Tensor CUDA Stream API

A CUDA Stream is a linear sequence of execution that belongs to a specific CUDA device. The PyTorch C++ API supports CUDA streams with the CUDAStream class and useful helper functions to make streaming operations easy. You can find them in CUDAStream.h. This note provides more details on how to use Pytorch C++ CUDA Stream APIs.

Acquiring CUDA stream

Pytorch's C++ API provides the following ways to acquire CUDA stream:

1. Acquire a new stream from the CUDA stream pool, streams are preallocated from the pool and returned in a round-robin fashion.

```
CUDAStream getStreamFromPool(const bool isHighPriority = false, DeviceIndex device = -1);
```

> **Tip**
>
> You can request a stream from the high priority pool by setting isHighPriority to true, or a stream for a specific device by setting device index (defaulting to the current CUDA stream's device index).

2. Acquire the default CUDA stream for the passed CUDA device, or for the current device if no device index is passed.

```
CUDAStream getDefaultCUDAStream(DeviceIndex device_index = -1);
```

> **Tip**
>
> The default stream is where most computation occurs when you aren't explicitly using streams.

3. Acquire the current CUDA stream, for the CUDA device with index `device_index`, or for the current device if no device index is passed.

```
CUDAStream getCurrentCUDAStream(DeviceIndex device_index = -1);
```

> **Tip**
>
> The current CUDA stream will usually be the default CUDA stream for the device, but it may be different if someone called `setCurrentCUDAStream` or used `StreamGuard` or `CUDAStreamGuard`.

Set CUDA stream

Pytorch's C++ API provides the following ways to set CUDA stream:

1. Set the current stream on the device of the passed in stream to be the passed in stream.

```
void setCurrentCUDAStream(CUDAStream stream);
```

> **Attention!**
>
> This function may have nosthing to do with the current device. It only changes the current stream on the stream's device. We recommend using `CUDAStreamGuard`, instead, since it switches to the stream's device and makes it the current stream on that device. `CUDAStreamGuard` will also restore the current device and stream when it's destroyed

2. Use `CUDAStreamGuard` to switch to a CUDA stream within a scope, it is defined in CUDAStreamGuard.h

> **Tip**
>
> Use `CUDAMultiStreamGuard` if you need to set streams on multiple CUDA devices.

CUDA Stream Usage Examples

1. Acquiring and setting CUDA stream on the same device

```
// This example shows how to acquire and set CUDA stream on the same device.
// `at::cuda::setCurrentCUDAStream` is used to set current CUDA stream

// create a tensor on device 0
torch::Tensor tensor0 = torch::ones({2, 2}, torch::device(torch::kCUDA));
```

```
// get a new CUDA stream from CUDA stream pool on device 0
at::cuda::CUDAStream myStream = at::cuda::getStreamFromPool();
// set current CUDA stream from default stream to `myStream` on device 0
at::cuda::setCurrentCUDAStream(myStream);
// sum() on tensor0 uses `myStream` as current CUDA stream
tensor0.sum();

// get the default CUDA stream on device 0
at::cuda::CUDAStream defaultStream = at::cuda::getDefaultCUDAStream();
// set current CUDA stream back to default CUDA stream on devide 0
at::cuda::setCurrentCUDAStream(defaultStream);
// sum() on tensor0 uses `defaultStream` as current CUDA stream
tensor0.sum();
```

```
// This example is the same as previous example, but explicitly specify device
// index and use CUDA stream guard to set current CUDA stream

// create a tensor on device 0
torch::Tensor tensor0 = torch::ones({2, 2}, torch::device(torch::kCUDA));
// get a new stream from CUDA stream pool on device 0
at::cuda::CUDAStream myStream = at::cuda::getStreamFromPool(false, 0);
// set the current CUDA stream to `myStream` within the scope using CUDA stream guard
{
  at::cuda::CUDAStreamGuard guard(myStream);
  // current CUDA stream is `myStream` from here till the end of bracket.
  // sum() on tensor0 uses `myStream` as current CUDA stream
  tensor0.sum();
}
// current CUDA stream is reset to default CUDA stream after CUDA stream guard is destroyed
// sum() on tensor0 uses default CUDA stream on device 0 as current CUDA stream
tensor0.sum();
```

Attention!

Above code is running on the same CUDA device. *setCurrentCUDAStream* will always set current CUDA stream on current device, but note that *setCurrentCUDASteram* actually set current stream on the device of passed in CUDA stream.

2. Acquiring and setting CUDA streams on multiple devices.

```
// This example shows how to acquire and set CUDA stream on two devices.

// acquire new CUDA streams from CUDA stream pool on device 0 and device 1
at::cuda::CUDAStream myStream0 = at::cuda::getStreamFromPool(false, 0);
at::cuda::CUDAStream myStream1 = at::cuda::getStreamFromPool(false, 1);

// set current CUDA stream to `myStream0` on device 0
at::cuda::setCurrentCUDAStream(myStream0);
// set current CUDA stream to `myStream1` on device 1
at::cuda::setCurrentCUDAStream(myStream1);

// create a tensor on device 0, no need to specify device index since
// current device index is 0
torch::Tensor tensor0 = torch::ones({2, 2}, torch::device(at::kCUDA));
// sum() on tensor0 use `myStream0` as current CUDA stream on device 0
tensor0.sum();

// change the current device index to 1 by using CUDA device guard within a braket scope
{
  at::cuda::CUDAGuard device_guard{1};
  // create a tensor on device 1
  torch::Tensor tensor1 = torch::ones({2, 2}, torch::device(at::kCUDA));
  // sum() on tensor 1 uses `myStream1` as current CUDA stream on device 1
  tensor1.sum();
}

// current device is reset to device 0 after device_guard is destroyed

// acquire a new CUDA stream on device 1
at::cuda::CUDAStream myStream1_1 = at::cuda::getStreamFromPool(false, 1);
// create a new tensor on device 1
torch::Tensor tensor1 = torch::ones({2, 2}, torch::device({torch::kCUDA, 1}));

// change the current device index to 1 and current CUDA stream on device 1
// to `myStream1_1` using CUDA stream guard within a scope
{
  at::cuda::CUDAStreamGuard stream_guard(myStream1_1);
  // sum() on tensor1 use `myStream1_1` as current CUDA stream on device 1
  tensor1.sum();
}

// current device is reset to device 0 and current CUDA stream on device 1 is
// reset to `myStream1`

// sum() on tensor1 uses `myStream1` as current CUDA stream on device 1
tensor1.sum();
```

3. Working with CUDA multistream guard

```
// This example shows how to use CUDA multistream guard to set
// two streams on two devices at the same time.

// create two tensor, one on device 0, one on device 1
torch::Tensor tensor0 = torch::ones({2, 2}, torch::device({torch::kCUDA, 0}));
torch::Tensor tensor1 = torch::ones({2, 2}, torch::device({torch::kCUDA, 1}));

// acquire new CUDA streams from CUDA stream pool on device 0 and device 1
at::cuda::CUDAStream myStream0 = at::cuda::getStreamFromPool(false, 0);
at::cuda::CUDAStream myStream1 = at::cuda::getStreamFromPool(false, 1);

// set current CUDA stream on device 0 to `myStream0` and
// set current CUDA stream on device 1 to `myStream1` CUDA using multistream guard
{
  at::cuda::CUDAMultiStreamGuard multi_guard({myStream0, myStream1});

  // sum() on tensor0 uses `myStream0` as current CUDA stream on device 0
  tensor0.sum();
  // sum() on tensor1 uses `myStream1` as current CUDA stream on device 1
  tensor1.sum();
}

// current CUDA stream on device 0 is reset to default CUDA stream on device 0
// current CUDA stream on device 1 is reset to default CUDA stream on device 1

// sum() on tensor0 uses default CUDA stream as current CUDA stream on device 0
tensor0.sum();
// sum() on tensor1 uses defualt CUDA stream as current CUDA stream on device 1
tensor1.sum();
```

Attention!

`CUDAMultiStreamGuard` does not change current device index, it only changes the stream on each passed in stream's device. Other than scope controlling, this guard is equivalent to calling `setCurrentCUDAStream` on each passed in stream.

4. A skeleton example for handling CUDA streams on multiple devices

```
// This is a skeleton example that shows how to handle CUDA streams on multiple devices
// Suppose you want to do work on the non-default stream on two devices simultaneously, and we
// already have streams on both devices in two vectors. The following code shows three ways
// of acquiring and setting the streams.

// Usage 0: acquire CUDA stream and set current CUDA stream with `setCurrentCUDAStream`
// Create a CUDA stream vector `streams0` on device 0
std::vector<at::cuda::CUDAStream> streams0 =
  {at::cuda::getDefaultCUDAStream(), at::cuda::getStreamFromPool()};
// set current stream as `streams0[0]` on device 0
at::cuda::setCurrentCUDAStream(streams0[0]);

// create a CUDA stream vector `streams1` on device using CUDA device guard
std::vector<at::cuda::CUDAStream> streams1;
{
  // device index is set to 1 within this scope
  at::cuda::CUDAGuard device_guard(1);
  streams1.push_back(at::cuda::getDefaultCUDAStream());
  streams1.push_back(at::cuda::getStreamFromPool());
}
// device index is reset to 0 after device_guard is destroyed

// set current stream as `streams1[0]` on device 1
at::cuda::setCurrentCUDAStream(streams1[0]);


// Usage 1: use CUDA device guard to change the current device index only
{
  at::cuda::CUDAGuard device_guard(1);

  // current device index is changed to 1 within scope
  // current CUDA stream is still `streams1[0]` on device 1, no change
}
// current device index is reset to 0 after `device_guard` is destroyed


// Usage 2: use CUDA stream guard to change both current device index and current CUDA stream.
{
  at::cuda::CUDAStreamGuard stream_guard(streams1[1]);

  // current device index and current CUDA stream are set to 1 and `streams1[1]` within scope
}
// current device index and current CUDA stream are reset to 0 and `streams0[0]` after
// stream_guard is destroyed


// Usage 3: use CUDA multi-stream guard to change multiple streams on multiple devices
{
  // This is the same as calling `torch::cuda::setCurrentCUDAStream` on both streams
  at::cuda::CUDAMultiStreamGuard multi_guard({streams0[1], streams1[1]});

  // current device index is not change, still 0
  // current CUDA stream on device 0 and device 1 are set to `streams0[1]` and `streams1[1]`
}
// current CUDA stream on device 0 and device 1 are reset to `streams0[0]` and `streams1[0]`
// after `multi_guard` is destroyed.
```