计算机科学
COMPUTER SCIENCE

面向低资源芯片的高效自适应卷积神经网络加速器

庞明义, 魏祥麟, 张云祥, 王斌, 庄建军

引用本文

庞明义, 魏祥麟, 张云祥, 王斌, 庄建军. 面向低资源芯片的高效自适应卷积神经网络加速器[J]. 计算机科学, 2025, 52(4): 94-100.

PANG Mingyi, WEI Xianglin, ZHANG Yunxiang, WANG Bin, ZHUANG Jianjun. Efficient Adaptive CNN Accelerator for Resource-limited Chips [J]. Computer Science, 2025, 52(4): 94-100.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241000099

面向低资源芯片的高效自适应卷积神经网络加速器

庞明义[1] 魏祥麟[2] 张云祥[2] 王 斌[2] 庄建军[1]
1 南京信息工程大学电子与信息工程学院 南京 211800
2 国防科技大学第六十三研究所 南京 210007
(202312490285@nuist.edu.cn)

摘 要 文中提出了一种面向非GPU类低资源芯片的自适应卷积神经网络加速器(Adaptive Convolutional Neural Network Accelerator,ACNNA),其可根据硬件平台资源约束和卷积神经网络结构自适应生成对应的硬件加速器。通过可重构特性,ACNNA可有效加速包括卷积层、池化层、激活层和全连接层在内的各种网络层组合。首先,设计了一种资源折叠式多通道处理引擎(Processing Engine,PE)阵列,将理想化卷积结构进行折叠以节省资源,在输出通道上展开以支持并行计算。其次,采用多级存储与乒乓缓存机制对流水线进行优化,有效提升数据处理效率。然后,提出了一种多级存储下的资源复用策略,结合设计空间探索算法,针对网络参数调度硬件资源分配,使低资源芯片可部署层次更深且参数更多的网络模型。以LeNet5和VGG16网络模型为例,在Ultra96 V2开发板上对ACNNA进行了验证。结果显示,采用ACNNA部署的VGG16最低仅消耗了原网络4%的资源量。在100MHz主频下,LeNet5加速器在2.05W的功耗下计算速率达0.37 GFLOPS;VGG16加速器在2.13W的功耗下计算速率达1.55 GFLOPS。与现有工作相比,所提方法的FPS提升超过83%。

关键词:硬件加速;卷积神经网络;设计空间探索策略;现场可编程门阵列
中图分类号 TP391

Efficient Adaptive CNN Accelerator for Resource-limited Chips

PANG Mingyi[1],WEI Xianglin[2],ZHANG Yunxiang[2],WANG Bin[2] and ZHUANG Jianjun[1]
1 School of Electronics and Information Engineering,Nanjing University of Information Science & Technology,Nanjing 211800,China
2 63rd Research Institute,National University of Defense Technology,Nanjing 210007,China

Abstract This paper proposes an adaptive convolutional neural network accelerator(ACNNA) for non-GPU chips with limited resources,which can adaptively generate hardware accelerators based on resource constraints of hardware platform and convolutional neural network structures. Through its reconfigurable feature,ACNNA can effectively accelerate various layer combinations including convolutional layers,pooling layers,activation layers,and fully connected layers. Firstly,a resource folding multi-channel processing engine(PE) array is designed,which folds the idealized convolutional structure to save resources and unfolds on the output channel to support parallel computing. Secondly,multi-level storage and ping-pong caching mechanisms are adopted to optimize the pipeline,effectively improving data processing efficiency. Then,a resource reuse strategy under multi-level storage is proposed,which combined with the design space exploration algorithm can more reasonably schedule hardware resource allocation for network parameters,so that low resource chips can deploy deeper and more parameterized network models. Taking LeNet5 and VGG16 network models as examples,this paper validate ACNNA on the Ultra96 V2 development board. The results show that the ACNNA deployment of VGG16 consumes only 4% of resources of original network. At 100MHz main frequency,LeNet5 accelerator achieves a computing rate of 0.37 GFLOPS with a power consumption of 2.05W; VGG16 accelerator has a computing speed of 1.55 GFLOPS at a power consumption of 2.132W. Compared with existing work,ACNNA increases Frames Per Second(FPS) by over 83%.

Keywords Hardware acceleration,Convolutional neural network,Design space exploration strategy,Field programmable gate array

1 引言

卷积神经网络(Convolutional Neural Network,CNN)在图像分类、语音识别和自然语言处理[1-4]等多个领域表现出色。深层次的网络结构、多层次的特征提取能力以及大规模数据集的训练,使得CNN模型能够自动学习和提取数据中的关键特征,完成高效的模式识别和分类任务。为提升网络性能,CNN网络结构日益复杂[5],参数量快速增加,对硬件

到稿日期:2024-10-21 返修日期:2025-02-22
通信作者:魏祥麟(weixianglin@nudt.edu.cn)

部署平台的计算和存储资源提出了很高的要求。

CNN 部署主要分为云端部署和边缘端设备部署两种方式。云端部署通常采用高性能的图形处理单元(Graphics Processing Unit，GPU)进行模型推理，适合需要大量计算资源和高性能的应用场景。边缘端设备部署通常将神经网络模型部署在低资源芯片上。相比 GPU 平台，低资源芯片计算资源有限，难以支撑复杂的推理计算。推理硬件平台主要包括应用型专用集成电路(Application-Specific Integrated Circuit，ASIC)、中央处理器(Central Processing Unit，CPU)、GPU 和现场可编程门阵列(Field Programmable Gate Array，FPGA)。ASIC 是一种为特定应用设计的集成电路，具有极高的性能和能效，但其设计周期长，成本高，缺乏灵活性，无法适应快速变化的应用需求。CPU 并行计算能力有限，能效较低。GPU 具有强大的并行计算能力，被广泛用于深度学习任务的训练和推理，但能效比较低。FPGA 是一种可编程的集成电路，具有高度并行性和硬件级别的定制特性，能效比 CPU 和 GPU 更高[6]，为 CNN 硬件部署提供了良好的加速器设计验证平台。

基于 FPGA 平台实现 CNN 加速器已受到广泛关注[7-11]，并且取得了较好的加速效果。但当前方法往往针对给定硬件约束对特定 CNN 模型进行加速器的设计，缺乏通用和自适应的加速器设计方案；并且，当前方法要求模型开发者熟悉硬件架构，需针对性开展 IP 核和访存结构设计，实现难度较大。对此，本文面向低资源芯片，提出了一种高效自适应卷积神经网络加速器实现方案 ACNNA，主要贡献包括 3 个方面。

1)设计了一种资源折叠式的多通道处理引擎(Processing Engine，PE)阵列，将理想化的卷积单元结构进行折叠以节省资源，在输出通道上展开以支持并行计算。

2)设计了一种支持存储与计算资源复用的系统结构，采用多级存储方式并利用乒乓缓存机制对流水线进行优化，提升数据处理效率。此外，提出了一种设计空间探索算法，其能够合理地针对网络参数调度硬件资源分配，使得低资源芯片能够部署层次更深且参数更多的网络模型。

3)基于 Ultra 96 V2 开发板，使用 ACNNA 对 LeNet5 和 VGG16 进行了部署。结果表明，采用 ACNNA 部署的 VGG16 最低仅消耗了原网络 4%的资源量。100 MHz 工作频率下，LeNet5 加速器在 2.05 W 的功耗下计算速率达 0.37 GFLOPS；VGG16 加速器在 2.13 W 的功耗下计算速率达 1.55 GFLOPS。与现有工作相比，ACNNA 的每秒帧数(Frames Per Second，FPS)提升超过 83%。

本文第 2 章介绍了相关工作；第 3 章给出了加速器设计方案；第 4 章是实验设置及结果评估。

2 相关工作

2.1 卷积神经网络

LeNet5 是 CNN 的开端，该网络包含 6 万余个参数[12]。VGG16 网络参数量达 1.38 亿[13]，在图像分类任务中获得了极高的精度。复杂 CNN 模型对存储空间和计算资源的需求急剧增加，然而，现实中广泛使用的边缘设备资源不足，其片上缓存容量仅以 MB 为单位，十分有限。例如，ZYNQ 7020 与 XCZU3EG 硬件平台的片上存储容量分别为 0.61MB 与 0.95MB，最高仅能容纳 25 万浮点数。因此，将复杂且庞大的模型高效部署到低资源芯片上面临巨大挑战。

2.2 可重构硬件加速器

CNN 模型通常在 Keras，TensorFlow，Pytorch 和 Caffe 等环境中设计开发训练，但此类高级语言训练的模型无法直接用于 FPGA 部署。为了实现高级语言设计和硬件实现之间的映射，高层次综合(High-Level Synthesis，HLS)工具应运而生[14]。HLS 工具能够将高级编程语言如 C，C++，SystemC 编写的程序自动转换为硬件描述语言的代码[15]。尽管如此，若想在 FPGA 上实现网络模型的灵活部署，设计人员仍需具备专业的硬件知识，以分配硬件资源，找到最佳配置，且需要较长的开发周期。业界针对各种网络模型提出的多种硬件加速方案[7-11]，但当前方法仅适用于特定网络，难以适配到多种网络模型。Chen 等提出的 Eyeriss 加速器[16]，通过可重构架构和行驻留数据流来优化能效，减少数据移动。该方法通过局部重用数据和应用压缩技术，实现了高吞吐量和能效。Tarek 等提出了一种利用云服务器工具实现从 Python 到比特流的端到端深度神经网络部署系统[17]。Mousouliotis 等创建了自动化流程工具 CNN Grinder[18]，其中包含一些模板引导用户，并提供了标准化的步骤和结构，使得用户可以按照预定的流程进行设计和验证，最终将 CNN 映射到特定的 FPGA 平台。Venieris 等提出了一种高层次自动化设计框架 fpgaConv-Net[19]，以增强规则和不规则 CNN 模型的映射，并使用基于同步数据流(SDF)的自动化设计技术快速探索架构替代方案。Acosta 等允许用户选择数据集并自定义 CNN 模型[20]，自动构建基于 FPGA 的定制硬件 CNN 加速器。Mazouz 等提出了一种基于 MATLAB 的自动化设计流程，用于在 FPGA 部署可重构的 CNN 模型[21]。设计人员获取 CNN 架构后，引入新的延迟和空间约束选项，通过循环重新排序、展开和流水线处理，框架能够自动生成多种设计方案。Wang 等提出的 TensorMap[22]，能够为空间加速器和张量计算生成优化的配置。Andrulis 等提出的 CiMLoop[23]，使得设计者可以描述、建模和映射工作负载到电路和架构。Wu 等提出的 Amoeba 加速器[24]，使用内核分割方法优化数据流，能够在不牺牲效率的情况下支持具有任意内核大小的 CNN。Jia 等提出的 XVDPU[25]，基于 Versal 芯片的 AI 引擎来加速 CNN，通过多批处理、共享权重、特征平稳映射和长负载权重方法提高数据复用率。Nag 等提出了一种可配置的硬件加速器 ViTA[26]，其通过流水线优化提高模型推理速度，利用层间多层感知机优化减少数据传输和计算开销。Xu 等提出的 Flare[27]，通过整合矢量点积以统一卷积和全连接层，将输入特征图的行向量视为基本处理单元，在消除数据重排的同时平衡了处理延迟和资源消耗。部分相关工作与本文方法的对比如表 1 所列。

现有大多数方案在部署过程中仍需设计人员调控相关参数，影响了快速部署和调整的能力。ACNNA 利用多通道 PE 阵列，将理想化卷积单元结构进行折叠以节省资源，在输出通道展开以并行加速计算，采用乒乓缓存对流水线进行优化。此外，提出的设计空间探索算法可根据硬件资源限制调整映

射配置，确保在硬件资源约束下选择最佳性能方案。本文方法通过引入自动化的设计工具链，从方案生成到加速器部署整个过程大幅削减了人工介入，使得加速器的部署与调整流程更加高效，能够快速适应不同应用场景和计算需求。

表 1 部分相关工作与本文方法的对比

Table 1 Comparison of related work with the proposed method

现有方案及特性	ACNNA
需要手动设计加速器：Grinder[18]，TensorMap[22]，CNN CiMLoop[23]，Amoeba[24]，Flare[27]	自动化设计工具链，减少人工干预
资源优化配置复杂：fpgaConvNet[19]，Amoeba[24]，ViTA[26]，Flare[27]	设计空间探索算法，自动优化配置
自动化设计加速器：文献[20]基于 GUI，文献[21]基于 MATLAB	基于脚本指令 Tcl 与高层次综合工具的自动化设计
依赖特定硬件平台：XVDPU[25]	可适配不同硬件平台，通用性强
针对特定网络设计，难以支持大规模网络部署加速	灵活折叠卷积单元，支持大规模网络部署加速

3 加速器设计方案

3.1 自动化框架

ACNNA 使用开放神经网络交换（Open Neural Network Exchange，ONNX）模型文件作为输入，自动生成比特流并部署，设计人员无需掌握专业硬件设计知识就可实现 CNN 的高效部署。ACNNA 的自动化框架如图 1 所示，可分为 4 个层次：模型层、设计层、系统层、部署层。

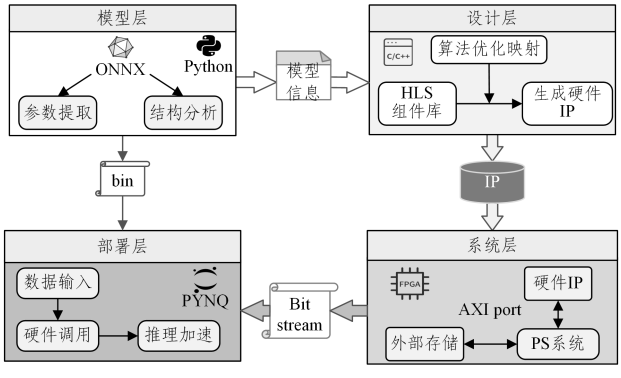


图 1 ACNNA 工作流

Fig. 1 Workflow of ACNNA

1）模型层解析模型结构，提取并记录各模型层的配置参数，如卷积核大小、步长、池化方式等。ACNNA 采用 ONNX 的网络模型作为输入。ONNX 是一个开放标准，支持多种深度学习框架（如 PyTorch，TensorFlow，MXNet[28] 等），具有较好的通用性。

2）设计层使用 HLS 工具，根据模型层所提取的参数调用预先构建好的 C/C++模板代码作为算子，映射整个模型结构并进行仿真验证。在映射同时进行设计空间探索，优化硬件资源分配和利用，确保在有限的硬件资源下实现最佳性能。

3）系统层利用 Tcl 脚本将生成的 IP 核集成到指定 FPGA 设计中，并生成最终的比特流文件与硬件描述文件。

4）部署层采用 PYNQ[29] 软件平台进行部署。PYNQ 平台集成了 Jupyter notebook 的开源框架并提供了与 FPGA 交互的接口，用户可使用比特流与硬件描述文件对 FPGA 部署网络模型。

通过这种系统化的工具链设计，ACNNA 能够高效灵活地使用 ONNX 模型作为输入，生成对应的硬件加速器，并在 PYNQ 平台上实现高性能的网络推理。

3.2 加速器系统结构

加速器的系统结构由片外存储、两级片上缓存、PE 阵列及控制单元组成，如图 2 所示。片外存储用于存放全部的特征和权重数据。两级片上缓存分为全局缓存与单元缓存。全局缓存用于存储多输出通道的特征数据、权重数据和输出特征，以支持多输出通道的并行计算与输出。单元缓存则用于存储单次卷积所需的特征与数据，作为乒乓缓存机制中的中间数据运载单元。PE 阵列中包含多个 PE 单元，在网络模型中将具有相同宽与高的卷积核称为一类卷积核，阵列类型的数量根据网络模型中卷积核的类型数量确定，单个阵列中 PE 单元的数量则由设计算法得出的计算并行度指定，并行度每增加 1，则阵列中 PE 单元的数量增加 1 个。片上全局缓存大小由系统根据设计空间探索算法在资源限制下得出的最佳并行度进行分配，而单元缓存则只与卷积核大小相关，这将在 3.5 节中详细介绍。

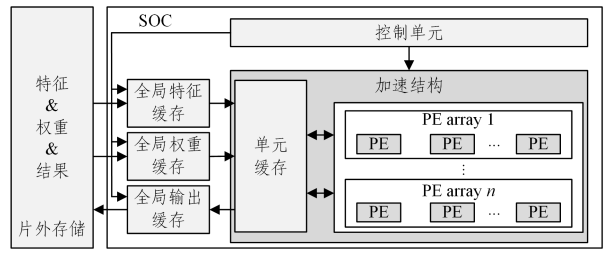


图 2 系统结构

Fig. 2 System structure

当卷积层进行推理时，系统从片外存储读取所需特征与权重数据并写入对应的片上全局缓存，随后从全局特征与权重缓存中使用单元缓存运载数据至 PE 阵列运算并输出结果至输出缓存，再由输出缓存将多通道的输出特征传递至片外存储。通过循环此计算流程，可对系统所需的存储和计算资源进行复用，提高资源利用率，在部署大规模卷积神经网络时节省硬件资源开销。

3.3 计算引擎 PE 及阵列设计

多数 CNN 模型中，卷积层的计算量占总运算量的 90% 以上，因此加速 CNN 的关键挑战在于如何利用硬件高效计算卷积层的运算。卷积层作为 CNN 中的核心部分，负责提取输入特征图中的局部特征，其运算过程可以描述为卷积核在输入特征图上按一定的步长滑动，每次滑动都将卷积核与特征图当前窗口的对应元素逐一相乘并累加，再将多个输入通道结果累加，得到一个输出通道的对应点输出特征。

以 VGG16 网络中 3×3 卷积过程的最大并行度映射为例，其理想化的 PE 结构如图 3 所示。整个计算过程被展开为 5 个级别：第一级包含了 3×3 矩阵的 9 个乘法操作，第二级至第五级则将第一级的 9 个计算结果进行逐级累加，整个计算过程并行执行，卷积的结果暂存于输出寄存器中。该结构虽能最大并行化单次卷积计算，但需要实例化每个计算结构，在计算过程中，每个乘法器或加法器只执行一次计算，导致资源利用率较低。

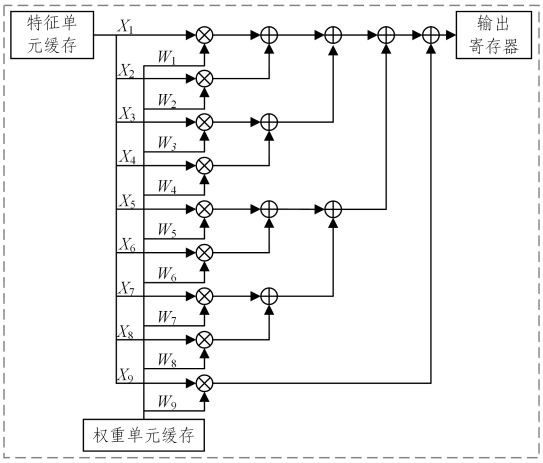


图 3 理想化 3×3 卷积结构

Fig. 3 Idealized 3×3 convolutional structure

为提高资源利用率，本文提出了一种资源折叠式的多通道 PE 阵列，如图 4 所示。同样以 3×3 卷积过程为例，该结构将图 3 中的理想化结构在单元内进行折叠复用，在通道间展开。图 4 中的特征与权重缓存为单元缓存，输出缓存为全局缓存，由于数据均存储在双端口 RAM 中，故只需配置两个乘法器即可同时支持两个乘法运算并输出结果到寄存器中，其后配置多个加法器进行累加运算，最终输出至输出缓存。当卷积核大小为 $k\times k$ 时，其映射的 PE 阵列与图 4 中的结构相似，单个 PE 单元需配置两个乘法器与 $\lceil 2\log_2 k\rceil+1$ 个加法器，其中 $\lceil\cdot\rceil$ 代表向上取整。若 PE 阵列计算并行度为 n，则阵列需配置 $2n$ 个乘法器与 $n\times(\lceil 2\log_2 k\rceil+1)$ 个加法器。

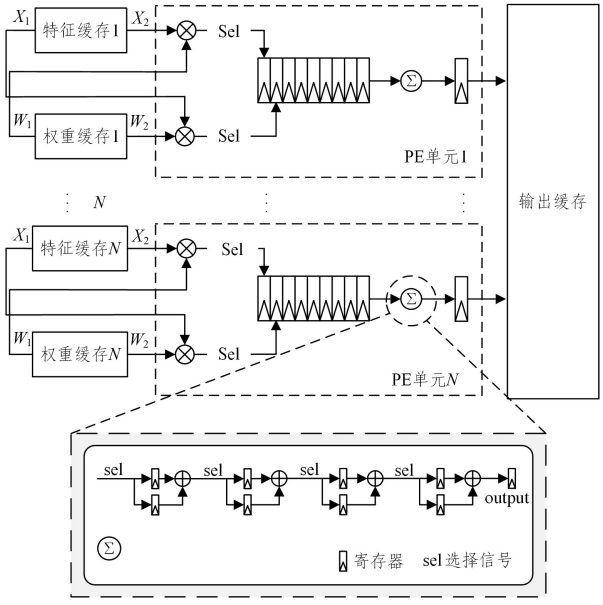


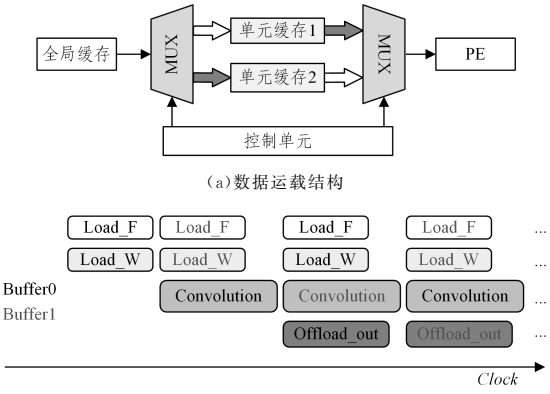
图 4 3×3 卷积核的多通道 PE 阵列

Fig. 4 Multi-channel PE array with 3×3 convolutional kernel

由于 PE 单元在输出通道上相互独立，因此组合多个 PE 单元可构成一个支持在输出通道上并行计算的 PE 阵列，且该阵列计算并行度与其内包含的 PE 单元数量一致，增加阵列中的 PE 单元数量，则会对应增加硬件的计算并行度，由此可提高卷积层通道间的计算效率以及系统吞吐量。实质上，通过调整通道计算的并行度即可根据需求对存算资源和数据吞吐量进行平衡，实现更高效的系统运作。ACNNA 将模型层内具有相同宽与高的卷积核记为一类卷积核，若整个网络同时存在多种大小的卷积核即多类卷积核，则设计不同类型的 PE 单元及阵列用于计算。

3.4 流水线优化

为进一步提高从全局缓存至 PE 单元的数据运载效率，ACNNA 采用了乒乓缓存机制，在加载数据时使用两个单元存储作为乒乓 RAM 进行交替加载，能够有效减少 PE 对数据等待的时间并提高设计的吞吐量。PE 中的特征与权重缓存均采用了乒乓缓存机制，其传输示意图如图 5(a)所示。在从全局缓存向单元缓存 1 传输数据的同时，PE 可以并行地从单元缓存 2 中读取数据进行卷积操作，当单元缓存 2 中的数据计算完成后，PE 可以从单元缓存 1 中继续读取数据进行计算，其工作流水线如图 5(b) 所示。此模式能够省掉除第一次以外的所有数据加载时间，使 PE 在第一次加载数据后始终处于工作状态，提高其运算效率。



(a)数据运载结构

(b)乒乓缓存流水线

图 5 乒乓缓存机制

Fig. 5 Ping-pong buffer mechanism

相比之下，流水线分段方法虽然也能提高并行性，但数据加载与运算具有依赖关系，导致 PE 在计算过程中增加了大量的数据等待时间。乒乓缓存是一种以存储空间换取传输带宽的机制，本设计中的单元缓存仅需容纳单次卷积所需数据，不仅可以提升数据运载效率，而且能降低资源消耗。

3.5 资源分配与设计空间探索

在 FPGA 上部署神经网络时，片上存算资源(BRAM 和 DSP)通常最为受限。本文提出了支持资源复用的硬件系统结构与设计空间探索算法来进行资源优化分配。ACNNA 根据网络各层参数及类别，将需分配的资源区分为静态和动态资源。静态资源定义为不跟随计算并行度而改变大小的资源，包括全局缓存中的特征缓存以及网络中的通用池化层与全连接层占用的资源，这些资源的大小设置为支持网络中的最大参数，用于支持不同层的复用。设在卷积神经网络的所有卷积层中，最大的输入通道数为 C_{in}，最大的输入特征的高和宽为 W_{in} 和 H_{in}，卷积核的高和宽均为 K_n，最大输出特征的高和宽为 W_{out} 和 H_{out}，全连接层最大的输入通道数和输出通道数分别为 Fc_{in} 和 Fc_{out}，数据位宽为 D_{w}，则特征缓存大小为 $C_{\text{in}}\times W_{\text{in}}\times H_{\text{in}}\times D_{\text{w}}$。在本工作中，为贴合大多数网络的输出通道数，池化层与全连接层在输出通道的计算并行度均设为 2，则池化层的输出缓存为 $2\times W_{\text{out}}\times H_{\text{out}}\times D_{\text{w}}$，全连接层的

输入和输出缓存大小分别为 $Fc_{in}\times D_w$ 和 $Fc_{out}\times D_w$，权重缓存的大小为 $2\times Fc_{in}\times D_w$。动态资源指跟随计算并行度而变化的资源，包括全局缓存中的权重和输出缓存与单元缓存中的特征和权重缓存。若计算并行度为 p，则全局缓存中的权重缓存大小为 $p\times C_{in}\times K_n^2\times D_w$，输出缓存大小为 $p\times W_{out}\times H_{out}\times D_w$，单元缓存中特征与权重缓存大小均为 $p\times K_n^2\times D_w$。

ACNNA 根据卷积层输出通道的数量确定系统的计算并行度，当并行度为卷积层输出通道数的因子时，对应的 PE 阵列在运算与数据运载的复用过程中不会进行额外的冗余操作。反之，若卷积层计算并行度超过其输出通道数或不为该层的因子，则会在最后一次数据加载过程中存在多余通道数据的加载和非必要计算，造成额外的数据运载时延与能量消耗。将网络模型中的同一类卷积层设计为同一并行度时，此类卷积层映射为硬件则是复用同一 PE 阵列进行卷积计算。当网络模型存在不同的卷积核大小时，在硬件上的体现为例化不同的 PE 阵列及单元缓存空间以分别适配。

最佳计算并行度 pb，使得在硬件平台资源限制下加速器中的并行计算单元达到最多。本文提出的设计空间探索算法如算法 1 所示，可分为两个部分。第一部分为可用并行度探测：以硬件资源限制与网络模型参数为输入，首先构建实现模型推理的最简加速器与卷积层 PE 阵列并得到对应资源量，再求解出硬件资源的冗余空间与该空间下卷积层 PE 阵列并行度的上限，遍历卷积层输出通道数因子，筛选出小于该上限的可用并行度。第二部分为并行度择优：将可用并行度作为输入，遍历每一种可用并行度，根据必要操作数与设计工具综合得出的不同操作的单位时间，计算不同并行度下的运算时间与冗余传输时间和，并将其作为评估指标进行比较，将最少时间和作为最优指标来选取最优计算并行度。

算法 1 设计空间探索算法

输入：CNN 各层参数列表 $C_p[\ ]$，卷积核大小 S，层数 L，硬件资源约束 R

输出：并行度列表 p[ ]

```
init R_t, P[] / * 初始化资源占用与并行度列表 * /
get_max_param(C_p[ ], M_p[ ]) / * 获取最大参数 * /
set_min_Model(M_p[ ], R_e) / * 例化最简模型 * /
set_convPE(S, R_c) / * 例化卷积层 * /
R_d = R - (R_e - R_c) / * 计算设计空间 * /
p_max = Round(R_d / R_c) / * 获取并行度上限 * /
for i from 1 to L do / * 遍历卷积层 * /
    get_factor(C_p[i], x[m]) / * 获取通道数因子 * /
    for j from 1 to m do / * 遍历所有因子 * /
        while x[j] < p_max do
            record(x[ ], p[ ]) / * 记录可用并行度 * /
        end while
    end for
end for
init T_min = +∞; / * 初始化最短时间 * /
init pb = -1 / * 初始最佳并行度 * /
for k from 1 to n do / * 遍历可用并行度 * /
    T_s = count_spare_T(p[k]) / * 冗余传输时间 * /
    T_c = count_computed_T(p[k]) / * 运算时间 * /
    T = T_s + T_c
    if T < T_min do / * 若消耗时间最少 * /
        pb = p[k] / * 记录为最佳并行度 Pb * /
        T_min = T
    end if
end for
return pb
```

4 实验评估

4.1 实验环境

本文采用了 Ultra96 V2 开发板作为硬件平台，该平台嵌入了基于 Xilinx Zynq 系列的 ZU3EG SBVA484 芯片，芯片内部集成了处理系统（Processing System，PS）和可编程逻辑（Programmable Logic，PL）。PS 端搭载了四核 Cortex-A53 处理器，使得其能够运行 PYNQ 平台，PL 端搭载 FPGA 逻辑资源。本文在 ACNNA 的设计层和部署层中使用的工具分别为 Vitis HLS 2023.2 和 Vivado 2023.2。实验中，选取了 LeNet5 与 VGG16 两种网络进行部署测试，模型数据集、参数量、操作数量如表 2 所列。

表 2 模型配置

Table 2 Model configuration

模型	数据集	参数量/万	操作数/FLOPs
LeNet5	MNIST	60000	828520
VGG16	CIFAR10	2110000	26316960

4.2 资源与性能分析

通过本文提出的设计空间探索算法得到的 LeNet5 与 VGG16 的全精度模型部署的可用并行度列表及最佳计算并行度如表 3 所列。算法给出 LeNet5 网络的最佳并行度为 8，VGG16 网络的最佳并行度为 16。在 HLS 工具为设计方案自动生成 IP 核后，经 Vivado 综合实现测得 LeNet5 的部分并行度方案资源占用如图 6 所示，VGG16 的部分并行度方案资源占用如图 7 所示，图中给出的资源量为整个网络模型的资源消耗（包含池化层与全连接层，为匹配多数池化层与全连接层的输出通道数，此类层计算并行度经验性地设为 2）。

表 3 可用并行度列表

Table 3 List of available parallelism

模型	可用并行度	最佳并行度
LeNet5	2,4,6,8,12,16,20	8
VGG16	2,4,8,16	16

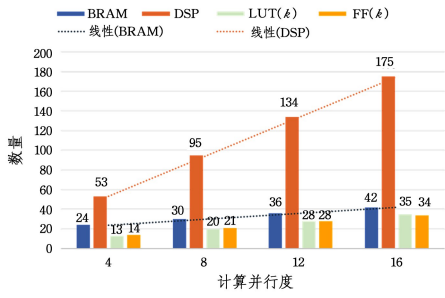


图 6 LeNet5 不同并行度下的资源量

Fig. 6 Resource usage of LeNet5 under different parallelism

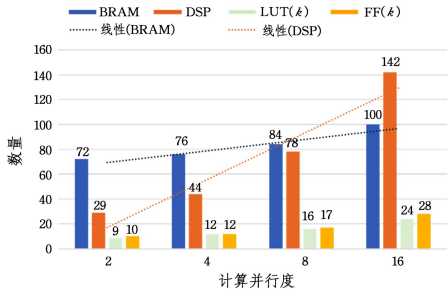


图 7 VGG16 不同并行度下的资源量

Fig. 7 Resource usage of VGG16 under different parallelism

由图 6、图 7 可知，随着卷积层计算并行度的提升，两种网络模型部署所需的存储与计算资源均提升，并几乎呈线性增加。由图 7 可知，VGG16 加速器在卷积层计算并行度为 2 的方案中，所使用的 BRAM 数量为 72，将该数据进行换算得到其对应所能容纳的参数量仅为该模型参数量的 4%，由此可验证本文所提出的支持资源复用的系统结构可有效节约片上资源。

LeNet5 与 VGG16 在不同计算并行度方案部署下的功率和吞吐量如表 4 所列，帧率由模型在 PYNQ 端部署后测得，其中“6(1),8”代表第一位卷积层并行度为 6 而其余为 8。在 100 MHz 工作频率下，LeNet5 网络选用并行度为 8 的方案的吞吐量超过了并行度为 16 的方案，这是因为 LeNet5 网络的 3 个卷积层输出通道数分别为 6,16,120。在并行度为 16 的设计方案中，由于第一层和第三层的输出通道数均不能被并行度整除，因此 PE 阵列在第一卷积层和第三卷积层时分别加载了 10 个通道与 8 个输出通道的冗余数据，造成了额外的数据加载时延。并行度为 8 时，仅在第一卷积层加载了 2 个输出通道的冗余数据，远低于并行度为 16 时的冗余数据加载。虽然 PE 阵列计算效率与其并行度成正比关系，但对于 LeNet5 这类小模型，高并行度所带来的加速效果难以抵消其所带来的冗余计算影响。因此，并行度为 16 的设计方案在有效吞吐量方面低于并行度为 8 的设计方案。同时，这也证明了本文所提出的设计空间探索算法对并行度择优的有效性。

表 4 不同并行度下的功率与吞吐量

Table 4 Power and throughput at different parallelism

模型	并行度	功率/W	FPS
LeNet5 (fp32)	2	1.946	223
	6(1),8	2.143	445
	8(最佳)	2.050	447
	12	2.082	414
	16	2.141	433
VGG16 (fp32)	4	2.003	25
	4(1−4),8	2.095	29
	8	2.058	43
	8(1−2),16	2.194	54
	16(最佳)	2.132	59

由 VGG16 网络部署的测试结果可得，吞吐量与计算并行度呈非线性关系，这是因为加速器中存在数据传输的带宽限制。在并行度数量方面，采取多并行度的方案相比于单并行度方案并没有带来较高增益，反而增加了不必要的资源开销，导致资源利用率下降。因此，在本方法的模型部署过程中采用多并行度是不必要的。

经计算，LeNet5 和 VGG16 网络在其最佳并行度下的吞吐量分别为 0.37 GFLOPs 和 1.55 GFLOPs。为更好地评估本文提出的 ACNNA 方法，将其与其他将 LeNet5 部署在 MNIST 数据集上进行推理的方法[30-31]进行性能对比，结果如表 5 所列。对比文献中采用的 ZYNQ 7020 与 ZC702 的硬件平台资源完全一致，本文采用的 Ultra96 V2 同属 XILINX 的 ZYNQ 平台，同为 ARM+FPGA 的架构，区别在于后者的 PS 端为 Cortex-A53，而前者为 Cortex-A9，PL 端的资源量更多。相比文献[31]中的卷积神经网络设计实现方法，ACNNA 性能提升超过 83%，能够在更短的时间内实现推理。

表 5 与现有工作的性能对比

Table 5 Performance comparison with existing work

方法	模型	硬件平台	FPS
文献[30]	LeNet5(float32)	ZYNQ 7020	42
文献[31]	LeNet5(int16)	ZC702	244
ACNNA	LeNet5(float32)	Ultra96 V2	447

结束语 本文提出了一种面向低资源芯片的自适应卷积神经网络加速器，从硬件资源限制到对网络模型进行综合考量，为卷积层设计了资源折叠式的多通道 PE 阵列，通过与片上资源搭配，支持在模型的计算层复用，减少了资源消耗，并且设计者可以通过对并行度的指定实现在存算资源和性能间的自由选择。实验结果表明，ACNNA 能够高效且灵活地部署 CNN 模型，最低仅占用 4%的资源量，最高吞吐量相比于现有工作提升 83%。

本文设计的方法更倾向于节省片上存储资源，由于使用了片外存储和两级片上缓存，因此存在大量中间数据运载操作，一定程度上影响了模型部署后的性能。接下来将考虑对存储结构与数据搬运流水线进行进一步优化，适当增加片上存储资源使用，以提升模型部署后的性能。

参 考 文 献

[1] CHEN X, XIE L X, WU J, et al. Cyclic CNN: Image Classification With Multiscale and Multilocation Contexts [J]. IEEE Internet of Things Journal, 2021, 8: 7466-7475.

[2] HUANG L, CHEN C, YUN J T, et al. Multi-Scale Feature Fusion Convolutional Neural Network for Indoor Small Target Detection [J]. Frontiers in Neurorobotics, 2022, 16: 881021.

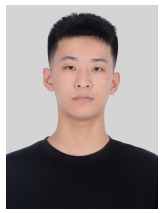
[3] HEMA C R, MÁRQUEZ F P G. Emotional speech Recognition using CNN and Deep learning techniques [J]. Applied Acoustics, 2023, 211: 109492.

[4] JANG B, KIM M, HARERIMANA G, et al. Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism [J]. Applied Sciences, 2020, 10: 5814.

[5] SYED R T, MARKO S, ULBRICHT M, et al. Towards Reconfigurable CNN Accelerator for FPGA Implementation [J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2023, 70: 1249-1253.

[6] BJERGE K, SCHOUGAARD J H, LARSEN D E. A scalable and efficient convolutional neural network accelerator using HLS for a system-on-chip design [J]. Microprocess and Microsystems, 2021, 87: 104363.

[7] ZHANG Z C, MAHMUD M A, KOUZANI A Z, et al. FitNN: A Low-Resource FPGA-Based CNN Accelerator for Drones [J]. IEEE Internet of Things Journal, 2022, 9: 21357-21369.
[8] LI, S Z, WANG Q, JIANG J F, et al. An Efficient CNN Accelerator Using Inter-Frame Data Reuse of Videos on FPGAs [J]. IEEE Transactions on Very Large Scale Integration(VLSI) Systems, 2022, 30: 1587-1600.
[9] YAN S, LIU Z, WANG Y, et al. An FPGA-based MobileNet Accelerator Considering Network Structure Characteristics [C]// 31st International Conference on Field-Programmable Logic and Applications(FPL). 2021: 17-23.
[10] WANG B, WEI X L, WANG C, et al. Adaptive design and implementation of automatic modulation recognition accelerator [J]. Journal of Ambient Intelligence and Humanized Computing, 2024, 15: 1-17.
[11] BAO C, XIE T, FENG W B, et al. A Power-Efficient Optimizing Framework FPGA Accelerator Based on Winograd for YOLO [J]. IEEE Access, 2020, 8: 94307-94317.
[12] LECUN Y, BOTTOU B, BENGIO Y, et al. Gradient-based learning applied to document recognition [C]// Proceedings of the IEEE. 1998: 2278-2324.
[13] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-ScaleImage Recognition [J]. arXiv: 1409. 1556, 2014.
[14] RIAZATI M, DANESHTALAB M, SJODIN M, et al. AutoDeepHLS: Deep Neural Network High-level Synthesis using fixed-point precision [C]// 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS). 2022: 122-125.
[15] NANE R, SIMA V M, PILATO C M, et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, 35: 1591-1604.
[16] CHEN Y H, KRISHNA T, EMER J S, et al. Eyeriss: An energy efficient reconfigurable accelerator for deep convolutional neural networks [J]. IEEE Solid-State Circuits, 2017, 52: 127-138.
[17] BELABED T, SILVA V R, QUENON A, et al. A Novel Automate Python Edge-to-Edge: From Automated Generation on Cloud to User Application Deployment on Edge of Deep Neural Networks for Low Power IoT Systems FPGA-Based Acceleration [J]. Sensors, 2021, 21: 6050.
[18] MOUSOULIOTIS P G, PETROU L P. CNN-Grinder: From Algorithmic to High-Level Synthesis descriptions of CNNs for Low-end-low-cost FPGA SoCs [J]. Microprocessors and Microsystems, 2020, 73: 102990.
[19] VENIERIS S I, BOUGANIS C. fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs [C]// 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines(FCCM). 2016: 40-47.
[20] RIVERA-ACOSTA M, ORTEGA-CISNEROS S, RIVER A. Automatic Tool for Fast Generation of Custom Convolutional Neural Networks Accelerators for FPGA [J]. Electronics, 2019, 8: 641.
[21] MAZOUZ A, BRIDEGS C P. Automated Offline Design-Space Exploration and Online Design Reconfiguration for CNNs [C]// 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems(EAIS). 2020: 1-9.
[22] WANG F, SHEN M, LU Y, et al. TensorMap: A Deep RL-Based Tensor Mapping Framework for Spatial Accelerators [J]. IEEE Transactions on Computers, 2024, 73: 1899-1912.
[23] ANDRULIS T, EMER J S, SZE V. CiMLoop: A Flexible, Accurate, and Fast Compute-In-Memory Modeling Tool [C]// 2024 IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS). 2024: 10-23.
[24] WU X, WANG M, LIN J, et al. Amoeba: An Efficient and Flexible FPGA-Based Accelerator for Arbitrary-Kernel CNNs [J]. IEEE Transactions on Very Large Scale Integration(VLSI) Systems, 2024, 32: 1086-1099.
[25] JIA X, ZHANG Y, LIU G, et al. XVDPU: A High Performance CNN Accelerator on the Versal Platform Powered by the AI Engine [C]// 2022 32nd International Conference on Field-Programmable Logic and Applications(FPL). 2022: 1-9.
[26] NAG S, DATTA G, KUNDU S, et al. ViTA: A Vision Transformer Inference Accelerator for Edge Applications [C]// 2023 IEEE International Symposium on Circuits and Systems (ISCAS). 2023: 1-5.
[27] XU Y, LUO J, SUN W. Flare: An FPGA-Based Full Precision Low Power CNN Accelerator with Reconfigurable Structure [J]. Sensors, 2024, 24: 2239.
[28] CHEN T Q, LI M, LI Y, et al. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems [J]. arXiv: 1512. 01274, 2015.
[29] WANG E, DAVISJ J, CHEUNG P Y. A PYNQ-Based Framework for Rapid CNN Prototyping [C]// 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines(FCCM). 2018: 223-223.
[30] CHEN S H, WU J M, PENG K J et al. Design and Implementation of Convolutional Neural Network Accelerator Based on ZYNQ Platform [J]. Chinese Automation and Information Engineering, 2024, 45(1): 30-34.
[31] WANG Y L, XIE K L, CHEN S Y, et al. A universal design on hardware acceleration of convolutional neural networks [J]. Chinese Computer Science Engineering, 2023, 45(4): 577-581.



PANG Mingyi, born in 2001, postgraduate. His main research interests include hardware acceleration and edge computing.

WEI Xianglin, born in 1985, Ph.D, associate researcher. His main research interests include edge computing, deep learning and wireless network security.

(责任编辑:何杨)