PR #22316 完整报告

sgl-project/sglang

[Reland] DeepSeek-R1-0528-w4a8: DeepEP Low Latency Dispatch Adopts FP8 Communication

合并时间：2026-04-10 14:56

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22316

执行摘要

本 PR 为 DeepSeek-R1-0528-w4a8 模型实现了 DeepEP 低延迟调度中的 FP8 通信优化，通过新增 Triton 内核和调整相关逻辑，将通信格式从 BF16 改为 FP8，减少带宽消耗并提升吞吐量约 10%。尽管优化显著，但 review 中揭示了兼容性风险，需关注后续修复。

功能与动机

为什么做？从 PR body 的 profiling 数据可知，当 DeepEP 启用时，DeepSeek-R1-0508-W4AFP8 模型的通信延迟是 DeepSeek-R1-0528 模型的两倍，根源是 DeepEP Dispatch 使用 BF16 通信导致带宽消耗增加，影响推理性能。因此，目标是通过改用 FP8 通信来优化性能，降低延迟。

实现拆解

1. 新增 Triton 量化内核：在 python/sglang/srt/layers/moe/ep_moe/kernels.py 中，新增以下核心函数，实现 per-token 到 per-tensor 的 FP8 量化转换，以适应 cutlass_w4a8_moe 操作符接口。

```python @triton.jit def _fp8_per_token_quant_to_per_tensor_quant_kernel( x_ptr, x_scale_ptr, x_scale_stride0, x_scale_stride1, x_scale_stride2, masked_m_ptr, output_scale_ptr, output_ptr, m, k, K_SCALE_BLOCK_SIZE: tl.constexpr, K_BLOCK_SIZE: tl.constexpr, ): pid_k, pid_m, pid_e = ( tl.program_id(axis=0), # 处理 k 维度的分块 tl.program_id(axis=1), # 处理 token 维度 tl.program_id(axis=2), # 处理专家维度 ) pid_m_dim = tl.num_programs(1)

    token_id = pid_m
    last_effective_id = tl.load(masked_m_ptr + pid_e)  # 获取每个专家的有效 token 数

    if token_id >= last_effective_id:
        return  # 跳过无效 token，优化性能
    output_scale_val_inv = 1.0 / tl.load(output_scale_ptr).to(tl.float32)  # 计算反量化尺度
    k_offsets = pid_k * K_BLOCK_SIZE + tl.arange(0, K_BLOCK_SIZE)
    scale_offsets = (k_offsets // K_SCALE_BLOCK_SIZE) * x_scale_stride2  # 计算尺度偏移

    x_ptrs = x_ptr + pid_e * m * k + k_offsets
    output_ptrs = output_ptr + pid_e * m * k + k_offsets
    x_scale_ptrs = x_scale_ptr + pid_e * x_scale_stride0 + scale_offsets
```

```
    for tok_idx in tl.range(token_id, last_effective_id, pid_m_dim):
        hidden = tl.load(x_ptrs + tok_idx * k).to(tl.float32)  # 加载量化输入
        scale_fp32 = tl.load(x_scale_ptrs + tok_idx * x_scale_stride1).to(tl.float32)  # 加载 per-token 尺度
        hidden = hidden * scale_fp32 * output_scale_val_inv  # 转换为 per-tensor 量化
        tl.store(output_ptrs + tok_idx * k, hidden.to(output_ptr.dtype.element_ty))  # 存储结果
```

1. 修改 MoE 计算函数：在 python/sglang/srt/layers/moe/cutlass_w4a8_moe.py 中，调整 cutlass_w4a8_moe_deepep_ll 函数，将输入从单个张量 a 改为 a_states 和 a_scales，并调用新内核 fp8_per_token_to_per_tensor_quant_triton 替代原有的 per_tensor_quant_fp8，确保 FP8 量化正确集成到计算流程中。
2. 调整调度器逻辑：在 python/sglang/srt/layers/moe/token_dispatcher/deepep.py 中，修改 _dispatch_core 函数，删除对 envs.SGLANG_DEEPEP_BF16_DISPATCH 的检查，直接设置 use_fp8 = True，强制启用 FP8 通信，简化配置但引入兼容性风险。
3. 移除断言限制：在 python/sglang/srt/layers/moe/ep_moe/layer.py 中，删除 forward_cutlass_w4afp8_masked 函数中的断言，该断言原本阻止 W4AFP8 模型使用 FP8 调度，现在允许其使用 FP8 通信以匹配优化。
4. 更新量化层逻辑：在 python/sglang/srt/layers/quantization/w4afp8.py 中，修改 apply_deepep_ll 函数，添加 hidden_scales 参数的传递，以支持新量化逻辑。

测试与基准：PR body 提供了准确性测试（如 gsm8k 和 ceval 数据集）和性能基准（在 H20 GPU 上吞吐量提升约 10%），但未新增单元测试文件，依赖现有集成测试。

关键源码片段

python/sglang/srt/layers/moe/ep_moe/kernels.py

新增核心 Triton 内核函数，实现 per-token 到 per-tensor 的 FP8 量化，是性能优化的基础组件。

```
@triton.jit
def _fp8_per_token_quant_to_per_tensor_quant_kernel(
    x_ptr,
    x_scale_ptr,
    x_scale_stride0,
    x_scale_stride1,
    x_scale_stride2,
    masked_m_ptr,
    output_scale_ptr,
    output_ptr,
    m,
    k,
    K_SCALE_BLOCK_SIZE: tl.constexpr,
    K_BLOCK_SIZE: tl.constexpr,
):
    pid_k, pid_m, pid_e = (
        tl.program_id(axis=0), # 维度 k 的程序 ID
```

```
        tl.program_id(axis=1), # 维度 m（token）的程序 ID
        tl.program_id(axis=2), # 维度 e（专家）的程序 ID
    )
    pid_m_dim = tl.num_programs(1) # m 维度的程序数量

    token_id = pid_m
    last_effective_id = tl.load(masked_m_ptr + pid_e) # 加载每个专家的有效 token 数量

    if token_id >= last_effective_id:
        return # 跳过无效 token
    output_scale_val_inv = 1.0 / tl.load(output_scale_ptr).to(tl.float32) #
    计算输出尺度的倒数用于反量化
    k_offsets = pid_k * K_BLOCK_SIZE + tl.arange(0, K_BLOCK_SIZE) # 计算 k 维度的偏移
    scale_offsets = (k_offsets // K_SCALE_BLOCK_SIZE) * x_scale_stride2 #
    计算尺度张量的偏移量

    x_ptrs = x_ptr + pid_e * m * k + k_offsets # 输入数据指针
    output_ptrs = output_ptr + pid_e * m * k + k_offsets # 输出数据指针
    x_scale_ptrs = x_scale_ptr + pid_e * x_scale_stride0 + scale_offsets # 尺度数据指针

    for tok_idx in tl.range(token_id, last_effective_id, pid_m_dim):
        hidden = tl.load(x_ptrs + tok_idx * k).to(tl.float32) # 加载 per-token 量化数据
        scale_fp32 = tl.load(x_scale_ptrs + tok_idx * x_scale_stride1).to(tl.float32) # 加载 per-
        token 尺度
        hidden = hidden * scale_fp32 * output_scale_val_inv # 转换为 per-tensor 量化
        tl.store(output_ptrs + tok_idx * k, hidden.to(output_ptr.dtype.element_ty)) # 存储结果

def fp8_per_token_to_per_tensor_quant_triton(
    x: torch.Tensor,
    x_scale: torch.Tensor,
    masked_m: torch.Tensor,
    output_scale: torch.Tensor,
    output: torch.Tensor,
):
    K_SCALE_BLOCK_SIZE = 128 # 每个尺度块的大小
    assert len(x.shape) == 3 and x.size(2) % K_SCALE_BLOCK_SIZE == 0 # 验证输入形状
    assert x.is_contiguous() # 确保输入是连续内存
    assert x_scale.size(2) == x.size(2) // K_SCALE_BLOCK_SIZE # 验证尺度张量形状
    assert output_scale.numel() == 1 # 输出尺度应为标量

    K_BLOCK_SIZE = 1024 # 内核处理的块大小
    assert x.size(2) % K_BLOCK_SIZE == 0
    grid = (x.size(2) // K_BLOCK_SIZE, 32, x.size(0)) # 设置计算网格：k 维度分块、32 个 token
    并行、专家数
    _fp8_per_token_quant_to_per_tensor_quant_kernel[grid](
        x,
        x_scale,
        *x_scale.stride(), # 传递尺度张量的步幅
        masked_m,
```

```
        output_scale,
        output,
        x.size(1), # m
        x.size(2), # k
        K_SCALE_BLOCK_SIZE=K_SCALE_BLOCK_SIZE,
        K_BLOCK_SIZE=K_BLOCK_SIZE,
        num_warps=8, # 设置 warp 数以优化性能
    )
```

评论区精华

review 中，Todobe 和 OrangeRedeng 提出了关键质疑：

> Todobe: “环境变量 SGLANG_DEEPEP_BF16_DISPATCH 与 W4AFP8 权重无关。该变量用于非量化权重。直接删除此处会导致非量化权重无法正确使用 deepep。”
>
> OrangeRedeng: “我确认此更改破坏了使用 wna16 量化方案的模型，例如 Kimi-K2.5。”

讨论结论是 OrangeRedeng 提出了快速修复 PR #22822，建议通过传递量化类型参数来解决兼容性问题，而非直接修改代码。这揭示了性能优化与向后兼容性之间的设计权衡。

风险与影响

技术风险：

- 回归风险：修改可能影响其他量化方案（如 wna16），导致模型无法正确运行。
- 兼容性问题：移除环境变量依赖可能破坏现有配置，特别是对于非量化模型。
- 性能风险：新 Triton 内核的正确性和效率需验证，错误可能引入精度损失或性能下降。
- 缺少测试覆盖：没有新增单元测试，依赖集成测试和基准，可能隐藏边界情况。

影响范围：

- 用户：吞吐量提升约 10%，改善推理体验和效率。
- 系统：减少通信带宽消耗，提升资源利用率和扩展性。
- 团队：代码变更涉及核心调度和量化模块，需要确保不影响其他功能；维护性因环境变量移除而简化，但可能引入配置复杂性。

关联脉络

- 历史 PR：PR #21232 “perf optimization for eplb” 同属 DeepSeek 模型性能优化，涉及 eplb 算法，与本 PR 在性能优化和 DeepSeek 集成方面形成连续脉络。
- 关联修复：PR #22822 是针对本 PR 中兼容性问题的快速修复，展示了社区如何响应 review 反馈，平衡性能优化与系统稳定性。
- 演进趋势：结合近期历史 PR（如 #22606 优化流式响应性能），可见仓库正持续优化核心路径性能，特别是在调度和量化领域，本 PR 是这一趋势的具体体现。