推理引擎 OpenPPL 实战训练营

商汤 sensetime

OpenPPL CUDA技术解析

2022年1月13日

主讲人介绍

李天健

商汤科技异构计算工程师
OpenPPL CUDA 架构负责人

- 博士毕业于上海交通大学计算机系
- 研究方向为计算机体系结构，在 TCAD、DAC、ICCD、ICCAD、ITC 等国际会议期刊上发表多篇论文
- 目前在商汤科技高性能计算部门负责 CUDA 架构的 PPL 研发与优化

课程提纲

第四期课程将为大家介绍开源高性能深度学习推理引擎是如何炼成的，并重点探讨 CUDA 引擎相关内容，对模型推理的前中后端的技术进行解析，包括图融合、内存管理、Runtime 优化等。

1、OpenPPL CUDA 简介

2、OpenPPL CUDA 技术解析

3、OpenPPL CUDA 性能

实战训练营

推理框架这件小事儿
OpenPPL CUDA技术分享

李天健

2022.1.13

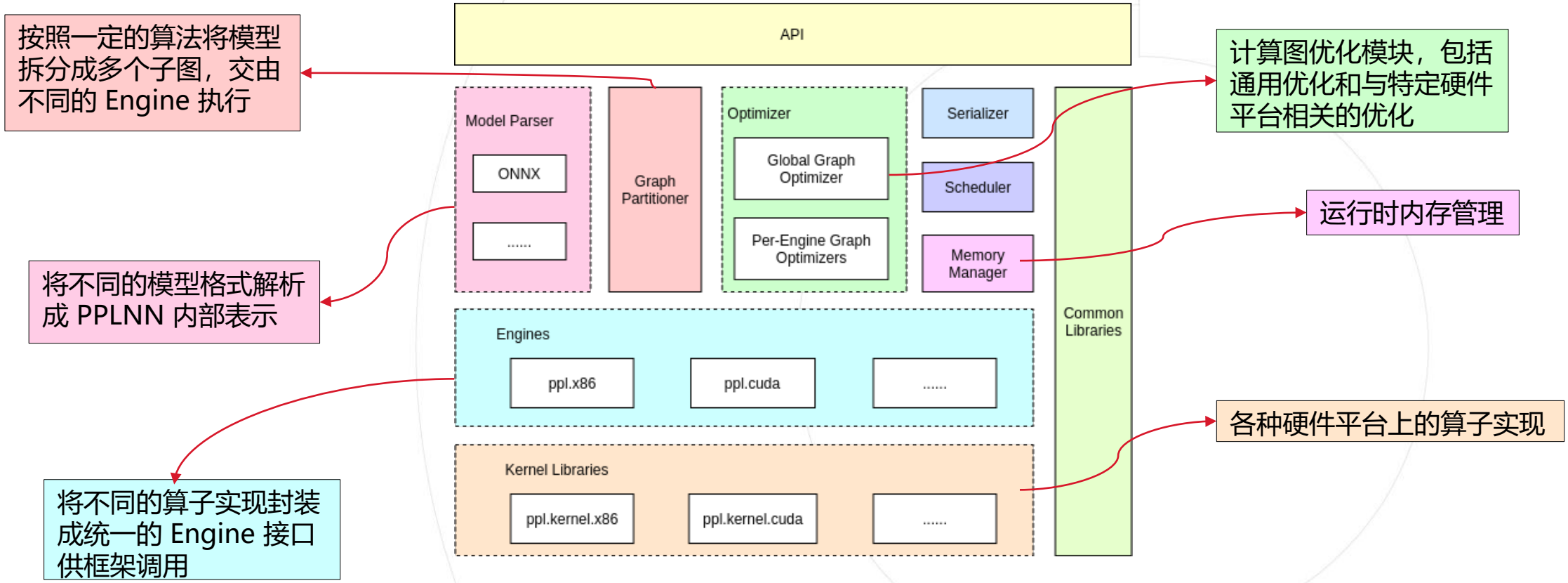
商汤 sensetime

Outline

Part 1	推理引擎概述
Part 2	OpenPPL CUDA技术解析
Part 3	OpenPPL CUDA使用方式

推理框架概述

- OpenPPL 是商汤HPC团队研发的支持多后端的深度学习推理部署引擎，目前已经支持的后端包括X86 CPU、CUDA GPU、RISC-V。



https://github.com/openppl-public/ppl.nn

商汤 sensetime

Outline

OpenPPL CUDA技术解析

- Graph Common 优化
  - 卷积融合：Conv+Activation(relu/prelu/clip/sigmoid..) , Conv+Add
  - 并行Op融合：Cast, Squeeze …
- Arch-Based 优化
  - Shape融合：Shape+Gather, Shape+Concat, Shape+Div
  - 特殊Op融合：channel shuffle
  - Constant Fold
- 排布推导

Shape融合

- ONNX Shape算子
  - 算子频率高，相关操作包含大量Kernel，影响推理性能；
  - 相关操作多为Memory操作，对数据排布敏感，影响后续的数据排布推导；
- Shape算子融合
  - 构建系数矩阵替换Shape算子
  - Pattern匹配，更新系数矩阵，完成融合

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$M\ gather \rightarrow$$

$$M_1 = (0 \quad 1 \quad 0 \quad 0 \quad 0)\ concat\ M_2 = (0 \quad 0 \quad 1 \quad 0 \quad 0) \rightarrow$$

$$M_3 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

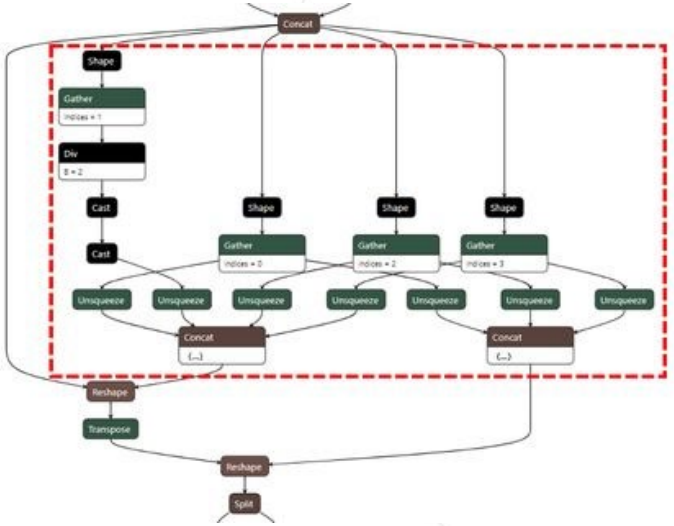
$$v = (N \quad C \quad H \quad W \quad 1)$$

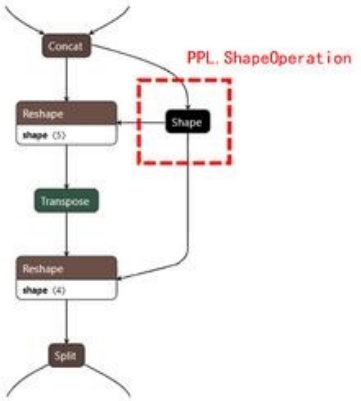
$$\mathrm{R}{=}v * M_3^T = (C, H)$$

https://github.com/openppl-public/ppl.nn

Shape融合

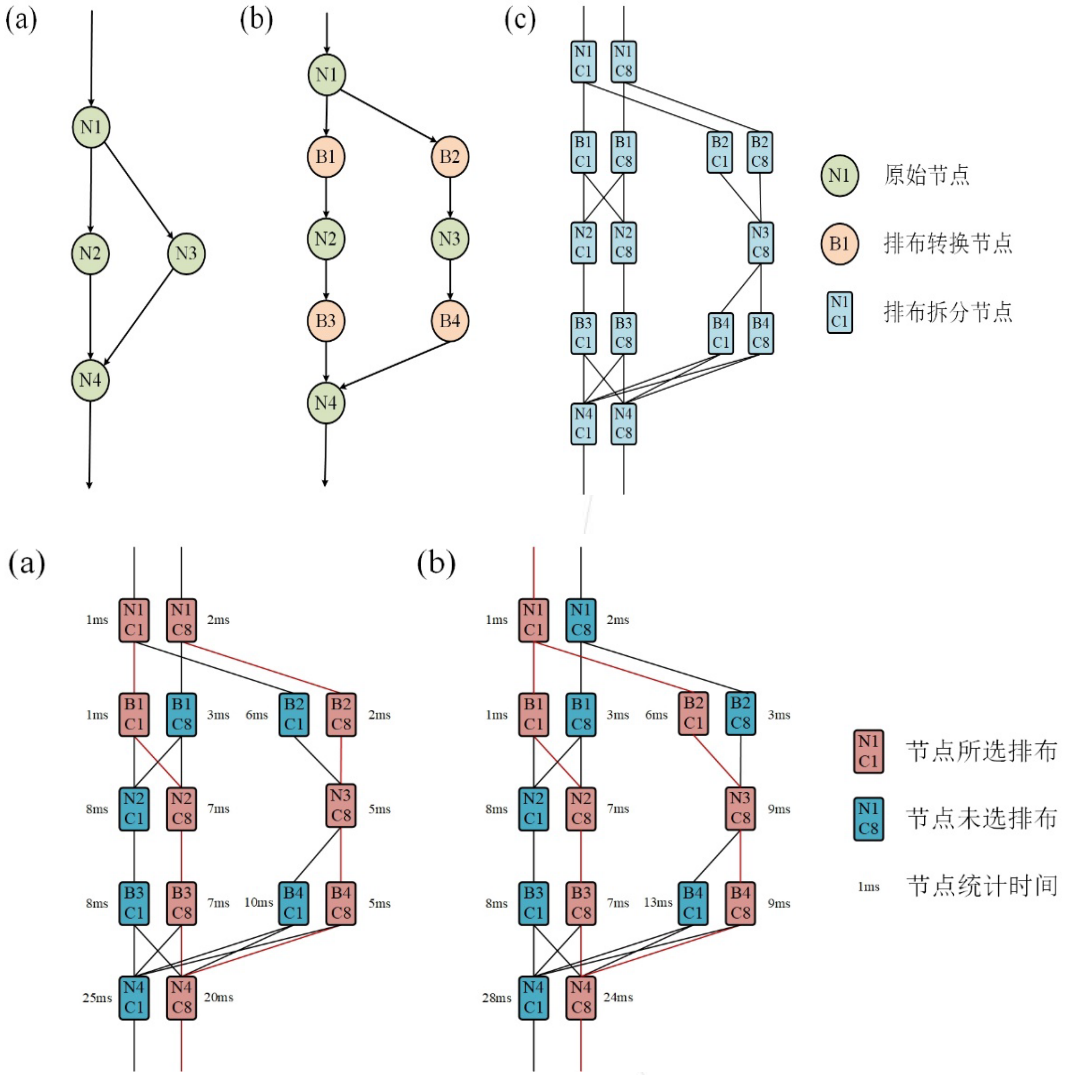
- ONNX Shape算子
  - 算子频率高，相关操作包含大量Kernel，影响推理性能；
  - 相关操作多为Memory操作，对数据排布敏感，影响后续的数据排布推导；
- Shape算子融合
  - 构建系数矩阵替换Shape算子
  - Pattern匹配，更新系数矩阵，完成融合
- 融合效果





https://github.com/openppl-public/ppl.nn

排布选择

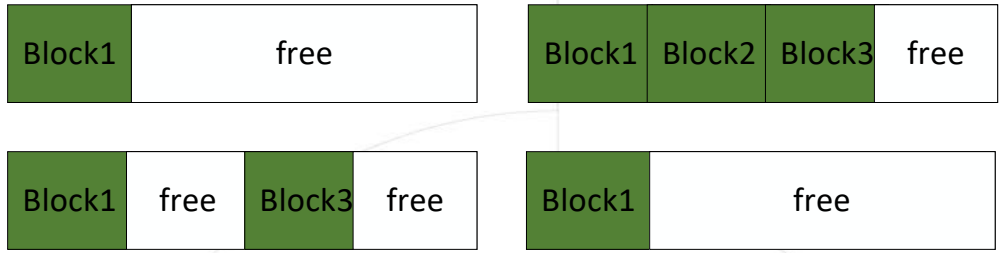


- 模型转换
  - 插入排布选择算子
  - 算子拆分
- 执行时间统计
  - 时间统计的真实性
- 算子排布选择
  - 最短路径回溯算法

https://github.com/openppl-public/ppl.nn

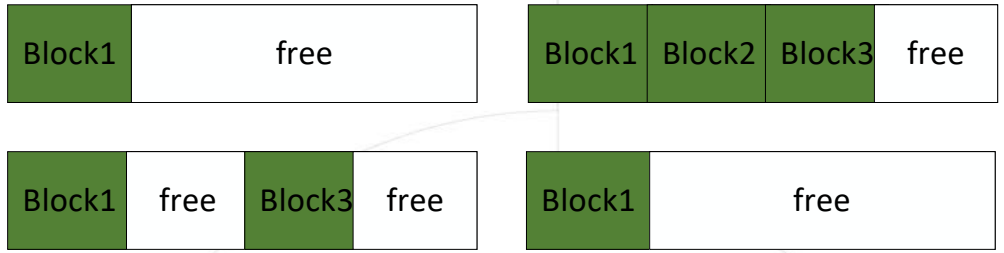
内存管理

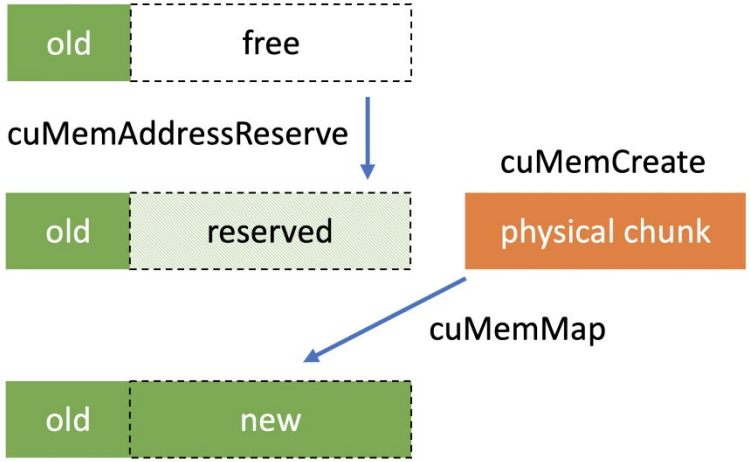
- Compact Memory Manager
  - 维护空闲/使用中的Block列表
  - Alloc->空闲列表中最适合的
  - Free ->Block的前继和后继的地址如果连续，可以将多个Free Block 合并为一个大的



内存管理

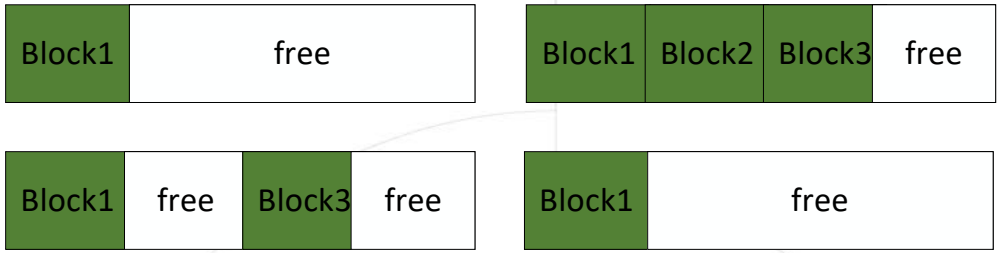
- Compact Memory Manager
  - 维护空闲/使用中的Block列表
  - Alloc->空闲列表中最适合的
  - Free ->Block的前继和后继的地址如果连续，可以将多个Free Block 合并为一个大的
- CUDA VM Management API
  - cuNemAdressReserve
  - cuMemMap

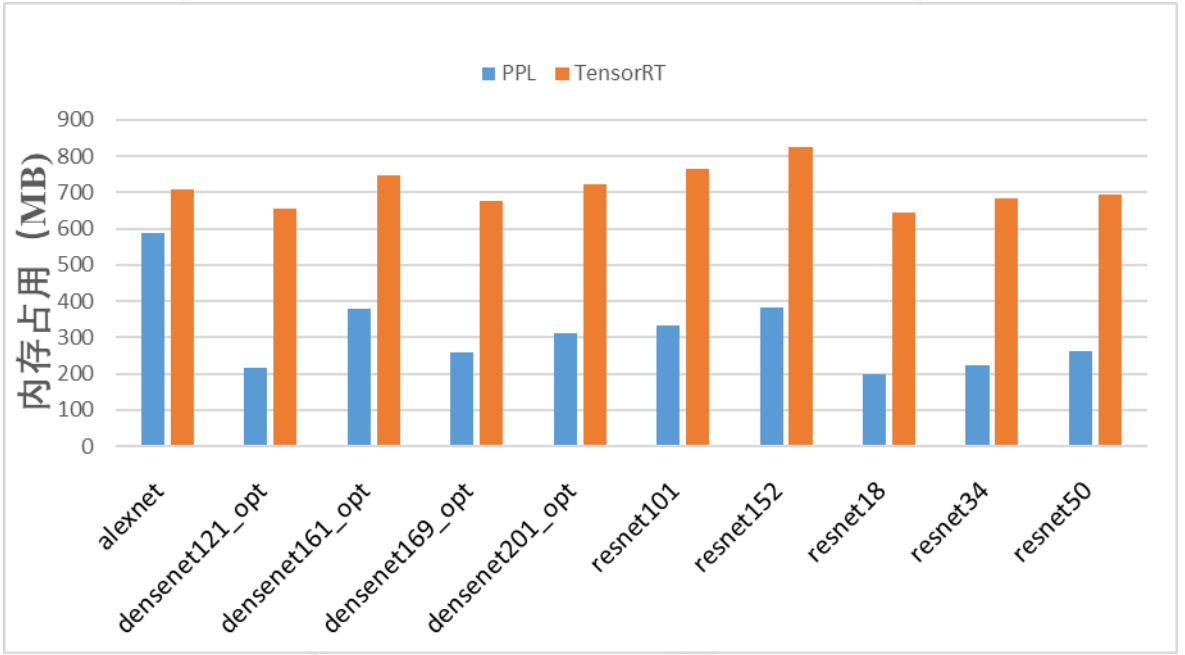




内存管理

- Compact Memory Manager
  - 维护空闲/使用中的Block列表
  - Alloc->空闲列表中最适合的
  - Free ->Block的前继和后继的地址如果连续，可以将多个Free Block 合并为一个大的
- CUDA VM Management API
  - cuNemAdressReserve
  - cuMemMap
- 整合到一起
  - 显存占用大幅下降
  - 在vGPU下API支持不完善



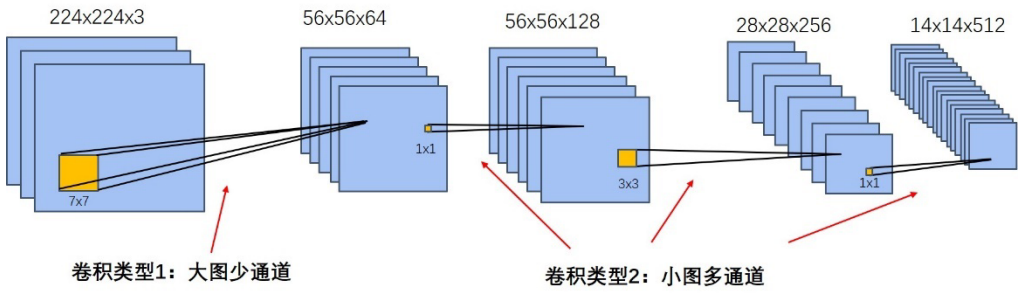


https://github.com/openppl-public/ppl.nn

Kernel Optimization

- 卷积优化
  - TensorCore 硬件计算单元，NV提供多条mma指令对TensorCore进行编程
  - 数据排布，目前OpenPPL CUDA的主要数据排布包括NHWC
  - 卷积算法，核心算法为基于偏移地址的隐式矩阵乘法

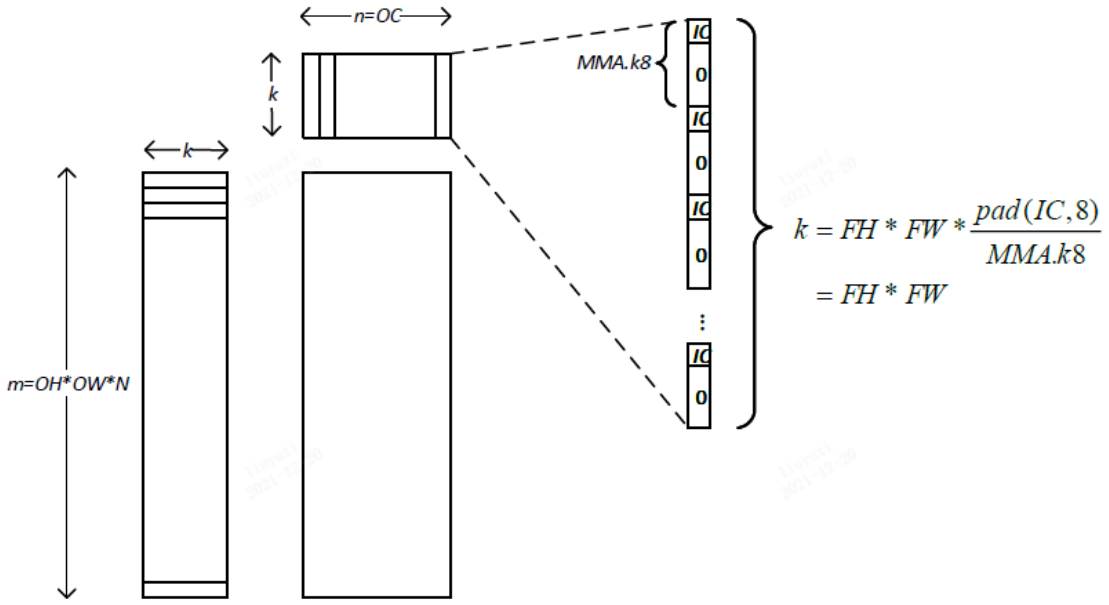
- 卷积分类：
  - 大图小通道
  - 小图大通道
  - 大图大通道




```
for (i = 0; i < OC; i += 1) {
    oc = i;
    for (j = 0; j < N × OH × OW; j += 1) {
        n = j / (OH × OW);
        oh = (j % (OH × OW)) / OW;
        ow = (j % (OH × OW)) % OW;
        for (k = 0; k < FH × FW × IC; k += 1) {
            // channel– interleaved sliding order
            icx   = k / (FH × FW × X);
            k_res = k % (FH × FW × X);

            fh = k_res / (FW × X);
            fw = (k_res % (FW × X)) / X;
            x  = (k_res % (FW × X)) % X;

            ic = icx × X + x;
            ih = oh × stride − pad + fh;
            iw = ow × stride − pad + fw;

            out_{n,oh,ow,oc} += in_{n,ih,iw,ic} × filter_{oc,fh,fw,ic};
        }
    }
}
```

大图小通道 – IDXN算法

- 卷积特点
  - OH/OW很大，保证Block的数量足够多
  - 输入通道K比较小，特别是小于mma.MxNxK（1688/8816）中K，需要显示地填充0去适配TensorCore,导致整体的利用率较低



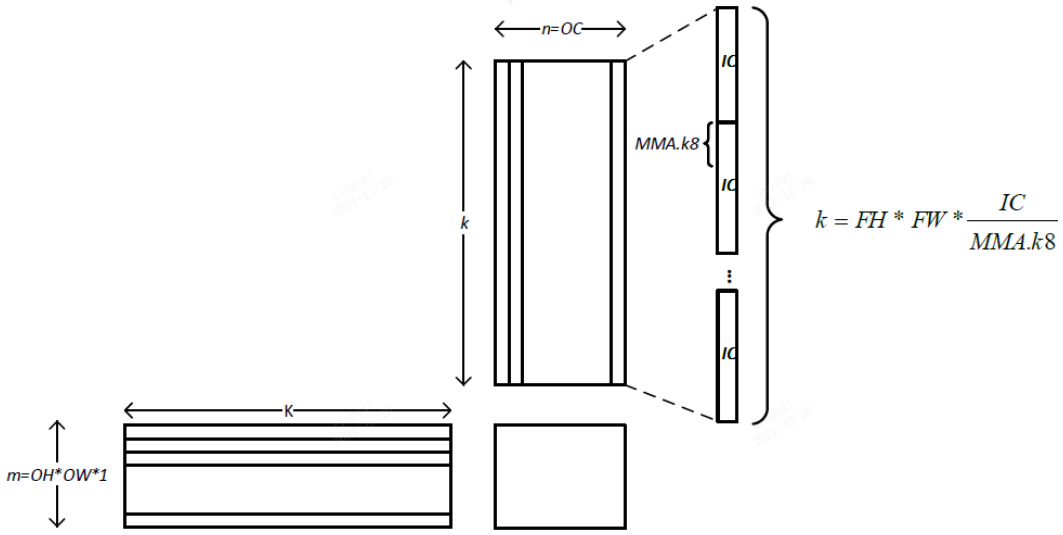
- IDXN算法
  - 修改计算顺序
  - 不使用Shared Memory进行暂存数据，K的维度较小，循环次数很少，使用share memory进行缓存和double buffer带来的收益较少，还会增加数据读取的延迟

Algorithm 1: IDXN Algorithm.

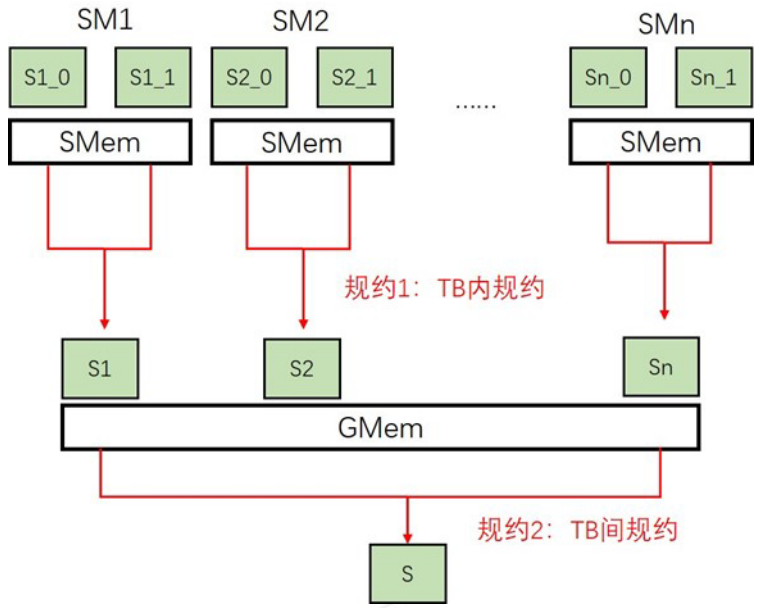
```
1  input_frag[w_m][w_k];
2  filter_frag[w_n][w_k];
3  output_frag[w_m][w_n];
   /* calculate accumulative delta LUT; */
4  __shared__ in_id_smem[];
5  if tid ≤ FH × FW then
6  |  in_id_smem[tid] ← calculate_offset();
7  end
8  for kgroup ← 0 to k_idxn by 1 do
   |  /* load into registers directly; */
9  |  input_frag[][] ← w_m × w_k of input;
10 |  filter_frag[][] ← w_n × w_k of filter;
11 |  foreach 16 × 8 matrix in output_frag do
12 |  |  output_frag[][] += input_frag[][] * filter_frag[][];
13 |  end
14 end
   /* no sync, store to gmem directly; */
15 output ← output_frag[][]
```

小图大通道 – 2PK算法

- 卷积特点
  - OH/OW尺寸很小（小batch），gridsize较小，SM的占用率较低
  - 输入通道K很大，SplitK的算法解决并行性不足的问题



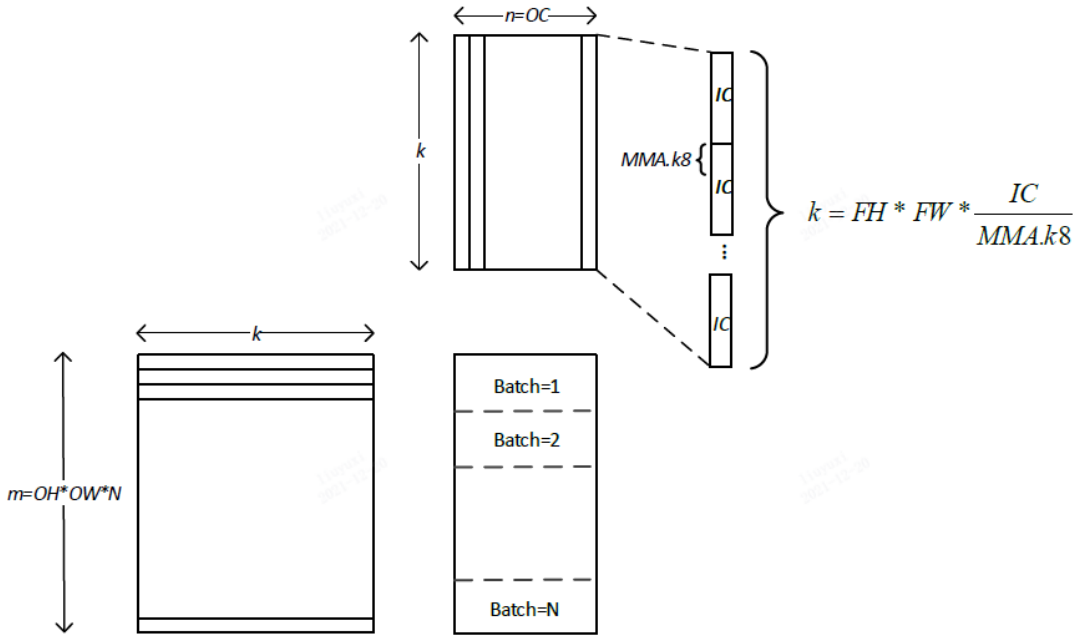
- 2SPK算法
  - 双层维度规约，Inter-Block/Intra-Block Reduce
  - 通道层IC进行划分外，也对卷积核FW*FH进行划分



https://github.com/openppl-public/ppl.nn

大图大通道 – Swizzle

- 卷积特点
  - OH/OW尺寸大，输入输出通道很大，计算密集型
  - 卷积核通道相较于特征图小很多，可以采用将Filter常驻L2-Cache的方式优化



- Swizzle算法
  - Swizzle， 优化L2-Cache
  - 分组牛耕

Algorithm 2: SWZL Algorithm.

```
1  __shared__ input_smem[b_m][b_k];
2  __shared__ filter_smem[b_n][b_k];
3  input_frag[w_m][w_k];
4  filter_frag[w_n][w_k];
5  output_frag[w_m][w_n];
   /* swizzle TB ID; */
6  tbx, tby ← Boustrophedon_map(blockIdx.x, blockIdx.y);
7  for k ← 0 to k_nhwc by b_k do
       /* load data into smem; */
8      input_smem ← b_m × b_k of input;
9      filter_smem ← b_n × b_k of filter;
10     for c ← 0 to b_k by w_k do
           /* load data from smem by ldmatrix; */
11         input_frag[][] ← w_m × w_k of input_smem;
12         filter_frag[][] ← w_n × w_k of filter_smem;
13         foreach 16 × 8 matrix in output_frag do
14             output_frag[][]+=input_frag[][]*filter_frag[][];
15         end
16     end
17 end
18 __syncthreads();
   /* rearrange for vectorized output; */
19 __shared__ out_smem[] ← output_frag[][];
20 __syncthreads();
21 outv4[] ← out_smem[];
22 output ← outv4[];
```

运行时编译

- 卷积配置多样
- 卷积实现方式分块复杂
  - 计算库体积
  - 初始化时间
- 融合对性能影响
  - Local Memory
  - 3000 vs 8000 行SASS

```
(base) litianjian@4miij32ad3i4:~/workspace/test/accuracy_
ptxas info    : 0 bytes gmem
ptxas info    : Compiling entry function '_Z52nvSwzlConv_
ptxas info    : Function properties for _Z52nvSwzlConv_hr
    48 bytes stack frame, 44 bytes spill stores, 44 bytes
ptxas info    : Used 255 registers, 32768 bytes smem, 157
```

```
(base) litianjian@4miij32ad3i4:~/workspace/test/accura
ptxas info    : 0 bytes gmem
ptxas info    : Compiling entry function '_Z52nvSwzlCo
75'
ptxas info    : Function properties for _Z52nvSwzlConv
    0 bytes stack frame, 0 bytes spill stores, 0 bytes
ptxas info    : Used 254 registers, 32768 bytes smem,
```

https://github.com/openppl-public/ppl.nn

运行时编译

- 定义卷积计算解空间
  - Warp分块： Per-Warp
  - Block分块：Per-CTA 的warp分块
  - K维度分块： K/S表示Block内规约的倍数
  - Pipeline stage分块：double buffer的使用
  - 解R： （B_m, B_n, W_m, W_n, K, S, pipe）

M \ N	8	16	32	64
16	16x8	16x16	16x32	16x64
32	32x8	32x16	32x32	32x64
64	64x8	64x16	64x32	64x64
128	128x8	128x16	128x32	\

M \ N	1	2	4
1	1x1	1x2	1x4
2	2x1	2x2	2x4
4	4x1	4x2	\

K / S	8	16	32
1x	k8_s8	k16_s16	k32_s32
2x	k16_s8	k32_s16	k64_s32
4x	k32_s8	k64_s16	k128_s32

https://github.com/openppl-public/ppl.nn

运行时编译

- 定义卷积计算解空间
- 代码生成与封装
  - 宏定义生成不同的代码模块
  - 根据卷积的一个解，封装对应的卷积代码
  - 依据融合信息，生成融合代码

运行时编译

- 定义卷积计算解空间
- 代码生成与封装
- 快速算法选择
  - 数据分析得到经验公式F，F根据卷积和输入的配置，得到候选集合
  - JIT候选集合，选出最优解

运行时编译

- 定义卷积计算解空间
- 代码生成与封装
- 快速算法选择
- 运行时编译机制
  - CUDA提供了NVRTC接口
  - 添加cuda_module、cuda_module_manager多个模块对运行时编译进行管理

运行时编译

- 定义卷积计算解空间
- 代码生成与封装
- 快速算法选择
- 运行时编译机制
- Runtime
  - Manager维护一个<Op, Module>的表
  - 调用Kernel Launch执行Kernel

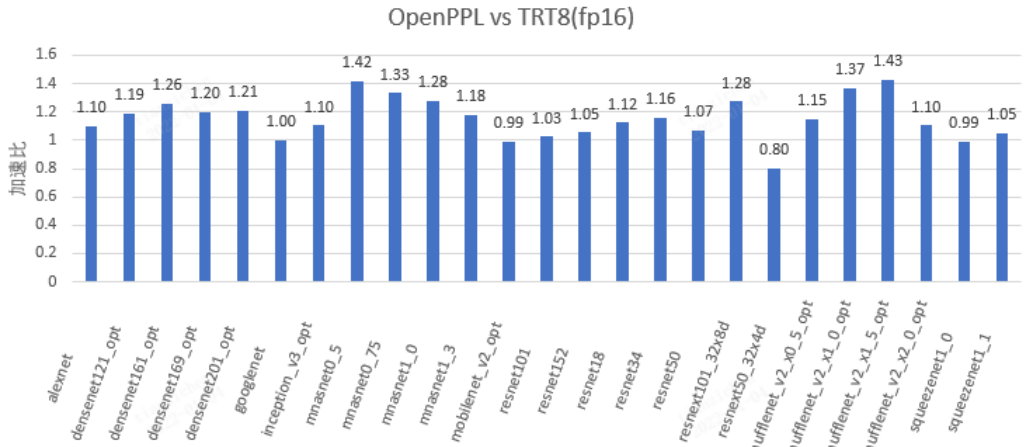
运行时编译

- 运行时编译优势及性能对比

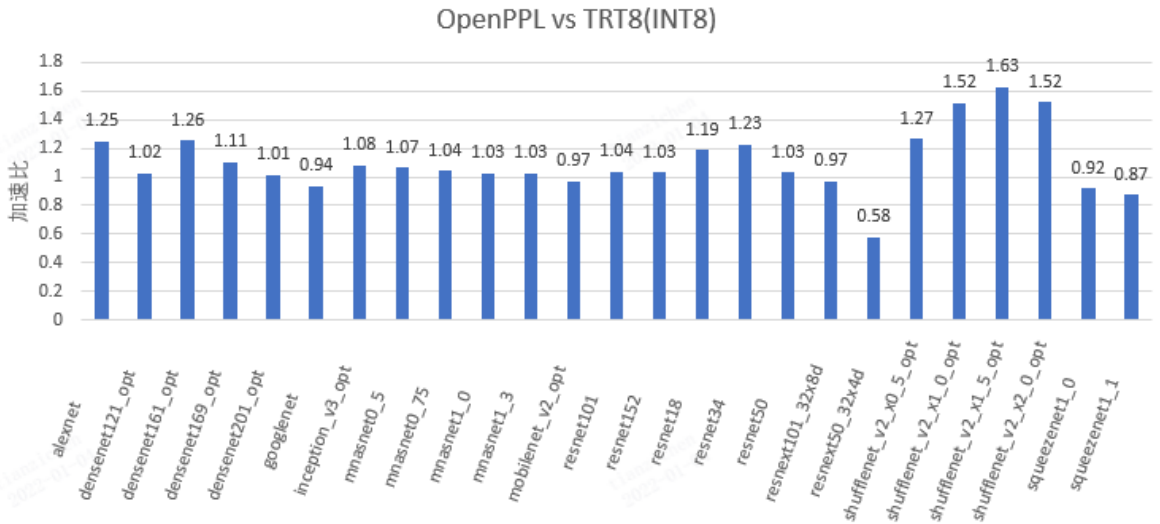
	运行时编译	静态编译
计算库体积 （OpenPPL编译Arch75架构）	库体积为10M	库体积为200M
初始化时间 （Resnet50，Batch=32）	预处理时间大约在200秒左右	预处理时间超过1000秒
部署灵活性	支持在不同设备上在线选择算法，无需保存算法信息	依据不同设备保存不同算法，设备数较多时需要对保存的算法进行管理
推理性能	网络测试结果显示相比静态编译，运行时编译有5%的性能提升	

性能结果

- Tesla T4上OpenPPL 与 TensorRT 8.0的性能比较



CUDA FP16



CUDA INT8

https://github.com/openppl-public/ppl.nn

商汤 sensetime

Outline

Part 1	推理引擎概述
Part 2	OpenPPL CUDA技术解析
Part 3	OpenPPL CUDA使用方式

PPLCUDA 使用方式

- 混合精度支持

$$/pplnn\ --use-cuda\ --onnx-model\ model.onnx\ --inputs\ input.bin\ --in\ -shapes\ n\_c\_h\_w\ --quant-file\ quant.json\ [--warmup-iterations\ m]\ --enable-profiling$$

```
quant info: {
    input.1 : {
        bit width : 8,
        per channel : false,
        sym : false,
        algorithm : KL,
        quant flag : true,
        scale : 0.020705883175719017,
        zero point : 128.0,
        tensor max : 2.640000104904175,
        tensor min : -2.640000104904175,
        q max : 255,
        q min : 0
    }
},
op info: {
    Conv 0 : {
        data type : INT8
    }
}
```

知乎 @李天健

- 性能测试

$$./pplnn\ --use-cuda\ --onnx-model\ model.onnx\ --inputs\ input.bin\ --in\ -shapes\ n\_c\_h\_w\ --kernel-type\ INT8\ [--warmup-iterations\ m]\ --enable-profiling$$

https://github.com/openppl-public/ppl.nn

Q&A

OpenPPL 微信公众号

OpenPPL QQ 交流群

- OpenPPL 主页：https://openppl.ai/
- OpenPPL GitHub 主页：https://github.com/openppl-public
- OpenPPL 知乎账号：https://www.zhihu.com/people/openppl

商汤 sensetime

团队介绍

- HPC 团队肩负着商汤最核心计算系统引擎的研发重任，同时定义下一代计算系统的规格和方案。作为「数据 - 算法 - 算力」的 AI 平台闭环中的算力环节，为 AI 技术在行业落地的过程中提供高速度、高效能、功能完善和稳定可靠的算力基础设施。

- 目前团队支持公司大量落地业务，产品部署量超数亿，服务上亿用户；同时承接国家重大科技创新项目以及推进开源计划，欢迎同学们加入！👏

工作日常

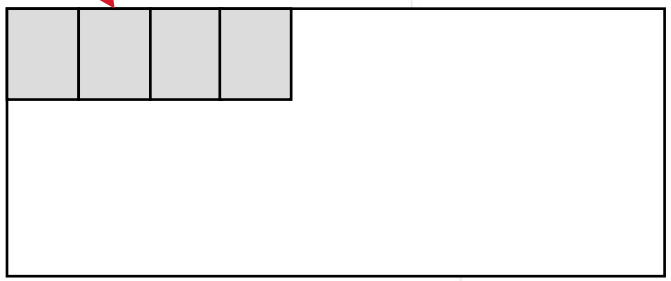
1. 【传统艺能】高性能计算、异构计算
2. 【涉及的处理器体系结构】CPU, GPU, DSP, NPU
3. 【日常玩具】
   - GPU 和 CPU 管够
   - 各种新架构开发板
   - 各种没上市的手机
4. 【黑科技】
   - 体系结构逆向工程
   - 面向极致性能调优的后端编译优化技术
5. 【开源计划】OpenPPL
   - 全自研深度学习推理引擎，挑战业内顶级性能
   - https://github.com/openppl-public
6. 【学术成果】ISCA, ASPLOS, NIPS, DAC, IPDPS

岗位连接：https://zhuanlan.zhihu.com/p/429466308

补充材料

(blockIdx.x, blockIdx.y)

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)



(0,0)	(3,0)	(0,2)	(3,2)	(0,4)	(3,4)	(0,6)
(0,1)	(3,1)	(0,3)	(3,3)	(0,5)	(3,5)	(1,6)
(1,0)	(4,0)	(1,2)	(4,2)	(1,4)	(4,4)	(2,6)
(1,1)	(4,1)	(1,3)	(4,3)	(1,5)	(4,5)	(3,6)
(2,0)	(5,0)	(2,2)	(5,2)	(2,4)	(5,4)	(4,6)
(2,1)	(5,1)	(2,3)	(5,3)	(2,5)	(5,5)	(5,6)