



FPGA Overlay 处理器加速 AI 计算

何磊^{1*}, 王堃², 吴晨^{3,4,1}, 陶卓夫⁵, 时霄⁶, 苗斯元⁵, 陆少强^{3,4,1}

1. 宁波东方理工大学 (暂名) 信息科学与技术学部, 宁波 315200, 中国

2. 复旦大学微电子学院, 上海 200433, 中国

3. 芯粒 CAD 和制造浙江省工程研究中心, 宁波 315201, 中国

4. 宁波数字孪生 (东方理工) 研究院, 宁波 315201, 中国

5. Electrical and Computer Engineering, University of California, Los Angeles, Los Angeles 90095, USA

6. 东南大学计算机科学与工程学院, 南京 211189, 中国

* 通信作者. E-mail: lhe@eitech.edu.cn

收稿日期: 2024-11-19; 修回日期: 2025-02-18; 接受日期: 2025-03-18; 网络出版日期: 2025-04-03

“科创甬江 2035” 关键技术 (批准号: 2024Z283) 资助项目

摘要 目前, 神经网络模型的计算复杂度与更新迭代速度正呈现出爆炸式增长的趋势. 为了在设备上高效使用神经网络算法, 加速神经网络的技术得到了学术界与工业界的关注. 主流的神经网络加速器, 主要包含两大类. 第一类通常针对特定类型的神经网络进行定制化加速, 提升神经网络算法的性能. 然而, 这类定制加速器方案存在可扩展性较差, 计算数据流固定等局限性. 这种局限性使得这些方案无法灵活应对需要频繁切换网络模型的应用场景, 尤其在高动态切换和实时响应的场景中表现不佳. 第二类则是基于指令集 Overlay 处理器. 这类方案首先提出一套针对人工智能 (artificial intelligence, AI) 加速的指令集, 在面向新型网络时, 可通过扩展指令实现快速支持. 与此同时, 通过编译器在软件侧实现神经网络计算图的优化、指令分配和映射的优化; 通过硬件架构在硬件侧完成数据流、并行化等加速优化. 对此, 本文首先分析了专用定制神经网络加速器与通用覆盖神经网络处理器 (Overlay neural network processing) 的设计理念和架构特点. 其次通过对这两类加速器在设计方案、灵活性以及应用适用性方面的详细比较, 本文深入探讨了通用 Overlay 神经网络处理器, 强调其在易用性、扩展性和通用性方面的显著优势. 再次对现有通用 Overlay 神经网络处理器的现场可编程门阵列 (field-programmable gate array, FPGA) 原型进行了系统分类, 并对每一类处理器的设计原理、实现方法以及实际应用中的表现进行了深入剖析. 最后本文讨论了通用 Overlay 神经网络处理器在设计过程中面临的挑战, 并展望了该技术在未来发展中的潜在方向和应用场景, 以为今后的研究工作提供参考和借鉴.

关键词 人工智能, 现场可编程门阵列, 专用定制加速器, 通用覆盖加速处理器, 编译器, 硬件加速

引用格式: 何磊, 王堃, 吴晨, 等. FPGA Overlay 处理器加速 AI 计算. 中国科学: 信息科学, 2025, 55: 796–836, doi: 10.1360/SSI-2024-0351

He L, Wang K, Wu C, et al. FPGA Overlay processor for AI computing. Sci Sin Inform, 2025, 55: 796–836, doi: 10.1360/SSI-2024-0351

1 神经网络以及硬件加速背景

深度神经网络 (deep neural networks, DNN)^[1] 在海量应用场景上, 相比于传统算法, 存在巨大的优势, 例如精度高、大数据适应性强. 因此, 各种不同的 DNN 模型被提出, 以针对不同的细分场景进行特定的优化. 例如, 在计算机视觉领域, 卷积神经网络 (convolutional neural network, CNN) 被广泛应用于目标识别、目标检测的图像领域^[2]. 在自然语言处理领域, 从最开始的循环神经网络 (recurrent neural network, RNN)^[3] 到现在基于 Transformer 的生成式预训练模型 (generative pre-trained transformer, GPT)^[4], 极大程度地提升了机器翻译和文本生成的性能. 在数据分析领域, 图神经网络 (graph neural network, GNN)^[5] 在处理图结构数据方面展示出了优越性能与极大潜力.

然而, 上述神经网络往往通过增加模型复杂度去兼容场景需求以及提升精准度, 这导致模型数据量与计算量指数型增长. 从 1998 年的 LeNet-5^[6] 到 2024 年的 ChatGPT-4¹⁾, 模型的参数量从 0.17 MB 增长到 3.6 TB, 计算量从 435 KFLOPS 增加到了 3700 TFLOPS. 当然也有一些工作通过提出先进的网络结构来减少模型的参数量, 但往往带来了精度的损失^[7,8]. 如何在模型大小和模型精度之间保持平衡, 一直以来都是神经网络设计的重点和难点.

硬件加速的首要问题是选用何种硬件平台满足神经网络计算所要求. 通用中央处理器 (central processing unit, CPU), 因其高灵活性适合处理复杂的控制流任务, 能够非常好地支持各类神经网络的推理部署, 但在大规模并行计算中性能较差. 通用图形处理器 (graphics processing unit, GPU), 拥有大量的计算核心, 能够高效并行处理矩阵运算, 适用于深度学习的推理部署, 尤其在图像处理和自然语言处理等任务中表现优异, 但依旧存在高能耗和高内存需求的问题. 专用集成电路芯片 (application specific integrated circuit, ASIC), 通过定制化设计实现了极高的计算效率和能效, 但其设计周期长、灵活性差, 无法适应神经网络算法的快速发展. 相比之下, 现场可编程门阵列 (field programmable gate array, FPGA), 因其可现场重构性, 可以根据不同的神经网络模型进行硬件逻辑的动态调整, 实现了低延时、低功耗和高性能的计算, 成为一种高可行性的神经网络加速方案. 然而, 基于 FPGA 的神经网络加速器设计, 除了考虑到高性能和高能效比的问题, 还存在以下挑战.

(1) 将神经网络优化部署到 FPGA 上, 开发周期和难度都远大于部署到 CPU 或者 GPU 上. 主流神经网络开发框架 Tensorflow 或 Pytorch, 都只能支持 CPU 或 GPU, 而无法快速部署到 FPGA 上.

(2) 相比于 CPU 与 GPU, FPGA 开发者直接面向底层硬件电路设计, 开发者从零搭建完整功能硬件电路, 需要涉及电路设计的各个方面. 而且开发使用的 RTL (register transfer level) 代码开发难度较大, 难以加速研发周期.

为解决上述问题, 文献 [9, 10] 提出使用自动化编译的方式, 自动生成神经网络加速器的 RTL 代码. 他们设计了一些 RTL 模板和参数, 并通过多目标优化的方式确认最优参数, 从而生成针对某个特定神经网络的 RTL 代码. 然而, 这种方式依旧需要重新进行综合、布局布线及时序调优, 依旧难以加速研发周期.

Overlay 处理器 (Overlay processing unit, OPU) 则是通过设计神经网络的通用指令集的方式来解决上述问题. OPU 针对神经网络算法中的通用算子设计粒度适中的指令集, 兼顾指令的灵活性和使用效率. 配合指令集设计的硬件微架构, 进一步使用并行化、流水线等技术, 优化数据流, 提升加速处理器的吞吐量. 与此同时, 编译器一方面考虑算子融合、量化等计算图的优化, 另一方面考虑计算资源分配、指令排序等充分发挥硬件架构的算力, 进一步提升神经网络端到端的运行速度. OPU 的核心是针对神经网络设计的指令集, 从最初的设计只支持卷积神经网络, 通过简单的扩展指令集, 已快速支持轻量级卷积神经网络, 基于 Transformer 的语言模型和大语言模型, 以及图神经网络. 例如从 CNN 到 Transformer, 指令集层面只增加了一组非线性计算指令. 而到现在支持最新的 Mamba2 网络^[11], 在指令上并不需要做过多扩展, 更多的是在编译器上, 进行计算图、数据流层面的优化. 另一方面, 使用

1) GPT-4 technical report, 2024. <https://arxiv.org/abs/2303.08774>.

OPU 的指令集和硬件微架构设计,当前已在 FPGA 上完成原型验证,也同样可以快速进行 ASIC 芯片实现.

对此,本文充分调研当前的神经网络加速器工作.总结来说,本文的主要贡献如下.

首先,分析了专用定制神经网络加速器,分析其在硬件架构设计和软件设计上如何解决上述问题.

其次,总结了 Overlay 处理器的设计特点及设计理念,并且对比了 Overlay 处理器和专用定制加速器的区别.

再次,我们基于支持的神经网络类型,对现有通用 Overlay 神经网络处理器的 FPGA 原型设计进行系统的分类,并针对每一类处理器的设计原理、实现方案及实际应用进行详细探讨.

最后,我们总结了基于 FPGA 的 Overlay 处理器在设计中面临的挑战,并展望了该技术的发展方向与前景.

2 基于 FPGA 的专用定制神经网络加速器

基于 FPGA 的神经网络硬件加速研究一直受到广泛关注.一个设计良好的基于 FPGA 的专用定制神经网络加速器需要充分考虑神经网络计算中的并行性,从而降低端到端延迟和提升吞吐量.与此同时,得益于 FPGA 的可重构性,这类专用定制神经网络加速器也能够快速地被重构成支持新的神经网络.实现一个高性能的基于 FPGA 的专用定制神经网络加速器,通常需要针对特定的神经网络算法,考虑并行架构的探索、存储带宽的优化和软硬件接口设计.本节介绍基于 FPGA 的专用定制神经网络加速器的硬件架构及软件设计.

2.1 专用定制神经网络加速器的硬件架构设计

专用定制神经网络加速器的硬件架构,是针对某个特定的神经网络设计和优化的.一般来说,一个专用定制神经网络加速器,主要针对以下两个方面进行优化.(1) 优化存储处理模块,通过优化片上和片外存储器的数据排列,数据交互,从而充分利用数据带宽,减轻带宽限制带来的性能瓶颈.具体来讲,部分工作通过动态数据量化和特定的数据排列策略^[12~14]来提高片外存储的带宽利用率.然而,另一部分工作通过提出单次加载策略^[15,16]或集成存储计算功能的技术^[17~19]来减少数据传输,从而减轻带宽限制的瓶颈.

(2) 优化计算模块,通过加速计算量大的网络层,例如卷积层、全连接层等,或者优化复杂运算的后处理层,例如非线性激活等,来提升运行效率.文献^[20~22]通过优化点积计算单元;文献^[23~27]通过优化实现高性能的并行架构;文献^[24,26,28~32]提出针对特定算法的固定数据流方法,有效提升了计算吞吐量和处理效率.此外,为了应对多样的神经网络模型需求,大量的工作采用了多任务功能划分方式^[25,30,33~35],针对稀疏矩阵运算的特定架构优化^[31,34,36~38],特别是为 Transformer 模型设计的可配置处理单元和数据选择优化策略^[21,33,35,38~41],都有效提升了特定模型的计算吞吐量.虽然这样的架构,也可以用来支持不同的神经网络算法,但这些模块的核心参数(例如片上存储器的位宽和容量,核心计算模块的数量以及并行度等),都是针对特定的神经网络算法进行优化的.以 2015 年 FPGA 的工作^[42]为例,他们提出了 Roofline 模型,针对某个特定的神经网络中的每一层进行评估,并选择最优的 Tiling 大小,从而确定硬件架构的设计参数.按照文中提到的案例,由于 64×7 是最优的 Tiling 大小,他们将计算引擎设计为接收 7 个输入特征图的数据和 7 个参数的数据.这样的设计,针对特定案例可以得到最优的吞吐量性能.然而,当神经网络的结构和参数变化后,最优的 Tiling 大小也会变化,从而对应的计算引擎参数变化,即需要一个新的专用定制的神经网络加速器硬件.

2.2 专用定制神经网络加速器的软件设计

专用定制神经网络加速器,在软件设计层面,通常采用了高级编译技术和智能调度算法来优化

神经网络算法的执行. 这些工作通过系统化的数据流和循环操作分析, 以及优化算法推理的调度策略 [14~16, 19, 21, 22, 25~27, 30~35, 38, 41, 43], 不仅提高了数据处理的效率, 还减少了数据移动. 另一部分工作实现从模型到硬件代码的高层次综合以及自动化转化, 并且通过调整计算过程和使用状态机, 进一步从软件侧优化神经网络的推理性能 [9, 15~18, 20, 21, 23~26, 28~30, 35~37, 39~41, 44]. 虽然这些工作在软件侧都有自动化的编译工具来辅助优化神经网络的推理过程, 但是, 这些工作都旨在生成一套优化的 RTL 代码, 针对一个新的神经网络算法, 依旧需要重新生成硬件比特流. 具体来讲, Caffeine [43] 提出了一套从高层次模型 (Caffe) 到底层硬件加速器的自动化编译软件流. 他们将整个流程分成软件自动化和硬件自动化两个步骤. 在软件自动化中, 首先, 对神经网络进行模型解析, 提取需要加速的计算图. 随后, 利用他们提出的统一的计算图表示形式, 表示卷积计算和全连接计算. 再者, 对这些计算图, 在考虑数据复用、数据排列等方面对 weights 数据进行重排优化, 以达成最优的带宽使用策略. 在硬件自动化流程中, 他们提出优化的 Roofline 模型, 综合考虑 DSP (digital signal processor) 数量、片上存储量、片外存储带宽等约束, 根据预先定义硬件模板, 自动生成 RTL 代码, 并完成综合、布局布线 and 生成 FPGA 所需的比特流. 同样地, Auto-Compiler [9] 首先针对神经网络中不同的算子和计算流程手动设计优化的设计模板, 并定义可优化的参数. 针对每个神经网络, 他们在软件侧针对这些参数进行优化, 将神经网络映射到这些手动优化的硬件设计模板上, 在性能上可以和纯手动设计的专用定制神经网络加速器相当.

总结来讲, 专用定制神经网络加速器的软件设计, 重点在考虑神经网络算法以及目标的 FPGA 板卡, 重新生成一套优化的 RTL 代码, 通过重新综合、布局布线生成一个优化的专用定制神经网络加速器. 在这种情况下, 每个神经网络算法或每一个 FPGA 板卡都会对应一个专用定制的加速器.

2.3 专用定制神经网络加速器的特点总结

专用定制神经网络加速器, 针对特定的神经网络进行优化, 因此可以获得极致的资源利用率, 从而保证端到端延时和吞吐量上的优势. 然而, 这类加速器的缺点也非常明显. (1) 设计工作量巨大. 针对新的神经网络, 专用定制神经网络加速器都需要重新进行架构设计、数据流优化等, 这导致巨大的工作量. 尽管某些自动化软件可以根据既定的架构产生针对新的神经网络的加速器 RTL 代码, 但是需要重新完成综合、布局布线、时序优化的 FPGA 流程, 同样需要巨大的工作量. (2) 灵活性低. 专用定制神经网络加速器, 往往针对某个神经网络的参数特别定制, 很难在保证高性能的情况下, 扩展到新型神经网络的新参数上. 例如一个针对 CNN 定制的神经网络加速器, 很难高效地扩展到基于 Transformer 的神经网络. (3) 应用适用性差. 针对复杂的深度学习场景, 往往需要多个神经网络的级联, 专用定制神经网络加速器需要在运行时对 FPGA 重新编程 (重新加载特定神经网络的比特流), 会非常低效; 然而针对边缘计算的场景, 甚至是无法实现的.

3 基于 FPGA 的通用 Overlay 神经网络处理器方案

为解决专用定制神经网络加速器的缺点, 基于 FPGA 的通用 Overlay 神经网络处理器应运而生. 通用 Overlay 神经网络处理器, 在保证针对神经网络加速性能的前提下, 解决专用定制神经网络设计工作量大、灵活性低和应用适应性差的缺点. 本节从 Overlay 处理器的定义、优势、设计挑战和现用工作这几方面进行介绍.

3.1 Overlay 处理器的定义

基于 FPGA 的 Overlay 处理器, 是一种可配置的硬件加速处理器. 首先, 针对一系列神经网络算法的计算方式进行分析, 并抽象成指令集. 其次, 设计实现对应指令集的微架构. 最后, 完成从神经网络算法到指令集的编译映射. 基于 FPGA 的 Overlay 处理器, 其本质为使用一套电路架构, 通过重新

编译生成指令来覆盖多种神经网络模型的硬件加速方案.

3.2 Overlay 处理器的优势

基于 FPGA 的 Overlay 处理器,一般具有以下三点优势.(1) 并行度高. Overlay 处理器,通常利用 FPGA 的特性,设计高度并行的计算核心,同时执行多个运算单元,从而加速神经网络中的核心计算模块.(2) 可重配置性. Overlay 处理器允许用户根据不同的神经网络,进行重新配置进行优化,并实现灵活支持多种神经网络的目的.与此同时,Overlay 处理器通常设计一套指令集,在保证使用同一套微架构的前提下,通过编译器重新编译不同的神经网络,大大降低了支持不同神经网络的开发周期.(3) 高能效比.相比于通用 CPU 和 GPU,基于 FPGA 的 Overlay 处理器可以提供更高的能效比,尤其是在低功耗场景下表现出色.

3.3 Overlay 处理器的设计挑战

Overlay 处理器,在神经网络加速层面体现出巨大的优势,然而,设计一个高性能的 Overlay 处理器,依旧存在严峻的挑战.(1) 指令集设计. Overlay 处理器的核心是指令集,如何设计一套完备、高效的领域专用的指令集,是 Overlay 处理器设计的第一大挑战.指令集的设计,一方面要能够保证足够的灵活性,以支持不同的神经网络;另一方面要保证计算核心的高性能,以实现 Overlay 处理器的高能效比.(2) 硬件微架构设计.虽然 FPGA 可以支持高并行度,但 FPGA 上资源有限,如何充分利用有限的 FPGA 资源实现高并行度是非常有挑战的.与此同时,Overlay 处理器同样存在带宽限制的问题,如何充分利用 FPGA 的存储带宽,充分发挥 FPGA 上并行的计算核心,是另一大设计挑战.(3) 编译器设计.编译器不仅要能够把不同的神经网络高效地映射到 Overlay 处理器上,还需要能够充分利用 Overlay 处理器的算力,以最高效率地发挥 Overlay 处理器的计算能力.

3.4 现有的 Overlay 处理器工作总览

考虑到上述优势及设计挑战,OPU 系列处理器已针对 Overlay 处理器做了深入和广泛的研究.近年来,多种创新的 Overlay 处理器方案被提出,包括统一的 FPGA 编译框架^[45]、打破深度神经网络与 FPGA 加速器之间编译壁垒的方法^[46],以及基于图的多 DNN 调度策略^[47].OPU 的核心是一套力度适中的针对神经网络加速的通用指令集.这指令集从设计之初只针对 CNN,随着神经网络的发展,这套指令集可简单地扩展为支持轻量级卷积神经网络、Transposed 卷积神经网络、基于 Transformer 的语言模型及大语言模型,以及图卷积神经网络.本文后续针对 OPU 系列处理器按照支持的网络类型进行分类,并对其硬件架构和编译器做详细介绍(第 5~7 节).

4 基于 FPGA 的 Overlay 处理器的基本结构

基于 FPGA 的 Overlay 处理器通常由指令集、编译器和硬件处理器三大模块组成,结合低精度定点、低精度浮点(low-precision floating point, LPFP)量化优化方法,进一步提升了推理效率与硬件适配性.如图 1 所示,处理器通过接收多个异构神经网络模型,经由图级中间表示(graph-level intermediate representation, graph-level IR)和指令级中间表示(instruction-level IR)两个关键优化阶段,实现了从模型到 FPGA 的高效映射.该架构设计实现了神经网络的系统化硬件部署,进一步充分发挥 FPGA 在通用处理器构建中的架构优势.

4.1 软件侧设计 —— 端到端的编译器设计

深度学习模型的结构发展迅速,从早期基于二维卷积的 CNN,到近期基于 Transformer 的大语言模型(large language model, LLM),其算子的种类与形状多样性之大,远超深度学习加速硬件的开发迭代速度.这使得这些模型在硬件上难以及时得到充分的优化,无法充分发挥硬件的性能.编译器,作为

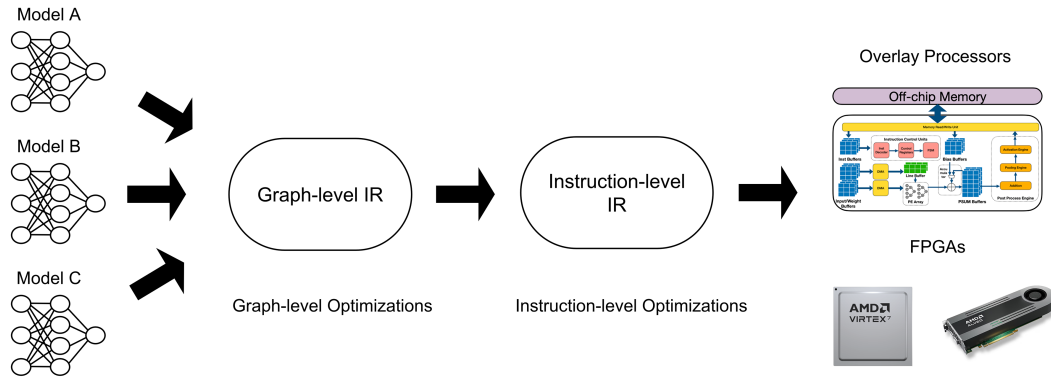


图1 (网络版彩图) Overlay 处理器基本架构.

Figure 1 (Color online) Architecture overview of Overlay processors.

深度学习模型和硬件处理器之间的桥梁,在开发速度、泛用性、性能上都起到了关键的提升作用.首先,在开发速度上,编译器能把模型映射到硬件的过程自动化,提升了工程效率.其次,在泛用性上,编译器提供了模块化的软件框架,能快速支持不同模型和不同硬件,避免重复的工程任务.最后,在最重要的性能上,编译器提供了一个平台,开发者可以通过优化算法,寻求在硬件约束下的最佳性能.在编译器设计方面,研究者们提出了多种优化方案,包括可扩展的算子生成框架^[48]、基于层融合的流水线架构^[49]、灵活的稀疏矩阵乘法设计^[50],以及打破数据布局与循环优化壁垒的方法^[51].

上述优点的实现离不开一个好的中间表示(IR).首先,中间表示需要能够表示模型各个算子的计算方式.其次,中间表示需要表示硬件的约束,例如有限的支持指令、有限的数据访问方式等.由此,编译器能够完成从模型到硬件的映射,确保各种分析和优化的进行.

基于FPGA的Overlay处理器的编译器一般由前端和后端两个部分组成.前端将深度学习模型描述为数据流图,并进行图级别的优化,后端将图递降(lowering)成Overlay处理器的指令,并进行指令级别的优化,最后生成Overlay处理器可以执行的机器码.在下面的章节里,在4.1.1小节介绍中间表示,4.1.2小节介绍前端的设计与优化,4.1.3小节介绍后端的设计和优化.

4.1.1 编译器中间表示设计

编译器在前端使用数据流图(dataflow graph)作为IR.图2(a)展示了常用的Transformer Block的一部分,包括KQV乘法和self-attention.数据流图中,每个节点是一个深度学习算子(例如,矩阵乘法(matrix multiplication, MM)、卷积、偏置加法、Softmax激活函数等),每一条边表示数据流方向,同时包含数据张量(tensor)的形状,这表示前端将进行算子层面的优化.

编译器在后端使用指令作为IR.图2(b)展示了KQV乘法其中一个的乘法指令表示.这里的指令直接从Overlay处理器抽象而来,包括外部数据读写(例如,Load指令描述片外到片内的数据读、Store指令描述片内到片外的数据写)和计算(例如,MatMul指令描述片内的矩阵计算).更多指令的详细信息在4.2小节中介绍.Overlay处理器的编译器在后端会保留loop结构,避免结构化的信息丢失在大量的指令中,方便在优化的过程中进行高层次(high-level)的数据流变化(transformation),同时也能够提升编译速度.

4.1.2 编译器前端设计与优化

编译器前端的语法分析器(parser)能够从各个深度学习框架(例如TensorFlow, PyTorch, Jax等)开发的模型分析并建立数据流图.目前,有很多开源的深度学习编译器,例如TVM, GLOW都在这方面有很好的研究.然而,Overlay处理器的编译器,往往会基于开源的深度学习编译器进行扩展和补充,

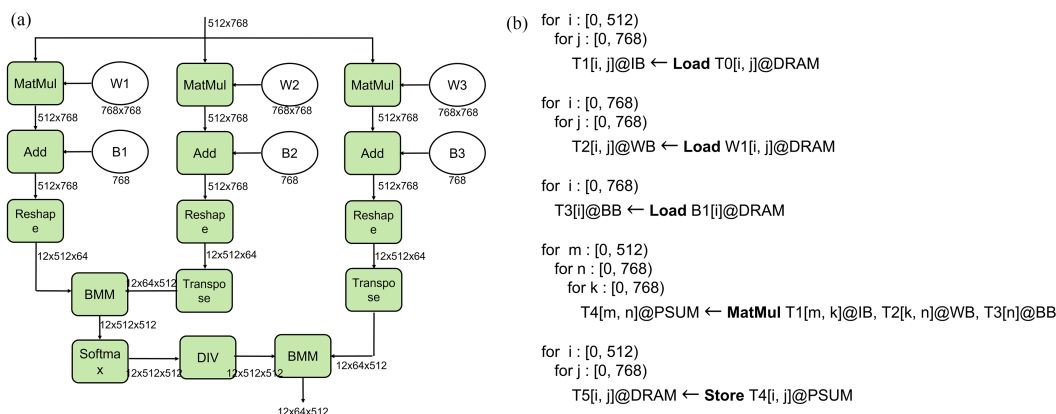


图 2 (网络版彩图) 编译器的中间表示形式示例. (a) 前端的图级 IR; (b) 后端的指令级 IR.

Figure 2 (Color online) IR of compilers. (a) Front-end graph level IR; (b) back-end instruction level IR.

并且结合硬件特征,在图级别做特定的优化.下面介绍两种常见的优化方式设计: Constant Propagation 和算子融合.

Constant Propagation. 传统的 Constant Propagation 能够将 runtime static 的值提前计算好,从而减少 runtime 的计算.然而,传统的 Constant Propagation 只对计算图进行分析,不会改变计算图的结构,这样会错过一些优化的机会.在 OPU^[52] 中,他们会适当改变一些算子的计算顺序,并且保证改变的计算顺序不影响计算图的语义 (semantics),从而提升 Constant Propagation 的机会.

算子融合. 硬件上片外的数据读写与片上的数据读写相比速度缓慢,算子融合的目的是将算子之间的中间结果保留在片上,从而减少片外的数据读写.然而,算子之间的融合方式受限于硬件上的数据流设计.为了找到有效的算子融合方案,编译器的前端需要将硬件上的数据流描述成硬件计算流图,与模型的数据流图进行匹配 (pattern matching).以 OPU 为例,他们使用的融合方案是矩阵乘法或卷积,与 elementwise 或激活或转置等进行融合,从而实现 90% 以上的 PE 利用率.

4.1.3 编译器后端设计与优化

编译器后端将前端优化后的计算数据流图作为输入,将每个融合后的算子各自递降成多个硬件指令的 loopnest.递降的过程基于预选准备的 Overlay 处理器指令库.与此同时,编译器后端通常还会进行一系列的 loop optimization,使得计算与数据读取操作尽可能地重叠,充分利用外部存储的带宽,从而提升 PE 的利用率.随后,编译器后端会展开所有的 loopnest,再对线性的指令列表进行 allocation 和 scheduling,以追求 PE 利用率的进一步最大化.最后,编译器后端会根据指令之间的 dependency,生成指令之间的控制链,并生成指令的机器码.在后端的优化过程中,也需要调用 event-driven 的仿真器以确保后端优化的准确性.

4.2 指令集设计——灵活粒度的指令集设计

对 Overlay 处理器而言,指令集的设计是关键.(1) 指令集需要足够灵活,能够覆盖不同神经网络算法中的不同算子.(2) 指令集需要有足够的可扩展性,能够快速扩展到支持新的算子,以支持新的神经网络算法.(3) 根据指令集设计的硬件处理器,需要足够高效,从而保证 Overlay 处理器的高性能.当然,这三条关键指标很难同时达到,如何在这三者之间进行权衡,设计出最有利于 Overlay 处理器的指令集,非常具有挑战性.

4.2.1 常见的指令集设计

寒武纪^[53] 在 2016 年,提出了一套针对神经网络的指令集.他们分析了当时存在的神经网络架构

和算子,提出了一种以 load-store 为架构的指令集,其中包含了标量运算、向量运算、矩阵运算、逻辑运算、数据传输和控制六大类指令.他们的指令集设计以灵活性为目标,指令设计粒度较小,可以灵活地支持神经网络中不同算子.他们在当时存在的 10 种典型的神经网络上的测试也证明了这套指令集的灵活性.但这套指令集也存在指令量大,能效比较低等问题.

Abdelfattah 等^[54]在 2018 年提出了针对 CNN 的 FPGA Overlay 处理器.他们同样提出了一套针对 CNN 的指令集.他们提出了一种轻量级的超长指令字 (very-long instruction word, VLIW) 架构来提供针对不同网络的灵活性.他们的指令集粒度较大,往往一个卷积层只需要使用 10 条左右的指令来表示.这样的设计大大减少了指令的数量,但也牺牲了灵活性和扩展性,很难高效地支持不同的神经网络.

OPU 在 2019 年提出了一套领域专用的针对卷积神经网络的指令集,这套指令集旨在优化指令的粒度,相比上述两套指令集,既保证了在神经网络领域内的灵活性,也保证了计算单元的高效性.他们设计了条件指令 (conditional instruction, C-type) 和非条件指令 (unconditional instruction, U-type) 两大类指令.其中条件指令包括外部存储器读、外部存储器写、片上数据获取、计算、后处理和指令获取六类,用来描述具体的算子操作和设定执行条件.而非条件指令是跟随对应的条件指令存在,用来传递对应条件指令的执行参数.

4.2.2 基于 FPGA 的 Overlay 处理器的指令集设计目标

基于 FPGA 的 Overlay 处理器的指令集设计,还需要考虑到 FPGA 实现的处理器特性.一方面,要考虑指令集的灵活性和扩展性,需要兼顾不同指令粒度的设计,既能够对常用的算子级的大粒度计算过程进行直接的概括,也能够对一些不常用的细粒度的运算进行支持和扩展.另一方面,基于 FPGA 的 Overlay 处理器,往往使用数据流结构以提升执行的效率.在指令集设计时,也需要兼顾到 FPGA 通用处理器的数据流架构,能够保证指令的执行效率.

4.3 硬件侧设计 —— 对应指令集的电路微架构设计

Overlay 处理器的微架构设计也是充满挑战的.一方面,它需要尽可能简化控制流,另一方面,也需要在运行时支持功能调整.基于这样的考虑,Overlay 处理器会做参数化的架构设计,并且参数寄存器由外部编译器修改与更新,用这样的办法支持不同的功能.在微架构粒度设计上,会保持与指令集粒度相匹配.与此同时,微架构设计也会考虑 FPGA 的具体特性 (例如片上 DSP, 片上 Block RAM 等),在并行度和流水线层面做出考虑,尽可能提升整体性能.一般来说,Overlay 处理器的微架构会用纯数据流的架构来提升性能,与此同时,还会包含指令解析器用于解析指令,并对核心计算模块给出关键的控制信号,保证计算的正确性.后文会对 OPU 系列的 Overlay 处理器在不同神经网络应用场景下的架构设计做出具体介绍.

4.4 软硬协同优化 —— 指令架构设计

LPFP^[55]架构设计直观降低了计算精度,达到减少 FPGA 中的内存使用、提升运算速度和降低功耗的目的,优化 FPGA 在深度学习等应用中的性能.通过精度与效率的平衡,有限资源的 FPGA 中得以实现高效计算.实施 LPFP 优化硬件设计,提高处理器和存储系统的性能,确保在大规模计算任务中保持较低的延迟和功耗.

4.4.1 LPFP 方法与实现

LPFP 量化通过降低数据精度,优化 CNN 的计算效率,同时保持模型性能.其核心包括融合操作和量化过程.在融合操作中,将批量归一化层、最大池化层等与卷积层、全连接层整合,减少冗余计算并优化硬件效率;无法融合的层 (如平均池化层) 独立处理,确保数据流一致性.量化通过动态选择基

于数据分布的比例因子,结合饱和度处理和四舍五入技术,最大限度降低量化误差.硬件实现中,量化数据格式与比例因子在推理阶段固定,降低运行开销.FPGA设计中,量化权重和偏置存储于外部存储器,推理时通过符号位异或、尾数乘法和指数加法完成运算,随后由对齐模块转换为定点格式.尽管可能引入细微精度损失,但实验验证对整体性能影响极小.

4.4.2 处理器架构优化与综合效能

LPFP 量化需求推动了专用处理器架构的设计,核心模块包括浮点功能单元 (floating-point functional unit, FPFU)、片上存储系统 (on-chip memory system, MS) 和中央控制模块 (central control module, CCM). FPFU 由多个并行处理单元构成,优化 LPFP 数据格式计算,采用流水线结构实现高吞吐量卷积和全连接操作.片上存储系统通过乒乓架构与直接存储器访问 (direct memory access, DMA) 模块优化数据流与带宽配置,减少延迟,确保计算连续性.中央控制模块协调 FPFU 和 MS 工作,通过并行模式提升推理速度.

LPFP 量化与专用处理器架构结合,实现了 FPGA 上 CNN 推理的高效执行.量化减少了数据冗余,并通过比例因子优化降低误差;处理器架构优化了存储与计算,达成性能与功耗平衡,为未来高效硬件计算提供了可靠解决方案.

5 卷积神经网络的 Overlay 处理器设计

现场可编程门阵列 (FPGA) 凭借其强大的并行计算能力,成为加速深度卷积神经网络 (CNN) 的理想平台.近年来,Overlay 处理器已广泛应用于传统 CNN 模型和网络中,如文献 [52,56~59] 等.本节深入探讨基于卷积神经网络的 Overlay 处理器设计,并进行全面总结.具体内容包括:卷积运算单元的基本设计、非线性运算单元的构建、轻量级卷积神经网络 Overlay 处理器的设计、混合精度卷积神经网络 Overlay 处理器的设计,最后对现有 CNN 网络 Overlay 处理器的实验数据进行对比分析.

5.1 面向普通矩阵计算的基本运算阵列设计

对于 CNN 加速器和处理器而言,如何在同一处理单元 (processing element, PE) 结构中适应不同的运算层,是一个关键的设计挑战.这要求 PE 单元具备高度的可重构性,以灵活应对各种计算需求.在 CNN 的卷积运算中,涉及的矩阵数据主要包括特征图 (feature map, FM) 和卷积核权重 (kernel weight, KW) 两部分.传统的 PE 设计通常倾向于在单个卷积核内挖掘并行性,这种设计尽管简洁,但存在以下两大缺陷.

(1) 特定尺寸卷积核的优化设计方案,通常难以推广到其他尺寸的卷积核.许多传统的 FPGA 设计只针对常见的 3×3 卷积核进行了架构优化,这极大限制了其在处理其他尺寸卷积核时的性能.

(2) 在对 KW 进行并行优化的同时,FM 数据的管理复杂度相应增加,这对 FPGA 有限的片上块随机存储器 (block random access memory, BRAM) 的容量提出了更大的挑战.

为了解决第一个问题,OPU^[52] 提出了一种解决方案,即将大尺寸的卷积核拆分为多个 3×3 的卷积核来实现.这种方法有效规避了尺寸不匹配带来的性能瓶颈.针对第二个问题,OPU 则提出了更高层级的并行处理方式.在卷积运算过程中,FM 的并行性体现在输入通道数上,而 KW 的并行性则体现在输出通道数上.如图 3 所示,OPU 在 FM 和 KW 这两个维度上同时引入了并行处理.具体而言,在每个时钟周期内,所有输入通道的数据切片会被同时读取,并与对应的卷积核元素并行处理.这种输入数据的并行读取方式契合了自然的数据存储模式,再结合卷积核内的并行计算,相较于传统的 PE 设计,能够更充分地利用 BRAM 带宽,并显著提升计算的并行度,从而大幅优化了卷积运算的效率.

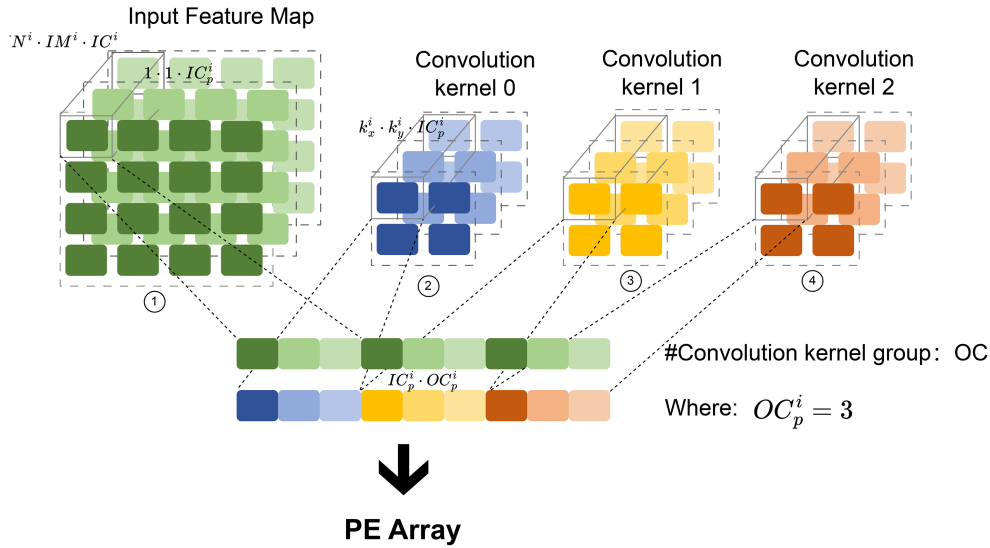


图3 (网络版彩图) OPU 的 FM 间、KW 间的并行数据处理.

Figure 3 (Color online) Parallel data processing between FM and KW of OPU.

5.2 非线性非卷积运算单元设计

与 VGGNet^[60] 和 AlexNet^[61] 相比, ResNet^[62] 和 GoogleNet^[63] 表现出了优异的性能, 并降低了模型推理的计算量. 然而, 它们也引入了需要额外新的非线性和非卷积操作. OPU 对于这些模型提供了额外的硬件支持.

5.2.1 Inception 模块

GoogleNet 引入了 Inception 模块这一重要概念, 其核心思想是在同一层中使用多个不同大小的卷积核同时进行卷积操作, 然后将各个结果进行拼接. 通过这种方式, Inception 模块能够在不同尺寸的卷积运算下实现多尺度特征提取和数据降维的功能, 并结合 ReLU 激活函数增加非线性特性, 增强了网络的表达能力.

Inception 模块的计算过程要求将各输入通道的数据进行逐通道 (channelwise) 的拼接操作. 在 OPU 的内存存储模式下, 前一层输出数据的内存地址是按照通道维度连续存储的, 因此, 在进入下一层的 Inception 模块时, 输入数据的拼接过程无需引入额外的计算操作或延迟. 这种高效的数据组织方式进一步提升了网络的处理速度和计算效率, 充分利用了 FPGA 的并行计算能力.

5.2.2 残差模块

残差模块的核心操作是矩阵元素的加法, 这本质上并不是一个计算密集型的过程, 但残差计算可能引发额外的片外数据通信, 从而影响性能. 为此, OPU 将残差加法操作嵌入到后处理的流水线计算过程中. 残差加法的时间被后处理阶段的计算时间所掩盖, 使得额外的块数据加载不会对整体计算性能造成负面影响. 这种设计不仅有效减少了片外数据通信带来的延迟, 还进一步优化了网络的整体计算效率, 充分发挥了 FPGA 的并行处理优势.

图 4 展示了 CNN 中残差加法模块的数据流架构, 主要包含三条计算路径: (1) 采用三级卷积串联结构, 通过逐层特征提取实现深度特征表达; (2) 使用单级卷积直通路保留原始特征信息; (3) 结合池化和卷积操作, 在降维的同时提取空间特征. 这种多路径设计在保持信息完整性的同时, 实现了不同尺度特征的融合.

OPU 的创新点在于对残差模块的流水线优化设计. 通过将残差加法、ReLU 激活和池化操作重组

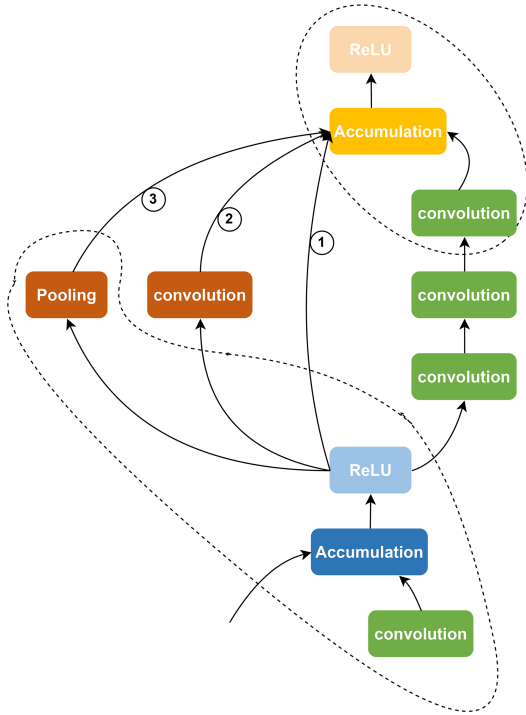


图4 (网络版彩图) CNN中残差加法的数据流.

Figure 4 (Color online) Dataflow for residual add in CNN.

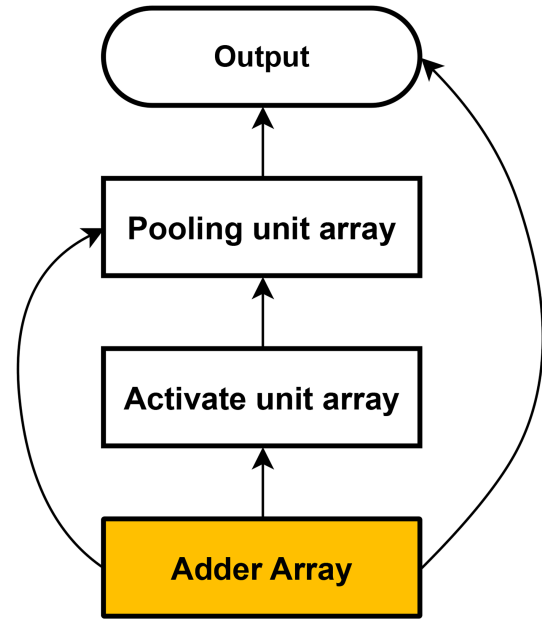


图5 (网络版彩图) OPU中后处理运算模块硬件设计.

Figure 5 (Color online) Architecture of the post-process module in OPU.

并嵌入到前序卷积层的后处理流水线中, 系统实现了计算和数据移动的重叠执行. 这种流水级优化策略具有双重效益: 一方面通过计算重叠掩盖了残差加法的延迟开销, 另一方面将局部数据访问限制在片上缓存区域, 显著降低了内存带宽压力.

图5展示了OPU中残差加法模块的硬件设计. 该硬件架构采用流水线结构, 从底部到顶部依次包含加法器阵列、激活单元阵列和池化单元阵列, 确保数据的顺序处理. 系统设计了两条数据通路: 一条完整的流水线主路径, 和一条直接输出的旁路. 该设计通过多路选择器来选择不同的功能模块, 实现灵活的数据路径调度. 这种模块化、可配置的设计不仅提升了硬件资源的利用率, 也有效减少了额外的数据传输开销, 使得残差加法与其他操作无缝集成在计算流程中, 进一步优化了网络的执行性能. 通过这种灵活的硬件架构设计, 系统可以实现残差加法与其他处理单元的紧集成, 在保证功能完整性的基础上, 通过并行处理和流水线操作提升计算效率.

5.3 轻量级卷积神经网络的 Overlay 处理器设计

Light-OPU^[56] 是基于FPGA实现的通用Overlay轻量级CNN处理器, 可以兼容传统CNN与Light-OPU模型. Light-OPU可以将LW-CNN的运算操作通过编译和灵活的指令集设计高效调度到相应硬件上, 同时指令的执行过程支持动态调度.

5.3.1 Light-OPU 指令集架构

Light-OPU采用CISC型指令框架, 每个指令的执行可能需要花费几百个周期. 其中, 长指令可解码为不同数量的32位RISC型短指令, 具体详见表1.

解码后的32位RISC指令分为两种类型: 条件指令(C-type)和无条件指令(U-type). C-type指令由操作码、触发条件码(trigger condition index, TCI)和地址码组成, 其中触发条件码明确指示了

表 1 Light-OPU CISC 指令解码后的 RISC 指令类型.

Table 1 RISC types of the decoded CISC in Light-OPU.

C type instruction	Memory read	FM load	Read feature map data
		KW load	Read convolution kernel weights
	Memory write	Writing back feature map data	
	Data fetch	FM reuse	Tradition conv
		KW reuse	DW-conv
	Compute	FM share	Tradition conv
		KW share	DW-conv
	Post process	Functions of weightless layers such as pooling layers and activation layers	
Instruction read	Update the address of the next instruction in instruction memory		

指令执行所需的先决条件. Light-OPU 采用动态乱序发射流水线结构来执行指令. 在这种乱序执行模式下, 为了实现有效的指令控制与调度, Light-OPU 根据指令操作间的依赖关系, 为每条指令设计了触发条件列表 (trigger condition list). 在指令执行过程中, 通过修改每条指令的触发条件索引 (trigger condition index, TCI) 来动态更新其执行所需的先决条件.

U-type 指令则提供相应 C-type 指令所需的运算参数. 为优化指令调度, Light-OPU 配置了多个指令单元 (instruction unit). 每个指令单元由 1 条 C-type 指令和多条 (可能为 0 条) U-type 指令组成. U-type 指令的主要功能是更新权重参数, 这并非指训练过程中参数的更新, 而是指获取新的参数以供后续计算使用.

5.3.2 Light-OPU 计算单元对 LW-CNN 的支持

传统 CNN 的卷积运算 (convolution, CONV) 与广泛应用于轻量级卷积神经网络的深度可分离卷积在运算过程上存在显著差异.

深度可分离卷积是 MobileNet 等轻量级卷积神经网络的基本单元, 通常与逐点卷积结合使用. 深度可分离卷积中的深度卷积相比于传统卷积运算显著减少了计算量. 在深度可分离卷积中, 每个卷积核的通道数均为 1, 即每个卷积核专门负责输入特征图的一个通道的运算. 因此, 在深度卷积中, 卷积核数量 (num_kernal) 等于输入特征图通道数 (Channel_in) 及输出特征图通道数 (Channel_out), 即 $\text{num_kernal} = \text{Channel_in} = \text{Channel_out}$. 如果需要改变输出特征图的通道数, 可以在 DW-CONV 后接一个逐点卷积, 即普通的 1×1 卷积, 从而实现通道数的调整.

Light-OPU 在前一代 OPU 的基础上, 设计了计算单元, 使其能够兼容这两类卷积运算及其他卷积算子. 与 OPU 相同, Light-OPU 处理的卷积核主要为 3×3 尺寸, 而对于大尺寸卷积核, 则可将其拆分为多个 3×3 卷积核进行实现.

5.3.3 Light-OPU 编译器

Light-OPU 的配套编译器可以将 CNN 网络配置信息转化为 Light-OPU 上运行的指令程序. 编译器的流程如图 6 所示, 主要实现两个目标. (1) 网络重构, 将网络计算重新定义为硬件友好的操作. 网络重构包括从网络配置解析到操作重新排序的步骤. (2) 硬件映射, 将重制定的网络映射到具有最小执行延迟的硬件. 硬件映射涵盖从网络切片到指令生成的步骤.

在网络重构之前, 需要进行网络配置解析, 以提取与网络结构相关的信息, 包括计算层参数和连接关系等. 网络配置信息来源于张量流、Keras 或 ONNX 生成的冻结模型文件. 网络重构主要包括以下步骤. (1) 计算层分组: 计算层分组将相邻层连接成计算块. 每个计算块由一个卷积层或完全连接层作为主导, 后续接池化层、激活层或残差层. 外部内存访问仅发生在计算块之间, 从而减少通信延迟. (2) 算子融合: 算子融合是硬件加速器配套编译器的典型操作. 在某些 CNN 中, 诸如批处理归一

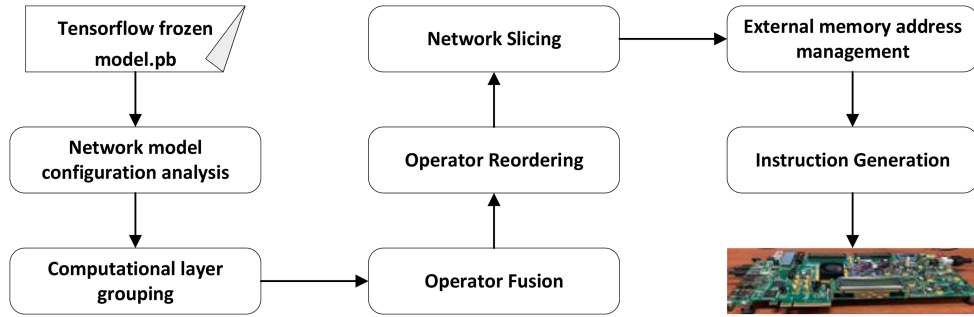


图6 (网络版彩图) Light-OPU 的编译流程.

Figure 6 (Color online) Compiling flow of Light-OPU.

化 (batch normalization) 和补零 (padding) 等层可以完全合并到前面的卷积层中. (3) 算子重排序: 算子重排序在某些处理器中由编译器完成. 在 Light-OPU 的设计中, 此步骤通过乱序发射流水线在硬件层面实现.

为了实现良好的运算性能并在较短时间内完成指令程序的运行, 编译器生成的指令需要合理映射至硬件. 在 Light-OPU 中, 硬件映射主要体现在网络切片的优化上. 网络切片包含两个层次: 其一是二维特征图切片, 以适应片上缓冲区的大小限制; 其二是通道切片, 这是由于片上计算引擎资源有限, 需控制通道并行度.

5.4 转置卷积神经网络的 Overlay 处理器设计

Uni-OPU^[57] 是为了兼容不同类型的转置卷积 (transposed convolution, TCONV) 和传统卷积网络 (CONV) 而设计的高效统一硬件加速的 CNN 网络 Overlay 处理器. 为实现对不同 TCONV 网络的兼容性以及高效的硬件加速, Uni-OPU 开发了一个深度优化的编译器.

5.4.1 转置卷积网络

TCONV 已被广泛应用于许多深度学习网络架构中. 流行的深度学习平台, 如张量流、Keras^[61] 和 Caffe^[62], 有特定的功能来执行 TCONV, 但许多网络使用上采样层和 CONV 层来实现它. 最终输出 FM 尺寸与 TCONV 输入 FM 尺寸的关系如下所示:

$$W_{\text{out}} = \frac{(W_{\text{up}} - K)}{S} + 1, \quad H_{\text{out}} = \frac{(H_{\text{up}} - K)}{S} + 1, \quad (1)$$

其中,

$$W_{\text{up}} = W_{\text{in}} \times \text{up} + 2 \times \text{Pad}, \quad H_{\text{up}} = H_{\text{in}} \times \text{up} + 2 \times \text{Pad}. \quad (2)$$

在 TCONV 中, 第一次上采样步骤可以通过多种插值方法实现. 其中传统的 TCONV 使用零插入法. 然而, 这种方法会在最终输出图像中产生不同尺度的棋盘伪影. 棋盘效应源于在第二次卷积步骤中, 非零输入值的不均匀重叠, 这一现象是由零插入操作引起的. 为了避免引入这些伪影, Uni-OPU 评估了多种上采样算法, 以促进权重绑定并减少不均匀重叠. 在这些方法中, 基于最近邻插值 (nearest neighbor interpolation) 的上采样效果最佳. 该方法下的所有棋盘伪影均已被消除. 这两种转置卷积分别称为零转置卷积 (zero transposed convolution, Zero-TCONV) 和最近邻转置卷积 (nearest neighbor transposed convolution, NN-TCONV).

5.4.2 Uni-OPU 编译器概述

Uni-OPU 是为了兼容不同类型的 TCONV 和 CONV 网络实现高效的统一硬件加速而设计的 CNN Overlay 处理器, 需要同时支持 Zero-TCONV 和 NN-TCONV 运算. 为此, 软件编译的过程将网络配置

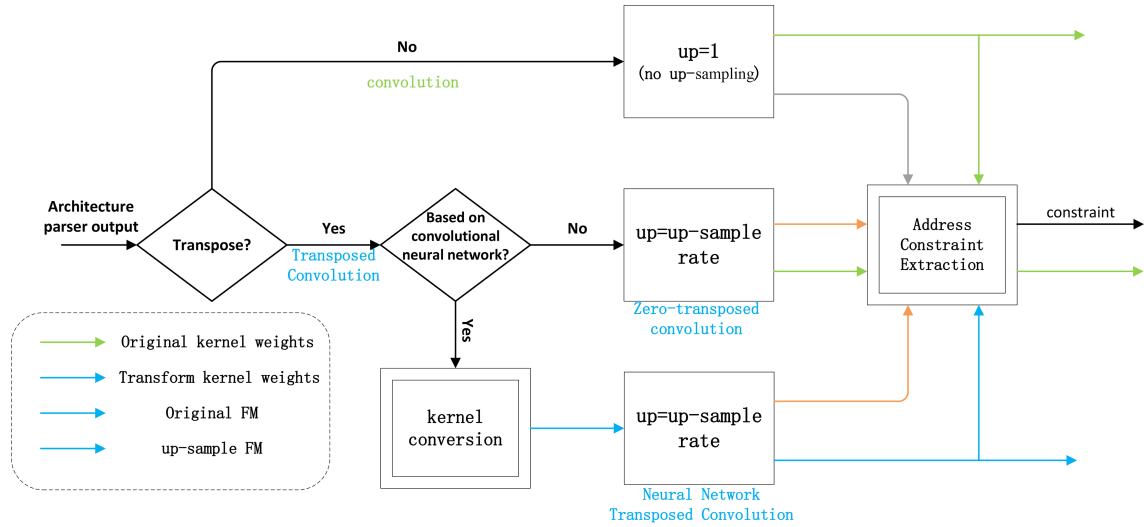


图7 (网络版彩图) Uni-OPU 编译流程中的网络重构流程。

Figure 7 (Color online) Network reconfiguration in compiling flow for Uni-OPU.

文件转换为处理器可执行的指令序列。和 Light-OPU 类似, Uni-OPU 的编译器主要实现的也是网络重构和硬件映射两个目标。Light-OPU 在网络重构时主要考虑的是算子融合, 而 Uni-OPU 在编译器的设计过程中引入了更多的优化, 其中最重要的就是算子重排。算子重排要求重新制定计算数据和操作的顺序, 旨在达到两个目的: (1) 减少 TCONV 中的运算低效问题; (2) 统一 3 种不同层次的计算模式, 即 NN-TCONV 层、Zero-TCONV 层和 CONV 层计算。

Uni-OPU 编译流程中的网络重构流程如图 7 所示, 主要包含两个功能模块: 卷积核转换和地址约束提取。卷积核转换对卷积核权重进行预处理, 以减少冗余操作。地址约束提取则计算出控制统一计算模式所需的约束集合。

具体来说, Uni-OPU 的编译器首先根据架构解析结果检查当前层是否为 TCONV 层。如果当前层是普通的 CONV 层, Uni-OPU 将通过上路径并基于该层的原始卷积核权重和原始特征图进行地址约束提取。如果当前层是 TCONV 层, Uni-OPU 将进一步决定其具体类型。针对 Zero-TCONV 层, Uni-OPU 将在其原始卷积核权重和上采样特征图上进行地址约束提取。针对 NN-TCONV 层, Uni-OPU 将首先进行卷积核转换, 计算所有可预加的权重组合, 并形成新的卷积核权重集合。然后, 再对转换后的卷积核权重和上采样特征图进行地址约束提取。最终, 计算重构的输出结果是一组地址约束参数和转换后的卷积核权重, 这些参数和权重可以直接应用于原始特征图。

5.4.3 Uni-OPU 编译流程中的卷积核转换

卷积核转换的核心是对权重进行预加, 以跳过上采样过程并减少计算需求。这需要识别一个卷积核的所有可预加组合。一种直接的方法是在上采样后的特征图上应用卷积核, 并记录所有出现的组合。然而, 这种方法会产生重复的组合, 因此效率不高。相反, Uni-OPU 考虑创建一个大小为 $up \times up$ 的辅助卷积核, 并将其滑动过原始的卷积核权重。在一个卷积核窗口内的所有权重可以形成一个可预加组合, 这样的搜索时间仅需 $16/36 = 44\%$ 。然后, 将每个提取的组合中的元素进行预加, 以创建转换后的卷积核权重。这两个步骤可以合并为一个卷积操作, 其中原始卷积核权重作为输入特征图, 卷积核则是一个所有元素都为 1, 尺寸为 $up \times up$ 的辅助卷积核。卷积的输出特征图即为重构后的卷积核。

生成某一层新的卷积核权重 $kernew$ 的过程在算法 1 中进行了描述。假设 C_{in} 和 C_{out} 分别表示输入和输出通道的数量。从原始卷积核集合中提取每个 2-D 卷积核, 并对其进行辅助卷积。填充大小设置为 $up - 1$ 。得到的 2-D 输出将作为转换后卷积核权重的一部分插入到 $kernew$ 中。

算法 1 Uni-OPU 中的卷积核转换.

```

1: Function KERNEL CONVERSION ( $\text{ker}_{\text{ori}}, \text{up}$ );
2:  $\text{ker}_{\text{new}} \leftarrow \text{empty}$ ,  $\text{aux}_{\text{ker}} \leftarrow \text{ones}(\text{up}, \text{up})$ ,  $\text{pad} \leftarrow \text{up} - 1$ ;
3: for  $i \leftarrow 1, C_{\text{out}}$  do
4:   for  $j \leftarrow 1, C_{\text{in}}$  do
5:      $\text{tmp\_ker} = \text{CONV}(\text{aux}_{\text{ker}}, \text{ker}_{\text{ori}}, \text{pad})$ ;
6:      $\text{ker}_{\text{new}}.\text{push\_back}(\text{tmp\_ker})$ ;
7:   end for
8: end for

```

算法 2 计算模式统一化算法.

```

1: Function COMPUTATION ( $\text{fm}, \text{ker}$ );
2: for  $i \leftarrow 0, (C_{\text{out}}/P_{\text{out}}) - 1$  do
3:    $\triangleright P_{\text{out}}$  parallelism;
4:   for  $j \leftarrow 0, (C_{\text{in}}/P_{\text{in}}) - 1$  do
5:      $\triangleright P_{\text{in}}$  parallelism;
6:     for  $k \leftarrow 0, K_x \times K_y - 1$  do
7:        $\text{cur}_k \leftarrow \text{ker}[k, j, i]$ ,  $x_0 = x_{os}[k]$ ,  $y_0 = y_{os}[k]$ ;
8:       for  $x_i \leftarrow (x_{is}[k], x_{ie}[k])$  with step  $s_{ix}$  do
9:         for  $y_i \leftarrow (y_{is}[k], y_{ie}[k])$  with step  $s_{iy}$  do
10:           $\text{out}[x_0, y_0, i] += \text{input}[x_i, y_i, j] \times \text{cur}_k$ ;
11:           $x_0 += s_{ox}$ ,  $x_0 = (x_0 > x_{oe}) ? x_{oe} : x_0$ ;
12:           $y_0 += s_{oy}$ ,  $y_0 = (y_0 > y_{oe}) ? y_{oe} : y_0$ ;
13:        end for
14:      end for
15:    end for
16:  end for
17: end for
End Function

```

5.4.4 Uni-OPU 编译流程中的地址约束提取

在完成卷积核转换后, Uni-OPU 的编译器将卷积操作 (CONV) 和零/最近邻转置卷积 (Zero/NN-TCNV) 转换为相同的计算模式, 具体过程在算法 2 中进行了描述. 该计算模式与输入/输出通道并行. 对于一组输入特征图 (FM) 和卷积核权重, Uni-OPU 提取每个卷积核权重 $\text{ker}[k, j, i]$, 并计算其与输入特征图中选定元素的乘积. 元素的选择基于起始和结束地址的约束, 以及步幅. 每个乘积结果会被加到相应的输出地址, 这一地址可以通过输出特征图的起始和结束约束以及步幅进行计算. 如果卷积核经过重构, 可以将每个卷积核元素在二维卷积核中的位置 (与 C_{in} 和 C_{out} 无关) 映射回原始卷积核, 从而完成地址约束提取的过程.

5.5 混合精度卷积神经网络的 Overlay 处理器设计

在卷积神经网络推理中, 低精量化已被证明有效地降低了计算复杂性和带宽需求. 混合精度 CNN 能够在保持准确度的同时受益于低精度. MP-OPU^[58] 是一个混合精度的卷积神经网络的 Overlay 处理器, 充分利用了混合精度在传统和轻量级 CNN 中的优势. MP-OPU 的微架构考虑了混合精度权重和激活值的计算核心共享, 结合编译器的调度以提升计算效率.

如图 8(a) 所示, 每个处理单元 (PE) 具有多个计算通道以实现并行计算, 并且有一个通道组合模块负责根据不同的并行模式组合或选择计算结果. 每个计算通道由混合精度乘法器 (mixed-precision multiplier, MPM) 阵列和加法树组成. 为了充分利用 FPGA 上的资源, MP-OPU 中使用 DSP 切片来实现混合精度乘法器, 并将一个 DSP 切片分解为多个低精度乘法器, 以提高资源使用效率. 由于卷积

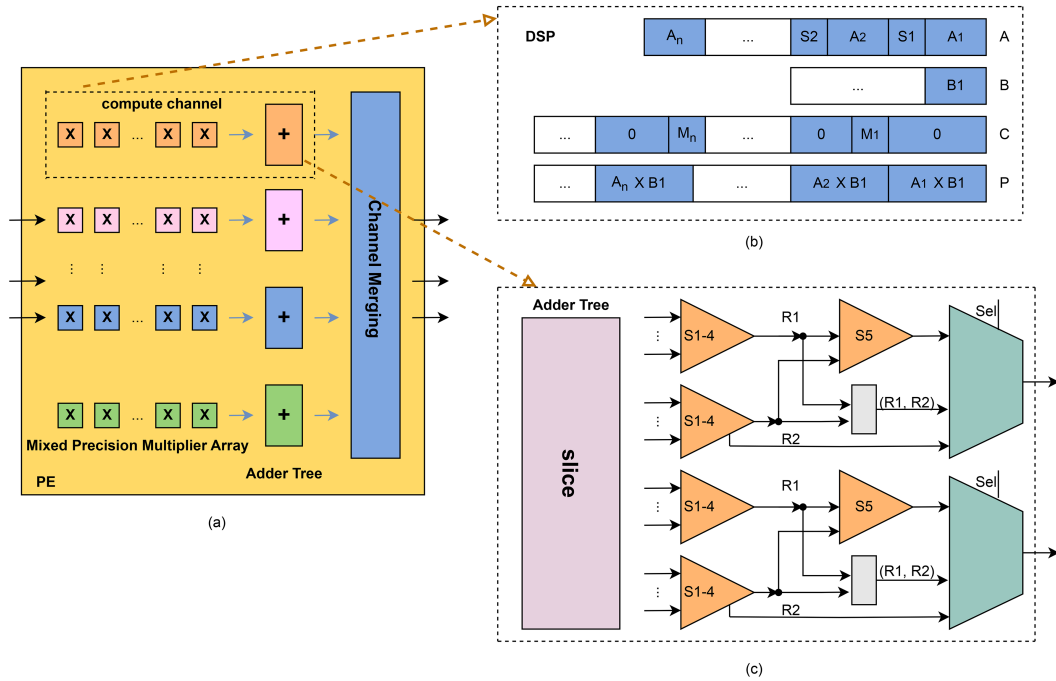


图 8 (网络版彩图) MP-OPU 中混合精度的计算单元设计. (a) MP-OPU 中的 PE 设计; (b) MP-OPU 中的 DSP 设计; (c) MP-OPU 中的加法器树设计.

Figure 8 (Color online) Computation unit for mixed-precision in MP-OPU. (a) PE architecture in MP-OPU; (b) DSP architecture in MP-OPU; (c) adder-tree architecture in MP-OPU.

神经网络 (CNN) 中的权重和激活值通常采用不同的位宽量化, MP-OPU 提出了分解规则, 以支持乘数和被乘数的不同位宽. 如图 8(b) 所示, 混合精度乘法过程可以共享相同的乘法器 B_1 , 并使用 n 个被乘数 (A_1, A_2 到 A_n) 将 n 个乘法器紧凑地集成到一个 DSP 中. 具体来说, 每个 DSP 被配置为执行乘法加法操作 ($P = A \times B + C$), 并根据不同的精度组合将不同的操作数 (A, B 和 C) 适配到乘法加法器上. 由于低位有效位的有符号乘法可能影响高位有效位, MP-OPU 为每个低精度乘法器引入了一个修正位, 在探索了所有 A_i 和 B_1 的组合后得出第 i 个修正位的计算公式:

$$M_i = \begin{cases} \text{Sign}(B_1) \oplus \text{Sign}(A_i), & B_1 \neq 0 \text{ and } A_i \neq 0, \\ 0, & \text{otherwise,} \end{cases} \quad (3)$$

其中, $\text{Sign}(x)$ 表示数据 x 的符号, $\oplus$ 表示异或操作. 通过这种方式, 只需要将 P 切分成 n 个部分就可以获得每个低精度乘法器的乘积. 在 MP-OPU 的每个计算通道中, DSP 的数量被固定配置. 由于每个 DSP 能够分解为多个低精度乘法器, MPM 阵列的输出数量取决于乘法器的分解程度. 因此, 加法树的后续设计具备可编程性, 具体通过参数寄存器配置, 如图 8(c) 所示. 为确保加法树能够处理不同位宽的输入并保持运算精度, MP-OPU 对加法树结构进行了精细设计. 首先, 对每个输入信号进行位宽切片和符号扩展, 保证数据对齐. 随后, 这些对齐后的数据输入至四级加法树进行分层处理. 最终, 两个四级加法树的输出结果或进一步累加, 或依据特定计算模式串联操作. 此外, 加法树的某些阶段可根据计算需求进行旁路配置, 从而优化深度卷积运算的效率与资源利用率.

为了加速卷积神经网络 (CNN) 的计算, MP-OPU 中使用了四级并行处理, 如图 9 所示. 对于输出通道并行性, 需要使用一个激活值去乘以不同的卷积核, 生成不同输出通道的不同激活值, 如图 9(a) 所示. 在输入特征图并行性 (如图 9(b) 所示) 中, MP-OPU 将计算同一输入特征图中不同激活值与对应卷积核的卷积, 以产生一个输出特征图的激活值. 对于输入通道并行性 (如图 9(c) 所示), MP-OPU 将不同输入通道中的激活值与相应卷积核并行计算乘法, 然后求和, 得出一个输出通道的结果 (或中

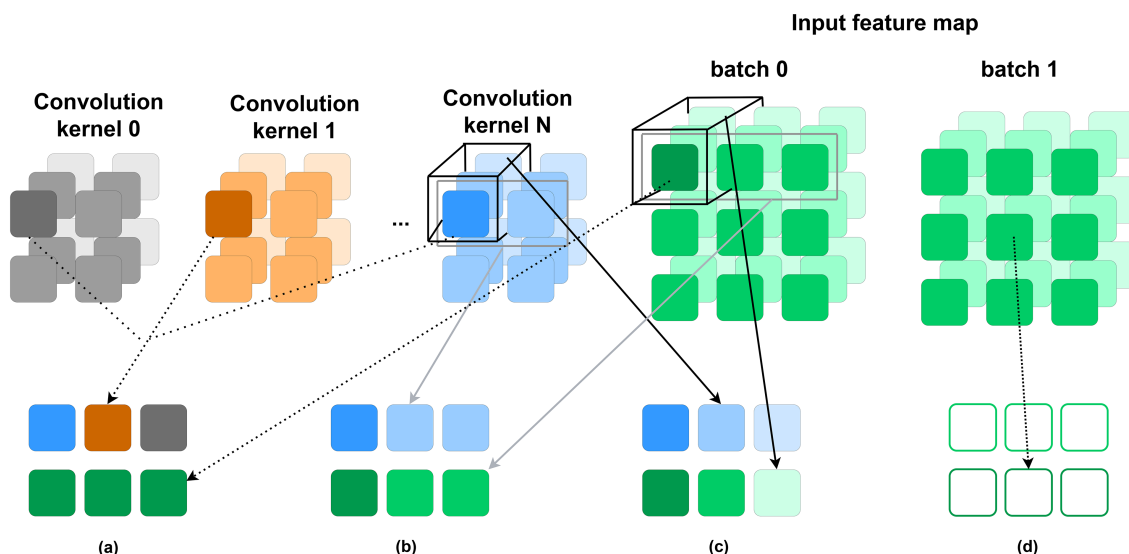


图 9 (网络版彩图) MP-OPU 中的并行机制. (a) 输出通道并行; (b) 输入特征图数据内并行; (c) 输入通道并行; (d) 批并行.

Figure 9 (Color online) Parallelism in MP-OPU. (a) Output channel parallelism; (b) input feature map parallelism; (c) input channel parallelism; (d) batch parallelism.

间结果). 批处理并行性将并行计算来自不同输入图像的激活值. 这 4 个级别的并行性是根据不同层的类型和参数进行选择的. 例如, 传统卷积层将在每个计算通道中使用输入通道并行性, 并在每个 PE 中的不同计算通道之间使用输出通道并行性. 由于在深度卷积层中, 每个输出特征图通道是通过对一个卷积核和一个输入特征图通道进行卷积生成的, 因此 MP-OPU 在每个计算通道中利用输入特征图并行性, 并在每个 PE 中的不同计算通道之间利用了输出通道并行性. 为了简化设计, 将所有 PE 设置为独立, 仅在不同 PE 之间进行批处理并行性.

5.6 实验数据对比

本节展示了当前基于 FPGA 的 CNN Overlay 处理器的推理数据, 包括工作频率、吞吐量、查找表、触发器、数字信号处理器和块随机存取存储器的资源占用情况, 如表 2 所示. 4 种处理器架构 (OPU, Light-OPU, Uni-OPU 和 MP-OPU) 在不同 FPGA 平台上的实现效果各具特色. 其中, OPU 在 XC7K325T 平台实现最高单 DSP 吞吐量 (0.48 GOPS), 资源占用适中; Light-OPU 在相同平台下吞吐量较低 (0.21 GOPS), 但查找表资源消耗更多; Uni-OPU 在 XC7Z100 平台表现出较大的 DSP 和 BRAM 使用量; MP-OPU 则在 XC7VX690T 平台展现出平衡的双吞吐量性能 (0.38/0.29 GOPS), 但资源消耗最高.

从整体性能来看, OPU 在资源效率和吞吐量上取得了最佳平衡, 适合通用场景; Light-OPU 虽然性能较低但可能更适合资源受限的应用场景; Uni-OPU 的大量 DSP 和 BRAM 使用表明其更适合复杂的计算任务; 而 MP-OPU 的双吞吐量设计则显示出其在并行处理方面的优势, 适合高并发的应用场景, 尽管这需要更多的硬件资源支持.

6 Transformer 网络的 Overlay 处理器设计

Overlay 处理器已被应用于多种类型的模型和网络中, 除了传统的 CNN 网络以外, 目前也有一些对 Transformer 网络进行加速的 Overlay 处理器设计, 例如文献 [21, 41, 64~69]. 本节将对基于 Transformer 网络的 Overlay 处理器进行详细介绍和总结, 内容具体分为以下几部分: 基本运算单元

表 2 基于 FPGA 的 CNN Overlay 处理器实验数据.

Table 2 Experimental results on FPGA-based Overlay processors for CNNs.

	Device	Frequency (MHz)	Throughput/DSP (GOPS)	LUT	FF	DSP	BRAM
OPU (2019) [52]	XC7K325T	200	0.48	129927	233996	817	165
Light-OPU (2020) [56]	XC7K325T	200	0.21	173522	241175	704	193.5
Uni-OPU (2020) [57]	XC7Z100	200	N/A	117971	247238	1987	494.5
MP-OPU (2021) [58]	XC7VX690T	200	0.38/0.29	278548	324033	3072	912

设计、非线性函数近似单元设计、存储系统设计、多核的 Transformer Overlay 处理器设计、稀疏的 Transformer 大模型 Overlay 处理器设计,最后对现有的 Transformer 网络 Overlay 处理器的实验数据进行对比.

6.1 基本运算单元设计

Transformer Overlay 处理器的基本运算单元负责进行网络中最主要的运算,即矩阵乘法的计算. 现有的基本运算单元有多种基本结构,包括基于加法树的结构,基于脉动阵列的结构,和其他特殊结构. 例如, DIF-LUT [70] 提出了可扩展的非线性激活函数近似方法,以及 PreVV [71] 提出的基于提前值验证的数据流优化方案. 本节对这些基本运算单元的结构进行总结和分类.

6.1.1 基于加法树的运算单元设计

文献 [21, 41, 64~66, 68] 的基本运算单元,都是基于 PE 阵列 + 可变加法树的结构. 可变加法树是指加法树在第几级进行输出是可控制的,或加法树的分叉方式是可以控制的,从而可以适配不同的输出数量需求.

具体来说,在 NPE, HPTA, DFX 和 MCore-OPU 中采用了基于加法树的高效并行处理架构,如图 10 所示. 每个 PE 单元包含定点乘法器和局部累加器,通过 SIMD 方式并行执行乘-累加运算,专门优化了矩阵乘法等计算密集型任务. 这些 PE 单元的输出通过多级加法树网络进行数据归约,采用对数深度设计显著降低了关键路径延迟. 加法树的每级加法器都采用流水线设计,在固定时钟频率下工作,保证了稳定的数据吞吐量. 加法树的可变特性通过选择性激活不同层级的输出节点来实现 64 到 2 的输出规模调整. 这种设计在降低硬件复杂度的同时,也带来了输出灵活性的权衡. 由于所有 PE 持续处于启用状态,系统能保持最大计算密度,但输出规模只能在预设档位间切换,限制了对不同规模任务的适应性. 每个 PE 的计算结果需要经过固定的归约路径,这种确定性路由虽然简化了控制逻辑,但也影响了系统处理不规则计算模式的能力.

相比之下,Transformer-OPU 和 LTrans-OPU 中的 PE 阵列采用了动态可配置机制,如图 11 所示. 这种架构通过顶部的配置寄存器接收并解析计算指令,实现了细粒度的 PE 阵列控制. 数据分发器根据指令配置动态调整数据路由策略,不仅控制 PE 的启用状态,还能优化数据在加法树网络中的流动路径.

具体实现上,系统采用了双状态控制机制:首先是 PE 级别的动态配置,通过状态寄存器和数据分发器的协同工作,实现 PE 单元的选择性激活. 其次是加法树网络层面的路径配置,通过标记加法器节点的有效性状态,构建最优的数据归约路径. 这种多层次的配置机制使系统能够根据实际计算需求,灵活调整计算资源的分配和数据流的组织方式.

从实现效果看,该架构在计算密度和灵活性之间实现了新的平衡:虽然动态配置会导致部分 PE 处于闲置状态,降低了硬件利用率,但系统获得了更细粒度的输出控制能力. 通过移位寄存器的后处理,系统能够适应更多样化的输出需求,特别适合处理动态变化的计算模式. 这种取舍在处理 Transformer 等结构可变的深度学习模型时展现出明显优势.

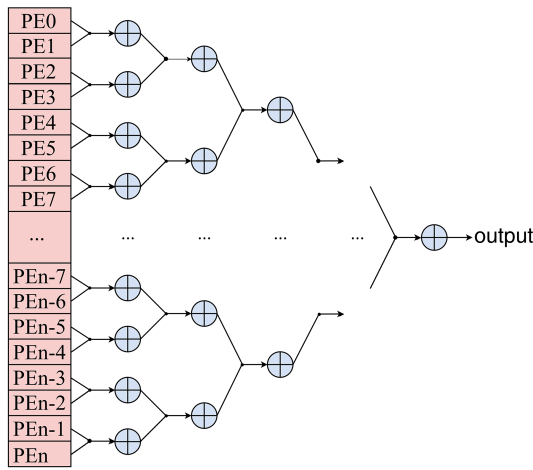


图 10 (网络版彩图) 传统的基于加法树的运算单元结构.

Figure 10 (Color online) Computational architecture with traditional add tree.

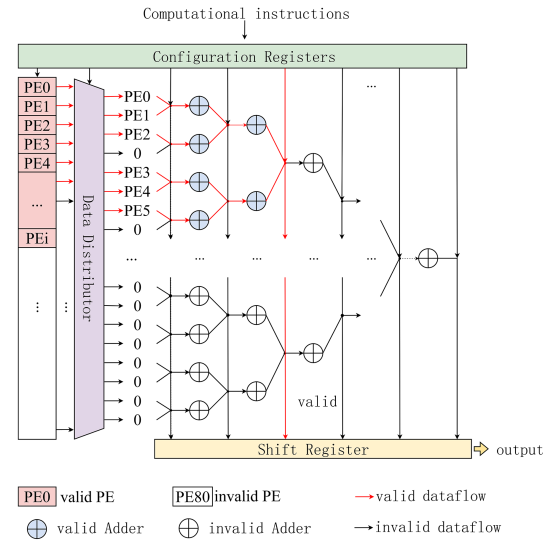


图 11 (网络版彩图) 具有可变 PE 阵列和加法树的运算单元结构.

Figure 11 (Color online) Computational architecture with reconfigurable PE array and add tree.

6.1.2 基于脉动阵列和其他类型的运算单元设计

ChatOPU^[69] 中的基本运算单元是基于脉动阵列结构的, 在这一结构下, 数据在时钟的驱动下在阵列中前向传播. ChatOPU 提出的脉动阵列结构利用了一种特殊的对角线数据流, 具体将在后续章节中详述.

FET-OPU^[67] 中的运算单元利用了改进的 dsp-packing 技术, 使得一个 PE 单元可以完成两个 8 bit 有符号乘法. 这一运算规则需要在两个乘法运算中保留一个公共乘数, 而基于加法树和基于脉动阵列的运算单元结构都不能满足这一要求. FET-OPU 设计了一种独特的矩阵乘法范式, 在不同轮次下, 计算左乘矩阵的不同列向量与右乘矩阵的不同行向量之间的乘积, 如图 12 所示. 在这种运算方式下, 不同轮次的矩阵运算可以共享相同的乘法器, 这符合 dsp-packing 技术的需求. FET-OPU 还对其运算单元进行了优化, 加入了额外的符号修正逻辑和溢出补偿逻辑, 解决了传统 dsp-packing 技术^[72]中, 无法进行两个有符号 8 bit 数乘法和数据溢出的问题. FET-OPU 中的每个 PE 还连接了两个 FIFO, 在缓存输出数据的同时, 可以进行数据拆分和融合, 数据转置等操作.

除了上述特征, 一些特殊结构和配置下的 Transformer Overlay 处理器的运算单元还有一些其他的设计和优化, 例如针对多核^[68]、多板^[41]、稀疏化大模型^[69]的 Transformer Overlay 处理器. 对于这些 Overlay 处理器的运算单元细节会在后面的章节进行详述.

6.2 非线性函数近似单元的设计

Transformer 网络中除了矩阵乘法运算外, 还有一系列如 gelu, softmax 和 layernorm 的非线性运算. 然而这些非线性函数的运算逻辑较为复杂, 难以在 FPGA 上直接实现. 同时, 由于这些非线性函数嵌入在变压器网络的基本块中, 它们的低效率成为一个重要的性能瓶颈^[73], 因此需要对这些非线性函数进行近似. 现有的非线性函数近似方法, 主要包括使用泰勒 (Taylor) 展开, 稀疏化或分段线性函数进行近似. 在这些方法中, 基于分段线性函数近似的方法能够在性能精度和复杂度上进行最佳的权衡.

NPE 使用了一种通过手工的非均匀分段找到最佳近似的方法. 主要分为 3 个手工步骤: 确定分

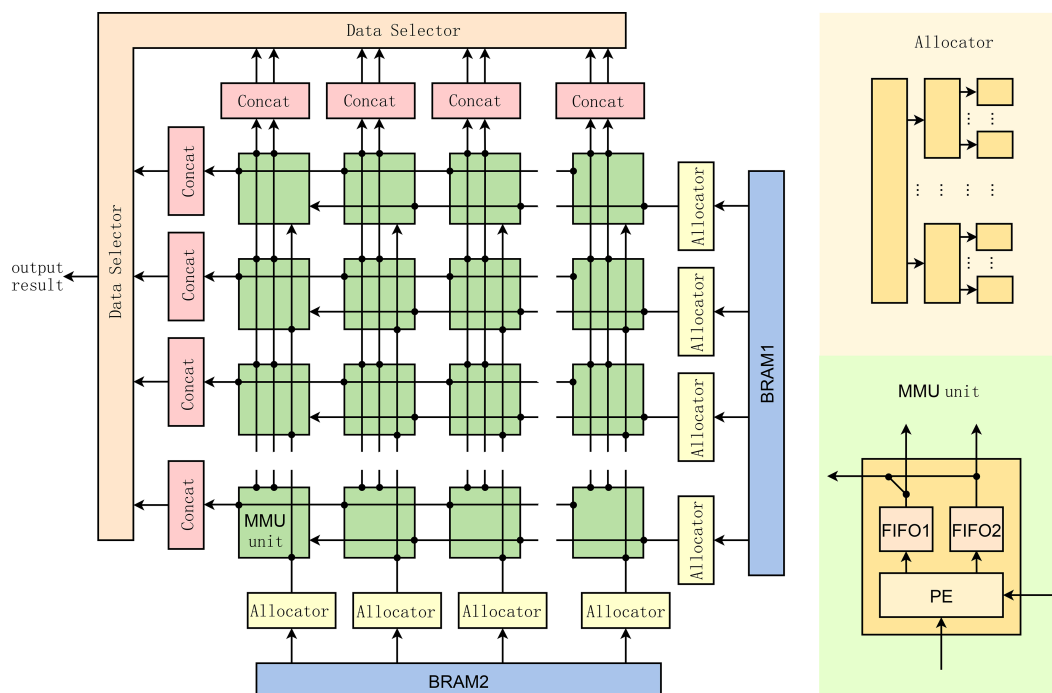


图 12 (网络版彩图) FET-OPU 中利用 DSP-packing 的运算单元结构.

Figure 12 (Color online) Computation unit with DSP-packing in FET-OPU.

段区间、确认区间函数斜率和确认区间函数截距. NPE 中提到, 即使是次优的分段方式, 也不会导致推理精度的明显损失. 同时, NPE 为 Transformer 网络中的非线性函数设计了专门的非线性向量处理单元, 支持 16、32 和 64 位操作, 以便在中间计算中允许更高的位精度.

NPE 中的手工分段线性近似方法取得了较好的效果, 但这一方法需要手工定义复杂的优化规则和仔细的调优. LTrans-OPU 和 FET-OPU 使用一种基于简单神经网络的自动分段方法, 如图 13 所示. 这一方法基于 NN-LUT^[74] 中的思想, 即认为一个带有 relu 操作的单层神经网络, 可以表示一个分段近似函数. 图 13(a) 部分展示了一个单层神经网络结构, 包含输入节点 x 和输出节点 $f(x)$, 中间通过 N 个隐藏节点 (n_1 至 n_N) 连接, 每个连接都有对应的权重参数 k 和偏置参数 b . 通过训练这一网络的参数, 可以直接获取最优的分段区间和各段斜率, 即最优的分段方法. 基于这一方法, 如图 13(b) 所示, 任意单输入的非线性函数可以转化为分段线性函数, 以便通过查找表 (look-up table, LUT) 直接实现. 输入 x 首先经过定位单元进行处理, 然后通过查找表 (LUT) 获取对应的参数值 (k_1-k_4 和 b_1-b_4), 最终通过乘法和加法运算得到函数输出值 $f(x)$. 这种设计不仅实现了对非线性函数的有效近似, 还能通过硬件结构实现高效的计算处理, 为 Transformer 网络中的非线性运算提供了一种实用的优化方案.

6.3 存储系统的设计

在 Transformer-OPU, LTrans-OPU 和 FET-OPU 中, 参数数据被分为两类: 大规模数据和小规模数据. 大规模数据包括输入数据和线性层参数, 这些参数从 CPU 通过 PCIE 接口发送到片外 DDR, 并动态传递到片上存储器. 小规模数据包括非线性参数和量化参数, 这些参数直接从 CPU 预写到片上存储器.

HPTA 提出了一个高效的存储子系统. HPTA 的通用主存储器中实现了 3 个具有相同宽度和深度的带缓冲区的 RAM 模块, 以提供 PE 不同的使用数据并易于重用, 从而允许更有效地使用内存. 同时主存储器仲裁器使用多路复用器来管理 RAM 的数据和地址接口, 以仲裁对 RAM 的访问, 并利用乒乓存储结构, 确保数据可以高效且无冲突地传输, 如图 14 所示.

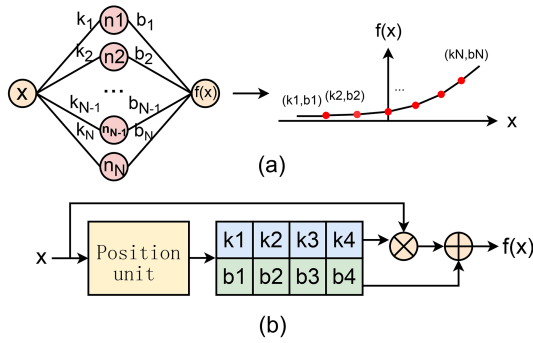


图 13 (网络版彩图) 基于神经网络自动确定分段线性近似方法的流程图. (a) 基于神经网络确定分段区间和斜率的示意图; (b) 基于 LUT 查找表的分段线性函数硬件实现方法.

Figure 13 (Color online) Dataflow of NN-based piecewise linear approximation. (a) NN-based piecewise intervals and slopes determination; (b) LUT-based hardware implementation for piecewise function.

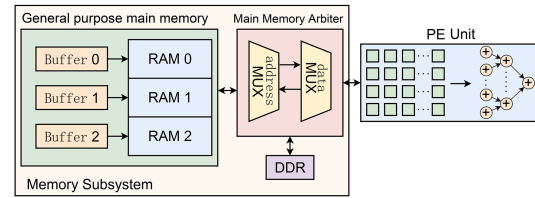


图 14 (网络版彩图) HPTA 中高效存储子系统细节图.
Figure 14 (Color online) Details of the efficient memory system in HPTA.

MCore-OPU 提出了一种简单而有效的张量溢出策略, 以解决来自 KV 缓存的累积的片上内存压力. 具体来说, MCore-OPU 将早期 KV 张量固定在片上存储器上, 并根据后进先出 (LIFO) 规则溢出后期 KV 张量, 从而节省早期 KV 张量的片外内存流量.

6.4 多核的 Transformer Overlay 处理器设计

在现有的基于 FPGA 的 Transformer 加速方案中, 吞吐量受到缓慢的片外存储器、逻辑结构上的复杂路由以及跨芯片的长数据路径的限制. 多种优化设计方案被提出, 例如, 灵活高效的处理器架构^[67]、低延迟的 Overlay 处理器设计^[66], 以及充分利用 FPGA 资源的部署方案^[75]. 其中吞吐量的一个重要的限制是 FPGA 上的 IP 的使用情况不佳, 其峰值频率没有得到很好的利用. 同时, 在 U200 等面向数据中心的大容量 FPGA 中, DDR 和 DSP 分散在不同的芯片上, 因此多核设计或许能取得更优的性能.

为此, MCore-OPU 提出了一种基于 FPGA 的多核 Overlay 处理器, 以优化基于 Transformer 网络的核内计算和核间通信. 该工作在 PE 阵列和其他部分之间分离时钟域以提高 DSP 频率, 同时在 FPGA 上实例化了多个小内核, 而不是单个大内核, 通过移位寄存器和级联 dsp 减少 IP 布线, 以确保芯片关键路径不会限制 DSP 的工作频率.

需要注意的是, 多核系统在带来了更快的计算速度的同时, 带来了更高的同步开销, 特别是对片外存储器的访问. 因此, MCore-OPU 提出了一个快速的片上同步数据路径, 以减少昂贵的片外内存流量, 同时将同步过程纳入流水线中, 从而实现非线性函数的计算效率的保留. 同时, MCore-OPU 提出一种特殊的度量方法, 以找到一种核内计算效率最大、核间通信代价最小的分配方案.

在指令集层面, MCore-OPU 从 OPU 指令集中, 增加了两个新的指令功能, 分别控制内核和 DDR 之间片外存储器访问的地址, 以及配置同步路由信息.

6.4.1 双频计算引擎

MCore-OPU 提出了一种双频计算引擎, 将规则的 PE 阵列的时钟域, 与其他不规则结构部分的时钟域分离, 以避免对 DSP 部分的频率约束. 如图 15 所示, 具体来说, MCore-OPU 通过将深度减少一半并将 bank 扩展两次来重塑 PE 数组的缓冲区. 因此, 数据可以由 bank 以单倍频率顺序写入缓冲区, 并由 PE 阵列以两倍频率并行地从所有 bank 中读取. 上述方案保证了输入存储器带宽能够匹配 PE

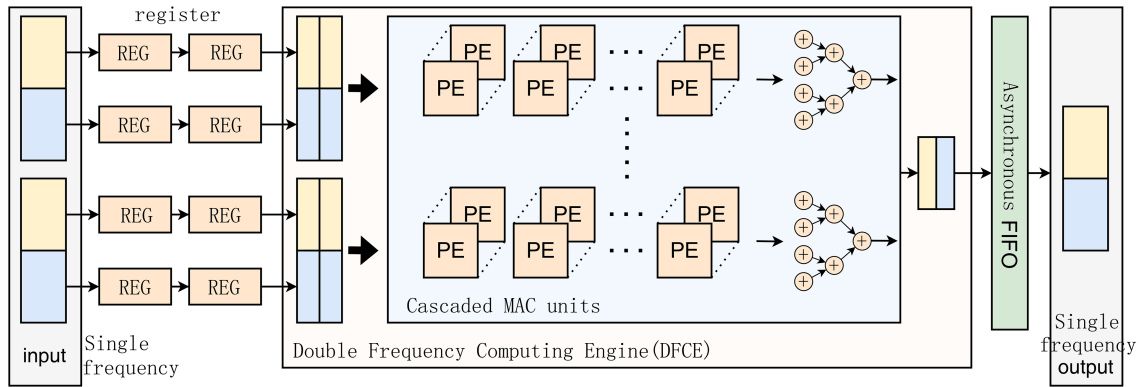


图 15 (网络版彩图) 基于级联 MAC 单元的双频计算引擎示意图.

Figure 15 (Color online) Dual-frequency computation engine based on cascaded MAC unit.

阵列双倍的工作频率, 并保持双倍的吞吐量. 在 PE 阵列架构中, MCore-OPU 没有使用 MAC 单元的深度加法器树来限制 FPGA 上的工作频率, 而是使用带有小加法器树的级联 MAC 单元来实现高工作频率. 从高频域到低频域的转换由异步 FIFO 实现.

6.4.2 片上同步路由

由于计算核心通常放置在 FPGA 上的不同超逻辑区域 (SLRs) 上, 且跨 SLRs 间的超长线路专用寄存器有限, 因此必须限制路由间的链路设计. 图 16 展示了系统的流水线执行时序, 以横向时间轴和纵向核心编号构建坐标系, 呈现了从核心 1 到核心 n 的完整处理流程. 每个核心遵循“本地输入 – 本地累加 – 同步累加”的处理顺序, 采用错时方式执行: 当核心 1 进入本地累加阶段时, 核心 2 才开始其本地输入, 依此类推, 形成规律的阶梯状执行结构. 这种设计将同步操作整合到计算单元的流水线中, 采用了错时并行的方式进行处理. 每个核心的执行过程都遵循相同的模式, 但通过时间上的错开执行, 使得在任一时刻, 不同核心都处于不同的处理阶段, 从而形成了一个高效的流水线结构.

MCore-OPU 对带有 128 位数据链路的路由器采用环形拓扑结构, 以确保跨 SLR 链路不会超过限制, 并使每个路由可以与它的左右相邻的路由实现通信. 同时, MCore-OPU 将同步作为计算单元流水线的一部分, 防止流水线被频繁打断带来的性能损失, 如图 16 所示. 这种错时并行的处理方案确保在任一时刻, 不同核心都处于不同的处理阶段, 从而形成高效的流水线结构, 实现核心间的协同工作.

6.5 稀疏的 Transformer 大模型 Overlay 处理器设计

针对 Transformer 模型的稀疏性特征, 研究者们提出了多种优化方案, 包括充分利用 FFN 和注意力机制稀疏性的方法^[76]、基于固定长度 token 剪枝的优化^[77]、利用 MOE 稀疏性的设计^[78]、支持 N:M 稀疏模式的架构^[79], 以及可扩展的注意力机制加速框架^[80]和模式剪枝方法^[81].

先前的基于 FPGA 的对大语言模型的加速工作, 往往倾向于为特定的结构化剪枝定制 PE 结构, 而很少利用非结构化剪枝. 这是由于非结构化剪枝不规则的稀疏模式, 难以利用数据重用, 因此难以充分利用非结构化稀疏性带来的参数减少.

而非结构化剪枝带来的参数压缩效果, 其实往往能够超过结构化剪枝带来的影响. 因此对于具有非结构化稀疏性的大语言模型, 通过精细的 PE 结构和稀疏编码设计, 来平衡编码和解码阶段之间的计算和内存效率, 以获得整体效率是很重要的. Chat-OPU 为了解决这一困难应运而生.

6.5.1 对角线数据流

在用传统的脉动阵列进行稀疏计算时, 对非结构化稀疏数据的压缩破坏了原始数据中的规则行和列, 从而导致数据重用性较差.

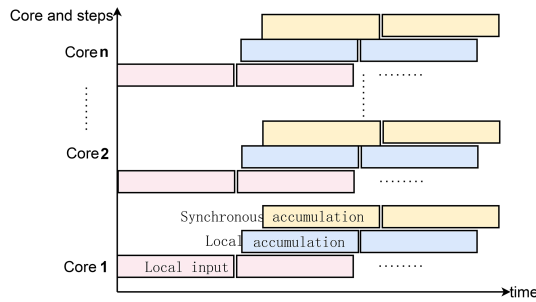


图 16 (网络版彩图) 包含同步步骤的流水线设计思想示意图。

Figure 16 (Color online) Pipeline design with synchronization.

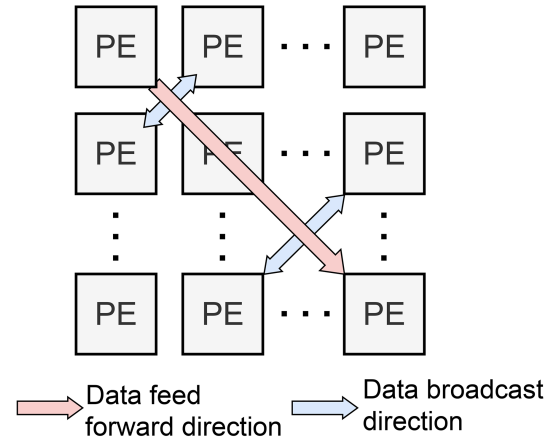


图 17 (网络版彩图) 基于对角线数据流的脉动阵列示意图。

Figure 17 (Color online) Systolic array with diagonal dataflow.

因此, Chat-OPU 提出了一种新颖的对角线数据流, 这一数据流可以在脉动阵列上实现高效数据重用, 同时用于稀疏和密集矩阵乘法, 如图 17 所示. 具体来说, 在 Chat-OPU 中, 所有输入从左上角一起送入到脉动阵列, 并沿着对角线转发到右下角. 对于每一步, 输入沿着与前馈方向正交的 PE 进行广播. 通过这种方式, 输入在所有 PE 之间共享, 而不是在单行 PE 之间共享, 以便更好地重用, 而且没有延迟开销.

为了最大限度地提高计算效率, Chat-OPU 还提出了一个有效的编码和解码方案. 在编码阶段, 通过提前用位掩码对非零元素位置进行编码, 以最大限度地减少片外内存通信开销. 在解码阶段, Chat-OPU 开发了一个高效的硬件解码器, 使在运行时能够获得数据原始矩阵中的位置和其他指标.

6.5.2 存储并重用 KV 张量

每个大语言模型的解码步骤都需要前一步的查询张量 K 和值张量 V 来计算交叉注意力. 这意味着 KV 张量的有效范围相互重叠, 并将随着推理过程的深入, 不断增加片上存储器的压力. 一旦片上存储器不能容纳所有的 KV 张量, 处理器必须将它们溢出到片外存储器中, 并在未来的步骤中重新加载, 从而导致昂贵的片外存储器流量.

因此, Chat-OPU 将早期的 KV 张量固定在片上存储器中, 并根据后进先出 (last-in first-out, LIFO) 规则溢出后期 KV 张量, 早期的 KV 张量不会被溢出, 以减少重新加载存储器的开销. 此外, Chat-OPU 将 KV 张量的溢出和重新加载以及写入片上存储器的操作合并在一起, 以提高片外带宽利用率.

6.6 其他大模型加速器

当前研究主要围绕计算架构创新、内存优化、编译框架升级三个维度展开, 通过算法-硬件协同设计突破传统 GPU 的性能瓶颈. 在计算架构方面, 可配置稀疏计算与混合精度量化成为核心突破方向. 例如, FlightLLM^[28] 提出动态可重构的稀疏 DSP 链, 支持非结构化稀疏模式 (如块稀疏、N:M 稀疏), 在 75% 稀疏度下实现计算效率提升 1.6 倍, 同时通过长度自适应编译技术将指令存储开销降低为原来的 1/500, 可动态适配 2048 输入长度的复杂场景; EDGE-LLM^[82] 则设计对数尺度结构化稀疏引擎, 结合 FP16*INT4 混合精度单元, 在 ChatGLM2-6B 模型上达到 1.67 倍于 A100 GPU 的吞吐量, 边缘设备能效提升 7.4 倍. 此外, SECDA-LLM^[83] 开发块浮点 (block floating point, BFP) 量化加速器, 通过超块向量处理器实现 3.5 比特/权重的压缩率, 在边缘 FPGA 上获得 11 倍延迟优化.

表 3 基于 FPGA 的 Transformer Overlay 处理器的实验数据.

Table 3 Experimental results of FPGA-based Overlay processor for transformer.

Design	Device	Core Num.	Freq. (MHz)	Latency (ms)	Throughput (GOPS)	DSP	BRAM	Power (W)
NPE (2021) ^[64]	Zynq Z-7100	1	200	13.96	808.01	2020	369	20
Transformer-OPU (2023) ^[65]	Xilinx Alveo U200	1	200	9.63	1171.31	3953	1113	N/A
LTrans-OPU (2023) ^[66]	Xilinx Alveo U200	1	200	4.51	2501.07	6144	1129	N/A
FET-OPU (2023) ^[67]	Xilinx Alveo U200	1	200	6.82	1653.76	4864	1357	7.39
HPTA (2023) ^[21]	Xilinx ZCU102	1	200	6.09	1852.19	2307	345	5
DFX (2022) ^[41]	Xilinx Alveo U280	4	200	4.82	2340.21	3533	1192	45
MCore-OPU (2024) ^[68]	Xilinx Alveo U200	4	300/600	11.74	1783.32	2316	892.5	N/A
ChatOPU (2024) ^[69]	Xilinx Alveo U200	1	300	23.14	487.46	1961	416.5	N/A

在内存与数据流优化领域,研究者聚焦于降低片外访问开销与提升带宽利用率. FlightLLM 的 Always-On-Chip 解码方案通过混合精度激活缓存,将解码阶段带宽利用率从 35.6% 提升至 65.9%; DFX 框架利用 HBM (high-bandwidth memory) 全通道优化与定制数据流,将 GPT-2 单令牌处理延迟压缩至 75.45 ms,速度超越 GPU 5.58 倍. 针对大规模模型部署,多 FPGA 分布式内存架构展现出潜力: FPGA 空间加速框架通过张量并行模型,在 Alveo U280 集群上实现 BERT 模型 13.4 倍加速, GPT-2 解码能效提升 5.7 倍,验证了千亿参数模型在 FPGA 集群部署的可行性.

编译与部署技术的智能化升级显著降低开发门槛. FlexRun 构建模块化指令集架构,支持动态重构计算单元参数,结合设计空间探索算法,使 BERT/GPT2 推理速度超越 V100 GPU 2.69 倍; SECDA-LLM 集成 SystemC 仿真与 HLS 协同设计流程,实现端到端 LLM 加速器快速原型验证. 异构协同加速方案(如 EDGE-LLM 的 CPU-FPGA 任务划分策略)进一步优化端到端性能,通过 CPU 处理动态编译与轻量算子, FPGA 专注密集计算,编译延迟降低 20%.

FlightLLM 在 Versal VHK158 FPGA 上实现 1.2 倍于 A100 的吞吐量,能效达 GPU 6 倍; 四 FPGA 集群 (DFX) 相比同规模 V100 GPU 集群,能效提升 3.99 倍且成本效益提高 8.21 倍. 在边缘场景, SECDA-LLM 于 PYNQ-Z1 设备部署 TinyLlama 模型,单令牌生成延迟仅 1.7 s,较 ARM CPU 加速 11 倍,验证了资源受限场景的实用性. 当前研究趋势呈现稀疏化-量化深度融合、编译系统智能化、多 FPGA 可扩展性三大特征,这些进展为 FPGA 在实时交互、边缘推理等场景替代 GPU 提供了技术基础,同时也为千亿参数 LLM 的高效部署开辟了新路径.

6.7 实验数据对比

本节对目前现有的基于 FPGA 的 Transformer Overlay 处理器的推理数据进行展示. 包括工作核心数、工作频率、推理时延、吞吐量、DSP 和 BRAM 资源占用、功耗情况等,如表 3 所示,以供参考和对比. 表中展示了 8 种 FPGA-based Transformer Overlay 处理器的设计实现数据. LTrans-OPU 和 DFX 性能最为出色,吞吐量分别达到 2501.07 和 2340.21 GOPS,时延均控制在 5 ms 内. 从资源使用看, LTrans-OPU 使用最多 DSP (6144) 但获得最佳性能, ChatOPU 则以最少资源 (DSP: 1961, BRAM: 416.5) 实现基础功能.

这些设计反映了不同的优化策略. 一部分设计(如 DFX 和 MCore-OPU)通过多核心架构提升性能,另一些(如 HPTA)则专注于提高资源利用效率,还有的(如 ChatOPU)则着重于降低资源开销. 这种多样化的设计思路为不同应用场景提供了丰富的选择空间,展现了 FPGA 在 Transformer 处理器实现上的灵活性.

7 图神经网络的 Overlay 处理器设计

应用于多种类型的模型和网络中的 Overlay 处理器,除了在传统 CNN 和 Transformer 网络有出色的性能表现,在图神经网络领域也有卓越的贡献.本节详细介绍和总结了基于图神经网络 (GNN) 的 Overlay 处理器,内容具体分为以下几部分:面向推理 GNN Overlay 处理器、面向训练的 GNN Overlay 处理器设计,以及高度集成的 Overlay 处理器的基本架构,最后对现有的图神经网络 Overlay 处理器的实验数据进行对比.

7.1 面向推理的图神经网络的 Overlay 处理器设计

图卷积网络 (graph convolutional networks, GCNs) 能够高效地处理非欧几里得 (Euclidean) 图数据.然而,GCN 在计算和内存访问过程中存在显著的不规则性,这限制了传统神经网络加速器的应用效率.此外,现有的专用 GCN 加速器往往需要大量内存资源,使得它们难以在资源受限的边缘设备上实现.现有的研究有灵活的内存优化并行图挖掘处理器^[84],以及面向边缘渲染的内存效率优化方案^[85].作为 GCN 网络推理加速专用的 Overlay 处理器,LW-GCN^[86]设计核心聚焦于以下几个方面:软件层面通过高效的稀疏矩阵预处理和编译器设计优化网络推理性能;硬件层面采用了灵活的通用架构设计,结合优化的内存管理策略和计算资源调度机制,以实现推理过程中的高效数据处理与计算任务执行;LW-GCN 使用统一的方法进行稀疏矩阵乘法 (sparse matrix-multiplication, SpMM) 和一般矩阵乘法 (general matrix-multiplication, GEMM).

7.1.1 PCOO 压缩量化

在软件预处理阶段,LW-GCN 算法首先对输入数据进行压缩,然后将图卷积网络 (GCN) 的工作负载分配至不同的处理单元 (PEs).在数据压缩过程中,由于 GCN 中的邻接矩阵和第一层输入矩阵通常具有高度稀疏性,LW-GCN 通过仅处理有效非零信息来优化存储效率,降低计算的复杂度.LW-GCN 据此提出了数据包的概念,在设计中使用 PCOO 格式用于高效表示稀疏矩阵.

具体而言,LW-GCN 将一行中的所有元素视为一个数据包,每个非零元素 ($X_{i,j}$) 被格式化为按位格式.前两位包含每个非零元素的开始和结束的行信息,接下来的位表示有效位 (valid digits, VLD),用于区分注入的空元素.这三类信息充当数据包的头部.其余的位则作为有效载荷,包含每个非零元素的列信息和数值.LW-GCN 使用 $\log_2(T)$ 位表示 ($X_{i,j}$) 在块内的列位置 ($j \bmod T$).剩余的 (H) 位表示非零元素的值.这种数据表示方式意味着仅仅需要 $(3 + \log_2(T) + H)$ 位来表示稀疏矩阵中的每个非零元素.PCoo 压缩稀疏矩阵的具体过程如算法 3 所示.

LW-GCN 对比了常用压缩格式 (CSR/CSC/COO) 与 PCOO 的存储消耗,结果如表 4 所示.对于所有数据集,PCOO 格式在存储效率上与 CSR 和 CSC 相当,并且比 COO 更加高效.由于 LW-GCN 将 MM 操作视为与 SpMM 相同的操作,LW-GCN 将 MM 中的左侧稠密矩阵格式化,来适应统一的处理单元.稠密矩阵首先按常规方式存储,然后让所有行共享相同的列信息,数据存储格式就转化为 PCOO 格式.这样,LW-GCN 只需要额外的 $(3 + \log_2(T) + \text{Column_Size})$ 位来存储稠密矩阵的中间步骤.

为减少内存消耗,LW-GCN 对 GCN 的矩阵进行了量化处理,采用 16 位定点数 (SINT16) 量化特征和权重,4 位定点数 (SINT4) 量化稀疏矩阵的非零元素.为了确保计算过程中的精度,所有中间结果均以 32 位有符号定点数 (SINT32) 存储.LW-GCN 在多个数据集上对 GCN 和 GraphSAGE 的量化策略进行评估,实验发现在精度损失低于 0.2% 的情况下,LW-GCN 能够显著优化内存与性能.

7.1.2 外积切片

在分配与调度中为了减少内存消耗,LW-GCN 采用了外积切片方法,如图 18 所示.LW-GCN 将

算法 3 稀疏矩阵预处理算法.

Require: $X \in \mathbb{R}^{m \times n}$, T , K ;
 tiles, sor, eor, vld $\leftarrow []$, $T \times 4$, $T \times 2$, T ;
for $t \leftarrow 0$ to $n - 1$ by T **do**
 rows $\leftarrow []$ for $0 : K$;
 for $i \leftarrow 0$ to m **do**
 row $\leftarrow []$;
 for $j \leftarrow t : (t + T - 1)$ **do**
 if $X_{i,j} \neq 0$ **then**
 row.append($j \bmod T + \text{vld}$);
 end if
 end for
 row $\leftarrow [0]$ if row is empty else row;
 row[0] $\leftarrow \text{row}[0] + \text{sor}$;
 row[-1] $\leftarrow \text{row}[-1] + \text{eor}$;
 rows[$i \bmod K$].extend(row);
end for
end for
 Fill zeros until rows are rectangular;
 tiles.append(rows.transpose());
return tiles;

表 4 各数据集上的存储要求对比.

Table 4 Comparison of memory requirements of datasets.

Dataset	Row	Col	Non-zeros	CSR	CSC	COO	PCOO
Cora Features	2708	1433	49216	781 kb	810 kb	1.33 Mb	886 kb
Cora Edges	2708	2708	10556	207 kb	207 kb	296 kb	201 kb
CiteSeer Features	3327	3703	105,165	1.74 Mb	1.74 Mb	2.94 Mb	2.00 Mb
CiteSeer Edges	3327	3327	9104	192 kb	192 kb	255 kb	173 kb
PubMed Features	19717	500	105165	13.2 Mb	13.2 Mb	18.3 Mb	15.8 Mb
PubMed Edges	19717	19717	9104	202 kb	202 kb	301 kb	195 kb

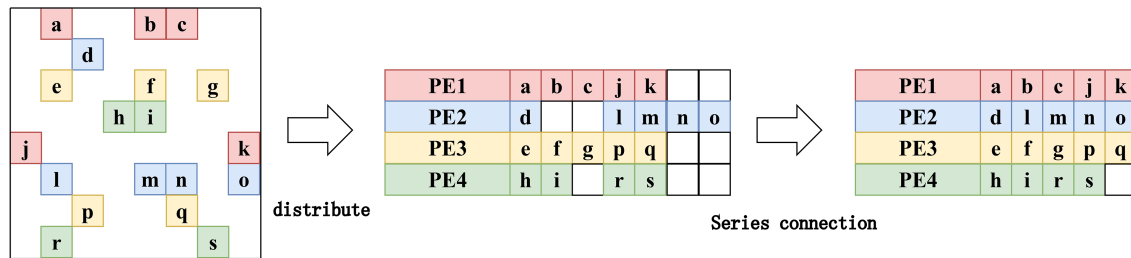


图 18 (网络版彩图) 计算引擎的循环分配设计.

Figure 18 (Color online) Loop allocation for computation engine.

输入数据划分为 X 的 T 列切片和 W 的 T 行切片. 硬件每次处理一对切片, 最后累加所有切片结果来生成整体计算结果. 对于每对切片, LW-GCN 执行算法 4 所示的预处理步骤, 以平衡工作负载并减少数据量.

如图 18 所示, LW-GCN 在稀疏矩阵 X 的乘法中, 将每行非零元素分配给同一个 PE, 并通过轮询分配不同行任务避免累加器的冲突. 由于图的邻接矩阵中不同行的稠密度差异可能过大, 大多数 PE 可能在完成较稀疏行的计算后进入闲置状态. 为此, LW-GCN 采用串联行分配策略, 即每个 PE 在

算法 4 预处理算法流程.

Require: $\text{tile} \in \mathbb{R}^{N \times K}$, $\text{depth } d = 16$;
 $\text{used}, \text{row} \leftarrow [0 \text{ for } 0 : K], [-1 \text{ for } 0 : K]$;
 $\text{result}, \text{share}, \text{block}, j \leftarrow [], \{\}, \{\}, 0$;
while $\text{sum}(\text{used}) < N \times T$ **do**
 $i \leftarrow \text{used}_j$;
 if $\text{tile}_{i,j} \in \text{share}$ or $\text{tile}_{i,j} \bmod d \notin \text{block}$ **then**
 $\text{row}_j \leftarrow \text{tile}_{i,j}$;
 $\text{share.append}(\text{tile}_{i,j})$;
 $\text{block.append}(\text{tile}_{i,j} \bmod d)$;
 $\text{used}_j \leftarrow \text{used}_j + 1$;
 else
 $\text{row}_j \leftarrow 0$;
 end if
 if $\min(\text{row}) \neq -1$ **then**
 $\text{result.append}(\text{row})$;
 $\text{row}, \text{share}, \text{block} \leftarrow [-1 \text{ for } 0 : K], \{\}, \{\}$;
 end if
 $j \leftarrow (j + 1) \bmod K$;
end while
Fill zeros until the result is rectangular;
return result;

完成前一行的计算后立即开始处理下一行的任务, 以确保 PE 连续工作, 减少闲置时间. 由于矩阵行的稠密度与其行号之间不存在显著相关性, 依据大数法则, 每个 PE 分配的行稠密度趋于均衡. 此外, LW-GCN 在必要时向串联行尾部注入空元素, 进一步平衡工作负载, 最大化 PE 效率.

考虑到片上内存容量限制, 权重矩阵 W 的多行信息存储在同一内存切片中, 且任意时刻只能访问一行数据, 稀疏矩阵 X 的多 PE 访问可能会引发数据冲突. 为此, LW-GCN 采用多内存系统和数据复制策略减少 RAW 数据冲突. 此外, LW-GCN 通过插入空元素 (VLD0) 进一步缓解了冲突. 但这样不可避免地增加了推理延迟和内存资源消耗, 同时多端口内存的实现也会加剧片上内存资源的消耗. 因此在设计中, LW-GCN 在内存使用与延迟之间进行了细致的优化平衡, 保障性能和资源利用的平衡.

预处理过程在算法 3 和 4 中总结为两个主要步骤, 其时间和空间复杂度与每个稀疏切片中的非零元素总数线性相关. 稠密切片则被量化为 SINT4 格式, 并保持数据结构不变传递至硬件. 预处理器最终生成指令, 实现各层和步骤的序列化执行.

7.1.3 基于数据复制的多端口矩阵存储单元设计

LW-GCN 的整体架构如图 19 所示, LW-GCN 的微架构由外围接口、外部内存接口、顶层控制、用于稀疏-密集矩阵乘法的 PE 数组和片上缓冲区组成. 顶层控制模块负责获取和解码指令, 然后将其传递给各个模块.

计算 LW-GCN 一个切片的整体工作流程如图 19 所示. 在初始加载步骤中, 密集输入数据从外部内存传输到密集数据内存 (high-density data memory, DDM). 在计算步骤中, 稀疏输入数据流入边权重内存 (EWM), PE 数组从 EWM 中获取 X 数据, 从 DDM 中获取 W 数据, 同时并行执行 MAC 操作. 中间结果将存储在输出矩阵内存备份 (output matrix memory backup, OMMB) 中, 并可在后续计算中视为左矩阵. 在完成每个聚合或组合步骤的所有切片对后, LW-GCN 将输出移动到 OMMB.

在 SpMM 过程中, 为了在不同 PE 上并行计算不同的行, 稀疏输入的多个非零元素在片上的存储位置是不固定的, 所以提前预测非零元素的列位置格外困难. 特别是, 可能会有多个 PE 需要来自同一 DDM 的不同地址. 而片上内存的读取能力有限 (双端口 RAM 仅支持从最多两个端口读取, 而 PE 数

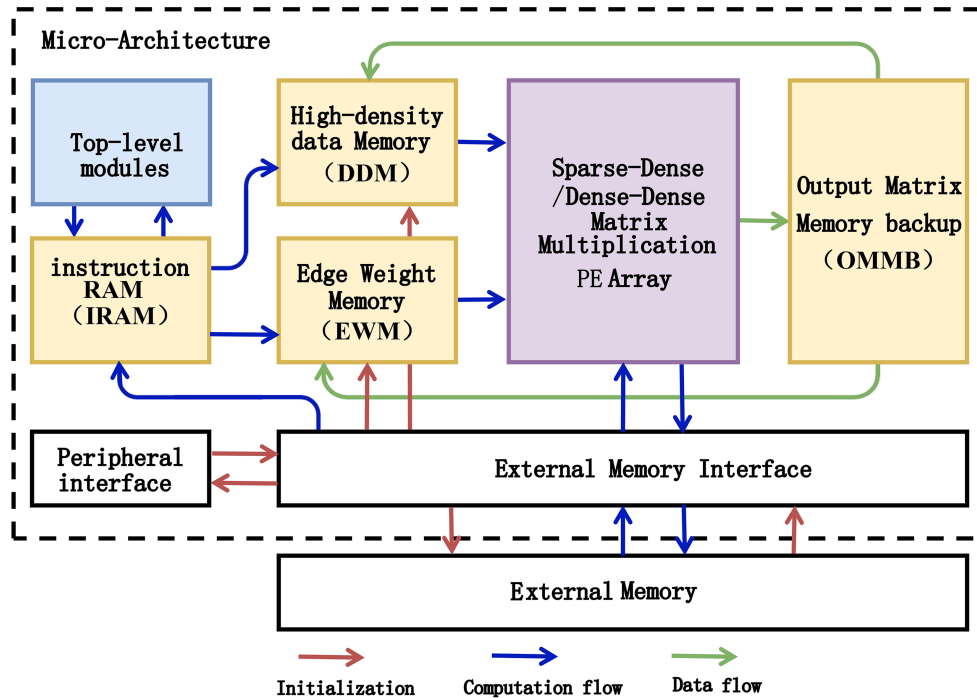


图 19 (网络版彩图) LW-GCN 的整体架构设计.
Figure 19 (Color online) Overall architecture of LW-GCN.

量通常大于两个), 访问的限制有可能导致 RAW 数据冲突. 在 LW-GCN 的微架构中, LW-GCN 通过数据复制和行分组构建了一个多端口内存, 来存储一个切片的权重, 从而减少 RAW 数据冲突.

此外, 在预处理过程中, LW-GCN 将密集数据复制为 r 个副本, 用于不同的内存切片, 进一步减少了 RAW 数据冲突的发生. 理想情况下, 当 r 等于 PE 的数量时, 每个 PE 都有一个专用的密集数据副本, 从而避免 RAW 数据冲突. 但这会导致较大的片上内存需求, 在现实中并不可行. 因此, LW-GCN 设定相对较小的 r , 以在可接受的资源利用率下解决部分 RAW 数据冲突, 并引入行分组以进一步减少 RAW 数据冲突的发生.

此外, LW-GCN 将每个密集数据副本划分为 n 个组, 并独立存储每个组, 其中每个组包含了 g 行数据. 具体而言, LW-GCN 将 W_j 存储在组 $(j \bmod g)$ 中, 这样 RAW 数据冲突仅可能发生在元素 X_{ia} , j_a 和 X_{ib} , j_b 之间, 发生冲突的条件为 $(j_a \bmod g) = (j_b \bmod g)$ 且 $j_a \neq j_b$. 与未分组内存相比, 显著降低了 RAW 冲突的发生概率. 尽管片上内存的深度需要达到最低要求以充分利用存储资源, LW-GCN 仍能够通过使用大量的行组在统计上降低 RAW 冲突的概率. 但较大的 g 值会增加数据分配给处理单元的复杂性, 从而导致布局布线过程的复杂化和资源消耗的增加. 数据复制和行分组都有效地减少了 RAW 数据冲突的概率. 对于剩余的冲突, LW-GCN 通过注入空元素并将其作为闲置周期进行处理来尽可能地避免.

7.1.4 基于优先解码器的调度单元设计

如图 20(a) 所示, PE 组的数量与内存的数量相对应, 确保每个 PE 组可以访问其相应的密集数据内存条, 进而避免 RAW 数据冲突. 内存选择器和数据分发器根据单个 PE 生成的地址调度适当的密集数据. 在分发地址到内存时, LW-GCN 采用优先解码器, 允许不同的 PE 能够从同一内存的相同地址获取数据.

需要强调的是, 数据复制仅适用于密集输入, 而不适用于稀疏输入. 如图 20(b) 所示, 压缩的稀疏数据直接流向每个 PE. 数据首先通过 PCOO 解码器, 其中 $\log_2(T)$ 位表示的列索引被解码为内存地

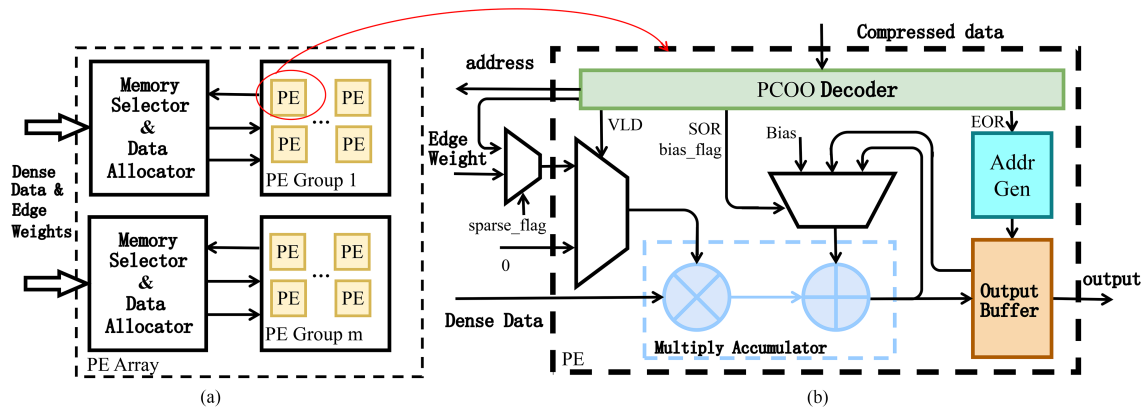


图 20 (网络版彩图) LW-GCN 中 (a) PE 阵列设计和 (b) 单个 PE 控制逻辑。

Figure 20 (Color online) (a) Architecture of PE array and (b) control unit of each PE in LW-GCN.

址以获取密集数据. 当有效位 ($VLD = 1$) 被检测到时, PE 将相应的值路由至其乘法器; 否则, 它将假定当前值为注入的空元素 (例如, 由于 RAW 数据冲突或等待其他 PE 完成), 并将 0 路由至乘法器. 由于多行被串联供 PE 使用, LW-GCN 使用开始行 (SOR) 和结束行 (EOR) 信号来指示行的起始和结束. 在每个计算步骤中, SOR 控制累加器的输入, 可以是其先前结果 ($SOR = 0$) 或保存在输出中间结果缓冲区 (OMMB) 中的前一个切片的中间结果 ($SOR = 1$). 与此同时, EOR 信号控制当前结果存储到输出缓冲区的地址生成, 并增加内部跟踪的行号 ($EOR = 1$).

矩阵乘法 (MM) 的计算在 PE 中遵循相同的工作流程. 由于左矩阵是密集的, 所有行共享相同的行和列信息. 随后, 稀疏标志信号指示所选的数据. 当 LW-GCN 处理 MM, 即 $sparse_flag = 0$ 时, 它将选择存储在边权重存储器 (edge weight memory, EWM) 中的边权重; 否则, 将选择从 PCOO 解码器解码的值. 通过这种方式, LW-GCN 能够在统一的 PE 中同时执行矩阵乘法 (MM) 和稀疏矩阵乘法 (SpMM), 从而提高 PE 在计算图卷积网络 (GCN) 组合和聚合时的工作效率.

7.2 面向训练的图神经网络的 Overlay 处理器设计

GNN 的训练与推理过程存在差异. 推理仅需前向传播生成预测, 而训练需同时进行前向传播和反向传播以计算梯度并更新模型参数. 其中, 反向传播通过前向传播计算输出和损失, 逐层反向计算梯度, 并通过优化算法更新参数, 因而需要额外存储中间结果, 导致训练计算复杂度和内存需求远高于推理过程.

为解决大型 GCN 工作负载与硬件约束的矛盾, SkeletonGCN^[87] 提出了一种基于 FPGA 的 GNN 加速器, 实现了软硬件的协同优化, 提高训练效率. 特别地, SkeletonGCN 通过 CPU 与 FPGA 调度控制充分利用不同硬件特性, 加速各种算子, 引入了一种适用于 GNN 训练的新型压缩格式和专用架构设计. 本节主要对 SkeletonGCN 的设计理念进行详细介绍.

7.2.1 统一 PE 架构

为提高计算效率, SkeletonGCN 提出了一种统一的 PE 架构以执行前向传播和反向传播, 如图 21(a) 所示. 该架构的主要操作包括稀疏-密集矩阵乘法 (SpMM)、密集矩阵乘法 (MM) 和转置密集矩阵乘法 (transposed matrix-multiplication, TMM), 其基本模块为乘加单元 (matrix-vector multiplication with caching, MACC). $M \times N$ 的 MACC 数组支持 SpMM, MM 和 TMM 的并行执行, 其中每一行的所有 MACC 单元共享相同的输入数据. 该架构高效地将数据输入到 MACC 中, 能够在不同的工作负载下实现良好的性能.

SkeletonGCN 采用密集的 PCOO 格式 (CPCOO) 对稀疏矩阵进行压缩, 并将压缩后的数据存储于

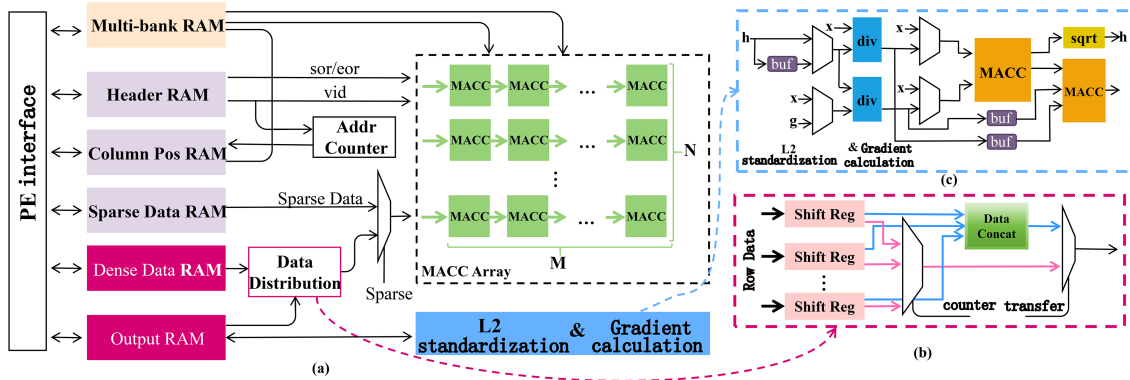


图 21 (网络版彩图) SkeletonGCN 整体架构. (a) SkeletonGCN 的统一 PE 架构, 前向传播和反向传播处理; (b) 内存加载和存储逻辑; (c) L2 标准化 & 梯度计算逻辑.

Figure 21 (Color online) Overall architecture of SkeletonGCN. (a) Unified PE architecture with forward and backward propagation; (b) memory load and control; (c) L2 normalization & computation of gradient.

3 个独立的 RAM 中, 如图 21(a) 所示. 计算时, 头部数据从头部 RAM 读取, 利用有效位标志 (VLD) 信号启用地址计数器, 生成主体 RAM 地址. 其中, 主体 RAM 提供非零元素的列位置以定位相应密集数据. 这种设计简化解压逻辑, 仅需几个导线和一个计数器控制. 非零稀疏数据和对应密集数据输入 MACC 数组, SOR 和 EOR 信号控制计算和结果保存. 完全流水线化架构使 MACC 单元大部分时间保持计算, 显著提高了 DSP 效率.

虽然 MM 和 TMM 共享计算操作, 但它们存在不同的内存访问需求. 为减轻负担并提高 DSP 效率, SkeletonGCN 设计了统一的内存加载和存储逻辑, 并增加数据分发模块来控制 MACC 数组的数据流, 如图 21(b) 所示. 同时, 片上内存宽度设为 $N \times N \times 16$ (SINT16), 保持每个周期输出活跃数据, 使 MACC 处于活动状态, 从而维持高 DSP 效率.

SkeletonGCN 的 L2 标准化及梯度计算模块将 MACC 数组的结果作为输入, 实现了计算的流水线处理. 由于该模块运算串行性, 大部分计算单元都获得了重用, 节省了资源利用率, 如图 21(c) 所示. 对于同时用于 L2 标准化及梯度计算的数据, SkeletonGCN 对其进行缓冲以消除冗余计算. 考虑到权重的更新, SkeletonGCN 设计独立模块从 PE 数组获取输出实现完全流水线化的计算, 并将 Adam 中的超参数设定为最接近的 2 的幂, 简化乘法运算为移位操作, 提高了计算效率.

7.2.2 数据分步通信

通信模块负责处理外部内存与 FPGA 之间以及不同 PE 之间的数据传输. CPU 与 FPGA 之间的接口采用 PCIe Gen3 X16, 仅传输初始数据、前向阶段的最终结果以及反向阶段的第一梯度. DDR4 也作为外部内存, 用于存储不同训练周期的初始数据. 片上缓冲区采用乒乓设计, 以有效隐藏 DDR4 与 FPGA 之间的通信延迟, 从而优化计算时间. 由于一个层的计算任务在不同 PE 上并行分配, SkeletonGCN 还需在各 PE 之间传输数据. 此外, SkeletonGCN 会收集一个层的所有数据, 并将其输入至权重更新模块, 以在反向传播阶段更新权重. 考虑到 FPGA 的特性, 即不同逻辑区域之间长连接数量的限制, SkeletonGCN 在 PE 接口模块中使用 FIFO 队列来控制不同 PE 之间的数据带宽.

对于 SpMM, MM 和 TMM, SkeletonGCN 使用轮询法将左矩阵的不同行分配给不同 PE 的不同行, 如图 22 中的简单示例所示. 通过这种方式, SkeletonGCN 就可以将非零元素的行信息隐藏在 MACC 数组的行索引下, 从而进一步优化 PCOO 数据压缩. 此外, 左矩阵中的元素逐个输入到 MACC (绿色箭头), 右矩阵的元素逐行输入到 PE (紫色箭头). 由于无法执行整个矩阵, SkeletonGCN 在左矩阵中进行行划分, 在右矩阵中进行列划分, 以适应 MACC 数组中的一个切片. 每个 PE 只能获得结果矩阵的一部分行. 当结果矩阵在后续步骤中用作左矩阵时, SkeletonGCN 将在每个 PE 内部传播结果, 减少

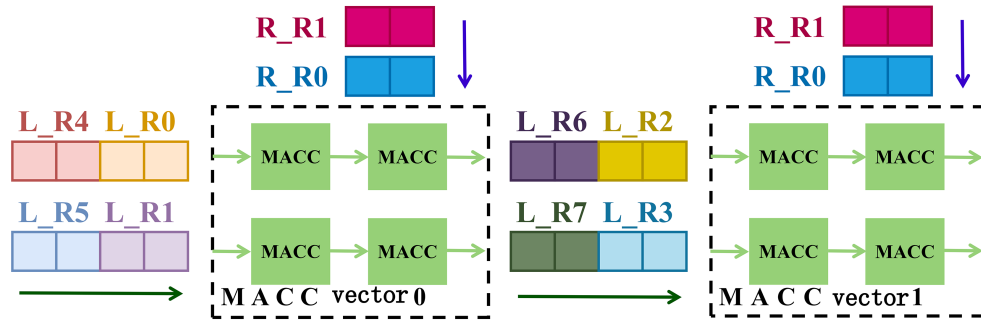


图 22 (网络版彩图) SkeletonGCN 中不同算子的不同执行流程.

Figure 22 (Color online) Computation flow for different operations in SkeletonGCN.

了在不同 PE 之间的通信.

7.2.3 工作负载分配

在前向和后向传播过程中, FPGA 执行稀疏-密集矩阵乘法 (SpMM)、密集矩阵乘法 (MM)、转置密集矩阵乘法 (TMM) 以及权重更新中的逐元素操作. 由于 softmax 函数和类别交叉熵损失在硬件实现上的计算成本较高, SkeletonGCN 将这些操作分配给 CPU. 同样地, 图采样和数据预处理任务也被指定给 CPU 处理.

在工作负载的分配中, SkeletonGCN 的整体架构为: PE 数组模块执行前向和后向传播阶段的所有操作, 而权重更新模块负责在反向传播完成后更新权重. 通信模块则管理 CPU, FPGA 与外部内存 (DDR4) 之间的数据传输, 以及不同 PE 之间的通信. SkeletonGCN 将子图的大小设置为合适的值, 以确保 FPGA 的片上内存能够处理子图的所有数据. 因此, FPGA 在训练一个子图的过程中只需访问外部内存三次: (1) 获取前向传递所需的初始数据; (2) 将最后一层的结果返回给 CPU 计算损失和梯度; (3) 获取用于反向传播的最后一层梯度.

在 GCN 的训练过程中, 计算开销较大的操作被分配给 FPGA, 而其他操作则由 CPU 处理. 为了提升整体训练性能, SkeletonGCN 实现了 CPU 和 FPGA 之间的大部分操作的并行处理. 在数据初始化的第一轮后, CPU 持续进行下一个周期的数据采样和预处理, 同时 FPGA 则执行当前周期的前向传播操作. 前向阶段完成后, CPU 被中断以传输前向传播的结果, 并计算 softmax 函数、损失以及相应的梯度. 计算完成后, 梯度被传回 FPGA 以进行反向传播, CPU 则重新开始为下一个周期准备数据. 此外, 由于子图之间是相互独立的, 多核 CPU 能够实现并行工作, 进一步提升了整体训练效率.

7.3 高度集成的 GNN Overlay 处理器设计

在图神经网络的 Overlay 处理器, 即 Graph-OPU^[88,89] 中, 数据执行路径由指令决定, 具备与多种 GNN 模型兼容的能力, 大大优化了传统加速器的单一固定数据路径的缺陷. 其中, GNN 操作根据 OPU 指令集架构 (instruction set, ISA) 框架以及额外参数的指引进行适配, 增强了指令执行机制, 提高了底层硬件的效率.

7.3.1 C/U-type 指令划分

与 OPU 类似, Graph-OPU 的设计将每条 32 位复杂指令分为条件指令 (C-type) 和无条件指令 (U-type). C-type 指令用于指定目标操作并设定触发条件, 而 U-type 指令为其配对的 C-type 指令提供相应的操作参数. Graph-OPU 继承了 OPU 指令集架构中的指令单元与块设计, 明确了每条指令的低位 (type) 和高位 (immediate, immi), 这一设计针对 GNN 中频繁的数据操作进行了保留, 确保能够提供更多操作细节, 同时最大程度上保持操作的连续性和完整性.

在 GNN 中, 矩阵操作通常导致指令块内出现大量重复指令, 而 GNN 的数据结构和操作具有不规则性, 例如在稀疏矩阵乘法 (SpMM) 和通用矩阵乘法 (GEMM) 之间的交替. 为了应对这种不规则性, Graph-OPU 采用 2 位操作码将 C-type 指令划分为四类: 内存访问、数据加载、计算和后处理, 分别对应不同的微架构模块. 指令在各自模块内执行, 而通过 4 位独热编码向量 (depen) 表示各模块间的依赖关系, 从而提高 Graph-OPU 操作中的并行性与执行效率.

7.3.2 基于参数控制的微架构任务管理

内存访问管理高带宽内存 (HBM) 与板载内存之间的数据传输, 以及从板载内存到 HBM 的反向传输. 内存管理模块通过识别模式位切换操作模式, 并通过数据类型标识数据源, 可以是稀疏数据、权重或其他类型. mini 参数控制 HBM 的 mini-switch 模块. 数据加载则将数据从板载内存移动到计算模块. 其操作模式可以通过调整数据复制和并行参数来定制, 以符合计算模块的数据需求.

7.3.3 IR 直接映射

Graph-OPU 利用 TVM 和 LLVM 将用户的模型文件 (.pth 或 .ONNX) 转换为 Graph-OPUIISA. 其中 TVM 使模型的计算步骤成型并有效地调度它们. 考虑到 Graph-OPU 架构, TVM 优化了计算以提高性能和能效. TVM 为 Graph-OPU 硬件生成 LLVM 中间表示代码 (IR), 将向量类型的 IR 代码匹配到微架构的计算引擎和内在函数, 从而确保高效的微架构支持. LLVM 使用 IR 和内在函数生成 GNN 模型的 Graph-OPU ISA 指令. IR 到 ISA 的转换采用直接映射. 如果没有匹配的内在函数, 指令将无法分割, 用户必须设计自定义方法以实现所需的操作. 例如, GAT 的 EFE 操作涉及两次顺序的乘法和加法, 这些操作通过使用初始结果作为后续操作的权重输入, 映射到 LLVM 内在函数上. 与通过重新配置进行模型切换相比, 基于 OPU-ISA 的编译器显著减少了时间消耗.

7.4 实验数据

本节对目前现有的基于 FPGA 的 Transformer Overlay 处理器的推理数据进行展示. 包括工作频率、查找表、触发器、BRAM 资源占用、DSP 和功耗情况等, 如表 5^[89~92] 所示. 表中呈现了 4 种 FPGA-based GNN Overlay 处理器的性能指标. Graph-OPU 表现出最强的模型兼容性, 能够支持从基础的 GCN 到复杂的 GraphSage 等多种模型架构. 在处理效率方面, 尽管各设计的资源占用差异较大 (如 DSP 使用量从 1048 到 8740 不等), 但 Graph-OPU 凭借其平衡的资源分配达到了最高的 DSP 效率 (0.157).

不同设计方案体现出独特的优化思路. FlowGNN 在保持较高工作频率 (300 MHz) 的同时, 显著降低了资源开销; 而 FP-GNN 则为支持 GAT 模型的特定需求, 增加了大量逻辑资源. 这些数据反映出 FPGA 实现 GNN 加速器时在通用性、性能和资源开销之间的不同权衡策略.

8 不同 Overlay 处理器的对比

针对前文提到的 CNN, Transformer 和 GCN 通用 Overlay 加速器的典型工作, 我们在架构、计算策略、存储策略、流水策略以及支持的神经网络方面进行对比, 如表 6 所示. 从中可以看到, OPU, Light-OPU, Uni-OPU, LPFP, MP-OPU 这些针对不同 CNN 加速的工作都使用乘法器阵列和加法树的核心计算架构, 配合全流水的后处理模块, 完成整体的计算. 这些工作使用相同的流水策略, 在核心计算模块和单个网络 block 内使用全流水计算, 而在不同的模块间和不同的 block 间串行计算. 另外, 这些工作主要根据不同的 CNN 的特征及数据表示形式 (INT8, FP8, 混合精度), 优化计算策略和存储策略. 在此基础上, NPE 首次扩展到对 BERT 的支持. 在架构上, 主要扩展了后处理模块, 使用 piecewise 的方式支持非线性函数的近似计算. 之后, FET-OPU 又在此基础上对 Transformer 及其扩展模型进行

表 5 与其他基于 FPGA 的端到端 Overlay GNN 加速器对比.

Table 5 Comparisons of FPGA-based Overlay processors for GNN.

	FPGA	Model	Freq. (MHz)	LUT	FF	BRAM	DSP	Efficiency (Throughput/DSPs)
GCN								
Light-GCN								
Graph-OPU (2024) [89]	Alveo-U50	GAT	225	475k	427k	927	2742	0.157
GIN								
GraphSage								
DeepBurning-GL (2020) [90]	Alveo-U50	GNN	200	671k	505k	403	5892	0.043
FP-GNN (2022) [91]	VCU128	GCN	225	717k	517k	1792	8192	0.126
		GAT		1068k	722k	1792	8740	
FlowGNN (2023) [92]	Alveo-U50	GCN	300	229k	192k	185	1048	N/A
		GAT		149k	134k	335	2488	
		gin		263k	166k	204	1741	

表 6 不同 Overlay 处理器的对比.

Table 6 Comparison of different OPUs.

Work	Architecture	Computational strategy	Storage strategy	Pipeline strategy	Support network
OPU (2019) [52]	MAC array, adder trees pipelined post-process off-chip/on-chip co-optimized memory	Two 8 bit MACs in one DSP, parallelisim in channel	Single DDR, ping-pong buffer, tile in channel	Fully pipelined in computation core and post-process modules, serial between modules, controlled by instructions, pipelined in block, serial between blocks	CNNs: VGG, YOLO
Light-OPU (2020) [56]					Lightweight CNNs: MobileNet, ShuffleNet, Xception
Uni-OPU (2020) [57]		Multiple low precision MACs in one DSP, parallelism in channel, pixel and batch	Two DDRs, ping-pong buffer, tile in channel and pixel		GAN, UNet SRCNN, FCN8s
LPFP (2021) [55]					CNNs: VGG ResNet, DenseNet
MP-OPU (2021) [58]					VGG, YOLO MobileNetV1/V2
NPE (2021) [64]	MAC array, adder trees non-linear post-process	8 bit/16 bit MAC in one DSP, instruction driven	Single DDR, ping-pong buffer tile in matrix	Parallel in instructions pipelined in core	BERT and RoBERTa, DistilBERT
FET-OPU (2023) [67]	Cascaded MM MAC array non-linear post-process	Two 8 bit MACs in DSP C/U-type instructions	Single DDR ping-pong BRAM		Transformer, ViT, Swin BERT, RoBERTa, GPT
MCore-OPU (2024) [68]	4 Cores and 4 DDRs dual frequency DSP, non-linear post-process	Multiple cores and clock domains dual frequency in DSP instruction control	4 DDRs, column-/row-wise tiling data allocation ping-pong Buffer	Instruction driven, pipelined in single core sync inserted in pipelines	Transformers, BERT, ViT GPT-2, LLaMA
ChatOPU (2024) [69]	Diagonal systolic array, off-chip/on-chip memory				OPT-350M OPT-1.3B
LW-GCN (2022) [86]	Cascaded MAC array for MM and SpMM	Distributed by groups for dense matrix, parallel between PEs	4 DDRs, co-optimized with on-chip ping-pong compressed sparse matrix	Parallel between PEs, pipelined in PE and block serial between blocks controlled by instructions	GCN GraphSAGE
SkeletonGCN (2022) [87]	buffer, PCOO format non-linear post-process				GCN GraphSAINT
Graph-OPU (2024) [89]	MAC array, selective adder trees, off-/on-chip memory non-linear post-process	Dynamic dataflow parallel between PEs, C-/U-type instructions	28 channel HBM optimized PCOO format for sparse matrix	Pipelined modules depen management dynamic dataflow	GCN, Light-GCN GAT, GIN GraphSAGE

了支持和优化. MCore-OPU 和 ChatOPU 则扩展了多核计算架构, 使用与核数相同的片外存储数量, 利用多核同步路由实现片外与片内, 不同计算核之间的数据交互. 与此同时, 这两项工作首次引入了时钟域分离, 使 DSP 工作在其他控制逻辑的两倍频下, 从而进一步提升计算效率. ChatOPU 中首次提出对角线的脉动阵列架构, 以提升稀疏化的大语言模型中矩阵乘法和稀疏矩阵乘法的数据复用, 从而提升计算效率. LW-GCN 和 SkeletonGCN 则是针对 GCN 的推理和训练做出优化. 两者使用类似的架构和策略, 其核心都是统一的密集矩阵乘法和稀疏矩阵乘法计算架构. 虽然 SkeletonGCN 针对 GCN 的训练过程, 但 GCN 反向传播过程的核心依旧为矩阵乘法, 所以 SkeletonGCN 在 LW-GCN 基础上扩展了后处理模块, 增加了 L2-Normalization 的非线性处理逻辑. Graph-OPU 进一步扩展了对不

同 GCN 网络推理的支持. Graph-OPU 使用选择性的加法树, 动态调整计算路径来支持稀疏矩阵乘法. 同时, 各个 PE 间并行计算, 由指令控制可动态实现并行度调整. 另外, Graph-OPU 首次使用 HBM 作为外部存储, 利用多通道访问的方式, 提升数据存取效率, 进一步提升整体计算性能.

更具体的, 我们针对 CNN, GCN 和 Transformer 通用 Overlay 加速器的典型工作, 在指令集 Overlay 率、算子粒度、编译流程、硬件效率、能效对比方面进行对比分析, 如表 7 所示. 在 CNN 加速器领域, 研究者通过多级指令集扩展和编译协同优化突破能效瓶颈. OPU 通过构建 Overlay 卷积操作的指令集 (加载/计算/存储事件), 结合粗粒度算子融合策略与分层切片编译技术, 实现 91% 的 MAC 效率, 较传统 GPU 提升 5.35 倍能效. Light-OPU 针对轻量化网络需求, 创新性设计深度可分离卷积专用指令, 通过输入/核内/特征图三级并行拆分与操作重排序编译策略, 在嵌入式平台实现 3 倍能效优势. Uni-OPU 通过统一卷积与转置卷积的块级计算模式, 配合网络切片编译优化, 使特定网络能效提升达 15.04 倍. 量化方面, 低精度浮点融合 (LPFP) 与动态混合精度 (MP-OPU) 分别通过编译时量化参数优化和数据重排策略, 实现 1.5~12.9 倍硬件效率提升.

针对图神经网络加速, 稀疏处理与编译优化协同成为关键. LW-GCN 首创 PCOO 稀疏压缩格式指令, 结合多路 MAC 阵列架构与编译时稀疏矩阵解码优化, 实现 912 倍能效提升. SkeletonGCN 扩展训练指令集和自动稀疏切片编译, 显著降低动态功耗. Graph-OPU 提出动态 SpMM/GEMM 切换指令, 配合 LLVM IR 映射的编译流程优化, 达成 5305 倍能效增益.

Transformer 加速器设计呈现多核架构与动态适配两大趋势. NPE 采用分段多项式近似指令和定点量化编译策略, 以 3 倍资源效率优势突破非线性函数计算瓶颈. FET-OPU 通过矩阵转置专用指令集和行级编译优化, 实现 231 倍能效提升. 在多核优化方向中, MCore-OPU 创新核间同步指令与编译时参数分配策略, 获得 5.81 倍 DSP 吞吐率优势; ChatOPU 则通过可配置脉动阵列架构支持混合数据流, 配合稀疏位掩码编译技术, 在 50% 稀疏度下实现 2.29 倍 GPU 吞吐率. 动态重构方面, ITLE 采用可重构脉动阵列指令, 结合循环展开编译优化, 使 MAC 利用率达 GPU 的 1.94 倍.

跨架构对比研究表明, 编译协同优化呈现显著领域特性: CNN 加速侧重算子融合与量化参数, GCN 加速依赖稀疏格式编解码优化, 而 Transformer 加速则通过多核调度与稀疏编码动态适配. 为了提高指令集覆盖率与编译灵活性, 未来研究需在动态指令扩展、异构计算单元协同编译等方向持续突破.

9 发展趋势与技术展望

9.1 发展趋势

随着 AI 应用的不断深入, 神经网络专用处理器的创新设计正成为推动技术进步的关键. 当前市场中, GPU 凭借其强大的通用计算能力占据主导地位, 而基于 FPGA 的 Overlay 处理器则以其独特的可重构性、能效等优势, 在推理加速、实时处理等特定应用场景中展现出独特价值. 回顾技术发展历程, 自 OPU^[52] 首次提出 Overlay 处理器架构以来, 这一技术已从最初面向 CNN 的单一加速器逐步发展为支持多种深度学习模型的通用平台. Light-OPU^[56] 和 Uni-OPU^[57] 的研究工作在轻量级网络优化和统一卷积处理方面实现了突破. 同时, 业界也在探索提高 FPGA 易用性的解决方案, 如 OverGen^[93] 通过领域特定覆盖生成框架提高了设计效率, Seiba^[94] 创新性地集成 HLS 电路与 Overlay 处理器以减少开发周期中的 FPGA 重编译次数. 然而, 随着深度学习模型的日益复杂化, 如何支持更复杂的网络结构并充分发挥硬件资源潜力, 成为了推动技术继续创新的重要动力.

深度学习模型的持续演进正推动 Overlay 处理器技术走向新的发展阶段. 目前, 研究重点主要聚焦于 3 个方向: 图神经网络 (GNN) 的性能提升、Transformer 的编译优化以及新兴的 Mamba 状态空间模型实现.

表 7 针对不同应用场景的 Overlay 处理器的对比.

Table 7 Comparison of OPUs for different network types.

Work	Type	ISA coverage	Operation granularity	Compiling flow	Hardware efficiency	Energy efficiency
OPU	CNN	Convolution, pooling, activation, control	Coarse (load/store, compute, post-process)	Unified IR, p-fusion/r-fusion, quantization	91% MAC efficiency on average	5.35× higher than Titan Xp
Light-OPU	CNN	Convolution, depthwise convolution, group convolution, channel reorganization	Parallelism in input/kernel/feature map for convolution and depthwise convolution	Computation blocks, operator fusion, and reorder for continuous layers	5.5× latency reduction and 3.0× better energy efficiency than Jetson TX2	1.3×–8.4× speedup than other FPGA accelerators
Uni-OPU	CNN	Zero-inserting-based, nearest-neighbor-resizing transposed convolution (zero-TCONV, NN-TCONV)	Block-wise CONV and TCONV layers	Unified CONV and TCONV, slicing, parallelism and scheduling	1.90× and 1.63× latency reduction than Titan Xp GPU on zero-TCONV and NN-TCONV	15.04× and 12.43× better energy efficiency than Titan Xp GPU on zero-TCONV and NN-TCONV
LPFP	CNN	Convolution, pooling, ReLU, fully-connected, and low-precision floating-point	Multiple operation units, parallel in multiple activations and weights	Fused convolution and continuous layers, low-precision floating-point quantization	1.5× high throughput than FPGA accelerators on average	39% to 5.16× better energy efficiency than FPGA accelerators
MP-OPU	CNN	Convolution, point-wise convolution, mixed precision	Dynamically optimize precision for different layers and operations	Allocate different precisions, operation fusion, slicing, and data reorder for different layers	12.9× latency reduction	2.2× per DSP throughput improvement than other FPGA accelerators
LW-GCN	GCN	Dense/sparse-dense matrix multiplication, PCOO format	Unified DMM and SDMM, reconfigurable MAC array	Compressed sparse matrix with PCOO format, INT4 quantization for input	60×, 12×, and 1.7× latency reduction than CPU, GPU, and FPGA accelerators, respectively	912×, 511×, and 3.87× higher energy efficiency than CPU, GPU, and FPGA accelerators
SkeletonGCN	GCN	Adam weight updates, L2 normalization, and compact PCOO during training	High efficient SpMM, DMM and transposed MM	Automated exploration of slicing and computation parameters	11.3× speedup than GraphACT	Power reduction using quantization and compression
Graph-OPU	GCN	SpMM, MM, batch normalization, selective adder tree	Unified computation engine for SpMM and GEMM	LLVM-based IR and mapping to SA instructions	1.36×, 1654×, 63× speedup than CPU, GPU, and FPGA	1.41×, 5305×, and 422× higher energy efficiency than CPU, GPU, and FPGA
NPE	Transformer	Matrix multiplication, non-linear operation, non-step, step, and index load/store operations	Piecewise approximation for multiple non-linear functions, highly pipelined	IR from NLP models, fixed point data for weights, and feature map	3× resource reduction on FPGA, with similar performance	4× and 6× power reduction than CPU and GPU
FET-OPU	Transformer	MM, addition, transpose, slicing, concatenation, DMA, non-linear	PE array with DSP packing, instruction controlled non-linear for accuracy and resource	Row-wise optimization, layer slicing, memory allocation, and parallelism	1.56×–8.21× speedup than FPGA and ASIC accelerators	231×, 5.85×–66.36×, and 5.28×–6.24× than CPU, GPU, and FPGA/ASIC accelerators
ITLE	Transformer	Reconfigurable PE array, row-column mask packing, buffer controller	Column-wise reconfigurable systolic array	Unified IR for computation graph and operation details, loop unrolling, and block allocation	1.94× MAC efficiency than GPU	1.16× higher normalized throughput than GPU
MCore-OPU	Transformer	ISA for sync between cores, matrix allocation, cross memory fetch	Independent or cooperated for different cores, balanced loads for different cores	Compiling for each core, parameterized multi-core allocation	1.24×–1.39× higher normalized throughput than FPGA	5.31×–5.81× higher normalized throughput than GPU
Chat OPU	Transformer	Encoding and decoding control array behavior, managing KV loading and popping	Diagonal data flow performs both sparse and dense matrix multiplications	Unstructured sparsity, and the positions of non-zero elements are encoded using bit masks	At 50% unstructured sparsity, OPT outperforms GPU by 2.29× in throughput	1.63× higher normalized throughput than FPGA accelerators

GNN. 在社交网络分析、推荐系统等领域得到了广泛应用,能够有效处理复杂的图结构数据,捕捉节点之间的关系,具备较强的表达能力.近年来,研究的重点逐渐转向如何提升图神经网络 (GNN) 在性能上的表现,特别是在大规模图数据处理和实时更新方面. LW-GCN^[86] 开创性地实现了 SpMM 和 MM 操作的统一处理,为提升资源利用效率提供了新思路. SkeletonGCN^[87] 在此基础上实现了训练推理的一体化支持,并通过创新的量化方案提升了系统效率. S-LGCN^[95] 通过软硬件协同设计在通用性和专用性之间取得了良好平衡. 针对数据流应用的架构模板^[96],通过参数化设计和轻量级数据包网络为这类不规则计算提供了新的实现思路. 在大规模稀疏图数据处理和动态图结构实时更新方面,改进数据调度策略和专用缓存结构设计将成为提升系统效率的关键.

Transformer. 在自然语言处理、机器翻译等任务中发挥着重要作用,其擅长处理长距离的依赖

关系,逐渐成为许多应用的基础架构.随着研究的深入,Transformer 在 FPGA 上的研究重点逐渐转向其编译和硬件加速性能的优化.HPTA 架构^[21]率先实现了高性能的 Transformer 加速,DFX 系统^[41]通过创新的多芯片互联方案突破了单芯片计算限制,而 ChatOPU^[69]引入的非结构化稀疏计算则为大规模模型部署开辟了新路径.而最新的研究成果主要集中在访存^[78,97]和稀疏方案^[79]的优化上.时分复用 FPGA 覆盖架构^[98]为解决大模型资源限制提供了新思路,通过更高效地共享有限硬件资源支持复杂计算.尽管超大规模模型的内存带宽瓶颈和注意力计算的二次复杂度问题仍待解决,但优化内存访问模式和探索高效的稀疏计算方案正在为这些挑战提供新的解决思路.

Mamba.作为序列建模的新范式,其线性复杂度的选择性状态空间机制正在为硬件加速开创新的可能.其主要应用于时序预测和序列建模任务,它凭借其线性复杂度的选择性状态空间机制,在硬件加速领域展现出独特的优势,尤其适合处理大规模时序数据.作为一项新兴技术,Mamba 在 FPGA 平台上的实现仍处于探索阶段.状态空间计算中的数据依赖制约了并行度的提升,选择性机制的硬件实现和计算资源的动态调度也面临着优化空间.针对这些问题,业界正在探索流水线架构以缓解数据依赖,尝试通过脉动阵列提升并行性能,并致力于优化资源分配策略.粗粒度 FPGA Overlay 处理器^[99]的即时编译特性有望为 Mamba 模型的快速迭代优化提供支持.这些技术创新正推动 Mamba 模型向更高效的实现迈进.

9.2 技术展望

Overlay 处理器在多个应用领域展现出独特优势.在边缘计算中,凭借其可重构性和低资源消耗,它非常适合部署在智能终端,支持本地化推理,实现实时决策和响应.而在工业物联网中,这些处理器则作为现场数据处理单元,能够高效支持设备状态监测和时序预测等任务,为数据分析提供强大支持.此外,Overlay 处理器在自动驾驶中的表现也十分突出,它能够融合处理来自视觉、激光雷达等不同传感器的数据,提升车辆的感知和决策能力.在移动计算场景下,凭借其出色的计算性能和低功耗特性,提供流畅的用户体验.同时,随着神经网络算法的不断进步,这些处理器在生物信息处理和智慧城市等新兴领域也展现了巨大的潜力.

纵观 FPGA Overlay 处理器的技术演进过程,其发展路线始终紧跟时兴的神经网络发展趋势.从早期的 CNN 处理^[52,56,57,100],到应对 GNN 的不规则计算^[86,87,95],再到支持 Transformer 的大规模并行^[78,79,97]和探索 Mamba 的新型架构,Overlay 处理器在保持可重构性的同时,不断提升对新型计算模式的支持能力.这一发展历程印证了 FPGA 在适应 AI 技术快速演进方面的独特价值.

最后,本文主要从软、硬件两个角度分析展望未来的创新研究方向:硬件层面上,提出更灵活的异构计算结构,如面向数据流应用的架构模板^[96],设计兼容性更强、运算效率更高的计算架构,以实现针对不同神经网络的智能适配;软件层面上,更高性能的编译优化技术,如粗粒度 Overlay 处理器的即时编译^[99]和 Seiba 的快速开发方法^[94],通过指令集拓展支持不断更新的神经网络、提升任务调度策略优化资源分配等方式,进一步释放硬件计算潜能;虽然在通用 AI 计算领域 GPU 的主导地位短期内难以撼动,但 FPGA Overlay 处理器在边缘计算、实时处理等专业场景中的优势将更加突出.伴随着这些关键技术的突破,FPGA Overlay 处理器必将在推动 AI 计算向更高效、更灵活方向发展的进程中贡献更大力量.

10 总结

为了应对定制化神经网络加速器在扩展性方面的局限,部分现有工作提出了基于 FPGA 的通用 Overlay 神经网络处理器的概念.这类 Overlay 处理器的核心思想是,通过设计统一的通用电路架构,能够适用于某一类神经网络模型的推理与部署,进而避免在模型切换时耗费大量时间进行重新设计与综合.目前针对不同类型的神经网络模型,如 CNN,Transformer 以及 GNN 等,已有相应的 Overlay 处

理器的设计实现. 本文对基于 FPGA 的通用 Overlay 神经网络处理器的定义与架构进行了系统性总结, 并深入探讨了其在设计上的优势和挑战. 针对各种类型的通用 Overlay 神经网络处理器, 本文还详细分析了其设计细节, 并对不同方案的实际效果进行了对比, 证明了此类设计在广泛应用中的优势及其未来发展的潜力.

Overlay 处理器技术可以通过指令集、计算架构以及编译技术不断更新, 根据应用场景的变化以适应不同的应用场景. 例如, Mamba 系列的神经网络可通过更新指令和编译方式来适应计算范式. 硬件架构正朝向更加灵活的异构计算结构发展, 编译优化技术的突破将进一步释放硬件潜能. 未来, 随着算子库的不断完善, FPGA Overlay 处理器将在边缘计算和实时处理等专业场景中发挥更大优势, 推动 AI 计算向更高效、更灵活的方向发展.

参考文献

- 1 Schmidhuber J. Deep learning in neural networks: an overview. *Neural Netws*, 2015, 61: 85–117
- 2 Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks. *Commun ACM*, 2017, 60: 84–90
- 3 Moro G, Fabbri M, et al. Dow Jones trading with deep learning: the unreasonable effectiveness of recurrent neural networks. In: *Proceedings of International Conference on Data Science, Technology and Applications*, 2018. 142–153
- 4 Radford A. Improving language understanding by generative pre-training. 2018. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- 5 Wu Z, Pan S, Chen F, et al. A comprehensive survey on graph neural networks. *IEEE Trans Neural Netw Learn Syst*, 2020, 32: 4–24
- 6 Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. *Proc IEEE*, 1998, 86: 2278–2324
- 7 Howard A G, Zhu M, Chen B, et al. MobileNets: efficient convolutional neural networks for mobile vision applications. 2017. ArXiv:1704.04861
- 8 Sandler M, Howard A G, Zhu M, et al. Inverted residuals and linear bottlenecks: mobile networks for classification, detection and segmentation. 2018. ArXiv:1801.04381
- 9 Ma Y, Cao Y, Vrudhula S, et al. An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks. In: *Proceedings of International Conference on Field Programmable Logic and Applications*, 2017. 1–8
- 10 Zhang X, Wang J, Zhu C, et al. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In: *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018. 1–8
- 11 Gu A, Dao T. Mamba: linear-time sequence modeling with selective state spaces. 2024. ArXiv:2312.00752
- 12 Qiu J, Wang J, Yao S, et al. Going deeper with embedded FPGA platform for convolutional neural network. In: *Proceedings of ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, 2016. 26–35
- 13 Sohrabizadeh A, Wang J, Cong J. End-to-end optimization of deep learning applications. In: *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020. 133–139
- 14 Ma Y, Cao Y, Vrudhula S, et al. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In: *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017. 45–54
- 15 Marino K, Zhang P, Prasanna V K. Me-ViT: a single-load memory-efficient FPGA accelerator for vision transformers. In: *Proceedings of IEEE International Conference on High Performance Computing, Data, and Analytics*, 2023. 213–223
- 16 Wang T, Gong L, Wang C, et al. ViA: a novel vision-transformer accelerator based on FPGA. *IEEE Trans Comput-Aided Des Integr Circuits Syst*, 2022, 41: 4088–4099
- 17 Arora A, Bhamburkar A, Borda A, et al. CoMeFa: deploying compute-in-memory on FPGAs for deep learning acceleration. *ACM Trans Reconfigurable Technol Syst*, 2023, 16: 1–34
- 18 Arora A, Ghosh M, Mehta S, et al. Tensor slices: FPGA building blocks for the deep learning era. *ACM Trans Reconfigurable Technol Syst*, 2022, 15: 1–34
- 19 Fu X, Ren Q, Wu H, et al. P³ ViT: a CIM-based high-utilization architecture with dynamic pruning and two-way ping-pong macro for vision transformer. *IEEE Trans Circuits Syst I*, 2023, 70: 4938–4948
- 20 Véstias M, Duarte R P, de Sousa J T, et al. Efficient design of low bitwidth convolutional neural networks on FPGA with optimized dot product units. *Trans ACM Reconfig Technol Syst*, 2022, 16: 1–36
- 21 Han Y, Liu Q. HPTA: a high performance transformer accelerator based on FPGA. In: *Proceedings of International*
<https://www.sciengine.com/doi/10.1360/SSI-2024-0351>

- Conference on Field-Programmable Logic and Applications, 2023. 27–33
- 22 Yu J, Ge G, Hu Y, et al. Instruction driven cross-layer CNN accelerator for fast detection on FPGA. *ACM Trans Reconfigurable Technol Syst*, 2018, 11: 1–23
 - 23 Liu Z, Dou Y, Jiang J, et al. Throughput-optimized FPGA accelerator for deep convolutional neural networks. *ACM Trans Reconfigurable Technol Syst*, 2017, 10: 1–23
 - 24 Zhao Z, Cao R, Un K F, et al. An FPGA-based transformer accelerator using output block stationary dataflow for object recognition applications. *IEEE Trans Circuits Syst II*, 2023, 70: 281–285
 - 25 Sarkar R, Liang H, Fan Z, et al. Edge-MoE: memory-efficient multi-task vision transformer architecture with task-level sparsity via mixture-of-experts. In: *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, 2023. 01–09
 - 26 Luo Z, Lu L, Jin Y, et al. Calabash: accelerating attention using a systolic array chain on FPGAs. In: *Proceedings of International Conference on Field-Programmable Logic and Applications*, 2023. 242–247
 - 27 Wang J, Guo L, Cong J. AutoSA: a polyhedral compiler for high-performance systolic arrays on FPGA. In: *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021. 93–104
 - 28 Zeng S, Liu J, Dai G, et al. FlightLLM: efficient large language model inference with a complete mapping flow on FPGAs. In: *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024. 223–234
 - 29 Basalama S, Sohrabizadeh A, Wang J, et al. FlexCNN: an end-to-end framework for composing CNN accelerators on FPGA. *ACM Trans Reconfigurable Technol Syst*, 2023, 16: 1–32
 - 30 Fang C, Guo S, Wu W, et al. An efficient hardware accelerator for sparse transformer neural networks. In: *Proceedings of IEEE International Symposium on Circuits and Systems*, 2022. 2670–2674
 - 31 Kwon E, Song H, Park J, et al. Mobile accelerator exploiting sparsity of multi-heads, lines, and blocks in transformers in computer vision. In: *Proceedings of Design, Automation & Test in Europe Conference & Exhibition*, 2023. 1–6
 - 32 Dong P, Sun M, Lu A, et al. HeatViT: hardware-efficient adaptive token pruning for vision transformers. In: *Proceedings of IEEE International Symposium on High-Performance Computer Architecture*, 2023. 442–455
 - 33 Qin Y, Wang Y, Deng D, et al. Fact: FFN-attention co-optimized transformer architecture with eager correlation prediction. In: *Proceedings of Annual International Symposium on Computer Architecture*, 2023. 1–14
 - 34 Fan H, Chau T, Venieris S I, et al. Adaptable butterfly accelerator for attention-based NNS via hardware and algorithm co-design. In: *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, 2022. 599–615
 - 35 Plagwitz P, Hannig F, Teich J. TRAC: compilation-based design of transformer accelerators for FPGAs. In: *Proceedings of International Conference on Field-Programmable Logic and Applications*, 2022. 17–23
 - 36 Meng Y, Kuppannagari S, Kannan R, et al. DynaMap: dynamic algorithm mapping framework for low latency CNN inference. In: *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021. 183–193
 - 37 Khataei A, Singh G, Bazargan K. Approximate hybrid binary-unary computing with applications in BERT language model and image processing. In: *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023. 165–175
 - 38 Peng H, Huang S, Chen S, et al. A length adaptive algorithm-hardware co-design of transformer on FPGA through sparse attention and dynamic pipelining. In: *Proceedings of ACM/IEEE Design Automation Conference*, 2022. 1135–1140
 - 39 Niu Y, Kannan R, Srivastava A, et al. Reuse kernels or activations? A flexible dataflow for low-latency spectral CNN acceleration. In: *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020. 266–276
 - 40 Sun M, Li Z, Lu A, et al. FILM-QNN: efficient FPGA acceleration of deep neural networks with intra-layer, mixed-precision quantization. In: *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2022. 134–145
 - 41 Hong S, Moon S, Kim J, et al. DFX: a low-latency multi-FPGA appliance for accelerating transformer-based text generation. In: *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, 2022. 616–630
 - 42 Zhang C, Li P, Sun G, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: *Proceedings of ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, 2015. 161–170
 - 43 Zhang C, Sun G, Fang Z, et al. Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks. In: *Proceedings of ACM Turing Award Celebration Conference-China*, 2023. 47–48
 - 44 Ye W, Zhou X, Zhou J, et al. Accelerating attention mechanism on FPGAs based on efficient reconfigurable systolic array. *ACM Trans Embed Comput Syst*, 2023, 22: 1–22
 - 45 Qian K, Lu H, Liu Y, et al. End-to-end compilation is all FPGAs need: a unified overlay-based FPGA compiler for deep learning. In: *Proceedings of Asia and South Pacific Design Automation Conference*, 2025
 - 46 Qian K, Liu Z, Lu H, et al. Autohammer: breaking the compilation wall between deep neural network and overlay-based FPGA accelerator. In: *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate*

- Arrays, 2024
- 47 Wang Y, Li Y, Chen J, et al. Taichi: efficient execution for multi-DNNs using graph-based scheduling. In: Proceedings of Design, Automation and Test in Europe Conference, 2025
 - 48 Liu Y, Wang T, Dong Y, et al. Translib: an extensible graph-aware library framework for automated generation of transformer operators on FPGA. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2024
 - 49 Zhou X, Li S, Lu H, et al. PipeFuser: building flexible pipeline architecture for DNN accelerators via layer fusion. In: Proceedings of Asia and South Pacific Design Automation Conference, 2024
 - 50 Tang E, Li S, Zhou H, et al. AispGEMM: accelerating imbalanced SpGEMM on FPGAs with flexible interconnect and intra-row parallel merging. In: Proceedings of Design, Automation and Test in Europe Conference, 2025
 - 51 Xu Z, Xu J, Peng H, et al. ALT: breaking the wall between data layout and loop optimizations for deep learning compilation. In: Proceedings of ACM European Conference on Computer Systems, 2023
 - 52 Yu Y, Wu C, Zhao T, et al. OPU: an FPGA-based overlay processor for convolutional neural networks. *IEEE Trans VLSI Syst*, 2019, 28: 35–47
 - 53 Liu S, Du Z, Tao J, et al. Cambricon: an instruction set architecture for neural networks. *SIGARCH Comput Archit News*, 2016, 44: 393–405
 - 54 Abdelfattah M S, Han D, Bitar A, et al. DLA: compiler and FPGA overlay for neural network inference acceleration. In: Proceedings of International Conference on Field Programmable Logic and Applications, 2018. 411–4117
 - 55 Wu C, Wang M, Chu X, et al. Low-precision floating-point arithmetic for high-performance FPGA-based CNN acceleration. *ACM Trans Reconfigurable Technol Syst*, 2022, 15: 1–21
 - 56 Yu Y, Zhao T, Wang K, et al. Light-OPU: an FPGA-based overlay processor for lightweight convolutional neural networks. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2020. 122–132
 - 57 Yu Y, Zhao T, Wang M, et al. Uni-OPU: an FPGA-based uniform accelerator for convolutional and transposed convolutional networks. *IEEE Trans VLSI Syst*, 2020, 28: 1545–1556
 - 58 Wu C, Zhuang J, Wang K, et al. MP-OPU: a mixed precision FPGA-based overlay processor for convolutional neural networks. In: Proceedings of International Conference on Field-Programmable Logic and Applications, 2021. 33–37
 - 59 Zhou S, Meng S, Tian H, et al. Edge-bit: software-hardware co-design for optimizing binarized transformer networks inference on edge FPGA. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2024
 - 60 Wang L, Guo S, Huang W, et al. Places205-VGGNet models for scene recognition. 2015. ArXiv:1508.01667
 - 61 Alom M Z, Taha T M, Yakopcic C, et al. The history began from AlexNet: a comprehensive survey on deep learning approaches. 2018. ArXiv:1803.01164
 - 62 Targ S, Almeida D, Lyman K. ResNet in ResNet: generalizing residual architectures. In: Proceedings of International Conference on Learning Representations, 2016
 - 63 Khan R U, Zhang X, Kumar R. Analysis of ResNet and GoogleNet models for malware detection. *J Comput Virol Hack Tech*, 2019, 15: 29–37
 - 64 Khan H, Khan A, Khan Z, et al. NPE: an FPGA-based overlay processor for natural language processing. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2021
 - 65 Bai Y, Zhou H, Zhao K, et al. Transformer-OPU: an FPGA-based overlay processor for transformer networks. In: Proceedings of Annual International Symposium on Field-Programmable Custom Computing Machines, 2023. 221–221
 - 66 Bai Y, Zhou H, Zhao K, et al. LTrans-OPU: a low-latency FPGA-based overlay processor for transformer networks. In: Proceedings of International Conference on Field-Programmable Logic and Applications, 2023. 283–287
 - 67 Bai Y, Zhou H, Zhao K, et al. FET-OPU: a flexible and efficient FPGA-based overlay processor for transformer networks. In: Proceedings of IEEE/ACM International Conference on Computer Aided Design, 2023. 1–9
 - 68 Lu S, Zhao T, Zhang R, et al. An FPGA-based multi-core overlay processor for transformer-based models. In: Proceedings of International Symposium of Electronics Design Automation, 2024. 697–702
 - 69 Tiandong Z, Shaoqiang L, Chen W, et al. ChatOPU: an FPGA-based overlay processor for large language models with unstructured sparsity. In: Proceedings of IEEE/ACM International Conference on Computer Aided Design, 2024
 - 70 Liu Y, He X, Yu J, et al. DIF-LUT: a simple yet scalable approximation for non-linear activation function on FPGA. In: Proceedings of International Conference on Field-Programmable Logic and Applications, 2023
 - 71 Zou K, Zhang Y, Zhang Z, et al. Prev: eliminating store queue via premature value validation for dataflow circuit on FPGA. In: Proceedings of Design, Automation and Test in Europe Conference, 2025
 - 72 Nguyen D, Kim D, Lee J. Double MAC: doubling the performance of convolutional neural networks on modern FPGAs. In: Proceedings of Design, Automation & Test in Europe Conference & Exhibition, 2017. 890–893
 - 73 Chollet F. Keras. 2015. <https://keras.io>
 - 74 Yu J, Park J, Park S, et al. NN-LUT: neural approximation of non-linear operations for efficient transformer inference.

- In: Proceedings of ACM/IEEE Design Automation Conference, 2022. 577–582
- 75 Wang H, Bai Y, Yu J, et al. TransFRU: efficient deployment of transformers on FPGA with full resource utilization. In: Proceedings of Asia and South Pacific Design Automation Conference, 2024
 - 76 Wang H, Zhang Y, Yu J, et al. FFAS-trans: fully exploiting FFN and attention sparsity for transformer on FPGA. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2024
 - 77 Shi K, Zhang M, Zhao K, et al. Fitop-trans: maximizing transformer pipeline efficiency through fixed-length token pruning on FPGA. In: Proceedings of International Conference on Field-Programmable Logic and Applications, 2024
 - 78 Lin X, Tian H, Xue W, et al. FLAME: fully leveraging MoE sparsity for transformer on FPGA. In: Proceedings of Design Automation Conference, 2024
 - 79 Zhang M, Cao J, Shi K, et al. FNM-trans: efficient FPGA-based transformer architecture with full n:m sparsity. In: Proceedings of Design Automation Conference, 2024
 - 80 Liu Z, Zhou X, Sun F, et al. Attentionlib: a scalable optimization framework for automated attention acceleration on FPGA. In: Proceedings of Design, Automation and Test in Europe Conference, 2025
 - 81 Cao J, Lin X, Zhang M, et al. PP-transformer: enable efficient deployment of transformers through pattern pruning. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2023
 - 82 Yu Z, Wang Z, Li Y, et al. EDGE-LLM: enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In: Proceedings of the 61st ACM/IEEE Design Automation Conference, 2024
 - 83 Haris J, Saha R, Hu W, et al. Designing efficient LLM accelerators for edge devices. 2024. ArXiv:2408.00462
 - 84 Li G, Zhang R, Yu J, et al. Flop: a flexible memory-optimized processor for parallel graph mining on FPGA. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2024
 - 85 Liu Z, Zhou X, Sun F, et al. Famers: an FPGA accelerator for memory-efficient edge-rendered 3D Gaussian splatting. In: Proceedings of Design, Automation and Test in Europe Conference, 2025
 - 86 Tao Z, Wu C, Liang Y, et al. LW-GCN: a lightweight FPGA-based graph convolutional network accelerator. *ACM Trans Reconfigurable Technol Syst*, 2023, 16: 1–19
 - 87 Wu C, Tao Z, Wang K, et al. SkeletonGCN: a simple yet effective accelerator for GCN training. In: Proceedings of International Conference on Field-Programmable Logic and Applications, 2022. 445–451
 - 88 Tang E, Li S, Chen R, et al. Graph-OPU: a highly flexible FPGA-based overlay processor for graph neural networks. *ACM Trans Reconfigurable Technol Syst*, 2024, 17: 1–33
 - 89 Chen R, Zhang H, Li S, et al. Graph-OPU: a highly integrated FPGA-based overlay processor for graph neural networks. In: Proceedings of International Conference on Field-Programmable Logic and Applications, 2023. 228–234
 - 90 Liang S, Liu C, Wang Y, et al. Deepburning-GL: an automated framework for generating graph neural network accelerators. In: Proceedings of International Conference on Computer-Aided Design, 2020. 1–9
 - 91 Cai H, Zhang H, Zhao D, et al. FP-GNN: a versatile deep learning architecture for enhanced molecular property prediction. *Brief Bioinf*, 2022, 23: bbac408
 - 92 Sarkar R, Abi-Karam S, He Y, et al. FlowGNN: a dataflow architecture for real-time workload-agnostic graph neural network inference. In: Proceedings of IEEE International Symposium on High-Performance Computer Architecture, 2023. 1099–1112
 - 93 Liu S, Weng J, Kupsh D, et al. OverGen: improving FPGA usability through domain-specific overlay generation. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, 2022. 35–56
 - 94 Wilson D, Stitt G. Seiba: an FPGA overlay-based approach to rapid application development. In: Proceedings of International Conference on ReConFigurable Computing and FPGAs, 2019. 1–8
 - 95 Li S, Chen R, Tang E, et al. S-LGCN: a software-hardware co-design for accelerating lightgcn. In: Proceedings of Design, Automation and Test in Europe Conference, 2024
 - 96 Drewes A, Burtsev V, Gurumurthy B, et al. An architectural template for FPGA overlays targeting data flow applications. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops, 2024
 - 97 Bai Y, Zhao K, Liu Y, et al. CSTrans-OPU: an FPGA-based overlay processor with full compilation for transformer networks via sparsity exploration. In: Proceedings of Design Automation Conference, 2024
 - 98 Li X, Maskell D L. Time-multiplexed FPGA overlay architectures. *ACM Trans Des Autom Electron Syst*, 2019, 24: 1–19
 - 99 Jain A K, Maskell D L, Fahmy S A. Coarse grained FPGA overlay for rapid just-in-time accelerator compilation. *IEEE Trans Parallel Distrib Syst*, 2021, 33: 1478–1490
 - 100 Choudhury Z, Shrivastava S, Ramapantulu L, et al. An FPGA overlay for CNN inference with fine-grained flexible parallelism. *ACM Trans Archit Code Optim*, 2022, 19: 1–26

FPGA Overlay processor for AI computing

Lei HE^{1*}, Kun WANG², Chen WU^{3,4,1}, Zhuofu TAO⁵, Xiao SHI⁶, Siyuan MIAO⁵ & Shaoqiang LU^{3,4,1}

1. *College of Information Science and Technology, Eastern Institute of Technology, Ningbo, Ningbo 315200, China*

2. *School of Microelectronics, Fudan University, Shanghai 200433, China*

3. *Chiplet CAD and Manufacturing Engineering Research Center of Zhejiang Province, Ningbo 315201, China*

4. *Ningbo Institute of Digital Twin, Eastern Institute of Technology, Ningbo 315201, China*

5. *Electrical and Computer Engineering, University of California, Los Angeles, Los Angeles 90095, USA*

6. *School of Computer Science and Engineering, Southeast University, Nanjing 211189, China*

* Corresponding author. E-mail: lhe@eitech.edu.cn

Abstract Currently, the computational complexity and update speed of neural networks are experiencing an explosive growth trend. To efficiently use neural networks on devices, technologies that accelerate neural networks have gained attention from both academia and industry. There are two categories of main neural network accelerators. The first is typically customized for specific types of neural networks to enhance inference performance. However, these schemes have certain limitations, such as poor scalability and a rather fixed computational data flow path. These limitations make these schemes inflexible in dealing with applications that require frequent switching of network models, especially in scenarios with high dynamic switching and real-time response. The second one is an instruction-based Overlay processing unit. The schemes of the Overlay processing unit first propose an instruction set for neural networks, which can easily be extended for coming networks. At the same time, a compiler is designed to perform optimization on the computation graph, allocation of computation, and scheduling of inputs from the software view. Meanwhile, the microarchitecture is designed for dataflow and parallelism optimizations from the hardware point of view. To this end, this study first analyzes the design philosophy and architectural features of customized neural network accelerators and Overlay processing units. Secondly, by comparing these two types of accelerators in terms of design scheme, flexibility, and applicability, this study delves into the general Overlay processing unit, emphasizing its significant advantages in terms of ease of use, scalability, and versatility. The study then systematically categorizes the existing FPGA prototypes of the Overlay processing units and provides an in-depth analysis of the design principles, implementation methods, and performance in practical applications for each category of processors. Finally, this study discusses the challenges faced in the design process of general Overlay processing units and looks forward to the potential directions and application scenarios of this technology in future development, with the aim of providing references and insights for future research work.

Keywords artificial intelligence, FPGA, customized accelerator, Overlay processing unit, compiler, hardware acceleration