PR #17392 完整报告

sgl-project/sglang

Add BF16 support to EP-MoE for DeepGEMM

合并时间：2026-05-14 02:06

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/17392

执行摘要

- 一句话：DeepGEMM EP-MoE 新增 BF16 支持
- 推荐动作：该 PR 值得阅读，尤其对 DeepGEMM wrapper 模式和自定义 Triton kernel 实现感兴趣的同学。设计上延续 FP8 架构，代码模块化清晰。建议关注后续是否添加自动化测试。

功能与动机

用户尝试在 BF16 模型上启用 EP-MoE 时发现 DeepGEMM 后端不支持 BF16 数据类型。本项目通过利用 DeepGEMM 内置的 grouped BF16 GEMM kernel 填补这一空白。

实现拆解

1. DeepGEMM wrapper 层（entrypoint.py、compile_utils.py）：新增 grouped_gemm_nt_bf16_masked 和 grouped_gemm_nt_bf16_contig 函数，以及对应的 _BF16GroupedMaskedWarmupExecutor、_BF16GroupedContWarmupExecutor 预热执行器，遵循 FP8 相同模式。
2. MoE 执行器（deep_gemm.py）：在 DeepGemmRunnerCore.run() 中添加权重 dtype 分支，当权重为 bf16 时调用新增的 _run_bf16_contiguous_gemm 或 _run_masked_bf16_gemm；这两个函数直接调用 BF16 GEMM 并处理激活 / 降幅。
3. Triton 内核（ep_moe/kernels.py）：新增 _silu_and_mul_kernel Triton kernel 和 silu_and_mul_masked_fwd 调度函数，专门用于 BF16 激活；同时修改 ep_scatter 使其在非 FP8 模式下跳过 scale 处理（通过新增 IS_FP8 参数）。
4. 环境变量与条件分支（server_args.py、deepep.py）：若 SGLANG_DEEPEP_BF16_DISPATCH 启用且为 MUSA 平台，则自动选择 deep_gemm 作为 runner backend；dispatcher 在 BF16 分发时跳过 FP8 量化逻辑。
5. 非量化 MoE 集成（unquant.py、fused_moe_triton/layer.py、compressed_tensors/compressed_tensors.py）：将 UnquantizedFusedMoEMethod 扩展以支持 use_deep_gemm 标志，使其可以在 forward 时创建 DeepGemmMoeQuantInfo 并设置 use_fp8=False，从而触发 BF16 分支。

关键文件：

- python/sglang/srt/layers/deep_gemm_wrapper/compile_utils.py（模块 编译工具；类别 source；类型 core-logic；符号 _BF16GroupedContWarmupExecutor, init, execute,

_BF16GroupedMaskedWarmupExecutor）：核心编译工具，新增 BF16 分组 GEMM 的 warmup executor，并扩展 DeepGemmKernelType 枚举和内存预算计算

- python/sglang/srt/layers/moe/moe_runner/deep_gemm.py（模块 MoE 执行器；类别 source；类型 core-logic；符号 _run_bf16_contiguous_gemm, _run_masked_bf16_gemm）：MoE 执行器核心，新增 BF16 的 contiguous 和 masked GEMM 运行路径，并根据权重 dtype 路由
- python/sglang/srt/layers/moe/ep_moe/kernels.py（模块 EP 内核；类别 source；类型 core-logic；符号 _silu_and_mul_kernel, silu_and_mul_masked_fwd）：新增 BF16 专用的 Triton SiLU + Multiply fused kernel，并扩展 ep_scatter 支持非 FP8 模式
- python/sglang/srt/layers/deep_gemm_wrapper/entrypoint.py（模块 GEMM 接口；类别 source；类型 core-logic；符号 grouped_gemm_nt_bf16_masked, grouped_gemm_nt_bf16_contig）：DeepGEMM 包装器入口，新增 BF16 分组 GEMM 的公开接口
- python/sglang/srt/layers/quantization/unquant.py（模块 非量化方法；类别 source；类型 dependency-wiring）：非量化 MoE 方法调整，使 DeepGEMM 后端能正确处理 BF16 权重（设置 use_fp8=False）

关键符号：_BF16GroupedContWarmupExecutor.init, _BF16GroupedContWarmupExecutor.execute, _BF16GroupedMaskedWarmupExecutor.init, _BF16GroupedMaskedWarmupExecutor.execute, _run_bf16_contiguous_gemm, _run_masked_bf16_gemm, _silu_and_mul_kernel, silu_and_mul_masked_fwd, grouped_gemm_nt_bf16_masked, grouped_gemm_nt_bf16_contig

关键源码片段

python/sglang/srt/layers/deep_gemm_wrapper/compile_utils.py

核心编译工具，新增 BF16 分组 GEMM 的 warmup executor，并扩展 DeepGemmKernelType 枚举和内存预算计算

```
class _BF16GroupedContWarmupExecutor(_BaseWarmupExecutor):
    """BF16 版本的 grouped contiguous GEMM 预热执行器。
    与 FP8 版本不同，这里直接使用 bf16 张量，无需 scale 处理。
    """
    def __init__(self, max_m: int, n: int, k: int, num_groups: int):
        # 分配 BF16 输入、权重和输出张量
        self.a = torch.empty((max_m, k), device="cuda", dtype=torch.bfloat16)
        self.b = torch.empty((num_groups, n, k), device="cuda", dtype=torch.bfloat16)
        self.m_indices = torch.zeros((max_m,), device="cuda", dtype=torch.int32)
        self.out = torch.empty((max_m, n), device="cuda", dtype=torch.bfloat16)

    def execute(self, m):
        # 调用 DeepGEMM 的 BF16 连续布局 grouped GEMM 进行预热
        deep_gemm.m_grouped_bf16_gemm_nt_contiguous(
            self.a[:m], self.b, self.out[:m], self.m_indices[:m]
        )
```

```
class _BF16GroupedMaskedWarmupExecutor(_BaseWarmupExecutor):
    """BF16 版本的 grouped masked GEMM 预热执行器。
    与连续版本不同，这里需要 masked_m 和 expected_m 参数。
    """
    def __init__(self, max_m: int, n: int, k: int, num_groups: int):
        # 分配 BF16 张量，形状为 [num_groups, max_m, k] 以适应 masked 布局
        self.a = torch.empty((num_groups, max_m, k), device="cuda", dtype=torch.bfloat16)
        self.b = torch.empty((num_groups, n, k), device="cuda", dtype=torch.bfloat16)
        self.masked_m = torch.zeros((num_groups,), device="cuda", dtype=torch.int32)
        self.out = torch.empty((num_groups, max_m, n), device="cuda", dtype=torch.bfloat16)

    def execute(self, m):
        # 设置 masked_m 为实际值 m
        self.masked_m.fill_(m)
        deep_gemm.m_grouped_bf16_gemm_nt_masked(
            self.a, self.b, self.out, self.masked_m, m
        )
```

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

MoE 执行器核心，新增 BF16 的 contiguous 和 masked GEMM 运行路径，并根据权重 dtype 路由

```
def _run_bf16_contiguous_gemm(
    self,
    runner_input: DeepGemmRunnerInput,
    quant_info: DeepGemmMoeQuantInfo,
    running_state: dict,
) -> torch.Tensor:
    # 从 runner_input 和 running_state 中提取必要参数
    hidden_states = runner_input.hidden_states
    all_tokens = running_state["all_tokens"]
    hidden_states_shape = running_state["hidden_states_shape"]
    m_indices = runner_input.m_indices

    N = quant_info.w13_weight.size(1)
    K = hidden_states_shape[1]
    w13_weight = quant_info.w13_weight
    w2_weight = quant_info.w2_weight

    # 第一步 grouped GEMM: (M, K) x (E, N, K) -> (M, N)
    gateup_output = torch.empty(
        (all_tokens, N),
        device=hidden_states.device,
        dtype=torch.bfloat16,
    )
    deep_gemm_wrapper.grouped_gemm_nt_bf16_contig(
        hidden_states, w13_weight, gateup_output, m_indices,
    )
    dispose_tensor(hidden_states)
```

```
    # 第二步激活：SiLU + Multiply，根据平台选择实现
    if not _is_musa:
        down_input = torch.empty(
            (all_tokens, N // 2),
            device=gateup_output.device,
            dtype=torch.bfloat16,
        )
        _legacy_silu_and_mul(gateup_output.view(-1, N), down_input)
    else:
        # MUSA 平台使用 torch 原生 SwishGLU
        down_input = _silu_and_mul_musa(gateup_output.view(-1, N))
    del gateup_output

    # 第三步 grouped GEMM: (M, N/2) x (E, K, N/2) -> (M, K)
    down_output = torch.empty(
        (all_tokens, K),
        device=down_input.device,
        dtype=torch.bfloat16,
    )
    deep_gemm_wrapper.grouped_gemm_nt_bf16_contig(
        down_input, w2_weight, down_output, m_indices,
    )
    dispose_tensor(down_input)
    return down_output
```

评论区精华

Review 中 BBuf 提出了两个问题：

1. (MUSA 检查) 在 server_args.py 中为什么需要 is_musa() 检查？作者 froststeam 解释目前 MUSA 平台 EP MoE 只有 DeepGEMM 实现，此检查避免影响其他平台。
2. (import 位置) 在 unquant.py 中建议将 from sglang.srt.environ import envs 移到文件顶部。作者接受并已移动。

- MUSA 平台 check 的合理性 (design): 接受解释，保持检查仅在 MUSA 平台生效。
- 导入语句位置优化 (style): 已按建议移动 import。

风险与影响

- 风险：核心变更涉及 MoE 推理路径中的 GEMM 和激活函数，可能影响 BF16 模型的正确性和性能。当前缺少自动化端到端测试（仅单点 warmup 测试），精度验证仅依赖 Benchmark 日志。MUSA 平台特有的条件分支可能引入维护复杂度，且可能与非 DeepGEMM 后端产生交互。
- 影响：影响使用 DeepGEMM 作为 MoE runner backend 的 BF16 模型（如 Qwen3.5-35B-A3B）。对 MUSA 平台用户，设置 SGLANG_DEEPEP_BF16_DISPATCH=1 可启用；CUDA 用户同样适用。不影响 FP8 模型或非 EP-MoE 场景。
- 风险标记：核心路径变更，缺少自动化测试，MUSA 平台专用分支

关联脉络

- 暂无明显关联 PR