推理引擎 OpenPPL 实战训练营

商汤 sensetime

模型“大小”与推理速度的那些事儿

2022年1月6日星期四

课程安排

课程安排	主讲人	课程时间
第一期：商汤自研AI推理引擎 OpenPPL 的实践之路	高洋	2021年12月07日
第二期：编程工作坊：基于 OpenPPL 的模型推理与应用部署	欧国宇	2021年12月16日
第三期：OpenPPL之通用架构下的性能优化概要	许志耿	2021年12月29日
第四期：模型大小与推理速度的那些事儿	**田子宸**	**2022年01月06日**
第五期：性能调优实战（CUDA篇）	李天健	2022年01月13日
第六期：性能调优实战（x86篇）	梁杰鑫	敬请期待
第七期：OpenPPL+RISC-V 指令集初探	焦明俊/杨阳	敬请期待
第八期：OpenPPL 在 ARM Server 上的技术实践	许志耿/邱君仪	敬请期待
第九期：量化工具实践	纪喆	敬请期待

「商汤学术」公众号

可以回复“抽奖”试试哦

主讲人介绍

田子宸

商汤科技异构计算工程师
PPL CPU & 加速器开发人员

- 本硕毕业于浙江大学
- 目前在商汤科技高性能计算部门负责参与 CPU、加速器等架构方向的 PPL NN/CV 研发与优化

• 课程提纲

第四期课程将探讨影响模型推理速度的因素与设计高效深度学习推理应用，描述模型“大小”的评价方法与多种评价指标，并分析这些指标对模型实际部署落地时推理速度的影响。

1. 模型“大小”的四种描述指标
2. 计算量、访存量与推理速度的关系
3. 影响推理速度的其他因素
4. 如何实现高效的深度学习推理应用

实战训练营

模型“大小”与推理速度的那些事儿

——影响模型推理速度的因素与设计高效深度学习推理应用的探讨

田子宸

2022/1/6

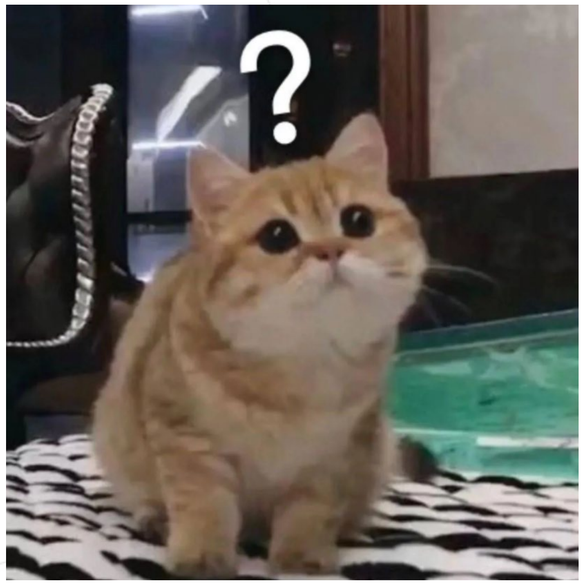
OpenPPL 跨年更新内容

- CUDA 后端大幅优化，增加 INT8 量化推理支持
- 发布了模型离线量化工具 PPQ
- X86 进行了大量优化，并添加了 SSE 指令集支持
- 初步实现了 RISC-V fp32/fp16 V 指令的支持
- PPL.cv X86/CUDA 添加大量算子并优化、支持 ARM 高频算子

- 详情请参考：https://zhuanlan.zhihu.com/p/452245355

一些常见的想法 & 问题

- 模型的“大”与“小”究竟该如何描述？只用计算量合适吗？
- 模型的计算量越小，推理速度就越快吗？
- 这个模型计算量和参数量明明变小了，为啥推理反而变慢了呢？
- 模型部署下去后算的不够快，我该从哪分析，又该怎么优化？
- 什么样的模型对硬件更加友好？
- 如何构建一个高效的深度学习推理应用？
- HPC 的同学平时说的那些东西都啥意思？我听不懂？



目录

Part 01

模型“大小”的四种描述指标

- 计算量
- 参数量
- 访存量
- 内存占用

计算量

- 模型运行一次所需的计算次数
- 反映了模型对**硬件计算单元能力**的需求
- 单位：OPs（Operations）或 FLOPs（Floating Point Operations）
- 计算方法：模型中所有算子的计算量之和
  - 单个算子的计算量：每种算子各不一致

$$FLOPs_{Conv} = N \times OC \times OH \times OW \times IC/G \times KH \times KW \times 2$$

- **计算量不单独影响模型推理速度，计算量低 ≠ 推理速度快**

参数量

- 模型中所有参数的总和
  - 在 CNN 模型中，主要是 Conv/FC/DeConv 的 weight，其他层占比很小
- 是模型文件大小的决定性因素
  - 参数量大的模型会
    - 占用更多的磁盘空间
    - 使得 APK 包体积膨胀
  - 缓解路径
    - 设计网络时减少参数量
    - 模型压缩存储
- 不直接影响推理速度，但会影响内存占用和模型初始化时间

访存量

- 模型运行一次需要访问存储单元的字节数
- 反映了模型对**存储单元带宽**的需求
- 单位：Bytes（或者 KB/MB/GB）
- 计算方法：
  - 模型访存量 ≈ 模型中所有算子的访存量之和
  - 单个算子的访存量：输入 Tensor + 输出 Tensor 的字节数

$$MACs_{Conv} = MACs_{Input} + MACs_{Output} + MACs_{Weight}$$
$$= (N \times IC \times IH \times IW + N \times OC \times OH \times OW + OC \times IC/G \times KH \times KW) \times sizeof(data\_type)$$

- **对推理速度至关重要，且易被忽略**

内存占用

- 模型运行时所占用的内存/显存大小
- 平均占用 & 最大占用
- 内存占用受模型本身以及软件实现影响
- 对推理速度没有直接影响，但在很多场景下有要求

小结

模型“大小”指标

每种指标反映了模型不同方面的需求，需要根据实际情况综合考虑

01 **计算量** **对推理速度有重要影响**

对硬件计算能力的需求

02 **参数量**

对磁盘大小/软件包大小的需求

03 **访存量** **对推理速度有重要影响**

对硬件存储器带宽的需求

04 **内存占用**

对系统内存的需求

Part 02

计算量、访存量与推理速度的关系

- 计算密度与 RoofLine 模型
- 计算密集型算子与访存密集型算子
- 计算密度与推理时间

计算密度

- 计算密度：程序计算量与访存量的比值，又称计算访存比
- 单位：FLOPs/Byte

$$I = \frac{FLOPs}{MACs}$$

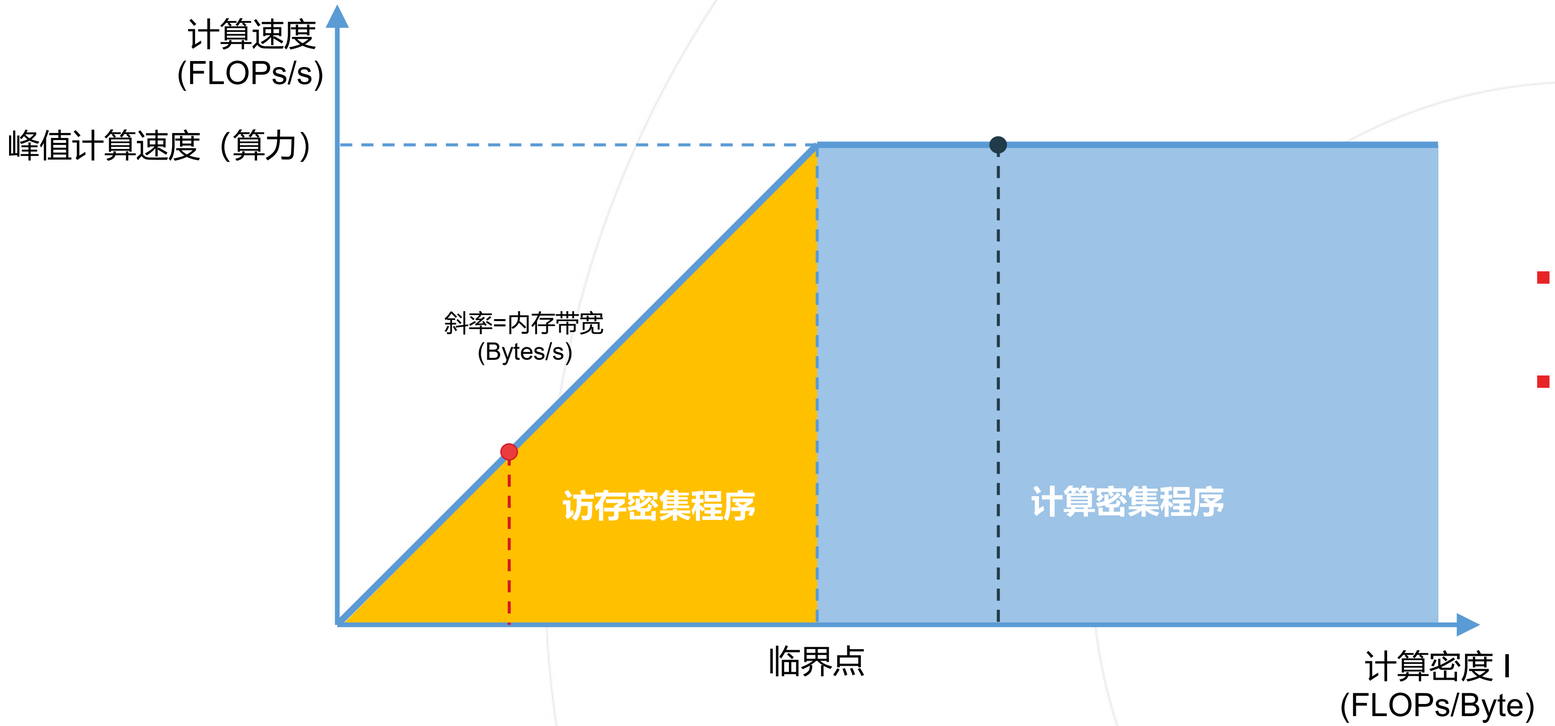
  - EltwiseAdd fp32：

$$I = \frac{FLOPs}{MACs} = \frac{N \times C \times H \times W}{N \times C \times H \times W \times 3 \times sizeof(float)} = \frac{1}{12}\ FLOPs/Byte$$

- 用于反映一个程序是计算更为密集，还是访存更为密集
  - 计算密集型程序：compute bound
  - 访存密集型程序：memory bound

RoofLine 模型

- 一种用于评估程序在硬件上能达到的**性能上界**的数学模型



- 访存密集程序：计算速度=计算密度×带宽
- 计算密集程序：计算速度=峰值计算速度

$$计算速度(FLOPs/s) = min(计算密度 \times 带宽, 峰值计算速度)$$

计算密集型算子与访存密集型算子

- 网络中的算子可根据计算密度，分为**计算密集型算子**和**访存密集型算子**
  - **典型**的算子分类

访存密集型算子

Concat、Eltwise Add、ReLU、MaxPooling

……

计算密集型算子

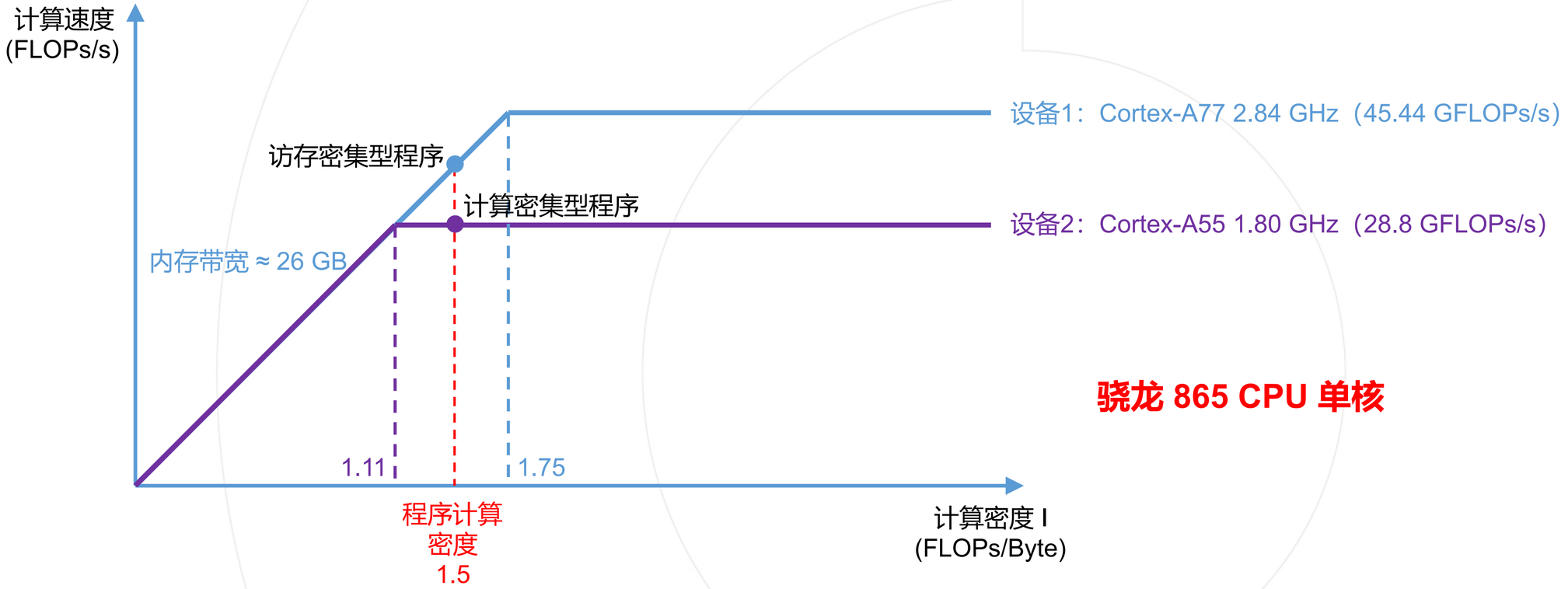
Conv、DeConv、FC、MatMul、LSTM

……

- 算子分类不绝对
  - 同一个算子在不同硬件上的性质可能会不同
  - 同一个算子在不同参数下，计算密度不同，甚至会导致算子性质发生变化

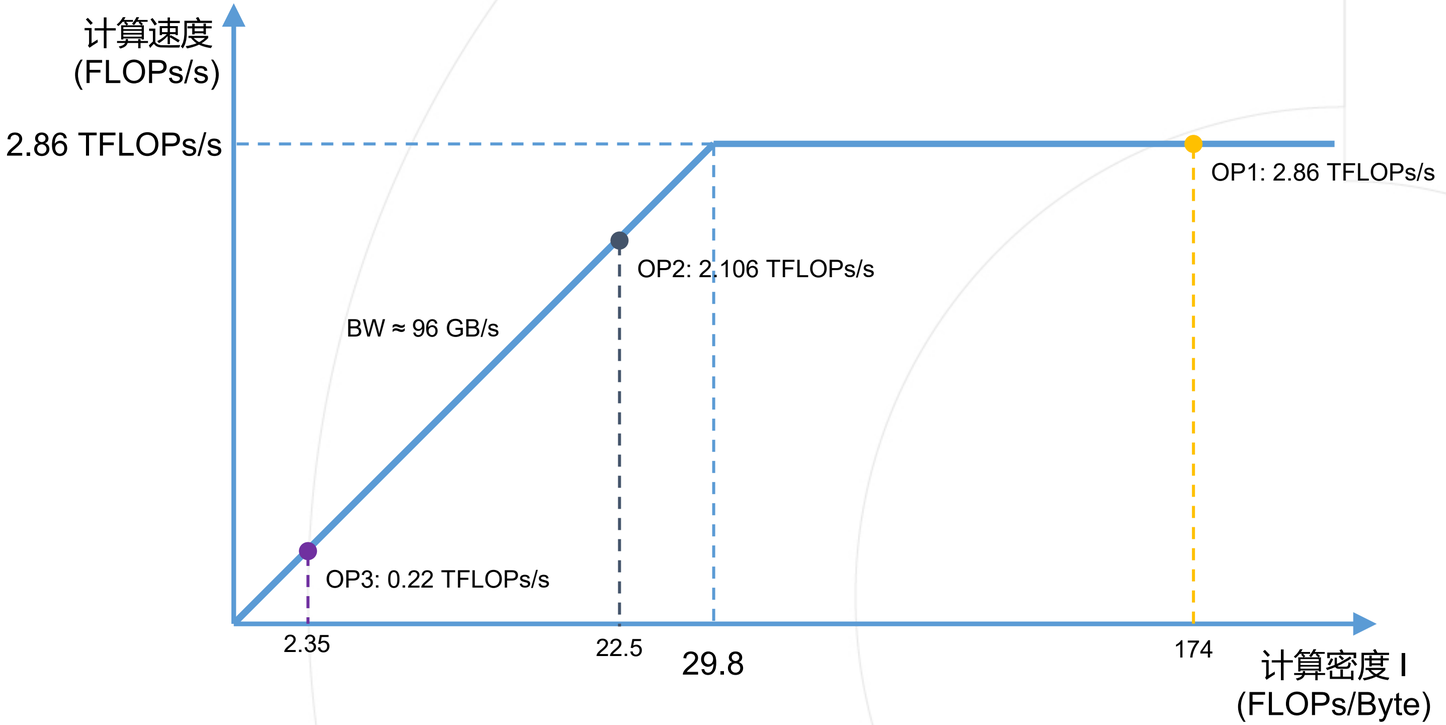
计算密集型算子与访存密集型算子

- 在不同的硬件设备上，同一个程序的性质可能会不同



计算密集型算子与访存密集型算子

- 同一种算子在不同参数下，计算密度不同，甚至会导致算子性质发生变化



计算密度越大，越有可能提升硬件的计算效率，充分发挥硬件性能

序号	batch	ic	ih,iw	kh,kw	stride	pad	dilation	group	oc	oh,ow	计算量(MFLOPs)	访存量(MB)	计算密度	计算效率	理论计算速度	
1	1	128	28	3	1	1	1	1	128	28	231.211	1.328	174.088	100.00%	2.860	计算密集
2	1	512	7	1	1	0	1	1	1024	7	51.380	2.287	22.465	73.64%	2.106	访存密集
3	1	128	28	3	1	1	1	128	128	28	1.806	0.770	2.346	7.69%	0.220	访存密集

计算密度与推理时间

- 计算效率高 ≠ 运行时间短

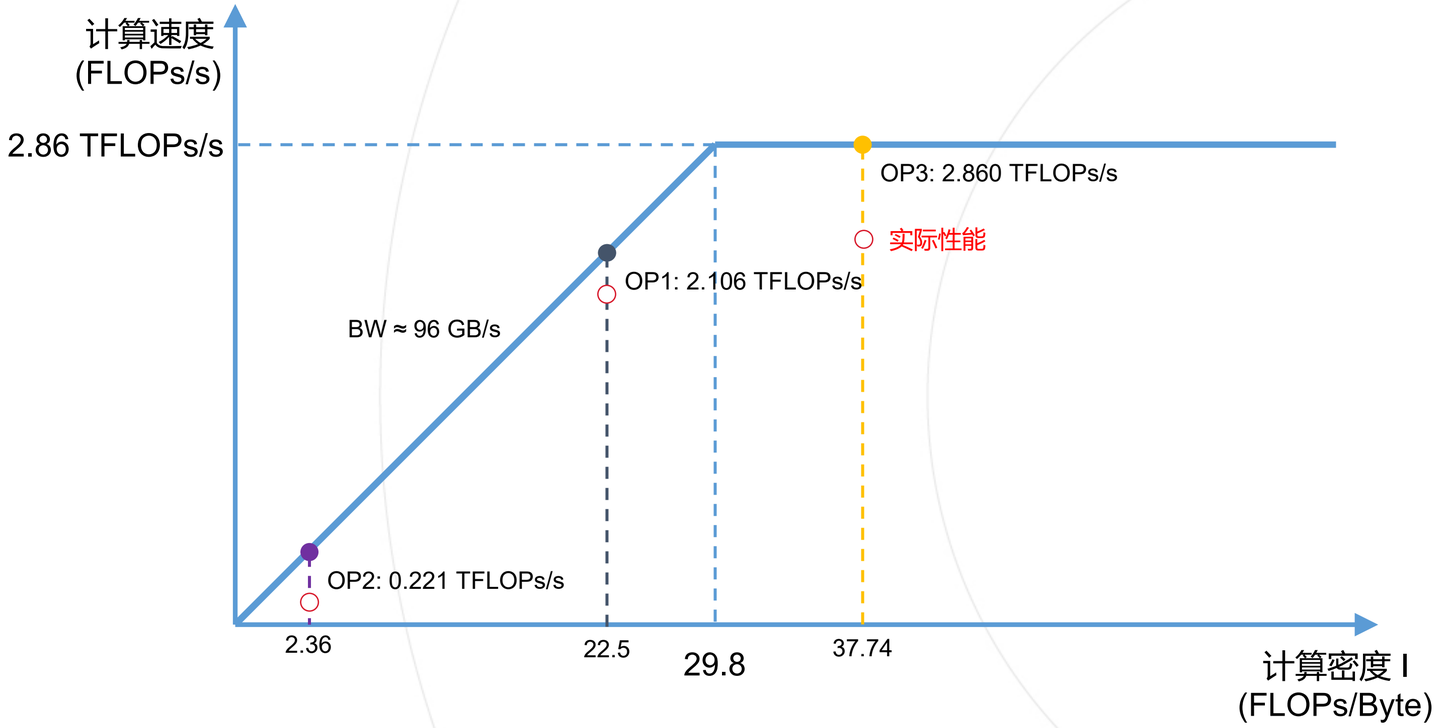
$$\text{运行时间} = \frac{\text{计算量}}{\text{计算速度}} = \begin{cases} \dfrac{\text{访存量}}{\text{内存带宽}}, & \text{访存密集型算子} \\ \dfrac{\text{计算量}}{\text{峰值计算速度}}, & \text{计算密集型算子} \end{cases}$$

- 根据 RoofLine 模型，计算量、访存量与推理时间的关系：
  - 访存密集型算子：推理时间跟访存量呈线性关系
  - 计算密集型算子：推理时间跟计算量呈线性关系

计算密度与推理时间

- 实例

序号	batch	ic	ih,iw	kh,kw	stride	pad	dilation	group	oc	oh,ow	计算量(MFLOPs)	访存量(MB)	计算密度	理论计算速度	理论推理时间(us)
1	1	512	7	1	1	0	1	1	1024	7	51.380	2.287	22.465	2.106	24.40
2	1	16	448	3	1	1	1	16	16	448	57.803	24.501	2.359	0.221	261.34
3	1	16	448	3	1	1	1	1	16	448	924.844	24.509	37.735	2.860	323.37



- RoofLine 模型给出的是**性能上界**
- 实际性能会因其他因素而离上界有一定距离

Part 03

影响推理速度的其他因素

- 硬件因素
- 系统环境因素
- 软件实现因素

硬件因素

- 一些硬件因素会导致程序无法达到理论计算峰值：
  - 功耗导致的频率变化

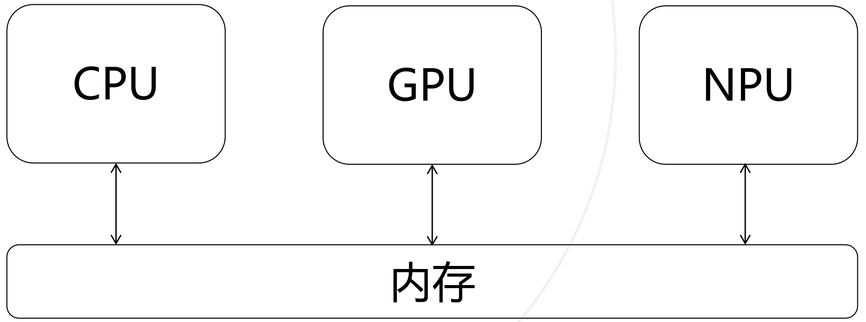
10980XE 不同核心数、不同指令集下的**频率(MHz)**

核心数 \ 指令集	AVX512	AVX	Normal
1~2	3700	3900	4500
3~4	3500	3700	4200
5~8	3400	3600	4100
9~12	3200	3600	4100
13~16	2900	3400	3900
17~18	2800	3300	3800

  - 硬件在设计上计算与访存无法充分掩盖
    - ARM 小核（Cortex-A53/A55）的静态执行流水线
    - 低端 Risc-V 芯片的阻塞式 Cache
  - 一些硬件实现上的 bug
- 硬件算力峰值需要实测
  - 高叔叔《浮点峰值那些事儿》：https://zhuanlan.zhihu.com/p/28226956

系统环境因素

- 操作系统中的其他任务，乃至操作系统本身也会对推理速度产生影响
  - Android 大小核调度问题
    - CPU占用率不足时，程序会被操作系统调度到小核
    - 解决方法：绑核
  - 缺页问题
    - Linux 写时拷贝机制引发的缺页问题：第一次写入时，速度异常的慢
    - 解决方法：大块内存不频繁申请&释放 or 使用内存池
  - 其他任务对硬件资源的挤占：推理时间 1+1>2
    - 对计算资源的挤占
    - 对内存带宽等其他资源的挤占



- 需要关注测试环境和实际部署系统环境的差异，最好能在实际系统环境下测试

软件实现因素

- 一个任务的软件实现好坏，对性能有重大影响

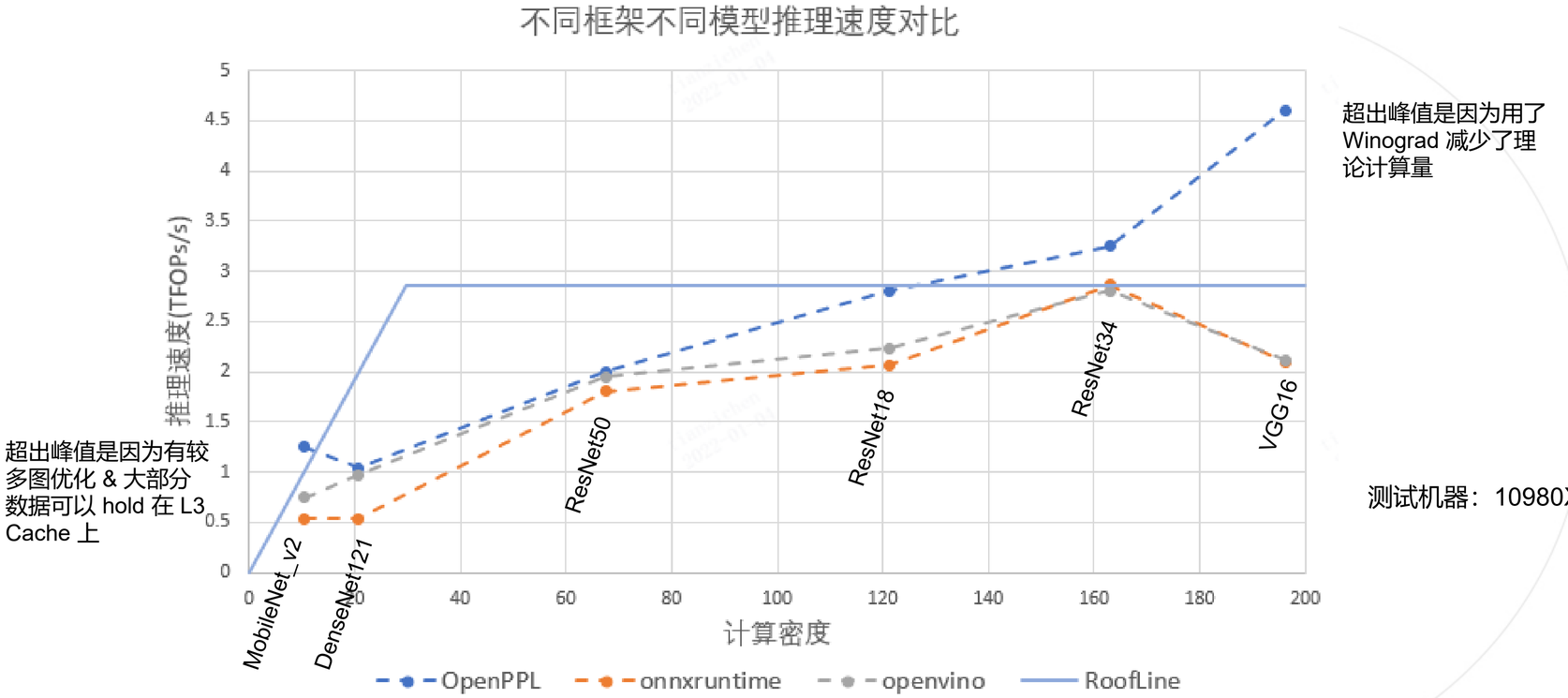
```
# naive implement: 17 s
for m in range(M):
    for n in range(N):
        for k in range(K):
            C[m, n] += A[m, k] * B[k, n]

# numpy implement: 0.138 s
C = np.dot(A, B)
```

- 对于深度学习推理框架来说，软件实现上的影响因素有：

软件实现因素

- 软件对性能的影响涉及到较多因素，往往呈现出较强的**非线性**，很多时候只能具体情况具体分析
- **实测是性能最准确的评估方式**，性能问题欢迎与 HPC 同学多多交流



不同框架不同模型的推理速度

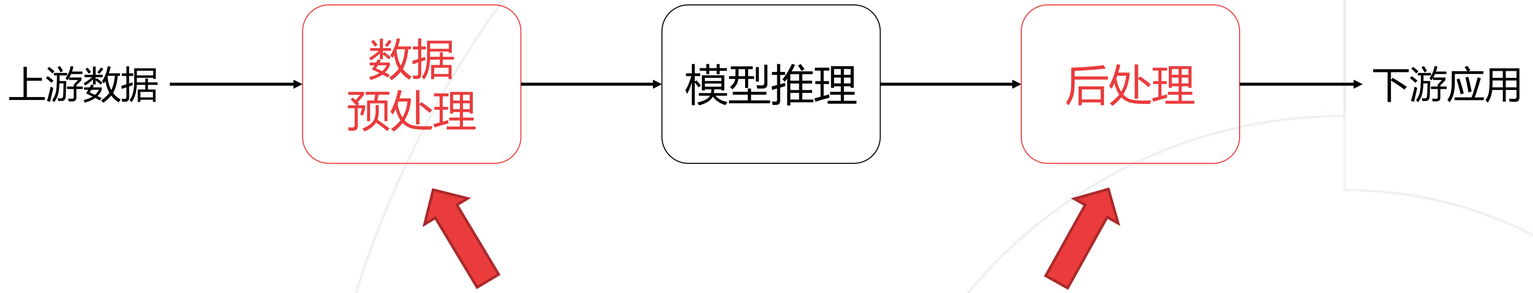
Part 04

如何实现高效的深度学习推理应用

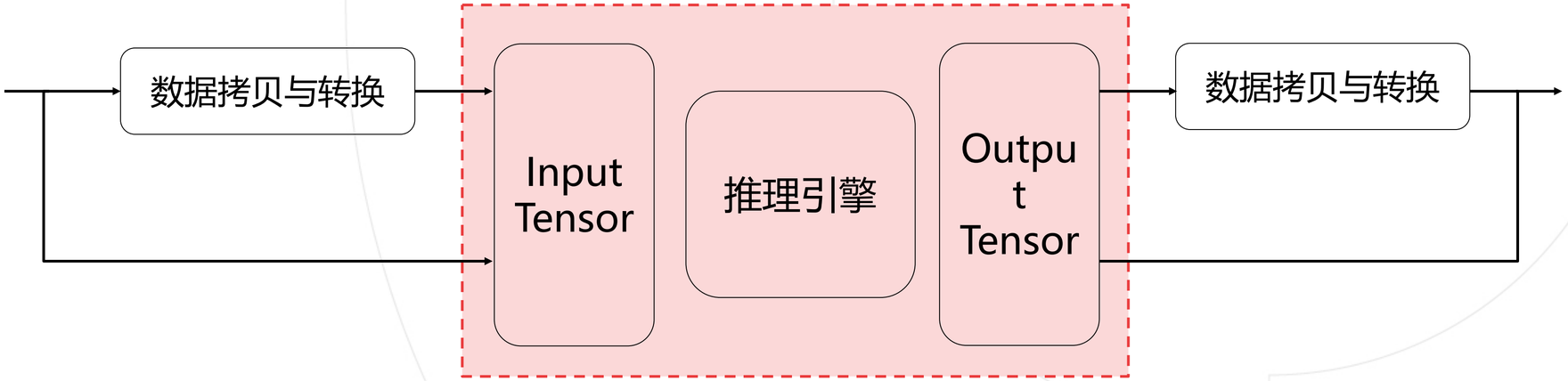
- 设计高效合理的应用软件框架
- 设计高效的深度学习网络结构
- 选用高效的推理框架
- 高性能深度学习推理应用方法论

设计高效合理的应用软件框架

- 预处理 & 后处理同样需要加速

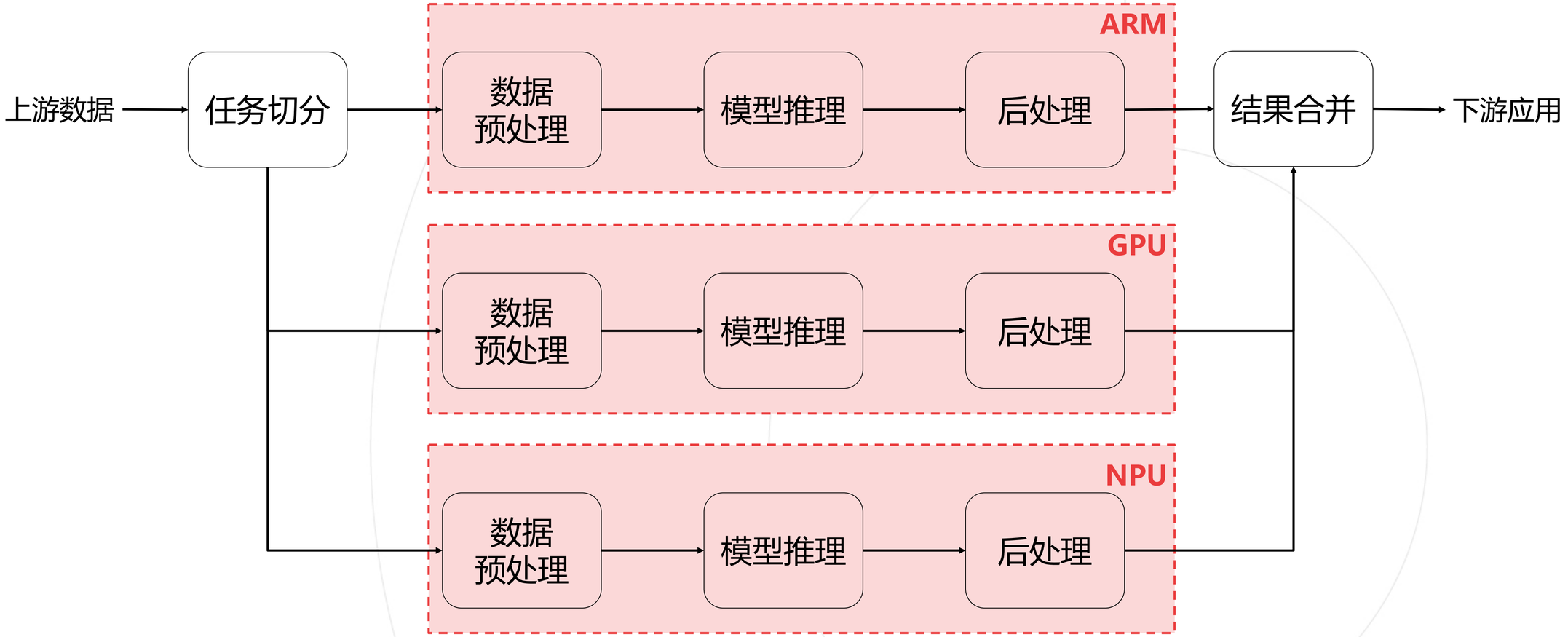


- 减少不必要的内存拷贝：零拷贝输入输出模式



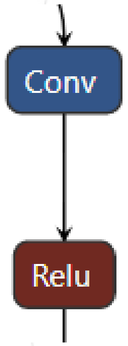
设计高效合理的应用软件框架

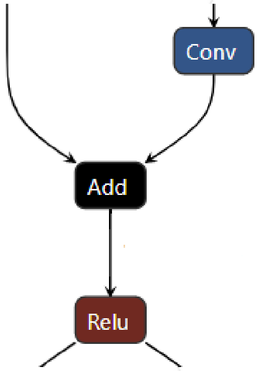
- 在具有多种设备的平台上（如高通 SoC），充分利用多后端算力

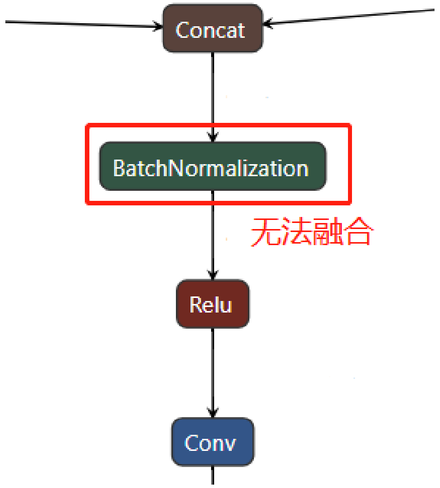


设计高效的深度学习网络结构

- 根据硬件平台的特性，调整网络的计算访存比（多平台部署可考虑设计多种模型）
  - 对于算力较弱的平台，受限于硬件计算能力，可以设计低计算量的网络来加快推理
  - 对于算力强的平台，一味降低计算量不一定有收益，需要关注访存量
- 网络子结构尽量选用经典结构，以最大可能匹配图优化规则







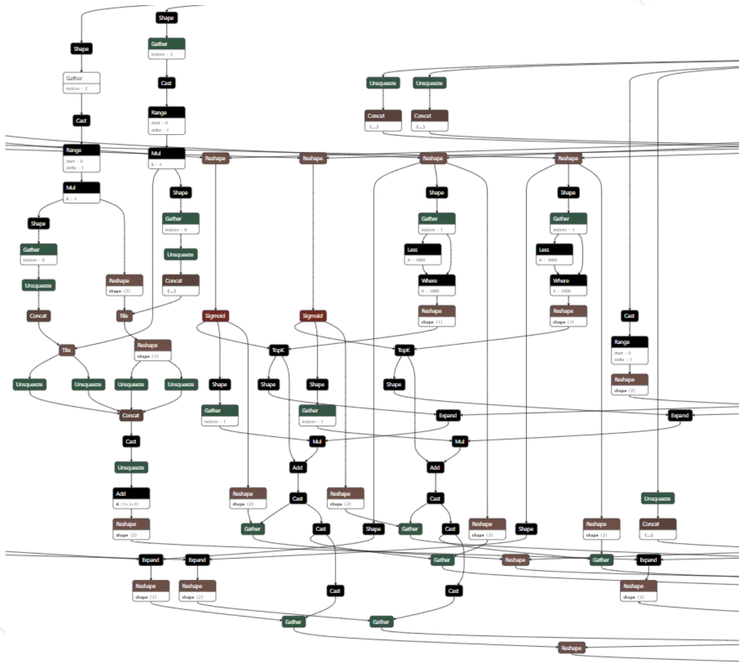
- 算子尽量使用常用参数，软件会对其做特殊优化
  - Conv：F3S1、F1S1、F3S2、F1S2、DwConv F3S1、DwConv F3S2 等等

设计高效的深度学习网络结构

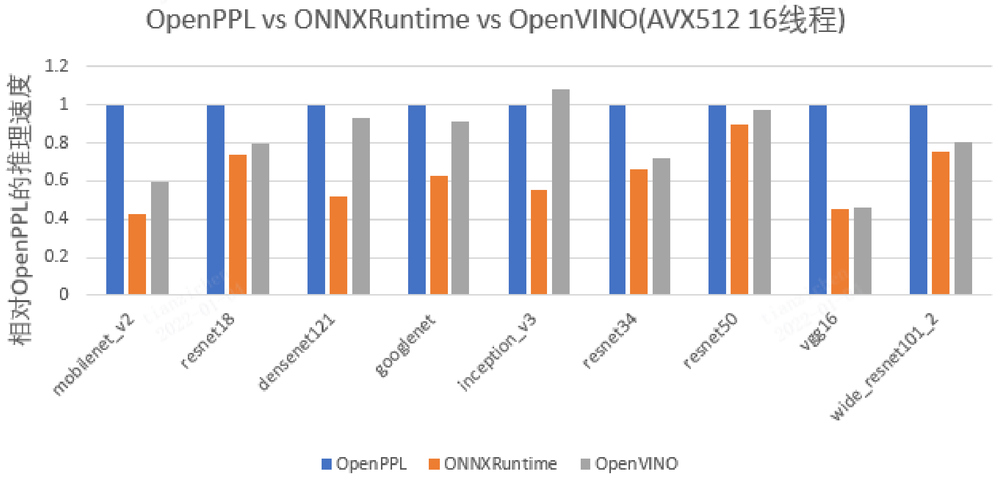
- 设计网络时，channel 数尽量选择 4/8/16 的倍数
  - 推理框架会使用特殊的数据排布，channel 数会向上取整

数据排布	Channel 取整数	选用的架构/精度
N4CHW	4	ARM fp32
N8CHW	8	ARM fp16、X86 FMA/SSE
N16CHW	16	X86 AVX512

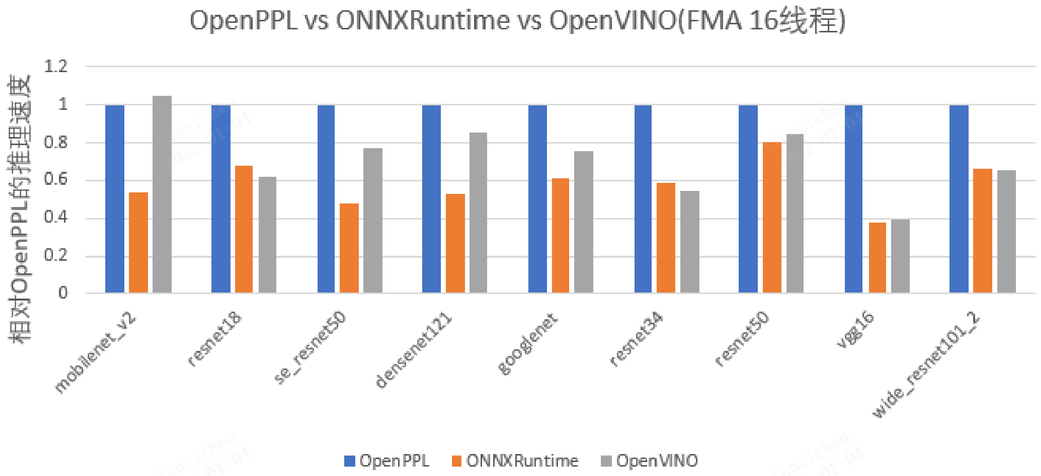
- “碎而深”的网络会有较多算子调用开销：
  - 函数调用、网络拓扑、内存池、线程池开销
  - 严重时甚至会成为网络的性能瓶颈
  - 使用 onnxsim 等工具对细碎算子进行合并
  - 定制大算子



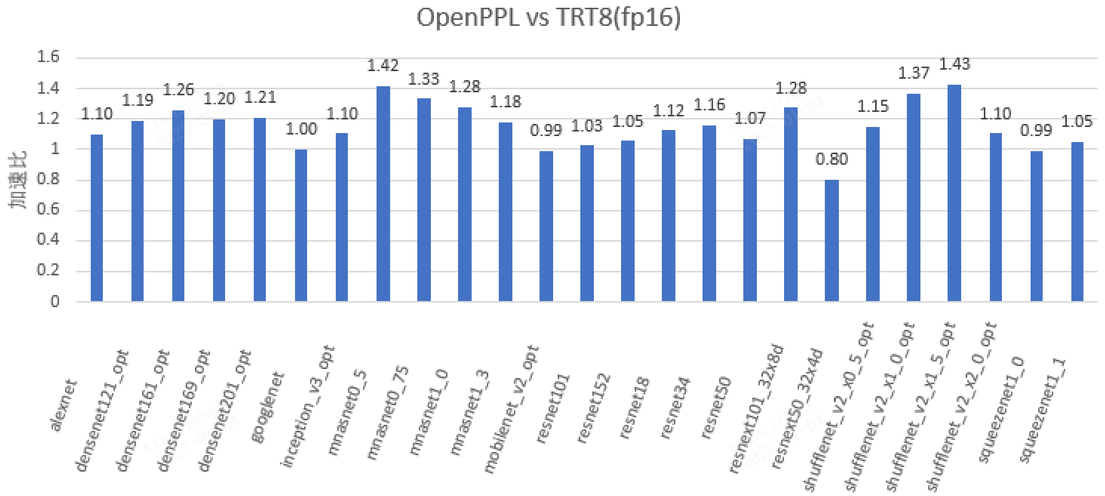
选用高效的推理框架



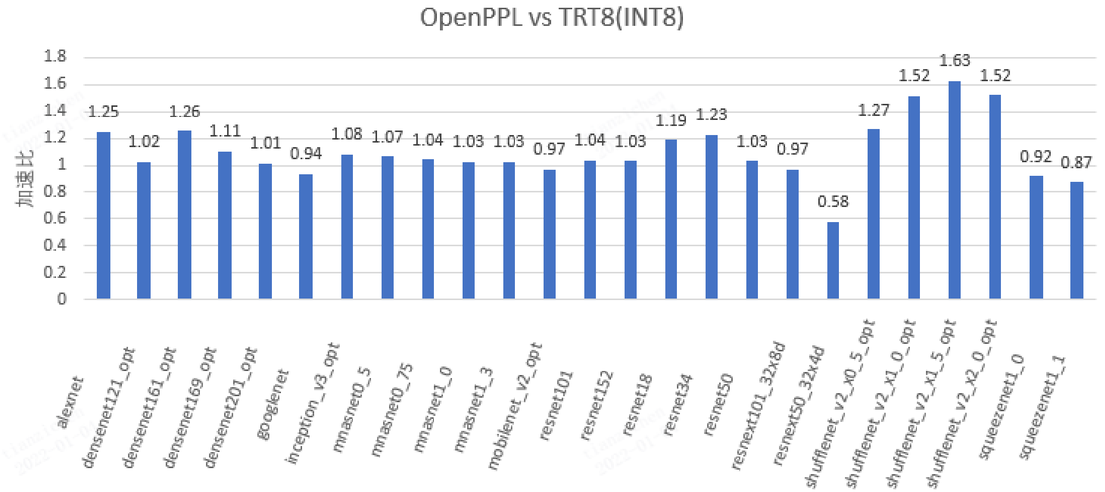
X86 FP32 AVX512



X86 FP32 FMA



CUDA FP16



CUDA INT8

设计高性能的深度学习推理应用的方法论

- 不再单独以计算量作为模型“大小”的评价指标，而是多指标综合评判模型
- **实测**是最准确的性能评估方式
- 在设计模型初期就将推理性能作为设计指标，测试模型性能，迭代模型结构
- 注意测试环境与实际部署环境的差异，尽可能在实际部署的环境下测试性能
- 深度学习推理应用需要从硬件选择、软件结构、模型设计、推理框架**全流程**优化
- 遇到模型推理性能问题时，可以逐层 profiling 确定性能瓶颈，与 HPC 同学保持紧密沟通，具体问题具体分析

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