

PR #50656 完整报告

vllm-project/vllm

[Kimi-K3] Add option to shard the shared expert instead of replicating

合并时间: 2026-08-03 23:41

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/50656>

执行摘要

- 一句话: Kimi-K3 共享专家可选 TP 分片, decode 省显存提速
- 推荐动作: 值得精读。三个看点: 其一, 实现上“reduce-scatter 双职责”(跨 TP 求和 + 恢复序列分片) 是 SP 下省一次集合通信的干净范例; 其二, PR 正文的测量方法论值得借鉴——profiling 与内存预算双证据定位问题、只信有 trace 佐证的 ITL 数字、明确标注方差与 crossover 为估计值; 其三, 测试刻意用“互不相交 token 切片”建模, 能抓住“按 feature 拼接 partial”这类会产生看似正常实则错误输出的回归, 测试设计本身有教学价值。阅读时可重点结合 jeejeelee 的 AR 合并疑问与作者自述的 FusedMoE / 跨节点后续。

功能与动机

PR body 用两条独立证据定位问题: 一是 profiling——decode 步中共享专家 GEMM 对是仅次于 MegaMoE routed GEMM 的第二大开销项, 内核本身约 5.9 TB/s 是健康的, 问题在字节数 (共享专家 gate_up 176.2 MB + down 88.1 MB = 264.2 MB/层 /rank, 约 92 层合计约 22.6 GiB/rank); 二是部署内存预算——按 dense/TP + experts/(TP*DP) 预估的每 GPU 用量比实际少约 28 GiB, 其中约 23 GiB 来自复制的共享专家。佐证细节:

low_latency_gemm.py 的 KIMI_K3_PROJECTIONS 只为 shared_gate_up_proj / shared_down_proj 收录了 TP 分片形状的条目, 复制形状在 SP 下落到 cuBLAS。作者同时指出这是 vLLM 全局惯例 (DeepSeek-V4 同样复制共享专家并有显式注释), Kimi-K3 只因共享专家异常大而成为 outlier; TP 分片 + 每层两个集合通信只在低 token 数下占优, 因此 opt-in 默认关, 面向 P/D 分离的 decode 实例, prefill 应关闭。

实现拆解

变更入口: `vllm/envs.py` 新增全局开关 `VLLM_KIMI_K3_SHARD_SP_SHARED_EXPERT` (默认 0), 注释明确适用范围——仅 Kimi-K3、仅低 token 数场景、面向 P/D 分离部署的 decode 实例, prefill 与统一服务不应开启。命名在 review 中由最初的 `VLLM_KIMI_K3_SHARD_SP_MLP` 调整而来。

实现分 5 步:

1. 配置入口 (`vllm/envs.py`): 注册布尔开关并配详细注释, 默认关闭, 保证对现有行为零影响。
2. 分片判定函数 (`vllm/models/kimi_k3/nvidia/model.py`): 新增纯函数 `shard_sequence_parallel_mlp()`, 把“是否分片”收敛为四个条件: 开关开启、

use_sequence_parallel=True、调用方 eligible=True、tp_size > 1 且 hidden_size 与 intermediate_size 均能被 tp_size 整除。整除性检查避免分片后在 divide() 处出错，不满足则保守回退复制。

3. **KimiMLP 构造与 forward 改造**：新增 can_shard_sequence_parallel 参数；replicate = use_sequence_parallel and not shard_sequence_parallel 统一决定两个投影的 disable_tp；分片时 down_proj.reduce_results=False，把归约职责从 RowParallel 内部 all-reduce 移交给块尾 sp_reduce_scatter，避免重复归约；forward 在分片模式下首尾各插入 sp_all_gather / sp_reduce_scatter，整块仍是每方向一个集合通信。
4. 调用点接线：shared_experts 传 can_shard_sequence_parallel=self.use_mega_moe——仅 MegaMoE 路径直连共享专家时开放分片，FusedMoE 路径由 runner 接管归约并假设复制布局，故不开放；非 MoE 层的稠密 mlp 传 can_shard_sequence_parallel=True。
5. 测试配套 ([tests/models/kimi_k3/test_sequence_parallel.py](#))：新增 test_shard_sequence_parallel_mlp_gating (6 组参数化覆盖 opt-in、非 SP、FusedMoE 不 eligible、tp=1、6144 % 5 整除失败等分支) 与 test_sharded_sequence_parallel_mlp_matches_replicated (以每 rank 持有互不相交 token 切片建模，逐 token 对比分片版与复制版输出)。该等价性测试专门防住早期“按 feature 维拼接 partial”的错误设计——那种实现运行干净但输出退化。无部署 / schema 配套改动；开关与 gpu_memory_utilization 显存预算天然兼容，KV cache 自动受益。

关键文件：

- vllm/models/kimi_k3/nvidia/model.py (模块 模型实现；类别 source；类型 core-logic；符号 shard_sequence_parallel_mlp, KimiMLP)：PR 核心源码：新增 shard_sequence_parallel_mlp() 分片判定函数；改造 KimiMLP 支持 SP 下 TP 分片 (disable_tp 翻转、down_proj.reduce_results 移交、forward 首尾插入 sp_all_gather / sp_reduce_scatter)；并在 shared_experts (仅 MegaMoE 路径) 与 稠密 mlp 两个调用点接线。
- tests/models/kimi_k3/test_sequence_parallel.py (模块 序列并行；类别 test；类型 test-coverage；符号 test_shard_sequence_parallel_mlp_gating, test_sharded_sequence_parallel_mlp_matches_replicated)：新增两个高价值测试：test_shard_sequence_parallel_mlp_gating 参数化覆盖 6 种开关分支 (含 opt-in、FusedMoE 不 eligible、tp=1、6144 % 5 整除失败)；test_sharded_sequence_parallel_mlp_matches_replicated 用“每 rank 互不相交 token 切片”建模验证分片版与复制版逐 token 等价，专门防住早期错误设计。
- vllm/envs.py (模块 环境配置；类别 source；类型 configuration；符号 VLLM_KIMI_K3_SHARD_SP_SHARED_EXPERT)：新增 VLLM_KIMI_K3_SHARD_SP_SHARED_EXPERT 全局开关 (默认关)，注释明确适用范围与 P/D 取舍；命名在 review 中由 VLLM_KIMI_K3_SHARD_SP_MLP 调整而来。

关键符号：shard_sequence_parallel_mlp, KimiMLP.init, KimiMLP.forward, test_shard_sequence_parallel_mlp_gating, test_sharded_sequence_parallel_mlp_matches_replicated

关键源码片段

tests/models/kimi_k3/test_sequence_parallel.py

新增两个高价值测试：test_shard_sequence_parallel_mlp_gating 参数化覆盖 6 种开关分支（含 opt-in、FusedMoE 不 eligible、tp=1、6144 % 5 整除失败）；

test_sharded_sequence_parallel_mlp_matches_replicated 用“每 rank 互不相交 token 切片”建模验证分片版与复制版逐 token 等价，专门防住早期错误设计。

以下是防回归的关键测试（按合并后版本整理，注释为分析补充）：

```
def test_sharded_sequence_parallel_mlp_matches_replicated(default_vllm_config):
```

```
    """分片版 SP MLP 必须对每个 token 复现复制版的结果。
```

```
    关键点：每个 rank 持有互不相交的 token 切片，所以光有权重切片
    无法独立算完自己的 token——必须 all-gather 完整 token 集、按各自
    intermediate 切片算 partial 再 reduce-scatter。若改成按 feature 维
    拼接各 rank 的结果，会静默混入不同 token 的特征，产出看似正常
    实则完全错误的输出（早期实现就是这么挂的，本测试专门防回归）。
    """
```

```
    tp_size, hidden, intermediate, tokens_per_rank = 4, 16, 12, 3
```

```
    torch.manual_seed(0)
```

```
    num_tokens = tp_size * tokens_per_rank
```

```
    x = torch.randn(num_tokens, hidden)
```

```
    gate_weight = torch.randn(intermediate, hidden)
```

```
    up_weight = torch.randn(intermediate, hidden)
```

```
    down_weight = torch.randn(hidden, intermediate)
```

```
    act_fn = SiluAndMul()
```

```
    # 复制版：单 rank 持完整权重，对全部 token 直接计算。
```

```
    replicated = act_fn(x @ torch.cat([gate_weight, up_weight]).T) @ down_weight.T
```

```
    shard = intermediate // tp_size
```

```
    # 分片版：每个 rank all-gather 完整 token 集，只算自己的切片。
```

```
    partials = [
```

```
        act_fn(
```

```
            x
```

```
            @ torch.cat(
```

```
                [
```

```
                    gate_weight[r * shard : (r + 1) * shard],
```

```
                    up_weight[r * shard : (r + 1) * shard],
```

```
                ]
```

```
            ).T
```

```
        )
```

```
        @ down_weight[:, r * shard : (r + 1) * shard].T
```

```
        for r in range(tp_size)
```

```
    ]
```

```
    reduced = torch.stack(partials).sum(0)
```

```
    # reduce-scatter: rank r 只保留自己的 token 切片并核对结果。
```

```
    for r in range(tp_size):
```

```
        mine = reduced[r * tokens_per_rank : (r + 1) * tokens_per_rank]
```

```
        expected = replicated[r * tokens_per_rank : (r + 1) * tokens_per_rank]
```

`torch.testing.assert_close(mine, expected, atol=1e-5, rtol=1e-5)`

评论区精华

- 与纯 TP 的对比：WoosukKwon 在 PR 评论提问 “How's the perf compared to pure TP? ④”。作者随后补充 GB200 DP8xTP4 + DecodeBenchConnector 的复现 vs 分片对比：并发 64 提速 +17.5%，128 时 +9.2%，256~1024 落在 +3.5%~+6.1%，到 1536 转为 -0.2%——收益随并发增大单调衰减直至反转，印证正文“约 700 token (TP8) 为 crossover 估计值”。
- 环境变量命名（已解决）：jeejeelee 在 `vllm/envs.py` 提出 NIT 改名建议，把 `VLLM_KIMI_K3_SHARD_SP_MLP` 改成 `VLLM_KIMI_K3_SHARD_SP_SHARED_EXPERT` 并顺带重写注释；作者回复 “much better, agreed will change it :)”，最终合并代码采用新名。
- 集合通信可否再合并（未闭环）：jeejeelee 在 forward 的 `sp_reduce_scatter` 处提问 “How about combining this RS and the sequential AG into one AR?”，未见直接答复。PR 正文对 “reduce-scatter 双职责” 有辩护（相比 all-reduce 后再 reshuffle，块尾一个 RS 同时求和并恢复序列分片），可视作部分回答，但 “RS 与下一处 AG 合并成一次 AR” 的想法保持开放。
- 作者自述后续：需要跟进非 MegaMoE 场景；跨节点 TP 时可能希望只做节点内分片（需新建 parallel group，成本较高）。
 - 与纯 TP 的性能对比及收益随并发的衰减 (performance): 数据回答了问题：分片收益随并发 / token 数增大单调衰减直至反转，与正文“约 700 token (TP8) 为 crossover 估计值”一致；作者自认未做拥堵终止并发扫描。
 - 环境变量命名：`VLLM_KIMI_K3_SHARD_SP_MLP` → `VLLM_KIMI_K3_SHARD_SP_SHARED_EXPERT` (style): 已采纳，最终 `envs.py` 与 `model.py` 均使用 `VLLM_KIMI_K3_SHARD_SP_SHARED_EXPERT`。
 - reduce-scatter 与后续 all-gather 能否合并为一次 all-reduce (performance): 未闭环；正文设计说明可视为部分回答，但 “RS + 下一处 AG → 一次 AR” 的合并想法保持开放，作者自述后续将跟进非 MegaMoE 场景与节点内 TP 分组。

风险与影响

- 风险：
 - FusedMoE 路径开关静默失效：`can_shard_sequence_parallel` 仅在 MegaMoE 路径为 True，走 FusedMoE 时开关不报错也不生效，部署侧需确认 MoE 后端。
 - 全局开关与 P/D 两难：prefill 与 decode 在同一开关下结论相反，聚合部署中单个 flag 必然有一方处于劣势——这是作者明示的已知限制，只能在 P/D 分离时按角色设置 env。
 - 每层新增 2 个集合通信：收益随并发增大衰减直至反转（DecodeBench 在并发 1536 时转负），饱和点未测，crossover 是估计值而非实测。
 - 整除性约束：`intermediate_size % tp_size` 与 `hidden_size % tp_size` 不满足时静默回退复制（不会崩溃），但用户可能误以为已分片；gating 测试覆盖了 `tp=5` (`6144 % 5 ≠ 0`) 分支。

- 测量空白：性能数据每臂单次运行、间隔数小时，TTFT +10.7% 与吞吐 -2.6% 落在运行间方差内；且基于 pruned checkpoint 的开发构建，不代表发布配置；未覆盖跨节点 TP、FusedMoE 组合与 ROCm 路径。
- 影响：
 - 对部署方：Kimi-K3 P/D 分离的 decode 实例可用一个环境变量换取每 rank 约 18 GiB 显存（同 gpu_memory_utilization 下 KV cache +23%）与 decode 延迟约 10% 的下降；prefill 实例与聚合部署建议保持默认关闭。PR 被列入 v0.27.0 cherry picks 里程碑，说明将随下个版本发布。
 - 对系统行为：默认关闭、仅 Kimi-K3 NVIDIA + MegaMoE 路径生效，其余模型与路径零影响；权重加载、并行组、KV cache 计算逻辑均无改动。
 - 对团队与代码演进：shard_sequence_parallel_mlp + AG/RS 结构为其他共享专家体量大的模型提供了可复用的 SP 分片模板，同时留下 FusedMoE 支持与跨节点 TP 的节点内分组两项待办。
 - 风险标记：FusedMoE 路径开关静默失效，全局开关与 P/D 两难，每层新增 2 个集合通信，TP 整除性约束，收益随并发增大反转

关联脉络

- PR #51253 [ROCm][Perf] Kimi-K3 Shard Latent MoE up-projection for ROCm path: 同属“Kimi-K3 投影分片省带宽”优化主线：该 PR 在 ROCm 侧对 latent MoE up-projection 做 TP 分片，本 PR 在 NVIDIA 侧解决共享专家在 SP 下被复制的问题，共用 vllm/models/kimi_k3/ 目录结构（amd/ 与 nvidia/ 对称布局），分片与归约设计可互相借鉴。
- PR #50185 attn_res kernel latency improvements: Kimi-K3 decode 延迟优化群的另一支：该 PR 在 CUDA kernel 层优化 attn_res 算子，本 PR 在模型层优化权重布局，两者目标一致（decode ITL 下降），共同构成近期 Kimi-K3 性能工作线。