Cuda Pipeline 同步机制

pipeline

它实现上是一个 proxy pattern, `cuda::pipeline`是每个 thread访问 `pipeline_shared_state`的 proxy

- pipeline_shared_state的模板参数也仅仅是描述pipeline会被共享的范围,和 barrier类似.
- `pipeline_shared_state`需要在共享的内存区域创建
- thread_scope的 pipeline是性能最优秀的, 它不使用任何共享资源, 用`cuda::pipeline<cuda::thread_scope_thread> pipeline = cuda::make_pipeline()`直接创建
- **make_pipeline**是一个同步操作**,**它用于初始化**`pipeline_shared_state`,**为当前线程确定**role,** 并通过线程间通信确定**group**内 **producer/consumer**的数量
- pipeline逻辑上是一个**fifo, head in, tail out,** 这个**pipeline**的元素称为**stage**
- pipeline这个 proxy有三种可能的角色, consumer, producer, both
- fifo的最大容量是编译期创建pipeline_shared_state时指定的,当 fifo中的 stage满时,后续的 producer将在 acquire时被阻塞.
- 对于 role为 producer的 pipeline:
  - pipeline.producer_acquire(); 表明当前thread开始尝试向fifo中 push数据,并需要lock相关的资源
  - 当 producer_acquire成功后,当前thread的后续async指令都可以发射到acquire获得的stage中
  - pipeline.producer_commit();提交当前thread的 async任务
  - 当 group内所有的producer都 commit后, 对应的stage将正式被push到 fifo中,这个stage将会在内部所有async任务完成后被标记为ready
- 对于 role为 consumer的 pipeline
  - pipeline.consumer_wait(); 表明当前thread需要从fifo中取数据,并需要锁定相关资源
  - 当 consumer_wait()成功后,说明 group中的 producer提交的一个stage已经 ready, 可以开始处理数据
  - pipeline.consumer_release(); 表明当前thread的任务已经完成.
  - 当 group内所有的consumer都 release后,这个 stage将被 pop出 fifo
- 注意: stage只有执行顺序的逻辑意义, fifo tail对应的是哪些数据搬运或者计算需要由用户自行track

```
// 申请一个pipeline, 同步API, 会自动记录producer和 consumer的数量
cuda::pipeline pipeline = cuda::make_pipeline(block, &shared_state, thread_role);
if (thread_role == cuda::pipeline_role::producer) {
  // Only the producer threads schedule asynchronous memcpys:
  pipeline.producer_acquire();
  size_t shared_idx = fetch_batch % stages_count;
  size_t batch_idx = fetch_batch;
  size_t global_batch_idx = block_batch(batch_idx) + thread_idx;
  size_t shared_batch_idx = shared_offset[shared_idx] + thread_idx;
```

```
  cuda::memcpy_async(shared + shared_batch_idx, global_in + global_batch_idx, sizeof(int), pipeline);
  // □□□□□
  pipeline.producer_commit();
 }
 if (thread_role == cuda::pipeline_role::consumer) {
  // Only the consumer threads compute:
  // □□□□□
  pipeline.consumer_wait();
  size_t shared_idx = compute_batch % stages_count;
  size_t global_batch_idx = block_batch(compute_batch) + thread_idx;
  size_t shared_batch_idx = shared_offset[shared_idx] + thread_idx;
  compute(global_out + global_batch_idx, *(shared + shared_batch_idx));
  pipeline.consumer_release();
 }
```

Revision #1
Created 11 January 2025 09:46:27 by Colin
Updated 12 January 2025 06:33:50 by Colin