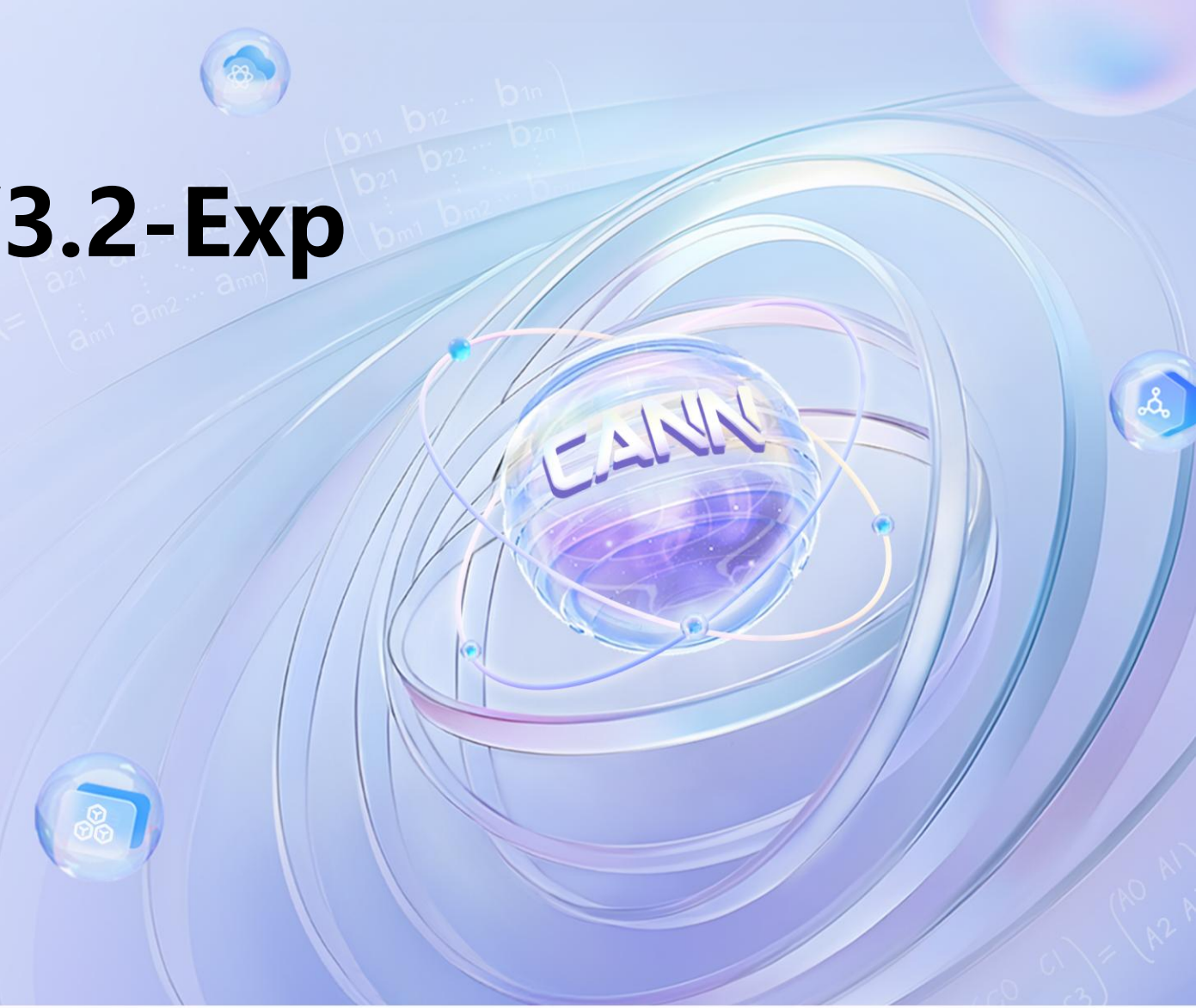
NPU DeepSeek-V3.2-Exp 推理优化实践



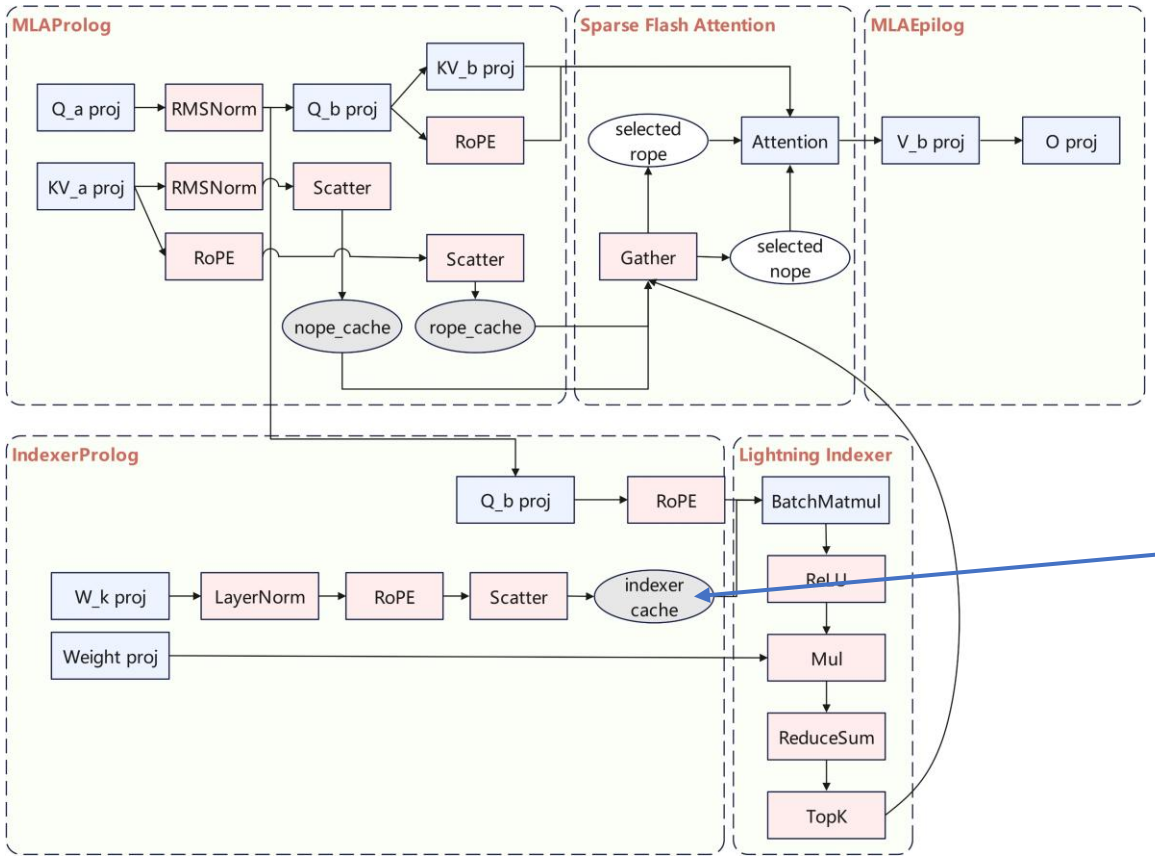
https://gitcode.com/cann

CANN

目录

DeepSeek V3.2 Exp模型解析—结构&内存

相较DeepSeek-V3.1，DeepSeek-V3.2-Exp主要新增了**Lightning Indexer**稀疏模块，旨在从长序列中稀疏选择TopK个Token用于计算注意力



https://gitcode.com/cann

- **权重内存**

相较DeepSeek-V3.1，主要新增了Indexer模块中的Q_b proj、W_k proj和Weight proj三个Linear层，参数量新增0.85B左右，较为轻量

- **KVCache内存**

为了保障模型效果，DeepSeek-V3.2-Exp仍然缓存了完整MLA的Full KVCache。与此同时，为了提高推理效率，避免decode阶段重复计算Indexer Key，需要**新增缓存Indexer Key Cache**

以每个rank处理4batch 64K序列长度为例，BF16场景新增的Indexer Key Cache约为4GB左右，W8A8场景约2GB

- **计算量**

相较DeepSeek-V3.1，

CANN

DeepSeek V3.2 Exp模型解析—Prefill MLA计算流

方案一： MLA Naive + Sparse Mask

方案一	Batch	M	K	N
BMM1(Q*K^T)	1*128	64K	192	64K
BMM2(P*V)	1*128	64K	64K	128

性能评估：方案一计算量和原始的Full Attention一致，但是无法拿到DSA的稀疏计算收益，长序列场景下性能不佳。

方案二： MLA Naive + Sparse Attention

方案二	Batch	M	K	N
BMM1(Q*K^T)	1*64K*128	1	192	2048
BMM2(P*V)	1*64K*128	1	2048	128

性能评估：方案二的优点在于BMM的计算量较小，相对原始的Full Attention计算量减小了64K/2048=32倍，但是存在以下问题：

- BMM的M轴为1，矩阵乘法计算效率较低
- BMM的HBM访存量相较原始的Full Attention激增2048倍，将会面临访存瓶颈

方案三： MLA Absorb + Sparse Attention

方案三	Batch	M	K	N
BMM1(Q*K^T)	1*64K	128	576	2048
BMM2(P*V)	1*64K	128	2048	512

性能评估：

- 方案三的计算量增加了3倍左右，但其BMM的M轴为128，对于矩阵乘法更为友好
- 方案三的HBM访存量相对方案二降低几十倍，访存耗时更低

结论：

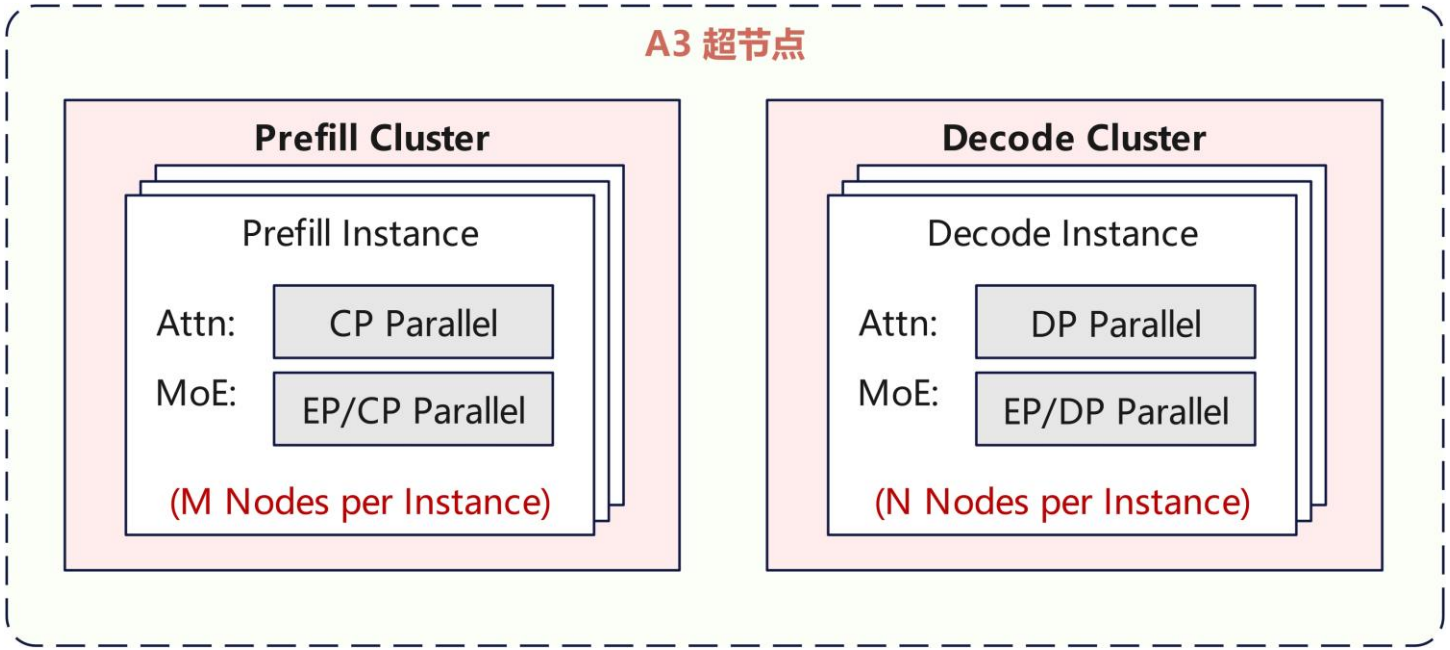
综合考虑计算和访存耗时，以及长序列应用场景，本实践选择基于方案三(MLA Absorb + Sparse Attention)来完成Prefill部署，从而Prefill和Decode的MLA计算流可以归一。

目录

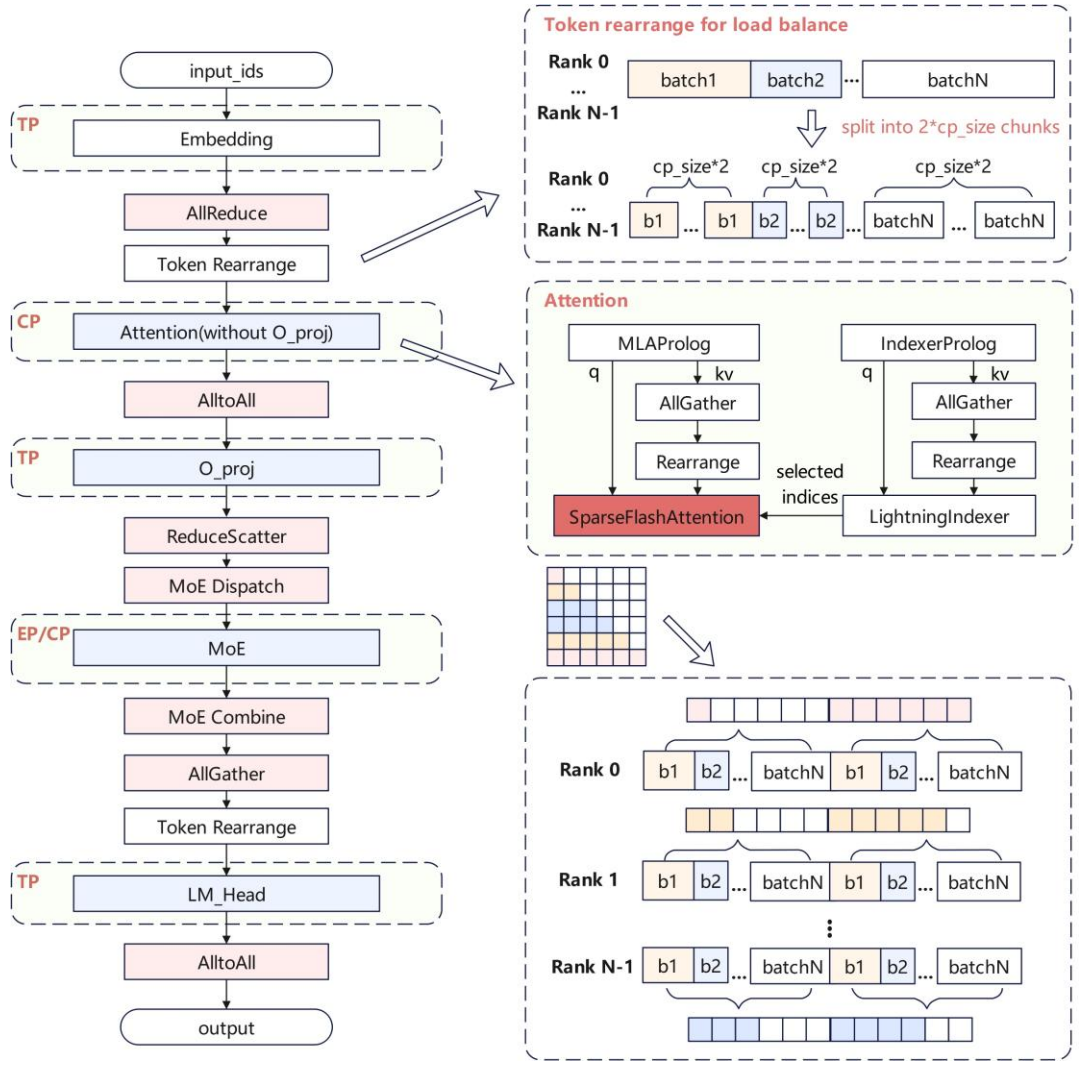
基于A3集群的整网优化方案解析—并行策略

Atlas A3推荐部署策略如下图所示，Prefill使用M个节点部署，Decode使用N个节点部署，每个节点包含8卡。

推荐根据资源数量、SLA等约束， BF16场景 M在4~8，N在4~16内动态调整；W8A8场景 M在2~8，N在2~16内动态调整。



基于A3集群的整网优化方案解析—Prefill并行策略



https://gitcode.com/cann

Q：为什么Attention模块不使用Data Parallel?

A：DP每个rank都需要推理不同的超长序列，总计算量和内存较大，TTFT较高，用户体验较差

Q：为什么Attention模块不使用Tensor Parallel?

A：DSA的Indexer模块针对head轴做了Reduce Sum操作，TP切分会产生大量的通信开销，影响整网性能；并且Prefill MLA使用了Absorb形式，kv head num为1，不适合TP切分

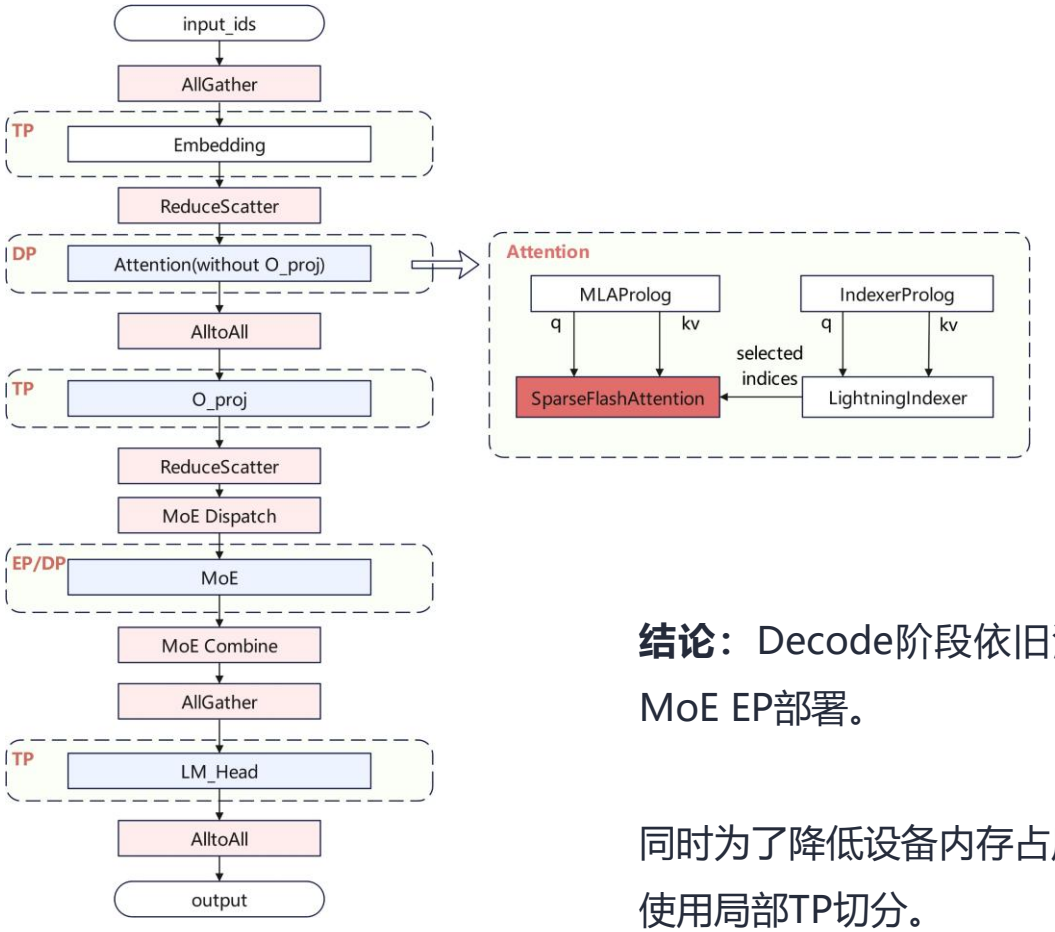
Q：为什么Attention模块选用Context Parallel?

A：CP可以让多个rank均摊长序列请求的计算，单rank的计算量和activation内存都较小，TTFT较为可控，用户体验更好；但需要注意不同rank的负载均衡处理

结 论： DeepSeek-V3.2-Exp 模 型 Prefill Attention 选 用 Context Parallel(CP)并行，MoE模块则沿用DeepSeek-V3.1的EP并行，兼顾吞吐与时延。

CANN

基于A3集群的整网优化方案解析—Decode并行策略



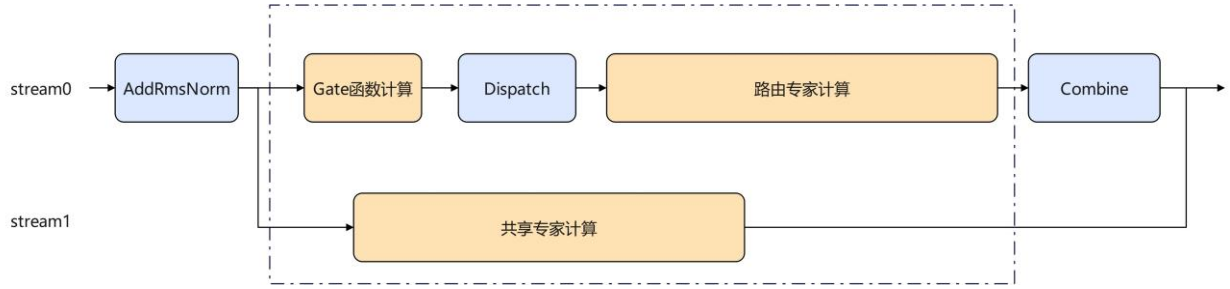
结论：Decode阶段依旧沿用DeepSeek-V3.1的部署策略，选用Attention DP + MoE EP部署。

同时为了降低设备内存占用，缓解权重访存瓶颈， O_proj/LM_Head/Embedding使用局部TP切分。

DeepSeek V3.2 Exp NPU推理优化介绍—多流并行

CANN提供了多流并行的机制，可支持计算/通信并行等。由于A3芯片为CV分离架构，也可支持Cube/Vector并行。MLAProlog/IndexerProlog/MOE模块均可使用多流并行加速

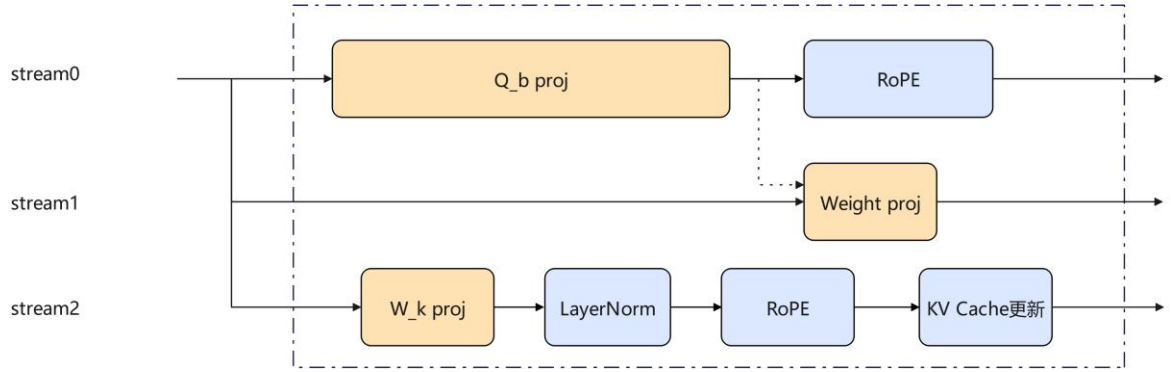
MOE多流并行



简单代码示例：

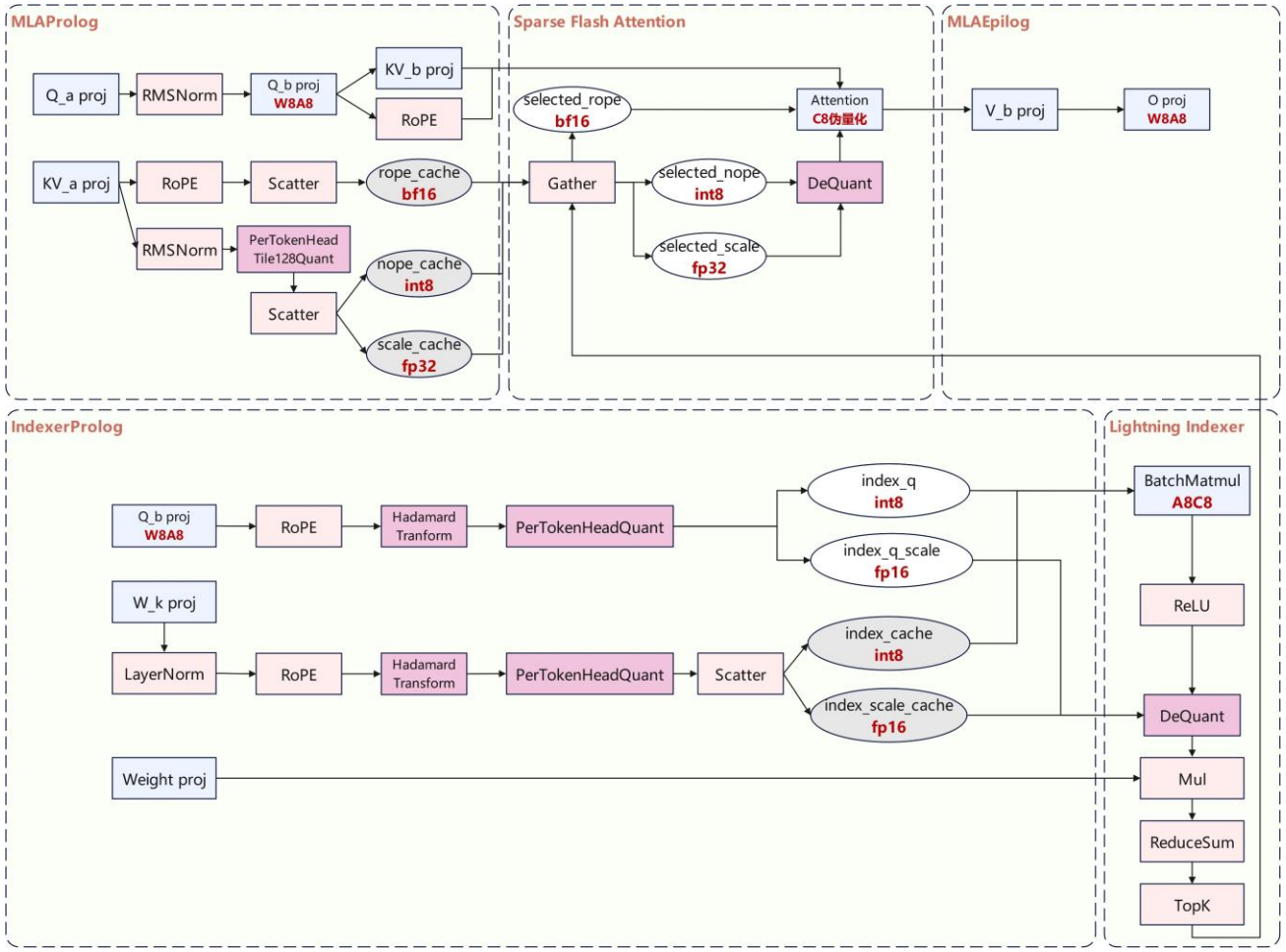
```
with torchair.scope.npu_stream_switch(stream_tag, stream_priority):
    # shared_expert use another stream
    hidden_states_share = self.shared_experts(hidden_states)
# router_experts use main stream
hidden_states_router = self.router_experts(hidden_states)
hidden_states = hidden_states_share + hidden_states_router
```

IndexerProlog多流并行



目录

整网量化方案



- **MLAProlog**：除q_b_proj和w_o_proj量化到W8A8，其他Linear均不量化；KVCache量化到C8
- **Sparse Flash Attention**：存8算16
- **IndexerProlog**：q_b_proj量化到W8A8，其他Linear均不量化
- **Lightning Indexer**：BatchMatmtul量化到A8C8，KCache量化到C8
- **MoE**：路由专家和共享专家GroupedMatmul量化到W8A8

Future Plan

- 量化：未来进一步开发低比特量化版本，探索KVCache量化压缩算法，软硬协同优化NPU计算效率，降低系统时延
- KVCache Offload：长序列场景会面临KVCache的内存瓶颈，可参考Shadow KV和Infinite LLM，达到类似的KVCache Offload效果
- MegaKernel：Decode阶段仍然存在较多流水并行空间，可通过PyPTO实现更大范围的MegaKernel，完成多核MPMD并行调度，提升计算效率

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem

Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货

关注CANN公众号获取资讯

https://gitcode.com/cann