推理引擎 OpenPPL 实战训练营

商汤 sensetime

OpenPPL之通用架构下的性能优化概要

2021年12月29日

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（x86篇）	梁杰鑫	敬请期待
第六期：性能调优实战（CUDA篇）	李天健	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

许志耿

商汤科技高级异构计算工程师
PPL CPU & 加速器架构负责人

- 本硕毕业于上海交通大学计算机系，研究方向为高性能计算，在 PARCO、ICPP、IPDPS 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 CPU、加速器等架构方向的 PPL 研发与优化

课程提纲

第三期课程将交流分享在 CPU 等通用架构下，基于微基准测试、性能分析以及底层调优的性能优化整体思路。

1. **工欲善其事，必先 Benchmark**：硬件架构特性分析
2. **AI 中的 AI**：基于 Arithmetic Intensity，探索优化方向
3. **斗榫合缝**：基于微架构的指令级调优
4. **性能优化流程总结**

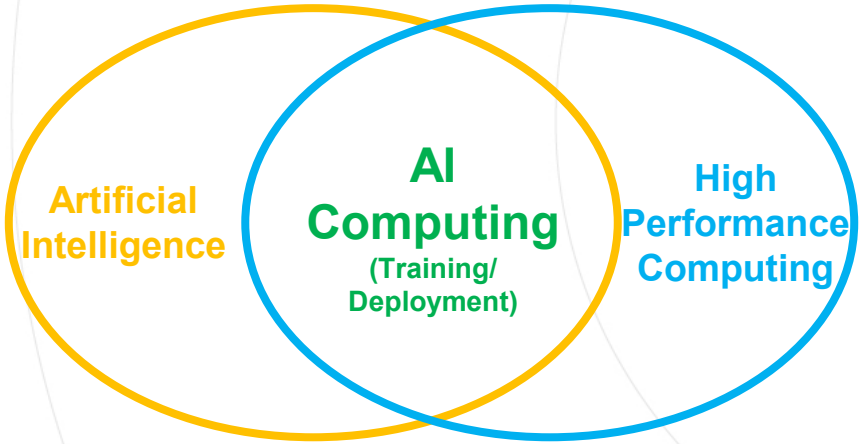
OpenPPL性能优化分享——
通用架构下的性能优化概要

许志耿

2021.12.29

AI Computing

- AI Computing：HPC + AI
- 推理部署：基于目标硬件平台，实现AI算法的高效推理过程
- 推理系统在AI应用部署中的定位
  - **“承上”，“比算法更懂硬件”—— 架构特性**
  - “启下“，“比硬件更懂算法”—— 计算特性

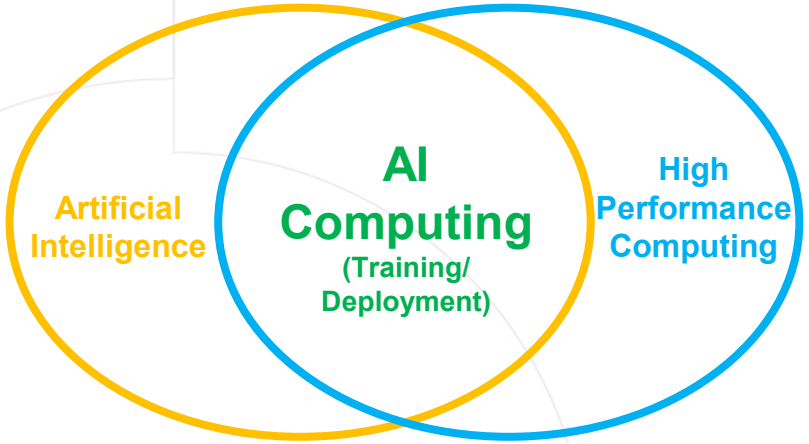


商汤 sensetime

Outline

Part 1. 基于Micro-benchmark的硬件架构特性分析

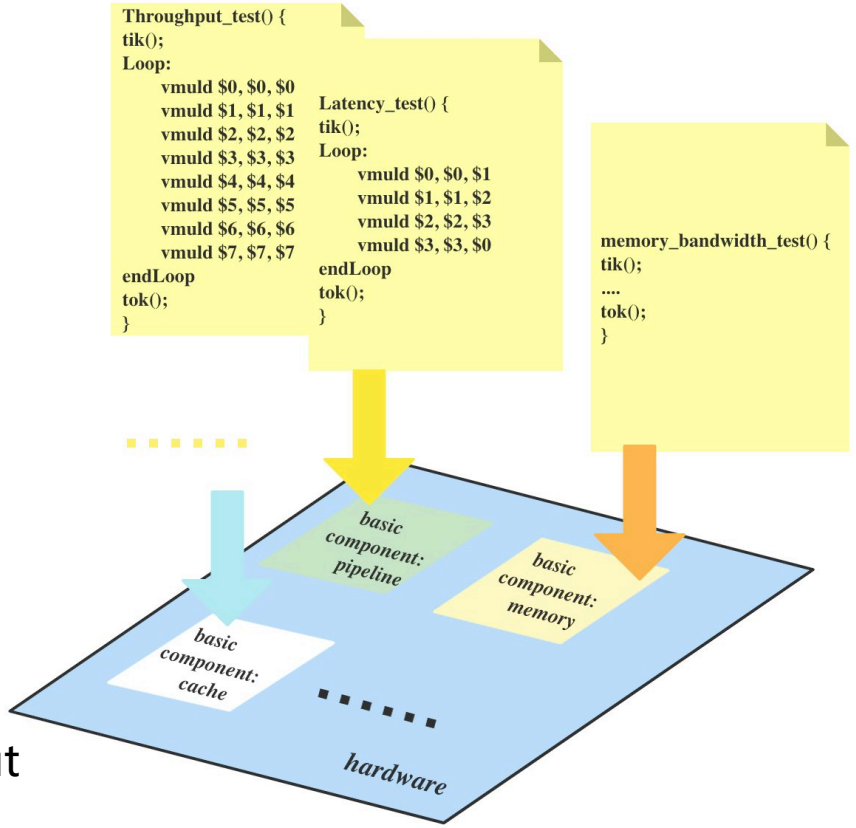
- AI Computing：HPC + AI
- 推理部署：基于目标硬件平台，实现AI算法的高效推理过程
- 推理系统在AI应用部署中的定位
  - ❑ **“承上”，“比算法更懂硬件”—— 架构特性**
  - ❑ “启下“，“比硬件更懂算法”—— 计算特性
- 如何获取架构特性?
  - ❑ 厂商架构文档 —— 是否足够开放？完整？准确?
  - ❑ 工欲善其事，必先利其器 —— 基于micro-benchmark进行架构特性测试
- Micro-benchmark：“A micro-benchmark is either a program or routine to measure and test the performance of a single component or task.” [1]
  - ❑ 针对单一架构组件的性能测试程序



Ref. [1] Poggi N. (2019) Microbenchmark. In: Sakr S., Zomaya A.Y. (eds) Encyclopedia of Big Data Technologies. Springer, Cham. https://doi.org/10.1007/978-3-319-77525-8_111

Part 1. 基于Micro-benchmark的硬件架构特性分析

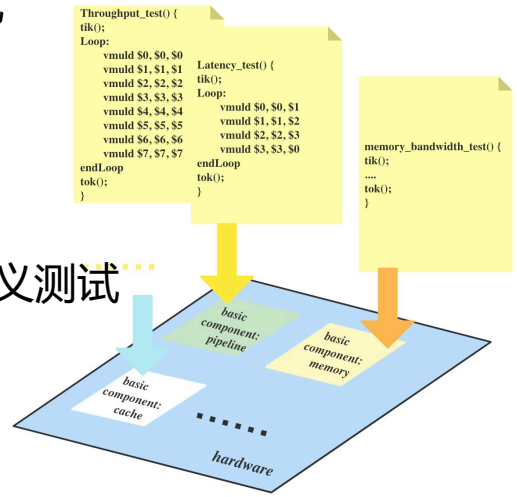
- Micro-benchmark
  - 测试对象：尽可能从功能独立的单一架构组件入手
  - 测试方法：设计基于简单代码/指令片段的小型程序
  - 测试目的：充分体现架构组件的性能



- 与性能相关的架构组件举例
  - 计算 —— 流水线微架构（pipeline）：instruction latency & throughput
  - 访存 —— 存储层次（memory hierarchy）：global/local memory, cache latency & bandwidth
  - 通信 —— 互联（comm）：inner/intra node/cluster/core communication latency & bandwidth
  - 特色架构组件

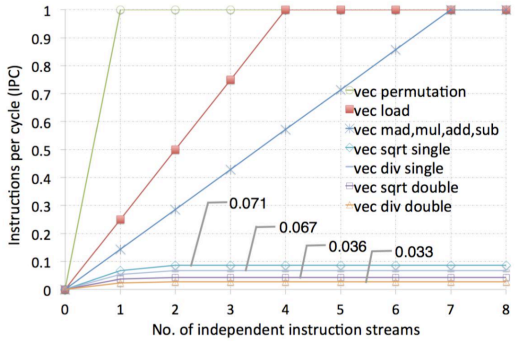
Part 1. 基于Micro-benchmark的硬件架构特性分析

- Micro-benchmark设计
  - 基于独立指令流的**指令吞吐**（IPC）测试
  - 基于依赖序列的**指令延迟**测试
  - 基于pointer-chasing的**各级访存延迟**测试
  - 基于stream的**各级访存带宽**测试
  - 基于ping-pong的**通信延迟**测试
  - 基于sync的单向延迟测试
  - 基于架构问题最小复现集的自定义测试
  - …



向量指令吞吐

- 以instruction per cycle (IPC) 作为度量



- At least 7 FMA to achieve peak performance.
- The throughput of SPM access and arithmetic vector instructions are all 1, except div and sqrt.
- The IPC of div and sqrt are quite low with a reciprocal throughput of 30 (1/0.033) and 28 (1/0.036) respectively in double precision, implying the succeeding independent double-precision div (sqrt) has to stall for 30 (28) cycles before it can be issued.

点对点 RLC 延迟测试

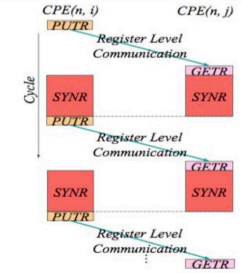


Figure 7: Measuring P2P RLC

Listing 6: Kernel pseudo code of P2P RLC latency measurement

```
1  //CPE(n, i):
2  LOOP:
3  //send data to CPE j
4  putr $data, j
5  //sync in row with CPE j
6  synr $mask
7
8  //CPE(n, j):
9  LOOP:
10 //receive data from CPE i
11 getr $data
12 //sync in row with CPE i
13 synr $mask
```

- Result:
  - T_putr(1 cycle) + T_getr(1 cycle) + T_data_trans = 10 cycles → T_data_trans = 10-1-1= 8 cycles
  - We test every pair of CPEs in the same row/col.

RLC latency in row:

No. IN ROW	0	1	2	3	4	5	6	7
0	/	8	8	8	9	9	9	9
1	8	/	8	8	9	9	9	9
2	8	8	/	8	9	9	9	9
3	8	8	8	/	9	9	9	9
4	8	8	8	8	/	9	9	9
5	8	8	8	8	9	/	9	9
6	8	8	8	8	9	9	/	9
7	8	8	8	8	9	9	9	/

RLC latency in col:

No. IN COL	0	1	2	3	4	5	6	7
0	/	8	9	9	8	8	9	9
1	8	/	9	9	8	8	9	9
2	8	8	/	9	8	8	9	9
3	8	8	9	/	8	8	9	9
4	8	8	9	9	/	8	9	9
5	8	8	9	9	8	/	9	9
6	8	8	9	9	8	8	/	9
7	8	8	9	9	8	8	9	/

Table[i,j] means the latency of transferring data from i_{th} CPE to j_{th} CPE in the same ROW/COL.

Micro-benchmark 测试样例[2]

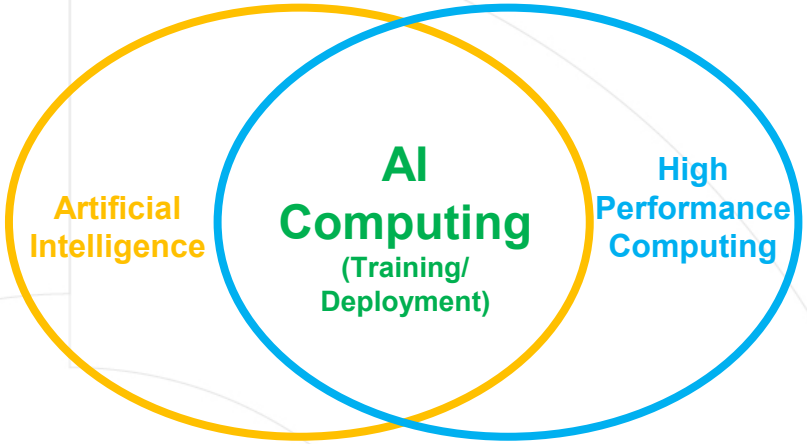
Ref. [2] Xu Z, Lin J, Matsuoka S. Benchmarking sw26010 many-core processor[C]//2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2017: 743-752.

商汤 sensetime

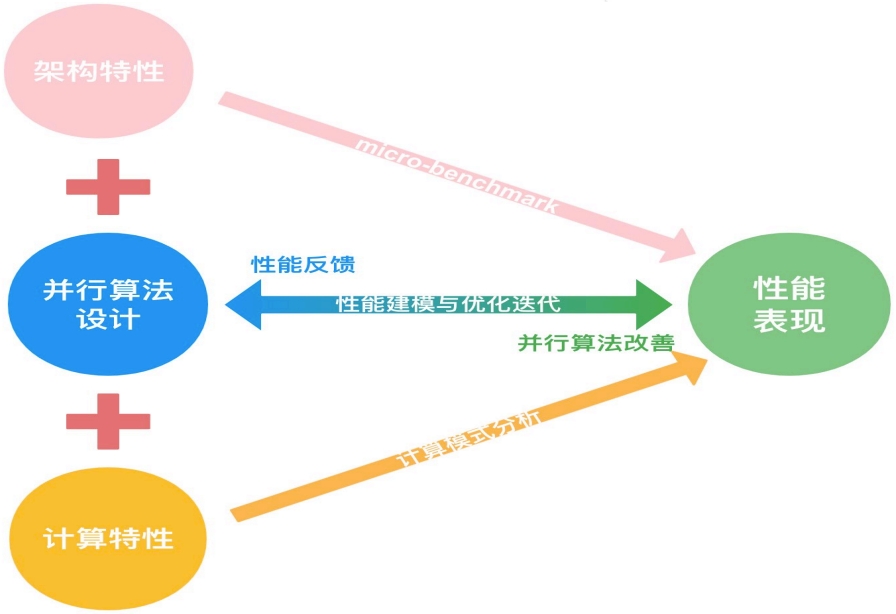
Outline

Part 2. 基于Arithmetic Intensity的性能分析

- 推理部署在AI产业中的定位
  - “承上”，“比算法更懂硬件”—— 架构特性
  - **“启下”，“比硬件更懂算法”—— 计算特性**



- 计算特性：计算过程的计算模式、访存模式、扩展引入的通信等
- 计算特性/架构特性/性能表现与性能优化的关系
  - 计算性能优化，本质上是基于给定的计算模式，在目标硬件平台上，依据架构特性，为获取高性能实现，寻找最优的计算、访存、通信行为的**执行序列（并行算法）**的过程。
- 并行算法设计常常是一个迭代的过程
  - 前馈：**根据计算特性、架构特性，进行性能建模，指导优化方向，设计并行算法。**
  - 反馈：**根据性能表现反馈，寻找性能瓶颈与潜在的架构特性的影响，进行架构特性分析与并行算法改良，再次进入前向过程。**



Part 2. 基于Arithmetic Intensity的性能分析

- 性能分析中**量化指标**的必要性
  - 关联计算与访存、计算特性与架构特性
  - 辅助分析与建模
- 计算强度 Arithmetic Intensity
  - Arithmetic intensity is a measure of floating-point operations (FLOPs) performed by a given code (or code section) relative to the amount of memory accesses (Bytes) that are required to support those operations.[3]
  - Flops / Bytes：每搬运单位数量的数据，硬件可以进行的计算次数
  - 一定程度上刻画了计算流程与硬件平台在计算与访存方面的特征
  - **计算模式**作为计算&访存的**需求方**，**硬件平台**作为计算&访存的**资源供给方**，两者的**供需匹配程度决定了性能的表现**
- 如何关联计算模式与硬件平台的计算强度，进而分析供需程度?
  - Roofline Model

Ref. [3] https://docs.nersc.gov/performance/arithmetic_intensity/

Part 2. 基于Arithmetic Intensity的性能分析

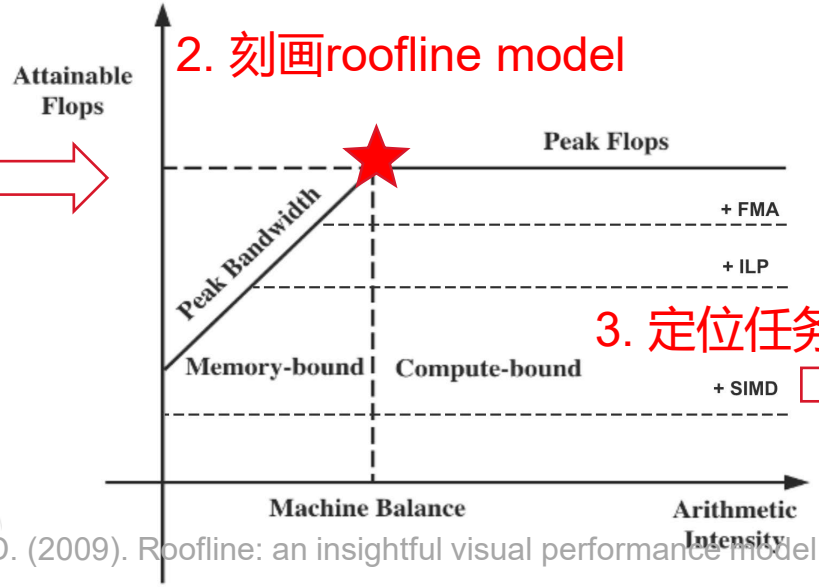
- 构建Roofline Model [4]
  - 横轴：计算强度
  - 纵轴：可达到的性能上限（Attainable FLOPS），取与硬件理论峰值相比的最小值
  - 斜率：目标存储层次带宽
  - 将硬件的计算/访存能力与计算任务的需求进行可视化与分析

- 在Roofline Model中分析计算任务瓶颈
  - 选择正确的访存层次（global memory/LLC/local memory等）刻画Roofline Model
  - 正确计算任务的计算强度（访存层次、访存量、计算量等）
  - 将任务计算强度与Machine Balance对比，判断性能瓶颈，Memory bound/Compute bound
  - 判断性能上限，与当前实际性能进行对比

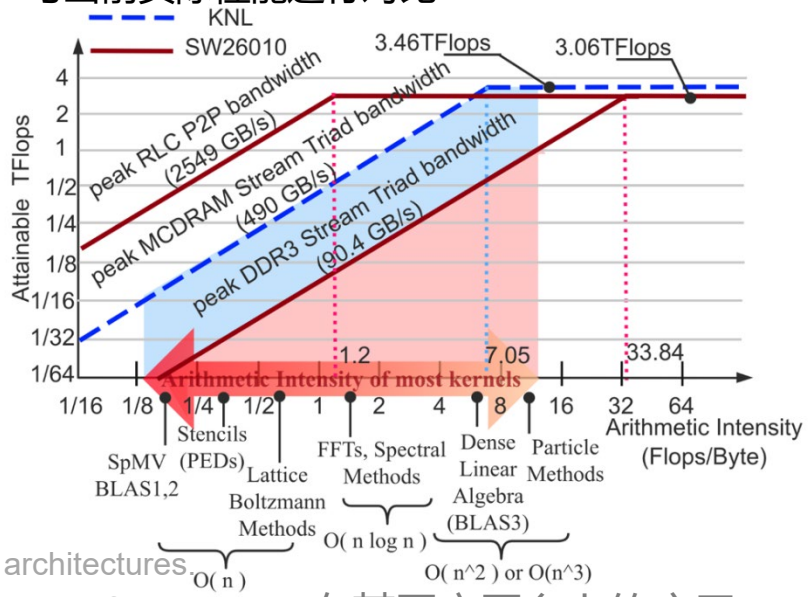
1. 评估计算强度

$$\text{Attainable Flops} = \text{Min}\begin{cases}\text{Peak Flops}\\ \text{AI x Peak Bandwidth (GB/S)}\end{cases}$$

$$\text{AI} = \text{FLOPS} / \text{Bytes}$$



3. 定位任务瓶颈



Roofline Model在某国产平台上的应用 [5]

Ref. [4] Williams, S., Waterman, A., & Patterson, D. (2009). Roofline: an insightful visual performance model for multicore architectures. Communications of the ACM, 52(4), 65-76.
[5] Lin, J., Xu, Z., Cai, L., Nukada, A., & Matsuoka, S. (2018). Evaluating the SW26010 many-core processor with a micro-benchmark suite for performance optimizations. Parallel Computing, 77, 128-143.

Part 2. 基于Arithmetic Intensity的性能分析

- 简单示例

❑ 基于内积的GEMM外围naïve算法分析（为简化说明，考虑C = A * B）

❑ gemm size为M,N,K，假设在LLC/local memory只做了一级分块 bm, bn, bk

❑ 假设数据较大，需要在主存与cache中迁移，选取主存作为存储层次分析对象

❑ 则单次K向迭代中

```
For bm in M:
    for bn in N:
     for bk in K:
            load block_A, block_B;
            compute block_C += block_A * block_b;
     store block_C;
```

  ❑ 访存量为 block_A(bm * bk) + block_B(bn * bk) + block_C(bm * bn) / (K / bk)

  ❑ 计算量为2 * bm * bn * bk

  ❑ 考虑到内积计算方式对矩阵A的复用，以及B的重复加载，还需要根据分块个数(M/bm, N/bn, K/bk)，重新调整访存量

❑ 则计算访存比为与M、N、K、bm、bn、bk相关的表达式 F(M, K, K, bm, bn, bk)

❑ Cache上矩阵分块占用容量为B(bm, bn, bk)，需考虑双缓冲

❑ 考虑到gemm kernel的寄存器分块，bm, bn, bk最好是寄存器分块rm, rn, rk的整倍数

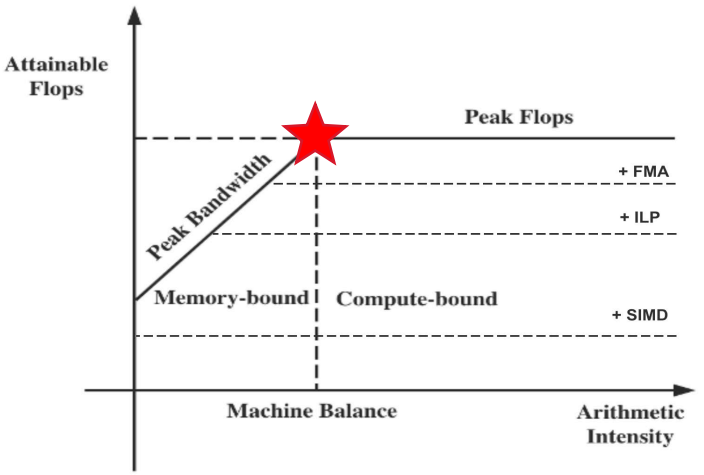
❑ 如果希望gemm达到峰值，则需求解满足如下约束的不等式，找到合适的分块大小

  ① F(M, K, K, bm, bn, bk) >= Machine Balance

  ② B(bm, bn, bk) <= Cache Size (e.g., LLC);

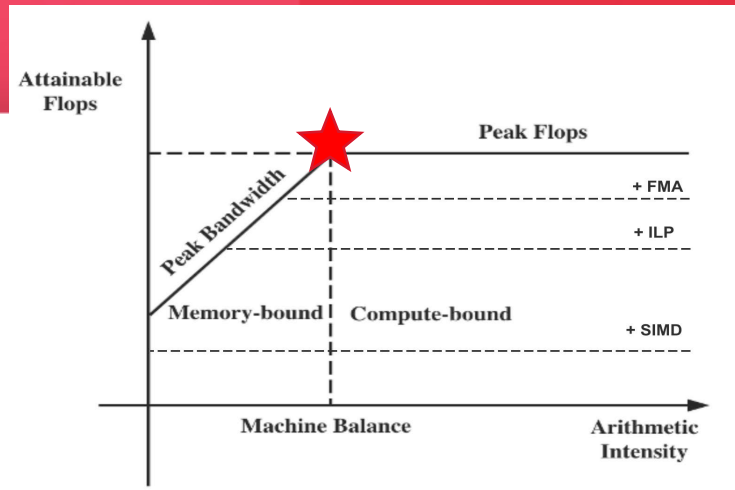
  ③ bm % rm == 0; bn % rn == 0; bk % rk ==0;

  ④ 为了使汇编kernel效率足够高，bk需要尽可能大



❑ 根据最终分块大小算出计算强度F，进而在Roofline Model中定位上限和瓶颈，以及根据实际性能反馈，改善算法流程

Part 2. 基于Arithmetic Intensity的性能分析



- Roofline Model 前置条件 & 局限性
  - 根据是否使用FMA/SIMD/ILP选取性能上界
  - 假定计算与访存可以overlap
  - 需选取正确的存储层次带宽
  - 计算指令可以fully-pipelined (throughput-oriented model)，否则需降低性能上界（latency bound）
  - 错误的计算性能上界/存储层次/计算访存无法掩盖将导致实际性能表现与建模存在较大偏差
  - 适合用于评估卷积/矩阵乘的并行算法性能上限与优化方向；或评估长尾算子的瓶颈类型（memory bound/compute bound）
- 同一计算任务在不同架构上，瓶颈类型可能不同
  - 核心在于分析不同硬件架构的Machine Balance和任务的计算强度
  - 例如：在端侧arm平台，conv常常是compute-bound
  - 若云侧大算力处理器的local memory空间有限，或内存带宽有限，则conv将成为memory bound，优化思路截然不同

硬件平台/架构特性	端侧处理器	云侧处理器
算力量级（TOPS）	1~10	~100
带宽量级（GB/S）	10~100	100~1000
Machine Balance	10~100	100~1000

商汤 sensetime

Outline

Part 3. 基于微架构的底层调优

- Compute-bound类型的调优与底层微架构息息相关
- 调优目标是最大化指令吞吐
- 常见调优方法举例
  - 最大化寄存器复用率，进而增大gemm汇编kernel的寄存器分块，隐藏ld/st指令延迟，提高指令吞吐率
  - 依据指令延迟，以及指令间依赖关系，交错指令发射时机，减少气泡
  - 寄存器双缓冲/分时复用，隐藏迭代间的数据加载开销
  - 依据指令发射端口数量和发射限制，group相关指令（尤其是VLIW类架构）
  - loop unrolling并寻找上述指令调度优化机会，提升ILP
  - 以软件模拟算法替代SFU运算（在SFU的IPC较低的情况下），并辅以loop unrolling等等

Outline

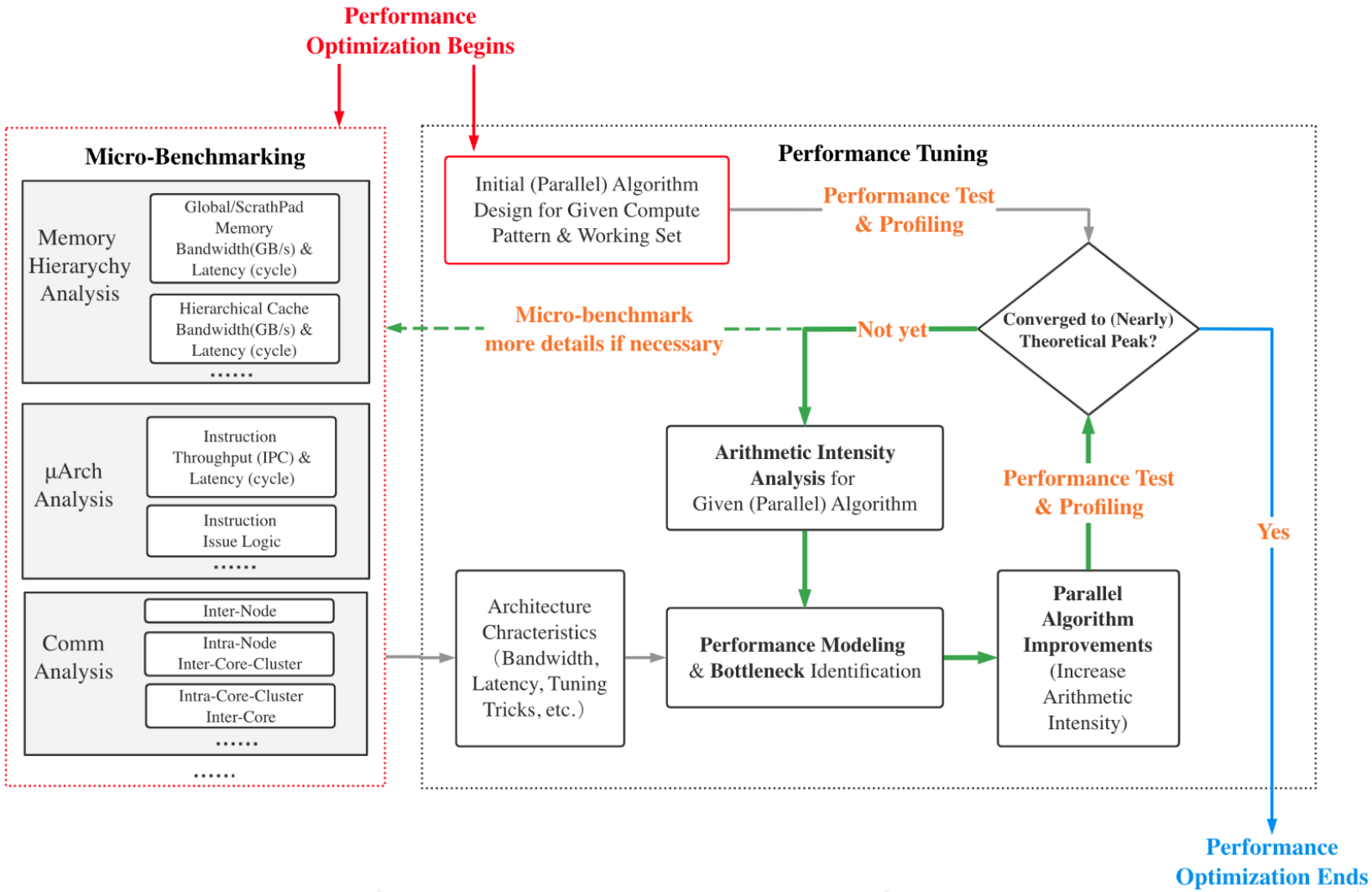
Part 4. 性能优化总结

- 优化流程

❑ Step1. 基于micro-benchmark，分析获取架构计算/访存/通信特性

❑ Step2. 结合计算任务的计算模式，设计并行算法

❑ Step3. 分析性能瓶颈，结合架构特性，针对性调优计算/访存瓶颈，或返回Step2. 改善并行算法设计，迭代直至性能收敛



Q&A

https://github.com/openppl-public
https://openppl.ai/
https://www.zhihu.com/people/openppl