一种基于 Zynq 的 CNN 加速器设计与实现

许 杰,张子恒,王新宇,佟 诚,梅 青,肖 建*
(南京邮电大学 电子与光学工程学院、微电子学院,江苏 南京 210023)

摘 要:卷积神经网络是一种前馈神经网络,它的人工神经元可以响应部分覆盖范围内的临近单元,对于大型图像处理有出色表现。文中设计了一种基于 Zynq 芯片的 CNN 加速器,以期在资源和功耗受限的 FPGA 中实现运算性能加速。该加速器采用数据量化的方式将网络参数从 64 位双精度浮点数转化为 16 位定点数;针对 CNN 不同层的特性和要求,设计了不同的网络结构和优化策略。卷积层和全连接层采用循环分块、循环流水及循环展开等方法进一步改进,而池化层采用流水线的优化方式。亦设计了 FPGA 和外部存储器的缓存策略,减少 FPGA 和外部存储器的数据传输量。以 CIFAR-10 数据集下的图像识别为例,在 Zynq7020 实验平台上进行板级测试,实验结果表明,100 MHz 的工作频率下,平均识别时间为 15.5 ms,相对于单核 CPU 方案实现了 144 倍的加速。

关键词:Zynq;卷积神经网络;硬件加速;现场可编程逻辑门阵列;数据量化;CIFAR-10
中图分类号:TP39 **文献标识码**:A **文章编号**:1673-629X(2021)11-0108-06
doi:10.3969/j.issn.1673-629X.2021.11.018

Design and Implementation of CNN Accelerator Based on Zynq

XU Jie,ZHANG Zi-heng,WANG Xin-yu,TONG Cheng,MEI Qing,XIAO Jian*
(School of Electronic and Optical Engineering,School of Microelectronics,Nanjing University of Posts and Telecommunications,Nanjing 210023,China)

Abstract:Convolutional neural network is a feed-forward neural network whose artificial neurons can respond to neighboring units within partial coverage and perform well in large-scale image processing. A CNN accelerator based on the Zynq chip is designed to accelerate the computing performance in the FPGA with limited resources and power consumption. The accelerator uses data quantization to quantify network parameters from 64-bit double-precision floating-point numbers to 16-bit fixed-point numbers. According to the characteristics and requirements of different layers of CNN,different network structures and optimization strategies are designed. The convolutional layer and the fully connected layer are further improved by the methods of loop tiling,loop pipeline and loop unrolling,and the pooling layer uses the pipeline optimization method. A cache strategy for FPGA and external memory is designed to reduce the amount of data transfer between FPGA and external memory. Taking image recognition under the CIFAR-10 data set as an example,a board-level test was performed on the Zynq7020 experimental platform. The experiment shows that the average recognition time is 15.5 ms at a working frequency of 100 MHz,which is 144 times faster than the single-core CPU solution.

Key words:Zynq;convolutional neural network;hardware acceleration;FPGA;data quantification;CIFAR-10

0 引 言

卷积神经网络(convolutional neural network,CNN)广泛应用在图像识别[1]、目标检测[2]等领域。CNN 所需的计算量和数据量巨大,单核数量和访存带宽有限的中央处理器(central processing unit,CPU)不能满足其吞吐量和存储量的设计要求,研究人员将 CNN 加速器的实现主要集中在图形处理器(graphics processing unit,GPU)、专用集成电路[3-4](application specific integrated circuit,ASIC)芯片和现场可编程逻辑门阵列[5-8](field programmable gate array,FPGA)等三类器件上。

GPU 可以提供较高的并行度、矩阵计算和浮点运算能力,但巨大的功耗消耗使其难以应用在功耗与面积受限的应用平台。ASIC 能够在特定功能上进行强化,具有更高的处理速度和更低的能耗。但是研发成本高,前期研发投入周期长、灵活性较差。FPGA 比特级细粒度定制的结构、流水线并行计算的能力和高效的能耗,使得其在 CNN 加速器的研究中具有极大的

收稿日期:2020-12-09 **修回日期**:2021-04-13
基金项目:国家自然科学基金面上项目(61974073)
作者简介:许 杰(1995-),男,硕士研究生,研究方向为电路与系统;通讯作者:肖 建,教授,研究方向为人工智能和嵌入式的研究与应用。

优势。

同时,FPGA 是半定制的硬件,通过编程可重定义其内部配置和链接,具有较大的灵活性。

近年来,越来越多的 CNN 加速器设计被提出。Zhang 等人[9]提出了一种基于 roofline 模型在 FPGA 上的加速卷积神经网络。Qiu 等人[10]提出了一种 FPGA 设计来加速嵌入式系统上的大规模图像分类挑战的 CNN。苏黎世联邦理工学院团队基于 SqueezeNet[11]模型训练出了适合在 FPGA 上运行的 ZynqNet 网络,并针对网络实现了 FPGA 硬件加速[12]。Li 等人[13]提出了一种端到端的基于 FPGA 的 CNN 加速器,将所有的层都映射到片上,各层之间流水化实现,但只适合资源较多的 FPGA。

文中先对 Caffe[14]神经网络框架范例中的 cifar10_quick[15]网络做微小的调整,然后设计相应的硬件加速器,并对每层的硬件加速策略做详细的阐述。最后,在 Xilinx 的 Zynq7020 实验平台上进行评估。

1　网络模型简介

1.1　CIFAR-10 数据集

CIFAR-10[16]数据集由 10 类 32×32 的彩色图片组成,一共包含 60 000 张图片,每一类包含 6 000 图片。其中 50 000 张图片作为训练集,10 000 张图片作为测试集。

1.2　网络结构简介

cifar10_quick 模型出自 Caffe 神经网络框架中关于 CIFAR-10 数据集的样例,由 3 层卷积层、3 层池化层和 2 层全连接层构成,输入 32×32×3 的图像数据,输出 10 类物体分类的结果。选择其中数值最大的结果输出,即为最终分类的结果。cifar10_quick 模型 3 个池化层对应的步长为 2,池化窗口的大小为 3×3,训练 5 000 个批次,达到的准确率为 75% 左右。文中保持 3 个池化层的步长为 2 不变,池化滤波器的大小修改为 2×2,训练后准确率达到 80%。图 1 中展示了修改后 cifar10_quick_v1 模型的网络结构。

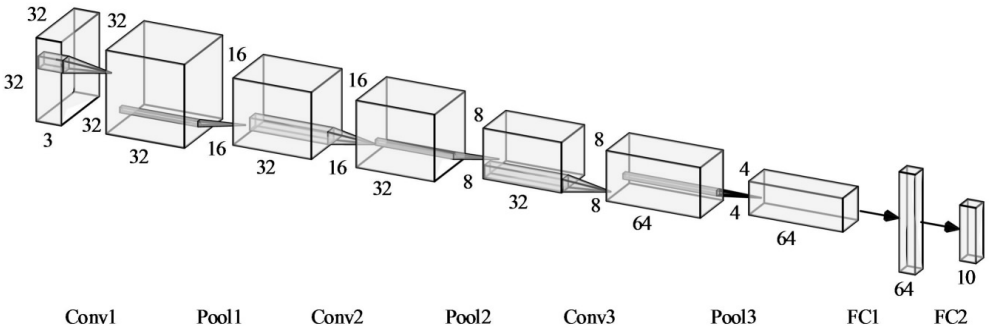

图 1　cifar10_quick_v1 网络结构

1.3　网络模型参数量化

CNN 模型训练完毕后,可以提取相应的权重和偏置。采用 GPU 进行训练,默认保存数据的类型为双精度浮点数,每一个数据占用 64 bit 的位宽。在 FPGA 中,逻辑资源有限,故使用定点数表示相应的权重、偏置以及输入特征及输出特征。

首先,分析网络参数的范围,确定最终的量化位数,保证量化的数据不会比原有数据失真太多。然后,确定浮点数包含的位数,即量化精度,量化后的整数范围不能比原有数据小,同时尽可能地提升浮点数所能表示的精度。值得注意的是,最高位为符号位,量化的定点数要支持负数。所有数据的量化需满足下列的公式(1)[17]:

$$2^{W-P-1} > \max(|D_{\max}| \times 2^{P}, |D_{\min}| \times 2^{P}) \quad (1)$$

其中,P 为量化精度,W 为量化位宽,D 表示量化前的浮点数据。

文中采用 16 位定点数进行存储,相比于 GPU 的 64 bit 的双精度浮点数,可以减少 3/4 的存储量。浮点数转换为定点数极大地节省了逻辑资源,相较于浮点运算,定点运算在速度上具有明显优势。

2　CNN 加速器优化

一个卷积神经网络由卷积层、池化层、全连接层以及非线性函数等四部分组成,还可能包括局部响应归一化[18](local response normalization,LRN)层和批归一化[19](batch normalization,BN)层,以及 Dropout[20]层等。而卷积层占据了 90% 的计算量[21],全连接层占据了大量的参数量,故将重点放在卷积层和全连接层的优化上,池化层作为整个卷积神经网络的一部分,亦在优化范围内。

2.1　软硬件协同设计框架

cifar10_quick_v1 模型是在 Zynq 芯片中实现的,而 Zynq 芯片主要由两部分组成:双核 ARM Cortex-A9 构成的处理系统(processing system,PS)和等价于一片 FPGA 的可编程逻辑(programmable logic,PL)。值得注意的是,Zynq7020 芯片属于 Zynq-7000 系列的低端产品,FPGA 逻辑资源有限,不适合将整个卷积神经网络映射到 Zynq 的 PL 端进行加速[13]。

如何充分利用 Zynq 的 PS 端的双核处理器系统与 PL 端的 FPGA 逻辑进行协同设计,就显得非常重要。FPGA 工作频率虽然不高,却有着极大的并行化处理优势,将加速的部分部署在 PL 端,能够很大程度上加速卷积神经网络的运行。PS 端工作频率高,与 PL 端高度的并行化处理相比,在计算上却没有明显的优势,其优势在于强大的管理和调度能力。

软硬件协同设计框架如图 2 所示。卷积神经网络的权重和偏置及输入特征存储在外部存储器 DDR3 中,加速时 Zynq 的 PS 端将各层的权重和偏置以及输入特征由 DDR3 加载给 PL 端,PL 端进行各层的硬件加速,计算完毕后,再将各层运算的结果输出到 DDR3。按照先后顺序,直到最后一层计算完毕,输出最终的运算结果。

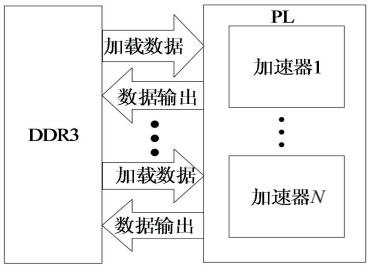

图 2 软硬件协同设计框架

由此可知,整个卷积神经网络的优化涉及两个方面,一是各层之间数据传输优化,减少数据传输的时间,二是各层内部计算的优化,主要是优化计算速度,减少 PL 端和 DDR3 内存之间的数据交互。

2.2 池化层优化

池化(也称为子采样)用于减少特征图的空间尺寸,从而在网络中减少参数和计算量。定期插入池化层在连续的卷积层之间,其独立于每一个输入的深度切片,并在特征图上求最大值调整输入特征的大小。最常见的形式是过滤器大小为 2×2 的池化层,每一次滤波器的移动最多可获取 4 个样本从而丢弃了 75% 的输入数据。除了常用的最大池化外,一些 CNN 也用于执行其他功能,例如平均池化和最小池化等[22]。

若无优化,池化层的实现,不论是取数据的过程,还是计算的过程,都是顺序执行。而 FPGA 逻辑资源有限,故只采用流水线的方式加速池化层的运行。

关于流水线的优化方式,以最大池化为例,池化层的滤波器大小为 2×2,需选取滤波器对应的 4 个数据中最大的数据。具体地,先取滤波器中的两个数据,选取较大的数据,暂存为 tmp_max1,再取剩余的两个数据,选取较大的数据,暂存为 tmp_max2,tmp_max1 和 tmp_max2 进行比较,较大的数据即为最终的结果。从两个数据的读取、比较到临时结果的暂存,是一个连续的过程。若未采用流水线,剩余的两个数据的运算过程,必须等待前两个数据的运算全部执行完毕后,才能开始执行。若采用流水线,前两个数据的读取完毕后,紧接着就是后两个数据的读取过程,同时前两个数据进行比较。下一个时钟周期,后两个数据的比较与前两个比较结果的暂存同步执行。直到算出最终的结果,后两个数据的运算过程与前两个数据的运算过程相比,只有一个时钟周期的间隔,极大地增加了运算速度。

池化层内部的计算采用流水线的方式实现,池化层与全连接层之间的数据传输,也需要优化。若池化层单独进行优化,则卷积层计算完毕后,将输出特征暂存到 DDR3,池化层再将该输出特征加载到 Zynq 的 PL 端,池化层优化计算后,输出结果到 DDR3。运算过程中没有很复杂的运算,却增加了数据存取过程,花费了大量的时间。若卷积层和池化层合并,即卷积层运算的结果直接传输到池化层参与运算,减少 DDR3 与 PL 端的数据交互,速度上会有很大的提升,如图 3 所示。

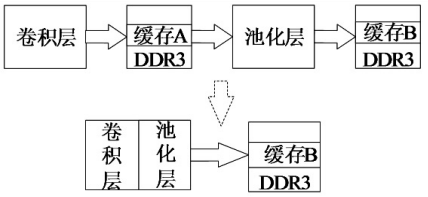

图 3 池化层缓存优化示意图

2.3 卷积层优化

卷积层的计算如图 4 所示。卷积层接收 CHin 个特征图作为输入,每个输入特征图由一个带有 $K \times K$ 大小的核的移动窗口卷积以在一个输出特征图中生成一个像素。移位窗口的步幅为 S(通常为 1,小于 K),总共 CHout 个输出特征图将形成下一个卷积层的输入特征图集[9]。

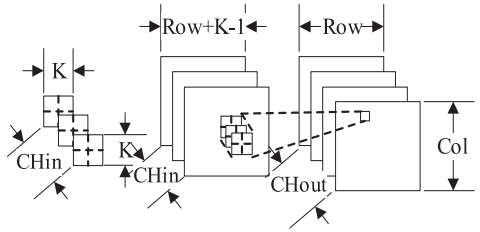

图 4 卷积层计算框图

卷积层的计算可用公式(2)表示:

$$\mathrm{Out}[\mathrm{cho}][r][c] = \sum_{\mathrm{chi}=0}^{\mathrm{CHin}-1}\sum_{kr=0}^{K-1}\sum_{kc=0}^{K-1} W[\mathrm{cho}][\mathrm{chi}][\mathrm{kr}][\mathrm{kc}] \times \mathrm{In}[\mathrm{chi}][r+\mathrm{kr}][c+\mathrm{kc}] \tag{2}$$

其中,Out 是输出特征变量,W 是输入权重变量,In 是输入特征变量,Row 是输出特征对应的行数,Col 是输

出特征对应的列数, chi 是输入通道数 CHin 对应的变量,cho 为输出通道数 CHout 对应的变量,kr 和 kc 分别是权重滤波器的行数或列数 K 对应的变量, r 是输出特征的行数 Row 对应的变量, c 是输出特征的列数 Col 对应的变量。

2.3.1　整体框架

卷积层优化的整体框架如图 5 所示。每一层卷积层的计算,都是 Zynq 的 PS 端将 DDR 存储的输入特征、权重以及偏置加载到 PL 端,PL 端加速卷积层的运算,计算的结果作为池化层的输入特征,池化层计算完毕后,再将最终的结果输出到 DDR3 中暂存。池化层从 4 个数据中提取出一个数据,故从卷积层的计算结果,到池化层的输出特征,可以节省 3/4 的内存空间,同时减少 DDR3 和 PL 端数据交互的时间。

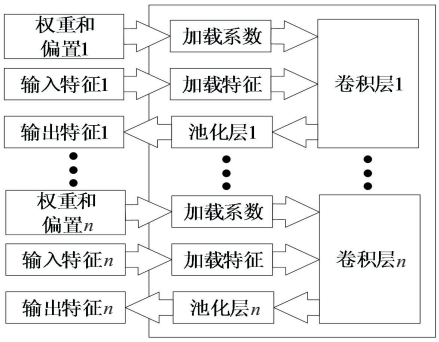

图 5　卷积层优化整体框图

2.3.2　卷积层输入输出通道分块策略

卷积层的计算量是最大的,充分利用 FPGA 并行化处理的优势能够加速卷积的运算。FPGA 的逻辑资源有限,仅将输入输出通道分块处理,同时分块后的输入输出通道的卷积计算并行化实现。

输入输出通道分块,则分块后的每次卷积运算不需要所有的权重、偏置以及输入特征参与。所有的输入数据可以按照输入输出通道分块,根据分块的大小加载部分的权重、偏置以及输入特征到 Zynq 的 PL 端。另一方面,若权重和偏置以及输入特征,完全按照输入输出通道分块,分块后各个分块大小所包含的数据量可能非常小,就会增加 DDR3 和 PL 端的数据交互次数,极大地增加 FPGA 访问外部存储器的时间。

因此,从 DDR3 加载权重、偏置以及输入特征时,仅按照输入通道分块读取。如图 6 所示,输入通道数用 CHin 表示,以 ITILE 作为输入通道分块的基数,则输入通道分块的大小 n 为 CHin/ITILE。首先,输入偏置的数据量与输出通道数一致,一次性将所有的输入偏置加载到 Zynq 的 PL 端;其次,输入特征按照输入通道分块,分步加载输入特征;最后,输入权重仅按照输入通道分块,所有输出通道对应的权重都一次性加载到 PL 端。也即每次加载到 PL 端的输入数据只有输入通道上不完整,将当前输入通道分块的所有输出通道的卷积计算(在 Conv 函数中实现)完毕后,再加载下一个输入通道分块的数据,直到所有的输入通道的数据加载并计算完毕,将所有输入通道分块卷积计算的结果累加,就可以获得最终的卷积运算的结果。一层卷积层的卷积运算完毕后,才将最终的输出特征传输到 DDR3,亦减少了 FPGA 和外部存储器的数据交互时间。

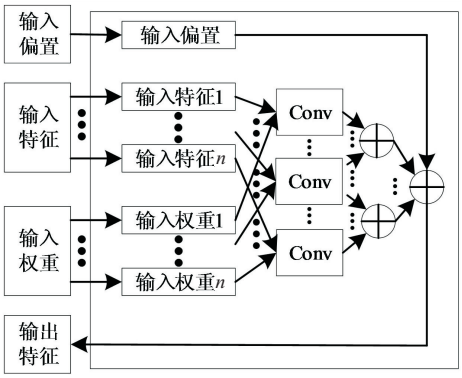

图 6　卷积层加载输入数据示意图

2.3.3　卷积层输入权重重组

为了保证输入权重从 DDR3 加载到 Zynq 的 PL 端的地址变化的连续性,减少 DDR3 地址的跳变,引入权重重组的策略,实施过程如图 7 所示。默认情况下,权重的加载过程是先按照第一个输出通道对应的输入通道加载数据,该输出通道对应的所有输入通道的权重加载完毕后,切换下一个输出通道的权重,直到所有输出通道的权重加载完毕。然而,文中仅按照输入通道分块加载权重,先计算部分输入通道对应的所有输出通道卷积运算的结果,故 DDR3 地址的变化过程必然不是连续的。为了减少 DDR3 地址的跳变,权重进行重新排序,即先将所有输出通道对应 ITILE 个输入通道的权重加载到 PL 端,然后加载下一部分所有输出通道对应的 ITILE 个输入通道的权重,直到所有的输入输出通道的权重加载完毕。

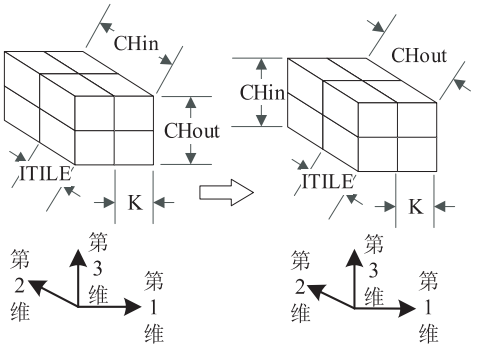

图 7　卷积层权重重组示意图

2.3.4　卷积层输入输出通道分块并行化

输出通道分块仅体现在卷积的计算上,输出通道数为 CHout,以 OTILE 作为输出通道分块的基数,则

输出通道分块的大小 m 为 CHout/OTILE。不同输出通道的卷积计算完全是独立的,计算所需的权重在不同输出通道完全是不同的,将不同输出通道分块的卷积计算完毕即可,无需将不同输出通道的结果进行累加。

输入输出通道分块后的计算如图 8 所示。分块后,共有 ITILE 个输入通道,OTILE 个输出通道,ITILE 个输入特征和 ITILE 个权重乘积运算并行,再将 ITILE 个乘积运算的结果累加,该过程对于 OTILE 个输出通道是完全并行的。对于每一个输出点的 ITILE 个输入通道,每一个输入通道权重滤波器内 $K * K$ 个元素与输入特征一一对应。将不同输入通道的权重滤波器内元素与对应输入特征元素的乘积累加(采用流水线的优化方式),就可以获得一个输出点的运算结果,最终会有 OTILE * Row * Col 个结果输出。

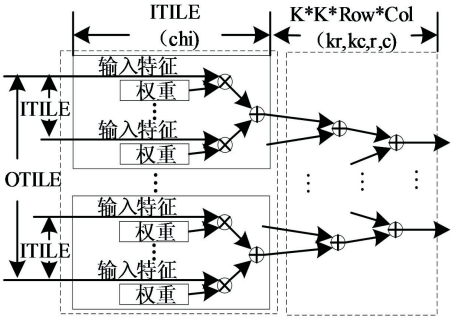

图 8 卷积层分块并行化示意图

2.4 全连接层优化

全连接层的计算比卷积层的计算简单,只需将一个输出通道的不同输入通道的权重与对应的输入特征相乘,得到的所有乘积累加,再加上对应输出通道的偏置,就能得到该输出通道的结果。

全连接层的计算用公式(3)表示:

$$\mathrm{Out}[\mathrm{fco}] = \sum_{\mathrm{fci}=0}^{\mathrm{FCin}-1} W[\mathrm{fco}][\mathrm{fci}] \times \mathrm{In}[\mathrm{fci}] \qquad (3)$$

其中,Out 是输出特征变量,W 是输入权重变量,In 是输入特征变量,FCin 是输入通道数,fci 是输入通道数 FCin 对应的变量,fco 是输出通道数对应的变量。

2.4.1 全连接层输出通道分块策略

全连接层的不同输出通道的计算所需的权重不同,输入特征却完全相同,即不同输出通道全连接层的计算是相互独立的,可以并行化实现。然而,所有输出通道的并行化计算,所需的 FPGA 逻辑资源是巨大的,故借鉴卷积层的分块策略,每次只计算部分输出通道的结果。

将 DDR3 存储的全连接层对应的输入特征,一次性全部加载到 PL 端。输入偏置亦是如此,不过加载到 PL 端后,按照输出通道分块进行重组。输入权重则不同,按照输出通道分块进行数据的加载,每次只加载部分输出通道的数据。计算完一个输出通道分块后,再加载下一个输出通道分块的权重,直到所有的输出通道计算完毕。每一个输出通道的全连接层的计算是相互独立的,故每个输出通道分块的计算(在 FC 函数中实现)也是相互独立的,所有输出通道分块计算完毕后,整合所有输出通道的结果,再将最终的输出特征输出到 DDR3,亦减少了 FPGA 与 DDR3 的数据交互时间,如图 9 所示。

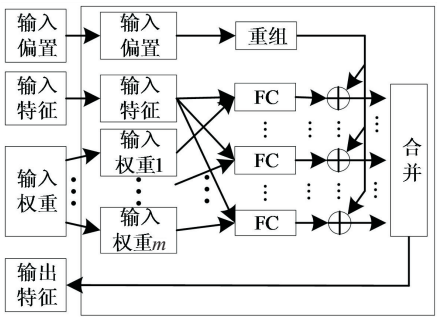

图 9 全连接层加载输入数据示意图

2.4.2 全连接层输出通道分块并行化

全连接层输出通道分块的并行化实现和卷积层输入输出通道分块的并行化实现类似,区别在于并行化的 FTILE 个输出通道中,只有一个输入通道的输入特征和权重的乘积运算是并行化实现的,各输入通道之间运算的切换采用流水线的优化方式。即 FTILE 个输出通道中,第一个输入通道的乘积运算计算完毕后,接着下一个输入通道进行乘积运算,直到所有的输入通道计算完毕。将所有输入通道的结果相加,总共有 FTILE 个输出结果,对应于 FTILE 个输出通道。全连接层的输出通道数用 FCout 表示,则输出通道分块的大小为 FCout/FTILE。

前文分别对卷积层、池化层、全连接层进行了优化,整个卷积神经网络优化完毕。

3 性能评估

3.1 实验环境

本设计在 Xilinx Zynq7020 平台中进行验证,双核 Cortex-A9 的 PS 端和 Zynq 的 PL 端都工作在 100 MHz 的频率下。Zynq 的 PS 端从 SD 卡中读取 CIFAR-10 数据集的测试集作为测试数据,10 000 张图片分张进行测试,CNN 加速器在不同的 OTILE、ITILE 和 FTILE 的参数条件下进行测试。

3.2 性能对比

共 4 组参数进行实验对比。软件优化时,使用单核 Cortex-A9 进行测试,其他组采用不同 OTILE、ITILE 和 FTILE 的参数配置进行对比。记录各层优化后的平均计算时间和加速比,生成的表格如表 1 所示。

表 1　不同参数配置下性能对比

	软件优化 单核 Cortex-A9	OTILE=4, ITILE=4, FTILE=2	OTILE=8, ITILE=4, FTILE=4	OTILE=8, ITILE=8, FC 层不加速
Conv1+Pool1/s	0.555 6	0.005 9	0.003 8	0.003 8
Conv2+Pool2/s	1.118 8	0.009 9	0.005 8	0.003 7
Conv3+Pool3/s	0.549 7	0.006 9	0.004 8	0.003 8
FC1/s	0.011 4	0.001 4	0.001 0	0.011 4
FC2/s	0.000 1	0.000 0	0.000 0	0.000 0
总时长/s	2.235 6	0.024 0	0.015 5	0.022 0
加速比	1.00	93.15	144.23	101.61

3.3　资源使用情况

不同 OTILE、ITILE 和 FTILE 的参数配置进行硬件优化,会有不同的资源占用情况与之对应,制成的表格如表 2 所示。

表 2　不同参数配置下资源占用情况对比　　%

优化方式	LUT	LUTRAM	FF	BRAM	DSP
OTILE=4, ITILE=4, FTILE=2	69	15	38	43	37
OTILE=8, ITILE=4, FTILE=4	74	16	39	58	45
OTILE=8, ITILE=8, FC 层不加速	69	10	29	71	57

4　结束语

在 Zynq7020 实验平台中对 cifar10_quick_v1 整个 CNN 模型进行设计,通过对卷积层输入输出通道分块及并行化设计,对池化层流水线优化,以及全连接层输出通道分块及并行化设计,充分利用 FPGA 的并行化优势,同时减少 FPGA 与 DDR3 的数据交互。最终,相比于单核 Cortex-A9 实现整个网络结构,采用 OTILE=8,ITILE=4,FTILE=4 的参数优化的网络可以达到 144 倍的加速,亦占用了 Zynq 的 PL 端大部分的逻辑资源。

参考文献:

[1] 龚　安,郭文婷. 基于卷积神经网络的皮肤癌识别方法[J]. 计算机技术与发展,2020,30(10):167-172.

[2] 李　欣,张　童,厚佳琪,等. 基于深度学习的多角度人脸检测方法研究[J]. 计算机技术与发展,2020,30(9):12-17.

[3] HAILESELLASIE M T, HASAN S R. MulNet: a flexible CNN processor with higher resource utilization efficiency for constrained devices [J]. IEEE Access, 2019, 7: 47509-47524.

[4] CHEN Y H, YANG T J, EMER J S, et al. Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices[J]. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2019, 9(2): 292-308.

[5] 仇　越,马文涛,柴志雷. 一种基于 FPGA 的卷积神经网络加速器设计与实现[J]. 微电子学与计算机,2018,35(8):68-72.

[6] 李申煜. 基于 Zynq 的卷积神经网络加速器设计[D]. 北京:北京理工大学,2016.

[7] 赵东升. 基于 HLS 的高效深度卷积神经网络 FPGA 实现方法[D]. 西安:西安电子科技大学,2019.

[8] 窦　阳,卿粼波,何小海,等. 基于 FPGA 的 CNN 加速器设计与实现[J]. 信息技术与网络安全,2019,38(11):96-101.

[9] ZHANG C, LI P, SUN G, 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. New York: ACM, 2015: 161-170.

[10] QIU J, SONG S, WANG Y, et al. Going deeper with embedded FPGA platform for convolutional neural network[C]// Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays. Monterey: ACM, 2016: 26-35.

[11] IANDOLA F N, HAN S, MOSKEWICZ M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size[J]. arXiv: 1602.07360, 2016.

[12] GSCHWEND D. ZynqNet: an FPGA-accelerated embedded convolutional neural network[J]. arXiv: 2005.06892, 2020.

[13] LI H, FAN X, JIAO L, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks[C]//2016 26th international conference on field programmable logic and applications (FPL). Lausanne: IEEE, 2016: 1-9.

[14] JIA Y, SHELHAMER E, DONAHUE J, et al. Caffe: convolu-

(下转第 121 页)