

# 基于 RISC-V 的深度可分离卷积神经网络加速器

曹希或 陈鑫 魏同权

(华东师范大学计算机科学与技术学院 上海 200062)

**摘要** 人工智能时代, RISC-V 作为一种新兴的开源精简指令集架构, 因其低功耗、模块化、开放性和灵活性等优势, 使之成为一种能够适应不断发展的深度学习模型和算法的新平台. 但是在硬件资源及功耗受限环境下, 基础的 RISC-V 处理器架构无法满足卷积神经网络对高性能计算的需求. 为了解决这一问题, 本文设计了一个基于 RISC-V 的轻量化深度可分离卷积神经网络加速器, 旨在弥补 RISC-V 处理器的卷积计算能力的不足. 该加速器支持深度可分离卷积中的两个关键算子, 即深度卷积和点卷积, 并能够通过共享硬件结构提高资源利用效率. 深度卷积计算流水线采用了高效的 Winograd 卷积算法, 并使用  $2 \times 2$  数据块组合拼接成  $4 \times 4$  数据片的方式来减少传输数据冗余. 同时, 通过拓展 RISC-V 处理器端指令, 使得加速器能够实现更灵活的配置和调用. 实验结果表明, 相较于基础的 RISC-V 处理器, 调用加速器后的点卷积和深度卷积计算取得了显著的加速效果, 其中点卷积加速了 104.40 倍, 深度卷积加速了 123.63 倍. 与此同时, 加速器的性能功耗比达到了 8.7 GOPS/W. 本文的 RISC-V 处理器结合加速器为资源受限环境下卷积神经网络的部署提供了一个高效可行的选择.

**关键词** 神经网络; 深度可分离卷积; RISC-V; Winograd 快速卷积; 硬件加速

**中图法分类号** TP18 **DOI号** 10.11897/SP.J.1016.2024.02536

## Efficient Accelerator for Depthwise Separable Convolutional Neural Networks Based on RISC-V

CAO Xi-Yu CHEN Xin WEI Tong-Quan

(School of Computer Science and Technology, East China Normal University, Shanghai 200062)

**Abstract** In the era of artificial intelligence, RISC-V, as an emerging open-source Reduced Instruction Set Computing architecture, has become a new platform capable of adapting to evolving deep learning models and algorithms due to its advantages such as low power consumption, modularity, openness, and flexibility. However, in environments with constrained hardware resources and power, the basic RISC-V processor architecture falls short of meeting the high-performance computing demands of convolutional neural networks. To address this issue, this paper introduces a lightweight depthwise separable convolutional neural network accelerator based on RISC-V, aiming to compensate for the insufficient convolutional computation capabilities of RISC-V processors. The accelerator supports two key operators in depthwise separable convolution: depthwise convolution and pointwise convolution, and enhances resource utilization efficiency through shared hardware structures. The depthwise convolution computation pipeline employs an efficient Winograd convolution algorithm and reduces data redundancy by combining  $2 \times 2$  data blocks into  $4 \times 4$  data tiles. Additionally, by extending RISC-V instructions, the accelerator achieves more flexible configuration and invocation. Experimental results demonstrate

significant acceleration in pointwise and depthwise convolution computations compared to the basic RISC-V processor, with a speedup of 104.40x for pointwise convolution and 123.63x for depthwise convolution. Meanwhile, the performance-to-power ratio of the accelerator reaches 8.7 GOPS/W. The combination of the RISC-V processor and the accelerator presented in this paper offers an efficient and viable choice for deploying convolutional neural networks in resource-constrained environments.

**Keywords** neural networks; depthwise separable convolution; Reduced Instruction Set Computer-V; Winograd fast convolution; hardware acceleration

## 1 引言

近年来,随着硬件性能不断提升,神经网络领域也经历了快速发展<sup>[1]</sup>.特别是卷积神经网络在图像相关任务中表现出色,能够自动检测图像或视频等数据中的模式和特征.它可应用于图像分类、脑机接口、物体检测、医学图像分析、自动驾驶等多个领域,这将人工智能引入了众多实际场景,提高了各种应用程序的性能和智能水平,为人们的日常生活带来了巨大便利.然而,由于卷积神经网络模型参数量与计算量十分庞大,并且主要计算类型与计算量集中在卷积操作,其大量的乘累加计算与庞大的模型参数量对资源受限的边缘智能设备的模型部署带来了严峻挑战<sup>[2]</sup>.

第五代精简指令集处理器(Reduced Instruction Set Computer-V, RISC-V)是一种源自加州大学伯克利分校的全面开放、可扩展的架构处理器,允许任何人免费使用并遵照其规范自定义设计处理器核心. RISC-V 采用了模块化的指令集架构,允许设计各种类型的微处理器,以满足不同应用场景的需求. RISC-V 具有小巧、低功耗、易于扩展实现、完全开放等特点,它逐渐成为一种能够适应不断发展的深度学习模型和算法的新平台.相比于图形处理器<sup>[3]</sup>(Graphics Processing Unit, GPU), RISC-V 处理器拥有更低的功耗;相比于 ARM 指令集中央处理器<sup>[4]</sup>(Central Processing Unit, CPU), RISC-V 处理器无需高昂的授权费用,且结构更加灵活.

但是仅仅使用基础的 RISC-V 处理器并不能满足神经网络对高性能计算的要求,尤其是在边缘计算和物联网等硬件资源及功耗受限的环境下.因此在 RISC-V 架构下,常采用处理器与加速器协同工作的模式来高效地部署神经网络.定制化的加速器

可以根据特定任务的需求和性能目标进行设计,提供更好的性能和效率.通过指令扩展功能, RISC-V 指令集能够方便地调用加速器来加速运算.

深度可分离卷积常用于计算能力有限的场景中,其通过将标准卷积拆分为深度卷积和点卷积,显著减少了模型的参数数量.使用 RISC-V 处理器结合加速器的架构来实现深度可分离卷积神经网络的部署和加速,这是一种有前景的资源受限环境下的神经网络部署方案.然而,目前对于这一领域的研究相对有限,相关工作主要存在以下三个方面的缺点:

(1) 现有的加速器<sup>[5-6]</sup>在计算深度可分离卷积时,通常会为点卷积和深度卷积分别开发独立的计算模块,未能实现这两种算子的模块共享,导致加速器对硬件资源的过度消耗.

(2) 大多数的现有加速器<sup>[7-9]</sup>在进行深度卷积计算时采用传统的卷积核与输入数据直接相乘相加的方式,未能结合相关算法进行优化,因此加速器的计算效率有待提升.

(3) 在 RISC-V 处理器调用加速器时,有部分工作<sup>[10]</sup>没有对相关指令或函数进行封装,导致调用过程相对繁琐.针对不同的网络模型,需要手动编写特定的代码来进行模型部署,这影响了系统的易用性.

为解决以上问题,本文设计了一个深度可分离卷积神经网络加速器.该加速器能够分别加速深度可分离卷积的两个算子,且具有较低的硬件资源消耗及功耗.在 RISC-V 处理器端,拓展了自定义指令,并对 TensorFlow Lite 库的卷积函数进行修改,以实现加速器的灵活配置和自动调用.总的来说,本文的主要贡献包括 3 个方面:

(1) 基于 Winograd 快速卷积算法和模块重用,本文设计了一款高效的深度可分离卷积硬件加速器.在深度卷积计算流程中,采用了 Winograd 算法,减少了硬件乘法器的使用量,同时通过  $2 \times 2$  数

据块组合拼接成  $4 \times 4$  数据片的方式降低了传输数据的冗余,点卷积计算与深度卷积共享部分硬件结构,提高了硬件资源的利用率。

(2) 利用 RISC-V 的指令拓展功能,本文设计了一套深度可分离卷积指令集,涵盖了处理器调用加速器所需的各类操作,包括数据输入、结果取回、开始计算等。通过修改 TensorFlow Lite 库的卷积函数,实现了处理器对加速器的灵活自动调用,方便了深度可分离卷积神经网络模型在边缘端的部署,加快了推理速度。

(3) 基于硬件设计工具,在现场可编程门阵列(Field Programmable Gate Array, FPGA)硬件上实现了加速器设计,部署了量化后的 MobilenetV1 网络。实验结果表明该加速器具有极低的功耗(仅为 0.123 W)和显著的能耗比(达 10.05 GOPS/W)。相比于 RISC-V 处理器,加速器可以将点卷积加速 104.40 倍,将深度卷积加速 123.63 倍。

本文第 2 节介绍 RISC-V 架构、Winograd 快速卷积算法和深度可分离卷积的相关技术及研究;第 3 节和第 4 节分别从硬件结构设计和软件指令设计的角度,介绍本文提出的基于 RISC-V 的深度可分离卷积加速器的实现细节;第 5 节为实验部分,将本文加速器在实际硬件上进行实现,评估加速器的资源使用情况和加速效果,并与已有加速器进行对比;第 6 节对全文进行总结。

## 2 相关工作

### 2.1 基于 RISC-V 的神经网络加速器

复杂指令集处理器(Complex Instruction Set Computer, CISC)和精简指令集处理器(Reduced Instruction Set Computer, RISC)是两种不同的计算机体系结构。CISC 通常包含大量指令,其中一条指令可以执行多个低级操作,甚至包括复杂的内存访问和控制流程。x86 架构是 CISC 的代表,其指令集非常庞大,支持复杂的操作,适用于通用计算任务。RISC 设计目标是通过减少指令集的复杂性来提高计算机性能,采用更简单、更基本的指令,每条指令执行的操作较为简单,执行时间较短。RISC-V 是最新一代 RISC,完全开源,可以免费使用。

RISC-V 的指令集是模块化和可扩展的。它分为基本指令集(RV32I、RV64I 等)和各种标准和自定义的扩展,如乘法扩展(M)、原子操作扩展(A)、

浮点数扩展(F)等。这种模块化的设计允许系统设计者选择适合其需求的指令集子集,避免了使用不必要功能的开销,提高了灵活性。在传统的 GPU、ARM 等平台难以满足高能效的情况下,RISC-V 作为一种开放的指令集架构,为研究人员提供了一个灵活且可定制的平台,用于实现高效的神经网络加速。RISC-V 在神经网络加速领域具有极大潜力,特别是在对低功耗、灵活性和定制性有要求的场景中。

针对 RISC-V 平台设计专用硬件加速器是提高神经网络性能的一种主要方法,研究人员通过定制指令集和硬件架构,实现了专门针对神经网络计算的高效加速器。Hoyer 等人<sup>[11]</sup>针对心房颤动场景开发了专门的硬件加速器。首先,他们对用于心房颤动检测的人工神经网络进行了优化,以满足在 RISC-V 控制器上进行推理的最低要求。随后,对一个 32 位浮点型神经网络进行了分析,并为了减小芯片面积,将其量化为 8 位定点数据类型。基于这一数据类型,研究团队开发了专门的加速器。该加速器包括单指令多数据硬件以及用于激活函数的加速器,实现了对设计网络的高效推理。然而,该加速器的结构是固定的,只能支持单一的网络结构。

Askarihemmat 等人<sup>[12]</sup>提出了一种深度神经网络(Deep Neural Network, DNN)加速器,该加速器采用可在位级别配置的专用处理元件,用于进行任意精度的推理。该 DNN 加速器包含 8 个处理元件,由 RISC-V 处理器进行控制。该加速器在无需硬件重新配置的情况下提供了运行时可编程性,并且无论目标 FPGA 大小如何,都可以使用多个量化级别来加速 DNN。然而,该加速器中的矩阵向量单元消耗了较多的硬件资源和能耗,其功耗达到了 21.066 W,因此在一些低功耗的场景下可能不太适用。

Liang 等人<sup>[13]</sup>提出了一种新的张量计算指令集扩展 TCX,它使用变长张量扩展了 RISC-V 指令集。该扩展具有多维寄存器文件、维度寄存器和完全通用的张量指令。作者实现了一个支持张量扩展的张量加速器,采用无序的 RISC 微架构。TCX 能够无缝集成到 RISC-V 指令集中,并为可扩展的硬件加速器实现提供软件兼容性。TCX 支持不同类型的张量计算,如标准卷积、深度卷积、跨步卷积、点卷积、全连接和池化。然而,该加速器在计算卷积时,并未结合相关卷积算法进行优化。与传统的卷积核与输入数据直接相乘相加的计算方式相比,Winograd 快速卷积算法可以减少乘法次数,更加节约硬件资源。

## 2.2 Winograd 快速卷积算法

Winograd 算法是由科学家 Shmuel Winograd 提出的一种最小滤波算法, 使用此算法可降低滤波计算过程中所需乘法次数。Winograd 算法还可以应用于一维和二维卷积, 适用于不同大小的卷积核中。根据该算法, 由长度为  $r$  的一维卷积核卷积计算生成长度为  $m$  的输出, 表示为  $F(m, r)$ , 需要的乘法数量可以不是  $m \times r$ , 而是最少降低到  $\mu(F(m, r)) = m + r - 1$ 。以  $F(2, 3)$  为例, 使用 Winograd 计算所需的乘法数量为  $\mu(F(2, 3)) = 2 + 3 - 1 = 4$ , 而非  $2 \times 3 = 6$ , 乘法次数从 6 次降低到了 4 次。二维的 Winograd 算法公式为

$$\mathbf{Y} = \mathbf{A}^T [[\mathbf{G}\mathbf{g}\mathbf{G}^T] \odot [\mathbf{B}^T \mathbf{d}\mathbf{B}]] \mathbf{A} \quad (1)$$

其中,  $\mathbf{g}$  代表权重数据矩阵,  $\mathbf{d}$  代表输入数据矩阵,  $\mathbf{Y}$  为输出数据矩阵,  $\mathbf{A}, \mathbf{G}, \mathbf{B}$  均为已知参数用于变换的矩阵, 角标 T 表示转置,  $\odot$  表示矩阵对应位置相乘的操作。

通过 Winograd 算法, 使用  $r \times r$  大小的卷积核进行卷积计算, 生成  $m \times m$  大小的输出。这个过程可以表示为  $F(m \times m, r \times r)$ 。Winograd 算法所需的最小乘法数为  $(m + r - 1)^2$ , 而原始卷积算法需要  $m \times m \times r \times r$  次乘法。例如, 对于  $F(2 \times 2, 3 \times 3)$ , 最小乘法次数为 16, 相比原始卷积所需的 36, 减少了 5/9 的乘法次数。

常规的 Winograd 算法支持  $r=2$  和  $r=3$  的二维卷积算子, 且切片大小一般不超过 6。  $F(2 \times 2, 3 \times 3)$  是 Winograd 适用的理想大小, 过大的切片尺寸或卷积核尺寸会带来较大的损失精度<sup>[14]</sup>。Winograd 算法在处理小型卷积核时, 可以在保持高精度的同时实现显著的性能提升。因此, 使用 Winograd 算法是加速卷积神经网络的重要优化方法之一。2016 年, Lavin 等人<sup>[15]</sup>首次将 Winograd 算法应用到卷积神经网络 (Convolutional Neural Network, CNN) 中, 利用该算法减少了卷积的乘法次数, 从而提升了卷积计算速度。

FPGA 平台因其灵活的结构特性更容易实现并发挥 Winograd 算法的优势。在计算资源和功耗有限的情况下, Winograd 卷积可以显著减少乘法运算的数量, 带来巨大的收益。Mo 等人<sup>[16]</sup>在 FPGA 上成功利用 Winograd 算法加速深度卷积, 并设计了点卷积和全连接的计算单元。然而, 所有计算模块按照网络模型结构的顺序以流水线方式连接, 每个计算模块都独立分配了硬件资源, 导致大量硬件资源

的消耗。DiCecco 等人<sup>[17]</sup>在 CPU-FPGA 异构平台上修改了 Caffe 框架, 实现了基于 FPGA 的 Winograd 卷积引擎, 可以与在 CPU 上运行的其他层一起工作。Xiao 等人<sup>[18]</sup>开发了一个自动化工具链, 简化了使用 Vivado HLS 将 Caffe 模型映射到 FPGA 比特流的过程, 并使用动态规划来选择是否使用 Winograd 算法。

在边缘端平台上, Wu 等人<sup>[19]</sup>、Zhang 等人<sup>[20]</sup>提出了一种高效的推理方法, 名为 DREW。该方法将深度重用技术与 Winograd 算法相结合, 以进一步加速 Winograd 卷积。平均加速幅度为 2.06 倍, 且精度损失极小 ( $< 0.4\%$ )。这一方法创新地通过结合重用技术与 Winograd 算法, 丰富了对 Winograd 算法的理解和应用, 显著提升了 CNN 推理的效率。

除了在上述平台的应用, 一些研究工作还将 Winograd 算法应用于其他非传统平台。Lin 等人<sup>[21]</sup>在 ReRAM 上成功地应用了 Winograd 算法, 通过采用切片的策略, 有效提升了数据的重复利用率。Winograd 算法也与随机计算、近似计算进行了结合<sup>[22-23]</sup>。Ghaffar 等人<sup>[24]</sup>则在 DRAM 架构上实现了神经网络的量化卷积。Chen 等人<sup>[25]</sup>在向量数字信号处理器上实现了三维 Winograd 卷积。在 RISC-V 平台中, Wang 等人<sup>[26]</sup>构建了一个自定义指令, 该指令可以执行  $F(2 \times 2, 3 \times 3)$  的 Winograd 卷积。然而, 这项研究还处于初步阶段, 尚未对完整的神经网络进行部署。

## 2.3 深度可分离卷积

深度可分离卷积与标准卷积不同, 它将卷积过程分为深度卷积和点卷积两个步骤, 既能有效提取图像数据特征, 又能降低卷积的参数数量和运算量。各类卷积的计算过程如图 1 所示: 标准卷积中, 卷积核通道数量与输入数据一致, 卷积核数量等于输出通道数量, 需要将每个通道相同二维坐标的数据乘积累加, 得到一个输出通道的结果。而在深度可分离卷积中, 输入数据先进行深度卷积, 每个卷积核只有一个通道, 卷积核数量和输出通道数量与输入通道数相同, 各通道只需进行乘法操作, 无需累加求和; 然后使用点卷积进行线性连接, 点卷积相当于卷积核大小固定为  $1 \times 1$  的标准卷积。

深度可分离卷积能够极大地降低模型参数数量和计算量。假设输入数据的 CHW 大小为  $C_{in} \times N \times N$ , 标准卷积的卷积核大小为  $C_{in} \times K \times K$ , 输出通道数为  $C_{out}$ , 并且步长为 1 时, 则标准卷积的参数数量和计

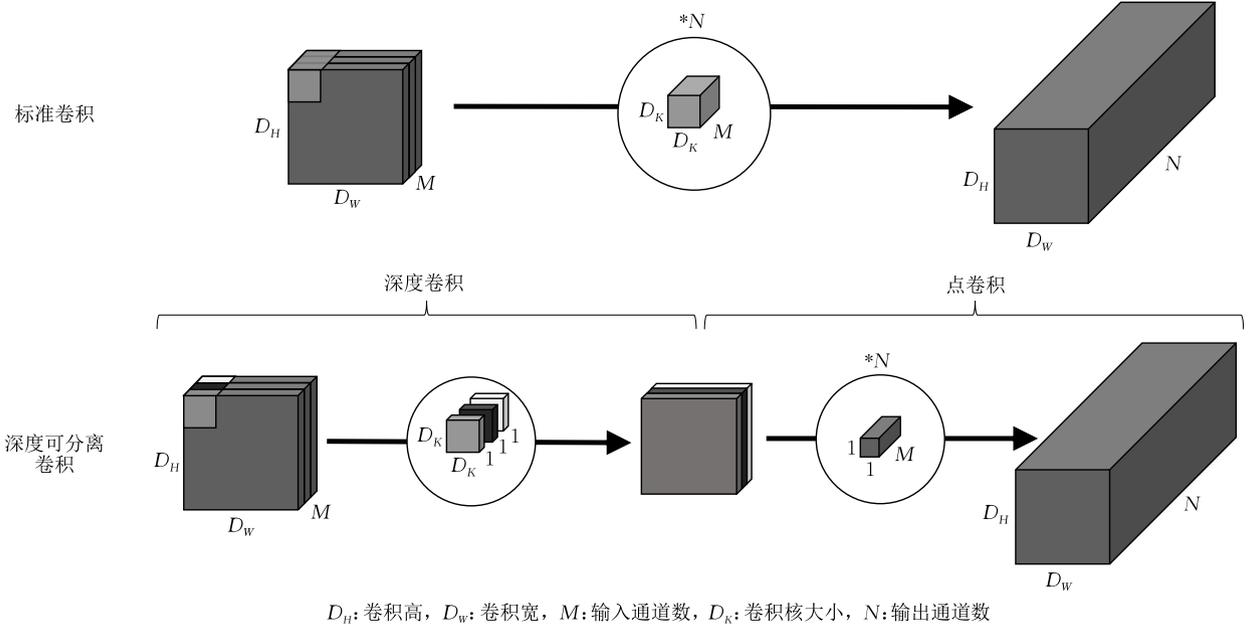


图 1 各类卷积计算过程

算量分别为

$$P_{SC} = C_{in} \times K \times K \times C_{out} \quad (2)$$

$$C_{SC} = C_{in} \times N \times N \times K \times K \times C_{out} \quad (3)$$

而生成相同大小输出的深度可分离卷积所需的参数量和计算量为

$$P_{DSC} = C_{in} \times K \times K + C_{in} \times C_{out} \quad (4)$$

$$C_{DSC} = C_{in} \times N \times N \times K \times K + C_{in} \times N \times N \times C_{out} \quad (5)$$

标准卷积和深度可分离卷积的参数量和计算量比值可以计算如下:

$$R_p = \frac{P_{DSC}}{P_{SC}} = \frac{1}{C_{out}} + \frac{1}{K^2} \quad (6)$$

$$R_c = \frac{C_{DSC}}{C_{SC}} = \frac{1}{C_{out}} + \frac{1}{K^2} \quad (7)$$

深度可分离卷积最早公开于 Sifre 等人<sup>[27]</sup> 研究刚性运动的博士学位论文中,之后谷歌使用深度可分离卷积并基于 Inception V3 改进得到了 Xception<sup>[28]</sup>,接着谷歌又设计了轻量级神经网络 MobileNet<sup>[29]</sup>,深度可分离卷积的出色性能在两个模型中得到体现,MobileNet 与 VGG16<sup>[30]</sup> 的准确率基本相同,计算量和参数量却远低于后者。

综上所述,RISC-V 平台中采用 Winograd 算法对深度可分离卷积进行加速是一种软硬件友好的神经网络边缘部署方案。这种方法可以减少硬件乘法单元的使用和能源消耗,同时利用 RISC-V 的指令扩展功能,软件端可以灵活便捷地调用加速器。然而,从以上相关研究中可以看出,这一方案仍有待探索和完善。本文旨在通过设计一个基于 RISC-V 的

深度可分离卷积神经网络加速器,填补现有研究的不足,为该方案的深入研究提供参考。引入该加速器,期望在性能和效率方面取得改进。这不仅有助于进一步推动神经网络边缘部署的发展,也为 RISC-V 平台上的深度学习应用提供了可行的选择。

### 3 深度可分离卷积神经网络加速器硬件设计

本小节对深度可分离卷积神经网络加速器的硬件结构设计进行介绍,首先介绍了硬件系统的整体架构,其包括 RISC-V 处理器、加速器和一些其他外设,接着介绍了加速器内的各个模块,并分析了优化的深度卷积存取方式所减少的传输数据量,最后对点卷积、深度卷积的数据流进行说明。

#### 3.1 系统架构

本文的硬件系统主要由 RISC-V 处理器和深度可分离卷积加速器组成。RISC-V 处理器与加速器、DRAM 内存、UART 串口等相连,如图 2 所示。RISC-V 处理器通过 USB 连接的 UART 串口与终端相连,以进行调试。程序文件、神经网络模型参数、输入图像数据等也通过 USB 的 UART 串口传输并存储到 DRAM 内存中。

RISC-V 处理器利用 RISC-V 架构的指令扩展功能,添加了数条 R 型用户自定义扩展指令,形成卷积指令集,以便调用加速器。加速器与 RISC-V 处

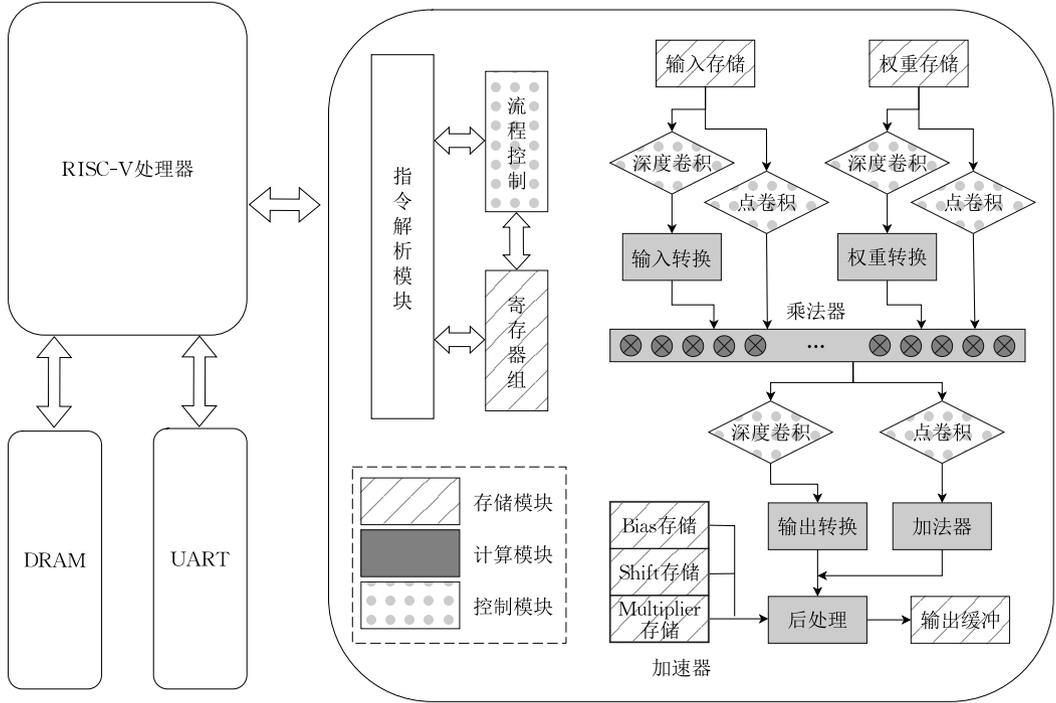


图 2 系统架构

理器相连接,但未直接连通 DRAM 内存.数据及控制信号通过卷积指令集在 RISC-V 处理器和加速器之间交互,并将计算结果返回给 RISC-V 处理器.

RISC-V 处理器负责程序的执行,共有五级流水线:取指、译码、执行、访存、写回.神经网络模型的加载、图像数据的输入、调试信息的统计和输出、程序的流程控制等工作,均由 RISC-V 处理器完成.在神经网络推理过程中,对加速器支持的点卷积和深度卷积算子,RISC-V 处理器通过扩展的卷积指令集调用加速器计算;对于其他算子,则由 RISC-V 处理器自身计算.本文设计的深度可分离卷积加速器支持加速计算卷积核宽高为  $3 \times 3$ 、步长为 1 的深度卷积和点卷积.

深度可分离卷积加速器内部包括指令解析模块、存储模块、计算模块和控制模块.对于深度卷积和点卷积这两种组成深度可分离卷积的算子,若单独开发硬件模块将消耗较多硬件资源.考虑到这两种算子都需要使用存储模块、乘法器单元和后处理单元,因此本文在两种算子的数据流中重用了这些模块.通过控制模块使各单元在不同算子计算模式下以不同方式工作,实现不同的算子加速功能,从而减少所需的硬件资源.

### 3.2 加速器接口与指令解析模块

RISC-V 处理器支持加速器拓展功能,具有一个加速器接口,设计好的加速器连接到该接口上.

具体来说,RISC-V 处理器与加速器通过多个信号相连,加速器的具体信号内容如表 1 所示.RISC-V 处理器需要执行一个拓展的自定义指令时,会将 Plugin\_bus\_cmd\_valid 信号置为 1,表示当前接口上的拓展指令内容有效,并将指令的 10 位指令识别码 Plugin\_bus\_cmd\_payload\_function\_id、32 位输入数据 Plugin\_bus\_cmd\_payload\_inputs\_0 和 32 位输入数据 Plugin\_bus\_cmd\_payload\_inputs\_1 传输到接口上.同时,若 Plugin\_bus\_cmd\_ready 为 1,表明加速器已准备好接收并处理新的拓展指令,将提取接口传入的指令数据,然后进行相关处理.等待加速器处理完成后,加速器会返回一个 32 位的输出数据,并将 Plugin\_bus\_rsp\_valid 信号置为 1.同时,当 RISC-V 准备好接收返回的输入数据时,将 Plugin\_bus\_rsp\_ready 信号置为 1,取接口上的返回数据,一条拓展指令对加速器的调用执行完毕.

表 1 加速器接口信号

信号名称	位宽	方向
Plugin_bus_cmd_valid	1	输入
Plugin_bus_cmd_ready	1	输出
Plugin_bus_cmd_payload_function_id	10	输入
Plugin_bus_cmd_payload_inputs_0	32	输入
Plugin_bus_cmd_payload_inputs_1	32	输入
Plugin_bus_rsp_valid	1	输出
Plugin_bus_rsp_ready	1	输入
Plugin_bus_rsp_payload_outputs_0	32	输出

RISC-V 处理器在执行程序的过程中,任何操作码字段指定为 CUSTOM0 的指令都将被识别为拓展指令,并被发送到加速器接口。RISC-V 处理器识别到拓展的自定义指令后,会调用加速器进行处理,通过多种功能的自定义指令组成的卷积指令集实现加速计算的功能。指令中的两个 32 位数据和一个 10 位的指令识别码将被发送给加速器。

加速器中的指令解析模块负责解析传入的数据和指令。不同的指令会根据指令识别码在加速器中分配不同编号的寄存器,用于缓存该指令携带的数据,待需要返回给 RISC-V 处理器的 32 位数据准备就绪后,也会存储在该寄存器中。需要返回的数据准备就绪后,从寄存器组传递给指令解析模块,最终返回给 RISC-V 处理器。由于不同指令需要的返回数据准备时间不同,因此每条指令执行所需的时钟周期数也会有所差异。

### 3.3 控制模块

控制模块包括流程控制单元和数据流流向选择单元。当 RISC-V 发出开始计算的指令后,流程控制单元会检测输入数据是否准备就绪,以及输出缓冲区是否还有剩余空间。如果这两个条件都满足,流程控制单元将发出开始信号,启动存储模块和计算模块以开始计算。

加速器可以分别对深度卷积和点卷积两种算子进行加速。这两种算子在数据流中共用输入存储和权重存储单元。然而,由于这两种算子的计算模式中,输入数据与权重数据的取出方式不同,因此需要由流程控制单元发出信号,指明当前的计算模式,以确保存储单元在正确的模式下工作。此外,在这两种算子的计算模式下,各个单元的数据流向也有所不同。数据流向选择单元会根据当前的计算模式,将数据流向不同的位置。

根据卷积输入数据和权重数据的尺寸不同,存储模块需要以不同的频率输出下一组数据,相应的控制信号由流程控制单元计算并产生。在点卷积计算模式下,当处于相同二维坐标的所有通道数据累加完成时,流程控制单元会发出清零信号,将加法器置零。可以说,控制模块协调了加速器内部各个单元,确保它们协同工作,以保持正常运行。

### 3.4 存储模块

存储模块负责存储指令携带的一般数据、输入数据、权重数据、Bias、Shift、Multiplier 及输出数据。对于指令携带的一般数据,主要用于说明卷积输入矩阵的大小、当前计算模式等信息,采用 32 位宽的寄存器进行存储。其余与卷积相关的数据则使用 32

位宽的块随机存取存储器(Block Random Access Memory, BRAM)进行存储。本文中将要部署的网络参数量化为 8 位宽,因此存储器的每个存储单元可以保存 4 个 8 位数据。

输入存储被分为四组 BRAM,每组 BRAM 的位宽为 32 比特。在点卷积计算模式下,这些 BRAM 以乒乓缓冲的方式工作。有两组 BRAM 在存储完数据后进行计算,每个时钟周期可以并行提供 64 位数据,而另外两组 BRAM 则可以接收新的数据。这种设计允许数据存储和计算同时进行,从而提高了计算效率。

在深度卷积计算模式下,加速器采用了  $F(2 \times 2, 3 \times 3)$  的 Winograd 算法进行加速计算。首先,需要对输入特征矩阵进行切片处理,将其切割成  $4 \times 4$  大小的数据片,即 Winograd 算法公式中的输入数据矩阵  $d$ ,才能参与 Winograd 算法的计算。如果将计算过程中所需的数据片完整地由 RISC-V 处理器传输到加速器存储中,由于数据片之间存在重叠区域,这将导致大量的数据冗余。因此,本文结合软硬件层面设计了一套数据片的存取与组装流程,在消除冗余的同时,增大了带宽,使得在一个时钟周期内可以取出一个数据片的数据。

深度卷积计算模式下的输入数据片存取方式如图 3 所示。输入存储被分为四组,特征输入数据矩阵的每个坐标数据大小为 8 比特,四个相邻的图像数据可以拼接成一个 32 比特的  $2 \times 2$  数据块,即如图 3 中长度为  $2 \times 2$  的正方形小方框所示。一条拓展指令可以从 RISC-V 处理器将一个数据块传输到加速器中,而一个  $2 \times 2$  的数据块正好可以存储在一组存储器的一行中。数据片的切片顺序相当于在特征输入矩阵内使用  $4 \times 4$  大小的方框,以步长为 2 的方式在左右和上下方向上滑动,方框滑动到的地方就是要提取的数据片。例如,在图 3 的输入数据矩阵中,0、1、2、3 号数据块组成第一个数据片,1、4、3、6 号数据块组成第二个数据片,4、5、6、7 号数据块组成第三个数据片。

本文的加速器通过将四个  $2 \times 2$  的数据块拼接成一个  $4 \times 4$  的数据片来获取数据片。输入数据矩阵中的数据块按照行、列和通道的优先顺序传递给加速器,例如图 3 中的数据块顺序为 0、1、4、5、 $\dots$ 、2、3、6、7、 $\dots$  的顺序。对于按顺序到达的输入数据,四组存储器依次轮流存储这些数据块,这样连续的两个数据块将被存储在相邻的两组存储器中。例如,图 3 中的数据块 0、1、4、5 分别存储在第 0、1、2、3 组输入存储器中。

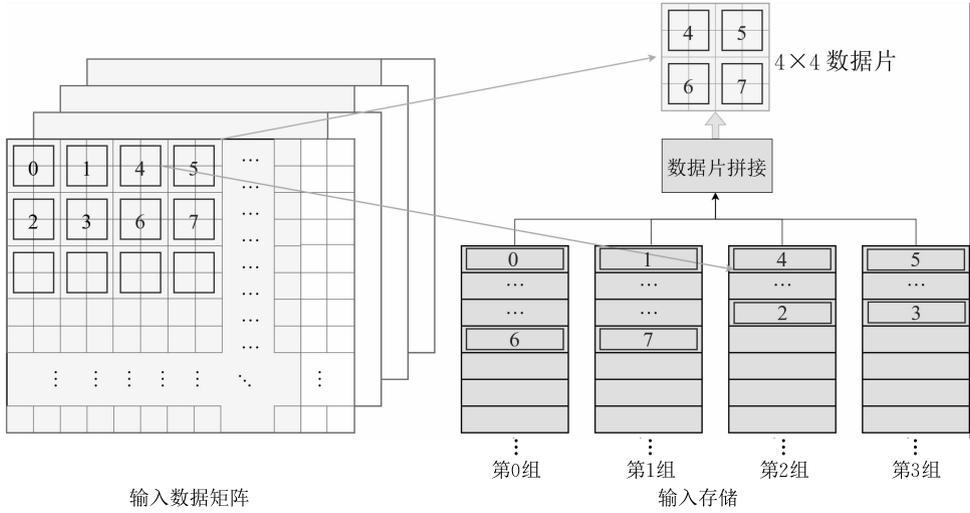


图 3 深度卷积输入数据存取方式

每个  $4 \times 4$  数据片由 4 个  $2 \times 2$  数据块组成, 四个相邻的数据块可以组合成一个数据片, 如图 3 中, 长为宽为  $4 \times 4$  的方框所示——编号为 4、5、6、7 的四个数据块可构成一个数据片。为了实现每个数据片所需要的四个数据块分别存储在四组不同的输入存储中, 并提高数据块的提取带宽, 需要在每行数据块传入后, 额外传入空白的数据块以占位存储器。这样可以在一个时钟周期内同时从四组输入存储中提取数据, 拼凑成一个完整的数据片。例如, 在图 3 中, 编号为 4、5、6、7 的四个数据块分别存储在第 2、3、0、1 这四组不同的输入存储中, 可以在一个时钟周期内同时从四组存储中提取出 4、5、6、7 这四个数据块, 通过数据片拼接步骤得到一个完整的数据片。

以上数据片拼接方法有效避免了数据传输过程中的数据冗余。设图像输入矩阵每一行的数据块数量为  $N_w$ , 每一列的数据块数量为  $N_h$ , 每个数据块的内存大小为  $D$ , 需要传入的空白数据块数量为  $N_{\text{null}}$ , 则有:

$$N_{\text{null}} = \begin{cases} 2, & \text{若 } N_w \% 4 = 0 \\ 1, & \text{若 } N_w \% 4 = 1 \\ 0, & \text{若 } N_w \% 4 = 2 \\ 3, & \text{若 } N_w \% 4 = 3 \end{cases} \quad (8)$$

一个输入通道内用于 Winograd 算法所需产生的数据量大小为

$$D_a = 4D(N_w - 1)(N_h - 1) \quad (9)$$

而加入了空白数据块后, 无需将每个数据片都单独传给加速器, 所需传输的数据大小为

$$D_t = D(N_w + N_{\text{null}})N_h \quad (10)$$

因此, 共减少传输的数据量为

$$D_m = D_a - D_t \approx 3DN_w N_h \quad (11)$$

输入存储内部计算出组成下一个数据片的四个数据块各自的地址后, 从四个 BRAM 中取出并将其拼接成一个完整的数据片, 每个时钟向下一单元传送一个数据片的数据。

### 3.5 计算模块

计算模块包含三个矩阵转换、乘法器、加法器和后处理单元。加速器内各个单元相互连接, 以数据流水线的方式计算卷积。

乘法器单元包含 16 个 12 位乘法器, 最多能够并行进行 16 组乘法运算。在计算深度卷积时, 流程选择将计算结果送入输出转换模块; 而在计算点卷积时, 流程则选择将计算结果送入加法器。如图 4 所示, 加法器单元用于点卷积的计算过程, 采用树形结构, 每次可以计算 8 组 24 位操作数的相加结果, 输出数据为每次的 8 组加法结果与上一时钟周期的输出数据的累加结果。直到当前点卷积输出像素的所有维度数据都累加完毕, 此时流程控制单元发出清零信号, 将加法器的输出置为零。后处理单元从相关存储单元中提取 Bias、Shift 和 Multiplier 等数据, 对得到的加法器结果进行偏移量叠加和量化处理。

在计算点卷积层时, 每次计算得到一个坐标的输出数据, 输出数据按照通道、行、列的顺序进行计算和输出。在加速器的点卷积计算模式下, 一个坐标的输出结果的计算数据流如下:

输入存储器在每个时钟周期提供来自 8 个通道的 8 个 8 位输入数据, 权重存储器同步地提供对应的 8 个 8 位权重数据。这些数据被传输到乘法器中; 经过乘法计算后, 得到 8 组乘积。加法器对这 8 组乘积进行求和, 求和结果与初始输出值相加, 加法器的

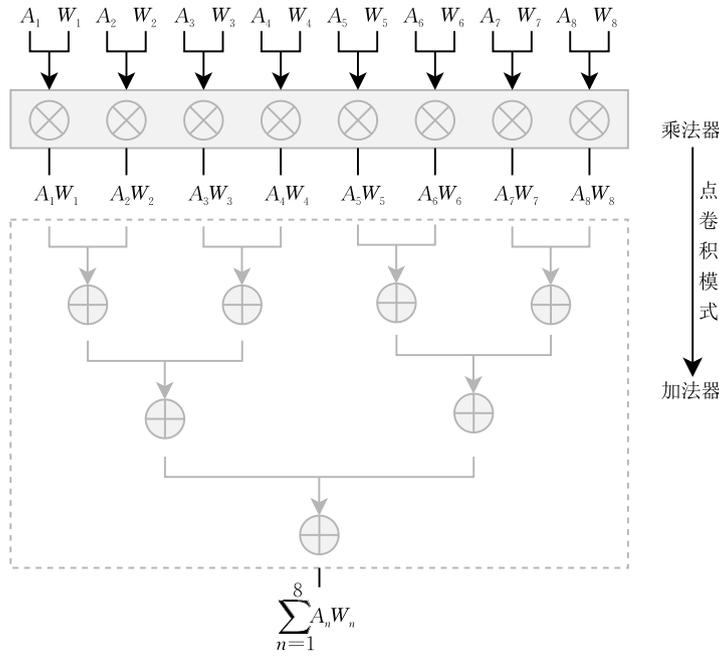


图 4 加法器结构

初始输出为 0. 重复上述步骤, 直到所有通道的数据都完成累加. 将累加结果送入后处理单元, 随后将加法器的输出清零. 经过后处理的累加结果被送入输出缓冲器, 等待结果取回指令以提取计算结果.

计算深度卷积层时, 采用  $F(2 \times 2, 3 \times 3)$  的 Winograd 算法进行计算, 计算过程所需固定参数为

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \\ 0 & -1 \end{bmatrix} \quad (12)$$

每次计算可以得到  $2 \times 2$  大小的输出数据块, 输出数据块按照行、列和通道的顺序进行计算和输出. 在深度卷积计算模式下, 加速器的一个输出数据块的计算数据流如下:

输入存储每个时钟周期提供一个  $4 \times 4$  大小的输入数据片, 记为  $\mathbf{d}$ , 权重存储同步地提供对应通道的  $3 \times 3$  大小的权重数据, 记为  $\mathbf{g}$ ; 接着  $\mathbf{d}$  被送入输入转换, 进行  $\mathbf{B}^T \mathbf{d} \mathbf{B}$  的矩阵转换计算, 结果记为  $\mathbf{U}$ , 同步地,  $\mathbf{g}$  被送入权重转换, 进行  $\mathbf{G} \mathbf{g} \mathbf{G}^T$  的矩阵转换计算, 结果记为  $\mathbf{V}$ ;  $\mathbf{U}$ 、 $\mathbf{V}$  计算完成后, 送入到乘法器中对相同位置进行乘法计算, 得到  $\mathbf{U} \odot \mathbf{V}$ ; 将  $\mathbf{U} \odot \mathbf{V}$  送入输出转换后计算得到  $\mathbf{A}^T [\mathbf{U} \odot \mathbf{V}] \mathbf{A}$ , 即卷积结果  $\mathbf{Y}$ ;  $\mathbf{Y}$  进行后处理之后送入输出缓冲器, 等待结果取

回指令将计算结果取回.

## 4 基于 RISC-V 的卷积指令集软件设计

本小节介绍了 RISC-V 处理器端的相关软件设计. 首先, 利用 RISC-V 的扩展指令功能, 拓展了 17 条 R 型 RISC-V 指令, 形成了一套卷积指令集, 以便灵活调用加速器. 接着, 利用该卷积指令集修改了 TensorFlow Lite 库中的卷积函数, 使神经网络推理时能够自动调用加速器进行计算.

### 4.1 卷积指令集设计

TensorFlow Lite 是一个移动端神经网络库, 可用于在移动设备、微控制器和其他边缘设备上部署模型. 本文在 RISC-V 处理器上基于 TensorFlow Lite 库进行神经网络的部署. 为了实现处理器对深度可分离卷积加速器的调用, 需要在 RISC-V 处理器端拓展相关用户自定义指令.

指令扩展是 RISC-V 指令集的重要特性, 允许根据应用需求和特定领域的要求, 向基本指令集添加额外的指令和功能. 指令扩展通过引入新的指令来丰富指令集, 以支持不同的应用场景. R 型指令的一般格式如图 5 所示. R 型指令用于 RISC-V 处理器中寄存器与寄存器之间的数据操作, 长度为 32 位, 包含多个字段. 每条指令接收两个 32 位的数据输入, 分别来自寄存器 rs1 和 rs2, 返回的 32 位的结果数据存储在寄存器 rd 中, opcode 字段指明指令类型, funct3 和 funct7 字段用于细分和标识

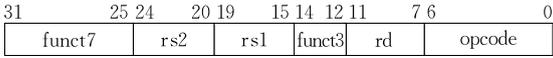


图 5 R 型指令格式

不同的指令。

拓展后的指令在代码中使用 asm 内联汇编的方式进行使用,将内联汇编的不同指令代码段使用 C++ 的 #define 功能封装成函数的形式. 在程序需要的地方就可以以函数的形式,直接调用拓展后的指令。

表 2 卷积指令集的指令内容

指令名称	功能说明	funct7	传入参数	返回参数
SET_SWITCH	设置计算模式	0100011	点卷积:0;深度卷积:1	—
SET_INPUT_WIDTH	设置参数	0100101	输入数据矩阵宽度	—
SET_INPUT_DEPTH_WORDS	设置参数	0001010	输入数据矩阵通道数	—
SET_OUTPUT_DEPTH	设置参数	0001011	输出数据矩阵通道数	—
SET_INPUT_OFFSET	设置参数	0001100	输入数据的 offset 值	—
SET_OUTPUT_OFFSET	设置参数	0001101	输出数据的 offset 值	—
SET_ACTIVATION_MIN	设置参数	0001110	量化激活的最小值	—
SET_ACTIVATION_MAX	设置参数	0001111	量化激活的最大值	—
SET_OUTPUT_BATCH_SIZE	设置参数	0010100	点卷积每个计算批次的输出通道数	—
SET_NUM_TILE	设置参数	0100100	深度卷积每个计算批次的输入数据片数	—
STORE_OUTPUT_MULTIPLIER	存储数据	0010101	Multiplier 数据	—
STORE_OUTPUT_SHIFT	存储数据	0010110	Shift 数据	—
STORE_OUTPUT_BIAS	存储数据	0010111	Bias 数据	—
STORE_FILTER_VALUE	存储数据	0011000	权重数据	—
STORE_INPUT_VALUE	存储数据	0011001	输入数据	—
MACC_RUN	开始计算	0100001	—	—
GET_OUTPUT	取回数据	0100010	—	计算结果

## 4.2 深度可分离卷积函数

TensorFlow Lite 库中的算子可以替换为自定义的实现方式. 利用卷积指令集中的指令重新编写点卷积、深度卷积的函数代码. 当神经网络推理过程中用到这两种算子时,会自动使用改写后的算法,调用加速器进行计算,实现计算加速. 由于加速器的存储容量有限,可能无法一次存储全部的卷积输入数据或权重数据,所以计算时分批次进行. 点卷积按输出通道进行分批,每批次计算部分通道下的输出数据,每个批次计算时,将输入数据按行列坐标分组,每组输入数据为输入数据矩阵中相同行列坐标下所有通道的数据. 开始计算后,得到一组输出数据,每组输出数据为相同行列坐标下当前批次通道范围的数据. 点卷积计算的算法如算法 1 所示,步 5、6 用于触发乒乓缓冲,将计算与存储并行,从而加快计算速度。

### 算法 1. 点卷积计算算法.

输入: 输入数据矩阵 *input*, 权重数据矩阵 *filter*, 卷积参数 *params*、*Bias*、*Shift*、*Multiplier*

为实现不同的功能,本文共拓展了 17 条 R 型 RISC-V 指令,形成一套卷积指令集,可以对加速器进行灵活的调用,硬件端也已经根据自定义指令设置了相关指令解析模块. 本文设计的拓展指令 opcode 段均为 CUSTOM0,表明这是一条拓展指令,每条指令分配了不同大小的 funct7 字段进行区分. 指令使用 rs1 字段传入 1 个 32 位数据,funct3 和 rs2 字段保留备用. 封装后卷积指令集的具体指令内容如表 2 所示。

输出: 点卷积输出数据矩阵 *output*

1. 初始化: 设置计算模式为点卷积,设置卷积相关参数;
2. FOR EACH 计算批次
3. 计算输出数据通道的起始值 *base*、结束值 *end*;
4. 将 *base* 至 *end* 通道的 *Bias*、*Shift*、*Multiplier* 数据和权重数据存储到加速器中;
5. 将一组输入数据存储到加速器中;
6. 发出开始计算信号;
7. FOR EACH 分组输入数据 ∈ 剩余的输入数据
8. 将一组输入数据存储到加速器中;
9. 取回一组输出数据;
10. 发出开始计算信号;
11. END FOR
12. 取回一组输出数据;
13. END FOR

深度卷积按行进行分批,每批次计算部分行的输出数据. 每个批次计算时,将输入数据按通道分组,每组输入数据为输入数据矩阵中一个通道内当前批次行范围的数据. 开始计算后,得到一组输出数据,每组输出数据为一个通道内当前批次行范围的

数据. 深度卷积计算的算法如算法 2 所示.

### 算法 2. 深度卷积计算算法.

输入: 输入数据矩阵  $input$ , 权重数据矩阵  $filter$ , 卷积参数  $params$ 、 $Bias$ 、 $Shift$ 、 $Multiplier$

输出: 深度卷积输出数据矩阵  $output$

1. 初始化: 设置计算模式为深度卷积, 设置卷积相关参数;
2. 将  $Bias$ 、 $Shift$ 、 $Multiplier$  数据和权重数据存储到加速器中;
3. FOR EACH 计算批次
4.   FOR EACH 分组输入数据
5.     将一组输入数据存储到加速器中;
6.     发出开始计算信号;
7.     取回一组输出数据;
8.   END FOR
9. END FOR

算法 1 和算法 2 在具体实现时需要调用拓展的卷积指令集中的指令. 例如, 算法中的“将一组输入数据存储到加速器中”操作通过多次调用“STORE\_INPUT\_VALUE”指令进行实现, 算法中的“发出开始计算信号”操作通过调用“START\_RUN”指令进行实现, 算法中的“取回一组输出数据”操作通过多次调用“GET\_OUTPUT”指令进行实现.

## 5 系统实现及评估

本小节在实际的 FPGA 开发板上实现了所设计的软硬件相关内容, 统计了实现后的硬件资源使用情况, 并通过对深度可分离卷积神经网络进行部署测试, 对加速器性能评估, 得到加速比、峰值性能等实验结果.

### 5.1 FPGA 平台实现

本文设计的硬件系统实现平台为 Arty-A7, 如图 6 所示. 该开发板由 Digilent 公司推出, 搭载 Xilinx 公司的 XC7A100T 芯片, 设计用于教育、原型设计和嵌入式系统开发, 为用户提供一个灵活、功能强大的平台.

本文的 RISC-V 处理器使用 SpinalHDL 语言开发, 加速器则使用 Amaranth 硬件描述语言进行开发. 硬件比特流文件由 Vivado 2022 软件通过综合、布局布线以及编译得到, Vivado 运行环境为 Ubuntu18.04. 系统时钟频率设置为 75 MHz, 建立时间裕度为 0.151 ns. 表 3 展示了此系统的硬件资源使用情况, 整个系统的资源占用很低, 对芯片各类资源的使用率均在 20% 以下.

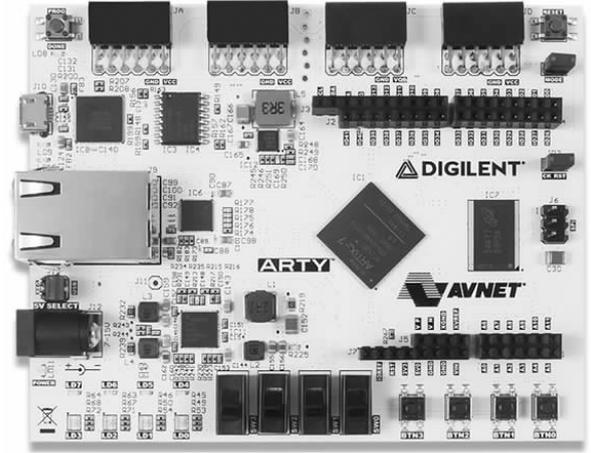


图 6 Arty-A7 FPGA 开发板

表 3 硬件资源使用

硬件资源	RISC-V 处理器	加速器	其余外设	总使用量	总使用率/%
LUT	3027	5434	2624.0	11085.0	17.84
Registers	1747	2449	2261.0	6457.0	5.09
BRAM	6	8	10.5	24.5	18.15
DSPs	4	32	0	36.0	15.00

本文中的 RISC-V 处理器采用五级流水线执行, 详细配置如表 4 所示. 该 RISC-V 处理器支持用户自定义拓展指令和连接加速器拓展功能. 在指令执行的指令解码阶段, RISC-V 处理器会识别当前指令的类型, 判断是否为用户拓展指令. 若是, 处理器将调用加速器来执行相应的运算, 同时向加速器传输相关指令代码. 加速器完成相应运算后, 在指令执行流水线的写回阶段, 将结果写回到相应的寄存器. 该 RISC-V 处理器支持性能计数器状态寄存器 (Performance Counter State Registers, PerfCSRs), 此寄存器用于收集和提供有关处理器性能的信息, 允许程序员和系统工具跟踪和分析程序的性能特征, 例如指令执行次数、缓存命中率等.

表 4 RISC-V 处理器配置

配置项	配置参数
核心数量	1
位宽	32
指令集	RV32IM
I-Cache 大小	8KB
D-Cache 大小	8KB
是否使用加速器	是
PerfCSRs 个数	8

RISC-V 处理器端使用 TensorFlow Lite 库部署神经网络. TensorFlow Lite 库下支持 MobilenetV1、MobilenetV2、Xception 等常见的使用了深度可分离卷积的神经网络. TensorFlow Lite 格式的网络模型

由 TensorFlow 模型转换而来。为测试所设计的深度可分离卷积加速器的加速效果,本文在服务器端使用 TensorFlow 训练神经网络,训练设备为 Intel(R) Xeon(R) CPU E5-2678。训练数据集为 CIFAR-10。CIFAR-10 数据集是一个接近普适物体的彩色图像数据集,含有 60 000 张大小为  $32 \times 32$  的图像。

网络模型训练完成后使用 TensorFlow Lite 的 converter 函数进行格式转换及模型量化,量化方法为训练后整数量化,量化精度为 uint8。转换时使用数据集中 1000 个输入数据来代表典型输入值,估算所有输入数据的动态范围。TensorFlow 的模型文件转换后得到量化的 TensorFlow Lite 模型文件。以 MobilenetV1 模型为例,使用 converter 量化并转换后,模型文件大小由 37.32 MB 降低到了 3.36 MB,大小仅为原来的 9%,同时准确率几乎没有下降。

## 5.2 加速器性能评估

TensorFlow Lite 库代码、相关测试代码使用 C++ 语言进行开发,开发工具为 Visual Studio Code,网络模型数据也转换为 C++ 数组形式进行

加载。使用 RISC-V GCC 工具链对 C++ 代码进行编译,测试代码通过 PerfCSRs 统计出不同函数执行所花费的时钟周期数。编译后的可执行文件通过串口连接个人电脑进行烧录,同时执行过程中的输出信息也通过串口传输显示在个人电脑端。经过测试,系统功能正常,可以使用加速器正确地对 MobilenetV1、MobilenetV2、Xception 等网络进行推理,加速器计算结果与单独使用 RISC-V 处理器计算结果一致,同时推理速度有显著的提升。

本文测试了所设计的深度可分离卷积加速器在不同输入和输出数据大小下的卷积层加速效果,测试结果如表 5 所示。从表 5 可以看出,不同的输出数据大小及输出数据大小情况下,加速器有着不同的加速比表现。其中当点卷积的输入尺寸为  $128 \times 16 \times 16$ ,输出尺寸为  $128 \times 16 \times 16$  时,点卷积的加速比可以达到 104.40 倍;当深度卷积的输入尺寸为  $8 \times 32 \times 32$ ,输出尺寸为  $8 \times 32 \times 32$  时,深度卷积加速比可以达到 123.63 倍。相比于 RISC-V 处理器,加速器的单层计算性能有着数十倍甚至上百倍的提升。

表 5 加速器端到端加速效果

网络层	卷积核大小/ ( $N \times C \times H \times W$ )	输入数据大小/ ( $C \times H \times W$ )	输出数据大小/ ( $C \times H \times W$ )	RISC-V 处理器计算时间/ ( $10^3 \times$ 时钟周期)	加速器计算时间/ ( $10^3 \times$ 时钟周期)	单层加速比
点卷积	$128 \times 128 \times 1 \times 1$	$128 \times 8 \times 8$	$128 \times 8 \times 8$	13 626	211	64.57×
点卷积	$64 \times 32 \times 1 \times 1$	$32 \times 32 \times 32$	$64 \times 32 \times 32$	32 691	343	95.31×
点卷积	$256 \times 128 \times 1 \times 1$	$128 \times 16 \times 16$	$256 \times 16 \times 16$	122 467	1173	104.40×
深度卷积	$32 \times 1 \times 3 \times 3$	$32 \times 16 \times 16$	$32 \times 16 \times 16$	3143	83	37.87×
深度卷积	$3 \times 1 \times 3 \times 3$	$3 \times 32 \times 32$	$3 \times 32 \times 32$	1930	21	91.90×
深度卷积	$8 \times 1 \times 3 \times 3$	$8 \times 32 \times 32$	$8 \times 32 \times 32$	8901	72	123.63×
深度卷积	$128 \times 1 \times 3 \times 3$	$128 \times 16 \times 16$	$128 \times 16 \times 16$	15 848	299	53.00×

TensorFlow Lite 库的输入数据的存储排列方式设置为 HWC 排列,此排列方式更利于计算点卷积。若要使用加速器对深度卷积进行计算,则需要先在 RISC-V 处理器端将输入数据进行预处理,转换为 CHW 排列,因此使用加速器后深度卷积的实际总推理时间为 RISC-V 处理器端的数据格式转换时间加上加速器的计算时间。此外,一般的深度可分离卷积神经网络中不仅包含加速器支持的深度可分离卷积算子,还包含了标准卷积、Relu、池化等加速器不支持的算子,此类算子仅能使用 RISC-V 处理器计算。因此,使用加速器的网络端到端加速比,并不如单独的点卷积或深度卷积层的表现出色。具体地,对使用了深度可分离卷积的网络模型的端到端测试结果如表 6 所示。表 6 中的第二列表示的是在仅使用 RISC-V 处理器推理时,那些可以被加速器计算支持的深度卷积层的总推理时间占整个

模型推理时间的比例。同理,表 6 中的第三列表示的是点卷积的占比。支持加速的深度卷积层和点卷积层占比越高,模型的可加速上限,即理论的端到端加速比越高。

表 6 加速器端到端加速效果

模型	支持加速的 深度卷积层 占比/%	支持加速的 点卷积层 占比/%	端到端加速比	
			理论上限	实际结果
MobilenetV1	10.0	83.7	15.87×	6.84×
MobilenetV2	11.4	62.0	3.76×	3.18×
Xception	18.5	48.0	2.99×	2.63×

以 MobilenetV1 网络为例,在使用 RISC-V 处理器推理时,支持加速的深度卷积层、点卷积层耗时占比分别为 10.0% 和 83.7%,也就是说,还有耗时共占 6.3% 的网络层算子不受加速器支持。因此,调用加速器后,若将支持加速的卷积层以极小的时间完成计算,可得到理论的端到端加速比上限为

15.87 倍. 实际推理测试时, 该网络的端到端加速比为 6.84 倍.

在点卷积计算模式下, 加速器使用了乒乓缓冲结构在两个缓冲区之间轮流传递数据, 使得在一个缓冲区执行计算的同时, 另一个缓冲区可以接收新的数据, 以此优化数据传输和处理效率, 提高系统的整体性能. 使用算法 1 所示算法即可利用加速器内的乒乓缓冲, 若没有 5、6 两步, 而是按照存储数据、开始计算、取回数据的顺序调用加速器, 则并没有工作在乒乓缓冲模式下. 图 7 对比了使用乒乓缓冲结构前后推理速度的变化, 该图中点卷积一、点卷积二的参数分别与表 5 中第二组、第三组点卷积一致. 根据表中数值, 使用乒乓缓冲结构后推理速度加快了约 25%.

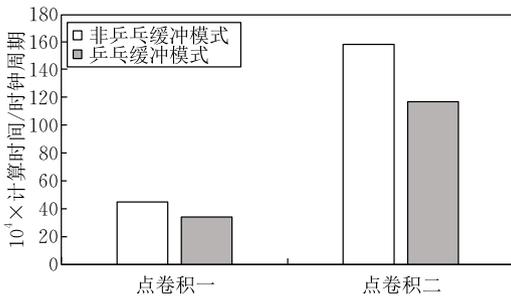


图 7 乒乓缓冲模式推理速度比较

### 5.3 相关对比

神经网络加速器在提供足够计算性能的同时最

好能够保持低功耗, 特别是在边缘计算和物联网设备上, 这类场景具有严格的资源限制, 包括功耗和计算能力. 表 7 展示了本文加速器在资源和性能功耗比方面与其他研究的对比. 表 7 中所列文献均设计了各自的卷积神经网络加速器, 并在 FPGA 上实现, 可以高效处理图像数据. 文献[31, 33]和本文加速器均由一颗 CPU 进行控制和调用, 不同的是, 文献[31, 33]中采用 ARM 架构, 而本文采用 RISC-V 架构, 得益于 RISC-V 的相关特性, 本文的系统更为灵活精简, 整体功耗也更低. 文献[34]通过软硬件结合设计轻量化推理架构, 实现了神经网络的推理加速. 尽管在性能功耗比方面高于本文的加速器, 但本文的 GOPs 是对方的 5.09 倍. 在许多实际应用场景中, 性能提升对于计算密集型任务尤为重要, 能够显著减少处理时间, 从而提高整体系统效率. 文献[35]通过优化 MobileNet 并行处理加速器来提高推理速度. 根据实验结果, 本文的方法在功耗上仅为其 1.09%. 在移动设备、远程传感器及其他依赖电池的设备中, 本文的方法能够显著减少能源消耗. 相比于表 7 中的文献[31-35], 本文的加速器拥有最低的功耗和硬件资源消耗. 尽管本文的加速器性能较弱, 但功耗仅为 0.123 W, 峰值性能可达 1.07 GOPs, 能耗比达到了 8.70 GOPs/W, 高于表 7 中的文献[31-33], 满足在资源不足且功耗受限的情况下部署神经网络的需求.

表 7 资源和性能对比

加速器	算法模型	平台	工作频率/ MHz	精度	LUT	BRAM	DSPs	性能/ GOPs	功耗/W	性能功耗比/ (GOPs · W <sup>-1</sup> )
文献[31]	MobileNetV1	ZYNQ-7020	130	INT16	37749	96.0	219	18.32	2.750	6.66
文献[32]	MobileNetV1	ZynpZCU102	270	INT8/16	17614	365.5	256	17.73	3.450	5.14
文献[33]	MobileNetV1	Zynq-7000	230	INT16	38228	132.5	144	63.51	12.730	4.99
文献[34]	MobileNetV1	XCK325T	200	INT8	173522	193.5	704	0.21	—	17.90
文献[35]	Optimized MobileNet	Zynq AXZU5EV	100	INT8	16474	901.0	666	—	6.510	—
本文加速器	MobileNetV1	Arty-A7	75	INT8	5434	8.0	32	1.07	0.123	8.70

文献[36]设计了一种基于 RISC-V 架构的卷积神经网络处理器, 该处理器可以利用 CNN 的并行性, 更加灵活. 作者设计了矢量存储指令、矢量加载指令、矢量加法指令和卷积运算指令来加速卷积过程的执行. 然而, 该工作仅在仿真平台进行了验证, 并没有实际上板烧录; 卷积计算并未结合相关算法进行优化, 消耗较多计算资源, 如 127 个 DSP 单元; 缺少针对模型部署方面的工作, 模型的部署方式较为繁琐. 相比之下, 本文的加速器通过了实际的硬件烧录, 充分验证了方案的可行性; 通过 Winograd 算法和模块重用方式, 在加速的同时降低了硬件资源

的消耗, 如加速器仅消耗 32 个 DSP 单元; 基于 TensorFlow Lite 框架和框架内算子函数的修改, 实现了网络模型文件的便捷部署, 且模型在推理时可自动调用加速器进行加速计算.

实验还将本文的深度可分离卷积加速器与 CPU、GPU 平台进行了对比, 对比结果如表 8 所示. 对比的 CPU 型号为 Intel(R) Xeon(R) Silver 4214R, GPU 型号为 NVIDIA GeForce RTX3080Ti. 使用 MobileNet 网络模型在不同平台上进行推理, 测试得到性能指标. CPU 用 Linux 的 s-tui 命令测得实时功耗, GPU 使用 nvidia-smi 命令测得实时功耗,

从而计算出性能功耗比. 可以看出 GPU 平台在面对小尺寸深度可分离卷积层时, 并不能完全发挥其性能优势. 相比于 CPU、GPU 平台, 本文的加速器在性能功耗比上具有巨大优势.

表 8 不同平台性能及功耗对比

平台	精度	性能/ GOPS	功耗/W	性能功耗比/ (GOPS · W <sup>-1</sup> )
CPU	FP32	1.31	101	0.013
GPU	FP32	3.92	113	0.035
本文加速器	INT8	1.07	0.123	8.70

## 6 结 论

本文设计了一个轻量级的基于 RISC-V 的深度可分离卷积神经网络加速器. 相比于标准卷积, 深度可分离卷积可以在保持神经网络识别精度的前提下, 减少网络的参数量和计算量. 该加速器支持对深度可分离卷积的两个算子, 即深度卷积和点卷积, 分别进行加速, 并且两个加速模块之间共享存储模块、乘法器单元、后处理单元等结构, 提高了硬件资源利用率. 深度卷积加速流水线采用了 Winograd 快速卷积算法, 且专门优化了数据存取方式来减少传输数据冗余. RISC-V 处理器拓展了自定义指令对加速器进行灵活的配置及调用, 并且采用了 TensorFlow Lite 库, 用拓展指令修改算子实现函数, 以支持高效部署常规图像神经网络以及推理加速.

经过对多个使用了深度可分离卷积的神经网络进行测试, 结果显示, 相比于 RISC-V 处理器, 本文加速器可以将点卷积加速 104.40 倍, 将深度卷积加速 123.63 倍, 且功耗极低, 仅为 0.123 W, 性能功耗比达 8.7GOPS/W, 满足在资源不足且功耗受限的情况下部署神经网络的需求. 本文相关代码已开源至 <https://github.com/2270142596/WinoFPGA>.

## 参 考 文 献

- [1] Krichen M. Convolutional neural networks: A survey. *Computers*, 2023, 12(8): 151
- [2] Tu Fengbin, Yin Shouyi, Ouyang P, et al. Deep convolutional neural network architecture with reconfigurable computation patterns. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017, 25(8): 2220-2233
- [3] Krzywaniak A, Czarnul P, Proficz J. Dynamic GPU power capping with online performance tracing for energy efficient GPU computing using DEPO tool. *Future Generation Computer Systems*, 2023, 145: 396-414
- [4] Zhou Xin, Dou Yong, Li Rongchun, et al. A pipelining

strategy for accelerating convolution neural networks on ARM CPUs. *Concurrency and Computation: Practice and Experience*, 2022, 34(2): e6102

- [5] Srivastava H, Sarawadekar K. A depthwise separable convolution architecture for CNN accelerator//*Proceedings of the 2020 IEEE Applied Signal Processing Conference (ASPCON)*. Kolkata, India, 2020: 1-5
- [6] Liao Jiawen, Cai Liangwei, Xu Yuan, et al. Design of accelerator for MobileNet convolutional neural network based on FPGA//*Proceedings of the 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. Chengdu, China, 2019, 1: 1392-1396
- [7] Xuan Lei, Un K F, Lam C S, et al. An FPGA-based energy-efficient reconfigurable depthwise separable convolution accelerator for image recognition. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2022, 69(10): 4003-4007
- [8] Li Baoting, Wang Hang, Zhang Xuchong, et al. Dynamic dataflow scheduling and computation mapping techniques for efficient depthwise separable convolution acceleration. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021, 68(8): 3279-3292
- [9] Li Guoqing, Zhang Jingwei, Zhang Meng, et al. Efficient depthwise separable convolution accelerator for classification and UAV object detection. *Neurocomputing*, 2022, 490: 1-16
- [10] Pan S Y, Lee S Y, Hung Y W, et al. A programmable CNN accelerator with RISC-V core in real-time wearable application //*Proceedings of the 2022 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*. Tainan, China, 2022: 1-4
- [11] Hoyer I, Utz A, Lüdecke A, et al. Design of hardware accelerators for optimized and quantized neural networks to detect atrial fibrillation in patch ECG device with RISC-V. *Sensors*, 2023, 23(5): 2703
- [12] Askarihemmat M, Wagner S, Bilaniuk O, et al. BARVINN: Arbitrary precision DNN accelerator controlled by a RISC-V CPU//*Proceedings of the 28th Asia and South Pacific Design Automation Conference*. New York, USA, 2023: 483-489
- [13] Liang Tailin, Wang Lei, Shi Shaobo, et al. TCX: A RISC style tensor computing extension and a programmable tensor processor. *ACM Transactions on Embedded Computing Systems*, 2023, 22(3): 1-27
- [14] Tong Gan, Huang Li-Bo. A review of research on Winograd fast convolution. *Journal of Frontiers of Computer Science & Technology*, 2022, 16(5): 959-971(in Chinese)  
(童敢, 黄立波. Winograd 快速卷积相关研究综述. *计算机科学与探索*, 2022, 16(5): 959-971)
- [15] Lavin A, Gray S. Fast algorithms for convolutional neural networks//*Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Las Vegas, USA, 2016: 4013-4021
- [16] Mo Zhuofeng, Luo Dehan, Wen Tengpeng, et al. FPGA implementation for odor identification with depthwise separable convolutional neural network. *Sensors*, 2021, 21(3): 832

- [17] DiCecco R, Lacey G, Vasiljevic J, et al. Caffeinated FPGAs: FPGA framework for convolutional neural networks//Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT). Xi'an, China, 2016: 265-268
- [18] Xiao Qingcheng, Liang Yun, Lu Liqiang, et al. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs//Proceedings of the 54th Annual Design Automation Conference. New York, USA, 2017: 1-6
- [19] Wu Ruofan, Zhang Feng, Guan Jiawei, et al. Drew: Efficient Winograd CNN inference with deep reuse//Proceedings of the ACM Web Conference 2022. Lyon, France, 2022: 1807-1816
- [20] Zhang Feng, Wu Ruofan, Guan Jiawei, et al. Expanding the edge: Enabling efficient Winograd CNN inference with deep reuse on edge device. *IEEE Transactions on Knowledge and Data Engineering*, 2023, 35(10): 10181-10196
- [21] Lin Jilan, Li Shuangchen, Hu Xing, et al. CNNWire: Boosting convolutional neural network with Winograd on ReRAM based accelerators//Proceedings of the 2019 on Great Lakes Symposium on VLSI. New York, USA, 2019: 283-286
- [22] Lentaris G, Chatzitsompanis G, Leon V, et al. Combining arithmetic approximation techniques for improved CNN circuit design//Proceedings of the 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS). Glasgow, UK, 2020: 1-4
- [23] Wang Huizheng, Zhang Zaichen, You Xiaohu, et al. Low-complexity Winograd convolution architecture based on stochastic computing//Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP). Shanghai, China, 2018: 1-5
- [24] Ghaffar M M, Sudarshan C, Weis C, et al. A low power in-DRAM architecture for quantized CNNs using fast Winograd convolutions//Proceedings of the International Symposium on Memory Systems. Washington, USA, 2020: 158-168
- [25] Chen Weiwen, Wang Yaohua, Yang Chao, et al. Hardware acceleration implementation of three-dimensional convolutional neural network on vector digital signal processors//Proceedings of the 2020 4th International Conference on Robotics and Automation Sciences (ICRAS). Wuhan, China, 2020: 122-129
- [26] Wang Shihang, Zhu Jianghan, Wang Qi, et al. Customized instruction on RISC-V for Winograd-based convolution acceleration//Proceedings of the 2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP). NJ, USA, 2021: 65-68
- [27] Sifre L, Mallat S. Rigid-motion scattering for texture classification. *arXiv preprint arXiv:1403.1687*, 2014
- [28] Chollet F. Xception: Deep learning with depthwise separable convolutions//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Honolulu, USA, 2017: 1251-1258
- [29] Howard A G, Zhu Menglong, Chen Bo, et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017
- [30] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014
- [31] Zhao Sijie, Gao Shangshang, Wang Rugang, et al. Acceleration and implementation of convolutional neural networks based on FPGA. *Digital Signal Processing*, 2023, 141: 104188
- [32] Wang Zilun, Mao Wendong, Yang Peixiang, et al. An efficient FPGA accelerator for point cloud//Proceedings of the 2022 IEEE 35th International System-on-Chip Conference (SOCC). Belfast, UK, 2022: 1-6
- [33] Zhai Jiaqi, Li Bin, Lv Shunsen, et al. FPGA-based vehicle detection and tracking accelerator. *Sensors*, 2023, 23(4): 2208
- [34] Yu Yuanxuan, Zhao Tiandong, Wang Kun, et al. 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
- [35] Yang Dingkun, Luo Zhiyong. A parallel processing CNN accelerator on embedded devices based on optimized MobileNet. *IEEE Internet of Things Journal*, 2023, 10(21): 18844-18852
- [36] Li Zhenhao, Hu Wei, Chen Shuang. Design and implementation of CNN custom processor based on RISC-V architecture//Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). Zhangjiajie, China, 2019: 1945-1950



**CAO Xi-Yu**, Ph. D. candidate. His main research interests include mobile edge computing and Internet of Things.

**CHEN Xin**, M. S. His main research interests include machine learning and hardware acceleration.

**WEI Tong-Quan**, Ph. D., associate professor. His main research interests include Internet of Things, mobile edge computing, and cloud computing.

## Background

With the improvement of hardware performance, Convolutional Neural Network (CNN) have achieved significant breakthroughs in fields such as image processing, medical image analysis, and autonomous driving. However, due to the substantial growth in model parameters and computational requirements, particularly on resource-constrained edge devices, the efficient deployment of CNN has become a formidable challenge.

In the realm of CNN acceleration, primary research platforms include GPUs, ARM architecture CPUs, RISC-V, and others. GPUs often require considerable power consumption, while the ARM architecture is characterized by a relatively closed design and high licensing fees. The use of RISC-V processors combined with accelerators emerges as a prospective deployment solution for neural networks in resource-constrained environments. However, research in this field is currently limited. Existing accelerators often develop independent computation modules for different neural network operators, leading to significant consumption of hardware resources. Some studies employ traditional convolutional kernels and direct multiplication-addition computation methods when computing deep convolutions, lacking optimization through relevant algorithms, resulting in untapped potential for improved accelerator computational efficiency. Some studies

have failed to encapsulate relevant instructions or functions, making the calling process rather cumbersome.

This paper presents the design of a depthwise separable convolutional neural network accelerator, combining RISC-V processors with an accelerator in a collaborative mode, aiming to address the current shortcomings in research. Specifically, the paper adopts the Winograd fast convolution algorithm and module reuse design to achieve efficient computation for depthwise convolutions and point convolutions. Through the extension of RISC-V instructions, a set of deep separable convolution instructions is designed, enabling the processor to flexibly and automatically invoke the accelerator. Ultimately, the accelerator is implemented on FPGA hardware, demonstrating low power consumption and a significant performance-to-power ratio. Specifically, the accelerator achieves a 104.40x speedup for point convolutions and a 123.63x speedup for depthwise convolutions. Simultaneously, the performance power efficiency of the accelerator reaches 8.7 GOPS/W.

This work was supported by the General Program of the National Natural Science Foundation of China (No. 62272169), the Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX), and the project of Shanghai Trusted Industry Internet Software Collaborative Innovation Center.