PR #52212 完整报告

vllm-project/vllm

[ROCm][DSV4][Perf] Optimize Triton sparse-MLA decode on gfx950

合并时间：2026-08-17 03:16

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52212

执行摘要

- 一句话：gfx950 专用 DSV4 稀疏 decode 内核，吞吐 +2.7%
- 推荐动作：值得精读。该 PR 在性能工程层面展示了 gfx950 专项内核设计（512 宽 QK dot、4×128 PV 累加器、guard tile 与压缩组 peeling）、workload-aware split 策略、以及通过数据契约（NaN-free + provenance 门控）换取跳过 scrub 的性能收益。review 中关于 gfx942 共享 selector 范围控制的交锋是典型的架构特化边界案例，值得作为团队评审参照。但需特别注意 commit 9c4d637d 夹带的 MRV1 回退与 PR 主题不符，建议维护者单独跟踪该行为变更。

功能与动机

DeepSeek-V4 稀疏 MLA 在 gfx950/MI355X 上做长上下文、低 batch decode 时，现有 in-tree Triton split-K 路径的并行度和寄存器数据流存在明显瓶颈。PR body 明确指出目标是优化默认 in-tree Triton 实现，而不引入 AITER/Gluon 运行时依赖（修正后的 AITER/Gluon 仅作为性能对照）。作者在 8k/1k 服务 A/B 中验证了输出吞吐 +2.754%、P99 TTFT -7.919% 的收益，并强调这是单次运行的配对无回归结论而非统计显著结论。

实现拆解

1. gfx950 专用 partial 内核与数据流改造：在 vllm/v1/attention/ops/rocm_aiter_mla_sparse.py 中新增 _sparse_attn_decode_gfx950_partial_kernel 及配套 _load_fp8_ds_mla_gfx950_nope_exact_chunk、_load_fp8_ds_mla_gfx950_tail128、_decode_e8m0_scales_triton。内核将 448 维 NoPE 与 64 维 RoPE 合并为一次 512 宽 QK dot，PV 侧使用 4 个 128 宽累加器避免寄存器溢出，按 BK32 guarded tile 处理 main/SWA 工作，并保留 masked tail、attention sink 与 invalid-index 语义。
2. workload-aware split 选择与 graph-safe 自适应：新增 _decode_gfx950_num_splits，按设备占用率目标选择 split 数，短负载上限 4、低 batch 长上下文最多 32；同时引入 build_for_cudagraph_capture 标记与 for_cudagraph_capture 元数据字段，使捕获期使用固定最大 grid，运行时按行选择活跃 splits，_copy_ragged_to_graph_buffers 在 gfx950 上保留 graph-stable base pointer 并暴露 source 容量（indptr 仍携带真实 NNZ）。
3. 压缩缓存 NaN-free 数据契约：在 vllm/models/deepseek_v4/common/ops/fused_compress_quant_cache.py 中为 compress_norm_rope_store_triton 与两阶段 finalizer 增加 SANITIZE_CACHE_NANS（仅 gfx950 开启），将 UE8M0 scale code 上限从 255 收紧

到 254，并将 RoPE 结果中 NaN/Inf 规范化为 0；vllm/models/deepseek_v4/amd/rocm.py 新增 _trust_dsv4_extra_cache_nan_free provenance 门控（要求 fp8_ds_mla 缓存、无 KV 传输、存在 extra cache），只有满足时才跳过解码期 scrub，否则保持默认 scrub 行为。

4. cp-gather 去特化避免编译爆炸：_cp_gather_indexer_quant_cache_gfx950_kernel 通过 do_not_specialize=["num_batches"] 去掉 token 数 constexpr 特化，batch 数改为 runtime 标量，block 地址乘法改用 int64 防 packed 布局溢出；分派处按 _ON_GFX950 选择 gfx950 或 legacy 内核。
5. 测试与 CI 配套：tests/kernels/attention/test_rocm_triton_attn_dsv4.py 重构了缓存读取 helper 支持批量 slot 读取，新增 poison/scrub 默认行为测试、provenance 门控测试、ragged graph buffer view 跟踪测试与 gfx950 专用 requires_gfx950 用例；tests/kernels/test_compressor_kv_cache.py 新增 gfx950-only cp-gather 分派测试（FakeKernel 断言参数个数与 grid）、NaN-free 规范化与 legacy scrub 等价性测试，并扩展 indexer gather 测试为多序列场景。

关键文件：

- vllm/models/deepseek_v4/amd/rocm.py（模块 模型层；类别 source；类型 data-contract；符号 _trust_dsv4_extra_cache_nan_free, build_for_cudagraph_capture, _copy_ragged_to_graph_buffers, DeepseekV4ROCMAiterMLASparseMetadata）：ROCm DSV4 入口层：新增 NaN-free provenance 门控 _trust_dsv4_extra_cache_nan_free、graph-capture 标记 for_cudagraph_capture 与 build_for_cudagraph_capture，并调整 _copy_ragged_to_graph_buffers 使 gfx950 保留 graph-stable base pointer 同时暴露 source 容量，是自适应 split 与跳过 scrub 的数据契约来源
- vllm/v1/attention/ops/rocm_aiter_mla_sparse.py（模块 注意力内核；类别 infra；类型 core-logic；符号 _cp_gather_indexer_quant_cache_gfx950_kernel, _decode_e8m0_scales_triton, _load_fp8_ds_mla_gfx950_nope_exact_chunk, _load_fp8_ds_mla_gfx950_tail128）：变更核心：新增 gfx950 专用 Triton 稀疏 decode partial 内核族（512 宽 QK dot、4x128 PV 累加器、BK32 guarded tiles）、_decode_gfx950_num_splits 32-split 选择器、UE8M0 scale 解码、以及去特化的 gfx950 cp-gather 内核，是整个性能收益的载体
- vllm/models/deepseek_v4/common/ops/fused_compress_quant_cache.py（模块 压缩缓存；类别 infra；类型 core-logic；符号 compress_norm_rope_store_triton, _fused_kv_compress_norm_rope_insert_sparse_attn, _finalize_norm_rope_quant_store_sparse_attn, _launch_two_stage_sparse_attn_compressor）：写入端配套：compress_norm_rope_store_triton 在两阶段压缩写入路径上启用 SANITIZE_CACHE_NANS，将 scale code 上限从 255 收紧到 254 并规范化 RoPE 非有限值，是解码端跳过 scrub 的前提
- tests/kernels/attention/test_rocm_triton_attn_dsv4.py（模块 内核测试；类别 test；类型 test-coverage；符号 _on_gfx950, _poison_fp8_ds_mla_cache_row, _read_fp8_ds_mla_cache_rows, _launch_sparse_decode_reduce）：核心测试配套：重

构缓存读取 helper 为批量 slot 读取，新增 poison/scrub 默认行为测试、provenance 门控测试、ragged graph buffer view 跟踪测试与 gfx950 专用用例，覆盖新数据契约与内核路径

- tests/kernels/test_compressor_kv_cache.py（模块 压缩测试；类别 test；类型 test-coverage；符号 _on_gfx950, test_cp_gather_despecialized_kernel_is_gfx950_only, FakeKernel, _decode_dsv4_cache_row）：压缩缓存测试配套：验证 gfx950-only cp-gather 去特化内核的分派行为与参数契约，并用单遍 / 两阶段双 writer 验证 NaN 规范化输出与 legacy scrub 等价

关键符号：_trust_dsv4_extra_cache_nan_free, build_for_cudagraph_capture, _copy_ragged_to_graph_buffers, _decode_gfx950_num_splits, _sparse_attn_decode_gfx950_partial_kernel, _sparse_attn_decode_gfx950_partial_loaded_tile, _load_fp8_ds_mla_gfx950_nope_exact_chunk, _load_fp8_ds_mla_gfx950_tail128, _decode_e8m0_scales_triton, _cp_gather_indexer_quant_cache_gfx950_kernel, compress_norm_rope_store_triton

关键源码片段

vllm/models/deepseek_v4/amd/rocm.py

ROCm DSV4 入口层：新增 NaN-free provenance 门控 _trust_dsv4_extra_cache_nan_free、graph-capture 标记 for_cudagraph_capture 与 build_for_cudagraph_capture，并调整 _copy_ragged_to_graph_buffers 使 gfx950 保留 graph-stable base pointer 同时暴露 source 容量，是自适应 split 与跳过 scrub 的数据契约来源

```
# gfx950 上跳过 extra-cache scrub 的信任门控。
# 要求：gfx950 + fp8_ds_mla 缓存 + 无 KV 传输 + 确实存在 extra cache。
# 任一条件不满足时保持旧语义，解码端仍逐元素清除非有限值。
def _trust_dsv4_extra_cache_nan_free(
    kv_cache_dtype: str,
    has_kv_transfer: bool,
    has_extra_cache: bool,
) -> bool:
    return (
        _ON_GFX950
        and kv_cache_dtype == "fp8_ds_mla"
        and not has_kv_transfer
        and has_extra_cache
    )


def _copy_ragged_to_graph_buffers(
    ragged_indices: torch.Tensor,
    ragged_indptr: torch.Tensor,
    ragged_indices_buffer: torch.Tensor,
    ragged_indptr_buffer: torch.Tensor,
    num_rows: int,
    max_entries_per_row: int,
```

```
) -> tuple[torch.Tensor, torch.Tensor]:
    """将动态 ragged 元数据拷贝进持久 CUDA graph 缓冲区。"""
    indptr_out = ragged_indptr_buffer[: num_rows + 1]
    indptr_out.copy_(ragged_indptr, non_blocking=True)

    max_entries = max(num_rows * max_entries_per_row, 1)
    ragged_out = ragged_indices_buffer[:max_entries]
    source_entries = ragged_indices.numel()
    if source_entries > 0:
        ragged_out[:source_entries].copy_(ragged_indices, non_blocking=True)
    if _ON_GFX950:
        # 保留 graph-stable 的 base pointer，同时把视图收窄到本次 source 容量，
        # 供 split selector 在无同步情况下读取；indptr 仍携带真实 NNZ。
        ragged_out = ragged_out[: max(source_entries, 1)]
    return ragged_out, indptr_out
```

vllm/v1/attention/ops/rocm_aiter_mla_sparse.py

变更核心：新增 gfx950 专用 Triton 稀疏 decode partial 内核族（512 宽 QK dot、4×128 PV 累加器、BK32 guarded tiles）、_decode_gfx950_num_splits 32-split 选择器、UE8M0 scale 解码、以及去特化的 gfx950 cp-gather 内核，是整个性能收益的载体

```
@triton.jit(do_not_specialize=["num_batches"])
def _cp_gather_indexer_quant_cache_gfx950_kernel(
    kv_cache_ptr, kv_cache_scale_ptr, k_fp8_ptr, k_scale_ptr,
    block_table_ptr, cu_seqlen_ptr, token_to_seq_ptr,
    block_size, block_table_stride, kv_cache_stride, kv_cache_scale_stride,
    LAYOUT: tl.constexpr, HEAD_DIM: tl.constexpr,
    BLOCK_TILE_SIZE: tl.constexpr, HEAD_TILE_SIZE: tl.constexpr,
    num_batches, BLOCK_TABLE_WIDTH: tl.constexpr, NUM_BLOCKS: tl.constexpr,
):
    # 与 legacy 内核的关键差异：token 数不再作为 constexpr 参与特化
    # （避免每次请求 shape 变化都触发重编译），batch 数改为 runtime 标量。
    tid = tl.program_id(0)
    offset = tl.arange(0, HEAD_DIM)
    batch_id = tl.load(token_to_seq_ptr + tid)
    valid_batch = (batch_id >= 0) & (batch_id < num_batches)
    safe_batch_id = tl.where(valid_batch, batch_id, 0)
    batch_start = tl.load(cu_seqlen_ptr + safe_batch_id, mask=valid_batch, other=0)
    batch_end = tl.load(cu_seqlen_ptr + safe_batch_id + 1, mask=valid_batch, other=0)
    batch_offset = tid - batch_start
    valid_token = valid_batch & (tid >= batch_start) & (tid < batch_end)
    if not valid_token:
        return
    # …块表查找后，packed 布局下 block_id * stride 可能超过 32 位范围，
    # 因此先转 int64 再做偏移计算，防止地址溢出。
    block_id = tl.load(
        block_table_ptr + safe_batch_id * block_table_stride + block_table_id,
        mask=valid_block_table, other=-1,
    )
```

```
    valid_block = valid_block_table & (block_id >= 0) & (block_id < NUM_BLOCKS)
    safe_block_id = tl.where(valid_block, block_id, 0).to(tl.int64)
    # …按 LAYOUT 计算 src 偏移并 gather 到 k_fp8 / k_scale 输出…
    tl.store(k_scale_ptr + tid, scale_val)
    tl.store(dst_k_ptr + offset, val, mask=valid_block)


def cp_gather_indexer_k_quant_cache_triton(
    k_cache, k_fp8, k_fp8_scale, block_table, cu_seqlen, token_to_seq,
):
    # gfx950 走去特化内核；其余架构保持 legacy 内核与原有参数个数
    #  （legacy 仍会把 num_tokens 等作为 constexpr 特化）。
    if _ON_GFX950:
        _cp_gather_indexer_quant_cache_gfx950_kernel[grid](
            *kernel_args, num_batches, block_table.shape[1], num_blocks,
        )
    else:
        _cp_gather_indexer_quant_cache_kernel[grid](
            *kernel_args, num_tokens, cu_seqlen.shape[0] - 1,
            block_table.shape[1], num_blocks,
        )
```

vllm/models/deepseek_v4/common/ops/fused_compress_quant_cache.py

写入端配套：compress_norm_rope_store_triton 在两阶段压缩写入路径上启用 SANITIZE_CACHE_NANS，将 scale code 上限从 255 收紧到 254 并规范化 RoPE 非有限值，是解码端跳过 scrub 的前提

```
# gfx950 专用：把压缩缓存的 scale code 上限从 255 收紧到 254，
# 并把 RoPE 的 NaN/Inf 规范化为 0，从而使解码端可以信任缓存不含 NaN，
# 跳过逐元素 scrub —— 这是解码性能收益的关键前提。
# 非 gfx950 平台保持 legacy 255 上限与不清理行为。
if head_dim == 512:
    kernel = _fused_kv_compress_norm_rope_insert_sparse_attn
    num_warps = 4
    kernel_kwargs = {"SANITIZE_CACHE_NANS": _ON_GFX950}
elif use_fp4_cache:
    kernel = _fused_kv_compress_norm_rope_insert_indexer_mxfp4_attn
    num_warps = 1
    kernel_kwargs = {}
else:
    kernel = _fused_kv_compress_norm_rope_insert_indexer_attn
    num_warps = 1
    kernel_kwargs = {}

# 内核内部：按 SANITIZE_CACHE_NANS 选择 scale code 上限并清理 NaN。
max_encoded: tl.constexpr = 254.0 if SANITIZE_CACHE_NANS else 255.0
encoded = tl.maximum(tl.minimum(encoded, max_encoded), 0.0)
tl.store(
    scale_ptr + scale_idx,
```

```
        encoded.to(tl.uint8),
    )
    # …RoPE 旋转后，若启用清理则把 NaN 置零，避免污染后续 softmax。
    if SANITIZE_CACHE_NANS:
        result = tl.where(result == result, result, 0.0)
```

评论区精华

Review 中唯一实质技术交锋是 jiacao-amd 在 vllm/v1/attention/ops/rocm_aiter_mla_sparse.py 的 _decode_num_splits 改动上提问："Is it safe to change splits range for gfx942?"——原代码把共享 selector 的 split 上限从 16 提高到 32，而该函数同时服务 gfx942。作者 Fangzhou-Ai 回复承认范围控制不当，并在 5dfdac01ed 中恢复共享 selector 与测试为原始 16-split 上限，仅 _decode_gfx950_num_splits 允许 32；同时恢复 legacy generic/gfx942 稀疏内核与 cp-gather launch，而 adaptive graph path、narrowed ragged view、direct-output path 与 cp-gather 去特化全部显式由 _ON_GFX950 门控。该线程状态为已解决。其余审核为 AndreasKaratzas 的 LGTM 批准与两次 CI 触发，Claude bot 均因 fork PR 自动 review 被禁用而未产生有效意见。

- 共享 split selector 上限改动是否会波及 gfx942 (design): reviewer 指出后作者在 5dfdac01ed 恢复共享 selector 与测试为原始 16-split 上限，仅 _decode_gfx950_num_splits 允许 32；同时恢复 legacy generic/gfx942 内核与 cp-gather launch，gfx950 特有路径全部由 _ON_GFX950 门控。

风险与影响

- 风险：
  1. gfx942 回归风险：最初共享 _decode_num_splits 上限被扩到 32，若未回复将改变 gfx942 的 split 行为；review 后已恢复 16 上限，但仍需关注最终合入时是否残留共享路径污染。
  2. NaN-free 缓存契约正确性：SANITIZE_CACHE_NANS 仅由 gfx950 写入端保证，读取端通过 _trust_dsv4_extra_cache_nan_free 门控信任。若存在未经验证的第三方写入路径（如 KV 传输或未来新写入内核）绕过规范化，extra_cache 中残留 NaN 会直接参与 softmax，测试中的 poison 用例只覆盖了 gfx950 默认 scrub 路径。
  3. CUDA graph 捕获路径复杂度：build_for_cudagraph_capture 与固定最大 grid + 按行动态 work 的设计依赖 for_cudagraph_capture 标记正确传播；_copy_ragged_to_graph_buffers 收窄视图但不改变 base pointer，若后续调用方误解 source 容量与 NNZ 的差异，可能越界或漏读。
  4. MRV1 回退范围蔓延：commit 9c4d637d 将 ROCm DSV4 默认从 MRV2 回退到 MRV1（为规避 ROCm 上 decode TPOT 回归），该改动与"gfx950 稀疏 decode 内核优化"标题不一致，且 co-author 包含非仓库维护者，需要确认是否单独评审。
  5. 性能不确定性：P90 TTFT 在 A/B 中回退 4.092%，作者也声明这是单次运行的配对结论；高并发下 split 选择的稳定性仍需更多重复实验验证。 - 影响：影响范围集中在 ROCm gfx950（MI355X）平台上的 DeepSeek-V4 稀疏 MLA decode 路径：服务吞吐提升约 2.7%，P99/P99.9 TTFT 改善 4.8%-7.9%，P90 TTFT 略有回退。gfx942 与通

用路径行为保持不变（AOT 审计确认 gfx942 HSACO 字节一致）。对团队而言，该 PR 确立了一个值得复用的模式：用 provenance 位显式门控"跳过安全 scrub"的优化，以及 graph capture 期固定资源、运行期动态选择工作的设计；同时它也演示了跨架构共享代码做架构特化时容易产生的范围控制问题。对下游用户的直接影响是 gfx950 上 DSV4 长上下文服务延迟改善，无新依赖、无配置项变化。 - 风险标记：gfx942 共享路径回归（已修复），NaN-free 缓存数据契约，CUDA graph 捕获复杂度，MRV1 回退范围蔓延，编译缓存膨胀（cp-gather 去特化）

关联脉络

- PR #51430 [MRV2] Move DSV4 to Model Runner v2: commit 9c4d637d 引用 #51430/#51768：本 PR 将 ROCm DSV4 默认回退到 MRV1 以规避 MRV2 在 ROCm 上的 decode TPOT 回归，与 MRV2 迁移方向相反
- PR #49544 [ROCm][Perf] gfx942: use FlyDSL fp8 MQA logits kernel: 同为 ROCm 稀疏 MLA 内核性能优化（gfx942），本 PR 的 gfx950 内核设计与其形成对照并复用同一 rocm_aiter_mla_sparse.py 模块
- PR #51318 [Bugfix][DSv4] Revert adaptive C128A metadata packing: 与稀疏 MLA decode 元数据稳定性直接相关，本 PR 的 ragged 视图收窄与 graph buffer 语义需与已回退的元数据打包保持兼容
- PR #51538 [Bugfix] Make DSV4 sparse MLA work end-to-end for plain decode, MTP, and DSpark: 同一稀疏 MLA decode 路径的端到端正确性修复，本 PR 在其基础上做 gfx950 性能特化
- PR #52084 [Perf][DSV4] Optimize sparse top-k metadata kernels for higher prefill throughput: 同一 DSV4 性能优化线：top-k 元数据与 decode 内核共同构成稀疏 MLA 的完整性能路径
- PR #52401 [Bugfix] Pick the DeepSeek V4 eager cudagraph region per model runner: 涉及 DSV4 CUDA graph region 选择，与本 PR 的 graph-safe 自适应 split 与回退 MRV1 决策相关