PR #36119 完整报告

sgl-project/sglang

[AMD][DSV4] perf: MXFP8 MoRI dispatch to match the w4a8 MoE input format

合并时间：2026-08-28 11:45

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36119

执行摘要

- 一句话：新增 MXFP8 MoRI dispatch，AMD DSV4 吞吐提升约 6-10%
- 推荐动作：值得精读。核心看点有三：一是“发送侧量化、按 live token 而非填充缓冲计数”的性能洞察，直击 AITER 量化网格按 padded 行数分配的开销来源；二是用 scale dtype（fp32 vs e8m0）而非 dispatch 枚举来区分输入格式，解耦了调度器与 runner 的枚举依赖；三是用启动期 inspect 签名探测 + bf16 回退管理跨仓库依赖，把“静默出错”的字节布局问题转化为显式警告。配套的契约测试思路（不依赖 GPU 却能钉住最容易静默回归的布局常量）也值得借鉴。

功能与动机

DeepSeek-V4 的 MoE 使用 per_1x32（MXFP4 权重）量化，AITER 需要携带 group-32 e8m0 微缩放的 fp8 激活，而 MoRI 现有三种 dispatch dtype 都无法产出该格式：bf16 无缩放（接收侧必须量化）、fp8 是 group-128 fp32 缩放（group 大小错误触发 fp8->bf16 upscale 往返）、fp4 缩放正确但 payload 需 upscale_mxfp4。bf16 只是因为无需 upscale kernel 才成为默认，但把 bf16->fp8 转换推给了接收侧——AITER 的量化网格按输入行数（MoRI 填充后的接收缓冲，3072*8=24576 行）而非实际 ~112 个 live token 分配，开销集中在接收侧。PR body 明确提出“quantizes on the send side, over live tokens, emitting the fp8 + group-32 e8m0 layout the MoE kernels consume directly”。

实现拆解

1. 扩展 DispatchDtype 枚举与布局计算（python/sglang/srt/layers/moe/token_dispatcher/moriep.py）：新增 mxfp8 = "mxfp8_blockwise" 成员；在 init_mori_op 中为 mxfp8 计算 scale_dim = hidden_size // MXFP4_BLOCK_SIZE（每 32 通道一个 scale，DSV4 隐藏维度 7168 → 224 个 scale），scale_type_size 取 torch.float8_e8m0fnu.itemsize（1 字节），确保 dispatch 缓冲按 e8m0 布局精确分配，不欠分配也不浪费。
2. 环境变量接线与 aiter 能力探测（moriep.py）：_apply_dispatch_dtype_override 解析 SGLANG_MORI_DISPATCH_DTYPE=mxfp8 时，先调用新增的 _aiter_supports_mxfp8_dispatch()（lru_cache 缓存，用 inspect 检查 aiter.get_hip_quant 及其 per_1x32 实例是否接受 scale_type 参数）验证 aiter 能消费该布局；不支持则 warning_once 并回退 bf16。默认仍是 bf16，不改变既有行为。
3. 接收侧跳过 upscale（python/sglang/srt/layers/moe/moe_runner/aiter.py 的 _pre_permute_deepep_to_aiter）：新增 is_mx_fp8_dispatch 判断——a1_scale 存在、

dtype 为 float8_e8m0fnu 且非 fp4 payload；命中时跳过 is_w4a4 分支的 fp8->bf16 upscale，fp8 + e8m0 数据直接交给 fused_moe。与普通 fp8 dispatch 的区分依据是 scale 的 dtype（fp32 vs e8m0），而非 dispatch 枚举本身。

4. 测试配套：新增 test/registered/unit/layers/test_moriep_mxfp8_dispatch.py（81 行，9 个 CPU 用例），锁定枚举唯一性、MXFP4_BLOCK_SIZE=32、7168//32=224、e8m0 1 字节、env 映射到枚举成员、空 live token 批的 scale 形状 (0, 224)；经 register_cpu_ci 注册到 base-a-test-cpu 套件。
5. 性能验证：MI355X（gfx950）、TP8 + DP8 attention + EP8/MoRI + EAGLE MTP、8192 in / 1024 out，同构建仅切换环境变量，每 conc 跑一轮；所有数据点均超出 0.5-2.7% 的 run-to-run 噪声带。

conc	吞吐 bf16 (tok/s)	吞吐 mxfp8 (tok/s)	吞吐提升	TPOT bf16 (ms)	TPOT mxfp8 (ms)	TPOT 改善
4	1022.03	1121.97	+9.8%	33.30	30.19	-9.3%
8	1818.01	1969.84	+8.4%	35.52	32.73	-7.9%
16	3080.51	3277.43	+6.4%	42.12	39.52	-6.2%
32	4527.58	4905.24	+8.3%	57.79	53.17	-8.0%
64	6264.62	6827.92	+9.0%	85.21	78.19	-8.2%

关键文件：

- python/sglang/srt/layers/moe/token_dispatcher/moriep.py（模块 专家分发；类别 source；类型 dependency-wiring；符号 DispatchDtype, _aiter_supports_mxfp8_dispatch, init_mori_op, _apply_dispatch_dtype_override）：变更入口与核心接线：新增 DispatchDtype.mxfp8 枚举、_aiter_supports_mxfp8_dispatch 启动探测、init_mori_op 中 e8m0 布局计算（scale_dim = hidden_size // 32、1 字节 scale）、_apply_dispatch_dtype_override 的环境变量解析与 bf16 回退。
- python/sglang/srt/layers/moe/moe_runner/aiter.py（模块 专家执行；类别 source；类型 core-logic；符号 _pre_permute_deepep_to_aiter）：接收侧核心逻辑：_pre_permute_deepep_to_aiter 中通过 a1_scale.dtype == float8_e8m0fnu 识别 mxfp8 dispatch，跳过 fp8->bf16 upscale，让 fp8 数据直达 fused_moe。
- test/registered/unit/layers/test_moriep_mxfp8_dispatch.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_mxfp8_member_exists_and_is_distinct, test_scale_group_size_is_32, test_scale_dim_matches_group_32_layout, test_e8m0_scale_is_one_byte）：9 个 CPU 契约用例锁定字节布局与 env 接线，覆盖最容易静默回归的点（scale group、scale 数量、e8m0 字节数、空 token 批形状）；注册进 base-a-test-cpu CI 套件。

关键符号：_aiter_supports_mxfp8_dispatch, _apply_dispatch_dtype_override, init_mori_op, _pre_permute_deepep_to_aiter

关键源码片段

python/sglang/srt/layers/moe/token_dispatcher/moriep.py

变更入口与核心接线：新增 DispatchDtype.mxfp8 枚举、_aiter_supports_mxfp8_dispatch 启动探测、init_mori_op 中 e8m0 布局计算（scale_dim = hidden_size // 32、1 字节 scale）、_apply_dispatch_dtype_override 的环境变量解析与 bf16 回退。

```
# moriep.py：mxfp8 dispatch 模式的核心接线
# 1) 探测当前 aiter 能否消费 group-32 e8m0 布局；用 lru_cache 缓存，
# 避免每次初始化都重复 import + inspect。
@functools.lru_cache(maxsize=1)
def _aiter_supports_mxfp8_dispatch() -> bool:
    """确认 aiter 是否支持 fp8 激活 + group-32 e8m0 微缩放。

    这里必须探测而不是假设：旧版 per_1x32 量化照样返回 fp8，
    但附带连续 fp32 缩放，MoE 侧会把 fp32 字节误读为 e8m0，
    产生垃圾输出而非异常。通过 inspect 检查签名，把问题拦在
    启动期并回退到 bf16。
    """
    try:
        import inspect

        from aiter import get_hip_quant

        return "scale_type" in inspect.signature(get_hip_quant).parameters or any(
            "scale_type" in inspect.signature(f).parameters
            for f in (get_hip_quant(QuantType.per_1x32),)
        )
    except Exception:
        return False


# 2) 枚举新增 mxfp8 成员：fp8 载荷 + group-32 e8m0 微缩放，
# 与 per_1x32（MXFP4 权重）MoE kernel 的输入格式完全一致。
class DispatchDtype(Enum):
    bf16 = "bfloat16"
    fp8 = "float8_blockwise"
    fp4 = "mxfp4_blockwise"
    mxfp8 = "mxfp8_blockwise"


# 3) 环境变量接线（_apply_dispatch_dtype_override 内）：默认仍是 bf16；
# mxfp8 仅在 aiter 支持时启用，否则回退 bf16 并给出明确警告，
# 避免静默的字节布局错配。
elif dispatch_dtype == "mxfp8":
    if _aiter_supports_mxfp8_dispatch():
        self.dispatch_dtype = DispatchDtype.mxfp8
    else:
```

```
        logger.warning_once(
            "SGLANG_MORI_DISPATCH_DTYPE=mxfp8 requires an aiter "
            "build whose per_1x32 quant accepts scale_type "
            "(for the group-32 e8m0 byte layout the MoE kernels "
            "consume). This aiter does not, so the send-side "
            "quant would emit continuous fp32 scales and the "
            "MoE would read them as e8m0 bytes. Falling back to "
            "bf16 dispatch."
        )
```

python/sglang/srt/layers/moe/moe_runner/aiter.py

接收侧核心逻辑：_pre_permute_deepep_to_aiter 中通过 a1_scale.dtype == float8_e8m0fnu 识别 mxfp8 dispatch，跳过 fp8->bf16 upscale，让 fp8 数据直达 fused_moe。

```
# aiter.py：_pre_permute_deepep_to_aiter 中判断是否跳过 upscale 往返。
# mxfp8 dispatch 已携带 fp8 数据 + group-32 e8m0 缩放，正是 per_1x32
# 想要的输入格式，可直接交给 fused_moe；只有 fp8 dispatch 的
# group-128/fp32 缩放才需要 dequant 往返，两者的区分依据是
# a1_scale 的 dtype（fp32 vs e8m0）。
is_mx_fp8_dispatch = (
    a1_scale is not None
    and a1_scale.dtype == torch.float8_e8m0fnu
    and not is_fp4_dispatch
)

if (
    is_w4a4
    and a1_scale is not None
    and not is_fp4_dispatch
    and not is_mx_fp8_dispatch
):
    # W4A4 权重 + 普通 FP8 dispatch：先 dequant FP8 -> BF16；
    # FP4 per_1x32 路径需要 BF16 输入。
    hidden_states = upscale(
        hidden_states, a1_scale, num_local_tokens, output_dtype
    )
    a1_scale = None
```

test/registered/unit/layers/test_moriep_mxfp8_dispatch.py

9 个 CPU 契约用例锁定字节布局与 env 接线，覆盖最容易静默回归的点（scale group、scale 数量、e8m0 字节数、空 token 批形状）；注册进 base-a-test-cpu CI 套件。

```
# 契约测试：锁定 mxfp8 dispatch 的字节布局，防止“能跑但悄悄退化”。
# 错误 group 大小或错误 scale dtype 的 fp8 payload 仍能运行，
# 但会重新引入本模式要消除的 upscale 往返，症状只有吞吐损失。
def test_scale_dim_matches_group_32_layout():
    """每 32 通道一个 scale；错配会欠分配 scale buffer，
    导致 kernel 越界读取。"""
```

```
    assert HIDDEN % MXFP4_BLOCK_SIZE == 0
    assert HIDDEN // MXFP4_BLOCK_SIZE == 224


def test_empty_token_batch_scale_shape():
    """decode 可能给某 rank 0 个 live token；空分支也必须产出
    形状正确的 scale tensor，否则 all-to-all 会失步。"""
    scale = torch.empty((0, HIDDEN // MXFP4_BLOCK_SIZE), dtype=torch.float8_e8m0fnu)
    assert scale.shape == (0, 224)
    assert scale.dtype == torch.float8_e8m0fnu
```

评论区精华

该 PR 没有 review 行内评论，核心讨论集中在 issue 评论与合入前的跨仓库协调：

- karverma-amd 在 issue 评论中说明配套 aiter PR（ROCm/aiter#4954）为 fused_moe 增加 12 行 a8w4 mxfp8 passthrough 分支，两边可任意顺序合并；若设置了 SGLANG_MORI_DISPATCH_DTYPE=mxfp8 但 aiter 缺少配套改动，8b0cd59 新增的启动探测会检测到 per_1x32 量化不接受 scale_type，记录警告并回退 bf16。
- 作者明确指出"Probed rather than assumed"的原因：旧版 per_1x32 量化仍返回 fp8 但带连续 fp32 缩放，MoE 侧会把 fp32 字节误读为 e8m0 产生垃圾输出而非异常，因此必须把问题拦截在启动期。
- HaiShaw 直接 APPROVED 并通过 /tag-and-rerun-ci 触发 CI 复跑；4 个 commit 中后两个为格式化与 CI 注册的收尾（pre-commit、CPU CI 注册），说明过程经过了 lint 与测试基建的打磨。
- aiter 配套改动、合并顺序与回退机制 (design): 通过启动期签名探测 + bf16 回退消除跨仓库依赖风险；HaiShaw 批准并触发 /tag-and-rerun-ci。

风险与影响

- 风险：
  - 跨仓库依赖：mxfp8 模式的收益依赖 ROCm/aiter#4954 合入；缺省时虽有启动探测兜底回退，但若 aiter 侧把 scale_type 参数改名或改为 keyword-only 意外路径，inspect 探测可能误判，需要保持两侧联动维护。
  - 字节布局静默回归：fp8 payload 配上错误 scale group（128）或错误 scale dtype（fp32）仍能运行，只是悄悄退回 upscale 往返，症状仅是吞吐损失；本 PR 用 CPU 契约测试钉住布局，但 GPU 端真实 kernel 路径没有被自动化单测覆盖（测试不依赖 GPU）。
  - scale dtype 启发式判定：aiter.py 用 a1_scale.dtype == torch.float8_e8m0fnu 区分 mxfp8 与普通 fp8 dispatch；若未来 fp8 dispatch 更改 scale dtype 或出现新的 e8m0 变体，该判定可能误判分支。
  - 空 live token 批：decode 可能给某 rank 0 个 token，空分支必须产出形状正确的 scale tensor，否则 all-to-all 失步；测试覆盖了 (0, 224) 形状，但真机多 rank 场景仍依赖 benchmark 验证。

    - CI 状态：PR Test (Base) 显示失败而 Extra 通过，材料未给出失败原因，合入前应确认该失败与本 PR 无关。
- 影响：
    - 用户影响：仅 AMD（MI355X/gfx950 等）且启用 MoRI 后端 + DSV4 类 per_1x32 权重的用户，显式设置 SGLANG_MORI_DISPATCH_DTYPE=mxfp8 可获得 6.4%-9.8% 吞吐提升与 6.2%-9.3% TPOT 改善；默认为 bf16，其余用户零影响。
    - 系统影响：改动集中在 MoE dispatch 与 aiter runner 两条窄路径，不触碰调度器、KV cache 等核心模块；新增 1 个 CPU CI 测试文件，CI 负担约 5 秒。
    - 团队影响：需要与 ROCm/aiter 仓库协调发布节奏（aiter 侧先合入则立即受益，后合入则自动回退），是跨仓库协作的典型样例。
    - 风险标记：依赖配套 aiter 改动，字节布局静默回归，GPU 路径无单测覆盖，默认 bf16 的 opt-in 特性

关联脉络

- PR #35374 [Kernel] Add H200 MoE configs for Qwen3.5 and Qwen3.6: 同属 MoE 性能调优脉络（H200 FP8 吞吐提升 vs 本 PR MI355X MXFP8 吞吐提升），说明 MoE 量化 / 调度是持续优化重点，可对照不同硬件上的收益路径。
- PR #36529 [Fix][XPU/ROCm/NPU] Defer sgl_kernel.quantization import in expert_pack: 同为 ROCm/XPU 侧 MoE 量化路径的兼容性修复，与本次 aiter 依赖接线守护同一类“导入 / 量化时点”问题。
- PR #36736 [AMD][CI] Merge the four MI35x DeepSeek-V3.2 nightly jobs into two to save runtime: AMD MI35x 平台 CI 建设与本 PR 的 AMD DSV4 验证呼应，且本 PR 的单元测试也注册进了 CPU CI 套件。