AI编译器-后端优化

Auto Tuning

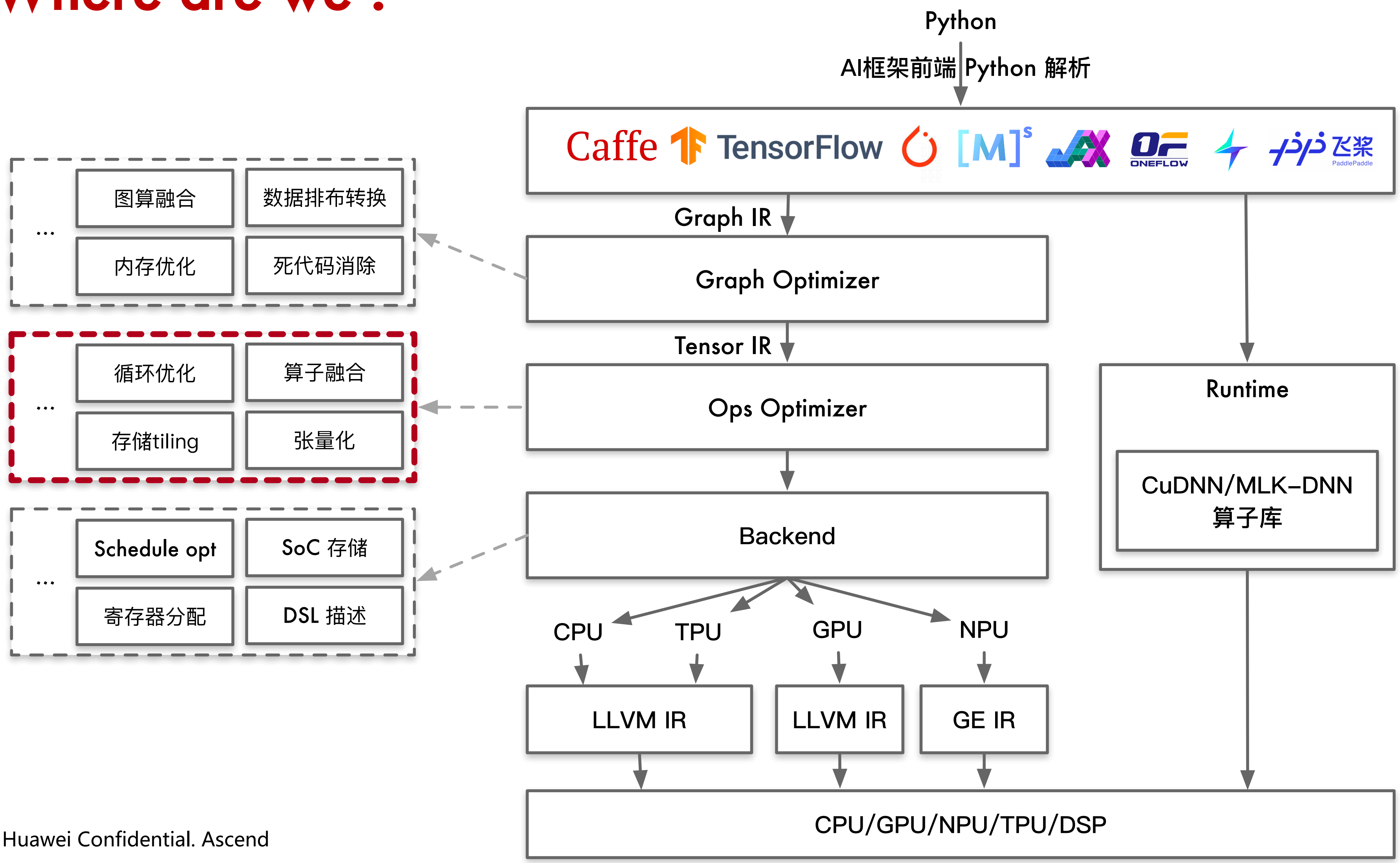
ZOMI

Talk Overview

I. AI 编译器后端优化

- 后端优化概念
- 算子计算与调度
- 算子调度优化
- Auto-Tuning
- Polyhedral

Where are we ?

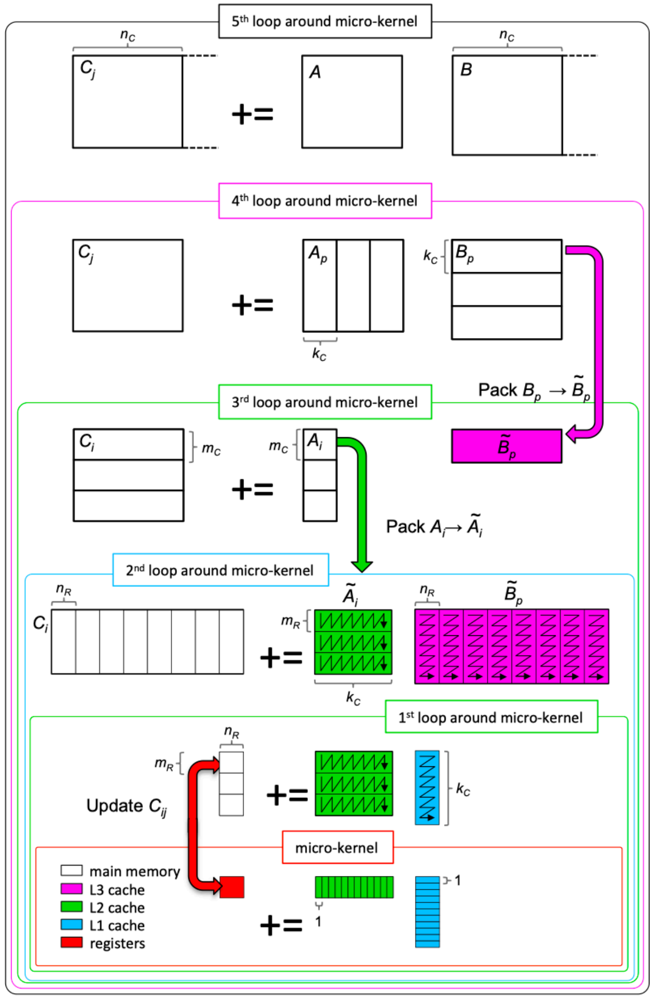


Python
AI框架前端 Python 解析
Caffe
TensorFlow
ONEFLOW
飞桨
PaddlePaddle
Graph IR
Graph Optimizer
Tensor IR
Ops Optimizer
Backend
CPU
TPU
GPU
NPU
LLVM IR
LLVM IR
GE IR
CPU/GPU/NPU/TPU/DSP
Runtime
CuDNN/MLK-DNN
算子库
…
图算融合
数据排布转换
内存优化
死代码消除
…
循环优化
算子融合
存储tiling
张量化
…
Schedule opt
SoC 存储
寄存器分配
DSL 描述

Auto Tuning

- 对于给定的程序和目标架构，找到最优的编译优化方法。使用哪些优化方法？选择什么参数集？用什么顺序应用优化方法以达到最佳性能？

BLISlab: A Sandbox for Optimizing GEMM



Loop 5 $\quad$ **for** $j_c = 0 : n-1$ **steps of** n_c

$\qquad \mathcal{J}_c = j_c : j_c + n_c - 1$

Loop 4 $\qquad$ **for** $p_c = 0 : k-1$ **steps of** k_c

$\qquad\quad \mathcal{P}_c = p_c : p_c + k_c - 1$

$\qquad\quad B(\mathcal{P}_c, \mathcal{J}_c) \to B_c$ $\qquad$ // Pack into B_c

Loop 3 $\qquad\quad$ **for** $i_c = 0 : m-1$ **steps of** m_c

$\qquad\qquad \mathcal{I}_c = i_c : i_c + m_c - 1$

$\qquad\qquad A(\mathcal{I}_c, \mathcal{P}_c) \to A_c$ $\qquad$ // Pack into A_c

$\qquad\qquad$ // Macro-kernel

Loop 2 $\qquad\qquad$ **for** $j_r = 0 : n_c - 1$ **steps of** n_r

$\qquad\qquad\quad \mathcal{J}_r = j_r : j_r + n_r - 1$

Loop 1 $\qquad\qquad\quad$ **for** $i_r = 0 : m_c - 1$ **steps of** m_r

$\qquad\qquad\qquad \mathcal{I}_r = i_r : i_r + m_r - 1$

$\qquad\qquad\qquad$ // Micro-kernel

Loop 0 $\qquad\qquad\qquad$ **for** $k_r = 0 : k_c - 1$

$$C_c(\mathcal{I}_r, \mathcal{J}_r) \mathrel{+}= A_c(\mathcal{I}_r, k_r) \; B_c(k_r, \mathcal{J}_r)$$

$\qquad\qquad\qquad$ **endfor**

$\qquad\qquad$ **endfor**

$\qquad\quad$ **endfor**

$\qquad$ **endfor**

endfor

Auto Tuning

- 优化选择 Optimize Selection
- 优化顺序 Optimize Sequence

Auto Tuning in AI

- **更低实验开销**：1）聚焦算子或者 kernel 级别的优化，而非整个程序。2）Cost Model 在CPU上模拟NPU执行，训练和推理推理的模拟速度要求足够快。
- **特定领域结构**：针对神经网络算子或者 kernel 级别的优化，1）主要是高度循环化、张量化、并行化特点进行优化；2）大量相类似的算子计算模式。

Question?

- 如何可以让机器匹配手写优化性能？

1. 建立一个足够大的搜索空间，保证可能的人工手写优化全部包含在这个搜索空间里面
2. 快速地搜索这个这个空间，获取优化的实现

Auto Tuning in AI

1. 参数化 Parameterization
2. 成本模型 Cost Model
3. 搜索算法 Search Algorithm

```
for (int n = 0; n < o_n; ++n) {
        for (int c = 0; c < o_c; ++c) {
            for (int j = 0; j < o_h; ++j) {
                for (int i = 0; i < o_w; ++i) {
                    int d_start = n * i_c * i_h * i_w + j * i_w + i;
                    int temp = 0;
                    for (int kk = 0; kk < k_c; ++kk) {
                        for (int kj = 0; kj < k_h; ++kj) {
                            for (int ki = 0; ki < k_w; ++ki) {
                                int k_idx = kk * k_h * k_w + kj * k_w + ki;
                                int d_idx = d_start + kk * i_h * i_w + kj * i_w + ki;
                                temp += inputs->data[d_idx] * kernel->data[k_idx];
                            }
                        }
                    }
                    res[n * o_c * o_h * o_w + j * o_w + i] = temp;
                }
            }
        }
```

Auto Tuning in AI

1. 参数化 Parameterization
2. 成本模型 Cost Model
3. 搜索算法 Search Algorithm

- 参数化 Parameterization：对调度优化问题进行建模，参数化优化空间一般由可参数化变换（Loop）的可能参数取值组合构成，因此需要调度原语进行参数化表示。Halide 将算法和调度解耦，TVM 提供调度模板。

```
bx, tx = s[C].split(C.op.axis[0], factor=64)
s[C].bind(bx, tvm.te.thread_axis("blockIdx.x"))
s[C].bind(tx, tvm.te.thread_axis("threadIdx.x"))
fadd = tvm.build(s, [A, B, C], target)
```

Auto Tuning in AI

1. 参数化 Parameterization
2. 成本模型 Cost Model
3. 搜索算法 Search Algorithm

- 成本模型 Cost Model：用来评价某一参数化下的调度性能，根据对调度额评价来指导最搜索到最优的调度策略。可以从运行时间、内存占用、编译后指令数来评价。实现方式主要有1）基于NPU硬件的黑盒模型；2）基于模拟的预定义模型；3）ML-Base 模型，通过机器学习模型来对调度性能进行预测。

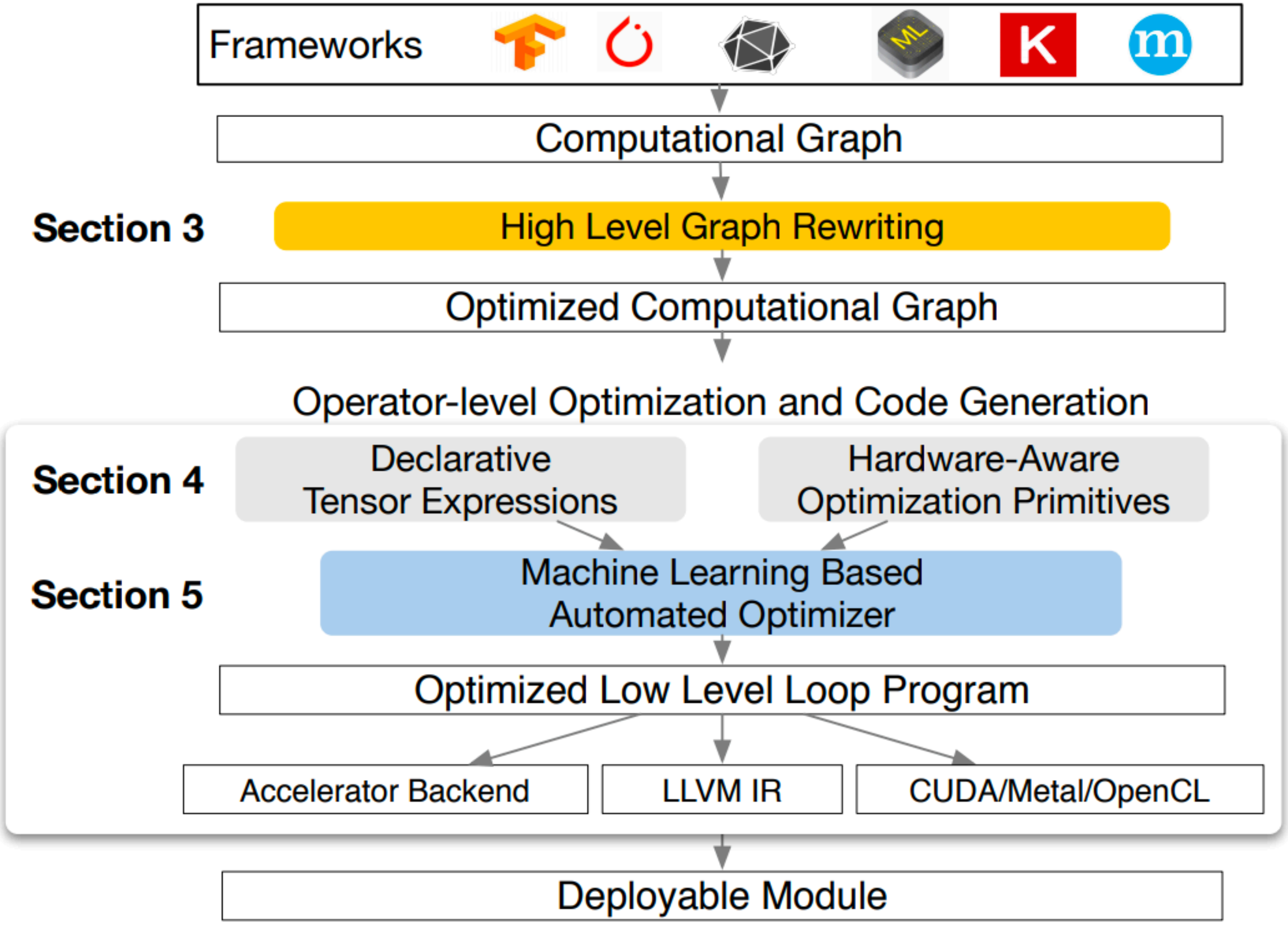
Auto Tuning in AI

1. 参数化 Parameterization
2. 成本模型 Cost Model
3. 搜索算法 Search Algorithm

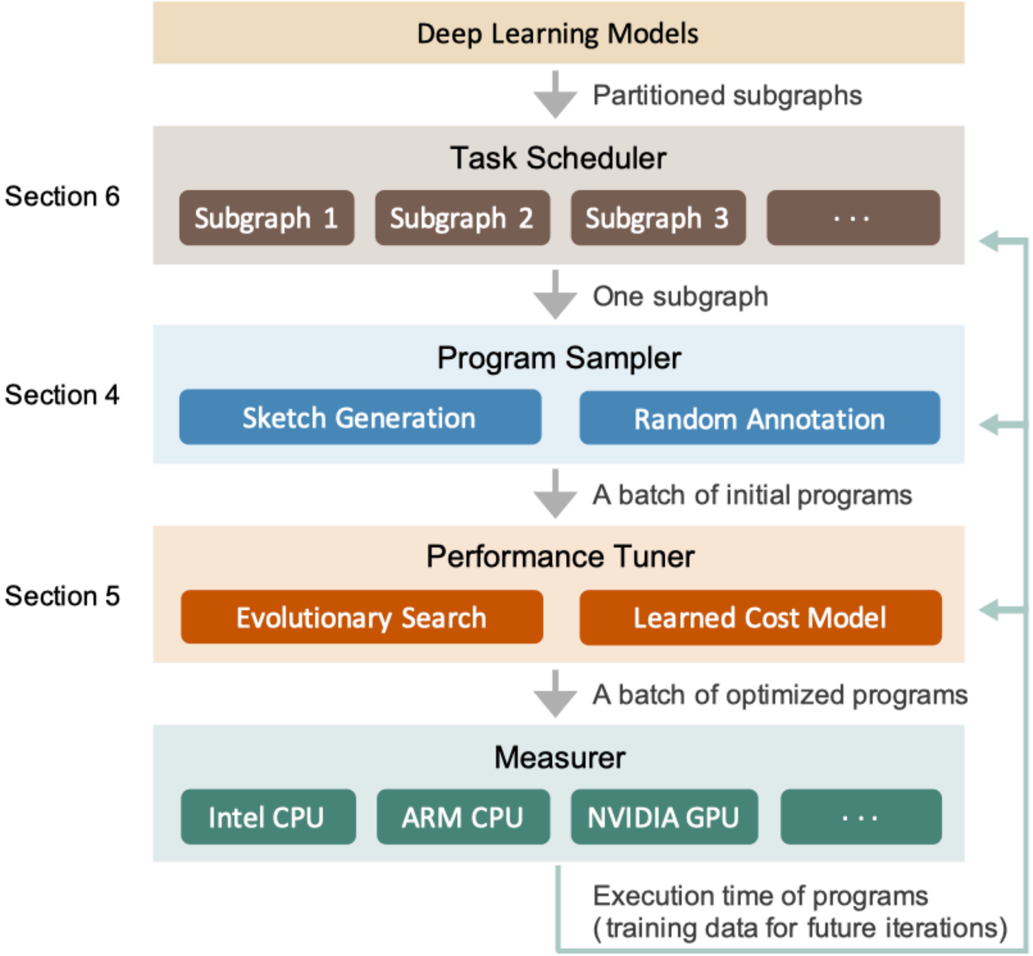
- 搜索算法 Search Algorithm：确定初始化和搜索空间后，在搜索空间找找到达到性能最优的参数配置。常用的搜索算法有1）遗传算法、2）模拟退火算法、3）强化学习等。

Auto Tuning in AI

1. 参数化 Parameterization
2. 成本模型 Cost Model
3. 搜索算法 Search Algorithm



Ansor : Generating High-Performance Tensor Programs for Deep Learning



Inference

- Li, Mingzhen, et al. "The deep learning compiler: A comprehensive survey." *IEEE Transactions on Parallel and Distributed Systems* 32.3 (2020): 708-727.
- Ning, Chao, and Fengqi You. "Optimization under uncertainty in the era of big data and deep learning: When machine learning meets mathematical programming." *Computers & Chemical Engineering* 125 (2019): 434-448.
- Xu, Zhiying, et al. "ALT: Breaking the Wall between Graph and Operator Level Optimizations for Deep Learning Compilation." *arXiv preprint arXiv:2210.12415* (2022).
- Baghdadi, Riyadh, et al. "A deep learning based cost model for automatic code optimization." *Proceedings of Machine Learning and Systems* 3 (2021): 181-193.
- Chen, Tianqi, et al. "TVM: end-to-end optimization stack for deep learning." *arXiv preprint arXiv:1802.04799* 11.2018 (2018): 20.
- Purkayastha, Arnab A., et al. "LLVM-based automation of memory decoupling for OpenCL applications on FPGAs." *Microprocessors and Microsystems* 72 (2020): 102909.
- Loop Optimizations: how does the compiler do it? https://johnysswlab.com/loop-optimizations-how-does-the-compiler-do-it/
- Loop Optimizations: taking matters into your hands https://johnysswlab.com/loop-optimizations-taking-matters-into-your-hands/
- Understanding Latency Hiding on GPUs - UC Berkeley EECS. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-143.pdf
- Reinforcement Learning and Adaptive Sampling for Optimized DNN Compilation, https://arxiv.org/abs/1905.12799

BUILDING A BETTER CONNECTED WORLD

THANK YOU