PR #25810 完整报告

sgl-project/sglang

perf(dsv4): add MHC token-count prewarm

合并时间：2026-05-21 13:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25810

执行摘要

- 一句话：DSV4 MHC pre 启动预热，消除冷启动尾部延迟
- 推荐动作：建议阅读 model_runner.py 中的钩子扩展方式和 mhc.py 中代表桶计算逻辑，这是模型特定预热与框架解耦的典型设计。此外，prewarm_mhc_token_counts 方法对显存和时间的权衡（通过 del 及时释放）也值得借鉴。本 PR 无显著风险，可正常合入。

功能与动机

DeepSeek V4 MHC pre 在稀有的 token-count bucket 首次进入服务进程时，会触发 20-40s 的前向 stall（PR body 原文：'In internal DSV4 profiling runs, the long 20-40s forward stalls aligned with first-seen MHC pre split buckets in sparse mixed/decode tail.'）。这严重影响尾延迟和吞吐量稳定性。本 PR 将首次加载成本转移到启动预热阶段，使线上流量和基准测试不再为此买单。

实现拆解

1. 新增 token-count 代表桶计算（python/sglang/srt/layers/mhc.py）：定义 get_mhc_pre_token_count_representatives 函数，基于 _compute_num_split_for_mhc_pre 枚举每个 split bucket，并取每个 bucket 中最大的 token count 作为代表值，避免预热所有 token count。
2. DecoderLayer 层预热方法（python/sglang/srt/models/deepseek_v4.py 内 DeepseekV4DecoderLayer）：新增 prewarm_mhc_token_counts 方法，遍历 attn 和 ffn 两条路径，对每个代表 token count 构造虚拟 residual 张量，调用 self.hc_pre，同步 CUDA 后释放。prewarm_mhc_token_count_buckets 方法负责调用代表桶计算后触发预热并返回使用的 token counts。
3. 模型级预热 hook（python/sglang/srt/models/deepseek_v4.py 内 DeepseekV4ForCausalLM）：新增 kernel_warmup(self, model_runner) 方法，作为模型专属预热钩子。它首先检查 is_hybrid_swa 及两个环境变量开关（SGLANG_OPT_DEEPGEMM_HC_PRENORM、SGLANG_OPT_USE_TILELANG_MHC_PRE），然后从 model_runner.server_args.chunked_prefill_size 推导 max_num_tokens（DP attention 下自动被 DP size 归一），最后委托给第一层 decoder layer 的 prewarm_mhc_token_count_buckets。
4. NextN 模型暴露（python/sglang/srt/models/deepseek_v4_nextn.py）：DeepseekV4NextnDecoderLayer 新增 prewarm_mhc_token_count_buckets 委托给内部

的 self.decoder，使得 NextN 包装路径也能触发预热。

5. ModelRunner 框架扩展（python/sglang/srt/model_executor/model_runner.py）：修改 kernel_warmup 方法，在原有的 FlashInfer autotune 之后，通过 getattr(self.model, 'kernel_warmup', None) 调用模型专属钩子，保持框架通用性。
6. 配置与守卫：无新增用户配置项；预热范围直接复用 --chunked-prefill-size，仅当两个环境变量同时启用且为 hybrid SWA 时才执行，不影响其他模型或功能。

关键文件：

- python/sglang/srt/models/deepseek_v4.py（模块 模型层；类别 source；类型 core-logic；符号 prewarm_mhc_token_counts, prewarm_mhc_token_count_buckets, kernel_warmup）：核心预热逻辑所在：在 DeepseekV4DecoderLayer 中新增 prewarm_mhc_token_counts 和 prewarm_mhc_token_count_buckets，在 DeepseekV4ForCausalLM 中新增 kernel_warmup 钩子，以及模型级 prewarm_mhc_token_count_buckets 委托。整个预热的实现骨架和条件守卫均在此文件。
- python/sglang/srt/layers/mhc.py（模块 MHC 层；类别 source；类型 core-logic；符号 get_mhc_pre_token_count_representatives）：新增 get_mhc_pre_token_count_representatives 函数，它是预热逻辑的“调度”核心：通过遍历 1..max_num_tokens 并调用已有的 _compute_num_split_for_mhc_pre 为每个 split bucket 选取代表 token count，确保每个 bucket 只预热一个代表值，大幅减少预热总次数。
- python/sglang/srt/model_executor/model_runner.py（模块 模型执行器；类别 source；类型 data-contract）：扩展了 kernel_warmup 框架方法，在原有的 FlashInfer autotune 之后，通过 getattr(self.model, 'kernel_warmup', None) 触发模型专属预热钩子。这一变化是框架与模型解耦的关键设计点。
- python/sglang/srt/models/deepseek_v4_nextn.py（模块 模型层；类别 source；类型 data-contract；符号 prewarm_mhc_token_count_buckets）：为 NextN 包装模型新增 prewarm_mhc_token_count_buckets 委托方法，确保使用 NextN 路径时预热也能正常触发。改动很小但保证了功能完整性。

关键符号：DeepseekV4DecoderLayer.prewarm_mhc_token_counts, DeepseekV4DecoderLayer.prewarm_mhc_token_count_buckets, DeepseekV4ForCausalLM.kernel_warmup, DeepseekV4ForCausalLM.prewarm_mhc_token_count_buckets, get_mhc_pre_token_count_representatives, ModelRunner.kernel_warmup

关键源码片段

python/sglang/srt/models/deepseek_v4.py

核心预热逻辑所在：在 DeepseekV4DecoderLayer 中新增 prewarm_mhc_token_counts 和 prewarm_mhc_token_count_buckets，在 DeepseekV4ForCausalLM 中新增 kernel_warmup 钩子，以及模型级 prewarm_mhc_token_count_buckets 委托。整个预热的实现骨架和条件守卫均在此文件。

```
# python/sglang/srt/models/deepseek_v4.py

class DeepseekV4DecoderLayer(nn.Module):
```

```
    # ... 省略现有 __init__ 和 hc_pre ...

    def prewarm_mhc_token_counts(
        self, token_counts: Tuple[int, ...], device: torch.device
    ) -> None:
        """对每个代表 token count 分别运行 attn 和 ffn 路径的 hc_pre。"""
        paths = (
            ("attn", self.hc_attn_fn, self.hc_attn_scale, self.hc_attn_base, self.input_layernorm),
            ("ffn", self.hc_ffn_fn, self.hc_ffn_scale, self.hc_ffn_base, self.post_attention_layernorm)
            ,
        )
        with torch.inference_mode():
            for num_tokens in token_counts:
                for path_name, hc_fn, hc_scale, hc_base, norm in paths:
                    tic = time.perf_counter()
                    # 创建 dummy 输入，shape 与真实推理保持一致
                    residual = torch.empty(
                        (num_tokens, self.hc_mult, self.hidden_size),
                        dtype=torch.bfloat16,
                        device=device,
                    )
                    y, post, comb, _ = self.hc_pre(residual, hc_fn, hc_scale, hc_base, norm=norm)
                    # 及时释放，避免 OOM
                    del residual, y, post, comb
                    torch.cuda.synchronize()
                    logger.info(
                        "DeepSeek V4 MHC prewarm path=%s num_tokens=%s completed in %.3fs",
                        path_name, num_tokens, time.perf_counter() - tic,
                    )

    def prewarm_mhc_token_count_buckets(
        self, max_num_tokens: int, device: torch.device
    ) -> Tuple[int, ...]:
        """计算代表 token counts 并执行预热，返回实际使用的 token counts。"""
        from sglang.srt.layers.mhc import get_mhc_pre_token_count_representatives
        token_counts = get_mhc_pre_token_count_representatives(
            max_num_tokens, self.hc_mult * self.hidden_size
        )
        if not token_counts:
            return token_counts
        logger.info(
            "DeepSeek V4 MHC prewarm max_num_tokens=%s representative token counts: %s",
            max_num_tokens, token_counts,
        )
        self.prewarm_mhc_token_counts(token_counts, device)
        return token_counts
```

python/sglang/srt/layers/mhc.py

新增 get_mhc_pre_token_count_representatives 函数，它是预热逻辑的“调度”核心：通过遍历 1..max_num_tokens 并调用已有的 _compute_num_split_for_mhc_pre 为每个 split bucket 选取代表 token count，确保每个 bucket 只预热一个代表值，大幅减少预热总次数。

```
# python/sglang/srt/layers/mhc.py

def get_mhc_pre_token_count_representatives(
    max_num_tokens: int, hc_hidden_size: int
) -> Tuple[int, ...]:
    """Return one token-count representative for each MHC pre split bucket.

    MHC pre 内部会根据 token count 和 hidden size 决定 split 数量 (n_splits)。
    同一个 n_splits 的 token count 共享相同的 kernel 编译结果，因此只需预热
    每个 bucket 内的最大 token count（即 bucket 触发的最后一个 token count），
    即可覆盖整个 bucket 的编译开销。
    """
    if max_num_tokens <= 0:
        return tuple()

    representatives_by_split: dict[int, int] = {}
    for num_tokens in range(1, max_num_tokens + 1):
        n_splits = _compute_num_split_for_mhc_pre(num_tokens, hc_hidden_size)
        # 同一个 n_splits 只保留最大的 num_tokens
        representatives_by_split[n_splits] = num_tokens

    return tuple(
        representatives_by_split[n_splits]
        for n_splits in sorted(representatives_by_split)
    )
```

python/sglang/srt/model_executor/model_runner.py

扩展了 kernel_warmup 框架方法，在原有的 FlashInfer autotune 之后，通过 getattr(self.model, 'kernel_warmup', None) 触发模型专属预热钩子。这一变化是框架与模型解耦的关键设计点。

```
# python/sglang/srt/model_executor/model_runner.py

class ModelRunner:
    # ...
    def kernel_warmup(self):
        """
        Warmup and tune kernels before cuda graph capture.
        Covers framework-level warmups and optional model-specific warmups.
        """
        if self.device != "cuda":
            return

        if self._should_run_flashinfer_autotune():
            self._flashinfer_autotune()
```

```
# 模型可以通过定义同名 hook 注册自己的预热逻辑（如 DeepSeek V4 MHC pre）
model_kernel_warmup = getattr(self.model, "kernel_warmup", None)
if model_kernel_warmup is not None:
    model_kernel_warmup(self)
```

评论区精华

审阅者 Fridge003 在评论中指出 prewarm_mhc_token_counts 内部调用了 deep_gemm.tf32_hc_prenorm_gemm 的 JIT 编译，建议后续将其包装到 python/sglang/srt/layers/deep_gemm_wrapper/entrypoint.py 中以复用其他 DeepGEMM 核的预热管道，但认为本PR较为紧急，可留待未来改进。该thread状态为已关闭，未产生分歧。

- 是否将 deep_gemm.tf32_hc_prenorm_gemm 包装到 deep_gemm_wrapper 中 (design): 认同该方向，但 PR 紧急，留待未来优化（deferred）。

风险与影响

- 风险：
  - 启动时间增加：预热会引入约 28s 的额外启动耗时（见 benchmark 日志），对于频繁重启或弹性扩缩容场景可能不可忽略。
  - 条件守卫复杂度：预热受三层条件控制（is_hybrid_swa、两个环境变量），若环境变量间不一致或未来修改 hc_pre 分支逻辑，预热可能跳过或重复执行。
  - 仅限特定配置：预热仅作用于 DSV4 + hybrid SWA + deep_gemm hc prenorm + TileLang MHC pre 组合，不影响其他模型或配置，但维护时需同步更新。
  - 无单元测试：本次未添加直接测试文件，预热路径的正确性依赖集成测试（如后续的 DSV4 flash FP4 测试）。
  - DP attention 下 chunked_prefill_size 归一：PR body 已说明 DP 场景下 ServerArgs 会提前除以 dp_size，ModelRunner 不应再次划分；但在未来重构中需注意此假设是否保持。
  - 影响：影响范围：仅限 DeepSeek V4 模型且开启 SGLANG_OPT_DEEPGEMM_HC_PRENORM=1 和 SGLANG_OPT_USE_TILELANG_MHC_PRE=1 的 hybrid SWA 场景。对用户透明，无需新增配置参数。影响程度：消除罕见 token bucket 首次到达时的 20-40s 长 stall，稳定尾延迟并带来约 4% 的吞吐提升（B300 benchmark 结果）。启动时间增加约 28s，属于一次性成本。团队影响：降低了 DSV4 推理服务的性能抖动，便于容量规划和 SLA 保障。
- 风险标记：启动时间增加约 28s，三层条件守卫可能遗漏未来变体，无单元测试覆盖预热路径，DP 下 chunked_prefill_size 归一假设需持续维护

关联脉络

- PR #25646 fix deepseek v4 hisparse: 同为 DSV4 MHC 模块的 bugfix，修改 mhc.py 中的压缩器逻辑，与本次预热功能在同一功能线上。

- PR #25860 add git gemm warpper for dispatch_bf16_fp32_backend: 引入 DeepGemm wrapper，与 Fridge003 建议将 MHC pre 内核包装到 wrapper 的思路同向。