一种微指令序列调度数据流的星载卷积神经网络 FPGA 加速器

郭子博[1)] 刘 凯[1)] 胡航天[1)] 李奕铎[1)] 璩泽旭[2)]

[1)](西安电子科技大学计算机科学与技术学院 西安 710000)

[2)](中国空间技术研究院西安分院 西安 710000)

摘 要 卷积神经网络(Convolutional Neural Network,CNN)是目前主流视觉算法不可或缺的关键部分.为提高CNN模型推理速度,学界提出了众多异构加速方法以满足不同场景下的多元加速需求.但如何在资源与能耗受限的在轨卫星上稳定高效地加速CNN仍是极具挑战的课题.为此,本文通过软硬件协同设计,着力优化微指令编码、指令级并行和运算级并行3个加速器设计的关键部分,在星上常见的Xilinx VX690T FPGA芯片上设计实现了一种微指令序列调度数据流的CNN加速器.在软件层面,本文提出一种可扩展的微指令编码格式及相应的编译方法.通过卷积循环分块和算子融合策略实现图级别优化,生成加速器可执行的微指令序列.在硬件层面,本文设计实现了一个由微控制器与逻辑运算器组成的RTL级CNN加速器.微控制器通过粗粒度流水线实现各类指令的并行执行.逻辑运算器通过DSP48E1计算资源级联所构建的计算阵列实现卷积算子的细粒度并行运算.实验结果表明,加速器设计功耗10.68 W,在加速YOLOV3Tiny算法时,峰值吞吐率(Runtime Max Throughput,RMT)达到378.63 GOP/s,计算资源利用效率(MAC Efficiency,ME)达到91.5%.相较典型GPU加速方法,本文的加速器有14倍能效提升.相较同类FPGA加速器,ME有6.3%以上的提升.

关键词 卷积神经网络;微指令序列;现场可编程逻辑门阵列;遥感目标检测;微处理器设计

中图法分类号 TP303 **DOI 号** 10.11897/SP.J.1016.2022.02047

An FPGA-Based Microinstruction Sequence Driven Spaceborne Convolution Neural Network Accelerator

GUO Zi-Bo[1)] LIU Kai[1)] HU Hang-Tian[1)] LI Yi-Duo[1)] QU Ze-Xu[2)]

[1)](*School of Computer Science and Technology, Xidian University, Xi'an* 710000)

[2)](*CAST-Xi'an Institute of Space Radio Technology, Xi'an* 710000)

Abstract Recently, with the evolvement of space remote sensing technology, the main earth observation device has been gradually transitioning from the single-satellite to a constellation composed of light and small satellites. A constellation of several high-resolution satellites collects hundreds of TBs (Terabytes) of RSI (Remote Sensing Image) data every day. The traditional satellite-to-ground data transmission mechanism has been unable to match the massive remote sensing data processing. In-orbit satellites need to improve their data processing capabilities to deal with increasingly complex observation missions. Meanwhile, in the field of RSI processing, deep learning algorithms based on CNN(Convolutional Neural Network) have become the mainstream method due to their excellent performance. However, the computation-intensive and memory-intensive features have brought many challenges to the deployment of CNN. Academia and industry propose

收稿日期:2021-09-26;在线发布日期:2022-04-11.本课题得到国家自然科学基金(62171342,61850410523)和空间测控通信创新探索基金(201701B)资助.**郭子博**,博士研究生,中国计算机学会(CCF)学生会员,主要研究方向为嵌入式片上系统、深度神经网络模型压缩技术.E-mail:zbguo@stu.xidian.edu.cn.**刘 凯**(通信作者),博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为高速图像视频编码、嵌入式片上系统.E-mail:kailiu@mail.xidian.edu.cn.**胡航天**,硕士研究生,主要研究方向为嵌入式片上系统.**李奕铎**,硕士研究生,主要研究方向为深度神经网络模型压缩技术.**璩泽旭**,硕士,高级工程师,主要研究方向为空间数据传输与处理技术.

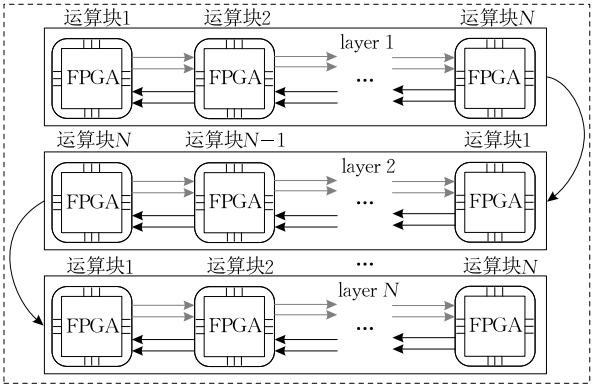
many specific acceleration methods for the CNN domain to cope with the various application scenarios. Numerous FPGA (Field Programmable Gate Array) and ASIC (Application Specific Integrated Circuit) accelerators have been designed to accelerate CNN in edge and data center scenarios. Compared with ASIC, FPGA has higher flexibility and faster development iteration speed, making it very suitable for spaceborne scenarios. In this paper, we propose a microinstruction driven CNN Accelerator for RSI processing on FPGA. This accelerator is jointly designed by software and hardware, which mainly optimizes microinstruction coding, instruction-level parallelism (Coarse-Grained Parallelism) and operation-level parallelism (Fine-Grained Parallelism) under the constraints of limited storage bandwidth and computing resources on satellites. At software level, we propose an extensible microinstruction encoding format and the corresponding compilation method (Micro Assembler). A microinstruction code covers 14 instructions in 4 types, which can schedule the dataflow between different components of the accelerator. The micro assembler performs graph-level optimization on the CNN topology by convolutional loop tiling and operator fusion, and then generates micro-instruction sequences that can be executed by the accelerator. At hardware level, we design and implement an RTL (Register Transfer Level) CNN accelerator, which is mainly composed of micro controller and logic operator. The micro controller achieves the parallel execution of different types of instruction by a 5-stage coarse-grained pipeline (Data Load, Data Fetch, Compute, Post Process, Write Back). The logic operator is a computing array with DSP48E1 hard core resources cascaded, which can achieve parallel execution of convolution operations by a 32-stage fine-grained pipeline. When the pipeline is established, the logic operator can complete 32×32 MAC (Multiply-accumulate) operations in one clock cycle. The performance of our proposed accelerator is evaluated on the Xilinx VX690T FPGA chip commonly found on satellites. The designed power consumption is 10.68 W. The RMT (Runtime Max Throughput) reaches 378.63 GOP/s, and the ME (MAC Efficiency) reaches 91.5%. When our accelerator is used as a coprocessor to accelerate the CNN object detection algorithm YOLOV3Tiny, the average accuracy of the RSI data set reaches 0.9 and the detection speed reaches 102 frames/s. The evaluation results show that our accelerator is 14 times more energy efficiency than the typical GPU acceleration method, and has more than 6.9% improvement in ME compared with other FPGA accelerators.

Keywords CNN; microinstruction sequences; FPGA; remote sensing object detection; microprocessor design

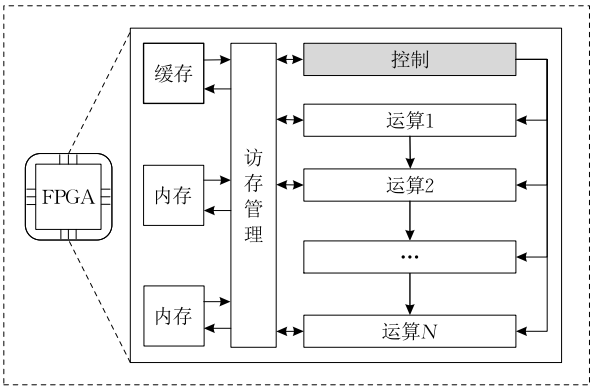
1 引 言

近年来,随着深度学习技术的发展,基于 CNN 的视觉算法展现出卓越性能[1-3].但计算密集、访存密集的特性给其在实际应用场景中的部署造成了障碍.为此,在通用处理器研究之外,学术界与工业界提出了诸多软硬件专用加速方法[3-6]以满足不同场景下 CNN 的多元加速需求.然而,在存储与计算资源受限的在轨卫星上加速 CNN 仍是一个极富挑战的任务.

现场可编程门阵列(Field Programmable Gate Array,FPGA)是星上常用的算法加速平台[7].其较 GPU[2]具有更低的功耗、较 ASIC[4-6]有更高的灵活性,是星载 CNN 加速器极佳的硬件选择.目前,基于 FPGA 的 CNN 加速器主要分为两类,分别为流模式(Stream)与单处理单元模式(Single Engine)[8-9].Stream 模式结构如图 1(a)所示[10],其将各优化的运算块连接形成流水线结构.当流水线建立后,各运算块即可实现并行运算.而大规模的并行运算使得该模式有较高的加速效率.但在获得高吞吐率的同时会牺牲加速器的模型适应性并消耗更多计算资源.Single Engine 模式如图 1(b)所示[11],该结构仅有一个运算单元,控制单元驱动数据在各运算单元



(a) Stream模式



(b) Single Engine模式

图 1　主流加速器结构图

间流传输以完成所有运算. 各运算单元间的粗粒度并行计算以及各运算单元内的细粒度并行计算实现了加速目标. 该模式有效降低了资源消耗，且控制码驱动数据流的形式也使其有更广泛的模型适应性. 但拥有这些优势的同时，Single Engine 模式也受到内存带宽的限制，难以达到与 Stream 模式相同的吞吐率.

在轨卫星有效载荷的资源十分珍贵. 以我国嫦娥二号探月卫星为例，166 kg 的载荷(含 136 kg 科学载荷和 30 kg 工程载荷)仅占发射重量 2480 kg 的 6.69%[12]. 且通常情况下，单个载荷还需进行冗余备份以提高其在空间辐射环境下的容错性[13]. 故用于专用算法加速的 FPGA IP 核(Intellectual Property Core)需要尽量压缩资源使用. 高吞吐率的 Stream 模式大量的资源消耗使其并不适合部署在资源受限的星载环境. 另外，卫星的服务周期一般以年计数[14]，Single Engine 模式对于不断更新的 CNN 模型有更快的适配速度. 显然，灵活性更高、资源消耗更少的 Single Engine 模式在星载场景中更具优势.

减少各级缓存之间的数据通讯所造成的计算中断，最大限度利用内存带宽与计算资源是 Single Engine 加速器的设计核心. 为此，本文提出一种软硬件协同设计的加速器. 软硬件设计架构如图 2 所示，在软件层面，本文定义了一种微指令编码格式，设计了该格式的编译方法(微汇编器). 其通过卷积循环分块和算子融合策略实现 CNN 网络拓扑结构的图级别优化，生成一组微指令序列. 在硬件层面，本文设计了微控制器与逻辑运算器. 前者解析并控制微指令序列的执行，通过控制各类指令流水执行来隐藏数据传输时间. 后者执行 CNN 各算子运算，通过细粒度流水线缩短运算时间，有效利用计算资源.

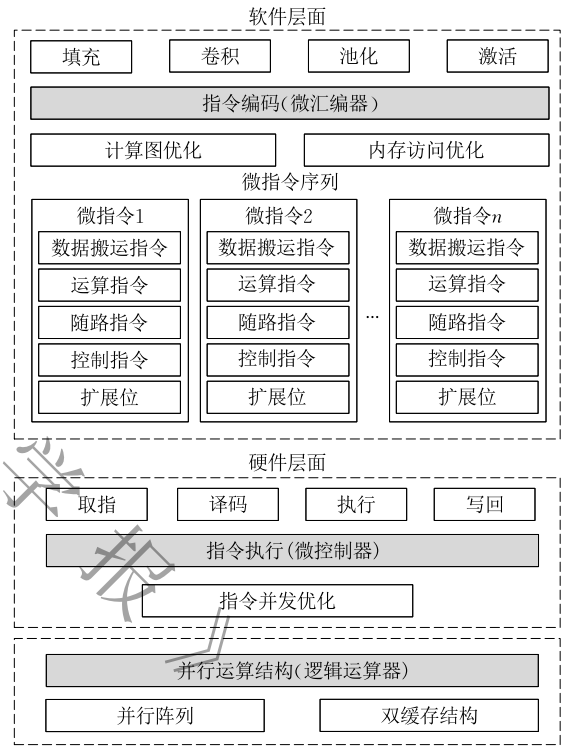


图 2　加速器软硬件设计架构图

本文主要贡献如下：

(1) 提出一种微指令架构. 该架构由可扩展的微指令编码与相应的编译方法(微汇编器)组成. 指令编码格式涵盖 4 类(数据搬运、随路、运算、控制) 14 条指令，可完整描述数据在加速器各组件间的流传输.

(2) 设计实现出一种基于微指令架构的 RTL 级 CNN 加速器. 加速器由用于解析与控制微指令并行执行的微控制器(粗粒度流水线)和用于执行运算指令的逻辑运算器(细粒度流水线)组成.

(3) 在 Xilinx VX690T FPGA 芯片进行了加速器效率评估，在芯片功耗 10.68 W 情况下，RMT 达到 378.63 GOP/s，ME 达到 91.5%. 评估结果表明，本

文的加速器较典型 GPU 加速方法有 14 倍能效提升，较同类 FPGA 加速器有 6.9%以上的 ME 提升.

本文第 2 节详述星载智能算法与 FPGA 加速器的相关工作；第 3 节与第 4 节分别从微指令架构与并行运算架构角度阐述本文提出的加速器的设计细节；第 5 节评估加速器在执行遥感目标检测时的性能，并将本文提出的加速器和已有加速器进行比较讨论；第 6 节总结全文得出结论.

2 相关工作

2.1 星载智能算法研究

目前，在星载智能算法的应用与研究方面已有诸多进展. 我国长光卫星技术公司研制的吉林一号[15]光谱卫星于 2019 年 1 月“一箭双星”发射，该卫星搭载了基于多核 DSP 的模式识别技术，具有森林火点和海面船舶的自动识别、搜寻和定位功能，有效提高了卫星数据回传效率与信息获取的时效性. 但 DSP 硬件平台的并行度比之 FPGA 与 ASIC 不具优势，难以加速卷积神经网络这类大规模并行运算.

除卫星已应用的技术，还有一些以星载智能算法为背景的研究. Wei 等人提出一种应用于遥感任务的混合类型神经网络模型量化方法[16]. 在 Xilinx KC705 FPGA 上设计出一个 7 层神经网络的加速结构，在 MSTAR 遥感数据集上的分类准确度达到 97.42%，但固定的硬件设计使其难以泛化至其它模型. Nong 等人提出一种基于嵌入式设备的轻量化遥感目标检测方法[17]，在 YOLOV3Tiny 网络模型的基础上增加了空间注意力模块以增强遥感目标的特征，并在嵌入式平台 Nvidia Jetson Xavier NX 上对算法进行评估，实时检测速度可达 32.5 帧/s. 该方法有效提升了 YOLOV3Tiny 模型的检测精度，并使用低功耗设备实现了算法加速. 但通用设计的嵌入式设备在加速效率方面明显弱于 FPGA 与 ASIC，且这类设备的抗辐射加固方法较少，在空间环境下的稳定性还需更多实验验证. Wang 等人提出了一种星载 SAR 成像与智能处理的单片多处理架构[18]，探索了 SAR 成像算法与 CNN 算法的流水执行方法. 该架构使用一种带状 Tile 化数据处理方案，实现了多任务模型地流水执行. 在 Synopsys 仿真平台下，基于该架构的芯片在加速 VGG-11 网络时，能效可达 5.4TOPS/W. 该文详述了深度流水线的设计细节，但并未详述运算指令编码与执行的设计细节.

2.2 FPGA 加速器研究

FPGA 加速器设计方面，Stream 模式与 Single Engine 模式都有诸多研究着力优化与解决其设计模式自身的缺陷.

在 Stream 模式方面：Li 等人提出了一种端到端的 CNN 加速器[19]，其将 AlexNet 模型[2]的所有层映射到一块 FPGA 芯片上，不同层可在流水线结构中并行计算. 该加速器在 156 MHz 时钟频率情况下吞吐率达到了 565.94 GOP/s. Liu 等人提出了一种自动化生成 CNN 加速器硬件代码的实现方法[20]. 该方法设计了 4 种典型的参数化细粒度并行结构与 1 种 CNN 高级描述语言. 其通过解析描述语言中各层的参数来实例化不同层的硬件代码，有效提升了开发效率，加快了加速器对新模型的适应速度. Nguyen 等人提出一种高吞吐率的 RTL 级 CNN 并行加速结构[21]，在 Xilinx VC707 FPGA 上高能效加速了 YOLOV2Tiny 模型[22]. 该结构将压缩后的权重直接存入 BRAM 从而减少高能耗的 DDR 访问，优化片上存储资源的使用. Farrukh 等人提出了一种基于 Booth Multiplier 和 Wallace Tree Adders 的硬件结构[23]. 该结构有效降低了 FPGA 硬核 DSP 资源的使用，在 Xilinx ZC706 FPGA 上高能效加速了 YOLOV2Tiny 模型[22].

在 Single Engine 模式方面：Gokhale 等人提出一种高效 CNN 加速器 Snowflake[24]. 该加速器将多维卷积运算分解为多组一维矢量运算，每组运算称为一个 Trace. 并基于 Trace 设计了控制指令与 vMAC 并行运算单元，通过数据重组实现了更高的运算单元利用率. 在 Xilinx ZC706 FPGA 平台，该加速器加速效率达到 116.5GOP/s，ME 达到了 91%. Zhang 等人提出了一种卷积和全连接统一表示的 CNN 加速器 Caffeine[25]. 其将卷积和全连接运算均转换为矩阵乘运算进行统一表示，并设计了粗细两种粒度的流水线，优化了计算资源与带宽的利用率. 另外，其使用了高层次综合技术（High-level Synthesis，HLS）来实现加速器结构以适应不同 FPGA 平台. 在 Xilinx KU060 与 VX690T FPGA 上，加速器峰值加速效率分别达到 365 GOP/s 与 636 GOP/s. Guo 等人提出了一种将 CNN 模型部署至嵌入式 FPGA 设备的完整工具链 Angel-Eye[26]. 其包含有效降低存储资源的量化策略、提高模型适应性的编译工具以及充分利用计算资源的加速器并行

结构. 在 Xilinx Zynq XC7Z045 FPGA 平台上，加速器达到 137GOP/s(16 bits)的加速性能. Abdelfattah 等人提出了名为 DLA 的深度学习加速器[27]，该加速器引入了一种针对深度神经网络推理的可重配机制，使用一个超长指令字(Very Long Instruction Word，VLIW)来实现数据流控制，而该部分仅占用约 1%的资源开销. 在 Intel Arria 10 FPGA上，加速器实现了 ResNet101[28] 与 GoogleNet[3] 模型. Yu 等人提出了一种可重配 CNN 处理器(Overlay Processor Unit，OPU)[29]，该处理器采用了一种复杂的非顺序指令架构，每条指令执行时间不定且指令间存在依赖关系. 凭借这种细粒度指令，OPU 有着较高的计算并行度，在资源较少的 Xilinx KC705 FPGA 上实现了 ResNet[28]、VGG[30]、YOLOTiny[31] 等网络模型. Yu 等人提出了一种用于快速目标检测任务的 CNN 加速器[11]. 该加速器使用一种转移最低分策略进行指令集优化，最大程度上减少 DDR 的访问以降低功耗，有较高的运算效率.

3　微指令架构

本文提出的加速器总体架构如图 3 所示，其分为 HostPC 端的微指令架构(Micro Instruction Architecture，MIA)与 FPGA 端的并行运算架构(Parallel Operation Architecture，POA)两部分. MIA 微汇编器将 CNN 前向传播过程编译为数条微指令构成的序列(序列预加载到片上指令寄存器 ROM). POA 微控制器逐条将微指令序列解析为控制码，进而驱动数据在逻辑运算器间流传输，完成数据流调度(Data Schedule).

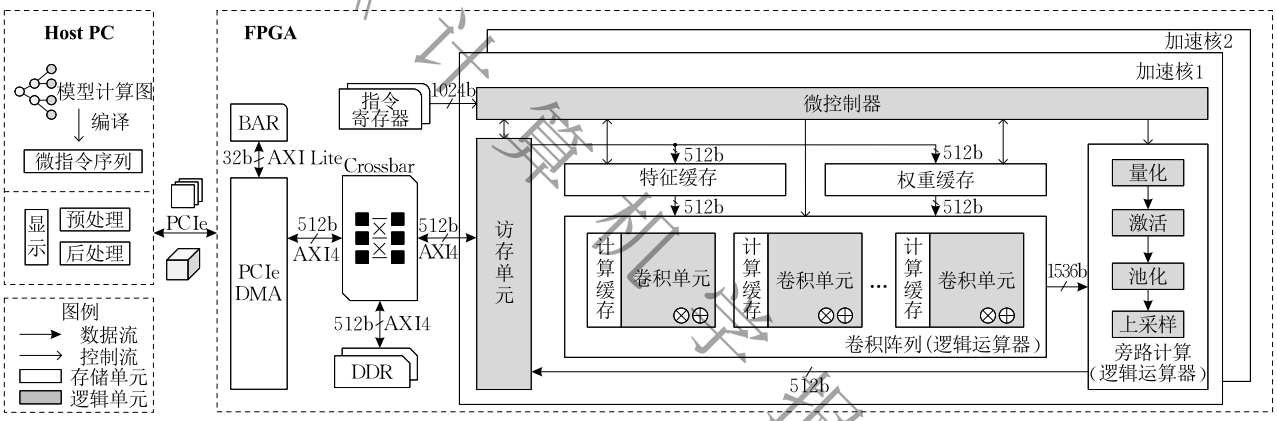


图 3　加速器架构图

MIA 架构基于 FPGA 存储资源特性定义了一组可扩展的微指令编码格式. 基于 CNN 模型与外部存储设备特性设计了该格式的微汇编器.

3.1　微指令编码格式

与复杂指令集系统(Complex Instruction Set Computing，CISC)固定的缓存与运算器分布不同，FPGA 可根据需求自行定义片上资源. 例如，对于 Block RAM 等片上缓存可自定义深度与数据位宽. 故与 CISC 等传统指令集架构以字节为编码单位不同，本文使用更小的位(bit)作为编码单位以压缩存储资源使用. 与 VLIW 架构类似，本文将 4 类(数据搬运指令、随路指令、运算指令与控制指令)长度各异的指令打包为一条微指令.

一条微指令包含数据在加速器各组件间流传输的所有控制码，编码格式如表 1 所示. 数据搬运指令由 6 条涉及存储访问的指令组成. 各指令名称明确了其源与目标的存储类型与数据类型. 源与目标存储分为片外内存(m)、片上缓存(r)、立即数(i). 传输的数据类型分为特征图(f)、权重(w)、偏置(b). 例如 mrmovf 中第一个 m 表示数据源为片外内存、r 表示目标为片上缓存、最后的 f 表示数据类型为特征图. 随路指令由 psum、quant、relu 3 条指令组成，该类指令在数据流传输过程中执行，无需复杂逻辑运. 指令编码中主要包含了一些控制信息. 例如，psum 指令表明了数据流是否需要累加中间结果，quant 明确了量化移位信息. 运算指令是 CNN 中必须的逻辑运算，包括 conv、pad、pool、upsample 4 条指令. 4 条指令由不同的逻辑运算单元并发执行，指令编码中包含了各运算的参数信息. 控制指令仅设计了结束标识指令 end，标识了该条微指令是否为微指令序列中最后一条.

表 1 微指令编码

指令类型	指令名	地址	长度
数据搬运指令	mrmovf	0x000	109 bits
	rrmovf	0x06d	74 bits
	mrmovw	0x0b7	109 bits
	rrmovw	0x124	74 bits
	rimovb	0x170	9 bits
	immovf	0x179	73 bits
随路指令	psum	0x16e	2 bits
	quant	0x1c2	13 bits
	relu	0x1fe	1 bit
运算指令	pool	0x1cc	3 bits
	upsample	0x1ce	1 bit
	pad	0x1cf	6 bits
	conv	0x1d5	26 bits
控制指令	end	0x1fd	1 bit

与 VLIW 通过汇编器明确指令并发关系不同，本文的微指令间有明确依赖关系，依靠微控制器保证指令执行顺序. 这样固定的数据流路径将消除指令间执行顺序的控制码，简化编码格式与微汇编器的设计. 单条微指令编码长度为 1024 位(bits)，前 510 位为有效位，其余位为预留扩展位. 当使用扩展位时，需在汇编器与微控制器中添加扩展指令的生成与解析方法.

3.2 微汇编器设计

一条微指令是加速器在一次数据流调度中各组件控制码的集合. 微汇编器作为媒介，则需将数据在 CNN 模型的前向传播过程划分为加速器中数次数据流调度，后根据每次调度信息生成二进制微指令编码. 编译流程如图 4 所示，对于计算图与模型，微汇编器首先进行适应硬件数据流的计算图优化并生成运算与随路指令. 后依据运算需求明确涉及访存的数据搬运指令.

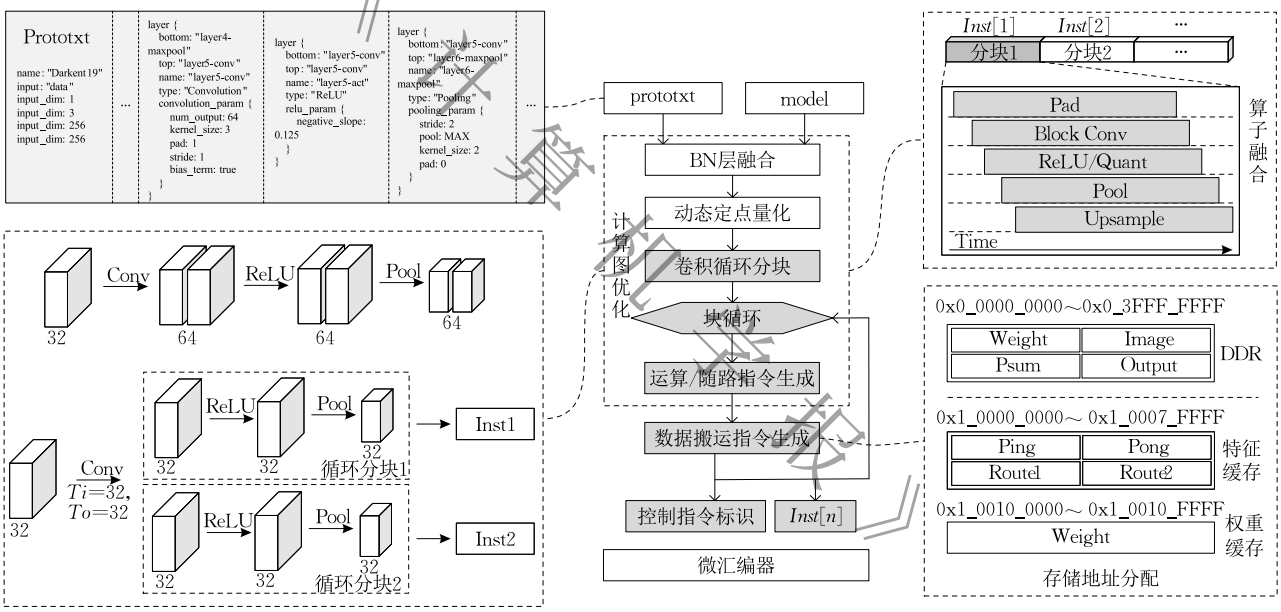


图 4 微汇编器编译流程

3.2.1 计算图优化

计算图是网络模型拓扑结构的一种描述，描述了数据在模型中的流动过程. 不同深度学习框架有不同的描述风格. 本文选用 Caffe[32] 中的“Prototxt”文件表述模型结构，如图 4(左上)所示. 其中“Layer”中包含了模型各层的信息. bottom 为输入的前置层数据，top 为该层输出数据，type 为层类型，param 为层参数.

对于计算图和模型，微汇编器会进行适应硬件的计算图优化. 优化包括：

(1) 通过 BN 层[33] 融合可进行部分预计算，减少推理计算量.

(2) 通过动态定点量化方法对模型进行参数压缩，减少存储资源使用.

(3) 对于单层规模较大的卷积算子进行如图 4(左下)的循环分块划分.

(4) 根据卷积分块将其他算子拆分后融合至一条数据流中. 如图 4(右上)所示，一条微指令(一个分块)中融合了多类运算.

其中，BN 层融合与动态定点量化是嵌入式设备常见的模型压缩方法[25,29,34]. 但对模型进行量化压缩后，冗杂的 CNN 模型参数与并不充裕的 FPGA 片上缓存之间仍有较大差距. 图 5 展示了 5 种常见 FPGA 的片上缓存大小与 4 种规模较小的 CNN 模

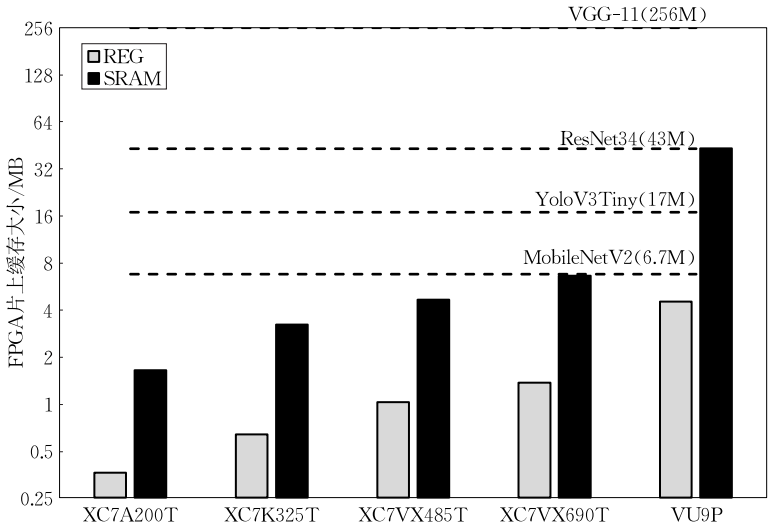


图 5　Xilinx 各系列 FPGA 片上缓存统计图

型(16 bits 量化)参数大小对比.可以看出,即使是星上常见的高性能 FPGA 芯片 VX690T,其 6.6 MB 的片上 SRAM 也难以将常见 CNN 模型的全部参数存储到片上缓存.为降低中间特征图与权重对片上缓存的依赖,CNN 加速器常用卷积循环分块策略进行卷积算子优化[9].该策略通过增加循环层次的方式减少单循环体内的运算规模,从而降低单次数据流调度所需的片上缓存大小.

本文使用了一种沿通道维度分块的策略对卷积算子进行循环分块.该策略在原始卷积的基础上加入了输出维度(内层)与输入维度(外层)的循环,当定义输入维度分块因子为 Ti,输出维度为 To 时,内层循环次数 $Itero$ 与外层循环次数 $Iteri$ 如式(1)、(2).循环分块卷积运算如式(3),单个分块仅进行输入特征(F_{in})的 Ti 维度与 To 个卷积核(W)的 Ti 维度的卷积运算.

$$Itero=C_o/To \tag{1}$$

$$Iteri=C_i/Ti \tag{2}$$

$$F_{\text{out}}=\sum_{t}^{Iteri}\sum_{j}^{Itero}Conv(F_{\text{in}}(t\times Ti),W(j\times To,t\times Ti)) \tag{3}$$

图 4(左下)为 $C_i=32$,$C_o=64$,Ti,$To=32$ 时,卷积分块示意(为保证输入输出特征格式的一致性,Ti 应与 To 相同,后文以 T 指代 Ti 与 To).根据式(1)、(2)可得 $Itero$ 为 2,$Iteri$ 为 1,故仅需将输出特征沿通道维度拆分为 2 块即可.对于该卷积运算,中间特征与权重所占的片上存储就降低了一半.

另外,CNN 各层对前置层数据并无完全依赖性.在加速器中,数据在不同单元间是流传输,因此无需缓存全部中间特征图,而是采用“即来即算”的方式.故在卷积算子分块后可将其他算子融合至一次数据流调度.硬件中的流运算如图 4(右上)所示,可以看出当一段延时后 5 种运算将完全并发运行.因此汇编器以一个卷积分块为核心,将不同类型的其他层运算参数写入同一条微指令中,即一条微指令可驱动数据流完成一个卷积分块与其所需的其他类型算子运算.微指令序列的指令条数也由分块的个数决定(当分块数据量大于片上缓存时生成多条微指令表示).单层卷积所需的微指令条数如式(4)所示.

$$Inst\_num=\lceil C_o/T\times C_i/T\rceil \tag{4}$$

参数 T 与片上缓存设计、微指令序列条数和指令执行效率都有很强相关性,故 T 的选择十分重要.表 2 对比了加速 YOLOV3Tiny 模型时,不同分块参数对存储资源与计算效率的影响.

表 2　不同 T 参数的加速效率对比

T	最大分块大小/KB	有效计算占比	加速效率
8	4.5	0.96	7.68
16	9.0	0.94	15.04
32	18.0	0.84	**26.88**
64	36.0	0.35	22.40

本文提出的逻辑运算器并行计算一个分块的全部维度,即在流水线建立之后,一个时钟周期可并行计算 T 个乘累加运算.若将串行计算速度设为 1,T 则是其加速效率.但当通道数 C_i,C_o 小于 T 时会产生大量无效运算,这会损失通过并行流水线所获得的性能增益.而 CNN 浅层特征图尺寸较大,通道数较低,故 T 参数较大时会造成大量无效计算,如 T 为 64 时有效的计算量仅占总体计算量的 35%,则其加速效率就仅为 22.40.而 T 为 32 时其加速效率最高.

另外，由于本文使用了镁光 MTKTF51264Hz DDR3 颗粒，其数据线为 64 bits，针对该颗粒 Xilinx 存储控制器(Memory Interface Generator，MIG)可实现长度最长为 8 的突发读写. MIG 的 AXI 接口提供了[32，64，128，256，512]的可选数据位宽. 各位宽的读写效率如图 6、图 7 所示. 在连续读写 4 KB 与 256 KB 数据时，尽管在带宽利用率方面 512 bits 逊于其他总线带宽，但其在读写速度上有明显优势. 而本加速器最大适应 16 bits 量化，T 为 32 时能完全利用 512 bits 总线带宽. 换言之，在一个时钟周期内逻辑运算器可获取一个分块中一个像素的全部维度进行计算. 故本文选择 $T=32$ 进行卷积循环分块并进行算子融合生成运算与随路指令.

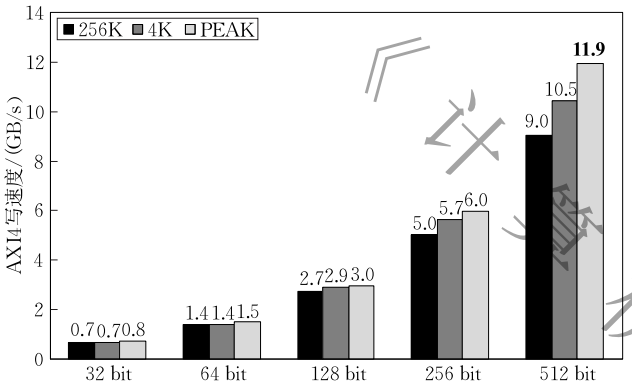


图 6 AXI4 接口 DDR3 写效率

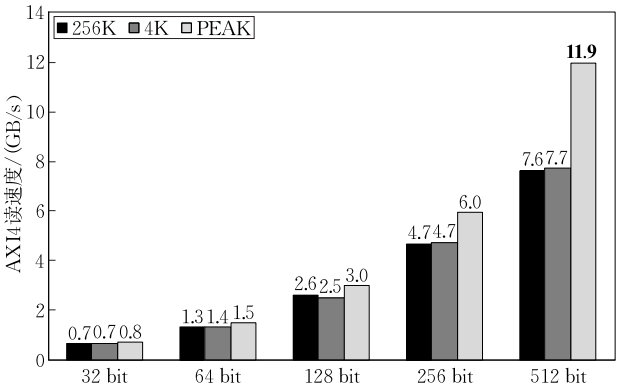


图 7 AXI4 接口 DDR3 读效率

3.2.2 内存访问优化

运算与随路指令明确了一条微指令参与运算的数据量大小，微汇编器依据数据量生成数据搬运指令. 在生成数据搬运指令时优先使用片上缓存进行指令间需复用的中间结果. 各级存储分配如图 4(右下)所示，优化策略如图 8 所示. 由于片上特征缓存采用双端口读写，可通过乒乓结构同时读写，当缓存乒(Ping)参与运算时运算结果写回乓缓存(Pong)，反之亦然. 故在生成写回指令 immovf 时优先分配片上缓存地址，如此可规避后续指令的片外内存访问(mrmovf). 另外，对于 CNN 中常见的路由层(需跨层使用的特征图)，可在特征缓存的 Route1 与 Route2 进行暂存.

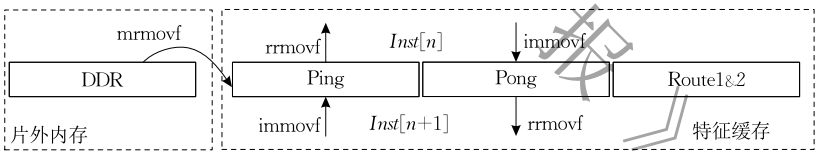


图 8 数据搬运指令优化

3.3 微指令实例

微汇编器生成微指令序列的整体算法流程如算法 1 所示. 首先，根据硬件存储地址初始化数据搬运指令基地址. 之后开始逐条生成微指令，每个卷积层将被划分为 $C_o/T\times C_i/T$ 个微指令执行，在循环体内根据融合的计算图参数执行 3 种类型的指令生成函数. 最后，标识控制指令，微调数据搬运超出片上缓存的微指令.

算法 1. 微指令序列生成算法.

输入：计算图(IR)，分块系数(T)

输出：微指令序列($Inst$)

```
1.  Initial( )                    //初始化存储基地址
2.  Num=0                         //初始化微指令序列下标
3.  FOR t=0 TO t=N                //N 为卷积层数
4.    //t 卷积层输入分块循环
5.    FOR i=0 TO i=IR[t].Ci/T
6.      //t 卷积层输出分块循环
7.      FOR j=0 TO j=IR[t].Co/T
8.        //根据层参数生成该条微指令的运算指令
9.        OpInstGenerator(Inst[Num][0x1cc])
10.       //根据层参数生成该条微指令的随路指令
11.       FlowInstGenerator(Inst[Num][0x16e])
12.       //计算该条微指令数据量生成数据搬运指令
13.       MovInstGenerator(Inst[Num][0x000])
14.       Num=Num+1
15.     END FOR
16.   END FOR
17. //在序列最后一条微指令将控制指令 end 置为有效
18. Inst[Num][0x1fd]=1
19. FineTuning( ) //微调特征图超出片上缓存大小的
                  微指令
20. END FOR
```

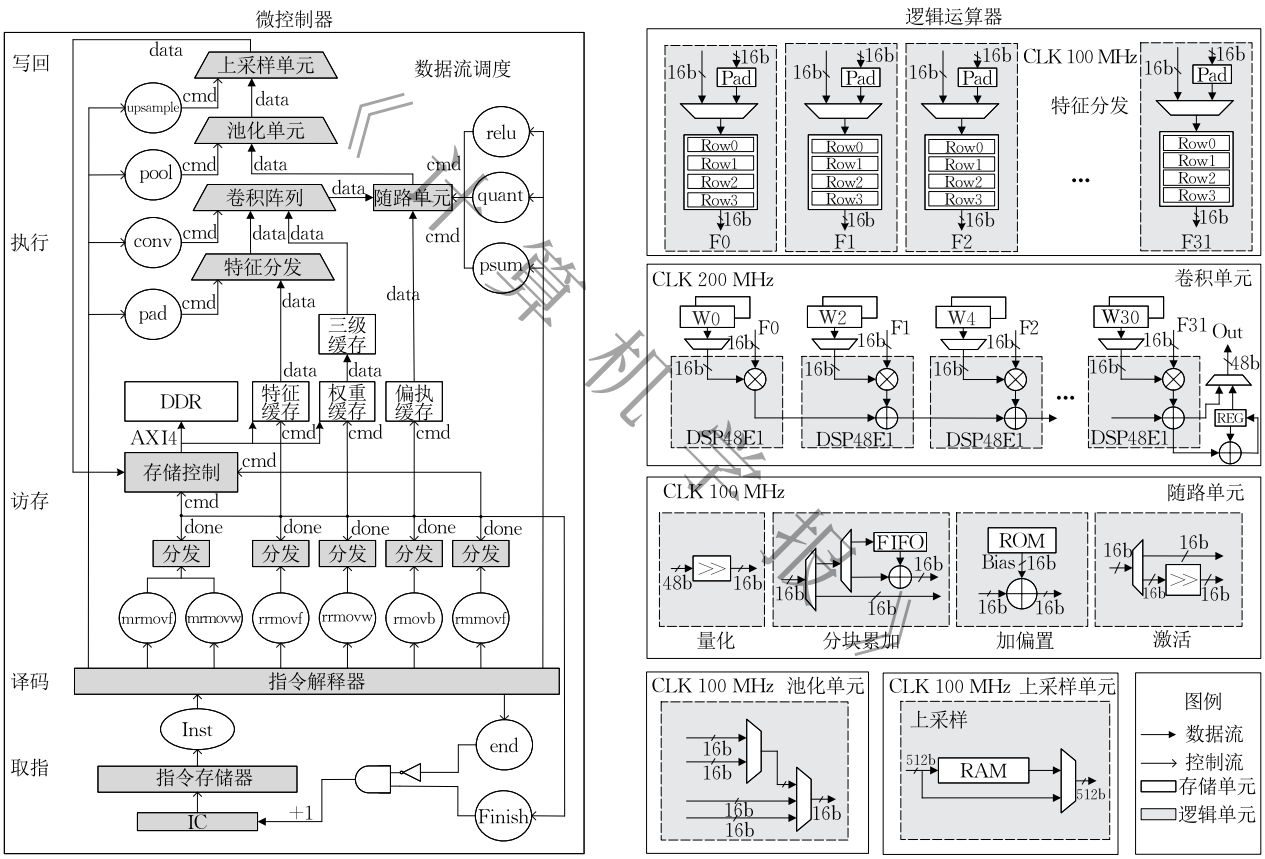
附录 1 为一条完整二进制微指令示意，这条微指令包含了一次数据流调度的全部控制码. 该次调度为输入特征图(宽×高×通道：64×256×32)经过填充、卷积(1×1 步长为 1)、激活、量化、池化(2×2 步长为 2)运算后写回输出特征图(宽×高×通道：32×128×32)的过程. 数条微指令构成的微指令序列将包含 CNN 推理所需的全部控制码.

4　并行运算架构

在硬件层面，本文设计了 RTL 加速器以并行执行 MIA 生成的微指令序列. 如图 9 所示，加速器的 POA 包含了指令级并行结构(微控制器)与运算级并行结构(逻辑运算器). 微控制器首先解析并分发运算与随路指令至逻辑运算器各运算单元(特征分发、卷积阵列、随路单元、池化单元、上采样单元)，其次解析数据搬运指令并驱动数据在逻辑运算器内流传输. 逻辑运算器则并行执行各类运算与随路指令.

4.1　微控制器设计

微控制器将二进制微指令解析为各运算单元控制码，进而驱动存储控制单元完成数据在各逻辑运算器间的流传输. 其调度流程如图 9(a)所示，包含了取指、译码、访存、执行、写回过程.



(a) 微控制器　　(b) 逻辑运算器

图 9　并行运算架构示意

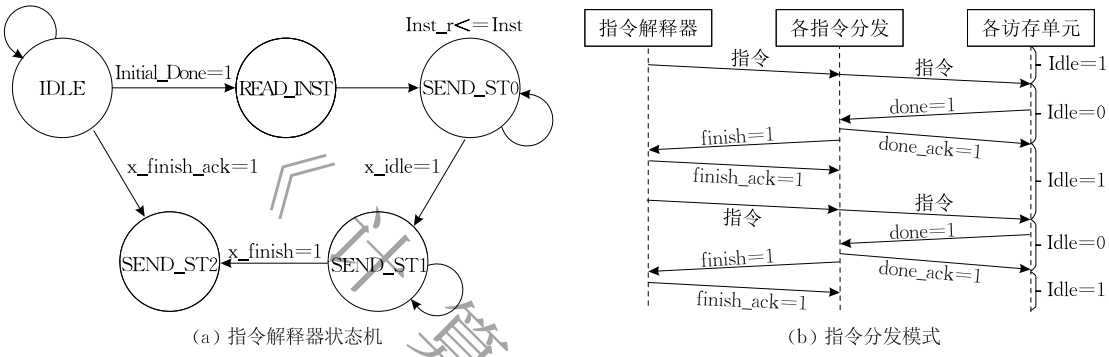
4.1.1　取指

本文的微指令序列预加载在片上的指令存储器(ROM)中，可通过类似刷新固件(Firmware)的形式在综合前预加载不同的指令序列，实现不同网络模型的推理. PCIe 传输数据完成后，初始化信号转变为高电平，指令解释器开始译码. 而指令存储器从基地址 0x000 开始取指，当数据搬运指令执行状态均为 finish 时写回完成，该条微指令执行结束. 若该条微指令并非序列最后一条，则指令计数器(Instruction Counter，IC)加 1 同时地址加 1，开始取下条指令.

4.1.2　译码

微指令的解析与分发由指令解释器状态机与分发状态机组成. 如图 10(a)所示，指令解释器共有 5 种状态，分别为初始的空闲状态(IDLE)、读指令状态(READ_INST)、初始化状态(SEND_ST0)、指令分发状态(SEND_ST1)、指令响应状态(SEND_ST2). 这 5 种状态贯穿整个译码、访存执行、写回过程. 当外部权重与图像数据加载完成后，译码过程开始状

态由 IDLE 跳转至 READ_INST，此时从指令存储器中读出微指令；后状态直接跳转至 SEND_ST0，此时对各单元待接收控制信号进行初始化赋值；当所有指令分发单元处于空闲状态时，状态跳转至 SEND_ST1，此时按照微指令编码规则将微指令不同字段进行拆分并赋值给各单元控制信号，完成指令码到控制信号的转变. 该过程结束后，译码过程结束；当各单元执行结束后跳转至 SEND_ST2 状态，此时回复所有分发单元响应信号(ACK)，进而将各分发单元状态恢复至空闲状态，等待下一条指令分发.



(a) 指令解释器状态机　　(b) 指令分发模式

图 10　译码单元示意

4.1.3　访存、执行与写回

译码完成后 3 组随路指令，4 组运算指令被发送至随路单元与各逻辑运算单元. 6 组数据搬运指令被分发到访存单元进而完成各级缓存间的数据传输. 各运算单元接收控制参数后等待数据流进行运算，如图 9(a)所示，cmd 为控制参数，data 为数据流.

本文的加速器存储体系分片外内存(DDR)、片上缓存(Block RAM)与运算单元缓存(Distribute RAM)3 级. 片外内存与片上缓存使用 AXI4 总线协议进行数据传输，同时片上缓存与 3 级运算单元缓存间的数据传输使用普通接口(Native Port)，两种数据传输可并发进行. 5 组数据搬运指令执行流水线如图 11 所示.

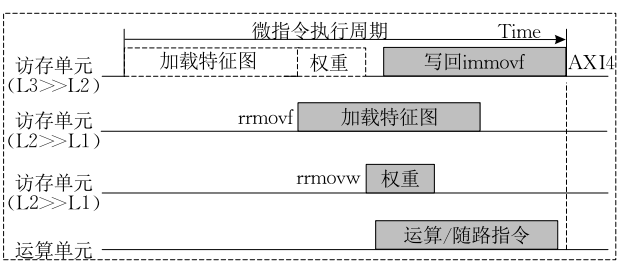


图 11　数据搬运指令执行流水线

其中 mrmovf(加载特征图)与 mrmovw(权重)并非每条微指令都需执行，其包含的 enable 信号标识了是否需要从 DDR 加载数据. 另外，4 组运算指令与随路指令执行流水线如 3.2 节图 4(右上)所示.

SEND_ST1 状态时 14 条指令已译码完成并分发至运算单元与分发单元. 运算单元接收控制参数后等待数据进行运算. 分发单元则将指令分发至各缓存单元进行数据搬运，驱动数据在各运算单元间流传输. 与此同时，分发单元作为媒介将访存进程反馈至指令解释器，交互模式如图 10(b)所示，5 个分发单元与访存单元间通过 done 与 done_ack 信号进行交互，与指令解释器间通过 finish 与 finish_ack 信号进行交互. 所有分发单元完成后表明数据已完成各单元间的流传输. 指令解释器回复确认信号后进入空闲状态等待序列下一条指令的译码与执行.

4.2　逻辑运算器设计

微控制器控制各类指令并发执行所形成的粗粒度流水线(指令级并行)与逻辑运算器各与运算单元内部的细粒度流水线(运算级并行)支撑了加速器的运算能力. 逻辑运算器各运算单元的结构如图 9(b)所示.

4.2.1　卷积阵列

图像三维卷积运算的本质是图像各通道维度像素与对应卷积核权重的乘累加运算. 一个 32 通道维度像素与两个卷积核进行 1×1 卷积运算的过程如图 12 所示，输出特征图的一个像素就需要 32 次乘累加运算.

本文的卷积逻辑运算器是 16 个卷积单元构成的运算阵列. 如图 9(b)所示，每个卷积单元由 32 个级联的 Xilinx DSP48E1 硬核 IP 与 64 个权重三级缓存(Distribute RAM)组成. 其中第 1 级 DSP 计算模式为 $A\times B$，其余为 $A\times B+PCIN$. 在最后一级 DSP 之后是一个暂存逻辑. 当卷积核大于 1×1 时，输出结果需要累加多个时钟的运算结果，故需对中间结果进行累加并存入 REG(寄存器)类型变量中，当达到累加次数后输出结果. 如此每个卷积单元就构成了 32 级的乘累加器(Multiply Accumulate，MAC).

每个卷积单元的细粒度流水运算模式如图 12

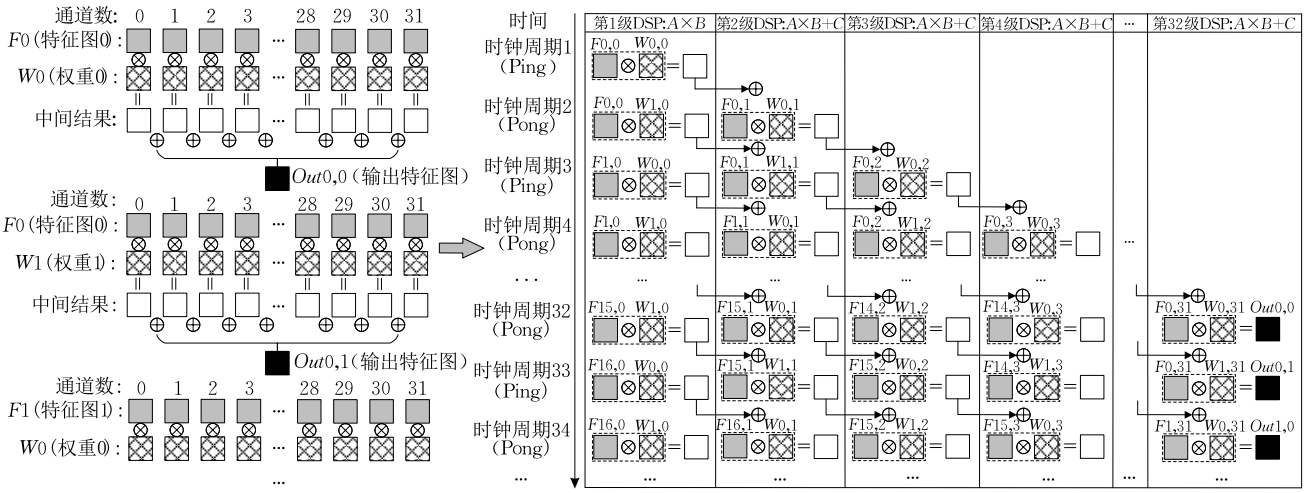


图 12　卷积运算细粒度流水线

所示，纵方向代表时钟周期，横方向代表 DSP 级数. 在一个时钟周期内，每级 DSP 接收上一级 DSP 的结果进行乘累加运算，而当 32 个时钟周期之后输出第一个结果. 换言之，在流水线建立之后，每个卷积单元在一个时钟周期内可得到输出特征图的一个像素. 另外，为释放 DSP48E1 的性能，本文的卷积单元使用了两倍于系统的时钟频率. 在乒(Ping)时钟周期与乓(Pong)时钟周期送入相同的特征图与不同的权重. 因此，每个卷积单元在一个系统时钟周期内，可完成图 12 所示的特征图($F0$)与两个卷积核($W0$,$W1$)的 1×1 卷积运算. 同理，16 个卷积单元组成的卷积阵列可实现 32 个卷积核的并行运算. 故本文的卷积逻辑运算器仅使用 16×32 个 DSP48E1 资源就构建了 32×32 的 MAC 阵列. 在一个系统时钟周期内最高可实现 1024 个乘累加运算，相对于同频率串行运算有 1024 倍的速度提升.

4.2.2　特征分发单元

为保证卷积阵列的稳定数据流，本文设计了如图 9(b)所示的特征分发逻辑. 特征数据在 rrmovf 指令驱动下从特征缓存流传输至分发单元. 分发单元将 512 位(bits)的数据进行 pad 后拆分为 16 bits 分别缓存至 32 个缓存块. 每个缓存块包含 4 个 Block RAM，交替缓存并分发数据至卷积单元阵列，在阵列计算的同时其他行可进行缓存以此提高数据分发效率. 16 个卷积单元共享相同的特征数据以提高特征数据复用.

另外，权重数据在 rrmovw 指令驱动下从权重缓存中读出，后直接存入卷积单元中的三级乒乓缓存结构. 权重 $W0$,$W2$,$W4$,…,$W30$ 存入缓存乒(Ping)，$W1$,$W3$,$W5$,…,$W31$ 存入缓存乓(Pong). 当特征数据流水分发至卷积单元的 32 级 DSP48E1 时，Ping 与 Pong 交替分发权重数据. 如此，在特征分发的 1 个时钟周期内，每个卷积单元就完成了两个卷积核的运算.

4.2.3　随路单元与旁路逻辑单元

如图 9(b)所示，随路单元包含了量化、分块累加、加偏置与激活逻辑. 分块累加通过随路指令 psum 控制 FIFO 的读写. 量化与激活则接收 quant 与 relu 指令进行移位运算.

$$LeakyReLU(x)=\begin{cases}x, & x>0\\ \lambda x, & x\le 0\end{cases} \tag{5}$$

$$ShiftLeakyReLU(x)=\begin{cases}x, & x\ge 0\\ x\gg -\log_2\lambda, & x<0\end{cases} \tag{6}$$

另外，对于激活层常用的带泄漏线性整流函数(Leaky ReLU)如式(5)，本文使用移位替代乘法以提高随路单元的运算效率，调整后的计算公式如式(6)(参数 λ 的取值范围：$0<\lambda<1$).

如图 9(b)所示，旁路逻辑单元包含池化逻辑单元与上采样逻辑单元，主要使用缓存与比较器构建. 数据在随路单元与逻辑运算单元间流传输的同时各单元也就完成了其相应的逻辑运算.

5　实验与分析

本文使用 Verilog HDL 实现了 RTL 级的加速器，在 Xilinx VX690T FPGA 上进行了单核与双核实验. EDA 工具为 Vivado2020.2，仿真工具为 ModelSim10.6d. 实际测试环境为 CPU：Intel Core i7-7700 CPU；操作系统：Ubuntu16.04；PCIe 驱动：Xilinx XDMA. 本文选择星载图像处理中常见的遥感目标检测任务进行实验分析，实验流程如图 13 所示.

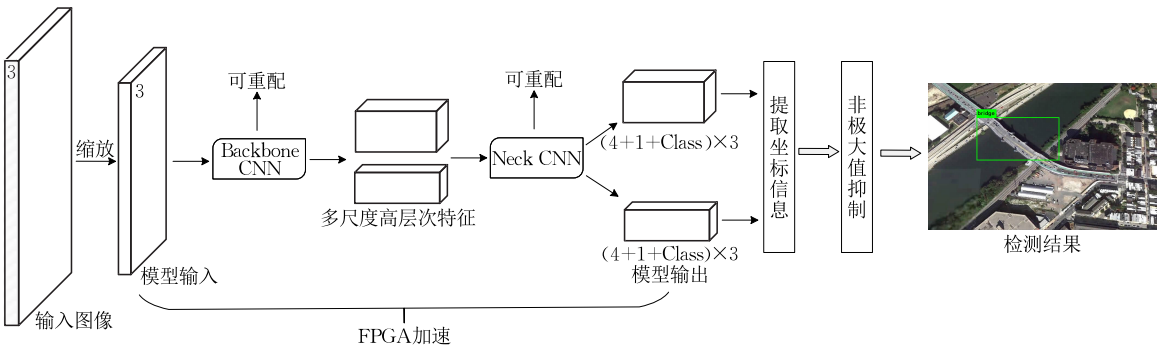


图 13 遥感目标检测实验流程图

5.1 模型优化实验

本文选取 24 层轻量化目标检测算法模型 YOLOV3Tiny[35] 作为实验模型. 评估了本文的量化方法在 CAESAR-Radi 数据集[36] 与高分数据集两个遥感数据集上的目标检测效果.

CAESAR-Radi SAR 遥感数据集以我国高分三号 SAR 数据和 Sentinel-1 SAR 数据为主数据源,由长宽均为 256 像素的 39 729 个船舶切片组成. 高分数据集则来自我国某高分卫星,由采样后长宽均为 256 像素的 579 幅遥感图像组成,其中包含汽车、军用车辆、航空母舰、驱逐舰、巡洋舰、商船、飞机、军用飞机、直升机 9 类目标.

图 14、图 15 为 YOLOV3Tiny 模型在本文的优化策略下,在上述两种数据集中的性能表现. 其中全精度模型在 DarkNet 框架下进行训练,量化在 Caffe 框架下实现. 全精度表示原始模型精度,在融合 BN 层后精度不会发生变化. 量化表示加入循环分块策略与动态定点量化策略后的模型精度. 纵坐标为目标检测领域常见精度指标平均精度均值(mean Average Precision,mAP),横坐标表示测试 mAP 时交并比(IoU)参数的阈值选择(常见选择为 0.5).

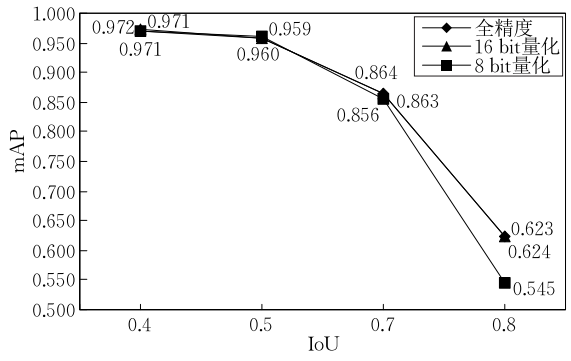


图 14 高分数据集检测精度图

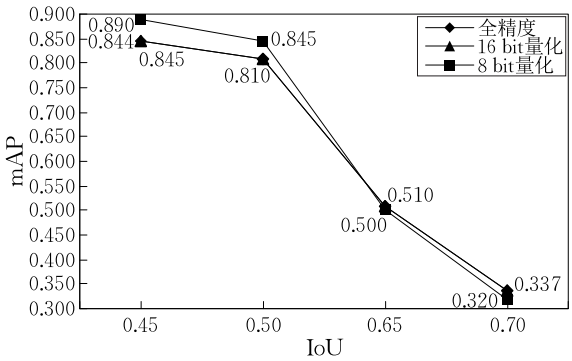


图 15 CAESAR-Radi 数据集检测精度图

图 14 展示了在高分数据集上,量化模型与原始模型的精度对比. 当 IoU 阈值小于 0.7 时,两种精度的量化模型与全精度模型相比没有精度损失. 当阈值为 0.8 时,16 bits 模型与全精度相比无损失,但 8 bits 模型有 0.078 的精度损失. 图 15 则展现了模型在 CAESAR-Radi 数据集上的检测精度对比. 16 bits 量化模型与全精度模型在各阈值下都展现了近乎相同的精度. 8 bits 模型尽管在阈值较小时,精度有一定优势. 但随着阈值变大(大于 0.65),检测精度有一定程度的损失.

由精度对比可知,16 bits 量化与全精度模型相比几乎无精度损失. 而 8 bits 模型在 IoU 阈值较低情况下有着无损甚至更高的精度,但随着阈值的增高,其表现出了精度损失. 说明其稳定性仍不如全精度和 16bits 模型. 但在 IoU 常见设置 0.5 时,两种量化模型均展现了无损的性能. 且量化策略可有效降低存储资源的使用,如表 3 所示. 故本文使用 16bits 量化模型进行加速器效率评估.

表 3 量化模型大小

	模型大小/MB	单层最大参数量
原始	34.8	18 MB
16 bits 量化	16.8	18 KB
8/16 bits 量化	8.5	9 KB

5.2 加速器性能实验与分析

5.2.1 加速器资源消耗评估

本文提出的加速器在 Xilinx VX690T FPGA 芯片的资源消耗如表 4 所示. 表中第 1 行为微控制

器单元与指令寄存器(ROM)的资源消耗. 可以看出本文的指令控制逻辑对片上资源消耗较少，主要资源消耗均小于 2.2%. 表中第 2 行为加速器全部逻辑资源消耗. 由于单个加速器所有资源消耗均未超过 50%，故本文还进行了双核实验，资源消耗如表中第 3 行所示.

表 4　FPGA 资源消耗表

	LUT	FF	BRAM	DSP
指令	2653 (0.06%)	1312 (0.15%)	32 (2.18%)	0 (0.00%)
单核	153767 (35.50%)	136143 (15.71%)	602.5 (40.99%)	517 (14.36%)
双核	268092 (61.89%)	231912 (26.77%)	1134 (77.14%)	1034 (28.72%)

各加速器逻辑单元对于主要逻辑资源 LUT 的消耗占比如图 16 所示. 除与上位机通信使用的 XDMA 核与控制 DRAM 的内存控制器，加速器主要逻辑单元中卷积阵列单元资源消耗最大. 同时卷积阵列使用了 512 个 DSP 资源，是逻辑资源消耗最大的单元. 图 17 展示了主要存储资源 BRAM 的消耗占比，为保证卷积单元的运算效率，特征缓存与随路单元中的分块累加单元消耗了大量 BRAM 来缓存中间特征图.

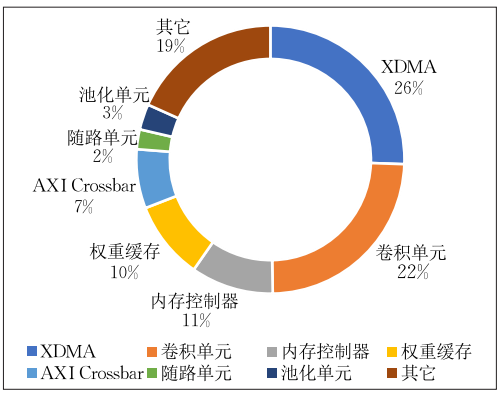


图 16　LUT 资源消耗分布

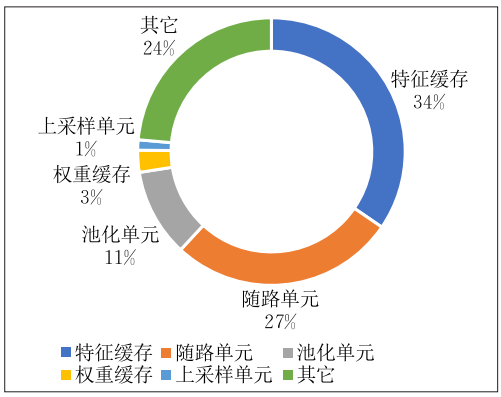


图 17　BRAM 资源消耗分布

根据资源消耗评估，卷积阵列单元作为主要逻辑运算器消耗了更多的资源，但仍有资源余量可供微指令序列扩展其他逻辑运算.

5.2.2　加速器性能评估

图 18 为 YOLOV3Tiny 网络模型 13 层卷积算子的运算量. 该运算量特指乘或加运算，单位为 1×10^9 个操作(Giga Operations，GOP). 灰色为 3×3 卷积，白色为 1×1 卷积. 其中 3×3 卷积运算在模型的乘累加计算量中占比达高达 91.3%，是模型推理核心运算. 微汇编器将该模型编译为 1403 条微指令组成的序列. 各层的微指令条数如图 19 中点线所示(其他层算子已融合至卷积层). 除指令条数，图 19 柱状图展示了各层微指令序列执行效率(双核). 该数据通过式 7 运算得到. 其中 OP 为运算量，T 为运算延时. 各层 OP 如图 18 所示，T 由时钟计数器与时钟主频推算得到.

$$Throughtput=OP/T \tag{7}$$

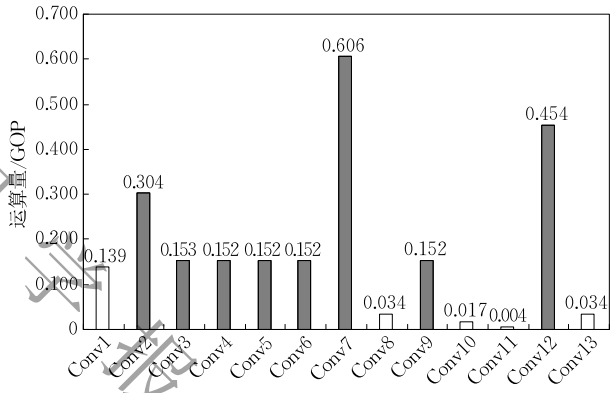


图 18　YOLOV3Tiny 模型卷积层运算量示意图

由图 19 可知，运算吞吐率与微指令条数之间有一定负相关性. 本文沿通道方向进行卷积分块与微指令划分，这使得通道较深层的微指令条数增多. 而微指令条数的增多会带来不可避免的流水中断. 同时，1×1 卷积本身更少的数据复用导致 Conv8、Conv10、Conv11 需要更多次的数据搬运，进而影响了运算效率. 但总体上运算量大的 3×3 卷积都有较高的吞吐率，而峰值吞吐率也达到了 378.6 GOP/s，十分接近计算资源(DSP 使用量：1034)的理论峰值吞吐率 413.6 GOP/s. 另外，影响微指令执行时间的因素还有是否访问 DDR 与指令序列长度. 由图 19 可以看出由于微指令序列 12～19 无需访问 DDR，故其与计算量相同的序列 10～11 相比有更高的吞吐率. 在使用有限的片上存储资源的条件下，微汇编器的内存调度优化策略优先将结果写回到片上存储来减少外存访问. 因此，微指令序列 12～1403 跳过了 mrmovf 指令执行，提高了执行效率.

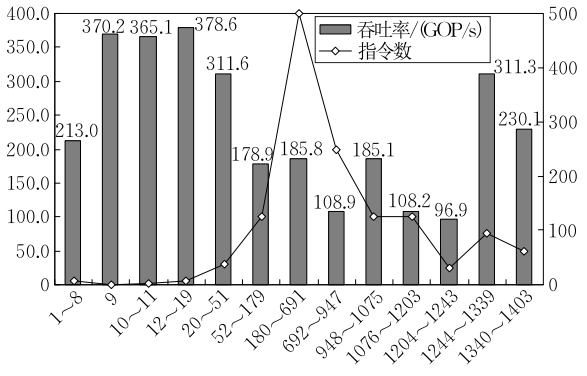


图 19 YOLOV3Tiny 各层运算吞吐率与微指令条数示意图

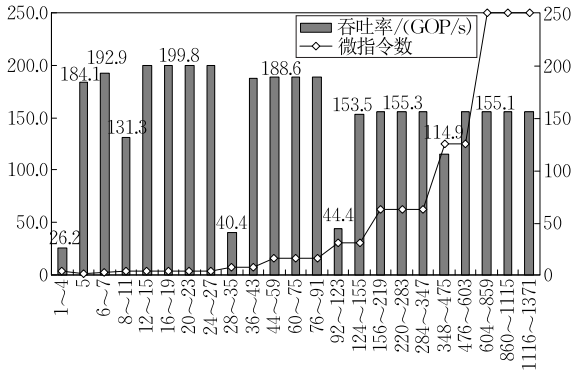


图 21 ResNet18 各层运算吞吐率与微指令条数示意图

除了 YOLOV3Tiny 的模型实验，本文还进行了 ResNet18 模型的性能评估. 本文将 ResNet18 的 23 层骨干网络进行了微指令编译，共生成 1371 条微指令. 并对该网络模型进行了 RTL 行为级仿真. 网络各层运算量如图 20 所示. 各层微指令条数与运算吞吐率如图 21 所示. 由于 ResNet 的残差结构，在评估时增大了片上缓存以存储中间结果，故只进行了单核测试. 指令的执行性能上与 YOLOV3Tiny 模型基本一致，在 3×3 卷积上有很高的吞吐率，峰值吞吐率达到了 199.8GOP/s，十分接近计算资源

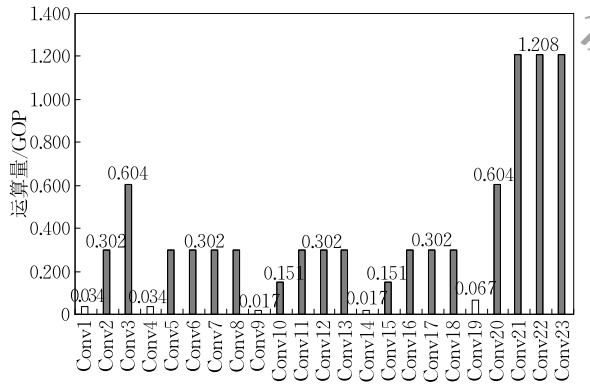


图 20 ResNet18 模型卷积层运算量示意图

(DSP 使用量:517)的理论峰值吞吐率 206.8GOP/s.

上述实验说明，通过微指令架构的数据流调度优化，本文提出的加速器在运算量较大的多数层达到了接近理论性能极限的峰值吞吐率. 但在运算量小但通道较深的层，由于指令序列过长与数据复用较低导致了一定的性能损失.

5.3 对比实验与分析

5.3.1 能效对比评估

星载系统对能耗有很强的敏感性，故本节对比了本文的 FPGA 加速器与其它通用硬件平台加速方法在能效方面的表现. 本节测试了相同 CNN 模型(YOLOV3Tiny)在不同平台上的加速能效，结果如表 5 所示. 测试平台分别为 Intel Core i7 8700 CPU、NVIDIA GTX1080 GPU、Xilinx VX690T FPGA. 软件框架为 DarkNet，在编译选项不选择 GPU 时，测试 266 张遥感数据集图像耗时 67 s，开启 GPU 时为 3 s. 故 CPU 与 GPU 帧率分别为 3.9 帧/s 与 88.7 帧/s. 而本文加速器的帧率分别为 51 帧/s 与 102 帧/s. 相较 GPU，本文在能效方面有约 14(单核)与 19.4(双核)倍的提升.

表 5 能效对比表

	工艺/nm	精度	平均每帧处理时间/ms	帧率/(帧/s)	功耗/W	能效/(帧/s/W)
CPU	14	Float	256.0	3.9	65.00	0.06
GPU	16	Float	11.0	88.7	180.00	0.49
本文(单核)	**28**	**16 bits fixed**	**19.6**	**51.0**	**7.39**	**6.90**
本文(双核)	**28**	**16 bits fixed**	**9.8**	**102.0**	**10.68**	**9.55**

5.3.2 微指令编码效率对比评估

作为一个 Single Engine 加速器，指令编码效率对汇编器复杂程度、指令执行效率都有较大影响. 表 6 对比了本文与文献[11]的指令编码效率. 为了统一对比，本文使用指令序列密度作为编码效率评价标准，即单位指令空间(MB)所包含的运算量. 相较文献[11]，本文 YOLOV3Tiny 模型的指令密度高出其 YOLOV1 模型 5.67 个 GOP. 这意味着本文的指令文件每 1MB 所驱动的操作数比文献[11]多 5.67 个. 这样的增益来源于微汇编器的计算图优化，将大规模数据流按照硬件结构设计，合理的划分为多个小数据流. 而单条微指令驱动了更多的操作，故使得本文的指令序列条数较少，进而指令序列有更高的密度. 但本文的单条微指令字过长，数据搬运

指令与运算指令均有较大的压缩编码空间，这亦是后续研究工作的方向.

表 6　指令编码效率对比

	模型	模型规模/GOP	指令数	指令大小/MB	密度/(GOP/MB)
TRETS[11]	YOLOV1	30.60	238132	3.810	8.03
TRETS[11]	VGG-A	16.80	101133	1.620	10.37
本文	**YOLO3T**	**2.33**	**1403**	**0.171**	**13.70**
本文	**ResNet18**	**8.62**	**1371**	**0.167**	**51.61**

5.3.3　计算效率对比评估

之前类似的诸多工作在不同的 FPGA 平台上实现多样的 CNN 模型加速. 但各加速器在模型选取、FPGA 选取与资源消耗等方面不尽相同. 故本文未选取直接性能指标来评估加速器的优劣，而是采用 *ME* 作为评价指标与其他相似工作进行比较. *ME* 是硬件效率与性能的重要评价标准[24,29]，计算方法如式(8).

$$ME=RMT/TTR \tag{8}$$

其中 *RMT* 为实际峰值吞吐率，*TTR* 为理论吞吐率上界(Theoretical Roof Throughput，TTR). *ME* 表现了设计方法对计算资源的利用效率. 优秀的设计 *ME* 将趋向 100%. 在 FPGA 平台 MAC 单元主要由 DSP 硬核资源构成，而某一设计的 DSP 使用量与其工作的时钟频率(f)决定了其 *TTR*，计算方法如式(9)，单位为 GOP/s.

$$TTR=DSP_{num}\times 2\times f \tag{9}$$

表 7 对比了本文与之前同样使用 VX690T FPGA 芯片的加速器计算效率. 相比文献[20]，本文在使用了其 72.0%的计算资源获得了 1.7 倍的吞吐率提升，*ME* 也有 14.2%的提升. 相较文献[25]、文献[19]，本文 *ME* 分别有 14.2%与 6.9%的提升. 换言之，文献[25]使用本文 2.7 倍的计算资源却仅获得了 1.7 倍的吞吐率提升. 同样，文献[19]使用本文 2.1 倍的计算资源得到了 1.5 倍的性能增益. 而本文之所以获得高 *ME* 是因为本文依据 DDR 最大带宽选择卷积循环分块参数，并以循环分块为核心规划数据流，逻辑运算器同样根据微指令进行设计. 这样的协同设计使得卷积大规模的乘累加运算合理的映射到了计算阵列上，有效利用了计算资源.

表 7　计算资源效率对比表

	平台	时钟频率/MHz	量化精度/bits	DSP 使用量	*TTR*/(GOP/s)	*RMT*/(GOP/s)	*ME*/%
TCAD[25]	VX690T	150	16	2833	849.9	**636.0**	74.8
FPT[20]	VX690T	100	16	1436	287.2	222.1	77.3
FPL[19]	VX690T	156	16	2144	668.9	565.9	84.6
本文	VX690T	100/200	16	1034	413.6	378.6	**91.5**

6　结　论

本文提出了一种面向 CNN 的微指令编码格式，并设计了一个微指令序列调度数据流的 RTL 级 CNN FPGA 加速器. 通过微汇编器的图级别优化与微控制器的数据流调度优化，加速器在保证灵活性的同时提升了运算效率. 实验证明加速器在星上常见的遥感目标检测任务上展现了优异性能. 相较于 CPU 与 GPU，本文的加速器具有更高的能效. 相较于其他 FPGA 加速器，本文的加速器在计算资源利用效率方面更具优势.

参　考　文　献

[1] Lecun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553): 436-444

[2] Krizhevsky A, et al. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 2017, 60(6): 84-90

[3] Szegedy C, et al. Going deeper with convolutions//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Boston, USA, 2015: 1-9

[4] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit//Proceedings of the Annual International Symposium on Computer Architecture. Toronto, Canada, 2017: 1-12

[5] Chen Y, Luo T, Liu S, et al. DaDianNao: A machine-learning supercomputer//Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture. Cambridge, UK, 2014: 609-622

[6] Chen Y H, Yang T J, Emer J, et al. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2019, 9(2): 292-308

[7] Gu Yi-Kun, Ni Feng-Lei, Liu Hong. Fault-tolerance design of Xilinx FPGA with self-hosting configuration management. Journal of Astronautics, 2012, 33(10): 1519-1527(in Chinese)
(顾义坤，倪风雷，刘宏. Xilinx FPGA 自主配置管理容错设计研究. 宇航学报，2012，33(10)：1519-1527)

[8] Wang Chao, Wang Teng, Ma Xiang, Zhou Xue-Hai. Research progress on FPGA-based machine learning hardware acceleration. Chinese Journal of Computers, 2020, 43(6): 1161-1182(in Chinese)

(王超，王腾，马翔，周学海. 基于 FPGA 的机器学习硬件加速研究进展. 计算机学报，2020，43(6)：1161-1182)

[9] Wu Yan-Xia，Liang Kai，Liu Ying，Cui Hui-Min. The progress and trends of FPGA-based accelerators in deep learning. Chinese Journal of Computers，2019，42(11)：2461-2480(in Chinese)
(吴艳霞，梁楷，刘颖，崔慧敏. 深度学习 FPGA 加速器的进展与趋势. 计算机学报，2019，42(11)：2461-2480)

[10] Geng T，Wang T，Sanaullah A，et al. A framework for acceleration of CNN training on deeply-pipelined FPGA clusters with work and weight load balancing//Proceedings of the International Conference on Field Programmable Logic and Applications. Dublin，Ireland，2018：394-398

[11] Yu J，Ge G，Hu Y，et al. Instruction driven cross-layer CNN accelerator for fast detection on FPGA. ACM Transactions on Reconfigurable Technology and Systems，2018，11(3)：1-23

[12] Ye Pei-Jian，Huang Jiang-Chuan，Sun Ze-Zhou，et al. The process and experience in the development of Chinese lunar probe. SCIENTIA SINICA Technologica，2014，44(6)：543-558(in Chinese)
(叶培建，黄江川，孙泽洲等. 中国月球探测器发展历程和经验初探. 中国科学：技术科学，2014，44(6)：543-558)

[13] Zhang Zhai，Liu Yan，Huang Li-Li. Self-repairing method for BRAM based on cold backup multi-mode redundancy structure. Acta Aeronautica et Astronautica Sinica，2021，42(7)：546-557(in Chinese)
(张砦，刘燕，黄莉莉. 基于冷备份多模冗余结构的 BRAM 自修复方法. 航空学报，2021，42(7)：546-557)

[14] Li Tan，Shen Juan，Chen Sai-Qi，Tang Meng-Hui. Life constraints of the LEO remote sensing satellite platform. Bulletin of Surveying and Mapping，2014，(S1)：40-42(in Chinese)
(李潭，沈娟，陈塞崎，唐梦辉. 低轨遥感卫星平台长寿命制约因素研究. 测绘通报，2014，(S1)：40-42)

[15] He Xiao-Jun，Li Zhu-Qiang，Qin Xiao-Bao，et al. Technological innovation and application achievements of Jilin No. 1 spectral satellite. Satellite Application，2020，(3)：18-26(in Chinese)
(贺小军，李竺强，秦小宝等. 吉林一号光谱卫星技术创新与应用成果. 卫星应用，2020，(3)：18-26)

[16] Wei X，Liu W，Chen L，et al. FPGA-based hybrid-type implementation of quantized neural networks for remote sensing applications. Sensors，2019，19(4)：924

[17] Nong Yuan-Jun，Wang Jun-Jie. Real-time object detection in remote sensing images based on embedded system. Acta Optica Sinica，2021，41(10)：179-186(in Chinese)
(农元君，王俊杰. 基于嵌入式的遥感目标实时检测方法. 光学学报，2021，41(10)：179-186)

[18] Wang Shi-Yu，Zhang Sheng-Bing，Huang Xiao-Ping，Chang Li-Bo. Single-chip multiprocessing architecture for space-borne SAR imaging and intelligent processing. Journal of Northwestern Polytechnical University，2021，39(3)：510-520(in Chinese)
(王时雨，张盛兵，黄小平，常立博. 星载 SAR 成像与智能处理的单片多处理架构. 西北工业大学学报，2021，39(3)：510-520)

[19] Li H，Fan X，Jiao L，et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks//Proceedings of the International Conference on Field Programmable Logic and Applications. Lausanne，Switzerland，2016：1-9

[20] Liu Z，Dou Y，Jiang J，et al. Automatic code generation of convolutional neural networks in FPGA implementation//Proceedings of the International Conference on Field-Programmable Technology. Xi'an，China，2016：61-68

[21] Nguyen D T，Nguyen T N，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

[22] Redmon J，Farhadi A. YOLO9000：Better，faster，stronger//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Honolulu，USA，2017：7263-7271

[23] Farrukh F U D，Zhang C，Jiang Y，et al. Power efficient tiny YOLO CNN using reduced hardware resources based on booth multiplier and WALLACE tree adders. IEEE Open Journal of Circuits and Systems，2020，1：76-87

[24] Gokhale V，Zaidy A，Chang A X M，et al. Snowflake：An efficient hardware accelerator for convolutional neural networks//Proceedings of the IEEE International Symposium on Circuits and Systems. Baltimore，USA，2017：1-4

[25] Zhang C，Sun G，Fang Z，et al. Caffeine：Toward uniformed representation and acceleration for deep convolutional neural networks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems，2018，38(11)：2072-2085

[26] Guo K，Sui L，Qiu J，et al. Angel-Eye：A complete design flow for mapping CNN onto embedded FPGA. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems，2017，37(1)：35-47

[27] Abdelfattah M S，Han D，Bitar A，et al. DLA：Compiler and FPGA overlay for neural network inference acceleration//Proceedings of the International Conference on Field Programmable Logic and Applications. Dublin，Ireland，2018：411-418

[28] He K，Zhang X，Ren S，et al. Deep residual learning for image recognition//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas，USA，2016：770-778

[29] Yu Y，Wu C，Zhao T，et al. OPU：An FPGA-based overlay processor for convolutional neural networks. IEEE Transactions on Very Large Scale Integration Systems，2019，28(1)：35-47

[30] Simonyan K，Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv：1409.1556，2014

[31] 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

[32] Jia Y，Shelhamer E，Donahue J，et al. Caffe：Convolutional architecture for fast feature embedding//Proceedings of the ACM International Conference on Multimedia. Orlando，USA，2014：675-678

[33] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift//Proceedings of the International Conference on Machine Learning. Lille, France, 2015: 448-456

[34] Gysel P, Motamedi M, Ghiasi S. Hardware-oriented approximation of convolutional neural networks. arXiv:1604.03168, 2016

[35] Redmon J, Farhadi A. YOLOv3: An incremental improvement. arXiv:1804.02767, 2018

[36] Wang Y, Wang C, Zhang H, et al. A SAR dataset of ship detection for deep learning under complex backgrounds. Remote Sensing, 2019, 11(7): 765

附录 1.

一条完整的微指令编码示例如下所示：

#数据搬运指令				
地址	微指令码	**mrmovf**:		#数据搬运指令,特征图数据由 DDR 加载至特征缓存
0x000:	0x1	.enable	1	#指令是否执行
0x001:	0x000000000	.srcAddress	0x000000000	#源地址
0x025:	0x100000000	.desAddress	0x100000000	#目的地址
0x049:	0x000100000	.length	1048576	#特征图数据量(字节)64×256×32×16/8(尺寸:64×256×32)
		rrmovf:		#数据搬运指令,特征图数据由特征缓存加载至运算单元
0x06d:	0x1	.enable	1	
0x06e:	0x1	.waitma	1	#是否等待 DDR 数据加载完成
0x06f:	0x100000000	.srcAddress	0x100000000	#源地址
0x093:	0x000100000	.length	1048576	#加载数据量
		mrmovw:		#数据搬运指令,权重数据由 DDR 加载至权重缓存
0x0b7:	0x1	.enable	1	#指令是否执行
0x0b8:	0x000100000	.srcAddress	0x000100000	#源地址
0x0dc:	0x100100000	.desAddress	0x100100000	#目的地址
0x100:	0x000000020	.length	32	#加载数据量(AXI 总线冲突读写次数)1×1×32×32×16/512
		rrmovw:		#数据搬运指令,权重缓存加载权重至运算单元 1 级缓存
0x124:	0x1	.enable	1	#指令是否执行
0x125:	0x1	.waitma	1	#是否等待 DDR 数据加载完成
0x126:	0x100100000	.srcAddress	0x100100000	#源地址
0x14a:	0x000000020	.length	32	
		rimovb:		#数据搬运指令,加载偏执到运算单元
0x170:	0x1	.enable	1	
0x171:	0x00	.address	0x00	#加载偏执的首地址
		immovf:		#数据搬运指令,写回结果
0x179:	0x1	.enable	1	
0x17a:	0x003000000	.address	0x003000000	#写回地址
0x19e:	0x000040000	.length	262144	#写回数据量 32×128×32×16/8(输出尺寸为 32×128×32)
#随路指令				
		psum:		#随路指令,部分和累加
0x16e:	0x0	.enable	0	
0x16f:	0x0	.type	0	#读或写部分和缓存(FIFO)
		quant:		#随路指令,量化(移位实现)
0x1c2:	0x1	.shift	1	#后量化数据移位方向(0 左 1 右)
0x1c3:	0x10	.length	16	#16 位
0x1f9:	0x0	.shiftb	0	#前量化数据移位方向(0 左 1 右)
0x1fb:	0x0	.lengthb	0	#不移动
		relu:		#随路指令,激活(移位实现)
0x1fe:	0x0	.type	0	#0 为激活,1 为不激活
#运算指令				
		pool:		#运算指令,池化
0x1cc:	0x1	.enable	1	
0x1cd:	0x1	.stride	1	#步长
0x1f7:	0x0	.pad	0	#是否填充(步长为 1 时)
		upsample:		#运算指令,上采样
0x1ce:	0x0	.enable	0	
		pad:		#运算指令,填充
0x1cf:	0x0	.enable	0	
0x1d1:	0x0	.type	0	#填充类型
		conv:		#运算指令,卷积
0x1d5:	0x100	.width	256	#输入特征图宽
0x1df:	0x040	.high	64	#输入特征图高
0x1e9:	0x1	.kernel	1	#卷积核(权重)尺寸
0x1ec:	0x1	.stride	1	#步长
#控制指令				
0x1fd:	0x0	**end**:	0	#控制指令,是否为指令序列最后一条指令
0x1fe～0x3ff:	0x0			#预留扩展位

GUO Zi-Bo, Ph. D. candidate. His research interests focus on the design system on chip and deep neural network model compression technology.

LIU Kai, Ph. D., professor, Ph. D. supervisor. His research interests include design embedded system on chip and high-speed image video coding.

HU Hang-Tian, M. S. candidate. His research interests focus on embedded system on chip.

LI Yi-Duo, M. S. candidate. His research interests focus on deep neural network model compression technology.

QU Ze-Xu, M. S., senior engineer. His research interests focus on data transmission and processing on space.

Background

In order to improve the inference speed of CNN model, Academia and industry have proposed many CNN-domain-specific acceleration methods to cope with the various application scenarios. However, how to accelerate CNN stably and efficiently on orbit satellites with limited resources and energy consumption is still a challenging proposition.

Compared with ASIC, FPGA has higher flexibility and faster development iteration speed, which make it very suitable for spaceborne scenarios. At present, the research on parallel architecture of FPGA accelerator is mainly divided into two categories: Stream mode and Single Engine mode. Stream structure has high throughput but consumes more resources. Single Engine structure has only a single operation unit, and the data is driven by the control unit and transmitted repeatedly between the operation units to complete all operations. It achieves the purpose of acceleration through the concurrent operation among all operation units and the parallel operation within each operation unit. Although it cannot achieve the high throughput of Stream mode, the resource consumption will be greatly reduced. The key to the design of Single Engine accelerator is the inter-layer Dataflow Schedule and the parallel structure design of Single operation unit.

For low resource and low power consumption scenarios of in-orbit satellites, this work chooses Single Engine design pattern. A kind of register transfer level CNN accelerator for microinstruction sequence scheduling data stream is designed and implemented on Xilinx XC7VX690T FPGA platform commonly used on the satellite. The interlayer data flow scheduling is optimized by the concurrent parsing and execution of multiple sets of instructions in the micro control unit. A cascade array structure of multiplicative accumulator is used to realize the parallel computation in layer. Experimental results show that the designed power consumption of the accelerator is 10.68 W, and the average accuracy of the remote sensing image data set is 0.9 when it is used as a coprocessor to accelerate the CNN object detection algorithm YOLOV3Tiny, and the detection speed is up to 102 frames/s. Compared with CPU and GPU, the accelerator has higher energy efficiency. Compared with similar FPGA accelerators, it also has an improvement of more than 6.9 percentage points in ME(MAC Efficiency).

This work has been supported in part by the National Natural Science Foundation of China (Grant Nos. 62171342, 61850410523) and the Innovation Fund for Space TT&C Communications (Grant No. 201701B).