PR #27436 完整报告

sgl-project/sglang

[diffusion] Enable breakable CUDA graph (BCG) for diffusion DiTs

合并时间：2026-07-08 14:45

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27436

执行摘要

- 一句话：为扩散 DiT 模型启用 Breakable CUDA Graph
- 推荐动作：该 PR 是扩散推理性能优化的关键里程碑，核心架构设计（BCG Runner + Prompt Padding）具有较高参考价值。建议团队阅读 runner.py、breakable_cuda_graph.py 和 prompt_padding.py 理解整体机制。未来添加新模型时务必参考已有 padder 实现，并在 CI 中添加对应 BCG 回归测试。

功能与动机

扩散 DiT 模型的前向中包含动态注意力等无法被单一 CUDA Graph 捕获的操作，导致整个前向必须运行在 Eager 模式。Breakable CUDA Graph 允许将图安全的操作（MLP、调制层等）捕获为多个图段，中间用 Eager 断点衔接，从而在保留动态灵活性同时最大化 CUDA 图收益。PR Body 提供了 Qwen-Image 和 Z-Image 等模型的 Profiler 对比，直观显示 BCG 模式下 GPU 执行时间缩短。

实现拆解

1. BCG 原语重定位：将 python/sglang/srt/model_executor/runner_backend_utils/breakable_cuda_graph/ 下的核心原语（BreakableCUDAGraph、BreakableCUDAGraphCapture、eager_on_graph 等）迁移到新的共享包 python/sglang/srt/breakable_cuda_graph/，并在原位置保留向后兼容的重新导出模块。新增 get_current_replay_token 用于扩散 replay token 机制。
2. 扩散 BCG Runner：新增 python/sglang/multimodal_gen/runtime/breakable_cuda_graph/runner.py，定义 BaseBreakableCUDAGraphRunner 和 DiffusionBreakableCUDAGraphRunner。Runner 封装目标模块，处理签名提取（基于 kwargs 的形状和 dtype）、图捕获、回放和输出克隆。签名不匹配时自动回落 Eager。支持 CPU 输入到 capture device 的拷贝。
3. Prompt Padding 框架：新增 python/sglang/multimodal_gen/runtime/breakable_cuda_graph/prompt_padding.py，提供 first_tensor、select_text_bucket、pad_tensor_dim、pad_nested_dim 等工具。通过将不同长度的文本条件填充到预定义桶，使得同一 resolution 的图可以跨 prompt 复用。桶选择采用最小适配原则，超出最大桶时退回 Eager。
4. Model-Specific Padders：为每个支持模型在 breakable_cuda_graph/model_padders/ 下实现专属 pad 函数：pad_qwen_prompt_kwargs（Qwen-Image）、pad_zimage_prompt_kwargs（Z-Image，需重建 caption 频率和 mask）、

pad_ideogram_prompt_kwargs（Ideogram，需解析 indicator 标记和动态分段 mask），以及 GLM-Image 等模型的 padder。

5. 集成与服务参数：修改 python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py 中的 _predict_noise，插入 _maybe_get_bcg_runner 和 _bcg_run 调用链。新增 server_args.py 中的 --enable-breakable-cuda-graph 和 --bcg-text-buckets 参数，以及模型支持验证函数。Warmup 阶段强制使用 server-based warmup 以捕获所有 timestep 和 CFG 分支。
6. 测试与 CI：新增 test_diffusion_bcg_service_validation.py 等测试文件，对支持模型添加 BCG 模式的端到端验证。调整部分 CI 配置以适应 BCG 引入的内存和精度阈值变化。

关键文件：

- python/sglang/multimodal_gen/runtime/breakable_cuda_graph/model_padders/zimage.py（模块 Z-Image 衬垫；类别 source；类型 data-contract；符号 is_zimage_transformer, _first_caption_tensor, _caption_seq_len, _pad_caption）：Z-Image 模型专属 padder，展示最复杂的 caption 频率重建和 mask 处理逻辑，是理解 BCG 文本填充的典型范例。
- python/sglang/multimodal_gen/runtime/breakable_cuda_graph/runner.py（模块 BCG 运行器；类别 source；类型 dependency-wiring；符号 _env_int, _env_float, _map_tensors, _flatten_tensors）：BCG Runner 核心文件，定义了 BaseBreakableCUDAGraphRunner 和 DiffusionBreakableCUDAGraphRunner，实现图签名匹配、捕获与回放全套逻辑。
- python/sglang/srt/breakable_cuda_graph/breakable_cuda_graph.py（模块 BCG 原语；类别 source；类型 dependency-wiring；符号 _check_cuda_bindings, get_current_stream, get_current_replay_token, _capture_status）：模型无关的 BCG 原语层，被 LLM 和扩散运行时共享，包含流追踪、图分段、弱引用等关键机制。
- python/sglang/multimodal_gen/runtime/breakable_cuda_graph/prompt_padding.py（模块 Prompt 衬垫；类别 source；类型 dependency-wiring；符号 first_tensor, select_text_bucket, pad_tensor_dim, pad_nested_dim）：通用的 prompt 填充和桶选择工具，被所有模型 padder 依赖，定义了关键数据契约。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py（模块 去噪阶段；类别 source；类型 dependency-wiring；符号 _bcg_is_warmup, _bcg_run, _bcg_text_buckets, _bcg_pad_prompt_kwargs）：BCG 集成入口，修改了主 denoising 循环以支持 BCG runner 调用。
- python/sglang/multimodal_gen/runtime/server_args.py（模块 服务参数；类别 source；类型 core-logic；符号 _normalized_bcg_model_refs, resolved_bcg_text_buckets, _validate_breakable_cuda_graph, _adjust_breakable_cuda_graph_support）：添加 BCG 相关的服务参数和模型支持验证，控制 BCG 启用和行为。

关键符号：BreakableCUDAGraph, BreakableCUDAGraphCapture, eager_on_graph, break_graph, BaseBreakableCUDAGraphRunner, DiffusionBreakableCUDAGraphRunner, pad_qwen_prompt_kwargs, pad_zimage_prompt_kwargs, pad_ideogram_prompt_kwargs, pad_glm_image_prompt_kwargs, select_text_bucket, _bcg_run, _maybe_get_bcg_runner

关键源码片段

python/sglang/multimodal_gen/runtime/breakable_cuda_graph/model_padders/zimage.py

Z-Image 模型专属 padder，展示最复杂的 caption 频率重建和 mask 处理逻辑，是理解 BCG 文本填充的典型范例。

```
# python/sglang/multimodal_gen/runtime/breakable_cuda_graph/model_padders/zimage.py
# 展示 pad_zimage_prompt_kwargs 核心逻辑

def pad_zimage_prompt_kwargs(
    call_kwargs: dict, current_model: Any, buckets: tuple[int, ...]
) -> dict:
    # 从 encoder_hidden_states 中提取第一个 caption 张量
    caption = _first_caption_tensor(call_kwargs.get("encoder_hidden_states"))
    if caption is None:
        return call_kwargs

    seq = _caption_seq_len(caption)
    cap_freq = None
    freqs_cis = call_kwargs.get("freqs_cis")
    if isinstance(freqs_cis, (tuple, list)) and len(freqs_cis) == 2:
        cap_freq = bcg_utils.first_tensor(freqs_cis[0])
    cap_freq_len = int(cap_freq.shape[0]) if torch.is_tensor(cap_freq) else seq

    # 选择最小的适配桶，超出最大桶则回落 eager
    bucket = bcg_utils.select_text_bucket(max(seq, cap_freq_len), buckets)
    if bucket is None:
        return call_kwargs

    # 只保留 BCG 相关的关键字
    out = {
        key: value
        for key, value in call_kwargs.items()
        if key in {
            "hidden_states", "timestep", "guidance",
            "encoder_hidden_states", "encoder_attention_mask",
            "freqs_cis",
        }
    }

    # 填充 encoder_hidden_states 到 bucket 长度
    out["encoder_hidden_states"] = _pad_caption(
        out["encoder_hidden_states"], target=bucket
    )
    # 填充频率信息（可能重建）
    out["freqs_cis"] = _pad_caption_freqs(
        out.get("freqs_cis"), current_model, target=bucket
```

```
        )
        # 生成并填充 mask
        if "encoder_attention_mask" in out:
            out["encoder_attention_mask"] = _caption_mask(
                out, caption=caption, seq=seq, bucket=bucket
            )
        else:
            out["encoder_attention_mask"] = _caption_mask(
                call_kwargs, caption=caption, seq=seq, bucket=bucket
            )
        return out
```

python/sglang/multimodal_gen/runtime/breakable_cuda_graph/runner.py

BCG Runner 核心文件，定义了 BaseBreakableCUDAGraphRunner 和 DiffusionBreakableCUDAGraphRunner，实现图签名匹配、捕获与回放全套逻辑。

```
# python/sglang/multimodal_gen/runtime/breakable_cuda_graph/runner.py
# 展示 BaseBreakableCUDAGraphRunner 的 capture 和 __call__ 核心逻辑

class BaseBreakableCUDAGraphRunner:
    def capture(self, *args, **kwargs) -> None:
        """为当前签名捕获 CUDA 图段，离线广播；服务期间不再调用。"""
        sig = _signature_kwargs(kwargs)
        if sig in self._graph_cache:
            return # 已经捕获过
        with BreakableCUDAGraphCapture(self._graph, device=self._device):
            output = self._module(*args, **kwargs)
        self._graph_cache[sig] = (output, copy.deepcopy(self._graph.segments))

    def __call__(self, *args, **kwargs):
        sig = _signature_kwargs(kwargs)
        entry = self._graph_cache.get(sig)
        if entry is None:
            # 签名未缓存，回退到 eager 模式
            return self._module(*args, **kwargs)
        expected_output, _ = entry
        # 将输入拷贝到静态缓冲区
        for buf, live in zip(self._input_buffers, _flatten_kwargs(kwargs)):
            buf.copy_(live, non_blocking=True)
        # 回放图段
        for seg in self._graph.segments:
            seg.replay()
        # 读取并克隆输出（避免被后续回放覆写）
        return _clone_output(expected_output)
```

python/sglang/srt/breakable_cuda_graph/breakable_cuda_graph.py

模型无关的 BCG 原语层，被 LLM 和扩散运行时共享，包含流追踪、图分段、弱引用等关键机制。

```
# python/sglang/srt/breakable_cuda_graph/breakable_cuda_graph.py
# 展示 BreakableCUDAGraphCapture 的核心生命周期

class BreakableCUDAGraphCapture:
    """上下文管理器，进入时启动捕获模式，退出时收集图段。"""
    def __init__(self, graph, device):
        self._graph = graph
        self._device = device
        self._stream = torch.cuda.Stream(device=device)
        self._segments = []

    def __enter__(self):
        _current_capture_var.set(self)
        _current_stream_var.set(self._stream)
        _install_wait_stream_hook() # 追踪 fork/join
        self._stream.__enter__()
        return self

    def __exit__(self, *exc):
        # 结束当前活跃图段
        if _is_stream_capturing(self._stream):
            seg = torch.cuda.CUDAGraph()
            seg.capture_end()
            self._segments.append(seg)
        _uninstall_wait_stream_hook()
        self._stream.__exit__(*exc)
        _current_capture_var.set(None)
        _current_stream_var.set(None)
        _forked_streams_var.set(None)

    @property
    def segments(self):
        return list(self._segments)
```

评论区精华

Review 中三个关键问题由 gemini-code-assist[bot] 提出：

- 流同步竞态条件（Critical）：图捕获在独立流上进行，回放时若当前流未等待输入拷贝完成，可能读取垃圾数据。BBuf 通过 _replay 中的 event.synchronize() 解决。
- _clone_output 不支持 dict/ModelOutput（High）：若模型返回 dict 或 ModelOutput，_clone_output 直接返回引用导致输出被后续步骤覆写。BBuf 添加了对 dict 和 ModelOutput 的克隆支持。
- _weak_ref_if_tensor 缺少 dict 支持（Medium）：中间张量不能弱引用，阻止共享 mem pool 回收。BBuf 添加了 dict 分支。 此外，Oasis-Git 建议 runner 继承 base、capture 在初始化时完成、模型辅助函数分离到独立文件，这些后续被采纳。mickqian 关注的 getattr 使用和初始化位置也已优化。

- 流同步竞态条件 (correctness): BBuf 在 _replay 中添加 event.synchronize()，确保输入拷贝完成后再回放。
- _clone_output 不支持 dict/ModelOutput (correctness): BBuf 添加了对 dict 和 ModelOutput 的递归克隆支持。
- _weak_ref_if_tensor 缺少 dict 支持 (correctness): BBuf 在 breakable_cuda_graph.py 中添加了 dict 分支的弱引用处理。
- BCG 架构设计：runner 继承 vs 直接注入 (design): BBuf 重构后采用 BaseBreakableCUDAGraphRunner 和 DiffusionBreakableCUDAGraphRunner 的继承模式，capture 在 warmup 时完成。
- 模型特定辅助函数位置 (design): BBuf 将 padder 移至 breakable_cuda_graph/model_padders/。

风险与影响

- 风险：
  - 正确性风险：流同步处理仍需细粒度验证，尤其多流场景下；桶选择边界（seq 等于桶大小时）需确认填充函数行为。
  - 内存风险：每个（resolution, bucket, timestep 分支）组合产生独立图段，显存可能显著增长。用户需根据模型调整 --bcg-text-buckets。
  - 兼容性风险：频繁 main 合并引入的 BCG 原语 API 变化可能影响 LLM 侧稳定性，需持续跟踪 CI。
  - 测试覆盖缺口：多 resolution 切换、超桶回落 Eager 等场景缺乏覆盖。
- 影响：
  - 用户：使用支持模型时传递 --enable-breakable-cuda-graph 获得性能提升，代价是 warmup 时间延长和显存增加。其他模型无影响。
  - 系统：新增 sglang.srt.breakable_cuda_graph 共享包，影响线程内 stream 同步和 contextvar 管理；扩散 warmup 阶段变重。
  - 团队：未来添加新扩散模型需编写 padder 并加入支持列表；BCG 原语的双侧维护通过共享包减轻。
  - 风险标记：核心路径变更，内存使用增加，流同步竞态风险

关联脉络

- 暂无明显关联 PR