水线的填充时间,略长与探索结果;DSP 效率由于整体延时的差异,略低于探索结果.

从结果对比可以得出,该设计空间探索方法可以有效地平衡流水线,从而实现吞吐率的最大化.同时,探索结果与硬件实现结果的误差较小,说明该探索结果对实际加速器的设计具有一定的指导意义.

5 结束语

本文基于细粒度流水线架构,提出了一种设计空间探索方法,该方法通过对 DSP 的多阶段分配实现流水线平衡;利用可调整的中间值缓存协调 BRAM 与 DDR 带宽;通过深度可分解卷积轻量化原始网络,减少整体计算量.该方法通过对片上资源的合理分配,实现计算吞吐率的最大化.通过该方法的探索结果配置硬件模板,在 ZC-706 FPGA 上实现了 YOLO2-tiny 网络,所设计的硬件加速器吞吐率为 424 GOPs,能效比为 47 GOPs/w,整体延时为 30 ms.

参考文献:

[1] ZHANG S, HE G, CHEN H B, et al. Scale adaptive proposal network for object detection in remote sensing images[J]. IEEE Geoscience and Remote Sensing Letters, 2019, 16(6): 864-868.

[2] ZHANG C, LI P, SUN G, et al. Optimizing fpga-based accelerator design for deep convolutional neural networks[C]// 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey: ACM/SIGDA, 2015:161-170.

[3] MOTAMEDI M, GYSEL P, AKELLA V, et al. Design space exploration of FPGA-based deep convolutional neural networks[C]// 2016 Asia and South Pacific Design Automation Conference. Macao: IEEE, 2016: 575-580.

[4] ZHANG X, WANG J, 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. San Diego: IEEE, 2018: 1-8.

[5] HOWARD A G, ZHU M, CHEN B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[6] WAI Y J, BIN MOHD YUSSOF Z, BIN SALIM S I, 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.

[7] YU J, GUO K, HU Y, et al. Real-time object detection towards high power efficiency[C]// 2018 Design, Automation & Test in Europe Conference & Exhibition. Dresden: IEEE, 2018: 704-708.

作者简介:

郭 谦 男,(1995-),硕士研究生.研究方向为神经网络算法与硬件加速器实现. E-mail:919828045@qq.com.

贺光辉 男,(1980-),博士,副教授.研究方向为无线通信系统算法和高能效电路设计与深度神经网络硬件加速器设计.