中国科学: 信息科学 2022 年 第 52 卷 第 6 期: 1040–1052
SCIENTIA SINICA Informationis
论文

面向机器学习系统的张量中间表示

庄毅敏[1,2,4], 文渊博[1,3,4], 李威[1,4], 郭崎[1*]

1. 中国科学院计算技术研究所计算机体系结构国家重点实验室, 北京 100190
2. 中国科学院大学, 北京 100049
3. 中国科学技术大学计算机科学与技术学院, 合肥 230026
4. 上海寒武纪信息科技有限公司, 上海 201308
* 通信作者. E-mail: guoqi@ict.ac.cn

收稿日期: 2020–12–24; 修回日期: 2021–03–12; 接受日期: 2021–04–16; 网络出版日期: 2022–06–10

国家自然科学基金 (批准号: 61925208)、北京市自然科学基金 (批准号: JQ18013)、中国科学院战略性先导科技专项 (批准号: XDB32050200)、中国科学院稳定支持基础研究领域青年团队计划 (批准号: YSBR-029) 和中国科学院青年创新促进会资助项目

摘要 随着各类机器学习算法的广泛应用, 高能效地定制机器学习系统受到越来越多的关注. 定制机器学习系统高效部署的关键在于其编程与编译环境. 中间表示是编程与编译环境的核心, 用于连接上层编程语言和底层硬件指令. 当前的中间表示或是面向上层算法或是面向以标量处理为核心的传统处理器, 难以高效应对以张量处理为核心的机器学习系统. 本文提出了面向机器学习系统的张量中间表示, 以提升机器学习系统的编程和运行效率. 具体而言, 我们定义了一系列张量类型, 张量操作及张量存储空间, 并在此基础上进行张量处理优化. 我们将所提出的张量中间表示对 TVM 的底层标量中间表示进行了扩展并在典型机器学习系统上进行了实验. 我们探索了原有中间表示没有发掘的优化并取得了 1.62~2.85 倍的性能提升, 同时在典型算子的开发效率上平均提升了 5.46 倍.

关键词 机器学习系统, 编程与编译, 张量处理, 中间表示, 编程效率

1 引言

以深度学习为代表的各类机器学习算法在多个领域 (如机器视觉、语音识别、机器翻译、机器人等) 都取得了巨大的成功. 为了提升机器学习算法的处理效率, 研究人员提出了大量领域定制系统, 即机器学习系统. 例如, 中国科学院计算技术研究所的 DianNao 系列架构 [1~4], Google 的 TPU [5], MIT 的 Eyeriss [6] 和 NVIDIA 的 Tensor Core [7] 等. 这些机器学习系统和传统处理器架构相比有显著区别, 如定制的运算单元 (如 Bfloat16 [1)], 比特串行单元 [8] 等), 复杂的片上存储以及领域专用指令集 (如矩阵乘法指令 [5], 卷积运算指令 [9]) 等.

1) Using Bfloat16 with tensorflow models. https://cloud.google.com/tpu/docs/bfloat16.

引用格式: 庄毅敏, 文渊博, 李威, 等. 面向机器学习系统的张量中间表示. 中国科学: 信息科学, 2022, 52: 1040–1052, doi: 10.1360/SSI-2020-0398
Zhuang Y M, Wen Y B, Li W, et al. A tensor intermediate representation for machine learning systems (in Chinese). Sci Sin Inform, 2022, 52: 1040–1052, doi: 10.1360/SSI-2020-0398

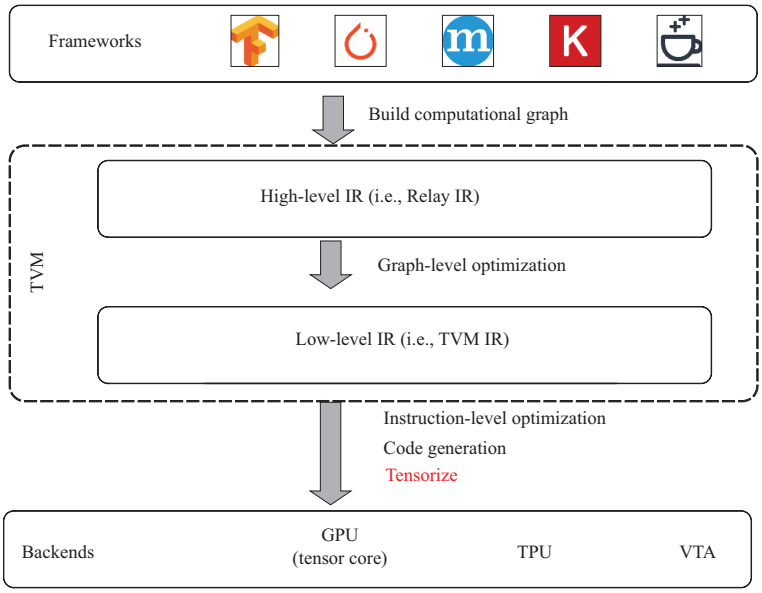

图 1 (网络版彩图) TVM 软件栈

Figure 1 (Color online) Overview of TVM stack

作为机器学习系统应用部署的关键, 其编程与编译环境主要包括 4 个层次: 高层编程语言 (如 Python, 领域专用语言 [10] 等[2]), 编程框架 (如 TensorFlow [11], PyTorch [12] 和 CNTK [13] 等), 高性能库 (如 NVIDIA 的 cuDNN [3] 和 Intel 的 oneDNN [4] 等) 和机器学习编译器 (如 TVM [14] 和 XLA [5] 等). 通常用户采用高层编程语言进行编程, 编程和编译系统将其翻译成机器学习系统的底层硬件指令.

中间表示用于连接高层编程语言和底层硬件指令. 当前的中间表示大致可以分为两类: (1) 基于图的中间表示, 如 TensorFlow 的数据流图, Relay IR [15], XLA 的 HLO (high level optimizer) IR 等. 这类中间表示描述机器学习应用的计算图结构, 例如数据流、运算操作和运算操作之间的依赖关系等信息. 由于这类中间表示通常没有硬件信息, 主要进行常量折叠、操作融合等各种硬件架构无关的通用优化. (2) 基于指令/代码的中间表示, 如 LLVM IR, TVM IR, Glow [16] 的 Low-Level IR 等. 这类中间表示通过标量形式描述具体的代数运算行为. 在这类中间表示上通常进行循环优化、冗余代码删除等传统编译器中常用优化, 然后根据不同后端进行硬件相关的优化和代码生成. 图 1 给出了以 TVM 软件栈为例的各种中间表示情况. 在 TVM 中, 以 Relay IR 作为图级别的中间表示, 以 TVM IR 作为代码级别中间表示. 由于机器学习系统一般具有张量运算能力, 代码生成过程中可以进行张量化 (tensorize) 处理以充分利用机器学习系统的运算能力.

当前中间表示主要是面向以标量处理为核心的传统处理器, 而没有考虑以张量处理为核心的机器学习系统, 导致在编程效率和代码生成上都存在问题. 图 2 以卷积操作为例, 结合中间表示到 Tensor Core 的下降过程具体阐释该问题. 在图级别中间表示中, Relay IR 直接以张量的形式表达操作的运算行为. 从 Relay IR 向 TVM IR 的下降过程, 张量操作被分解成由标量乘法和加法操作构成的多重循环. 然后, 在 TVM IR 中间表示上, 通过分块优化后, 重新进行张量化操作, 生成 Tensor Core 提供的

2) TensorFlow, PyTorch 的用户接口也可以看作是嵌入的领域专用语言 (embedded domain specific language).
3) NVIDIA cuDNN. https://developer.nvidia.com/cudnn.
4) Intel oneDNN. http://oneapi-src.github.io/oneDNN.
5) XLA: optimizing compiler for machine learning. https://www.tensorflow.org/xla.

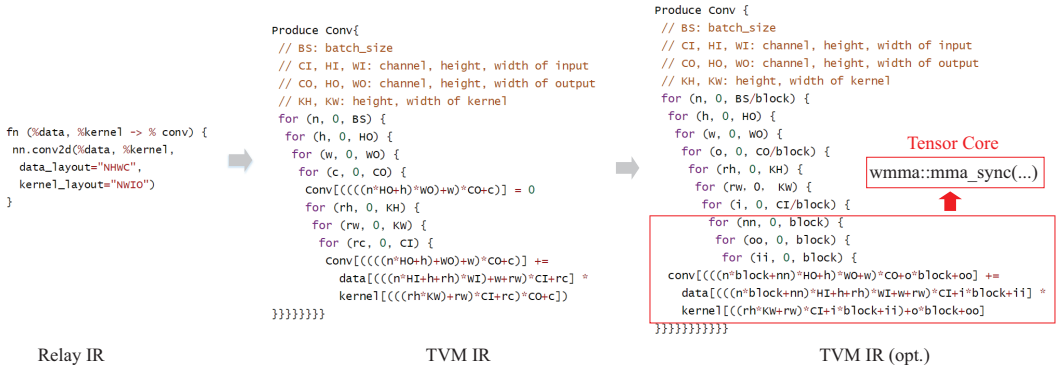

图 2　(网络版彩图) 中间表示下降过程

Figure 2　(Color online) The process of lowering IR

wmma::mma_sync 矩阵函数. 这种面向标量处理器的中间表示, 在以机器学习系统为后端的下降过程中, 形成了 “张量 – 标量 – 张量” 的处理过程, 带来了两个方面的问题. 在编程效率方面, 编译器或者程序员需要自动或手动从标量程序中恢复出张量语义, 这个过程增加了额外的负担并且容易产生大量的错误. 在代码生成方面, 不同机器学习系统的张量原语不同并且不能相互兼容, 这使得在底层中间表示上, 编译器需要为不同机器学习系统分别作张量化处理. 而张量语义的缺失也使得针对机器学习系统的张量层次优化难以被应用挖掘, 同时, 在张量层次进行的优化也需要在不同系统上分别实现, 带来了大量重复的代码生成工作.

针对当前机器学习系统的编程和编译环境中间表示的问题, 我们提出了张量中间表示, 以提升编程和运行效率. 该中间表示基于机器学习系统的架构特点, 提供了张量数据类型、张量操作类型和张量存储类型. 其中, 张量数据类型支持机器学习系统专有数据类型以及不同数据类型的转换, 张量操作类型提供了对向量和张量操作的直接表达能力, 张量存储类型增加了对片上存储的描述以使得程序员可以管理片上存储. 由于张量中间表示贴近底层硬件特点, 可以方便进行各类编译优化和高性能程序编写. 我们在 TVM 中实现了所提出的张量中间表示, 并在此基础上增加了 Tensor Core 的后端. 与 TVM 中原有的中间表示相比, 不需要使用复杂的张量化方法将标量中间表示转换为 Tensor Core 的张量原语, 这显著提升了开发效率. 更重要的是, 张量语义信息可以结合后端硬件特性进行统一搜索, 在更大空间中进行优化. 实验结果表明, 在典型的卷积和矩阵乘法算子上, 和原有中间表示相比, 基于张量中间表示的算子开发效率提升了 5.46 倍, 在部分算子上通过张量中间表示的优化方法可以取得 1.62 倍以上的性能提升.

本文主要贡献总结如下:

- 提出了面向机器学习系统的张量中间表示, 包括张量数据类型、操作类型和存储类型等.
- 在 TVM 中实现了面向机器学习系统的张量中间表示, 包括增加张量描述信息、拓展张量中间表示和实现张量层次的优化等.
- 在 V100 GPU 上验证了所实现张量中间表示的有效性. 实验结果表明, 和 TVM 原有标量中间表示相比, 典型算子开发效率平均提升了 5.46 倍, 部分算子运行性能提升了 1.62~2.85 倍.

本文的其余部分组织结构为: 第 2 节介绍相关工作; 第 3 节介绍所提出的张量中间表示定义; 第 4 节介绍张量中间表示的实现; 第 5 节介绍基于张量中间表示的优化; 第 6 节介绍实验评估及结果; 第 7 节讨论张量中间表示的更多实现可能; 第 8 节对全文进行了总结.

2 相关工作

2.1 机器学习系统

以深度学习为代表的各类机器学习算法具有计算和访存密集等特点. 为了高效地处理各种不同机器学习算法, 各类领域专用机器学习系统不断涌现. 目前, 主流机器学习系统结合了算法特性和硬件特性, 通过并行化的硬件设计结合算法数据流的特征, 取得了显著的性能和能效的提升. 例如, DianNao [1] 架构通过 3 块片上存储, 分别存放输入神经元、输出神经元和权重数据, 从而获得低延时和低功耗的收益. TPU [5] 包含了 256×256 的 8 位矩阵乘单元阵列和 28 MB 大小的片上内存, 最高可以达到 92 TeraOp/s (TOPS) 的吞吐. Eyeriss v2 [17] 通过定制化的 NOC (network on chip) 取得了很好的神经元和权值的复用, 并且通过支持稀疏矩阵运算取得更高的算力. Fused DNN [18] 提出了一种融合计算的处理器, 通过局部的数据运算融合多个计算层, 从而减少对 DRAM 的访存.

2.2 机器学习编译器

研究人员提出了多种具有灵活接口和优化功能的机器学习编译器. TVM [14] 是一个端到端的编译器栈, 它可以支持机器学习任务部署在不同的编程框架以及不同的硬件架构上. TensorFlow XLA 是一种领域专用的线性代数编译器, 能够在 TensorFlow 框架下, 加快模型的运行速度. Facebook 设计了一个两重中间表示的编译器 Glow [16], 其包括高阶中间表示和低阶中间表示. Intel 提出图优化编译器 nGraph [19], 通过中间表示对不同目标架构进行抽象, 将用户程序优化并生成不同目标代码.

2.3 中间表示

中间表示是编译器的重要组成, 通过中间表示可以解耦高层机器学习框架和底层硬件架构. 机器学习编译器通常使用多种层次的中间表示, 高层的中间表示通常描述算法信息并进行硬件无关的优化, 低层的中间表示主要负责硬件相关的优化和代码生成. 其中, 基于图的中间表示作为一种高层的中间表示, 被许多机器学习编译器广泛采用. 例如, TensorFlow 设计了一种图结构的中间表示, 包括用于表示操作结点的描述对象 Operation 和用于表示操作之间的数据流的描述对象 Tensor. TVM 使用 Relay IR 来描述计算图. Relay IR 是一种函数式可微的编程语言, 它支持多种数据类型, 支持控制流和递归等, 能够描述复杂的机器学习模型.

在硬件架构无关优化的基础上, 机器学习编译器使用基于指令/代码的中间表示来实现硬件架构相关的优化和代码生成过程. 对于优化后的 HLO, XLA 根据硬件架构特性进行优化, 通过调用硬件平台提供的高性能运算库或者通过 LLVM IR 等低层中间表示进行优化和代码生成. TVM IR 在中间表示层级提供了多种调度原语, 例如循环顺序交换、分块计算等. 程序员或编译器可以通过这些调度原语实现对不同操作的优化. TVM IR 可以生成不同后端的代码, 从而实现对不同硬件架构的支持.

3 张量中间表示的定义

3.1 概述

对于当前以标量为主的低层中间表示存在的不足, 本文结合了当前机器学习系统的特征, 提出了一种面向机器学习系统的张量中间表示. 相比于传统的标量操作, 例如算术运算、逻辑运算、比较运算、函数调用以及条件分支等, 该张量中间表示包含传统的标量操作并且增加了向量和张量操作的表达能力. 具体地, 该张量中间表示包括张量数据类型、张量操作类型和张量存储类型.

表 1 张量操作类型示例

Table 1 Examples of tensor operations

Type		Operation code	Example
Scalar	Arithmetic	add, sub, mult, div ...	add.dType dstReg, srcReg1, srcReg2
	Logic	eq, less, great ...	less.dType dstReg, srcReg1, srcReg2
	Control	jump, branch, sync ...	jump dstPC
	Data Access	load, store, move ...	load.dType dstReg, srcReg
Vector/Tensor description		tensor	tensor.dType name(dim1, dim2, ...)
Vector	Arithmetic	multv, addv, subv ...	multv(outTensor, inTensor)
	Logic	minv, maxv	minv(dstReg, inTensor)
	Data Access	loadv, storev, movev ...	loadv.dstRam.srcRam(outTensor, inTensor)
	Convert	convert	convert.dstType.srcType.mod(outTensor, inTensor, [scale], [bias])
Tensor	Function	conv, pool, gemm ...	conv(outTensor, inTensor, wTensor, kernel, stride, ...)

3.2 张量数据类型

机器学习模型, 特别是深度学习模型, 能够容忍低精度的数据类型而不影响应用程序效果. 因此, 具有更低功耗和更高运算能力的低精度数据类型被广泛应用于机器学习系统. 在张量数据类型的设计上, 我们充分考虑了机器学习系统带来的新的数据类型. 本小节将具体介绍张量数据类型的设计.

在数据类型方面, 张量数据类型支持传统编程语言 (如 C/C++ 语言等) 所支持的数据类型, 包括 float, int 和 bool 等类型. 在此基础上, 张量数据类型支持两种被当前机器学习系统广泛应用的数据类型, 也就是 16 位浮点数据类型 (包括 half 和 bfloat) 和量化数据类型 (包括 quant16 和 quant8). half 和 bfloat 数据类型与 float 数据类型表示方法类似, 但是具有更少的指数位或者有效位, 例如, half 数据类型包括 1 比特符号位, 5 比特指数位和 10 比特有效位. 量化数据类型通过整型数据加上量化因子和偏置表示浮点型数据的数值, 具体的关系表达式如下: $\text{value}_{\text{fp}} = \text{value}_{\text{int}} \times \text{scale} + \text{bias}$, 其中, scale 和 bias 被称为量化参数.

机器学习系统存在着众多数据类型, 使得数据类型转换操作在机器学习系统中非常常见. 为此, 张量中间表示提供给用户不同数据类型之间转换的操作. 由于数据类型较多并且不同数据类型对于机器学习系统的性能影响较大, 在类型转换过程中通常需要显式指明数据类型, 特别是在量化数据类型的转换过程中, 需要显式提供具体的量化参数. 需要注意的是, 低精度数据类型转换到高精度数据类型可以直接进行转换, 例如 short 到 float 的类型转换, 但是, 高精度数据类型到低精度数据类型转换需要额外指明舍入方式以减少精度失真带来的影响. 具体地, 我们提供了 6 种舍入方式: 向零舍入 (round toward zero)、向无穷舍入 (round off zero)、向下舍入 (round down)、向上舍入 (round up)、向偶数舍入 (round to even) 和向奇数舍入 (round to odd). 具体的类型转换操作的实现将在第 3.3 小节介绍.

3.3 张量操作类型

张量中间表示提供了对向量和张量操作的直接表达能力. 表 1 给出了张量操作类型的部分示例. 张量操作类型包含了标量操作, 并在标量操作的数据类型上拓展了机器学习系统引入的新兴数据类型. 对于向量和张量的操作, 我们新增了向量/张量描述符用于描述向量/张量的数据类型, 数据摆放格式和各个数据维度大小等信息. 张量操作使用该描述符作为输入/输出参数配合其他辅助参数完成操作的描述.

图 3 (网络版彩图) 中间表示优化过程对比. (a) 传统中间表示的优化过程; (b) 张量中间表示的优化过程

Figure 3 (Color online) Optimization comparison on two IR. (a) The optimization process of traditional IR; (b) the optimization process of Tensor IR

张量操作的支持使得机器学习应用的代码编写具有更高的效率, 包括编程框架中的核函数、高性能计算库和机器学习编译器等代码的编写. 图 3 提供了通过传统中间表示和张量中间表示进行 Tensor Core 优化编程的过程, 具体阐明了张量中间表示带来的编程效率的提升. 在传统中间表示中, 高层中间表示被分解成由标量运算构成的多层循环. 编译器需要从复杂的标量循环中找到 warp 的划分方式, 而后需要改变标量运算过程进行调度优化, 最后生成对应后端的代码. 这个优化过程非常烦琐并且容易引入错误. 在张量中间表示中, 基于张量操作的描述使得中间表示具有操作的算法信息, 能够直观地进行数据分块和优化, 通过简单的循环分解可以高效地进行代码生成. 相比于传统中间表示, 张量中间表示能更直观地描述张量运算行为, 使得编译器和程序员能够更高效地进行开发和探索更多的优化方法.

3.4 张量存储类型

张量中间表示需要能够支持不同的机器学习系统, 而不同机器学习系统的架构通常具有较大的差异, 这给张量中间表示的设计带来了很大的挑战. 本小节总结了当前机器学习系统具有的一般性特征, 参考 DianNao 架构构建了一个典型架构, 该典型架构可以迅速拓展到其他机器学习系统. 基于该典型架构构建中间表示, 既保证张量中间的表达能力, 又具有多平台的兼容能力.

机器学习系统通常具有复杂的片上存储. 典型的机器学习系统一般具有专门的权值存储空间, 例如 TPU, DianNao, Eyeriss 等架构, 通过特殊的权值存储获取更高的运算效率而神经元和其他数据则存放在另外的存储空间上; 或者具有多核/多线程共享的存储空间, 例如 GPU 中的 Shared Memory.

典型架构如图 4 所示. 我们设置了两块片上存储空间, 分别用于权值和神经元数据的存储. 此外, 还有运算单元、控制单元和 DRAM 存储空间等基本结构. 张量中间表示基于该典型架构进行构建, 在数据操作相关的操作上, 显式指定数据存储位置和流动方向; 而在代码生成阶段, 根据特定机器学习架构, 将存储空间映射到具体的片上存储上.

我们的张量存储类型是对该典型架构的片上存储进行描述, 也就是对权值存储空间和神经元存储空间的描述. 它具有两个方面的优点: (1) 该张量存储类型使得机器学习系统的片上存储对程序员可见, 程序员可以显式管理片上存储并进行关键的性能优化; (2) 基于典型的机器学习系统, 可以使得该

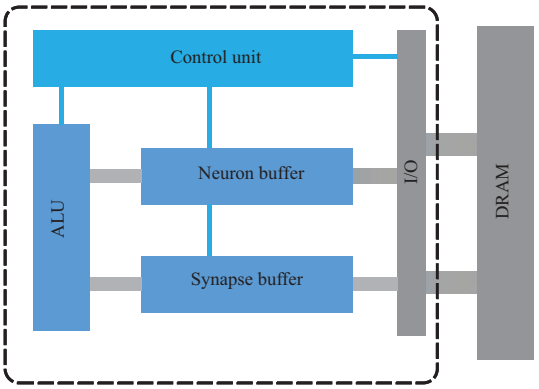

图 4 (网络版彩图) 典型架构

Figure 4 (Color online) Typical architecture

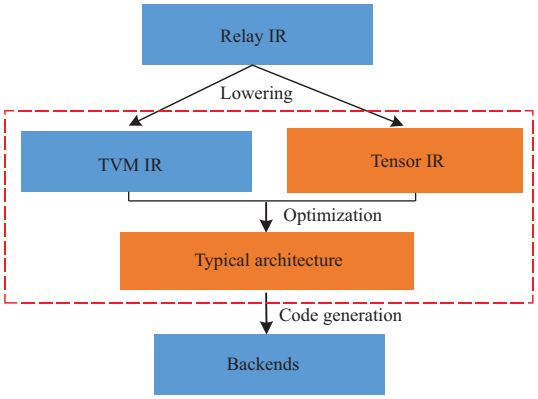

图 5 (网络版彩图) 张量中间表示实现

Figure 5 (Color online) The implementation of Tensor IR

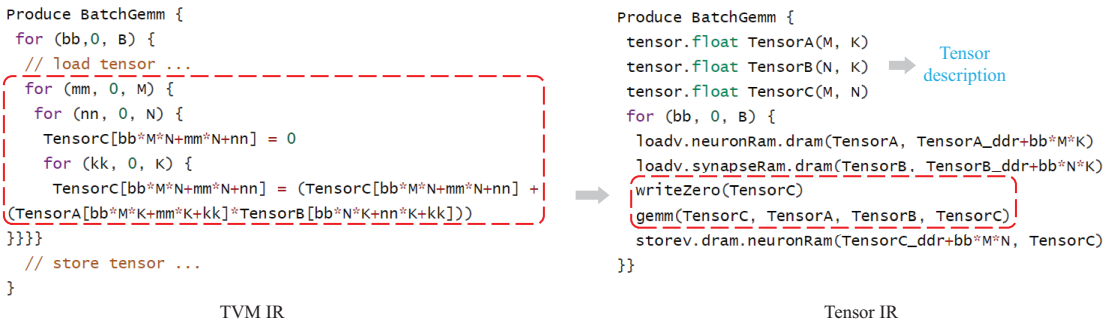

图 6 (网络版彩图) 张量中间表示实现示例

Figure 6 (Color online) Example of the implementation of Tensor IR

张量中间表示支持不同的机器学习系统, 具有更好的兼容性.

4 张量中间表示的实现

我们通过在 TVM 上对底层标量中间表示进行张量化的拓展实现对张量中间表示的支持. 本小节具体介绍张量中间表示在 TVM 中的实现.

如图 5 所示, 张量中间表示使用 Relay IR 作为输入, 在底层中间表示中, 拓展了张量语义描述信息, 包括张量的数据类型、存储和操作的描述. 在从 Relay IR 下降到 TVM IR 的过程中, 我们使用张量中间表示替换其中标量部分的下降过程. 最后, 结合 TVM IR、张量中间表示和典型架构进行优化, 生成目标后端代码.

图 6 以 BatchGemm 操作的实现为例进一步阐明张量中间表示的实现过程. 在张量表示中, 我们使用张量描述信息和张量访存操作替换原有中间表示的数据空间声明和访存行为, 使用张量运算操作替换多重循环的标量运算过程, 而在计算地址偏移等过程中仍使用原有中间表示的标量运算逻辑.

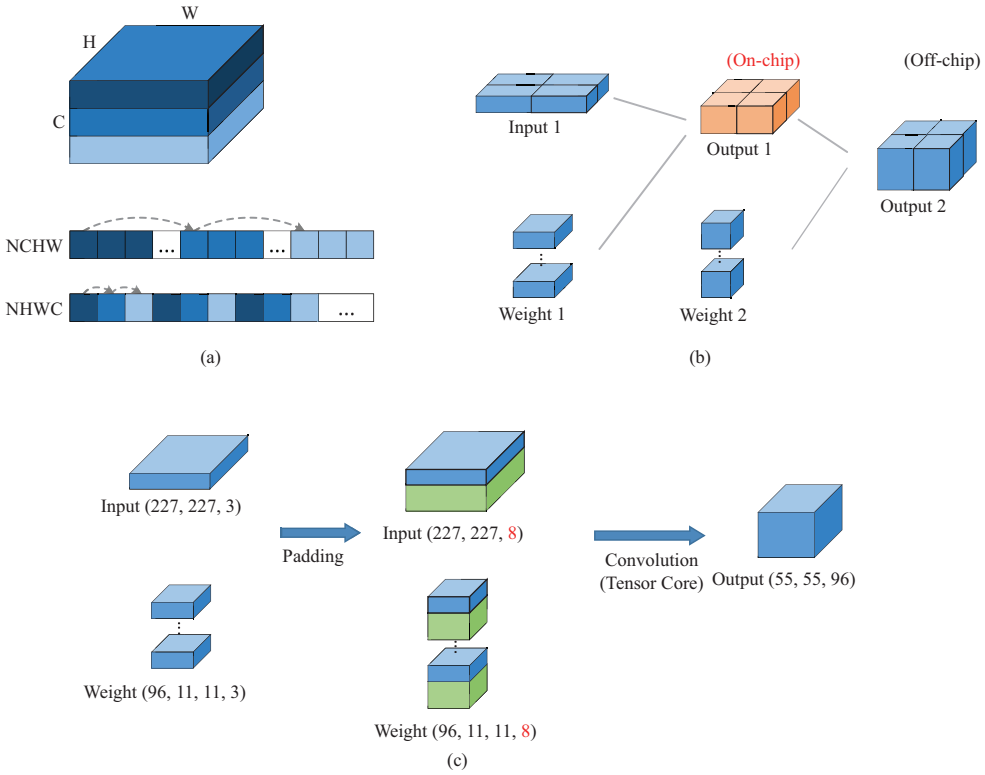

图 7 (网络版彩图) 张量中间表示的优化. (a) 数据布局优化; (b) 操作融合优化; (c) 补充冗余数据优化

Figure 7 (Color online) The optimizations base on Tensor IR. (a) Data layout optimization; (b) operation fusion optimization; (c) tensor padding optimization

5 基于张量中间表示的优化

张量中间表示使得底层中间表示具有更多数据和运算操作的信息, 能够提供更多潜在的优化机会. 与图级别中间表示相比, 图级别中间表示仅能够进行计算图相关的算法优化, 而张量中间表示同时具有算法和架构相关的信息, 通过结合张量语义信息和硬件架构信息, 能够挖掘更多的优化手段. 本节讨论张量中间表示带来的优化方法.

5.1 存储空间优化

张量中间表示具有更多张量数据属性的描述, 包括数据的维度信息、摆放方式和数据类型等. 基于这些数据, 可以进行存储相关的优化. 编译器可以根据算法和硬件架构特点调整最合适的数据摆放方式, 例如, 对于 GPU-TensorCore 架构可以调整卷积操作输入和权值的摆放顺序, 如图 7(a) 所示, 通过对数据预处理, 从 NCHW 摆放变换为 NHWC 摆放可以减少非连续数据的访存带来的访存性能下降问题.

5.2 张量运算优化

操作融合优化被广泛应用于机器学习系统, 通过将两个操作之间交互的输入输出数据暂存在片上存储中, 可以减少片上存储和片外存储之间的数据交互, 从而减少这部分访存带来的开销. 图 7(b) 展示了两个连续的卷积操作. 受限于有限的片上存储空间, 通常需要对数据进行分块, 因此操作的融合

表 2 测试例配置

Table 2 Configuration of experimental operations

	Cout	H/W	Cin	Kernel	Stride	Network
Conv1	96	227	3	11	4	Alexnet [20]
Conv2	486	32	1024	3	1	SSD_COCO [21]
Conv3	324	64	512	3	1	SSD_COCO
Conv4	64	56	64	3	1	ResNet50 [22]
Gemm1	96	–	24	–	–	MobileNetV3 [23]
Gemm2	1000	–	2048	–	–	ResNet50
Gemm3	1000	–	1536	–	–	Inception-v4 [24]

需要依次执行不同分块的多个卷积运算, 需要编译器完成运算的调度工作. 传统中间表示需要将多个操作的循环合并在一起并且需要精确的坐标分析, 实现上非常困难, 而通过张量表示的运算行为, 可以直观获取操作的运算行为, 从而更容易进行分块和优化.

机器学习系统专用加速运算模块通常会设置严格的形状限制以更高效地完成运算, 例如, Tensor Core 中要求矩阵乘必须按照 16×16, 8×32 或者 32×8 的分块大小. 在传统标量中间表示中, 在不满足运算约束的情况下, 无法使用这些运算模块. 而在张量中间表示中, 可以对张量数据进行数据变换操作, 例如通过补充冗余数据、转置和维度折叠等, 使得运算可以满足约束条件从而获得更高的运算效率. 如图 7(c) 所示的卷积操作, 对特征维度补充冗余数据, 使得运算可以在 Tensor Core 上完成计算, 从而取得更高的运算速度.

6 实验评估

本节介绍具体的实验方法和实验结果.

6.1 实验环境

我们以 TVM 作为实验使用的编译器框架, 使用 NVIDIA 的 V100 GPU 作为硬件后端. V100 具有 640 个 Tensor Core, 能够达到 112 TFLOPS 的峰值算力. 实验对应的 CUDA 和 cuDNN 的版本分别为 v10.0 和 v7.4.

在基准测试例上, 我们选取了卷积和矩阵乘作为实验测试例. 这些操作具有一定的代表性. 首先, 这些操作被广泛应用于真实机器学习模型之中; 其次, 这些操作的优化是整个机器学习模型优化的关键; 最后, 它们都具有张量表达能力.

实验测试例的规模信息如表 2 [20~24] 所示. 我们在 TVM 编译器框架上比较张量中间表示与原有中间表示在测试例上的性能差异. 在编程效率上, 通过比较两种中间表示在中间表示的优化和下降阶段所需的代码行数[6)]间接体现效率的差异.

6.2 实验结果

图 8 展示了张量中间表示和原有中间表示在 V100 上的性能. 我们以原有中间表示作为基准对比张量中间表示带来的性能提升. 实验结果表示, 张量中间表示在 Conv1∼Conv3 和 Gemm1 测试例上, 带来了 1.62∼2.85 倍的性能提升, 而在 Conv4, Gemm2 和 Gemm3 上, 与原有中间表示性能持平.

6) 仅统计有效行数, 不包括冗余换行、空行和异常判断等.

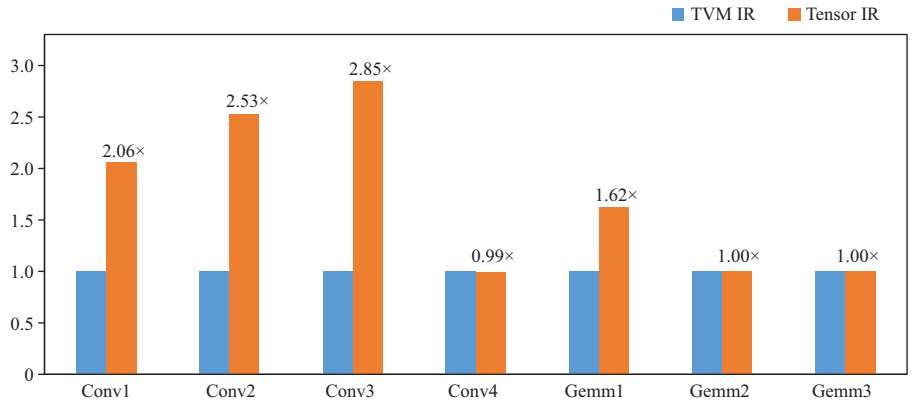

图 8 (网络版彩图) 张量中间表示和原有中间表示在 V100 GPU 上的性能对比

Figure 8 (Color online) Performance comparison of Tensor IR and TVM IR on V100 GPU.

表 3 两种中间表示优化和下降阶段代码行数对比

Table 3 Lines of code comparison of two IRs on optimizing and lowering

	Conv	Gemm	Average
TVM IR	194	144	–
Tensor IR	36	26	–
Ratio	5.39×	5.54×	5.46×

该性能差异主要由两个原因产生. (1) 在一些测试例上, 张量中间表示通过数据变换操作使得运算操作满足 Tensor Core 的约束, 可以在 Tensor Core 上运算, 带来了新的性能提升. (2) 在其他测试例上, 张量中间表示的优化方式与原有中间表示优化方式一致, 因此性能上基本持平. 具体地, 在测试例 Conv1∼Conv3 和 Gemm1 上, 原有中间表示受限于 Tensor Core 的分块限制, 无法使用具有更高运算效率的 Tensor Core 运算模块. 而张量中间表示具有张量描述带来的算法信息, 可以通过张量的维度变换, 例如 Padding, Reshape 等行为, 在不改变运算结果的情况下, 使得运算满足硬件架构的限制条件, 从而获得更高的运算效率.

我们的实验表明, 张量中间表示能够在运算操作上发现新的优化方法, 带来了 1.62 倍以上的性能提升. 在实际网络中, 这些操作通常具有较大的运算量, 容易成为运算瓶颈. 而在网络级别的优化上, 张量中间表示也能够带来一些新的优化方法, 这将在后续的研究中继续探索.

在编程效率方面, 张量中间表示和原有中间表示的代码行数如表 3 所示. 使用张量中间表示相比于原有中间表示在 Conv 和 Gemm 操作上分别有 5.39 倍和 5.54 倍的效率提升. 我们分析效率上的提升主要来源于以下 3 个方面. (1) 原有中间表示由多重循环构成, 每重循环都需要单独指定拆分调度方式, 例如 split, reorder 等, 并且需要描述循环之间的嵌套关系. (2) 原有中间表示需要对所有使用存储空间进行单独声明, 而张量中间表示对空间的管理已经包含在张量描述符和张量操作里面. (3) 原有中间表示需要针对循环嵌套的标量运算过程进行变换, 形成带有张量语义的程序片段, 然后进行张量化操作, 带来额外的编程开销.

7 讨论

当前的张量中间表示是在 TVM 中实现的, 其实现方式借鉴了 TVM 的优化调度原语和搜索方法. 但是所提出的张量中间表示并不局限于 TVM, 同样可以应用于其他机器学习编译器, 如 XLA 和 MLIR. 在 XLA 中, 张量中间表示可以以 HLO IR 作为输入, 在底层构造张量语义, 同时提供张量优化方法. 用户程序下降到 HLO IR 后, 从 HLO IR 下降到张量中间表示, 然后再进行优化, 生成目标后端代码. 在 MLIR 中, 通过拓展张量方言 (Dialect) 提供张量表示的支持. 在方言下降过程中, 替换图级别相关方言的张量相关部分的下降过程, 使其下降到张量方言上. 然后在张量方言上进行优化, 生成目标后端代码. 与原本的 XLA 或 MLIR 相比, 张量中间表示不需要将图级别中间表示转换为 LLVM IR 等标量表示, 减少了张量化过程的开销, 同时张量中间表示也保留了张量语义信息, 能够根据后端架构特征进行优化. 本文所提出的张量中间表示同样可以支持其他机器学习系统, 如 Google TPU 和 Cambricon MLU 等.

8 结论

本文提出了面向机器学习系统的张量中间表示. 张量中间表示支持机器学习系统专有数据类型, 能够直接表达向量和张量运算并且能够在中间表示层次提供对机器学习片上存储空间的管理. 与传统中间表示相比, 张量中间表示提供了更多的张量信息, 结合底层硬件的特点, 能够在更大的空间中探索更深层次的优化方法. 在编程上, 张量中间表示不需要通过复杂而易错的张量化方法将标量中间表示转换为机器学习系统的各种张量指令或者原语, 使得开发效率平均有 5.46 倍的提升.

我们未来的工作将会探索张量中间表示的进一步优化工作, 包括基于张量存储的片上空间管理优化、基于张量操作的多操作融合优化等; 同时将张量中间表示部署于更多的机器学习系统, 使得张量中间表示具有更好的多平台兼容性和更高的性能.

参考文献

1 Chen T S, Du Z D, Sun N H, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014. 269–284

2 Chen Y J, Luo T, Liu S L, et al. DaDianNao: a machine-learning supercomputer. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014. 609–622

3 Liu D F, Chen T S, Liu S L, et al. PuDianNao: a polyvalent machine learning accelerator. In: Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems, 2015. 369–381

4 Du Z D, Fasthuber R, Chen T S, et al. ShiDianNao: shifting vision processing closer to the sensor. In: Proceedings of ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), 2015. 92–104

5 Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture, 2017. 1–12

6 Chen Y H, Emer J, Sze V. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. In: Proceedings of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016. 367–379

7 Markidis S, Chien S W D, Laure E, et al. NVIDIA Tensor Core programmability, performance & precision. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2018. 522–531

8 Judd P, Albericio J, Hetherington T, et al. Stripes: bit-serial deep neural network computing. In: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, 2016. 1–12

9 Liu S L, Du Z D, Tao J H, et al. Cambricon: an instruction set architecture for neural networks. In: Proceedings of ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016. 393–405

10 Truong L, Barik R, Totoni E, et al. Latte: a language, compiler, and runtime for elegant and efficient deep neural networks. In: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2016. 209–223

11 Abadi M, Barham P, Chen J, et al. TensorFlow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, 2016. 265–283

12 Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of Advances in Neural Information Processing Systems, 2016. 8026–8037

13 Seide F, Agarwal A. CNTK: Microsoft's open-source deep-learning toolkit. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016. 2135–2135

14 Chen T Q, Moreau T, Jiang Z H, et al. TVM: an automated end-to-end optimizing compiler for deep learning. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, 2018. 578–594

15 Roesch J, Lyubomirsky S, Weber L, et al. Relay: a new IR for machine learning frameworks. In: Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, 2018. 58–68

16 Rotem N, Fix J, Abdulrasool S, et al. Glow: graph lowering compiler techniques for neural networks. 2018. ArXiv:1805.00907

17 Chen Y H, Yang T J, Emer J S, et al. Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices. IEEE J Emerg Sel Top Circ Syst, 2019, 9: 292–308

18 Du X Z, El-Khamy M, Lee J, et al. Fused DNN: a deep neural network fusion approach to fast and robust pedestrian detection. In: Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV), 2017. 953–961

19 Cyphers S, Bansal A K, Bhiwandiwalla A, et al. Intel nGraph: an intermediate representation, compiler, and executor for deep learning. 2018. ArXiv:1801.08058

20 Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks. Commun ACM, 2017, 60: 84–90

21 Liu W, Anguelov D, Erhan D, et al. SSD: single shot MultiBox detector. In: Proceedings of European Conference on Computer Vision, 2016. 21–37

22 He K M, Zhang X Y, Ren S Q, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. 770–778

23 Howard A, Scandler M, Chu G, et al. Searching for MobileNetV3. In: Proceedings of the IEEE International Conference on Computer Vision, 2019. 1314–1324

24 Szegedy C, Ioffe S, Vanhoucke V, et al. Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the 31st AAAI Conference on Artificial Intelligence, 2017. 4278–4284

A tensor intermediate representation for machine learning systems

Yimin ZHUANG[1,2,4], Yuanbo WEN[1,3,4], Wei LI[1,4] & Qi GUO[1*]

1. *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190*, China*;
2. *University of Chinese Academy of Sciences, Beijing* 100049*, China*;
3. *School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230026*, China*;
4. *Cambricon Technologies Corporation Limited, Shanghai* 201308*, China*
* Corresponding author. E-mail: guoqi@ict.ac.cn

Abstract With the wide deployment of various machine learning algorithms, highly energy-efficient customized machine learning systems have gained popularity. The machine learning compilers are crucial to machine learning systems. The intermediate representation is the key to programming and compilation environments, and it connects the high-level programming language and the lower-level instruction set architectures. The current state-of-the-art intermediate representations are either oriented to high-level algorithms or classical processors based on scalar processing, but they cannot be effectively implemented on tensor-based machine learning systems. To address this problem, we propose a tensor intermediate representation for machine learning systems to improve programming productivity and performance. Concretely, we define a series of tensor types, tensor operations, and tensor memories and optimize the tensor processing based on these definitions. To validate our proposal, we extend the proposed tensor intermediate representation to the low-level scalar intermediate representation of TVM and perform experiments with Tensor Core on a typical machine learning system. Experimental results show that we explore optimizations that are not discovered in the original intermediate representation and achieve 1.62×∼2.85× performance improvement. Besides, the tensor intermediate representation improves the efficiency of programming by 5.46× on average.

Keywords machine learning systems, programming & compiling, tensor processing, intermediate representation, programming efficiency