# PR #23122 完整报告

sgl-project/sglang

[NPU] DFlash Speculative Decoding Support NPU

合并时间：2026-05-30 15:13

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23122

## 执行摘要

- 一句话：适配 DFlash 推测解码至 Ascend NPU 平台
- 推荐动作：该 PR 是 DFlash 支持 NPU 的必要适配，展示了如何通过条件分支和硬件特定算子扩展新后端。对于想了解 SGLang 硬件适配模式的读者，值得精读。同时，建议跟进后续可能增加的单元测试和更多模型的验证。

## 功能与动机

Given that dFlash has been merged, this PR adapts the changes for NPU based on the following draft: https://github.com/sgl-project/sglang/pull/18343.

## 实现拆解

1. 新增 NPU 融合 forward 路径：在 dflash.py 的 DFlashAttention 类中新增 forward_prepare_npu 方法，调用 sgl_kernel_npu 的 split_qkv_rmsnorm_rope 融合算子，一次性完成 QKV 投影、RMSNorm 和 RoPE，避免多次 kernel launch。forward 方法通过全局 _is_npu 标志选择路径。
2. 注册 ascend 注意力后端：在 dflash_worker.py 中将 "ascend" 加入 supported_draft_backends 元组，使 NPU 场景可选用 ascend 后端。在 _append_target_hidden_sequential 中，对 NPU 调用 forward_prepare_npu 替换原有的 kv_proj_only + apply_k_norm + apply_k_rope 三步操作。
3. 处理 DFlash 草稿输入：在 npu_graph_runner.py 的 replay 方法中，当是 DFlash 草稿模型且 input_embeds 不为空时，将其拷贝到 buffer 中以支持图重放。
4. 支持非因果注意力：在 ascend_backend.py 的注意力 forward 中，根据 layer.attn_type == AttentionType.ENCODER_ONLY 动态选择 mask=None 和 sparse_mode=0，解除原有因果掩码限制，适配 DFlash 的非因果草稿注意力。
5. 配套调整：在 dflash.py 和 dflash_worker.py 中添加 is_npu 导入和条件判断，确保 NPU 路径仅在相应硬件上启用。

关键文件：

- python/sglang/srt/models/dflash.py（模块 草稿模型；类别 source；类型 data-contract；符号 forward_prepare_npu, DFlashAttention.forward）：核心模型文件，新增 NPU 特定的 QKV 融合算子路径 forward_prepare_npu，并在 forward 中增加 NPU 条件分支。

- python/sglang/srt/speculative/dflash_worker.py（模块 推测工作流；类别 source；类型 dependency-wiring；符号 _append_target_hidden_sequential, supported_draft_backends）：工作流文件，注册 ascend 注意力后端，并调整 context KV 填充路径以使用 NPU 融合算子。
- python/sglang/srt/hardware_backend/npu/graph_runner/npu_graph_runner.py（模块 图执行器；类别 source；类型 core-logic；符号 replay）：NPU graph runner，处理 DFlash 草稿输入时需要复制 input_embeds 到 buffer。
- python/sglang/srt/hardware_backend/npu/attention/ascend_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 forward_mtp）：NPU 注意力后端，针对 ENCODER_ONLY 注意力类型解除因果掩码约束，适配 DFlash 非因果注意力。

关键符号：DFlashAttention.forward_prepare_npu, DFlashAttention.forward, AscendBackend.forward_mtp, NpuGraphRunner.replay, DFlashWorker._append_target_hidden_sequential

## 关键源码片段

### python/sglang/srt/models/dflash.py

核心模型文件，新增 NPU 特定的 QKV 融合算子路径 forward_prepare_npu，并在 forward 中增加 NPU 条件分支。

```
class DFlashAttention(nn.Module):
    # ... ( 初始化代码略 )

    def forward_prepare_npu(self, positions, hidden_states):
        # NPU 融合路径：一次 GEMM 后使用 sgl_kernel_npu 的 split_qkv_rmsnorm_rope
        # 同时完成 QKV 拆分、RMSNorm 和 RoPE，避免多次 kernel launch
        qkv, _ = self.qkv_proj(hidden_states)
        if self.attn.layer_id == 0:
            # 只在第一层预计算 cos/sin 表，后续层复用
            self.rotary_emb.get_cos_sin_with_position(positions)
        q, k, v = split_qkv_rmsnorm_rope(
            qkv,
            self.rotary_emb.position_sin,
            self.rotary_emb.position_cos,
            self.q_size,
            self.kv_size,
            self.head_dim,
            eps=self.q_norm.variance_epsilon,
            q_weight=self.q_norm.weight,
            k_weight=self.k_norm.weight,
            q_bias=getattr(self.q_norm, "bias", None),
            k_bias=getattr(self.k_norm, "bias", None),
        )
        return q, k, v

    def forward(self, positions, hidden_states, forward_batch):
```

```
        # 根据硬件选择计算路径
        if _is_npu:
            q, k, v = self.forward_prepare_npu(positions, hidden_states)
        else:
            qkv, _ = self.qkv_proj(hidden_states)
            q, k, v = qkv.split([self.q_size, self.kv_size, self.kv_size], dim=-1)
            q, k = apply_qk_norm(q, k, self.q_norm, self.k_norm, self.head_dim)
            q, k = self.rotary_emb(positions, q, k)
        # 共享的非因果注意力计算
        attn_output = self.attn(q, k, v, forward_batch)
        output, _ = self.o_proj(attn_output)
        return output
```

## python/sglang/srt/hardware_backend/npu/attention/ascend_backend.py

NPU 注意力后端，针对 ENCODER_ONLY 注意力类型解除因果掩码约束，适配 DFlash 非因果注意力。

```
if forward_batch.forward_mode.is_draft_extend():
    actual_seq_lengths = (
        np.array(forward_batch.extend_seq_lens_cpu).cumsum().tolist()
    )
else:
    actual_seq_lengths = np.arange(
        self.speculative_num_draft_tokens,
        self.speculative_num_draft_tokens + query.shape[0],
        self.speculative_num_draft_tokens,
    )
# DFlash 使用 ENCODER_ONLY 非因果注意力，需调整 mask 与 sparse_mode
if layer.attn_type == AttentionType.ENCODER_ONLY:
    mask = None
    sparse_mode = 0 # 无稀疏约束
else:
    mask = self.mtp_mask
    sparse_mode = 3 # 默认因果稀疏模式

attn_output, _ = torch.ops.npu.npu_fused_infer_attention_score(
    query,
    k_cache,
    v_cache,
    block_table=self.forward_metadata.block_tables,
    block_size=self.page_size,
    num_heads=layer.tp_q_head_num,
    num_key_value_heads=layer.tp_k_head_num,
    input_layout="TND",
    atten_mask=mask, # 根据注意力类型动态传入
    scale=layer.scaling,
    actual_seq_lengths=actual_seq_lengths,
    actual_seq_lengths_kv=actual_seq_lengths_kv,
    sparse_mode=sparse_mode, # 根据注意力类型动态选择
```

）

## 评论区精华

该 PR 没有经过人工审查讨论，仅由 sglang-npu-bot 自动批准。Body 中提供了详细的精度和速度基准测试，表明在 Ascend910 上 DFlash 可达到 1.3-3x 加速，精度与基线相当。

- 暂无高价值评论线程

## 风险与影响

- 风险：
  1. 测试覆盖不足：变更仅依赖一个模型（Qwen3-8B）和单一数据集（GSM8K）进行验证，缺乏对其他模型和更多场景的测试。
  2. 代码隔离风险：通过全局变量 _is_npu 进行条件分支，若 CUDA 或 NPU 路径有未同步的更改，可能导致行为不一致。
  3. 缺少单元测试：没有添加针对 NPU 特定路径的单元测试，回归风险较高。
  4. 性能依赖：NPU 融合算子 split_qkv_rmsnorm_rope 依赖于 sgl_kernel_npu 库，该库的更新可能影响性能或正确性。 - 影响：对用户：Ascend NPU 用户现在可以启用 DFlash 推测解码来加速推理，无需等待人工适配。对系统：增加了约 68 行代码，主要集中在条件分支和 NPU 特定调用，维护成本可控。影响范围限定在 NPU 后端和 DFlash 模块。 - 风险标记：缺少测试覆盖，仅单模型验证，NPU 特定路径缺乏回归，依赖硬件平滑运行

## 关联脉络

- PR #18343 [WIP] DFlash Speculative Decoding Support: 本 PR 基于该草稿进行适配