CUDA IPC Refcounting implementation explained

Since shared CUDA memory belongs to the producer process, we need to take special precautions to make sure that it is stays allocated for entire shared tensor life-span.

It could be done manually by syncing on an event:

```
# Producer
queue.put(tensor)
event.wait()

# Consumer
tensor = queue.get()
safe_to_use_tensor = tensor.clone()
event.set()
```

However, this requires blocking producer process (and gets overcomplicated in case of multiple consumers and handling various race-conditions).

Instead, we implement cross-process reference counting for shared CUDA (and HIP) tensors, which will take care of keeping producers memory allocated for entire tensor's life-span.

Details of implementation follow.

At the moment of sending tensor, we are wrapping DataPtr of the tensor with additional structure CudaIPCSentData. It still points to the same memory, but have other behavior on destruction.

Instead of simply removing the allocated block, it checks if there are any active references to this block (references are stored in shared memory files described by CudaIPCRefCountersFile structure). If such exists, instead of deleting blocks DataPtr it is moved to the global state CudaIPCSentDataLimbo.

Each individual CudaIPCRefCountersFile contains multiple reference counters for multiple tensors. Current implementation sequentially provides next available reference counter by increasing offset.

CudaIPCSentDataLimbo is keeping references to data blocks which are not in use by producer process (i.e., tensor when out of scope), but still in use (or will be in use) by a consumer. It also tries to reduce the number of stored blocks by scanning the limbo list for blocks whose ref count has gone to zero on various events such as CudaCaching allocator haven't found any suitable block for the next allocation, the attempt of any shared block deletion, explicit call of cuda_ipc_collect.

Consumer's side wraps received data into the different structure CudaIPCReceivedData. On destruction, it takes care of decreasing reference count to the received tensor.