推理引擎 OpenPPL 实战训练营

商汤 sensetime

OpenPPL+RISC-V 指令集初探

2022年2月25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

焦明俊

商汤科技高性能计算工程师
OpenPPL CPU 开发人员

- 硕士毕业于复旦大学
- 目前在商汤科技高性能计算部门负责参与 CPU 架构方向的 PPL NN/CV 研发与优化

课程提纲

OpenPPL RISC-V 是针对全志 AllWinner D1 平台 RISC-V 架构的深度学习推理引擎，本次分享将结合 RISC-V V 指令集以及全志 D1 开发板的微架构特点，对推理引擎的算法设计进行技术解析。

1. OpenPPL RISC-V 简介
2. RVV 指令集概述
3. AllWinner D1 微架构特点介绍
4. OpenPPL RISC-V 算法设计
5. OpenPPL RISC-V 性能展示

实战训练营

推理框架这件小事儿
OpenPPL RISCV 技术分享

商汤 AI 推理引擎 OpenPPL 实战训练营

焦明俊 2022.02.24

Outline

Outline

RVV 指令集概述

- RISCV 是一个基于精简指令集 risc 原则的开源指令集架构，其指令使用模块化设计，包括基本指令集以及额外可选择的扩展指令集。

指令集名称	描述	版本	状态[a]
基本指令集			
RV32I	基本整数指令集, 32位	2.1	冻结
RV32E	基本整数指令集(嵌入式系统), 32位, 16 寄存器	1.9	开放
RV64I	基本整数指令集, 64位	2.1	冻结
RV128I	基本整数指令集, 128位	1.7	开放
标准扩展指令集			
M	整数乘除法标准扩展	2.0	冻结
A	不可中断指令(Atomic)标准扩展	2.1	冻结
F	单精确度浮点运算标准扩展	2.2	冻结
D	双倍精确度浮点运算标准扩展	2.2	冻结
G	所有以上的扩展指令集以及基本指令集的总和的简称	不适用	不适用
Q	四倍精确度浮点运算标准扩展	2.2	冻结
L	十进制浮点运算标准扩展	0.0	开放
C	压缩指令标准扩展	2.0	冻结
B	位操作标准扩展	0.93	开放
J	动态指令翻译标准扩展	0.0	开放
T	顺序存储器访问标准扩展	0.0	开放
P	单指令多资料流（SIMD）运算标准扩展	0.2	开放
V	向量运算标准扩展	1.0RC	开放
N	用户中断标准扩展	1.1	开放

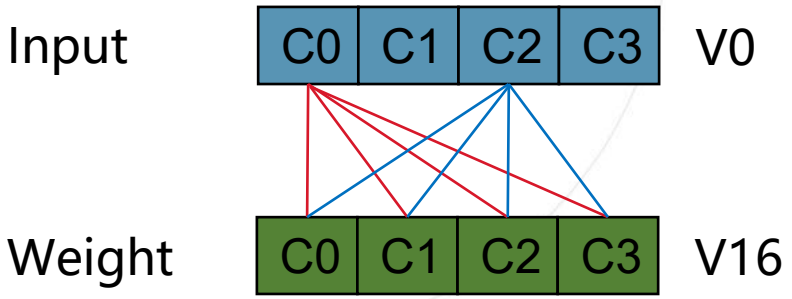
Ref. https://en.wikipedia.org/wiki/RISC-V

RVV 指令集概述

- 向量设置/控制指令
  - vsetvl、vsetvli

    用来配置 vl、vtype 寄存器
- 向量加载存储指令
  - vle.v、vse.v
- 读取/修改向量寄存器指令
  - vmv.v.v、vmv.v.x、vmv.v.i
  - vrgather.v.v、vrgather.v.x、vrgather.v.i

- 向量算术运算指令
  - vfadd.vv、vfadd.vf
  - vfsub.vv、vfsub.vf
  - vfmul.vv、vfmul.vf
  - vfdiv.vv、vfdiv.vf
  - vfmacc.vv、vfmacc.vf
    - 对比 arm neon： vfma



Outline

AllWinner D1 微架构特点

- C906 处理器的体系结构特点（用户手册）
  - 支持 RV64 IMA[FD]C[V]指令子集
  - 5级单发射顺序执行流水线
  - 一级哈佛结构的指令/数据缓存（各32K）
  - 算力可达 4GFlops（@1GHz）
  - 遵循 RVV 向量扩展标准（RVV 0.7.1）
  - 支持 int8/int16/int32/int64/fp16/fp32/fp64/bfp16 向量运算

Ref. https://www.t-head.cn/product/C906?spm=a2ouz.12986968.0.0.7bfc2be0qiQoyn

AllWinner D1 微架构特点

- 实测性能参数
  - 指令吞吐

    通过构建无数据依赖的相互独立的指令流来测试每条指令的执行周期数(CPI)
  - 指令延迟

    通过构建具有写后读相关性的指令流序列来测试不同指令的执行延迟
  - 访存带宽

    通过构建 Cache 命中率为百分百和零的两种指令流分别测试 Cache 和主存 DDR 的带宽

Ref. 许志耿《OpenPPL之通用架构下的性能优化概要》

AllWinner D1 微架构特点

- 实测性能参数
  - 指令吞吐、指令延迟

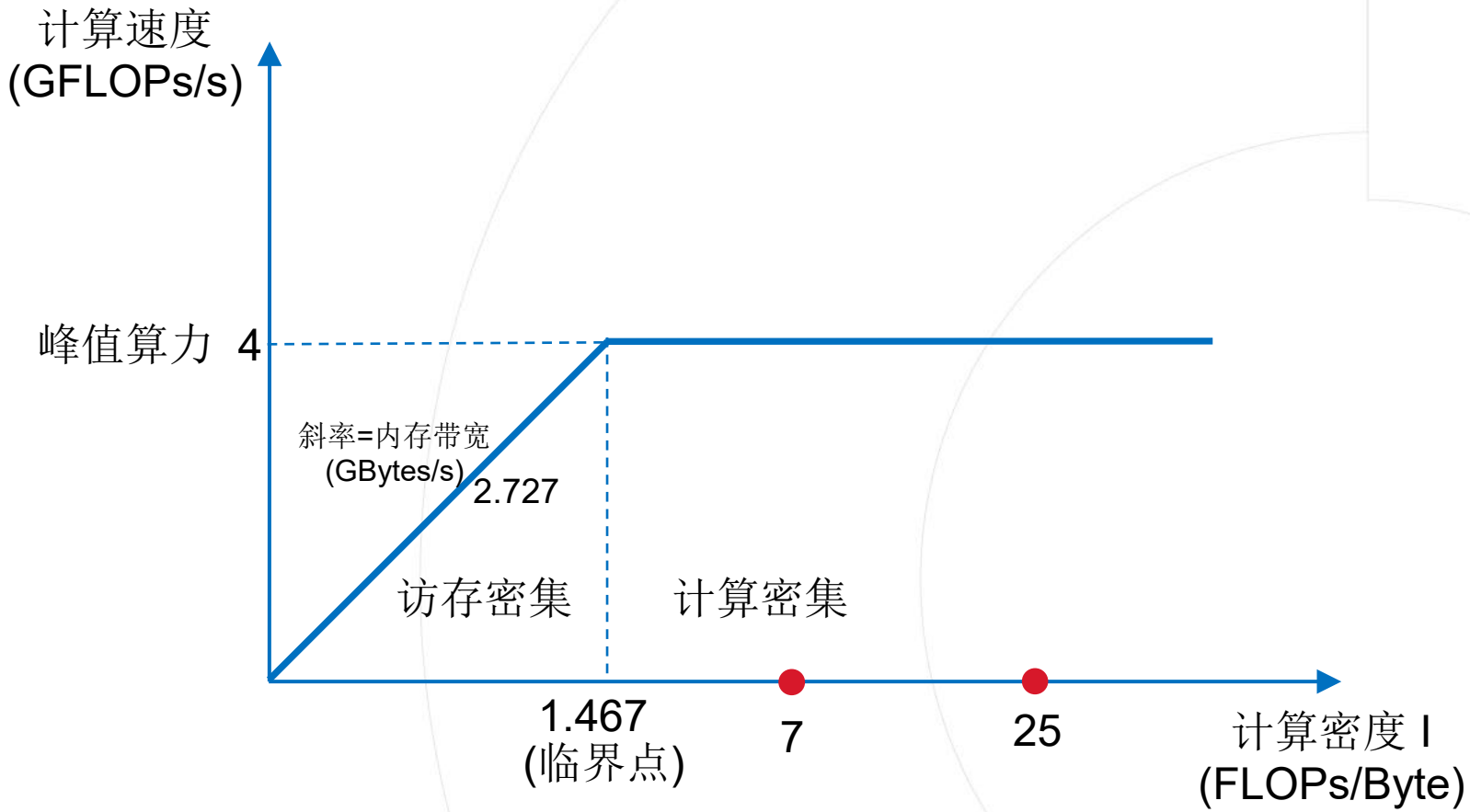
指令	指令吞吐(CPI)	执行延迟
vfadd/sub/mul.vv、vfadd/sub/mul.vf	1	3
vfmacc.vv、vfmacc.vf	2	4
vrgather.vi	2	3
vrgather.vi + vfmacc.vv	3	5
vle.v、vse.v	2	2

  - 访存带宽

    Cache： 8 GB/s

    Memory : 2.727 GB/s　(DDR3 792 MHz)

AllWinner D1 微架构特点

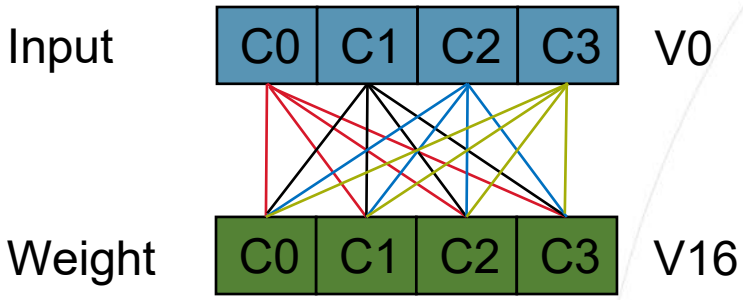


Ref. https://zhuanlan.zhihu.com/p/34204282 《模型大小与推理速度的那些事儿》

Outline

通用 GEMM KERNEL 的设计

采取 NBCX 的数据排布




```
vrgather.vi  v8,  v0,  0          vrgather.vi  v8,  v0,  0
vrgather.vi  v9,  v0,  1          vrgather.vi  v9,  v0,  1
vrgather.vi  v10, v0,  2          vfmacc.vv    v20, v16, v8
vrgather.vi  v11, v0,  3          vrgather.vi  v10, v0,  2
vfmacc.vv    v20, v16, v8         vfmacc.vv    v20, v16, v9
vfmacc.vv    v20, v16, v9         vrgather.vi  v11, v0,  3
vfmacc.vv    v20, v16, v10        vfmacc.vv    v20, v16, v10
vfmacc.vv    v20, v16, v11        vfmacc.vv    v20, v16, v11
```

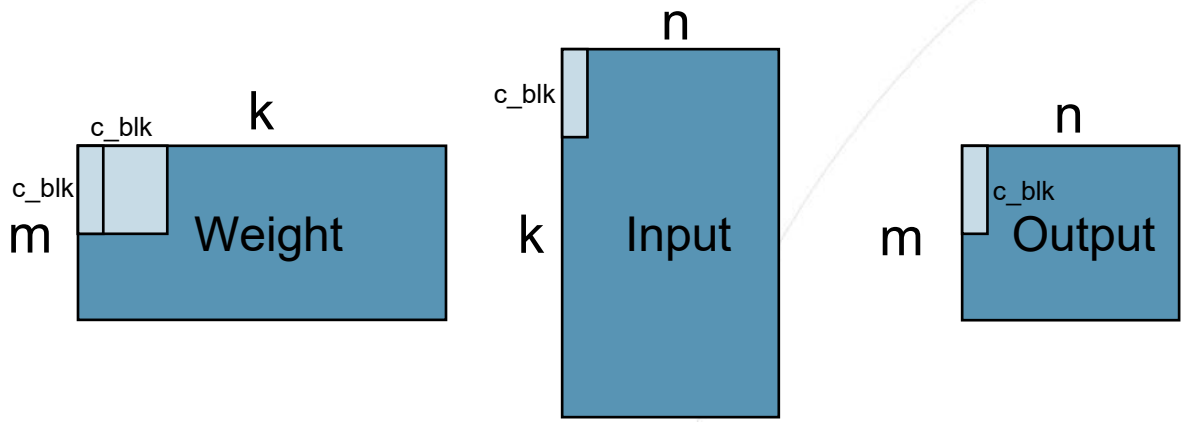
由于需要使用 gather 指令：

- 占用了向量寄存器
- 添加了与计算无关的寄存器间数据传递指令

左：2 * 4 + 2 * 4 = 16 (cycle)

右：2 + 3 * 3 + 2 = 13 (cycle)

通用 GEMM KERNEL 的设计



假设数据都在Cache中并且k足够大，在寄存器的约束下寻找最合适的kernel size (m/n)来达到最高的gemm_kernel性能。

- NumVregs = m / c_blk * c_blk + n + m / c_blk * n + c_blk <= 32
- m % c_blk = 0

$$\text{Utilization} = \frac{ComputeCycles}{ComputeCycles + LoadStoreCycles + VregsTransCycles}$$

- ComputeCycles = m * n * 2 * (k / c_blk)
- LoadStoreCycles = (m + n) * 2 * (k / c_blk) + m / c_blk * n * 2
- VregsTransCycles = c_blk * n * 2 * (k / c_blk)

weight/input : vle.v、vse.v
compute : vfmacc.vv

- NumVregs = m / c_blk * c_blk + m / c_blk * n <= 32
- m % c_blk = 0

$$\text{Utilization} = \frac{ComputeCycles}{ComputeCycles + LoadStoreCycles}$$

- ComputeCycles = m * n * 2 * (k / c_blk)
- LoadStoreCycles = (m * 2 + n * 1) * (k / c_blk) + m / c_blk * n * 2

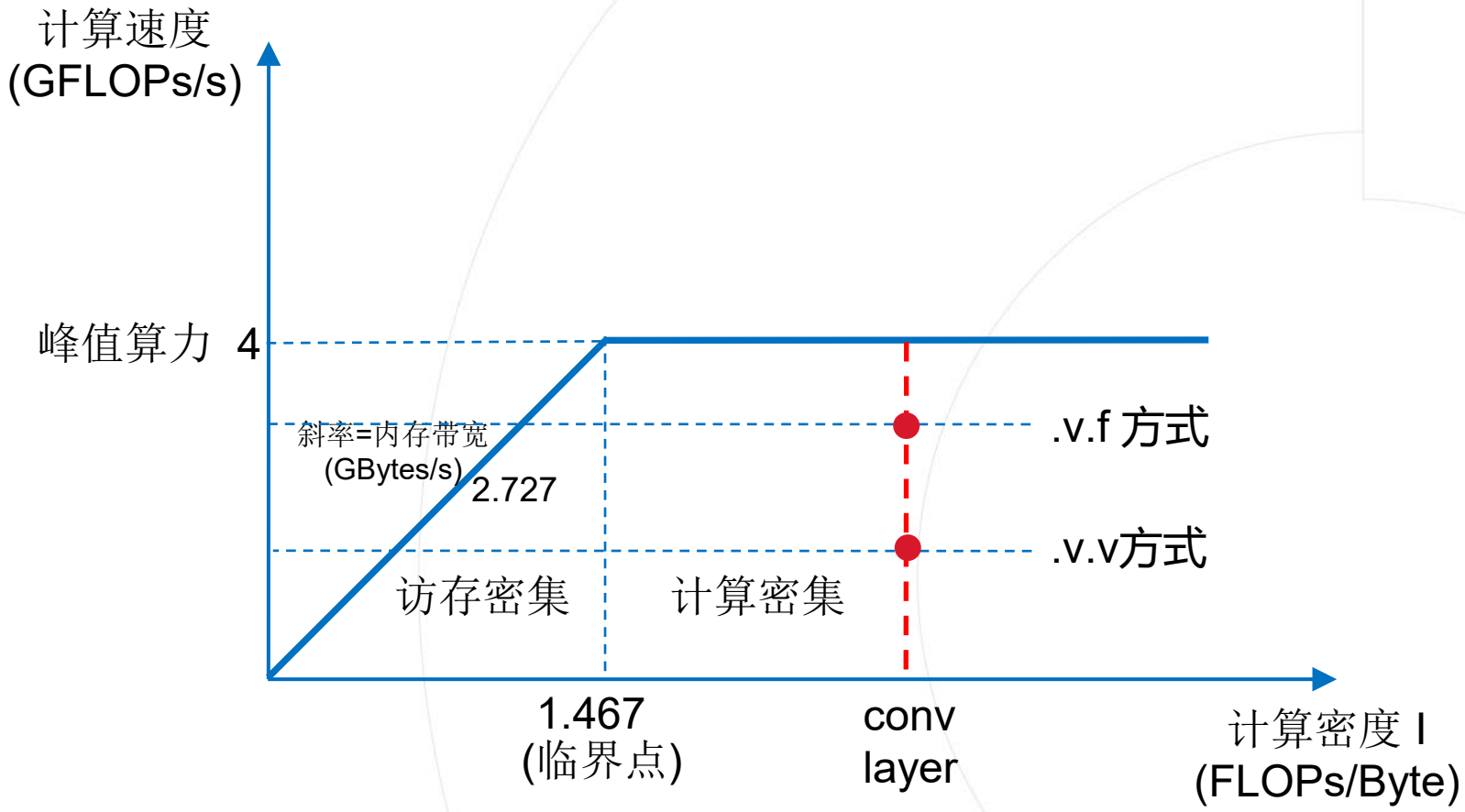
weight : vle.v、vse.v
input : flw
compute : vfmacc.vf

通用 GEMM KERNEL 的设计

```
m:8 n:1 uti:0.3199600049993751
m:8 n:2 uti:0.3808957000446362
m:8 n:3 uti:0.40671503468728487
m:8 n:4 uti:0.4209833908896563
m:8 n:5 uti:0.43003527633126154
m:8 n:6 uti:0.43628926887462366
m:8 n:7 uti:0.4408689448175757
m:8 n:8 uti:0.4443672973442111
m:16 n:1 uti:0.3901844231062338
m:16 n:2 uti:0.4847566748721833
```

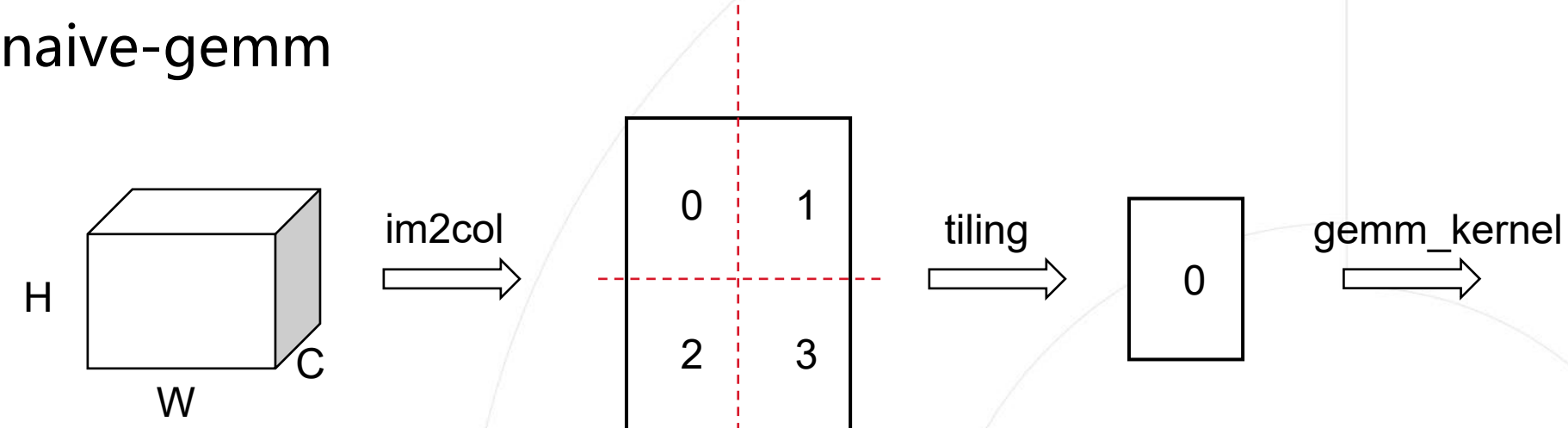
```
m:8 n:1 uti:0.4847566748721833
m:8 n:2 uti:0.6398400399900025
m:8 n:3 uti:0.7162174764524853
m:8 n:4 uti:0.7616780720023802
m:8 n:5 uti:0.7918342097123415
m:8 n:6 uti:0.8133008577782485
m:8 n:7 uti:0.829360855278382
m:8 n:8 uti:0.8418283459388359
m:8 n:9 uti:0.85178749676513
m:8 n:10 uti:0.8599261001007726
m:8 n:11 uti:0.8667015481210181
m:8 n:12 uti:0.872429853459048
m:8 n:13 uti:0.8773363562069966
m:8 n:14 uti:0.8815860677916072
m:8 n:15 uti:0.8853025936599423
m:8 n:16 uti:0.8885803540437348
m:8 n:17 uti:0.891492717700801
m:8 n:18 uti:0.8940975590826187
m:8 n:19 uti:0.8964411434047808
m:8 n:20 uti:0.8985608985608986
m:8 n:21 uti:0.9004874290212894
m:8 n:22 uti:0.9022460030117587
m:8 n:23 uti:0.9038576669788004
m:8 n:24 uti:0.9053400919486031
m:16 n:1 uti:0.4922130359546241
m:16 n:2 uti:0.6528946697271104
m:16 n:3 uti:0.7326147095297147
m:16 n:4 uti:0.7802499238037184
m:16 n:5 uti:0.8119251506501745
m:16 n:6 uti:0.8345104857111811
m:16 n:7 uti:0.8514277569249774
m:16 n:8 uti:0.8645727794663964
m:24 n:1 uti:0.4947497262127166
m:24 n:2 uti:0.6573654027219036
```

通用 GEMM KERNEL 的设计

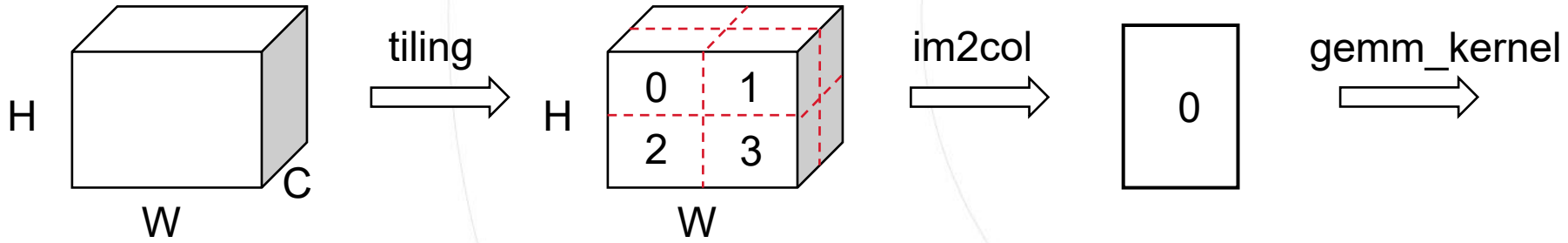


gemm 实现的两种选择

- naive-gemm



- tile-gemm



gemm 实现的两种选择

- naive-gemm
  - 需要开辟一块额外内存空间用来存放整张图的 im2col 结果
  - im2col 过程仅需一次
  - 由于Cache大小的局限性，无法合并 im2col 与 tiling 流程
  - 非 1x1 卷积核下，无额外 tiling 开销

- tile-gemm
  - 仅需一小块 tile 大小的内存空间来存放 tile 的 im2col 结果
  - 由于 Output Station 的循环计算策略，im2col 过程需要执行多次
  - 可合并 tiling 与 im2col 流程，并将结果直接送入 Cache 中
  - 非 1x1 卷积核下，在原图上 tiling 划分会有重叠数据开销

gemm 实现的两种选择

Models	tile-gemm only	add naive-gemm support
resnet18	2207.35 ms	1706.72 ms
resnet34	3564.58 ms	2881.52 ms
resnet50	9214.03 ms	6211.02 ms
resnet101	15989.94 ms	10440.48 ms
resnext50_32x4d	13359.08 ms	7759.88 ms

快速卷积算法-Winograd

- direct conv
  - Td = OC * OH * OW * IC * f * f * 2
- winograd conv

$$Y = A^T\left[\left[GgG^T\right] \odot \left[B^T dB\right]\right] A$$

  - $Tw = \frac{OH\_pad}{b} \times \frac{OW\_pad}{b} \times IC \times OC \times (b+f-1)^2 \times 2$ (点乘运算)

$$+(b+f-1)^3 \times IC \times \frac{OH\_pad}{b} \times \frac{OW\_pad}{b} \times 2 \times 2$$ (B^TdB)

$$+ b(b+f-1)^2 \times OC \times \frac{OH\_pad}{b} \times \frac{OW\_pad}{b} \times 2$$

$$+b^2(b+f-1) \times OC \times \frac{OH\_pad}{b} \times \frac{OW\_pad}{b} \times 2$$ (A^TMA)

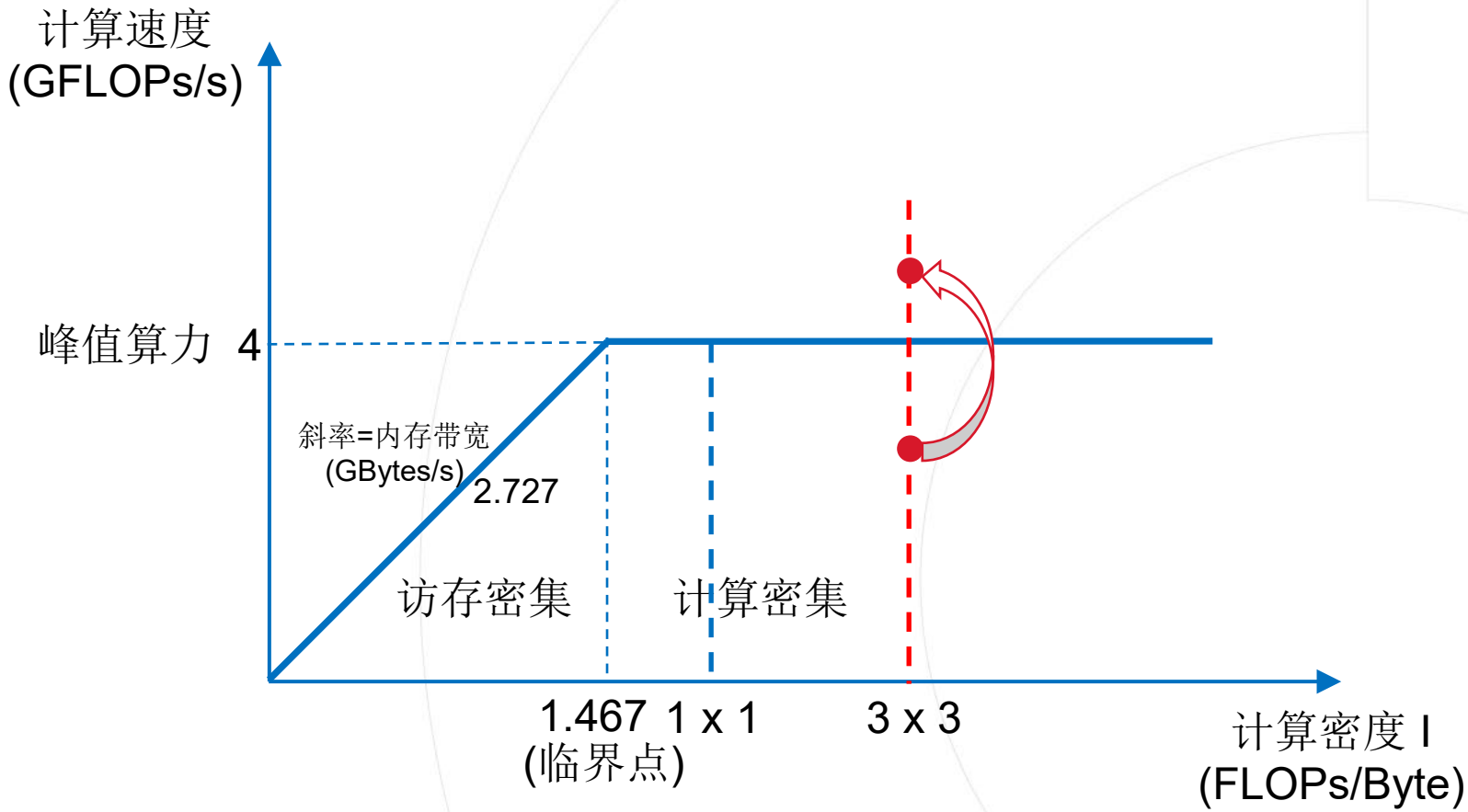
- 加速比
  - $\frac{Td}{Tw} \approx \frac{f^2}{\frac{(b+f-1)^2}{b^2}+\frac{2(b+f-1)^3}{b^2 \times OC}+\frac{(b+f-1)^2}{b \times IC}+\frac{b+f-1}{IC}}$

快速卷积算法-Winograd

- 加速比
  - $\frac{Td}{Tw} \approx \frac{f^2}{\frac{(b+f-1)^2}{b^2}+\frac{2(b+f-1)^3}{b^2\times OC}+\frac{(b+f-1)^2}{b\times IC}+\frac{b+f-1}{IC}}$
- 分析
  - 随着 IC/OC 的增大，$T_B{}^T{}_{dB}$ / $T_A{}^T{}_{MA}$ 两者减小，加速效益更明显
  - 以上 IC/OC 较大的前提下，$\frac{Td}{Tw} \approx \frac{1}{(\frac{1}{f}+\frac{1}{b}-\frac{1}{bf})^2}$
    - f 固定，随着 b 的增大，加速效益更明显
    - b 固定，随着 f 的增大，加速效益更明显
      - f 为 1 时，加速比为 1

winograd	理论加速比
b2f3	9/4=2.25
b4f3	4/1=4
b6f3	81/16=5.06

快速卷积算法-Winograd



减少了实际计算复杂度

⇩

相同算力下减少了计算时间

⇩

在传统卷积计算复杂度下，
变相提升了计算速度

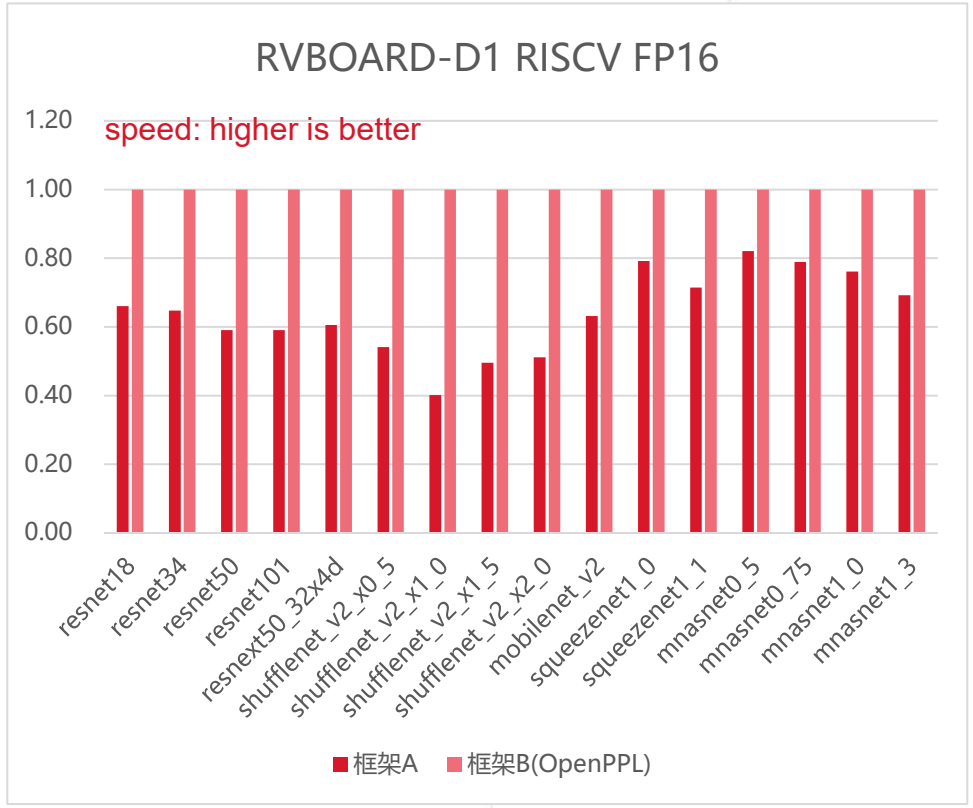
Outline

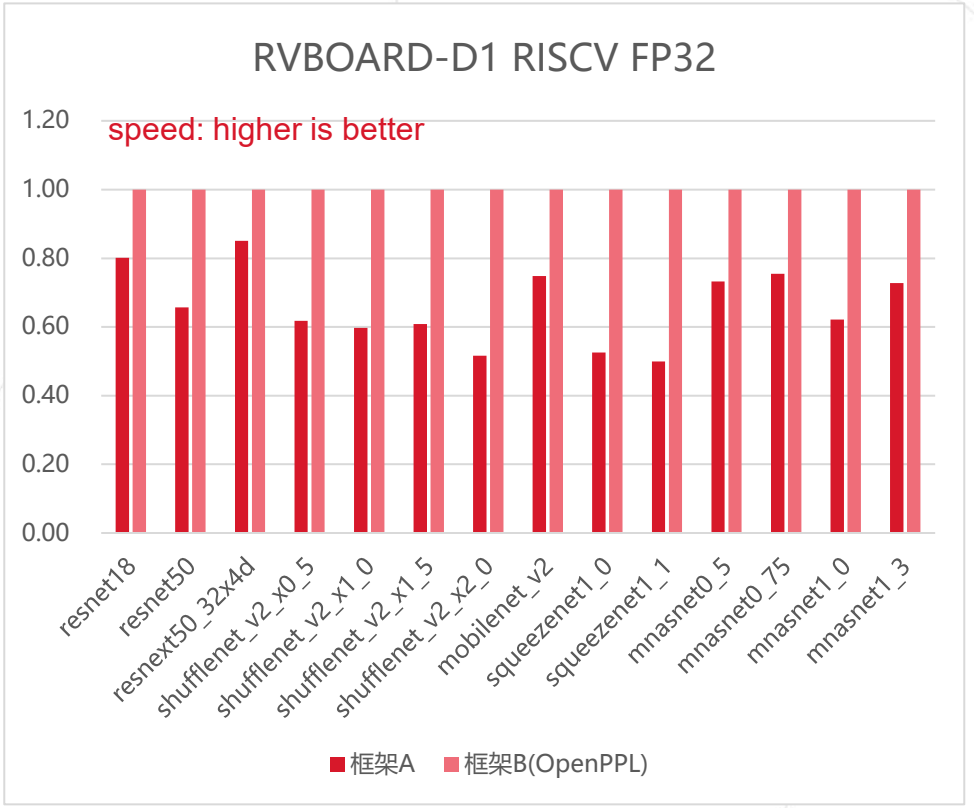
OpenPPL RISCV 使用方式

- fp32 测试
  - ./pplnn --use-riscv -- onnx-model model.onnx --inputs input.bin
- fp16 测试
  - ./pplnn --use-riscv --use-fp16 --onnx-model model.onnx --inputs input.bin
- 其他命令设置项
  - --help
  - --enable-profiling
  - --save-inputs
  - --save-outputs
  - --min-profiling-iterations
  - --wg-level

Ref. https://github.com/openppl-public/ppl.nn/blob/master/docs/en/riscv-doc/benchmark_tool.md

性能结果





OpenPPL在 fp16 和 fp32 精度上均能够达到平均 1.5 倍于框架A的性能

OpenPPL RISCV 开发计划与展望

- 支持更多的网络模型和算子
  - 语义分割、目标检测等网络
  - 视觉 CV 算子
- 支持
  - 兼容 RVV 0.7.1 和 RVV 1.0 标准版本
  - 可变长 VLEN 支持

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