CUDA Streams

Романенко А.А.
arom@ccfit.nsu.ru
Новосибирский государственный университет

CUDA потоки

* Поток (stream)– логическая последовательность зависимых асинхронных операций, независимая от операций в других потоках
* Параллельные потоки могут потенциально быть более эффективными, чем «обычное» исполнение за счет совмещения исполнения ядер и передачи данных, двунаправленной передачи данных.

CUDA потоки

* По умолчанию все операции ассоциированы с нулевым потоком
* Асинхронная передача данных (cudaMemcpyAsync)
* может работать только с pinned памятью (cudaMallocHost or cudaHostRegister).
* Ядро можно запустить в отдельном потоке указав kernel<<<…, stream>>>(…)
* Синхронизация – cudaStreamSynchronize(N)

Нулевой поток

* Используется, если не указано иначе
* Полностью синхронен
  * Как если бы cudaDeviceSynchronize() вставлен до и после каждой CUDA операции
* Исключения (асинхронно с операциями на CPU)
  * Запуск ядра
  * cudaMemcpy*Async
  * cudaMemset*Async
  * cudaMemcpy на том же устройстве
  * H2D cudaMemcpy <= 64kB

Compute capability

* Compute Capability 1.0+
  * Async kernel execution
* Compute Capability 1.1+ (i.e. C1060)
  * Async data copy (single engine)
  * asyncEngineCount
* Compute Capability 2.0+ (i.e. C2050)
  * Parallel kernel execution (Fermi – up to 16 GPU kernels)
    * concurrentKernels
  * Second copy engine

Overlapping calculation and data transfer

* It’s possible if:
  * Device has computer compatibility 1.1 and higher
  * property **asyncEngineCount** > 0
* Not supported if one copy CUDA Arrays or 2D arrays allocated with cudaMallocPitch()
* Could be blocked by environment variable CUDA_LAUNCH_BLOCKING = 1

GPU parameters

```
./deviceQuery
..
Concurrent copy and execution:              Yes with 2 copy engine(s)
..
..
Support host page-locked memory mapping:    Yes
Concurrent kernel execution:                Yes
```

Example

```
cudaMalloc ( &dev1, size ) ;
double* host1 = (double*) malloc ( &host1, size ) ;
...
cudaMemcpy( dev1, host1, size, H2D ) ;
kernel2 <<< grid, block, 0, 0 >>> ( ..., dev2, ... ) ;
kernel3 <<< grid, block, 0, 0 >>> ( ..., dev3, ... ) ;
cudaMemcpy( host4, dev4, size, D2H ) ;
```

- There is no parallelism. Default CUDA stream (0) is used.

Example

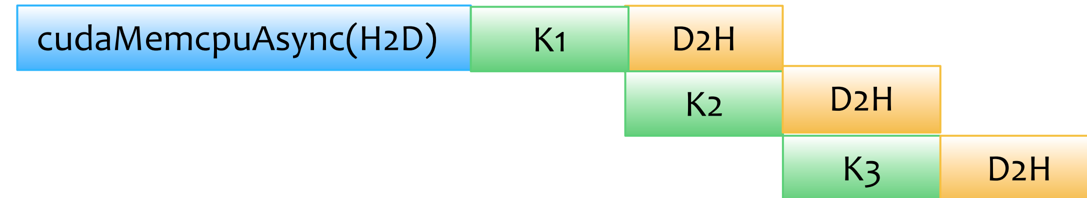
```
cudaMalloc ( &dev1, size ) ;
double* host1 = (double*) malloc ( &host1, size ) ;
...
cudaMemcpy( dev1, host1, size, H2D ) ;
kernel2 <<< grid, block>>> ( ..., dev2, ... ) ;
CPU_code();
kernel3 <<< grid, block>>> ( ..., dev3, ... ) ;
cudaMemcpy( host4, dev4, size, D2H ) ;
```

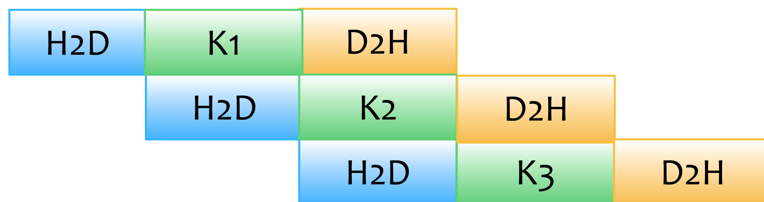
- Kernel is executed asynchronously with CPU code

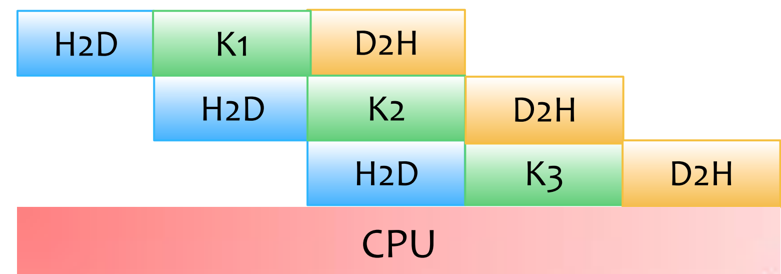
Example

```
cudaStream_t stream1, stream2, stream3, stream4;
cudaStreamCreate ( &stream1);
...
cudaMalloc ( &dev1, size );
cudaMallocHost( &host1, size ); //pinned memory required
...
cudaMemcpyAsync( dev1, host1, size, H2D, stream1);
kernel2 <<< grid, block, 0, stream2>>> ( ..., dev2, ... );
kernel3 <<< grid, block, 0, stream3>>> ( ..., dev3, ... );
cudaMemcpyAsync( host4, dev4, size, D2H, stream4);
CPU_code();
...
```

Levels of parallelism







Stream scheduling

* **Fermi has 3 execution queue:**
  * 1 compute queue
  * 2 copying queues– one for H2D, one for D2H
* **CUDA operations are set for execution as they were declared**
* **CUDA kernels are executed async only from different streams**
* **Synchronous operations block any other operations in all CUDA streams.**

Example 1

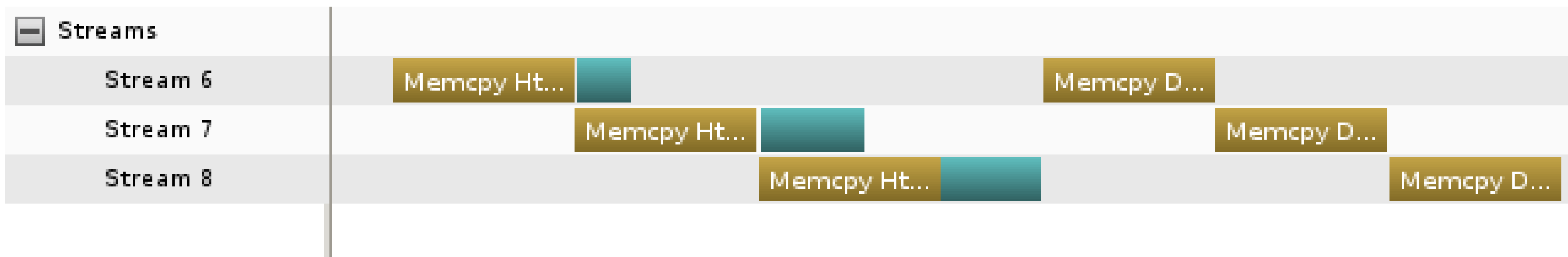
```
for (int i = 0; i < 3; ++i)
      cudaMemcpyAsync(inputDevPtr + i * size,
                      hostPtr + i * size, size,
                      cudaMemcpyHostToDevice, stream[i]);

for (int i = 0; i < 3; ++i)
      MyKernel<<<size/512, 512, 0, stream[i]>>>
                  (outputDevPtr + i * size,
                   inputDevPtr + i * size, size);

for (int i = 0; i < 3; ++i)
      cudaMemcpyAsync(hostPtr + i * size,
                      outputDevPtr + i * size, size,
                      cudaMemcpyDeviceToHost, stream[i]);
```

Example 2

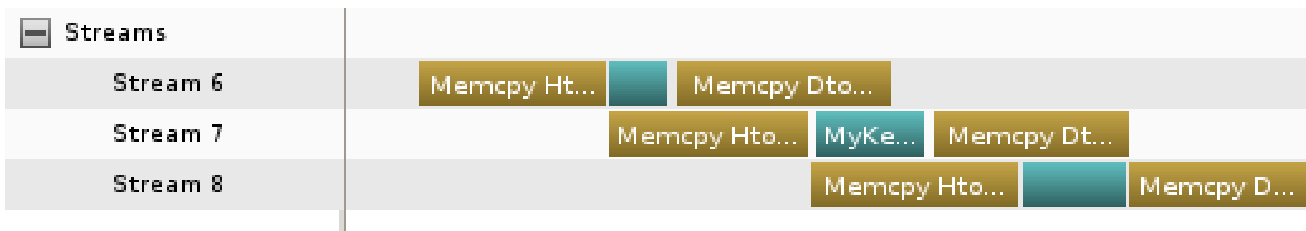
```
for(int i=0; i < 3; ++i) {
      cudaMemcpyAsync(inputDevPtr + i * size,
                      hostPtr + i * size, size,
            cudaMemcpyHostToDevice, stream[i]);

      MyKernel<<<size/512, 512, 0, stream[i]>>>
            (outputDevPtr + i * size,
             inputDevPtr + i * size, size);

      cudaMemcpyAsync(hostPtr + i * size,
                outputDevPtr + i * size, size,
               cudaMemcpyDeviceToHost, stream[i]);
}
```

Timeline

Example 1



Example 2



* CUDA streams и events:
  * Связаны с GPU
  * Каждый GPU имеет свой поток по умолчанию (0)
* Используя CUDA streams и events:
  * Kernel может исполняться только в потоке текущего GPU
  * Передача данных может проводиться в потоке любого GPU
  * CUDA Event может записываться только в потоке того же GPU
* Synchronization, querying:
  * Любой event или stream может быть синхронизирован