CUDA Streams

CUDA Streams

- Stream – a logical sequence of dependent asynchronous operations, independent from operations in other streams.

- Parallel streams can potentially be more efficient than regular execution on combination of kernels and host ⇔ device transfers, feeding both I/O and computational units simultaneously.

CUDA Streams

- By default, all operations are performed in stream #0
- Asynchronous data transfers (cudaMemcpyAsync) can work only with pinned memory (cudaMallocHost or cudaHostRegister).
- Stream could be assigned to kernel, using launch config: <<<..., stream>>>(...)

Devices, Streams and Events

- CUDA streams and events:
  - Are bound to particular GPU, *current* one in the moment of stream/event creation
  - Each GPU has default stream (0)
- Using CUDA streams and events:
  - Kernel can be executed only in stream of the current GPU
  - Data transfer can be performed in stream of any GPU
  - CUDA Event can be recorded only in stream of the same GPU
- Synchronization, querying:
  - Any event or stream can be synchronized
    - > Even if event/stream is bound to the current GPU

Example 1

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;

cudaSetDevice( 0 );
cudaStreamCreate( &streamA );     // streamA and eventA are bound to device #0
cudaEventCreaet( &eventA );

cudaSetDevice( 1 );
cudaStreamCreate( &streamB );     // streamB and eventB are bound to device #1
cudaEventCreate( &eventB );

kernel<<<..., streamB>>>(...);
cudaEventRecord( eventB, streamB );
cudaEventSynchronize( eventB );
```

OK:

- device #1 is set as current
- streamB and eventB are bound to device #1

Example 2

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;

cudaSetDevice( 0 );
cudaStreamCreate( &streamA );    // streamA and eventA are bound to device #0
cudaEventCreaet( &eventA );

cudaSetDevice( 1 );
cudaStreamCreate( &streamB );    // streamB and eventB are bound to device #1
cudaEventCreate( &eventB );

kernel<<<..., streamA>>>(...);
cudaEventRecord( eventB, streamB );
cudaEventSynchronize( eventB );
```

ERROR:

- device #1 is set as current
- streamA is bound to device #0 (not current!)

Example 3

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;

cudaSetDevice( 0 );
cudaStreamCreate( &streamA );    // streamA and eventA are bound to device #0
cudaEventCreaet( &eventA );

cudaSetDevice( 1 );
cudaStreamCreate( &streamB );    // streamB and eventB are bound to device #1
cudaEventCreate( &eventB );

kernel<<<..., streamB>>>(...);
cudaEventRecord( eventA, streamB );
```

ERROR:

- eventA is bound to device #0
- streamB is bound to device #1

Example 4

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;

cudaSetDevice( 0 );
cudaStreamCreate( &streamA );     // streamA and eventA are bound to device #0
cudaEventCreaet( &eventA );

cudaSetDevice( 1 );
cudaStreamCreate( &streamB );     // streamB and eventB are bound to device #1
cudaEventCreate( &eventB );

kernel<<<..., streamB>>>(...);
cudaEventRecord( eventB, streamB );

cudaSetDevice( 0 );
cudaEventSynchronize( eventB );
kernel<<<..., streamA>>>(...);
```

device #1 is set as current

device #0 is set as current

Example 4

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;

cudaSetDevice( 0 );
cudaStreamCreate( &streamA );     // streamA and eventA are bound to device #0
cudaEventCreaet( &eventA );

cudaSetDevice( 1 );
cudaStreamCreate( &streamB );     // streamB and eventB are bound to device #1
cudaEventCreate( &eventB );

kernel<<<..., streamB>>>(...);
cudaEventRecord( eventB, streamB );

cudaSetDevice( 0 );
cudaEventSynchronize( eventB );
kernel<<<..., streamA>>>(...);
```

OK:

- device #0 is set as current
- it is allowed to sync/query events/streams bound to devices, other than the current

Example 4

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;

cudaSetDevice( 0 );
cudaStreamCreate( &streamA );     // streamA and eventA are bound to device #0
cudaEventCreaet( &eventA );

cudaSetDevice( 1 );
cudaStreamCreate( &streamB );     // streamB and eventB are bound to device #1
cudaEventCreate( &eventB );

kernel<<<..., streamB>>>(...);
cudaEventRecord( eventB, streamB );

cudaSetDevice( 0 );
cudaEventSynchronize( eventB );
kernel<<<..., streamA>>>(...);
```

OK:

- device #0 is set as current
- it is allowed to sync/query events/streams bound to devices, other than the current
- device #0 will not start executing kernel until device #1 will finish executing its kernel