

PR #39646 完整报告

sgl-project/sglang

dsv4.1: standalone kernels and Python wrappers

合并时间: 2026-09-16 13:31

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39646>

执行摘要

- 一句话: 抽出 DSV4.1 独立内核与 Python 封装, 作为 PR 栈首层合入
- 推荐动作: 值得精读, 尤其是准备接 #38798 / #39648 的工程师与负责 DSV4.1 内核维护的同学。
- 优先看三处设计决策: small_gemm_bf16.py 的 MAX_M 策略上限与 can_use_* 契约 (调用方负责回退, 内核只做「形状正确即最快」); mxfp8_epilogue.py 的 UE8M0 舍入 + 128x4 swizzle 偏移 + padding 槽位裁剪策略 (多 CTA 写集合不相交的隐含证明); q_rope_store.py 的 prefill/decode 分支与 int64 偏移、PDL 开关。
- 阅读时把每个 assert 当成调用契约而非防御式编程: 这些断言定义了调度层必须满足的形状、stride 与设备前提。
- 跟进项建议: (a) 按 BBuf 的意见把 candidate_blocks.py / indexer_postprocess.py 的独立覆盖补进本层或明确移交 #39648; (b) 为 engram_gather 补 owned / unowned ID 的用例; (c) 刷新 PR 描述中已过期的 byte-identical 声明与文件统计, 避免后续审计引用失效结论; (d) 如计划支持非 Blackwell 平台, 尽早确认 can_use_* 谓词在所有调用点的使用方式。

功能与动机

PR 描述明确写出动机: Extract standalone DeepSeek V4.1 kernels and Python wrappers from #38798 as the first layer of a linear PR stack, 即把 33 个文件、+3,494 行的 DSV4.1 功能拆成可独立评审、可独立验证的层——内核与封装层放本 PR, Keep model dispatch, scheduler/cache ownership, and runtime integration in #38798, 并把 Top-K 实现隔离到 #39648。作者给出的验证标准是: 23 个内核 / 封装文件与来源快照 01d34d7b 逐字节一致、Python 函数 AST 等价、抽取出的测试在单台 GB300 上全绿。评审者 BBuf 又补了一层动机: 分层必须让每层自带覆盖, 否则 this leaves the standalone layer without independent coverage and makes a regression here appear as a Top-k regression later in the stack。

实现拆解

1. 内核按算子族落位: 在 `python/sglang/kernels/ops/` 下新增五组目录——`embeddings/` (`engram_hash.py`、`engram_gather.py`、`engram_gate.py`)、`layernorm/` (`hc_combine_norm.py`、`mxfp8_epilogue.py`、`mhc_post_split_h.py`、`rmsnorm_fp32.py`)、`gemm/` (`small_gemm_bf16.py`, 并在 `gemm/__init__.py` 重新导出)、`speculative/` (

row_argmax.py)、attention/dsv4/ (candidate_blocks.py、indexer_postprocess.py、flashmla_sched_meta.py、q_rope_store.py、wo_a_bf16.py) ; C++ 侧新增 jit/csrc/deepseek_v4/flashmla_sched_meta.cuh 与 jit/csrc/gemm/small_gemm_bf16.cuh, 并把原 gemm/utils.cuh 改名为 dot_product.cuh 复用归约原语。

2. 统一封装契约: JIT 侧用 @cache_once + load_jit(...) 装载, trait 字符串决定实例化哪个 C++ 模板, make_cpp_args(is_arch_support_pdl()) 决定是否编入 PDL 支持; Torch 侧用 @register_custom_op(op_name=..., mutates_args=["out"]) 注册, 使原地写出的 buffer 在 CUDA Graph / torch.compile 下被正确追踪; 对外暴露 can_use_* 谓词与 MAX_M = 32 的策略上限, _arch_supported() 要求非 HIP 且 get_jit_cuda_arch().major >= 9, 形状不符时内核直接抛错而不是静默回退, 回退责任显式交给调用方。
3. 数值一致性契约 (本层最核心的隐含约定): ue8m0_scale 复刻 FlashInfer 的 UE8M0 正向舍入 (含次正规、指数封顶 254), mxfp8_epilogue 的 scale 走 128x4 swizzle 偏移, rmsnorm_mxfp8 用额外 CTA 把 padding scale 槽位清零, 目标是 rmsnorm + mxfp8_quantize(..., is_sf_swizzled_layout=True) 逐位相同; wo_a_bf16.py 的 _wo_a_reduce_quant 先 BF16 舍入再量化并在同一 kernel 清零 padding scale; q_rope_store.py 用 tl.fma 固定 even/odd 运算顺序以匹配 deepseek_rope_kernel; hc_combine_norm.py 先 combine 再显式 to(tl.bfloat16).to(tl.float32) 复现未融合路径的舍入; candidate_blocks.py 保留 torch.topk 的 tie 行为, 只把成员资格下推给 mask kernel; engram_hash.py 文档串声明算术必须与 EngramHasher 的 torch 路径逐位一致。
4. 流水线与形状分支配套: candidate_row_lens 与 q_rope_store 在支持 PDL 的架构上使用 tl.extra.cuda.gdc_wait() / gdc_launch_dependents() 并自动附带 launch_pdl=True, 说明这些 kernel 被设计为与前序 kernel 重叠; q_rope_store 另设 prefill 分支 (rows >= 4096 and heads == 16), 把偏移提升到 int64 以容纳超过 2 GiB 的 padding 输出。
5. 测试与 CI 配套: 携带 test/registered/kernel/attention/test_flashmla_sched_meta.py、test/registered/kernel/layernorm/test_mxfp8_epilogue.py、test/registered/kernel/attention/test_dsv4_q_rope_store.py, 均以 register_cuda_ci(est_time=..., stage="base-b-kernel-unit", runner_config=...) 挂到 kernel 单测流水线, 并用 pytestmark 限定 Blackwell + CUDA。测试策略统一为「对齐参考实现»: FlashMLA 调度元数据只比较前 7 个字 (DEFINED = slice(0, 7), 尾字 _pad 参考实现从不写入), attention 输出用 view(torch.int16) / view(torch.int32) 做位相等比较; MXFP8 用 FlashInfer 的 mxfp8_quantize 与 flashinfer.norm.rmsnorm 做 bitwise 基准; Q RoPE store 用 fused_rope_inplace 做参照并断言 padding 区未被改写。
6. 评审驱动的收尾改造: 22 个 commit 中有大量合并同类文件、改名与补校验——wo_a_bf16_gemv 并入 wo_a_bf16、hc_combine_norm_mxfp8 并入 hc_combine_norm、n128k512/n32k5120 并入 small_gemm_bf16、decoding_sched_meta 改名 flashmla_sched_meta、fast_row_argmax 改名 row_argmax、gemm/utils.cuh 改名 dot_product.cuh, 并针对 BBuf 的契约校验意见补了 commit 101686f validate engram_gather arguments。这些收尾使「逐字节一致」的声明不再成立, 也是评审随后要求刷新 PR 描述的直接原因。

关键文件:

- `python/sglang/kernels/ops/embeddings/engram_hash.py` (模块 哈希嵌入; 类别 `infra`; 类型 `infrastructure`; 符号 `_engram_commit_history_kernel`, `engram_commit_history`, `_engram_hash_kernel`, `_launch_hash_kernel`) : 本 PR 体量最大的新增实现 (+363), 承载 DSV4.1 engram 的 n-gram 哈希、历史提交 (commit) 与屏蔽 (cummax block 传播) 语义, 并显式声明算术必须与 `sglang.srt.layers.engram.EngramHasher` 的 torch 路径逐位一致, 是后续模型层最依赖的数值契约之一。
- `python/sglang/kernels/ops/gemm/small_gemm_bf16.py` (模块 矩阵乘法; 类别 `infra`; 类型 `infrastructure`; 符号 `_jit_small_gemm_module`, `_n128k512_gemm_custom_op`, `_n32k5120_gemm_custom_op`, `_arch_supported`) : 定义了两个固定形状 decode GEMM 的接口契约: trait 化 JIT 装载、custom op 原地写 out、`can_use_*` 谓词 + `MAX_M = 32` 策略上限、`_arch_supported()` 的架构门禁。后续 #38798 的模型层就是按这套约定做 dispatch 与回退。
- `python/sglang/kernels/ops/layernorm/mx_fp8_epilogue.py` (模块 量化后处理; 类别 `infra`; 类型 `infrastructure`; 符号 `ue8m0_scale`, `mx_fp8_epilogue`, `_rmsnorm_mx_fp8_kernel`, `rmsnorm_mx_fp8`) : MXFP8 数值契约的单一来源: `ue8m0_scale` 复刻 FlashInfer 的 UE8M0 正向舍入, `mx_fp8_epilogue` 负责 128x4 swizzle 布局的 scale 写入, `rmsnorm_mx_fp8` 用额外 CTA 清除 padding 槽位。`hc_combine_norm`、`wo_a_bf16` 都复用它, 是量化侧一致性的公共基座。
- `python/sglang/kernels/ops/attention/dsv4/candidate_blocks.py` (模块 候选块筛选; 类别 `infra`; 类型 `infrastructure`; 符号 `_maximum_with_nan`, `_candidate_scores_kernel`, `_candidate_mask_kernel`, `_publish_candidate_mask_kernel`) : DSV4.1 索引器 (indexer) 稀疏选择的后处理核心: 把页内块打分、可见性 mask 与已发布 mask 合并到一次扫描, 并保留 `torch.topk` 的 tie 行为; `candidate_row_lens` 还示范了 PDL 依赖写法。BBuf 的全局评论正是围绕它与 `indexer_postprocess` 缺少独立测试展开。
- `python/sglang/kernels/ops/layernorm/hc_combine_norm.py` (模块 归一化层; 类别 `infra`; 类型 `infrastructure`; 符号 `_hc_combine_norm`, `_hc_combine_norm_prefill`, `hc_combine_norm`, `_hc_combine_norm_mx_fp8_kernel`) : 把四条超连接流的加权合并与 RMSNorm 融合, 并复现未融合路径「先存 BF16 再进 RMSNorm」的舍入; `decode / prefill` 走两套 kernel, `hc_combine_norm_mx_fp8` 再复用 `mx_fp8_epilogue`。是数值一致性与 CTA 划分策略都值得看的例子。
- `python/sglang/kernels/ops/attention/dsv4/wo_a_bf16.py` (模块 输出投影; 类别 `infra`; 类型 `infrastructure`; 符号 `_wo_a_bf16_gemv_kernel`, `wo_a_bf16_gemv`, `_wo_a_partial`, `_wo_a_reduce`) : WO-A 投影的三个变体 (单 token GEMV、split-K 小 batch GEMM、带 MXFP8 epilogue 的融合版), 其中 `_wo_a_reduce_quant` 在同一 kernel 内完成 FP32 归约、BF16 舍入、E4M3 量化与 padding scale 清零, 是「部分和缓冲复用 + 量化融合」的代表实现。
- `python/sglang/kernels/ops/attention/dsv4/q_rope_store.py` (模块 位置编码; 类别 `infra`; 类型 `infrastructure`; 符号 `_q_rope_store`, `_q_rope_store_prefill`, `q_rope_store`) : 把 Q 的 64 维 RoPE 与写入稀疏 attention 所需 padding 缓冲合并成一次 kernel, 用 `tl.fma` 固定运算顺序以对齐 `deepseek_rope_kernel`; `decode` 支持 PDL, `prefill` 分支用 `int64` 偏移处理超过 2 GiB 的输出。

- `python/sglang/kernels/ops/embeddings/engram_gather.py` (模块 哈希嵌入; 类别 `infra`; 类型 `infrastructure`; 符号 `_engram_gather_kernel`, `engram_gather`): 唯一被逐行评审过的文件: BBuf 在这里指出内核按 `ids.numel()` 线性化、写出 `row * DIM` 个元素却没有任何输入契约校验, 可能静默错读写; 随后 `commit 101686f` 补上参数校验, 是「`wrapper` 必须显式校验」这条评审规则的来源。
- `test/registered/kernel/attention/test_flashmla_sched_meta.py` (模块 调度元数据; 类别 `test`; 类型 `test-coverage`; 符号 `_cache`, `_call`, `_ours`, `_lengths`): 本层最具代表性的测试: 用 FlashMLA 自身的 `tile_scheduler_metadata` 作为参考, 只比较前 7 个已定义字 (`_pad` 字参考实现不写), 并在 FlashMLA 不做 `split-KV` 的形状上显式 `skip`; `extra cache` 场景还把内核产出的元数据塞回 FlashMLA 验证 `out/lse` 位相等。
- `test/registered/kernel/layernorm/test_mxfp8_epilogue.py` (模块 量化后处理; 类别 `test`; 类型 `test-coverage`; 符号 `test_bitwise_identical_to_norm_then_quantize`, `check`, `test_dynamic_graph`, `test_wo_a_partial_quant_matches_quantize`): 量化契约的回归防线: 用 FlashInfer 的 `mxfp8_quantize` 与 `flashinfer.norm.rmsnorm` 作为基准, 逐位比较 `y / q / swizzle` 后的 `sf`, 并验证 `padding scale` 全零; CUDAGraph replay 用例专门覆盖「`scale padding` 也必须重算」这一容易漏掉的场景。
- `python/sglang/kernels/ops/attention/dsv4/flashmla_sched_meta.py` (模块 调度元数据; 类别 `infra`; 类型 `infrastructure`; 符号 `_jit_flashmla_sched_meta_module`, `flashmla_sched_meta`): 把 FlashMLA 的 `tile scheduler` 元数据计算搬进 SGLang 自己维护的 JIT 模块 (配合 `csrc/deepseek_v4/flashmla_sched_meta.cuh`), 是 DSV4.1 稀疏 `decode` 与前缀缓存组合 (`extra cache`) 的关键路径; 测试与实现同名, 便于对照阅读。

关键符号: `engram_hash_ids_and_commit`, `engram_hash_ids`, `engram_commit_history`, `_engram_hash_kernel`, `engram_gather`, `fused_engram_gate`, `candidate_block_logits`, `candidate_row_lens`, `hc_combine_norm`, `hc_combine_norm_mxfp8`, `ue8m0_scale`, `mxfp8_epilogue`, `rmsnorm_mxfp8`, `wo_a_bf16_small_batch`, `wo_a_bf16_small_batch_mxfp8`, `_wo_a_reduce_quant`, `n128k512_gemm_bf16`, `n32k5120_gemm_bf16`, `can_use_n128k512_gemm`, `can_use_n32k5120_gemm`, `q_rope_store`, `flashmla_sched_meta`, `row_argmax`, `filter_topk_pages`, `mhc_post_split_h`, `rmsnorm_fp32`

关键源码片段

`python/sglang/kernels/ops/gemm/small_gemm_bf16.py`

定义了两个固定形状 `decode GEMM` 的接口契约: `trait` 化 JIT 装载、`custom op` 原地写 `out`、`can_use_*` 谓词 + `MAX_M = 32` 策略上限、`_arch_supported()` 的架构门禁。后续 #38798 的模型层就是按这套约定做 `dispatch` 与回退。

```
"""两个固定形状的 decode bf16 GEMM: x[m, 512] @ w[128, 512].T 与
x[m, 5120] @ w[32, 5120].T。与 tiny_gemm_bf16 共用 dot_product_vec 的归约顺序,
因此同一行的结果不随 batch 组成变化, 可逐位复现。
"""
```

```
# MAX_M 只是调用方的策略上限: 内核本身按 blockIdx.y 切行, 任意 m 都能算;
# 超过 32 行就让调用方走通用 GEMM, 避免小 GEMM 在大 batch 下失去收益。
```

```
MAX_M: int = 32
```

```
@cache_once
```

```
# trait 决定实例化 SmallGemmKernel<N128K512Trait> 还是 <N32K5120Trait>;
```

```
# 是否编入 PDL 支持由 make_cpp_args(is_arch_support_pdl()) 在编译期决定。
```

```
def _jit_small_gemm_module(trait: str) -> Module:
```

```
    args = make_cpp_args(is_arch_support_pdl())
```

```
    return load_jit(
```

```
        f"small_gemm_bf16_{trait}",
```

```
        *args,
```

```
        cuda_files=["gemm/small_gemm_bf16.cuh"],
```

```
        cuda_wrappers=[("run", f"SmallGemmKernel<{trait}, {args}>::run")],
```

```
        extra_cuda_cflags=["-O3"],
```

```
    )
```

```
# 通过 custom op 注册并声明 out 为原地写参数，保证 CUDA Graph / torch.compile 下
```

```
# 输出 buffer 的别名关系被正确追踪。
```

```
@register_custom_op(op_name="n128k512_gemm_bf16", mutates_args=["out"])
```

```
def _n128k512_gemm_custom_op(x, w, out) -> None:
```

```
    _jit_small_gemm_module("N128K512Trait").run(x, w, out)
```

```
@cache_once
```

```
def _arch_supported() -> bool:
```

```
    # 只支持 Blackwell 及更新架构的 CUDA 路径；HIP (ROCm) 直接判负。
```

```
    return not is_hip_runtime() and get_jit_cuda_arch().major >= 9
```

```
def can_use_n128k512_gemm(n: int, k: int, m: int, max_m: int = MAX_M) -> bool:
```

```
    """调用方先用它判形状，False 时退回通用 GEMM；内核自身不隐式回退。"""
```

```
    return n == 128 and k == 512 and 1 <= m <= max_m and _arch_supported()
```

```
@debug_kernel_api
```

```
def n128k512_gemm_bf16(x, w, out=None) -> torch.Tensor:
```

```
    """等价于 torch.nn.functional.linear(x, w)，w 固定为 [128, 512]。
```

```
    x 允许是行切片视图 (stride(1) == 1)，但每行必须 32 字节对齐，因为行是按  
    整向量加载的；形状不符时直接抛错，而不是静默回退到通用实现。
```

```
    """
```

```
    m = x.shape[0]
```

```
    if out is None:
```

```
        out = torch.empty((m, 128), dtype=torch.bfloat16, device=x.device)
```

```
    if m == 0: # m == 0 不启动内核，避免空 grid 的启动开销
```

```
        return out
```

```
    _n128k512_gemm_custom_op(x, w, out)
```

```
    return out
```

python/sglang/kernels/ops/layernorm/mxftp8_epilogue.py

MXFP8 数值契约的单一来源: `ue8m0_scale` 复刻 FlashInfer 的 UE8M0 正向舍入, `mxftp8_epilogue` 负责 128x4 swizzle 布局的 scale 写入, `rmsnorm_mxftp8` 用额外 CTA 清除 padding 槽位。hc_combine_norm、wo_a_bf16 都复用它, 是量化侧一致性的公共基座。

```
@triton.jit
```

```
def ue8m0_scale(amax):
```

```
    # 复刻 FlashInfer 的 UE8M0 正向舍入 (舍次正规) : 先按 1/448 归一化, 再根据
    # 指数与尾数决定是否进位, 指数封顶 254 (留下 2**127) ; 0 与负值映射为 0。
    normalized = amax * (1.0 / 448.0)
    bits = normalized.to(tl.int32, bitcast=True)
    exponent = (bits >> 23) & 255
    mantissa = bits & 0x7FFFFF
    bump = (mantissa != 0) & ~((exponent == 0) & (mantissa <= 0x400000))
    sf = tl.where(normalized <= 0, 0, tl.minimum(exponent + bump.to(tl.int32), 254))
    inv = tl.where(sf == 0, 0, ((254 - sf) << 23)).to(tl.float32, bitcast=True)
    return sf, inv
```

```
@triton.jit
```

```
def mxftp8_epilogue(y, row, Q, S, K: tl.constexpr, BLOCK: tl.constexpr,
                    GROUPS: tl.constexpr, g_lo, g_hi):
    # 只写 [g_lo, g_hi) 内的 group, 使一行可以拆给多个 CTA; 每个 group 仍然
    # 自己算 amax 与 scale, 因此结果与 standalone mxftp8_quantize 逐位一致。
    GP: tl.constexpr = BLOCK // 32
    g = tl.arange(0, GP)
    gmask = (g < GROUPS) & (g >= g_lo) & (g < g_hi)
    e = tl.arange(0, 32)
    idx = g[:, None] * 32 + e[None, :]
    v = tl.reshape(y.to(tl.float32), (GP, 32))
    amax = tl.max(tl.abs(v), 1)
    sf, scale = ue8m0_scale(amax)
    # 量化边界与 FlashInfer 一致: 先乘 scale 再夹到 e4m3 的 [-448, 448]
    q = tl.minimum(tl.maximum(v * scale[:, None], -448.0), 448.0).to(tl.float8e4nv)
    tl.store(Q + row * K + idx, q, gmask[:, None])
    # FlashInfer 的 128x4 swizzle:
    # off = (g // 4) * 512 + ((row % 32) * 4 + ((row // 32) % 4)) * 4 + (g % 4)
    off = (g // 4) * 512 + ((row % 32) * 4 + ((row // 32) % 4)) * 4 + (g % 4)
    tl.store(S + off, sf.to(tl.uint8), gmask)
```

```
@triton.jit
```

```
def _rmsnorm_mxftp8_kernel(X, W, Y, Q, S, SX: tl.constexpr, M: tl.constexpr,
                           K: tl.constexpr, EPS: tl.constexpr,
                           BLOCK: tl.constexpr, GROUPS: tl.constexpr):
    row = tl.program_id(0)
    if row < M:
        h = tl.arange(0, BLOCK)
```

```

v = tl.load(X + row * SX + h, h < K, 0).to(tl.float32)
weight = tl.load(W + h, h < K, 0).to(tl.float32)
inv = tl.rsqrt(tl.sum(v * v, 0) / K + EPS)
y = (v * inv * weight).to(tl.bfloat16)
tl.store(Y + row * K + h, y, h < K)
# 关键点：量化输入就是上面写出的 BF16 值，所以与 rmsnorm 后再独立量化逐位相同
mxfp8_epilogue(y, row, Q, S, K, BLOCK, GROUPS, 0, GROUPS)
else:
    # 多出来的 CTA 只负责把 scale 的 padding 槽位清零；这些槽位与上面
    # 真实行的写入集合互不相交，因此不需要额外同步。
    off = (row - M) * 512 + tl.arange(0, 512)
    pad_row = ((off // 4) % 4) * 32 + ((off // 16) % 32)
    tl.store(S + off, 0, (off < GROUPS * 128) & (pad_row >= M))

```

```

def rmsnorm_mxfp8(x, weight, eps):
    """RMSNorm 一次产出 (y_bf16, y_q, y_sf)，与 rmsnorm 后再独立量化逐位相同。"""
    m, k = x.shape
    assert 0 < m <= 8 and k % 32 == 0 # 仅服务 BS<=8 的 decode / verify 形状
    ...
    # 网格尾部额外挂 cdiv(s.numel(), 512) 个 CTA，专门清零 scale padding
    _rmsnorm_mxfp8_kernel[(m + triton.cdiv(s.numel(), 512),)](...)
    return y, q, s

```

test/registered/kernel/attention/test_flashmla_sched_meta.py

本层最具代表性的测试：用 FlashMLA 自身的 tile_scheduler_metadata 作为参考，只比较前 7 个已定义字（_pad 字参考实现不写），并在 FlashMLA 不做 split-KV 的形状上显式 skip；extra cache 场景还把内核产出的元数据塞回 FlashMLA 验证 out/lse 位相等。

```

# FlashMLA 的 DecodingSchedMeta 末尾有一个它自己从不写入的 _pad 字，torch::empty
# 留下的内容不确定，所以对比时只看前面 7 个已定义字。
DEFINED = slice(0, 7)

```

```

def _ours(like_meta, like_splits, topk_length, topk, *, extra_topk_length=None, extra_topk=0):
    """用被测内核重算调度元数据，输出形状对齐 FlashMLA 分配的同形状 buffer。"""
    from sglang.kernels.ops.attention.dsv4.flashmla_sched_meta import flashmla_sched_meta

    meta = torch.empty_like(like_meta)
    splits = torch.empty_like(like_splits)
    flashmla_sched_meta(
        meta, splits,
        topk_length=topk_length,
        extra_topk_length=extra_topk_length,
        block_size_n=BLOCK_SIZE_N, # 64
        fixed_overhead_num_blocks=FIXED_OVERHEAD, # 5, 与 FlashMLA 的固定开销对齐
        topk=topk,
        extra_topk=extra_topk,
    )

```

```
return meta, splits
```

```
@pytest.mark.parametrize("b", [1, 6, 64])
@pytest.mark.parametrize("s_q", [1, 6])
@pytest.mark.parametrize("topk", [2048])
@pytest.mark.parametrize("mode", ["full", "random", "zeros", "ones", "mixed"])
def test_matches_flashmla_schedule(b, s_q, topk, mode):
    kv = _cache()
    topk_length = _lengths(b, topk, mode, b * 1000 + s_q * 37 + topk + len(mode))
    _, _, sched = _call(kv, b, s_q, topk, topk_length)
    if sched.tile_scheduler_metadata is None:
        # 该形状下 FlashMLA 不做 split-KV，没有可比对的参考实现，直接跳过
        pytest.skip("FlashMLA did not split the KV for this shape")
    meta, splits = _ours(sched.tile_scheduler_metadata, sched.num_splits, topk_length, topk)
    assert torch.equal(meta[:, DEFINED], sched.tile_scheduler_metadata[:, DEFINED])
    assert torch.equal(splits, sched.num_splits)

def test_extra_cache_schedule(b, topk, extra_topk):
    ...
    # 把内核产出的元数据塞回 FlashMLA 再跑一次，验证 attention 输出与参考实现逐位相同
    out, lse, _ = _call(kv, b, s_q, topk, topk_length, extra=extra, meta=(meta, splits))
    assert torch.equal(out.view(torch.int16), ref_out.view(torch.int16))
    assert torch.equal(lse.view(torch.int32), ref_lse.view(torch.int32))
```

评论区精华

- BBuf (COMMENTED, 全局评论) 要求把 `test_dsv4_indexer_postprocess.py` 与本 PR 引入的 `candidate_blocks.py` / `indexer_postprocess.py` 放在同一个 PR: Right now #39646 adds both helpers, while their only end-to-end coverage lands in #39648。他给出的后果判断是回归会被错误归因: makes a regression here appear as a Top-k regression later in the stack; 同一评论还指出 PR 描述仍声称抽取文件与来源快照 byte-identical, 但后续清理 commit 已改动签名与注释, 验证章节与文件统计需要刷新。作者未在该线程内书面回应, PR 随后被 APPROVED 并合并。
- BBuf (inline, engram_gather.py) 要求补齐 wrapper 契约校验: 内核用 `ids.numel()` 线性化、按 row 索引并写出 `row * DIM` 个元素, 但当时没有检查 `ids` 是否连续 / 整型 / CUDA、`ids` 与 `out` 是否同设备、`out.shape == (n, dim)`, 并警告 A non-contiguous ids or undersized output can silently read/write the wrong memory; 还要求校验 `dim > 0`、`block_size > 0`, 并补一个覆盖 `owned` 与 `unowned` ID 的正确性测试。参数校验部分由 commit 101686f 落地, 测试部分在本次提交列表中未见对应 commit。
- CI 侧讨论: 作者用 `/rerun-test test_dsv4_q_rope_store.py test_flashmla_sched_meta.py test_mxfp8_epilogue.py` 触发重跑, bot 报告 1-gpu-h100 (1 个测试) 与 4-gpu-b200 (2 个测试) 全部通过; PR 描述中 Base / Extra / AMD ROCm 三个检查呈 `no_entry_sign` 或 🚫, 随后作者执行 `/tag-and-rerun-ci`。

- `candidate_blocks / indexer_postprocess` 的端到端覆盖归属 (testing): 作者未在该评论线程内书面回应, PR 随后被 APPROVED 并合并; 两个 helper 仍留在本层, 独立覆盖缺失的问题被带入栈的后续 PR (#39648 的 `test_topk_v2.py` 等)。PR 描述中也未见 `byte-identical` 声明的正式修订说明——描述里反而新增了 AST equality 与 GB300 测试结果等段落, 说明部分内容已更新但统计口径仍不一致。
- `engram_gather wrapper` 缺少输入契约校验 (correctness): 作者以 `commit 101686f` `validate engram_gather arguments` 补上参数校验 (合并进本 PR); 评论中要求的 `owned / unowned ID` 正确性测试在本次提交列表与文件清单里没有对应的新增测试文件, 仍属待补项。
- 抽取测试的 CI 门禁与 skip 面 (testing): 定向重跑通过, 测试可执行性得到确认; 但覆盖依赖具体架构 (Blackwell) 与参考实现的分支行为, H100 / AMD 等平台的 CI 结果为 `no_entry_sign` 或未覆盖, skip 面需要读者自行评估。

风险与影响

- 风险:
 1. 本层在 main 上不可达, 存在「死代码老化」风险: 模型 `dispatch`、调度器与缓存所有权都在 #38798, 本 PR 的 kernel 没有任何调用者, 一旦 #38798 延迟或被改写, 这些实现只被自身单测覆盖, 接口契约 (如 `can_use_*`、`MAX_M = 32`、`x.stride(0) >= 8192`) 可能在真实调用方接入时才发现不匹配。
 2. 数值一致性契约强依赖外部参考实现: `ue8m0_scale / 128x4 swizzle` 对标 `FlashInfer mxfp8_quantize`, `flashmla_sched_meta` 对标 `sgl_kernel.flash_mla` 的 `DecodingSchedMeta`, `q_rope_store` 对标 `deepseek_rope_kernel` 的运算顺序, 测试用 `torch.equal / assert_close(atol=0)` 做硬约束。任一侧升级 (FlashInfer、FlashMLA 版本或 `_pad` 字语义变化) 都会让这些用例变红甚至静默失配; 测试也承认参考侧 `_pad` 字内容不确定, 只能比较前 7 个字。
 3. 覆盖面窄: `_arch_supported()` 与三个测试的 `pytestmark` 都把范围锁在 Blackwell + CUDA, PR 描述的 AMD ROCm 检查为 `no_entry_sign` / ❷; H20、NPU、XPU 与 HIP 路径完全没有回归防护, 而 #38798 合入后这些平台若误走 `can_use_*` (谓词已排除 HIP 但依赖调用方正确判断), 问题不会在本层暴露。
 4. 大量 `assert` 即报错、无回退路径: `hc_combine_norm` 限定 `0 < m <= 8 or 4096 <= m <= 65536` 与 `(m, 20480)` 形状, `wo_a_bf16_small_batch` 限定 `2 <= m <= 8` 且 `x.shape[1:] == (2, 4096)`, `small_gemm_bf16` 明确 `other shapes raise rather than fall back`。这些约束在调度层写错分支时会直接抛异常而非降级, 属于「快速失败换正确性」的取舍, 需要调用方严格配合。
 5. 提取正确性声明已过期: BBuf 指出 `byte-identical` 与文件统计未随清理 `commit` 刷新 (合并态为 25 个文件 / +2,741 / -19, 而 PR 描述声称 23 个内核文件 +2,459、总计 33 个文件 +3,494), 后续审计若直接引用该声明会得到错误结论; `candidate_blocks.py / indexer_postprocess.py` 的独立覆盖仍然缺失, 回归可能被记到 #39648 头上。
 6. 性能收益无法在本层验证: PDL 重叠 (`gdc_wait / gdc_launch_dependents`)、每行 4 个 CTA 的 `_hc_combine_norm` 冗余统计、`split-K` 小 batch GEMM 等设计都只降延迟

不改数值，单测只能证明正确性；真正的收益与是否被调度层正确启用只能等 #38798 的端到端基准。

• 影响：

- 对现有用户：无行为变化。本 PR 不修改任何已有推理路径，只是把新 kernel 与封装铺进 `python/sglang/kernels/ops/`，DSV4.1 能力是否可用取决于后续 #38798 与 #39648 是否按时合入。
- 对系统与仓库结构：确立 `sglang/kernels/ops/{embeddings,layernorm,gemm,speculative,attention/dsv4}` 这一内核库布局，以及 `jit/csrc` 头文件命名（`dot_product.cuh`、`small_gemm_bf16.cuh`、`flashmla_sched_meta.cuh`），后续 DSV4.1 系列 PR 会继续按此路径增量交付；`register_custom_op(..., mutates_args=["out"])` 与 `debug_kernel_api`、`@cache_once` 的组合成为该目录的默认写法。
- 对存储与显存：`rmsnorm_mxfp8` / `hc_combine_norm_mxfp8` 额外分配 128 行对齐的 swizzled scale buffer，`q_rope_store` 的 prefill 分支为超过 2 GiB 的 padding 输出引入 int64 偏移；这些都是后续显存预算评估必须纳入的项。
- 对团队协作：22 个 commit 显示出明显的「先抽取、再命名收敛、再补契约校验」的演进节奏；BBuf 的两条意见确立了本栈的 review 规则——同一 PR 引入的 helper 应尽量自带端到端覆盖，wrapper 必须显式校验输入契约。这条规则会直接作用于 #39648（Top-k 与候选选择）、#39677（Rust 侧模块）等后续层。
- 对 CI 成本：新增 base-b-kernel-unit stage 下 4-gpu-b200 与 1-gpu-large 的调度，`est_time` 分别为 25s / 120s / 40s，且只在 Blackwell 上执行；这部分 GPU 资源占用是持续成本，而 `flashmla_sched_meta` 测试还会在 FlashMLA 不切分 KV 的形状上静默 skip（描述称 66 个形状）。
- 风险标记：本层无运行时调用方，数值契约强依赖外部参考实现，候选 / 索引内核缺独立测试，仅 Blackwell 有测试覆盖，PR 描述与最终代码不一致，形状断言即报错无回退

关联脉络

- PR #39648 dsv4.1: Top-k kernels and candidate selection: 同一线性 PR 栈的下一层，也是本 PR 的隔离对象：PR 描述写明 Existing Top-k implementation changes are isolated in #39648，BBuf 的评论又指出本 PR 引入的 `candidate_blocks.py` / `indexer_postprocess.py` 端到端覆盖落在 #39648，两层是直接的上下游依赖关系。
- PR #39677 dsv4.1: Rust extension modules for image preprocessing, KV pool names, and PD bootstrap: 同属 dsv4.1 系列抽取 PR 栈（同一作者、同一命名前缀、同为从大 PR 拆出的独立层），从 Rust 侧补齐图像预处理、KV 池名与 PD bootstrap，与本 PR 的 Python/Triton 层互为对偶，需一起评估 DSV4.1 的整体交付节奏。
- PR #36534 [DSV4][BCG] Optimize the heavy memory use of C4 Indexer when BCG is enabled: 同一条 DSV4 索引器 / 注意力演进线：该 PR 在 `deepseek_v4_backend.py` 与 `prefill_cuda_graph_runner.py` 上处理 C4 Indexer 的显存开销，而本 PR 的 `candidate_blocks.py`、`indexer_postprocess.py`、`flashmla_sched_meta.py` 正是这条索引器路径的算子底座，两者的显存与调度假设相互影响。