中国科学: 信息科学 2019 年 第 49 卷 第 3 期: 277–294
SCIENTIA SINICA Informationis
面向智能应用的定制计算加速器技术专题 • 评述

面向卷积神经网络的 FPGA 设计

卢丽强[1], 郑思泽[1], 肖倾城[1], 陈德铭[2], 梁云[1*]

1. 北京大学信息科学技术学院, 北京 100871, 中国
2. University of Illinois at Urbana-Champaign, Illinois, Urbana-Champaign 61820, USA
* 通信作者. E-mail: ericlyun@pku.edu.cn

收稿日期: 2018–10–30; 接受日期: 2019–02–26; 网络出版日期: 2019–03–19

国家自然科学基金 (批准号: 61672048) 和北京自然科学基金 (批准号: L172004) 资助项目

摘要 近年来, 卷积神经网络作为深度学习中的常用算法, 被广泛地应用在计算机视觉的任务中. FPGA 凭借它的高并行计算、低功耗和可重复配置的特点在实现卷积神经网络的多种加速器中显示了优异的特性. 近几年, 使用 FPGA 加速卷积神经网络的方法已经被人们广泛地探索, 但是大部分设计的性能都受限于片上乘法器数量. 快速算法在卷积操作中能够有效降低运算中乘法的数量, 进而达到平衡资源的效果. 本文首先介绍 4 种实现卷积神经网络的算法: 传统的空间卷积算法、矩阵乘法、Winograd 算法和 FFT 算法. 同时介绍国内外对于不同算法在硬件上的实现, 以及相应的优化手段, 并且总结使用 FPGA 加速 CNN 的发展历程.

关键词 卷积神经网络, FPGA, 卷积算法, 快速算法, Winograd, FFT

1 引言

深度学习这一概念在 2006 年被人提出, 通过增加人工神经网络的深度增加算法提取数据特征的能力. 卷积神经网络 (convolutional neural network, CNN) 作为深度学习中被广泛采用的算法, 被人们实现在大量的应用中, 例如人脸识别、车道检测、立体视觉匹配、语音识别 [1~4], 如图 1 所示. 但是卷积神经网络的计算量庞大, 这就要求设计与其计算量相对应的加速器. 图形处理器 (graphic processing units, GPU)、现场可编程逻辑门阵列 (field-programmable gate array, FPGA)、专用集成电路 (application specific integrated circuit, ASIC) 是常用的神经网络加速器. 在这些加速器中, FPGA 因为它的高度并行计算、低功耗、可重复编程的特性而被人们广泛采用. 此外, 高层次综合 (high level synthesis, HLS) 为软件开发人员提供了良好的编程环境, 降低了 FPGA 的开发周期, 使得 FPGA 编程更加有效率.

引用格式: 卢丽强, 郑思泽, 肖倾城, 等. 面向卷积神经网络的 FPGA 设计. 中国科学: 信息科学, 2019, 49: 277–294, doi: 10.1360/N112018-00291
Lu L Q, Zheng S Z, Xiao Q C, et al. Accelerating convolutional neural networks on FPGAs (in Chinese). Sci Sin Inform, 2019, 49: 277–294, doi: 10.1360/N112018-00291

(a) (b) (c) (d)

图 1 (网络版彩图) 卷积神经网络的各种应用

Figure 1 (Color online) CNN applications. (a) Face recognition; (b) lane detection; (c) stereo matching; (d) speech recognition

大量研究表明, 卷积神经网络的大部分计算都是卷积操作, 而卷积层包含的是大量的乘加运算, 先前利用 FPGA 加速神经网络的工作性能通常都受限于片上 (On-Chip) 数字处理单元 (digital signal processor, DSP) 的数量. 通常来说单个 DSP 只能在一个时钟周期内完成一次乘加操作. 因此, 如果想要 FPGA 加速的性能进一步提升, 我们需要从提高 DSP 利用效率方面入手. 在 FPGA 框架设计方面, 从 2015 年开始, 人们已经对各个优化手段作了充分研究, 例如在哪一维度做并行计算, 如何做数据定点化工作, 怎么降低带宽需求等等. 这些设计大部分是采用空间卷积算法的方式计算卷积, 优化操作大多基于对循环的优化, 例如循环展开, 循环重排序. 也有一些设计是将卷积操作展开成矩阵乘法操作, 这种做法可以利用成熟的通用矩阵乘法完成运算, 但是会造成冗余的数据复制. 基于上述两种卷积操作的加速器的性能通常受限于片上 DSP 数量, 近年来为了进一步加速 CNN 在 FPGA 上的运算速度, 人们开始利用一些快速算法来降低卷积操作中的乘法数量, 并将这些算法部署到 FPGA 上, 如 Winograd 算法, FFT 算法 (fast Fourier transform algorithm). 快速算法主要分为 3 个步骤, 首先将输入图像与卷积核转换到特殊域, 例如在 FFT 算法中将图像先转换到频域. 接下来将转换后的数据进行点对点相乘, 最后将相乘的结果逆变换回时域. 快速算法的优势就在于能够有效减少乘法次数, 但是快速算法在变换域的过程会有较多的常数乘法.

本文旨在从卷积的算法层面来分析近年来采用 FPGA 加速 CNN 的发展历程. 本文首先归纳总结了实现卷积操作的 4 种算法, 分别是传统的六层循环卷积、通用矩阵乘法、Winograd 快速算法和 FFT 算法. 通过算法层面的分析, 我们再介绍国内外对于不同算法在硬件上的实现, 以及相应的优化手段. 最后我们对国内的技术对比分析, 并且总结使用 FPGA 加速 CNN 的发展历程.

2 现场可编程逻辑门阵列 (FPGA)

2.1 FPGA 简介

FPGA 是一种半定制电路, 设计师可以通过配置开关状态来达到想要的逻辑操作. 如图 2, FPGA 的逻辑部分 (programmable logic, PL) 主要由可配置逻辑块 (configurable logic block, CLB) 和用于接口的输入/输出块 (input/output block, IOB) 构成. 图 2(a) 展示了 FPGA 的逻辑框架示意图. CLB 是逻辑单元的小规模编组, 从图 2(a) 中可以看出, CLB 排列为二维阵列, 每个 CLB 中包含两个逻辑片 (logic slice) 并且紧邻一个开关矩阵. 逻辑片主要由查找表 (lookup table, LUT) 和触发器 (filp-flop, FF) 构成. FPGA 的主要开发流程包括: 设计输入、综合 (synthesis)、功能仿真 (simulation)、实现 (implementation)、时序 (timing) 检查、比特流 (bitstream) 生成.

在学术研究领域, 大部分深度学习加速器都采用了片上系统的架构 (system on chip, SoC), 也就是 CPU+FPGA 的模式, 通过 CPU 上的操作系统来调度 FPGA 的运行. 这种架构可以使 FPGA 加

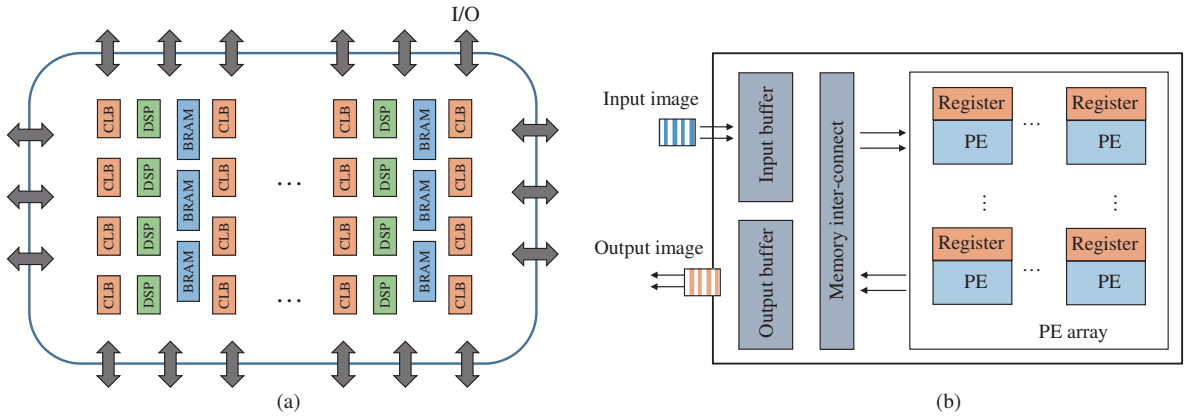


图 2 (网络版彩图) FPGA 框架与采用 FPGA 加速 CNN 的基本架构

Figure 2 (Color online) FPGA structure and a general architecture for accelerating CNNs on FPGA. (a) FPGA structure; (b) architecture for accelerating CNNs on FPGA

速器和 CPU 能通过片上总线进行紧耦合, 从而提高数据带宽, 提高执行效率, 减少加速器的调度代价. ARM 与 FPGA 之间会共享一块 DDR, ARM 与 FPGA 之间通过 AXI (advanced extensible interface) 接口实现互联, 进行处理器端到 FPGA 端的数据传输. 当今许多卷积神经网络加速器设计都在 Xilinx 公司的 Zynq 系列芯片或者 Intel SoC FPGA 上完成. 这些系列芯片包含双核或四核的 ARM 处理器. 通过其公司推出的软件开发包可以实现深度学习加速器软件部分的功能. 深度学习的发展对于 FPGA 性能与带宽也有了越来越高的要求. 为了能够应对日益增长的计算量, FPGA 上需要搭载更多的计算资源与更高的带宽. 提高资源密度的方式就是提高芯片的工艺水平, Xilinx 公司的 Virtex Ultrascale+ 系列采用了 16 nm 的工艺与 FinFET 技术. Intel 公司 Stratix 10 系列的芯片采用了 14 nm 的工艺与 Tri-Gate 的技术. 为了增加带宽, 这两款芯片使用了 HBM 技术 (high bandwidth memory), 采用 HBM 技术后可以达到 200~400 GB/s 的带宽.

图 2(b) 展示了采用 FPGA 加速卷积神经网络的基本设计框图. 从数据缓存的层次上来看, 大致可以分为 3 层: 片外存储, 片上缓存, 寄存器. 片外存储通常采用的是 DRAM 存储器, 片上缓存为 Block RAM, 由于片上缓存无法存放整个输入图像, 因此片上通常只存储部分输入图像. 对输入图像不同的划分方案对应着不同的存储策略与带宽需求, 通常来说需要最大化输入图像的重利用来减小带宽需求. 输入缓存与计算单元之间通过数据传输控制单元相连, 每次将计算单元所需要的数据传入计算单元内部的寄存器中. 由于 FPGA 支持高度的并行计算, 通常有大量的计算单元同时计算. 不同的并行策略对应着计算单元阵列中不同的连接方式.

2.2 高层次综合

高层次综合 (high level synthesis, HLS) 是最近被广泛采用的开发工具. 这里的 “高层次” 是相对于 HDL (hardware description language) 更高一个级别的语言. 首先, 我们认为底层是可以执行用户逻辑的物理器件, 具体可以细分到各个 LUT, DSP, 寄存器. HDL 相对这个底层要稍微高级一点, 它描述了最后期望电路中寄存器和寄存器之间发生的可以被解释执行的操作, 隐藏了技术层面的细节. 而高层次所对应的语言则隐藏了更多的 “细节”, 例如 C, C++, OpenCL. 它们从算法层级上描述了我们所期望的功能, 但是具体对应着怎么样的逻辑资源和其连接关系并没有展现. 高层次综合里的 “综合” 与 2.1 小节中的综合不同, 前文所指的综合是逻辑综合, 它是将 HDL 描述映射到门级网表的过

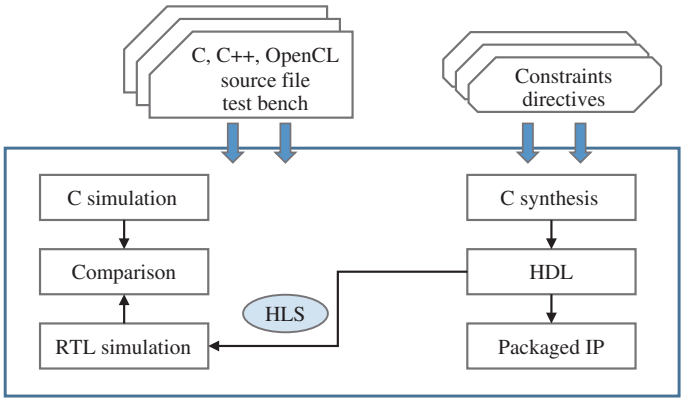


图 3 (网络版彩图) 高层次综合流程示意图

Figure 3 (Color online) Process flow of HLS

程. 而高层次综合是指将高层的 C, C++ 代码先分析解释为 HDL 描述, 再进行逻辑综合, 因此综合所需时间也更长. 图 3 展示了基于 HLS 的开发流程. 它可以将 C++ 描述的算法综合到 RTL (resistor transistor logic) 级别的实现, 并且可以添加各类硬件优化的引导语句 (directives). 从图 3 中可以看出, 输入包括两个部分, 用于正确性验证的测试代码和用于综合的 C++ 代码. 输出包括了标准的 RTL 级别的 VHDL 和 Verilog 代码, 并且将设计封装成 IP 供 Vivado 或者第三方使用.

3 卷积算法

3.1 非快速算法

3.1.1 空间卷积算法

主要的计算公式为 $\mathrm{Out}(o,p,q)=\sum_{c<C,i<K,j<L}F(o,c,i,j)\times\mathrm{In}(c,p\times\mathrm{st}_r+i-\mathrm{pad},q\times\mathrm{st}_c+j-\mathrm{pad})$. 其中 In 为输入数据, 如一张图片, 一般情况下是三维张量, 其第一维是通道数 C, 第二维是图片高度 H, 第三维是图片宽度 W. F 对应着卷积层中的 Filter, 也就是网络的参数所在, 有四个维度, O 表示输出通道数, C 表示输入通道数, K 为 Filter 高度, L 为 Filter 宽度. 另外的输入 $\mathrm{st}_r,\mathrm{st}_c,\mathrm{pad}$ 分别为 Filter 的沿高度方向滑动距离与沿宽度方向滑动距离以及边界处补全长度 (padding length).

空间卷积算法非常直观, 易于实现, 可以短时间内部署到 CPU, GPU 等多种设备上运行, 在此算法基础上也可以进行很多优化提高算法效率, 这也从侧面表示出, 这个算法的效率并不是最好的, 需要 $O(OPQCKL)$ 次乘加运算, 典型地, 当 $L=K$, $P=H-K+1$, $Q=W-K+1$ 且 $H>K$, $W>K$ 时, 需要 $O(OHWK^2C)$ 次运算.

3.1.2 通用矩阵乘法算法 (GEMM)

第 2 种算法是利用通用矩阵乘法实现卷积操作的算法, 称为 GEMM (general matrix multiplication). 主要思想是将原来的张量展开为矩阵, 然后直接利用成熟的矩阵乘法程序完成卷积操作. 可以参考图 4 理解 GEMM 算法流程.

算法先要将输入 In 和 Filter 分别转换为可以进行矩阵乘积的形式, 对于 In 来说, 就是将会与 Filter 做乘法并被加在一起的元素展开成连续存储的一行, 这个展开过程要考虑到 Filter 的滑动距离

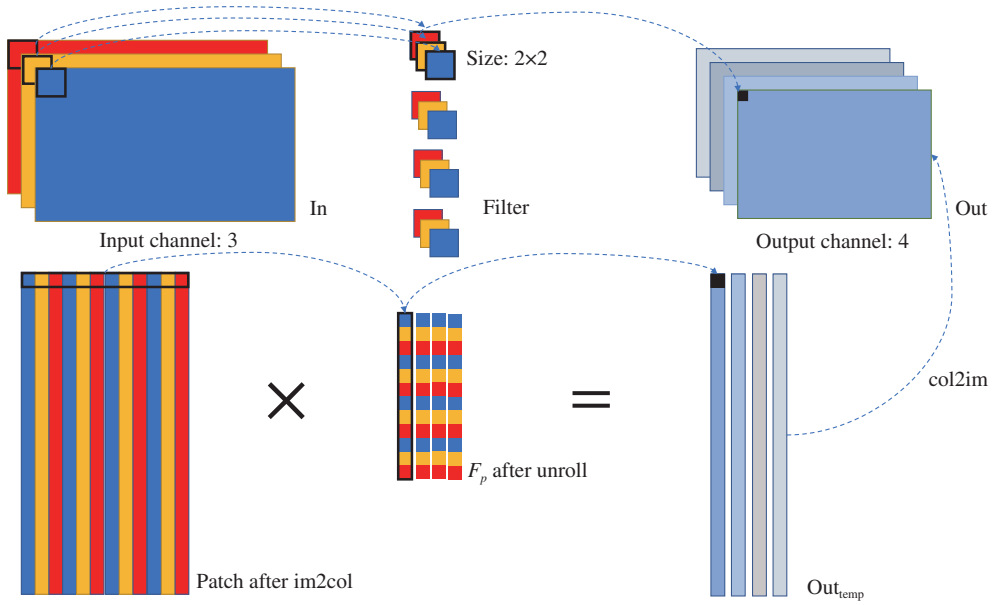


图 4 (网络版彩图) 将卷积操作转化为 GEMM

Figure 4 (Color online) Converting convolution operation to GEMM operation

st_r, st_c 以及补全问题 pad, 所以用到的函数 im2col 需要这些数字作为参数, 将变换后的结果存放在张量 Patch 中, 相应地, Filter 也要展开, 展开方式满足一列与 Patch 的一行元素一一对齐, 从而保证一次矩阵向量乘法后结果就是卷积运算的结果. 最后要将矩阵向量乘法的输出通过 col2im 函数转换为多通道二维图像形式的张量 Out.

这里也只是呈现了 GEMM 方法中的一种, 用到的主要转换方法为 im2col, 同时还存在很多其他方法, 如 im2row, kn2col, kn2row 等, im2col 与 im2row 是从输入图像的角度展开, 而 kn2col 和 kn2row 则是从 Filter 角度出发, 无论哪种方法, 面临的问题是, 一方面并没有减少所必须的计算数目 (乘法和加法数目), 另一方面又产生了巨大的内存开销和读写需求, 有的工作针对内存开销问题给出了解决思路 [5]. GEMM 方法的优势在于能够利用非常成熟的矩阵乘法程序, 这类程序已经做到了针对常用设备 (如 CPU, GPU 等) 的非常好的优化, 性能方面得到充分的提升, 所以能够复用这类程序为卷积操作的性能提升提供了可能, 也免去了专门为卷积操作进行复杂优化的人力工作.

3.2 快速算法

在非快速算法的卷积算法中输出特征图中的每个元素被单独计算, 快速算法在运算过程时对输入图像中局部元素同时计算. 本文假设局部区域的大小为 $n \times n$, 滤波器大小为 $r = K = L$, 我们使用快速算法生成大小 $m \times m$ $(n = m + r - 1)$ 局部输出元素. 基于快速算法的卷积描述如下:

$$\text{Out}_t = \text{Inverse\_Transform}[\text{Transform}(\text{In}_t) \odot \text{Transform}(F)], \tag{1}$$

其中 $\odot$ 表示元素的点对点相乘, In_t, Out_t 表示局部的输入输出块. 根据式 (1), 快速算法可以分为 3 个步骤:

- 输入块与卷积核变换. 第一步是将输入块与卷积核转换到特殊域, 并且保持尺寸大小相同. 其中 Winograd 与 FFT 的区别在于转换函数的不同.

• 点对点乘法 (element-wise matrix multiplication, EWMM). 第二步是将转换后的输入块与卷积核进行点对点相乘. 其中 Winograd 与 FFT 的区别在于 Winograd 算法中是实数乘法, 而 FFT 中是复数乘法.

• 逆变换. 最后一步是将上述结果进行逆变换, 得到最终时域的结果. 其中 Winograd 与 FFT 的区别在于转换函数的不同.

接下来我们将对 Winograd 与 FFT 算法进行更多的细节讨论.

3.2.1 FFT 算法

FFT 算法是通信领域常用的信号处理算法, FFT 算法利用了数字信号在时间域的卷积操作等价于频率域的乘法操作的性质. 1965 年, Cooley-Turkey 提出快速傅里叶变换 (FFT) 来加速机器的计算速度, 这可以使运算复杂度从 $O(N^2)$ 减少到 $O(N\log N)$ 的复数乘法, 且不会出现精度损失. 对于输入信号序列 $\{X(N)\}$, 其离散傅里叶变换 (discrete Fourier transformation, DFT) 可以表示如下:

$$X(k)=\sum_{n=0}^{N-1}x(n)\cdot \mathrm{e}^{-\mathrm{i}2\pi kn/N}. \tag{2}$$

Cooley-Turkey 1 维 FFT 计算如下: 首先将序列不断按照奇数项与偶数项对半分,

$$\begin{cases} X(k)=X_1(k)+W_N^kX_2(k), \\ X\left(k+\dfrac{N}{2}\right)=X_1(k)-W_N^kX_2(k), \end{cases} \quad k=0,1,2,\ldots,N/2-1, \tag{3}$$

其中 $X_1(k)$, $X_2(k)$, W_N^k 表示如下:

$$\begin{cases} X_1(k)=\displaystyle\sum_{r=0}^{N/2-1}x(2r)W_N^{2rk}, \\ X_2(k)=\displaystyle\sum_{r=0}^{N/2-1}x(2r+1)W_N^{2rk}, \end{cases} \quad W_N^k=\mathrm{e}^{-2\mathrm{i}\pi k/N}. \tag{4}$$

被划分后的两项又可以看作是新的一个 FFT 运算, 序列长度为原始长度的一半. 对划分后的序列继续进行对半划分, 最终原始序列被转换为多个两点 DFT, 这也被称作蝶形运算. 由于在 CNN 模型中, 输入图像与卷积核均是二维, 因此我们实际采用时应为二维 FFT 算法. 二维 FFT 的操作可以分为多次一维 FFT 操作, 具体为首先对二维矩阵的各行进行一维 FFT 操作, 而后对各列进行 FFT 操作.

将快速算法应用于卷积操作, 乘法来源于式 (1) 中的点乘操作. 采用传统的卷积操作计算式 (1) 需要 m^2r^2 次乘法, 采用 FFT 算法后仅需要 n^2 次复数乘法. 对此实际上还有进一步优化, 一般来说需要消耗 4 次实数乘法来完成一次复数乘法, 实际上通过数学变形只需要 3 次乘法:

$$\begin{aligned} (a+ib)\times(c+id)&=(ac-bd)+i(ad+bc), \\ (ac-bd)&=b(c-d)+c(a-b), \\ (ad+bc)&=b(c-d)+d(a+b). \end{aligned} \tag{5}$$

另一方面, 由于在现实的 CNN 模型中输入都是实数值, 利用 Hermitian 对称性可以将复数乘法次数从 $n\times n$ 降低到 $n\times(\lfloor\frac{n}{2}\rfloor+1)$. 具体来说, 是因为 FFT 变换后的矩阵存在共轭对称性:

$$\overline{X(i,j)}=X(-i \bmod n,-j \bmod n). \tag{6}$$

利用上述两种优化手段后, 最终实数乘法次数可以降低为 $3n(\lfloor \frac{n}{2} \rfloor + 1) \approx 1.5$ 每个输入像素点.

3.2.2 Winograd 算法

Winograd 算法于 1980 年由数学家 Winograd 提出 [6]. Winograd 算法与 FFT 算法类似, 但是 Winograd 转换过程由常数矩阵乘法实现. 首先我们来看一个简单的例子,

$$\mathrm{In}_t = \begin{bmatrix} z_0 & z_1 & z_2 & z_3 \end{bmatrix}^{\mathrm{T}}, \quad F = \begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix}^{\mathrm{T}}, \quad \mathrm{Out}_t = \begin{bmatrix} y_0 & y_1 \end{bmatrix}^{\mathrm{T}}.$$

上述例子中, 采用传统方法计算 y_0, y_1 需要 6 次乘法. 而采用 Winograd 算法则只需要 4 次乘法, 具体操作如下:

$$\begin{bmatrix} z_0 & z_1 & z_2 \\ z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 + m_4 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}. \tag{7}$$

m_1, m_2, m_3, m_4 为

$$\begin{aligned} &m_1 = (z_0 - z_2)x_0, \quad m_2 = (z_1 + z_2)\frac{x_0 + x_1 + x_2}{2}, \\ &m_4 = (z_1 - z_3)x_2, \quad m_3 = (z_2 - z_1)\frac{x_0 - x_1 + x_2}{2}. \end{aligned} \tag{8}$$

为了便于理解, 上述操作可以表示为多个矩阵乘的形式,

$$\mathrm{Out}_t = A^{\mathrm{T}}[(GF) \odot (B^{\mathrm{T}}\mathrm{In}_t)]. \tag{9}$$

$$B^{\mathrm{T}} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}, \quad A^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}.$$

对于二维 Winograd 情况, 我们可以对一维情况进行嵌套迭代得到,

$$\mathrm{Out}_t = A^{\mathrm{T}}[(GFG^{\mathrm{T}}) \odot (B^{\mathrm{T}}\mathrm{In}_t B)]A. \tag{10}$$

在 Winograd 算法中, 乘法次数为 n^2 次实数乘法.

3.3 算法总结

表 1 [7~35] 对 4 种算法进行了总结. 在 FPGA 实现中空间卷积算法被广泛采用. 由于循环层数较多, 这种算法存在广泛的探索空间, 不同的循环顺序与并行方式对应着不同数据重利用维度与访存模式, 其核心计算为乘加操作 (multiply-and-accumulate, MAC), 计算单元的计算取决于具体的循环顺序与并行策略, 例如在输入通道采用并行计算, 则其对应的是向量内积. 将卷积操作展开成矩阵乘法是在 CPU 与 GPU 平台上常用的手段, 并且被人们广为优化. 矩阵乘法的通用性更强, 例如适用于卷积操作中不同的卷积核大小, 不同的步长. 但是矩阵乘法的缺点就是存在数据的冗余复制, 由于 FPGA 的片上内存资源较少, 因此采用这种方法的相关工作较少. FFT 算法是信号处理的常用手段, 但是由于其操作在复数域进行, 因此对于乘法的减少相对较少. Winograd 算法流程与 FFT 类似, 但是运算过程均在实数域完成. 相比于 FFT, Winograd 的转换代价更高, 这是因为随着 Winograd 算法参数增大 (n, m, r), 常数矩阵中的值会骤增.

表 1 用于卷积的 4 种算法总结与国内外相关工作

Table 1 The conclusion of four convolution algorithms and related work

Algorithm	Arithmetic reduction	Transformation overhead	Inner computation	Domestic related work	Abroad related work
Spatial	None	None	MAC	[7∼13]	[14∼25]
GEMM	None	Low	Vector product	[26]	[27]
Winograd	High	High	EWMM	[28∼30]	[31, 32]
FFT	Medium	Medium	EWMM (complex)	[33]	[34, 35]

从 2015 年开始, FPGA 成为加速卷积神经网络的平台, 当时侧重点主要还是将空间卷积这一操作映射到 FPGA 的具体实现中. 随着时间推移, 越来越多的加速器架构被人们提出, 如何高效地使用 FPGA 加速卷积神经网络已经被人们广泛研究, 大部分设计中最终性能都受限于片上 DSP 数量. 因此, 为了进一步加速 FPGA, 人们开始将目光移向了快速算法. 快速算法能够有效降低卷积操作的乘法次数, 提高加速比, 相比于非快速算法, 快速算法需要一些额外的操作, 这些操作大部分都是常数乘法, 在硬件实现过程中, 这些常数乘法会被转换为多个位运算相加的操作, 位运算可以不需要消耗片上的 DSP 资源, 仅使用 LUT 阵列就可以实现位运算. 从近两年的研究现状来看, 基于快速算法的工作在逻辑资源使用方面确实要高于非快速算法的工作. 此外, 快速算法是以一个输入块进行操作, 因此对于片上缓存的容量要求更高. 并且快速算法加快了整体的运算过程, 因此对于片上与片外数据带宽需求也更大. 综上所述, 快速算法的操作流程异于传统的卷积算法, 因此基于快速算法的新的 FPGA 架构也被提出. 第 4 节将会简述国内外关于 4 种卷积算法的相关工作.

4 国内外相关工作简述

4.1 国内相关研究工作

4.1.1 基于空间卷积算法的设计

北京大学和美国加利福尼亚大学洛杉矶分校 (University of California, Los Angeles) 合作 [8], 提出了 FPGA 上实现 CNN 模型的过程中, 需要充分考虑计算吞吐与内存带宽的匹配问题, 因此他们根据天花板模型 (roofline model) 设计了分析方法. 通过使用循环展开、分块等变换优化 CNN 网络的计算吞吐和访存带宽, 达到最好性能和最小资源使用. 实验中, 他们在 VC707 FPGA 开发板上实现了一个 CNN 模型并和先前工作进行比较, 结果表明在 100 MHz 频率下峰值性能有 61.62 GFLOPS, 优于先前工作.

清华大学主导完成的工作 [7] 中, 他们提出基于 FPGA 的针对 CNN 的加速器设计, 用于 ImageNet 大规模图形分类应用. 经过对 CNN 网络的深入分析, 他们提出卷积层是计算集中的, 而全连接层是访存集中的, 相应地, 他们提出动态精度数据量化方法, 配合设计卷积器, 达到对两种网络层次的高效加速, 既提高了带宽, 又增加了资源利用率. 实验表明, 他们在 VGG 16 模型上使用 8/4 比特量化时精度损失仅有 0.4%, 为了充分利用外部内存带宽, 他们还设计了一种数据排列方法. 他们在 Xilinx Zynq ZC706 开发板上实现的 VGG16-SVD 网络, 达到了 4.45 fps 的帧率和 86.66% 的 top-5 准确率, 仅仅用 16 比特量化. 实验中卷积层的平均性能为 187.8 GOP/s, 全连接层则是 137.0 GOP/s, 超过了之前的工作.

复旦大学的工作 [10] 中, 他们将 CNN 模型的所有网络层都映射到 FPGA 的一个芯片上以提高不同层的并发度, 通过流水线结构增加了吞吐. 同时, 他们还提出了寻找最优并行策略的方法, 提高吞吐和资源利用. 此外, 他们还实现了全连接层的批计算方法, 提高了访存带宽的利用. 通过在全连接层使用两种不同的计算模式, 可以极大减少片上缓存的需求. 实验中, 他们在 Xilinx VC709 上实现了 AlexNet, 峰值性能为 565.94 GOP/s, 工作时钟频率为 156 MHz, 这一结果超过了先前工作.

北京大学和美国加利福尼亚大学洛杉矶分校合作 [11] 他们在 FPGA 上利用脉动阵列 (systolic array) 实现 CNN, 以达到高资源利用率下的高时钟频率. 并提供性能和资源利用的分析模型, 同时构建了自动搜索设计空间的框架和 C 程序源码转换为使用脉动阵列的 CNN 实现的机制. 实验结果表明, 他们的框架能够为实际 CNN 模型产生加速代码, 达到 461 GFLOPS 的浮点计算性能以及 1.2 TOPS 的 8∼16 比特定点计算性能.

Iception 与 Residual 是现代 CNN 中两种重要结构, 清华大学在 2018 年的工作 [12] 中指出当前 FPGA 平台上的映射方法不能充分适应 Iception 和 Residual 中不同层之间的数据局部性以及其他许多特征, 因此并没有充分发挥 FPGA 计算存储资源的能力. 他们提出 LCP (layer clusters paralleling) 映射方法, 把不同的层根据参数、数据局部性特征划分到不同的集群中, 然后把不同的集群放在 FPGA 的不同部分上进行加速. 实验中, 他们在 Xilinx VC709 FPGA 上实现了 GoogLeNet 和 ResNet-50 中 Iception/Residual 模块, 结果表明 LCP 方法能达到与 baseline 相比 4.03 倍的加速和文献 [18] 相比最佳实现 2.00 倍的加速.

大部分 FPGA 加速器都是用于处理 CNN 的推理过程, 近年来也有在 FPGA 上实现 CNN 训练的相关工作. 清华大学与哈尔滨工业大学 [13] 提出, 使用 FPGA 加速 CNN 训练过程需要解决 3 个方面问题: (1) 模块化. 一个 CNN 模型包含了多种类型的层, 所以为了完成在 FPGA 上训练 CNN, 需要对不同的层进行模块化设计. (2) 统一的数据路径. 为了减少模块间耦合的消耗并提高整个架构的有效性, 需要设计统一的数据流路径, 便于最好地利用存储带宽. (3) 运行时重配置性. 为了适应整个 CNN 模型的训练过程, 需要整个架构具有可重配置性. 文献 [13] 提出 CPU-FPGA 协同处理架构, 其中 CPU 用做控制器, FPGA 用来加速计算, 同时 FPGA 上的 DRAM 用于存储每个计算模块的输入输出数据. 在该架构中, 模块控制器负责根据不同的层配置定制不同的计算模块, 并且按照特定顺序将模块重新配置到 FPGA 上: 在一个训练周期中先是从底层到顶层的前向计算模块, 然后是从顶层到底层的反向计算模块. 数据控制器将训练数据分成不同的 minibatch, 并将它们加载到 DRAM 中用于训练.

4.1.2 基于 GEMM 的设计

北京大学和美国加利福尼亚大学洛杉矶分校合作 [26], 他们提出了 FP-DNN (field programmable DNN) 框架, 将用 TensorFlow 描述的 DNN 模型作为输入, 自动生成在 FPGA 开发板上的硬件实现. FP-DNN 利用高性能计算引擎和精确设计的通信优化策略来执行预测 (inference) 过程. 他们利用 FP-DNN 实现了 CNN, LSTM-RNN, Residual Net 等, 实验结果表明无论是性能还是灵活性上, 他们的框架都提供了很好的支持.

由北京大学与美国加利福尼亚大学洛杉矶分校、Falcon-computing 公司合作的工作 Caffeine [9] 中, 他们设计并实现了一个软硬件协同代码库, 提供在 FPGA 上实现 CNN 的全面加速应用. 他们将卷积层运算和全连接层运算都表示成矩阵乘法, 然后专注于 FPGA 的计算和访存优化. 在设计 Caffeine 时, 他们使用了高层次综合并为使用者提供了硬件描述参数. 此外, Caffeine 可以结合进 Caffe 框架中使用. 实验表明, 他们的设计能提供 365 GOPS (在 Xilinx KU060 FPGA 上) 和 636 GOPS (在 Virtex7

690t FPGA 上) 的峰值性能. 而且在 FCN 层上能做到相比以往工作 100 倍的加速比. 除了性能的提升, 在节能方面他们的工作也有显著的成果. 并且该研究小组在后续工作 [36] 中对于片上存储进一步优化, 减少了带宽压力.

4.1.3 基于 Winograd 的设计

国防科技大学在 2018 年的工作 [28] 中指出, 当前很多的 FPGA 上 CNN 加速工作仅关注二维卷积, 然而加速三维卷积同样有重要意义. 三维卷积在计算复杂度和访存开销上都提出了新的挑战. 他们提出了一个统一的、基于模板的框架提供对二维和三维卷积操作的加速. 他们同时提出了辅助设计空间搜索的分析模型. 实验中, 他们在 S2C VUS440 上实现了 VGG16 和 C3D 模型, 结果为 VGG16 上 1.13 TOPS 和 C3D 上 1.11 TOPS 的性能. 他们实现的 C3D 做到了相比 CPU 实现 13 倍的加速和 60 倍的节能以及相比 GPU 6.4 倍的节能效果.

北京大学和香港中文大学在 2017 年的工作 [29, 33] 中指出, 传统的 CNN 卷积层算法 (即空间卷积算法和 GEMM 算法) 往往受限于 FPGA 的计算单元数目, 而 Winograd 快速算法可以极大减少计算复杂度, 从而提高 FPGA 上的卷积操作效率. 他们首先提出了在 FPGA 上部署 Winograd 算法的新方法, 通过利用行缓冲 (line buffer) 结构高效地重用数据. 同时, 他们还采用了流水线和并行技术. 他们针对复杂的设计空间提出了分析模型来预测资源使用和性能表现, 这个模型被用来指导设计空间的搜索. 实验中, 他们使用 Xilinx ZCU102 开发板在 AlexNet 上得到了 1006.4 GOP/s 的卷积层平均性能和 854.6 GOP/s 的总体平均性能; 在 VCC16 上得到了 3044.7 GOP/s 的卷积层平均性能和 2940.7 GOP/s 的总体平均性能. 并且该研究小组在工作 [37] 中将 Winograd 算法与稀疏化相结合, 使性能进一步提升. 在北京大学和香港中文大学合作的另一项工作 [30] 中, 他们同样使用了 Winograd 算法, 特别的是他们提出了一个可以合并 CNN 中许多层的架构, 提高了层间数据复用率, 并且他们在异构平台上设计了对 CNN 中每一层的合并策略最优化的算法. 通过实现一个自动工具链, 他们能够将 Caffe 模型映射到 FPGA 上. 实验中他们用 VGG 和 AlexNet 证明了他们的设计能够达到相比先前采用层合并策略的加速工作 1.99 倍的加速.

4.2 国外相关研究工作

4.2.1 基于空间卷积算法的设计

帝国理工大学 (Imperial College London) [14] 将 CNN 的处理过程视为一种流结构 (streaming architecture), 将 CNN 描述为 SDF (synchronous data flow) 模型, 并进行设计空间探索, 设计一套转换库实现 CNN 模型从 SDF 到 FPGA 上的映射, 最终输出可综合的 Vivado HLS 硬件设计. 相比于前人基于 FPGA 的 CNN 设计及优化, 该作者提出的优化和映射方法可以包含 CNN 的卷积层、池化层和非线性层, 可以吸收所有 FPGA 平台的参数. 从 SDF 到硬件的映射的方法有 3 种: SDFG 划分以及 FPGA 资源的重配置; 粗粒度的折叠, 实现途径是参数化一个层 (layer) 的展开的程度, 如果资源足够可以完全展开并行, 也可以只展开一半, 分两次处理; 细粒度折叠, 实现途径是参数化点乘的并行程度, 同样是完全并行或者时分复用. 实验显示可以达到 12.73 GOP/s 的性能和 7.27 GOP/s/W 的能耗比.

纽约州立大学石溪分校 (Stony Brook University) Michael Ferdman 研究小组的研究重点主要在片内外访存带宽需求上 [16~18]. 其在文献 [16] 中指出, 在计算神经网络时, 由于是逐层计算, 在计算每层的前后都要读写特征图片 (feature map). 这占据了近 50% 的片外存储 (off-chip memory) 访问. 因此文中提出了层间计算融合的模式 (fusion), 与传统的加速器不同, 在 fusion 计算模式中, 一层结果计算完

成后将会进行下一层的计算而不是被存储到片外中. 为了尽可能地减少片内外的数据传输, 文献 [16] 还提出了一个优化框架, 探索如何划分融合网络的各个层. 经过 FPGA 上进行验证后, 文献 [16] 得到了可减少高达 95% 的片外存储访问的结果. 文献 [17] 提出了 Escher 加速器, 给定带宽的预算, 合理地选择输入图像的数量 (batch size) 可以提高带宽利用率. 在 Virtex-7 690T FPGA 平台上, Escher 能够降低 2.4 倍的带宽需求. 先前的加速器配置通常只能针对特定网络才能达到最佳性能, 实际上随着层变换, 单一加速器的设计很可能只能达到次优的性能. 因此, 该小组还提出通过资源划分来提高硬件效率 [18]. 具体来说, 他们设计了多个卷积层处理单元来并行地计算 CNN 模型, 并且设计了优化算法来指导如何进行资源划分.

亚利桑那州立大学 (Arizona State University) 从空间卷积算法入手, 探索使用 FPGA 加速 CNN [19~21] 的各种优化手段. 在文献 [19] 中指出, HLS 工具可以快速实现设计, 但是硬件实现效率低下, 而传统的 RTL 设计虽然能够精细优化, 但是开发效率低. 因此, 文献 [19] 提供可扩展的解决方案, 集成了高层次综合灵活的特性和底层 RTL 的优化. 该工作核心是一个 CNN 编译器, 它能分析 CNN 的结构和参数, 并自动生成一组模块化和可扩展的计算模块来加速各种深度学习算法. 对于 Alexnet 和 NIN 网络, 该工作在 Stratix-V GXA7 FPGA, 分别达到 114.5 GOP/s 和 117.3 GOP/s. 文献 [20, 21] 在 Altera Arria 10 GX 1150 FPGA 上实现的 VGG 网络能够达到 624.25 GOP/s 的性能.

4.2.2 基于 GEMM 的设计

2014 年, 美国加利福尼亚大学洛杉矶分校的 Cong 等 [27] 从数学层面分析了卷积操作的特点, 并且采用矩阵乘法的方式来计算卷积. 文中针对卷积的矩阵乘法提出来多种优化方式并与传统的矩阵乘法进行对比. 2016 年, 亚利桑那州立大学基于 OpenCL 编程框架在 FPGA 上实现 CNN 加速, 也是采用了矩阵乘法来计算卷积操作. 最终在 Intel Altera P395-D8 平台上达到了 136.5 GOP/s 的性能.

4.2.3 基于 Winograd 的设计

Intel [31] 提出了基于 Intel OpenCL 的 FPGA 深度学习加速器 DLA (deep learning accelerator). 该加速器采用了 Winograd 算法, 并且利用了 Altera 平台上一个 DSP 可以处理两个半精度浮点数乘法的特性. 在具体实现中, DLA 采用了 1 维的 Winograd 算法, 能够将乘法次数减少两倍, 最终在 Intel Altera Arria-10 平台上能够达到 1.38 TOP/s 的性能. 南加利福尼亚大学 (University of Southern California) 的 Prasanna 等 [32] 采用二维 Winograd 算法, 并建立了性能评估模型. 文献 [32] 还对比了 Winograd 算法与 FFT 算法, 并指出 FFT 算法对内存的需求更大, 因为数据为复数, 并且配置参数相同的情况下, Winograd 的加速比要优于 FFT 算法.

4.2.4 基于 FFT 的设计

南加利福尼亚大学的 Prasanna 等 [34] 也采用了 FFT 算法来加速卷积神经网络. 文献 [34] 设计采用了 8×8 的 FFT 尺寸进行加速, 在输入输出通道维度做并行计算. 最终在 CPU-FPGA 异构平台 HARP (Intel heterogeneous architecture research platform) 上测试, 实验结果显示 123.5 GOP/s 的性能. 但在文献 [34] 中, 并没有采用 Hermitian 对称的优化, 也没有使用三次乘法来算复数乘法的优化.

近年来, 也有在 FPGA 上采用 FFT 进行 CNN 训练的相关工作 [35]. 文献 [35] 的特点在于, 整个训练都在频域进行. 传统的前向过程中, 每一层的输出都需要转换回时域再进行下一层的计算, 实际上这会导致很大的冗余变换操作. 基于此, 文献 [35] 提出了频域的训练方式, 能够充分利用 FFT 算法

表 2 非传统卷积算法在 FPGA 上的实现总结

Table 2 The implementation of non-conventional convolution algorithms on FPGA

Algorithm	Winograd				FFT	GEMM		
	[28]	[29]	[38]	[31]	[34]	[39]	[9]	[26]
FPGA	VCU440	ZCU102	Virtex7 VX690T	Arria10 GX1150	Stratix5 QPI	Stratix5 GXA7	Kintex KU060	Stratix-V GSMD5
DSP	2880	2520	3683	1576	224	256	1058	1590
Logic (K)	5541	600	505	246	201	228	150	172
Frequency (MHz)	200	200	200	303	200	194	200	150
Precision	16 bit	16 bit	FP32	FP16	16 bit	16 bit	16 bit	FP16
Network	VGG	VGG	Alexnet	Alexnet	VGG	Alexnet	VGG	VGG
Performance (GOP/s)	821	3045	46	1382	123	66	360	364.36
Power	–	23.6	–	44.3	13.2	33.9	25.0	25.0

带来的加速效果. 该设计中采用了 Hermitian 对称的优化, 与 3 乘法来算复数乘法的优化. 通常卷积需要用 0 将尺寸扩大到与输入图像相同, 该设计通过辛格函数卷积权值来减少权值的存储空间.

5 在 FPGA 上部署 CNN 的总结与分析

5.1 非传统卷积算法在 FPGA 上实现性能总结

表 2 [9, 26, 28, 29, 31, 34, 38, 39] 列出了近年非传统卷积算法在 FPGA 上实现的情况, 可以看出在快速算法实现方面, 国内外水平相近. 从表 2 中可以看出, Winograd 的使用最为广泛, 这也得益于 Winograd 算法能够带来比 FFT 更大的运算量减少. 对于表 2, 我们可以比较各类算法所对应的资源效率与能源效率, 例如文献 [29] 的 DSP 资源效率为 1.21 GOP/s/DSP, 文献 [34] 的 DSP 效率为 0.55 GOP/s/DSP, 文献 [26] 的 DSP 资源效率为 0.23 DSP/s/DSP. 通过观察这个比例, 我们可以发现其实这比例与快速算法带来的乘法减少量相近. 在文献 [29] 中采用 6×6 的 Winograd 参数, 这可以带来 4 倍乘法量减少. 文献 [34] 采用 8×8 二维 FFT, 可以带来 3 倍左右的乘法减少.

5.2 传统卷积算法在 FPGA 上实现总结

根据之前所述, 绝大多数的 FPGA 皆是对传统卷积算法进行实现. 因为传统卷积算法共有六重循环, 有着庞大的设计探索空间, 并且 FPGA 灵活可重配的特性进一步增大了设计的空间. 表 3 展示了 2016 年到 2018 年的 FPGA 的部分加速器, 该表可以从两个方面来分析.

(1) 性能增加. 从 2015~2016 年间, FPGA 加速器的设计初具雏形, 这段时间内, 学者探讨的主要还是如何将卷积神经网络部署到 FPGA 中, 对于如何去最大化性能, 如何合理利用带宽并未透彻研究. 因此可以看到在 2016 年的 FPGA 加速器性能只有几十到几百左右的 GOP/s. 随着加速器的范式不断稳定, 人们开始更加着重于性能的优化, 例如最新的工作 [23, 31] 中 FPGA 频率可以到达 300 MHz 以上, 性能急剧上升到 TOP/s 级别.

(2) 网络变换. 从表 3 [7, 9~12, 14, 20, 21, 23, 24, 26, 39] 中可以看出, FPGA 加速器的设计趋势都是顺应深度学习网络的发展趋势. VGG 和 Alexnet 是 2014~2015 年的热门网络, 因此 2016 年的 FPGA 加速器均是针对该网络而设计的. 随后网络发展趋势朝着更深, 拓扑结构更复杂的方向发展, 例如 Resnet

表 3 传统卷积算法在 FPGA 上的实现总结

Table 3 The implementation of the conventional convolution algorithms on FPGA

	2016					2017				2018		
	[9]	[14]	[7]	[39]	[10]	[11]	[20]	[26]	[23]	[12]	[21]	[24]
FPGA	Virtex VX690T	Zynq Z7020	Zynq Z7045	Stratix GSD8	Virtex7 VX690T	Arria10 GT1150	Arria10 GT1150	Stratix-V GSMD5	Arria10 GT1150	Virtex VX690T	Arria10 GT1150	Kintex KU115
DSP	3600	220	900	1963	2144	1576	1576	1590	1576	3600	1576	5520
Logic (K)	693	53.2	250	695	274	246	246	172	246	693	246	1451
Frequency (MHz)	150	125	150	120	156	232	150	150	370	200	200	–
Precision	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	FP32	16 bit	16 bit	16 bit
Nework	VGG	VGG	VGG	Alexnet	Alexnet	VGG	VGG	Resnet	VGG	GoogleNet	Resnet	YOLO
Performance (GOP/s)	354	49	137	72.4	566	1171	645	226	866	870	707	2109
Power	25	–	9.6	19.1	30	45	50	25	41.7	–	50	22.3

表 4 利用 FPGA 加速卷积神经网络不同的优化方向

Table 4 Optimization techniques of accelerating CNNs on FPGA

	Design space exploration	Data transfer optimization	On-chip resource optimization	Hardware unit generatio
Related work	[8, 11, 20, 21, 29, 40, 41]	[11, 12, 16, 23]	[18, 25, 30]	[15, 22∼24, 42]

和 GoogleNet 采用了分支结构. 应对网络变化, FPGA 加速器的设计也需要作出调整, 例如如何处理分支, 如何兼容更多的层.

5.3 FPGA 优化总结

给定了某种卷积算法后, 如何将其合理地部署到 FPGA 上也是学界讨论的重点, 各个维度的优化被人们不断提出. 表 4 [8, 11, 11, 12, 15, 16, 18, 20∼23, 23∼25, 29, 30, 40∼42] 将其分为 4 大类: 设计空间探索, 数据通路优化, 片上资源优化, 硬件单元生成.

设计空间探索. 大多数设计空间探索的工作都是针对空间卷积算法, 由于循环层数众多, 因此存在广泛的探索空间. 探索过程的目标为最大化 FPGA 性能, 而相应的限制为网络参数与 FPGA 资源限制. 大多数工作与文献 [8] 类似, 从计算资源与数据传输带宽两个角度入手建立模型. 文献 [20, 21] 将优化空间卷积算法根据循环分为 3 个方面: 循环展开 (unrolling)、循环划分 (tiling)、循环交换 (interchange). 文中系统地整理了前人基于空间卷积算法的工作, 并且定量地描述了这 3 种优化所对应的带宽需求, 片上存储需求, 由此建立了设计探索空间.

数据通路优化. 数据通路优化可分为片上数据传输和片外数据传输. 文献 [11] 采用脉动阵列结构传输数据, 减少计算单元对片上缓存的访问, 可以使设计达到极高的频率. 相比于当今热门加速器 GPU, FPGA 的带宽有限, 通常只有 4∼20 GB/s, 大多数文献的优化目标都是降低片外的访存带宽需求. 文献 [16] 首次提出 fusion 的技术, 通过将中间结果一直保留到下一层的计算来减少带宽需求. 文献 [12] 从不同网络层的带宽需求不同角度出发, 来平衡带宽利用.

片上资源优化. 片上资源优化旨在提高 FPGA 内各个资源的利用率. 文献 [30] 指出各个算法对于资源的需要不相同, 因此采用异构算法可以提高 FPGA 的资源利用率. 硬件单元生成的工作在近两年出现的较多, 其目的是提高开发效率, 通过一些高层的描述就能自动生成高效的 RTL 电路. 近年来,

随着深度学习的普及, 卷积神经网络已经在日常生活中使用的端设备上被实现. 但是, 与服务器级设备不同, 边缘设备相对较小, 因此资源有限. 因此, 当我们在边缘设备上实现机器学习算法时, 资源使用和硬件优化的控制起着重要作用. 文献 [25] 针对卷积神经网络探索各种资源方面的优化和设计技术, 其关键思想是 Backward Pipeline Scheduling 和 Latency Balancing, 它们优化 CNN 层之间的流水线, 以显著减少处理单个图像的整体延迟.

硬件单元生成. 近两年来, 随着 FPGA 设计被不断完善, 基于空间卷积算法的 FPGA 加速器设计更加侧重于自动化设计, 顶层设计优化, IP 化. 硬件单元生成旨在缩短开发周期, 可以达到快速部署深度学习到 FPGA 的目标. ACT 实验室 [22, 42] 提出了 DnnWeaver 框架, 可以根据 Caffe [43] 中的高层次的网络描述和 FPGA 具体参数生成加速器. 为了在保持自动化的同时保持性能的优势, DnnWeaver 使用手动优化设计模板生成加速器. 威斯康星大学 (University of Wisconsin-Madison) [23] 针对目前被广泛采用的 OpenCL 开发流程提出核心难点是优化 OpenCL 内核以有效利用 FPGA 中灵活的硬件资源. 传统的 OpenCL 开发工具简单地通过各种编译器选项优化 OpenCL 内核代码, 这不能达到计算密集型和数据密集型工作负载 (例如卷积神经网络) 的理想性能. 相较于普通的 OpenCL 实现, 该工作有 4.41 倍的性能提升, 能在 Intel Ateral Arria-10 平台上达到 1.79 TOP/s 的性能. 伊利诺伊大学厄巴纳 – 香槟分校 (University of Illinois Urbana-Champaign) 的 Chen 等 [24] 提出端到端的自动化工具 AccDNN. AccDNN 集成了高度优化的 RTL IP, 涵盖了网络中多种类型的层. 同时 AccDNN 能够自动地进行设计空间探索, 找出最优的并行策略, 提高数据的重利用.

6 展望

6.1 FPGA 深度学习加速器工具链开发

为了促进深度学习快速发展, 现已有许多软件库与开源框架来帮助人们迅速、高效地执行深度学习网络, 例如 Caffe, Torch, Tensorflow. 这些开源框架提供高级抽象的 API 使人们能简便地运行深度学习网络, 并且提供相应的性能模型来指导在 GPU, CPU 上的实现. FPGA 作为当今主流的加速器之一, 如何与软件层面相互衔接也受到人们高度重视. 另一方面, 高层次综合 (HLS) 工具已经被证明可以将高级抽象语言映射到 FPGA 的硬件设计. 并且随着 HLS 的不断推广, 许多公司的软件工具链都支持 HLS 开发流程. 现有 Xilinx 的 Vivado HLS, Intel FPGA OpenCL SDK, Maxeler 的 MaxCompiler 等工具, 以及 LegUp 采用常用的编程语言, 如 C, C ++, OpenCL 和 Java 促进功能正确的硬件设计的开发. HLS 的发展无疑可以缩小 FPGA 硬件与软件之间的隔阂. FPGA 加速器的一个发展方向是自动化, 模块化. 当应用场景切换时, FPGA 加速器需要快速应对调整, 例如网络结构的变换, 网络层参数的变换. 近年来已经出现部分工作将深度学习中各个计算操作模块化、IP 化以供上层调度. 也有工作开发了完整的从网络配置文件到 FPGA 实现的框架. 表 5 [9, 14, 26, 29, 42, 44~46] 总结了 CNN 部署到 FPGA 工具链的相关总结. 从 FPGA 平台中可以看出, 大部分平台都是 SoC 平台, 采用 CPU+FPGA 的结构, 通过 CPU 来控制 FPGA. 绝大多数的设计都采用了 HLS, 这可以更好地与软件开源框架的接口相吻合. DnnWeaver 没有采用 HLS, 其将给定的高级 DNN 规范转换为表示数据流的指令集架构 (instruction set architecture, ISA), DnnWeaver 编译器包含了优化算法, 可以对 DNN 进行切分、调度和批处理等操作, 以达到最大限度地重用数据, 并最好地利用目标 FPGA 的内存和其他资源. 文献 [45] 采用 SPIRAL 来生成 FFT 硬件单元, SPIRAL 是从 2005 年发展起来的 FFT 硬件语言生成工具. 在 FPGA 与软件接口层面, 大部分工作都采用了已有的开源框架来对 FPGA 进行调度, 也有工作采用特

表 5 CNN 部署到 FPGA 工具链一览表

Table 5 Tool-chains for deploying CNNs on FPGA

	[14]	[44]	[42]	[9]	[26]	[45]	[29]
FPGA	Zynq ZC702	Zynq ZC702 Zynq ZC706	Zynq Z7045 Stratix-V SGSD5	Kintex KU060 Virtex7 VX960T	Stratix-V GSMD5	Intel HARP	Zynq ZC706 Zynq ZCU102
Language	HLS	HDL	ISA	HLS	OpenCL	SPIRAL [46]	HLS
Software interface	Caffe & Torch	Caffe	Caffe	Caffe	Tensorflow	Specific input	C++ program

定的程序来调度 FPGA. 例如文献 [29] 通过手写 C++ 程序来控制 FPGA.

6.2 加速稀疏神经网络

一方面, 当前神经网络发展趋势倾向于更深, 拓扑结构更复杂, 因此, 神经网络稀疏化是将庞大的神经网络部署到 FPGA 上的有效手段. 神经网络能够稀疏化是因为网络本身存在冗余权值, 文献 [47] 证明了 Alexnet 网络和 VGG 网络可以被压缩十倍以上. 尽管稀疏化理论上减少了总体计算量, 但是稀疏化带来的计算不规则性也影响了 FPGA 性能, 加速稀疏卷积神经网络的难点主要在于数据编解码带来的额外开销与计算单元工作时的负载不均衡问题. 目前采用 FPGA 来加速稀疏卷积神经网络的相关工作非常少, 但是在专用集成电路研究领域 (application specific integrated circuit, ASIC) 中, 研究学者已经开始不断提出相关的加速器架构. FPGA 作为神经网络加速器中最有前景的平台之一, 如何在其上高效地加速稀疏神经网络仍是亟待解决的问题, 相信在未来两年内关于这方面的设计会不断被研究学者提出.

参考文献

1 He K M, Zhang X Y, Ren S Q, et al. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015. 1026–1034

2 Ross G, Jeff D, Trevor D, et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014. 580–587

3 Joseph R, Santosh D, Ross G, et al. You only look once: unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. 779–788

4 Karen S, Andrew Z. Very deep convolutional networks for large-scale image recognition. 2015. ArXiv: 1409.1556

5 Anderson A, Vasudevan A, Keane C, et al. Low-memory GEMM-based convolution algorithms for deep neural networks. 2017. ArXiv:1709.03395v1

6 Winograd S. Arithmetic Complexity of Computations. Philadelphia: Siam, 1980

7 Qiu J T, Wang J, Yao S, et al. Going deeper with embedded FPGA platform for convolutional neural network. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2016. 1–10

8 Zhang C, Li P, Sun G Y, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2015. 1–10

9 Zhang C, Fang Z M, Zhou P P, et al. Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. In: Proceedings of International Conference on Computer Aided Design (ICCAD), Austin, 2016. 1–9

10 Li H M, Fan X T, Jiao L, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In: Proceedings of International Conference on Field Programmable Logic and Applications (FPL), 2016. 1–9

11 Wei X C, Yu C H, Zhang P, et al. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In: Proceedings of Annual Design Automation Conference (DAC), Austin, 2017. 1–6

12 Lin X H, Yin S Y, Tu F B, et al. LCP: a layer clusters paralleling mapping method for accelerating inception and residual networks on FPGA. In: Proceedings of Annual Design Automation Conference (DAC), San Francisco, 2018. 1–6
13 Zhao W L, Fu H H, Luk W, et al. F-CNN: an FPGA-based framework for training convolutional neural networks. In: Proceedings of Annual IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), London, 2016. 1–8
14 Venieris S I, Bouganis C. fpagConvNet: a framework for mapping convolutional neural networks on FPGAs. In: Proceedings of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2016. 1–8
15 Zhao R Z, Luk W. Optimizing CNN-based object detection algorithms on embedded FPGA platforms. In: Proceedings of Americas Regional Council meeting (ARC), 2017. 1–13
16 Alwani M, Chen H, Ferdman M, et al. Fused-layer CNN accelerators. In: Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016. 1–12
17 Shen Y M, Ferdman M, Milder P. Escher: a CNN accelerator with flexible buffering to minimize off-chip transfer. In: Proceedings of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2017. 1–8
18 Shen Y M, Ferdman M, Milder P. Maximizing CNN accelerator efficiency through resource partitioning. In: Proceedings of International Symposium on Computer Architecture (ISCA), Toronto, 2017. 1–13
19 Ma Y F, Suda N, Cao Y, et al. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In: Proceedings of International Conference on Field Programmable Logic and Applications (FPL), 2016. 1–8
20 Ma Y F, Cao Y, Vrudhula S, et al. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Monterey, 2017. 1–10
21 Ma Y F, Cao Y, Vrudhula S, et al. Optimizing the convolution operation to accelerate deep neural networks on FPGA. In: Proceedings of Symposia on VLSI Technology and Circuits (VLSI), Hawaii, 2018. 1–14
22 Sharma H, Park J, Mahajan D, et al. From high-level deep neural models to FPGAs. In: Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016. 1–12
23 Zhang J L, Li J. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Monterey, 2017. 1–10
24 Zhang X F, Wang J S, Zhu C, et al. AccDNN: an IP-based DNN generator for FPGAs. In: Proceedings of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2018. 1–8
25 Liu X H, Kim D H, Wu C, et al. Resource and data optimization for hardware implementation of deep neural networks targeting FPGA-based edge devices. In: Proceedings of Annual Design Automation Conference (DAC), San Francisco, 2018. 1–6
26 Guan Y J, Liang H, Xu N Y, et al. FP-DNN: an automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In: Proceedings of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2017. 1–8
27 Cong J, Xiao B J. Minimizing computation in convolutional neural networks. In: Proceedings of International Conference on Artificial Neural Networks, Hamburg, 2014. 1–10
28 Shen J Z, Huang Y, Wang Z L, et al. Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Monterey, 2018. 1–10
29 Lu L Q, Liang Y, Xiao Q C, et al. Evaluating fast algorithms for convolutional neural networks on FPGAs. In: Proceedings of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2017. 1–8
30 Xiao Q C, Liang Y, Lu L Q, et al. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In: Proceedings of the 54th Annual Design Automation Conference (DAC), 2017. 1–6
31 Aydonat U, O'Connell S, Capalija D, et al. An OpenCL deep learning accelerator on Arria 10. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Monterey, 2017. 1–10

32 Podili A, Zhang C, Prasanna V, et al. Fast and efficient implementation of convolutional neural networks on FPGA. In: Proceedings of Annual IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2017. 1–8

33 Lu L Q, Liang Y, Xiao Q C, et al. Evaluating fast algorithms for convolutional neural networks on FPGAs. In: Proceedings of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2018. 1–14

34 Zhang C, Prasanna V. Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Monterey, 2017. 1–10

35 Ko J H, Mudassar B, Na T, et al. Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation. In: Proceedings of Annual Design Automation Conference (DAC), Austin, 2017. 1–6

36 Wei X C, Liang Y, Li X H, et al. TGPA: tile-grained pipeline architecture for low latency CNN inference. In: Proceedings of 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018. 1–8

37 Lu L Q, Liang Y. SpWA: an efficient sparse winograd convolutional neural networks accelerator on FPGAs. In: Proceedings of the 55th Annual Design Automation Conference (DAC), 2018

38 DiCecco R, Lacey G, Vasiljevic J, et al. Caffeinated FPGAs: FPGA framework for convolutional neural networks. In: Proceedings of International Conference on Field-Programmable Technology (FPT), 2016. 1–8

39 Suda N, Chandra V, Mohanty A, et al. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2016. 1–10

40 Motamedi M, Gysel P, Akella V, et al. Design space exploration of FPGA-based deep convolutional neural networks. In: Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC), 2016. 1–6

41 Venieris S I, Bouganis C S. Design space exploration of FPGA-based deep convolutional neural networks. In: Proceedings of IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2016. 1–8

42 Sharma H, Park J, Amaro E, et al. DnnWeaver: from high-level deep network models to FPGA acceleration. In: Proceedings of the Workshop on Cognitive Architectures, 2016

43 Jia Y Q, Shelhamer E, Donahue J, et al. Caffe: convolutional architecture for fast feature embedding. 2014. ArXiv: 1408.5093

44 Wang Y, Xu J, Han Y H, et al. DeepBurning: automatic generation of FPGAbased learning accelerators for the neural network family. In: Proceedings of the Design Automation Conference (DAC), 2016. 110–115

45 Zeng H Q, Chen R, Zhang C, et al. A framework for generating high throughput CNN implementations on FPGAs. In: Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA), 2018. 117–126

46 Puschel M, Moura J M F, Johnson J R, et al. SPIRAL: code generation for DSP transforms. In: Proceedings of the IEEE, Special Issue on Program Generation, Optimization, and Adaptation, 2005. 232–275

47 Han S, Mao H Z, Dally W J. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. 2015. ArXiv:1510.00149

Accelerating convolutional neural networks on FPGAs

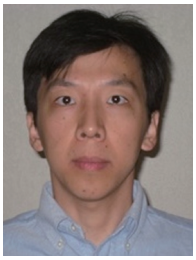
Liqiang LU[1], Size ZHENG[1], Qingcheng XIAO[1], Deming CHEN[2] & Yun LIANG[1*]

1. *Electronics Engineering and Computer Science, Peking University, Beijing* 100871*, China*;
2. *University of Illinois at Urbana-Champaign, Champaign* IL 61820*, USA*
* Corresponding author. E-mail: ericlyun@pku.edu.cn

Abstract In recent years, convolutional neural networks (CNNs) have become widely adopted for computer vision tasks. FPGAs have been adequately explored as a promising hardware accelerator for CNNs owing to their high performance, energy efficiency, and reconfigurability. However, previous FPGA methods, which are based on the conventional convolutional algorithm, are often bounded by the computational capability of FPGAs. This paper first introduces four convolution algorithms: 6-loop algorithm, general matrix-matrix multiplication (GEMM), Winograd algorithm, and fast Fourier transform (FFT) algorithm. Then, we present the implementations of these algorithms at home and abroad, and also introduce their corresponding optimization techniques.

Keywords CNN, FPGA, convolution algorithm, fast algorithm, Winograd, FFT

Liqiang LU was born in 1994. He received his B.S. degree in 2017 from the Institute of Microelectronics, Peking University, Beijing, China. He is currently a Ph.D. candidate in the School of EECS, Peking University. His research focuses on algorithm-level and architecture-level optimizations of FPGA for machine learning applications.



Yun LIANG was born in 1981. He received his Ph.D. degree in Computer Science from the National University of Singapore in 2010 and worked as a research scientist in UIUC before joining PKU. He is an associate professor (with tenure) in the School of EECS, Peking University, China. His research focuses on heterogeneous computing (GPUs, FPGAs, and ASICs) for emerging applications, such as artificial intelligent and big data, computer architecture, compilation techniques, programming model and program analysis, and embedded system design.