基于非一般类算子融合方法及硬件架构设计

王 莹[1,2],高 岚[1*],张 哲[3],刘 昕[1],武毅雄[1],张伟功[1]
(1. 首都师范大学信息工程学院,北京 100048;2. 首都师范大学数学科学学院,北京 100048;
3. 山西农业大学软件学院,山西晋中 030801)

摘 要: 针对传统算子融合算法在异构计算系统跨计算单元时的失效性问题,本文提出一种优化后的算子融合策略,并针对新型融合算法进行了硬件设计实现. 论文基于传统算子融合算法的设计初衷,在端侧异构计算系统部署深度学习算法时,分析算子融合覆盖率对推理任务计算性能的影响,挖掘跨计算单元算子融合的可能性,设计可以提升算子融合覆盖率的改进算法模型;同时,通过构建以 CPU(Central Processing Unit)+GPU(Graphics Processing Unit)+DLA((Deep Learning Accelerator))组成的异构计算平台,为改进后的算子融合策略提供结构更加耦合的多层级存储共享结构. 实验结果表明,与优化前的算子融合算法相比,改进后的算子融合策略可以有效提升算子融合覆盖率,部署在 Xilinx 公司 FPGA(Field-Programmable Gate Array)开发板上进行目标检测网络推理实验. 结果表明,本文提出的设计方案,针对 YOLOX-Nano 的推理过程可实现 62.67% 推理计算性能提升,计算加速比为 2.68;针对 YOLOv5s 的推理过程可实现 71.10% 推理计算性能提升,计算加速比为 3.46.

关键词: 深度学习;算子融合;卷积神经网络;异构计算;FPGA;GPU
基金项目: 国家自然科学基金(No.62202317)
中图分类号: TP302.8 **文献标识码:** A **文章编号:** 0372-2112(2025)09-3299-11
电子学报 URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20250312

Operator Fusion Method and Hardware Architecture Design Based on Non-Standard Operators

WANG Ying[1,2], GAO Lan[1*], ZHANG Zhe[3], LIU Xin[1], WU Yi-xiong[1], ZHANG Wei-gong[1]
(1. *College of Information Engineering*, *Capital Normal University*, *Beijing* 100048, *China*;
2. *School of Mathematical Science*, *Capital Normal University*, *Beijing* 100048, *China*;
3. *Faculty of Software Technologics*, *Shanxi Agricultural University*, *Jinzhong*, *Shanxi* 030801, *China*)

Abstract: To address the failure of traditional operator fusion algorithms in heterogeneous computing systems when crossing different computing units, this paper proposes an optimized operator fusion strategy and implements a hardware design for the novel fusion algorithm. Building upon the original design intentions of traditional operator fusion, we analyze the impact of operator fusion coverage on inference performance when deploying deep learning algorithms on edge-side heterogeneous computing systems. We explore the feasibility of cross-unit operator fusion and design an improved fusion algorithm model that enhances fusion coverage. Furthermore, a heterogeneous computing platform composed of CPU (Central Processing Unit), GPU (Graphics Processing Unit) and DLA (Deep Learning Accelerator) is constructed, incorporating a tightly coupled multi-level shared memory architecture tailored for the optimized fusion strategy. Experimental results demonstrate that the proposed fusion strategy significantly improves operator fusion coverage compared to the unoptimized version. Deployed on a Xilinx FPGA (Field-Programmable Gate Array) development board for object detection network inference, the proposed design achieves a 62.67% performance improvement and a 2.68× speedup for YOLOX-Nano inference, and a 71.10% performance improvement and a 3.46× speedup for YOLOv5s inference.

Key words: deep learning; operator fusion; convolutional neural network; heterogeneous computing system; FPGA; GPU
Foundation Item(s): National Natural Science Foundation of China (No.62202317)

收稿日期:2025-04-22;录用日期:2025-09-08;责任编辑:覃怀银
*通讯作者:高岚

1 引言

随着人工智能的发展,计算机视觉算法和技术不断迭代更新,卷积神经网络已经成为一种公认的、适合图像处理的一种最基本的网络结构,在其特征提取优势作用下,计算机视觉任务应用取得了可以超越人类的视觉分辨精度[1,2]. 将以卷积神经网络为主的算法模型与嵌入式端侧系统相结合也已经成为计算机视觉算法研究和应用的热点方向,包括目标检测、自动驾驶、虚拟现实等[3~5]. 然而,在物联网端侧人工智能需求的引领下,经过不断探索和发展,相关技术发展瓶颈已经开始凸显. 一方面,不协调的软硬件发展严重限制了算法性能发挥,算法模型对端侧系统的计算能力的要求更高[6];另一方面,端侧系统在满足算力需求的同时,仍需应对系统体积、电源供应、散热等严峻问题[7]. 特别是在计算资源和存储资源有限的嵌入式终端设备上,这类问题更加突出[8].

为了应对上述挑战,相关的领域研究包括专用深度学习加速器研究、异构计算架构研究和算子融合技术研究等. 专用深度学习加速器(Deep Learning Accelerator,DLA)是一种面向深度学习算法计算特征需求设计的领域架构定制计算单元,典型的研究成果有:Google TPU 采用 ASIC(Application-Specific Integrated Circuit)设计,通过脉动阵列架构快速处理神经网络所需的大量乘法和加法运算[9,10];NVDLA 是 NVIDIA 公司开发的一个用于加速推理的开源框架,开源的 Verilog RTL 模型支持通过不同的参数化配置综合实现不同的电路规模[11];GCONV Chain 是一种将整个 CNN(Convolutional Neural Network)计算转换为一系列标准通用卷积(GCONV)的方法,并且所提出的 GCONV 支持在现有的 CNN 加速器上进行高效处理,该方法减少了 70.6% 性能开销,并降低了约 68.7% 系统能耗[12]. 而无论是哪种专用 DLA 处理器,都需要外部 CPU 通用设备参与控制与调度协助. 例如,在 SiFive 与 NVIDIA 的一项合作中,将 RISC-V+NVDLA 成功运行在 SiFive Freedom 平台上,并实现了 YOLOv3 目标检测算法部署[13];NVIDIA 公司推出的 Xavier 系统级芯片,采用 CPU(Central Processing Unit)、GPU(Graphics Processing Unit)、深度学习加速器 NVDLA 等六种处理器,可以实时运行数十种算法[14];AMD 公司推出的 Versal Prime 自适应计算加速平台,是一个完全支持软件编程的异构计算平台,将标量引擎、自适应引擎和 DSP 引擎相结合,支持用户定制自己的特定领域专用计算架构;AMD 的另一个异构计算研究方向 Chiplet,支持将 X86 CPU、GPU 及 AI 加速器一起集成在 2D 或 3D 封装中[15];Cambricon-Q 是中科院寒武纪提出的一种混合加速器计算架构,由一个 ASIC 加速器核心和一个真实的近数据处理 NDP 引擎组成[16];由此可见,以 CPU+xPU 组成的异构计算系统成为当前主流高效计算架构的实现方法. 算子融合[17~21]技术研究主要针对深度学习模型算子层的合并优化,有效减少中间特征数据读写操作引发的内存访问,加速模型推理计算过程、降低数据传输延迟和访存开销,从而提升算法模型的运行效率. 目前,算子融合已经成为深度学习模型优化的重要手段. 算子融合方法,主要包括垂直融合、水平融合和复合融合. 其中,垂直融合是将计算图中具有前后依赖关系的算子层进行合并,水平融合是指同类型的算子层进行合并;垂直融合和水平融合是不改变原有函数规则的算子合并,而复合融合是以数学方法指导两个或以上算子层进行合并操作,合并后最终得到一个计算过程更加简化的等价新函数替代原有算子层运算. 关于算子融合的相关研究有:TensorFlow XLA 在 TPU 训练阶段会采用计算图优化,减少数据搬移,提升模型训练效率;TVM 是一个开源的深度学习编译器和运行时系统,其算子融合过程分为算子图构建和融合优化,实现将多个算子组合成一个更大的算子[22];DNNVM 是一种集成循环和数据布局优化器、汇编器、运行时支持器和验证环境的框架,通过启发式子图同构算法来检测出所有可能的融合机会[23];TensorRT 是 Nvidia 公司提供的深度学习推理优化器和运行时库[24],内部提供了算子融合功能,支持将多个算子合并为一个高效的复合算子,优势在于通过自动算子融合优化计算图,减少 Kernel 调度次数,因此主要用于 GPU 计算优化,适用 NVIDIA 硬件;DNNFusion 是一种用于深度神经网络加速的技术,可以与各种编程语言和编译器结合使用,如 TensorFlow、PyTorch、Caffe 等,利用不同框架的相关功能或库从而实现算子融合. Open AI Lab 开源的边缘设备推理引擎 Tengine 内部自动算子优化工具[25],采用 Halid 框架可以对计算图分析,可以实现算子融合自动化及新代码生成,目的减少指令执行及中间计算结果的访存次数.

尽管针对系统性能与功耗的问题正在不同研究领域得到不断的丰富与推进,但是传统背景下异构系统的算子融合策略具有一定的局限性,因而满足异构计算系统中跨计算单元的算子融合技术研究尚且不多. 本文基于常规算子融合方法进行分析,总结提炼一种能适用于端侧异构计算系统的非一般类算子融合模型,并设计实现了一种高效的可支持改进后算子融合策略的异构计算架构. 首先,找到影响算子融合覆盖率的因素,在典型异构计算系统中分析算子融合优化算法与硬件计算资源之间的需求关系. 其次,结合改进后的算子融合策略需求,专门设计了多层级共享存储的异构计算架构,支持深度学习推理过程在端侧异构计算系统中的高效应用. 最后,基于 FPGA(Field-

Programmable Gate Array)硬件平台进行实验,与传统异构计算系统下进行深度学习推理算子融合方法相比,本文提出的扩张型算子融合优化方法和改进后的异构计算架构,可以有效提升神经网络的推理速度.

2　非一般类算子融合方法

2.1　传统算子融合算法

2.1.1　算子融合覆盖率定义

深度学习类算法模型通过多层卷积神经网络执行推理任务,通过不断地重复卷积、激活和池化的计算过程实现多尺度特征提取. 传统一般类型算子融合优化的研究目标是减少深度学习算法计算过程对存储器的频繁访问,改善中间特征数据在算子层之间的传输效率,加速推理计算过程. 其判别条件如下:(1)网络模型中连续两个或两个以上的关系节点满足预设依赖关系,如卷积→激活、卷积→池化、卷积→激活→池化等;(2)对满足依赖关系的节点须同时映射在同一个运算单元中.

用 $G=(V,E)$ 表示网络算子层节点与节点之间的关系,其中,V 是算子层节点的集合,如式(1)所示:

$$V=\{v_i|0\leqslant i\leqslant \text{num\_layer}-1\} \tag{1}$$

其中,v_0 是起始节点,$v_{\text{num\_layer}+1}$ 是终止节点,num_layer是网络层节点数量,节点 v_i 用 (cls_func, cls_pu) 表示,cls_func 和 cls_pu 分别是节点的运算类型和运算载体. E 是算子层关系集合,如式(2)所示:

$$E=\{e_j|0\leqslant j\leqslant \text{num\_rls}-1\} \tag{2}$$

每个 e_j 用 (V_p,V_n) 表示连接的两个算子层节点 p 和 n,num_rls是算子节点关系总数量. 定义融合覆盖率,如式(3)所示:

$$R_{\text{fusion}}=\text{num\_opd}/\text{num\_rls} \tag{3}$$

num_opd是被优化的关系边的数量,其中关系边的数量直接对应相邻算子层切换过程中对存储访问的依赖,被优化的关系边的数量越多,越能减少相邻算子层对存储器的访问次数,进而减少数据在两个算子层之间的等待时间,提升推理过程的运算效率.

2.1.2　算子融合覆盖率分析

本文以图像分类算法VGG16和目标检测YOLOv5s算法在硬件加速器NVDLA上部署为例,分析传统优化策略对算法模型的融合覆盖率.

(1)对于VGG16网络,其特征提取主干网络共包括13个卷积层、13个激活层以及5个池化层,共31个算子层,并以Convolution→ReLU和Convolution→ReLU→MaxPooling的形式组成规律性迭代. 这两种规律性迭代与2.1.1节的常规算子融合依赖关系吻合,可直接进行算子层融合优化,融合操作后VGG16特征提取主干网络层算子关系数量由31个减少至13个,有18项算子层关系被优化,占总关系数量41.9%,即 $R_{\text{fusion}}=41.9\%$.

(2)对于YOLOv5s网络的推理结构,其工作流程仍以串行逻辑依赖特征为主体,且满足Convolution→Sigmoid→elewiseMul形式组成的规律性迭代的有57组. 其中,有7组与elewiseAdd算子组成ResUnit结构、进一步有10组与Concat算子组成CPS结构. 理想状态下,首先,将Convolution和Sigmoid与elewiseMul进行算子融合操作,将生成新算子称为CSeM. 此时,算子关系数量由120个减少至74个,有114项算子层关系被优化,占总关系数量60.6%,算子融合覆盖率为 $R_{\text{fusion}}=60.6\%$;其次,将融合范围扩展至CPS结构,可以进一步使Convolution与elewiseAdd或Concat进行算子融合操作,可实现算子数量由118个减少至57个,即算子融合覆盖率可提升至69.6%,即 $R_{\text{fusion}}=69.6\%$. 但是,在以深度学习加速器(以NVDLA为例)为加速计算核心的系统中,其不具备Sigmoid激活算子,导致算子融合前提条件不满足,无法执行算子融合,即 $R_{\text{fusion}}=0$.

由此可见,运算单元内部可执行的算子运算种类是决定推理运算效率的基础,算子功能缺失问题不仅会影响加速计算过程的连续性,同时会影响网络模型算子融合覆盖率. 特别是在当前以规律性迭代为主要特征的推理任务中,算子功能缺失问题对网络模型算子融合覆盖率的影响则更为严重. 而通过其他计算单元补充该运算过程,又不能满足常规算子融合策略的判别条件. 因此,本文将诸如此类具有依赖关系且不能通过传统算子融合方法进行计算过程加速优化的问题,总结为需要研究一种非一般类算子融合方法,将判别条件(2)扩展至可满足在不同计算单元之间实现算子融合优化,实现理想状态下的算子合并对提升算子融合覆盖率具有十分重要的意义.

2.2　非一般类算子融合优化方法

2.2.1　算子融合存储模型优化

随着网络模型的不断发展以及物联网端侧场景应用的日益复杂化,局限在同一个计算单元内的计算过程合并可能性也越来越小. 根据2.1节分析,在一般类算子融合执行过程中,算子缺失是影响计算单元内部实现算子合并和融合覆盖率的主要因素,算子融合覆盖率越低,推理计算过程在不同计算单元之间的计算过程切换越频繁,不仅会导致了计算延迟增加,也导致了不必要的访问开销. 本节将挖掘非一般类算子融合的可能性,具体研究如何突破传统一般类算子融合的局限性,将网络模型的计算过程可合并范围扩展至多个不同计算单元,提高网络优化融合覆盖率,从而提升端侧系统计算的高效性.

为了兼顾算子功能缺失问题以及对主控CPU执行常规任务的效率问题，假设异构计算系统的计算核心包括CPU、GPU和DLA，其中CPU主要负责完成常规计算任务和过程控制类运算，GPU主要负责配合DLA完成深度学习算法模型中的补充算子功能运算. 卷积算子(conv2d)与激活算子(activate)的融合是一个最具代表性的可融合神经网络关系节点，融合前的计算过程分两步进行，如式(4)和式(5)所示：

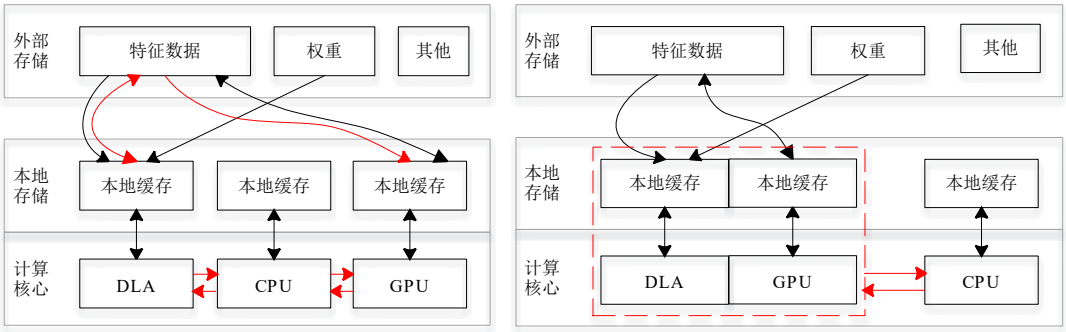
$$\text{ft\_tmp} = \text{conv2d}(\text{input}, \text{weight}) \tag{4}$$

$$\text{output} = \text{activate}(\text{ft\_tmp}) \tag{5}$$

融合后计算过程如式(6)所示：

$$\text{output} = \text{activate}\big(\text{conv2d}(\text{input}, \text{weight})\big) \tag{6}$$

图1是以GPU运算单元补充激活算子功能前后的对比图，图中展示了将参与运算的特征数据、权重数据存放在外部存储器时，融合前后的存储结构与数据流动变化，不难看出，算子融合的技术优势在于：(1)可以减少计算核心单元对外部存储器的访问次数，推理过程中的中间结果的存放位置不是外部存储空间，而是距离计算核心较近的本地存储区；(2)缩短数据传输的延迟时间，减少高延迟存储器访问的等待；(3)减少对CPU的控制过程依赖. 这种典型的算子融合策略在指导具有规律性迭代的计算过程优化中，可以有效提升计算的高效性.



(a) 算子融合前的存储结构与数据流动　　(b) 算子融合后的存储结构与数据流动

图1　算子融合前后存储模型优化对比图

2.2.2　存储调度优化生成算法

由于多分支结构是当前主流卷积神经网络的另一项主要结构特征，因而数据依赖关系也呈现出对远距离数据的单向延迟需求. 在2.2.1节算子融合存储模型优化思想下，本小节将继续研究如何生成存储模型的调度分配算法.

首先，定义并初始化系统中各类存储单元，如算法1所示.

(1)运算单元内部的高速缓存存储容量如式(7)所示：

$$\text{Buf} = \{\text{buf}_{\text{puID}} | 0 \leqslant \text{puID} \leqslant \text{PU\_NUM} - 1\} \tag{7}$$

其中，puID表示运算单元编号；PU_NUM表示系统中运算单元数量；buf_{puID}则表示编号为puID的运算单元可提供的最大高速缓存容量，单位：Byte. 同时，以$\text{freebuf}_{\text{puID}}$表示puID的运算单元可提供的剩余高速缓存容量. 初始状态下，每个运算单元的内部空闲缓存容量即为其最大容量buf_{puID}. 在运行过程中，当收到存储分配请求req_buf时，如果满足$\text{req_buf} < \text{freebuf}_{\text{puID}}$分配条件，则执行存储分配；同时，更新剩余存储容量$\text{freebuf}_{\text{puID}}$为$\text{freebuf}_{\text{puID}} - \text{req_buf}$.

(2)系统提供在片上存储容量如式(8)所示：

$$\text{OnChip} = \{\text{ram}_{\text{rID}} | 0 \leqslant \text{rID} \leqslant \text{RAM\_NUM} - 1\} \tag{8}$$

其中，rID表示RAM存储模块编号；RAM_NUM表示系统中RAM存储模块数量；ram_{rID}则表示编号为rID的存储模块可提供的最大存储容量，单位：Byte. 同时，以$\text{freeram}_{\text{rID}}$表示rID存储单元可提供的剩余存储容量. 初始状态下，每个片内共享存储模块容量即为其最大容量ram_{rID}. 在运行过程中，当收到存储分配请求req_ram时，如果满足$\text{req_ram} < \text{freeram}_{\text{rID}}$分配条件，则执行存储分配；同时，更新剩余存储容量$\text{freeram}_{\text{rID}}$为$\text{freeram}_{\text{rID}} - \text{req_ram}$.

(3)系统提供在片外存储容量如式(9)所示：

$$\text{OffChip} = \{\text{ddr}_{\text{dID}} | 0 \leqslant \text{dID} \leqslant \text{DDR\_NUM} - 1\} \tag{9}$$

其中，dID表示外部存储器DDR编号；DDR_NUM表示系统中外部存储器DDR数量；ddr_{dID}则表示编号为dID的存储模块可提供的最大存储容量，单位：Byte. 同时，以$\text{freeddr}_{\text{dID}}$表示dID存储单元可提供的剩余存储容量. 初始状态下，每个片外共享存储模块容量即为其最大容量ddr_{dID}. 在运行过程中，当收到存储分配请求req_ddr时，如果满足$\text{req_ddr} < \text{freeddr}_{\text{dID}}$分配条件，则执行存储分配；同时，更新剩余存储容量$\text{freeddr}_{\text{dID}}$为$\text{freeddr}_{\text{dID}} - \text{req_ddr}$.

算法1　存储单元初始化与容量更新算法

输入:

运算单元数量:PU_NUM

每个运算单元内部最大缓存容量:buf_{puId}

片上共享存储模块数量:RAM_NUM

每个片内共享存储模块可提供的最大存储容量:ram_{rId}

片外共享存储模块数量:DDR_NUM

每个片外共享存储模块可提供的最大存储容量:ddr_{dId}

输出:

每个运算单元内部空闲的缓存容量:$freebuf_{puId}$

每个片内共享存储模块空闲的存储容量:$freeram_{rId}$

每个片外共享存储模块空闲的存储容量:$freeddr_{dId}$

Mem-init:

$freebuf_{puID} = buf_{puID}$

$freeram_{rID} = ram_{rID}$

$freeddr_{dID} = ddr_{dID}$

Mem-update:

IF ($req\_buf < freebuf_{puID}$)

　$freebuf_{puID} = freebuf_{puID} - req\_buf$

IF ($req\_ram < freeram_{rID}$)

　$freeram_{rID} = freeram_{rID} - req\_ram$

IF ($req\_ddr < freeddr_{dID}$)

　$freeddr_{dID} = freeddr_{dID} - req\_ddr$

其次,为每个融合后的算子节点建立内部节点之间存储分配调度分配方案. 区别于一般性算子融合方法,结合本文2.2节已讨论的改进算子融合方法形成的融合新算子(简称fusedNewOP)支持将一个连续的运算过程分配在不同计算单元中进行,因此需要根据融合后的算子fusedNewOP生成三种存储分配调度队列selfBuf、sharedRAM和sharedDDR.

如算法2描述,遍历每个fusedNewOP的运算过程$subOP_i$,查看当前运算过程和下一个后续运算过程$subOP_{i+1}$所映射的运算单元. 首先,对于前后连续的算子操作,算法需要分别获取前一个算子操作prePUType和下一个算子操作nextPUType的处理单元类型以及两个算子操作之间传递数据的预期存储需求expMem. 其次,算法根据前后算子操作的处理单元类型是否相同,采取不同的存储分配优先级,具体地:情况1,当所映射的为同一个运算单元且满足$\exp Mem < freebuf_{puID}$时,将$subOP_i$和$subOP_{i+1}$的数据交换过程添加进selfBuf队列;否则,将$subOP_i$和$subOP_{i+1}$的数据交换过程添加进sharedRAM队列. 情况2,当所映射的为不同运算单元且满足$\exp Mem < freeram_{rID}$时,将$subOP_i$和$subOP_{i+1}$的数据交换过程添加进sharedRAM队列;否则,将$subOP_i$和$subOP_{i+1}$的数据交换过程添加进sharedDDR队列. 同时,在每次队列添加后执行队列管理更新. 通过算法1的存储分配策略,判断计算任务的上下文是否发生跨计算单元的数据交换,以此来合理地进行存储分配决策,尽可能优化数据访问的效率和数据搬运的开销.

算法2　融合算子内部存储管理分配

FOR ($i = 0, i < fusedNewOP\_NUMBER, i++$)

　prePUType = getPUType($subOP_i$)

　nextPUType = getPUType($subOP_{i+1}$)

　exp Mem = getMem Re q($subOP_i$, $subOP_{i+1}$)

　IF (prePUType == nextPUType)

　　IF (exp Mem < $freebuf_{puID}$)

　　　allocate(buf_{puID})

　　　Mem - update($freebuf_{puID}$)

　　ELSE IF (exp Mem < $freeram_{rID}$)

　　　allocate(ram_{rID})

　　　Mem - update($freeram_{rID}$)

　ELSE IF (prePUType ≠ nextPUType)

　　IF (exp Mem < $freeram_{rID}$)

　　　allocate(ram_{rID})

　　　Mem - update($freeram_{rID}$)

　　ELSE

　　　allocate(ddr_{dID})

　　　Mem - update($freeddr_{dID}$)

根据多分支结构特征所带来的远距离数据依赖关系,需要针对融合后的新算子之间进行存储管理二次优化生成,从而实现将算子融合优化扩展至卷积神经网络的完整推理过程,具体如算法3描述.

第一步,生成分支节点集合,计算每个分支节点branchNode与其具有后向依赖关系节点的距离,即统计每一个作为数据源的分支节点s与目的节点d之间的融合新算子个数,记为newOPCnt(s, d). 第二步,统计节点s到其节点d之间的所有算子数量,获取涉及的算子操作列表并遍历这个列表,累加所有操作的内存需求,得到总的存储需求量为$\sum \exp Mem_{s \to d}$. 具体存储分配时,判断只有当$\sum \exp Mem_{s \to d} < freeram_{rID}$时,将当前分支节点branchNode的输出特征数据分配在系统的片上RAM存储区中;否则,将当前分支节点branchNode的输出特征数据分配在系统的片外DDR存储区中.

3　硬件架构设计

非一般类算子融合方法不改变原有算子层依赖关系和运算表达式,是一种更符合计算过程加速推理的算子融合优化调度方法. 为了支持该方法的实现,本文在传统异构计算系统的基础上,设计了一种更加高效的异构计算架构模型和多层级共享存储规划方案,整体架构如图2所示.

算法3 融合算子间存储管理分配

```
FOR (i = 0, i < numBranchNodes, i ++) {
  int s = branchNodes[i]
  for  (d = 0, d < numBranchNodes, d ++){
    if (s == d) continue;
    int newOPCnt = count_new_ops(s, d)
    int totalMem Re q = 0
    int* newOPCnt = count_new_ops(s, d)
    int len = newOps[0]
      for (j = 1; j <= len; j ++)
        totalMem Re q += get_memory_req(newOps[j])
        if (totalMem Re q < freeram_rID)
          allocate_to_ram(s)
        else
          allocate_to_ddr(s)
  }
}
```

3.1 异构计算架构建模

本节将介绍针对异构计算平台的硬件设计方案，包括异构计算核心电路建模与数据传输通路电路建模.

首先，本文提出一种由通用处理器CPU、GPU和深度学习加速器DLA组成的异构算力核心，将优势互补的运算单元进行集成可以有效弥补DLA算子缺失所导致的算子融合率降低问题，为改进后的算子融合算法提供基本的算力支持，提高算子融合的可行性. 通过这样一种系统级的紧耦合架构设计，CPU作为主要过程控制单元，负责完成样本图像的获取和输入、推理过程计算子任务的调度以及推理结果的信息展示；深度学习加速器DLA采用可配置算子模块化设计，负责完成CNN网络卷积、池化等算子的加速计算；GPU采用可配置多核设计，负责完成专用计算单元无法支持的算子计算，如YOLOv5s中的sigmoid、element-wise等运算.

其次，以CPU+GPU+DLA组成的异构计算核心中，为了支持网络推理过程中数据在不同计算单元之间的灵活流动，各运算单元以及存储部件通过交叉总线单元和系统总线形成两级互联. 一方面，通过交叉总线单元(switch)可以建立各功能部件的数据交换通路，支持共享数据在不同计算单元之间的有序调，为确保数据在系统中的流动不会造成冲突，从而优化数据访问延迟. 在设计中，交叉总线单元的带宽需求，如式(10)所示：

$$\exp \mathrm{BW}_{\mathrm{switch}}=\sum_{0}^{\mathrm{PU}_{\mathrm{NUM}-1}} \mathrm{BW}_{\mathrm{puID}} \tag{10}$$

其中，$\mathrm{BW}_{\mathrm{puID}}$是计算单元puID的数据传输带宽；PU_NUM是计算单元的数量.

另一方面，为了在计算单元内部快速访问存储资源，设计中引入了系统总线和RAM存储的连接. 每个计算单元通过系统总线可以访问最靠近的存储区域，从而减少数据访问的延迟. 在设计中，将计算单元puID内部系统总线的带宽需求表示，如式(11)所示：

$$\exp \mathrm{BW}_{\mathrm{sysbus}}^{\mathrm{puID}}=\mathrm{UnitNUM}_{\mathrm{puID}} \times \mathrm{BW}_{\mathrm{local}} \tag{11}$$

其中，$\mathrm{UintNUM}_{\mathrm{puID}}$是计算单元puID中运算部件的数量；$\mathrm{BW}_{\mathrm{local}}$是每个运算部件的本地存储带宽需求.

通过以上设计能够确保数据在不同计算单元之间传输时有效避免出现瓶颈问题，可支持高效的数据流动. 在数据传输过程中，满足“数据即时到达，计算即刻进行”的原则，有助于优化内存访问的时序，使得各计算单元能够在最短的时间内访问所需数据.

3.2 多层级共享存储方案

为了支持前面第2节所提出的算子融合优化方案，同时遵循通过算子融合策略提升系统执行推理任务效率的设计初衷. 本文为异构计算平台设计了多层级共享存储结构，将优化后的融合算子内部计算过程数据交换以及远距离多分支融合算子间的数据交换映射到不同层级的存储区域，如图3所示.

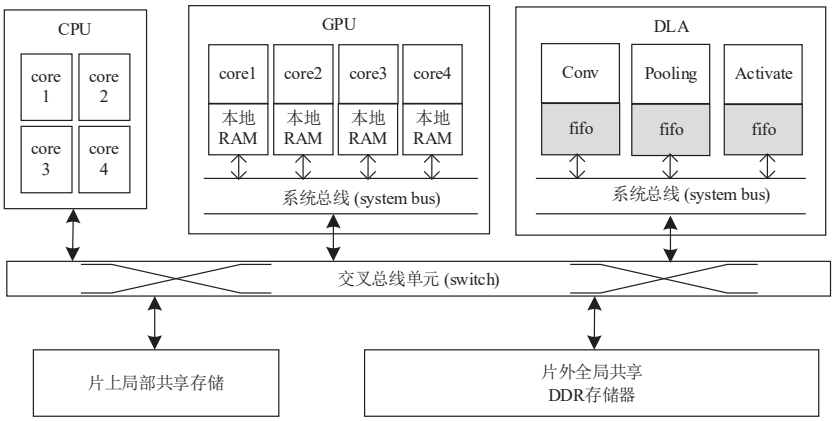


图2 异构计算系统整体架构图

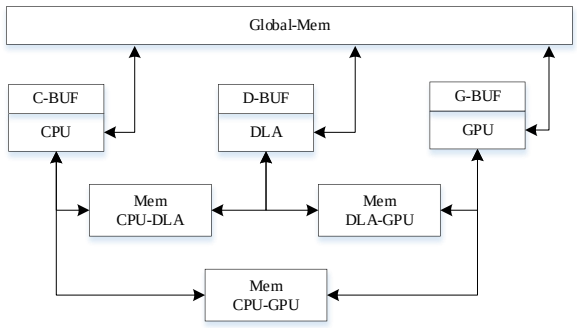


图3　多层级存储建模

首先,存储结构设计是影响计算过程执行效率的主要因素,影响性能的主要因素有访存次数和访存模式,而单一的BRAM(Block Random Access Memory)片上存储体容易受到访问带宽的限制. 因此,构建一种可以支持两两设备间共享数据的存储架构,既可以满足任意相邻计算过程的快速数据共享,又可以增加并行计算过程的同时刻存储访问需求. 具体地,设计中采用可编程逻辑器件内部片内BRAM存储资源,建立异构计算系统核心设备之间的局部共享数据存放区. 以深度学习加速器DLA为例,设计两个片上BRAM存储器$\mathrm{Mem}_{\mathrm{CPU\text{-}DLA}}$、$\mathrm{Mem}_{\mathrm{DLA\text{-}GPU}}$,分别用于DLA与CPU、DLA与GPU的共享数据存放. 直观上看,通过两两设备间共享数据存储区的建立,可以在外部共享存储Global-Mem和本地X-BUF之间建立数据高效传输的桥梁,每一组两两设备间采用独立的访问接口,支持连续的计算过程在运行中实现数据的预先存取,可以提高带宽、吞吐量及流水计算的执行效率.

其次,将各局部共享存储区进行统一编址管理,从而减少内部总线互联复杂度. 共享存储区大小分配采用参数化设计,用参数$\mathrm{CMem}_{[\mathrm{cpu\text{-}dla}]}$、$\mathrm{CMem}_{[\mathrm{dla\text{-}gpu}]}$和$\mathrm{CMem}_{[\mathrm{cpu\text{-}gpu}]}$分别表示异构计算系统三个共享区的存储需求,具体存储容量分配取决于核心计算过程的中间特征数据量以及计算所需的权重等参数数据. 对于面向深度学习加速应用的场景计算需求,int8和int16是首选数据类型,因此,BRAM位宽$\mathrm{BW}_{\mathrm{bram}}$为8 bit或16 bit.

存储器空间地址按1 K空间对齐,即BRAM低位地址固定为addLow=[9: 0].

存储器局部片选用存储器空间地址高位判断,因此,BRAM高位地址为addHigh=[AWH−1: 10]. 其中,AWH是存储器高位地址总位宽,计算表达式如下:

$$\mathrm{AWH}=\left\lceil \log_2 \mathrm{CMem}_{[\mathrm{cpu-dla}]} \right\rceil+\left\lceil \log_2 \mathrm{CMem}_{[\mathrm{dla-gpu}]} \right\rceil+\left\lceil \log_2 \mathrm{CMem}_{[\mathrm{cpu-gpu}]} \right\rceil.$$

图4所示是本文所提出的算子融合方法对应的存储分配与数据流模型,这里假设输入数据input是前一个运算过程的输出,由于它是一个远距离分支的中间特征数据,因此存放在片外$\mathrm{DDR}_{\mathrm{Mem}}$中. 经过融合算法后Conv→Sig mo id→ElewiseMul形成一个完整的跨计算单元的新算子,其中计算过程Conv映射在DLA设备中执行、Sigmoid和ElewiseMul映射在GPU设备中执行;中间特征数据①和中间特征数据④分配在片上局部存储器$\mathrm{BramMem}_{[\mathrm{dla\text{-}gpu}]}$,中间特征数据②和中间特征数据③分配于本地运算单元的内部BUF缓冲存储区中. 同理,对于输出特征数据仍需进一步根据它的后继节点判断决定其存储位置. 如此设计的优势在于:(1)对于跨计算单元的运算过程,根据数据量和访问频率进行判断,可以更有效地利用有限的片上高速存储资源;(2)在同一个运算单元内部,高效利用本地内部BUF存储区,进一步减少数据访问延迟,有助于提高整体存储资源利用率;(3)通过合理的存储分配,达到优化数据流传输路径的目的,明显减少了依赖CPU进行数据调用和数据同步的时间开销,有助于保持计算过程的连续性,提高整体推理过程运算效率.

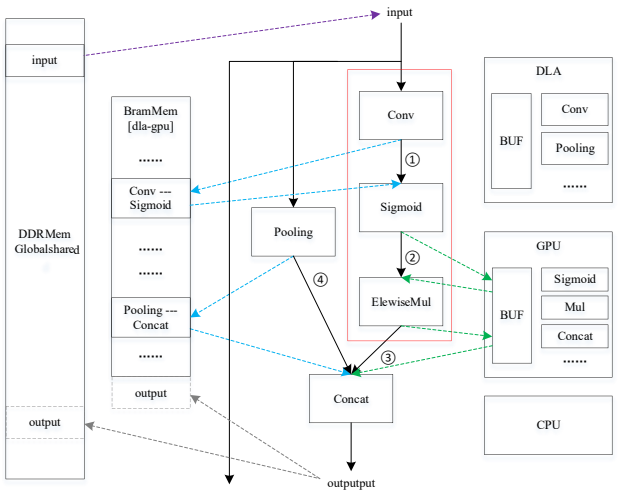


图4　算子融合过程中的存储映射分配与数据流

4　实验

为了证明本文提出的非一般类算子融合算法的有效性和硬件架构设计方法的可行性,实验设计时选用具有普遍性和代表性的计算机视觉深度学习网络模型进行对比实验,通过消融实验验证非一般类算子融合策略与具有多层级存储共享结构的异构计算系统的协同效果,具体实验设计如下:

(1)基准实验,采用传统算子融合策略的Tengine框架和传统异构计算系统.

(2)算法优化实验,采用本文提出的非一般类算子融合策略和传统异构计算系统.

(3)硬件优化实验,采用传统算子融合策略的Tengine框架和具有多层级存储共享结构的异构计算

系统.

(4)协同优化实验,采用本文提出的非一般类算子融合策略和具有多层级存储共享结构的异构计算系统.

4.1 实验平台建立

本文使用可配置性、灵活性更强的FPGA器件作为硬件开发平台主体,FPGA则选择Xilinx公司ZYNQ系列芯片.

首先,采用FPGA内嵌ARM软核作为异构计算系统中的CPU处理器,使用Vivado开发环境,通过IP核技术调用FPGA内部集成硬件,选择开源硬件设计GPU处理器和深度学习加速器,将GPU和DLA两种计算单元集成在一块FPGA逻辑器件内部. 其中,深度学习加速器选用Nvidia公司开源的NVDLA设计,数据精度选择8位定点数进行测试,核心的PE阵列大小配置为32×8的MAC阵列形式,卷积运算单元CBUF缓存的Bank数量为32、CBUF缓存的Bank宽度为32、CBUF缓存的Bank深度为128;GPU选用佐治亚理工大学开源的Vortex GPU,是一款基于RISC-V指令集架构的开源通用图形处理单元(GPGPU),配备1个计算集群(Cluster),每个Cluster内包含4个计算核心(Core),每个计算核心支持4个Warp调度单元,每个Warp包含4个线程(Thread),L2 Cache缓存大小为128 KB、L3 Cache缓存大小为1 MB. 其次,片内局部共享存储器选择FPGA芯片内部集成的BRAM,总容量为1 MB;片外全局共享存储器选择外部DDR存储器,总容量1 GB. 再次,在FPGA内部选择使用AXI总线将各个运算单元和存储区互联. 最后,为了更精准地统计推理过程的时间开销,本文专门设计硬件性能评测模块进行时间开销统计.

4.2 消融实验测试情况

为验证本文提出的优化方法,实验选取了具有代表性的目标检测网络YOLOX-Nano和YOLOv5s. 在统一硬件环境下(CPU主频1.6 GHz,NVDLA、GPU、存储器访问以及AXI总线等FPGA部分逻辑设计工作频率200 MHz),设计并开展了针对算法优化与硬件结构优化的消融实验. 实验对比涵盖了基准、单独算法优化、单独硬件优化以及协同优化四种模式,并且对各实验模式分别进行了数据结果统计.

(1)基准实验

基准实验采用传统算子融合策略的Tengine框架和传统异构计算系统,在该模式下,YOLOX-Nano网络模型的子图数量为63个,YOLOv5s网络模型的子图数量为241个. 分别统计执行目标检测推理的结果数据统计如表1所示,具体地,执行YOLOX-Nano网络执行总时间为1.318 s,其中,异构计算推理用时为1.135 s;YOLOv5s网络执行总时间为18.369 s,其中,异构计算推理用时为15.088 s.

(2)算法优化实验

算法优化实验采用本文提出的非一般类算子融合策略和传统异构计算系统,在该模式下,对于YOLOX-Nano网络模型子图数量可由63个减少至13个,实现算子融合覆盖率为79.4%;对于YOLOv5s网络模型子图数量可由241个减少至58个,实现算子融合覆盖率为75.1%. 分别统计执行目标检测推理的结果数据统计如表1所示,具体地,执行YOLOX-Nano网络执行总时间为0.596 s,其中,异构计算推理用时为0.482 s;YOLOv5s网络执行总时间为7.654 s,其中,异构计算推理用时为4.554 s.

(3)硬件优化实验

硬件优化实验采用传统算子融合策略的Tengine框架和具有多层级存储共享结构的异构计算系统,在该模式下,各模型子图的中间特征数据可以在局部高效共享存储区进行缓存. 分别统计执行目标检测推理的结果数据统计如表1所示,具体地,执行YOLOX-Nano网络执行总时间为1.313 s,其中,异构计算推理用时为1.130 s;YOLOv5s网络执行总时间为18.318 s,其中,异构计算推理用时为15.079 s.

(4)协同优化实验

协同优化实验采用本文提出的非一般类算子融合策略和具有多层级存储共享结构的异构计算系统,在该模式下已形成完整的算法优化与硬件优化协同. 分别统计执行目标检测推理的结果数据统计如表1所示,具体地,执行YOLOX-Nano网络执行总时间为0.538 s,其中,异构计算推理用时为0.424 s;YOLOv5s网络执行的总时间为7.491 s,其中,异构计算推理用时为4.360 s.

表1 消融实验结果数据统计 单位:s

时间	YOLOX-Nano				YOLOv5s			
	基准	算法优化	硬件优化	协同优化	基准	算法优化	硬件优化	协同优化
推理	1.135	0.482	1.130	0.424	15.088	4.554	15.079	4.360
总执行	1.318	0.596	1.313	0.538	18.369	7.654	18.318	7.491

4.3 整体性能分析

根据消融实验结果数据统计,对比基准实验和协同优化实验的性能变化,结果如表2所示.

在YOLOX-Nano网络中,算子融合前模型子图数量63个,采用本文提出的优化后的算子融合策略,主干推理网络子图数量可减少至13个,实现算子融合覆盖率为79.4%. 根据实际实验测量数据的统计结果,

YOLOX-Nano网络执行总体性能提升为59.22%,推理计算部分的性能提升为62.67%,整体运行加速比为2.45,推理运算部分加速比为2.68. 分别统计每个异构调度子图的时间数据,经计算得出,在优化异构调度策略作用下,融合后的13个跨计算单元调度子图推理过程性能均有提升,平均每个跨计算单元调度子图的计算耗时可减少71.54%.

在YOLOv5s网络中,算子融合前模型子图数量241个,采用本文提出的优化后的算子融合策略,主干推理网络子图数量可减少至58个,实现算子融合覆盖率为75.1%. 根据实际实验测量数据的统计结果,YOLOv5s网络执行总体性能提升为59.22%,推理计算部分的性能提升为71.10%,整体运行加速比为2.45,推理运算部分加速比为3.46. 分别统计每个异构调度子图的时间数据,经计算得出,在优化异构调度策略作用下,融合后的58个跨计算单元调度子图推理过程性能均有提升,平均每个跨计算单元调度子图的计算耗时可减少70.66%. 由此可见,采用本文提出的非一般类算子融合策略以及具有多层级存储共享结构的异构计算系统可有助于卷积神经网络整体运算加速的实现.

表2　算子融合性能实验结果数据统计

模型	总性能提升/%	推理性能提升/%	子图性能提升/%	整理加速比	推理加速比
YOLOX-Nano	59.22	62.67	71.54	2.45	2.68
YOLOv5s	59.22	71.10	70.66	2.45	3.46

4.4　仿真性能分析

经仿真分析,如表3所示,在保持CPU主频不变的情况下,将NVDLA、GPU、存储器访问以及AXI总线等FPGA部分逻辑设计的工作频率假设提升至1 GHz,以此作为未来在ASIC实现中可能达到性能水平的推算与评估. 在该设定下,对于YOLOX-Nano的所有Tensor数据均可以分配在高速的片内存储空间中,主频提升的作用效果能够体现在全部特征提取子图中. 因此,推理一个完整YOLOX-Nano网络,总运行时间可缩短至0.283 s,异构计算推理用时可缩短至0.088 s. 对于YOLOv5 s而言,仅有部分Tensor数据可以分配在高速片内存储空间中,主频提升的效果只作用于部分特征提取计算子图. 因此,推理一个完整YOLOv5s网络,总运行时间可缩短至5.566 s,异构计算推理用时可缩短至2.434 s.

表3　主频1 GHz仿真性能分析　　单位:s

模型	推理时间	执行总时间
YOLOX-Nano	0.088	0.283
YOLOv5s	2.434	5.566

5　结论

本文围绕提升网络模型推理加速性能展开研究. 首先,本文分析了传统算子融合算法的局限性以及影响算子融合效率的因素,提出一种非一般类的改进型算子融合方法,使其可以实现跨计算单元执行计算过程合并优化,提高算子融合覆盖率;该融合方法对任意相互关联的算子层具有一定通用性,即适用于网络模型中具有前后依赖关系且需要在计算单元之间实现快速数据交换的运算过程. 其次,本文在传统异构计算基本架构基础之上,针对端侧异构计算平台进行了多层级存储结构建模,为了可以更好地支持改进算子融合策略,在可编程逻辑器件FPGA平台上进行了真实异构计算平台搭建,通过共享存储与管理等逻辑功能设计使得优化的算子融合策略与硬件计算架构更加契合. 实验结果证明,本文提出的算子融合优化算法和改进后的异构计算架构,可以减少对片外存储器的访问依赖,有助于降低加速计算部件的运算过程执行时间,从而有效减少深度学习算法模型的推理耗时. 实验结果表明,针对YOLOX-Nano的完整推理过程可实现59.22%的计算速度提升,其中,推理计算性能提升为62.67%;针对YOLOv5s的完整推理过程可实现59.22%的计算速度提升,其中,推理计算性能提升为71.10%. 文章以卷积神经网络运算为核心的目标检测任务为例展开研究,但本文提出的方法不局限于此项单一任务,同样可适用于语义分割、时序预测以及以Transformer为代表的其他任务网络.

参考文献

[1] VOULODIMOS A, DOULAMIS N, DOULAMIS A, et al. Deep learning for computer vision: A brief review[J]. Computational Intelligence and Neuroscience, 2018, 2018: 7068349.

[2] DIWAN T, ANIRUDH G, TEMBHURNE J V. Object detection using YOLO: Challenges, architectural successors, datasets and applications[J]. Multimedia Tools and Applications, 2023, 82(6): 9243-9275.

[3] ZHANG J Q, CHEN X R, SONG M C, et al. Eager pruning: Algorithm and architecture support for fast training of deep neural networks[C]//2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA). Piscataway: IEEE, 2019: 292-303.

[4] BOCHKOVSKIY A, WANG C Y, LIAO H M. YOLOv4: Optimal speed and accuracy of object detection[EB/OL]. (2020-04-23)[2025-03-21]. https://arXiv.org/abs/2004.10934.

[5] FLYNN J, NEULANDER I, PHILBIN J, et al. Deep stereo: Learning to predict new views from the world's imag-

ery[C]//2016 IEEE Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE, 2016: 5515-5524.

[6] KWON H, LAI L Z, PELLAUER M, et al. Heterogeneous dataflow accelerators for multi-DNN workloads[C]//2021 IEEE International Symposium on High-Performance Computer Architecture. Piscataway: IEEE, 2021: 71-83.

[7] XU H, NAZHAMAITI M, LIU Y D, et al. Utilizing direct photocurrent computation and 2D kernel scheduling to improve in-sensor-processing efficiency[C]//2020 57th ACM/IEEE Design Automation Conference. Piscataway: IEEE, 2020: 1-6.

[8] VILIM M, RUCKER A, OLUKOTUN K. Aurochs: An architecture for dataflow threads[C]//2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture. Piscataway: IEEE, 2021: 402-415.

[9] JOUPPI N P, YOON D H, ASHCRAFT M, et al. Ten lessons from three generations shaped google's TPUv4i: Industrial product[C]//2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture. Piscataway: IEEE, 2021: 1-14.

[10] NORRIE T, PATIL N, YOON D H, et al. The design process for google's training chips: TPUv2 and TPUv3[J]. IEEE Micro, 2021, 41(2): 56-63.

[11] ZHAO K J, WANG J, ZANG D. A convolutional neural network accelerator based on NVDLA[C]//2021 The 5th International Conference on Algorithms, Computing and Systems. New York: ACM, 2021: 43-47.

[12] ZHANG J Q, CHEN X R, RAY S. GCONV chain: Optimizing the whole-life cost in end-to-end CNN acceleration[C]//IEEE Transactions on Computers. Piscataway: IEEE, 2021: 2300-2312.

[13] REDMON J, FARHADI A. YOLOv3: An incremental improvement[EB/OL]. (2018-04-08)[2025-03-21]. https://arXiv.org/abs/1804.02767.

[14] RODRIGUEZ-FERRANDEZ I, TALI M, KOSMIDIS L, et al. Sources of single event effects in the NVIDIA Xavier SoC family under proton irradiation[C]//2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design. Piscataway: IEEE, 2022: 1-7.

[15] ZHAO Y W, LIU C, DU Z D, et al. Cambricon-Q: A hybrid architecture for efficient training[C]//2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture. Piscataway: IEEE, 2021: 706-719.

[16] DAS SHARMA D, PASDAST G, QIAN Z G, et al. Universal chiplet interconnect express (UCIe): An open industry standard for innovations with chiplets at package level[J]. IEEE Transactions on Components, Packaging and Manufacturing Technology, 2022, 12(9): 1423-1431.

[17] NIU W, GUAN J X, WANG Y Z, et al. DNNFusion: Accelerating deep neural networks execution with advanced operator fusion[C]//Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. New York: ACM, 2021: 883-898.

[18] CAI X Y, WANG Y, ZHANG L. Optimus: An operator fusion framework for deep neural networks[J]. ACM Transactions on Embedded Computing Systems, 2023, 22(1): 1-26.

[19] PARK J, NAUMOV M, BASU P, et al. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications[EB/OL]. (2018-11-29)[2025-03-21]. https://arXiv.org/abs/1811.09886.

[20] SHUVO M M H, ISLAM S K, CHENG J L, et al. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review[J]. Proceedings of the IEEE, 2023, 111(1): 42-91.

[21] YANG Y, DIAO B Y, LIU H D, et al. An evolutionary search-based operator fusion method with binary representation for deep learning inference acceleration[M]//Pattern Recognition. Cham: Springer Nature Switzerland, 2024: 32-45.

[22] CHEN T, MOREAU T, JIANG Z, et al. TVM: An automated end-to-end optimizing compiler for deep learning[C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). California: USENIX Association, 2018: 578-594.

[23] XING Y, LIANG S, SUI L Z, et al. DNNVM: End-to-end compiler leveraging heterogeneous optimizations on FPGA-based CNN accelerators[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(10): 2668-2681.

[24] JEONG E, KIM J, TAN S, et al. Deep learning inference parallelization on heterogeneous processors with TensorRT[J]. IEEE Embedded Systems Letters, 2022, 14(1): 15-18.

[25] OPEN AI LAB. Tengine[EB/OL]. (2021-10-08)[2025-03-21]. https://github.com/OAID/Tengine.

作者简介

王　莹　女,1984年5月出生于北京市.现为首都师范大学数学科学院博士研究生.主要研究方向为深度学习视觉算法、异构计算系统与领域专用加速器.

E-mail: wangyingstudio@163.com

高　岚　女,1987年8月出生于河北省涿州市.现为首都师范大学信息工程学院副教授,硕士生导师.主要研究方向为计算机体系结构及并行编程优化,并行编程模型,以及计算机片上存储系统的设计及优化.

E-mail: gaolan@cnu.edu.cn

张　哲　男,2002年1月出生于山西省临猗市.现为山西农业大学软件学院本科生.主要研究方向为软件工程.

E-mail: zzpaxli@163.com

刘　昕　男,1998年1月出生于黑龙江省齐齐哈尔市.现为首都师范大学信息工程学院硕士研究生.主要研究方向为计算机体系结构.

E-mail: lxemail163@163.com

武毅雄　男,1997年1月出生于山西省运城市.现为首都师范大学信息工程学院硕士研究生.主要研究方向为计算机体系结构.

E-mail: wuyixiong163@163.com

张伟功　男,1967年4月出生于山西省临猗市.现为首都师范大学信息工程学院教授,博士生导师.主要研究方向为高可靠嵌入式计算机体系结构、异构计算系统与深度学习算法加速计算.

E-mail: zwg771@cnu.edu.cn