PR #24933 完整报告

sgl-project/sglang

Amd/deepseek v4 rebase main 0509

合并时间：2026-05-19 00:15

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24933

执行摘要

- 一句话：为 ROCm 平台添加 DeepSeek V4 模型支持，新增 HIP 注意力后端与 Triton 内核
- 推荐动作：值得精读的文件包括 deepseek_v4_backend_hip_radix.py（理解 ROCm 后端架构）、compress_hip.py（HIP 专用压缩设计）和 tilelang_kernel.py（TileLang 内核实现与 monkey-patch 技巧）。建议重点关注环境变量治理和条件编译模式，后续可借鉴到其他平台适配。

功能与动机

PR body 明确说明：'Enable deepseek v4 model support (merge to main) to ROCm platform'，目标是让 DeepSeek V4 系列模型在 AMD GPU 上首次可运行，作为后续优化的基础，并逐步迁移 amd/deepseek_v4 分支上的优化工作。

实现拆解

1. 新增 HIP 注意力后端：创建 deepseek_v4_backend_hip_radix.py，实现 DeepseekV4HipRadixBackend，继承 AttentionBackend、CompressorBackendMixin 和 C4IndexerBackendMixin。定义 DSV4AttnMetadata 数据结构，包含分页索引、压缩元数据等。_create_flashmla_metadata 在 HIP 平台返回 None，不依赖 flash_mla 库；init_compression_metadata 导入 Triton 内核。
2. HIP 专用压缩器：新增 compress_hip.py，实现 CompressorHip（继承 Compresso 基类）。使用 Triton RMS Normalize 内核替代 CUDA 版本；use_fused_compress 固定为 False，通过环境变量 SGLANG_OPT_USE_FUSED_COMPRESS 选择性开启 HIP fused 压缩。提供 overlap_transform 等方法用于预填充阶段。
3. Flash MLA 入口适配：新增 hip_flash_mla.py，暴露 flash_mla_with_kvcache_entrypoint 函数，根据 SGLANG_HACK_FLASHMLA_BACKEND 环境变量选择“tilelang”或“torch”后端；flash_mla_with_kvcache_torch 用于调试对比。
4. TileLang/Triton 内核扩展：在 tilelang_kernel.py 中大幅新增 fp8_paged_mqa_logits_kernel 系列、dpsk_v4_fp8_partial_kernel 和 dpsk_v4_fp8_attention_fwd 函数，处理 FP8 分页 MQA 的 logits 计算和稀疏注意力解码。同时增加了 tilelang 适配器 bug 的 monkey-patch（_legalize_result_idx_safe）。

5. 内存管理与状态扩展：修改 deepseek_v4_compress_state.py，为 KVAndScore 添加 from_kv_score、clone、cat 等便捷方法，CompressStatePool 根据 is_hip 使用不同的内存分配策略。其他适配包括 deepseek_v4_rope.py 添加 fused_norm_rope_inplace_triton 融合内核，deepseek_v4.py 调整导入路径，attention_registry.py 注册新后端。

关键文件：

- python/sglang/srt/layers/attention/deepseek_v4_backend_hip_radix.py（模块 注意力后端；类别 source；类型 core-logic；符号 _pad_last_dim, _create_flashmla_metadata, _create_dummy_paged_compress_data, DSV4AttnMetadata）：核心新增文件，实现了 HIP 专用的 DeepSeek V4 注意力后端，包含 DSV4AttnMetadata 数据结构、flash_mla 元数据管理、分页压缩数据创建等关键逻辑，是整个 ROCm 适配的入口。
- python/sglang/srt/layers/attention/dsv4/compress_hip.py（模块 压缩器；类别 source；类型 core-logic；符号 _rms_normalize_kernel, rms_normalize_triton, DeepseekRefRMSNorm, init）：新增的 HIP 专用压缩器，使用 Triton RMS Normalize 内核，实现 CompressorHip 类，展示 ROCm 上与 CUDA 差异化实现的关键逻辑。
- python/sglang/srt/layers/attention/nsa/tilelang_kernel.py（模块 TileLang 内核；类别 source；类型 core-logic；符号 _legalize_result_idx_safe, fp8_paged_mqa_logits_kernel, fp8_paged_mqa_logits, tilelang_fp8_paged_mqa_logits）：大幅修改并新增大量 TileLang 内核，包括 fp8_paged_mqa_logits_kernel 和 dpsk_v4_fp8_attention_fwd，是 ROCm 计算性能的关键。包含 tilelang 适配器 bug 的绕过补丁。
- python/sglang/srt/layers/attention/hip_flash_mla.py（模块 FlashMLA 适配；类别 source；类型 dependency-wiring；符号 flash_mla_with_kvcache_entrypoint, flash_mla_with_kvcache_torch, _assert_close）：新增 Flash MLA 入口函数，统一调度 torch、tilelang、kernel 三种后端，是 ROCm 上 MLA 计算路由的核心。
- python/sglang/srt/mem_cache/deepseek_v4_compress_state.py（模块 压缩状态；类别 source；类型 core-logic；符号 shape, from_kv_score, new_empty, setitem）：扩展 KVAndScore 数据结构和 CompressStatePool 内存分配策略，增加便捷方法并支持 HIP 路径。
- python/sglang/srt/layers/deepseek_v4_rope.py（模块 融合 RoPE；类别 source；类型 core-logic；符号 _fused_norm_rope_kernel, fused_norm_rope_inplace_triton）：新增 fused_norm_rope_inplace_triton 内核，将 RMSNorm 和 RoPE 融合到一个 Triton 内核中，减少访存。
- python/sglang/srt/layers/quantization/fp8.py（模块 量化层；类别 source；类型 dependency-wiring；符号 _require_fp4_dtype）：量化模块调整，支持 FP4 等类型，确保兼容 ROCm 上的新内核。
- python/sglang/srt/models/deepseek_v4.py（模块 模型定义；类别 source；类型 data-contract）：模型定义文件调整，引入新的 HIP 后端导入路径，确保模型初始化正确。

关键符号：pad_last_dim, _create_flashmla_metadata, DSV4AttnMetadata.get_flashmla_metadata, DSV4AttnMetadata.copy, _rms_normalize_kernel, rms_normalize_triton, CompressorHip.use_fused_compress, flash_mla_with_kvcache_entrypoint, fp8_paged_mqa_logits_kernel,

dpsk_v4_fp8_attention_fwd, _legalize_result_idx_safe, fused_norm_rope_inplace_triton, KVAndScore.from_kv_score, KVAndScore.clone

关键源码片段

python/sglang/srt/layers/attention/deepseek_v4_backend_hip_radix.py

核心新增文件，实现了 HIP 专用的 DeepSeek V4 注意力后端，包含 DSV4AttnMetadata 数据结构、flash_mla 元数据管理、分页压缩数据创建等关键逻辑，是整个 ROCm 适配的入口。

```
# 文件：sglang/srt/layers/attention/deepseek_v4_backend_hip_radix.py
from __future__ import annotations

import torch
import torch.nn.functional as F
from sglang.srt.utils import ceil_align

# 常量定义
SWA_WINDOW = 128
C4_TOPK = 512
PAGE_INDEX_ALIGNED_SIZE = 64


def _pad_last_dim(x, multiples_of: int = PAGE_INDEX_ALIGNED_SIZE):
    """将张量最后一维补齐到 multiples_of 的整数倍，填充值为 -1。"""
    if x is None:
        return None
    curr_size = x.shape[-1]
    target_size = ceil_align(curr_size, multiples_of)
    return F.pad(x, pad=(0, target_size - curr_size), mode="constant", value=-1)


def _create_flashmla_metadata():
    """在HIP（ROCm）上返回 None，避免依赖 flash_mla 库。"""
    from sglang.srt.utils import is_hip
    if is_hip():
        return None
    import flash_mla
    return flash_mla.get_mla_metadata()[0]


def _create_dummy_paged_compress_data(compress_ratio: int):
    """HIP 平台暂不支持 paged compress，返回 None。"""
    return None


@dataclass
class DSV4AttnMetadata:
    """存储 DeepSeek V4 注意力后端所需的所有元数据。
    包含 page_table、raw_out_loc、各种压缩层级（c1/c4/c128）的索引和 flash_mla 元数据。
```

```
    """
    page_size: int
    page_table: torch.Tensor
    raw_out_loc: torch.Tensor
    cuda_int32_kwargs: dict
    seq_lens_casual: torch.Tensor
    positions_casual: torch.Tensor
    swa_page_indices: torch.Tensor
    swa_topk_lengths: torch.Tensor
    c4_sparse_topk: int
    # 可选字段
    c4_out_loc: Optional[torch.Tensor] = None
    c4_topk_lengths_raw: Optional[torch.Tensor] = None
    c4_topk_lengths_clamp1: Optional[torch.Tensor] = None
    # 动态初始化字段
    c4_sparse_topk_lengths: torch.Tensor = field(init=False)
    c4_sparse_page_indices: torch.Tensor = field(init=False)
    c128_out_loc: Optional[torch.Tensor] = None
    c128_page_indices: Optional[torch.Tensor] = None
    c128_topk_lengths_clamp1: Optional[torch.Tensor] = None
    c1_flashmla_metadata: FlashMLASchedMeta = field(init=False, repr=False)
    c4_flashmla_metadata: FlashMLASchedMeta = field(init=False, repr=False)
    c128_flashmla_metadata: FlashMLASchedMeta = field(init=False, repr=False)

    @property
    def positions(self) -> torch.Tensor:
        return self.positions_casual

    def get_flashmla_metadata(self, compress_ratio: Literal[0, 4, 128]):
        """根据压缩比返回对应的 flash_mla 元数据。"""
        if compress_ratio == 0:
            return self.c1_flashmla_metadata
        elif compress_ratio == 4:
            return self.c4_flashmla_metadata
        elif compress_ratio == 128:
            return self.c128_flashmla_metadata
        else:
            raise ValueError(f"invalid {compress_ratio=}")
```

python/sglang/srt/layers/attention/dsv4/compress_hip.py

新增的 HIP 专用压缩器，使用 Triton RMS Normalize 内核，实现 CompressorHip 类，展示 ROCm 上与 CUDA 差异化实现的关键逻辑。

```
# 文件：sglang/srt/layers/attention/dsv4/compress_hip.py
from functools import cached_property
import triton
import triton.language as tl
from sglang.srt.layers.attention.dsv4.compressor import Compressor as _CompressorBase
```

```
@triton.jit
def _rms_normalize_kernel(x_ptr, weight_ptr, eps, stride_row, dim,
                          BLOCK_SIZE: tl.constexpr, HAS_WEIGHT: tl.constexpr):
    """Triton 实现的 RMS 归一化内核，支持可选的逐元素权重缩放。"""
    pid = tl.program_id(0)
    offs = tl.arange(0, BLOCK_SIZE)
    mask = offs < dim
    base = pid * stride_row
    x = tl.load(x_ptr + base + offs, mask=mask, other=0.0).to(tl.float32)
    mean_sq = tl.sum(x * x, axis=0) / dim
    rms_inv = tl.rsqrt(mean_sq + eps)
    out = x * rms_inv
    if HAS_WEIGHT:
        weight = tl.load(weight_ptr + offs, mask=mask, other=0.0)
        out = out * weight
    tl.store(x_ptr + base + offs, out, mask=mask)


def rms_normalize_triton(x: torch.Tensor, eps: float, weight: torch.Tensor = None) -> torch.
Tensor:
    """对输入 x 做 RMS 归一化，可选权重。
    将输入展开为二维，逐行调用 Triton 内核。
    """
    dim = x.shape[-1]
    x_flat = x.view(-1, dim)
    num_rows = x_flat.shape[0]
    BLOCK_SIZE = triton.next_power_of_2(dim)
    grid = (num_rows,)
    _rms_normalize_kernel[grid](x_flat, weight, eps, x_flat.stride(0), dim,
                                BLOCK_SIZE=BLOCK_SIZE, HAS_WEIGHT=(weight is not None))
    return x


class CompressorHip(_CompressorBase):
    """HIP 平台专用的 Compressor，使用 Triton 内核完成归一化。
    默认关闭 fused_compress（避免 CUDA 特定优化），通过环境变量可选启用。
    """
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        self.norm = DeepseekRefRMSNorm(self.head_dim, eps=self.norm.variance_epsilon)

    @cached_property
    def use_fused_compress(self) -> bool:
        # HIP 上禁用 fused 压缩，改用 Triton 实现
        return False

    @cached_property
    def use_hip_fused_compress(self) -> bool:
        # 通过环境变量控制是否使用 HIP 本地的 fused 压缩
```

```
    return envs.SGLANG_OPT_USE_FUSED_COMPRESS.get()
```

python/sglang/srt/layers/attention/nsa/tilelang_kernel.py

大幅修改并新增大量 TileLang 内核，包括 fp8_paged_mqa_logits_kernel 和 dpsk_v4_fp8_attention_fwd，是 ROCm 计算性能的关键。包含 tilelang 适配器 bug 的绕过补丁。

```
# 文件：sglang/srt/layers/attention/nsa/tilelang_kernel.py
import functools
from tilelang.jit.adapter.base import BaseKernelAdapter as _BaseKernelAdapter

# ----------------------
# 绕过 tilelang 内部 bug：_legalize_result_idx 会就地修改 out_idx 列表
# 当同一个 @tilelang.jit 工厂编译两个不同参数计数的 prim_func 时，
# 第二次编译看到的索引已被第一次转换，导致适配器生成错误，运行时 IndexError。
# 补丁：在 mutation 前复制列表。
# ----------------------
if not getattr(_BaseKernelAdapter, "_legalize_result_idx_patched", False):
    _orig_legalize = _BaseKernelAdapter._legalize_result_idx

    def _legalize_result_idx_safe(self, result_idx):
        if isinstance(result_idx, list):
            result_idx = list(result_idx) # 复制避免原地修改
        return _orig_legalize(self, result_idx)

    _BaseKernelAdapter._legalize_result_idx = _legalize_result_idx_safe
    _BaseKernelAdapter._legalize_result_idx_patched = True


@functools.cache
def fp8_paged_mqa_logits_kernel(
    head_dim: int = 128,
    num_heads: int = 64,
    block_size: int = 64,
    clear_accum: bool = True,
    split_kv: int = 1,
):
    """构造一个 TileLang 内核，计算 FP8 分页 MQA 的 logits。
    使用符号化形状 (N, L, S, C)，由 tilelang 自动处理任意 batch 大小。
    """
    # 符号化变量声明
    N = T.symbolic("batch_size")
    L = T.symbolic("max_table_length")
    S = T.symbolic("max_seq_len")
    C = T.symbolic("num_blocks")
    B = block_size
    D = head_dim
    H = num_heads
    SK = int(split_kv)
```

```
BLOCK_BYTES = B * (D + 4)
SCALE_OFFSET = B * D

@tilelang.jit(pass_configs={**pass_configs, tilelang.PassConfigKey.TL_DISABLE_SAFE_
MEMORY_ACCESS: True})
def fp8_paged_mqa_logits(
    q: T.Tensor[(N, H, D), FP8],
    kvcache_u8: T.Tensor[(C, BLOCK_BYTES), UINT8],
    weight: T.Tensor[(N, H), FP32],
    seq_lens: T.Tensor[(N,), INT32],
    page_table: T.Tensor[(N, L), INT32],
    o: T.Tensor[(N, S), FP32],
) -> None:
    # 内核实现：每个 CU 处理一个 batch 和 split 后的分块
    with T.Kernel(N * SK) as bxs:
        bx = bxs % N
        pid_split = bxs // N
        seq_len = seq_lens[bx]
        np_total = T.ceildiv(seq_len, B)
        stride = T.ceildiv(np_total, SK)
        # ... 后续按分块计算 logits，存储在 o 中
        # 注意：clear_accum 和 split_kv 参数控制累加策略
        pass # 具体计算略

return fp8_paged_mqa_logits
```

评论区精华

主要讨论集中在几个方面：

- 设计：DarkSharpness 建议将 compressor 相关改动移到单独文件（如 compress_hip.py）以避免 diff 过大，PR 采纳该建议并重构。
- 精度：DarkSharpness 担心 tgemm.mm(x, y, otype=x.dtype).float() 中 bf16 in/out 再 cast 到 fp32 可能导致精度退化，PR 作者 kkHuang-amd 回应实测未见退化。
- 文件命名：DarkSharpness 对 hip_flash_mla.py 的必要性提出疑问，PR 作者表示会专门针对 ROCm 修复该文件。
- 风格：DarkSharpness 建议将 jit_kernel/deepseek_v4.py 中新增的 docstring 移动到文件开头，已接受。
  - compressor 文件拆分建议 (design): 作者已采纳，创建了 compress_hip.py。
  - bf16 精度问题 (correctness): 作者回应实测 GSM8K 未见精度退化，且 DSV4 要求 bf16 in fp32 out 但内核本身使用 fp32 累加。
  - hip_flash_mla.py 是否需要保留 (question): 作者回应会修复该文件以便在 ROCm 上专用运行，保留该文件但后续优化。
  - docstring 位置调整 (style): 作者接受该建议，后续提交中调整。

风险与影响

- 风险：主要技术风险包括：
  - CUDA 回归风险：多处使用了 is_hip() 分支，若未充分测试可能影响 CUDA 路径（已在 commit 中明确 fix breakage）。
  - 内核兼容性：新增的 Triton/TileLang 内核依赖 ROCm 环境，可能存在编译器兼容问题或性能瓶颈。
  - 环境变量复杂度：大量环境变量（如 SGLANG_HACK_FLASHMLA_BACKEND、SGLANG_OPT_USE_FUSED_COMPRESS）控制行为，配置不当可能导致运行时错误。
  - 缺少单元测试：PR 未添加自动化测试，仅提供手动 GSM8K 精度验证（0.948），未来回归风险较高。
  - 影响：对用户：AMD ROCm 用户首次能在 MI35x 上运行 DeepSeek V4 系列模型（flash/pro），开启后续优化。CUDA 用户无影响。对系统：新增了完整的 HIP 注意力后端和多个 Triton 内核，代码侵入性中等（通过 is_hip 隔离）。对团队：后续将有多个 PR 迁移剩余优化（压缩流融合、多 stream 等），此 PR 奠定架构基础。
  - 风险标记：缺少测试覆盖，环境变量复杂度，CUDA 回归风险，内核兼容性

关联脉络

- 暂无明显关联 PR