Journal of Computer Applications
计算机应用, 2022, 42(S1): 208 - 214
ISSN 1001-9081
CODEN JYIIDU
2022-06-30
http://www.joca.cn

文章编号:1001-9081(2022)S1-0208-07
DOI:10.11772/j.issn.1001-9081.2021081407

基于专用卷积神经网络加速器的编译器设计与实现

焦禹铭[1],吴　凯[1],郭风祥[2],王　昭[3],宋庆增[1*]

(1. 天津工业大学 计算机科学与技术学院,天津 300387;
2. 天津工业大学 电气工程学院,天津 300387;
3. 中国电子科技集团公司 信息科学研究院,北京 100086)

(*通信作者电子邮箱 qingzengsong@tjpu.edu.cn)

摘　要:不同框架深度学习模型部署是人工智能落地的核心,然而模型计算量和参数量过大、编程模型未统一导致了各种新型的专用卷积神经网络(CNN)加速器层出不穷,增加了模型的部署难度。对模型压缩和编译工具链这两个方面进行了改进:在模型压缩方面,提出新的通道剪枝标准,结合了通道的相关性和影响性以及输出通道对应的激活值,在保证精度的同时可以极大地削减卷积神经网络的计算量和参数量;在编译工具链方面,设计了一套自动的端到端优化堆栈,提出了针对基于现场可编程门阵列(FPGA)的深度学习编译器设计方法,并在中间表示中添加了所提出的排序标准的剪枝算法。实验结果表明,所设计的编译器于舰船目标检测的任务中,在通用设备上,保证精度损失不超过1%的情况下取得了1.3倍的加速效果;在专用的CNN加速器上取得了1.6倍的加速效果,在部署中能够有效地针对卷积网络进行加速。

关键词:现场可编程门阵列;模型压缩;深度学习编译器;中间表示;目标检测

中图分类号:TP314　　**文献标志码**:A

Design and implementation of compiler based on special convolutional neural network accelerator

JIAO Yuming[1], WU Kai[1], GUO Fengxiang[2], WANG Zhao[3], SONG Qingzeng[1*]

(1. *School of Computer Science and Technology, Tiangong University, Tianjin* 300387, *China*;
2. *School of Electrical Engineering, Tiangong University, Tianjin*300387, *China*;
3. *Information Science Academy, China Electronics Technology Group Corporation, Beijing* 100086, *China*)

Abstract: The deployment of deep learning models in different frameworks is deemed as the core of the implementation of artificial intelligence algorithms. However, various new-type special Convolutional Neural Network (CNN) accelerators emerge in endlessly caused by the oversize model calculation and parameter quantity and the inconsistent programming model, which has increased the difficulty of model deployment. The improvements has been done from two aspects: model compression and compilation tool chain. In terms of model compression, a new channel pruning standard was proposed, the correlation and influence of the channel were combined, and the activation value corresponding to the output channel was taken into account. It could greatly reduce the calculation and parameter amounts of convolutional neural network while ensuring the accuracy. In terms of compilation tool chain, a set of automatic end-to-end optimization stack was designed, a design method of deep learning complier based on Field Programmable Gate Array (FPGA) was proposed. Besides, the pruning algorithm with proposed sort standard was added to the intermediate representation. The experimental results show that in the task of ship target detection on general equipment, the designed compiler can achieve 1.3 times the acceleration effect while ensuring an accuracy loss of less than 1%. It can achieve 1.6 times the acceleration effect on the special CNN accelerator. In general, it can effectively accelerate the convolutional neural network in deployment.

Key words: Field Programmable Gate Array (FPGA); model compression; deep learning complier; intermediate representation; object detection

0　引言

随着深度学习及卷积神经网络(Convolutional Neural Network, CNN)的发展,研究人员对算法模型的开发迭代效率要求越来越高,为更方便、快速地开发深度学习算法,众多科技巨头公司开发了各种深度学习编程框架,常见的深度学习编程框架包括TensorFlow、PyTorch与Caffe等,因此,开发团队中难免有使用不同框架的开发人员,增大了开发团队的协作

收稿日期:2021-08-05;**修回日期**:2021-11-24;**录用日期**:2021-11-24。

作者简介:焦禹铭(1997—),男,黑龙江牡丹江人,硕士研究生,主要研究方向:深度学习系统、FPGA、高性能计算;　吴凯(1996—),男,天津人,硕士研究生,主要研究方向:高性能计算;　郭风祥(1997—),男,山西运城人,硕士研究生,主要研究方向:FPGA;　王昭(1992—),男,山东德州人,工程师,博士,主要研究方向:高性能计算;　宋庆增(1980—),男,天津人,副教授,博士,主要研究方向:FPGA、高性能计算。

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

难度。深度学习算法的落地场景对硬件的计算能力要求越来越高,于是近年来专用深度学习加速器领域迅速发展。但由于芯片研发周期长,流片成本大,编程模型却未统一,所以就带来了不同框架的新模型对不同专用CNN加速器所构成异构平台的部署问题。

为了应对深度学习模型日益复杂的情况,减轻深度学习芯片开发者向不同硬件移植不同模型算子的负担,学术界、工业界以及开源社区提出了深度学习编译器的概念。如TVM(Tensor Virtual Machine)[1]、MLIR(Multi-Level Intermediate Representation)[2]等,这些编译器可以结合大量的深度学习领域专有方法,例如算子融合、数据布局转换、循环重排等[3],以及传统编译器的成熟优化技术,也可以结合参数自动调优技术,将不同框架实现的模型在不同硬件上快速地部署,实现模型性能提升。

然而,对于专用CNN加速器,其算子支持通常跟不上深度学习模型的迭代速度,导致无法完全支持新型模型,虽然可以将模型算子转化为其他算子使模型运行,但是仍会导致精度效果不如预期。此外,现有的深度学习编译器未考虑在编译优化过程中整合模型压缩技术,且现有的通道剪枝算法的排序标准考虑也并不充分,为模型性能的进一步提升留下了优化空间。为此,亟须针对专用CNN加速器研制一套整合了模型压缩技术的深度学习编译器,提高深度学习模型在专用CNN加速器上的部署效率和运行性能。

因此,本文提出一种针对基于专用CNN加速器的深度学习编译框架海雀编译器(Auk Complier, AukC)。

本文主要工作如下:

1)设计了一套支持专用CNN加速器后端的模型部署端到端的优化堆栈;

2)提出了新型标准的通道剪枝算法,并适配了本文所设计的中间表示,降低了压缩算法在不同深度学习框架实现的工程复杂度;

3)通过子图拆分技术,将加速器不支持的算子/子图进行卸载,转化到其他深度学习编译器的中间表示进行联合优化。

1　相关工作

1.1　MMdnn

MMdnn(Model Management for deep neural network)是一个全面的跨框架解决方案,用于转换、可视化和诊断深度神经网络模型[4]。MMdnn具有一个通用的转化器在框架之间转换深度学习模型,可以使用一个框架训练模型并使用另一个框架部署,也支持模型的再训练,在转化过程中生成一些代码片段以简化以后的再训练或推理,同时,还提供了一款模型可视化工具,方便使用者对模型的直观了解。但是其设计的中间表示主要针对模型的转化,并不适用于在其上进行计算图的优化。

1.2　模型压缩

1.2.1　通道剪枝

剪枝分为结构化剪枝和非结构化剪枝,通常非结构化剪枝会破坏数据的局部性,而结构化剪枝在落地场景更常用,且更适用于未支持稀疏计算的CNN加速器。

在结构化剪枝的研究中,Zhuang等[5]提出的通道剪枝方法对硬件设备加速效果明显,对每一个通道对应的批量归一化层引入缩放因子,与通道的输出相乘。接着联合训练网络权重和这些缩放因子,根据缩放系数对通道进行排序剪枝,并微调剪枝后的网络,在工业界使用广泛,但需要在训练中加入约束,不适用于融入到深度学习编译器;Molchanov等[6]比较了诸多剪枝标准,包括以卷积核权重的大小进行修剪、以激活函数输出为剪枝标准,以互信息为剪枝的标准以及以泰勒展开式的第二项为剪枝标准;He等[7]提出FPGM(Filter Pruning via Geometric Median)算法,以几何形心为剪枝标准,考虑了卷积核范数标准差过小和范数最小值过小的情况,但不易于基于中间表示级别的实现;Prakash等[8]提出以滤波器之间的正交性作为排序标准的RePr(Re-initailizing-Pruning)算法,先将一层中每个通道卷积核张量的平铺,并对构成的矩阵归一化,再与自身的转置相乘得到一个通道数乘通道数维度大小的矩阵,任一行的数值表示其他通道对这一通道的相关性,考虑了特征之间的不必要重合。

但以上方法均未从通道影响力和独立性以及激活值的综合角度来分析排序标准,且现有深度学习编译器也未整合通道剪枝算法。针对以上问题,本文提出了综合考虑当前层对应的通道影响力和独立性以及每个通道对应激活值的通道排序标准,并在编译器中进行了实现。

1.2.2　量化

量化是一种将权重离散化的模型压缩方式,将分布连续的权重通过近似映射而达到离散的目的。深度神经网络的鲁棒性较强,将高精度的参数映射为低精度的参数,也是一种对数据集的正则化方式,可以提高模型泛化能力,但是将参数精度降低太多就会导致模型无法拟合数据集,而降低太少又起不到量化压缩的效果,所以将float32的参数转化为int8是精度和参数量的权衡,且量化后的网络经过硬件的定制化支持可以充分发挥其加速功效,比较适用于现场可编程门阵列(Field Programmable Gate Array, FPGA)、专用集成电路(Application Specific Integrated Circuit, ASIC)以及支持低比特运算指令集的硬件,对于模型的大小以及速度都会有所提升。

浮点数的量化方法为如式(1)~(3)[9]:

$$Q = \mathrm{round}\left(\frac{X}{scale} + ZP\right) \tag{1}$$

$$scale = \frac{X_{\max} - X_{\min}}{255} \tag{2}$$

$$ZP = \frac{-X_{\min}}{scale} \tag{3}$$

其中:Q表示量化之后的整型数;X表示原始浮点数;round是四舍五入函数;$scale$表示量化步长;ZP表示零点,这两个量化因子用来保证区间内的变量都能准确地映射到要量化定点位数的数值区间中;$X_{\min}$和$X_{\max}$分别表示原始浮点数的最小和最大阈值。

常用的量化方式有三种:训练后动态量化、训练后静态量化、感知量化训练。训练后动态量化需要在模型推理中重新计算输出激活值的范围来确定量化的放缩因子和偏移,浪费了推理时间;训练后静态量化使用校验数据集来提前确定激活值范围,不需要在推理中确定激活值范围,由于易用性和高性能,是工业界常用的量化方法;感知量化训练对精度损失最小,在量化中对模型进行训练,从而让网络参数能更好地适应量化带来的信息损失,但是操作繁琐,不利于中间表示级别的实现。

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

1.3 YOLO

YOLO(You Only Look Once)[10]是一种端到端的目标检测算法,是在面向目标检测任务中,边缘计算设备上最常用的检测算法。图 1 为 YOLO 的算法流程,首先对特征图预处理,再传入躯干卷积网络进行特征提取,再经过大小为 1×1 卷积核的 YOLO 头部层进行通道降维得到所需维度的特征图,最后对提取后的特征图进行解码和非极大值抑制得到目标检测框。

图 1 YOLO 算法流程

1.4 基于 FPGA 的神经网络加速器

图 2 为本文所部署的 FPGA 加速器软硬协同框架[11],将其作为编译器验证的原型机。

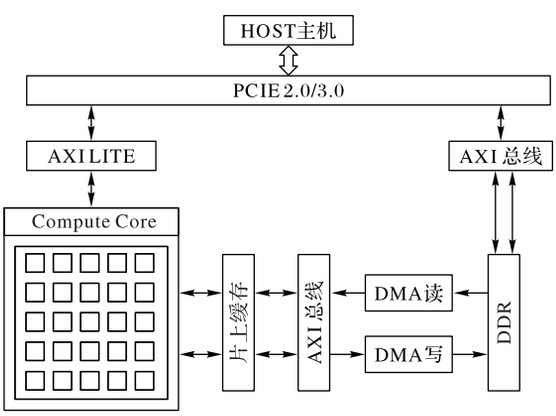

图 2 FPGA 加速器软硬协同加速框架

软件(HOST 端)负责计算任务的调度和控制,通过外部设备互联扩展(Peripheral Component Interconnect express, PCI-e)作为数据接口向硬件发送调度指令,硬件负责算子的计算。由于 FPGA 板卡的存储资源和计算资源限制,所以上位机将模型中一层所需的权重、参数以及图片通过 PCI-e 接口线存入 FPGA 上的双倍速率同步动态随机存储器(Double Data Rate SDRAM, DDR SDRAM)中,运算结束后,结果再通过高级可扩展接口(Advanced eXtensible Interface, AXI)总线写回 DDR 以进行下一层算子的计算。

图 3 为图 2 中的计算核心单元的内部架构图,通过通路选择模块(AXI Stream Switch)进行算子的选择,是可扩展处理单元(Processing Element, PE)并可重构的灵活硬件架构,基本可以实现现有的轻量化神经网络。

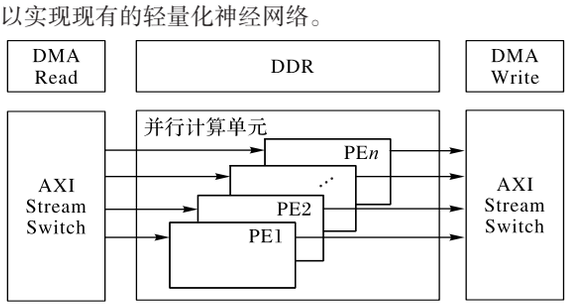

图 3 核心计算单元

2 设计流程及整体框架

AukC 是一个端到端的优化堆栈,架构如图 4 所示,编译器的前端输入为 TensorFlow、PyTorch、Caffe、MXNet 等深度学习框架的模型文件。首先经过解析模块,使用 MMdnn 模型解析器转化为 MMdnn 的 IR(Intermediate Representation),再通过计算图构造模块,将其转化为自定义的 AIR(Auk IR),再将 AIR 输入到计算图优化模块,对其进行剪枝、量化、算子融合、公共子表达式消除等优化,并将部分子图标注为 Relay,最后将子图进行指令选择,并将标注为 Relay 的子图卸载到 TVM,最终生成得到可执行代码。

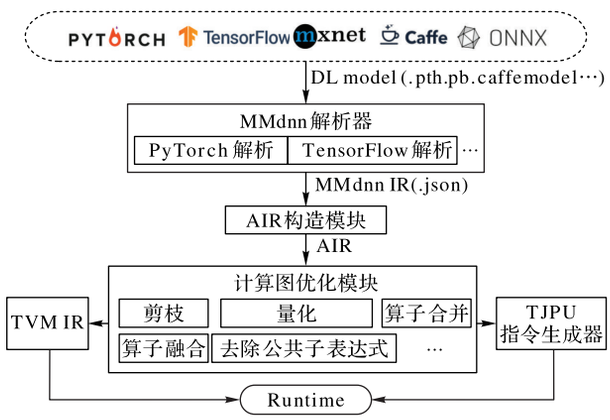

图 4 AukC 编译架构

2.1 AIR

MMdnn 的中间表示在将各种深度学习框架中的模型表示为统一的通用计算图方面起着重要作用[4],但是其设计目标是转换模型,所以简化了 MMdnn IR 的设计,将重心放在了图形语法上,不利于基于其上进行计算图的优化,所以要先将 MMdnn IR 转化为 AIR 计算图,以便在 AIR 上进行优化。

AIR 是对接深度学习模型和后端多个硬件的桥梁,根据硬件特征和算子定义好 IR 的抽象尤为重要,本文 AIR 设计主要针对后续模型压缩优化、模型融合优化、子图拆分、算子映射和对接到其他编译器 IR 为设计目标。

AIR 主要以类似图的邻接表存储方法存储,以便于遍历,顶点对应着不同的 tensor 操作,边代表着数据的依赖关系,对于顶点的结构,包含 op_name、id、def、use 以及 attr。其中:op_name 为顶点的算子类型,如卷积算子、批归一化算子等;id 为不同节点的唯一 id 号;def 为当前节点所需输入张量对应的产生节点;use 为当前节点数据输出到的节点 id;attr 中会根据不同的算子类型,有不同的属性,比如 conv 节点的属性包括通用属性如输入输出尺寸、步长等,以及编译器优化所需的标志如类别、剪枝标志、融合标志等。其中类别为算子的操作类型,主要分两种类型,计算算子及维度转化算子,计算算子包括逐点计算和累加计算,计算算子主要包括卷积、矩阵乘法等,维度转化包括上下采样、拼接等算子。

而每个节点对应的权重保存在计算图的哈希表中 Map〈id, weights〉,其中键为顶点对应唯一 id 号,值为节点所需的权重数据。

2.2 基于 AIR 的优化方法

基于 AIR 的优化,将模型所对应的计算图分配到专用加速器和通用处理器上,目的是增加硬件资源的利用率,使模型更高效地在异构平台上运行。

2.2.1 改进的通道剪枝算法

因为对硬件加速明显的是结构剪枝,尤其是通道剪枝,所

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

以本编译器采用通道剪枝方法,对于通道剪枝有三个重要因素,分别是通道影响性、独立性以及通道对应的激活值。在剪枝中不仅要考虑当前层的一个通道的所有卷积核,也要考虑与之相关的这层删减后,输出特征图通道数的减少所导致的输出特征图通道的缩减,所以本文提出式(4)中以通道层次的正交性和L2范数以及激活层次的输出特征通道L1范数作为通道排序标准。

$$S_i = \omega(\lambda \operatorname{rank}(\boldsymbol{\gamma}) + (1-\lambda)\operatorname{rank}(\|\boldsymbol{K}\|_2)) + (1-\omega)\operatorname{rank}(\|\boldsymbol{A}\|_1) \tag{4}$$

其中:S_i表示第i个卷积核通道的评分;ω和λ表示可设定的系数,范围为0~1;$\boldsymbol{\gamma}$表示通道间的正交性;$\|\boldsymbol{K}\|_2$表示卷积核的L2范数;$\|\boldsymbol{A}\|_1$表示激活值的L1范数;rank表示根据当前因子的通道重要性排名。

首先计算当前层通道之间的正交性,先将一层中每个通道卷积核张量平铺,并对构成的矩阵归一化,再将归一化的矩阵与矩阵的转置相乘,得到一个通道数乘通道数维度大小的矩阵,矩阵中每行的大小代表着其他通道对这一行对应通道的相关性大小,这一因子充分考虑了特征之间的不必要重合;再计算当前层每个通道的L2范数,进行重要性评估;最后计算每个卷积核通道对应输出的激活值的L1范数。剪枝算法根据S_i的值对通道进行排序,将重要性排名低的卷积核通道剪掉。

对于AIR,引入了一个简单的解释执行器,实现了其形式化可执行代码,但并不适合于部署,因为该执行器是通过遍历AIR来执行程序,每次要运行一个算子,就必须遍历每个子节点,且由于节点之间依赖性较强,会不可避免地产生巨大开销,同时也不支持对于专用加速器的调用。遍历中通过op_name可以寻找到每个算子对应的验证单元,用来进行剪枝激活值的确定等,即AIR都可以转化为函数调用的实现。

在遍历图中,将prune flag属性为true的顶点id放入可剪枝节点的列表进行存储,如use列表中有不可剪枝层需将这一id从列表中移除,并对列表中已有id的对应节点进行通道重要性评估。

对于剪枝后的微调,实现了一个反计算图生成器,可以将AIR重新映射回MMdnn IR,将MMdnn IR转化为用户所熟悉的深度学习编程框架,训练收敛后,再映射回AIR进行下一个阶段的优化。

2.2.2　算子合并

对于本文所部署的FPGA加速器中,将批标准化(Batch Normalization, BN)算子[12]简化或者融合到卷积算子中,对硬件设计的复杂度及运行速度都有较大的提升。算子简化是将BN的计算拆解为乘法和加法,依次执行,而融合到卷积算子中则是对卷积核权重的修改,在执行过程中,需要对def节点为卷积算子的BN算子,使用融合入卷积的合并方法。

BN层计算输入即为上一层卷积计算的输出,那么就可以将卷积的计算公式代入到BN层的计算公式,经过公式的推导即可得到新的融合卷积运算,如式(5)所示:

$$\boldsymbol{y} = \boldsymbol{w}_{\text{new}}\boldsymbol{x} + \boldsymbol{b}_{\text{new}} \tag{5}$$

其中:

$$\boldsymbol{w}_{\text{new}} = \frac{\gamma \boldsymbol{w}}{\sqrt{\sigma^2 + \varepsilon}} \tag{6}$$

$$\boldsymbol{b}_{\text{new}} = \frac{\boldsymbol{b} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \times \gamma + \beta \tag{7}$$

$\boldsymbol{y}$表示输出特征图;$\boldsymbol{x}$表示输入特征图;$\boldsymbol{w}_{\text{new}}$和$\boldsymbol{b}_{\text{new}}$表示新的卷积权重;$\boldsymbol{w}$和$\boldsymbol{b}$表示原始卷积权重;$\mu$表示小批量训练数据的平均值;$\sigma$表示小批量训练数据的方差;$\gamma$和$\beta$表示BN层的训练得到的放缩转移参数;$\varepsilon$表示一个极小的常数。将原卷积运算经过权重(式(6))和偏置(式(7))的修改,便可得到新的融合了BN的卷积(式(5)),减少了内存的搬运及BN算子的计算量。

2.2.3　基于计算图的量化方法

本编译器量化采用训练后静态量化的方式,首先确定需要量化的卷积算子,并对计算图插入伪量化节点,然后生成可执行的指令以及调度代码,在校验集合中运行32位浮点数推理,对于激活层采用KL散度(Kullback-Leibler Divergence, KLD)的量化方式,以使得量化后的数据分布和原始数据分布最接近,并求出式(1)中的*scale*和*ZP*值,并将节点类型转化为量化计算节点,用于FPGA的8位定点数运算。

2.2.4　算子融合

对于神经网络加速器来说,可以添加一个片上存储器来提高数据传输性能,将数据从DDR外部存储器传到片上缓存,进行计算后再从片上流回片外部存储器。但是对于大多数神经网络加速器来说,内存带宽的短缺会使得片上计算资源利用不充分,造成数据传输延迟上升,而在总执行时间中,大量时间也会用在数据从片外到片上,再从片上到片外的重复读写中。所以在本文所设计的编译器中,也根据当前研究[13-14]以及专用CNN加速器的特性添加了算子融合的技术,以更好地利用CNN加速器的计算资源。

如图5所示,算子融合方法包含:计算图的拆分,将模型拆解为多个子图;对子图进行遍历,确定融合起点与待融合节点;输出特征块拆分,拆为多个子块;根据子块数据访存量,判断是否将当前算子与下一算子融合;判断当前子图队列是否为空,为空则选择其他子图队列,非空则回到确定节点;所有子图队列为空则结束算子融合步骤。

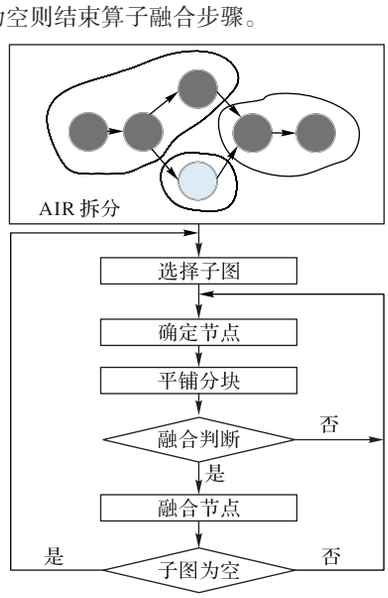

图5　算子融合流程

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

具体步骤如下：

1)子图拆分。

首先对整个AIR图进行拆解，目的是确定数据依赖关系以及异构平台的卸载。具体步骤如下：复制AIR图，以便在新的图上进行修改；创建算子列表，取任一个无数据依赖的算子节点，将id添加到列表中，并删除这一节点use列表中对应每个节点的def；重复选取无数据依赖的节点，直至计算图的每个节点use、def为空；根据每个节点attr中的类别，进行子图的拆分；将不同子图放到子图队列中。

2)确定节点。

通过获取单元，在某一子图队列中，得到节点id，作为起始节点，队列的下一个节点作为待融合节点。

3)输出拆分。

获取融合节点后，需对输出进行拆分。拆分单元的拆分参数会根据对应可用存储量、算子访存量以及计算量进行初始化，将待融合节点所输出的数据进行拆分，可拆成不同维度，每块拆分大小不全相等。

4)融合判断。

在每融合一个核函数时，需要将输入数据块、中间数据块与片上存储资源进行比较，即在图6中，T2为根据输出所拆分的一个输出块，当满足输入块T0+中间块T1<片上存储时，才判断为可融合，否则需要降低输出块T2的尺寸大小，也就是要减小本节所提到的拆分单元的拆分参数，由于算子是固定的，那么反向推导即可得知会减小T0和T1的大小，若不可减小拆分尺寸，需跳过该节点，重新在子图节点列表中进行节点的确定。

算子融合完成后会将算子的输出放在片上存储，以便下一被融合算子的读取，也就是减少了中间数据的写入和读取，因此节省了大量访存时间。

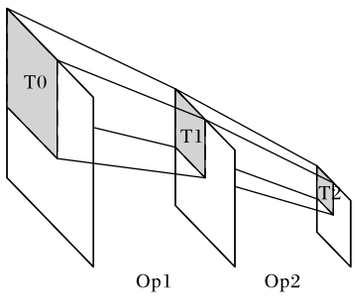

图6 拆分块所对应的感受野

2.2.5 *其他流程*

在遍历计算图中，还有一系列的其他编译器常见pass，如在遍历计算图节点时中遇到常量计算，将常量计算折叠直接计算为常数来替代表示式；在遇到非线性激活函数时替换为近似计算算子，取得加速效果；在遇到硬件不支持算子，通过2.2.4节中的子图拆分技术，标注部署平台为CPU执行后，将子图转化为TVM中的Relay IR，再联合TVM进行编译优化，得到可在CPU端执行的优化代码。

2.2.6 *后端*

编译器后端为将优化后的计算图，进行专用硬件指令生成的过程，并基于TVM将不同计算子图映射到异构平台上。

1)硬件指令。

FPGA加速器所采用的指令集为精简指令集，主要包含参数配置指令以及计算配置指令[15]。上位机通过PCI-e接口将指令发送到控制单元，由于计算资源的有限，上位机调度的单元为一个算子/子图，以降低资源占用，提高网络模型配置的通用性。由于硬件中指令的控制模块不进行复杂控制，计算流程取决于上位机的调度，在启动和结束后都要有上位机信号的交互，所以生成指令时要考虑到生成启动信号和读取中断结束信号。图7为常用卷积算子的指令配置，包含Control信号，决定是参数配置状态还是计算状态，Switch寄存器配置算子的选择，以及Reg0-7的算子通用寄存器，配置各个算子所需参数。

指令	Control_REG	Switch_REG	Reg0	Reg1	Reg2	…	Reg7
地址偏移	0×00	0×04	0×08	0×0C	0×10	…	0×24
操作码	IDLE COMPUTE PARAM REST	3×3 PW DW …	CHANNEL IN CHANNEL OUT KENNEL SIZE …	Control_REG	Control_REG	Control_REG	Control_REG

图7 指令集配置

从计算图转化为指令，先遍历AIR中的节点，根据节点的算子类型生成Switch寄存器的值，根据节点所需位置的大小，设置权重数据传入DDR的DMA读写地址，对于卷积、深度可分离卷积[16]等算子会根据卷积节点属性，针对寄存器生成卷积核大小、通道数以及*scale*和移位大小等指令。指令以32位宽为单位，依次写入图中寄存器。在每个算子的开始还会生成复位信号以及参数配置开始信号，在读取到算子计算完成信号后，会继续下一层算子的计算调度。

2)上位机调度代码。

上位机中，除了生成向FPGA的驱动接口发送的指令调度代码外，还需要将TVM针对不支持算子生成的后端代码进行添加，需要在TVM的后端代码前添加从FPGA的AXI接口读数据以及在TVM后端代码后将处理后的数据发送回FPGA。

3)权重生成。

本文所部署的FPGA加速器中，对于特定的int8卷积算子的支持，不可以采用原始模型float32的数据类型以及排布方式，需要根据计算图中量化后生成的*scale*和*ZP*根据式(1)对权重进行8-bit量化压缩，并根据硬件计算的分块大小以及子图拆分大小进行重新布局，本文所部署的加速器3×3卷积算子后端采用8个输入通道和8个输出通道同时计算的方式，1×1卷积由于常在网络的末尾层，所以采用32个输入通道和8个输出通道同时计算的方式，同时也需要在剪枝时加入8的倍数通道的约束进行规范的剪枝。

4)联合优化。

本文设计的编译器选择以TVM的Relay IR作为转化目

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

标,设计了Relay生成器。具体而言,需要遍历AIR,并根据节点信息,找到需要卸载的节点,通过预设的哈希对应,转化为Relay表示(如遇到不规则算子,需要在TVM端进行算子注册[1],并将其添加到预设的哈希对应表),并在原卸载节点处插入内存拷贝节点,以根据数据维度来进行异构设备的数据传输。通过编译器之间的联合优化,以提高AukC的扩展性以及在异构平台的兼容性,以卸载的方式解决CNN加速器不支持算子的问题。

3　实验与结果分析

本文所设计的编译器目的在于多深度学习编程框架的支持、模型部署的加速以及对于专用CNN加速器的支持,固以具有代表性的基于FPGA的CNN加速器作为实验部署的后端。

3.1　实验环境

本文实验过程需要两台服务器,分别为板卡服务器和调试服务器。板卡服务器具体配置为CPU Inter Core i9-9900K,系统Ubuntu 18.04.5;调试服务器使用CPU Inter Core i7-9750H,GPU为NVIDIA GeForce GTX 1660 Ti。实验采用的FPGA板卡为Xilinx Virtex-7 690T,本文设计的深度学习编译器在调试服务器进行编译,并采用图2中的软硬协同框架进行推理。

3.2　多框架模型自动化支持

以TensorFlow、PyTorch、Caffe2、MXNet等MMdnn支持转化的深度学习框架构建模型,因为本测试只测试对于多个深度学习框架的支持,所以将模型搭建为简易的VGG(Visual Geometry Group)16[17]分类网络,以PyTorch训练的模型为基准,将其权重转化为NumPy格式作为中间媒介,并将NumPy格式权重加载到搭好的其他深度学习框架模型中,最后以猫的图片作为测试用例,将图片输入到各个框架模型转化成AIR的Python函数表示,进行比对,得到相同结果。

3.3　模型加速效果

模型加速评估选取目标检测中的YOLOv4[18]算法为例,对合成孔径雷达(Synthetic Aperture Radar, SAR)船舶图像进行检测,由于本实验仅测量加速性能,所以未追求模型精度的极致。模型的Backbone采用类似VGG堆叠,如需精度的提升,此网络也适用于RepVGG(Re-parameterization VGG)[19]的结构重参数化方法进行训练转化。

本文实验所用模型的参数如表1所示,采用硬件所支持的算子搭建成的网络,Backbone采用CBR(卷积核大小为3×3、Padding为1的卷积算子,BN算子以及ReLU算子)堆叠而成,两个YOLO头部采用卷积核大小为1×1、padding为0的卷积算子,其中:Input表示输入图片大小;Op表示算子类型;Filters表示当前层通道数;Stride表示步长数,"√"为2,"—"为1;Head表示是否输出YOLO头部层;Output表示当前层输出图片大小。

表2为表1中的模型在GPU上剪枝效果的评估,实验中选取交并比(Intersection over Union, IoU)为0.5,通道排序标准式(4)中ω取值0.8,以更关注当前层通道,λ取值0.5,以均衡独立性与重要性的值,剪枝率选取为50%。其中PreTrain表示是否需要稀疏化预训练,方法中Baseline表示未剪枝的测试结果,Pruned[5]表示使用工业界常用通道剪枝算法,Pruned(本文)表示使用本文提出排序标准的剪枝算法。

表1　YOLOv4和CBR网络参数表

Input	Op	Filters	Stride	Head	Output
416×416×1	CBR	8	√	—	208×208×8
208×208×8	CBR	32	—	—	208×208×32
208×208×32	CBR	64	—	—	208×208×64
208×208×64	CBR	64	—	—	208×208×64
208×208×64	CBR	128	√	—	104×104×128
104×104×128	CBR	128	—	—	104×104×128
104×104×128	CBR	128	—	—	104×104×128
104×104×128	CBR	256	√	—	52×52×256
52×52×256	CBR	256	—	—	52×52×256
52×52×256	CBR	256	—	—	52×52×256
52×52×256	CBR	256	—	—	52×52×256
52×52×256	CBR	256	—	—	52×52×256
52×52×256	CBR	256	√	—	26×26×256
26×26×256	CBR	128	—	—	26×26×128
26×26×128	Conv,1×1	25	—	√	26×26×25
26×26×128	CBR	128	—	—	26×26×128
26×26×128	CBR	128	√	—	13×13×128
13×13×128	CBR	128	—	—	13×13×128
13×13×128	Conv,1×1	25	—	√	13×13×25

表2　剪枝加速效果

算法	PreTrain	F1-Score	速率/(frame·s^{-1})
Baseline	—	92.5	104.8
Pruned[5]	√	91.1	161.3
Pruned(本文)	×	91.8	168.8

从表2中可以得出,本文所提出的剪枝算法对比传统的通道剪枝算法,省去了稀疏化训练的步骤,增加了模型的适用范围,更适用于面向多框架的深度学习编译器,精度和速度都超过了传统的通道剪枝算法。

表3中,Baseline中的CPU和GPU为直接使用PyTorch部署的结果,且未使用AukC和TVM的图级优化,FPGA为直接使用指令生成器部署的方案,未经过AukC和TVM对计算图进行的调优;Baseline-pruned使用工业界最常用的通道剪枝算法[5]后经过PyTorch的部署,FPGA也是经过剪枝后的模型输入到指令生成器中;AukC为使用本文所提出的图剪枝率50%情况下的部署,且是未使用TVM联合调优的方案;方法TVM是作为对比,在远程服务器上进行20 h的auto schedule调优所得到的结果;AukC+TVM为使用AukC和TVM的部署堆栈,在CPU和GPU端是先经过AukC的调优,再经过TVM的调优所得到的结果。

表3　不同编译方法的速率对比　　单位: frame/s

算法	CPU	GPU	FPGA
Baseline	3.8	104.8	45.2
Baseline-pruned[5]	5.6	161.3	62.6
AukC	—	—	73.8
TVM[1]	4.9	135.5	—
AukC+TVM	7.7	173.4	73.8

表2~3的实验结果表明,本文提出的剪枝算法更适用于专用CNN加速器;在AukC+TVM上增加了对FPGA加速器后端的支持;且在本文实验所采用模型的目标检测应用场景中,精度损失不超过1%的情况下,相较于直接使用基于TVM的性能提升至原来1.3倍;对于专用CNN加速器的部署,相较于直接用指令生成器部署,性能提升至原来1.6倍。

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

4 结语

本文面向卷积神经网络对异构体系结构的支持和部署难题，设计了一种全自动的神经网络编译器（AukC），编译器整合了当下流行的神经网络优化技术，并对通道剪枝算法进行改进，适配了本文所设计的中间表示。实验结果表明，在保持精度损失不超过1%的情况下，通用设备加速效果可以提升至原来的1.3倍，专用CNN加速器可提升至原来的1.6倍，在精度损失要求不高的情况下，可以完成数倍的加速。至此本文设计了一套完整的端到端部署堆栈，完美支持专用CNN加速器（“海雀”神经网络加速器），并可以联合TVM进行结合使用部署到诸如CPU、GPU等通用后端设备，解决了编程模型未统一背景下专用加速器的兼容性问题，还可以将本文所提出的优化堆栈用于其他的专用CNN加速器上进行扩展，仅需要将指令生成器作一定的扩充。

下一步将考虑硬件结合编译器进行自动调优，添加优化中对功耗的约束，以更好地应用到边缘场景。

参考文献（References）

[1] CHEN T, MOREAU T, JIANG Z, et al. TVM: end-to-end optimization stack for deep learning [EB/OL]. [2018-10-05]. https://arxiv.org/pdf/1802.04799v1.pdf.

[2] LATTNER C, AMINI M, BONDHUGULA U, et al. MLIR: a compiler infrastructure for the end of Moore's law [EB/OL]. [2020-03-01]. https://arxiv.org/pdf/2002.11054v2.pdf.

[3] LI M, LIU Y, LIU X, et al. The deep learning compiler: a comprehensive survey [EB/OL]. [2020-08-28]. https://arxiv.org/pdf/2002.03794v4.pdf.

[4] LIU Y, CHEN C, ZHANG R, et al. Enhancing the interoperability between deep learning frameworks by model conversion [C]// ESEC/FSE 2020: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2020: 1320-1330.

[5] ZHUANG L, LI J, SHEN Z, et al. Learning efficient convolutional networks through network slimming [C]// ICCV 2017: Proceedings of the 2017 IEEE International Conference on Computer Vision. Piscataway: IEEE, 2017: 2755-2763.

[6] MOLCHANOV P, TYREE S, KARRAS T, et al. Pruning convolutional neural networks for resource efficient transfer learning [EB/OL]. [2017-06-08]. https://arxiv.org/abs/1611.06440v1.

[7] HE Y, LIU P, WANG Z, et al. Filter pruning via geometric median for deep convolutional neural networks acceleration [C]// CVPR 2019: Preceedings of 2019 IEEE Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE, 2019: 4335-4344.

[8] PRAKASH A, STORER J, FLORENCIO D, et al. RePr: improved training of convolutional filters [EB/OL]. [2019-02-25]. https://arxiv.org/pdf/1811.07275.pdf.

[9] JACOB B, KLIGYS S, BO C, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference [C]// CVPR 2018: Preceedings of 2018 IEEE Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE, 2018: 2704-2713.

[10] REDMON J, DIVVALA S, GIRSHICK R, et al. You only look once: unified, real-time object detection [C]// CVPR 2016: Preceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE, 2016: 779-788.

[11] 李磊，徐国伟，李文婧，等. 基于深度学习的舰船目标检测算法与硬件加速[J]. 计算机应用，2021，41(S1)：162-166.

[12] IOFFE S, SZEGEDY C. Batch normalization: accelerating deep network training by reducing internal covariate shift [C]// ICML 2015: Proceedings of the 32nd International Conference on International Conference on Machine Learning. New York: PMLR.org, 2015: 448-456.

[13] 黄明飞，王海涛，吕春莹. 一种应用于AI异构编译器的自动切分神经网络子图方法：CN111062467A[P]. 2020-04-24.

[14] 北京中科寒武纪科技有限公司. 一种用于神经网络的算子融合方法及其相关产品：CN110490309A[P]. 2019-11-22.

[15] 黄瑞，金光浩，李磊，等. 轻量化神经网络加速器的设计与实现[J]. 计算机工程. 2021，47(9)：185-190.

[16] HOWARD A G, ZHU M, CHEN B, et al. MobileNets: efficient convolutional neural networks for mobile vision applications [EB/OL]. [2017-4-17]. https://arxiv.org/pdf/1704.04861.pdf.

[17] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition [EB/OL]. [2015-04-10]. https://arxiv.org/pdf/1409.1556.pdf.

[18] BOCHKOVSKIY A, WANG C Y, LIAO H. YOLOv4: optimal speed and accuracy of object detection [EB/OL]. [2020-04-23]. https://arxiv.org/pdf/2004.10934.pdf.

[19] DING X, ZHANG X, MA N, et al. RepVGG: making VGG-style ConvNets great again [EB/OL]. [2021-03-29]. https://arxiv.org/pdf/2101.03697.pdf.

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net