面向 YOLO 神经网络的数据流架构优化研究

穆宇栋[1),2)] 李文明[1),2)] 范志华[1),2)] 吴 萌[1),2)] 吴海彬[1),2)]
安学军[1)] 叶笑春[1),2)] 范东睿[1),2)]

[1)](处理器芯片全国重点实验室(中国科学院计算技术研究所) 北京 100190)
[2)](中国科学院大学计算机科学与技术学院 北京 100049)

摘 要 YOLO 目标检测算法具有速度快、精度高、结构简单、性能稳定等优点,因此在多种对实时性要求较高的场景中得到广泛应用。传统的控制流架构在执行 YOLO 神经网络时面临计算部件利用率低、功耗高、能效较低等挑战。相较而言,数据流架构的执行模式与神经网络算法匹配度高,更能充分挖掘其中的数据并行性。然而,在数据流架构上部署 YOLO 神经网络时面临三个问题:(1)数据流架构的数据流图映射并不能结合 YOLO 神经网络中卷积层卷积核较小的特点,造成卷积运算数据复用率过低的问题,并进一步降低计算部件利用率;(2)数据流架构在算子调度时无法利用算子间结构高度耦合的特点,导致大量数据重复读取;(3)数据流架构上的数据存取与执行高度耦合、串序执行,导致数据存取延迟过高。为解决这些问题,本文设计了面向 YOLO 神经网络的数据流加速器 DFU-Y。首先,结合卷积嵌套循环的执行模式,本文分析了小卷积核卷积运算的数据复用特征,并提出了更有利于执行单元内部数据复用的数据流图映射算法,从而整体提升卷积运行效率;然后,为充分利用结构耦合的算子间的数据复用,DFU-Y 提出数据流图层次上的算子融合调度机制以减少数据存取次数、提升神经网络运行效率;最后,DFU-Y 通过双缓存解耦合数据存取与执行,从而并行执行数据存取与运算,掩盖了程序间的数据传输延迟,提高了计算部件利用率。实验表明,相较数据流架构(DFU)和 GPU(NVIDIA Xavier NX),DFU-Y 分别获得 2.527 倍、1.334 倍的性能提升和 2.658 倍、3.464 倍的能效提升;同时,相较 YOLO 专用加速器(Arria-YOLO),DFU-Y 在保持较好通用性的同时,达到了其性能的 72.97%、能效的 87.41%。

关键词 YOLO 算法;数据流架构;数据流图优化;卷积神经网络;神经网络加速
中图法分类号 TP301 **DOI 号** 10.11897/SP.J.1016.2025.00082

Optimization of Dataflow Architecture for YOLO Neural Networks

MU Yu-Dong[1),2)] LI Wen-Ming[1),2)] FAN Zhi-Hua[1),2)] WU Meng[1),2)] WU Hai-Bin[1),2)]
AN Xue-Jun[1)] YE Xiao-Chun[1),2)] FAN Dong-Rui[1),2)]

[1)](State Key Lab of Processors(Institute of Computing Technology, Chinese Academy of Sciences), Beijing 100190)
[2)](School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049)

Abstract The YOLO(You Only Look Once) Object Detection Algorithm stands out in the realm of computer vision for its remarkable speed, accuracy, and straightforward architecture. It's particularly favored in applications where real-time processing is crucial such as Autonomous Driving and Vehicle Detection. Despite these advantages, implementing YOLO neural networks on traditional control flow architectures presents significant drawbacks, including underutilization of computational resources, excessive energy consumption, and poor power efficiency. By contrast,

收稿日期:2023-10-18;在线发布日期:2024-06-20。本课题得到北京市科技新星计划资助(20220484054,20230484420)、北京市自然科学基金-昌平创新联合基金资助项目(L234078)、中国科学院青年创新促进会资助。**穆宇栋**,硕士研究生,中国计算机学会(CCF)学生会员,主要研究方向为数据流架构、数据流图映射及可重构架构。E-mail:muyudong19@mails.ucas.ac.cn。**李文明**,博士,副研究员,中国计算机学会(CCF)会员,主要研究领域为高通量处理器架构、数据流体系结构。**范志华**(通信作者),博士,助理研究员,中国计算机学会(CCF)会员,主要研究领域为数据流体系结构、可重构智能架构。E-mail:fanzhihua@ict.ac.cn。**吴 萌**,博士研究生,主要研究方向为数据流架构及高通量计算机架构。**吴海彬**,博士研究生,主要研究方向为数据流架构及高通量计算机架构。**安学军**,博士,正高级工程师,中国计算机学会(CCF)会员,主要研究方向为图处理架构及高性能互联网络。**叶笑春**,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为高通量计算机架构及软件模拟。**范东睿**,博士,研究员,中国计算机学会(CCF)会员,主要研究领域为高通量计算机架构及高性能计算机架构。

dataflow architectures offer a promising alternative. The execution pattern of the dataflow architecture allows an operation to be executed once its operators are ready, which are inherently aligned with the operational demands of neural network algorithms, enabling more effective exploitation of data parallelism and reducing the redundant data transfers. Nevertheless, adapting YOLO neural networks to conventional dataflow architectures will confront with three key issues: (1) YOLO neural networks are distinguished by their reliance on convolution operations that utilize small kernels. However, traditional dataflow architectures, with their inflexible Data Flow Graph (DFG) designs, are incompetent to handle these operations efficiently. This incompatibility results in a diminished rate of data reuse, consequently leading to suboptimal utilization of computational resources. (2) The execution mechanism between operators does not effectively utilize the linked structures of specific layers in YOLO neural networks. This is particularly evident in the interaction between the convolution and activation layers, where recurrent data transfers between the Processing Element (PE) array and on-chip memory lead to substantial execution overhead and diminished energy efficiency. (3) The inherent limitations of traditional dataflow architectures, which perform data transfer and processing in a bounded and sequential manner, result in significant delays. To overcome these obstacles, we present an enhanced dataflow architecture and introduce a YOLO accelerator, designated as DFU-Y. This solution encompasses three key strategic improvements: First, through a detailed analysis of the execution patterns of nested convolution loops, we have crafted an innovative Data Flow Graph (DFG) mapping algorithm with a particular focus on the challenges posed by convolutions with small kernels. This algorithm is meticulously designed to optimize the parallelization patterns across various dimensions within dataflow architectures, thereby enhancing data reuse at the level of individual Processing Elements and boosting computational efficiency. Second, DFU-Y integrates an operator fusion mechanism into the data flow graph. This innovative mechanism classifies operation nodes and strategically recouples operations with data dependencies. The aim is to maximize data reuse among interconnected operators and reduce superfluous data transfers. Third, to surmount the constraints of sequential data transfer and processing, DFU-Y utilizes a double buffering mechanism. This approach effectively separates data transfer from execution, enabling balanced task partitioning that mitigates data transfer delays and augments the utilization of computing units. Experiments show that compared with the dataflow architecture (DFU) and GPU (NVIDIA Xavier NX), DFU-Y achieves 2.527× and 1.334× of performance improvement and 2.658× and 3.464× of energy efficiency improvement respectively. Compared to the dedicated YOLO accelerator (Arria-YOLO), DFU-Y obtains 72.97% of acceleration rate and 87.41% of energy efficiency with good versatility. In end-to-end experiments, DFU-Y achieves an average energy efficiency of 7.11 FPS/W on different YOLO models.

Keywords　YOLO algorithm; dataflow architecture; optimization of data flow graph; convolutional neural network; neural network acceleration

1　引　言

目标检测是计算机视觉领域中一个有挑战的任务，其可以看作目标识别与目标定位的结合[1]。由于目标数量不定、大小不一、位置未知，目标检测往往较为复杂且实时性较差。近年来，YOLO 目标检测算法很好地适应了各种实时性要求较高的应用场景[2]，其全称为 You Only Look Once: Unified, Real-Time Object Detection，最早于 2016 年被提出[3]。该目标检测算法仅使用一个神经网络直接检测目标位置与类别，在精度允许范围内，目标识别速

度获得明显提升。

YOLO 神经网络最核心的特征在于网络层数少、算子种类相对单一、结构较为稳定。此外，相比于传统的卷积神经网络，YOLO 神经网络中卷积核大小为 1 的卷积层重复次数最多、运行时长占比极大，由此带来卷积数据可复用性差、计算部件利用率低等问题。

为加速 YOLO 神经网络，多种加速架构被提出。图形处理器（Graphics Processing Unit，GPU）平台[4-6]虽然可以通过大量的计算单元并行处理矢量数据从而加速神经网络的计算，但是受限于计算资源利用率较低、功耗较大，此类加速方法的能效表现普遍较差。面对新型算法的发展及其对计算资源需求的提高，现场可编程门阵列（Field Programmable Gate Array，FPGA）平台越来越受到学术界和工业界的青睐。FPGA[7-11]通过比特级的查找表（Look-Up Table，LUT）单元实现了强可重构性，并在运行速度与能效上取得良好表现。然而，比特级重构代价高昂，例如编译开销大、时钟频率低、开发成本高等[12-13]。相比之下，数据流架构[14-18]通过指令级可重构在灵活性和能效上取得了较好的平衡。

数据流计算思想天然与神经网络的执行方式相似，随着深度神经网络的发展，数据流架构也成为了研究热点[19-22]。在数据流架构中，计算任务通过数据流图表示，节点表示运算模块，节点间的有向边表示计算节点间的数据依赖。计算节点被触发执行的条件为其需要的所有操作数均已就绪[23]。得益于此种特殊的执行方式，相较于传统控制流架构，数据流架构不仅有利于挖掘数据级并行性，而且在通用性与能效表现间取得较好的平衡[24]。然而，由于 YOLO 神经网络卷积核大小较为特殊且算子间存在结构耦合，当前数据流架构在加速 YOLO 神经网络时，仍存在以下问题：首先，传统数据流架构对 YOLO 神经网络中运行时长占比较大的卷积核大小为 1 的卷积算子的数据流图映射往往不能有效挖掘计算部件内的数据复用，从而直接降低了计算部件利用率；其次，YOLO 神经网络中的算子间所具有的卷积与激活算子间的耦合关系常常被忽略，从而造成了算子间重复的数据存取，增加了数据传输开销；此外，由于硬件结构限制，当前数据流架构上的数据存取与计算串序执行，从而浪费了大量计算资源，降低了程序运行时的性能功耗比。

为解决上述问题，本文面向 YOLO 神经网络，对数据流架构进行了优化。本文分别从卷积数据流图映射、算子调度、访存部件设计三个方面展开研究，提出了面向 YOLO 神经网络的数据流架构加速器 DFU-Y（Dataflow Unit for YOLO），主要贡献包括：

（1）为提高 YOLO 神经网络中卷积核大小为 1 的卷积运算在执行单元内的数据复用，本文提出了一种针对性的数据流图映射算法，显著提高了计算部件利用率，减少了数据传输开销；

（2）为充分挖掘 YOLO 神经网络中算子间的结构耦合，本文提出数据流架构上的算子融合调度机制，将具有数据相关性的算子在数据流图层次上共同实现，从而灵活利用算子间的数据复用，减少数据访存；

（3）为提高计算部件利用率，掩盖因硬件结构导致的数据传输延迟，本文采用数据流架构上的双缓存存取-执行解耦合机制，以减少数据传输时长；

（4）实验表明，在小卷积核卷积运算中，相比同精度数据流架构 DFU[10]与 GPU（NVIDIA Xavier NX），DFU-Y 分别获得 2.527 倍、1.334 倍的性能提升和 2.658 倍、3.464 倍的能效提升；相较于 YOLO 专用加速器（Arria-YOLO），DFU-Y 在保持良好通用性的同时取得了 72.97%的加速比和 87.41%的能效。在端到端实验中，DFU-Y 在不同 YOLO 模型上取得了 7.11 FPS/W 的平均能效。

本文的结构为：第 2 节介绍本文的相关工作；第 3 节介绍本文的研究动机；第 4 节介绍本文提出的优化方案；第 5 节介绍实验相关配置；第 6 节对实验结果进行分析；第 7 节总结本文。

2 相关工作

本节将从 YOLO 算法特点分析、YOLO 中卷积算子特征研究、YOLO 加速器研究现状与数据流架构 DFU 四个方面介绍本文的相关工作。

2.1 YOLO 算法特点

相较于传统卷积神经网络，YOLO 神经网络在网络层组成与卷积核大小规模上具有显著不同，如表 1 所示。

表 1 常见卷积神经网络对比表

卷积神经网络	核心网络层	卷积层的卷积核大小
Lenet[25]	卷积层，最大池化层，全连接层	5×5
AlexNet[26]	卷积层，全连接层，最大池化层等	3×3，5×5，11×11
VGGNet[27]	卷积层，全连接层，最大池化层等	3×3
YOLO[28]	卷积层，最大池化层等	1×1，3×3，6×6

首先，YOLO 神经网络的网络层种类相对单一。不同于其余常见卷积神经网络，YOLO 神经网络的主体结构中不依赖复杂全连接层，相较而言卷积层重复次数最多、在所有层的运行时长中占比极大。

其次，YOLO 神经网络中的卷积运算的卷积核规模具有显著特征，即卷积核只有三种大小，且其中卷积核大小为 1 的卷积运算出现次数远远多于其余规模，并在运行时长中占比最大。以应用广泛的 YOLOv5s 为例，在 NVIDIA Jetson Xavier NX 8 GB 平台上，统计 YOLO 神经网络中关键算子的执行时长占比，结果如图 1 所示。图 1 中，Conv2d_1、Conv2d_3、Conv2d_6 分别指卷积核大小为 1、3、6 的二维卷积运算。根据实验结果，卷积核大小为 1 的卷积运算占 YOLO 神经网络运行总时长的近一半，远远高于其余算子的执行时长。而激活函数如 SiLU 算子的时间占比仅次于卷积，运行时长占比也高于 10%。因此，卷积核大小为 1 的卷积运算与激活算子是 YOLO 神经网络推理过程中的主要瓶颈。

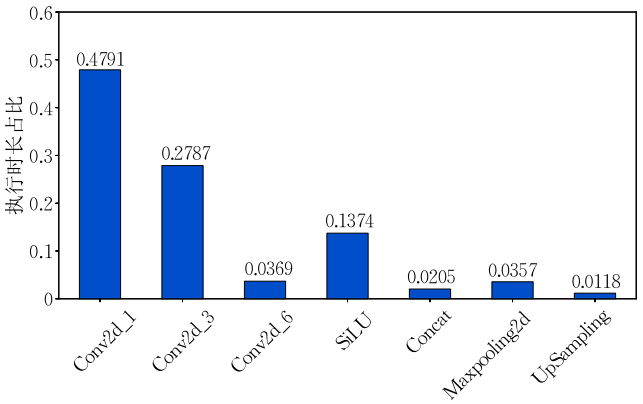


图 1　YOLO 算子运行时长占比图

最后，卷积运算与激活函数如 SiLU 算子在结构上高度耦合。卷积运算的结果常常作为 SiLU 算子的输入，如图 2 所示。提升两者的运行效率对于减少 YOLO 神经网络的运行时长具有重要意义。

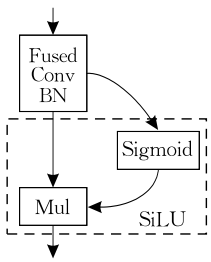


图 2　YOLO 神经网络中的算子耦合结构图

2.2　YOLO 卷积算子

如前文所述，YOLO 神经网络中，卷积核大小为 1 的卷积运算是制约其运行效率最关键的因素。相较于其余卷积核规模的卷积运算，当卷积核大小为 1 时，卷积的计算模式将发生显著变化，并由此带来新的计算特征。下面将分别介绍通用卷积运算与卷积核大小为 1 的卷积运算。

神经网络中的通用二维卷积运算公式如式(1)。式(1)中，各参数的含义如表 2 所示。

$$O[y][x][a][b]=\sum_{z=0}^{C-1}\sum_{i=0}^{h-1}\sum_{j=0}^{w-1}I[x][z][a\times s_h+i][b\times s_w+j]\times W[y][z][i][j]+B,$$
$$0\le y<M,\ 0\le x<N,\ 0\le a<P,\ 0\le b<Q,$$
$$P=\frac{H-h}{s_h}+1,\ Q=\frac{W-w}{s_w}+1 \tag{1}$$

表 2　卷积运算公式中的参数含义

参数	含　义
O	输出特征图像张量(Output Feature Image Tensor)
I	输入特征图像张量(Input Feature Image Tensor)
W	卷积核张量(Kernel Tensor)
C	输入图像通道(Channel) 数
N	输入图像组数(Batch Size)
H/W	输入图像高/宽(Input Image Height/Width)
M	卷积核组数
B	卷积偏置
h/w	卷积核高/宽(Kernel Height/Width)
s_h/s_w	高/宽上的偏移步长(Stride)
P/Q	输出图像高/宽(Output Image Height/Width)

将二维卷积运算的过程图像化，如图 3 所示。图 3 中，输入特征图像的通道与卷积核的对应通道相乘并累加，其结果作为输出特征图像的一个数据点，而不同卷积核对应不同输出特征图像通道。于是，卷积核通道数与输入特征图像通道数相同，卷积核组数与输出特征图像通道数相同，而输入、输出图像组数相同。考虑到步长，输出特征图像的宽高如式(1)中所示。

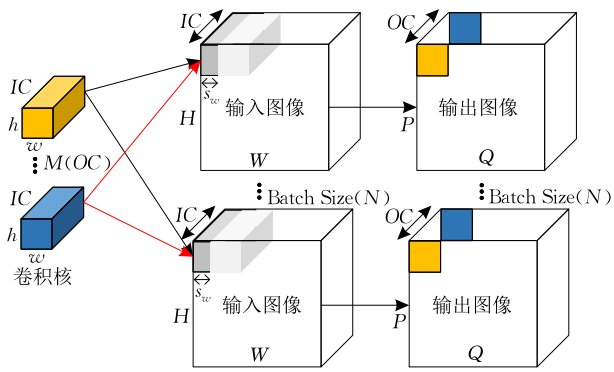


图 3　卷积运算图

对于卷积核大小为 1、输入图像组数为 1 的卷积运算，取式(1)中 $h=w=N=1$，相应卷积步长也均为 1，即 $s_h=s_w=1$。此时，卷积核大小为 1 的卷积

运算公式如式(2)。

$$O[y][a][b]=\sum_{z=0}^{C-1} I[z][a][b]\times W[y][z]+B,$$
$$0\leq y<M,\ 0\leq a<P,\ 0\leq b<Q,$$
$$P=H,\ Q=W \tag{2}$$

分析式(2)发现,对于卷积核大小为 1 的卷积运算,只需要在输入图像高(IH)、输入图像宽(IW)、输入通道数(IC)、输出通道数(OC)四个维度上进行循环乘加运算,即可得到最终输出特征图的结果。进一步地,由于卷积核大小为 1,输入图像高、宽维度上不具有数据相关性,因此可以将这两个维度上的循环合并为一个维度 IHW,此时卷积只需要在 IHW、IC、OC 三个维度上循环乘加运算即可得到最终结果。

此外,卷积核大小为 1 时,由于输入图像宽、高维度上将不存在数据相关,那么该维度上也将不存在数据复用的可能性。然而,现有研究往往忽略了 YOLO 神经网络中卷积所具有的此类特征,从而导致计算部件内部数据复用率显著降低,并进一步降低 YOLO 神经网络的运行效率。因此,结合此种类型的卷积运算的特点对数据流架构进行优化对提升 YOLO 神经网络的推理效率意义重大。

2.3 YOLO 加速相关研究

YOLO 算法因其实时性好、结构简单而被应用于各种边缘设备中。YOLO-ReT[4] 结合边缘 GPU 的计算资源,利用原始特征收集与重新分发(RFCR)模型对 YOLO 神经网络进行了优化,有效提高了 YOLO 神经网络的整体运行速度,然而能效仍有待优化。HyConv[5] 和 Sparsity-GPU[6] 均重点分析了 GPU 平台上卷积神经网络推理阶段所存在的特征图与权值稀疏性,并通过低比特量化有效提高了整体能效,但运行端到端推理时功率较大。

VC707-YOLO[7] 与 Arria-YOLO[8] 均为基于 FPGA 平台的针对 YOLO 神经网络的专用加速器,前者使用二进制权值的同时流水化实现 YOLO 神经网络的卷积层,并将所有卷积层分别映射到专用硬件块上,从而提高层间数据的复用率并减少外部内存的访存频率,后者则采取多级流水与部分层融合的方式加速算子运行。以 Zynq-Winograd[9] 为代表的加速器在 FPGA 平台上利用 Winograd 算法压缩了卷积计算量,从而有效提升 YOLO 算法整体运行速度、降低硬件功耗,但 Winograd 算法本身实现较为复杂,可移植性较差。以 REQ-YOLO[10] 为代表的 YOLO 加速器利用了 FPGA 平台比特级 LUT 单元的可重构性,采取了结合特定模型压缩方法优化硬件结构的方式综合提升算法的实现效率。例如,REQ-YOLO 首先采取了块循环基质压缩技术与异构权值量化技术压缩 YOLO 模型,再在硬件基础上实现特定功能部件以支持相应压缩方式,显著提高了算法运行效率。然而,模型压缩的加速方法会带来不同程度的精度损失,此外,为适配压缩模型,还需要额外的硬件资源开销,由此导致了灵活性较差且较难移植的问题。而以 MC-YOLO[11] 为代表的 YOLO 加速器则利用了多处理单元脉动阵列下的输入特征图像与权值组播优化,在多核系统上有效提高了数据复用,并进一步提高计算资源利用率,取得了性能与能效的良好平衡。

2.4 数据流架构 DFU

DFU[15] 是一种面向通用计算的粗粒度数据流架构,其处理单元之间采取数据流执行模型,而处理单元内部采取控制流执行模型。采取 DFU 作为基础数据流架构,下面将分别介绍 DFU 的整体结构、执行层次与传输层次。

DFU 的整体结构如图 4 所示。数据流处理器在外部主机的控制下工作,主要由二维处理单元(Processing Element,PE)阵列、微控制器、片上数据缓存(Data Buffer,DBUF)、片上配置缓存(Configuration Buffer,CBUF)和片上网络(Network on Chip,NoC)组成。其中处理单元阵列按照长宽均为 4 进行排列,共包含 16 个处理单元[29]。为提高计算部件利用率,每一个处理单元的计算部件包含四级流水,并采取了 SIMD 运行架构进行数据计算[30]。

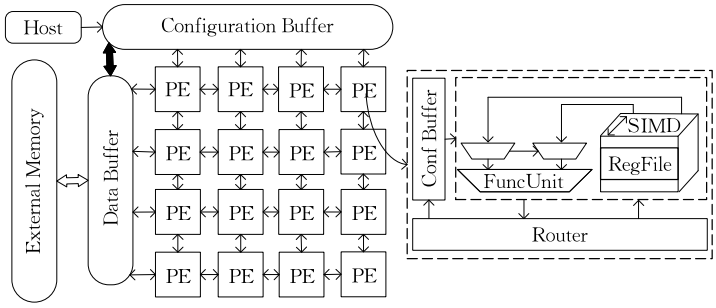


图 4 DFU 架构图

DFU 运行程序的执行层次如图 5 中所示。首先，在无数据相关性的维度，将计算任务划分为多个应用(APP)串序执行，以适应 DFU 片上内存的大小要求；然后，在每个 APP 内部，将应用切分为至多 4 个任务(Task)并行执行，以充分利用执行单元内的四级流水架构；在 Task 内部，将计算任务划分为若干子任务(Subtask)，每个子任务分别实现运算部分功能，实现方式为构建一套数据流图并将数据流图映射到计算单元阵列上执行。Subtask 之间串序执行，每个 Subtask 可多次循环执行，循环次数表示为 *Instance*。

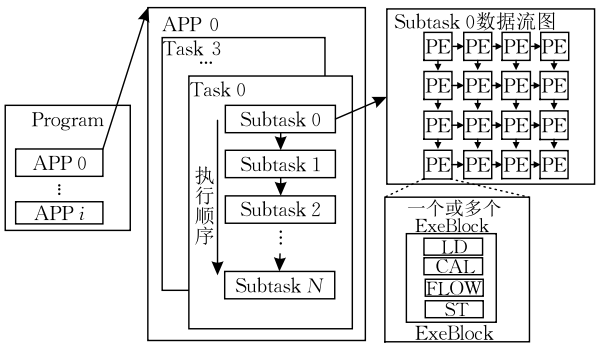


图 5　DFU 的执行模式示意图

DFU 的数据与指令的存储与传输主要在三个层次上进行、DFU 的片外主存与片上内存、片上内存与执行单元内部寄存器；执行单元之间。主机中的数据与指令存储在片外主存上，当主机的启动程序执行后，主控核控制直接内存存取部件(Direct Memory Access，DMA)将片外主存中的数据与指令通过总线 BUS 与 DFU 进行通信，这一层次上的数据传输对应 APP 层次上的数据处理。每次执行相同操作、不同数据的 APP 时，在首个 APP 完成配置文件与指令的传输后，其余 APP 只需从主存上存取对应数据，而不需要重复传输配置文件与指令。除上述片外主存与片上内存的数据传输由主机控制外，其余数据传输均由数据流架构上的数据流图的设计决定。

3　研究动机

3.1　数据复用性差

DFU 的执行模式与数据流图并不适合 YOLO 神经网络中运行时长占比最大的小卷积核卷积运算，并因此导致数据复用性较差。分析 DFU[15] 中的数据流图设计方案，卷积将通过一套数据流图在单个执行单元上依次完成输入数据读取、卷积循环乘加、输出数据存回，而数据复用一方面来源于卷积核高宽大于卷积步长时，输入图像数据中的重叠部分数据点通过片上网络在执行单元之间复用，另一方面来源于对卷积核数据的复用。因此，DFU 设计的卷积的数据流图 G_{DFG} 中，单个执行单元的平均数据复用量为

$$Reuse(G_{\mathrm{DFG}})=r_1\cdot(h-s_h)\cdot w+r_2\cdot(w-s_w)\cdot h+h\cdot w \tag{3}$$

上式中，r_1，r_2 代表计算任务规模与执行阵列拓扑结构带来的复用系数，大小在 0 至 1 间。其余变量的含义与表 2 中卷积公式的含义一致，式中前两项表示分别输入图像数据高、宽的复用，第三项表示卷积核本身的复用。然而，对 YOLO 神经网络中对单张输入图片、卷积核大小为 1 的卷积运算，由于卷积步长必然不小于卷积核高宽，因此传统数据流图下并不存在对输入图像数据的复用，式(3)中前两项均为 0；与此同时，由于卷积核宽高较小，因此卷积核的数据复用也较为有限。由此造成传统数据流架构的数据流图设计方案在执行 YOLO 中卷积核大小为 1 的运算时的数据复用率较低，大量计算资源被浪费。

3.2　算子间数据重复读取

DFU 的算子调度模式会导致大量重复数据读取。DFU 进行算子调度时，不会考虑算子间是否存在数据相关，执行不同算子时均需要在片外主存与片上内存间进行数据存取。在此调度模式下，卷积运算的结果需先存回至片外主存，执行激活算子时，再从片外主存上将相同的数据读取至片上内存中并进行运算。然而，根据对 YOLO 神经网络的分析，卷积运算与激活算子高度耦合，卷积运算的结果可以直接作为激活算子的输入，因此，当前算子间调度模式造成了大量额外的存取开销，并显著降低了计算部件利用率。

3.3　传输延迟高

DFU 的数据存取模式无法掩盖数据传输延迟。由于 DFU 上仅存在单个片上内存，因此当片上内存与片外主存进行数据存取时，执行单元阵列无法从片上内存中存取数据。此时，数据存取与执行将串序进行，数据存取过程中，执行单元阵列不执行操作。

神经网络往往具有很大的参数量，然而，受限于传输带宽，数据存取的延迟通常很大，因此片外主存与片上内存间的数据访问是神经网络硬件加速的瓶颈之一。以 YOLOv5s 部分卷积层为例统计 YOLO

神经网络卷积层在 DFU 上的运行时数据存取时长占比，如图 6 所示。图 6 中，数据存取时长占比平均高达 26.25%。因此，DFU 在执行 YOLOv5 神经网络时的串序存取-执行模式将增加程序执行时长，并造成执行单元阵列大量闲置，导致性能功耗比降低。

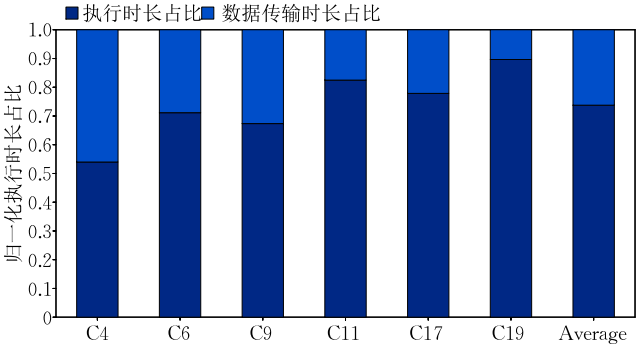


图 6 YOLO 神经网络部分卷积层的运行时长分布图

4 DFU-Y 架构

基于对 YOLO 神经网络与数据流架构 DFU 的分析，本文在 DFU 基本结构的基础上，设计了面向 YOLO 神经网络的数据流架构 DFU-Y(Data Flow Unitfor YOLO)，总体结构如图 7 所示。DFU-Y 的核心部件包括双缓存 SPM、执行单元阵列、微控制器等。其中，SPM 作为片上内存共有两个，两者彼此独立且功能完全相同，以完成数据存取功能并改善数据传输延迟问题；微控制器在原有基础上加入算子融合调度机制。DFU-Y 设计了三套片上网络：(1) 控制网络。用于传输指令、配置等控制信息；(2) 数据网络。用于 PE 间共享数据的传输；(3) 访存网络。用于 PE 阵列与片上存储间的数据传输。

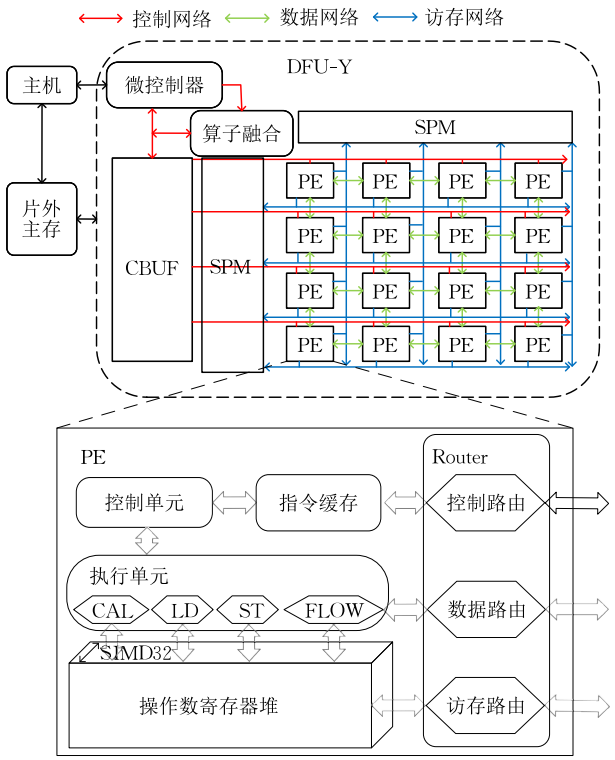


图 7 DFU-Y 总体架构

DFU-Y 的执行单元内部包括控制单元、执行单元、指令缓存、操作数寄存器堆和三套路由。控制信号由控制路由传入指令寄存器堆、再由控制单元解码；访存数据则由访存路由传入操作数寄存器堆、再由执行单元处理；执行单元将操作部件分类为计算、读取、存回、传输(FLOW)四种，其中传输部件可由数据路由完成 PE 间数据的快速传输。DFU-Y 在主机控制下执行操作，并与片外主存进行数据存取。

DFU-Y 的设计特点体现为：

(1) 在数据流图映射方面，为解决 YOLO 神经网络中卷积核大小为 1 的卷积运算数据复用差的问题，在 DFU-Y 中采用小卷积核卷积数据流图映射算法，以优化数据流图设计与映射，从而有效提高计算单元内部的数据复用，并提高计算部件利用率；

(2) 在调度方面，为解决 YOLO 神经网络中结构高度耦合的算子间数据重复读取问题，在微控制器中加入算子融合调度机制，将存在数据相关的算子通过数据流图的融合共同实现，避免数据的反复存取；

(3) 在访存部件设计方面，为解决传输延迟过高的问题，加入双缓存 SPM 作为片上内存，解耦合数据存取与执行阵列计算，并将两者并行执行，从而部分掩盖数据存取开销，综合提高算子执行效率。

后文将依次详细介绍 DFU-Y 的核心优化设计。

4.1 小卷积核卷积数据流图映射算法

DFU 设计的卷积数据流图不利于 YOLO 神经网络中卷积核大小为 1 的卷积运算的数据复用，针对此类卷积运算，重新设计数据流图映射算法。记数据流架构上的 SIMD 维度大小为变量 $simd$，执行单元阵列规模为 $PEvar$，执行单元阵列的拓扑结构为 TPE，数据流图的执行次数为 $instance$；由于对于卷积核大小为 1 的卷积运算，执行单元之间不存在数据传输，因此执行单元阵列的拓扑结构 TPE 不会产生影响。将卷积核大小为 1 的卷积在输入数据高宽、输入数据通道数、输出数据通道数的三个层次的嵌套循环分别记为 IHW、IC、OC。数据流图映射算法的输出为卷积计算维度到数据流架构参数的执行关系。

数据流图的设计重点在于提升指令级数据并行，并进一步提升数据复用[31]。当执行单元之间不存在空间维度上的数据复用时，可以从以下两个方面的提升数据复用：首先，考虑时间维度上的数据复用[32]，即将执行单元此次数据流图的节点输出作为同一执行单元下次数据流图的节点输入。分析卷积运算的三个循环维度发现，IC 维的累加计算一方面由于存在数据相关性而不利于挖掘指令级并行，另一方面则由于串序依次计算的特点而与上述时间维度上的数据复用目标相匹配。因此，将 IC 维度的计算通过数据流图的顺序执行次数 $Instance$ 实现将避免 IC 维度上结果的存取，从而在单个执行单元内提升时间维度上的数据复用。

其次，考虑提升单个执行单元内的数据复用。当卷积的 IC 维度通过数据流时间维度上的 $Instance$ 实现后，挖掘执行单元内数据复用的关键在于卷积中无数据相关性的 IHW、OC 维度在数据流 $PEvar$、$simd$ 上的映射执行方案，其中 $PEvar$ 将任务平均划分到不同执行单元上并行执行，而 $simd$ 则在执行单元内对多个操作数并行执行相同计算指令。分析执行小卷积核卷积运算时，单个执行单元内数据复用的来源。对于单个执行单元，读入一定大小的输入图像数据后，由于卷积核大小为 1，因此唯一可能的数据复用来源于卷积核不同 OC 维度数据的复用。分析将卷积 IC 维度映射到数据流图执行次数 $Instance$ 后，两种映射方案下，执行单元的内部平均数据复用量如式(4)。

$$Reuse(G_{\mathrm{DFG}})=\begin{cases}k\cdot\dfrac{OC}{PEvar}\cdot simd,\ OC\rightarrow PEvar, IHW\rightarrow simd\\ k\cdot\dfrac{OC}{simd}\cdot simd,\ IHW\rightarrow PEvar, OC\rightarrow simd\end{cases},$$
$$0<k\leq 1 \tag{4}$$

式中，k 表示由于执行单元阵列拓扑结构所带来复用参数，大小介于 0 至 1 之间。于是，数据流图构建算法如算法 1，其中各参数含义与表 2 保持一致。

算法 1.　小卷积核卷积数据流图映射算法。

输入：SIMD 维度大小 $simd$，执行单元阵列规模 $PEvar$，执行单元阵列拓扑 TPE，数据流图执行次数 $instance$，卷积核高/宽 h/w，输入数据宽高 IHW，输入通道数 IC，输出通道数 OC

输出：数据流图 $G_{\mathrm{DFG}}=(V,E)$、数据流图节点与执行单元阵列映射关系 $map(V,TPE)$

//首先判断卷积核大小是否为 1

IF $h==1\ \&\&\ w==1$ THEN
  $IC\rightarrow Instance$
  $Node(Instruction\text{-}Calculation)=MADD$
  //根据复用量选择不同的映射方式
  IF $k\cdot\dfrac{OC}{PEvar}\cdot simd\geq k\cdot\dfrac{OC}{simd}\cdot simd$ THEN
    $OC\rightarrow PEvar$, $IHW\rightarrow simd$
  ELSE THEN
    $IHW\rightarrow PEvar$, $OC\rightarrow simd$
  END IF
  FOR every p in $PEvar$
    $Map(Node,p)$
  END FOR
  //若卷积核大小不为 1，仍采取传统映射方式
ELSE
  Tradition G_{DFG}[15]
END IF

算法 1 首先判断卷积核是否大小为 1，若不为 1 则采取 DFU[15] 中的数据流图映射方案。在满足小卷积核的前提下，首先，将卷积 IC 维的运算映射到数据流图的循环次数 $Instance$ 维度上执行，且每一个数据流图的计算节点的基本指令为乘加运算(MADD)。然后，根据数据流架构 $simd$ 维度与执行单元阵列规模 $PEvar$ 的大小关系，确定卷积的 OC、IHW 维度与数据流架构的 $simd$、$PEvar$ 维度的执行关系，以达到最大化数据复用的目的。完成任务维度划分后，由于执行单元间不存在数据传输，因此将计算节点平均映射到执行单元阵列上。

以 DFU 数据流架构为例，其 $simd$ 大小为 32，而 $PEvar$ 大小为 16。当我们采取文献[15]中的数据流图映射方案，根据式(3)，此时复用数据量为 1，即卷积核本身。但是，根据算法 1 中的数据流图映射方案，卷积核的 OC 维度将映射到 $PE16$ 上、IHW 维度将映射到 $simd32$ 上，且 DFU 的执行单元阵列的拓扑结构为二维对称正方形，k 为 1。于是，此时的数据复用量将高达 $2\cdot OC$，为在 DFU 架构下所能取得的最佳数据复用量。优化后的映射方案如图 8 所示。

图 8 中，$Input\ Feature$ 指本次 Task 所需处理的输入数据，$Output\ Feature$ 则指输出数据。根据算法 1，卷积核的 OC 维数据将平均划分到不同 PE 中，且这个数据点在 $SIMD32$ 维度上重复；而 IHW 维度数据则读取到 $SIMD32$ 维度上。在单个执行单元内的一次 Subtask 中，首先将卷积核与输入高宽对应项相乘，并将结果保存；然后，在此执行单元内

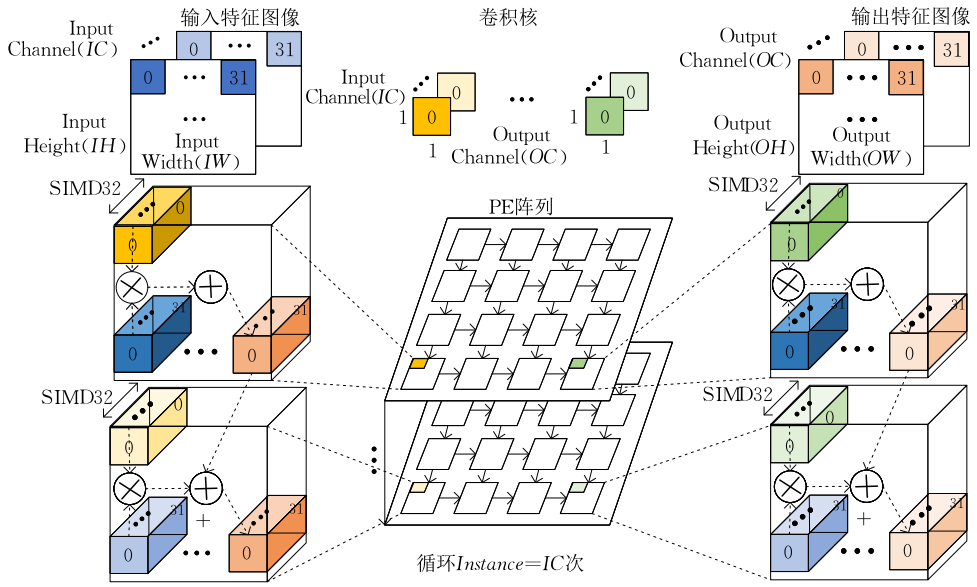


图 8 小卷积核卷积数据流图映射示意图

将依次执行卷积的 IC 维度计算，即每一次输入数据与卷积核的相乘的结果将与之前的结果进行累加。这个过程将通过对数据流图整体循环实现。

4.2 算子融合调度机制

DFU 在设计数据流图时不会考虑算子间的数据依赖关系，面对数据高度相关的算子时，常常造成大量数据重复读取，降低了程序运行效率。针对访存密集型多算子执行任务，算子融合是减少访存延迟、提升计算效率的常用方案[33]。算子融合是指将原本独立依次执行的多个算子所要实现的功能综合到同一个算子中完成。不同于传统控制流架构通过指令层次的算子融合[34]，数据流架构通过对数据流图中计算节点与有向边的融合，实现算子融合。两个算子的数据流图能够融合的前提是它们的数据流图节点数目一致、映射关系 $map(V,TPE)$ 相同而计算指令不同，且计算指令间存在明确的数据依赖关系。

数据流图中节点可分为读取节点、计算节点与存回节点三类，分别实现数据读入计算部件、完成计算指令与数据存回内存三种功能。针对不同类型节点，节点融合机制目标不同。对于读取与存回节点，节点融合需要消除重复指令，即读取或存回数据内存地址一致的指令，从而减小数据传输延迟。而对于计算节点，节点融合需要保证正确的数据相关性，即按照正确的先后顺序执行指令，这需要主机通过一个二进制数明确计算指令的数据依赖，设置为 1 表示第一个算子执行在前，设置为 0 表示第二个算子执行在前。

根据上述目标，本文在微控制器中加入算子融合调度机制，该机制如图 9 所示，运行步骤如下：

步骤 1. 节点分组。主机向微控制器发送算子融合命令，微控制器从 CBUF 中读取将要融合的算子的数据流图，并将节点按照有向边对应分组；

步骤 2. 节点融合。微控制器按顺序同时读取节点分组后的数据流图的一组对应节点，若节点均已读完，跳转至步骤 5；否则，判断是否为计算节点；若是，则跳转至步骤 3，否则跳转至步骤 4；

步骤 3. 计算节点融合。根据主机发送的计算指令数据依赖关系，将该组节点中的指令按照正确的先后执行关系进行合并，输出合并后的一组计算节点与有向边，跳转至步骤 2；

步骤 4. 读取、存回节点融合。保留该组节点中

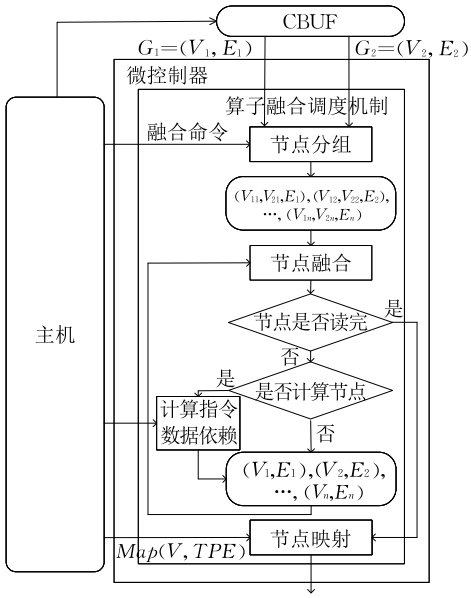


图 9 算子融合调度机制图

数据依赖在前的节点，输出该节点与其对应的有向边，跳转至步骤 2；

步骤 5. 节点映射。根据主机发送的原数据流图与执行单元阵列的映射关系 $map(V,TPE)$，返回新生成的数据流图并映射到执行单元阵列上，结束算子融合过程。

以 YOLO 神经网络中的小卷积核卷积运算与 SiLU 激活算子为例，由于卷积核大小为 1、输入数据间除输入通道维度外没有数据相关性，因此小卷积核卷积运算的数据流图结构与 SiLU 算子高度一致，两者最大的区别仅在于计算节点的指令不同，并且卷积运算的计算指令的输出为 SiLU 算子的计算指令的输入。因此，小卷积核卷积运算与 SiLU 算子非常适合采取算子融合调度机制。根据上述算子融合步骤，两者融合过程主要为将 SiLU 的计算节点的指令添加到小卷积核卷积运算计算节点的指令之后、结果回存之前，并保持其余节点不变。此外，两者的数据流图融合后，原有的映射关系同样保持不变。通过算子融合，将避免 SiLU 算子的重复数据存取与配置，从而提高计算部件利用率，增加神经网络的总体运行效率。

4.3　解耦合访存-执行模式

数据流架构上的卷积与 SiLU 算子属于访存密集型算子，主存与片上内存数据存取开销很大。然而，当前硬件结构下，由于仅存在一个片上内存，当与主存进行数据存取时，处理单元阵列将无法访问片上内存，从而被闲置，如图 10 所示。若能解耦访存与执行，从而并行执行数据存取与处理单元运算，将在很大程度上掩盖数据存取的开销、减少程序运行时长，进一步提升运行效率。

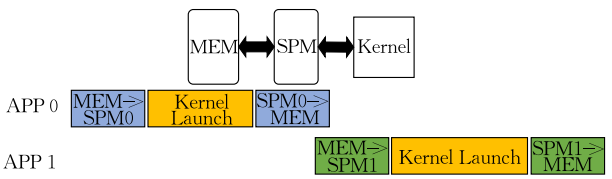


图 10　未解耦合执行模式图

双缓存乒乓机制常用于解耦访存与执行。具体过程为，通过两个片上存储器 SPM0、SPM1 交替与主存、执行单元阵列通信，将 Kernel 执行与数据存取并行，从而掩盖数据存取开销[35]。为适合用于数据流架构 DFU 的结构特点，采用 2 个大小一致、彼此独立的 SPM。当 SPM0 与 DFU 外的主存通信时，SPM1 可同时与执行单元阵列通信，并执行相应运算。当 SPM1 上的数据运算结束时，SPM0 中的数据再与 PE 进行通信并执行运算，同时将 SPM1 上的结果回存到主存 MEM。由此交替执行，直到执行完所有 APP。

为充分实现双缓存解耦合的优化效果，应合理划分单个 APP 的任务量大小。理想情况下，若数据在执行单元阵列上的执行时长能同时覆盖 SPM0 的数据回存时长与 SPM1 的数据读取时长，那么数据在主存与 SPM 间的传输延迟将完全被掩盖。此时，主存与 SPM 间的数据存取开销仅发生在第一个 APP 的数据读取以及最后一个 APP 的数据存回中。以 2 个 APP 为例，理想的双缓存并行运行过程如图 11 所示。

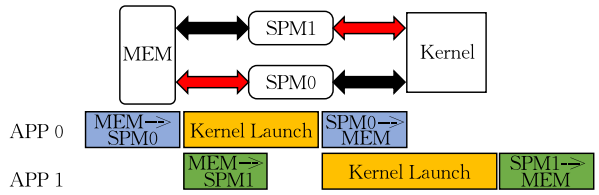


图 11　双缓存解耦合执行模式图

在图 11 中，共有两个计算任务，分为两组，编号为偶数的 APP 将从主存读入的数据存入基址为 SPM0 中，编号为奇数的 APP 将采用 SPM1 执行相同操作。在执行第一个 APP0 时，数据的读取时间将明显长于其余 APP 的数据读取，这是由于第一次在 PE 阵列上执行数据流图时，还需要传输数据流图的配置指令及映射关系等配置数据，而所有 APP 数据流图完全一致，因此后续 APP 不再需要传输数据流图的配置数据。对比图 10，图 11 中的执行单元阵列运算时长充分掩盖了数据存取延迟，不仅能提高计算部件利用率，还能在很大程度上减小程序运行时长。

5　实验设置

5.1　实验平台

本文基于模拟平台 SimICT[36]，使用 C 语言实现了时钟级精确的数据流加速器架构模拟器，以获取运行时长与利用率。SimICT 是一款中科院计算所自主研发的大规模并行模拟框架，通过对核心计算部件的 RTL 级硬件校准，可使模拟误差在 7% 以内[15]。同时，本文使用 Verilog 语言实现了对 DFU-Y 的 RTL 级仿真，在使用 Synopsys Design Compiler 工具进行综合时，采取 TSMC 的 12 纳米 GP standard VT library 工艺以获取面积与功耗。DFU-Y 模拟器的配置参数如表 3 所示。

表 3 DFU-Y 配置参数

模块	配置信息
微控制器	1×RISC-V 核，可执行算子融合调度机制
处理单元阵列	4×4，16 KB 指令缓存，144 KB 数据缓存，1 GHz，SIMD32，FP32（1TFLOPS）、INT8（4TOPS）
片上网络	2D Mesh，1 套 PE 通信网络，1 套控制网络，1 套访存网络
片上存储	SPM，Ping-Pong，3 MB
访存带宽	32.00 GB/s
总面积	57.04 mm^2
总功率	5 W

本实验中，微控制器用于控制执行单元阵列的执行过程，相较于 DFU 加入了算子融合调度机制。处理单元阵列构造与 DFU 基本保持一致，结合任务需求设计为 4×4 的规模，其中每个执行单元数据缓存大小为 144 KB、指令缓存大小为 16 KB，采取 SIMD32 执行模式。执行单元的主频设置为 1 GHz，彼此间通过 2 维 mesh 网络进行信息传输。片上内存共有 2 个，彼此完全独立，大小均设置为 3 MB，以存储 PE 阵列所能存取的最大数据量，并留出 0.5 MB 用于存储控制与配置信息。此时，SPM 大小不会对不同规模的卷积运算产生直接影响。本文通过在 SimICT 模拟器中接入 DRAMsim2 模拟器，实现访存延迟的评估。

使用基于 LLVM 平台设计的编译器为数据流架构上的算子编译数据流图与映射关系，并利用微控制器映射到处理单元阵列上以执行对应操作。

5.2 测试程序

实验选取 YOLOv2[37]、YOLOv2-Tiny、YOLOv3-Tiny 和 YOLOv5s[27] 作为测试程序。特别地，将 YOLOv2 和 YOLOv5s 中卷积核大小为 1 的卷积层列于表 4。除卷积核大小外，其余卷积层的卷积运算规模基本与表 4 一致，不再单独列出。实验中，BN 层与卷积层融合执行。BN 层计算公式为

$$O=\frac{I-\mu}{\sqrt{\sigma^2+\varepsilon}}*\gamma+\beta \tag{5}$$

式(5)中，O/I 分别为 BN 层输入/输出，μ、σ 分别为均值和方差，γ、β 分别为 BN 层的训练参数，ε 则是一个为避免分母为 0 而引入的极小正数。将式(5)代入式(1)，可得出融合后的卷积计算式(6)：

$$O[y][x][a][b]=\sum_{z=0}^{C-1}\sum_{i=0}^{h-1}\sum_{j=0}^{w-1}I[x][z][a\times s_h+i][b\times s_w+j]\times\frac{\gamma}{\sqrt{\sigma^2+\varepsilon}}\times W[y][z][i][j]+\left(\frac{B-\mu}{\sqrt{\sigma^2+\varepsilon}}\cdot\gamma+\beta\right) \tag{6}$$

对比式(1)与式(6)发现，BN 层与卷积层的融合仅对卷积核与偏置参数做推理前的静态线性变换预处理，不会对卷积运算的推理性能产生影响[11]，因此本文测试程序中不再考虑 BN 层。

表 4 YOLO 中小卷积核的卷积层

YOLO 版本	卷积层	输入图像规模（IH, IW, IC）	卷积核通道数与组数（IC, OC）
YOLOv5s	C3	160,160,64	64,32
	C4	160,160,32	32,32
	C6	160,160,64	64,64
	C8	80,80,128	128,64
	C9	80,80,64	64,64
	C11	80,80,128	128,128
	C14	80,80,256	256,64
	C16	40,40,256	256,128
	C17	40,40,128	128,128
	C19	40,40,256	40,40,256
	C22	40,40,512	512,128
	C25	20,20,256	256,256
YOLOv2	C4	104,104,128	128,64
	C7	52,52,256	256,128
	C10	26,26,512	512,256
	C15	13,13,1024	1024,512
	C17	13,13,1024	1024,512
	C21	26,26,512	512,64

DFU-Y 能够支持的数据格式包括 FP32 和 INT8。不同数据格式下，YOLO 模型的推理精度主要取决于量化与训练方法。在合理的量化方法下[38]，CNN 精度损失可控制在应用可接受的范围内。本文将在 DFU、DFU-Y 和 Xavier NX 平台上端到端执行 FP32 和 INT8 数据格式的 YOLOv5s 神经网络，并分析数据格式对推理性能的影响，而不再考虑数据格式对精度的影响。

此外，本文选取 AlexNet 与 VGG16 中部分具有代表性的卷积层作为 DFU-Y 通用性的测试程序，具体参数列于表 5。

表 5 AlexNet 与 VGG16 中部分代表性卷积层

神经网络	卷积层	输入图像规模（IH, IW, IC）	卷积核参数（$Size, IC, OC$）
AlexNet	C1	227,227,3	11,3,96
	C2	27,27,96	5,96,256
	C3	13,13,256	3,256,384
VGG16	C1	224,224,3	3,3,64
	C3	112,112,64	3,64,128
	C5	56,56,128	3,128,256
	C7	28,28,256	3,256,512

5.3 对比平台

首先，通过在 DFU-Y 上采取消融实验，对比不同优化方案相较于 DFU 所取得的性能提升。消融实验的步骤为，初始时，消除 DFU-Y 上的所有优化，使得 DFU-Y 与 DFU 完全一致，将实验结果作

为 Baseline；然后，在 DFU-Y 上复原小卷积核卷积数据流图映射算法，在相同条件下进行实验；以此类推，分别补充算子融合调度机制与双缓存解耦合，直至复原 DFU-Y，以分析每种优化方案带来的性能提升。

然后，选取英伟达 Jetson Xavier NX 作为边缘 GPU 对比平台。此平台有 384 个 CUDA 核，2 个深度学习加速器（Deep Learning Accelerator，DLA），INT8 精度下峰值算力高达 21 TOPS。GPU 上的算子通过 Pytorch 中的 Torch。nn 算子库实现，在 CUDA 核上执行运算。

此外，选取多种针对不同版本 YOLO 神经网络的 FPGA 加速器作为对比平台。不同实验平台的具体参数、数据格式和 YOLO 模型版本列于表 6 中。

表 6　实验对比平台参数

实验平台	硬件型号	主频/GHz	功率/W	片上内存	YOLO 模型	数据格式	性能
DFU-Y	数据流架构	1	5	2×3 MB SPM	YOLOv2，v2-Tiny，v3-Tiny，v5s	FP32/INT8	1 TFLOPS（FP32）/ 4 TOPS（INT8）
Jetson Xavier NX	Jetson Xavier NX 8 GB	0.845～1.1	15	11.125 MB 三级 Cache	YOLOv2，v2-Tiny，v3-Tiny，v5s	FP32/INT8	0.825 TFLOPS（FP32）/ 21 TOPS（INT8）
VC707-YOLO[7]	Virtex-7 VC707 FPGA	0.2	8.7	20.1 MB BRAM	YOLOv2-Tiny	Binary	464.7 GOPS
Arria-YOLO[8]	Arria-10 GX1150	0.19	26	28.9 MB M20K RAM	YOLOv2	INT8	566 GOPS
REQ-YOLO[10]	ADM-7V3 FPGA	0.2	23	6.6 MB BRAM	YOLOv3-Tiny	INT8	/
MC-YOLO[11]	Xilinx XCZU7EV	0.167	3.75	308 MB BRAM	YOLOv3-Tiny	INT8	168 GOPS

5.4　衡量指标

本文中，当分析卷积层执行效果时，采取执行时长作为加速性能衡量指标，采取性能功耗比反映硬件设计的能效特性。性能功耗比（Performance-per-Watt）的定义为单位功率的计算性能，单位为 $GOP/s \cdot W^{-1}$。

当分析神经网络端到端执行效果时，采取每秒帧数（Frames Per Second，FPS）衡量速度，采取单位功率下每秒帧数（FPS-per-Watt，FPS/W）反映硬件总体的能效特性。

特别地，本文中计算部件利用率定义为实际计算性能占理论峰值性能的百分比，计算公式为

$$\text{计算部件利用率} = \frac{\text{实际计算性能}}{\text{理论峰值性能}} \times 100\% \quad (7)$$

6　结果分析

为综合对比优化效果、提高分析效率，本文选取了表 4 中部分具有代表性的卷积核大小为 1 的卷积层作为分析对象，综合分析 DFU-Y 的优化性能。

6.1　数据格式实验

在 DFU、DFU-Y 和 Jetson Xavier NX 平台上，端到端执行 FP32 和 INT8 数据格式下的 YOLOv5s 神经网络，结果如图 12 所示。

实验结果表明，在执行 YOLOv5s 神经网络时，相较于 FP32，INT8 数据格式下 DFU、DFU-Y 分别

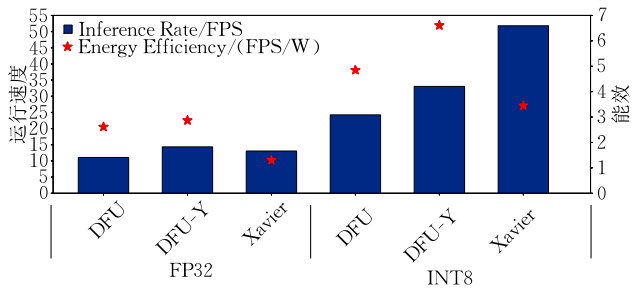


图 12　YOLOv5s 上不同数据格式实验结果图

取得 2.19、2.30 倍的速度提升和 1.86、2.31 倍的能效提升，主要因为 INT8 数据格式下峰值算力提升 4 倍，且数据存取次数减少、计算部件利用率更高。Xavier 在 FP32 数据格式下运行速度为 DFU-Y 的 0.91 倍，而在 INT8 数据格式下，运行速度高达 51.8 FPS，为 DFU-Y 的 1.57 倍。这主要由于 Xavier 有两个深度学习加速器，因此在 INT8 数据格式下运算速度更快。

总的来说，所有实验平台都能在 INT8 数据格式下取得更好的运行性能，DFU-Y 更是能取得 6.61 FPS/W 的最佳能效。在本文后续实验中，除特殊 FPGA 平台会特别注明外，所有实验均采取 INT8 数据格式。

6.2　DFU-Y 上的消融实验

图 13 展示了相较于 DFU，DFU-Y 上不同优化带来的运行时长减少与计算部件利用率收益。图 13 中，以各层 DFU 结果为基准，将运行时间归一化，“Prov1”指采取小卷积核卷积数据流图映射算法，

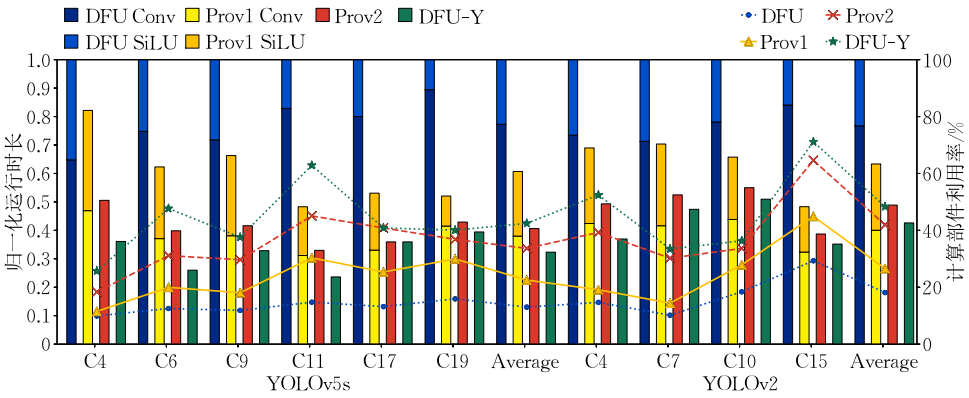


图 13 DFU-Y 上的消融实验结果图

“Prov2”指在“Prov1”的基础上采取算子融合调度机制,“DFU-Y”指在“Prov2”的基础上采取解耦合访存-执行模式,也即最终的 DFU-Y 整体优化结构;左侧纵轴表示以每层各自的 DFU 结果为基准进行归一化后的运行时长,表现为图 13 中的柱状图;右侧纵轴表示不同方案的计算部件利用率,表现为折。

在DFU上,YOLOv5s与YOLOv2中运行时长较大,而计算部件平均利用率分别为 13.02%、18.14%,表明 DFU 执行 YOLO 神经网络时的运行时长与计算部件利用率均存在较大优化空间。

采取小卷积核卷积数据流图映射算法的优化效果如图 13 中的“Prov1”所示。在执行时长方面,激活算子的运行时长未发生变化,而 YOLOv5s 与 YOLOv2 的算子平均总运行时长分别下降至 0.6069、0.6333,平均下降 37.99%。总体而言,*OC* 维更大的卷积层优化效果更佳。例如,YOLOv5s 中 C11 下降幅度最大、C4 下降幅度最小,YOLOv2 中 C15 下降幅度最大、C7 下降幅度最小。主要原因在于相同条件下,*OC* 维数据越多,数据复用的提高越明显。体现在计算部件利用率上,当采用此优化后,YOLOv5s 与 YOLOv2 分别提高到 22.52%、26.57%,是 DFU 的近两倍。

同时采取算子融合调度机制后的优化效果如图 13 中的“Prov2”所示。融合后将不再区分卷积运算与激活算子的执行时长;此时,YOLOv5s 与 YOLOv2 的算子平均总运行时长分别下降至 0.4063、0.4887,通过避免激活算子输入数据的反复存取,程序运行总时长平均下降 17.26%。YOLOv5s、YOLOv2 的平均计算部件利用率也分别提高到 33.64%、41.93%。

进一步采取双缓存解耦合存取-执行后的优化效果如图 13 中的“DFU-Y”所示。在执行时长上,YOLOv5s 与 YOLOv2 的算子平均总运行时长分别为 0.3232、0.4261,平均下降 7.285%。其中不同卷积层的减小幅度差异较大,卷积层数据量越大、单 APP 执行时长越长、APP 数越多,解耦合后所能掩盖的数据存取时长越大,由此带来的总运行时长减小也就越明显。特别地,以 YOLOv5s 中的 C17 层为代表的部分卷积层并未获得提升,这是由于 C17 数据量小,采取了单 APP 执行,因此无法掩盖传输延迟。

综合而言,相较于 DFU,DFU-Y 在 YOLOv5s、YOLOv2 上分别获得 2.707 倍、2.347 倍的性能提升,平均提升 2.527 倍。平均计算部件利用率分别从 13.02%、18.14%提高到 42.45%、48.32%,运行效率取得较大增益。

6.3 与其他平台的性能对比

图 14 展示了相较于 Xavier 平台,DFU 与 DFU-Y 在运行时长与计算部件利用率上的优化效果,图 14 中,Xavier 以 GPU 代称。以 GPU 运行时长为基准,对 DFU、DFU-Y 上的运行时长做归一化,左侧纵轴表示归一化运行时长,右侧纵轴表示计算部件利用率。

根据图 14,在 DFU 上运行 YOLOv5s、YOLOv2 的平均运行时长分别高达 2.430、1.673,平均计算部件利用率分别仅有 13.02%、18.14%,低于 GPU 上的 33.33%、31.61%。这主要由于 DFU 没有结合 YOLO 本身的特点进行结构优化。对比之下,DFU-Y 运行 YOLOV5s、YOLOv2 的平均运行时长分别下降至 0.7449、0.7549,性能分别提升 1.342 倍、1.325 倍,平均提升 1.334 倍;其中,YOLOv5s 的 C4、C6,YOLOv2 的 C4、C7 由于输入图像数据量大、*OC* 维多,取得的速度提升最为显著,而 YOLOv5s 中的 C19 由于输入图像宽高较小、数据存取次数较小,导

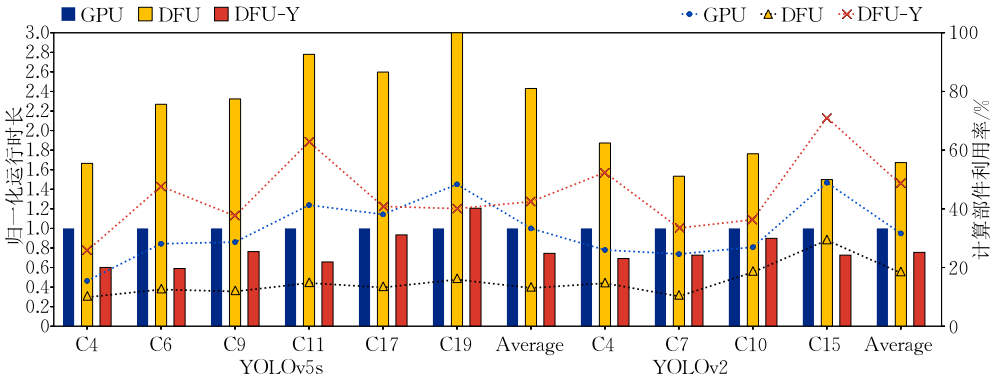


图 14　与 Xavier 运行时长结果对比图

致 DFU-Y 的表现稍逊于 GPU。在计算部件利用率上，DFU-Y 运行 YOLOv5s、YOLOv2 时分别提高到 42.45%、48.32%，显著高于 GPU。

图 15 展示了 DFU-Y 与 GPU 上的性能功耗比，图 15 中，以 GPU 的结果为基准对 DFU-Y 的结果进行了归一化。根据实验结果，GPU 上性能功耗比普遍较低，这主要由于 GPU 的计算部件利用率较低。相较而言，DFU-Y 上运行 YOLOv5s 与 YOLOv2 的平均性能功耗比分别高达 3.532、3.396，平均提升 3.464 倍。卷积层性能功耗比的提升效果与运行时长及计算部件利用率高度相关，即在 DFU-Y 上运行时长减少越多、计算部件利用率越高，性能功耗比也相应更高。

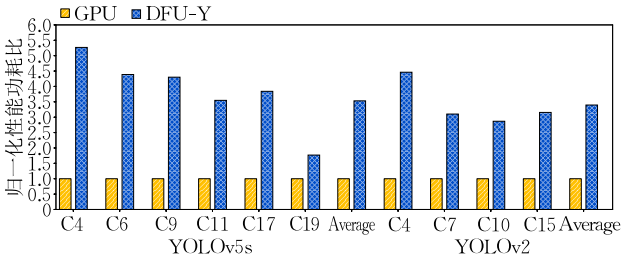


图 15　DFU-Y 与 Xavier 性能功耗比对比图

图 16 展示了与 YOLOv2 专用加速器 Arria-YOLO 相比，DFU 与 DFU-Y 的运行时长。图 16 中，以 Arria-YOLO 运行时长结果为基准进行归一化。DFU 相对 Arria-YOLO 的运行时长平均高达 3.037 倍，加速效果与专用加速器相差较大；而 DFU-Y 相较于 Arria-YOLO 运行时长平均为 1.371 倍，相较于 DFU 取得了 2.21 倍的性能提升。

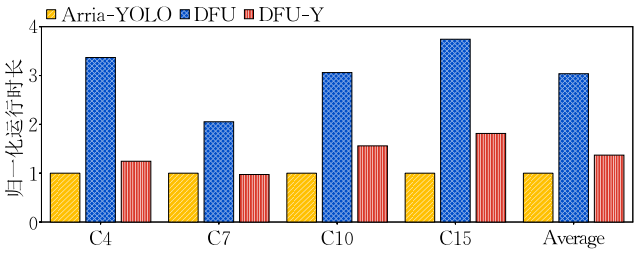


图 16　与 Arria-YOLO 运行时长对比图

图 17 展示了在不同实验平台上，端到端运行不同 YOLO 模型下的实验结果。图 17 中，柱状图表示以每个 YOLO 模型下 DFU 执行效果为基准进行归一化后的加速比，散点图表示以每个 YOLO 模型下 DFU 执行效果为基准进行归一化后的能效。YOLO 版本在横坐标中标出。

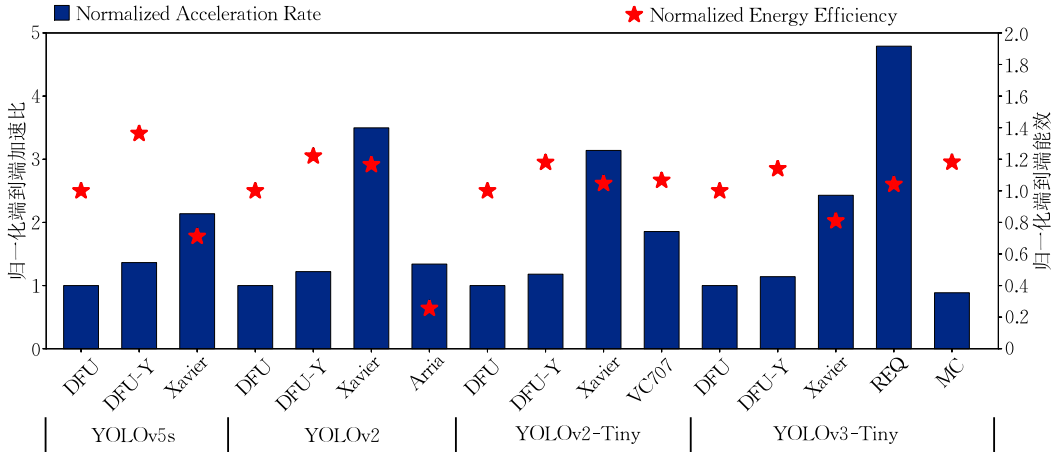


图 17　不同平台执行不同 YOLO 神经网络的端到端结果对比图

实验结果表明，在YOLOv5s、YOLOv2和YOLOv2-Tiny上，Xavier均取得了最高的加速比，这主要由于Xavier的峰值性能明显高于其他平台；而在YOLOv3-Tiny上，得益于块循环基质压缩技术与异构权值量化技术在硬件上定制化设计，REQ-YOLO取得了最高的加速比。相较而言，DFU-Y在YOLOv5s、YOLOv2、YOLOv2-Tiny和YOLOv3-Tiny上分别取得了1.36、1.22、1.18和1.14倍的加速比，平均加速比1.23。可以发现，由于YOLOv5s中卷积核大小为1的卷积运算比例最高，因此DFU-Y的加速效果最好；而YOLOv3-Tiny中卷积核大小为1的卷积运算比例较小，因此总的加速效果不如YOLOv5s。

DFU-Y在所有YOLO模型上的平均能效高达7.11 FPS/W。在YOLOv5s、YOLOv2和YOLOv2-Tiny上DFU-Y均取得了最佳的能效，主要因为DFU-Y在较高的峰值性能、较低的功率下，通过小卷积核数据流图映射算法取得了较好的数据复用和计算部件利用率，因此能效表现较为突出。特别地，在YOLOv2实验中，尽管Arria-YOLO在卷积核大小为1的卷积层中能效优异，然而，由于其端到端推理总功率高达26W，因此总体能效表现较差。而在YOLOv3-Tiny上，MC-YOLO由于采取了多核系统上的脉动阵列权值组播优化，能效最佳；相较而言此时DFU-Y的能效表现不如在YOLOv5s上，但仍取得了MC-YOLO能效的96.51%，与之接近。

6.4 DFU-Y执行其他网络

数据流架构DFU-Y在加速YOLO神经网络的同时，保持了良好的通用性。图18展示了相对于数据流架构DFU，DFU-Y在AlexNet、VGG16的部分代表性卷积层上的实验结果。图18中，DFU-Y在AlexNet与VGG16上的运行时长相对Xavier平均分别下降18.86%、27.61%，取得了与Xavier接近的执行效果。此时优化效果不如YOLO是因为AlexNet与VGG16的卷积核大小均不为1，因此优化中的小卷积核卷积数据流图映射算法无法取得增益。实验表明，DFU-Y能够在提升YOLO神经网络性能与能效的同时保持较好通用性。

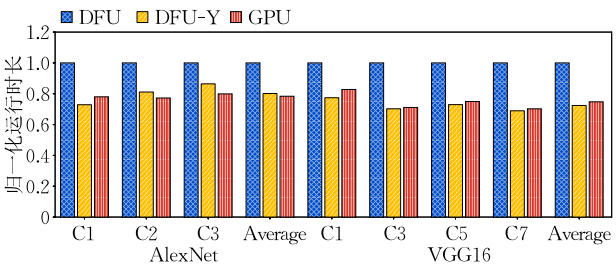
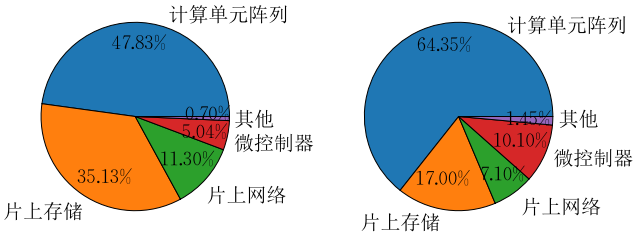


图18 DFU-Y的通用性实验结果图

6.5 代 价

本文中由于小卷积核卷积数据流图映射算法优化带来的编译开销很小。数据流图主要在编译器的编译阶段生成，而卷积运算的数据流图规模较小、编译逻辑较为简单，因此相较于数据流架构上的运算阶段，数据流图映射算法优化的额外编译代价可以忽略。

DFU-Y的硬件开销如图19所示。相较于DFU，DFU-Y的面积增加2.41%，额定功耗增加6.01%。硬件增加的开销主要来源于微控制器执行算子融合调度机制的执行功耗以及双缓存解耦带来的额外存取启动开销。相较于优化带来的增益，额外的硬件开销可以接受。



(a) 面积占比 (b) 功率占比

图19 DFU-Y的面积与功耗分布

7 总 结

本文结合YOLO神经网络的特点，提出了数据流架构上的小卷积核卷积数据流图映射算法与算子融合调度机制，并加入双缓存机制解耦合数据存取与执行，从而提出YOLO数据流加速器DFU-Y。在小卷积核卷积运算上相比INT8精度下的数据流架构DFU与GPU(NVIDIA Xavier NX)，DFU-Y分别获得2.527倍、1.334倍的性能提升和2.658倍、3.464倍的能效提升；相较于YOLO专用加速器(Arria-YOLO)，DFU-Y的能够在保持较好通用性的同时，取得了72.97%的性能与87.41%的能效增益。而在端到端实验中，DFU-Y表现出了良好的能效特性。本文目前仅针对YOLO算法神经网络中的卷积层进行了优化设计，未来工作将继续面向神经网络中的其他计算如MaxPooling、UPSampling在数据流架构上优化，并进一步提升DFU-Y在不同卷积神经网络上的通用性。

致 谢 感谢评审专家的中肯意见，使得本文的质量得以提升！

参 考 文 献

[1] Kang Kai, Ouyang Wanli, Li Hongsheng, et al. Object detection from video tubelets with convolutional neural networks//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, USA, 2016: 817-825

[2] Wu Wenkai, Chen Chienyu, Lee Jiannshu. Embedded YOLO: Faster and lighter object detection//Proceedings of the 2021 International Conference on Multimedia Retrieval. Taipei, China, 2021: 560-565

[3] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, USA, 2016: 779-788

[4] Ganesh P, Chen Yao, Yang Yin, et al. YOLO-ReT: Towards high accuracy real-time object detection on edge GPUs//Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. Waikoloa, USA, 2021: 1311-1321

[5] Li Xiaqing, Zhang Guangyan, Wang Zhufan, et al. HyConv: Accelerating multi-phase CNN computation by fine-grained policy selection. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(2): 388-399

[6] Oh C, So J, Kim S, et al. Exploiting activation sparsity for fast CNN inferenceon mobile GPUs. ACM Transactions on Embedded Computing Systems, 2021, 20(5): 1-25

[7] Nguyen D, Nguyen T, Kim H, et al. A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection. IEEE Transactions on Very Large Scale Integration Systems, 2019, 27(8): 1861-1873

[8] Xu Ke, Wang Xiaoyu, Liu Xinyang, et al. A dedicated hardware accelerator for real-time acceleration of YOLOv2. Real-Time Image Process, 2021, 18(3): 481-492

[9] Bao Chun, Xie Tao, Feng Wenbin, et al. A power-efficient optimizing framework FPGA accelerator based on Winograd for YOLO. IEEE Access, 2020, 8: 94307-94317

[10] Ding Caiwen, Wang Shu, Liu Ning, et al. REQ-YOLO: A resource-aware, efficient quantization framework for object detection on FPGAs//Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Seaside, USA, 2019: 33-42

[11] Adiono T, Ramadhan R, Sutisna N, et al. Fast and scalable multicore YOLOv3-Tiny accelerator using input stationary systolic architecture. IEEE Transactions on Very Large Scale Integration Systems, 2023, 31(11): 1774-1787

[12] Fuchs A, Wentzlaff D. The accelerator wall: Limits of chip specialization//Proceedings of the IEEE International Symposium on High Performance Computer Architecture. Washington, USA, 2019: 1-14

[13] Zhang Jiyu, Zhang Zhiru, Zhou Sheng, et al. Bit-level optimization for high-level synthesis and FPGA-based acceleration//Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays. Monterey, USA, 2010: 59-68

[14] Xiang Taoran, Ye Xiaochun, Li Wenming, et al. Accelerating fully connected layers of sparse neural networks with fine-grained dataflow architectures. Journal of Computer Research and Development, 2019, 56(6): 1192-1204(in Chinese)
(向陶然，叶笑春，李文明等. 基于细粒度数据流架构的稀疏神经网络全连接层加速. 计算机研究与发展，2019，56(6)：1192-1204)

[15] Fan Zhihua, Li Wenming, Tang Shengzhong, et al. Improving utilization of dataflow architectures through software and hardware co-design//Proceedings of the European Conference on Parallel Processing and Distributed Processing. Limassol, Cyprus, 2023: 245-259

[16] Wu Xin-Xin, Ou Yan, Li Wen-Ming, et al. Acceleration of sparse convolutional neural network based on coarse-grained dataflow architecture. Journal of Computer Research and Development, 2021, 58(7): 1504-1517(in Chinese)
(吴欣欣，欧焱，李文明等. 基于粗粒度数据流架构的稀疏卷积神经网络加速. 计算机研究与发展，2021，58(7)：1504-1517)

[17] Wu Xinxin, Fan Zhihua, Liu Tianyu, et al. LRP: Predictive output activation based on SVD approach for CNNs acceleration//Proceedings of the 2022 Conference and Exhibition on Design, Automation and Test in Europe. Leuven, Belgium, 2022: 831-836

[18] Fan Zhi-Hua, Wu Xin-Xin, Li Wen-Ming, et al. Dataflow architecture optimization for low-precision neural networks. Journal of Computer Research and Development, 2023, 60(1): 43-58(in Chinese)
(范志华，吴欣欣，李文明等. 面向低精度神经网络的数据流体系结构优化. 计算机研究与发展，2023，60(1)：43-58)

[19] Fan Zhi-Hua, Li Wen-Ming, Ye Xiao-Chun, et al. The research progress of dataflow computing: A brief survey. Frontiers of Data and Computing, 2021, 3(5): 65-81(in Chinese)
(范志华，李文明，叶笑春等. 数据流计算研究进展与概述. 数据与计算发展前沿，2021，3(5)：65-81)

[20] Fan Zhihua, Li Wenming Wang Zhen, et al. Accelerating convolutional neural networks by exploiting the sparsity of output activation. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(12): 3253-3265

[21] Yan Mingyu, Deng Lei, Hu Xing, et al. HyGCN: A GCN accelerator with hybrid architecture//Proceedings of the IEEE International Symposium on High Performance Computer Architecture. San Diego, USA, 2020: 15-29

[22] Yan Mingyu, Hu Xing, Li Shuangchen, et al. Alleviating

irregularity in graph analytics acceleration: A hardware/software co-design approach//Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. Columbus, USA, 2019: 615-628

[23] Nowatzki T, Gangadhar V, Ardalani N, et al. Stream-dataflow acceleration//Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture. Toronto, Canada, 2017: 416-429

[24] Nowatzki T, Gangadhar V, Sankaralingam K. Exploring the potential of heterogeneous Von Neumann/dataflow execution models//Proceedings of the ACM/IEEE 42nd Annual International Symposium on Computer Architecture. Portland, USA, 2015: 298-310

[25] LeCun Y, Boser B, Denker J, et al. Handwritten digit recognition with a back-propagation network//Proceedings of the 2nd International Conference on Neural Information Processing Systems. Denver, USA, 1989: 396-404

[26] Krizhevsky A, Sutskever I, Hinton G. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 2017, 60(6): 84-90

[27] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556, 2015: 1-14

[28] Jocher G. YOLOv5 by Ultralytics. 2020. https://github.com/ultralytics/yolov5. DOI:10.5281/zenodo.3908559

[29] Shen Xiaowei, Ye Xiaochun, Tan Xu, et al. An efficient network-on-chip router for dataflow architecture. Journal of Computer Science and Technology, 2017, 32: 11-25

[30] Shen Xiaowei, Ye Xiaochun, Tan Xu, et al. Memory partition for SIMD in streaming dataflow architectures//Proceedings of the 7th International Green and Sustainable Computing Conference. Hanzhou, China, 2016: 1-8

[31] Kong Xiangyu, Huang Yi, Zhu Jianfeng, et al. MapZero: Mapping for coarse-grained reconfigurable architectures with reinforcement learning and Monte-Carlo tree search//Proceedings of the 50th Annual International Symposium on Computer Architecture. Orlando, USA, 2023: 1-14

[32] Lee J, Carlson T. Ultra-fast CGRA scheduling to enable run time, programmable CGRAs//Proceedings of the 58th ACM/IEEE Design Automation Conference. San Francisco, USA, 2021: 1207-1212

[33] Zheng Size, Chen Siyuan, Song Peidi, et al. Chimera: An analytical optimizing framework for effective compute-intensive operators fusion//Proceedings of the IEEE International Symposium on High-Performance Computer Architecture. Montreal, Canada, 2023: 1113-1126

[34] Singh S, Perais A, Jimborean A, et al. Exploring instruction fusion opportunities in general purpose processors//Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture. Chicago, USA, 2022: 199-212

[35] Tan Xu, Shen Xiaowei, Ye Xiaochun, et al. A non-stop double buffering mechanism for dataflow architecture. Journal of Computer Science and Technology, 2018, 33: 145-157

[36] Ye Xiaochun, Fan Dongrui, Sun Ninghui, et al. SimICT: A fast and flexible framework for performance and power evaluation of large-scale architecture//Proceedings of the International Symposium on Low Power Electronics and Design. Beijing, China, 2013: 273-278

[37] Redmon J, Farhadi A. YOLO9000: Better, faster, stronger //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Honolulu, USA, 2017: 6517-6525

[38] Liu Zechun, Cheng Kwang-Ting, Huang Dong, et al. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. New Orleans, USA, 2022: 4932-4942

MU Yu-Dong, M. S. candidate. His main research interests include dataflow architecture, dataflow graph mapping and reconfigurable architecture.

LI Wen-Ming, Ph. D., associate professor. His main research interests include high-throughput processor architecture and dataflow architecture.

FAN Zhi-Hua, Ph. D., assistant professor. His main research interests include dataflow architecture and reconfigurable intelligent architecture.

WU Meng, Ph. D. candidate. Her main research interests include dataflow architecture and high-throughput computer architecture.

WU Hai-Bin, Ph. D. candidate. His main research interests include dataflow architecture and high-throughput computer architecture.

AN Xue-Jun, Ph. D., professor. His main research interests include Graph Processing Architecture and High Performance Interconnection Networks.

YE Xiao-Chun, Ph. D., professor. His main research interests include high-performance computer architecture and software simulation.

FAN Dong-Rui, Ph. D., professor. His main research interests include high-throughput computer architecture and high- performance computer architecture.

Background

The topic of this research focuses on enhancing the execution efficiency of the YOLO neural network, particularly emphasizing the optimization on dataflow architectures. YOLO neural networks are applied in various scenarios that require real-time performance such as Autonomous driving and vehicle detection, hence there is extensive research on YOLO's execution efficiency. However, previous studies have mainly concentrated on control flow architectures such as GPUs and FPGA-based domain-specific accelerators, which have apparent bottlenecks in energy efficiency and design cost, respectively.

The execution mechanism of dataflow architecture allows an operation to be executed once its operands are ready, which are inherently aligned with the acceleration of neural networks. This research group has accelerated sparse neural networks, scientific computing, and graph neural networks on dataflow architectures. Nevertheless, after analyzing the execution of YOLO neural network on the dataflow architecture, this research discovers that accelerating convolutional neural networks on dataflow architectures has not fully leveraged its advantages due to poor mapping effects of dataflow graphs of convolution operations with small kernels, redundant data transfers between operators and significant off-chip data access overhead.

This paper optimizes the execution on dataflow architectures targeting these issues. Firstly, it analyzes the parallelization pattern of the dataflow architecture and proposes dataflow graph mapping algorithm for convolution operations with small kernels to fully utilize the data reuse of nested loops in convolution. Secondly, to fully exploit the data reuse between structurally coupled operators, this paper introduces an operator fusion scheduling mechanism at the dataflow graph level to reduce data access frequency and enhance neural network execution efficiency. Lastly, by decoupling data access and execution through double buffering, this paper enables parallel execution of off-chip data access and on-chip data computation, thereby amortizing the data transmission delay and increasing the utilization rate of the function unit.

Experiments show that compared to representative dataflow architectures and GPUs, this topic achieves a performance improvement of 2.527× and 1.334×, respectively, and an energy efficiency improvement of 2.658× and 3.464×. In end-to-end experiments, this topic achieves an average energy efficiency of 7.11 FPS/W on different YOLO networks.

This paper was sponsored by the Beijing Nova Program (Nos. 20220484054, 20230484420), the Beijing Natural Science Foundation and Changping Innovation Joint Fund (No. L234078), the CAS Project for Youth Innovation Promotion Association.