PR #28333 完整报告

sgl-project/sglang

Call Flashinfer `mm_fp8` for per-tensor FP8 GEMMs on SM100

合并时间：2026-06-17 11:50

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28333

执行摘要

- 一句话：Flashinfer bmm_fp8 替代 CUTLASS 实现 SM100 FP8 线性层
- 推荐动作：建议仔细阅读 fp8_utils.py 中 flashinfer_bmm_fp8 的封装方式，以及 modelopt_quant.py 中条件路径的切换逻辑。该 PR 展示了如何在特定硬件上安全替换核心算子，并平衡通用性与性能。审阅中关于后端清理的建议也值得关注。

功能与动机

当前 FP8 per-tensor GEMM 使用 CUTLASS kernel，但该 kernel 并非真正的 per-tensor 设计（其 epilogue 中存在奇怪的重复缩放）。Flashinfer 的 bmm_fp8 不仅具有正确的 per-tensor 语义，还能在 cuBLAS、cuDNN 和 CUTLASS 之间自动调优，有望获得显著性能提升。

实现拆解

1. 新增 Flashinfer bmm_fp8 自定义操作(fp8_utils.py)：在 is_blackwell_supported() and is_flashinfer_available() 代码块中，注册名为 flashinfer_bmm_fp8 的 custom op。将输入从 [M,K] unsqueeze 为 [1,M,K]，权重从 [K,N] unsqueeze 为 [1,K,N]，调用 _raw_flashinfer_bmm_fp8 后再 view 回 [M,N]。同时新增便捷函数 apply_fp8_linear_bmm_flashinfer，封装了静态量化（static_quant_fp8）与 bmm 调用，并支持可选 bias。
2. 修改 ModelOptFp8LinearMethod 前向路径(modelopt_quant.py)：在 __init__ 中根据 is_sm100_supported() and is_flashinfer_available() 设置 enable_flashinfer_bmm 标志。apply 方法中优先使用新的 bmm 路径；process_weights_after_loading 中当启用 bmm 时，不再将 weight_scale 强制转换为 per-channel（因为 flashinfer bmm 原生支持 per-tensor scale）。
3. 调整 Flashinfer autotune 启动条件(model_runner.py)：在 _should_run_flashinfer_autotune 中新增 fp8_gemm_needs_autotune 判断，当使用 flashinfer_cutlass 后端或 modelopt FP8 量化且为 SM100 时，将 autotune 标志置为 True。
4. 为 FP8 量化内核添加 PDL 支持(fp8_kernel.py)：在 Triton kernel _static_quant_fp8 中新增 USE_PDL 参数，在 load 前后插入 gdc_wait() 和 gdc_launch_dependents() 以利用 SM100 的程序依赖启动特性。static_quant_fp8 函数根据 is_arch_support_pdl() 动态传入 PDL 参数。

关键文件：

- python/sglang/srt/layers/quantization/fp8_utils.py（模块 量化层；类别 source；类型 core-logic；符号 flashinfer_bmm_fp8, apply_fp8_linear_bmm_flashinfer）：核心文件：新增 flashinfer_bmm_fp8 custom op 和 apply_fp8_linear_bmm_flashinfer 封装函数，实现 per-tensor FP8 GEMM 的 flashinfer 调用。
- python/sglang/srt/model_executor/model_runner.py（模块 模型运行器；类别 source；类型 data-contract）：让 autotune 启动条件覆盖 FP8 per-tensor GEMM 场景，确保 SM100 上 flashinfer 自动调优正确触发。
- python/sglang/srt/layers/quantization/modelopt_quant.py（模块 量化配置；类别 source；类型 data-contract）：修改 ModelOptFp8LinearMethod，在 SM100 上启用 flashinfer bmm 路径，并相应调整 weight_scale 后处理逻辑。
- python/sglang/srt/layers/quantization/fp8_kernel.py（模块 量化内核；类别 source；类型 dependency-wiring；符号 _static_quant_fp8, static_quant_fp8）：为 FP8 静态量化 Triton kernel 添加 PDL 支持，利用 SM100 硬件新特性提升 kernel 级并发。

关键符号：flashinfer_bmm_fp8, apply_fp8_linear_bmm_flashinfer, _static_quant_fp8, static_quant_fp8, _should_run_flashinfer_autotune

关键源码片段

python/sglang/srt/layers/quantization/fp8_utils.py

核心文件：新增 flashinfer_bmm_fp8 custom op 和 apply_fp8_linear_bmm_flashinfer 封装函数，实现 per-tensor FP8 GEMM 的 flashinfer 调用。

```
# python/sglang/srt/layers/quantization/fp8_utils.py

# 在 Blackwell 支持且 flashinfer 可用时，导入并注册 custom op
if is_blackwell_supported() and is_flashinfer_available():
    from flashinfer import SfLayout
    from flashinfer import bmm_fp8 as _raw_flashinfer_bmm_fp8 # 新增：导入 bmm_fp8
    # ... 其他导入
    from sglang.srt.utils.custom_op import register_custom_op

    # 将 flashinfer bmm_fp8 包装为 custom op，避免 torch.compile 追踪内部 JIT
    @register_custom_op(
        op_name="flashinfer_bmm_fp8",
        mutates_args=[],
        fake_impl=lambda q_input, weight, x_scale, weight_scale, out_dtype: (
            q_input.new_empty((q_input.shape[0], weight.shape[1]), dtype=out_dtype)
        ),
    )
    def flashinfer_bmm_fp8(
        q_input: torch.Tensor, # [M, K] FP8 e4m3
        weight: torch.Tensor, # [K, N] FP8 e4m3 ( 列优先 )
        x_scale: torch.Tensor, # per-tensor 标量
        weight_scale: torch.Tensor, # per-tensor 标量
```

```
        out_dtype: torch.dtype,
    ) -> torch.Tensor:
        """通过 flashinfer bmm_fp8 计算 per-tensor FP8 矩阵乘法 (仅用于 SM100)。"""
        m, n = q_input.shape[0], weight.shape[1]
        # bmm_fp8 要求输入为 3D: [B, M, K] 和 [B, K, N]
        return _raw_flashinfer_bmm_fp8(
            q_input.unsqueeze(0), # [1, M, K]
            weight.unsqueeze(0), # [1, K, N]
            x_scale.reshape(1), # 保持为标量
            weight_scale.reshape(1),
            out_dtype,
            backend="auto", # 允许 flashinfer autotune
        ).view(m, n) # 消除 batch 维度

    # ... 其余代码

# 文件底部新增的便捷函数
def apply_fp8_linear_bmm_flashinfer(
    input: torch.Tensor,
    weight: torch.Tensor,
    weight_scale: torch.Tensor,
    input_scale: torch.Tensor,
    bias: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    """Per-tensor static FP8 linear via flashinfer bmm_fp8 (SM10X only)."""
    output_shape = [*input.shape[:-1], weight.shape[1]]
    input_2d = input.view(-1, input.shape[-1])
    # 使用给定的 per-tensor scale 进行静态量化
    qinput, x_scale = static_quant_fp8(input_2d, input_scale, repeat_scale=False)
    output = flashinfer_bmm_fp8(qinput, weight, x_scale, weight_scale, input.dtype)
    if bias is not None:
        output = output + bias
    return output.view(*output_shape)
```

python/sglang/srt/layers/quantization/fp8_kernel.py

为 FP8 静态量化 Triton kernel 添加 PDL 支持，利用 SM100 硬件新特性提升 kernel 级并发。

```
# python/sglang/srt/layers/quantization/fp8_kernel.py

# 在文件顶部新增导入
from sglang.jit_kernel.utils import is_arch_support_pdl

@triton.jit
def _static_quant_fp8(
    # ... 其他参数
    REPEAT_SCALE: tl.constexpr,
    USE_PDL: tl.constexpr = False, # 新增参数：是否启用程序依赖启动
):
    """使用给定 scale 对张量进行浮点8量化。"""
```

```
    g_id = tl.program_id(0)
    # ... 指针计算
    cols = tl.arange(0, BLOCK)
    mask = cols < N

    # SM100 PDL: 在 load 前等待之前的数据生产
    if USE_PDL:
        tl.extra.cuda.gdc_wait()

    y = tl.load(y_ptr + cols, mask=mask, other=0.0).to(tl.float32)
    y_s = tl.load(y_s_ptr).to(tl.float32)

    # PDL: 标记当前 kernel 的依赖已完成
    if USE_PDL:
        tl.extra.cuda.gdc_launch_dependents()

    y_s_inv = 1.0 / y_s
    y_q = tl.clamp(y * y_s_inv, fp8_min, fp8_max).to(y_q_ptr.dtype.element_ty)
    # ... 存储

def static_quant_fp8(
    x: torch.Tensor,
    x_s: torch.Tensor,
    repeat_scale: bool = False,
) -> Tuple[torch.Tensor, torch.Tensor]:
    # ... 参数检查
    BLOCK = triton.next_power_of_2(N)
    num_warps = min(max(BLOCK // 256, 1), 8)
    num_stages = 1
    # 检查硬件是否支持 PDL，动态传入编译参数
    pdl_kwargs = {"USE_PDL": True, "launch_pdl": True} if is_arch_support_pdl() else {}
    _static_quant_fp8[(M,)](
        x,
        x_q,
        x_s,
        x_s_repeat,
        x.shape[-1],
        N,
        fp8_min=FP8_MIN,
        fp8_max=FP8_MAX,
        BLOCK=BLOCK,
        REPEAT_SCALE=repeat_scale,
        num_warps=num_warps,
        num_stages=num_stages,
        **pdl_kwargs, # 传递 PDL 参数
    )
    # ... 返回
```

评论区精华

审阅者 Fridge003 建议未来清理 cutlass_fp8_supported 标志，统一为更清晰的 --fp8-gemm-backend 参数，并为 flashinfer gemm 创建独立的 fp8 gemm backend。该建议未在本 PR 中实现，但已在 review 中记录为后续 TODO。

- 清理 cutlass_fp8_supported 标志并统一 --fp8-gemm-backend (design): 已记录为 TODO，未在本 PR 中实现。

风险与影响

- 风险：
  1. 回归风险：新的 bmm 路径仅在 SM100 且 flashinfer 可用时激活（条件严格），不影响其他架构。但若 flashinfer 库版本不兼容 bmm_fp8 API，可能导致运行时错误，需确保依赖版本。
  2. 性能风险：autotune 条件新增 fp8_gemm_needs_autotune，在 modelopt FP8 + SM100 情况下会触发 autotune，可能增加启动延迟，但预期在线性层收益下可接受。
  3. 正确性风险：process_weights_after_loading 中当 enable_flashinfer_bmm 为 True 时跳过了 per-channel 转换，若 flashinfer bmm 实际需要 per-channel scale 则会导致精度错误。但 flashinfer bmm_fp8 文档表明支持 per-tensor scale，风险低。
  4. PDL 引入风险：gdc_wait / gdc_launch_dependents 依赖 SM100 硬件特性，is_arch_support_pdl() 已做保护，非 SM100 不会启用。 - 影响：用户影响：SM100 设备上使用 ModelOpt FP8 量化模型（如 Nemotron-3）的用户将自动获得性能提升，无需手动配置。其他架构用户无影响。系统影响：flashinfer autotune 增加启动时间和显存占用（autotune 结果缓存），但仅针对符合条件的模型。团队影响：代码设计提供了架构特定优化的参考模式，但增加了条件分支复杂度，需要后续重构建议。
- 风险标记：仅 SM100 生效，依赖 flashinfer bmm_fp8 API，autotune 启动条件变化，PDL 仅限于 SM100

关联脉络

- 暂无明显关联 PR