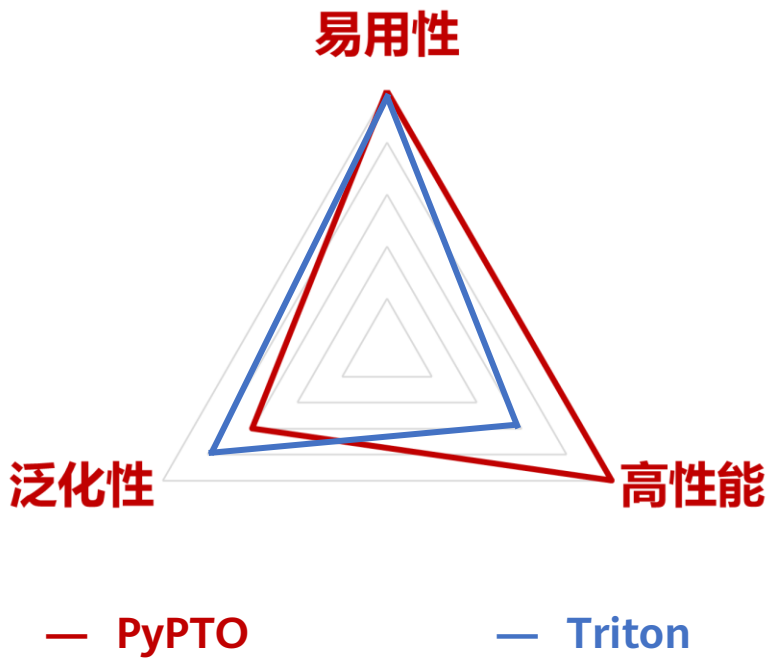
PyPTO：
简单易用、高性能、跨代兼容的编程范式

王子楠 CANN PyPTO技术专家

CANN

目录

PyPTO(Python Parallel Tile Operation)：聚焦易用性、高性能



- **易用性：降低开发门槛（好开发），简化开发流程（好使用）**
  - C++ → Python 语言编程
  - Tile DSL 的编程
  - 去除算子交付件，端到端调试最优
  - Debug/DFX/可视化
- **高性能：**
  - Kernel launch 开销、算子间 GM 访问 → 0
  - 更大范围的融合：SPMD → MPMD
- **跨代兼容**
  - 支持同一算子的多平台复用，具有可移植性

PyPTO—基于 Tile DSL 的编程，降低开发门槛

前端实现

```python
def softmax_compute (input_tensor, output_tensor):
    row_max = pto.row_max_single(input_tensor)
    sub = pto.sub(input_tensor, row_max) # input_tensor - row_max
    exp = pto.exp(sub)                   # sub.exp()
    esum = pto.row_sum_single(exp)  # exp.row_sum_single()
    output_tensor[:] = pto.div(exp, esum)     # exp / esum
```

根据算子数学表达式及计算逻辑，实现算子函数代码（硬件无关）

```python
def dynamic_softmax(input_tensor, output_tensor):
    tensor_shape = input_tensor.shape
    b = pto.symbolic_scalar(tensor_shape[0]) #动态Batch
    n1, n2, dim = tensor_shape[1:]
    tile_b = pto.symbolic_scalar(1) #静态TileBatch
    b_loop = b / tile_b
    with pto.function("SOFTMAX", [input_tensor], [output_tensor]):
        #配置for循环表达动态shape
        for idx in pto.loop(0, b_loop, 1, name="LOOP_L0_bIdx", idx_name="idx"):
            b_offset = idx * tile_b
            input_view = pto.view(input_tensor, [tile_b, n1, n2, dim], [b_offset, 0, 0, 0])

            output_tile = pto.tensor()
            pto.set_vec_tile_shapes(1, 4, 1, 64) #配置vector计算的tiling配置
            softmax_compute(input_view, output_tile)

            pto.assemble(output_tile, [b_offset, 0, 0, 0], output_tensor)
```

控制流 for if else

Machine 承载

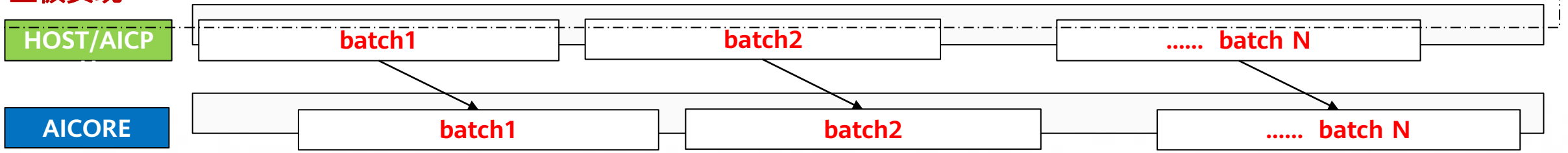
AICPU HOST CPU

计算流 +−x÷

PASS 承载

AICORE 执行

上板实现



CANN

PyPTO–基于 jit kernel 整网接入，简化算子开发流程

PyPTO 代码实现展示

```python
if not use_grouped_topk and custom_routing_function is None:
    bs, ne = router_logits.shape
    device_id = router_logits.device

    topk_weights = torch.zeros((bs, top_k), dtype=router_logits.dtype, device=device_id)
    topk_ids = torch.zeros((bs, top_k), dtype=torch.int32, device=device_id)
    inputs = [router_logits]
    outputs = [topk_ids, topk_weights]

    #将原实现torch_npu.softmax函数调用替换为pto.dynamic_softmax即可
    @pto.jit
    pto.dynamic_softmax(inputs, outputs)

    pto.device_synchronize()
    return topk_weights, topk_ids
```

Pytorch 的脚本与数据结构

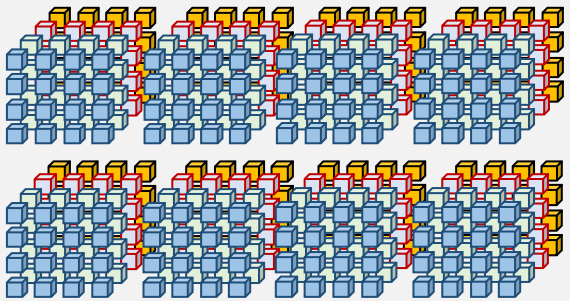
通过 Jit 的方式完成算子的整网接入：

- 简化算子开发流程：
- 去掉算子的所有交付件
- 去掉 **aclnn**
- 去掉 **GE** 入图
- 去掉 **FE** 的融合规则

CANN

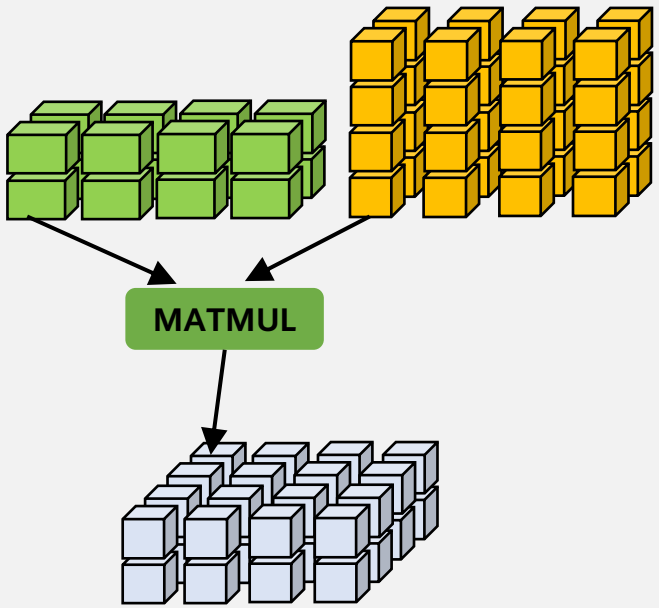
目录

Tile = smaller Tensor



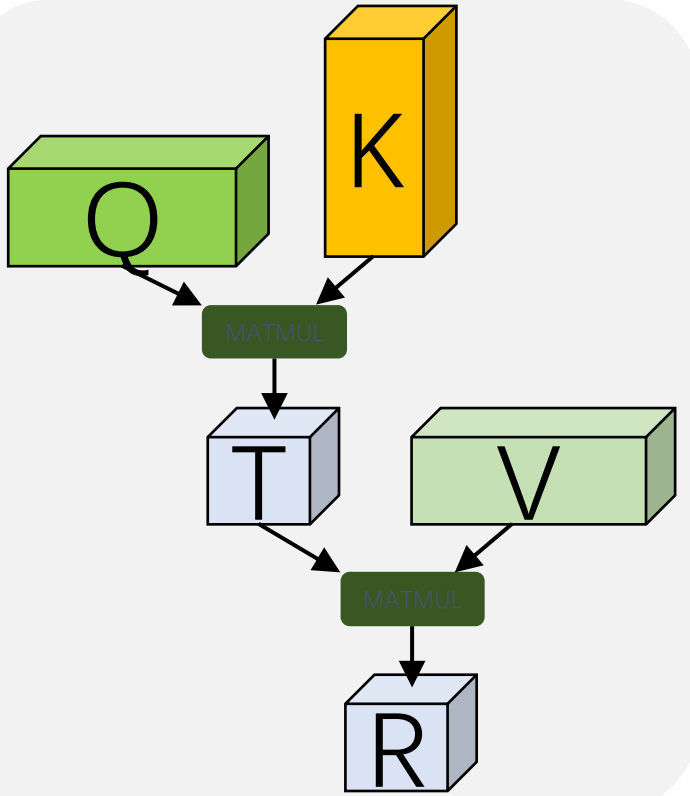
Scalar Operation（32bit）

- 灵活性高
- 控制复杂度高，硬件实现复杂
- 计算冗余，效率不高



Tile Operation（16KB）

- 适配硬件存储层级
- 刚好适配核内Buffer存储
- 抽象简单 控制适中

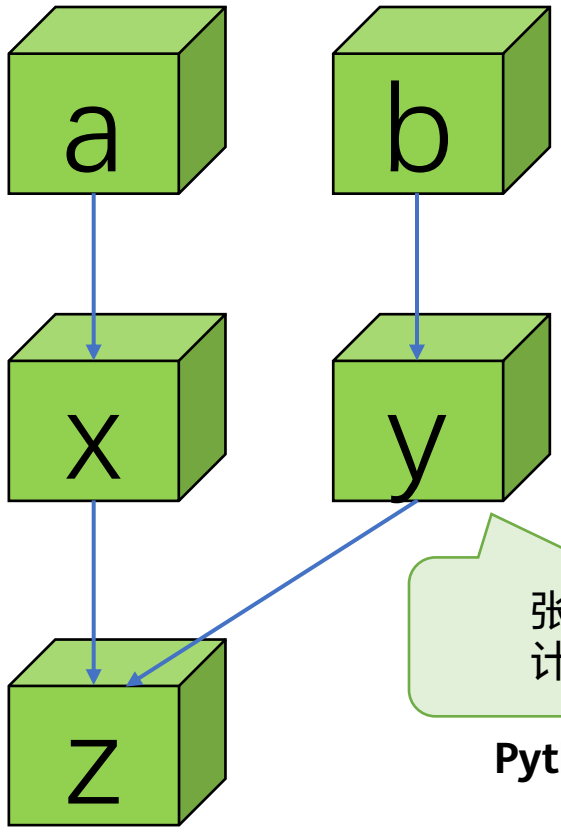


Tensor Operation（ 256MB ）

- 表达简单，python编程
- 数据量过大，塞不进芯片

CANN

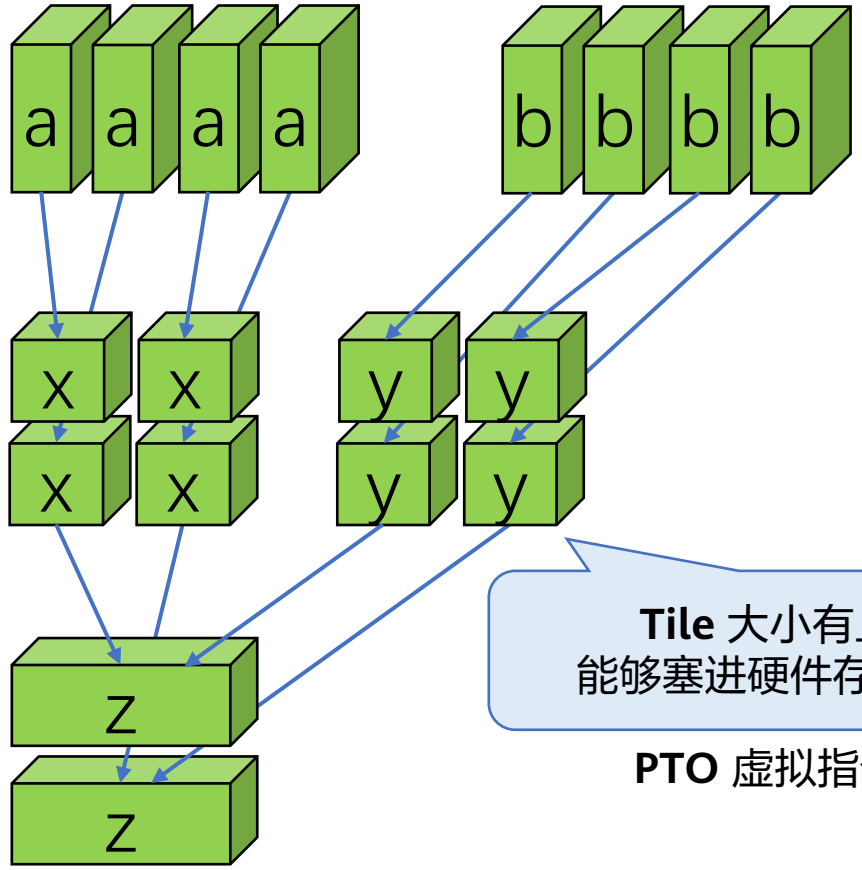
Tensor 运算 → Tile 运算



张量逻辑
计算表达

Python 前端

Tensor Operation



Tile 大小有上限
能够塞进硬件存储层级

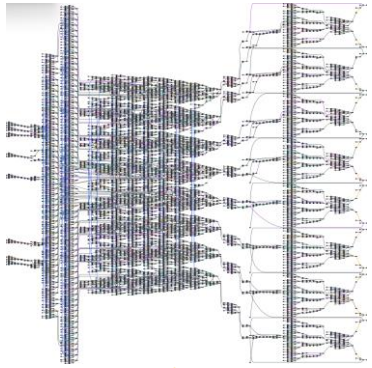
PTO 虚拟指令集

Tile Operation

CANN

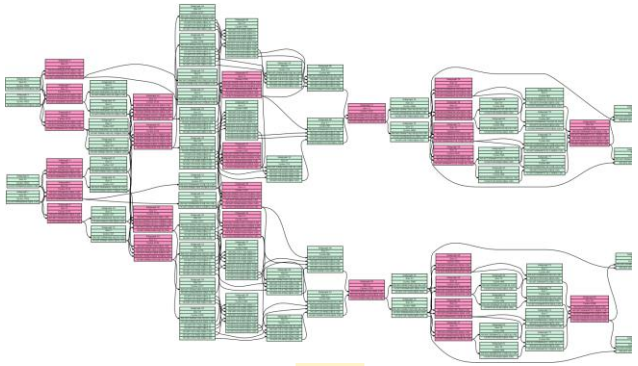
PyPTO 性能调优： Human-In-The-Loop

PyPTO 的整图依赖关系图

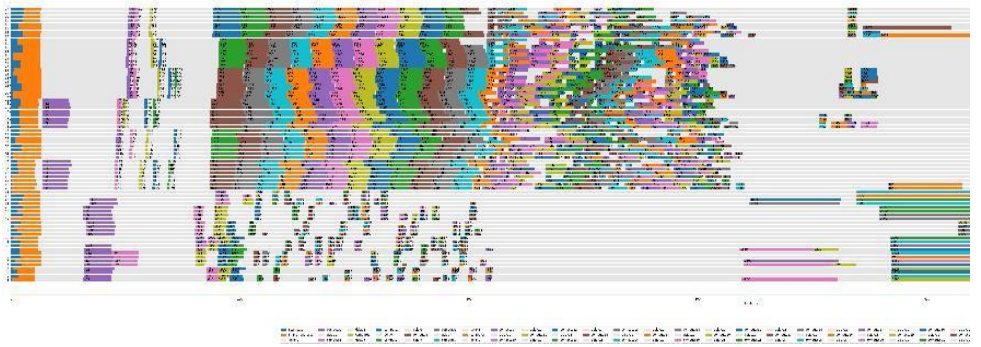


整图切分成子图

PyPTO 的函数依赖关系图



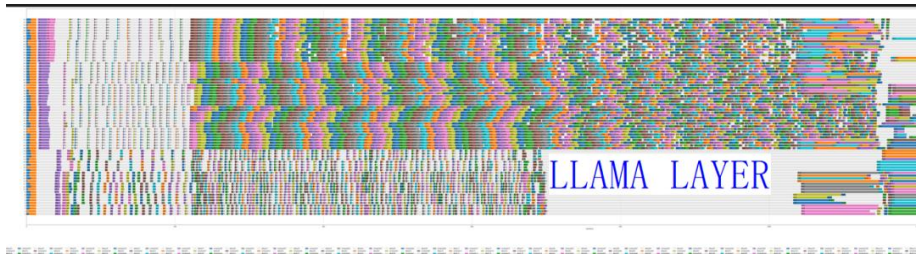
将多子图下发到多核上执行



获取真实的上板执行泳道图

Profiling 可视化

调整调度算法



从泳道图上识别不合理性能点

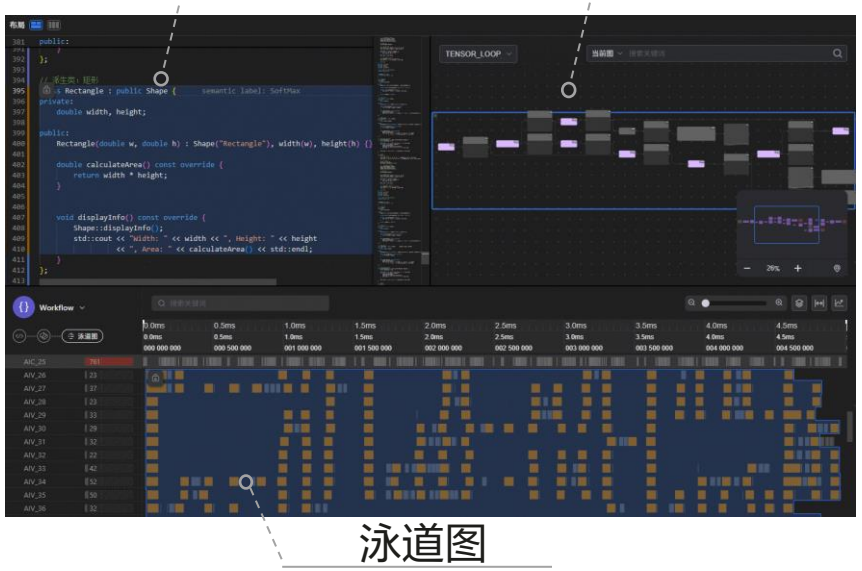
调整切图算法
调整每个层级 tensor/tile 块的颗粒度

CANN

PyPTO 优化方法：Human-In-The-Loop 调整 Tile 粒度

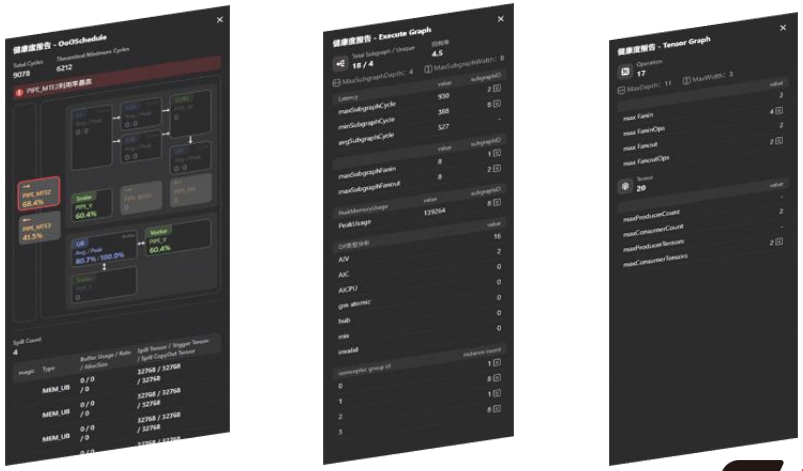
IDE：

- 提供 **Tile** 编程框架全流程辅助工具，包括编译、运行时状态的可视、**Tile** 编程框架工作流的作业能力
- 打通算子代码-算子计算流程图-算子执行泳道图之间实时联动
- 使能基于 Tile 计算图的精度调试的能力，提供基于 Pass 阶段、Tensor、OP 多种粒度的立体化精度调试能力
- 将专家经验固化成 IDE 专家系统能力，基于编译、运行时的数据，提供性能瓶颈诊断和性能调优建议



DFX：

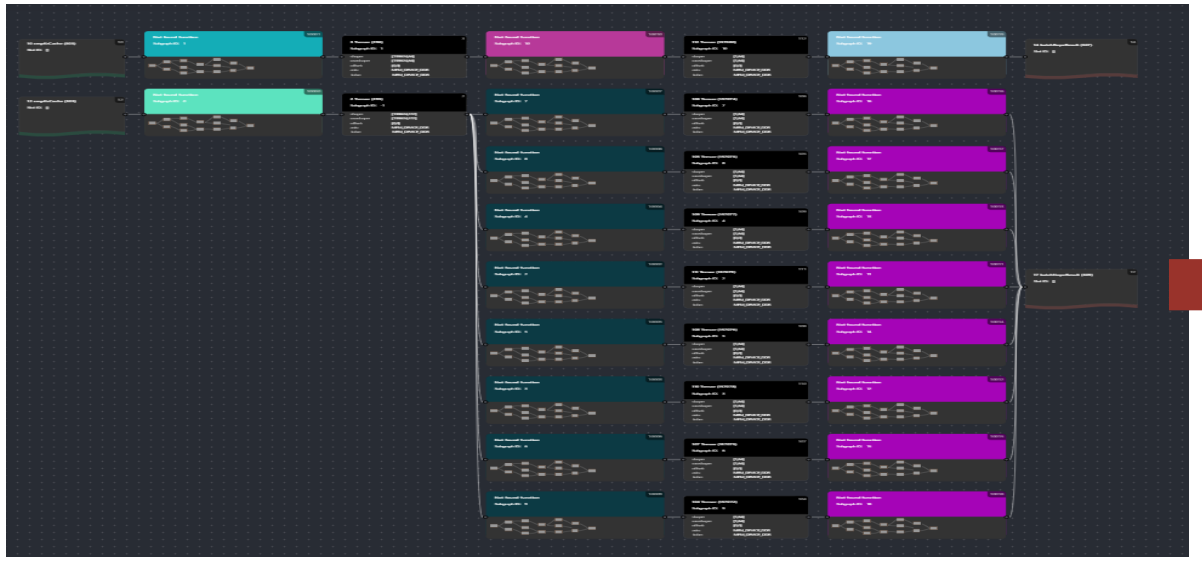
- 内置丰富的检查，并在异常时打印
- 健康报告：对关键指标进行分析和展示
- 精度工具
- 性能优化手段：
  - 融合算子
  - **MPMD**
  - **Tileshape**（内存 **vs** 并行 **tradeoff**）
  - 自定义 **tiling** 方法
  - 用户可以完整控制切分，执行过程和逻辑（切图，大页，**L2Prefetch**等）

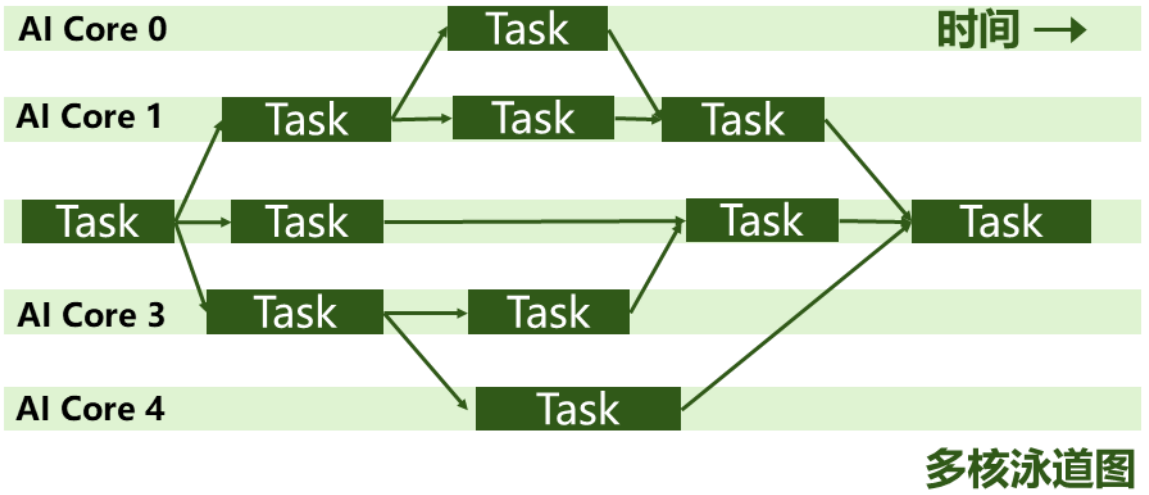


CANN

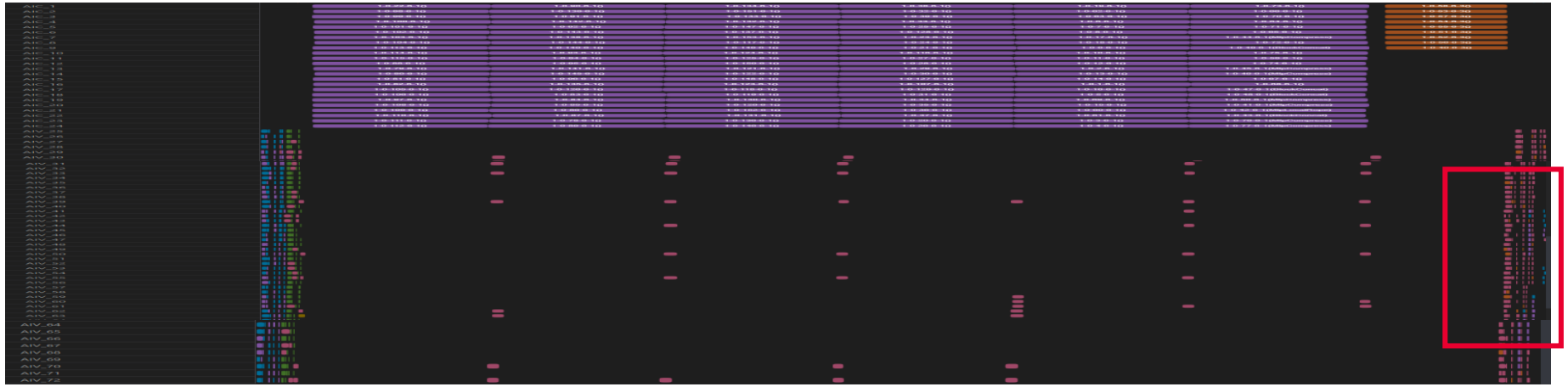
PyPTO 任务调度：MPMD 的调度实现

- 通过数据形成任务之间的依赖，可以通过拓扑序解依赖





- 采集上板数据，形成泳道图；关联前端计算逻辑



不同颜色是不同子图
可以看到MPMD调度

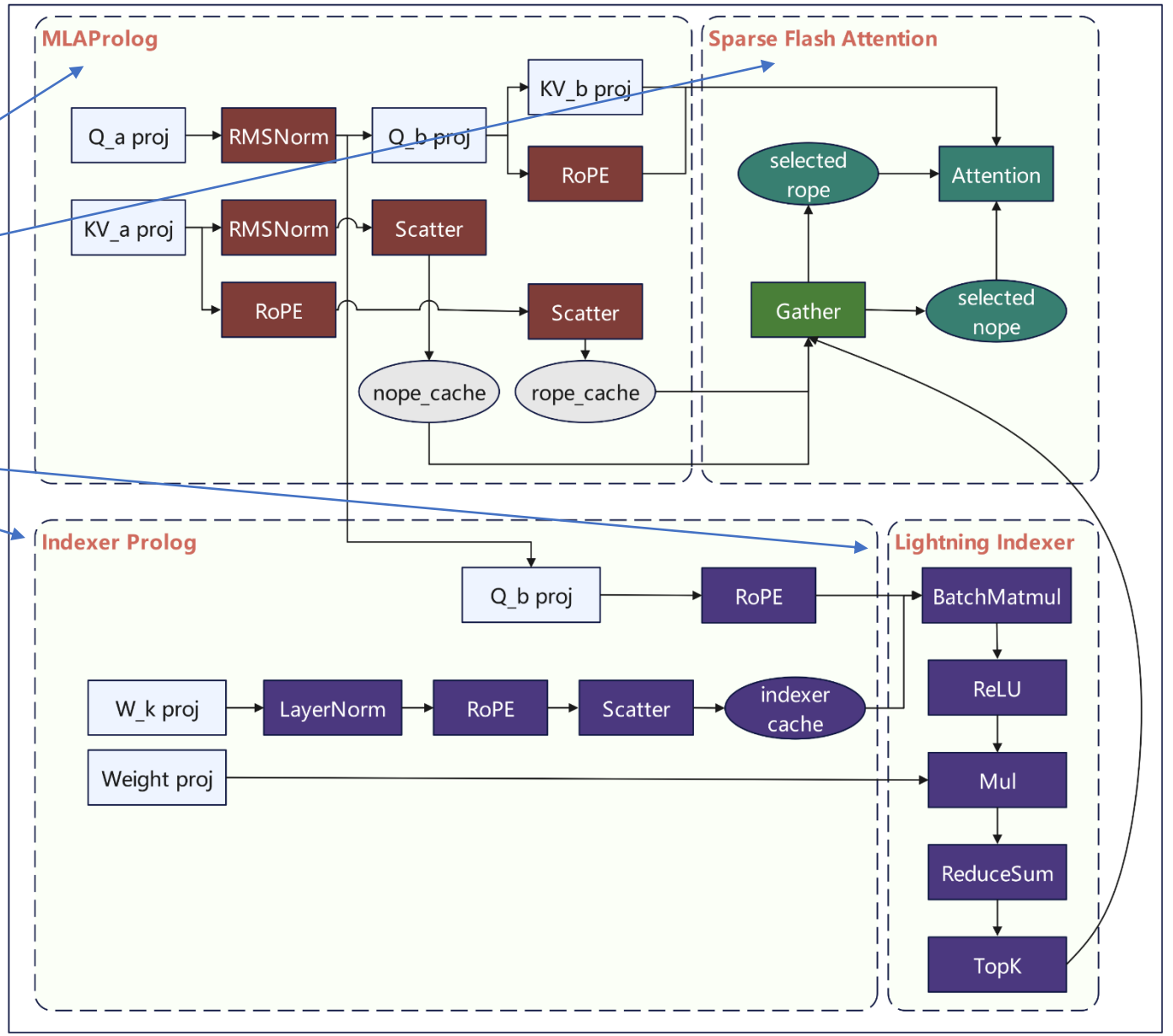
注：
- ✓ 算子：NSA kv cmp
- ✓ VSCode IDE v0.1.2

CANN

目录

DeepSeek V3.2-Exp 开发：自主控制融合范围

将 **Attention** 层拆解为 **4** 个子图并行开发；
通过框架融合成一个算子



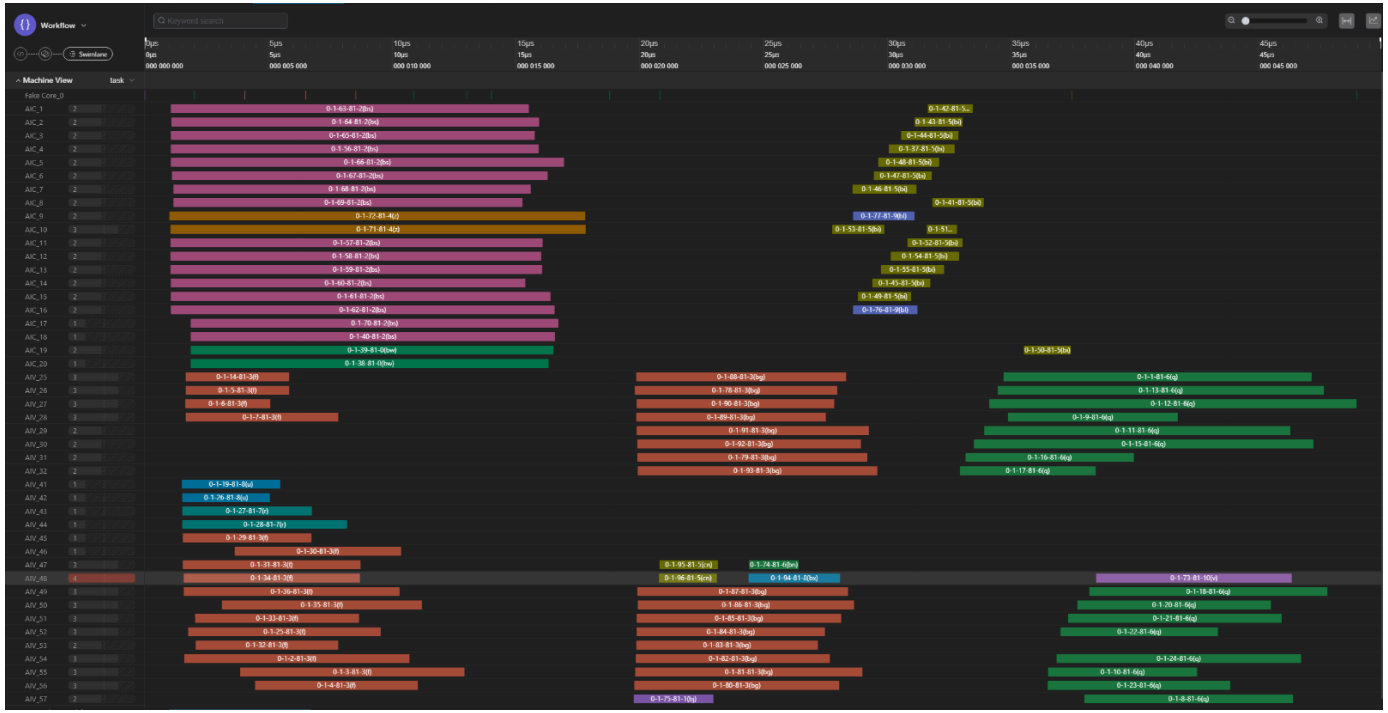
CANN

Indexer Prolog 调优实例

类别	Indexer Prolog 算子组成
PyPTO	1 / 个
	IndexerProlog
小算子	9 / 个
	QuantBatchMatmul, SplitVD, MatMul, InterleaveRope, ConcatV2D, Cast, LayerNorm, DynamicQuant, ScatterNdUpdate

计算流呈现如下特点：

- **Q, Cache 和 Weight** 计算互相独立且串行；
- **Q** 耗时最长，可以掩盖其他两部分耗时



Indexer Prolog 逼近理论极致性能

Q 计算阶段	Bound 类型	Bound 数	理论极致性能	PyPTO 实际性能
Linear	**MTE2**	**12MB**	**12/1.2/0.75=13.3**	**16us**
Dequant+RoPE	**MTE2**	**/**	**9.6**	**10us**
Hadamard	**Fixpipe**	**/**	**4**	**4.5us**
DynamicQuant	**Vector**	**21000cycle**	**21000/1800/0.66=17.7**	**18us**

- Linear 计算阶段有 **cache**，**weights** 计算抢占 **MTE2** 带宽，故 **Indexer Prolog** 算子的实际表现会比 Q 的计算理论极致稍差；
- 在 **Batch=4, MTP1** 的典型场景下，**Query** 的理论极致性能是 **45us**，**PyPTO** 达到 **48.5us**，已经逼近理论极致性能；
- **Hadamard Transform** 采用 **matmul** 实现，实际性能受到调度影响
- 一套 **tile** 切分，在各类场景（不同 **Batch/MTP** 的组合），平均优于小算子 1 倍性能；

PyPTO-Next

PyPTO

- **DeepSeek V3.2-Exp、Qwen3-235b、Qwen3-next 等多个大模型的 PyPTO 算子实现**
- 用户根据已有的 **Operation** 自主编写算子、模型，尝试更多想法
- 用户扩充 **TileOP**，实现社区共创

CANN

欢迎开发者体验和贡献

cann-recipes-infer

cann-recipes-train

cann-recipes-sig小组

cann-recipes交流群

特性介绍：
https://gitcode.com/cann/cann-recipes-infer/blob/master/docs/models/deepseek-v3.2-exp/deepseek_v3.2_exp_pypto_operator_guide.md

算子实例：
https://gitcode.com/cann/cann-recipes-infer/tree/master/ops/pypto

尝鲜体验：
https://zhuanlan.zhihu.com/p/1966608093946320154

欢迎广大开发者体验并参与贡献，如有疑问也可通过issue、SIG或者cann-recipes交流群联系我们！