Multi-GPU Programming

By: 苏辉

2013.11.12

设备，流，事件

- CUDA流 和事件是单设备（GPU）

  由当前的gpu决定

  每个设备有默认的流（aka 0-或 NULL-stream）
- 使用事件和流

  只有相同设备的流的事件才会被记录
- 同步/查询

  对任意的事件/流都可以同步/查询

Example 1

```
cudaStream_t streamA, streamB;
cudaEvent_t eventA, eventB;
cudaSetDevice( 0);
cudaStreamCreate( &streamA ); // streamA 和eventA 属于device-0
cudaEventCreaet( &eventA );
cudaSetDevice( 1 );
cudaStreamCreate( &streamB ); // streamB 和 eventB 属于device-1
cudaEventCreate( &eventB );
kernel<<<..., streamB>>>(...);
cudaEventRecord( eventB, streamB);
cudaEventSynchronize( eventB);
```

OK:
- device-1 is current
- eventB 和 streamB 属于device-1

Example 2

```
cudaStream_t streamA, streamB;
cudaEvent_t  eventA, eventB;
cudaSetDevice( 0);
cudaStreamCreate( &streamA ); // streamA 和eventA 属于device-0
cudaEventCreaet( &eventA );
cudaSetDevice( 1 );
cudaStreamCreate( &streamB ); // streamB 和 eventB 属于device-1
cudaEventCreate( &eventB );
kernel<<<..., streamA>>>(...);
cudaEventRecord( eventB, streamB);
cudaEventSynchronize( eventB);
```

错误
- device-1 is current
- streamA 属于device-0

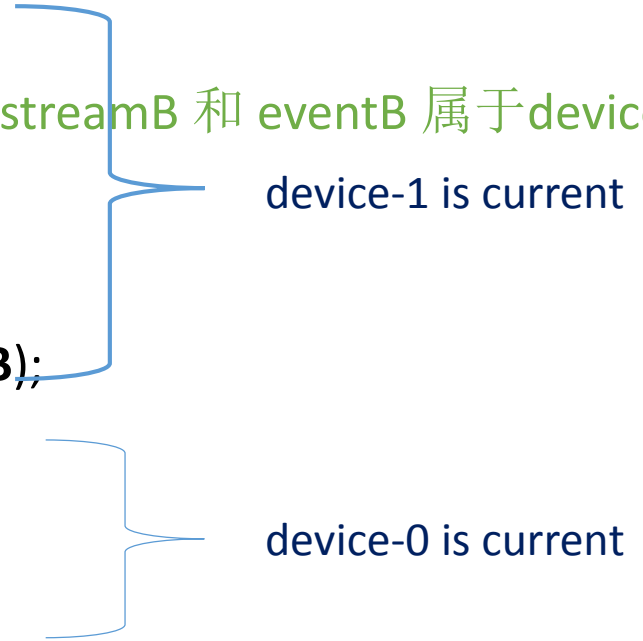
Example 3

```
cudaStream_t streamA, streamB;
cudaEvent_t  eventA, eventB;
cudaSetDevice( 0);
cudaStreamCreate( &streamA ); // streamA 和eventA 属于device-0
cudaEventCreaet( &eventA );
cudaSetDevice( 1 );
cudaStreamCreate( &streamB ); // streamB 和 eventB 属于device-1
cudaEventCreate( &eventB );
kernel<<<..., streamB>>>(...);
cudaEventRecord( eventA, streamB);
```

错误
- eventA 属于device-0
- streamB 属于device-1

Example 4

```
cudaStream_t streamA, streamB;
cudaEvent_t  eventA, eventB;
cudaSetDevice( 0);
cudaStreamCreate( &streamA ); // streamA 和eventA 属于device-0
cudaEventCreaet( &eventA );
cudaSetDevice( 1 );
cudaStreamCreate( &streamB ); // streamB 和 eventB 属于device-1
cudaEventCreate( &eventB );
kernel<<<..., streamB>>>(...);
cudaEventRecord( eventB, streamB);
cudaSetDevice( 0);
cudaEventSynchronize( eventB);
kernel<<<..., streamA>>>(...);
```



Example 4

```
cudaStream_t streamA, streamB;
cudaEvent_t  eventA, eventB;
cudaSetDevice( 0);
cudaStreamCreate( &streamA ); // streamA 和eventA 属于device-0
cudaEventCreaet( &eventA );
cudaSetDevice( 1 );
cudaStreamCreate( &streamB ); // streamB 和 eventB 属于device-1
cudaEventCreate( &eventB );
kernel<<<..., streamB>>>(...);
cudaEventRecord( eventB, streamB);
cudaSetDevice( 0);
cudaEventSynchronize( eventB);
kernel<<<..., streamA>>>(...);
```

OK:
- device-0 is current
- 同步/查询其他设备的事件/流是允许的
- device-0 不会执行内核除非device-1 完成它的内核函数

统一地址

- CPU和GPU分配使用统一的虚拟地址空间

  因此，驱动/设备可以判断数据所在的地址

  要求：64位系统

- GPU可以引用指针

  另一个GPU上的地址

  Host上的地址

UVA and Multi-GPU Programming

- 两个方面

  Peer-to-peer(P2P) memcopies

  使用另一个GPU的地址

- ---cudaDeviceEnablePeerAccess(peer_device, 0)

  允许current GPU访问peer_device GPU

  ---cudaDeviceCanAccessPeer(&accessible, dev_X, dev_Y)

  检查是否 dev_X可以访问dev_Y的内存

  返回0/1(第一个参数)

Example 5

```
int gpu1 = 0;
int gpu2 = 1;
cudaSetDevice( gpu1 );
cudaMalloc( &d_A, num_bytes );
int accessible = 0;
cudaDeviceCanAccessPeer( &accessible, gpu2, gpu1 );
if( accessible )
{
cudaSetDevice( gpu2 );
cudaDeviceEnablePeerAccess( gpu1, 0 );
kernel<<<...>>>( d_A);
}
```

虽然内核在 Gpu2上执行，它可以访问在gpu1上分配的内存（通过PCIe）

Peer-to-peer memcopy

- cudaMemcpyPeerAsync(void* dst_addr, int dst_dev,
  void* src_addr, int src_dev,
  size_t num_bytes, cudaStream_t stream)

两个设备之间拷贝字节

1）如果peer-access允许

字节在最短的PCIe路径上传输

2）如果peer-access不允许

CUDA驱动通过CPU memory传输

P2P Memcopy的好处

- 利于编程

  不必为了inter-GPU的交换而手动维护主机端的内存池

- 性能

  -----尤其当通信path不包括IOH

  (GPUs连着一个PCIe switch)

  - 单方向的传输速度达到6.6GB/s

  -----不相交的GPU-pairs能通信而不需要带宽

Inter-Gpu Commucation Cases

		Network nodes	
		Single	Multiple
Single process	Single-threaded		N/A
	Multi-threaded		N/A
Multiple processes			

GPUs可以通过P2P或者共享主机内存通信

GPUs可以通过主机端消息传递通信