# PR #25532 完整报告

sgl-project/sglang

[fp8] SM90 swap-AB scaled_mm dispatch (~1.16x kernel geomean, +5.8-18.5% end-to-end)

合并时间：2026-05-20 13:20

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25532

## 执行摘要

- 一句话：SM90 FP8 GEMM 引入 swap-AB 调度，小 batch 解码加速 1.16x
- 推荐动作：建议精读，特别是 fp8_gemm_sm90_dispatch.cuh 中基于 M/N 的分桶策略、swap-AB 的模板化实现以及 epilogue 的广播抽象。该 PR 展示了如何通过 CUTLASS 3.x EVT 灵活组合高效 GEMM 变体，对于未来 sgl-kernel 支持的优化有参考价值。

## 功能与动机

原始 SM90 FP8 调度对所有小 M 路径使用固定 tile <_64, _64, _128>，对于 M_orig << 64（如 decode batch=1），只有 1/64 的 M 维度是有效计算，大量填充浪费。Swap-AB 将 GEMM 转置为 D^T = B^T · A^T，使 kernel M 映射到大得多的 N_orig 维度，配合更小的 kernel-N tile（16, 32）减少无效填充，从而提升小 batch decode 性能。

## 实现拆解

1. 移植 CUTLASS 3.x Epilogue 组件：从 vLLM 上游引入 broadcast_load_epilogue_c3x.hpp 和 scaled_mm_epilogues_c3x.hpp，支持行 / 列 / 标量广播，实现运行时 per-tensor/per-channel 精度选择，避免标量在 CPU/GPU 间拷贝的性能问题。
2. 编写调度内核：在 fp8_gemm_sm90_dispatch.cuh 中定义 cutlass_3x_gemm_sm90_fp8 模板，通过编译期 swap_ab 标志控制内核布局转置；实现 cutlass_scaled_mm_sm90_fp8 函数，基于 M 和 N 将问题路由到 8 个调度桶，每个桶指定 tile shape、cluster shape 和是否 swap-AB。新增 M32_N8192（TileN=32）和 M128_smallN（Tile<_64,_64,_128>）桶以覆盖小 M- 中 N 区域。
3. 清理原有内联调度：在 fp8_gemm_kernel.cu 中删除约 355 行旧的 SM90 内联调度代码（DeviceGemmFp8RowwiseSm90、prepare_sm90_fp8_args 等），将 SM90 入口从两路分支（bf16/f16）改为单一路径 cutlass_scaled_mm_sm90_fp8。
4. 添加基准和测试：新增 bench_fp8_gemm_swap_ab.py，使用 triton 的 perf_report 对 M∈[1,128]×11 种 (N,K) 组合进行性能扫描；扩展 test_fp8_gemm.py，添加 test_accuracy_sm90_swap_ab，覆盖 15 个 (M,N) 边界形状 ×3 种 K ×2 种输出 dtype ×2 种 bias 选项，共计 180 个测试 case，并包含 cluster 不对齐的额外形状。

关键文件：

- sgl-kernel/csrc/cutlass_extensions/epilogue/broadcast_load_epilogue_c3x.hpp（模块 epilogue 加载器；类别 source；类型 core-logic；符号 CtaTileShapeMNK, Element, StrideMNL）：核心 epilogue 广播实现，支持运行时的 per-tensor/per-channel 选择，是 swap-AB 调度能灵活处理不同量化粒度的基础。
- sgl-kernel/csrc/cutlass_extensions/epilogue/scaled_mm_epilogues_c3x.hpp（模块 epilogue 计算；类别 source；类型 dependency-wiring）：定义了 ScaledEpilogue 系列 epilogue 类，将 scale 和 bias 融合到 GEMM 中，直接服务于 swap-AB 调度器的输出计算。
- sgl-kernel/csrc/cutlass_extensions/gemm/fp8_gemm_sm90_dispatch.cuh（模块 调度桶；类别 other；类型 dependency-wiring）：swap-AB 调度的核心文件，定义了 8 个调度桶和 dispatch 函数，控制不同形状下是否使用转置、tile 和 cluster 参数。
- sgl-kernel/csrc/gemm/fp8_gemm_kernel.cu（模块 内核入口；类别 other；类型 dependency-wiring）：FP8 GEMM 的入口文件，原内联 SM90 调度被替换为单一路径调用，删除 355 行死代码。
- sgl-kernel/tests/test_fp8_gemm.py（模块 正确性测试；类别 test；类型 test-coverage；符号 test_accuracy_sm90_swap_ab）：新增的 swap-AB 正确性测试，覆盖所有调度桶边界和 cluster 不对齐形状，确保精度无退化。
- sgl-kernel/benchmark/bench_fp8_gemm_swap_ab.py（模块 性能基准；类别 source；类型 core-logic；符号 sglang_scaled_fp8_quant, benchmark）：针对 swap-AB 调度做精确性能基准，覆盖所有桶的 (N,K) 形状和 M 范围，支持与 main 对比。

关键符号：cutlass_scaled_mm_sm90_fp8, cutlass_3x_gemm_sm90_fp8, sglang_scaled_fp8_quant, benchmark, test_accuracy_sm90_swap_ab, sglang_scaled_fp8_quant

## 关键源码片段

### sgl-kernel/tests/test_fp8_gemm.py

新增的 swap-AB 正确性测试，覆盖所有调度桶边界和 cluster 不对齐形状，确保精度无退化。

```
# 新增的测试覆盖调度桶边界，包括 cluster 不对齐的 M orig 值
# 每个 (M, N) 对与多种 K、dtype、bias 组合，共 180 个 case
SM90_SWAP_AB_MN_SHAPES = [
    (1, 128),
    (1, 4096),
    (8, 1024),
    (8, 8192),
    (16, 1280),
    (16, 8192),
    (17, 128),
    (17, 4096),
    (32, 1024),
    (32, 8192),
    (64, 1280),
    (64, 8192),
    (65, 4096),
```

```
    (96, 4096),
    (128, 4096),
    # Cluster-misaligned M orig 在 M64_smallN 桶中（TileN=16, cluster_N=4）
    # M orig 为 17/20/33/48 时 grid_N=ceil(M_orig/16)∈{2,2,3,3},
    # 不是 cluster_N=4 的倍数，必须显式测试
    (20, 128),
    (20, 1024),
    (20, 1280),
    (33, 128),
    (33, 1024),
    (33, 1280),
    (48, 128),
    (48, 1024),
    (48, 1280),
]

@pytest.mark.parametrize(
    "shape_mn", SM90_SWAP_AB_MN_SHAPES, ids=lambda s: f"M{s[0]}_N{s[1]}"
)
@pytest.mark.parametrize("K", [2048, 4096, 8192])
@pytest.mark.parametrize("with_bias", [True, False])
@pytest.mark.parametrize("out_dtype", [torch.bfloat16, torch.float16])
def test_accuracy_sm90_swap_ab(shape_mn, K, with_bias, out_dtype):
    M, N = shape_mn
    # 使用与 main 相同的参考实现对比
    _test_accuracy_once(M, N, K, with_bias, out_dtype, "cuda")
```

## sgl-kernel/benchmark/bench_fp8_gemm_swap_ab.py

针对 swap-AB 调度做精确性能基准，覆盖所有桶的 (N,K) 形状和 M 范围，支持与 main 对比。

```
# 针对性更强的 swap-AB 基准，输出格式与 bench_fp8_gemm.py 一致
# 对比 main 时只需重定向输出后 diff

# 覆盖每个调度桶边界的关键 (N,K) 形状
NK_SHAPES = [
    (1024, 4096),
    (1024, 8192),
    (1280, 4096), # N == kNThreshold 边界
    (4096, 4096),
    (4096, 8192), # N == kM128NThreshold 边界
    (8192, 4096),
    (8192, 8192),
    (14336, 4096),
    (14336, 8192),
    (28672, 4096), # Llama-3 70B MLP up_proj N
    (28672, 8192),
]

# CI 中只测 M=1 以保持快速；完整运行覆盖所有 M 桶
```

```
batch_sizes = [1] if IS_CI else [1, 8, 16, 17, 32, 48, 64, 96, 128]

def sglang_scaled_fp8_quant(input, scale=None):
    """量化辅助：将输入转为 FP8，返回 (fp8_tensor, scale)"""
    fp8_type_ = torch.float8_e4m3fn
    output = torch.empty_like(input, device=input.device, dtype=fp8_type_)
    is_static = scale is not None
    if not is_static:
        scale = torch.zeros(1, device=input.device, dtype=torch.float32)
    per_tensor_quant_fp8(input, output, scale, is_static)
    return output, scale

@triton.testing.perf_report(
    triton.testing.Benchmark(
        x_names=["batch_size"],
        x_vals=batch_sizes,
        x_log=False,
        line_arg="provider",
        line_vals=["sglang-fp8-bf16", "sglang-fp8-fp16"],
        ylabel="GB/s",
        plot_name="fp8 swap-AB scaled matmul",
    )
)
def benchmark(batch_size, provider, N, K):
    M = batch_size
    a = torch.ones((M, K), device="cuda") * 5.0
    b = torch.ones((N, K), device="cuda") * 5.0
    scale_a = torch.randn((M,), device="cuda", dtype=torch.float32)
    scale_b = torch.randn((N,), device="cuda", dtype=torch.float32)
    dtype = torch.float16 if "fp16" in provider else torch.bfloat16
    a_fp8, scale_a_fp8 = sglang_scaled_fp8_quant(a, scale_a)
    b_fp8, scale_b_fp8 = sglang_scaled_fp8_quant(b, scale_b)
    b_fp8 = b_fp8.t()
    ms, _, _ = triton.testing.do_bench_cudagraph(
        lambda: sgl_scaled_mm(a_fp8, b_fp8, scale_a_fp8, scale_b_fp8, dtype, bias=None),
        quantiles=[0.5, 0.2, 0.8],
    )
    gbps = lambda ms: (2 * M * N * K + M * N) * a.element_size() * 1e-9 / (ms * 1e-3)
    return gbps(ms), 0, 0
```

# 评论区精华

- 调度桶完整性：gemini-code-assist 指出初始实现缺少 PR 描述中的 M32_N8192 和 M128_smallN 桶，作者 yuan-luo 确认并修复。
- 参数顺序潜在隐患：BBuf 指出调用者需手动根据 swap 标记调整 (a_scales, b_scales) 顺序，是“潜伏的脚枪”。yuan-luo 在 cutlass_gemm_caller_sm90_fp8_scaled 内部通过 if constexpr (Gemm::swap_ab) 自动交换，消除了风险。

- Cluster 配置有效性：gemini-code-assist 对 M16_N1280 桶的 ClusterShape<_1,_2,_1> 提出质疑，认为 swap 后 N 维度只有一个 CTA，集群为 2 无效。yuan-luo 回应实际测试通过且该桶性能优势明显，保持原样。
- 大 M 配置简化：kaixih 询问 M>128 统一使用一个 Cutlass3xGemmDefault 配置与 main 的分桶配置相比性能表现。yuan-luo 提供基准数据，16 个形状几何平均加速 1.06x，无回归。
- 死代码清理：gemini-code-assist 和 BBuf 均指出 fp8_gemm_kernel.cu 中残留原内联调度代码，yuan-luo 随后删除。
  - 调度桶完整性（M32_N8192 和 M128_smallN 缺失）(design): yuan-luo 确认并修复，加入对应分支。
  - 尺度参数顺序的潜伏脚枪 (correctness): yuan-luo 在 caller 内部使用 if constexpr (Gemm::swap_ab) 自动交换，消除隐患。
  - M16_N1280 桶的 Cluster 配置有效性 (correctness): yuan-luo 经实证测试确认该配置能正常启动并取得 23-29% 提速，保持原样。
  - 大 M 配置简化的性能验证 (performance): yuan-luo 提供基准数据，16 个形状几何平均加速 1.06x，确认无退化。
  - 死代码清理（原内联 SM90 调度残留）(style): yuan-luo 随后删除约 355 行死代码。

## 风险与影响

- 风险：主要风险在于部分形状（M=64，N 很大）存在 ≤10% 的性能退化，虽整体桶平均仍为正，但可能影响极端配置的尾巴延迟。swap-AB 的尺度参数内部交换已由编译期分支保障，但若未来新增桶漏掉 swap_ab 标记将导致精度错误。该变更仅作用于 SM90（Hopper）架构，不影响 CPU、AMD 或 Blackwell。DeepSeek V3/R1 使用块量化的独立内核，不受影响。测试覆盖了所有调度桶边界和 cluster 不对齐场景，并包含 180 个正确性 case。
- 影响：对用户透明，无需更改模型或 API。FP8 动态量化模型（Llama, Qwen, Mistral, DeepSeek 密集层等）在 NVIDIA Hopper GPU 上 decode 吞吐可提升 5.8-18.5%，尤其 batch size 较小（≤32）时收益明显。对 Prefill 和大 batch 无影响。sgl-kernel 的 Python 封装 fp8_scaled_mm 签名与行为不变，下游 srt 代码无需改动。
- 风险标记：特定形状微退化限制，仅 SM90 架构，参数顺序自动交换，测试覆盖全桶边界

## 关联脉络

- 暂无明显关联 PR