PR #33128 完整报告

sgl-project/sglang

Support DeepGEMM for standard MoE dispatch

合并时间：2026-08-03 12:48

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33128

执行摘要

- 一句话：补齐 DeepGEMM 标准 MoE 调度，新增紧凑布局与 UE8M0 重量化
- 推荐动作：值得重点关注的设计点：masked/compact 布局启发式与 CUDA Graph 静态 buffer 的权衡、padding 行用 expert index -1 跳过无效数据的技巧、以及 UE8M0 直接量化激活以避免 2x 误差的精度处理。建议性能团队以本 PR 为基线，针对 compact 布局的 block_e 对齐与 masked/compact 交叉点做专项调优；合并者已批准，注意跟进 EP 路径 kernel 参数变更对其他调用点的影响。

功能与动机

PR body 明确说明：deep_gemm MoE runner 虽然可与标准 token dispatcher 组合，但现有集成隐含 DeepEP 的权重与激活布局——标准 MoE 层不会一致地把 gate/up 与 down 两组专家权重重量化为 UE8M0，标准调度路径也未提供 DeepGEMM 需要的紧凑 local-expert 布局。本次变更的目标是在保留 DeepEP 路径的同时，补齐标准调度下的 DeepGEMM 支持。

实现拆解

1. 统一标准层 UE8M0 权重重量化（python/sglang/srt/layers/quantization/fp8.py）：在 Fp8MoEMethod.process_weights_after_loading_block_quant 中删除对 DeepEPMoE 类型的断言，改为对 w13_weight 与 w2_weight 两个权重 /scale 对循环调用 requant_block_scale_ue8m0_for_deepgemm，并显式传入 output_dtype=torch.bfloat16 与 weight_shape，使标准 MoE 层也能被正确预重量化。
2. 新增 masked/compact 布局决策（python/sglang/srt/layers/moe/moe_runner/deep_gemm.py）：新增 _should_use_masked_standard_layout（num_experts > num_local_experts 且 num_local_experts <= 32 时走 masked）与 _get_compact_all_tokens（计算紧凑布局的静态行数上界）；pre_permute_standard_to_deep_gemm 据此分支：masked 路径复用 moe_ep_deepgemm_preprocess，compact 路径则用 ep_scatter 与 fused_moe_dispatch_index 构造 CUDA Graph 兼容的静态 buffer。
3. scatter 内核支持 padding 标记（python/sglang/kernels/ops/moe/ep_moe_kernels.py）：_fwd_kernel_ep_scatter_1 新增 num_valid_tokens_per_expert 参数，在 padding 行写入 expert index -1，使 DeepGEMM 跳过无效行；同时 moe_ep_deepgemm_preprocess 在 DEEPGEMM_SCALE_UE8M0 开启时直接用 per_token_quant_fp8_ue8m0_scatter 量化 FP8 激活，避免先量化再舍入 scale 带来的 2x 表示误差。

4. BF16 compact buffer 改为未初始化分配：因为 scatter 会在 grouped GEMM 读取前写满所有有效行，padding 行又由 -1 标记得以跳过，torch.empty 即可安全使用（对应 review 中 ch-wan 的提问）。
5. 测试配套：test_minimax_quant_scatter.py 增加 UE8M0 scale 一致性、紧凑上界参数化、masked/compact 端到端一致性测试；test_deepgemm_ue8m0_requant.py 增加标准层 requant 调用与 format_ue8m0 标记断言；注册的内核测试在 GB300 上 52 passed。

关键文件：

- python/sglang/srt/layers/moe/moe_runner/deep_gemm.py（模块 MoE 运行器；类别 source；类型 core-logic；符号 _should_use_masked_standard_layout, _get_compact_all_tokens, pre_permute_standard_to_deep_gemm）：核心实现：新增 masked/compact 布局决策与 CPUGraph 静态上界计算，改造 standard -> deep_gemm 的 pre-permute 路径。
- python/sglang/srt/layers/quantization/fp8.py（模块 量化层；类别 source；类型 dependency-wiring；符号 process_weights_after_loading_block_quant）：解除标准 MoE 层必须为 DeepEPMoE 的假设，w13/w2 统一按 UE8M0 重量化，是标准调度集成的权重侧前提。
- python/sglang/kernels/ops/moe/ep_moe_kernels.py（模块 MoE 内核；类别 source；类型 infrastructure；符号 _fwd_kernel_ep_scatter_1, ep_scatter, moe_ep_deepgemm_preprocess）：scatter 内核增加有效 token 数参数，padding 行写 -1，支撑 compact 布局与 CUDA Graph；FP8 激活改用 UE8M0 scale 直接量化。
- test/registered/kernels/ops/moe/test_minimax_quant_scatter.py（模块 量化测试；类别 test；类型 test-coverage；符号 test_standard_deepgemm_preprocess_quantizes_with_ue8m0_scale, test_compact_all_tokens_uses_tight_routing_independent_bound, test_standard_masked_runner_matches_compact_end_to_end）：覆盖标准调度 UE8M0 量化、紧凑上界参数化、masked 与 compact 端到端一致性，是本次功能正确性的核心验证。
- test/registered/unit/layers/quantization/test_deepgemm_ue8m0_requant.py（模块 重量化测试；类别 test；类型 test-coverage；符号 test_fp8_moe_requants_standard_layer_for_deepgemm）：CPU 单测验证标准层 requant 调用参数与 UE8M0 标记，防止 fp8.py 的权重处理回归。

关键符号：_should_use_masked_standard_layout, _get_compact_all_tokens, pre_permute_standard_to_deep_gemm, post_permute_deep_gemm_to_standard, _fwd_kernel_ep_scatter_1, ep_scatter, moe_ep_deepgemm_preprocess, process_weights_after_loading_block_quant

关键源码片段

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

核心实现：新增 masked/compact 布局决策与 CPUGraph 静态上界计算，改造 standard -> deep_gemm 的 pre-permute 路径。

```
def _should_use_masked_standard_layout(runner_config: MoeRunnerConfig) -> bool:
```

```
    # 当存在专家并行且本地专家数较少时，masked 布局的缓冲区不会随专家数膨胀
    return (
        runner_config.num_experts > runner_config.num_local_experts
        and runner_config.num_local_experts <= 32
    )


def _get_compact_all_tokens(
    num_assignments: int, num_experts: int, block_e: int = 128
) -> int:
    # 紧凑布局的静态行数上界：每个非空专家至少占一个 block_e，
    # 其余分配再按 block_e 分桶对齐，保证任意路由都不会越界
    max_nonempty_experts = min(num_assignments, num_experts)
    return block_e * (
        max_nonempty_experts + (num_assignments - max_nonempty_experts) // block_e
    )
```

python/sglang/srt/layers/quantization/fp8.py

解除标准 MoE 层必须为 DeepEPMoE 的假设，w13/w2 统一按 UE8M0 重量化，是标准调度集成的权重侧前提。

```
if not self.is_fp4_expert:
    weight_block_size = self.quant_config.weight_block_size
    # 标准 MoE 层也统一对 w13 与 w2 两组权重做 UE8M0 重量化，
    # 不再要求层类型必须是 DeepEPMoE
    for weight, weight_scale in (
        (layer.w13_weight, layer.w13_weight_scale_inv),
        (layer.w2_weight, layer.w2_weight_scale_inv),
    ):
        requant_block_scale_ue8m0_for_deepgemm(
            weight,
            weight_scale,
            weight_block_size,
            use_deepgemm_runner=will_use_deepgemm,
            output_dtype=torch.bfloat16,
            weight_shape=weight.shape[-2:],
        )
```

python/sglang/kernels/ops/moe/ep_moe_kernels.py

scatter 内核增加有效 token 数参数，padding 行写 -1，支撑 compact 布局与 CUDA Graph；FP8 激活改用 UE8M0 scale 直接量化。

```
@triton.jit
def _fwd_kernel_ep_scatter_1(
    num_recv_tokens_per_expert,
    num_valid_tokens_per_expert,
    expert_start_loc,
    m_indices,
    num_experts: tl.constexpr,
```

```
    BLOCK_E: tl.constexpr,
):
    # 每个 program 处理一个 expert 的填充行区间
    cur_expert = tl.program_id(0)
    cur_expert_start = tl.load(expert_start_loc + cur_expert)
    cur_expert_padded_token_num = tl.load(num_recv_tokens_per_expert + cur_expert)
    cur_expert_valid_token_num = tl.load(num_valid_tokens_per_expert + cur_expert)
    m_indices_start_ptr = m_indices + cur_expert_start
    off_expert = tl.arange(0, BLOCK_E)
    # 超出有效 token 数的 padding 行写入 -1，让 DeepGEMM 跳过
    for start_m in tl.range(0, cur_expert_padded_token_num, BLOCK_E, num_stages=4):
        offsets = start_m + off_expert
        tl.store(
            m_indices_start_ptr + offsets,
            tl.where(offsets < cur_expert_valid_token_num, cur_expert, -1),
        )
```

评论区精华

review 中 ch-wan 在 deep_gemm.py 的 diff 上提问 can we use torch.empty?，YAMY1234 回复已将 BF16 compact buffer 改为 torch.empty：scatter 写满所有有效行后才进入 grouped GEMM，padding 行带 expert index -1 会被 DeepGEMM 跳过。此外 hnyls2002 在 issue 评论中分享其被关闭的平行 PR #33342 的 benchmark：128 本地专家无 EP（block-fp8）下 compact 比 masked 慢 15%-19%，约在 8192 tokens 才交叉；48 本地专家 tp8/ep8（fp4）下慢 22%-30%，提示当前布局启发式在部分 shape 上并非最优。PR body 的对比测试也显示，标准调度下 DeepGEMM 全部低于 FlashInfer TRTLLM（慢 24%-42%），团队仍以功能正确性优先合入。

- BF16 compact buffer 可否使用 torch.empty (correctness): YAMY1234 确认已改为 torch.empty：scatter 会在 grouped GEMM 读取前写满所有有效行，padding 行带 expert index -1 会被 DeepGEMM 跳过。
- compact 布局在部分 shape 下比 masked 慢 (performance): 未当场解决；当前启发式（本地专家 > 32 时走 compact）需在更多 shape 上验证，是后续优化切入点。
- DeepGEMM 标准调度性能整体低于 FlashInfer TRTLLM (performance): 接受性能回退，作为功能正确性优先的一次集成；后续需专项调优。

风险与影响

- 风险：
  1. 性能回退：GB300 上 concurrency 1-128 全部劣化 23.99%-41.74%，若用户手动选择 deep_gemm runner 且规模类似会直接损失吞吐；PR 未新增用户可见选项，默认不受影响。
  2. 核心热路径变更：pre_permute_standard_to_deep_gemm 是 standard dispatch + deep_gemm 组合的必经路径，masked/compact 分支影响所有该组合请求；running_state 中新增 src2dst 等键，post-permute 依赖其正确性。

3. 未初始化 buffer 安全性：compact BF16 buffer 用 torch.empty，安全性依赖 scatter 写满所有有效行 + padding 写 -1；若路由异常或内核改动破坏该不变式，可能读取未初始化数据（不会崩溃但产生错误结果）。
4. kernel API 变化：_fwd_kernel_ep_scatter_1 与 ep_scatter 增加参数，所有调用点需同步更新；该文件同时被 DeepEP/ 其他 EP 路径使用，存在回归风险。
5. 覆盖有限：精度与性能只验证了 PP2/TP8/EP1、FP8 权重、Qwen3.5-397B 单一配置；FP4/MXFP8、其他并行配置与 Breakable Prefill Graph 未覆盖（log 显示预填充 graph 被兼容性 guard 跳过）。 - 影响：对用户：新增 DeepGEMM + standard dispatcher 的可运行组合，但当前性能显著低于 FlashInfer TRTLLM，适合以功能验证为目的的实验，不建议生产默认切换；无新增配置项。对系统：MoE 执行路径扩展为 DeepEP 布局与标准调度 compact/masked 双布局共存，CUDA Graph 静态形状由 _get_compact_all_tokens 保证；scatter kernel 变更影响 EP 相关路径。对团队：合入后为后续 DeepGEMM 标准调度性能优化（如 masked/compact 交叉点研究、JIT kernel 调优）奠定基础，同时关闭了 #33342 的平行实现。 - 风险标记：性能回退 24%-42%，核心 MoE 调度热路径变更，Blackwell 专属 UE8M0 路径，未初始化 buffer 依赖 scatter 写满，并行配置覆盖有限

关联脉络

- PR #33342 类似 DeepGEMM 标准调度变更（已关闭）：hnyls2002 指出该平行实现因本 PR 关闭，并提供了 compact vs masked 的 benchmark 数据。
- PR #36237 [MegaMoE] Respect padded MXFP8 scale row strides in pre-dispatch: 同为 MoE dispatch 预处理的 padding/scale 布局修复，与本 PR 的 compact 布局 padding 处理相关。
- PR #35188 [Bugfix] Fix int32 destination offset overflow in CUTLASS MoE pre-reorder: 两者都修改 ep_moe_kernels.py 中 MoE 预排序 / 预处理内核。
- PR #36097 Fix MXFP8 MoE weight sizing for non-gated models: 同样涉及 fp8.py 中 MoE 权重尺寸与量化处理路径。