PR #27867 完整报告

sgl-project/sglang

[DSv4] Loading Time Weight Dequant

合并时间：2026-07-07 09:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27867

执行摘要

- 一句话：DSV4 Flash 加载时 FP4→FP8 去量化，TP8 性能提升 1.56-2.23x
- 推荐动作：值得精读，尤其是 cast_e2m1fn_to_e4m3fn 的无损转换设计和 review 中关于环境变量检测的讨论。若需在 H20 上部署 DeepSeek V4 Flash 并利用 TP=8，此 PR 是关键依赖。

功能与动机

当前 DeepSeek V4 Flash 的可用权重（来自 deepseek-ai 或 sgl-project）不支持 TP=8，而在 H20 上 TP=8 性能更好。因此希望在权重加载时进行 FP4 到 FP8 的去量化，因为转换过程依赖于 TP 大小。详见关联 Issue #23602。

实现拆解

1. 环境变量注册：在 python/sglang/srt/environ.py 新增 SGLANG_DSV4_FP4_DEQUANT 环境变量（默认 False），作为去量化功能的开关。
2. 配置检测增强：在 python/sglang/srt/configs/model_config.py 的 ModelConfig.__init__ 中，当检测到 SGLANG_DSV4_FP4_DEQUANT 设置时，强制运行 FP4 专家布局自动检测（try_detect_fp4_experts），并将检测结果写回该环境变量，确保后续逻辑能正确识别。
3. 核心去量化函数：在 python/sglang/srt/layers/quantization/fp8.py 中新增 cast_e2m1fn_to_e4m3fn 函数，使用查找表将 packed 的 e2m1fn 值映射为 float32，再结合 block 级别的 scale 进行重缩放和偏移处理，最终输出标准 FP8 张量和对应的 scale_max_offset_bits。
4. 量化配置标志传递：在 python/sglang/srt/model_loader/loader.py 中，将环境变量值注入 Fp8Config.dequant_fp4_to_fp8 标志，同时 Fp8Config.__init__ 增加该成员变量。
5. MoE 方法路由调整：在 Fp8Config.get_quant_method 中，当 is_fp4_experts and dequant_fp4_to_fp8 时，直接返回普通 Fp8MoEMethod，跳过专门的 FP4 MoE runner（如 Marlin 或 FlashInfer MXFP4），并使用 assert 确保只有 auto 后端通过。
6. 端到端测试：在 test/registered/models_e2e/test_deepseek_v4_flash_fp4_h200.py 中新增 TestDSV4FlashFP4DequantTP8H200 测试类，以 TP=8 启动服务并注入环境变量，运行基础解码正确性和 GSM8K 精度测试。

关键文件：

- python/sglang/srt/layers/quantization/fp8.py（模块 量化层；类别 source；类型 core-logic；符号 cast_e2m1fn_to_e4m3fn）：核心变更文件，新增 cast_e2m1fn_to_e4m3fn 去量化函数，并修改 Fp8Config 和 get_quant_method 以支持 dequant 路径。
- test/registered/models_e2e/test_deepseek_v4_flash_fp4_h200.py（模块 端到端测试；类别 test；类型 test-coverage；符号 TestDSV4FlashFP4DequantTP8H200, setUpClass, tearDownClass）：新增 TP=8 去量化端到端测试类，验证精度和性能。
- python/sglang/srt/configs/model_config.py（模块 配置层；类别 source；类型 data-contract）：调整 DSV4 专家检测逻辑，当 dequant 环境变量设置时强制运行检测并写回结果。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 core-logic）：注册了核心环境变量 SGLANG_DSV4_FP4_DEQUANT。
- python/sglang/srt/model_loader/loader.py（模块 模型加载器；类别 source；类型 data-contract）：将 dequant 标志从环境变量注入 Fp8Config，实现配置传递。

关键符号：cast_e2m1fn_to_e4m3fn, Fp8Config.get_quant_method

关键源码片段

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

核心变更文件，新增 cast_e2m1fn_to_e4m3fn 去量化函数，并修改 Fp8Config 和 get_quant_method 以支持 dequant 路径。

```
# 预计算 FP4 到 float32 查找表 (e2m1fn 格式 )
DSV4_DEQUANT_FP4_TABLE = torch.tensor(
    [0.0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 6.0, # 正半部分
     0.0, -0.5, -1.0, -1.5, -2.0, -3.0, -4.0, -6.0], # 负半部分
    dtype=torch.float32,
)


def cast_e2m1fn_to_e4m3fn(
    x: torch.Tensor, scale: torch.Tensor
) -> tuple[torch.Tensor, torch.Tensor]:
    """
    将 packed e2m1fn (FP4) 权重无损转换为 e4m3fn (FP8) 格式。
    输入 x 的 shape 为 (out_dim, in_dim)，其中 in_dim 维度 packed 了 2 个 FP4 值。
    返回转换后的 FP8 张量及重缩放因子。
    """
    assert x.dtype == torch.int8, "输入必须是 int8（packed FP4）"
    assert x.ndim == 2
    out_dim, in_dim = x.size()
    in_dim *= 2 # 展开 packed 维度
    fp8_block_size = 128
    fp4_block_size = 32
    assert in_dim % fp8_block_size == 0 and out_dim % fp8_block_size == 0
    assert scale.size(0) == out_dim and scale.size(1) == in_dim // fp4_block_size
```

```
    # 1. 将 int8 视为 uint8，提取低 4 位和高 4 位
    x = x.view(torch.uint8)
    low = x & 0x0F
    high = (x >> 4) & 0x0F
    table = DSV4_DEQUANT_FP4_TABLE.to(x.device)
    # 2. 查表得到浮点值并展开：(out_dim, in_dim/2, 2) -> (out_dim, in_dim)
    x = torch.stack([table[low.long()], table[high.long()]], dim=-1).flatten(2)

    # 3. 准备重缩放：计算最大偏移量，使得 (max_fp4 * offset) 不溢出 fp8
    MAX_OFFSET_BITS = 6 # 6.0 * 2^6 = 384 < 448 (fp8 max)
    bOut = out_dim // fp8_block_size
    bIn = in_dim // fp8_block_size
    # 将 x reshape 为 (bOut, fp8_block_size, bIn, fp8_block_size) 并转置
    x = x.view(bOut, fp8_block_size, bIn, fp8_block_size).transpose(1, 2)
    # 将 scale 展平为 (bOut, bIn, 128*4)
    scale = scale.float().view(bOut, fp8_block_size, bIn, -1).transpose(1, 2).flatten(2)
    # 计算每个块的最大 offset bits
    scale_max_offset_bits = scale.amax(dim=-1, keepdim=True) / (2**MAX_OFFSET_BITS)
    offset = scale / scale_max_offset_bits
    # 将 offset 展开到每个 fp4 元素
    offset = offset.unflatten(-1, (fp8_block_size, -1)).repeat_interleave(fp4_block_size, dim=-1)
    # 4. 应用重缩放并恢复原始 shape
    x = (x * offset).transpose(1, 2).reshape(out_dim, in_dim)
    # 转换为 fp8 格式
    return x.to(torch.float8_e4m3fn), scale_max_offset_bits.squeeze(-1).to(torch.float8_e8m0fnu)
```

test/registered/models_e2e/test_deepseek_v4_flash_fp4_h200.py

新增 TP=8 去量化端到端测试类，验证精度和性能。

```
class TestDSV4FlashFP4DequantTP8H200(
    BasicDecodeCorrectnessMixin, GSM8KMixin, CustomTestCase
):
    """SGLANG_DSV4_FP4_DEQUANT=1: TP=8，加载时去量化后通过普通 FP8 MoE 路径推理。"""

    gsm8k_accuracy_thres = 0.93

    @classmethod
    def setUpClass(cls):
        cls.model = try_cached_model(MODEL) # 使用原始 FP4 模型
        cls.base_url = DEFAULT_URL_FOR_TEST
        # 启动 TP=8 服务器，注入 dequant 环境变量
        cls.process = popen_launch_server(
            cls.model,
            cls.base_url,
            timeout=SERVER_LAUNCH_TIMEOUT,
            other_args=[
                "--trust-remote-code",
                "--tp", "8",
                "--speculative-algorithm", "EAGLE",
```

```
                "--speculative-num-steps", "3",
                "--speculative-eagle-topk", "1",
                "--speculative-num-draft-tokens", "4",
                "--watchdog-timeout", "900",
            ],
            env={"SGLANG_DSV4_FP4_DEQUANT": "1"},
        )

    @classmethod
    def tearDownClass(cls):
        if hasattr(cls, "process") and cls.process:
            kill_process_tree(cls.process.pid)
```

评论区精华

> 讨论 1：环境变量检测逻辑（@BBuf 评论 python/sglang/srt/configs/model_config.py）
> 指出 is_set() 无法区分 0 和 false，应使用 .get() 进行条件判断。作者已修改为 .get() 方式。
>
> 讨论 2：后端兼容性风险（@BBuf 评论 python/sglang/srt/layers/quantization/fp8.py）
> 当已启用 dequant 时，若用户指定 --moe-runner-backend flashinfer_mxfp4 等不兼容后端，Fp8MoEMethod 可能无法创建相应 runner，导致运行时错误。建议要么拒绝组合，要么规范化后端。作者最终在 get_quant_method 中添加 assert 检查，仅允许 auto 后端通过。

- 环境变量检测逻辑 (correctness): 作者已修改为使用 .get() 并写回实际检测结果。
- 后端兼容性风险 (design): 作者在 get_quant_method 中添加 assert 检查，仅允许 auto 后端通过。

风险与影响

- 风险：
  - 兼容性风险：SGLANG_DSV4_FP4_DEQUANT 仅在 is_deepseek_v4 分支生效，非 DSV4 模型无影响。
  - 数值精度风险：去量化过程在 float32 进行，查找表 + 重缩放可能引入微小误差，但 MMLU 测试通过。
  - 后端冲突风险：若用户同时设置 --moe-runner-backend flashinfer_mxfp4，会触发 assert 阻止启动，避免静默错误。
  - 运行时依赖：需要 PyTorch 支持 torch.float8_e4m3fn，较老版本可能失败。
- 影响：
  - 用户：提供了一条新的部署路径，使得 TP=8 在 H20 上可获得显著性能提升（TPOT 1.56-2.23 倍），同时保持精度。
  - 系统：增加了加载时的一次性转换开销（FP4→FP8），但推理时完全走 FP8 流水线，无额外开销。

- 团队：需同时维护原生 FP4 MoE 和去量化 FP8 两条 MoE 路径，新增代码集中于 fp8.py ，相对独立。
- 风险标记：环境变量二义性，后端兼容性，数值精度风险

关联脉络

- 暂无明显关联 PR