PR #30971 完整报告

sgl-project/sglang

[minimax-m3] fp8 attention GEMMs on SM100 (fp8_e4m3 KV + trtllm_mha)

合并时间：2026-08-01 09:39

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30971

执行摘要

- 一句话：M3 注意力 GEMM 全面 fp8 化，prefill 最高提速 66%
- 推荐动作：值得精读。三个 commit 相互独立、各有用处：升序 topk 契约、page128 支持、fp8 模式。重点关注：m3_fp8_attn_gemm_enabled 的推导式开关与 kill switch 设计、per-tensor scale 以 Optional[float] = None 端到端传递并归一化到 unit_scale() 的边界约定、q_scale 折入 sm_scale（同时作用 QK dot 与 sink logit）而 k_scale 不能碰 sink 项的数学约束，以及 warp-wise 升序排序与 fmha_sm100 early-exit 机制的契约互动。测试方法论（fp8 vs 反量化 bf16 参考的 parity 策略 + CUDA graph 位精确断言）也值得直接复用。

功能与动机

PR body 明确说明：MiniMax-M3 目前所有注意力 GEMM 都以 bf16 运行——--kv-cache-dtype fp8_e4m3 时 fp8 cache 在加载时被加宽回 bf16（widening-dequant 契约），lightning-indexer cache 也保持 bf16。而 Blackwell 上 MSA fmha_sm100 内核与 trtllm-gen 的 dense 内核都有原生 fp8 路径，可以在 fp8_e4m3 下端到端运行 sparse / MSA / indexer / dense 注意力 GEMM。该改动主要加速 prefill（32k 输入吞吐最高 +66%），并在相同 KV token 预算下释放约 20 GB/GPU 的 indexer 缓存内存，GSM8K 精度持平（0.970 vs 0.968），三个 commit 各自独立可用。

实现拆解

第一步：无 flag 的模式推导与 M3 覆盖入口

- python/sglang/srt/server_args.py 新增 m3_fp8_attn_gemm_enabled(args)，由三条件推导：kv_cache_dtype == "fp8_e4m3"、attention_backend == "trtllm_mha"、is_sm100_supported()，并被 SGLANG_DISABLE_M3_FP8_ATTN_GEMM 环境变量抑制。
- python/sglang/srt/arg_groups/overrides.py 的 _minimax_m3_overrides：当后端未显式指定且 KV 为 fp8_e4m3 时，将 SM100 默认注意力后端从 fa4 翻转为 trtllm_mha，并把 page_size 置为 128；fp8_e5m2 时保持 fa4 + 加宽 Triton 路径并输出警告（fmha_sm100 的变体查找会静默派发 e4m3 内核，故 e5m2 不得进入 MSA）；用户显式指定后端时绝不覆盖。_mla_backend_page_constraints 放行 page_size 128（flashinfer >= 0.6.12）。

第二步：内核前置改造

- minimax_decode_topk.cuh：block-id 输出契约从“前填充、无序”改为“严格升序”。初始实现是 smem 中 O(k^2) rank 置换，经 review 后演化为 deepseek_v4 topk_impl.cuh 风格的 warp-wise ballot+popc 排序（32x32 / 64x64 分支），输出 bit-identical、快 2-4%；kMaxTopK 从 32 提升到 64 以维持既有 topk == 64 测试契约；ROCm 条件宏统一为 USE_ROCM。
- python/sglang/srt/layers/attention/trtllm_mha_backend.py：TRTLLMHAAttnBackend 构造期对 page_size >= 128 校验 trtllm-gen 前置条件（GQA 且 q_heads/kv_head > 1、QK/V head dim 相等、page 为 2 的幂），不满足直接抛 ValueError 并提示改用 --page-size 64；XQA 布局（SM90/SM120）有原生 page-128 内核，无需校验。

第三步：fp8 模式的核心接线

- python/sglang/srt/layers/attention/minimax_sparse_backend.py：新增 _quant_q_fp8，在 KV store 与 DP trim 之后将 q/idx_q 量化为 fp8_e4m3；forward_extend 与 forward_decode 的 sparse 调用点把所有 per-tensor scale（q_scale / k_scale / v_scale / idx_q_scale / idx_k_scale / idx_v_scale）以 Optional[float] = None 透传给内核。
- use_msa 判定重构：原先“MSA 是 bf16/fp16 only、fp8 主 KV 必须留在 Triton 加宽路径”；现在允许 uniform fp8_e4m3（fp8 模式 + e4m3 主池）进入 MSA，_prepare_msa_decode_meta 构建 CG plan 时传 is_fp8=True。use_dense_sparse_decode 在 fp8 模式下被禁用（_dense_sparse_main_decode 尚无 fp8 处理，注释标注为 follow-up）。
- scale 语义：Triton wrapper 把 q_scale 折入 sm_scale（同时作用于 QK dot 与 sink logit，而 k_scale 不得触碰 sink 项）；MSA 原生传递 scales 给 fmha_sm100；所有 scale 在 kernel 启动边界经 unit_scale() 归一化一次。

第四步：缓存池与 HiCache 配套

- python/sglang/srt/mem_cache/memory_pool.py：MiniMaxSparseKVPool 新增 get_kv_cache_quant_method()（经由 main_pool 转发）；set_kv_buffer / set_index_kv_buffer / set_index_k_buffer / set_fused_kv_index_buffer 的 scale 参数默认从 1.0 改为 None，None 表示单位 scale，避免非 None 时 MHATokenToKVPool 的 in-place div_ 多开 kernel；fused 路径因 dtype 相等性检查不适用 fp8 时，回退到带 scale 的分步存储。
- python/sglang/srt/mem_cache/memory_pool_host.py：HostPoolGroup.backup_from_device_all_layer 按子池归一化 backup 索引——page_first + write-back JIT 池使用 CPU 索引，其余池保留 CUDA 索引（必要时 CPU 到 device 异步拷贝并 record_stream），修复混合 host 池（fp8 + 非 fp8 并存）下的 HiCache 备份设备错配（该 commit 标注 cherry-pick 自 f8c25079）。

第五步：测试与验证配套

- test_fp8_attn_gemm.py（新增 455 行）：fp8 内核 vs 反量化 bf16 参考的 parity 策略，覆盖 step-3 decode/prefill、非单位 k_scale / v_scale 语义、bf16 回归与 unit-scale 等价性。

- test_msa_fp8_parity.py（新增 286 行）：MSA fp8 vs Triton fp8 双路 parity、fp8-vs-bf16 误差界、CUDA graph capture/replay 位精确（含 JIT 预热流处理）。
- test_trtllm_mha.py：page128 在 XQA、trtllm-gen GQA decode/extend/CUDA-graph 的正向用例，以及 MHA 布局构造期拒绝用例（断言 ValueError 文案含 "dynamic tokens-per-page"）。
- test_model_overrides.py：test_m3_fp8_attn_gemm_resolution 覆盖模式推导全组合（e4m3 / auto / e5m2、kill switch、非 SM100、显式后端不被覆盖）与 page 约束链。
- 精度与性能验证：GSM8K A/B（仅由 kill switch 区分）0.970 vs 0.968；bench_one_batch TP4 下 8k / 32k 输入 prefill 分别 +19.8% / +65.9%，decode 基本持平（104.5 vs 108.5 tok/s）；同 KV budget 下释放约 20 GB/GPU。

关键文件：

- python/sglang/srt/layers/attention/minimax_sparse_backend.py（模块 稀疏注意力；类别 source；类型 core-logic；符号 _quant_q_fp8, MiniMaxSparseAttnBackend）：fp8 注意力 GEMM 模式的核心接入点：新增 _quant_q_fp8 量化 q/idx_q，重构 use_msa 判定以允许 uniform fp8_e4m3 进入 MSA，并将 per-tensor scale 端到端透传给 sparse/MSA 内核。
- python/sglang/kernels/jit/csrc/minimax/minimax_decode_topk.cuh（模块 内核算子；类别 infra；类型 core-logic；符号 minimax_decode_topk_block_kernel, TopKTrait）：升序 block-id 输出契约改造的核心内核文件，也是 review 讨论最密集处；O(k^2) 排序最终演化为 warp-wise ballot+popc 排序（kMaxTopK 32->64），并统一 ROCm 宏为 USE_ROCM。
- python/sglang/srt/mem_cache/memory_pool_host.py（模块 缓存池；类别 source；类型 core-logic；符号 _backup_uses_cpu_host_indices, _kernel_index_device, _normalize_backup_indices）：HiCache backup 索引按子池归一化，修复混合 host 池（fp8 + 非 fp8）并存时的索引设备错配；是 PR 内单独 cherry-pick 的 bugfix commit。
- python/sglang/srt/layers/attention/minimax_sparse_ops/tests/test_fp8_attn_gemm.py（模块 单元测试；类别 test；类型 test-coverage；符号 qdq, build_decode_inputs, build_prefill_inputs, run_step3_decode）：新增 455 行单测，覆盖 sparse decode/prefill 的 fp8 vs 反量化 bf16 参考 parity、非单位 scale 语义与 bf16 回归，是 fp8 内核正确性的主测试面。
- python/sglang/srt/layers/attention/minimax_sparse_ops/tests/test_msa_fp8_parity.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_msa_fp8_decode_vs_triton_fp8, test_msa_fp8_decode_scales, test_msa_fp8_decode_capture_replay_bitexact）：新增 MSA fmha_sm100 fp8 与 Triton fp8 的 parity 测试，含 CUDA graph capture/replay 位精确验证，弥补 fmha_sm100 无上游 fp8 测试的空白。
- python/sglang/srt/layers/attention/trtllm_mha_backend.py（模块 注意力后端；类别 source；类型 dependency-wiring；符号 TRTLLMHAAttnBackend.init）：trtllm_mha 支持 page_size 128 的构造期校验：把不支持的 GQA 布局从 CUDA graph 捕获期的内核缺失错误提前到启动即 raise。

- python/sglang/srt/server_args.py（模块 服务参数；类别 source；类型 core-logic；符号 m3_fp8_attn_gemm_enabled）：新增 m3_fp8_attn_gemm_enabled()，是无 flag 推导式开关的权威定义，被 minimax_sparse_backend 与 overrides 共同引用。
- python/sglang/srt/arg_groups/overrides.py（模块 模型覆盖；类别 source；类型 core-logic；符号 _minimax_m3_overrides, _mla_backend_page_constraints）：_minimax_m3_overrides 在 fp8_e4m3 + SM100 下把默认后端从 fa4 翻转为 trtllm_mha 并置 page_size 128，且对 fp8_e5m2 告警；_mla_backend_page_constraints 放行 128。
- test/registered/attention/unittests/dense/test_trtllm_mha.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestTRTLLMMHAPage128TrtllmGen, test_page128_decode_cases, test_page128_extend_cases, test_page128_cuda_graph_decode_cases）：新增 page128 在 XQA 与 trtllm-gen（GQA）下的 decode/extend/CUDA graph 用例，以及 MHA 布局构造期拒绝用例。
- python/sglang/srt/layers/attention/minimax_sparse_ops/msa.py（模块 MSA 内核；类别 infra；类型 infrastructure；符号 _check_msa_dtypes）：fp8 模式下 MSA 的 dtype 校验与规模断言扩展（_check_msa_dtypes），支撑 uniform-e4m3 MSA 路径。

关键符号：m3_fp8_attn_gemm_enabled, _quant_q_fp8, _normalize_backup_indices, _backup_uses_cpu_host_indices, _kernel_index_device, get_kv_cache_quant_method, _minimax_m3_overrides, _mla_backend_page_constraints, _check_msa_dtypes

关键源码片段

python/sglang/srt/layers/attention/minimax_sparse_backend.py

fp8 注意力 GEMM 模式的核心接入点：新增 _quant_q_fp8 量化 q/idx_q，重构 use_msa 判定以允许 uniform fp8_e4m3 进入 MSA，并将 per-tensor scale 端到端透传给 sparse/MSA 内核。

```
# fp8 注意力 GEMM 模式的 q 量化入口。
# 约定与 KV 池一致：fp8 张量存储 value/scale，注意力内核再把 logits 乘回 scale
#  （`None` 表示单位 scale，不触发额外除法）。
def _quant_q_fp8(q: torch.Tensor, q_scale: Optional[float]) -> torch.Tensor:
    if q_scale is not None:
        q = q / q_scale
    return q.to(torch.float8_e4m3fn)


class MiniMaxSparseAttnBackend(AttentionBackend):
    def __init__(self, runner: ModelRunner):
        # 模式由配置推导：fp8_e4m3 KV + trtllm_mha + SM100，无 opt-in flag
        self.fp8_attn_gemm = m3_fp8_attn_gemm_enabled(runner.server_args)
        if self.fp8_attn_gemm:
            assert self.kv_pool.main_pool.dtype == torch.float8_e4m3fn, (
                "fp8 attn-GEMM mode requires an fp8_e4m3fn main KV pool, got "
                f"{self.kv_pool.main_pool.dtype}"
            )
        # ... 模型配置解析、sparse 块参数与 topk 设置 ...
```

```
        # MSA（fmha_sm100）在 fp8 模式下必须以 uniform fp8_e4m3 运行：
        # bf16 q + fp8 K/V 的组合不被其 uniform-dtype 内核支持；e5m2 会被
        # 变体查找静默派发成 e4m3 内核，因此永不允许进入 MSA。
        _main_kv_is_fp8 = self.kv_pool.main_pool.dtype in (
            torch.float8_e4m3fn,
            torch.float8_e5m2,
        )
        _msa_fp8_ok = (
            self.fp8_attn_gemm and self.kv_pool.main_pool.dtype == torch.float8_e4m3fn
        )
        self.use_msa = (
            not envs.SGLANG_DISABLE_MSA.get()
            and msa_available()
            and self.block_size_k == 128
            and self.kv_pool.page_size == self.block_size_k
            and self.topk_blocks in (4, 8, 16, 32)
            and (not _main_kv_is_fp8 or _msa_fp8_ok)
        )
```

python/sglang/srt/mem_cache/memory_pool_host.py

HiCache backup 索引按子池归一化，修复混合 host 池（fp8 + 非 fp8）并存时的索引设备错配；是 PR 内单独 cherry-pick 的 bugfix commit。

```
    def _backup_uses_cpu_host_indices(self, host_pool, io_backend) -> bool:
        # 分页优先（page_first）且支持 write-back JIT 的池，backup 阶段使用
        # CPU 侧索引即可；其余子池需要 CUDA 索引参与 kernel 写回。
        return (
            io_backend == "kernel"
            and getattr(host_pool, "layout", None) == "page_first"
            and getattr(host_pool, "can_use_write_back_jit", False)
        )


    def _normalize_backup_indices(self, entry, host_indices, device_indices, io_backend):
        # 非 kernel 后端不改变索引位置
        if io_backend != "kernel":
            return host_indices, device_indices

        # page_first + JIT 池：CUDA 索引降回 CPU，避免 kernel 路径误用设备指针
        if self._backup_uses_cpu_host_indices(entry.host_pool, io_backend):
            if host_indices.is_cuda:
                host_indices = host_indices.cpu()
            return host_indices, device_indices

        # 普通池需要设备索引：若仅有 CPU 索引则异步拷到目标设备，并注册
        # stream，保证异步拷贝在 CUDA 图 / 流语义下的生命周期正确
        if not host_indices.is_cuda:
            target_device = self._kernel_index_device(entry, device_indices)
            if target_device is not None:
```

```
            host_indices = host_indices.to(target_device, non_blocking=True)
            if host_indices.is_cuda:
                host_indices.record_stream(
                    torch.cuda.current_stream(host_indices.device)
                )
        return host_indices, device_indices


    def backup_from_device_all_layer(self, device_pool, host_indices, device_indices, io_backend,
    pool_transfers=None):
        # 主 KV（anchor）与附加池（pool_transfers）各自归一化索引后再写回，
        # 修复混合 host 池（fp8 池 + 非 fp8 池）并存时的索引设备错配
        anchor_host_indices, anchor_device_indices = self._normalize_backup_indices(
            self.anchor_entry, host_indices, device_indices, io_backend
        )
        self.anchor_entry.host_pool.backup_from_device_all_layer(
            self.anchor_entry.device_pool,
            anchor_host_indices,
            anchor_device_indices,
            io_backend,
        )
        for transfer in pool_transfers or []:
            entry = self.entry_map.get(transfer.name)
            if entry is None or transfer.host_indices is None:
                continue
            transfer_host_indices, transfer_device_indices = (
                self._normalize_backup_indices(
                    entry, transfer.host_indices, transfer.device_indices, io_backend
                )
            )
            entry.host_pool.backup_from_device_all_layer(
                entry.device_pool,
                transfer_host_indices,
                transfer_device_indices,
                io_backend,
            )
```

评论区精华

Review 讨论聚焦在 minimax_decode_topk.cuh 的升序排序实现：

1. DarkSharpness 指出初始 O(k^2) 排序不够高效，建议复用 deepseek_v4 topk_impl.cuh 的 warp-wise 排序；alumkal 采纳后单独消融验证输出 bit-identical、内核时间快 2-4%，并指出剩余差距主要来自向 gmem 的有序散射而非 rank 计算。
2. 对“能否去掉排序、直接 padding -1”的探讨：alumkal 论证排序分支（num_blocks > topk）与 -1 padding 分支（num_blocks <= topk）路径不相交，且升序是 fmha_sm100 early-exit 掩码契约（valid_blks 反向扫描 + 掩码循环提前退出）与 sparse prefill 的共同硬需求；实测排序仅约 0.08 us/launch，保留排序换取干净契约。

3. DarkSharpness 质疑 CTA 512 线程（16 warps）下 warp 排序的 stride 正确性：alumkal 解释 warps 切分 targets（warp w 负责 t = w, w+16...）、lanes 持 candidates 的 ballot+popc 设计、__ballot_sync(0xffffffff) 收敛性，并以 topk ∈ {16, 32, 33, 48, 64} 与 CPU 排序参考逐位验证通过。
4. DarkSharpness 指出 ROCm 条件应统一使用 build 定义的 USE_ROCM 宏而非 hipcc 内部宏 __HIP_PLATFORM_AMD__，已采纳。

- minimax_decode_topk 的升序排序实现效率与 warp-wise 排序建议 (performance): 已采纳 warp-wise ballot+popc 排序（32x32 / 64x64 分支），保留升序契约。
- 是否可以去掉排序、直接 padding -1 (design): 保留升序排序以维持干净契约。
- CTA 512 线程（16 warps）下 warp 排序 stride 的正确性 (correctness): 设计 sound，验证通过。
- ROCm 条件编译宏统一为 USE_ROCM (style): 已采纳，切换为 #ifdef USE_ROCM，并补充 wave64 场景的宽度 -32 __shfl_up 与 __ballot-free count_lt 分支。

风险与影响

- 风险：
  1. 无 flag 自动激活改变默认数值路径：fp8 模式一旦满足 kv_cache_dtype fp8_e4m3 + trtllm_mha + SM100 即生效，不依赖用户显式选择；GSM8K parity 背书了该决策，但长尾任务 / 其他精度敏感场景未被覆盖，依赖 kill switch 兜底。
  2. page128 行为变更：显式 --page-size 128 + trtllm_mha + 非 GQA 模型从“回退 64 + 警告”变为构造期 raise（ValueError），可能影响依赖旧行为的启动脚本。
  3. 依赖与硬件前提：page128 依赖 flashinfer >= 0.6.12 的 trtllm-gen 动态 tokens-per-page 内核；MSA fp8 首次前向会 JIT 编译 fmha_sm100 fp8 变体，冷缓存可耗时数分钟且跨 TP rank 串行，首 token 延迟可能显著放大。
  4. 功能裁剪：fp8 模式下 use_dense_sparse_decode 被禁用（_dense_sparse_main_decode 尚无 fp8 处理，注释明确标注 follow-up）。
  5. HiCache 回归面：memory_pool_host.py 的索引归一化改动覆盖 backup_from_device_all_layer 主 KV 与附加池两条路径，涉及 kernel/CPU 索引设备切换与 record_stream 生命周期，回归面集中于冷备与页缓存回流。
  6. 精度风险：fp8 PV MMA（P 量化到 e4m3）是主要误差源，测试容差按 6e-2 ~ 1e-1 设置，与 bf16 参考的偏差在可接受范围内，但未覆盖所有随机种子场景。 - 影响：用户影响：MiniMax-M3 + SM100 部署默认获得 prefill 性能提升（8k / 32k 输入分别 +19.8% / +65.9%）与约 20 GB/GPU 显存释放；可通过 SGLANG_DISABLE_M3_FP8_ATTN_GEMM 精确退回旧数值路径。非 M3 模型与非 SM100 硬件不受影响。 系统影响：改动横跨 server_args / overrides、trtllm_mha 后端、稀疏注意力内核、KV 池与 HiCache，无调度器改动；模式推导集中在一处（m3_fp8_attn_gemm_enabled），便于审计。 团队影响：为模型专属优化确立了“配置推导、无新 flag + kill switch”的范式，后续类似优化（如 Inkling 短卷积后端、统一内存后端）可复用该模式。

- 风险标记：无 flag 自动激活（默认行为变更），核心注意力路径变更，依赖 flashinfer >= 0.6.12，首次 JIT 编译分钟级延迟，构造期拒绝行为变更（page128 非 GQA），HiCache 索引归一化回归面

关联脉络

- PR #33023 feat(inkling): migrate short convs onto the ShortConv attention backend: 同属注意力后端抽象层改造：Inkling 短卷积迁移到注意力后端，与本 PR 的 minimax_sparse_backend 改动共同推进“模型专属算子后端化”方向，且都涉及 speculative decode 与元数据契约。
- PR #33046 [unified-memory] Support fa3, the default MLA backend on pre-Blackwell hosts: 同属 fp8/MLA + 注意力后端组合推导的配置模式：fa3 支持与统一内存默认配置修复，与本 PR 的 server_args/overrides 后端选择逻辑交互。
- PR #32972 [unified-memory] Let Kimi-Linear use the paged MLA attention backends: 验证了“fp8 KV + 特定 attention 后端”组合支持的一般化路径，与本 PR 的 fp8_e4m3 + trtllm_mha 组合及 page_size 约束放行模式一致。
- PR #33013 config: read resolved config via namespace accessors: 覆盖 server_args.py / overrides.py 的配置读取迁移，与本 PR 的 m3_fp8_attn_gemm_enabled 推导入口和 _minimax_m3_overrides 所在文件重叠，属于同一配置体系演进。