基于MLIR的FP8量化模拟与推理内存优化

徐金龙, 桂中华, 李嘉楠, 李颖颖, 韩林

引用本文

徐金龙, 桂中华, 李嘉楠, 李颖颖, 韩林. 基于MLIR的FP8量化模拟与推理内存优化[J]. 计算机科学, 2024, 51(9): 112-120.

XU Jinlong, GUI Zhonghua, LI Jia'nan, LI Yingying, HAN Lin. FP8 Quantization and Inference Memory Optimization Based on MLIR [J]. Computer Science, 2024, 51(9): 112-120.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230900143

基于 MLIR 的 FP8 量化模拟与推理内存优化

徐金龙[1,3]　桂中华[2]　李嘉楠[2]　李颖颖[3]　韩　林[1]
1 郑州大学国家超级计算郑州中心　郑州 450001
2 郑州大学计算机与人工智能学院　郑州 450001
3 信息工程大学四院　郑州 450001
(longkaizh@163.com)

摘　要　随着目标检测模型和语言大模型的迅速发展,网络模型正变得越来越庞大。为了更好地在端侧硬件上进行模型部署,通常采用模型量化技术对模型进行压缩。现有的模型量化策略主要基于 FP16,BF16 和 INT8 等类型实现。其中,8bit 数据类型在降低推理内存占用与部署开销方面最为显著,但 INT8 类型依赖特定的校准算法,未能很好地处理动态范围大、离群点多的模型。FP8 类型能够更好地拟合神经网络中的数据分布,同时具有多种数制,可在表达范围和表达精度上灵活调整。然而,当前 MLIR 系统缺乏对 FP8 类型量化的支持。为此,提出了一种基于 MLIR 系统的 FP8 量化模拟策略,包含 FP8E4M3 和 FP8E5M2 两种数制,通过对网络中的算子进行量化模拟,评估 FP8 两种数制对模型推理精度的影响。同时,针对推理引擎中存在的内存分配冗余问题,提出了一种基于定义使用链的内存复用策略,使得模型推理过程中的内存占用峰值进一步减小。实验选取了典型的 Yolov5s 和 Resnet50 模型进行测试,结果表明相较于现有的 INT8 量化策略,FP8 量化策略能够保持更好的模型精度,同时不依赖特定校准算法,部署更为简便。在模型精度上,测试用例分别达到了 55.5%和 77.8%的准确度,经过内存复用优化,内存占用峰值降低了约 15%～20%。

关键词:模型压缩;深度学习编译器;FP8 量化;MLIR;Yolov5s 模型
中图分类号　TP311

FP8 Quantization and Inference Memory Optimization Based on MLIR

XU Jinlong[1,3],GUI Zhonghua[2],LI Jia'nan[2],LI Yingying[3] and HAN Lin[1]
1 National Supercomputing Center in Zhengzhou,Zhengzhou University,Zhengzhou 450001,China
2 School of Computer and Artificial Intelligence,Zhengzhou University,Zhengzhou 450001,China
3 Fourth School,Information Engineering University,Zhengzhou 450001,China

Abstract　With the development of object detection models and language models,network models are becoming increasingly large. In order to better deploy the model on the end-to-end hardware,model quantization technology is usually used to compress the model. The existing model quantization strategies are mainly implemented based on FP16,BF16,INT8,and other types. Among them,the 8-bit data type is the most significant in reducing inference memory usage and deployment costs,but the INT8 type relies on specific calibration algorithms and fails to handle models with large dynamic ranges and multiple outliers well. The FP8 type can better fit the data distribution in neural networks,and has multiple formats that can be flexibly adjusted in terms of expression range and accuracy. However,the current MLIR lacks support for quantifying the FP8 type. To this end,a FP8 quantization simulation strategy based on MLIR is proposed,which includes two formats:FP8E4M3 and FP8E5M2. By quantifying and simulating the operators in the network,the impact of the two formats on the inference accuracy of the model is evaluated. A memory reuse strategy based on define use chain is proposed to address the issue of memory allocation redundancy in inference engines,further reducing the peak memory usage during the model inference process. Typical Yolov5s and Resnet50 models are selected for testing and verification,and the results show that,compared to the existing INT8 quantization strategy,the FP8 quantization strategy can maintain better model accuracy,and does not rely on specific calibration algorithms,making deployment more convenient. In terms of model accuracy,the test cases achieve an accuracy of 55.5% and 77.8%,respectively. After memory reuse optimization,the peak memory usage is reduced by about 15%～20%.

Keywords　Model compression,Deep learning compiler,FP8 quantification,MLIR,Yolov5s model

到稿日期:2023-09-25　返修日期:2024-01-27
基金项目:2022 年河南省重大科技专项(221100210600)
This work was supported by the Major Science and Technology Projects in Henan Province for 2022(221100210600).
通信作者:韩林(hanlin@zzu.edu.cn)

1 引言

随着深度学习的迅速发展，网络模型变得更加庞大，在计算和存储资源有限的边缘设备上进行模型部署变得越来越困难。模型量化[1]，是模型部署过程中一种通过降低模型数据位宽、减小模型规模及提高模型推理效率的技术。一些深度学习模型对数据的精度敏感度低，进行低精度推理可以节省模型所占用的内存空间，同时保持较高的模型精度。在低精度类型中，FP16 类型已经默认得到大多数硬件厂商的支持，BF16 与 INT8 类型的使用越来越广泛。以 INT8 为例，其值域在区间上均匀分布，依赖量化校准算法对数据进行缩放，量化后数据能在一定程度上近似网络模型中的数据分布。但是，当数据中出现离群点时，其量化效果不佳。GRAPHCORE、英伟达等厂商提出的符合高斯分布的 FP8 格式[2-3]理论上更符合网络模型中的数据分布，但 FP8 数据格式多样，不同网络中数据的值域不同，量化效果也不同，当前 FP8 量化技术仍处于发展阶段。

模型量化分为量化感知训练（Quantization Aware Training，QAT）和训练后量化（Post-Training Quantization，PTQ）。QAT 通过在网络训练中加入量化过程，让网络参数能更好地适应量化带来的信息损失，从而达到更高的网络精度。端侧部署场景中模型训练和部署往往分开进行，二者的计算能力和内存需求也存在较大的差异。PTQ 针对训练好的网络进行量化，不需要大量的数据参与计算，将模型量化与训练过程分离；缺点是量化的精度稍差于 QAT。QAT 更便于部署，使用更加广泛，量化过程需要利用模型中算子的输入、输出的类型信息进行量化推理。Pytorch 等框架[4-5]使用 Python 语言对深度神经网络进行建模，在模型运行时进行类型推导以获取量化的相关信息，因此失去了一些提前优化的机会。为了解决信息的获取以及优化的问题，可以利用编译器的中间表示（Intermediate Representation，IR）更好地为模型量化提供信息，同时还可以分析内存占用情况以进一步优化。

近些年，深度学习领域出现了一批代表性的编译器，如 TVM（Tensor Virtual Machine），MLIR（Multi-Level Intermediate Representation）和 XLA（Accelerated Linear Algebra）等[6-8]。这些编译器具备深度学习领域专有优化方法以及中间表示层级的分析技术，同时也具备传统编译器的优化技术，可以将不同框架编写的深度学习模型在硬件上快速部署。MLIR 编译器具备良好的拓展性，能够快速组建一套满足需要的编译流，以达到定制化编译优化与代码生成的目的。MLIR 编译器从高级 IR 向低级 IR 进行转换时，可以进行类型转换、代码生成，生成量化需要的类型信息，便于量化技术的实现。

由于 INT8 量化依赖校准算法，且数据分布不能很好地近似神经网络权重和激活值，同时当前 MLIR 系统仍缺乏对 FP8 量化的支持，因此本文基于深度学习编译器 TPU-MLIR[9]提出 FP8 量化模拟方案与内存优化策略。本文的贡献如下：

（1）从二进制层面实现了 FP32 与 FP8E4M3 和 FP8E5M2 之间的转换策略，将转换过程中的损失最小化；

（2）基于 TPU-MLIR 实现了一种 FP8 量化模拟策略，借助 MLIR 量化模块获取了较高的精度；

（3）实现了一种内存复用策略，降低了模型的内存占用峰值，以实现更大的网络模型推理。

2 面向 MLIR 的模型量化

通过大量的样本数据训练，深度学习模型能够学习到数据样本的特征。然而，在训练过程中，数据样本中不可避免地存在噪声，这使得训练好的网络对噪声具有一定的容忍度，从而增强了整个网络的鲁棒性。借助于训练后网络的鲁棒性，模型量化技术通过降低网络输入与权重等数据位宽，在带来轻微模型精度损失的情况下，节省了大量的内存资源与模型部署开销。

TPU-MLIR 是一款基于 MLIR 开发的多层级编译系统，用于将模型快速部署到 TPU 硬件单元上。该系统将模型部署分为前、中、后 3 个阶段，能够为模型量化提供丰富的类型信息，同时也为模型优化提供了空间。由于该系统是 MLIR 应用于边缘设备进行推理优化的最新工作，因此，在该系统上实现 FP8 量化能够为边缘设备上的模型部署提供一定的参考。

2.1 模型量化技术

随着深度学习模型与 AI 加速器件的迭代更新，Google 和 IBM 等厂商纷纷采用新的类型，并设计自己的浮点类型，以取得 AI 系统性能和精度的平衡。当前比较主流的量化数据类型包括 FP16，BF16 和 INT8 等。其中，FP16 等浮点格式设计与 FP32 格式类似，只是在阶码位数和小数位数上有所不同。以 FP32 存储格式为例，其包含 1 位符号、8 位指数和 23 位小数。计算式如式(1)所示：

$$f=(-1)^{s}2^{p-b}\left(1+\frac{d_1}{2}+\frac{d_2}{2^2}+\cdots+\frac{d_m}{2^m}\right) \tag{1}$$

其中，s 为符号位，p 为指数位，b 为指数偏移，d 表示小数位。FP32 数据表示范围广泛，精度高，在神经网络训练和推理中获得了很好的效果，但需要较大的内存空间和计算量，不适合边缘设备上的网络部署。通过模型量化将 FP32 模型以较低位宽的数据格式计算，以近似 FP32 模型的计算效果。量化公式如式(2)所示：

$$r=S(q-z) \tag{2}$$

其中，r 为 FP32 类型的真实数据，q 为量化后的低位宽格式数据，z 为零点表示偏移值，S 表示缩放因子。通过选择合适的 S 与 z 数据，可以很好地近似 r 值。

(1)FP16 和 BF16 量化

早在 2008 年，IEEE 浮点运算标准就引入了 FP16 作为存储格式。FP16 具有 1 位符号位、5 位指数位和 10 位小数位。相比 FP32，FP16 在精度和表达范围上有相应的下降的缩小，但在大多数网络模型中可以做到精度无损，同时减少网络模型 50%的内存占用。

FP16 的最大正数约为 65 504。由于部分深度学习模型

在训练中对精度要求低，对表达范围要求更高，因此 Google 团队提出了 BF16 浮点格式[10]，如图 1 所示。该格式的表示范围和 FP32 类型保持一致，但小数位只有 7 位，精度上略低于 FP16。根据量化式(2)，FP16 与 BF16 量化中的 S 一般取为 1.0，z 取为 0。

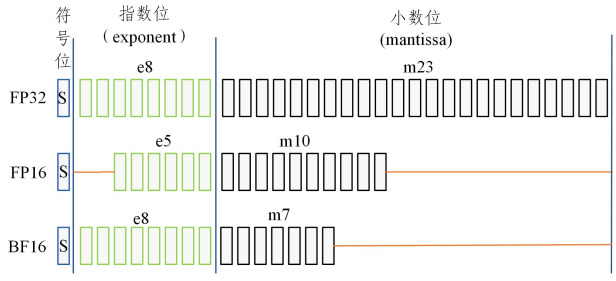

图 1 FP32，FP16 与 BF16 的格式对比

Fig. 1 Format comparison of FP32 and FP16 with BF16

(2)INT8 量化

INT8 量化[11]是 Google 在 TensorFlow Lite 中的量化方案。将 FP32 类型转换为 INT8 类型，相当于对数据以 8bit 整数进行再编码。根据符号位，可以将 FP32 数据映射到[−127，127]或[0，255]区间。INT8 量化的核心在于选取适当的缩放因子(即式(2)中的 S)，常用相对熵(又称 KL 散度)来衡量 INT8 量化数据和原来 FP32 数据分布之间的相似程度。INT8 的量化流程主要分为以下两个步骤：

①通过 KL 散度算法进行程序校准，计算出对称量化需要的缩放因子 S；

②根据量化式(3)进行量化推理，以整数计算近似 FP32 浮点数计算，其中 *round* 为取余操作。

$$q=round\left(\frac{r}{S}+z\right) \tag{3}$$

INT8 量化需要完整的校准程序，其数据在区间上均匀分布，对于一些具备较多离群点的模型效果不佳，选择合适的校准算法可以在大多数网络上获得较高的精度。

2.2 TPU-MLIR 编译系统

TPU-MLIR 系统前端接收训练后的神经网络模型，如开源项目 ONNX(Open Neural Network Exchange)[12]中已经训练好的模型，通过编译系统对模型进行算子融合、模型校准、数据切片、流水并行等优化，将模型高效部署到端侧硬件 TPU 上。该编译流程与常见的代码生成编译器略有不同，TPU-MLIR 更专注编译过程中的前端和中端转换，最终模型的推理依赖于 CPU 或 TPU 端算子库的分发。同时为了便于实现多种优化，TPU-MLIR 在中间表示层次提供了基于 Intel 开源项目 OneDNN(oneAPI Deep Neural Network Library)库[13]的 CPU 推理引擎，方便程序验证。TPU-MLIR 整体工作流程如图 2 所示，主要分为 4 步。

(1)通过前端转换模块从 TensorFlow Lite[13]，Caffe2 或 ONNX 等框架导入已训练的模型到 TPU-MLIR 中。

(2)模型被转换为 TPU-MLIR 中的高层级中间表示 TOP IR，TOP IR 将深度学习模型架构和权重数据信息建模在同一层，并应用图级别的优化遍对模型进行初步优化。

(3)TOP IR 被进一步转换为 TPU IR 表示，该步骤通过 IR 递降实现，为 IR 新增更多硬件相关的属性，进行 TPU 硬件相关的优化，如数据重排、流水并行等。

(4)模型被转换为序列化的 bmodel 文件，使用 TPU 推理执行。

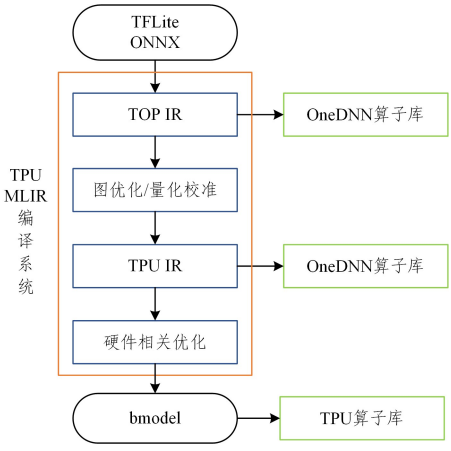

图 2 TPU-MLIR 的整体结构

Fig. 2 Overall structure of TPU-MLIR

3 FP8 量化策略支持

FP8 在节省内存方面与 INT8 类似，但 FP8 具备小数位，具有更高的精度。与 FP32 不同，FP8 并没有 IEEE 规定的指数位和小数位分配方式，也没有标准的指数偏移值限制，具有较强的灵活性。当前 MLIR 系统中的量化模块仅支持 INT8 量化，缺少 FP8 量化类型与对应的缩放因子，因此本文重新扩展了 MLIR 量化系统，以支持 FP8 量化策略。

Sun 等[14]于 2019 年提出的 HBF8，包含 1 位符号位、4 位指数位和 3 位小数位，指数偏移值设置为 11，在 AlexNet，DenseNet121 和 MobileNetV2 中获得了与 FP32 接近的精度；但偏移值过大会带来较小的表达范围，对于一些动态范围比较大的网络，精度下降明显。2022 年英伟达联合 ARM 和 AMD 等发表的 FP8 格式标准[15]包含 E4M3 和 E5M2 两种格式(见图 3)，并给出了 Bert 和 GPT3 模型的实验结果；但其忽略了量化缩放因子对网络的影响，同时缺乏对目标检测模型的量化研究。本文提出的 FP8 量化策略深入研究了缩放因子对两种不同 FP8 数制量化的影响，并对目标检测网络与残差网络进行 FP8 量化支持。

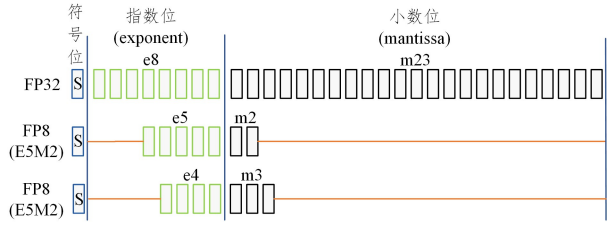

图 3 FP32 与 FP8E4M3 和 E5M2 的格式对比

Fig. 3 Format comparison of FP32 with FP8E4M3 and E5M2

3.1 数值转换规则的构建

FP32 数据对于计算过程中产生的无穷值、零值、NaN、

非规格化等特殊数值有着对应的二进制格式。FP8 数据的二进制编码采用 2022 年英伟达发表的 E4M3 格式和 E5M2 两种格式，这两者的主要区别在于 1 位二进制是分配给指数还是小数，这造成了表示范围和精度上的差异。FP8E4M3 具有 4 位指数和 3 位小数，二进制编码如表 1 所列。

表 1　FP8E4M3 二进制编码规则

Table 1　Binary encoding rules of FP8E4M3

值类别	二进制表示
指数偏置值	7
无穷值	无
NaN	$S.1111.111_2$
零值	$S.0000.000_2$
规格化最大值	$S.1111.110_2=1.75*2^8=448$
规格化最小值	$S.0001.000_2=2^{-6}$
非规格化最大值	$S.0000.111_2=0.875*2^{-6}$
非规格化最小值	$S.0000.001_2=2^{-9}$

表 1 中展示了 E4M3 编码表的详细规则，其中阶码的偏移值为 7，可以表示 NaN 和零值，但无法表示无穷值。由于 NaN 数据仅使用了两种编码进行表示，存在空闲编码，为了充分利用 FP8E4M3 的所有编码，规定当指数位全为 1 而小数位不全为 0 时仍表示正常值，这为 FP8E4M3 拓展出了额外的 7 个数值（256，288，320，352，384，416，448），因此 FP8E4M3 的表示范围为[−448，448]。同时，FP8E4M3 具有非规格化数值表示，指定整数位默认为 0，阶码固定为 −6，最小值为 2^{-9}。

表 2 中展示了 E5M2 编码表的详细规则，FP8E5M2 与 IEEE754 标准更加接近，其中指数的偏移值为 15，可以表示无穷、零值和 NaN 数据。同时，FP8E5M2 能够表示非规格化数值，即指定整数位默认为 0，阶码固定为 −14，最小值为 2^{-16}。与 FP8E4M3 相比，FP8E5M2 具有更大的表达范围以及更强的特殊值表示能力，但 FP8E5M2 由于仅具有 2 位小数位，其数值分布的离散程度比 FP8E4M3 更加明显，因此在表达精度上低于 FP8E5M2。

表 2　FP8E5M2 二进制编码规则

Table 2　Binary encoding rules of FP8E5M2

值类别	二进制表示
指数偏置值	15
无穷值	$S.11111.00_2$
NaN	$S.11111.\{01,10,11\}_2$
零值	$S.00000.00_2$
规格化最大值	$S.11110.112=1.75*215=57344$
规格化最小值	$S.00001.002=2^{-14}$
非规格化最大值	$S.00000.112=0.75*2^{-14}$
非规格化最小值	$S.00000.012=2^{-16}$
非规格化最小值	$S.00000.012=2^{-16}$

为了将 FP32 中的所有数值表示转换为 FP8 中两种格式对应的编码表示，需要对 FP32 数据编码作分类处理，主要分为以下 3 类：

(1)FP32 数据超出 FP8 表示的绝对最大值。

(2)FP32 数据小于 FP8 表示的接近 0 的最小值。

(3)FP32 数据处于 FP8 表示范围之中。

从数据范围上区分比较简便，但在数值的转换中需要不断地进行值域判断，效率较低。为了使数值转换更加高效，从 FP32 格式中的指数位即阶码进行区分。FP32 具有 8 位阶码，表示范围为[0，255]；指数偏移为 127，指数的真实表示范围为[−127，128]。以 FP32 转换为 FP8E4M3 为例，FP8E4M3 具有 4 位阶码，表示范围为[0，15]；指数偏移为 7，指数的真实表示范围为[−6，7]。因此，二者的转换可以分为以下 4 类：

(1)FP32 真实指数大于 FP8 真实指数的最大值。

(2)FP32 真实指数小于 FP8 真实指数的最小值。

(3)FP32 真实指数处于 FP8 真实指数表示区间。

(4)FP32 中特殊值（如 NaN、无穷、零值）转换到 FP8 表示。

对于 FP32 中真实指数小于 FP8 最小指数的情况，需要进一步分类。FP8 中非规格化数值可以用于近似表示 FP32 中较小指数的规格化数值，对于小于 FP8 非规格化数值的 FP32 数据，默认使用 FP8 中的 0 表示，FP32 到 FP8 的映射区间如图 4 所示。

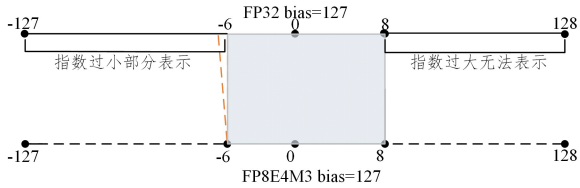

图 4　FP32 到 FP8 的映射过程

Fig. 4　Mapping process from FP32 to FP8

将 FP32 数值转换为 FP8 时，FP8 的指数和小数范围都更加小，因此会不可避免地存在数据损失。当使用 FP8 转换回 FP32 时，FP8 中的特殊值可以全部被 FP32 表示。FP8 中指数位于 −6 和 8 之间的规格化浮点值，也可以被 FP32 完整表示，因此转换过程只需要调整指数间的差值。以 FP8E4M3 为例，根据式(2)可计算出二者之间的指数差值，FP8E5M2 类型只需要将公式中的偏移值 7 更改为 15 即可。

$$E_{FP32}=E_{FP8}-7+127 \tag{4}$$

对于 FP8 的非规格化数值，FP32 同样可以完全表示，但要注意 FP32 使用规格化数值表示 FP8 非规格化数值，具体的转换过程如算法 1 所示。

算法 1　FP8E4M3 数值转换回 FP32

输入：所有 FP8E4M3 格式数据 FP8Data

输出：对应的 FP32 格式数据 FP32Data

```
1. for all FP8Data do
2.     w=FP8Data<<24;
3.     sign=w &0x80000000;
4.     exponent=w+w;
5.     将 FP8Data 规格化值转换为 FP32 规格化值
6.     normal_v=exponent>>5+0x3C000000;
7.     mask=141 <<23;
8.     bias=0x1.0p+14f;
9.     将 FP8Data 非规格化值转换为 FP32 规格化值
10.    cutoff=0x10000000;
11.    if exponent< cutoff then
12.    FP32Data=sign | denormal_v;
13.    else
```

```
14.   FP32Data=sign | normal_v;
15.   end if
16.   return FP32Data;
17. end for
```

算法 1 中减去 bias 常量是因为 FP8 为非规格化数值，默认小数点之前整数位为 0。当转换回 FP32 时，FP32 规格化数据默认小数点之前整数位为 1，因此转换后的 FP32 数值比 FP8 原始数值更大，需要减去 bias 恢复到 FP8 格式表示的值。

3.2 量化类型与属性构建

为了在 MLIR 的 IR 文件中表示量化相关的信息，需要使用 MLIR 内置的 Quant Dialect 模块，该模块主要专注于量化实现，提供了用于支持量化的相关数据类型、类型转换和类型校验系统，比如用于表示校准类型的 CalibratedQuantizedType，该类型在 IR 转换过程中用于携带当前网络层激活值的最值或 threshold 阈值信息，如图 5 所示。其中，FP32 表示量化校准过程使用的是 FP32 类型数据进行推理，括号中表示当前层输入或者输出值的最值，当两个值相等时表示用于计算对称量化缩放因子 S 的 threshold。

除此之外，还有 uniformQuantizeType 用于表示均匀量化类型，该量化类型包含原始数据的表达类型和量化后数据的存储类型两个部分。其中，表达类型支持 FP32，FP16 和 BF16 等，存储类型支持有符号和无符号 INT8 类型或其他宽度的整型，其中 Scale 是一个双精度值，Zeropoint 为一个整数值。

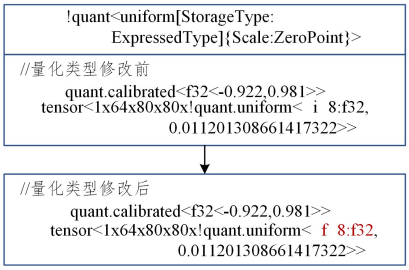

图 5 FP8 量化类型

Fig. 5 FP8 quantification type

由于 FP8 类型作为一种新兴类型，出现时间较晚，MLIR 中量化模块开发时未能考虑到新类型的出现，因此 MLIR 量化模块中并不支持将 FP32 表达类型转换为 FP8 存储类型。本文为了实现对 FP8 类型的量化支持，深入修改了 Quant Dialect 中的量化类型系统以及校验系统。为了保持最小侵入式改动，复用并拓展了均匀量化类型的存储类型支持，修改了类型解析器系统以满足 FP8 类型的解析，修改了 IR 打印器系统中对量化类型 FP8 的处理，在 IR 层面支持 FP32 到 FP8 的量化类型表达，如图 5 所示。

为了支持更细粒度的量化以评估 FP8 类型的量化效果，在 TPU-MLIR 中实现了 Per-channel 逐通道量化技术。当前 TPU-MLIR 中将权重数据文件和模型建模在同一层 IR 中，便于在模型推理之前计算出卷积核逐个通道的缩放因子，减少推理时的计算量。为了将提前计算出的缩放因子存储在 IR 中，对卷积算子的属性进行了扩展，为卷积算子添加一个浮点格式的属性 float_scale。添加后的卷积算子 IR 如图 6 所示，由于权重通道过多，只展示部分缩放因子值。

```
%5 = "tpu.Conv2D"(%2, %3, %4) {dilations = [1, 1], float_scale =
[0.05740785226225853, 0.0045583001337945461…], group = 1 : i64,
kernel_shape = [6, 6], kernel_zp = 0 : i64, pads = [2, 2, 2, 2], strides = [2, 2],
with_bias = true} :
(tensor<1x3x640x640x!quant.uniform<f8:f32,0.0022321607142857>>,
tensor<32x3x6x6xf32>, tensor<1x32x1x1xf32>) ->
tensor<1x32x320x320x!quant.uniform<f8:f32,0.16413830758928>>
```

图 6 FP8 逐通道量化支持

Fig. 6 FP8 per channel quantization support

3.3 缩放因子选取

由式(3)可知，缩放因子与零点值的选取直接决定了量化后模型的精度。其中，零点值根据低位宽数据分布区间确定，FP8 数据分布与 FP32 十分类似，不需要进行数据偏移，因此将零点值设置为 0。为了充分研究 FP8 类型缩放因子对模型精度的影响，本文将 FP8 缩放因子选取分为 3 步：首先，将缩放因子默认值设置为 1.0，即不对模型中的数据进行缩放，这样可以直接观察类型转换误差对模型精度的影响；然后，以 1.0 为基准选择 0.1，0.2 和 0.25 等常规值进行测试；最后，为了充分考虑到深度学习网络中各个网络层的激活值的差异，使用基于 Percentile99.99% 的校准策略[17]，计算缩放因子需要使用的 threshold 如式(5)所示。以 FP8E4M3 为例，Q_{max} 表示量化后数据的最大值为 448，Q_{min} 表示量化后数据的最小值为 −448。

$$S=\frac{threshold}{Q_{max}-Q_{min}} \tag{5}$$

考虑到更细粒度的逐通道量化策略，这里采用最大值策略统计出当前通道内数据的最大值作为当前通道缩放因子计算中的 threshold，通过与量化后最大值 448 相除得到缩放因子值。另外，取余策略选择数值计算中经常使用的 ROUNDING_HALF_TO_EVEN 方法。

为了在 MLIR IR 层面生成新的 FP32 到 FP8 量化数据类型，基于 TPU-MLIR 系统中的 Dialect Conversion 过程来完成该步骤。通过对 TOP IR 递降，生成具有 FP8 量化类型算子的 TPU IR。图 5 展示了从 FP32 数据类型转换为校准类型，进一步转换为 FP8 量化类型的过程，模拟策略转换如式(6)所示：

$$x_i^{(q)}=ToFP32(S*ToFP8(x_i^{(S)})) \tag{6}$$

其中：

$$x_i^{(S)}=clip\left(round\left(\frac{x_i}{S}+z\right),-448,448\right) \tag{7}$$

其中，x_i 表示 FP32 格式的真实数据值；S 为计算出的缩放因子；z 为零点，在这里采用对称量化，z 取 0；$round$ 为取余函数；$clip$ 为裁剪函数，以 FP8E4M3 为例，裁剪范围为 −448 到 448；ToFP8 函数为 2.1 节中构建的 FP32 到 FP8 的自动转换规则；ToFP32 函数是从 FP8 转换回 FP32 格式的算法 1 实现。采用缩放因子能够一定程度上提升模型的准确度是因为网络中的数值较小，过小的数据在转换为 FP8 过程中无法表示就会被舍弃而当作 0，通过一定程度的放大，可以让这些

数据对网络产生一定的影响，进而改善精度。FP8 量化模拟策略流程如下：

(1)通过小批量数据，基于 Percentile99.99%策略进行量化校准，获取 threshold，将模型类型从 FP32 转换为校准类型；

(2)通过 TOP IR 递降为 TPU IR 过程，生成新的 FP8 量化类型算子，通过量化公式计算出缩放因子 S 并添加到卷积算子的 float_scale 属性中；

(3)遍历 TPU IR，逐个算子进行推理，根据缩放因子 S，对算子的输入和权重参数进行缩放后取余、裁剪，转换到 FP8 格式后转换回 FP32 格式；

(4)输出网络中一些层的计算结果与 FP32 网络对应层进行对比。

3.4　误差分析

本方法中的量化误差来自量化后数据与真实数据之间的差值，如式(8)所示：

$$\sum R_i = x_i^q - x_i \tag{8}$$

其中，R_i 为量化后的值与真实值之间的误差。误差主要来源于 3 个部分：取余函数、裁剪函数和 ToFP8 函数。其中，取余函数会带来舍入误差，常见的有向上取整和向下取整等方法，本文通过选择数值计算中常用的 ROUNDING_HALF_TO_EVEN 方法最小化舍入误差。裁剪函数 clip 会将缩放后产生溢出的数据指定为当前数制的最大最小值，这会引入一些误差，导致转换回 FP32 类型时数据失真，可以通过选择合适的缩放因子减少该类型误差。ToFP8 函数因为小数位和指数位数较少，对 FP32 数据进行截断会不可避免地引入误差。网络中的每个权重数据和每层激活值都会在转换中产生误差，并在网络推理中进行传播。由于神经网络具有一定的鲁棒性，该误差反映到模型的精度验证上处于可接受范围。

4　推理引擎内存优化

在对 Yolov5s 网络进行 FP8 量化模拟时，进一步发现当前基于 TPU IR 的 CPU 端推理引擎存在优化空间。CPU 端推理引擎用于评估网络模型优化后的大小以及对模型计算结果进行快速验证，便于进一步部署到 TPU 硬件。当前 CPU 端推理引擎使用静态内存分配策略，优点是推理速度快，对于较小的模型十分友好；但对于稍大一些的模型，会占据过多的内存空间。本章通过结合 TPU IR 对网络模型进行分析，获取网络算子之间的数据流及控制流关系，进行内存优化，进一步减少模型推理过程中的内存占用峰值。

4.1　静态内存分配策略分析

TPU-MLIR 编译系统根据内部中间表示设计了一套强大的推理引擎，用于 TOP 层和 TPU 层 IR 的 CPU 端推理验证。该推理引擎分为 3 个步骤：

(1)遍历模型对应的 IR 文件，为整个模型的输入、输出、算子的权重、所有算子的输出进行内存空间分配；

(2)遍历模型对应的 IR 文件，为所有算子匹配对应的输入和输出空间；

(3)逐个算子调用对应的封装好的 OneDNN 静态链接库，针对当前算子的输入进行推理，将结果填充到输出空间。

当前的内存分配策略在模型的推理期间不再需要申请和释放额外的内存空间，模型推理速度快，但要求设备内存的最大容量必须大于步骤 1 中分配的内存空间，否则会造成内存分配失败，推理引擎无法执行。

与静态内存分配相对的是动态内存分配，即在推理过程中同时申请和释放内存，这会在一定程度上降低模型的推理速度。通过对模型的 IR 文件进行分析，发现模型中存在一些冗余的内存分配，可以通过内存复用策略减少静态内存分配阶段的内存峰值占用。内存复用策略基于静态内存分配方案实现，不会对推理速度产生影响。

4.2　内存复用支持

内存复用是多次对同一块内存单元进行读写的操作。在网络模型推理过程中，算子输出缓存区用于接收算子的输出数据，经过后续算子读取后，缓冲区不再被使用。可以将其他算子的计算结果写入不再被使用的缓冲区进行内存复用。网络模型中算子的输入和输出之间存在一定的数据流和控制流关系，算子的输出可能作为多个后续算子的输入，因此在进行内存复用时需分析计算图的特点，选择性地进行内存复用。

MLIR 编译器中输入和输出的对象都是一种基于 SSA (Static Single Assignment)的 Value 表示，编译器为每个 Value 对象维护一个包含定义和使用关系的 D-U 链，可以通过 Value 对象快速获取其使用者集合。借助 D-U 链可以分析出算子输入、输出缓冲区的使用情况，在不具备数据依赖时进行内存复用。

当前 Yolov5s 模型[16]具备大量 element-wise 算子，这些算子的输入、输出占据了相当一部分内存空间。内存复用还需要考虑内存空间的大小，根据推理引擎的设计，内存空间由算子的张量形状和元素类型决定。当前，Yolov5s 网络和 Resnet50 网络[17]中的激活算子 SILU 和 Add 算子的输入和输出具有相同的张量形状和类型，初步符合内存复用条件。通过修改推理引擎中的内存申请策略，判断特定算子的输入缓冲区的使用数量，当使用数量为 1 即仅仅是当前算子使用时，为当前算子的输出和输入分配同一块内存空间，进行内存复用。以激活算子为例，整体思路如图 7 所示。

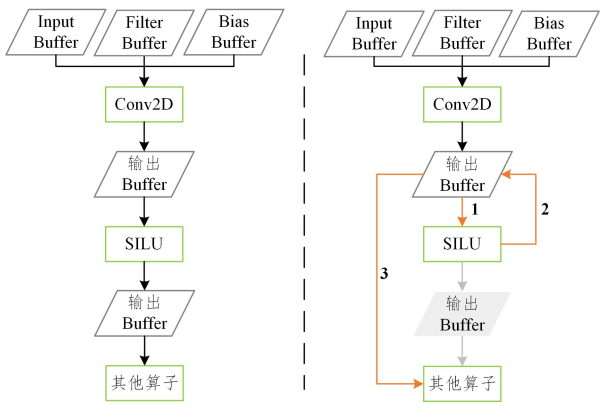

图 7　SILU 算子内存复用支持

Fig. 7　SILU operator memory reuse support

算法 2 展示了激活算子的内存复用策略，Add 等算子与之类似，同样需要判断输入的使用者数量。Add 算子的输入可能具有不同的形状，应选择更大的输入形状对应的空间进行复用。

算法 2　推理引擎算子内存复用策略

```
输入：element-wise 类算子如激活算子
1 if is a 〈tpu::ActiveOp〉(op) then
2.    activeop=dyn_cast〈tpu::ActiveOp〉(op);
3.    v=activeop.getInput();
4.    获取算子输入内存空间的使用者数量
5.    num=distance(v.user_begin(),v.user_end());
6.    opname=getName(v);
7.    param 为算子输入参数，map 容器存储所有
8.    算子输入和输出的内存空间对应的指针
9.    param->inputs=map[opname]->data();
10.   if num_uses==1 then
11.      将输出空间的指针指向输入空间地址
12.      param->outputs=map[opname]->data();
13.   else
14.      result=activeop->getResults();
15.      o_name=getName(result);
16.      param->outputs=map[o_name]->data();
17.   end if
18. end if
```

5　实验与结果分析

本文采用 TPU-MLIR 深度学习编译器和 TPU IR 层的 oneDNN 推理接口进行模型精度验证。相关的工具链版本为：TPU-MLIR v1.1.beta.1，MLIR 为 6e19eea02bbe7747cfca1f2a13287b9987ab959a 分支，使用的测试平台为 Ubuntu20.04 版本，CPU 为 x86 架构的 AMD R7 5800 @2.8 GHz，内存 16 GB。测试集中使用的 pycocotools 版本为 2.0.6。

5.1　测试集与实验方案

5.1.1　测试集

本文主要测试的网络对象为目标识别网络和分类网络。对于目标识别网络，使用公开的 COCO2017 以及 COCO2014 数据集[18]，验证每个数据集前 5 000 张照片的识别精度。对于分类网络，使用 ImageNet 经典数据集[19]，验证前 5 000 张照片的识别精度，测试集如表 3 所列。

表 3　测试集规模

Table 3　Test set size

测试集	输入图片规模	标注文件
COCO2017val	5 000	Annotation2017val
Coco2014val	41 000	Annotation2014val
ImageNet	50 000	分组代替标注

对于网络模型，采用开源项目 ONNX Model 库中预训练的 Yolov5s 和 Resnet50-v1-7 模型进行训练后量化 PTQ，不需要额外的模型训练工作。在模型推理的数量类型方面，分别使用 FP32，INT8，FPE4M3，FP8E5M2 这 4 种数据类型进行精度验证对比。

5.1.2　实验方案

为了测试 Yolov5s 网络模型的精度，使用 COCO 数据集官方提供的 Python 包 pycocotools 工具进行测试，取交并集为 0.5 时的精度数据。对于 Resnet50-v1-7 网络，采用 ImageNet2012 数据集进行测试，分别对推理结果中概率最大的标签号（即 TOP1 准确度）和推理结果中前 5 个概率最大的标签号（即 TOP5 准确度）进行统计，主要对模型中计算密集型算子卷积进行量化模拟。

为了测试 Yolov5s 和 RestNet-v1-7 网络的内存峰值占比，使用 Ubuntu 系统自带的 TOP 工具进行内存监测，执行 20 次求取平均值，然后结合网络中的理论数据进行分析。

为保证实验结果的准确性与精确度，每个测试集需要同时满足以下条件：(1)在该规模的测试集上至少进行 10 次实验；(2)内存测试上，测试 20 次及以上并取平均，得到最终的峰值结果。

5.2　结果与分析

5.2.1 网络精度对比

首先选取了 COCO2017 数据集进行测试，测试结果如图 8 所示。FP8E4M3 类型在缩放因子默认为 1.0 时，模型精度优于当前系统中的 INT8 的量化精度，与 FP32 类型相比，精度损失仅为 0.7%；将缩放因子选择为 0.2 时，可以进一步提升模型精度。相比 F8E4M3 类型，FP8E5M2 类型表示范围更大，但只有两位小数位，精度较低，在 Yolov5s 模型中精度损失比 F8E4M3 更加显著。在缩放因子默认为 1.0 时，模型精度仅为 52%，相比 FP32 类型，下降了 4.1%；将缩放因子设置为 0.1，会将精度损失缩小到 2.3%。

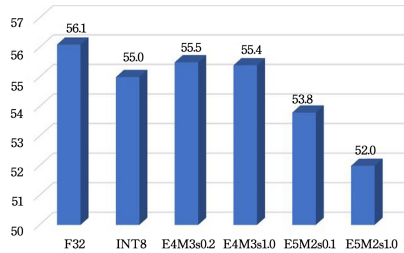

图 8　COCO2017 下的 Yolov5s mAP 测试（$IoU=0.5$）

Fig. 8　Yolov5s mAP test on COCO2017（$IoU=0.5$）

在 Percentile99.99% 校准策略下，两种 FP8 类型在 Yolov5s 模型中的平均精度分别为 31.3%和 1.2%，远低于缩放因子为 1.0 的情况，这是因为 Yolov5s 网络中的激活值与权重数据较小。对比 INT8 类型，FP8 数据类型在数据较小时数据密集度高，精度高，且由于小数位较少，在较大数值表达上密集度低，精度低。由式(7)可知，缩放因子的选择与 FP8 的表达范围相关，更大的表达范围会得到更小的缩放因子 S。选择过大的缩放因子会将网络中的数据映射到 FP8 中的较大数值，带来更大的误差。因此，FP8E5M2 类型与 FP8E4M3 类型在该策略下的精度较低。

在 COCO2014 数据集中得到了相同的结论，如图 9 所示。其中，Percentile99.99%校准策略下两种 FP8 格式的模型精度分别为 26.6%和 2.6%。

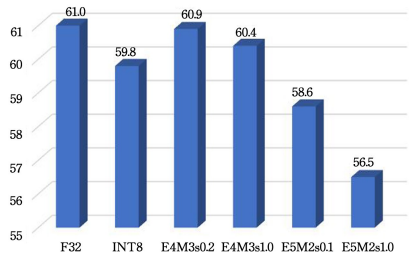

图 9 COCO2014 下的 Yolov5smAP 测试（*IoU*=0.5）

Fig. 9 Yolov5s mAP test on COCO2014(*IoU*=0.5)

对于 ResNet50 网络，在 ImageNet2012 测试中，两种 FP8 类型获得了与 INT8 类型接近的精度，结果如图 10 所示。在 FP8E4M3 类型缩放因子为 0.2 时，模型的 TOP1 准确率略高于 INT8。与 Yolov5s 网络模型测试结果类似，F8E4M3 类型模型的精度全面高于 F8E5M2 类型。在 Percentile99.99%校准策略下，两种 FP8 格式的模型的 TOP1 精度分别为 9.8%和 1.6%，TOP5 精度分别为 18.1%和 3.8%，远低于默认缩放因子为 1.0 时的精度，这是由于 ResNet50 网络中数据较小，将数据扩大到 FP8 类型较大数据时产生了较大误差。

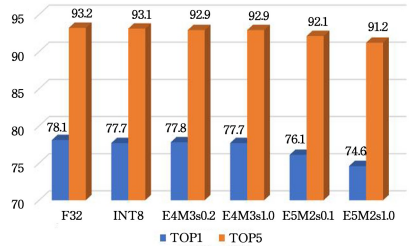

图 10 Resnet50 模型精度对比

Fig. 10 Comparison of Resnet50 model accuracy

对上述实验数据进行分析得知，使用 FP8E4M3 类型进行模型量化的效果略好于 INT8 类型，精度损失在可接受的范围内；而使用 FP8E5M2 类型量化时，精度损失大于 FP8E4M3 和 INT8 类型。因此，面向边缘端硬件部署值域比较小的模型时，应选择 FP8E4M3 格式进行量化，以获取最佳效果。同时，FP8 类型对缩放因子并不敏感，缩放因子为 1.0 时可以获得较高的模型预测精度。为了达到最佳模型精度，应该以 1.0 为基准进行测试，根据经验可以选择 0.1，0.2，0.25 和 0.5 等常规值进行测试，不需要依赖校准算法。相对 INT8 量化，FP8 量化可以省去模型校准的步骤，使得 FP8 的部署实施更为便捷。该策略在 CPU 端推理验证，是一种软件层面的模型量化策略，同样可以在支持 FP8 类型的端侧 TPU 等硬件上实现 FP8 类型的模型量化和数制转换工作，因此该策略具备一定的拓展性。

5.2.2 内存占用对比

为确保测试的合理性，对每组数据进行了 20 次测试并取平均值。对于 Yolov5s 网络中依赖关系单一的 element-wise 算子（如 SILU 和 Add 算子）经过定义使用链分析，算子输入值的使用者数量为 1，具备内存复用条件。通过多轮测试对比，内存复用前内存峰值占用约为 450～460MB，复用后内存占用峰值为 366～375MB，降低了约 20%，效果提升明显。对于 Resnet50v1-7 网络，其中也存在一些依赖关系单一的 element-wise 算子，如 Relu 和 Add 算子，具备内存复用条件。如图 11 所示，相比优化前，优化后内存峰值占用下降约 15%。

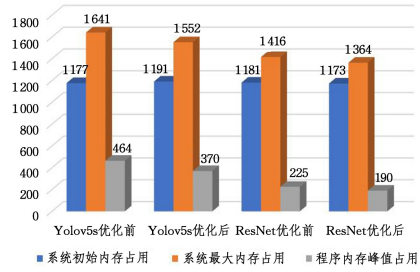

图 11 内存占用峰值对比

Fig. 11 Comparison of memory usage peaks

当前一些比较常见的网络模型，如 ViT-Base，EfficientNet 和 VGG 等，普遍存在大量的逐元素操作算子。对于具有如加法算子 AddOp、减法算子 SubOp，各种激活函数如 SigmoidOp、比较算子 CompOp、与或非算子等特征的网络，都可以借助该策略进行内存缓冲区使用情况的分析，通过将算子的输入与输出复用同一块内存缓冲区，优化模型推理过程中的内存占用。在硬件方面，当前内存复用方案在 CPU 端验证可以达到较好的效果，进一步地在个人移动终端设备如 ARM 硬件或边缘计算芯片如 RSIC-V 硬件上进行推理，也可以借助提出的内存复用技术在模型部署前对模型的数据流进行分析，改善模型在推理阶段的内存分配策略，节省内存空间，为更大的模型部署提供可能。

结束语 针对当前深度学习领域模型部署带来的巨大挑战，本文基于 TPU-MLIR 编译器对目标识别网络和分类网络模型网络进行了 FP8E4M3 和 FP8E5M2 的类型量化。其中，FP8E4M3 类型预测精度略高于 INT8 量化精度，相对于 FP32 类型，精度损失最低仅为 0.6%。通过实验进一步发现，FP8 格式对缩放因子的选取不敏感，不依赖特定校准算法，可以通过设置如 0.1 和 0.2 等经验值进行快速筛选，得出最优精度，相对于 INT8 量化来说，大大降低了部署的难度。基于编译器的内存缓存区使用情况分析与优化可以推广到包含逐点操作的所有网络中，以进一步降低模型的内存占用。当前支持 FP8 类型的硬件较少，如英伟达推出的 H100 芯片难以获取，无法在实际的硬件上测试具体的性能损失和功耗情况，这是下一步的工作方向。总之，FP8 的出现为推进模型量化研究提供了很好的思路，值得进一步研究。

参 考 文 献

[1] KRISHNAMOORTHI R. Quantizing deep convolutional networks for efficient inference: A whitepaper[J]. arXiv: 1806.08342, 2018.

[2] NOUNE B, JONES P, JUSTUS D, et al. 8-bit Numerical Formats for Deep Neural Networks[J]. arXiv: 2206.02915, 2022.

[3] KUZMIN A, VAN BAALEN M, REN Y W, et al. FP8 Quantization: The Power of the Exponent[C]// NeurIPS 2022. 2022.

[4] PASZKE A, GROSS S, MASSA F, et al. Pytorch: An imperative style, high-performance deep learning library[C]// Advances in Neural Information Processing Systems. 2019.

[5] ABADI M, AGARWAL A, BARHAM P, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems[J]. arXiv:1603.04467, 2016.

[6] LATTNER C, PIENAAR J A, AMINI M, et al. MLIR: A Compiler Infrastructure for the End of Moore's Law[J]. arXiv: 2002.11054, 2020.

[7] CHEN T, MOREAU T, JIANG Z, et al. TVM: end-to-end optimization stack for deep learning[J]. arXiv:1802.04799, 2018.

[8] LEARY C, WANG T. XLA: TensorFlow, compiled[C]// google tensorflow 2017. TensorFlow Dev Summit, 2017.

[9] HU P C, LU M, WANG L, et al. TPU-MLIR: A Compiler For TPU Using MLIR[J]. arXiv:2210.15016, 2022.

[10] KALAMKAR D D, MUDIGERE D, MELLEMPUDI N, et al. A Study of BFLOAT16 for Deep Learning Training[J]. arXiv: 1905.12322, 2019.

[11] JACOB B, KLIGYS S, CHEN B, et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference[C]// CVPR. 2018: 2704-2713.

[12] GASKILL B. Onnx: the open neural network exchange format [J]. Linux Journal, 2018(Apr. TN. 285): 157-161.

[13] LI J H, QIN Z N, MEI Y J, et al. oneDNN Graph Compiler: A Hybrid Approach for High-Performance Deep Learning Compilation [J]. arXiv:2301.01333, 2023.

[14] SUN X, CHOI J W, CHEN C Y, et al. Hybrid 8-bit Floating Point(HFP8) Training and Inference for Deep Neural Networks [C]// NeurIPS. 2019: 4901-4910.

[15] MICIKEVICIUS P, STOSIC D, BURGESS N, et al. FP8 Formats for Deep Learning[J]. arXiv:2209.05433, 2022.

[16] ZHOU X D, MENG Y, XIN X, et al. Research of YOLOv5s Model Acceleration Strategy in AI Chip[C]// ICCCS. 2023: 791-794.

[17] HE K M, ZHANG X Y, REN S Q, et al. Deep Residual Learning for Image Recognition[C]// CVPR. 2016: 770-778.

[18] LIN T Y, MAIRE M, BELONGIE S J, etal. Microsoft COCO: Common Objects in Context[C]// ECCV. 2014: 740-755.

[19] DENG J, DONG W, SOCHER R, et al. ImageNet: A large-scale hierarchical image database[C]// CVPR. 2009: 248-255.

XU Jinlong, born in 1985, Ph.D, master's supervisor. His main research interests include high-performance computing and paralle compilation.

HAN Lin, born in 1978, Ph.D, associate professor, is a senior member of CCF (No. 16416M). His main research interests include compiler optimization and high-performance computing.

(责任编辑:柯颖))