
先进编译实验室
Advanced Compiler
深度学习编译系统
先进编译实验室
Advanced Compiler
先进编译实验室
Advanced Compiler
先进编译实验室
Advanced Compiler
先进编译实验室
Advanced Compiler
先进编译
王
2022年

目录

01 深度学习工作流

02 深度学习编译系统背景

03 深度学习编译系统架构

深度学习工作流

背景—深度学习编译系统VS现有框架

依赖张量算子库，使用GPU加速深度学习模型的训练和推理

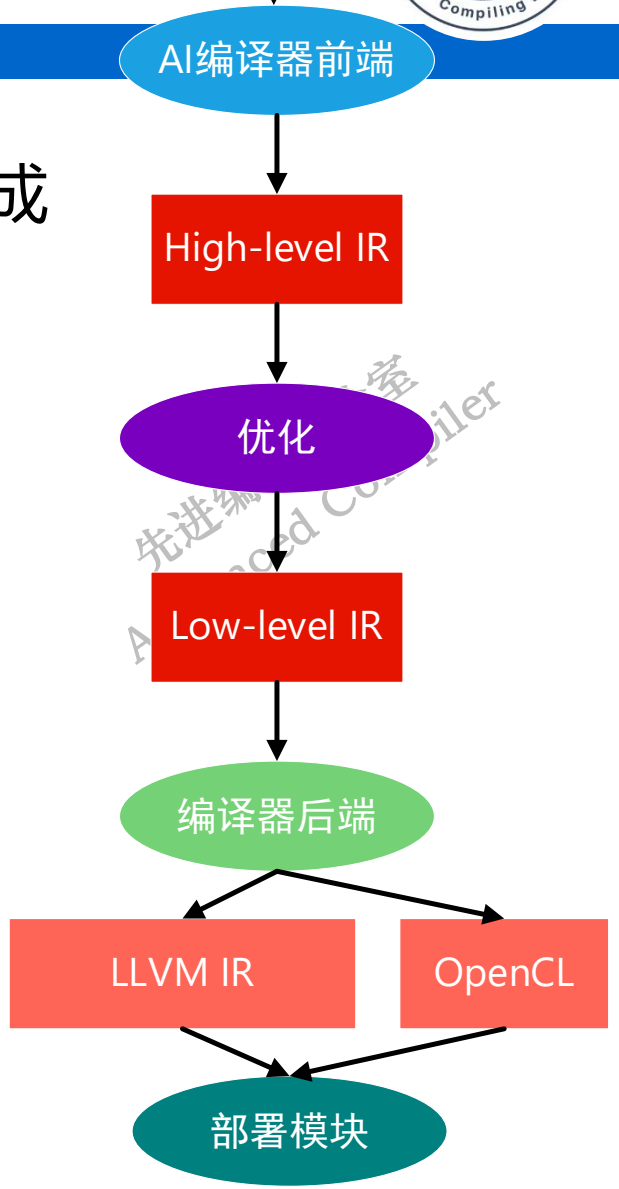
- 新算子使算子库的开发和维护工作量越来越大
- CPU、GPU、NPU等算子优化的移植性
- 更多可优化点的添加

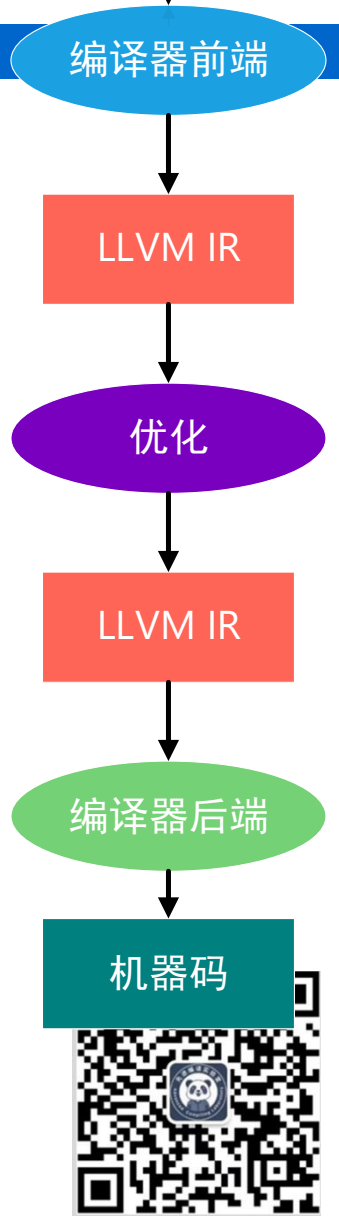
背景—深度学习编译系统VS传统编译器

相似：前端、硬件无关优化、硬件相关优化、代码生成

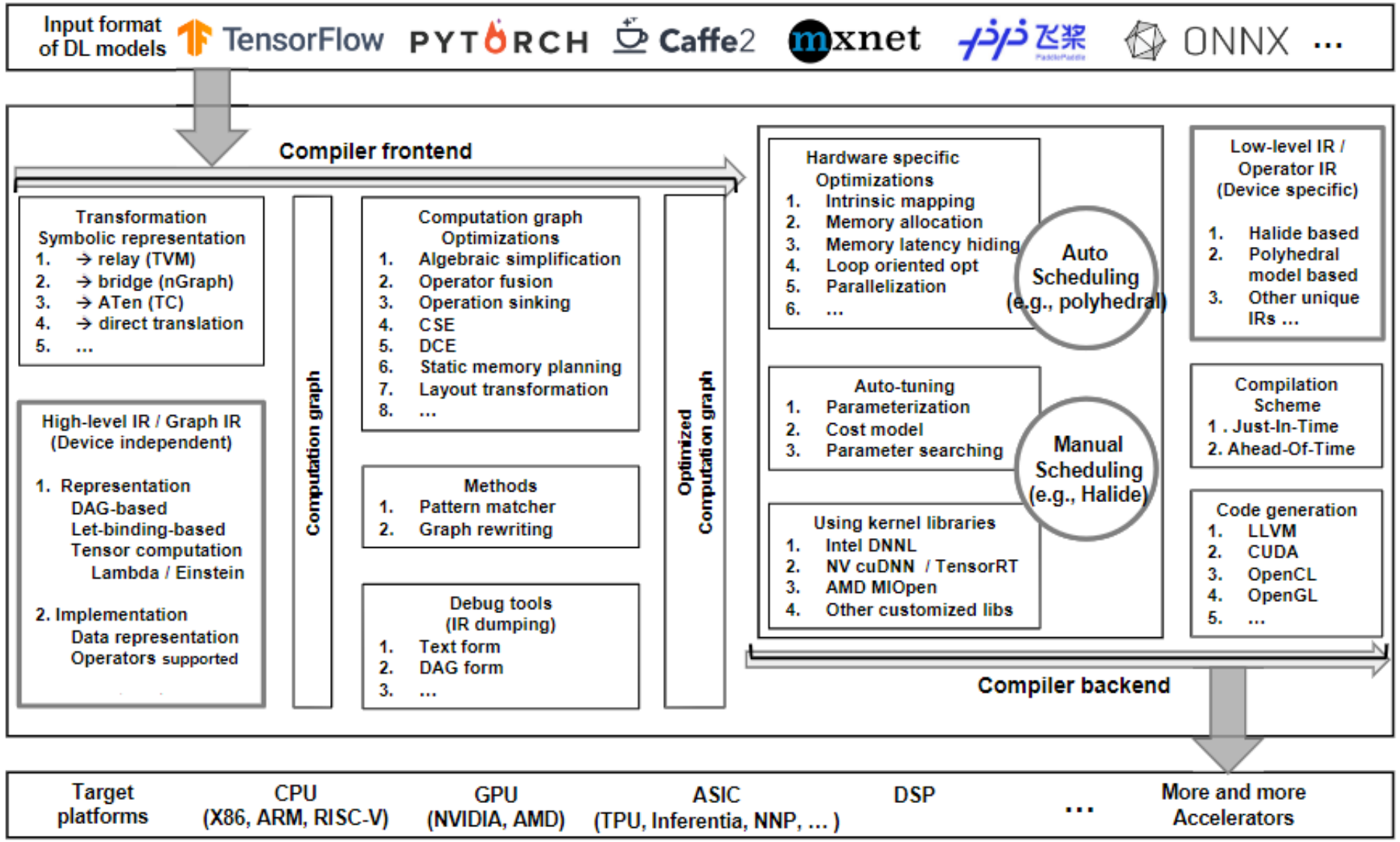
区别：

- 特定领域的编译系统
- 以Python为主的动态解释器语言的前端
- 多层IR设计（图层/算子层/codegen）
- 面向神经网络的特定优化
  - 自动微分
  - 量化/混合精度
  - 大规模并行
  - 张量运算/循环优化



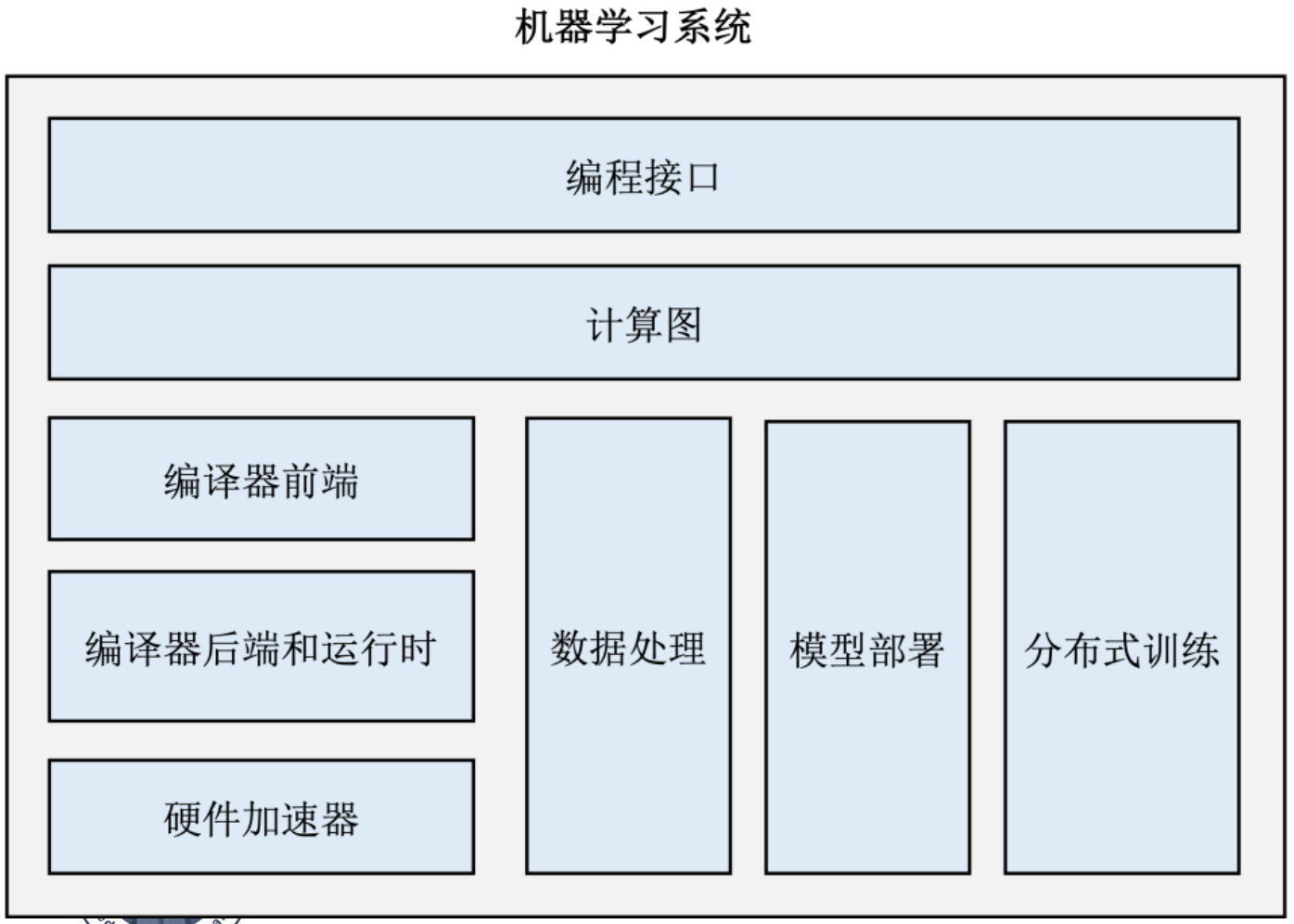


深度学习编译系统架构



- 前端
- 中间表示
- 后端

深度学习编译系统架构



- 硬件加速器
- 数据处理
- 模型部署
- 分布式训练

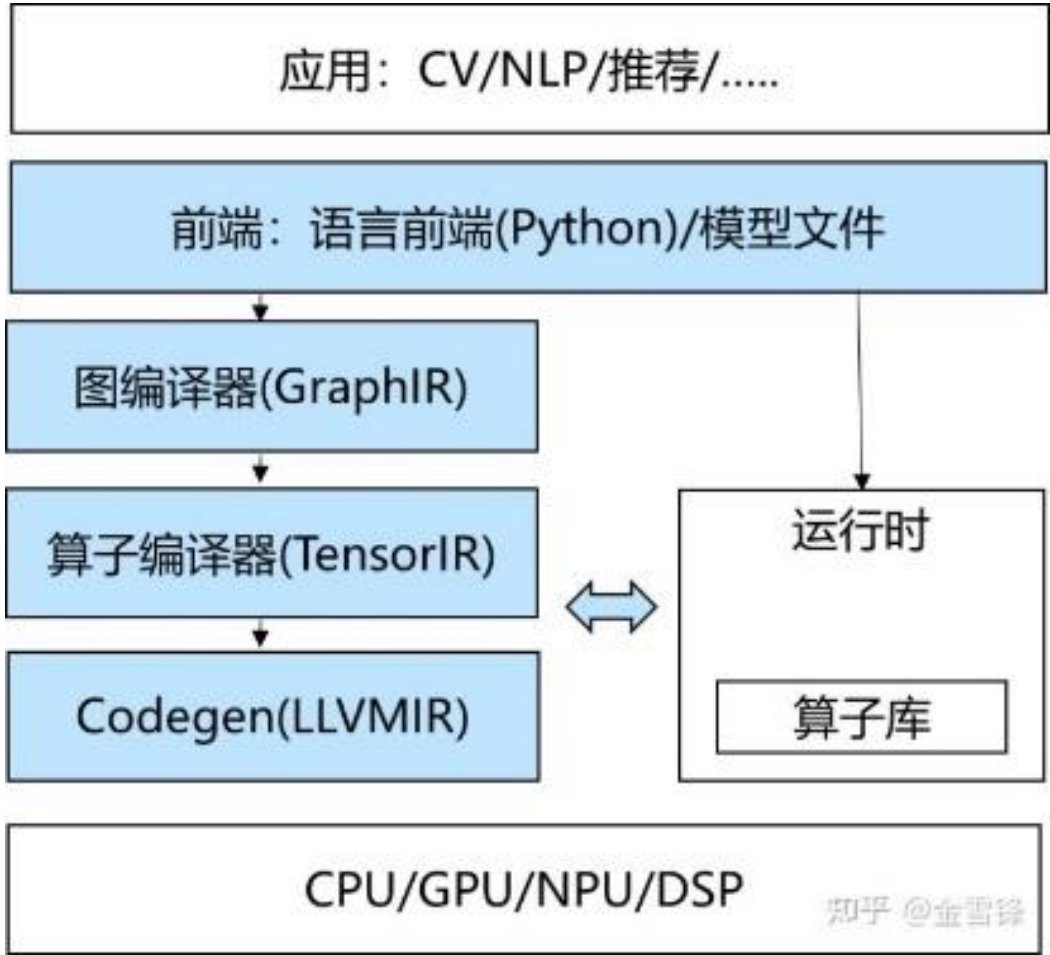
中间表示

IR设计要考虑从源代码到目标代码编译的完备性、编译优化的易用性和性能

张量表达、自动微分、JIT能力、隐式并行、循环优化、通用性

IR：深度学习编译系统中采用多级IR

- High-Level IR ： 也称为图形IR，表示计算和控制流，与硬件无关。高级IR的目标是在操作符和数据之间建立控制流和依赖关系，并为图形级优化提供接口
- Low-Level IR：用于针对不同硬件目标的特定于硬件的优化和代码生成。因此，低级别IR应该足够细粒度以反映硬件特性并表示特定于硬件的优化



中间表示

Single Layer | Dual Layers | Multiple Layers | Flexible and Exensible

program

Halide IR

LLVM Compiler

BIN binary

E.g. Halide

Resnet, DCGAN | LSTM | Training Loop

High-Level Differentiable IR

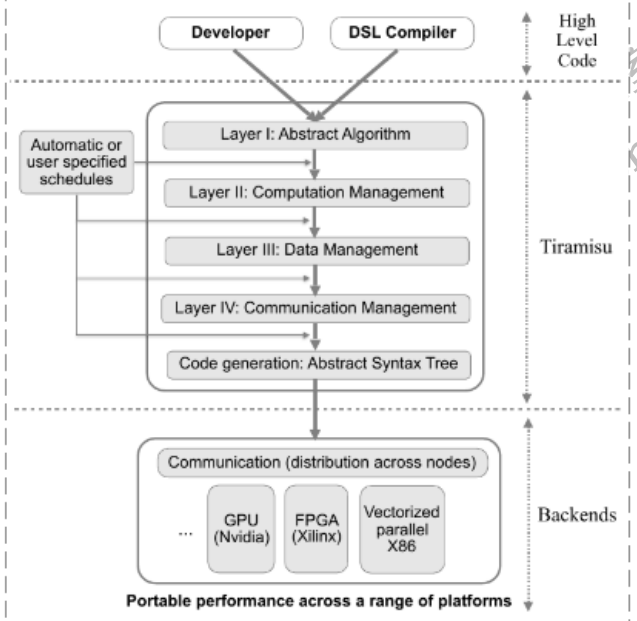
Tensor Expression IR

LLVM, CUDA, Metal | VTA

Edge FPGA | Cloud FPGA | ASIC

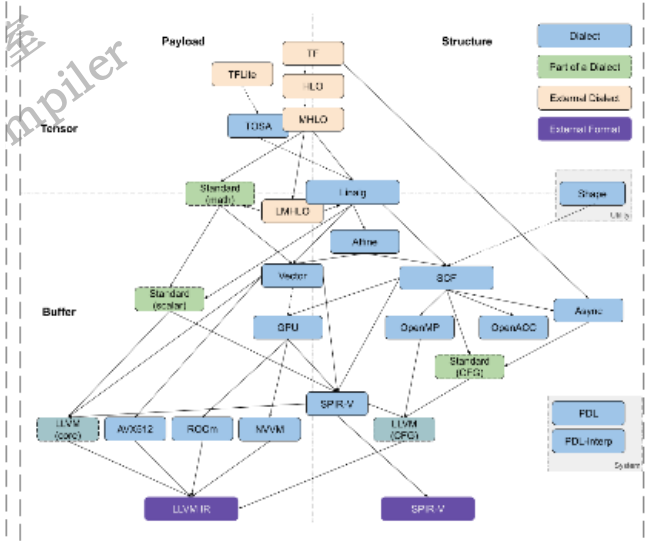
from https://tvmconf.org/slides/Jared-Roesch-Relay.pdf

E.g. TVM



from《Tiramisu: A Polyhedral Compiler for Expressing Fast and Portable Code》

E.g. Tiramisu



from https://llvm.discourse.group/t/codegen-dialect-overview/2723

E.g. MLIR

前端——计算图

Python静态化：AI网络→图IR

计算图：用来表示深度学习网络模型在训练与推理过程中计算逻辑与状态的工具，由张量Tensor、算子Operator以及有向线段表示的张量状态、依赖关系构成

作用：

- ◆ 对于输入数据、算子和算子执行顺序进行统一表达
- ◆ 定义中间状态和模型状态，从而帮助框架更好地管理内存
- ◆ 自动化分析模型定义、计算梯度
- ◆ 分析算子执行关系，发现将算子进行异步执行的机会，从而优化程序执行

分类：◆ 静态图　◆ 动态图

前端——计算图

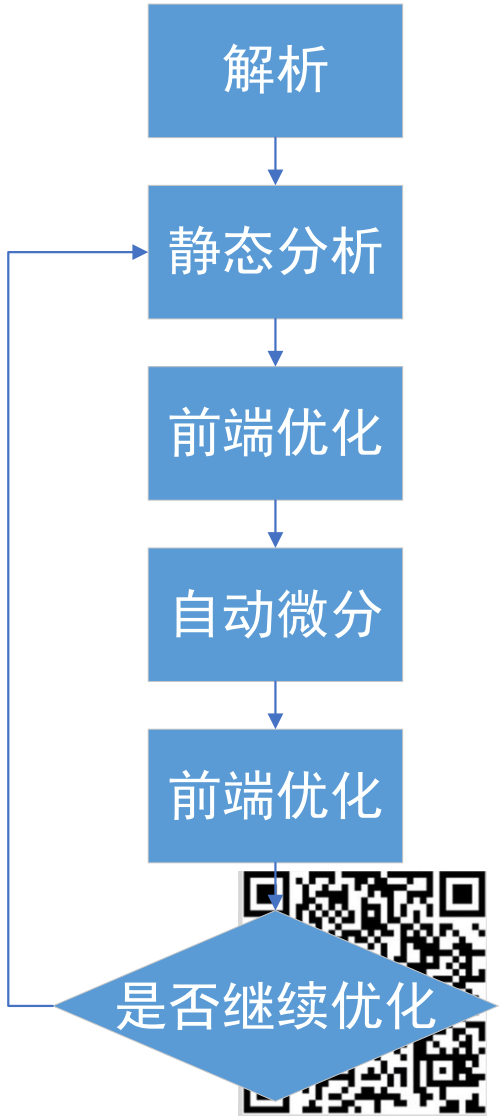
- 静态图：在代码执行前生成正反向图并完成编译优化
  - Tracing模式：框架把Python假执行一遍，记录算子执行序列作为正向图，并以此进行自动微分生成反向图，然后进行正反向图的编译优化
  - AST转换：框架获取Python代码的AST，通过编译技术转换成正向图，基于正向图生成反向图，同时进行编译优化
- 动态图：使用Python相关机制注册算子，正向利用Python解释执行，同时通过自动微分机制生成反向图，然后基于反向图进行梯度更新

前端——计算图

Python静态化：AI网络→图IR

基于抽象释义的静态分析：完成类型推导和特化

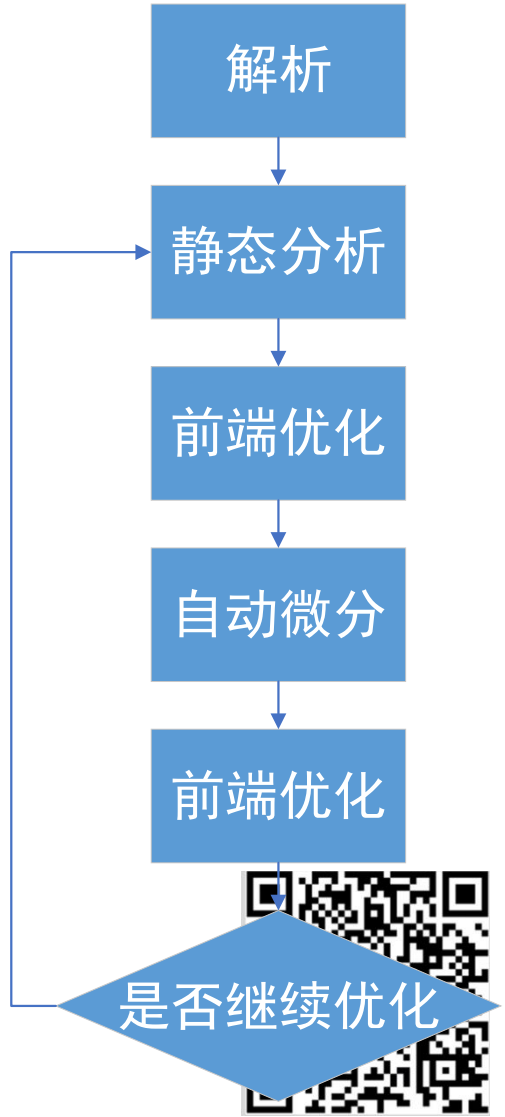
- 类型系统：类型的集合以及使用类型来规定程序行为的规则，用于定义不同的类型、指定类型的操作和类型之间的相互作用
- 静态分析：在不实际运行程序的情况下，通过词法分析、语法分析、控制流、数据流分析等技术对代码进行分析验证的技术



趋势：动静分离→动静结合→动静统一

前端——硬件无关优化

- 无用与不可用代码消除
- 常量传播
- 常量折叠
- 公共子表达式消除



前端——自动微分

微分求解：手动求解、数值微分、符号微分、自动微分

自动微分：将计算机程序中的运算操作分解为一个有限的基本操作集合，且集合中基本操作的求导规则已知，在完成每一个基本操作的求导后，使用链式法则将结果组合到整体程序的求导结果

根据链式法则的不同组合顺序可分为：

- 前向模式：从输入方向开始计算梯度值 $\frac{dy}{dx}=\frac{dy}{da}\left(\frac{da}{db}\left(\frac{db}{dc}\frac{dc}{dx}\right)\right)$
- 后向模式：从输出方向开始计算梯度值 $\frac{dy}{dx}=\left(\left(\frac{dy}{da}\frac{da}{db}\right)\frac{db}{dc}\right)\frac{dc}{dx}$

计算图优化

High-level IR：隐藏了一些底层运行的细节信息，无法直接映射到目标设备上的算子

计算图优化：对High-level IR上的各个节点进行拆分和融合，转换为可在硬件上执行的Low-level IR。通过等效图变换达到简化计算、减少资源开销、适配硬件的执行能力、提升执行性能。

- 通用硬件优化：在计算图中尝试匹配特定的子图结构，找到目标子图结构后通过等价替换方式将其替换成对硬件友好的子图结构
  - 算子融合
  - 自动算子生成
  - 图算融合
- 特定硬件优化
  - 由于硬件指令限制而做的优化
  - 特定于存储格式的优化

后端——算子选择

算子选择：针对优化的Low-level IR，由于对于每个节点可以选择不同的输入输出格式和数据类型等，为其选出最为合适的算子，生成完整的算子序列

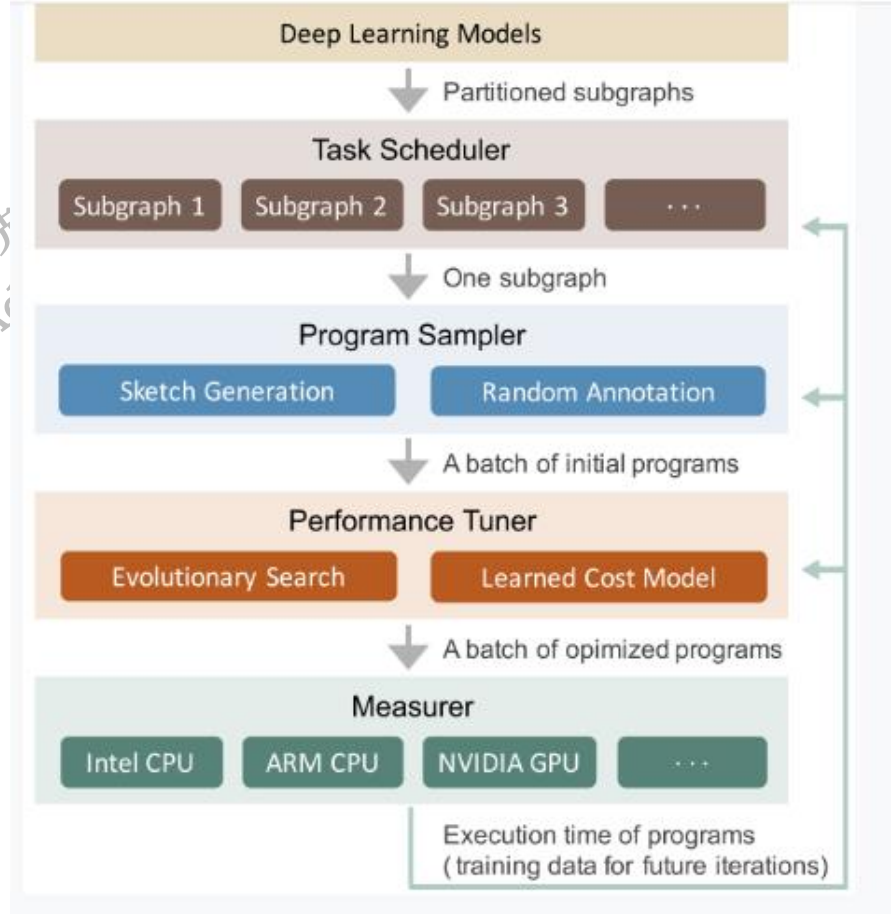
Ansor：

1、如何自动化的构造一个更大的搜索空间?

Ansor 使用了一个层次化的搜索空间

2、如何更有效的进行搜索?

Ansor 在搜索过程中增加了采样，先对完整的程序进行采样然 后再调整，提高了搜索效率



后端与运行时——内存分配与计算调度执行

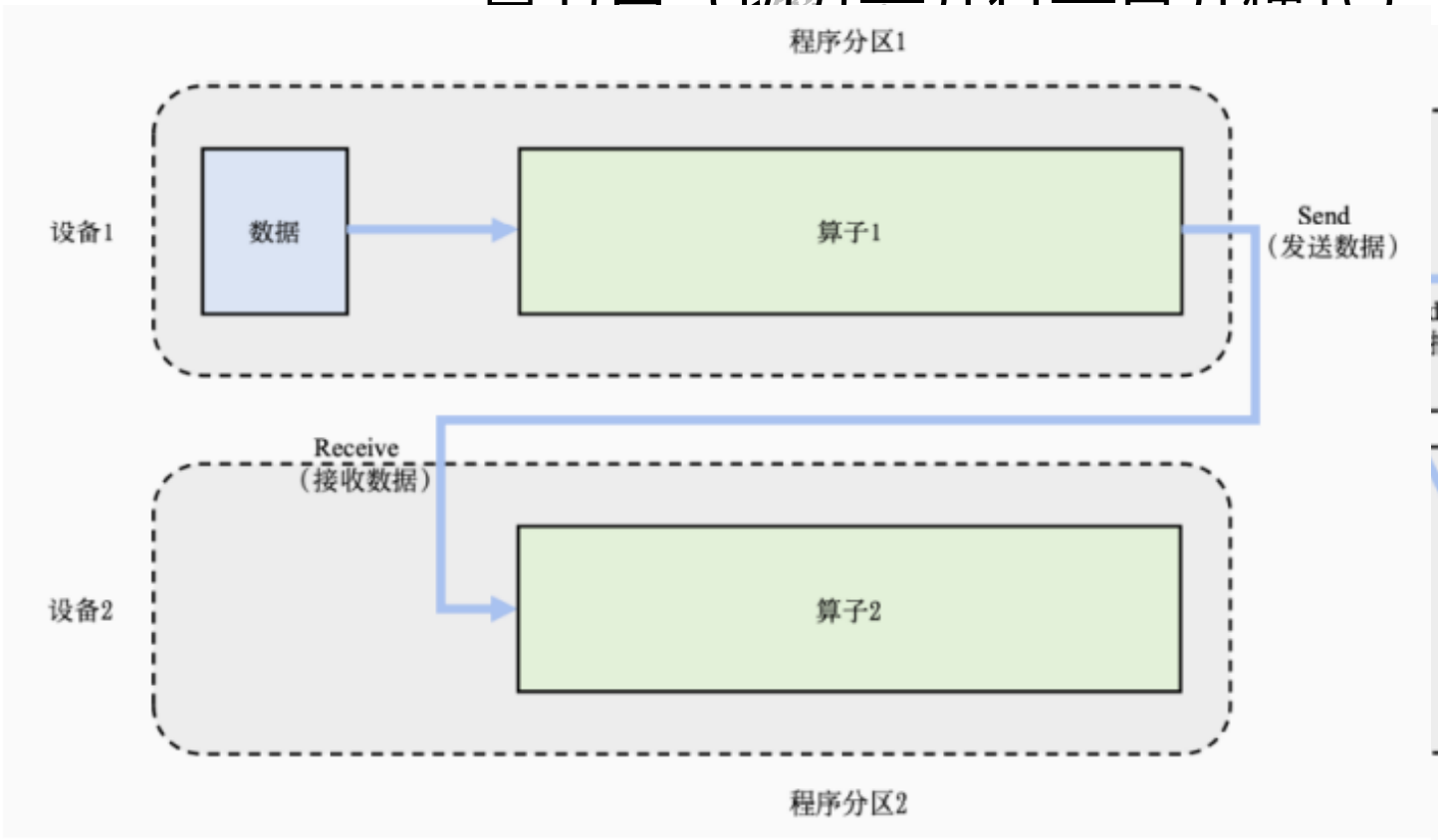
内存分配：遍历算子序列，根据IR图中算子的输入输出的形状、数据类型、存储格式等信息为每个算子计算分配相应的输入输出内存，然后将算子加载至设备上执行计算

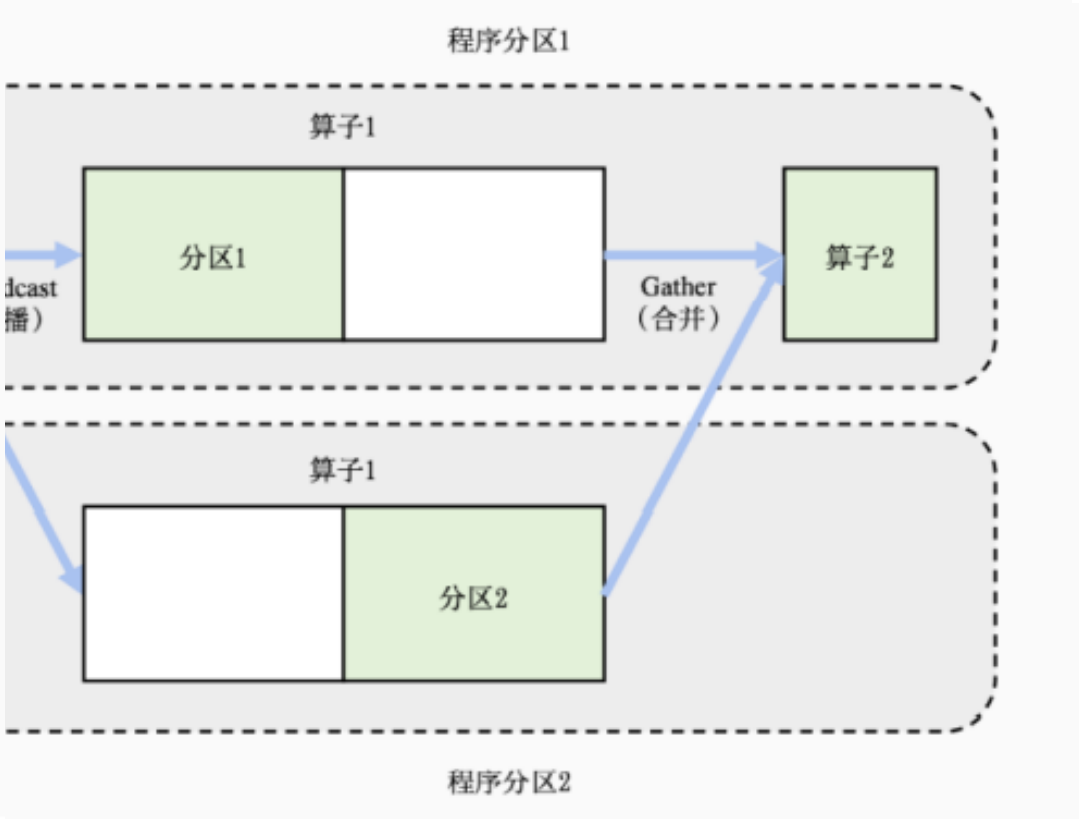
- 分配连续的内存
- 尽量减少为每个算子的输入输出分配不同的内存

计算调度执行：计算任务可通过运行时完成计算的调度与在硬件上的执行

模型训练

分布式训练：为应对单个机器上内存及算力资源不足，设计分布式训练系统将一个机器学习模型任务拆分成多个子任务，并将子任务分发给多个计算节点（切分—并行—合并模式），解决资源瓶颈。





数据处理

数据处理：包括数据加载、数据混洗、数组组装、数据发送等过程

数据处理模块设计需要满足下面的特性：

- 易用性：提供良好的编程抽象和接口使得用户方便构建一个数据处理流水、支持用户灵活地在数据流水中注册自定义算子
- 高效性：支持随机读取且具备高读取吞吐率的文件格式、合理的并行架构执行数据流水线：数据读取、数据预处理计算、芯片上模型计算三个步骤异步并行执行
- 保序性：使数据最终送入模型的顺序由数据混洗组件的数据输出顺序唯一确定

模型部署

模型部署：将训练好的模型部署到运行环境中进行推理的过程

- 训练模型到推理模型的转换
- 部署阶段的性能优化
- 模型压缩
  - 量化
  - 稀疏
  - 知识蒸馏
- 模型推理优化
- 模型安全保护

谢 谢!

先进编译实验室
王磊
2022年07月