TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

面向深度学习的自动化端到端优化编译器

arXiv:1802.04799v3 [cs.LG] 5 Oct 2018

arXiv:1802.04799v3 [cs。2018年10月5日

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan

陈天棋、莫雷、姜子恒、郑、阎爱迪

Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, Arvind KrishnamurthyPaul G. Allen School of Computer Science & Engineering, University of Washington

华盛顿大学计算机科学与工程学院

AWS, Shanghai Jiao Tong University, UC Davis, Cornell

AWS，上海交通大学，加州大学戴维斯分校，康奈尔大学

Abstract

摘要

L1D L1I L1D L1I

L1D L1I L1D L1I

L2 L2

L2·L2

L3

L3

RF RF RF RF

射频射频射频

L1/TX L1/TX

L1/德克萨斯L1/德克萨斯

L2

L2

SM SM Activation

SM SM激活

Buffer

缓冲器

Accum.Register

伏隔。注册

File

文件

Wgt.FIFO

Wgt。先入先出

CPU GPU 'TPU'

中央处理器图形处理器' TPU '

implicitly managed mixed explicitly managed

隐式管理混合显式管理

Memory Subsystem Architecture

内存子系统架构

There is an increasing need to bring machine learn-ing to a wide diversity of hardware devices.Current frameworks rely on vendor-specific operator libraries and optimize for a narrow range of server-class GPUs.Deploying workloads to new platforms - such as mo-bile phones, embedded devices, and accelerators (e.g., FPGAs, ASICs) - requires significant manual effort.We propose TVM, a compiler that exposes graph-level and operator-level optimizations to provide performance portability to deep learning workloads across diverse hardware back-ends.TVM solves optimization chal-lenges specific to deep learning, such as high-level op-erator fusion, mapping to arbitrary hardware primitives, and memory latency hiding.It also automates optimiza-tion of low-level programs to hardware characteristics by employing a novel, learning-based cost modeling method for rapid exploration of code optimizations.Experimen-tal results show that TVM delivers performance across hardware back-ends that are competitive with state-of-the-art, hand-tuned libraries for low-power CPU, mo-bile GPU, and server-class GPUs.We also demonstrate TVM's ability to target new accelerator back-ends, such as the FPGA-based generic deep learning accelerator.The system is open sourced and in production use inside several major companies.

越来越需要将机器学习应用到各种硬件设备中。当前的框架依赖于特定于供应商的运算符库，并针对范围狭窄的服务器级GPU进行优化。将工作负载部署到新平台，如移动电话、嵌入式设备和加速器(如FPGAs、ASICs)，需要大量的手动工作。我们提出TVM，这是一个公开图形级和操作级优化的编译器，为不同硬件后端的深度学习工作负载提供性能可移植性。TVM解决了深度学习特有的优化挑战，如高级运算符融合、到任意硬件原语的映射以及内存延迟隐藏。它还通过采用一种新颖的、基于学习的成本建模方法来快速探索代码优化，从而根据硬件特性自动优化低级程序。实验结果表明，TVM在硬件后端提供的性能与针对低功耗CPU、移动GPU和服务器级GPU的最先进的手动调整库具有竞争力。我们还展示了TVM针对新加速器后端的能力，例如基于FPGA的通用深度学习加速器。该系统是开源的，并在几家大公司内部生产使用。

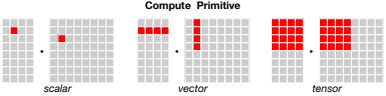


Figure 1: CPU, GPU and TPU-like accelerators re-quire different on-chip memory architectures and com-pute primitives.This divergence must be addressed when generating optimized code.

图1:中央处理器、图形处理器和类似TPU的加速器需要不同的片上内存架构和计算机原语。生成优化代码时，必须解决这种差异。

terms of memory organization, compute functional units, etc., as shown in Figure 1.

在内存组织、计算功能单元等方面。，如图1所示。

Current DL frameworks, such as TensorFlow, MXNet, Caffe, and PyTorch, rely on a computational graph in-termediate representation to implement optimizations, e.g., auto differentiation and dynamic memory man-agement [3, 4, 9].Graph-level optimizations, however, are often too high-level to handle hardware back-end-specific operator-level transformations.Most of these frameworks focus on a narrow class of server-class GPU devices and delegate target-specific optimizations to highly engineered and vendor-specific operator li-braries.These operator-level libraries require significant manual tuning and hence are too specialized and opaque to be easily ported across hardware devices.Providing support in various DL frameworks for diverse hardware back-ends presently requires significant engineering ef-fort.Even for supported back-ends, frameworks must make the difficult choice between: (1) avoiding graph optimizations that yield new operators not in the prede-fined operator library, and (2) using unoptimized imple-mentations of these new operators.

当前的动态链接库框架，如TensorFlow、MXNet、Caffe和PyTorch，依赖计算图中间表示来实现优化，例如自动微分和动态内存管理[3，4，9]。然而，图形级优化通常过于高级，无法处理硬件后端特定的操作符级转换。这些框架大多专注于服务器级GPU设备的狭窄类别，并将特定目标的优化委托给高度工程化和特定供应商的运营商图书馆。这些操作员级库需要大量的手动调整，因此过于专门化和不透明，难以跨硬件设备移植。目前，为不同的硬件后端提供各种不同的数据链路框架支持需要大量的工程支持。即使对于支持的后端，框架也必须在两者之间做出艰难的选择:(1)避免产生不在预先定义的运算符库中的新运算符的图形优化，以及(2)使用这些新运算符的未优化实现。

To enable both graph- and operator-level optimiza-

为了实现图形级和操作员级优化-

1 Introduction

1导言

Deep learning (DL) models can now recognize images, process natural language, and defeat humans in challeng-ing strategy games.There is a growing demand to deploy smart applications to a wide spectrum of devices, rang-ing from cloud servers to self-driving cars and embed-ded devices.Mapping DL workloads to these devices is complicated by the diversity of hardware characteristics, including embedded CPUs, GPUs, FPGAs, and ASICs (e.g., the TPU [21]).These hardware targets diverge in

深度学习模型现在可以识别图像，处理自然语言，并在挑战性的策略游戏中击败人类。从云服务器到自动驾驶汽车和嵌入式设备，越来越多的人需要在各种设备上部署智能应用。硬件特性的多样性使得将数据链路工作负载映射到这些设备变得复杂，包括嵌入式处理器、图形处理器、现场可编程门阵列和专用集成电路(如TPU [21])。这些硬件目标在

1

1

tions for diverse hardware back-ends, we take a fun-damentally different, end-to-end approach.We built TVM, a compiler that takes a high-level specification of a deep learning program from existing frameworks and generates low-level optimized code for a diverse set of hardware back-ends.To be attractive to users, TVM needs to offer performance competitive with the multi-tude of manually optimized operator libraries across di-verse hardware back-ends.This goal requires addressing the key challenges described below.

对于不同的硬件后端，我们采用一种有趣的、完全不同的端到端方法。我们构建了TVM，这是一个编译器，它从现有框架中获取深度学习程序的高级规范，并为不同的硬件后端生成低级优化代码。为了对用户有吸引力，TVM需要提供与跨硬件后端的多种手动优化的运营商库相竞争的性能。这个目标要求解决下面描述的关键挑战。

Leveraging Specific Hardware Features and Abstrac-tions.DL accelerators introduce optimized tensor com-pute primitives [1, 12, 21], while GPUs and CPUs con-tinuously improve their processing elements.This poses a significant challenge in generating optimized code for a given operator description.The inputs to hardware in-structions are multi-dimensional, with fixed or variable lengths;they dictate different data layouts;and they have special requirements for memory hierarchy.The system must effectively exploit these complex primitives to ben-efit from acceleration.Further, accelerator designs also commonly favor leaner control [21] and offload most scheduling complexity to the compiler stack.For spe-cialized accelerators, the system now needs to gener-ate code that explicitly controls pipeline dependencies to hide memory access latency - a job that hardware per-forms for CPUs and GPUs.

利用特定的硬件功能和抽象。DL加速器引入了优化的张量计算原语[1，12，21]，而图形处理器和中央处理器不断改进它们的处理元素。这在为给定的操作员描述生成优化代码方面带来了巨大的挑战。硬件指令的输入是多维的，具有固定或可变的长度；它们规定了不同的数据布局；它们对内存层次有特殊要求。系统必须有效地利用这些复杂的原语来避免加速。此外，加速器设计通常还倾向于更精简的控制[21]，并将大部分调度复杂性转移到编译器堆栈上。对于专用加速器，系统现在需要生成明确控制流水线依赖性的代码，以隐藏内存访问延迟——这是硬件为CPU和GPU执行的一项任务。

Large Search Space for Optimization Another chal-lenge is producing efficient code without manually tun-ing operators.The combinatorial choices of memory ac-cess, threading pattern, and novel hardware primitives creates a huge configuration space for generated code (e.g., loop tiles and ordering, caching, unrolling) that would incur a large search cost if we implement black box auto-tuning.One could adopt a predefined cost model to guide the search, but building an accurate cost model is difficult due to the increasing complexity of modern hardware.Furthermore, such an approach would require us to build separate cost models for each hard-ware type.

用于优化的大搜索空间另一个挑战是无需手动调整运算符就能产生高效的代码。内存访问、线程模式和新硬件原语的组合选择为生成的代码(例如循环片和排序、缓存、展开)创建了巨大的配置空间，如果我们实现黑盒自动调整，这将导致巨大的搜索成本。人们可以采用预定义的成本模型来指导搜索，但由于现代硬件的复杂性不断增加，很难建立准确的成本模型。此外，这种方法需要我们为每种硬件类型建立单独的成本模型。

TVM addresses these challenges with three key mod-ules.(1) We introduce a tensor expression language to build operators and provide program transformation primitives that generate different versions of the pro-gram with various optimizations.This layer extends Halide [32]'s compute/schedule separation concept by also separating target hardware intrinsics from transfor-mation primitives, which enables support for novel ac-celerators and their corresponding new intrinsics.More-over, we introduce new transformation primitives to ad-dress GPU-related challenges and enable deployment to specialized accelerators.We can then apply different se-quences of program transformations to form a rich space

东方神起用三个关键模块来应对这些挑战。(1)我们引入张量表达式语言来构建运算符，并提供程序转换原语，这些原语通过各种优化生成不同版本的程序。这一层扩展了卤化物[32]的计算/调度分离概念，还将目标硬件内部结构与转换原语分离，从而支持新型加速器及其相应的新内部结构。此外，我们引入了新的转换原语来应对与GPU相关的挑战，并支持部署到专业加速器。然后我们可以应用不同顺序的程序转换来形成一个丰富的空间

of valid programs for a given operator declaration.(2) We introduce an automated program optimization frame-work to find optimized tensor operators.The optimizer is guided by an ML-based cost model that adapts and im-proves as we collect more data from a hardware back-end.(3) On top of the automatic code generator, we introduce a graph rewriter that takes full advantage of high- and operator-level optimizations.

给定运算符声明的有效程序的。(2)引入自动程序优化框架来寻找优化的张量算子。优化器由基于最大似然的成本模型指导，当我们从硬件后端收集更多数据时，该模型会进行调整和改进。(3)在自动代码生成器的基础上，我们引入了一个图形重写器，它充分利用了高级和操作级优化。

By combining these three modules, TVM can take model descriptions from existing deep learning frame-works, perform joint high- and low-level optimizations, and generate hardware-specific optimized code for back-ends, e.g., CPUs, GPUs, and FPGA-based specialized accelerators.

通过结合这三个模块，TVM可以从现有的深度学习框架中获取模型描述，执行联合的高级和低级优化，并为后端(例如，CPU、GPU和基于FPGA的专用加速器)生成硬件特定的优化代码。

This paper makes the following contributions: • We identify the major optimization challenges in pro-

本文做出了以下贡献:我们确定了在支持方面的主要优化挑战

viding performance portability to deep learning work-loads across diverse hardware back-ends.

为不同硬件后端的深度学习工作负载提供性能可移植性。

• We introduce novel schedule primitives that take ad-vantage of cross-thread memory reuse, novel hardware intrinsics, and latency hiding.

我们引入了新颖的调度原语，这些原语利用了跨线程内存重用、新颖的硬件内在特性和延迟隐藏。

• We propose and implement a machine learning based optimization system to automatically explore and search for optimized tensor operators.

我们提出并实施了基于机器学习的优化系统，以自动探索和搜索优化的张量算子。

• We build an end-to-end compilation and optimiza-tion stack that allows the deployment of deep learning workloads specified in high-level frameworks (includ-ing TensorFlow, MXNet, PyTorch, Keras, CNTK) to diverse hardware back-ends (including CPUs, server GPUs, mobile GPUs, and FPGA-based accelerators).The open-sourced TVM is in production use inside several major companies.

我们构建了一个端到端的编译和优化堆栈，允许将高级框架中指定的深度学习工作负载(包括TensorFlow、MXNet、PyTorch、Keras、CNTK)部署到不同的硬件后端(包括CPU、服务器GPU、移动GPU和基于FPGA的加速器)。开源的TVM正在几家大公司内部生产使用。

We evaluated TVM using real world workloads on a server-class GPU, an embedded GPU, an embedded CPU, and a custom generic FPGA-based accelerator.Experimental results show that TVM offers portable performance across back-ends and achieves speedups ranging from 1.2× to 3.8× over existing frameworks backed by hand-optimized libraries.

我们在服务器级GPU、嵌入式GPU、嵌入式CPU和定制的通用基于FPGA的加速器上使用真实世界的工作负载来评估TVM。实验结果表明，TVM提供了跨后端的可移植性能，比手动优化库支持的现有框架实现了1.2倍到3.8倍的加速比。

2 Overview

2概述

This section describes TVM by using an example to walk through its components.Figure 2 summarizes execu-tion steps in TVM and their corresponding sections in the paper.The system first takes as input a model from an existing framework and transforms it into a computa-tional graph representation.It then performs high-level dataflow rewriting to generate an optimized graph.The operator-level optimization module must generate effi-cient code for each fused operator in this graph.Oper-ators are specified in a declarative tensor expression lan-

本节通过一个示例来介绍TVM的组件。图2总结了TVM中的执行步骤以及本文中相应的部分。该系统首先从一个现有的框架中获取一个模型作为输入，并将其转换成一个计算图形表示。然后，它执行高级数据流重写，以生成优化的图形。操作员级优化模块必须为该图中的每个融合操作员生成高效代码。运算符在声明性张量表达式中指定

2

2

Frameworks

结构

High Level Graph Rewriting

高级图形重写

Machine Learning Based Automated Optimizer

基于机器学习的自动优化器

Optimized Computational Graph

优化计算图

Computational Graph

计算图形

Hardware-Aware Optimization Primitives

硬件感知优化原语

Declarative

宣言的

Tensor Expressions

张量表达式

Optimized Low Level Loop Program

优化的低级循环程序

Accelerator Backend LLVM IR CUDA/Metal/OpenCL

加速器后端LLVM IR CUDA/Metal/OpenCL

Deployable Module

可部署模块

Operator-level Optimization and Code Generation

操作员级优化和代码生成

Section 3

第3节

Section 4

第4节

Section 5

第5节

Figure 2: System overview of TVM.The current stack supports descriptions from many deep learning frame-works and exchange formats, such as CoreML and ONNX, to target major CPU, GPU and specialized ac-celerators.

图TVM的系统概述。当前的堆栈支持许多深度学习框架和交换格式的描述，如CoreML和ONNX，以主要的CPU、GPU和专门的交流加速器为目标。

guage;execution details are unspecified.TVM identifies a collection of possible code optimizations for a given hardware target's operators.Possible optimizations form a large space, so we use an ML-based cost model to find optimized operators.Finally, the system packs the gen-erated code into a deployable module.

guage未指定执行细节。TVM为给定的硬件目标的操作符识别可能的代码优化的集合。可能的优化形成了一个很大的空间，所以我们使用基于最大似然的成本模型来寻找优化的运算符。最后，系统将生成的代码打包成一个可部署的模块。

End-User Example.In a few lines of code, a user can take a model from existing deep learning frameworks and call the TVM API to get a deployable module:

最终用户示例。在几行代码中，用户可以从现有的深度学习框架中获取一个模型，并调用TVM API来获得一个可部署的模块:

import tvm as t

将tvm导入为t

# Use keras framework as example, import model

#以keras框架为例，导入模型

graph, params = t.frontend.from\_keras(keras\_model) target = t.target.cuda()

graph，params = t . front . from \_ keras(keras \_ model)target = t . target . cuda()

graph, lib, params = t.compiler.build(graph, target, params)

graph，lib，params = t.compiler.build(graph，target，params)

This compiled runtime module contains three compo-nents: the final optimized computational graph (graph), generated operators (lib), and module parame-ters (params).These components can then be used to deploy the model to the target back-end:

这个编译的运行时模块包含三个组成部分:最终优化的计算图(graph)、生成的运算符(lib)和模块参数(params)。这些组件可用于将模型部署到目标后端:

import tvm.runtime as t

将tvm.runtime导入为t

module = runtime.create(graph, lib, t.cuda(0)) module.set\_input(\*\*params) module.run(data=data\_array)

module = runtime.create(graph，lib，t . cuda(0))module . set \_ input(\* \* params)module . run(data = data \_ array)

output = tvm.nd.empty(out\_shape, ctx=t.cuda(0)) module.get\_output(0, output)

output = tvm.nd.empty(out\_shape，CTX = t . cuda(0))module . get \_ output(0，output)

TVM supports multiple deployment back-ends in lan-guages such as C++, Java and Python.The rest of this paper describes TVM's architecture and how a system programmer can extend it to support new back-ends.

TVM支持C++、Java、Python等语言的多个部署后端。本文的其余部分描述了TVM的体系结构以及系统程序员如何扩展它来支持新的后端。

3 Optimizing Computational Graphs

3优化计算图

Computational graphs are a common way to represent programs in DL frameworks [3, 4, 7, 9].Figure 3 shows

计算图是在DL框架中表示程序的常用方式[3，4，7，9]。图3显示了

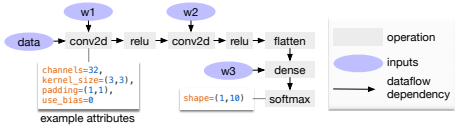


Figure 3: Example computational graph of a two-layer convolutional neural network.Each node in the graph represents an operation that consumes one or more ten-sors and produces one or more tensors.Tensor operations can be parameterized by attributes to configure their be-havior (e.g., padding or strides).

图3:两层卷积神经网络的示例计算图。图中的每个节点代表一个消耗一个或多个十元并产生一个或多个张量的运算。张量运算可以通过属性来参数化，以配置它们的行为(例如，填充或步幅)。

an example computational graph representation of a two-layer convolutional neural network.The main differ-ence between this high-level representation and a low-level compiler intermediate representation (IR), such as LLVM, is that the intermediate data items are large, multi-dimensional tensors.Computational graphs pro-vide a global view of operators, but they avoid specifying how each operator must be implemented.Like LLVM IRs, a computational graph can be transformed into func-tionally equivalent graphs to apply optimizations.We also take advantage of shape specificity in common DL workloads to optimize for a fixed set of input shapes.

两层卷积神经网络的示例计算图表示。这种高级表示与低级编译器中间表示(如LLVM)之间的主要区别在于中间数据项是大型多维张量。计算图提供了操作符的全局视图，但是它们避免了指定每个操作符必须如何实现。像LLVM IRs一样，计算图可以转换成功能等价的图来应用优化。我们还利用常见下行链路工作负载中的形状特异性，针对一组固定的输入形状进行优化。

TVM exploits a computational graph representation to apply high-level optimizations: a node represents an op-eration on tensors or program inputs, and edges represent data dependencies between operations.It implements many graph-level optimizations, including: operator fu-sion, which fuses multiple small operations together;constant-folding, which pre-computes graph parts that can be determined statically, saving execution costs;a static memory planning pass, which pre-allocates mem-ory to hold each intermediate tensor;and data layout transformations, which transform internal data layouts into back-end-friendly forms.We now discuss operator fusion and the data layout transformation.

TVM利用计算图表示来应用高级优化:节点表示张量或程序输入上的运算，边表示运算之间的数据依赖关系。它实现了很多图级优化，包括:运算符fusion，将多个小操作融合在一起；常数折叠，预先计算可以静态确定的图形部分，节省执行成本；静态存储器规划通道，其预先分配存储器来保存每个中间张量；和数据布局转换，它们将内部数据布局转换成后端友好的形式。我们现在讨论算子融合和数据布局转换。

Operator Fusion.Operator fusion combines multiple operators into a single kernel without saving the interme-diate results in memory.This optimization can greatly reduce execution time, particularly in GPUs and spe-cialized accelerators.Specifically, we recognize four categories of graph operators: (1) injective (one-to-one map, e.g., add), (2) reduction (e.g., sum), (3) complex-out-fusable (can fuse element-wise map to output, e.g., conv2d), and (4) opaque (cannot be fused, e.g., sort).We provide generic rules to fuse these operators, as follows.Multiple injective operators can be fused into another in-jective operator.A reduction operator can be fused with input injective operators (e.g., fuse scale and sum).Op-erators such as conv2d are complex-out-fusable, and we

运营商融合。运算符融合将多个运算符组合成一个内核，而不会将中间结果保存在内存中。这种优化可以大大减少执行时间，特别是在GPU和专用加速器中。具体来说，我们识别出四类图运算符:(1)内射(一对一映射，例如加法)，(2)归约(例如求和)，(3)复出可融合(可以将元素式映射融合到输出，例如conv2d)，以及(4)不透明(不能融合，例如排序)。我们提供通用规则来融合这些操作符，如下所示。多个内射算子可以融合成另一个内射算子。约简算子可以与输入内射算子融合(如融合尺度和和)。像conv2d这样的运放是复杂的，不可融合的，我们

3

3

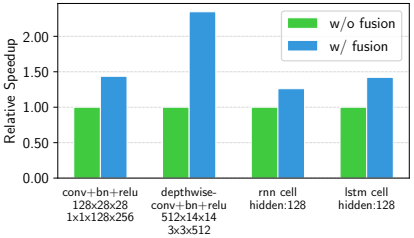


Figure 4: Performance comparison between fused and non-fused operations.TVM generates both operations.Tested on NVIDIA Titan X.

图4:融合和非融合操作之间的性能比较。TVM生成两个操作。在英伟达泰坦x上测试过。

can fuse element-wise operators to its output.We can apply these rules to transform the computational graph into a fused version.Figure 4 demonstrates the impact of this optimization on different workloads.We find that fused operators generate up to a 1.2× to 2× speedup by reducing memory accesses.

可以将元素操作符融合到它的输出中。我们可以应用这些规则将计算图转换成融合版本。图4展示了这种优化对不同工作负载的影响。我们发现融合算子通过减少内存访问产生了高达1.2倍到2倍的加速。

Data Layout Transformation.There are multiple ways to store a given tensor in the computational graph.The most common data layout choices are column major and row major.In practice, we may prefer to use even more complicated data layouts.For instance, a DL ac-celerator might exploit 4×4 matrix operations, requiring data to be tiled into 4×4 chunks to optimize for access locality.

数据布局转换。有多种方法可以在计算图中存储给定的张量。最常见的数据布局选择是主要列和主要行。实际上，我们可能更喜欢使用更复杂的数据布局。例如，DL加速程序可能利用4×4矩阵运算，要求将数据平铺成4×4块，以优化访问局部性。

Data layout optimization converts a computational graph into one that can use better internal data layouts for execution on the target hardware.It starts by spec-ifying the preferred data layout for each operator given the constraints dictated by memory hierarchies.We then perform the proper layout transformation between a pro-ducer and a consumer if their preferred data layouts do not match.

数据布局优化将计算图形转换为可以使用更好的内部数据布局在目标硬件上执行的图形。它从给定内存层次结构所规定的约束条件，为每个运算符指定首选数据布局开始。然后，如果生产者和消费者偏好的数据布局不匹配，我们就在他们之间执行适当的布局转换。

While high-level graph optimizations can greatly im-prove the efficiency of DL workloads, they are only as effective as what the operator library provides.Cur-rently, the few DL frameworks that support operator fu-sion require the operator library to provide an implemen-tation of the fused patterns.With more network opera-tors introduced on a regular basis, the number of possible fused kernels can grow dramatically.This approach is no longer sustainable when targeting an increasing num-ber of hardware back-ends since the required number of fused pattern implementations grows combinatorially with the number of data layouts, data types, and accel-erator intrinsics that must be supported.It is not feasi-ble to handcraft operator kernels for the various opera-tions desired by a program and for each back-end.To

虽然高级图形优化可以极大地提高DL工作负载的效率，但它们的效果仅取决于运算符库提供的内容。目前，少数支持操作符融合的数字语言框架要求操作符库提供融合模式的实现。随着更多的网络操作符被定期引入，可能的融合内核的数量会急剧增长。当针对越来越多的硬件后端时，这种方法不再是可持续的，因为所需的融合模式实现的数量随着必须支持的数据布局、数据类型和加速器本质的数量而增长。为程序和每个后端所需的各种操作手工制作操作符内核是不可行的。到

for y in range(1024):

对于y范围(1024):

for x in range(1024): C[y][x] = 0

对于(1024)范围内的x:C[y][x]= 0

for k in range(1024):

对于范围(1024)内的k:

C[y][x] += A[k][y] \* B[k][x]

C[y][x] += A[k][y] \* B[k][x]

for yo in range(128):

对于范围内的yo(128):

for xo in range(128):

对于范围(128)内的xo:

C[yo\*8:yo\*8+8][xo\*8:xo\*8+8] = 0 for ko in range(128):

C[yo\*8:yo\*8+8][xo\*8:xo\*8+8] = 0，适用于范围(128)内的ko:

for yi in range(8):

对于范围(8)中的yi:

for xi in range(8):

对于范围(8)内的xi:

for ki in range(8):

对于ki范围(8):

C[yo\*8+yi][xo\*8+xi] +=

C[yo\*8+yi][xo\*8+xi] +=

A[ko\*8+ki][yo\*8+yi] \* B[ko\*8+ki][xo\*8+xi]

a[ko \* 8+ki][yo \* 8+yi]\* B[ko \* 8+ki][XO \* 8+Xi]

inp\_buffer AL[8][8], BL[8][8] acc\_buffer CL[8][8] for yo in range(128):

范围(128)内yo的inp\_buffer AL[8][8]，BL[8][8] acc\_buffer CL[8][8]:

for xo in range(128): vdla.fill\_zero(CL)

对于xo范围(128): vdla.fill\_zero(CL)

for ko in range(128):

对于范围内的ko(128):

vdla.dma\_copy2d(AL, A[ko\*8:ko\*8+8][yo\*8:yo\*8+8]) vdla.dma\_copy2d(BL, B[ko\*8:ko\*8+8][xo\*8:xo\*8+8]) vdla.fused\_gemm8x8\_add(CL, AL, BL) vdla.dma\_copy2d(C[yo\*8:yo\*8+8,xo\*8:xo\*8+8], CL)

vdla.dma\_copy2d(AL，A[ko \* 8:ko \* 8+8][yo \* 8:yo \* 8+8])vdla . DMA \_ copy 2d(BL，B[ko \* 8:ko \* 8+8][xo\*8:xo\*8+8])vdla . fused \_ gemm 8 x8 \_ add(CL，AL，BL)vdla . DMA \_ copy 2d(C[yo \* 8:yo \* 8+8，XO \* 8:XO \* 8+8]，CL)

+ Cache Data on Accelerator Special Buffer

+在加速器专用缓冲区上缓存数据

A = t.placeholder((1024, 1024)) B = t.placeholder((1024, 1024)) k = t.reduce\_axis((0, 1024))

A = t.placeholder((1024，1024)) B = t.placeholder((1024，1024)) k = t.reduce\_axis((0，1024))

C = t.compute((1024, 1024), lambda y, x: t.sum(A[k, y] \* B[k, x], axis=k)) s = t.create\_schedule(C.op)

C = t.compute((1024，1024)，lambda y，x: t.sum(A[k，y] \* B[k，x]，axis = k))s = t . create \_ schedule(c . op)

schedule schedule transformation corresponding low-level code

进度计划转换对应的低级代码

+ Map to Accelerator Tensor Instructions

+映射到加速器张量指令

CL = s.cache\_write(C, vdla.acc\_buffer) AL = s.cache\_read(A, vdla.inp\_buffer) # additional schedule steps omitted …

CL = s.cache\_write(C，vdla . ACC \_ buffer)AL = s . cache \_ read(A，vdla.inp\_buffer) #省略额外的调度步骤…

s[CL].tensorize(yi, vdla.gemm8x8)

s[CL]。tensorize(yi，vdla.gemm8x8)

+ Loop Tiling

+循环平铺

yo, xo, ko, yi, xi, ki = s[C].tile(y, x, k, 8, 8, 8)

唷，xo，ko，yi，xi，ki = s[C]。平铺(y，x，k，8，8，8)

Figure 5: Example schedule transformations that opti-mize a matrix multiplication on a specialized accelerator.

图5:在专用加速器上优化矩阵乘法的示例调度转换。

this end, we next propose a code generation approach that can generate various possible implementations for a given model's operators.

为此，我们接下来提出一种代码生成方法，可以为给定模型的操作符生成各种可能的实现。

4 Generating Tensor Operations

4生成张量运算

TVM produces efficient code for each operator by gen-erating many valid implementations on each hardware back-end and choosing an optimized implementation.This process builds on Halide's idea of decoupling de-scriptions from computation rules (or schedule optimiza-tions) [32] and extends it to support new optimizations (nested parallelism, tensorization, and latency hiding) and a wide array of hardware back-ends.We now high-light TVM-specific features.

TVM通过在每个硬件后端生成许多有效的实现并选择优化的实现，为每个运营商生成高效的代码。这个过程建立在卤化物从计算规则(或调度优化)[32]中分离描述的思想上，并扩展它以支持新的优化(嵌套并行、张量化和延迟隐藏)和广泛的硬件后端。我们现在高光TVM特有的功能。

4.1 Tensor Expression and Schedule Space

4.1张量表达式和时间表空间

We introduce a tensor expression language to support au-tomatic code generation.Unlike high-level computation graph representations, where the implementation of ten-sor operations is opaque, each operation is described in

我们引入张量表达式语言来支持自动代码生成。与高级计算图形表示不同，在高级计算图形表示中，十进制操作的实现是不透明的，每个操作都在

4

4

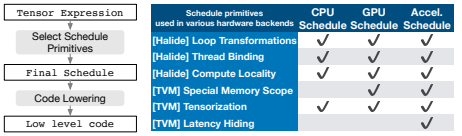


Figure 6: TVM schedule lowering and code generation process.The table lists existing Halide and novel TVM scheduling primitives being used to optimize schedules for CPUs, GPUs and accelerator back-ends.Tensoriza-tion is essential for accelerators, but it can also be used for CPUs and GPUs.Special memory-scope enables memory reuse in GPUs and explicit management of on-chip memory in accelerators.Latency hiding is specific to TPU-like accelerators.

图6: TVM计划降低和代码生成过程。该表列出了现有的卤化物和新的TVM调度原语，用于优化中央处理器、图形处理器和加速器后端的调度。张量化对于加速器来说是必不可少的，但它也可以用于CPU和GPU。特殊的内存范围支持GPU中的内存重用和加速器中片内内存的显式管理。潜伏隐藏是TPU加速器特有的。

an index formula expression language.The following code shows an example tensor expression to compute transposed matrix multiplication:

索引公式表达式语言。下面的代码显示了一个计算转置矩阵乘法的张量表达式示例:

m, n, h = t.var('m'), t.var('n'), t.var('h') A = t.placeholder((m, h), name='A') B = t.placeholder((n, h), name='B') k = t.reduce\_axis((0, h), name='k') C = t.compute((m, n), lambda y, x:

m，n，h = t.var('m ')，t.var('n ')，t.var('h') A = t.placeholder((m，h)，name='A') B = t.placeholder((n，h)，name='B') k = t.reduce\_axis((0，h)，name='k') C = t.compute((m，n)，lambda y，x:

t.sum(A[k, y] B[k, x], axis=k)) result shape

总和(A[k，y] B[k，x]，轴=k))结果形状

computing rule

计算规则

Each compute operation specifies both the shape of the output tensor and an expression describing how to compute each element of it.Our tensor expression language supports common arithmetic and math oper-ations and covers common DL operator patterns.The language does not specify the loop structure and many other execution details, and it provides flexibility for adding hardware-aware optimizations for various back-ends.Adopting the decoupled compute/schedule princi-ple from Halide [32], we use a schedule to denote a spe-cific mapping from a tensor expression to low-level code.Many possible schedules can perform this function.

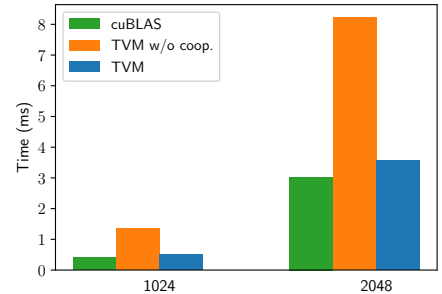
每个计算操作指定输出张量的形状和描述如何计算它的每个元素的表达式。我们的张量表达式语言支持常见的算术和数学运算，并涵盖常见的DL运算符模式。该语言没有指定循环结构和许多其他执行细节，它为各种后端添加硬件感知优化提供了灵活性。采用从卤化物[32]解耦的计算/调度原理，我们使用调度来表示从张量表达式到低级代码的特定映射。许多可能的时间表可以执行此功能。

We build a schedule by incrementally applying basic transformations (schedule primitives) that preserve the program's logical equivalence.Figure 5 shows an ex-ample of scheduling matrix multiplication on a special-ized accelerator.Internally, TVM uses a data structure to keep track of the loop structure and other information as we apply schedule transformations.This information can then help generate low-level code for a given final schedule.

我们通过增量应用基本转换(调度原语)来构建一个调度，以保持程序的逻辑等价性。图5显示了一个在特殊加速器上调度矩阵乘法的例子。在内部，当我们应用时间表转换时，TVM使用数据结构来跟踪循环结构和其他信息。这些信息有助于为给定的最终计划生成低级代码。

Our tensor expression takes cues from Halide [32], Darkroom [17], and TACO [23].Its primary enhance-ments include support for the new schedule optimiza-tions discussed below.To achieve high performance on many back-ends, we must support enough schedule primitives to cover a diverse set of optimizations on dif-ferent hardware back-ends.Figure 6 summarizes the operation code generation process and schedule primi-

我们的张量表达式从卤化物[32]、暗室[17]和TACO [23]中获得线索。它的主要改进包括支持下面讨论的新的计划优化。为了在许多后端上实现高性能，我们必须支持足够的调度原语，以覆盖不同硬件后端上的一组不同的优化。图6总结了操作代码生成过程和时间表primi-



Matrix Size

矩阵尺度

Figure 7: Performance comparison between TVM with and without cooperative shared memory fetching on ma-trix multiplication workloads.Tested on an NVIDIA Ti-tan X.

图7:在矩阵乘法工作负载上，有和没有协作共享内存获取的TVM之间的性能比较。在NVIDIA Ti-tan X上测试。

tives that TVM supports.We reuse helpful primitives and the low-level loop program AST from Halide, and we introduce new primitives to optimize GPU and ac-celerator performance.The new primitives are neces-sary to achieve optimal GPU performance and essen-tial for accelerators.CPU, GPU, TPU-like accelerators are three important types of hardware for deep learning.This section describes new optimization primitives for CPUs, GPUs and TPU-like accelerators, while section 5 explains how to automatically derive efficient schedules.

TVM支持的特权。我们重用有用的原语和卤化物的低级循环程序AST，并引入新的原语来优化GPU和加速器性能。新原语对于实现最佳图形处理器性能是必要的，对于加速器也是必要的。CPU、GPU、类TPU加速器是深度学习的三种重要硬件类型。这一节描述了用于中央处理器、图形处理器和类似TPU的加速器的新的优化原语，而第五节解释了如何自动获得高效的调度。

4.2 Nested Parallelism with Cooperation

4.2嵌套并行与协作

Parallelism is key to improving the efficiency of compute-intensive kernels in DL workloads.Modern GPUs offer massive parallelism, requiring us to bake par-allel patterns into schedule transformations.Most exist-ing solutions adopt a model called nested parallelism, a form of fork-join.This model requires a parallel sched-ule primitive to parallelize a data parallel task;each task can be further recursively subdivided into subtasks to ex-ploit the target architecture's multi-level thread hierarchy (e.g., thread groups in GPU).We call this model shared-nothing nested parallelism because one working thread cannot look at the data of its sibling within the same par-allel computation stage.

并行性是提高DL工作负载中计算密集型内核效率的关键。现代图形处理器提供了巨大的并行性，要求我们将并行模式烘焙成调度转换。大多数现有解决方案采用一种称为嵌套并行的模型，这是一种分叉连接的形式。这个模型需要一个并行sched-ule原语来并行化一个数据并行任务；每个任务可以进一步递归地细分为子任务，以探索目标体系结构的多级线程层次结构(例如，GPU中的线程组)。我们称这个模型为无共享嵌套并行，因为一个工作线程不能在同一个并行计算阶段查看其兄弟线程的数据。

An alternative to the shared-nothing approach is to fetch data cooperatively.Specifically, groups of threads can cooperatively fetch the data they all need and place it into a shared memory space.This optimization can take advantage of the GPU memory hierarchy and en-

无共享方法的一个替代方法是协作获取数据。具体来说，线程组可以协作获取它们都需要的数据，并将其放入共享内存空间。这种优化可以利用图形处理器内存层次结构和

Halide recently added shared memory support but without general memory scope for accelerators.

卤化物最近增加了共享内存支持，但没有加速器的一般内存范围。

5

5

able data reuse across threads through shared memory regions.TVM supports this well-known GPU optimiza-tion using a schedule primitive to achieve optimal per-formance.The following GPU code example optimizes matrix multiplication.

能够通过共享内存区域跨线程重用数据。TVM支持这种众所周知的图形处理器优化，使用调度原语来实现最佳性能。以下GPU代码示例优化矩阵乘法。

Barrier inserted automatically by compiler

编译器自动插入的障碍

All threads cooperatively load AS and BS in different parallel patterns

所有线程以不同的并行模式协同加载AS和BS

for thread\_group (by, bx) in cross(64, 64): for thread\_item (ty, tx) in cross(2, 2): local CL[8][8] = 0 shared AS[2][8], BS[2][8] for k in range(1024):

对于cross(64，64)中的thread\_group (by，bx):对于cross(2，2)中的thread\_item (ty，tx):范围(1024)中k的局部CL[8][8] = 0共享AS[2][8]，BS[2][8]:

for i in range(4):

对于范围(4)中的I:

AS[ty][i\*4+tx] = A[k][by\*64+ty\*8+i\*4+tx] for each i in 0..4:

0中每个I的AS[ty][I \* 4+tx]= A[k][乘\*64+ty\*8+i\*4+tx]..4:

BS[ty][i\*4+tx] = B[k][bx\*64+ty\*8+i\*4+tx]

BS[ty][I \* 4+tx]= B[k][bx \* 64+ty \* 8+I \* 4+tx]

memory\_barrier\_among\_threads()

memory \_ barrier \_ among \_ threads()

for yi in range(8):

对于范围(8)中的yi:

for xi in range(8):

对于范围(8)内的xi:

CL[yi][xi] += AS[yi] \* BS[xi]

CL[yi][xi] += AS[yi] \* BS[xi]

for yi in range(8):

对于范围(8)中的yi:

for xi in range(8):

对于范围(8)内的xi:

C[yo\*8+yi][xo\*8+xi] = CL[yi][xi]

C[yo\*8+yi][xo\*8+xi] = CL[yi][xi]

w, x = t.placeholder((8, 8)), t.placeholder((8, 8)) k = t.reduce\_axis((0, 8)) y = t.compute((8, 8), lambda i, j: t.sum(w[i, k] \* x[j, k], axis=k)) def gemm\_intrin\_lower(inputs, outputs): ww\_ptr = inputs[0].access\_ptr("r") xx\_ptr = inputs[1].access\_ptr("r") zz\_ptr = outputs[0].access\_ptr("w")

w，x = t.placeholder((8，8))，t.placeholder((8，8)) k = t.reduce\_axis((0，8)) y = t.compute((8，8)，lambda i，j: t.sum(w[i，k] \* x[j，k]，axis = k))def gemm \_ intrin \_ low(inputs，outputs): ww\_ptr = inputs[0]。access \_ ptr(" r ")xx \_ ptr = inputs[1]。access \_ ptr(" r ")ZZ \_ ptr = outputs[0]。access\_ptr("w ")

compute = t.hardware\_intrin("gemm8x8", ww\_ptr, xx\_ptr, zz\_ptr) reset = t.hardware\_intrin("fill\_zero", zz\_ptr)

compute = t . hardware \_ intrin(" gemm 8 x8 "，ww\_ptr，xx\_ptr，zz\_ptr)reset = t . hardware \_ intrin(" fill \_ zero "，ZZ \_ ptr)

update = t.hardware\_intrin("fuse\_gemm8x8\_add", ww\_ptr, xx\_ptr, zz\_ptr) return compute, reset, update

update = t . hardware \_ intrin(" fuse \_ gemm 8 x8 \_ add "，ww\_ptr，xx\_ptr，zz\_ptr)返回计算、复位、更新

declare behavior

宣布行为

lowering rule to generate hardware intrinsics to carry out the computation

降低规则以生成硬件内变量来执行计算

gemm8x8 = t.decl\_tensor\_intrin(y.op, gemm\_intrin\_lower)

gemm 8 x8 = t . decl \_ tensor \_ intrin(y . op，gemm \_ intrin \_ lower)

Additionally, we introduce a tensorize schedule primi-tive to replace a unit of computation with the correspond-ing intrinsics.The compiler matches the computation pattern with a hardware declaration and lowers it to the corresponding hardware intrinsic.

此外，我们引入一个张量化的时间表，用相应的内蕴来代替一个计算单位。编译器将计算模式与硬件声明相匹配，并将其降低到相应的硬件固有值。

Tensorization decouples the schedule from specific hardware primitives, making it easy to extend TVM to support new hardware architectures.The generated code of tensorized schedules aligns with practices in high-performance computing: break complex operations into a sequence of micro-kernel calls.We can also use the tensorize primitive to take advantage of handcrafted micro-kernels, which can be beneficial in some plat-forms.For example, we implement ultra low precision operators for mobile CPUs that operate on data types that are one- or two-bits wide by leveraging a bit-serial matrix vector multiplication micro-kernel.This micro-kernel accumulates results into progressively larger data types to minimize the memory footprint.Presenting the micro-kernel as a tensor intrinsic to TVM yields up to a 1.5× speedup over the non-tensorized version.

张量化将调度从特定的硬件原语中分离出来，使得扩展TVM以支持新的硬件架构变得容易。生成的张量化调度代码符合高性能计算的实践:将复杂的操作分解成一系列微内核调用。我们也可以使用张量原语来利用手工制作的微内核，这在某些平台上是有益的。例如，我们通过利用位串行矩阵向量乘法微内核，为在一位或两位宽的数据类型上运行的移动CPU实现超低精度运算符。这个微内核将结果累积成越来越大的数据类型，以最小化内存占用。将微内核表示为TVM固有的张量，比非张量化版本的速度提高了1.5倍。

Figure 7 demonstrates the impact of this optimiza-tion.We introduce the concept of memory scopes to the schedule space so that a compute stage (AS and BS in the code) can be marked as shared.Without explicit memory scopes, automatic scope inference will mark compute stages as thread-local.The shared task must compute the dependencies of all working threads in the group.Additionally, memory synchronization barriers must be properly inserted to guarantee that shared loaded data is visible to consumers.Finally, in addition to being use-ful to GPUs, memory scopes let us tag special memory buffers and create special lowering rules when targeting specialized DL accelerators.

图7展示了这种优化的影响。我们将内存范围的概念引入到调度空间中，这样计算阶段(代码中的AS和BS)就可以被标记为共享。如果没有显式内存范围，自动范围推断会将计算阶段标记为线程本地。共享任务必须计算组中所有工作线程的相关性。此外，必须正确插入内存同步屏障，以确保共享的加载数据对消费者可见。最后，除了对GPU有用之外，内存范围还允许我们标记特殊的内存缓冲区，并在针对特殊的DL加速器时创建特殊的降低规则。

4.3 Tensorization

4.3张量化

DL workloads have high arithmetic intensity, which can typically be decomposed into tensor operators like matrix-matrix multiplication or 1D convolution.These natural decompositions have led to the recent trend of adding tensor compute primitives [1, 12, 21].These new primitives create both opportunities and challenges for schedule-based compilation;while using them can improve performance, the compilation framework must seamlessly integrate them.We dub this tensorization: it is analogous to vectorization for SIMD architectures but has significant differences.Instruction inputs are multi-dimensional, with fixed or variable lengths, and each has different data layouts.More importantly, we cannot sup-port a fixed set of primitives since new accelerators are emerging with their own variations of tensor instructions.We therefore need an extensible solution.

DL工作负载具有很高的算术强度，通常可以分解为张量算子，如矩阵-矩阵乘法或1D卷积。这些自然分解导致了最近增加张量计算基元的趋势[1，12，21]。这些新的原语为基于时间表的编译带来了机遇和挑战；虽然使用它们可以提高性能，但是编译框架必须无缝地集成它们。我们称之为张量化:它类似于SIMD建筑的矢量化，但有显著的区别。指令输入是多维的，具有固定或可变的长度，并且每个都有不同的数据布局。更重要的是，我们不能支持一组固定的原语，因为新的加速器正在出现，它们有自己的张量指令变体。因此，我们需要一个可扩展的解决方案。

We make tensorization extensible by separating the target hardware intrinsic from the schedule with a mech-anism for tensor-intrinsic declaration.We use the same tensor expression language to declare both the behavior of each new hardware intrinsic and the lowering rule as-sociated with it.The following code shows how to de-clare an 8×8 tensor hardware intrinsic.

我们通过用张量内在声明的机制将目标硬件内在从调度中分离出来，使张量化可扩展。我们使用相同的张量表达式语言来声明每个新硬件的内在行为和与之相关的降低规则。下面的代码展示了如何对一个8×8的张量硬件内在进行去声明。

4.4 Explicit Memory Latency Hiding

4.4显式内存延迟隐藏

Latency hiding refers to the process of overlapping mem-ory operations with computation to maximize utilization of memory and compute resources.It requires different strategies depending on the target hardware back-end.On CPUs, memory latency hiding is achieved implic-itly with simultaneous multithreading [14] or hardware prefetching [10, 20].GPUs rely on rapid context switch-ing of many warps of threads [44].In contrast, special-ized DL accelerators such as the TPU [21] usually favor leaner control with a decoupled access-execute (DAE) architecture [35] and offload the problem of fine-grained synchronization to software.

延迟隐藏是指将内存操作与计算重叠的过程，以最大限度地利用内存和计算资源。根据目标硬件后端，它需要不同的策略。在中央处理器上，内存延迟隐藏是通过同时多线程[14]或硬件预取[10，20]实现的。图形处理器依赖于许多线程的快速上下文切换[44]。相比之下，TPU [21]等专用数据链路加速器通常更倾向于采用解耦的访问执行(DAE)架构的精简控制[35]，并将细粒度同步的问题转移给软件。

Figure 9 shows a DAE hardware pipeline that reduces runtime latency.Compared to a monolithic hardware de-sign, the pipeline can hide most memory access over-heads and almost fully utilize compute resources.To achieve higher utilization, the instruction stream must be augmented with fine-grained synchronization operations.Without them, dependencies cannot be enforced, leading to erroneous execution.Consequently, DAE hardware pipelines require fine-grained dependence enqueuing/d-equeuing operations between the pipeline stages to guar-

图9显示了减少运行时延迟的DAE硬件管道。与单片硬件设计相比，流水线可以隐藏大部分内存访问并几乎完全利用计算资源。为了获得更高的利用率，指令流必须用细粒度的同步操作来扩充。没有它们，依赖关系就无法执行，从而导致错误的执行。因此，DAE硬件管道需要在管道阶段之间进行细粒度的相关性排队/数据请求操作，以确保

6

6

vthread 0 ld ex ld … ld ex

vthread 0 ld ex ld … ld ex

Input: High-level Threaded Program

输入:高级线程程序

vthread 1 ld ex ld … ld ex

vthread 1 ld ex ld … ld ex

for vthread tx in range(2): acc\_buffer CL[8] inp\_buffer AL[8] for k in range(128):

对于范围(2)中的vthread tx:范围(128)中k的acc\_buffer CL[8] inp\_buffer AL[8]:

ld.dma\_copy2d(AL, AL[k][tx\*8:tx\*8+8]) ex.accumulate(AL, CL)

ld.dma\_copy2d(AL，AL[k][tx\*8:tx\*8+8])

read after write (RAW) dependence read after write (RAW) dependence push RAW dependence push WAR dependence pop RAW dependence pop WAR dependence

写后读(原始)依赖写后读(原始)依赖推原始依赖推战争依赖弹出原始依赖弹出战争依赖

vthread 0

vthread 0

vthread 1 barrier ld ex ld … ld ex ld ex ld … ld ex barrier

vthread 1屏障ld ex ld … ld ex ld ex ld … ld ex屏障

Inject Synchronization Instructions

注入同步指令

for vthread tx in range(2): acc\_buffer CL[8] inp\_buffer AL[8] ex.push\_dep\_to(ld) for k in range(128):

对于范围(2)中的vthread tx:范围(128)中k的ACC \_ buffer CL[8]InP \_ buffer AL[8]ex . push \_ dep \_ to(LD):

ld.pop\_dep\_from(ex)

ld.pop\_dep\_from(ex)

ld.dma\_copy2d(AL, AL[k][tx\*8:tx\*8+8]) ld.push\_dep\_to(ex) ex.pop\_dep\_from(ld)

ld.dma\_copy2d(AL，AL[k][tx \* 8:tx \* 8+8])LD . push \_ dep \_ to(ex)ex . pop \_ dep \_ from(LD)

ex.accumulate(AL, CL) ex.push\_dep\_to(ld) ld.pop\_dep\_from(ex)

ex .累加(AL，CL)ex . push \_ dep \_ to(LD)LD . pop \_ dep \_ from(ex)

l d

l d

l d

l d

e x

e x

e x

e x

l d

l d

l

l

d … e x e x

d … e x e x

Final Single Instruction Stream

最终单指令流

acc\_buffer CL[2][8] inp\_buffer AL[2][8] ex.push\_dep\_to(ld) ex.push\_dep\_to(ld) for k in range(128):

ACC \_ buffer CL[2][8]InP \_ buffer AL[2][8]ex . push \_ dep \_ to(LD)ex . push \_ dep \_ to(LD)针对范围(128)中的k:

ld.pop\_dep\_from(ex)

ld.pop\_dep\_from(ex)

ld.dma\_copy2d(AL[0], AL[k][0:8]) ld.push\_dep\_to(ex) ld.pop\_dep\_from(ex)

ld.dma\_copy2d(AL[0]，AL[k][0:8])LD . push \_ dep \_ to(ex)LD . pop \_ dep \_ from(ex)

ld.dma\_copy2d(AL[1], AL[k][8:16]) ld.push\_dep\_to(ex) ex.pop\_dep\_from(ld)

ld.dma\_copy2d(AL[1]，AL[k][8:16])LD . push \_ dep \_ to(ex)ex . pop \_ dep \_ from(LD)

ex.accumulate(AL[0], CL[0]) ex.push\_dep\_to(ld) ex.pop\_dep\_from(ld)

ex .累加(AL[0]，CL[0])ex . push \_ dep \_ to(LD)ex . pop \_ dep \_ from(LD)

ex.accumulate(AL[1], CL[1]) ex.push\_dep\_to(ld) ld.pop\_dep\_from(ex) ld.pop\_dep\_from(ex)

ex .累加(AL[1]，CL[1])ex . push \_ dep \_ to(LD)LD . pop \_ dep \_ from(ex)LD . pop \_ dep \_ from(ex)

Figure 8: TVM virtual thread lowering transforms a virtual thread-parallel program to a single instruction stream;the stream contains explicit low-level synchronizations that the hardware can interpret to recover the pipeline parallelism required to hide memory access latency.

图8: TVM虚拟线程降低将一个虚拟线程并行程序转换成单个指令流；该流包含硬件可以解释的显式低级同步，以恢复隐藏内存访问延迟所需的流水线并行性。

ld 0

ld 0

ex 0

ex 0

ld 1

ld 1

ex 1

ex 1

ld 2

ld 2

ex 2

ex 2

ld 3

ld 3

ex 3

ex 3

t

t

Monolithic Pipeline

整体管道

ld.perform\_action(ld0) ex.perform\_action(ex0) ld.perform\_action(ld1) ex.perform\_action(ex1) ...

LD . perform \_ action(ld0)ex . perform \_ action(ex0)LD . perform \_ action(ld1)ex . perform \_ action(ex1)...

Instruction Stream

指令流

ld 0

ld 0

ex 0

ex 0

ld 1

ld 1

ex 1

ex 1

ld 2

ld 2

ld 3

ld 3

ex 2

ex 2

ex 3

ex 3

execution savings

执行节省

Decoupled Access-Execute Pipeline

解耦的访问执行管道

write after read (WAR) dependence

读后写(WAR)依赖性

read after write (RAW) dependence

写后读(原始)相关性

ld.perform\_action(ld0) ld.push\_dep\_to(ex) ld.perform\_action(ld1) ld.push\_dep\_to(ex) ex.pop\_dep\_from(ld) ex.perform\_action(ex0) ex.push\_dep\_to(ld) ex.pop\_dep\_from(ld) ex.perform\_action(ex1) ex.push\_dep\_to(ld) ld.pop\_dep\_from(ex) ld.perform\_action(ld2) ...

LD . perform \_ action(ld0)LD . push \_ dep \_ to(ex)LD . perform \_ action(ld1)LD . push \_ dep \_ to(ex)ex . pop \_ dep \_ from(LD)ex . perform \_ action(ex0)ex . push \_ dep \_ to(LD)ex . pop \_ dep \_ from(LD)ex . perform \_ action(ex1)ex . push \_ dep \_ to(LD)LD . pop \_ dep \_ from(ex)LD . perform \_ action(ld2)...

Figure 9: Decoupled Access-Execute in hardware hides most memory access latency by allowing memory and computation to overlap.Execution correctness is en-forced by low-level synchronization in the form of de-pendence token enqueueing/dequeuing actions, which the compiler stack must insert in the instruction stream.

图9:解耦访问——通过允许内存和计算重叠，硬件中的执行隐藏了大部分内存访问延迟。执行正确性是由低级同步以独立令牌入队/出队动作的形式强制的，编译器堆栈必须将这些动作插入指令流中。

antee correct execution, as shown in Figure 9's instruc-tion stream.

确保正确执行，如图9的指令流所示。

Programming DAE accelerators that require explicit low-level synchronization is difficult.To reduce the programming burden, we introduce a virtual threading scheduling primitive that lets programmers specify a high-level data parallel program as they would a hard-ware back-end with support for multithreading.TVM then automatically lowers the program to a single in-struction stream with low-level explicit synchronization, as shown in Figure 8.The algorithm starts with a high-level multi-threaded program schedule and then inserts the necessary low-level synchronization operations to guarantee correct execution within each thread.Next, it interleaves operations of all virtual threads into a sin-gle instruction stream.Finally, the hardware recovers the

对需要显式低级同步的DAE加速器进行编程很困难。为了减轻编程负担，我们引入了一个虚拟线程调度原语，它允许程序员指定一个高级数据并行程序，就像支持多线程的硬件后端一样。然后，TVM通过低级显式同步自动将程序降低到单个指令流，如图8所示。该算法从高级多线程程序调度开始，然后插入必要的低级同步操作，以保证每个线程内的正确执行。接下来，它将所有虚拟线程的操作交织成一个单一的指令流。最后，硬件恢复

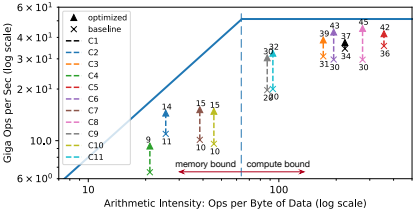


Figure 10: Roofline [47] of an FPGA-based DL ac-celerator running ResNet inference.With latency hid-ing enabled by TVM, performance of the benchmarks is brought closer to the roofline, demonstrating higher com-pute and memory bandwidth efficiency.

图10:运行ResNet推理的基于现场可编程门阵列的数字信号处理器的顶线[47]。通过TVM启用延迟隐藏，基准测试的性能更接近最高水平，显示出更高的计算机和内存带宽效率。

available pipeline parallelism dictated by the low-level synchronizations in the instruction stream.

指令流中的低级同步决定了可用的流水线并行度。

Hardware Evaluation of Latency Hiding.We now demonstrate the effectiveness of latency hiding on a cus-tom FPGA-based accelerator design, which we describe in depth in subsection 6.4.We ran each layer of ResNet on the accelerator and used TVM to generate two sched-ules: one with latency hiding, and one without.The schedule with latency hiding parallelized the program with virtuals threads to expose pipeline parallelism and therefore hide memory access latency.Results are shown in Figure 10 as a roofline diagram [47];roofline perfor-mance diagrams provide insight into how well a given system uses computation and memory resources for dif-ferent benchmarks.Overall, latency hiding improved performance on all ResNet layers.Peak compute utiliza-tion increased from 70% with no latency hiding to 88% with latency hiding.

延迟隐藏的硬件评估。我们现在展示了基于焦点FPGA的加速器设计中延迟隐藏的有效性，我们将在6.4小节中对此进行深入描述。我们在加速器上运行ResNet的每一层，并使用TVM生成两个sched-ules:一个带有延迟隐藏，另一个没有。带有延迟隐藏的调度使用虚拟线程并行化程序，以暴露流水线并行性，从而隐藏内存访问延迟。结果在图10中显示为屋顶线图[47]；屋顶线性能图提供了对给定系统如何为不同的基准使用计算和内存资源的洞察。总体而言，延迟隐藏提高了所有ResNet层的性能。峰值计算利用率从无延迟隐藏的70%增加到有延迟隐藏的88%。

7

7

5 Automating Optimization

5自动化优化

Given the rich set of schedule primitives, our remaining problem is to find optimal operator implementations for each layer of a DL model.Here, TVM creates a special-ized operator for the specific input shape and layout as-sociated with each layer.Such specialization offers sig-nificant performance benefits (in contrast to handcrafted code that would target a smaller diversity of shapes and layouts), but it also raises automation challenges.The system needs to choose the schedule optimizations - such as modifying the loop order or optimizing for the memory hierarchy - as well as schedule-specific param-eters, such as the tiling size and the loop unrolling factor.Such combinatorial choices create a large search space of operator implementations for each hardware back-end.To address this challenge, we built an automated sched-ule optimizer with two main components: a schedule ex-plorer that proposes promising new configurations, and a machine learning cost model that predicts the perfor-mance of a given configuration.This section describes these components and TVM's automated optimization flow (Figure 11).

给定丰富的调度原语集，我们剩下的问题是为DL模型的每一层找到最优的操作符实现。这里，TVM为与每个层相关的特定输入形状和布局创建一个特殊的操作符。这种专门化提供了显著的性能优势(与手工代码相比，手工代码的目标是较小的形状和布局多样性)，但它也带来了自动化挑战。系统需要选择调度优化(如修改循环顺序或优化内存层次结构)以及特定于调度的参数(如切片大小和循环展开因子)。这种组合选择为每个硬件后端创建了运营商实现的大搜索空间。为了应对这一挑战，我们构建了一个自动化的sched-ule优化器，该优化器包含两个主要组件:一个提出有前途的新配置的调度器，以及一个预测给定配置性能的机器学习成本模型。本节描述了这些组件和TVM的自动化优化流程(图11)。

5.1 Schedule Space Specification

5.1附表空间规格

We built a schedule template specification API to let a developer declare knobs in the schedule space.The tem-plate specification allows incorporation of a developer's domain-specific knowledge, as necessary, when specify-ing possible schedules.We also created a generic mas-ter template for each hardware back-end that automati-cally extracts possible knobs based on the computation description expressed using the tensor expression lan-guage.At a high level, we would like to consider as many configurations as possible and let the optimizer manage the selection burden.Consequently, the optimizer must search over billions of possible configurations for the real world DL workloads used in our experiments.

我们构建了一个时间表模板规范API，让开发人员在时间表空间中声明旋钮。当指定可能的时间表时，模板规范允许在必要时结合开发者的特定领域知识。我们还为每个硬件后端创建了一个通用的掩码模板，它根据使用张量表达式语言表示的计算描述自动提取可能的旋钮。在高层次上，我们希望考虑尽可能多的配置，并让优化器管理选择负担。因此，优化器必须为我们实验中使用的真实的DL工作负载搜索数十亿种可能的配置。

5.2 ML-Based Cost Model

基于5.2毫升的成本模型

One way to find the best schedule from a large configu-ration space is through blackbox optimization, i.e., auto-tuning.This method is used to tune high performance computing libraries [15, 46].However, auto-tuning re-quires many experiments to identify a good configura-tion.

从大型配置空间中找到最佳计划的一种方法是通过黑盒优化，即自动调整。该方法用于调整高性能计算库[15，46]。然而，自动调谐需要许多实验来确定良好的配置。

An alternate approach is to build a predefined cost model to guide the search for a particular hardware back-end instead of running all possibilities and measuring their performance.Ideally, a perfect cost model con-siders all factors affecting performance: memory access patterns, data reuse, pipeline dependencies, and thread-

另一种方法是建立一个预定义的成本模型来指导对特定硬件后端的搜索，而不是运行所有的可能性并测量它们的性能。理想情况下，一个完美的成本模型考虑了所有影响性能的因素:内存访问模式、数据重用、管道依赖和线程

Raspberry Pi

树莓皮

Tracker

追踪者

Mali GPU Nvidia GPU

马里GPU Nvidia GPU

TensorOp Specification

张量运算规范

Schedule Space Template

计划空间模板

Database

资料库

Device Cluster

设备集群

Schedule Explorer

计划资源管理器

ML Cost Model

边际成本模型

log

原木

query training

查询训练

data FPGA Board

数据现场可编程门阵列板

rpc

rpc

get\_perf

get\_perf

…

……

update

更新

Figure 11: Overview of automated optimization frame-work.A schedule explorer examines the schedule space using an ML-based cost model and chooses experiments to run on a distributed device cluster via RPC.To im-prove its predictive power, the ML model is updated pe-riodically using collected data recorded in a database.

图11:自动化优化框架的概述。时间表浏览器使用基于最大似然的成本模型检查时间表空间，并通过远程过程控制选择在分布式设备集群上运行的实验。为了提高其预测能力，最大似然模型是定期更新的，使用收集的数据记录在数据库中。

Method Category Data Cost Model Bias Need Hardware Info

方法类别数据成本模型偏差需要硬件信息

Learn from His-tory

向历史学习

Blackbox auto-tuning high none no no

黑盒自动调谐高无无无

Predefined cost model none high yes no

预定义成本模型无高是否

ML based cost model low low no yes

基于ML的成本模型低低否是

Table 1: Comparison of automation methods.Model bias refers to inaccuracy due to modeling.

表1:自动化方法的比较。模型偏差是指建模导致的不准确性。

ing patterns, among others.This approach, unfortu-nately, is burdensome due to the increasing complexity of modern hardware.Furthermore, every new hardware target requires a new (predefined) cost model.

ing模式，等等。这种方法，不可避免地，由于现代硬件的复杂性不断增加，是一种负担。此外，每个新的硬件目标都需要一个新的(预定义的)成本模型。

We instead take a statistical approach to solve the cost modeling problem.In this approach, a schedule explorer proposes configurations that may improve an operator's performance.For each schedule configuration, we use an ML model that takes the lowered loop program as in-put and predicts its running time on a given hardware back-end.The model, trained using runtime measure-ment data collected during exploration, does not require the user to input detailed hardware information.We up-date the model periodically as we explore more config-urations during optimization, which improves accuracy for other related workloads, as well.In this way, the qual-ity of the ML model improves with more experimental trials.Table 1 summarizes the key differences between automation methods.ML-based cost models strike a bal-ance between auto-tuning and predefined cost modeling and can benefit from the historical performance data of related workloads.

相反，我们采用统计方法来解决成本建模问题。在这种方法中，时间表浏览器提出可以提高操作员性能的配置。对于每个调度配置，我们使用一个ML模型，该模型将降低的循环程序作为输入，并预测其在给定硬件后端的运行时间。该模型使用探测期间收集的运行时测量数据进行训练，不需要用户输入详细的硬件信息。随着我们在优化过程中探索更多配置，我们会定期更新模型，这也提高了其他相关工作负载的准确性。通过这种方式，最大似然模型的质量随着更多的实验而提高。表1总结了自动化方法之间的主要区别。基于最大似然法的成本模型在自动调整和预定义的成本建模之间取得了平衡，并且可以从相关工作负载的历史性能数据中受益。

Machine Learning Model Design Choices.We must consider two key factors when choosing which ML model the schedule explorer will use: quality and speed.The schedule explorer queries the cost model frequently, which incurs overheads due to model prediction time and model refitting time.To be useful, these overheads must be smaller than the time it takes to measure per-

机器学习模型设计选择。我们必须考虑两个关键因素，当选择哪个ML模型的时间表探索者将使用:质量和速度。进度浏览器经常查询成本模型，由于模型预测时间和模型改装时间，会产生开销。为了有用，这些开销必须小于测量每条线所需的时间

8

8

0 100 200 300 400 500 600 700 800

0 100 200 300 400 500 600 700 800

Number of Trials

审判次数

0.00

0.00

0.25

0.25

0.50

0.50

0.75

0.75

1.00

1.00

1.25

1.25

1.50

1.50

Relative Speedup TVM: ML-based Model TVM: Blackbox Genetic Algorithm

相对加速模型:基于最大似然模型的模型:黑盒遗传算法

TVM: Random Search Baseline: cuDNN

TVM:随机搜索基线:cuDNN

Figure 12: Comparison of different automation methods for a conv2d operator in ResNet-18 on TITAN X. The ML-based model starts with no training data and uses the collected data to improve itself.The Y-axis is the speedup relative to cuDNN.We observe a similar trend for other workloads.

图12:TITAN X上ResNet-18中conv2d操作员不同自动化方法的比较。基于ML的模型从没有训练数据开始，并使用收集的数据来改进自己。Y轴是相对于cuDNN的加速。我们观察到其他工作负载也有类似的趋势。

for yo in range(4):

对于范围内的yo(4):

for xo in range(4):

对于范围(4)内的xo:

C[yo\*2:yo\*2+2][xo\*2:xo\*2+2] = 0 for ko in range(8):

C[yo\*2:yo\*2+2][xo\*2:xo\*2+2] = 0，适用于范围(8)中的ko:

for yi in range(2):

对于范围(2)中的yi:

for xi in range(2):

对于范围(2)中的xi:

C[yo\*2+yi][xo\*2+xi] +=

C[yo\*2+yi][xo\*2+xi] +=

A[k][yo\*2+yi] \* B[k][xo\*2+xi]

A[k][yo\*2+yi] \* B[k][xo\*2+xi]

xi yi k xo yo

伊稀k xo yo

C 2 4 4 16 64

C 2 4 4 16 64

A 1 2 16 16 64

A 1 2 16 16 64

B 2 2 16 64 64

B 2 2 16 64 64

Query: Loop AST Feature Extraction

查询:循环AST特征提取

cost prediction

成本预测

e.g. touched memory size

例如触摸的存储器大小

Schedule Explorer

计划资源管理器

XGBoost

XGBoost

alternatively, we can feed AST to TreeRNN

或者，我们可以把AST输入给TreeRNN

Figure 13: Example workflow of ML cost models.XG-Boost predicts costs based on loop program features.TreeRNN directly summarizes the AST.

图13:ML成本模型的示例工作流程。XG-Boost基于循环程序特性预测成本。TreeRNN直接总结AST。

formance on real hardware, which can be on the order of seconds depending on the specific workload/hardware target.This speed requirement differentiates our problem from traditional hyperparameter tuning problems, where the cost of performing measurements is very high rela-tive to model overheads, and more expensive models can be used.In addition to the choice of model, we need to choose an objective function to train the model, such as the error in a configuration's predicted running time.However, since the explorer selects the top candidates based only on the relative order of the prediction (A runs faster than B), we need not predict the absolute execution times directly.Instead, we use a rank objective to predict the relative order of runtime costs.

真实硬件上的性能，根据特定的工作负载/硬件目标，可能在几秒的数量级。这种速度要求将我们的问题与传统的超参数调整问题区分开来，在传统的超参数调整问题中，执行测量的成本相对于模型开销非常高，并且可以使用更昂贵的模型。除了模型的选择，我们还需要选择一个目标函数来训练模型，比如配置的预测运行时间的误差。然而，由于资源管理器仅根据预测的相对顺序来选择最佳候选项(A运行速度比B快)，因此我们不需要直接预测绝对执行时间。相反，我们使用等级目标来预测运行时间成本的相对顺序。

We implement several types of models in our ML opti-mizer.We employ a gradient tree boosting model (based on XGBoost [8]), which makes predictions based on fea-tures extracted from the loop program;these features in-

我们在ML优化器中实现了几种类型的模型。我们采用了一个梯度树增强模型(基于XGBoost [8])，该模型基于从循环程序中提取的特征进行预测；这些特征在-

clude the memory access count and reuse ratio of each memory buffer at each loop level, as well as a one-hot encoding of loop annotations such as "vectorize", "un-roll", and "parallel."We also evaluate a neural network model that uses TreeRNN [38] to summarize the loop program's AST without feature engineering.Figure 13 summarizes the workflow of the cost models.We found that tree boosting and TreeRNN have similar predictive quality.However, the former performs prediction twice as fast and costs much less time to train.As a result, we chose gradient tree boosting as the default cost model in our experiments.Nevertheless, we believe that both ap-proaches are valuable and expect more future research on this problem.

包括每个循环级别的内存访问计数和每个内存缓冲区的重用率，以及循环注释的一次性编码，如“矢量化”、“解卷”和“并行”我们还评估了一个神经网络模型，该模型使用TreeRNN [38]来总结没有特征工程的循环程序的AST。图13总结了成本模型的工作流程。我们发现树推进和树神经网络具有相似的预测质量。然而，前者执行预测的速度是后者的两倍，并且花费更少的时间进行训练。因此，我们在实验中选择梯度树增强作为默认的成本模型。尽管如此，我们相信这两种方法都是有价值的，并期望在这个问题上有更多的未来研究。

On average, the tree boosting model does prediction in 0.67 ms, thousands of times faster than running a real measurement.Figure 12 compares an ML-based opti-mizer to blackbox auto-tuning methods;the former finds better configurations much faster than the latter.

平均而言，树推进模型在0.67毫秒内完成预测，比运行实际测量快几千倍。图12比较了基于最大似然法的优化器和黑盒自动调谐方法；前者比后者更快地找到更好的配置。

5.3 Schedule Exploration

5.3进度探索

Once we choose a cost model, we can use it to select promising configurations on which to iteratively run real measurements.In each iteration, the explorer uses the ML model's predictions to select a batch of candidates on which to run the measurements.The collected data is then used as training data to update the model.If no ini-tial training data exists, the explorer picks random candi-dates to measure.

一旦我们选择了一个成本模型，我们就可以用它来选择有希望的配置，并在此基础上迭代运行实际测量。在每次迭代中，探索者使用最大似然模型的预测来选择一批候选对象来运行测量。然后，收集的数据被用作更新模型的训练数据。如果没有初始训练数据，探索者会选择随机的日期进行测量。

The simplest exploration algorithm enumerates and runs every configuration through the cost model, se-lecting the top-k predicted performers.However, this strategy becomes intractable with large search spaces.Instead, we run a parallel simulated annealing algo-rithm [22].The explorer starts with random configura-tions, and, at each step, randomly walks to a nearby con-figuration.This transition is successful if cost decreases as predicted by the cost model.It is likely to fail (reject) if the target configuration has a higher cost.The random walk tends to converge on configurations that have lower costs as predicted by the cost model.Exploration states persist across cost model updates;we continue from the last configuration after these updates.

最简单的探索算法通过成本模型枚举和运行每个配置，选择最佳的预测执行者。然而，这种策略在搜索空间大的情况下变得难以处理。相反，我们运行并行模拟退火算法[22]。探索者从随机配置开始，在每一步，随机走到附近的配置。如果成本按照成本模型的预测降低，这种转变是成功的。如果目标配置的成本较高，则可能会失败(拒绝)。随机游走倾向于收敛于成本模型预测的成本较低的配置。勘探状态在成本模型更新中持续存在；在这些更新之后，我们从最后的配置继续。

5.4 Distributed Device Pool and RPC

5.4分布式设备池和RPC

A distributed device pool scales up the running of on-hardware trials and enables fine-grained resource sharing among multiple optimization jobs.TVM implements a customized, RPC-based distributed device pool that en-ables clients to run programs on a specific type of de-vice.We can use this interface to compile a program on the host compiler, request a remote device, run the

分布式设备池扩大了硬件试验的运行，并支持多个优化作业之间的细粒度资源共享。TVM实现了一个定制的、基于RPC的分布式设备池，允许客户端在特定类型的设备上运行程序。我们可以使用这个接口在主编译器上编译程序，请求远程设备，运行

9

9

Name Operator H,W IC,OC K,S

名称操作员高、中、低

C1 conv2d 224, 224 3,64 7, 2

C1 conv2d 224，224 3，64 7，2

C2 conv2d 56, 56 64,64 3, 1

C2 conv2d 56，56 64，64 3，1

C3 conv2d 56, 56 64,64 1, 1

C3 conv2d 56，56 64，64 1，1

C4 conv2d 56, 56 64,128 3, 2

C4 conv2d 56，56 64，128 3，2

C5 conv2d 56, 56 64,128 1, 2

C5 conv2d 56，56 64，128 1，2

C6 conv2d 28, 28 128,128 3, 1

C6 conv2d 28，28 128，128 3，1

C7 conv2d 28, 28 128,256 3, 2

C7 conv2d 28，28 128，256 3，2

C8 conv2d 28, 28 128,256 1, 2

C8 conv2d 28，28 128，256 1，2

C9 conv2d 14, 14 256,256 3, 1

C9 conv2d 14，14 256，256 3，1

C10 conv2d 14, 14 256,512 3, 2

C10 conv2d 14，14 256，512 3，2

C11 conv2d 14, 14 256,512 1, 2

C11 conv2d 14，14 256，512 1，2

C12 conv2d 7, 7 512,512 3, 1

C12 conv2d 7，7 512，512 3，1

Name Operator H,W IC K,S

名称操作员高、中、低

D1 depthwise conv2d 112, 112 32 3, 1

D1纵深conv2d 112，112 32 3，1

D2 depthwise conv2d 112, 112 64 3, 2

D2深度conv2d 112，112 64 3，2

D3 depthwise conv2d 56, 56 128 3, 1

D3深度方向conv2d 56，56 128 3，1

D4 depthwise conv2d 56, 56 128 3, 2

D4深度conv2d 56，56 128 3，2

D5 depthwise conv2d 28, 28 256 3, 1

D5深度conv2d 28，28 256 3，1

D6 depthwise conv2d 28, 28 256 3, 2

D6深度conv2d 28，28 256 3，2

D7 depthwise conv2d 14, 14 512 3, 1

D7深度方向conv2d 14，14 512 3，1

D8 depthwise conv2d 14, 14 512 3, 2

D8深度方向conv2d 14，14 512 3，2

D9 depthwise conv2d 7, 7 1024 3, 1

D9深度方向conv2d 7，7 1024 3，1

Table 2: Configurations of all conv2d operators in ResNet-18 and all depthwise conv2d operators in Mo-bileNet used in the single kernel experiments.H/W denotes height and width, IC input channels, OC out-put channels, K kernel size, and S stride size.All ops use "SAME" padding.All depthwise conv2d operations have channel multipliers of 1.

表2:ResNet-18中所有conv2d运算符的配置，以及在单核实验中使用的Mo-bileNet中所有深度conv2d运算符的配置。高/宽表示高度和宽度、集成电路输入通道、集成电路输出通道、K内核大小和S步长。所有操作都使用“相同”填充。所有深度conv2d运算的通道乘数均为1。

function remotely, and access results in the same script on the host.TVM's RPC supports dynamic upload and runs cross-compiled modules and functions that use its runtime convention.As a result, the same infrastruc-ture can perform a single workload optimization and end-to-end graph inference.Our approach automates the compile, run, and profile steps across multiple devices.This infrastructure is especially critical for embedded de-vices, which traditionally require tedious manual effort for cross-compilation, code deployment, and measure-ment.

远程运行，并在主机上以相同的脚本访问结果。TVM的RPC支持动态上传，并运行使用其运行时约定的交叉编译模块和函数。因此，相同的基础结构可以执行单个工作负载优化和端到端图形推理。我们的方法跨多个设备自动执行编译、运行和配置步骤。这种基础设施对于嵌入式设备尤其重要，因为嵌入式设备传统上需要繁琐的手工操作来进行交叉编译、代码部署和测量。

6 Evaluation

6评估

TVM's core is implemented in C++ (∼50k LoC).We provide language bindings to Python and Java.Earlier sections of this paper evaluated the impact of several in-dividual optimizations and components of TVM, namely, operator fusion in Figure 4, latency hiding in Figure 10, and the ML-based cost model in Figure 12.We now fo-cus on an end-to-end evaluation that aims to answer the following questions:

TVM的核心是用C++实现的(50k LoC)。我们提供Python和Java的语言绑定。本文的前几部分评估了TVM的几个独立优化和组件的影响，即图4中的操作员融合、图10中的延迟隐藏和图12中的基于最大似然的成本模型。我们现在关注端到端评估，旨在回答以下问题:

• Can TVM optimize DL workloads over multiple platforms?

东方神起能否在多个平台上优化下行链路工作负载？

• How does TVM compare to existing DL frame-

TVM与现有的数字标牌框架相比如何-

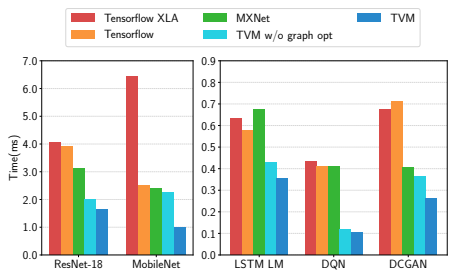


Figure 14: GPU end-to-end evaluation for TVM, MXNet, Tensorflow, and Tensorflow XLA.Tested on the NVIDIA Titan X.

图14:TVM、MXNet、张量流和XLA张量流的GPU端到端评估。在NVIDIA泰坦x上测试过。

works (which rely on heavily optimized libraries) on each back-end?

在每个后端工作(依赖于高度优化的库)?

• Can TVM support new, emerging DL workloads (e.g., depthwise convolution, low precision opera-tions)?

TVM能否支持新出现的下行链路工作负载(例如深度卷积、低精度运算)？

• Can TVM support and optimize for new specialized accelerators?

东方神起能否支持和优化新的专业加速器？

To answer these questions, we evaluated TVM on four types of platforms: (1) a server-class GPU, (2) an embed-ded GPU, (3) an embedded CPU, and (4) a DL accelera-tor implemented on a low-power FPGA SoC.The bench-marks are based on real world DL inference workloads, including ResNet [16], MobileNet [19], the LSTM Lan-guage Model [48], the Deep Q Network (DQN) [28] and Deep Convolutional Generative Adversarial Networks (DCGAN) [31].We compare our approach to exist-ing DL frameworks, including MxNet [9] and Tensor-Flow [2], that rely on highly engineered, vendor-specific libraries.TVM performs end-to-end automatic optimiza-tion and code generation without the need for an external operator library.

为了回答这些问题，我们在四种类型的平台上评估了TVM:(1)服务器级GPU，(2)嵌入式GPU，(3)嵌入式CPU，以及(4)在低功耗FPGA SoC上实现的DL加速器。基准是基于现实世界的DL推理工作负载，包括ResNet [16]，MobileNet [19]，LSTM语言模型[48]，深度Q网络(DQN) [28]和深度卷积生成对抗网络(DCGAN) [31]。我们将我们的方法与现有的依赖于高度工程化的、供应商特定的库的DL框架进行比较，包括MxNet [9]和张量流[2]。TVM执行端到端自动优化和代码生成，无需外部操作员库。

6.1 Server-Class GPU Evaluation

6.1服务器级GPU评估

We first compared the end-to-end performance of deep neural networks TVM, MXNet (v1.1), Tensor-flow (v1.7), and Tensorflow XLA on an Nvidia Titan X. MXNet and Tensorflow both use cuDNN v7 for con-volution operators;they implement their own versions of depthwise convolution since it is relatively new and not yet supported by the latest libraries.They also use cuBLAS v8 for matrix multiplications.On the other hand, Tensorflow XLA uses JIT compilation.

我们首先比较了Nvidia Titan X上的深度神经网络TVM、MXNet(1.1版)、张量流(1.7版)和张量流XLA的端到端性能。MxNet和张量流都使用cuDNN v7作为协同进化算子；他们实现了自己版本的深度卷积，因为它相对较新，还没有得到最新库的支持。他们还使用cuBLAS v8进行矩阵乘法。另一方面，XLA张量流使用JIT编译。

Figure 14 shows that TVM outperforms the base-lines, with speedups ranging from 1.6× to 3.8× due to both joint graph optimization and the automatic opti-mizer, which generates high-performance fused opera-

图14显示，TVM的性能优于基线，由于联合图形优化和自动优化器，加速范围从1.6倍到3.8倍，自动优化器产生高性能的融合运算

10

10

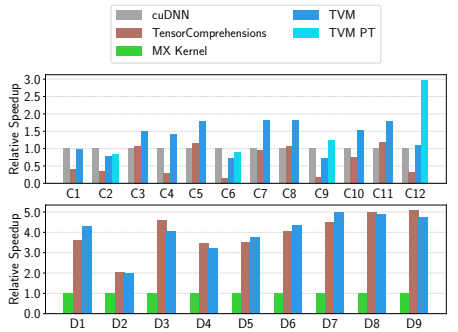


Figure 15: Relative speedup of all conv2d operators in ResNet-18 and all depthwise conv2d operators in Mo-bileNet.Tested on a TITAN X. See Table 2 for op-erator configurations.We also include a weight pre-transformed Winograd [25] for 3x3 conv2d (TVM PT).

图15:ResNet-18中所有conv2d算子和MoBilenet中所有深度conv2d算子的相对加速。在TITAN X上测试。关于操作器配置，见表2。我们还包括一个重量预转换Winograd [25]为3x3 conv2d (TVM PT)。

tors.DQN's 3.8 x speedup results from its use of un-conventional operators (4×4 conv2d, strides=2) that are not well optimized by cuDNN;the ResNet workloads are more conventional.TVM automatically finds optimized operators in both cases.

托尔斯。DQN的3.8倍加速是由于它使用了非传统算子(4×4 conv2d，跨距=2)，这些算子没有被cuDNN很好地优化；ResNet工作负载更传统。在这两种情况下，TVM都会自动找到优化的运算符。

To evaluate the effectiveness of operator level opti-mization, we also perform a breakdown comparison for each tensor operator in ResNet and MobileNet, shown in Figure 15.We include TensorComprehension (TC, com-mit: ef644ba) [42], a recently introduced auto-tuning framework, as an additional baseline.TC results in-clude the best kernels it found in 10 generations × 100 population × 2 random seeds for each operator (i.e., 2000 trials per operator).2D convolution, one of the most important DL operators, is heavily optimized by cuDNN.However, TVM can still generate better GPU kernels for most layers.Depthwise convolution is a newly introduced operator with a simpler structure [19].In this case, both TVM and TC can find fast kernels com-pared to MXNet's handcrafted kernels.TVM's improve-ments are mainly due to its exploration of a large sched-ule space and an effective ML-based search algorithm.

为了评估算子级优化的有效性，我们还对ResNet和MobileNet中的每个张量算子进行了分解比较，如图15所示。我们将最近引入的自动调整框架TensorIntelligence(TC，com-mit: ef644ba) [42]作为附加基线。TC结果包括它在10代× 100个群体×每个操作者2个随机种子中发现的最佳核(即每个操作者2000次试验)。2D卷积，一个最重要的DL算子，被cuDNN大量优化。但是TVM对于大多数层还是可以生成更好的GPU内核的。深度卷积是一种新引入的算子，具有更简单的结构[19]。在这种情况下，TVM和TC都可以找到与MXNet手工内核相比的快速内核。TVM的改进主要是由于它探索了一个大的规则空间和一个有效的基于最大似然的搜索算法。

6.2 Embedded CPU Evaluation

6.2嵌入式中央处理器评估

We evaluated the performance of TVM on an ARM Cor-tex A53 (Quad Core 1.2GHz).We used Tensorflow Lite (TFLite, commit: 7558b085) as our baseline system.Figure 17 compares TVM operators to hand-optimized

我们在ARM Cor-tex A53(四核1.2GHz)上评估了TVM的性能。我们使用Tensorflow Lite (TFLite，commit: 7558b085)作为我们的基线系统。图17比较了手动优化的TVM操作器

According to personal communication [41], TC is not yet meant to be used for compute-bound problems.However, it is still a good reference baseline to include in the comparison.

根据个人通信[41]，TC还不打算用于计算相关的问题。然而，它仍然是一个很好的参考基线，可以包括在比较中。

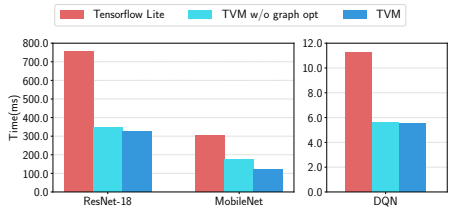


Figure 16: ARM A53 end-to-end evaluation of TVM and

图16: ARM A53端对端评估TVM和

TFLite.

TFLite。

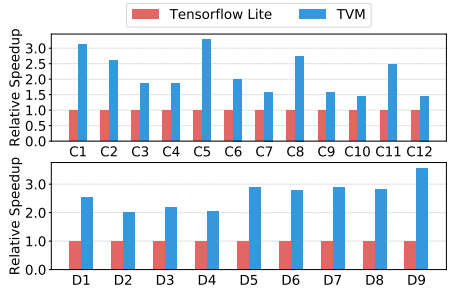


Figure 17: Relative speedup of all conv2d operators in ResNet-18 and all depthwise conv2d operators in mo-bilenet.Tested on ARM A53.See Table 2 for the con-figurations of these operators.

图17:ResNet-18中所有conv2d运算符和mo-bilenet中所有深度conv2d运算符的相对加速。在ARM A53上测试。这些运算符的配置见表2。

ones for ResNet and MobileNet.We observe that TVM generates operators that outperform the hand-optimized TFLite versions for both neural network workloads.This result also demonstrates TVM's ability to quickly opti-mize emerging tensor operators, such as depthwise con-volution operators.Finally, Figure 16 shows an end-to-end comparison of three workloads, where TVM outper-forms the TFLite baseline.

ResNet和MobileNet的。我们观察到，对于两种神经网络工作负载，TVM生成的运算符都优于手动优化的TFLite版本。这个结果也证明了TVM快速优化新兴张量算子的能力，如深度方向的协同进化算子。最后，图16显示了三种工作负载的端到端比较，其中TVM输出形成了TFLite基线。

Ultra Low-Precision Operators We demonstrate TVM's ability to support ultra low-precision infer-ence [13, 33] by generating highly optimized operators for fixed-point data types of less than 8-bits.Low-precision networks replace expensive multiplication with vectorized bit-serial multiplication that is composed of bitwise and popcount reductions [39].Achieving effi-cient low-precision inference requires packing quantized data types into wider standard data types, such as int8 or int32.Our system generates code that outperforms hand-optimized libraries from Caffe2 (commit: 39e07f7)

超低精度运算符我们通过为小于8位的定点数据类型生成高度优化的运算符，展示了TVM支持超低精度推理的能力[13，33]。低精度网络用矢量位串行乘法代替昂贵的乘法，矢量位串行乘法由按位和popcount缩减组成[39]。实现高效的低精度推理需要将量化的数据类型打包成更广泛的标准数据类型，如int8或int32。我们的系统生成的代码优于Caffe2手工优化的库(提交:39e07f7)

DCGAN and LSTM results are not presented because they are not yet supported by the baseline.

DCGAN和LSTM的结果没有提出，因为它们还没有得到基线的支持。

11

11

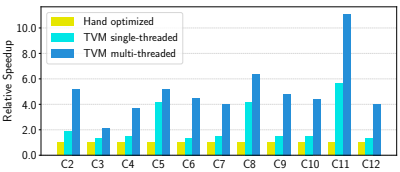


Figure 18: Relative speedup of single- and multi-threaded low-precision conv2d operators in ResNet.Baseline was a single-threaded, hand-optimized imple-mentation from Caffe2 (commit: 39e07f7).C5, C3 are 1x1 convolutions that have less compute intensity, result-ing in less speedup by multi-threading.

图18:ResNet中单线程和多线程低精度conv2d运算符的相对加速。基线是Caffe2的单线程、手动优化实现(提交:39e07f7)。C5，C3是1×1卷积，计算强度较小，导致多线程加速较慢。

[39].We implemented an ARM-specific tensorization intrinsic that leverages ARM instructions to build an ef-ficient, low-precision matrix-vector microkernel.We then used TVM's automated optimizer to explore the schedul-ing space.

[39]。我们实现了一个专用于ARM的内部张量化，它利用ARM指令来构建一个高效、低精度的矩阵向量微内核。然后，我们使用TVM的自动优化器来探索调度空间。

Figure 18 compares TVM to the Caffe2 ultra low-precision library on ResNet for 2-bit activations, 1-bit weights inference.Since the baseline is single threaded, we also compare it to a single-threaded TVM version.Single-threaded TVM outperforms the baseline, particu-larly for C5, C8, and C11 layers;these are convolution layers of kernel size 1×1 and stride of 2 for which the ul-tra low-precision baseline library is not optimized.Fur-thermore, we take advantage of additional TVM capa-bilities to produce a parallel library implementation that shows improvement over the baseline.In addition to the 2-bit+1-bit configuration, TVM can generate and opti-mize for other precision configurations that are unsup-ported by the baseline library, offering improved flexi-bility.

图18将TVM与ResNet上的Caffe2超低精度库进行了比较，用于2位激活、1位权重推断。由于基线是单线程的，我们还将其与单线程TVM版本进行了比较。单线程TVM优于基线，尤其是C5、C8和C11层；这些是内核大小为1×1、步长为2的卷积层，ul-tra低精度基线库未针对其进行优化。此外，我们利用额外的TVM能力来产生一个并行库实现，该实现比基线有所改进。除了2位+1位配置，TVM还可以生成和优化基线库未支持的其他精密配置，从而提高灵活性。

6.3 Embedded GPU Evaluation

6.3嵌入式图形处理器评估

For our mobile GPU experiments, we ran our end-to-end pipeline on a Firefly-RK3399 board equipped with an ARM Mali-T860MP4 GPU.The baseline was a vendor-provided library, the ARM Compute Library (v18.03).As shown in Figure 19, we outperformed the baseline on three available models for both float16 and float32 (DCGAN and LSTM are not yet supported by the base-line).The speedup ranged from 1.2× to 1.6×.

对于我们的移动GPU实验，我们在配备有ARM Mali-T860MP4 GPU的萤火虫-RK3399板上运行了我们的端到端流水线。基线是供应商提供的库，ARM计算库(18.03版)。如图19所示，我们在第16代和第32代的三个可用模型上都超过了基线(基线还不支持DCGAN和LSTM)。加速比在1.2×到1.6×之间。

6.4 FPGA Accelerator Evaluation

6.4现场可编程门阵列加速器评估

Vanilla Deep Learning Accelerator We now relate how TVM tackled accelerator-specific code generation on a generic inference accelerator design we prototyped on an FPGA.We used in this evaluation the Vanilla Deep

香草深度学习加速器我们现在讲述TVM如何在我们在FPGA上原型化的通用推理加速器设计上处理加速器特定的代码生成。我们在这次评估中使用了香草深度

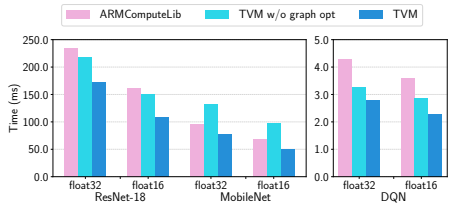


Figure 19: End-to-end experiment results on Mali-T860MP4.Two data types, float32 and float16, were evaluated.

图19:马里T860MP4上的端到端实验结果。评估了两种数据类型，float32和float16。

Learning Accelerator (VDLA) - which distills charac-teristics from previous accelerator proposals [12, 21, 27] into a minimalist hardware architecture - to demonstrate TVM's ability to generate highly efficient schedules that can target specialized accelerators.Figure 20 shows the high-level hardware organization of the VDLA architec-ture.VDLA is programmed as a tensor processor to efficiently execute operations with high compute inten-sity (e.g, matrix multiplication, high dimensional con-volution).It can perform load/store operations to bring blocked 3-dimensional tensors from DRAM into a con-tiguous region of SRAM.It also provides specialized on-chip memories for network parameters, layer inputs (nar-row data type), and layer outputs (wide data type).Fi-nally, VDLA provides explicit synchronization control over successive loads, computes, and stores to maximize the overlap between memory and compute operations.

学习加速器(VDLA) -将以前加速器提案[12，21，27]的特征提取到最低限度的硬件架构中-以展示TVM生成针对特定加速器的高效计划的能力。图20显示了虚拟数据实验室架构的高级硬件组织。VDLA被编程为张量处理器，以高效地执行具有高计算强度的运算(例如，矩阵乘法、高维卷积)。它可以执行加载/存储操作，将动态随机存取存储器中阻塞的三维张量带入静态随机存取存储器的邻接区域。它还为网络参数、层输入(nar-row数据类型)和层输出(宽数据类型)提供专门的片内存储器。最后，VDLA提供了对连续加载、计算和存储的显式同步控制，以最大化内存和计算操作之间的重叠。

Methodology.We implemented the VDLA design on a low-power PYNQ board that incorporates an ARM Cor-tex A9 dual core CPU clocked at 667MHz and an Artix-7 based FPGA fabric.On these modest FPGA resources, we implemented a 16×16 matrix-vector unit clocked at 200MHz that performs products of 8-bit values and accu-mulates them into a 32-bit register every cycle.The the-oretical peak throughput of this VDLA design is about 102.4GOPS/s. We allocated 32kB of resources for ac-tivation storage, 32kB for parameter storage, 32kB for microcode buffers, and 128kB for the register file.These on-chip buffers are by no means large enough to provide sufficient on-chip storage for a single layer of ResNet and therefore enable a case study on effective memory reuse and latency hiding.

方法论。我们在一个低功耗PYNQ板上实现了VDLA设计，该板集成了一个667MHz的ARM Cor-tex A9双核CPU和一个基于Artix-7的FPGA结构。在这些有限的现场可编程门阵列资源上，我们实现了一个时钟频率为200兆赫兹的16×16矩阵矢量单元，该单元执行8位值的乘积，并在每个周期将它们累积到32位寄存器中。这种VDLA设计的理论峰值吞吐量约为102.4千兆位/秒。我们为活动存储分配了32kB的资源，为参数存储分配了32kB的资源，为微码缓冲分配了32kB的资源，为寄存器文件分配了128kB的资源。这些片上缓冲区绝对不够大，无法为单层ResNet提供足够的片上存储，因此无法对有效的内存重用和延迟隐藏进行案例研究。

We built a driver library for VDLA with a C runtime API that constructs instructions and pushes them to the target accelerator for execution.Our code generation al-gorithm then translates the accelerator program to a se-ries of calls into the runtime API.Adding the specialized accelerator back-end took ∼2k LoC in Python.

我们用一个C运行时API为VDLA构建了一个驱动库，该API构造指令并将其推送到目标加速器执行。然后，我们的代码生成算法将加速器程序转换为一系列调用，并将其转换为运行时应用编程接口。添加专门的加速器后端用了Python中的2k LoC。

12

12

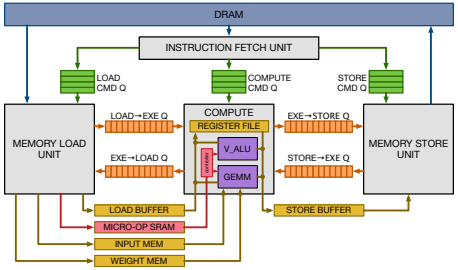


Figure 20: VDLA Hardware design overview.

图20: VDLA硬件设计概述。

End-to-End ResNet Evaluation.We used TVM to generate ResNet inference kernels on the PYNQ plat-form and offloaded as many layers as possible to VDLA.We also used it to generate both schedules for the CPU only and CPU+FPGA implementation.Due to its shal-low convolution depth, the first ResNet convolution layer could not be efficiently offloaded on the FPGA and was instead computed on the CPU.All other convolution lay-ers in ResNet, however, were amenable to efficient of-floading.Operations like residual layers and activations were also performed on the CPU since VDLA does not support these operations.

端到端ResNet评估。我们使用TVM在PYNQ平台上生成ResNet推理内核，并将尽可能多的层卸载到VDLA。我们还使用它来生成仅用于CPU和CPU+FPGA实现的时间表。由于其低卷积深度，第一个ResNet卷积层不能在现场可编程门阵列上有效卸载，而是在中央处理器上计算。然而，ResNet中的所有其他卷积层都可以进行有效的浮点运算。剩余层和激活等操作也在CPU上执行，因为VDLA不支持这些操作。

Figure 21 breaks down ResNet inference time into CPU-only execution and CPU+FPGA execution.Most computation was spent on the convolution layers that could be offloaded to VDLA.For those convolution lay-ers, the achieved speedup was 40×.Unfortunately, due to Amdahl's law, the overall performance of the FPGA accelerated system was bottlenecked by the sections of the workload that had to be executed on the CPU.We envision that extending the VDLA design to support these other operators will help reduce cost even further.This FPGA-based experiment showcases TVM's ability to adapt to new architectures and the hardware intrinsics they expose.

图21将ResNet推理时间分解为只执行CPU和CPU+FPGA执行。大部分计算花费在可以卸载到VDLA的卷积层上。对于这些卷积层，获得的加速比是40倍。不幸的是，由于Amdahl定律，FPGA加速系统的整体性能受到了必须在CPU上执行的工作负载部分的瓶颈。我们设想扩展VDLA设计以支持这些其他运营商将有助于进一步降低成本。这个基于现场可编程门阵列的实验展示了东方神起适应新架构及其揭示的硬件本质的能力。

7 Related Work

7相关工作

Deep learning frameworks [3, 4, 7, 9] provide convenient interfaces for users to express DL workloads and deploy them easily on different hardware back-ends.While ex-isting frameworks currently depend on vendor-specific tensor operator libraries to execute their workloads, they can leverage TVM's stack to generate optimized code for a larger number of hardware devices.

深度学习框架【3、4、7、9】为用户提供了方便的界面，表达DL工作负载，并轻松部署到不同的硬件后端。虽然现有的框架目前依赖于特定于供应商的张量算子库来执行它们的工作负载，但是它们可以利用TVM的堆栈来为大量硬件设备生成优化的代码。

High-level computation graph DSLs are a typical way to represent and perform high-level optimiza-tions.Tensorflow's XLA [3] and the recently introduced DLVM [45] fall into this category.The representations

高级计算图是表示和执行高级优化的典型方式。张量流的XLA [3]和最近引入的DLVM [45]属于这一类。陈述

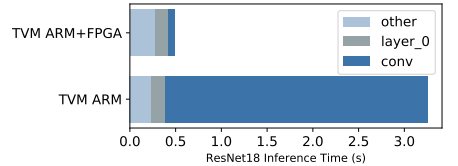


Figure 21: We offloaded convolutions in the ResNet workload to an FPGA-based accelerator.The grayed-out bars correspond to layers that could not be accelerated by the FPGA and therefore had to run on the CPU.The FPGA provided a 40x acceleration on offloaded convo-lution layers over the Cortex A9.

图21:我们将ResNet工作负载中的卷积转移到了一个基于FPGA的加速器上。灰色条对应的层不能被现场可编程门阵列加速，因此必须在中央处理器上运行。现场可编程门阵列在Cortex A9上的卸载卷积层上提供了40倍的加速。

of computation graphs in these works are similar, and a high-level computation graph DSL is also used in this paper.While graph-level representations are a good fit for high-level optimizations, they are too high level to optimize tensor operators under a diverse set of hard-ware back-ends.Prior work relies on specific lowering rules to directly generate low-level LLVM or resorts to vendor-crafted libraries.These approaches require sig-nificant engineering effort for each hardware back-end and operator-variant combination.

这些工作中的计算图是相似的，本文还使用了高级计算图DSL。虽然图级表示非常适合高级优化，但是它们的级别太高，无法在不同的硬件后端下优化张量算子。先前的工作依赖于特定的降低规则来直接生成低级LLVM或求助于供应商制作的库。这些方法需要对每个硬件后端和运营商不同的组合进行大量的工程工作。

Halide [32] introduced the idea of separating comput-ing and scheduling.We adopt Halide's insights and reuse its existing useful scheduling primitives in our compiler.Our tensor operator scheduling is also related to other work on DSL for GPUs [18, 24, 36, 37] and polyhedral-based loop transformation [6,43].TACO [23] introduces a generic way to generate sparse tensor operators on CPU.Weld [30] is a DSL for data processing tasks.We specifically focus on solving the new scheduling chal-lenges of DL workloads for GPUs and specialized accel-erators.Our new primitives can potentially be adopted by the optimization pipelines in these works.

卤化物[32]引入了分离计算和调度的思想。我们采用卤化物的见解，并在我们的编译器中重用它现有的有用的调度原语。我们的张量算子调度也与GPU的DSL[18，24，36，37]和基于多面体的循环变换[6，43]的其他工作相关。TACO [23]介绍了一种在CPU上生成稀疏张量算子的通用方法。Weld [30]是用于数据处理任务的DSL。我们特别关注为图形处理器和专用加速器解决下行链路工作负载的新调度挑战。我们的新原语可能会被这些工作中的优化管道采用。

High-performance libraries such as ATLAS [46] and FFTW [15] use auto-tuning to get the best perfor-mance.Tensor comprehension [42] applied black-box auto-tuning together with polyhedral optimizations to optimize CUDA kernels.OpenTuner [5] and existing hyper parameter-tuning algorithms [26] apply domain-agnostic search.A predefined cost model is used to automatically schedule image processing pipelines in Halide [29].TVM's ML model uses effective domain-aware cost modeling that considers program structure.The based distributed schedule optimizer scales to a larger search space and can find state-of-the-art kernels on a large range of supported back-ends.More impor-tantly, we provide an end-to-end stack that can take de-scriptions directly from DL frameworks and jointly opti-mize together with the graph-level stack.

像阿特拉斯[46]和FFTW [15]这样的高性能库使用自动调谐来获得最佳性能。张量理解[42]应用黑盒自动调整和多面体优化来优化CUDA内核。OpenTuner [5]和现有的超参数调整算法[26]应用了与域无关的搜索。预定义的成本模型用于自动调度卤化物[29]中的图像处理管道。东方神起的最大似然模型使用考虑程序结构的有效的领域感知成本建模。基于的分布式调度优化器可以扩展到更大的搜索空间，并且可以在大量支持的后端上找到最先进的内核。更重要的是，我们提供了一个端到端的堆栈，它可以直接从描述语言框架中提取描述，并与图级堆栈一起联合优化。

13

13

Despite the emerging popularity of accelerators for deep learning [11, 21], it remains unclear how a com-pilation stack can be built to effectively target these de-vices.The VDLA design used in our evaluation provides a generic way to summarize the properties of TPU-like accelerators and enables a concrete case study on how to compile code for accelerators.Our approach could potentially benefit existing systems that compile deep learning to FPGA [34,40], as well.This paper provides a generic solution to effectively target accelerators via ten-sorization and compiler-driven latency hiding.

尽管深度学习加速器越来越受欢迎[11，21]，但如何建立一个组合堆栈来有效地针对这些设备仍然不清楚。我们评估中使用的VDLA设计提供了一种概括类TPU加速器属性的通用方法，并支持关于如何为加速器编译代码的具体案例研究。我们的方法也可能有益于将深度学习编译到现场可编程门阵列的现有系统[34，40]。本文提供了一个通用的解决方案，通过十进制和编译器驱动的延迟隐藏来有效地定位加速器。

8 Conclusion

8结论

We proposed an end-to-end compilation stack to solve fundamental optimization challenges for deep learning across a diverse set of hardware back-ends.Our system includes automated end-to-end optimization, which is historically a labor-intensive and highly specialized task.We hope this work will encourage additional studies of end-to-end compilation approaches and open new op-portunities for DL system software-hardware co-design techniques.

我们提出了一个端到端的编译堆栈，以解决跨不同硬件后端深度学习的基本优化挑战。我们的系统包括自动化的端到端优化，这在历史上是一项劳动密集型和高度专业化的任务。我们希望这项工作将鼓励对端到端编译方法的进一步研究，并为分布式语言系统软硬件协同设计技术开辟新的机会。

Acknowledgement

确认

We would like to thank Ras Bodik, James Bornholt, Xi Wang, Tom Anderson and Qiao Zhang for their thorough feedback on earlier versions of this paper.We would also like to thank members of Sampa, SAMPL and Systems groups at the Allen School for their feedback on the work and manuscript.We would like to thank the anonymous OSDI reviewers, and our shepherd, Ranjita Bhagwan, for helpful feedbacks.This work was supported in part by a Google PhD Fellowship for Tianqi Chen, ONR award #N00014-16-1-2795, NSF under grants CCF-1518703, CNS-1614717, and CCF-1723352, and gifts from Intel (under the CAPA program), Oracle, Huawei and anony-mous sources.

我们要感谢拉斯·博迪克、詹姆斯·博恩霍尔特、王喜、汤姆·安德森和张桥对本文早期版本的全面反馈。我们还要感谢艾伦学校的桑巴、桑佩尔和系统小组成员对工作和手稿的反馈。我们要感谢匿名的OSDI评论家和我们的牧羊人兰吉塔·巴格万，感谢他们的有益反馈。本研究部分得到了谷歌陈天棋博士奖学金、奖#N00014-16-1-2795、国家自然科学基金(CCF-1518703、CNS-1614717和CCF-1723352)以及英特尔(根据计划)、甲骨文、华为和匿名人士的资助。

References

参考

[1] NVIDIA Tesla V100 GPU Architecture: The World's Most Ad-vanced Data Center GPU, 2017.

[1] NVIDIA Tesla V100 GPU架构:全球最先进的数据中心GPU，2017年。

[2] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANE , D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIEGAS , F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.Software available from tensorflow.org .

[2] ABADI，m .，AGARWAL，a .，BARHAM，p .，BREVDO，e .，CHEN，z .，CITRO，c .，CORRADO，G. S .，DAVIS，a .，DEAN，j .，DEVIN，m .，GHEMAWAT，s .，GOODFELLOW，I .，HARP，a .，IRVING，g .，ISARD，m .，JAI，y .，JOZEFOWICZ，r .，KAISER，l .，KUDLUR，m .，LEVENBERG，j .，MANE，d .，MONGA，r .，MOORE，s .，MURRAY，d .，OLAH，软件可从tensorflow.org获得。

[3] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., KUDLUR, M., LEVENBERG, J., MONGA, R., MOORE, S., MURRAY, D. G., STEINER, B., TUCKER, P., VASUDEVAN, V., WARDEN, P., WICKE, M., YU, Y., AND ZHENG, X. Tensor-flow: A system for large-scale machine learning.In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (2016), pp. 265-283.

[3] ABADI，m .，BARHAM，p .，CHEN，j .，CHEN，z .，DAVIS，A .，DEAN，j .，DEVIN，m .，GHEMAWAT，s .，IRVING，g .，ISARD，m .，KUDLUR，m .，LEVENBERG，j .，MONGA，r .，MOORE，s .，MURRAY，D. G .，STEINER，b .，TUCKER，p .，VASUDEVAN，v .，WARDEN，p .，WICKE，m .，YU，y .，AND ZHENG，X. Tensor-flow:大规模机器学习系统。在第12届USENIX操作系统设计和实现研讨会上(OSDI 16) (2016)，第265-283页。

[4] AGARWAL, A., AKCHURIN, E., BASOGLU, C., CHEN, G., CYPHERS, S., DROPPO, J., EVERSOLE, A., GUENTER, B., HILLEBRAND, M., HOENS, R., HUANG, X., HUANG, Z., IVANOV, V., KAMENEV, A., KRANEN, P., KUCHAIEV, O., MANOUSEK, W., MAY, A., MITRA, B., NANO, O., NAVARRO, G., ORLOV, A., PADMILAC, M., PARTHASARATHI, H., PENG, B., REZNICHENKO, A., SEIDE, F., SELTZER, M. L., SLANEY, M., STOLCKE, A., WANG, Y., WANG, H., YAO, K., YU, D., ZHANG, Y., AND ZWEIG, G. An introduction to computational networks and the computational network toolkit.Tech.Rep. MSR-TR-2014-112, August 2014.

[4] AGARWAL，a .，AKCHURIN，e .，BASOGLU，c .，CHEN，g .，CYPHERS，s .，DROPPO，j .，EVERSOLE，a .，GUENTER，b .，HILLEBRAND，m .，HOENS，r .，HUANG，x .，HUANG，z .，IVANOV，v .，KAMENEV，a .，KRANEN，p .，KUCHAIEV，o .，MANOUSEK，w .，MAY，a .，MITRA，b .，NANO，o，NAVARRO，g .，ORLOV，a .，PADMILAC，m .，PARTHASARATHI，技术。众议员MSR-TR-2014-112，2014年8月。

[5] ANSEL, J., KAMIL, S., VEERAMACHANENI, K., RAGAN-KELLEY, J., BOSBOOM, J., O'REILLY, U.-M., AND AMARAS-INGHE, S. Opentuner: An extensible framework for program au-totuning.In International Conference on Parallel Architectures and Compilation Techniques (Edmonton, Canada, August 2014).

[5] ANSEL，j .，KAMIL，s .，VEERAMACHANENI，k .，RAGAN-KELLEY，j .，BOSBOOM，j .，O 'REILLY，U.-M .，AND AMARAS-INGHE，S. Opentuner:一个可扩展的程序自动调谐框架。并行架构和编译技术国际会议(加拿大埃德蒙顿，2014年8月)。

[6] BAGHDADI, R., BEAUGNON, U., COHEN, A., GROSSER, T., KRUSE, M., REDDY, C., VERDOOLAEGE, S., BETTS, A., DONALDSON, A. F., KETEMA, J., ABSAR, J., HAASTREGT, S. V., KRAVETS, A., LOKHMOTOV, A., DAVID, R., AND HA-JIYEV, E. Pencil: A platform-neutral compute intermediate lan-guage for accelerator programming.In Proceedings of the 2015 International Conference on Parallel Architecture and Compila-tion (PACT) (Washington, DC, USA, 2015), PACT '15, IEEE Computer Society, pp. 138-149.

[6] BAGHDADI，r .，BEAUGNON，u .，COHEN，A .，GROSSER，t .，KRUSE，m .，REDDY，c .，VERDOOLAEGE，s .，BETTS，A .，DONALDSON，A. F .，KETEMA，j .，ABSAR，j .，HAASTREGT，S. V .，KRAVETS，A .，LOKHMOTOV，A .，DAVID，r .，和HAA-JIYEV，E. Pencil:一种用于加速器编程的平台中性计算中间语言。《2015年并行架构与编译国际会议论文集》(美国DC华府，2015年)，《并行架构与编译国际会议论文集》，IEEE计算机学会，第138-149页。

[7] BASTIEN, F., LAMBLIN, P., PASCANU, R., BERGSTRA, J., GOODFELLOW, I. J., BERGERON, A., BOUCHARD, N., AND BENGIO, Y. Theano: new features and speed improvements.Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[7] BASTIEN，f .，LAMBLIN，p .，PASCANU，r .，BERGSTRA，j .，GOODFELLOW，I. J .，BERGERON，a .，BOUCHARD，n .，和BENGIO，y . Anano:新特性和速度改进。深度学习和无监督特征学习NIPS 2012研讨会，2012。

[8] CHEN, T., AND GUESTRIN, C. Xgboost: A scalable tree boost-ing system.In Proceedings of the 22Nd ACM SIGKDD Inter-national Conference on Knowledge Discovery and Data Mining (New York, NY, USA, 2016), KDD '16, ACM, pp. 785-794.

[8]陈，t .，和盖斯特林，C. Xgboost:一个可扩展的树增强系统。《第22届美国计算机学会知识发现和数据挖掘国际会议论文集》(美国纽约州纽约市，2016年)，KDD 16，美国计算机学会，第785-794页。

[9] CHEN, T., LI, M., LI, Y., LIN, M., WANG, N., WANG, M., XIAO, T., XU, B., ZHANG, C., , AND ZHANG, Z. MXNet: A flexible and efficient machine learning library for heteroge-neous distributed systems.In Neural Information Processing Sys-tems, Workshop on Machine Learning Systems (LearningSys'15) (2015).

[9] CHEN，t .，LI，m .，LI，y .，LIN，m .，WANG，n .，WANG，m .，XIAO，t .，XU，b .，ZHANG，c .，AND ZHANG，Z. MXNet:一个灵活高效的异构分布式系统机器学习库。在神经信息处理系统中，机器学习系统研讨会(学习系统' 15) (2015)。

[10] CHEN, T.-F., AND BAER, J.-L. Effective hardware-based data prefetching for high-performance processors.IEEE Transactions on Computers 44, 5 (May 1995), 609-623.

[10]陈廷芳，和BAER，J-L .高性能处理器基于硬件的有效数据预取。美国电气和电子工程师协会计算机学报44，5(1995年5月)，609-623。

[11] CHEN, Y., LUO, T., LIU, S., ZHANG, S., HE, L., WANG, J., LI, L., CHEN, T., XU, Z., SUN, N., AND TEMAM, O. Dadiannao: A machine-learning supercomputer.In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitec-ture (Washington, DC, USA, 2014), MICRO-47, IEEE Computer Society, pp. 609-622.

[11] CHEN，y .，ROO，t .，LIU，s .，ZHANG，s .，HE，l .，WANG，j .，LI，l .，CHEN，t .，XU，z .，SUN，n .，AND TEMAM，O. Dadiannao:一台机器学习超级计算机。在第47届IEEE/ACM微体系结构国际研讨会(2014年，美国DC)会议录中，微47，IEEE计算机学会，第609-622页。

[12] CHEN, Y.-H., EMER, J., AND SZE, V. Eyeriss: A spatial ar-chitecture for energy-efficient dataflow for convolutional neural networks.In Proceedings of the 43rd International Symposium on Computer Architecture (Piscataway, NJ, USA, 2016), ISCA '16, IEEE Press, pp. 367-379.

[12]陈永海，EMER和SZE:卷积神经网络能量有效数据流的空间结构。《第43届国际计算机体系结构研讨会论文集》(皮斯卡塔韦，美国新泽西州，2016年)，ISCA '16，IEEE出版社，第367-379页。

[13] COURBARIAUX, M., BENGIO, Y., AND DAVID, J. Binarycon-nect: Training deep neural networks with binary weights during propagations.CoRR abs/1511.00363 (2015).

[13]库尔巴里奥，m .，BENGIO，y .，和大卫，j . Binarycon-connect:在传播期间用二进制权值训练深层神经网络。CoRR abs/1511.00363 (2015)。

14

14

[14] EGGERS, S. J., EMER, J. S., LEVY, H. M., LO, J. L., STAMM, R. L., AND TULLSEN, D. M. Simultaneous multithreading: a platform for next-generation processors.IEEE Micro 17, 5 (Sept 1997), 12-19.

[14] EGGERS，S. J .，EMER，J. S .，LEVY，H. M .，LO，J. L .，STAMM，R. L .，和TULLSEN，D. M .，同步多线程:下一代处理器的平台。IEEE微17，5(1997年9月)，12-19。

[15] FRIGO, M., AND JOHNSON, S. G. Fftw: an adaptive software ar-chitecture for the fft.In Acoustics, Speech and Signal Processing, 1998.Proceedings of the 1998 IEEE International Conference on (May 1998), vol. 3, pp. 1381-1384 vol.3.

[15] FRIGO，m .，AND JOHNSON，S. G. Fftw:一种自适应软件体系结构的快速傅里叶变换。声学，语音和信号处理，1998年。《1998年电气和电子工程师协会国际会议论文集》(1998年5月)，第3卷，第1381-1384页，第3卷

[16] HE, K., ZHANG, X., REN, S., AND SUN, J. Identity mappings in deep residual networks.arXiv preprint arXiv:1603.05027 (2016).

[16]何，金，张，x，任，s，孙，j .深剩余网络中的恒等式映射.arXiv预印本arXiv:1603.05027 (2016)。

[17] HEGARTY, J., BRUNHAVER, J., DEVITO, Z., RAGAN-KELLEY, J., COHEN, N., BELL, S., VASILYEV, A., HOROWITZ, M., AND HANRAHAN, P. Darkroom: Compiling high-level image pro-cessing code into hardware pipelines.ACM Trans.Graph.33, 4 (July 2014), 144:1-144:11.

[17] HEGARTY，j .，BRUNHAVER，j .，DEVITO，z .，RAGAN-KELLEY，j .，COHEN，n .，BELL，s .，VASILYEV，a .，HOROWITZ，m .，和HANRAHAN，P. Darkroom:将高级图像处理代码编译到硬件流水线中。ACM Trans。图表。33，4(2014年7月)，144:1-144:11。

[18] HENRIKSEN, T., SERUP, N. G. W., ELSMAN, M., HENGLEIN, F., AND OANCEA, C. E. Futhark: Purely functional gpu-programming with nested parallelism and in-place array updates.In Proceedings of the 38th ACM SIGPLAN Conference on Pro-gramming Language Design and Implementation (New York, NY, USA, 2017), PLDI 2017, ACM, pp. 556-571.

[18] HENRIKSEN，t .，SERUP，N. G. W .，ELSMAN，m .，HENGLEIN，f .，AND OANCEA，C. E. Futhark:纯函数式gpu编程，具有嵌套并行和就地数组更新。在第38届美国计算机学会编程语言设计和实现会议记录(美国纽约州纽约市，2017)，PLDI 2017，美国计算机学会，第556-571页。

[19] HOWARD, A. G., ZHU, M., CHEN, B., KALENICHENKO, D., WANG, W., WEYAND, T., ANDREETTO, M., AND ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications.CoRR abs/1704.04861 (2017).

[19] HOWARD，A. G .，ZHU，m .，CHEN，b .，KALENICHENKO，d .，WANG，w .，WEYAND，t .，ANDREETTO，m .，和ADAM，H. Mobilenets:用于移动视觉应用的高效卷积神经网络。CoRR abs/1704.04861 (2017)。

[20] JOUPPI, N. P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers.In [1990] Proceedings.The 17th Annual International Symposium on Computer Architecture (May 1990), pp. 364-373.

[20]通过增加一个小的全关联高速缓存和预取缓冲器来提高直接映射高速缓存的性能。在[1990]程序中。第17届计算机体系结构年度国际研讨会(1990年5月)，第364-373页。

[21] JOUPPI, N. P., YOUNG, C., PATIL, N., PATTERSON, D., AGRAWAL, G., BAJWA, R., BATES, S., BHATIA, S., BODEN, N., BORCHERS, A., BOYLE, R., CANTIN, P.-L., CHAO, C., CLARK, C., CORIELL, J., DALEY, M., DAU, M., DEAN, J., GELB, B., GHAEMMAGHAMI, T. V., GOTTIPATI, R., GUL-LAND, W., HAGMANN, R., HO, C. R., HOGBERG, D., HU, J., HUNDT, R., HURT, D., IBARZ, J., JAFFEY, A., JAWORSKI, A., KAPLAN, A., KHAITAN, H., KILLEBREW, D., KOCH, A., KUMAR, N., LACY, S., LAUDON, J., LAW,J., LE, D., LEARY, C., LIU, Z., LUCKE, K., LUNDIN, A., MACKEAN, G., MAG-GIORE, A., MAHONY, M., MILLER, K., NAGARAJAN, R., NARAYANASWAMI, R., NI, R., NIX, K., NORRIE, T., OMER-NICK, M., PENUKONDA, N., PHELPS, A., ROSS, J., ROSS, M., SALEK, A., SAMADIANI, E., SEVERN, C., SIZIKOV, G., SNEL-HAM, M., SOUTER, J., STEINBERG, D., SWING, A., TAN, M., THORSON, G., TIAN, B., TOMA, H., TUTTLE, E., VASUDE-VAN, V., WALTER, R., WANG, W., WILCOX, E., AND YOON, D. H. In-datacenter performance analysis of a tensor processing unit.In Proceedings of the 44th Annual International Symposium on Computer Architecture (New York, NY, USA, 2017), ISCA '17, ACM, pp. 1-12.

[21] JOUPPI，N. P .，YOUNG，c .，PATIL，n .，PATTERSON，d .，AGRAWAL，g .，BAJWA，r .，BATES，s .，BHATIA，s .，BODEN，n .，BORCHERS，a .，BOYLE，r .，CANTIN，p-l .，CHAO，c .，CLARK，c .，CORIELL，j .，DALEY，m .，DAU，m .，DEAN，j .，GELB，b .，GHAEMMAGHAMI，T. V .，GOTTIPATI，r .，GUL-LAND，w .，HAGMANN，r .，HOj .，LE，d .，LEARY，c .，LIU，z .，LUCKE，k .，LUNDIN，a .，MACKEAN，g .，MAGIORE，a .，MAHONY，m .，MILLER，k .，NAGARAJAN，r .，NARAYANASWAMI，r .，NI，r .，NIX，k .，NORRIE，t .，OMER-NICK，m .，PENUKONDA，n .，PHELPS，a .，ROSS，j .，ROSS，m .，SALEK，a .，SAMADIANI，e .，SEVERN，c .，SIZIKOV，g .，SNEL-HAM，《第44届计算机体系结构国际年会论文集》(美国纽约州纽约市，2017年)，ISCA '17，美国计算机学会，第1-12页。

[22] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Op-timization by simulated annealing.Science 220, 4598 (1983), 671-680.

[22]柯克帕特里克，s .，GELATT，C. D .，和韦奇，M. P .，通过模拟退火的优化。理科220，4598 (1983)，671-680。

[23] KJOLSTAD, F., KAMIL, S., CHOU, S., LUGATO, D., AND AMARASINGHE, S. The tensor algebra compiler.Proc.ACM Program.Lang.1, OOPSLA (Oct. 2017), 77:1-77:29.

[23] KJOLSTAD，f .，KAMIL，s .，CHOU，s .，LUGATO，d .，和AMARASINGHE，s .，张量代数编译器。继续。ACM计划。郎。1，OOPSLA(2017年10月)，77:1-77:29。

[24] KLOCKNER , A. Loo.py: transformation-based code generation for GPUs and CPUs.In Proceedings of ARRAY '14: ACM SIG-PLAN Workshop on Libraries, Languages, and Compilers for Ar-ray Programming (Edinburgh, Scotland., 2014), Association for Computing Machinery.

[24] KLOCKNER，A. Loo.py:基于转换的GPU和CPU代码生成。在ARRAY '14会议录:ACM SIG-PLAN关于Ar-ray编程的库、语言和编译器的研讨会(苏格兰爱丁堡。，2014)，计算机械协会。

[25] LAVIN, A., AND GRAY, S. Fast algorithms for convolutional neural networks.In 2016 IEEE Conference on Computer Vision

[25] LAVIN，a .，AND GRAY，s .，卷积神经网络的快速算法。在2016年IEEE计算机视觉会议上

and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016 (2016), pp. 4013-4021.

和模式识别，CVPR 2016，美国内华达州拉斯维加斯，2016年6月27-30日(2016)，第4013-4021页。

[26] LI, L., JAMIESON, K. G., DESALVO, G., ROSTAMIZADEH, A., AND TALWALKAR, A. Efficient hyperparameter optimiza-tion and infinitely many armed bandits.CoRR abs/1603.06560 (2016).

[26] LI，l .，JAMIESON，K. G .，DESALVO，g .，ROSTAMIZADEH，a .，AND TALWALKAR，a .高效超参数优化和无限多武装匪徒。CoRR abs/1603.06560 (2016)。

[27] LIU, D., CHEN, T., LIU, S., ZHOU, J., ZHOU, S., TEMAN, O., FENG, X., ZHOU, X., AND CHEN, Y. Pudiannao: A polyvalent machine learning accelerator.In Proceedings of the Twentieth International Conference on Architectural Support for Program-ming Languages and Operating Systems (New York, NY, USA, 2015), ASPLOS '15, ACM, pp. 369-381.

[27]刘，d .，陈，t .，刘，s .，周，j .，周，s .，特曼，o .，冯，x .，周，x .，和陈，y .普店脑:多价机器学习加速器。《第二十届国际程序设计语言和操作系统体系结构支持会议论文集》(纽约，纽约，美国，2015)，ASPLOS '15，ACM，第369-381页。

[28] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., ET AL.Human-level control through deep reinforcement learning.Nature 518, 7540 (2015), 529.

[28] MNIH，v .，KAVUKCUOGLU，k .，SILVER，d .，RUSU，A. A .，VENESS，j .，BELLEMARE，M. G .，GRAVES，a .，RIEDMILLER，m .，FIDJELAND，A. K .，OSTROVSKI，g .，ET AL .通过深度强化学习的人类水平控制。自然518，7540 (2015)，529。

[29] MULLAPUDI, R. T., ADAMS, A., SHARLET, D., RAGAN-KELLEY, J., AND FATAHALIAN, K. Automatically scheduling halide image processing pipelines.ACM Trans.Graph.35, 4 (July 2016), 83:1-83:11.

[29]自动调度卤化物图像处理管道。ACM Trans。图表。35，4(2016年7月)，83:1-83:11。

[30] PALKAR, S., THOMAS, J. J., NARAYANAN, D., SHANBHAG, A., PALAMUTTAM, R., PIRK, H., SCHWARZKOPF, M., AMA-RASINGHE, S. P., MADDEN, S., AND ZAHARIA, M. Weld: Re-thinking the interface between data-intensive applications.CoRR abs/1709.06416 (2017).

[30] PALKAR，s .，THOMAS，J. J .，NARAYANAN，d .，SHANBHAG，a .，PALAMUTTAM，r .，PIRK，h .，SCHWARZKOPF，m .，AMA-拉辛赫，S. P .，MADDEN，s .，和ZAHARIA，M. Weld:重新思考数据密集型应用程序之间的接口。CoRR abs/1709.06416 (2017)。

[31] RADFORD, A., METZ, L., AND CHINTALA, S. Unsupervised representation learning with deep convolutional generative adver-sarial networks.arXiv preprint arXiv:1511.06434 (2015).

[31]拉德福德，美国，梅兹，和钦塔拉，美国，无监督代表学习与深度卷积生成对抗网络。arXiv预印本arXiv:1511.06434 (2015)。

[32] RAGAN-KELLEY, J., BARNES, C., ADAMS, A., PARIS, S., DU-RAND, F., AND AMARASINGHE, S. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines.In Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (New York, NY, USA, 2013), PLDI '13, ACM, pp. 519-530.

[32] RAGAN-KELLEY，J .，BARNES，C .，ADAMS，A .，PARIS，S .，DU-RAND，F .，和AMARASINGHE，S .卤化物:用于优化图像处理流水线中的并行性、局部性和重新计算的语言和编译器。《第34届美国计算机学会编程语言设计与实现会议论文集》(美国纽约州纽约市，2013年)，PLDI '13，美国计算机学会，第519-530页。

[33] RASTEGARI, M., ORDONEZ, V., REDMON, J., AND FARHADI, A. Xnor-net: Imagenet classification using binary convolutional neural networks.In European Conference on Computer Vision (2016), Springer, pp. 525-542.

[33] RASTEGARI，m .，ORDONEZ，v .，REDMON，j .，和FARHADI，A. Xnor-net:使用二进制卷积神经网络的Imagenet分类。在欧洲计算机视觉会议(2016)，斯普林格，第525-542页。

[34] SHARMA, H., PARK, J., MAHAJAN, D., AMARO, E., KIM, J. K., SHAO, C., MISHRA, A., AND ESMAEILZADEH, H. From high-level deep neural models to fpgas.In Microarchitecture (MI-CRO), 2016 49th Annual IEEE/ACM International Symposium on (2016), IEEE, pp. 1-12.

[34] SHARMA，h .，PARK，j .，MAHAJAN，d .，AMARO，e .，KIM，J. K .，SHAO，c .，MISHRA，a .，和ESMAEILZADEH，h .从高级深层神经模型到fpgas。在微体系结构(CRO)，2016年第49届IEEE/ACM国际研讨会(2016)，IEEE，第1-12页。

[35] SMITH, J. E. Decoupled access/execute computer architectures.In Proceedings of the 9th Annual Symposium on Computer Archi-tecture (Los Alamitos, CA, USA, 1982), ISCA '82, IEEE Com-puter Society Press, pp. 112-119.

[35]解耦访问/执行计算机体系结构。《第九届计算机结构年会论文集》(美国加利福尼亚州洛斯阿拉密托斯，1982年)，ISCA 82年，美国电气和电子工程师协会计算机学会出版社，第112-119页。

[36] STEUWER, M., REMMELG, T., AND DUBACH, C. Lift: A func-tional data-parallel ir for high-performance gpu code generation.In Proceedings of the 2017 International Symposium on Code Generation and Optimization (Piscataway, NJ, USA, 2017), CGO '17, IEEE Press, pp. 74-85.

[36] STEUWER，m .，REMMELG，t .，AND DUBACH，C. Lift:一种用于高性能gpu代码生成的函数式数据并行ir。《2017年代码生成和优化国际研讨会论文集》(皮斯卡塔韦，美国新泽西州，2017年)，CGO '17，IEEE出版社，第74-85页。

[37] SUJEETH, A. K., LEE, H., BROWN, K. J., CHAFI, H., WU, M., ATREYA, A. R., OLUKOTUN, K., ROMPF, T., AND ODERSKY, M. Optiml: An implicitly parallel domain-specific language for machine learning.In Proceedings of the 28th International Con-ference on International Conference on Machine Learning (USA, 2011), ICML'11, pp. 609-616.

[37] SUJEETH，A. K .，LEE，h .，BROWN，K. J .，CHAFI，h .，WU，m .，ATREYA，A. R .，OLUKOTUN，k .，ROMPF，t .，AND ODERSKY，M. Optiml:一种隐含并行的机器学习领域特定语言。《第28届国际机器学习会议论文集》(美国，2011)，ICML'11，第609-616页。

[38] TAI, K. S., SOCHER, R., AND MANNING, C. D. Improved semantic representations from tree-structured long short-term memory networks.arXiv preprint arXiv:1503.00075 (2015).

[38]泰，K. S .，SOCHER，r .，AND MANNING，C. D .，改进的来自树型结构的长短期记忆网络的语义表示。arXiv预印本arXiv:1503.00075 (2015)。

15

15

[39] TULLOCH, A., AND JIA, Y. High performance ultra-low-precision convolutions on mobile devices.arXiv preprint arXiv:1712.02427 (2017).

[39]移动设备上的高性能超低精度卷积。arXiv预印本arXiv:1712.02427 (2017)。

[40] UMUROGLU, Y., FRASER, N. J., GAMBARDELLA, G., BLOTT, M., LEONG, P. H. W., JAHRE, M., AND VISSERS, K. A. FINN: A framework for fast, scalable binarized neural network infer-ence.CoRR abs/1612.07119 (2016).

[40] UMUROGLU，y .，FRASER，N. J .，GAMBARDELLA，g .，BLOTT，m .，LEONG，P. H. W .，JAHRE，m .，AND VISSERS，K. A. FINN:一种快速、可扩展二值化神经网络推理框架。CoRR abs/1612.07119 (2016)。

[41] VASILACHE, N. personal communication.

[41] VASILACHE，n .个人通信。

[42] VASILACHE, N., ZINENKO, O., THEODORIDIS, T., GOYAL, P., DEVITO, Z., MOSES, W. S., VERDOOLAEGE, S., ADAMS, A., AND COHEN, A. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions.CoRR abs/1802.04730 (2018).

[42] VASILACHE，n .，ZINENKO，o .，THEODORIDIS，t .，GOYAL，p .，DEVITO，z .，MOSES，W. S .，VERDOOLAEGE，s .，ADAMS，a .，AND COHEN，a . Tensor understants:与框架无关的高性能机器学习抽象。CoRR abs/1802.04730 (2018)。

[43] VERDOOLAEGE, S., CARLOS JUEGA, J., COHEN, A., IGNA-CIO GOMEZ , J., TENLLADO, C., AND CATTHOOR, F. Polyhe-dral parallel code generation for cuda.ACM Trans.Archit.Code Optim.9, 4 (Jan. 2013), 54:1-54:23.

[43] VERDOOLAEGE，s .，CARLOS JUEGA，j .，COHEN，a .，IGNA-CIO GOMEZ，j .，TENLLADO，c .，AND CATTHOOR，f .多面体-dral cuda并行代码生成。ACM Trans。阿奇特。代码Optim。9，4(2013年1月)，54:1-54:23。

[44] VOLKOV, V. Understanding Latency Hiding on GPUs.PhD thesis, University of California at Berkeley, 2016.

[44] VOLKOV，v .了解隐藏在GPU上的延迟。加州大学伯克利分校博士论文，2016。

[45] WEI, R., ADVE, V., AND SCHWARTZ, L. Dlvm: A mod-ern compiler infrastructure for deep learning systems.CoRR abs/1711.03016 (2017).

[45] WEI，r .，ADVE，v .，AND SCHWARTZ，L. Dlvm:深度学习系统的现代编译器基础设施。CoRR abs/1711.03016 (2017)。

[46] WHALEY, R. C., AND DONGARRA, J. J. Automatically tuned linear algebra software.In Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (Washington, DC, USA, 1998), SC '98, IEEE Computer Society, pp. 1-27.

[46]自动调谐线性代数软件。《1998年美国计算机学会/美国电气和电子工程师协会超级计算会议论文集》(1998年，美国DC华盛顿)，98年，美国电气和电子工程师协会计算机学会，第1-27页。

[47] WILLIAMS, S., WATERMAN, A., AND PATTERSON, D. Roofline: An insightful visual performance model for multicore architectures.Commun.ACM 52, 4 (Apr. 2009), 65-76.

[47]威廉姆斯，s .，沃特曼，a .，和帕特森，d .，屋顶线:一个深刻的多核架构的视觉性能模型。社区。ACM 52，4(2009年4月)，65-76。

[48] ZAREMBA, W., SUTSKEVER, I., AND VINYALS, O. Recurrent neural network regularization.arXiv preprint arXiv:1409.2329 (2014).

[48] ZAREMBA，W .，SUTSKEVER，I .，和VINYALS，O .递归神经网络正则化。arXiv预印本arXiv:1409.2329 (2014)。

16

16