The Deep Learning Compiler: A Comprehensive Survey

MINGZHEN LI*, YI LIU*, XIAOYAN LIU*, QINGXIAO SUN*, XIN YOU*, HAILONG YANG*†, ZHONGZHI LUAN*, LIN GAN§, GUANGWEN YANG§, and DEPEI QIAN*, Beihang University* and Tsinghua University§

<一> 深度学习编译器综述： Abstract & Introduction

算树平均数
昏昏沉沉工程师 (寻职中)
关注他

The Deep Learning Compiler: A Comprehensive Survey

The Deep Learning Compiler: A Comprehensive Survey

MINGZHEN LI*, YI LIU*, XIAOYAN LIU*, QINGXIAO SUN*, XIN YOU*, HAILONG YANG*†, ZHONGZHI LUAN*, LIN GAN§, GUANGWEN YANG§, and DEPEI QIAN*, Beihang University* and Tsinghua University§

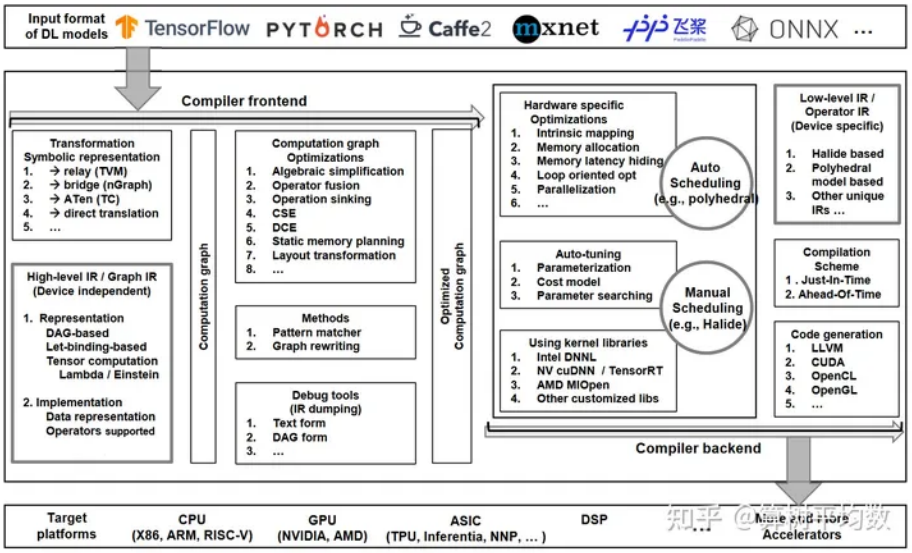
Abstract

在不同的深度学习 (DL) 硬件上部署各种深度学习 (DL) 模型的难度推动了社区对 DL 编译器的研究和开发。业界和学术界已经提出了几种深度学习编译器，例如 Tensorflow XLA 和 TVM。

类似地，深度学习编译器将**不同深度学习框架中描述的深度学习模型作为输入**，然后为不同的深度学习硬件生成优化的代码作为输出。然而，现有的综述都没有全面分析 DL 编译器的独特设计架构。

在本文中，我们通过详细剖析常用的设计，对现有的深度学习编译器进行了全面的调查，重点是面向深度学习的多级IR和前端(frontend)/后端(backend)优化。

我们对多级 IR 的设计进行了详细分析，并说明了常用的优化技术。最后，提出了一些深度学习编译器的潜在研究方向。这是第一篇关于深度学习编译器设计架构的综述论文，我们希望能为未来深度学习编译器的研究铺平道路。



Introduction

经过工业界和学术界的不断努力，人们提出了几种流行的深度学习框架，如TensorFlow、PyTorch、MXNet和CNTK，以简化各种深度学习模型的实现。

尽管上述深度学习框架各有优缺点，具体取决于其设计中的权衡，但在现有的各种深度学习模型支持新兴深度学习模型时，互操作性对于减少冗余工程工作变得非常重要。

为了提供互操作性，ONNX被提出，它定义了**表示深度学习模型的统一格式，以促进不同深度学习框架之间的模型转换。**

与此同时，芯片架构师设计定制深度学习加速硬件(DL Accelerators)以提高矩阵乘法等深度学习关键的算子的计算效率。互联网巨头（例如，Google TPU、Hisilicon NPU、Apple Bonic）、处理器供应商（例如，NVIDIA Turing 、Intel NNP）、服务提供商（例如，Amazon Inferentia、阿里巴巴含光），甚至初创公司（例如寒武纪、Graphcore ）都在投入大量的人力和资本来开发深度学习芯片，以提高深度学习模型的性能。

一般来说，深度学习硬件可以分为以下几类：

1）**软硬件协同设计的通用硬件**: 例如CPU、GPU添加了AVX512向量单元和张量核心等特殊硬件组件来加速DL模型

2）**为深度学习模型完全定制的专用硬件** : 对于谷歌TPU等专用硬件，专用集成电路（例如矩阵乘法引擎和高带宽存储器）的设计旨在将性能和能效提升到极致

3）**受生物脑科学启发的神经形态硬件**

为了丰富硬件多样性，将计算有效地映射到深度学习硬件非常重要。在通用硬件上，高度优化的线性代数库（基本线性代数子程序 (BLAS) 库（例如 MKL , cuBLAS））充当深度学习模型高效计算的基础。

以卷积运算为例，深度学习框架将卷积转换为矩阵乘法，然后调用 BLAS 库中的 GEMM 函数。此外，硬件供应商还发布了专门为深度学习计算而优化的库（例如MKL-DNN和cuDNN），包括前向和后向卷积、池化、归一化和激活。除此之外，还有更先进的工具来进一步加速深度学习操作。

例如，TensorRT 通过大量高度优化的 GPU 内核支持和图优化（例如层融合）和低比特位量化。在专用的深度学习硬件上，也提供了类似的库
模型的快速发展，从而无法有效地利用深度学

为了解决 DL 库和工具的缺点，并减轻在每个 DL 硬件上手动优化 DL 模型的负担。很快，工业界和学术界提出了几种流行的深度学习编译器，例如 TVM、Tensor Compressive、Glow、nGraph 和 XLA。

深度学习编译器将**深度学习框架中描述的模型定义作为输入**，并在各种深度学习硬件上**生成高效的代码实现作为输出。**模型定义和具体代码实现之间的转换针对模型规范和硬件架构进行了高度优化。

具体来说，它们结合了面向深度学习的优化，例如Layer和Operator融合（fusion），从而实现高效的代码生成(codegen)。此外，现有的深度学习编译器还利用通用编译器（例如 LLVM）的成熟工具链，这在不同的硬件架构之间提供了更好的可移植性。

与传统编译器类似，深度学习编译器也采用分层设计，包括前端(frontend)、中间表示（IR）和后端(backend)。然而，DL编译器的独特之处在于多级IR的设计和DL特定的优化。

在本文中，我们通过将编译器设计分解为前端、多级 IR 和后端，对现有的 DL 编译器进行了全面的调查，特别强调了 IR 设计和优化方法。

据我们所知，这是第一篇对 DL 编译器的设计进行全面调查的论文。具体来说，本文做出以下贡献：

- 我们剖析了现有深度学习编译器普遍采用的设计架构，并对关键设计组件进行了详细分析，例如多级IR、前端优化（包括node-level、block-level和dataflow-level优化）和后端优化（包括hardware-specific optimization、auto-tuning和optimized kernel libraries）。
- 我们从各个方面提供了现有深度学习编译器的全面分类，这与本次调查中描述的关键组件相对应。该分类的目标是为从业者考虑其需求提供关于选择 DL 编译器的指南，并为研究人员提供 DL 编译器的全面总结。
- 我们提供了深度学习编译器在 CNN 模型上的定量性能比较，包括成熟模型和轻量级模型。我们比较了端到端和每层（卷积层，因为它们主导推理时间）的性能，以显示优化的有效性。评估脚本和结果已开源1供参考
- 我们重点介绍了深度学习编译器未来发展的一些见解，包括dynamic shape和pre-/post-processing、高级auto-tuning、多面体模型、subgraph partitioning、量化、统一优化、可微分编程和隐私保护，我们希望推动这些发展DL 编译器社区的研究。

本文的其余部分安排如下:

第 2 节介绍了 DL 编译器的背景，包括 DL 框架、DL 硬件以及硬件 (FPGA) 特定的 DL 代码生成器。 第 3 节描述了 DL 编译器的通用设计架构。

第 4 节讨论 DL 编译器的关键组件，包括多级 IR、前端(frontend)优化和后端(backend)优化。

第 5 节介绍了全面的分类法。

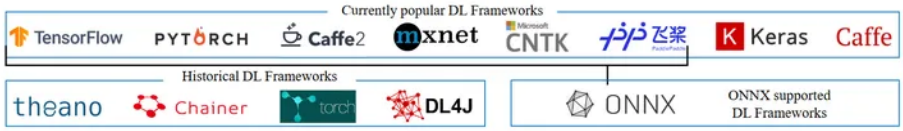
第 6 节提供了定量性能比较。

第 7 节重点介绍了 DL 编译器研究的未来方向。

Background

1. Deep Learning FrameWorks

在本节中，我们概述了流行的深度学习框架。该讨论可能并不详尽，但旨在为深度学习从业者提供指导。下图展示了 DL 框架的概况，包括当前流行的框架、历史框架和 ONNX 支持的框架。



TensorFlow: 在所有DL框架中，TensorFlow对语言接口的支持最全面，包括C++、Python、Java、Go、R和Haskell。

TensorFlow 采用了一个由 ***primitive operators extended with restricted control edges***组成的 dataflow graph 来表示一段可微的程序。

TensorFlow Lite 专为移动和嵌入式深度学习而设计，并提供 Android 神经网络 API。为了降低使用 TensorFlow 的复杂性，Google 采用 Keras 作为 TensorFlow 核心的前端。

此外，TensorFlow中的***eager-mode***应用了类似于PyTorch的方法来更好地支持动态计算图。

> "Dataflow graph of primitive operators extended with restricted control edges" 是一种描述计算过程的图结构，通常用于表示计算任务中的操作和它们之间的依赖关系。这个描述可能涉及到一些术语，我来解释一下：

1. **Dataflow Graph**: 数据流图是一种图结构，其中节点表示操作（例如算术运算、卷积、池化等），边表示操作之间的数据依赖关系。节点之间的边代表数据的流动路径，从一个操作的输出到另一个操作的输入。
2. **Primitive Operators**: 原始操作是指计算任务中的基本操作，通常是硬件或软件中可直接执行的基本指令，例如加法、乘法等。在深度学习中，原始操作可能是矩阵乘法、卷积等。
3. **Control Edges**: 控制边是指表示操作执行顺序的边。在一个数据流图中，操作之间不仅有数据依赖关系，还可能有执行顺序的依赖关系。控制边表示了操作之间的顺序关系，确保在正确的时间执行特定的操作。
4. **Extended with Restricted Control Edges**: "Extended with restricted control edges" 表示在数据流图中加入了一些限制性的控制边，这些边可能用于强制执行一些操作的先后顺序，或者限制某些条件下的执行路径。这些限制性控制边可能会影响操作的并发执行或优化。

综合起来，这个术语描述了一个基于原始操作的数据流图，其中除了数据依赖关系外，还包括了一些限制性的控制边，这些边可能在计算任务的执行过程中影响操作的执行顺序或路径。

这种描述方式可以帮助优化计算任务的执行，控制并发性，以及解决一些与操作顺序相关的问题。这在编写高性能的计算任务或者在特定条件下优化计算流程时可能会发挥作用。

> "Eager mode"（即"即时执行模式"）是指在深度学习框架中，计算图（Graph）在定义后立即执行操作，而不需要显式地构建静态计算图并进行显式编译的模式。
> 这种模式允许用户编写和调试代码时，可以像使用传统编程语言一样，逐行运行代码并查看中间结果，而不必等待整个计算图的构建和编译。
> 传统的深度学习框架，如TensorFlow 1.x版本，使用了静态计算图。在这种模式下，首先需要定义整个计算图的结构，然后将数据送入计算图进行计算。这种方式的好处是可以对计算图进行优化以提高性能，但缺点是开发过程相对较为繁琐，不够直观。
> 与此不同，"eager mode" 将计算图的构建和执行结合在一起，使得用户可以像使用Python等脚本语言一样进行操作，即时地查看计算结果，更加方便调试和快速原型开发。这在小规模实验、教育和探索性研究中特别有用。
> 需要注意的是，从 TensorFlow 2.0 开始，TensorFlow 默认启用了 Eager Execution（即 eager mode），这使得 TensorFlow 更加易于使用和学习，同时也支持了静态计算图的方式。这使得用户可以在需要时切换到静态计算图以获得更好的性能。其他深度学习框架，如 PyTorch，一直以来都是基于即时执行模式构建的。

Keras: 是一个用纯 Python 编写的高级神经网络库，用于快速构建 DL 模型。虽然 Keras 本身不是一个深度学习框架，但它提供了一个与 TensorFlow、MXNet、Theano 和 CNTK 集成的高级 API。借助 Keras，深度学习开发人员只需几行

常见的 DL 包集成，例如 scikit-learn。但Keras由于过度封装而不够灵活，导致添加算子或获取底层数据信息过于困难。

PyTorch: Facebook 用 Python 重写了基于 Lua 的深度学习框架 Torch，并在 Tensor 层面重构了所有模块。作为最流行的动态框架，PyTorch 嵌入了用于在 Python 中构建动态数据流图的原语，其中控制流在 Python 解释器中执行。 PyTorch 1.0 集成了 PyTorch 0.4 和 Caffe2 的代码库，创建了统一的框架。这使得 PyTorch 能够吸收 Caffe2 的优点，以支持高效的图形执行和移动部署。 FastAI是基于PyTorch上层封装的高级API层。它完全借用了 Keras 来简化 PyTorch 的使用。

Caffe/Caffe2: Caffe 是由加州大学伯克利分校为深度学习和图像分类而设计的。 Caffe 具有命令行、Python 和 MATLAB API。 Caffe的简单性使得源代码易于扩展，适合开发者深入分析。因此，Caffe主要定位于研究，这也使得它从一开始就流行到现在。 Caffe2 是基于原始 Caffe 项目构建的。 Caffe2 的代码结构与 TensorFlow 类似，但 API 更轻，并且更容易访问计算图中的中间结果。

MXNet: MXNet 支持多种语言 API，包括 Python、C++、R、Scala、Julia、Matlab 和 JavaScript 。它的目的是可扩展的，是从减少数据加载和 I/O 复杂性的角度设计的。 MXNet 提供不同的范例：声明式编程（如 Caffe 和 Tensorflow）以及命令式编程（如 PyTorch）。 2017年12月，亚马逊和微软联合发布了基于MXNet的Gluon，这是一个类似于Keras和FastAI的高级接口。 Gluon 支持灵活的动态图和高效的静态图。

CNTK: CNTK 可以通过 Python、C++ 和 C# API 或其自己的脚本语言（即 BrainScript）使用。CNTK 的设计易于使用且可用于生产中的大规模数据。然而，CNTK尚不支持ARM架构，这限制了它在移动设备上的使用。它使用类似于 TensorFlow 和 Caffe 的静态计算图，其中 DL 模型被视为通过有向图的一系列计算步骤。

PaddlePaddle: PaddlePaddle 的原始设计与 Caffe 类似，其中每个模型都可以表示为一组层。不过，PaddlePaddle v2参考TensorFlow采用了算子的概念，将层分解为更细粒度的算子，从而支持更复杂的DL模型。而PaddlePaddle Fluid与PyTorch类似，因为它提供了自己的解释器，以避免Python解释器的性能限制。

ONNX: 开放神经网络交换（ONNX）定义了可扩展的计算图模型，因此不同深度学习框架构建的计算图可以轻松转换为ONNX。借助 ONNX，在深度学习框架之间转换模型变得更加容易。例如，它允许开发人员构建 MXNet 模型，然后使用 PyTorch 运行该模型进行推理。如图1所示，ONNX已集成到PyTorch、MXNet、PaddlePaddle等中。对于尚未直接支持的几种深度学习框架（例如 TensorFlow 和 Keras），ONNX 为其添加了转换器。

Historical Frameworks: 由于 DL 社区的快速发展，许多历史的 DL 框架不再活跃。例如，PyTorch 已经取代了 Torch [20]。作为最古老的深度学习框架之一，Theano 已不再维护。Deeplearning4J 是一个基于 Java 和 Scala 的分布式深度学习框架，但由于缺乏大型开发者社区而变得不活跃。 Chainer 曾经是动态计算图的首选框架，但被具有类似功能的 MXNet、PyTorch 和 TensorFlow 所取代。

之前的一些工作比较了深度学习框架在不同应用（例如计算机视觉和图像分类）和不同硬件（例如CPU、GPU和TPU）上的性能。与它们不同的是，本次调查重点关注**深度学习编译器的研究工作，它提供了更通用的方法来在不同的硬件上有效地执行各种深度学习模型。**

2. Deep Learning Hardware

根据通用性，深度学习硬件可以分为三类：

1）通过硬件和软件优化可以支持深度学习工作负载的通用硬件；

2）专用硬件，通过定制的电路设计加速深度学习工作负载；

3）通过模仿人脑的神经形态硬件。

General-purpose Hardware: 深度学习模型最具代表性的通用硬件是图形处理单元（GPU），它通过多核架构实现高并行性。例如，Nvidia GPU 自 Volta 架构以来就引入了tensor cores。Tensor cores可以并行加速混合精度矩阵乘法累加计算，广泛应用于深度学习模型的训练和推理过程中。与硬件共同优化，NVIDIA还推出了高度优化的深度学习库和工具，例如cuDNN 和 TensorRT ，以进一步加速深度学习模型的计算。

Dedicated Hardware : 专用硬件是为DL计算完全定制的，以将性能和能源效率提高到极致。深度学习应用和算法的快速扩展促使许多初创公司开发专用深度学习硬件（例如 Graphcore GC2、Cambricon MLU270）。此外，传统硬件公司（如英特尔NNP、高通Cloud AI 100）和云服务提供商（如谷歌TPU、亚马逊Inferentia、阿里巴巴含光）也纷纷投资该领域。最著名的专用深度学习硬件是谷歌的 TPU 系列。 TPU 包括矩阵乘法器单元 (MXU)、统一缓冲区 (UB) 和激活单元 (AU)，后者由主机处理器使用 CISC 指令驱动。 MXU 主要由systolic array组成，该阵列针对执行矩阵乘法时的功耗和面积效率进行了优化。与 CPU 和 GPU 相比，TPU 仍然是可编程的，但使用矩阵作为primitive而不是vector或scalar。 Amazon Inferentia 最近也引起了人们的关注。该芯片有四个专为张量级运算而设计的 NeuroCore，并且具有large on-chip cache，以避免频繁的 memory access。

Neuromorphic Hardware: 神经形态芯片利用电子技术来模拟生物大脑。此类的代表产品有IBM的TrueNorth和Intel的Loihi。神经形态芯片（例如 TrueNorth）的人工神经元之间具有非常高的连接性。神经形态芯片还复制了类似于脑组织的结构：神经元可以同时存储和处理数据。传统芯片将处理器和内存分布在不同的位置，但神经形态芯片通常有许多微处理器，每个微处理器都有少量的本地内存。与TrueNorth相比，Loihi拥有更类似于大脑的学习能力。 Loihi 引入了脉冲时间依赖性突触可塑性模型 (STDP)，这是一种通过突触前和突触后脉冲的相对时间来调节突触强度的机制。然而，神经形态芯片距离大规模商业化生产还很遥远。尽管如此，在计算机科学领域，神经形态芯片可以帮助捕捉常规深度学习模型忽略的快速、终身学习的过程，而在神经学领域，它们有助于弄清楚大脑各个部分是如何工作的共同创造思想、感觉，甚至意识。

3. Hardware-specific DL Code Generator

> "Code Generator"（代码生成器）是一种软件工具，它用于将高级抽象表示（如源代码、中间表示等）转换为低级代码，通常是机器代码、汇编语言或其他编程语言的形式。代码生成器的主要目的是自动化代码编写的过程，从而提高开发效率，减少手动编写代码所需的工作量，并确保生成的代码符合特定的规范和要求。
> 代码生成器在不同的领域和情境下都有广泛的应用，包括以下几个方面：

1. **编译器**: 在编译器中，代码生成器将高级编程语言的源代码转换为目标机器代码，以便计算机能够直接执行。这涉及到词法分析、语法分析、语义分析和代码优化等多个阶段。
2. **中间代码生成**: 在一些编程环境中，代码生成器负责将高级语言源代码转换为中间表示（如三地址码、虚拟机代码等），这种中间表示通常用于后续的优化和平台无关性。
3. **框架和模板**: 一些应用程序框架或代码生成工具可以根据用户提供的配置信息，生成特定的代码结构、类、函数等，从而加速开发过程。例如，Web开发中的代码生成器可以根据数据模型自动生成数据库访问代码、API端点等。
4. **代码自动生成**: 代码生成器也可用于自动生成常见的代码模式，如单例模式、工厂模式等，从而减少开发人员的重复劳动。
5. **领域特定语言(DSL)生成**: 在某些领域中，为了更好地表达特定领域的问题，可以创建领域特定语言（DSL）。代码生成器可以将领域特定语言的代码转换为通用编程语言的代码，从而使领域问题得以更自然地解决。

总之，代码生成器在软件开发中扮演着自动化和提高效率的重要角色，可以显著减少手动编写代码的工作量，从而加速开发过程并降低错误的风险。

对于在**编译器**中，"Code Generator"（代码生成器）是编译过程中的一个阶段，它负责将经过前面的词法分析、语法分析、语义分析、语义处理和中间代码生成等步骤处理后的中间表示（如抽象语法树、三地址码等）转换为目标机器代码或者其他目标语言的代码。代码生成器是编译器的最后一个主要阶段，其主要目标是生成可在目标硬件或环境上运行的最终代码。

代码生成器的工作包括以下几个主要方面：

1. **指令选择（Instruction Selection）**：根据中间表示，选择适当的机器指令，将高级语义映射到底层指令。这包括了选择适当的寄存器、操作码、操作数等。
2. **寄存器分配（Register Allocation）**：将中间表示中的临时变量映射到实际的寄存器或内存位置，以便在目标代码中有效地使用寄存器。寄存器分配旨在最大限度地减少数据在内存和寄存器之间的移动次数。
3. **代码优化（Code Optimization）**：在生成目标代码的过程中，进行一些优化操作，以提高生成的代码的性能和效率。这可能涉及到常量折叠、循环展开、死代码消除等优化技术。
4. **目标代码生成（Target Code Generation）**：最终生成目标机器代码或目标语言的代码。这些代码应该能够在目标硬件或环境上正确地执行，并且符合所选的目标指令集体系结构。

代码生成器的设计和实现取决于编译器所针对的目标硬件架构、目标编程语言和优化目标。

不同的编译器可能使用不同的技术和策略来完成代码生成过程。代码生成的质量和效率对于生成的代码性能和可读性都有很大影响，因此代码生成器在编译器中扮演着关键的角色。

FPGA是包含可编程逻辑块阵列的可重新编程集成电路。程序员可以在制造后对其进行配置。除了可重新编程的特性外，FPGA 的低功耗和高性能特性使其广泛应用于通信、医疗、图像处理和 ASIC 原型设计等领域。

在深度学习领域，高性能CPU和GPU具有高度可重编程性，但耗电量大，而高能效ASIC则专门针对固定应用。然而，FPGA 可以弥补 CPU/GPU 和 ASIC 之间的差距，这使得 FPGA 成为深度学习的一个有吸引力的平台。

The High-Level Synthesis (HLS)编程模型使FPGA程序员能够使用C和C++等高级语言方便地生成有效的硬件设计。它避免了编写大量的Verilog或VHDL描述，从而降低了编程门槛并缩短了漫长的设计周期。

Xilinx Vivado HLS 和适用于 OpenCL 的英特尔 FPGA SDK 是两种针对各自 FPGA 的流行 HLS 工具。然而，即使使用 HLS，将 DL 模型映射到 FPGA 仍然是一项复杂的工作，因为 1) DL 模型通常由 DL 框架的语言而不是裸露的 C/C++ 代码来描述，2) DL 特定的信息和优化很困难被杠杆化。

针对FPGA的硬件特定代码生成器以DL模型或其领域特定语言（DSL）作为输入，进行特定领域（关于FPGA和DL）优化和映射，然后生成HLS或Verilog/VHDL，最后生成比特流。根据基于FPGA的加速器生成的架构可以分为两类：处理器架构（processor architecture）和流式架构（streaming architecture)。

The processor architecture: 与通用处理器有相似之处。这种架构的FPGA加速器通常由多个处理单元（PU）组成，这些处理单元由片上缓冲器和多个较小的处理引擎（PE）组成。它通常具有虚拟指令集（ISA），硬件的控制和执行的调度应由软件决定。

此外，静态调度方法避免了冯诺依曼执行的开销（包括取指令和解码）。硬件模板是具有可配置参数的通用且基本的实现。针对该架构的深度学习代码生成器采用硬件模板自动生成加速器设计。通过模板的可配置参数，代码生成器实现了可扩展性和灵活性。可扩展性意味着代码生成器可以生成从高性能到低功耗的 FPGA 设计，而灵活性意味着代码生成器可以生成具有不同层类型和参数的各种 DL 模型的设计。 PU的数量和每个PU的PE的数量是重要的模板参数。此外，耕种大小和批量大小也是将DL模型映射到PU和PE的重要调度参数。

所有这些参数通常是通过使用各种策略进行设计空间探索来确定的，例如结合性能模型和自动调整。 DNN Weaver 、AngelEye 、ALAMO 、FP-DNN、SysArrayAccel 是针对处理器架构的典型 FPGA DL 代码生成器。更重要的是，PU和PE通常负责粗粒度的基本运算，例如矩阵-向量乘法、矩阵-矩阵乘法、池化和一些逐元素运算。这些基本操作的优化主要以并行性和数据重用之间的权衡为指导，这与一般优化类似。

The streaming architecture: 与**pipeline**有相似之处。这种架构的 FPGA 加速器由多个不同的硬件模块组成，并且对于输入 DL 模型的每一层几乎都有一个硬件模块。对于深度学习模型的输入数据，这种加速器通过不同的硬件块以相同的顺序和层处理数据。

此外，通过流式输入数据，可以以流水线方式充分利用所有硬件块。然而，流式架构通常遵循一个初始假设，即目标 FPGA 上的片上内存和计算资源足以容纳 DL 模型，这给部署具有复杂层的深度模型带来了障碍。针对该架构的深度学习代码生成器可以通过利用FPGA的可重构性或采用动态控制流来解决这个问题。单个块的进一步优化类似于处理器架构的基本操作。

fpgaConvNet 、DeepBurning、Haddoc2 和 AutoCodeGen 是典型的相应 DL 代码生成器。对于将DL模型映射到FPGA的具体编译技术的详细概述。本次调查重点关注通用深度学习编译技术，这些技术可以应用于更广泛的深度学习硬件，而不是局限于FPGA。

3. COMMON DESIGN ARCHITECTURE OF DL COMPILERS

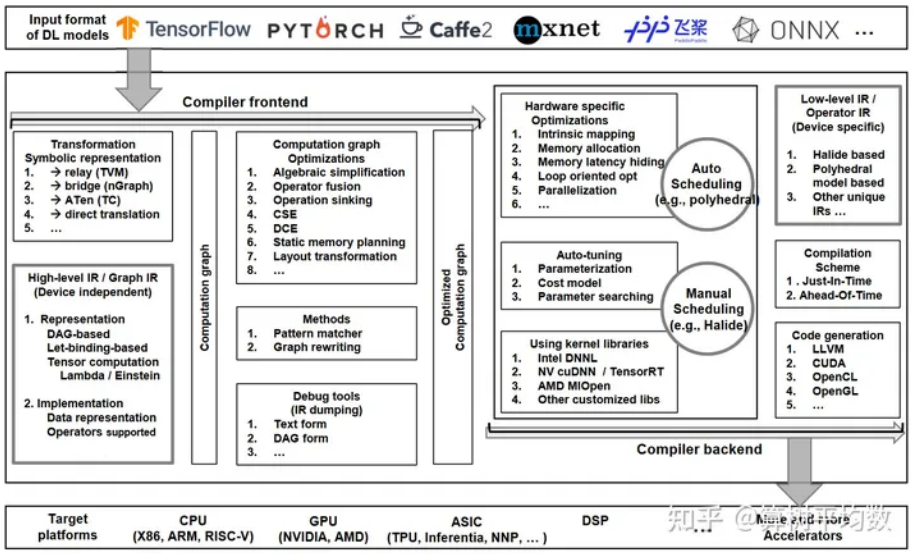
DL编译器的通用设计架构主要包含两部分：编译器前端和编译器后端。

如下图所示。中间表示（IR）分布在前端和后端。一般来说，IR是程序的抽象，用于程序优化。具体来说，DL 模型在 DL 编译器中被转换为多级 IR，其中高级 IR 驻留在前端，低级 IR 驻留在后端。

基于高级 IR，编译器前端负责独立于硬件的转换和优化。

基于低级 IR，编译器后端负责特定于硬件的优化、代码生成和编译。

本次综述重点关注 DL 编译器的设计原理。



> "IR" 在计算机科学中通常是指 "Intermediate Representation"（中间表示），它是在编译器、解释器或其他编程工具中用来表示程序或代码的中间形式。IR 是在源代码和目标代码之间的一个抽象层，通常用于在不同阶段之间传递信息、进行优化以及执行其他编译或解释相关的任务。
> 在编译器中，IR 具有以下几个主要目的：

1. **优化**: IR 提供了一个更易于进行代码优化的抽象层，因为它相对较接近源代码，但又不如目标代码那么底层。在 IR 层面进行优化可以在保留高级语义的同时，进行更有效的代码改进，例如常量折叠、循环优化等。
2. **跨平台性**: IR 可以是与特定目标平台无关的中间表示。这使得编译器可以在不同的硬件体系结构上生成目标代码，只需将 IR 转换为特定平台

3. **代码生成**: 编译器可以将 IR 转换为目标机器代码或者其他目标语言的代码。这允许在生成目标代码之前进行更多的优化，同时也支持生成多个目标代码版本。
4. **分析**: 编译器可以对 IR 进行静态分析，以检测潜在的错误、优化机会或者生成代码的统计信息。
5. **代码生成工具**: 一些代码生成工具（如代码生成器、模板引擎等）也可以使用 IR 来表示生成的代码，以便灵活地生成多种代码形式。

IR 的具体形式可以因编译器或工具的不同而异，可以是抽象语法树、三地址码、虚拟机指令等。IR 的选择通常取决于所处理的问题和应用场景。不同的编程语言、编译器和工具链都可能使用不同的 IR 来满足其特定需求。

The high-level IR:

也称为图 IR，表示计算和控制流，并且与硬件无关。高级IR的设计挑战是计算和控制流的抽象能力，它可以捕获和表达不同的深度学习模型。高级IR的目标是建立控制流以及操作符和数据之间的依赖关系，并提供用于图级优化的接口。它还包含丰富的编译语义信息，并为定制运算符提供可扩展性。高级 IR 的详细讨论见 4.1 节。

The low-level IR:

专为特定于硬件的优化和不同硬件目标上的代码生成而设计。因此，低级 IR 应该足够细粒度，以反映硬件特性并代表特定于硬件的优化。它还应该允许在编译器后端使用成熟的第三方工具链，例如 Halide 、多面体模型 和 LLVM 。低层 IR 的详细讨论见 4.2 节。

The frontend:

将现有深度学习框架中的深度学习模型作为输入，然后将模型转换为计算图表示（例如图 IR）。前端需要实现各种格式转换，以支持不同框架下的多种格式。计算图优化结合了通用编译器和深度学习特定优化的优化技术，减少了冗余并提高了图 IR 的效率。此类优化可以分为节点级（例如，nop 消除和零维张量消除）、块级（例如，代数简化、算子融合和算子下沉）和数据流级（例如，CSE、DCE、静态内存规划和布局转换）。在前端之后，生成优化的计算图并传递到后端。前端的详细讨论见 4.3 节。

The backend:

将高级 IR 转换为低级 IR 并执行特定于硬件的优化。一方面，它可以直接将高级IR转换为LLVM IR 等第三方工具链，以利用现有基础设施进行通用优化和代码生成。另一方面，它可以利用深度学习模型和硬件特性的先验知识，通过定制的编译过程来更有效地生成代码。常用的特定于硬件的优化包括硬件固有映射、内存分配和获取、内存延迟隐藏、并行化以及面向循环的优化。为了确定大优化空间中的最佳参数设置，现有的深度学习编译器广泛采用两种方法，例如自动调度（例如多面体模型）和自动调优（例如 AutoTVM）。优化的低级 IR 使用 JIT 或 AOT 进行编译，为不同的硬件目标生成代码。

编辑于 2024-01-04 17:43 · IP 属地中国香港

深度学习编译器