推理引擎 OpenPPL 实战训练营

商汤 sensetime

OpenPPL X86 技术解析

2022年2月18日星期五

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：OpenPPL CUDA技术解析	李天健	2022年01月13日
第六期：性能调优实战（x86篇）	**梁杰鑫**	**2022年02月17日**
第七期：OpenPPL+RISC-V 指令集初探	焦明俊/杨阳	2022年02月24日
第八期：OpenPPL 在 ARM Server 上的技术实践	许志耿/邱君仪	2022年03月03日
第九期：量化工具实践	纪喆	2022年03月10日

「商汤学术」公众号

可以回复“抽奖”试试哦

主讲人介绍

梁杰鑫

商汤科技异构计算工程师
OpenPPL X86 架构负责人

- 本科毕业于中山大学
- 目前在商汤科技高性能计算部门负责参与 CPU 方向的 PPL 研发与优化

OpenPPL X86 技术解析

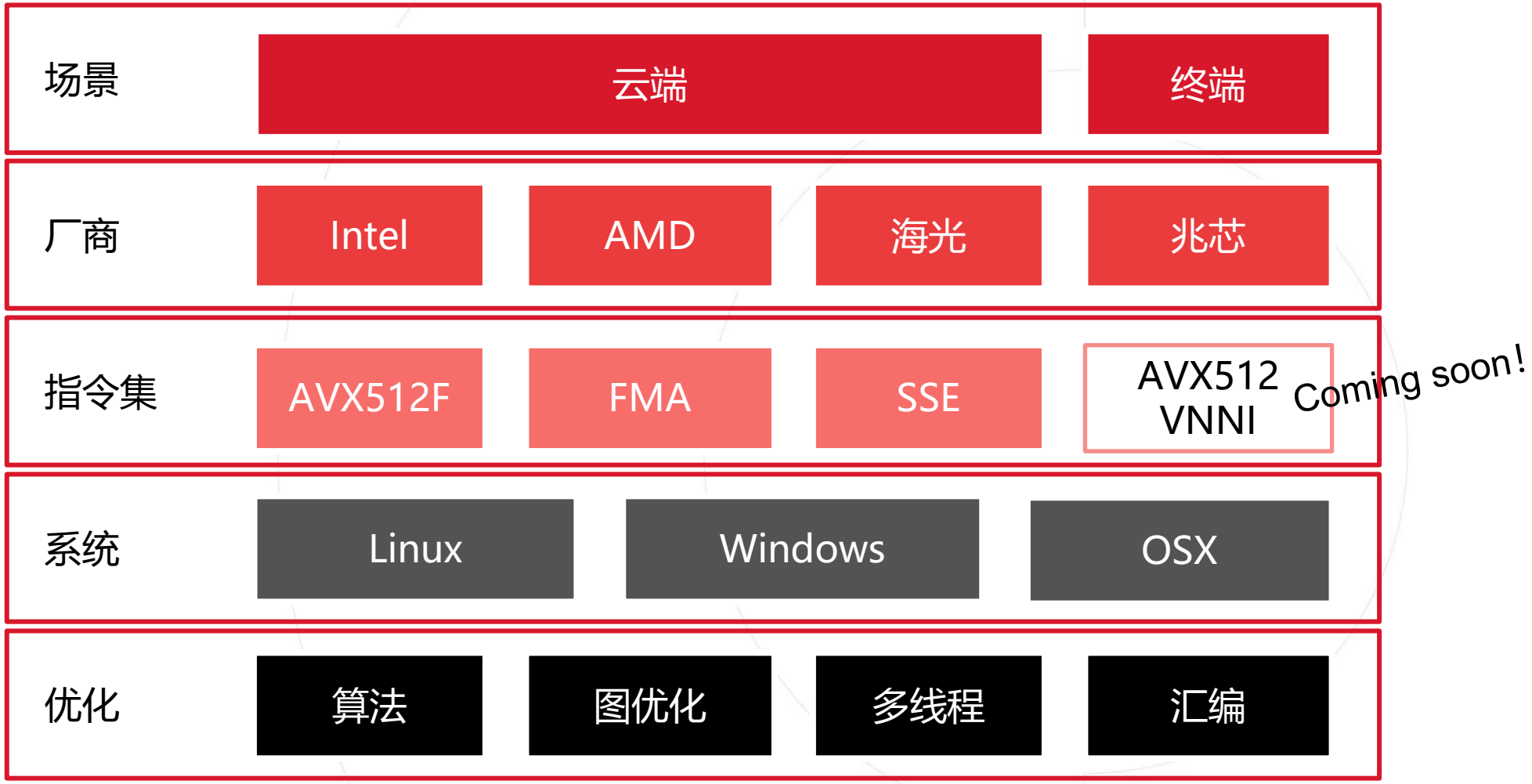
商汤 AI 推理引擎 OpenPPL 实战训练营

梁杰鑫 2022.02.17

Outline

OpenPPL X86 概述

OpenPPL X86 是针对高性能X86服务器优化的深度学习推理引擎后端。同时也配了多种场景，兼容多厂商，系统，并在各种指令集进行了高度优化。



X86服务器架构特点

核心/非核心

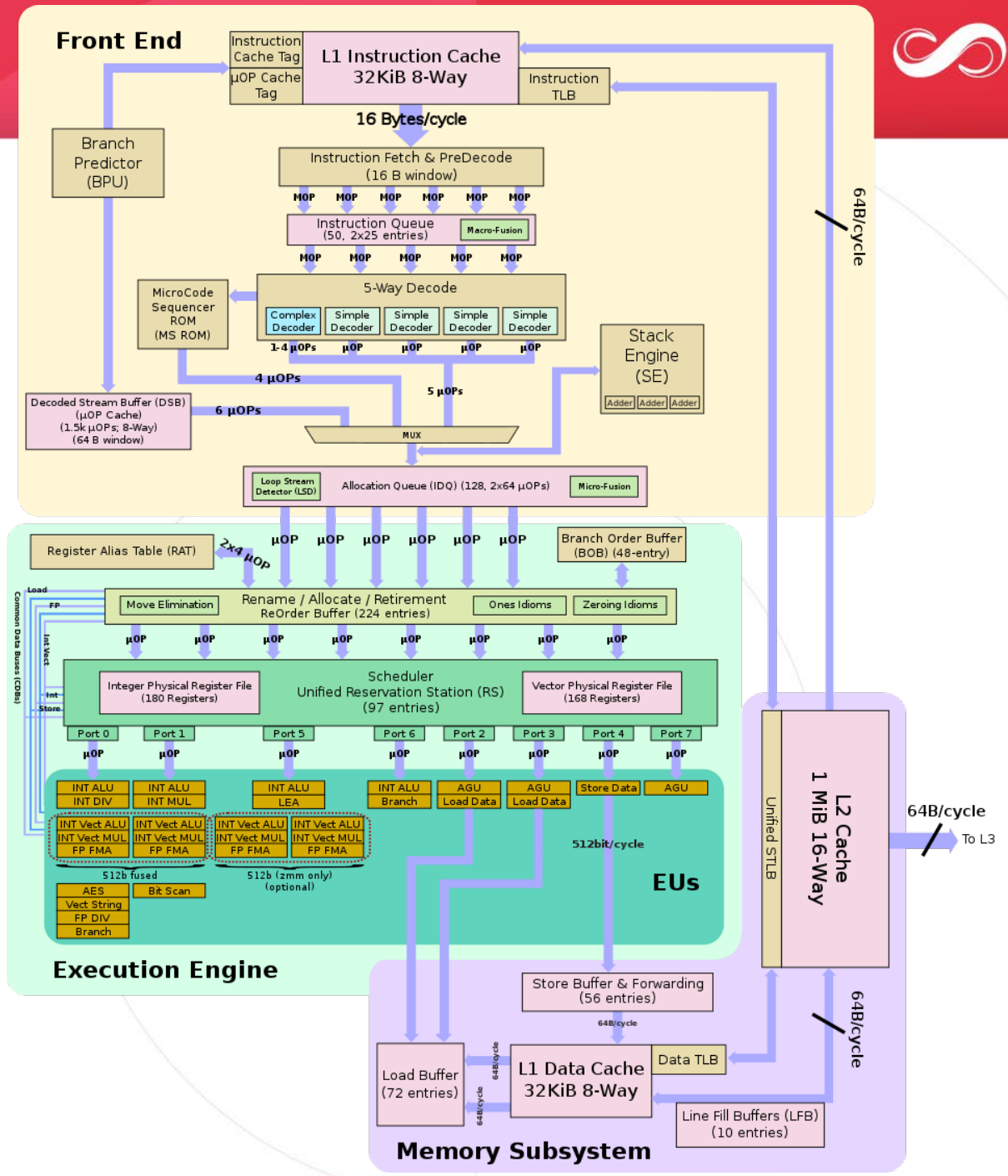
算力/带宽

访存瓶颈

算法设计

核心微架构

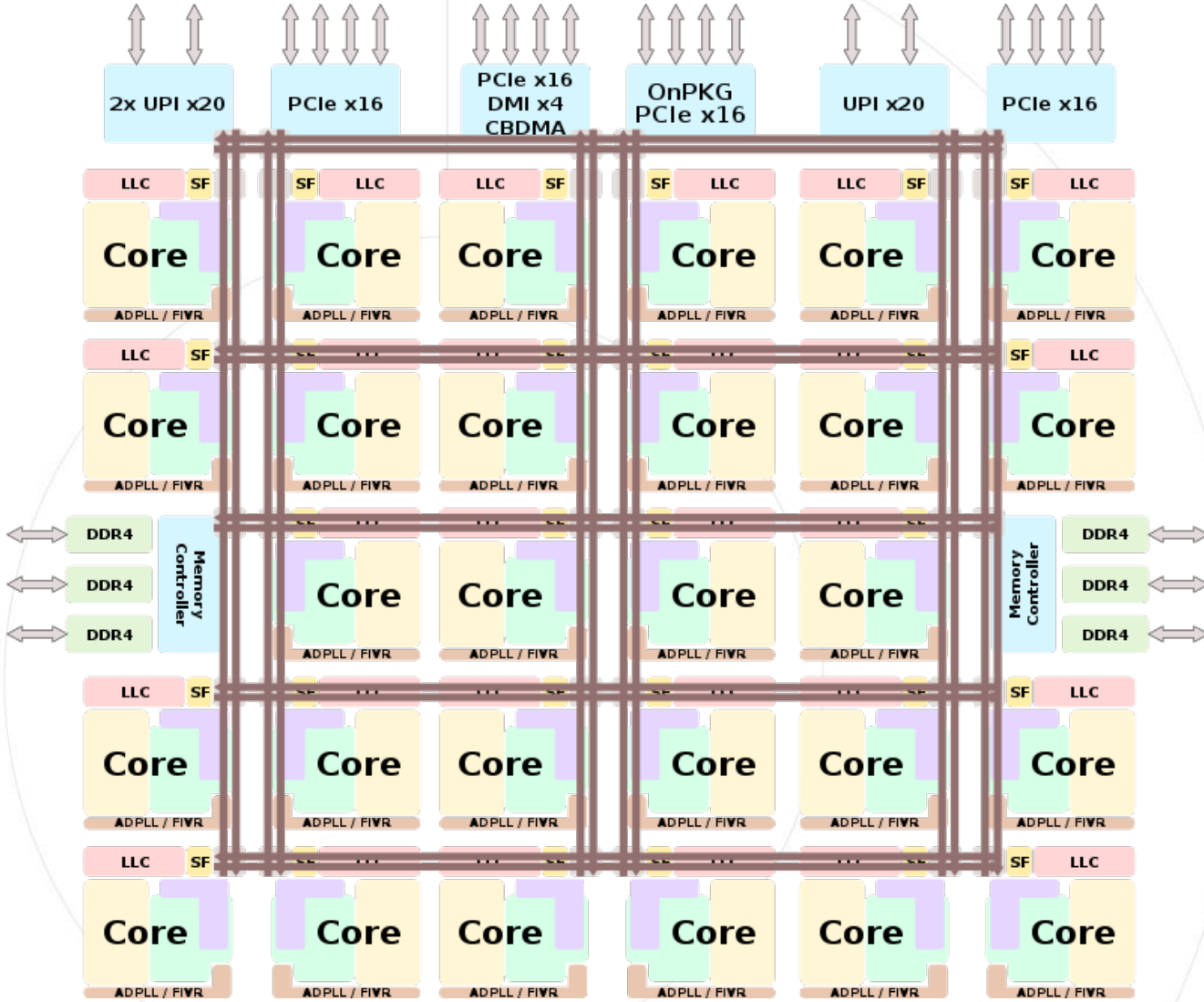
- Skylake X/Cascade Lake资源配置
  - 5MOP前端 + 6uOP后端
  - 256-bit Vector x 4（2可用）AVX2/FMA/SSE
  - 512-bit Vector x 2（低至x1）AVX512
    - 可能出现只有一个512-bit单元的情况
  - 2 Load + 1 Store + 2 FMA
  - 168 Vector 物理寄存器
  - 32KB L1-I Cache
  - 32KB L1-D Cache
  - 1MB L2 Cache



Ref[pic]. https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake

非核心结构——Intel

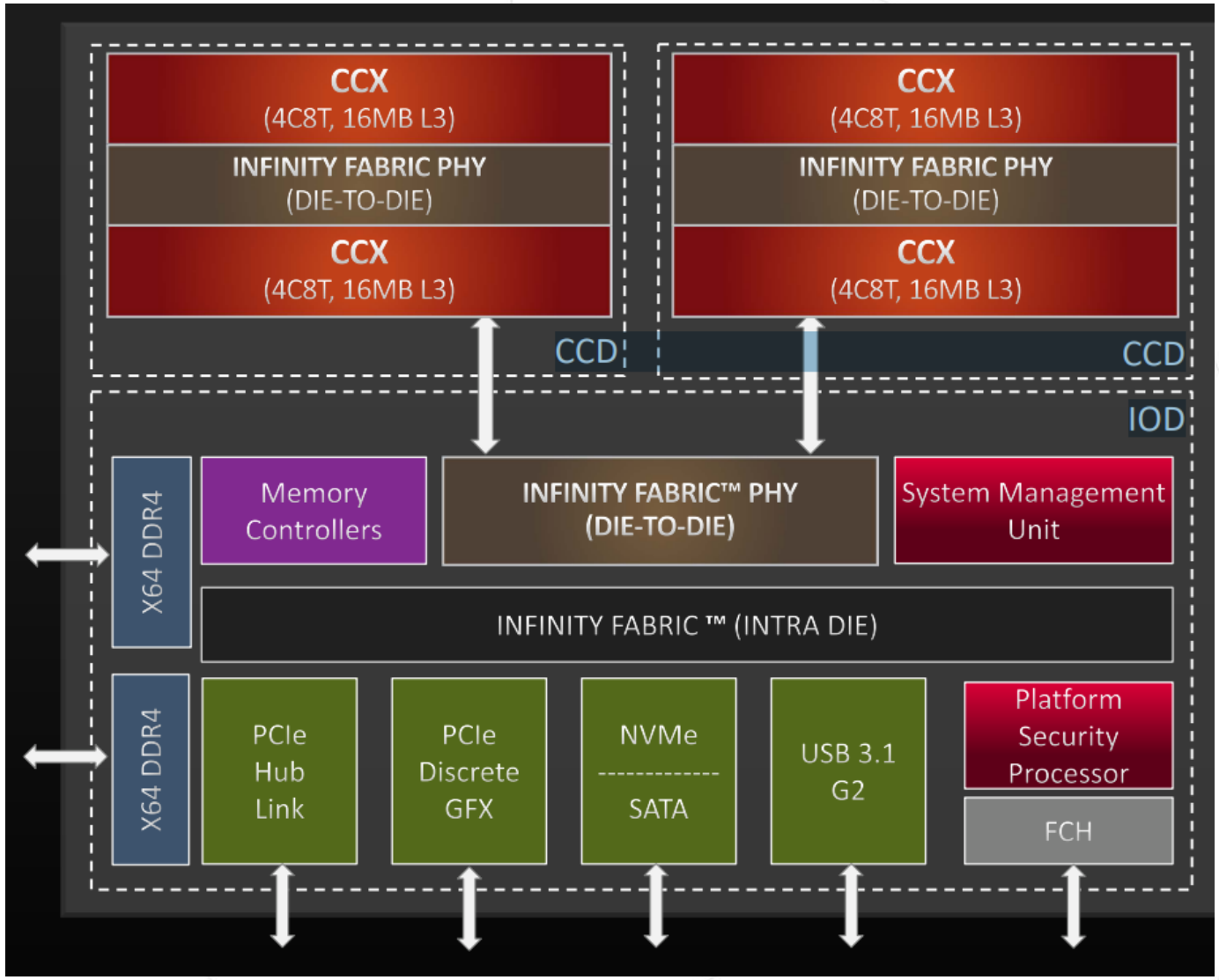
- Intel多核互联以及LLC分布
  - 核心之间，核心与其他部件通过Mesh总线连接
    - 每一条总线为半Ring总线
    - 路由逻辑：先垂直方向，后水平方向
  - 每个核心拥有一块LLC（LastLevelCache）（L3）
    - 分布式LLC，1.375MB/核心
    - 不同核心的LLC通过Mesh总线全核共享
    - 非包含式淘汰缓存，缓存内容与L2 Cache互斥
  - 物理核心编号与操作系统核心编号不一致[1]
    - 操作系统各种核心编号相邻的核心物理上并不相邻



Ref[pic]. https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake
Ref[1] Volcano: Stateless Cache Side-channel Attack by Exploiting Mesh Interconnect

非核心结构——AMD

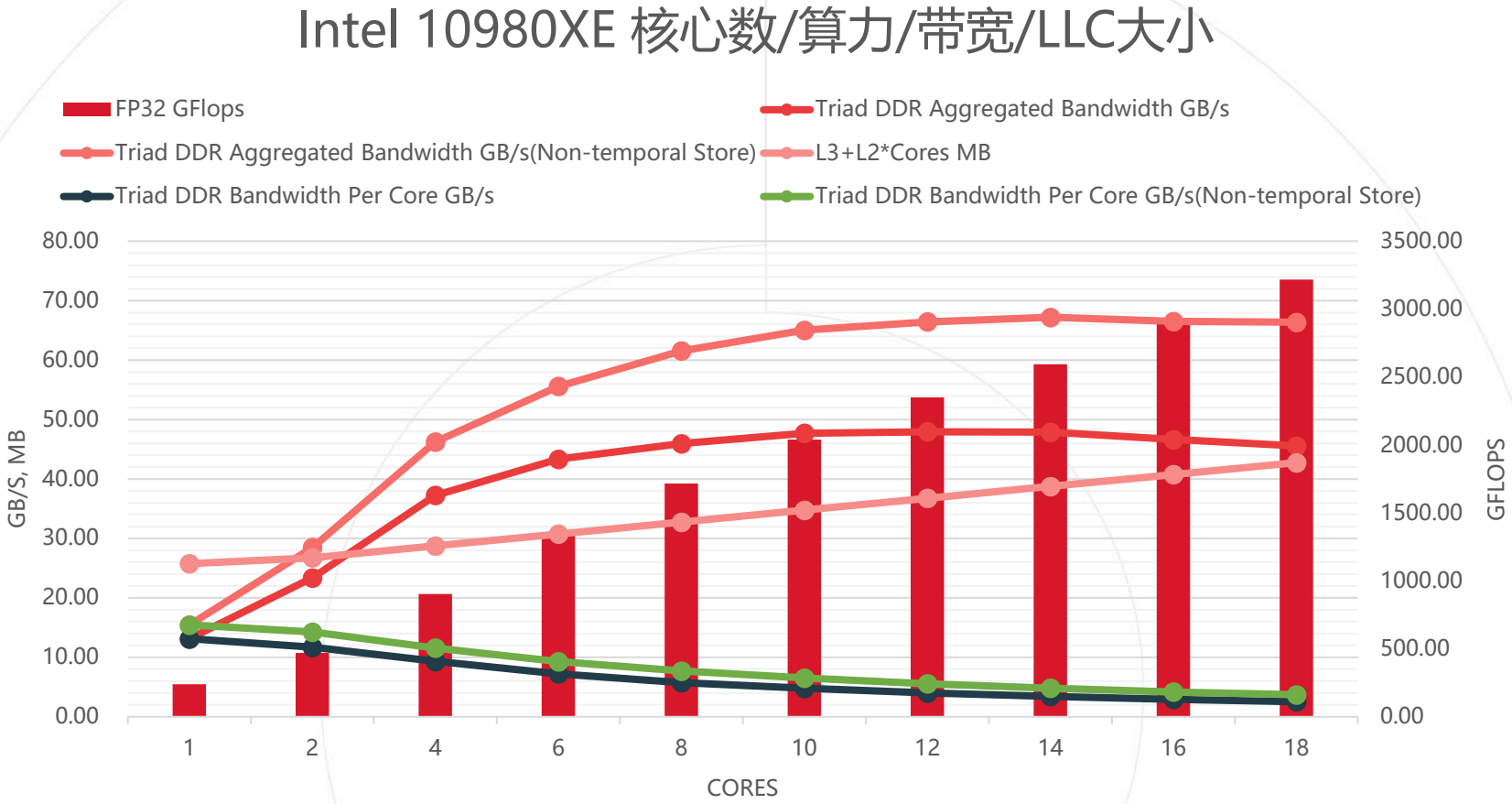
- AMD多核互联以及LLC分布
  - 每个核心拥有一块L3
    - 非包含式淘汰缓存，4MB/核心
    - CCX内所有核心共享L3
  - CCX之间通过IF总线互联



Ref[pic]. HOTCHIP 2019 AMD ZEN2 Sildes

算力/带宽

- 服务器平台的特点
  - 算力高，扩展性相对线性
  - 单核心带宽不高
  - 多核心带宽很低，可扩展性弱
- Non-temporal Store
  - 绕过缓存系统直接写入内存
  - 减少不必要的缓存读写
  - 使用前提：地址对齐16Byte



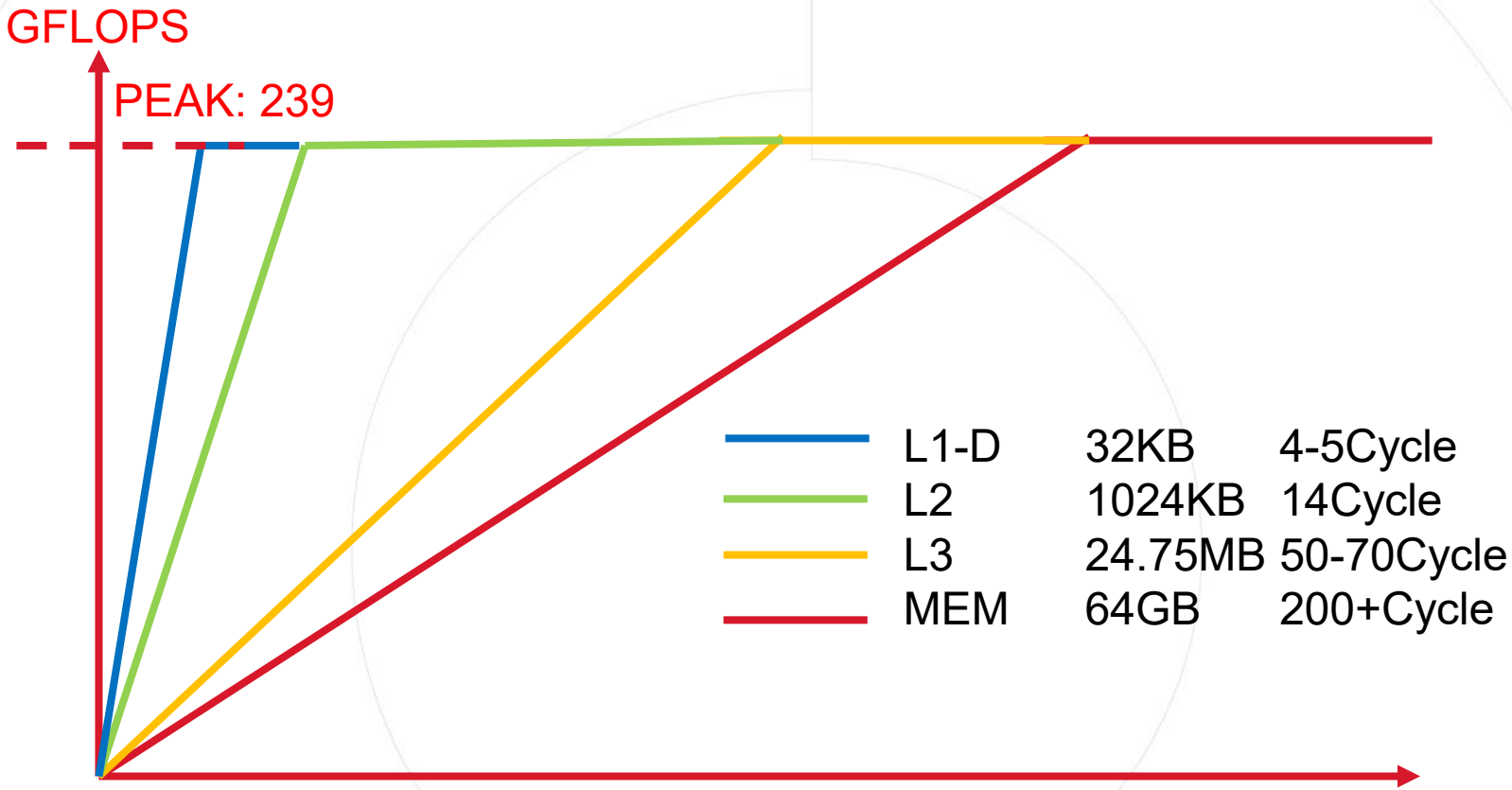
- 浮点峰值测试：https://github.com/pigirons/cpufp
- 访存带宽测试：https://github.com/jeffhammond/STREAM（需要自行改写AVX512）

算力/带宽

- Roofline

Scenes	Bandwidth GB/s
L1 RD Per Core	250
L1 WR Per Core	134
L2 RD Per Core	124
L2 WR Per Core	79
L3 Triad 1 Core	21
L3 Triad 18 Core	273

10980XE 各级缓存的大致速度

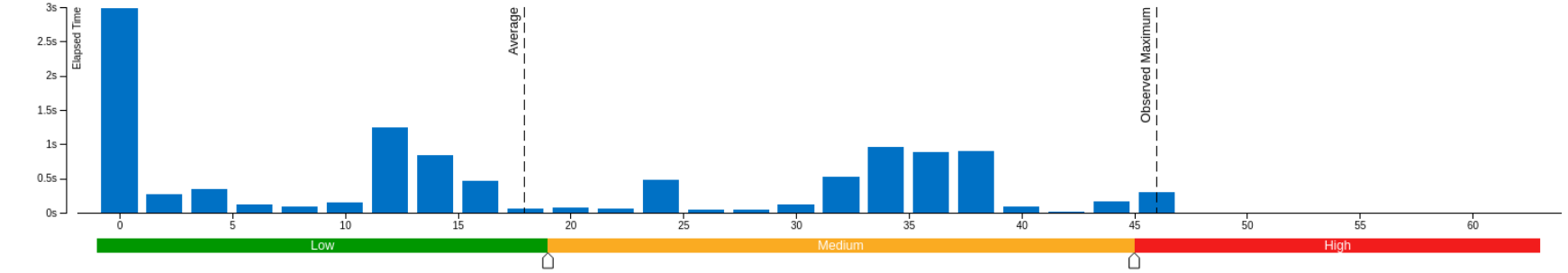


10980XE 单核各级存储结构的大小/延迟/Roofline

Ref. https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake

访存瓶颈

- 实际程序的访存瓶颈
  - Roofline模型的理想条件是计算访存完全掩盖，并不能完全反映实际情况
  - 有空间局部性，拐点与芯片缓存结构强相关，在各级Cache都有可能发生
  - 有时间局部性，瓶颈往往只发生在某一段时间，可以具体到某个代码段
- VTune抓取的DDR带宽（test_conv2d 18-thread Winograd on VGG16）



上图为整个程序的带宽直方图
下图为内存带宽占用的时间轴

已经很好了
但是对于细致调优来说
还是比较粗略

访存瓶颈

- 实际程序的运行时间
  - 例子：一种GEMM写法
  - 访存密集
    - Pack B, 1次
    - Pack A, [M/mb]次
    - Unpack C, [M/mb] x [N/nb]次
  - 计算密集
    - A^ x B_j, [M/mb] x [N/nb]次
  - 计算时间
    - Total Pack B + Total Pack A + Total Unpack C + Total A^ x B_j
    - 数据有多大？数据在哪一级存储？更优的写法？（可以复习《通用架构下的性能优化概要》）

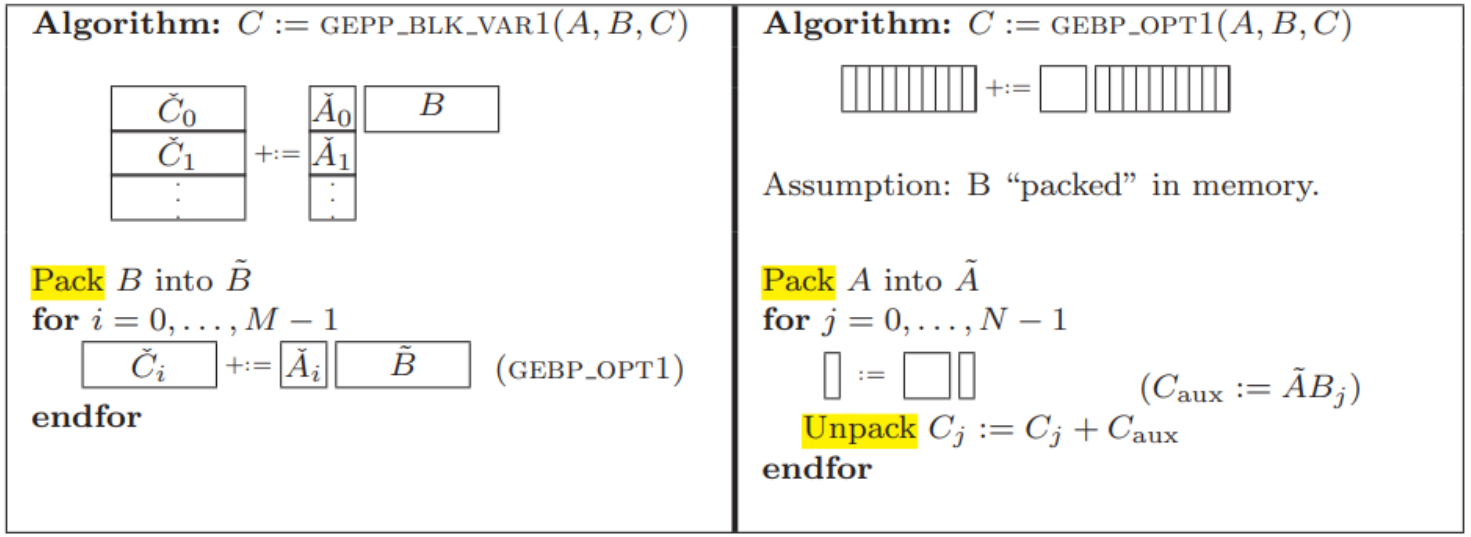
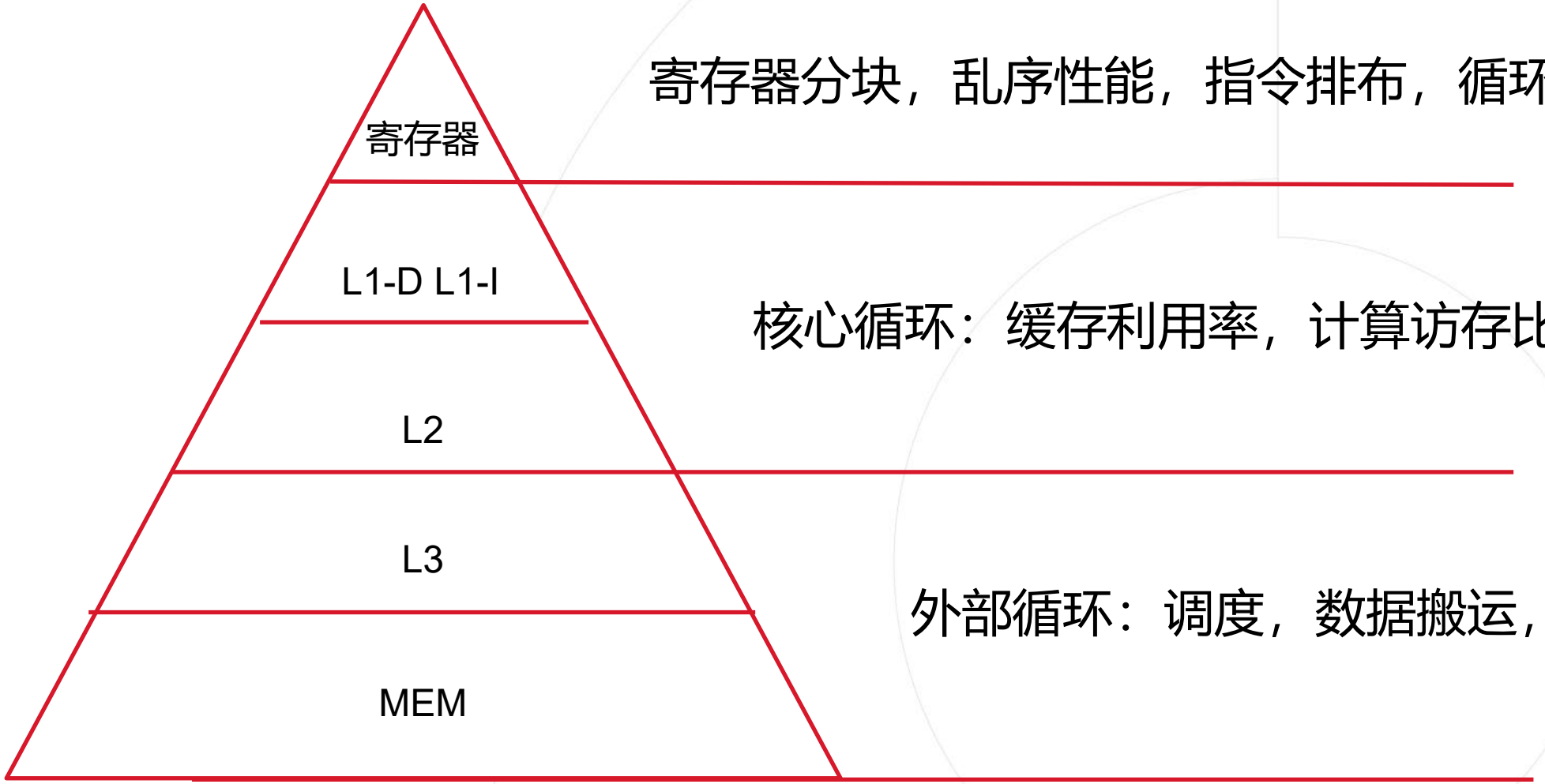


Fig. 8. Optimized implementation of GEPP (left) via calls to GEBP_OPT1 (right).

Ref. [pic] Anatomy of High-Performance Matrix Multiplication

算法设计



寄存器分块，乱序性能，指令排布，循环展开

核心循环：缓存利用率，计算访存比，预取

外部循环：调度，数据搬运，流程设计

OpenPPL X86 算法设计

数据排布

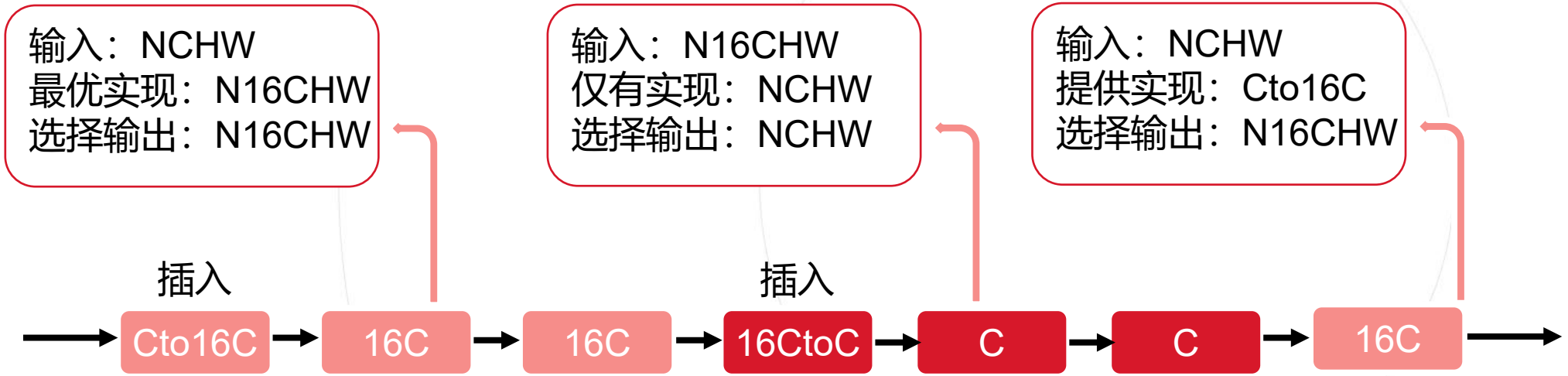
卷积算法

数据排布

- 输入输出：N16CHW（nChw16c）
  - NCHW：[N, C, H, W]
  - NHWC：[N, H, W, C]
  - N16CHW：[N, ⌈C/16⌉, H, W, 16]
    - 介于NCHW与NHWC之间，一定程度规避维度之间跨度过长
    - 天然的Packing和数据对齐，512-bit Vector 可以容纳16个float数据
  - 在行主元矩阵乘中，B矩阵几乎无法避免做Packing
    - NCHW: w x in(ic, ih*iw) = out(oc, oh*ow)，无法预处理B，C不对齐
    - NHWC: in(ih*iw, ic) x w = out(oh*ow, oc)，可以预处理B，可能需要Pack A矩阵，AC不对齐
    - N16CHW: in(⌈ic/16⌉, ih*iw, 16) x w = out(⌈oc/16⌉, oh*ow, 16)，A天然Packing，AC对齐

数据排布

- 数据排布选择策略
  - 不进行全局的数据排布优化
  - 贪心策略：选择每一个算子根据【输入数据排布】来选择局部最优的【输出数据排布】
  - 数据排布转换：插入Reorder算子



卷积算法——直接卷积

- 相比于IM2COL GEMM卷积，更利于多核计算，更容易提高效率
  - 同样的空间占用下，计算访存比高
  - 一定条件下可以免除数据Packing
- N16CHW下的直接卷积
  - 以3x3卷积为例
  - 汇编优化：
  - 高叔叔的《**关于sgemm_hsw的一点解释说明**》

https://zhuanlan.zhihu.com/p/426127316

input

(0,0),(0:15)	(0,1),(0:15)	(0,2),(0:15)	(0,3),(0:15)
(1,0),(0:15)	(1,1),(0:15)	(1,2),(0:15)	(1,3),(0:15)
(2,0),(0:15)	(2,1),(0:15)	(2,2),(0:15)	(2,3),(0:15)
(3,0),(0:15)	(3,1),(0:15)	(3,2),(0:15)	(3,3),(0:15)

KH=0,KW=0,IC=0,OC=0:15
KH=0,KW=0,IC=1,OC=0:15
KH=0,KW=0,IC=2,OC=0:15
KH=0,KW=0,IC=3,OC=0:15
KH=0,KW=0,IC=4,OC=0:15
KH=0,KW=0,IC=5,OC=0:15
KH=0,KW=0,IC=6,OC=0:15
KH=0,KW=0,IC=7,OC=0:15
KH=0,KW=0,IC=8,OC=0:15
KH=0,KW=0,IC=9,OC=0:15
KH=0,KW=0,IC=10,OC=0:15
KH=0,KW=0,IC=11,OC=0:15
KH=0,KW=0,IC=12,OC=0:15
KH=0,KW=0,IC=13,OC=0:15
KH=0,KW=0,IC=14,OC=0:15
KH=0,KW=0,IC=15,OC=0:15

weight

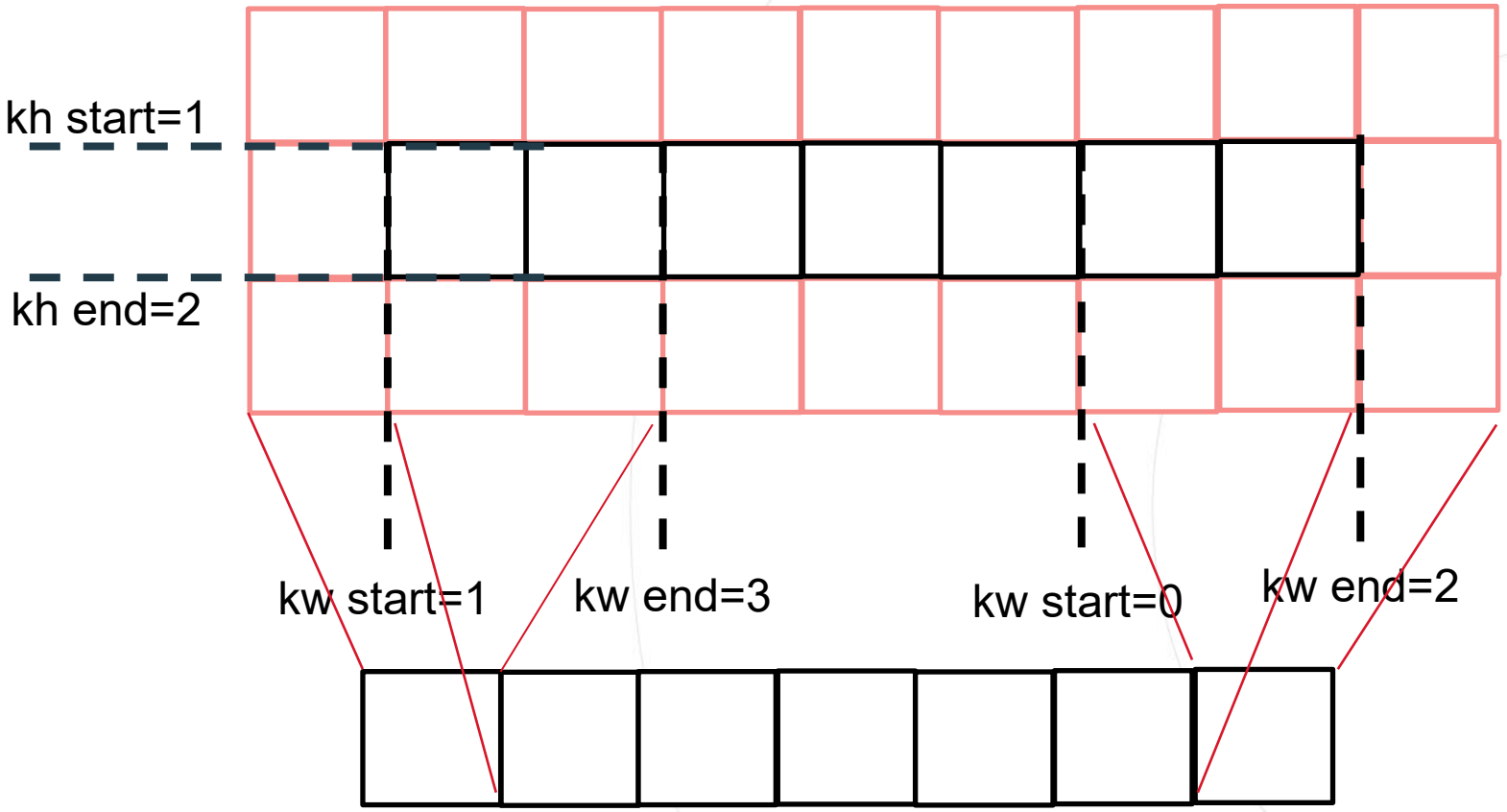
```
vector16 OUT = {0}
vector16 W = {0}
scalar IN = 0
FOR kh = 0:2
FOR kw = 0:2
  FOR ic = 0:15
    k = kh*3+kw
    IN = input(ih+kh,iw+kw,ic)
    W = weight(k*16*16,ic*16,0:16)
    OUT += IN * W
output(oh,ow,0:16) = OUT
```

output

(0,0),(0:15)	(0,1),(0:15)	(0,2),(0:15)
(1,0),(0:15)	(1,1),(0:15)	(1,2),(0:15)
(2,0),(0:15)	(2,1),(0:15)	(2,2),(0:15)

卷积算法——直接卷积

边界处理



Padding部分

非Padding部分

行边界Kernel
- 参数： kh start, kh end
- 参数： kw start, kw end
- 一次计算一个点

非行边界Kernel
- 参数：kh start，kh end
- 一次计算多个点，循环展开

卷积算法——直接卷积

- 不同输入图片尺寸下的调度设计

```
Big Input Image HxW
----------------------------
FOR gb 0:group-1 by Gb
FOR bb 0:batch-1 by Bb
FOR icb 0:IC-1 by ICb
  #OMP PARALLEL COLLAPSE(4)
  FOR g in Gb
  FOR b in Bb
  FOR ocb 0:OC-1 by OCb
  FOR oh 0:OH-1
    FOR oc in OCb by OCr
      compute ow border left
      compute ow by Wr
      compute ow border right
```

May use non-temporal store

```
Small Input Image HxW
----------------------------
FOR gb 0:group-1 by Gb
FOR bb 0:batch-1 by Bb
FOR icb 0:IC-1 by ICb
  #OMP PARALLEL COLLAPSE(4)
  FOR g in Gb
  FOR b in Bb
  FOR ic in ICb by 16
  FOR ih 0:IH-1
    make padding width input
  #OMP PARALLEL COLLAPSE(4)
  FOR g in Gb
  FOR b in Bb
  FOR ocb 0:OC-1 by OCb
  FOR oh 0:OH-1
    FOR oc in OCb by OCr
      compute ow by Wr
```

Hold data in Global L2/L3

卷积算法——Winograd

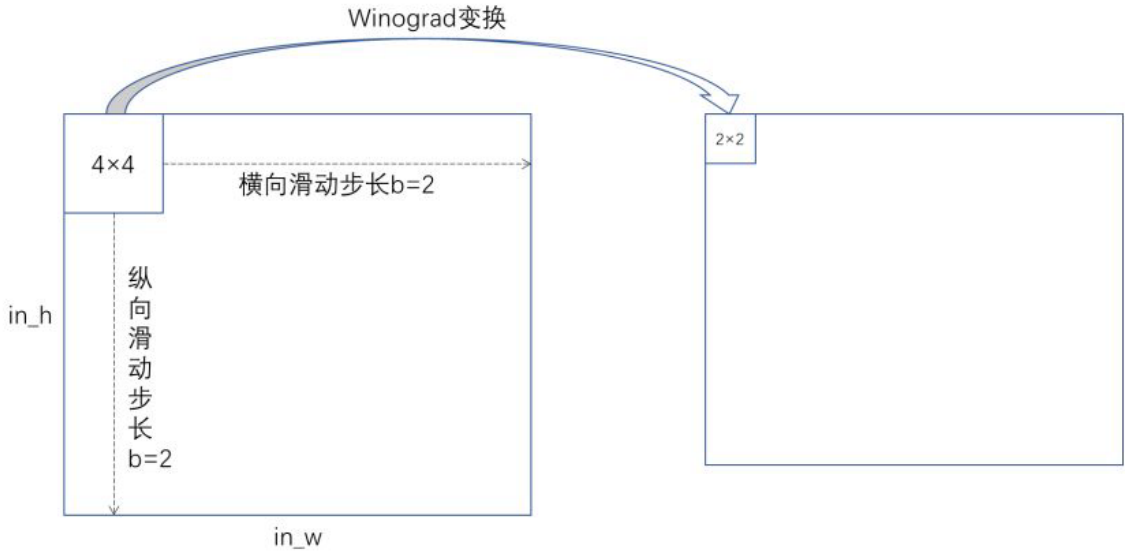
- Winograd卷积是什么
  - Andrew Lavin, Scott Gray, Fast Algorithms for Convolutional Neural Network, 2015
  - 灵感：Winograd，一种特殊的FFT变换；FFT卷积：时域卷积->频域乘积

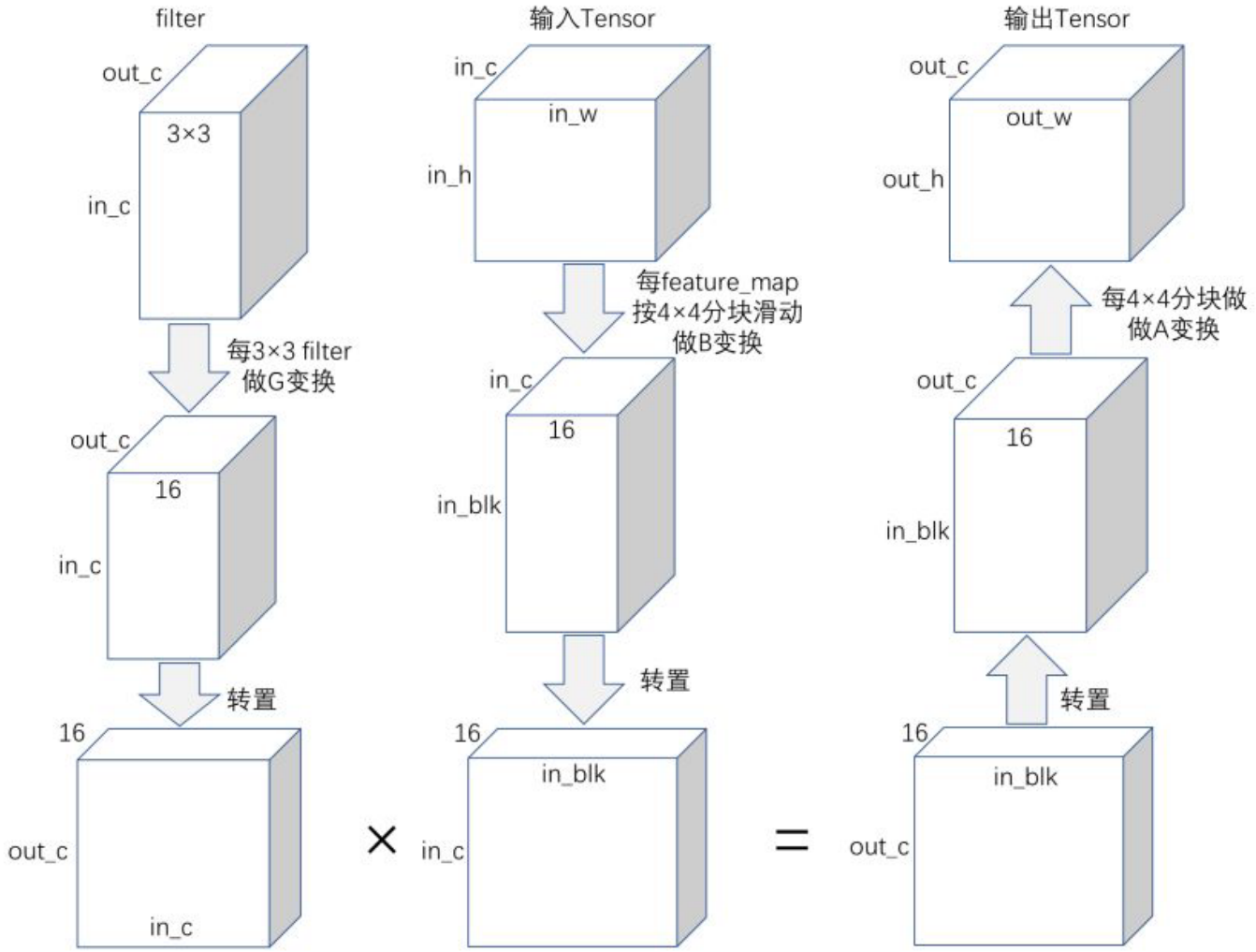
  - 直接卷积：$y = w * X$
  - 二维Winograd卷积:
    - 变换：$\nabla(a, A) = AaA^T$
    - 常数变换矩阵：y, X, w 分别对应 A, B, G
    - 卷积过程：$y = \nabla(\nabla(w, G) \times \nabla(X, B^T), A^T)$

    常数变换矩阵推导：https://github.com/andravin/wincnn

卷积算法——Winograd

- 以Winograd F(2x2, 3x3)为例





- 图文参考高叔叔的文章：《**“远超”理论浮点峰值**》
https://zhuanlan.zhihu.com/p/465739282

- 直接卷积计算量：

$$OH * OW * 3 * 3 * IC * OC * 2$$

- Winograd F(2x2, 3x3) 计算量：

$$\frac{IH}{4} * \frac{IW}{4} * 4^3 * IC * 2 + \frac{OH}{2} * \frac{OW}{2} * (4^2 * 2 * OC + 16 * IC * OC * 2)$$

卷积算法——Winograd

- 调度设计，TILES=batch* [OH/out_blk]*[OW/out_blk]

Big Input BLKs per thread

```
#OMP PARALLEL COLLAPSE(3)
FOR g 0:group-1
FOR ocb 0:OC-1 by OCb
FOR tb in 0:TILES-1 by Tb
  FOR icb 0:IC-1 by ICb
    TRANS_IN = TRANSFORM(IN[icb,tb])
    FOR oc in OCb by OCr
      FOR i 0:(in_blk*in_blk)-1
        TRANS_OUT(i,oc) += GEMM(i,oc)
      IF is last icb
        OUT[oc,tb] = TRANSFORM(TRANS_OUT)
```

Hold Data in Local L2/L3

Small Input BLKs per thread

```
FOR g 0:group-1
FOR tb in 0:TILES-1 by Tb
FOR icb 0:IC-1 by ICb
  #OMP PARALLEL
  TRANS_IN = TRANSFORM(IN[icb,tb])

  FOR ocb 0:OC-1 by OCb
    #OMP PARALLEL COLLAPSE(2)
    FOR i 0:(in_blk*in_blk)-1
    FOR oc in OCb by OCr
      TRANS_OUT(i, oc) += GEMM(i, oc)
    IF is last icb
      #OMP PARALLEL
      OUT[ocb,tb] = TRANSFORM(TRANS_OUT)
```

Hold data in Global L2/L3

OpenPPL X86 图优化

- Depthwise卷积融合
- 就地计算
- 其他融合

Depthwise融合

- MobileNet中的常见结构
- 数据量大于LLC，可能因带宽受限加速不明显

Conv3x3 计算访存比

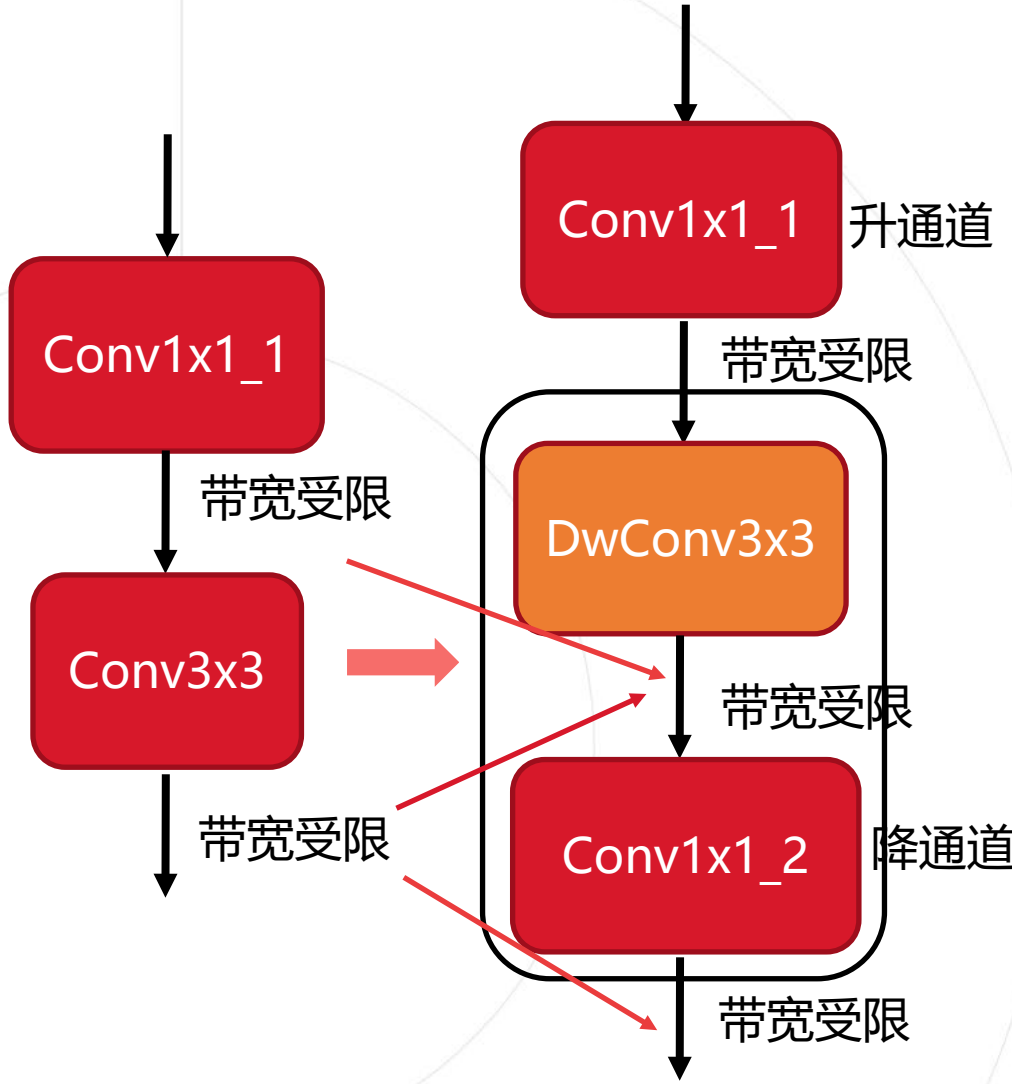
$$\frac{IC * OC * OH * OW * 3(KH) * 3(KW) * 2}{(IC * IH * IW + OC * OH * OW + IC * OC * 3 * 3) * sizeof(float)}$$

DwConv3x3 计算访存比

$$\frac{OC * 3 * 3 * OH * OW * 2}{(2 * OC * OH * OW + OC * 3 * 3) \times sizeof(float)}$$

Conv1x1_2 计算访存比

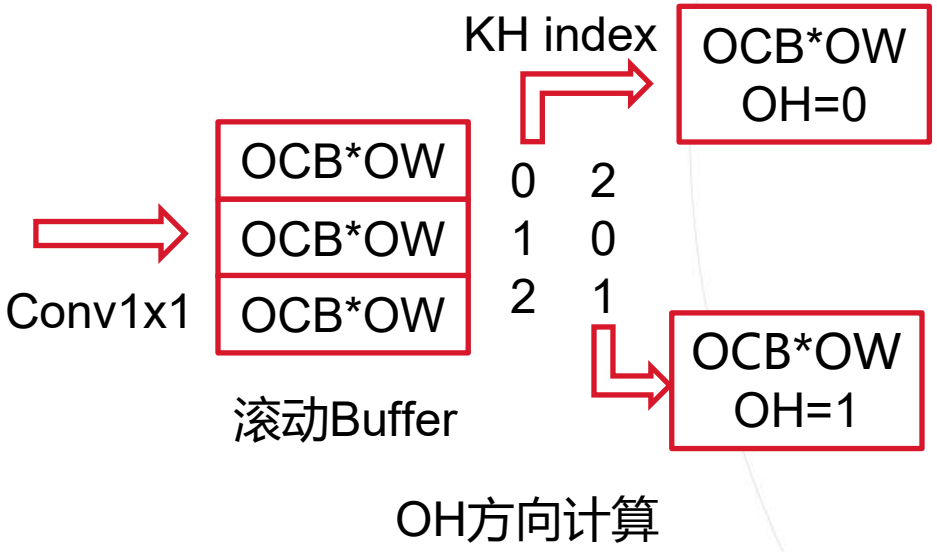
$$\frac{IC * OC * OH * OW * 2}{(IC * IH * IW + OC * OH * OW + IC * OC) \times sizeof(float)}$$



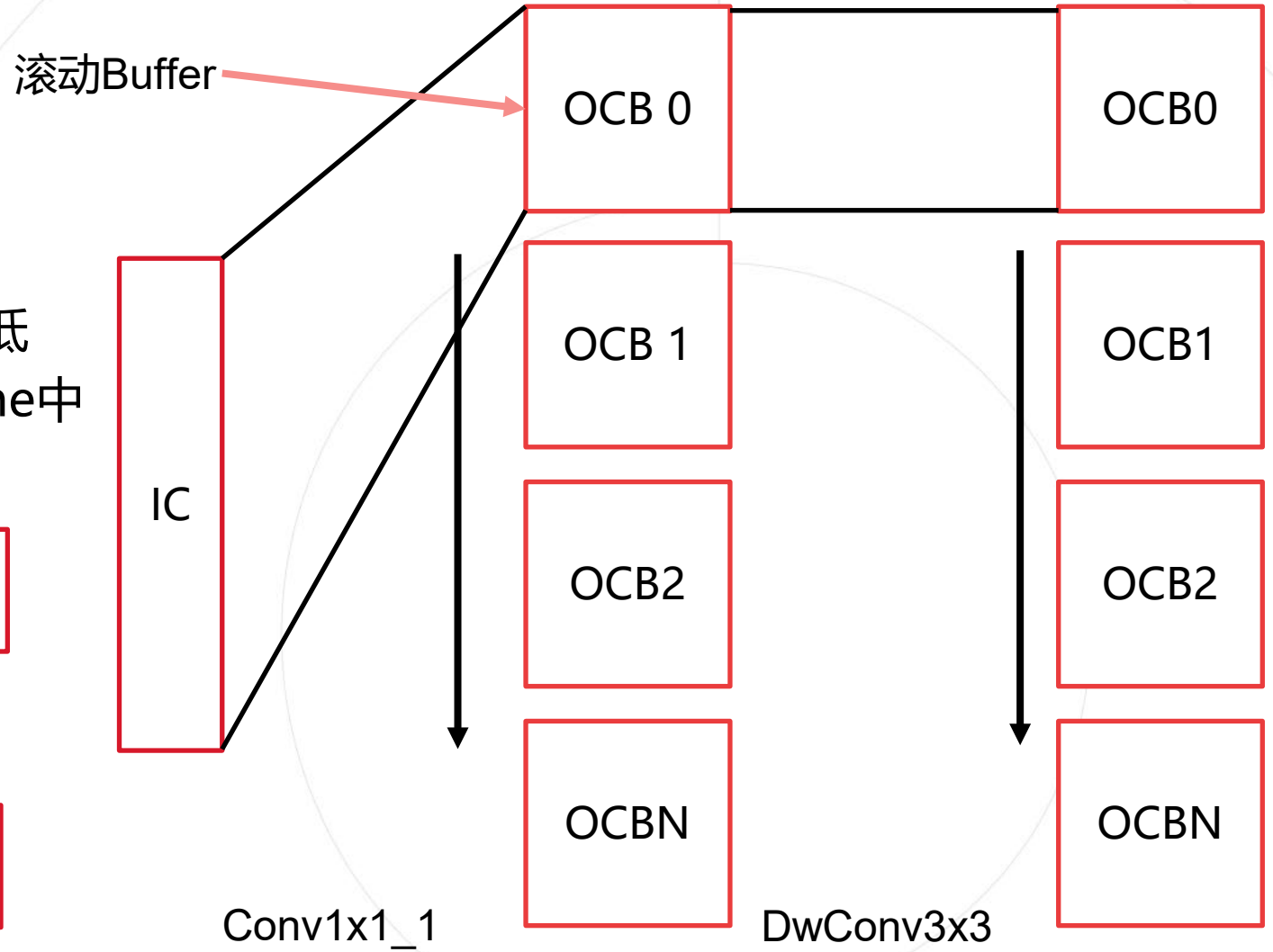
Depthwise融合

融合设计

- 前提条件：
  - 保证Conv1x1的效率不能降低
  - 中间计算结果能装进L2 Cache中



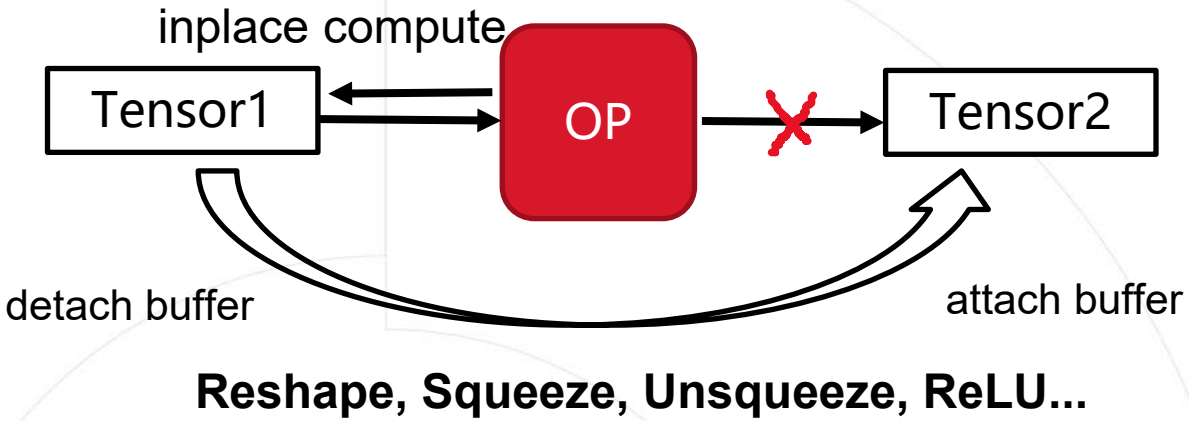
OH方向计算



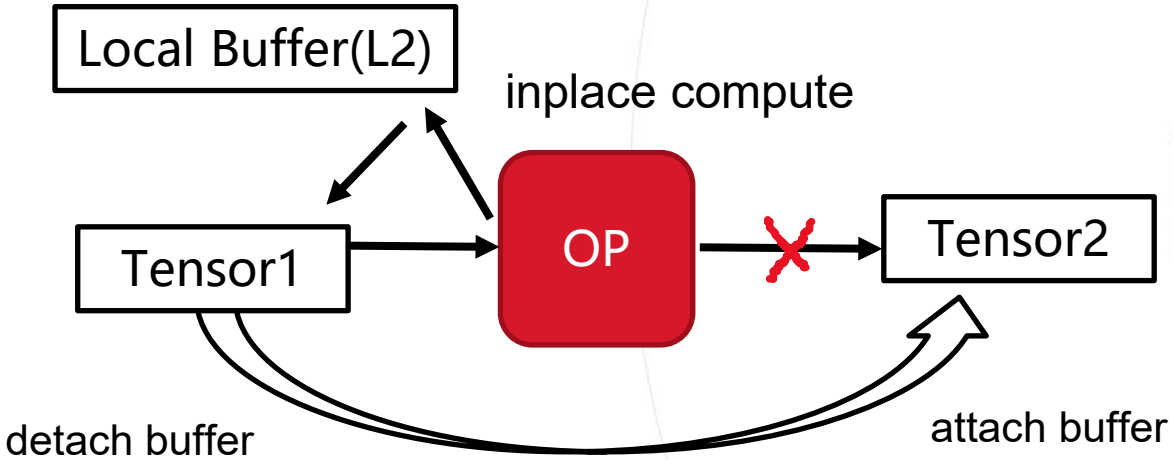
OC方向计算

就地计算

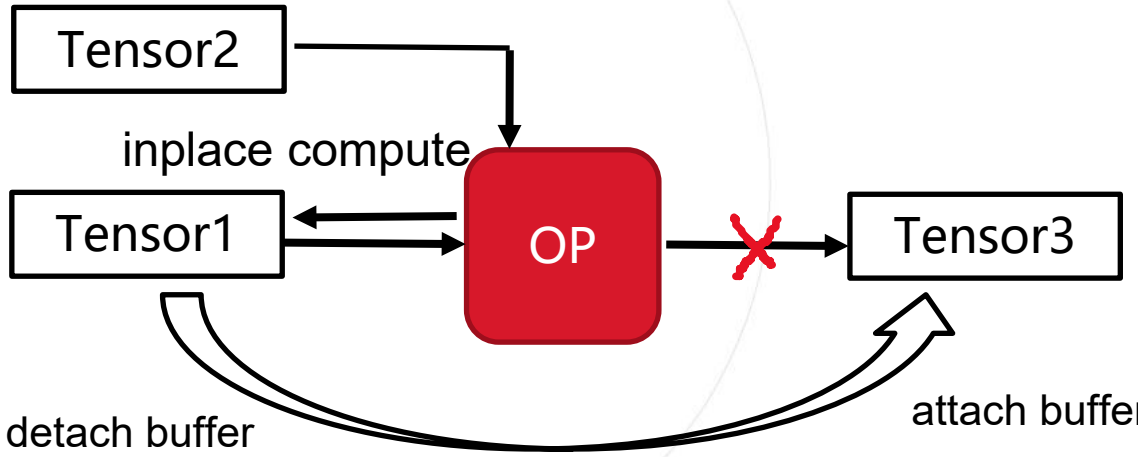
- 应用场景
  - 访存瓶颈
  - 数据不会重复使用
  - 输入+输出大小>LLC



Reshape, Squeeze, Unsqueeze, ReLU...



Reorder

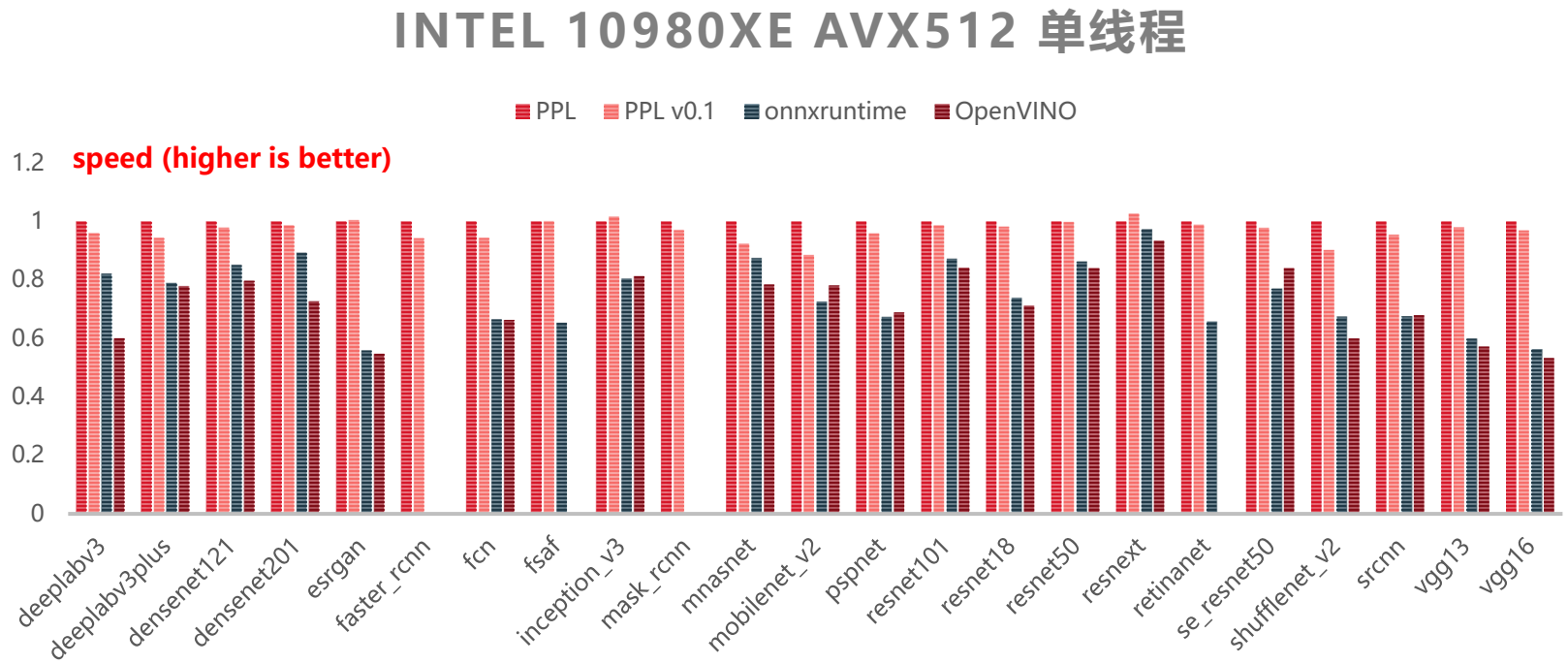


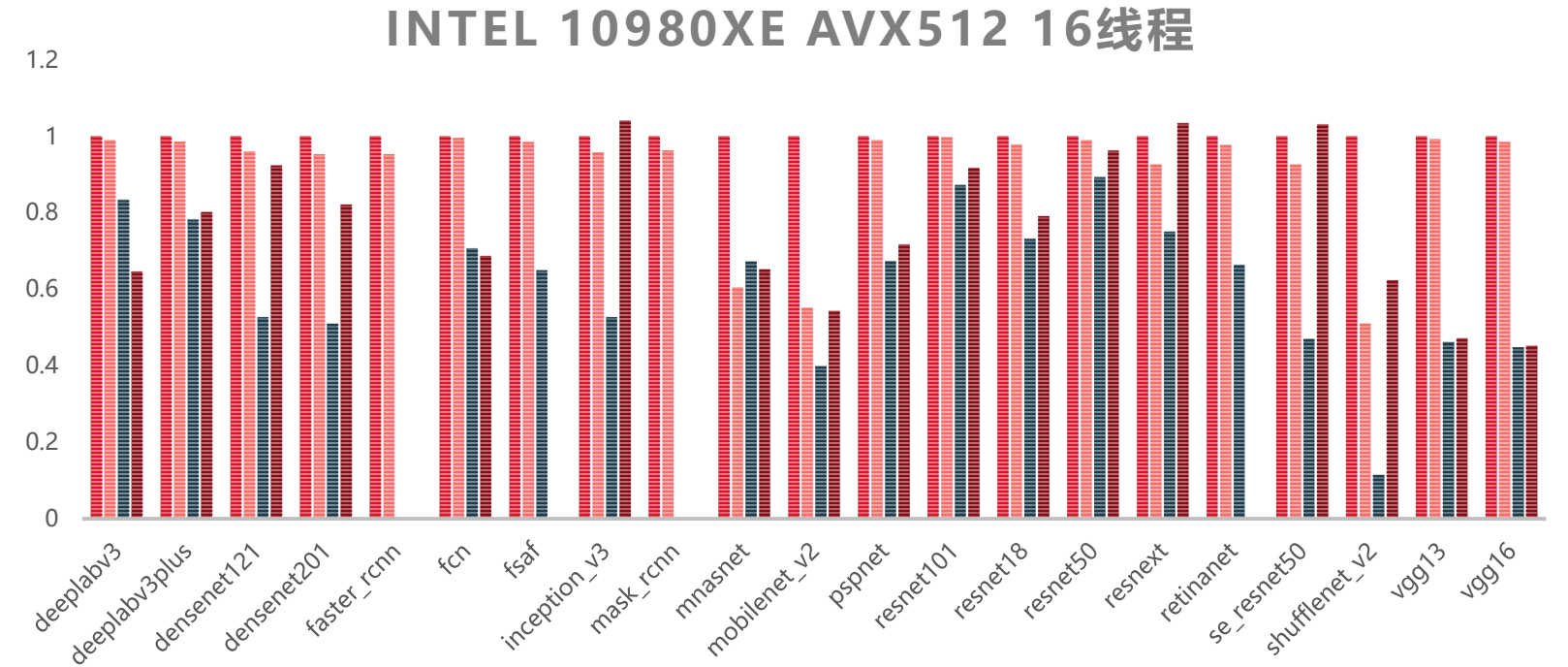
Add, Sub, Div, Mul...

其他融合

- 后处理融合
  - Conv+BN+ReLU/ReLU6+Add
  - Gemm+ReLU
  - Add+ReLU
  - BN+ReLU
- 特殊结构融合
  - ChannelShuffle
  - Swish

OpenPPL X86 性能





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

THANK YOU

Q&A

OpenPPL 微信公众号

OpenPPL QQ 交流群

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