PR #37524 完整报告

vllm-project/vllm

[Hardware][GPU] Profiler config additional to increase it scope and annotation details

合并时间：2026-07-18 04:39

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/37524

执行摘要

- 一句话：新增 graph capture 阶段 profiler 和详细 trace annotation 配置
- 推荐动作：建议仔细阅读 gpu_model_runner.py 中 capture_model 和 _warmup_and_capture 的改动，理解如何将 profiler 注入 capture 流程。gpu_worker.py 中 detailed annotation 的 roofline metric 计算方式值得参考。整体设计清晰，类型提示方面的讨论（AbstractContextManager vs ContextManager）也值得关注。

功能与动机

依据 PR body，目的是提高 profiling scope，为 attention 操作提供更详细的 trace 信息，便于分析不同 batch size 下的算子形状。Review 中 rasmith 询问 graph capture 期间 profiling 的价值，作者以 batch size 12 的 captured graph 为例说明可以获取操作的具体 shape。

实现拆解

1. 配置层(vllm/config/profiler.py)：在 ProfilerConfig 中添加 capture_torch_profiler（bool）和 detailed_trace_annotation（bool）字段，并在 _validate_profiler_config 中增加 capture_torch_profiler 仅当 profiler 为 'torch' 时的校验。
2. Graph capture profiler 集成(vllm/v1/worker/gpu_model_runner.py)：在 capture_model 中，根据 capture_torch_profiler 和 local_rank==0 的条件，创建 torch.profiler.profile 实例或 nullcontext，将 profiler 对象通过 _capture_cudagraphs 传递给 _warmup_and_capture。在 _warmup_and_capture 中，将 profiler 作为上下文管理器包裹在 _dummy_run 外，并添加 torch.profiler.record_function 标记。
3. 详细 trace annotation(vllm/v1/worker/gpu_worker.py)：在 annotate_profile 中，当 detailed_trace_annotation 启用时，计算每个请求的 seq_len、query_len，并按 context/generation 阶段汇总 sq、sk、sqsq、sqsk 指标，构造详细 annotation 字符串；否则保持原有简单格式。
4. 测试(tests/v1/worker/test_gpu_profiler.py)：新增 TestAnnotateProfile 测试类（test_simple_format_mixed 和 test_detailed_format_mixed），验证 annotation 字符串格式正确；新增 test_profiler_entered_during_capture 测试，验证 profiler 在 _warmup_and_capture 中被正确作为上下文管理器进入和退出。

关键文件：

- vllm/config/profiler.py（模块 配置层；类别 source；类型 core-logic；符号 ProfilerConfig, _validate_profiler_config）：配置项定义和验证，新增两个字段及其校验逻辑。
- vllm/v1/worker/gpu_model_runner.py（模块 执行器；类别 source；类型 core-logic；符号 capture_model, _warmup_and_capture, _capture_cudagraphs）：核心变更：在 capture_model 中条件创建 profiler，并传递到 capture 流程。
- vllm/v1/worker/gpu_worker.py（模块 工作进程；类别 source；类型 core-logic；符号 annotate_profile）：annotate_profile 方法扩展，支持详细 roofline 注释。
- tests/v1/worker/test_gpu_profiler.py（模块 测试；类别 test；类型 test-coverage；符号 TestAnnotateProfile, test_simple_format_mixed, test_detailed_format_mixed, test_profiler_entered_during_capture）：新增针对两个配置的单元测试，验证 annotation 格式和 profiler 上下文管理。

关键符号：capture_model, _warmup_and_capture, annotate_profile, _validate_profiler_config

关键源码片段

vllm/v1/worker/gpu_model_runner.py

核心变更：在 capture_model 中条件创建 profiler，并传递到 capture 流程。

```
# vllm/v1/worker/gpu_model_runner.py ( 变更后 )

@instrument(span_name="Capture model")
def capture_model(self) -> int:
    # ... 省略前面的代码
    set_cudagraph_capturing_enabled(True)

    # Setup torch profiler for graph capture traces (conditional)
    from vllm.distributed.parallel_state import get_world_group
    local_rank = get_world_group().local_rank
    enable_profiler = (
        local_rank == 0
    ) and self.vllm_config.profiler_config.capture_torch_profiler
    if enable_profiler:
        trace_dir = (
            self.vllm_config.profiler_config.torch_profiler_dir + "/capture_traces"
        )
        profiler = torch.profiler.profile(
            activities=[
                torch.profiler.ProfilerActivity.CPU,
                torch.profiler.ProfilerActivity.CUDA,
            ],
            record_shapes=True,
            profile_memory=True,
            with_stack=True,
            on_trace_ready=torch.profiler.tensorboard_trace_handler(
```

```
                trace_dir,
                worker_name=f"graph_capture_rank_{local_rank}",
                use_gzip=True,
            ),
        )
        logger.info_once(
            "Rank %d: Torch profiler enabled for CUDA graph capture, "
            "traces will be saved to: %s",
            local_rank, trace_dir,
        )
    else:
        profiler = nullcontext() # 或 nullcontext() 当不启用时

    with self._freeze_gc(), graph_capture(device=self.device):
        # ... 原来的 capture 流程
        for runtime_mode, batch_descs in self.cudagraph_dispatcher.get_capture_descs():
            self._capture_cudagraphs(
                batch_descriptors=batch_descs,
                cudagraph_runtime_mode=runtime_mode,
                profiler=profiler, # 传递 profiler
            )
        # ... encoder graph capture

    # ... 后续代码

# _warmup_and_capture 接收 profiler 参数
def _warmup_and_capture(
    self,
    desc: BatchDescriptor,
    cudagraph_runtime_mode: CUDAGraphMode,
    profile_seq_lens: int | None = None,
    allow_microbatching: bool = False,
    num_warmups: int | None = None,
    profiler: AbstractContextManager[Any] | None = None,
):
    if profiler is None:
        profiler = nullcontext()
    # ... warmup 和 capture 逻辑
    with (
        profiler, # 作为上下文管理器，期间 profiling 处于活跃状态
        torch.profiler.record_function(
            f"capture_{desc.num_tokens}_{cudagraph_runtime_mode.name}"
        ),
    ):
        self._dummy_run(
            desc.num_tokens,
            cudagraph_runtime_mode=cudagraph_runtime_mode,
            uniform_decode=desc.uniform,
            # ... 其他参数
```

```
        )
```

vllm/v1/worker/gpu_worker.py

annotate_profile 方法扩展，支持详细 roofline 注释。

```
  # vllm/v1/worker/gpu_worker.py ( 变更后 )

  def annotate_profile(self, scheduler_output):
      if not self.profiler:
          return nullcontext()
      self.profiler.step()
      iteration_details = compute_iteration_details(scheduler_output)

      if self.vllm_config.profiler_config.detailed_trace_annotation:
          # 计算 roofline 指标
          ctx_seq_len_sum = ctx_qq_compute = ctx_qk_compute = 0
          gen_seq_len_sum = gen_qq_compute = gen_qk_compute = 0
          total_scheduled_tokens = 0
          # 构建 num_computed_tokens 映射
          new_req_ids = {r.req_id for r in scheduler_output.scheduled_new_reqs}
          num_computed_tokens_ids = {
              r.req_id: r.num_computed_tokens
              for r in scheduler_output.scheduled_new_reqs
          }
          for req_id, nc in zip(
              scheduler_output.scheduled_cached_reqs.req_ids,
              scheduler_output.scheduled_cached_reqs.num_computed_tokens,
          ):
              num_computed_tokens_ids[req_id] = nc

          for req_id, num_tokens in scheduler_output.num_scheduled_tokens.items():
              query_len = num_tokens
              total_scheduled_tokens += query_len
              seq_len = num_computed_tokens_ids.get(req_id, 0) + query_len
              if (
                  scheduler_output.scheduled_cached_reqs.is_context_phase(req_id)
                  or req_id in new_req_ids
              ):
                  ctx_seq_len_sum += seq_len
                  ctx_qq_compute += query_len * query_len
                  ctx_qk_compute += query_len * seq_len
              else:
                  gen_seq_len_sum += seq_len
                  gen_qq_compute += query_len * query_len
                  gen_qk_compute += query_len * seq_len
          annotation = (
              f"execute_{total_scheduled_tokens}"
              f"_context_{iteration_details.num_ctx_requests}"
              f"(sq{iteration_details.num_ctx_tokens}"
```

```
                f"sk{ctx_seq_len_sum}"
                f"sqsq{ctx_qq_compute}"
                f"sqsk{ctx_qk_compute})"
                f"_generation_{iteration_details.num_generation_requests}"
                f"(sq{iteration_details.num_generation_tokens}"
                f"sk{gen_seq_len_sum}"
                f"sqsq{gen_qq_compute}"
                f"sqsk{gen_qk_compute})"
            )
        else:
            annotation = (
                f"execute_context_{iteration_details.num_ctx_requests}"
                f"({iteration_details.num_ctx_tokens})"
                f"_generation_{iteration_details.num_generation_requests}"
                f"({iteration_details.num_generation_tokens})"
            )
        return self.profiler.annotate_context_manager(annotation)
```

tests/v1/worker/test_gpu_profiler.py

新增针对两个配置的单元测试，验证 annotation 格式和 profiler 上下文管理。

```
# tests/v1/worker/test_gpu_profiler.py ( 新增部分 )

class TestAnnotateProfile:
    """Tests for Worker.annotate_profile() annotation string formatting."""

    def _annotate(self, detailed: bool) -> str:
        worker = MagicMock()
        worker.vllm_config.profiler_config.detailed_trace_annotation = detailed
        worker.profiler = MagicMock()

        ctx_req = MagicMock(req_id="ctx1", num_computed_tokens=0)
        cached = CachedRequestData(
            req_ids=["gen1"],
            resumed_req_ids=set(),
            new_token_ids=[],
            all_token_ids={},
            new_block_ids=[],
            num_computed_tokens=[10],
            num_output_tokens=[1],
        )
        sched = MagicMock(
            scheduled_new_reqs=[ctx_req],
            scheduled_cached_reqs=cached,
            num_scheduled_tokens={"ctx1": 4, "gen1": 1},
        )

        Worker.annotate_profile(worker, sched)
        return worker.profiler.annotate_context_manager.call_args[0][0]
```

```
    def test_simple_format_mixed(self):
        assert self._annotate(detailed=False) == (
            "execute_context_1(4)_generation_1(1)"
        )

    def test_detailed_format_mixed(self):
        # ctx1: sq=4, sk=4, sqsq=16, sqsk=16 | gen1: sq=1, sk=11, sqsq=1, sqsk=11 | bs=5
        assert self._annotate(detailed=True) == (
            "execute_5_context_1(sq4sk4sqsq16sqsk16)_generation_1(sq1sk11sqsq1sqsk11)"
        )


def test_profiler_entered_during_capture():
    """Verifies profiler is entered as context manager in _warmup_and_capture."""
    runner = MagicMock()
    runner.compilation_config.cudagraph_num_of_warmups = 0
    mock_profiler = MagicMock()

    GPUModelRunner._warmup_and_capture(
        runner,
        desc=MagicMock(num_tokens=4, uniform=True),
        cudagraph_runtime_mode=CUDAGraphMode.FULL,
        profiler=mock_profiler,
    )

    mock_profiler.__enter__.assert_called_once()
    mock_profiler.__exit__.assert_called_once()
```

评论区精华

1. rasmith 质疑 graph capture profiling 的价值：作者解释了不同 batch size 的 captured graph trace 可以揭示操作的 shape 信息，有助于性能诊断。rasmith 后续建议创建 RFC（已创建 #45069）。
2. dllehr-amd 建议使用 info_once：避免在每次 graph capture 时重复输出日志，作者已修改。
3. dllehr-amd 建议确保 bool 类型：避免字符串非空判断导致的意外行为，作者将 capture_torch_profiler 改为 bool 类型。
4. gemini-code-assist[bot] 指出类型提示不准确：profiler 参数可能为 nullcontext，建议使用 typing.ContextManager[Any]。最终代码使用了 AbstractContextManager[Any] | None 并保留了 None 检查。

- Graph capture profiling 的价值 (design): 作者认可了价值，并进一步创建了 RFC issue #45069。
- 变量命名和类型安全性 (correctness): 字段改为 bool，日志改为 info_once。
- 类型提示不准确 (style): 最终代码使用 AbstractContextManager[Any] | None，并保留 None 检查，部分采纳。

风险与影响

- 风险：新配置均默认关闭，不影响现有默认行为。风险点：
  - capture_torch_profiler 启用后会在 GPU 上产生额外 profiling 开销，但仅在 graph capture 阶段，不影响运行时。
  - 路径相关：torch_profiler_dir 必须为有效路径，且需有写入权限，否则可能导致启动失败。
  - 测试覆盖：新增测试覆盖了 annotation 格式和 profiler 上下文管理，但未覆盖复杂的多 rank 场景。
  - 影响：
  - 用户：可获得更丰富的 profiling 信息，可选择开启 capture 阶段 trace 和详细 roofline annotation。配置向后兼容。
  - 系统：graph capture 时若开启 profiler 会增加少量时间（trace 序列化），但仅发生在启动阶段。运行时 annotation 字符串长度增加，但对性能影响可忽略。
  - 团队：提升了 profiling 诊断能力，便于定位 attention 算子瓶颈。设计模式（profiler 作为上下文参数传递）可供其他阶段复用。
  - 风险标记：新配置默认关闭，profiler 路径有效性需验证，graph capture 阶段 overhead 仅影响启动，缺少多 rank 环境下的集成测试

关联脉络

- 暂无明显关联 PR