UltraAcc：基于FPGA流水架构的低功耗高性能CNN加速器定制设计

包振山 郭俊南 张文博 党鸿博
（北京工业大学信息学部 北京 100024）

摘 要 近年来，卷积神经网络（Convolutional Neural Network，CNN）在目标检测、场景分割、图片分类等领域的应用中取得了举世瞩目的成绩，然而随着应用对精度要求的不断提升、模型参数量不断增加、所需计算量不断增大，这使得在“边”“端”侧资源有限的平台上部署低延时应用极具挑战. 虽然采用GPU可以完成CNN模型加速计算的理论验证，但受限于GPU定制改造成本以及其自身功耗，无法在实际低功耗系统中应用. 相比而言，作为低功耗高性能系统，FPGA平台具备高性能计算能力以及硬件可重构的特点，适于完成CNN加速. 当前利用FPGA可重构性的定制计算技术虽然可整合加速器以应对多变的CNN应用场景，调整加速器结构以适配应用计算保证功耗效率. 然而，现有卷积神经网络FPGA加速器的瓶颈在于卷积神经网络算法适配性不佳，由此会导致计算间隙大、时延浪费、计算资源使用率低的问题. 本文针对CNN算法因局部参数共享所导致的计算密集的特点，重新组织数据流结构以适应并行运算. 针对资源有限制的FPGA板卡，本文自底向上定制了矩阵乘法、卷积计算、池化计算等单元，最终组成Ultra加速器（Ultra Accelerator，UltraAcc）；同时，本文设计了评估模型进行超参数调优，从底层单元到计算层单元再到整个计算链都做了对存储资源、计算资源、运行时延的评估，再配合神经网络训练的精度，从而实现在软件硬件两个方面平衡优化整个应用系统. Ultra加速器在Ultra96板卡上平均吞吐量可以达到126.72 GOPs，是IEEE/ACM DAC-SDC’19冠军方法的5.47倍. 采用Ultra加速器本团队参与了DAC-SDC’20低功耗目标检测的比赛，最终以精度IoU 0.65、速度FPS 212.73、消耗能量1.64 kJ夺得2020年该赛项冠军.

关键词 加速器；卷积神经网络；现场可编程门阵列；数据流水技术；软硬件协同
中图法分类号 TP391 **DOI号** 10.11897/SP.J.1016.2023.01130

UltraAcc：A Customized Low Power and High Performance CNN Accelerator with Dataflow on FPGAs

BAO Zhen-Shan GUO Jun-Nan ZHANG Wen-Bo DANG Hong-Bo
（*Faulty of Information Technology, Beijing University of Technology, Beijing* 100024）

Abstract Convolutional Neural Network (CNN) has remarkable application effect in object detection, semantic segmentation and image classification in recent years. In order to meet the requirements of high precision, CNN models with deep layers need to be constructed. Due to the large number of parameters of the CNN and its intensive computational demands, it is a great challenge to the deployment of CNN applications with low latency requirements on edge devices which are resource-limited. Although GPU can be used to complete theoretical verification of accelerated computation of CNN model. Due to the limitation of GPU customization cost and power consumption, it cannot be applied in the actual low-power system. In contrast, as a low power consumption and high performance system, FPGA has the characteristics of high performance

收稿日期：2022-05-24；在线发布日期：2023-01-10. 本课题得到国家自然科学基金（62072016）资助.**包振山**，硕士，副教授，中国计算机学会（CCF）会员，主要研究方向为智能计算系统、高效深度学习. E-mail：baozhenshan@bjut.edu.cn. **郭俊南**，硕士研究生，中国计算机学会（CCF）会员，主要研究方向为FPGA架构、神经网络加速器. **张文博**（通信作者），博士，副教授，中国计算机学会（CCF）会员，主要研究方向为机器学习算法和硬件系统协同设计. E-mail：zhangwenbo@bjut.edu.cn. **党鸿博**，硕士研究生，中国计算机学会（CCF）会员，主要研究方向为卷积神经网络和深度学习.

computing capability and reconfigurability, which are suitable for customized computing of CNNs. The method to solve the acceleration problem is to use the customized computing technology with FPGA reconfigurability. We can use the composable accelerator to deal with various CNN application scenarios and adjust the accelerator structure to suit the application to ensure power consumption efficiency. The bottleneck of the existing CNN accelerator on FPGA lies in the poor adaptation of CNN algorithm, which leads to the problems of large computing gap, the waste of latency and low utilization of computing resources. In this paper, we reorganize the dataflow structure to adapt to CNN parallel operation. According to the limited FPGA resources, the matrix multiplication, convolution calculation, pooling calculation and other units were customized from the bottom up to top, and the Ultra accelerator (UltraAcc) is proposed. An evaluation model is designed for hyperparameter tuning. From the bottom unit to the computing layer unit and then to the whole computing chain, storage resources, computing resources and latency are evaluated. With the precision result of CNN training, the whole application system is balanced and optimized from both software and hardware. The UltraAcc can achieve an average throughput of 126.72 GOPs on the Ultra96v2, 5.47 times higher than the first place method in IEEE/ACM DAC-SDC'19 on the same platform. The UltraAcc was used to participate in the DAC-SDC'20. And we won the first prize with accuracy of IoU 0.65, speed of FPS 212.73 and energy consumption of 1.64 kJ.

Keywords accelerator; CNN; FPGA; dataflow; hardware-software co-design

1 引 言

卷积神经网络(Convolutional Neural Network, CNN)在计算机视觉领域应用的突出表现引起了业界的极大关注[1].然而,为了达到高精度要求,需要构建层数较深和参数量巨大的CNN模型,这给低时延、资源有限的平台的部署带来了极大的挑战[2-4].

虽然在算法验证阶段,CNN通常使用并行处理器(例如GPU)进行加速,但它对功耗的高需求也限制了其应用领域的拓展.在低功耗高性能系统设计中,FPGA和ASIC一直是加速器设计的主流平台.然而加速器的设计一直伴随着两大问题——资源利用率低和加速器应用范围受限.虽然一些加速设计平台采用了既有硬件模块(比如TPU中采用脉动阵列),但这类硬件模块相对固定,需要有针对性地调整数据流才能充分发挥其高效运作能力,应对灵活多样的应用时表现并不能尽如人意.相较而言,由于FPGA在结构可定制化计算和可重构性方面有着与生俱来的优势[5],因而成为了CNN加速器设计的理想平台[6-7].

CNN高效加速本质上是对算法内在数据流和并行性两方面进行优化.常见的加速器设计分为两种,循环平铺加速器和数据流水加速器.循环平铺加速器注重并行度优化,通常抓取固定尺寸的数据,搬运数据至片上进行计算,如北大CECA实验室Zhang等人的工作[8].然而,这类固定尺寸的计算单元很难高效复用计算类型多样的CNN,因而产生了一些"无效计算",进而导致不必要的时延和功耗.数据流水加速器注重数据流结构,将整个网络放置到FPGA片上进行数据流水处理,可以发挥出FPGA板卡的全部性能.比如,Xilinx团队的FINN[9]将神经网络压缩为二值网络,且使待处理的所有数据满足流水结构.在其定制的二值神经网络下,FINN运行速度可以达到每秒21.9k帧.然而,针对CNN的数据流水加速器设计存在三个困难:(1)卷积算法数据依赖性高,中间结果的缓存是限制并行加速的最大障碍;(2)基础计算单元灵活性不足,不能适配同一神经网络中的不同计算层;(3)加速器可考量指标众多,欠缺有针对性、系统化的定制优化策略.

针对上述的问题,本文提出以数据流水为架构基础,加入循环平铺设计的Ultra加速方案.本方案的创新性为以下三点:

(1)针对数据调用特点设计了滑动窗口重排模块,以消除数据依赖性;

(2)采用灵活位宽和动态资源调度的方式提高基础单元灵活度,以增强并行加速粒度;

(3)建立优化模型评估机制,以便合理、高效地

调整模型超参数.

应用上述三个技术，我们按照图 1 的工作流程完成了 Ultra 加速器的设计. 首先，进入“定制神经网络”环节. 在这个阶段主要是依据 FPGA 资源限制和时序要求，对网络模型进行压缩，以减少计算量. 而后，进入“评估模型”环节. 在这个阶段主要考虑优化并行处理，既可以采用针对性的 CNN 底层乘加运算的优化设计，也可以采用 CNN 层间流水优化设计，以加快并行处理效率. 最后，通过反复迭代“优化模型”和“加速器配置”形成“加速器模板”. 不难看出，在图 1 所述流程中，评估模型处于中间环节，可以实现连通软、硬件设计的桥梁，这部分的设计需要考虑平衡精度、速度、资源使用等指标. 采用上述设计方式，能够有效贯穿硬件设计和软件设计，一方面考虑算法的性能需求压缩模型，另一方面根据应用平台的资源限制，通过定制化设计减少“无效计算”，进而有效达成整个网络模型的流水线优化设计.

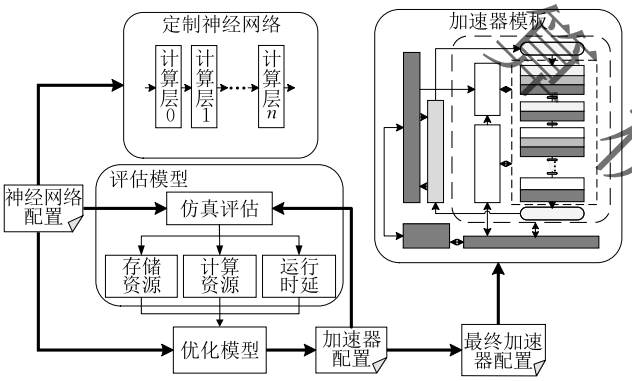

图 1　Ultra 加速器的工作流图

应用上述思想，我们团队在 IEEE/ACM DAC-SDC’20 低功耗目标检测比赛中获得冠军. 实验结果显示，Ultra 加速器在该赛项指定硬件平台 Ultra96 板卡上平均吞吐量可以达到 126.72 GOPs，是上届冠军方法的 5.47 倍. 在 DAC-SDC’20 比赛中，精度 IoU 达到 0.65、速度 FPS 达到 212.73，功耗 1.64 kJ.

综上所述，本文的贡献有以下三点：

(1) 根据卷积算法因局部共享参数而导致计算密集的特点，重新组织数据流结构降低数据依赖性，以适应并行计算.

(2) 提出了定制化 Ultra 加速器的设计思路. 在资源限定的 FPGA 平台环境上，设计了软-硬件贯穿的“定制神经网络-评估模型-迭代优化”的工作流程，从而达成以流式数据特点为核心的加速器定制化设计.

(3) 应用 Ultra 定制化加速器设计方案，分别针对 DAC-SDC 竞赛和 VGG 经典分类问题两个应用场景，给出了相应的解决方案.

本文第 2 节是相关的文献工作；第 3 节详细展示 Ultra 加速器的设计；第 4 节实验验证 Ultra 加速器性能；第 5 节是全文的总结.

2　相关工作

本节介绍了卷积算法的常用加速优化手段、软硬协同设计 CNN 加速器的发展情况以及 IEEE/ACM DAC-SDC 中加速器应用情况.

2.1　卷积循环嵌套优化

在 CNN 实现中主要有大量卷积运算需要频繁处理，因此对于卷积算法的优化处理是 CNN 加速器设计的关键问题. 卷积算法是一种典型的嵌套结构计算，其循环体较多，对嵌套结构优化处理是卷积加速的常见方法，包括循环展开、循环平铺、数据流优化等方法.

循环展开是一种提高计算并行性的技术. Nuzman 和 Zaks[10] 讨论了通过在单指令多数据(Single Instruction Multiple Data，SIMD)体系结构上应用向量化处理技术来展开循环并行性的方法. 北京大学 CECA 实验室的工作[8] 主要研究了不同类型数据上的循环展开与数据共享之间的关系. 通过分析得到循环展开方案，调整卷积算法中的循环顺序，去除部分数据依赖，保证其中两层循环可以完全展开，从而实现对 CNN 算法加速.

循环平铺是利用数据块在计算过程中经常出现的局部性特点来重用数据，以提高数据利用率. 具体方法是，将处理过程中所有的数据划分为更小的数据块，将其依次存储在本地缓冲区中，根据算法特点，增加其被重用率，以减少数据调入的置换时间. 在 Chen 等人的 Eyeriss 加速器系列研究[11-12] 中，循环平铺有效地减少了加速器与主存系统之间的数据转换，即芯片的片上-片下通信.

数据流优化是从数据整体角度入手的 CNN 嵌套循环优化方式. DnnWeaver[13] 是一个基于数据流的架构编译器，它通过利用数据流架构的属性，自动生成一个针对给定的 CNN 优化的加速器. 与 DnnWeaver 相比，有些基于 FPGA 的解决方案将数据流架构与网络流水线架构相结合，例如 MALOC[14] 提供了一种将所有层流水部署在片上的方法，DNNBuilder[15] 也是

自动化流水结构部署的框架. fpgaConvNet[16]是一种用于在 FPGA 上的优化卷积神经网络的端到端框架. 基于同步数据流(Synchronous Dataflow,SDF)范式定义了一组 SDF 转换方法,以便有效地指导加速器架构设计空间. 然而,由于每个单独的计算层都需要额外的资源开销,导致资源难以平衡利用,这是这类加速器需要考虑的主要问题. Shen 等人[17]提出的工作考虑了不同层独特的计算数据流和资源需求进行建模,从而更好地分配资源.

2.2 软硬协同设计

软硬协同设计方法综合考虑了软件算法应用指标要求和高效部署硬件架构的工程需求[18]. 在 CNN 加速研究领域,最初研究人员更多地关注借助于硬件资源约束模型的参数表示,即如何采用量化等方式压缩模型,进而将其部署在硬件平台上. 比如,Hiroki 等人[19]通过引入二值化网络作为特征提取的主干网络来重塑 YOLOv2[20],从而充分利用 FPGA 的低位宽计算的优势. Xilinx 研究团队提出的 FINN-R 框架[21]则进一步将量化神经网络(QNN)集成到基于 YOLO 的目标检测系统中.

随着研究的不断深入,研究人员在 CNN 优化加速设计方面采用更为深入的软硬协同设计方法. Synetgy[22]以 ShuffleNet 为基础,通过更改模型结构以适合 FPGA 加速,并设计了配套的计算单元,从而加速了卷积神经网络推理. Light-OPU[23]采用软硬件设计重新定制了轻量级基础操作单元,以实现高效加速,同时考虑了基础操作和传统卷积运算的计算引擎匹配问题,以提高资源使用效率和整体电力效率. Hao 等人[24]强调了自动化软件模型和硬件加速器协同设计实现加速器是神经网络加速的发展方向,而后 Hao 等人[25]又提出了第一个 FPGA/DNN 协同设计框架,包括面向硬件的自底向上的 DNN 模型设计,以及 DNN 驱动的自顶向下的 FPGA 加速器设计. 通过这种协同设计流程搜索的 SkyNet[26]在嵌入式 GPU 和 FPGA 上精度和吞吐量都取得了当时最优异的成绩.

2.3 DAC-SDC 赛事加速器设计

DAC-SDC 是 CCF A 类 IEEE/ACM 顶级 EDA 会议 Design Automation Conference(DAC)承办的基于 FPGA 平台的深度学习算法加速器设计大赛[27]. 该赛项的任务是设计轻量化的目标检测算法,从而探索在低功耗平台上实现无人机目标检测应用的性能上限. 鉴于 FPGA 在低功耗、可重构设计方面的优势[28],主办方将其设定为该赛项的主要应用平台. 硬件 FPGA 平台先后采用 Xilinx 提供的 Xilinx PYNQ Z1 板卡和 Xilinx Ultra96v2 板卡. 赛题属于单一目标检测任务,是无人机应用中最重要的任务之一[29]. 数据集的图片均由 DJI-UAV 无人机采集,展现了真实无人机应用中的数据特点. 不同于其他计算机视觉类的挑战赛,如 ImageNet[30]和 PASCAL VOC[31]只关注准确度指标,DAC-SDC 更看重吞吐量、功率和检测精度所组成的综合性能评估结果,其在关注检测精度的同时,充分考虑了无人机应用中实时化响应、资源/能量受限方面的实际需求,鼓励参赛者采用软硬件协同方式,针对实际性能需求构建定制化系统方案.

经过 2018 年和 2019 年的 DAC-SDC 竞赛,各参赛队伍已逐渐形成了一套较为成熟的设计思想,即高效应用循环平铺加速器的方案以达成比赛所需要求. 以 DAC-SDC'19 的冠军作品 SkyNet[26]为例,其采用循环平铺策略,通过搜索得到该方案设定下的最优结果. 而 DAC-SDC'20 并没有止步于此,他们继续 SkyNet 的设计思想,将速度提升至 52.4,精度达到 0.731. 然而,这类设计受循环平铺设计固有思路影响,仍然存在较多的"无效计算"(SkyNet 加速器的实际执行计算量是网络本身计算量的 2.5 倍左右),因而还有一定的提升空间.

本文中的 UltraAcc 在 DAC-SDC'20 夺得冠军,这主要得益于其注重数据流结构,在 FPGA 片上部署计算流水线,给每个计算单元分配独立的资源,避免计算冲突,形成高资源利用率、高并行度的加速系统. 在后续两年该赛项的比赛中,一些队伍延续该思想,在加速器设计方面又取得了一些新的提升.

本文研究软硬件协同数据流水架构的加速器设计方法. 其中,在硬件设计方面,本文针对卷积计算特点,优化设计了基础计算单元,并由此自底向上进行优化部署;在软件算法设计方面,本文则基于硬件资源特点,定制化了相应的量化神经网络. 根据上述思想,本文设计了 Ultra 加速器,以下将详细介绍.

3 Ultra 加速器设计

3.1 流水架构加速器设计总述

Ultra 加速器采用数据流水架构模式,由 ARM 和 FPGA 协作完成. 图 2 展示了该加速器的基本架构,该工作过程主要涉及五个模块:ARM/FPGA 共

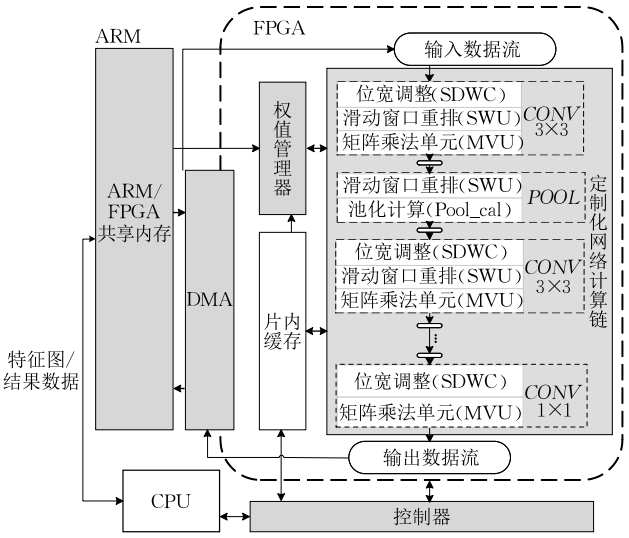

图 2　Ultra 加速器结构设计

享内存、DMA、控制器、定制化 CNN 计算链和权值管理器，以下将分别介绍.

ARM/FPGA 共享内存中存放待测数据及加速器所需的配置参数. DMA 用于 ARM 和 FPGA 之间高速数据流通信，其传输流式数据，在 ARM 端或 FPGA 端接收到数据后再进行转换. 图 3 展示了 ARM 端图像特征数据转换为数据流放入 DMA 的过程，其中 IH、IW、IC 分别表示特征图高度、宽度和通道数. 控制器部署在 ARM 端，负责将数据从共享内存放入 DMA，从而实现配置通信地址、激活加速器及通过调节特征图数量($batch$)控制流量. CNN 计算链是完成加速计算的核心模块，它由多层计算单元串联. 计算单元自底向上分为底层单元、一级流水单元、二级流水单元. 底层单元包括滑动窗口重排、矩阵乘法单元、流水数据转换单元；一级流水单元由底层单元组成，包括 $conv3\times3$、$conv1\times1$、$pool$ 等；完整的计算链由多个底层流水单元和一级流水单元组成，合称二级流水单元. 层与层之间设有缓存模块，以解除层间依赖关系实现流水结构. 每个单元的输入输出均为特征数据流，为了后文描述方便，表 1 规定了一些 CNN 计算中使用变量.

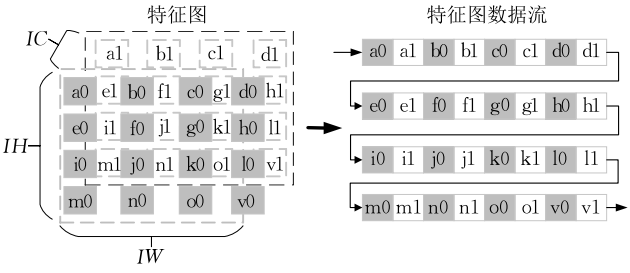

图 3　特征图转换为数据流

表 1　卷积神经网络计算变量定义

变量	解释
$batch$	代表一组特征图的个数
IW	代表输入特征图的宽度
IH	代表输入特征图的高度
IC	代表输入特征图的通道数
OW	代表输出特征图的宽度
OH	代表输出特征图的高度
OC	代表输出特征图的通道数
$width_{elem}$	代表特征数据占用位宽
$width_{weight}$	代表权值数据占用位宽
$width_{IN}$	代表输入数据流的位宽
$width_{OUT}$	代表输出数据流的位宽
K	代表滑窗的尺寸
S	代表滑窗的移动步长
P	代表填充像素宽度

Ultra 加速器共有三种权值加载模式，它们分别是全片上模式、全片外模式和混合模式. 全片上模式是将所有权值预先保存在片内缓存. 全片外模式是将所有权值存放在共享内存. 混合模式是片内缓存和共享内存中各存储一部分. 权值管理器根据卷积网络计算链中计算层的需要，从共享内存或片内缓存中获取权值地址并发放给对应的计算层.

3.2　数据流水架构

CNN 计算链采用流水设计以实现所有计算层高效并行. 为了达成这一目标，需要在系统构建阶段解除层间、层内的数据依赖，同时为每层配备足够的存储和计算资源.

(1) 减少层间数据依赖：单个计算层前后配置流水缓存管道. 层间数据依赖主要是指某层的计算凑够之前一层的结果. 当存在层间数据依赖时，为了促成每层都更高效地运行，需要在每层的前后配置相应的流水缓冲区，以减少等待时间. 随着应用不断深入，我们发现这样的设计不仅可以有效降低层间的数据依赖，同时也可以规范计算层接口以便完成快速开发.

(2) 减少层内数据依赖：针对算法数据特点，利用片上缓存和重排数据的方式解决. 层内数据依赖是指完成一次计算需要请求层内其他数据才能完成. 在理想情况下，只需一个操作周期就可以获取到所依赖的所有数据. 在流水缓存管道的设计前提下有以下两个问题：① 采用内存寻址的方式获取数据需要较高的操作周期数；② 不同操作可能多次请求同一数据，这类重复获取是不必要的. 为了解决上述两类问题，需要结合具体算法的数据依赖特点，利用片上缓存和重排列数据的方法以降低算法的依赖性，具体操作方法在 3.3.1 节详细介绍.

(3)每个计算层分配独立的计算资源和缓存资源.为了避免层间出现数据读写冲突,需要为每个计算层配备独立的缓存资源;为了减少冗余计算,需要为每个计算层配备独立的计算资源,以确保每个计算层在加速器开启后满负荷计算.

根据上述设计原则,Ultra 加速器充分利用底层硬件资源实现了高效计算.表 2 展现了 Ultra 加速器与单一循环平铺加速器在理论性能方面的对比.

表 2 加速器理论性能对比

比较情况	Ultra 加速器	循环平铺加速器
第 i 计算层运行时间	L_{xi}	L_{si}
网络整体运行时间	$\max(L_{xi})$	$\sum L_{si}$
计算资源使用率	$\frac{\sum C_{xi}}{C}$	$\frac{C_{si}}{C}$
存储资源使用率	$\frac{\sum M_{xi}}{M}$	$\frac{M_{si}}{M}$

如表 2,假定系统是一个含有 n 个计算层的网络计算链,其中 FPGA 平台的存储资源为 M、计算资源为 C,待完成 k 组数据的计算任务.若采用 Ultra 加速器,第 i 计算层所用存储资源为 M_{xi},计算资源为 C_{xi},运行时延为 L_{xi}.由于运行时每层都满负荷运算,因此满足以下条件:$\sum M_{xi} \leq M, \sum C_{xi} \leq C$.若采用循环平铺思想设计的加速器,第 i 个计算层所用存储资源为 M_{si},计算资源为 C_{si},运行时延为 L_{si}.由于只有单个计算层在进行运算,因此满足以下条件即可,$M_{si} \leq M, C_{si} \leq C$.

关于运行时延的对比. 如图 4 所示,可以看出 Ultra 加速器更适合每层计算量较为匀称的轻量级小网络.

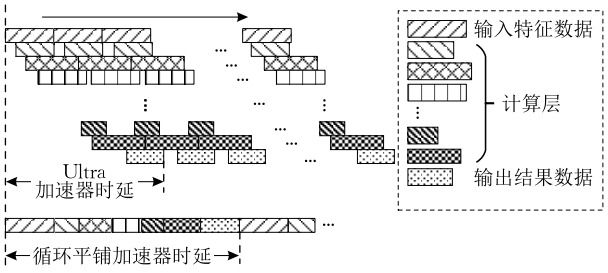

图 4 Ultra 加速器时延与循环平铺加速器时延对比

假定数据依赖所带来的时延远小于计算层运行时延,网络的整体运行时延几乎等同于 $\max(L_{xi})$.通常情况下,在运行时间上,Ultra 加速器比循环平铺加速器更具有优势.

关于资源使用情况的对比. 由于循环平铺加速器只需要满足当前计算需求,理想情况是将所有资源全部用于单层计算.而 Ultra 加速器要满足所有计算层同时运算,所以通常情况下 $M_{si} > M_{xi}, C_{si} > C_{xi}$.与之相关的单层计算时间 $L_{si} < L_{xi}$.但是由于硬件设备固化后不能做到完全复用.Ultra 加速器尽可能达到 $\sum M_{xi} \approx M$, $\sum C_{xi} \approx C$.

相比而言,循环平铺加速器在理想下被希望能实现对于任意 si,$M_{si} \approx M, C_{si} \approx C$,但这个想法很难实现.因此,相比之下 Ultra 加速器更容易实现资源的充分利用.

3.3 计算链设计

CNN 计算链由多层计算单元串联而成.为了便于进行优化处理,我们将计算单元自底向上分为底层单元、一级流水单元、二级流水单元.其中底层单元的优化设计中采用了滑动窗口重排、矩阵乘法单元优化设计和其他底层单元优化设计.以下将分别对这些内容详细介绍.

3.3.1 滑动窗口重排

CNN 中计算层的一大特点是基于一个 *kernel* 尺寸的滑窗依次移动进行局部计算,例如卷积计算、池化计算.每进行一个 *kernel* 尺寸的滑窗计算所依赖的数据是不连续的.这导致算法在层内的数据依赖性较高.设计滑动窗口重排(Sliding Window Unit,*SWU*)模块就是为了重新组织生成合适的数据流以降低数据依赖性便于 CNN、池化的并行计算.

图 5 以滑窗 *kernel* 为 2、计算步长 S 为 1 的计算

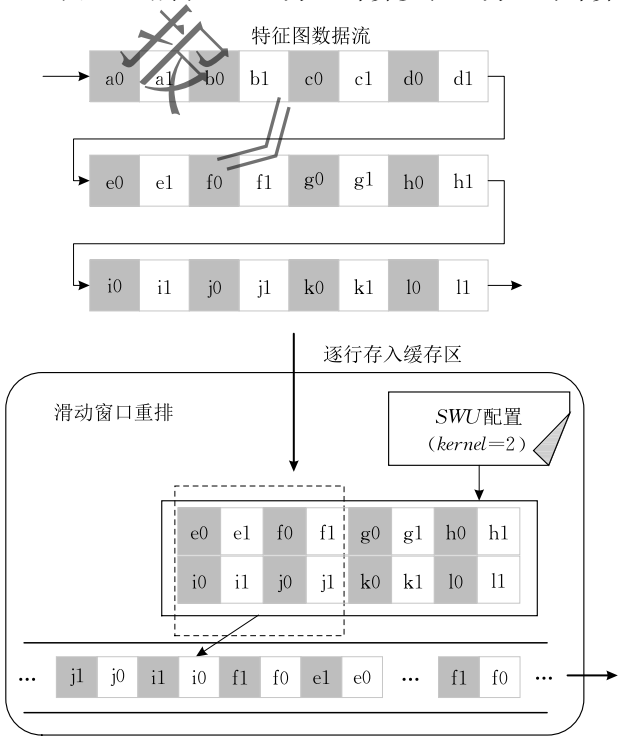

图 5 *SWU* 重排特征数据

任务为例，展示了 *SWU* 的特征数据存放情况. 其中流入数据包含通道数、特征图宽度、特征图高度，在此示例中分别为 2、4、4.

第一步，需要根据流入数据格式，确定 *SWU* 配置，申请足够的空间. 因为这种情况下 *kernel* 为 2，数据依赖发生在 2 行之间. 所以需要缓存 2 行内所有特征数据，需要开辟 *kernel*×通道数×特征图宽度的空间.

第二步，将数据依次放入缓存区. 根据滑窗尺寸 *kernel* 依次获取每次计算需要的数据并放入到输出流内.

第三步，更新缓存区数据. 当缓存区内所有数据放置完毕，根据步长加载新数据到缓存区. 例如当前步长是 1，那么当前缓存内的第一行已经不存在依赖关系，而第二行数据仍存在依赖性. 因此只需将新数据更新至存缓区第一行即可.

经过上述三步处理后，所有的计算所需的数据都如图 6 所示按卷积计算规则顺序存放，数据的依赖性降到了最低. 这类操作的本质是将数据复制了多份以达到高效获取数据的目的.

图 6　*SWU* 重排前后特征数据对比

此外，在 CNN 中还需要减少权值依赖关系. 图 7 展示了利用权值管理器对权值进行重排列的过程，其中 *IC*、*OC*、*K*、*K* 分别表示权值数据四个维度的尺寸. 重排操作包括调整权值的读取顺序，以及将同一滑窗内调用的权值进行位宽拼接. 因此可以实现一次读取多个数据，从而实现与 *SWU* 模块整理出的数据流高效匹配，进而完成相应的计算.

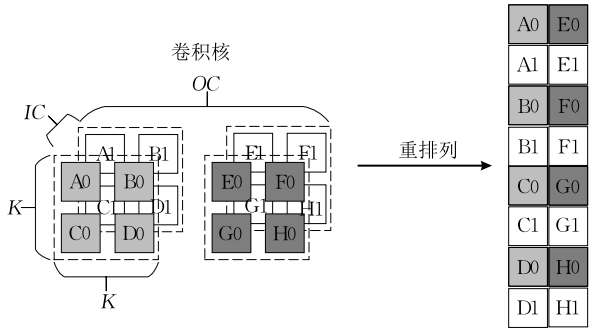

图 7　权值管理器重排权值数据

这种方法的本质是利用片上空间资源作为缓存，以减少从片外数据流中获取数据花费的时间. 片上缓存的读取访问更为灵活，可以有效避免与片外通信造成的时间间隙. 根据卷积、池化类型的算法，一个计算任务需要 $batch\times\left\lfloor\frac{IH-K}{S}+1\right\rfloor\times\left\lfloor\frac{IW-K}{S}+1\right\rfloor\times IC\times K^2$ 次数据读取. 在 *SWU* 中，每次读取时将数据进行缓存，根据算法复制多份并按序送入数据流，之后再将数据移出缓冲区，由此只需要进行 $batch\times IH\times IW\times IC$ 次数据读取.

3.3.2　矩阵乘法单元优化设计

根据以上描述，权值管理器分别从共享内存和片上缓存中获取权值数据，而后进行权值重排. 以下分别从向量计算单元和张量计算单元两方面分别介绍对矩阵乘法单元优化设计的方法.

(1) 向量计算单元

事先可以获取每个重排列数据的位宽拼接个数，记为 *SIMD*. *MVU* 根据 *SIMD* 申请乘加运算组(Multiplication and Addition Computations，MACs) 的计算资源. 而后将特征数据与拆解后的权值数据放入乘加运算组进行计算. 由此可以实现图 8 所示的在同一时刻针对一个特征数据完成与 *SIMD* 个向量数据的乘加操作.

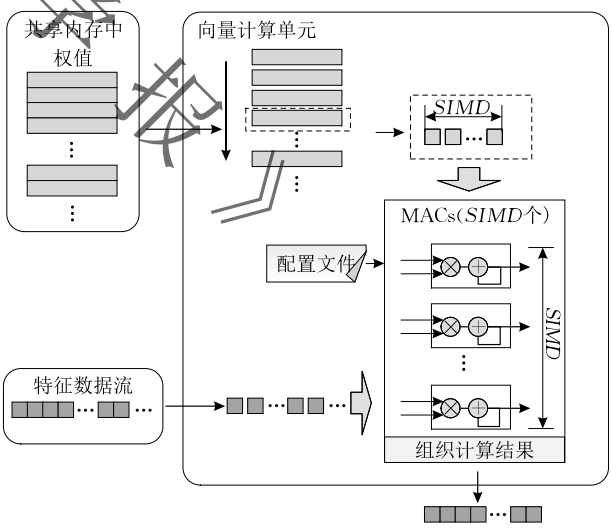

图 8　向量计算 *MVU* 单元

(2) 张量计算单元

相比向量计算单元中共享内存总线 AXI 传输的方式，片上缓存可以将权值数据拆分到多个独立存储地址. *MVU* 进行多通道的读取数据，可以在同一时间进行 *SIMD*×*PE* 次运算操作. 其中使用的通道数量即为并行度，记为 *PE*. *MVU* 根据 *SIMD*×

PE 申请 MACs 计算资源. 一方面获取个数为 PE 的特征数据向量. 另一方面每个读取通道将权值进行位宽拆解,得到单通道权值向量. 之后,如图 9 所示,再将通道组合,获得 $SIMD\times PE$ 的权值数据张量. 由此,当权值存储在片上缓存时可以并行实现特征数据向量与权值张量的乘加运算操作.

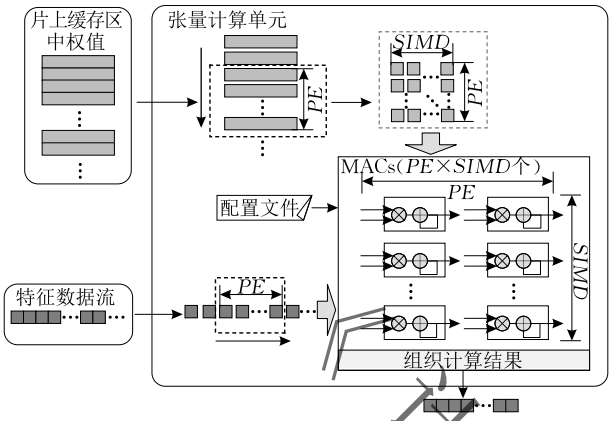

图 9 张量计算 MVU 单元

3.3.3 其他底层单元优化设计

图 10 展示了位宽调整单元调节数据流中单个数据的位宽大小的方式,该模块的作用是控制中间数据流的位宽大小,间接控制数据流中的数据个数,控制每个单元的流入流出速度,以保证中间缓存区可承受所有的中间结果数据.

池化核心单元对应一级流水单元最大池化的底层操作,即 $k\times k$ 个数据中选出最大值,并行方式实现类似乘加计算单元.

填充单元与卷积运算相关,主要是为了解决卷积过后图像缩小的问题,在滑动窗口重排前,向数据流中对应位置补充 0 即可.

数据流调整单元的实质是对数据流进行操作,主要用于实现神经网络中跳层、残差结构,其包括三种操作:将一个数据流复制为两个数据流、将两个数据流按位相加合成一个数据流、将两个数据流直接拼接成一个数据流.

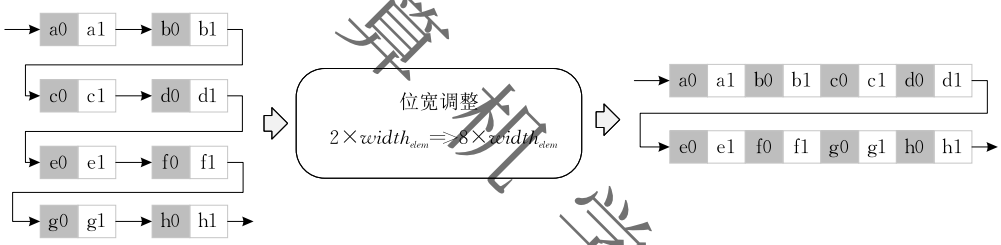

图 10 位宽调整单元

3.3.4 一级和二级流水单元

一级流水单元定义为计算层单元,按神经网络中的常用算法进行划分,包括卷积、池化、全连接计算等. 每个一级流水单元都是由底层单元组成的,中间用流水缓存进行连接,以实现单元内的流水,具体实现组合见表 3.

表 3 一级流水单元流水化处理

一级流水单元	流水化处理
$K\times K$ 卷积	填充→特征数据重排→位宽调整→标准卷积乘加计算→位宽调整
2×2 最大池化	特征数据重排→位宽调整→池化核心
全连接计算	位宽调整→标准卷积乘加计算→位宽调整

由于 1×1 卷积数据依赖只有 1 个通道的数据并行,不会造成卷积后图像缩小的问题,因此无需进行特征数据重排和填充,因此可以简化流程为“位宽调整→乘加计算→位宽调整”.

二级流水单元定义为整个神经网络,内部可以包含所有的一级流水单元以及底层单元位宽调整、数据流调整,以数据流缓冲区进行串联. 位宽调整单元用于调节数据流位宽大小以方便层间对接.

3.4 评估模型

评估模型是对每个单元的时延、计算资源、存储资源进行评估. 评估对象是计算链中主要单元,自底向上包括 $SDWC$、SWU、MVU、$conv3\times3$、$conv1\times1$、$pool$、net 等. 评估模型是从时延、计算资源、存储资源三个方面对加速器进行量化分析. 根据模型可以确定加速器的设计空间. 针对设计空间进行最优化搜索,以达成平衡资源和时延效率之间关系的目标.

3.4.1 粗粒度评估与评估指标

从 Ultra 加速器应用方案提出至最终形成 FPGA 应用系统将经过图 11 所示的三个步骤:加速器评估优化、高层次综合优化、硬件集成综合优化. 其中,加速器评估优化为本课题研究的主要内容,后两个环节应用的是 FPGA 厂商的开发工具,以实现对系统的部署.

上述三个环节都会对应用系统提供不同层面的优化:加速器评估优化从算法角度确定数据流组织

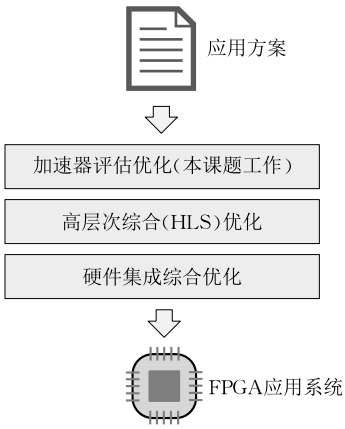

图 11　从应用方案到 FPGA 应用系统步骤

方式、确定计算并行程度；高层次综合优化对评估优化的结果从指令角度去除冗余指令，优化指令调度；硬件集成综合从硬件分配角度安排硬件资源的复用、完成硬件布线. 本课题中的工作属于加速器应用系统优化的前期阶段，重点在于将应用算法的并行方案进行映射，评估是为了确保该优化方案可以在 FPGA 上以最优的效果实现.

本课题对加速器粗粒度评估的工作主要从运行时延、计算资源、存储资源三个方面展开. 运行时延是主要关注的指标，期望时延尽可能的低；计算资源和存储资源对应的是应用平台的限制，主要关注评估结果不应超出平台所提供的硬件资源限制.

运行时延 $Latency_x$ 表示在 FPGA 板卡上运行计算单元 x 所用的时间，见式(1)，$Cycles_x$ 代表各个单元的运行周期数，f 代表 FPGA 的时钟主频，$\frac{1}{f}$ 即表示单个周期在 FPGA 上运行所需要的时间.

$$Latency_x = Cycles_x \times \frac{1}{f} \tag{1}$$

关于存储资源，这里只考虑 BRAM 的使用情况，由 FF、LUT 提供的存储由于量级较小而忽略不计. BRAM 的存储结构因 FPGA 厂商的不同而不同. 因此设置函数 $B()$ 用来评估硬件存储设备与所需空间的映射关系. 这里借鉴 Shen 等人[17] 对 Virtex-7 FPGA BRAM-18Kb 进行建模的分析方法来设计. 其中 B(数据位宽，空间数量)记作 $B(width, depth)$，以此表征其影响因素主要是存储数据的位宽和连续存储的空间数量.

经测算，在此类应用中 *MVU* 需要使用 *DSP* 实现，因而 *DSP* 资源容易成为这类应用的短板，因此在建模中只考虑 *DSP* 资源的使用，其余资源相对充足. 而 *DSP* 的工作方式通常因 FPGA 厂商的不同而不同，可容纳的计算位宽不同. 使用粗略估计的方式，设置函数 $D()$ 用于描述对应位宽乘法与 *DSP* 使用情况的映射关系. 此外 *DSP* 一般在并行运算中使用，所以与并行度有密切关系见式(2).

$$DSP = D(width_{elem}, PE \times SIMD) \tag{2}$$

3.4.2　底层单元评估

流水数据转换单元(Stream Data Width Converter，SDWC)用来调整数据流的位宽，以适应不同的计算要求. *SDWC* 的运行时延评估如式(3)、(4)，其中 $elem_{Input}$ 表示待转换数据的个数. 根据设计，运行时延与输出位宽和待转换的数据个数密切相关.

$$elem_{Input} = \frac{batch \times IW \times IH \times IC \times width_{elem}}{width_{IN}} \tag{3}$$

$$Cycles_{SDWC} = \begin{cases} elem_{Input}, & width_{IN} > width_{OUT} \\ elem_{Input} \times \dfrac{width_{IN}}{width_{OUT}}, & width_{IN} \le width_{OUT} \end{cases} \tag{4}$$

SWU 用来重排列数据，降低数据依赖性，匹配计算单元. *SWU* 根据功能可以划分为两个阶段进行运行时延评估，分别是加载数据至缓存区和从缓存区抓取数据重排后放置到输出数据流通道. 阶段一加载数据，应当将所有数据仅放入缓存区一次. 阶段二重排数据，应当与滑窗的尺寸 K 以及滑窗的移动次数相关. 缓存区设置大小为 $K \times IW \times IC \times width_{elem}$ bits，每次加载一行到缓存区，所用周期数为 IW；当缓存区满时进行重排数据，根据卷积算法和缓存区大小，可以重排数据 $\left\lfloor \frac{IW-K}{S}+1 \right\rfloor$ 次，每次重排涉及 K^2 个数据，因此所用周期数为 $\left\lfloor \frac{IW-K}{S}+1 \right\rfloor \times K^2$；之后加载一行新数据替换缓存区中的旧数据，两个阶段交替进行见图 12.

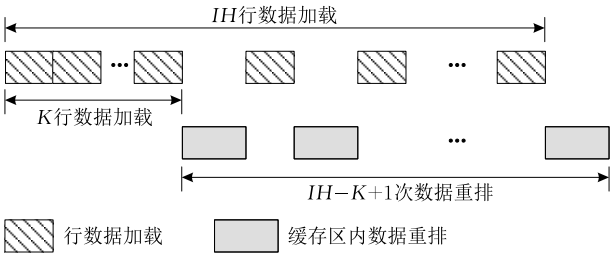

图 12　$S=1$ 情况下 *SWU* 运行示意图

根据卷积算法分析，一个 $IW \times IH \times IC$ 特征图需要进行 $batch \times IH$ 次行数据加载和 $batch \times$

$\left\lfloor\frac{IH-K}{S}+1\right\rfloor$次数据重排，由此可得到式(5)～(7)，其中$Cycles_{load}$表示加载环节需要的操作周期数，$Cycles_{order}$表示重排环节需要的操作周期数.

$$Cycles_{load}=batch\times IH\times IW \tag{5}$$

$$Cycles_{order}=batch\times\left\lfloor\frac{IH-K}{S}+1\right\rfloor\times\left\lfloor\frac{IW-K}{S}+1\right\rfloor\times K^2 \tag{6}$$

$$Cycles_{SWU}=Cycles_{load}+Cycles_{order} \tag{7}$$

对于片上空间的使用情况，SWU需要缓存K行数据，这里的数据对应的是一通道的数据，所以位宽为$IC\times width_{elem}$，见式(8).

$$Memory_{SWU\_buffer}=B(IC\times width_{elem},K\times(IW)) \tag{8}$$

标准卷积MVU是核心计算单元. 由于是并行计算单元，所以应当关注并行度. MVU主要完成的是卷积运行单元的乘加运算，卷积计算串行需要周期数$batch\times IC\times K^2\times OC\times(OW\times OH)$，经过$MVU$模块并行处理，有$SIMD$、$PE$两个维度的并行加速. MVU计算是对卷积计算的等价变形计算(见算法1). 在算法1中，由于之前SWU单元对数据顺序的转换，解除了数据依赖，MVU可以将原卷积计算卷积和K的两层循环与输入通道的循环进行合并，设置展开通道数为$SIMD$，之后将输出通道的循环设置展开通道数PE，进行并行乘加计算. MVU中涉及到的存储资源包括累加缓存数组acc和临时缓存量$store$，权值数据片上缓存，即放置在一个连续的空间内，见式(9)～(11).

$$Memory_{MVU\_store}=B\left(SIMD\times width_{elem},K^2\times\frac{IC}{SIMD}\right) \tag{9}$$

$$Memory_{MVU\_acc}=B(32,PE) \tag{10}$$

$$Memory_{WEIGHT\_onchip}=B\left(width,\frac{K^2\times IC\times OC\times width_{elem}}{width}\right) \tag{11}$$

由此可以得到评估式(12)，注意这里忽略了累加运行，只考虑了乘法运算. 上文提到MVU有三种模式. 其中向量模式下PE应当对应1.

$$Cycles_{MVU}=batch\times\frac{IC\times K^2}{SIMD}\times\frac{OC}{PE}\times(OW\times OH) \tag{12}$$

算法1. MVU计算.

输入：$Stream\langle ap\_int\langle width_{elem}\times SIMD\rangle\rangle$ IN

$ap\_int\langle width_{weight}\times SIMD\rangle WEIGHT$

$[PE]\left[\frac{IC\times K\times K}{SIMD}\times\frac{OC}{PE}\right]$

输出：$Stream\langle ap\_int\langle 32\times PE\rangle\rangle$ OUT

```
//IN 待处理数据流，WEIGHT 权值数组，
//OUT 完成池化输出数据流
1.  ap_int<width_elem × SIMD> temp←0;
2.  ap_int<32>  acc[PE];//累加缓存数组
3.  FOR b=0→batch DO{
4.    FOR ow=0→OW DO {
5.      FOR oh=0→OH DO {
6.        FOR i=0→IC×K² DO {
7.          temp←IN; //读取数据
8.          acc←0; //初始化 acc
9.          设置并行循环展开 SIMD 个通道;
10.         FOR o=0→OC DO {
11.           设置并行循环展开 PE 个通道;
12.           利用 WEIGHT 和 temp 完成乘加;
13.           利用 acc 累加结果;}
14.       OUT←acc;//输出结果}}}}
```

$Pool_cal$是池化的核心计算单元. 除了特征数据本身不涉及其他数据依赖问题，计算时只考虑通道IC维度的并行度$unroll_factor$即可. $Pool_cal$计算是对池化计算的并行优化(详见算法2). $Pool_call$可以将原池化计算输入通道的循环设置展开通道数为$unroll_factor$，进行并行比较运算. 由此可以推到出时延公式(13).

$$Cycles_{pool\_cal}=batch\times OW\times OH\times K^2\times\frac{IC}{unroll\_factor},\quad unroll\_factor\leq IC \tag{13}$$

算法2. Pool_cal 池化计算.

输入：$Stream\langle ap\_int\langle width_{elem}\times IC\rangle\rangle$ IN

输出：$Stream\langle ap\_int\langle width_{elem}\times IC\rangle\rangle$ OUT

```
//IN 待处理数据流，OUT 完成池化输出数据流
1.  ap_int<width_elem × IC>  result←0;
    //缓存结果变量
2.  FOR b=0→batch DO {
3.    FOR iw=0→OW DO {
4.      FOR ih=0→OH DO {
5.        FOR k=0→K² DO {
6.          ap_int<width_elem × IC>  temp←IN;
            //读取数据
7.          FOR ic=0→IC DO {
8.            设置并行循环展开 unroll_factor 个通道;
9.            利用 result 和 temp 完成比较运算;}
10.       OUT←result;//输出结果
11.       result←0;//清空缓存变量}}}}
```

3.4.3 一级和二级流水单元评估

根据上一章节中的解释，一级流水和二级流水分别是由多个底层单元拼接而成，底层单元可以满足“流水结构”. 在数据满载且不考虑时间间隙的情

况下，可视为所有单元均可以同时执行，因此流水单元的时延周期就是所有底层单元中最长时延周期数. 根据上章中对一级、二级流水具体单元的定义，详细评估公式(14)～(18).

$$Cycles_{conv3\times3}=\max(Cycles_{SWU},Cycles_{SDWC},Cycles_{MVU}) \tag{14}$$

$$Cycles_{conv1\times1}=\max(Cycles_{SDWC},Cycles_{MVU}) \tag{15}$$

$$Cycles_{FC}=\max(Cycles_{SDWC},Cycles_{MVU}) \tag{16}$$

$$Cycles_{pool}=\max(Cycles_{SWU},Cycles_{pool\_cal}) \tag{17}$$

$$Cycles_{net}=\max(Cycles_{layer1},Cycles_{layer2},\cdots,Cycles_{layern}),\ layer\in\{conv3\times3,conv1\times1,pool\} \tag{18}$$

观察评估模型可以发现以下几点：

(1) 并行参数选择加大时，对应的计算、存储的资源需要分配更多.

(2) 在流水结构下，时延周期由峰值的单元决定.

(3) 底层单元中 *MVU* 易成为时延周期最大的单元，并且有并行参数可以通过分配计算、存储资源来调节. *SWU* 因为通常与 *MVU* 连用，在存储资源方面会相应受到影响.

(4) 底层单元位宽调整并非自主可控，因此当位宽调整成为整个系统的瓶颈时，可以说明加速器已调整至最优.

(5) 一级流水单元中卷积由于算法特性计算量较大，因此通常是高时延周期数单元. 一方面可以通过调整对应底层单元的并行参数，另一方面还可以在设计应用神经网络时进行软硬协同的定制化设计.

针对评估模型构建设计空间进行调优加速器十分必要，加速器的并行参数选择应该尽量使各个单元的时延周期数保持一致.

3.4.4　优化模型

上一节确定了 Ultra 加速器的设计空间，本节针对设计空间将问题转化为最优化解决方案搜索问题. 搜索的目标是找出一套参数配置使得在所有资源都允许的情况下达到运行时延最小化. 优化模型包括四部分，搜索空间、约束条件、目标函数、搜索算法. 根据评估模型，可以分析得出一些结论，便于定义最优化问题的 4 个要素：

(1) *MVU* 中 *PE* 和 *SIMD* 是主要消耗计算资源的因素.

(2) 每个模块的计算位宽和模型参数是存储资源的主要影响因素.

(3) 每个流水单元的时延由耗时最长的子单元决定.

由此得出优化模型如下：每层的参数 *PE* 和 *SIMD*，其中 $m|n$ 表示 m 被 n 整除. 搜索算法采用随机搜索算法.

$$\text{搜索空间：}\begin{cases}0<PE<IC\\0<SIMD<K\times K\times IC\end{cases}$$

$$\text{约束条件：}\begin{cases}IC|PE\\(K\times K\times IC)|SIMD\\\sum M_{xi}<M\\\sum C_{xi}<C\end{cases}$$

目标函数：$\min(\max(L_{xi}))$

3.5　实现方案

这一节具体陈述从定制化神经网络算法到评估调优最终生成加速器的过程. 在软件设计部分，首先在 GPU 上对 CNN 进行低比特量化预训练；在硬件设计部分，主要基于 HLS 工具实现了 Utra 方案. 其中，硬件的综合和布线是通过 Vivado 工具完成的. 本文方案主要讨论推理时延和冗余计算率，而不是 CNN 的准确性，这是因为 UltraAcc 设计不影响 CNN 的精度. 在软件设计中，量化算法才是影响 CNN 精度的关键，本文采用本课题组提出的 LSFQ[32] 方法，该方法以 CNN 精度要求和底层硬件资源限制来确定相应的量化方案.

3.5.1　Ultranet 加速方案

针对DAC-SDC这一赛题，本文设计了匹配 Ultra-96v2 板卡性能的 Ultranet 神经网络进行目标检测任务. 输入特征图尺寸分别为 320×160. 经过简单估算，Ultranet 较轻量，完全可以适应张量计算模式. Ultranet 的调优后超参数配置如表 4，拟定运行环境每周期 8 ns.

表 4　Ultranet-Ultra 加速器方案

网络层	尺寸 (*IN*,*OUT*,*K*)	参数量/kb	计算量 MAC	*PE*	*SIMD*	预估时延/μs
Conv0	3,16,3	0.22	0.022	16	3	2.77
Pool0						0.39
Conv1	16,32,3	2.30	0.059	8	16	2.77
Pool1						0.10
Conv2	32,64,3	9.21	0.059	8	16	2.77
Pool2						0.02
Conv3	64,64,3	18.43	0.029	4	16	2.77
Pool3						0.01
Conv4	64,64,3	18.43	0.007	2	8	2.77
Conv5	64,64,3	18.43	0.007	2	8	2.77
Conv6	64,64,3	18.43	0.007	2	8	2.77
Conv7	64,64,3	18.43	0.007	2	8	2.77
Conv8	64,36,1	2.30	0.001	2	8	0.17
Total		106.18	0.198	—	—	2.77

分析调优模型给出的方案，找到瓶颈在 Conv0 中的 *SDWC*. 可以看到当前配置方案下，加速器的运行时延最大的是第一个层卷积中的 *SDWC* 单元. *SDWC* 运行时延完全是读取特征图数据. 根据公式，*SDWC* 要处理数据在该加速器中的上限即为 2.77 ms. 当前加速器下不能进一步优化. 可说明 Ultranet 已经最大程度优化.

3.5.2 VGG16Beta 加速方案

Ultranet 设计较为极致，全部权值可以放置在 Ultra96v2 板卡上运行. 此外本文选用 224×224 样本尺寸的 ImageNet VGG16 网络，其体积计算量较大，需要采用片外模式.

为了适应板卡计算，本文将 VGG16 稍作改动，改动后的网络模型称为 VGG16Beta. 将原浮点型网络量化到 4 bit. 最后一层卷积进行通道修改. VGG16Beta 以及其调优方案如表 5. 拟定运行环境每周期 8 ns.

表 5 VGG16Beta-Ultra 加速器方案

网络层	尺寸 (*IN*,*OUT*,*K*)	参数量 kb	计算量 MAC	*PE*	*SIMD*	预估时延/μs
Conv0_0	3,64,3	0.86	0.17	1	9	77.07
Conv0_1	64,64,3	18.43	3.70	1	64	231.21
Pool0						0.51
Conv1_0	64,128,3	36.86	1.85	1	32	231.21
Conv1_1	128,128,3	73.73	3.70	1	64	231.21
Pool1						0.13
Conv2_0	128,256,3	147.46	1.85	1	32	231.21
Conv2_1	256,256,3	294.92	3.70	1	64	231.21
Conv2_2	256,256,3	294.92	3.70	1	64	231.21
Pool2						0.03
Conv3_0	256,256,3	294.91	0.92	1	16	231.21
Conv3_1	256,256,3	294.91	0.92	1	16	231.21
Conv3_2	256,256,3	294.91	0.92	1	16	231.21
Pool3						0.01
Conv3_0	256,256,3	294.91	0.23	1	4	231.21
Conv3_1	256,256,3	294.91	0.23	1	4	231.21
Conv3_2	256,256,3	294.91	0.23	1	4	231.21
Pool4						0.002
FC	12544,1000	12544	0.012	ARM 平台上执行		
Total		15180.62	22.13	—	—	231.21

VGG16Beta 层数较多，片上的缓存空间全部用来流水中间结果存储. 采用片外存储模式存储权值数据. 瓶颈出现在 Conv0_1 内，*SIMD* 不能设置得更大，会导致片内空间不足，不能形成整个计算链流水结构. 调优模型给出的是可行的较优方案.

3.5.3 LeNet 加速方案

本文选用 MNIST-LeNet 类型网络进行加速器应用部署. 经过评估，MNIST-LeNet 计算量较小，可以适应张量计算模式. MNIST-LeNet-4bit 调优后的超参数配置如表 6 所示，拟定运行环境每周期 6 ns. MNIST-LeNet 数据集输入图像尺寸为 32×32 的黑白图像，卷积不做填充处理 padding.

表 6 MNIST-LeNet-4bit-Ultra 加速器配置

网络层	尺寸 (*IN*,*OUT*,*K*)	参数量/kb	计算量 MAC	*PE*	*SIMD*	预估时延/μs
Conv0	1,3,5	0.038	76800	2	2	154.2
Pool0						9.4
Conv1	3,3,2	0.45	921600	2	2	154.2
Pool1						7.2
FC1	300,48	7.2	14400	1	1	86.7
FC2	48,10	0.24	480	1	1	2.9
Total		7.928	1013280	—	—	154.2

分析调优模型给出的方案，找到瓶颈在 Conv0 中的 *SDWC*. 可以看到当前配置方案下，加速器的运行时延最大的是第一个层卷积中的 *SDWC* 单元. *SDWC* 运行时延完全是读取特征图数据. 根据公式，*SDWC* 要处理数据在该加速器中的上限即为 154.2 μs. 当前加速器下不能进一步优化. 可说明 MNIST-LeNet 已经最大程度优化.

4 实 验

本节将从多个角度对 Ultra 加速器进行测试. 通过对比当前先进的加速器工作，以及在 DAC-SDC 中进行了实战测试，来评价 Ultra 加速器的性能优越性以及实际应用的价值. 此外对评估模型进行了单独测试，评价该解决方案在实际应用中的有效性.

4.1 实验设置

(1) 软件与硬件环境. 本文实现 Ultra 加速器是在 Xilinx Vivado+HLS2018.3 上. 目标板卡是 Ultra96v2. Ultra96v2 包括 XCZU3EG 型号 FPGA 芯片，Cortex-A53 ARM 板卡. FPGA 芯片包含 216BRAMs、70560LUTs、141120FFs 和 360 个 DSPs.

(2) Ultra 加速器配置. 采用上节中两个实际问题的解决方案. 主干网络部署在 FPGA 端，其余应用计算(如目标检测 YOLO 部分，图像分类 softmax 部分)放置在 ARM 端完成. 在 Ultra96v2 的 ARM 上构建了基于 Xilinx PYNQ 框架的测试程序. 该程序将完成以下功能：将待测试数据加载到 DDR3 内存中；调用 FPGA 的 PYNQ 接口将 Ultra 加速器的硬件物理地址映射到内存中；给出信号，激活 FPGA

上 Ultra 加速器，同时启动定时器；在 Ultra 加速器的硬件执行结束时停止计时. 其中记录的执行时间就是 Ultra 加速器的运行时延.

(3) Ultra 加速器以及评估模型评测方法. 本文将从三个角度对加速器性能进行评测，加速器运行时延、加速器功耗、加速器资源利用情况. 使用评估调优模型给出了三个实际应用中的解决方案，对比实际运行情况，分析评估模型的应用效果. 具体实验方法如下：

(1) 将 Ultra 加速器与其他加速器进行对比.

(2) 本工作在参与了 DAC-SDC'20 之后加入了调优模型进行优化，给出了 Ultra 加速器优化后的成绩与 DAC-SDC 往年成绩的对比.

(3) 对比评估模型给出的预期结果和实际 Ultra 加速器运行结果.

4.2　与其他加速器性能比较

表 7 是 Ultra 加速器对比之前其他工作性能对比情况. Ultra 加速器在 Ultranet 情况下平均吞吐量达到了 126.72 GOPs. 对比 CECA 实验室的工作[8]提升了 2.06 倍. 由于不同的加速器研究工作使用了不同的 FPGA 平台，计算资源情况也有所不同. 为了提供一个公平的比较，表中讨论了性能密度[8]. 从表中可以看出 Ultra 加速器的性能密度要优于其他工作，本文的工作最高，达到了 1.44×10^{-2}，是 CECA 实验室工作的 17.73 倍. 本文对比了单个 DSP 的吞吐量，Ultra 加速器可以达到 0.35，高于其他公开这一指标的工作. 除了适应性最佳的 Ultranet，本文还测试了 Ultra 加速器在 VGG16Beta 上，也可以达到 9.29×10^{-3}. 说明 Ultra 加速器具备一定的兼容能力.

表 7　Ultra 加速器与以往工作加速器性能对比

	计算位宽	主频/MHz	FPGA 型号	FPGA 资源	神经网络模型	计算量	吞吐量	GOPs/SLICE	GOPs/DSP
[13]	32float	100	Virtex7 VX485T	75900 SLICE 2800 DSP	Alexnet	1.33GLOP	61.62 GOPs	8.12×10^{-4}	—
[28]	权值 11bit 激活 9bit	215	Zynq ZU3EG	8800 SLICE 360 DSP	SkyNet	0.46GMAC	23.15 GOPs	2.63×10^{-3}	—
[38]	权值 16bit 激活 16bit	125	Zynq 7Z020	13300 SLICE 220 DSP	VGG16	30.72GLOPs	38.3 GOPs	2.88×10^{-3}	0.17
[26]	权值 4bit 激活 4bit	250	Zynq ZU3EG	8800 SLICE 360 DSP	DiracDeltanet	0.24GMAC	47.09 GOPs	5.35×10^{-3}	—
[27]	权值 8bit 激活 8bit	200	Kintex7 XC7K325T	50950 SLICE 840 DSP	MobileNetV1	—	147.84 GOPs	2.9×10^{-3}	0.21
Ultra 加速器	权值 4bit 激活 4bit	166 125	Zynq ZU3EG	8800 SLICE 360 DSP	Ultranet VGG16Beta	0.20GMAC 11.1GMAC	126.72 GOPs 81.8 GOPs	1.44×10^{-2} 9.29×10^{-3}	0.35 0.23

表 8 是 Ultranet 的 Ultra 加速器与 SkyNet 加速器部署时资源消耗的对比情况. 因为流水设计中的基础计算单元更细致，所以 DSP 资源更容易被全部利用起来，并且所有 DSP 计算资源在运行中没有闲置状态. 相较于 SkyNet DSP 资源分给了 $conv3\times3$ 和 $conv1\times1$，由于 SkyNet 是循环平铺加速器逐层运算，$conv3\times3$ 和 $conv1\times1$ 不可能同时进行计算. 即便 SkyNet 使用了 92% 的 DSP 资源，但是实际应用计算时，DSP 资源一直是有闲置的. 因此相比之下，Ultranet DSP 资源浪费要少很多，能量消耗更少. 此外由于 Ultranet 网络设计得较小，所以片上的存储资源并没有完全用完.

表 8　FPGA 部署资源利用情况对比

资源利用率	DSP/%	BRAM/%	LUT/%	FF/%
Ultranet	100	68	59	31
SkyNet	92	96	83	38

LeNet由Lecun于1998年首次提出，用于 MNIST 数据集手写体识别应用[33]. LeNet 由两层卷积、两层池化、三层全连接构成. 这种 2 层卷积、2 层池化和全连接层的结构被称作 LeNet 型神经网络，由于其计算较为轻量级，常用于低资源 FPGA 加速的研究. 表 9 展示了 Ultra 加速器与之前 LeNet 类型的其他工作性能对比的情况. 其中，量化训练采用 LSFQ[32]方法，该方案在 4bit 的量化情况下运行时延达到 184 μs，相比 Youssef[34] 提速了 31.9%；在 8bit 量化下 top1 精度达到 98.74%，运行时延达到 192 μs.

表 9　LeNet CNN 加速器的性能对比

加速器	设备	Bit	Top-1	运行时延/μs
[34]	Zynq 7000	W16a16	98.12	270
[35]	Virtex7 485t	W8A16	98.16	490
本文工作	Ultra96v2	W4a4	98.72	184
	Ultra96v2	W8a8	98.74	192

4.3 DAC-SDC 往年成绩比较

表 10 中列出了 2019、2020 两年的参赛队伍前三名的成绩，包括精度 IoU(ground truth 与结果框的占比)，速度 FPS(1 秒内完成目标检测的帧数)以及能量消耗(单位：kJ)．可以看出本文参赛队伍时延是其余队伍的 4 倍，达到 220.76 fps．消耗能量方面只使用了 1.64 kJ，相当于其余队伍的 1/5．往年比赛队伍中只有 SystemsETHZ 使用的是数据流水加速器，其余队伍均采用的是循环平铺加速器．

表 10 DAC-SDC'19 和 DAC-SDC'20 FPGA 组成绩对比

年份	参赛队伍	神经网络	IoU	FPS	Energy/kJ
2019	SystemsETHZ	SqueezeNet	0.553	55.13	6.30
	XJTU_Tripler	ShuffleNetV2	0.615	50.91	9.45
	iSmart3	SkyNet	0.716	25.05	15.10
2020	iSmart	SkyNet	0.724	50.68	7.62
	ShanghaiTech_SkSkr	SkyNet	**0.731**	52.43	6.76
	BJUT_Runner(ours)	Ultranet	0.656	212.73	**1.64**
—	Ours	Ultranet	0.667	**220.76**	1.82

经过本文的改进，本文使用评估优化模型进行解决方案优化．最后一行成绩来自 DAC-SDC'21 在 4 月公开提交的成绩，可以看到本文将 FPS 成绩再次提高．本文认为该成绩已经达到 Ultranet 在 Ultra 加速器上运行的极限速度．(这里，赛方提供的速度成绩包含了测试数据读入时间，Ultranet 独立运行测试速度约为 320 FPS)．

从表 10 的 IoU 精度结果可以看出，SkyNet 的精度成绩更为优秀．对比 SkyNet 和 Ultranet 网络结构，SkyNet 使用了可分离卷积作为基础算法构建网络，可分离卷积的优势在于计算量和参数量远低于普通卷积，并且精度可以满足 DAC 赛题应用水平．在之后 2021 年、2022 年的冠军团队均采用了可分离卷积作为基础网络．精度主要．2020 年参赛的 Ultranet 经本文调优后该方案已充分利用了 Ultra96v2 平台资源，但是受限于普通卷积算法本身，模型改动空间较小，因此精度提升较小，可再提升的可能性比较低．

2021 年 DAC-SDC 冠军团队沿用流水结构加速器，在 SkyNet 基础上将单加速器串行计算架构替换成多加速器并行数据流水架构[36]．2022 年 7 月 DAC-SDC 冠军团队再报佳绩，通过 S2C、SharePE 提升了计算资源和带宽的利用率[37]，虽然具体细节还未披露，但在该团队前期发表的工作[38]来看，其针对可分离卷积提出了 P2D&D2P 等价变换计算流程以减少外部内存访问，进而降低了数据依赖性．

除了比赛队伍之间比较，本文横向对比了 Ultranet 网络在 CPU(Intel Xeon E5-2680 v4 2.40 GHz)、GPU(GeForce RTX 3090)、FPGA(Zynq ZU3EG)平台的运行效果，如图 13．采用每次输入一张 640×360 的 RGB 图像，共输入 1000 张图像，测试了运行效率 FPS 指标．可以看出 Ultra 加速器在 FPGA 运行速度是 CPU 的 6.45 倍，是 GPU 的 1.39 倍．可以说明基于 FPGA 的 Ultra 加速器在主流平台内是先进的．

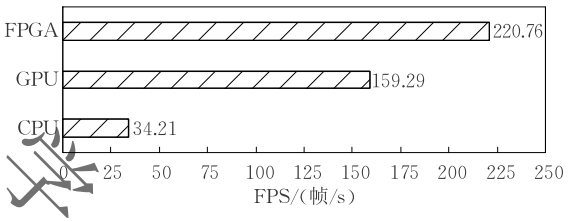

图 13 Ultranet 在 CPU、GPU、FPGA 平台运行对比

4.4 评估模型与实际情况比较

根据模型分析，影响卷积单元计算时延的参数共有 5 个，包括输入通道数 IN、输出通道数 OUT、内核尺寸 K、并行参数 PE、位宽并行参数 $SIMD$．对这 5 个参数进行控制变量，设置了 10 组测试样例见表 11．在 100 MHz 的情况下进行卷积单元测试，从输入特征图维度 W、H、IN，卷积参数 IN、OUT、

表 11 $K\times K$ 卷积测试样例

编号	特征宽度 W	特征高度 H	卷积核 K	输入通道数 IN	输出通道数 OUT	并行参数 PE	位宽并行参数 $SIMD$	预估值/ms	实际值/ms	误差率/%
1	320	160	3	32	64	4	16	147.46	99.13	48.75
2	320	160	3	32	64	2	8	589.24	393.50	49.74
3	320	160	3	64	32	4	8	294.91	198.04	48.92
4	320	160	3	16	96	4	8	221.18	147.90	49.55
5	320	160	3	16	96	8	8	110.59	74.30	48.85
6	320	160	1	64	64	2	8	131.07	87.71	49.45
7	320	160	1	64	64	4	16	32.77	22.14	48.03
8	320	160	1	64	128	4	16	65.54	44.25	48.11
9	320	160	1	32	128	4	8	32.77	22.41	46.25
10	320	160	1	32	128	8	16	16.38	11.45	43.07

K，并行参数 PE、$SIMD$ 中可以确认一个测试样例. 根据评估模型可以预估出运行时延，部署到 FPGA 上实测运行时间进行对照分析，计算出误差值和误差率$\left(\frac{\text{预估值}-\text{实际值}}{\text{实际值}}\right)$. 可以看出，评估模型给出的结果与实际结果误差值较大，平均误差值 54 ms，但是误差率基本保持一致，为 48.07%. 这样的误差是由于评估是粗粒度的，从设计上忽略了运算时硬件设备的时间间隙，另一方面是本文的优化位于高层，部署到板卡运行之前还经过了 HLS、Vivado 两轮优化. 误差率基本保持一致，可以说明评估与实际结果是成正相关关系的，从实际应用来说可以做到指导加速器方案设计，达到了粗粒度评估的预期. 由此可以初步得出结论，评估模型与实际结果是成正相关关系，具有指导方案设计的应用价值.

5　结束语

本文针对资源有限的 FPGA 高效部署卷积神经网络问题进行了研究. 本文提出了 Ultra 加速器从算法到硬件部署优化的定制化解决方案. Ultra 加速器采用流水架构，其中主要包括针对卷积计算的数据流重排模块以及卷积神经网络所需的基本计算单元，自底向上进行设计. 搭建评估优化模型进行加速器超参数的优化. 最终实验表明，Ultra 加速器在 Ultra96 板卡上平均吞吐量可以达到 126.72 GOPs，是 IEEE/ACM DAC-SDC'19 冠军方法的 5.47 倍. Ultra 加速器在资源较为有限的 FPGA 板卡上可以实现计算量较大的 VGG16Beta 网络，并可以达到 81.9 GOPs 的吞吐量. 用 Ultra 加速器参与了 DAC-SDC'20 低功耗目标检测的比赛，最终以精度 IoU 0.65、速度 FPS 212.73，能量消耗 1.64 kJ 夺得冠军.

参　考　文　献

[1] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521: 436-444. https://doi.org/10.1038/nature14539

[2] Wang Chao, Wang Teng, Ma Xiang, Zhou Xue-Hai. Research progress on FPGA-based machine learning hardware acceleration. Chinese Journal of Computers, 2020, 43(6): 1161-1182(in Chinese)
(王超，王腾，马翔，周学海. 基于 FPGA 的机器学习硬件加速研究进展. 计算机学报，2020，43(6)：1161-1182)

[3] Wu Yan-Xia, Liang Kai, Liu Ying, Cui Hui-Min. The progress and trends of FPGA-based accelerators in deep learning. Chinese Journal of Computers, 2019, 42(11): 2461-2480(in Chinese)
(吴艳霞，梁楷，刘颖，崔慧敏. 深度学习 FPGA 加速器的进展与趋势. 计算机学报，2019，42(11)：2461-2480)

[4] Zhang Z, Kouzani A Z. Resource-constrained FPGA/DNN co-design. Neural Computing and Applications, 2021, 33: 14741-14751

[5] Chen Yu-Ting, Cong Jing-Sheng, Gill M, et al. Customizable Computing. Beijing: China Machine Press, 2018(in Chinese)
(陈昱廷，丛京生，迈克尔·吉尔等. 可定制计算. 北京：机械工业出版社，2018)

[6] Guo K, Zeng S, Yu J, et al. A survey of FPGA-based neural network inference accelerators. ACM Transactions on Reconfigurable Technology and Systems, 2019, 12(1): 1-26

[7] Chen Y, Xie Y, Song L, et al. A survey of accelerator architectures for deep neural networks. Engineering, 2020, 6(3): 264-274

[8] Zhang C, Li P, Sun G, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks// Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. New York, USA, 2015: 161-170

[9] Yaman U, Nicholas J, Giulio G, et al. FINN: A framework for fast, scalable binarized neural network inference// Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. New York, USA, 2017: 65-74

[10] Nuzman D, Zaks A. Outer-loop vectorization: Revisited for short SIMD architectures//Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. Toronto, Canada, 2008: 2-11

[11] Chen Y, Krishna T, Emer J, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 2017, 52(1): 127-138

[12] Chen Y, Emer J, Sze V. Eyeriss: A spatial architecture for energy efficient dataflow for convolutional neural networks. ACM SIGARCH Computer Architect News, 2016, 44(3): 367-379

[13] Sharma H, Park J, Mahajan D, et al. From high-level deep neural models to FPGAs//Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Taipei, China, 2016: 1-12

[14] Gong L, Wang C, Li X, et al. MALOC: A fully pipelined FPGA accelerator for convolutional neural networks with all layers mapped on chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 37(11): 2601-2612

[15] Zhang X, Wang J, Zhu C, et al. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs//Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). San Diego, USA, 2018, 56:1-8

[16] Venieris S, Bouganis C. fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs. IEEE Transactions on Neural Networks and Learning Systems, 2019, 30(2): 326-342

[17] Shen Y, Michael F, Peter M. Maximizing CNN accelerator efficiency through resource partitioning. ACM SIGARCH Computer Architect News, 2017, 45(2): 535-547

[18] Lu Ye, Chen Yao, Li Tao, et al. Convolutional neural network construction method for embedded FPGAs oriented edge computing. Journal of Computer Research and Development, 2018, 55(3): 551-562(in Chinese)
(卢冶, 陈瑶, 李涛等. 面向边缘计算的嵌入式 FPGA 卷积神经网络构建方法. 计算机研究与发展, 2018, 55(3): 551-562)

[19] Hiroki N, Haruyoshi Y, Tomoya F, Shimpei S. A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA//Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey, USA, 2018: 31-40

[20] Joseph R, Ali F. YOLO9000: Better, faster, stronger//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, USA, 2017: 6517-6525

[21] Michaela B, Thomas B, Nicholas J, et al. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2018, 11(3): 1-23

[22] Yang Y, Huang Q, Wu B, et al. Synetgy: Algorithm-hardware co-design for ConvNet accelerators on embedded FPGAs//Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. New York, USA, 2019: 23-32

[23] Yu Y, Zhao T, Wang K, He L. Light-OPU: An FPGA-based overlay processor for lightweight convolutional neural networks//Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Seaside, USA, 2020: 122-132

[24] Hao C, Chen D. Deep neural network model and FPGA accelerator co-design: Opportunities and challenges//Proceedings of the 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT). Qingdao, China, 2018: 1-4

[25] Hao C, Zhang X, Li Y, et al. FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge//Proceedings of the 2019 56th ACM/IEEE Design Automation Conference (DAC). San Francisco, USA, 2019: 1-6

[26] Zhang X, Lu H, Hao C, et al. SkyNet: A hardware-efficient method for object detection and tracking on embedded systems//Proceedings of the Machine Learning and Systems (MLSys). Austin, USA, 2020: 1-14

[27] Xu X, Zhang X, Yu B, et al. DAC-SDC low power object detection challenge for UAV applications. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2019, 43(2): 392-403

[28] Chao H, Gu Y, Napolitano M. A survey of optical flow techniques for UAV navigation applications//Proceedings of the 2013 International Conference on Unmanned Aircraft Systems (ICUAS). Atlanta, USA, 2013: 710-716

[29] Torres-Sanchez J, Lopez-Granados F, Pena J. An automatic object-based method for optimal thresholding in UAV images: Application for vegetation detection in herbaceous crops. Computers and Electronics in Agriculture, 2015, 114(C): 43-52

[30] Russakovsky O, Deng J, Su H, et al. ImageNet large scale visual recognition challenge. International Journal of Computer Vision, 2015, 115(3): 211-252

[31] Everingham M, Van Gool L, Williams C, et al. The pascal visual object classes (VOC) challenge. International Journal of Computer Vision, 2010, 88(2): 303-338

[32] Bao Z, Fu G, Zhang W, et al. LSFQ: A low-bit full integer quantization for high-performance FPGA-based CNN acceleration. IEEE Micro, 2022, 42(2): 8-15

[33] Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86(11): 2278-2324

[34] Youssef E, Elsemary H, El-Moursy M, et al. Energy adaptive convolution neural network using dynamic partial reconfiguration//Proceedings of the 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS). Springfield, USA, 2020: 325-328

[35] Li Z, Wang L, Guo S, et al. Laius: An 8-bit fixed-point CNN hardware inference engine//Proceedings of the 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC). Guangzhou, China, 2017: 143-150

[36] Jiang W, Yu H, Liu X, et al. TAIT: One-shot full-integer lightweight DNN quantization via tunable activation imbalance transfer//Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC). 2021: 1027-1032

[37] https://byucci.github.io/dac_sdc_2022

[38] Li G, Zhang J, Zhang M, et al. Efficient depth-wise separable convolution accelerator for classification and UAV object detection. Neurocomputing, 2022, 490: 1-16

BAO Zhen-Shan, associate professor. His research interests include intelligent computing system and efficient deep learning.

GUO Jun-Nan, M. S. candidate. His research interests include FPGA architecture and neural network inference acceleration.

ZHANG Wen-Bo, associate professor. Her research interests include co-designing efficient algorithms and hardware systems for machine learning.

DANG Hong-Bo, M. S. candidate. His research interests include convolutional neural network and deep learning.

Background

Neural network accelerator has become a hot research topic in recent years because of its high performance and low power in practical applications. Many researchers believe that compared with GPU and CPU, FPGA can better meet the requirements of high precision, low power consumption and high time efficiency in practical neural network applications. IEEE/ACM Design Automation Conference (DAC) holds the System Design Contest (DAC-SDC). The task of DAC-SDC is to implement the object detection algorithm with deep neural networks on the FPGA. Researchers are encouraged to participate in this area of research. At present, neural network accelerators based on FPGA accelerate optimization from convolution, pooling and other algorithm layers. The accelerators adopt loop tiling, dataflow and pipeline to accelerate parallel, and then implement neural network inference by reuse. Such methods seldom consider the whole network and lack optimization between layers. This problem leads to the waste of latency and low utilization of computing resources. Aiming at this problem, this paper mainly studies the optimization method between layers of neural network. In this paper, the Ultra accelerator (UltraAcc) is proposed. We reorganize the dataflow structure to adapt the whole CNN parallel operation. An evaluation model is designed for hyper-parameter tuning. From the bottom unit to the computing layer unit and then to the whole computing chain, storage resources, computing resources and latency are evaluated. With the precision result of CNN training, the whole application system is balanced and optimized from both software and hardware. The UltraAcc can achieve an average throughput of 126.72 GOPs on the Ultra96v2, 5.47 times higher than the first place method in IEEE/ACM DAC-SDC'19 on the same platform. The UltraAcc was used to participate in the DAC-SDC'20. And UltraAcc won the first prize with accuracy of IoU 0.65, speed of FPS 212.73 and power consumption of 1.64 kJ.

This work is supported by the National Natural Science Foundation of China (No. 62072016).