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 1.0+
  * Параллельность исполненя GPU / CPU
* Compute Capability 1.1+ (i.e. C1060)
  * Добавлена поддержка асинхронных операций копирования (одно устройство)
  * asyncEngineCount
* Compute Capability 2.0+ (i.e. C2050)
  * Добавлена поддержка параллельности в запуске ядер (Fermi – 16 GPU ядер)
    * concurrentKernels
  * Добавлено второе устройство для поддержки двунаправленного копирования

Совмещение передачи данных с вычислениями

* Возможно если:
  * Устройство с вычислительными возможностями (computer compatibility) выше 1.1
  * Свойство asyncEngineCount устройства > 0
* Не поддерживается, если в копирование вовлечены массивы (CUDA Arrays) или 2D массивы, выделенные через cudaMallocPitch
* Может блокироваться переменной окружения CUDA_LAUNCH_BLOCKING, установленной в 1

Параметры GPU

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

Примеры

```
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 ) ;
```

* В потоке по умолчанию все операции синхронны

Примеры

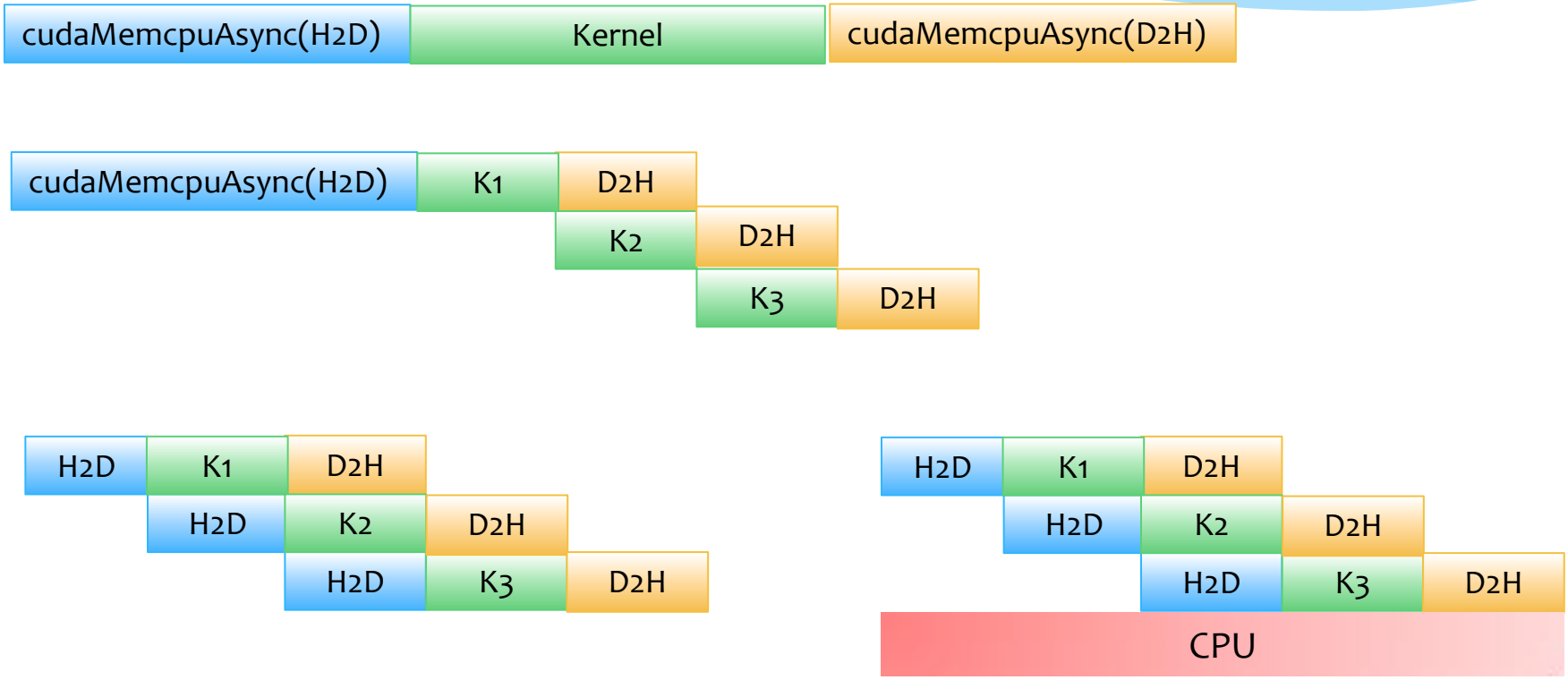
```
cudaMalloc ( &dev1, size ) ;
double* host1 = (double*) malloc ( &host1, size ) ;
...
cudaMemcpy( dev1, host1, size, H2D ) ;
kernel2 <<< grid, block>>> ( ..., dev2, ... ) ;
CPU_code();
kernel3 <<< grid, block>>> ( ..., dev3, ... ) ;
cudaMemcpy( host4, dev4, size, D2H ) ;
```

* По умолчанию исполнение ядра асинхронно с операциями на CPU

Примеры

```
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();
...
```

Уровни параллелизма



Диспетчеризация потоков

* **Fermi имеет 3 очереди исполнения**
  * 1 вычислительная
  * 2 очереди копирования – одна для H2D и одна для D2H
* **Операции CUDA отправляются на исполнение в той последовательности в которой объявлены**
  * Помещаются в соответствующие очереди
  * Операции в одном потоке зависимы и независимы между очередями
* **CUDA операция ставится на выполнение в очередь если:**
  * Предшествующая операция в той же очереди завершена,
  * Завершена операция в соответствующей очереди исполнения
  * Есть ресурсы
* **CUDA ядра исполняются параллельно только из разных потоков**
* **Синхронные операции блокируют все другие операции. Даже в других потоках**

Пример 1

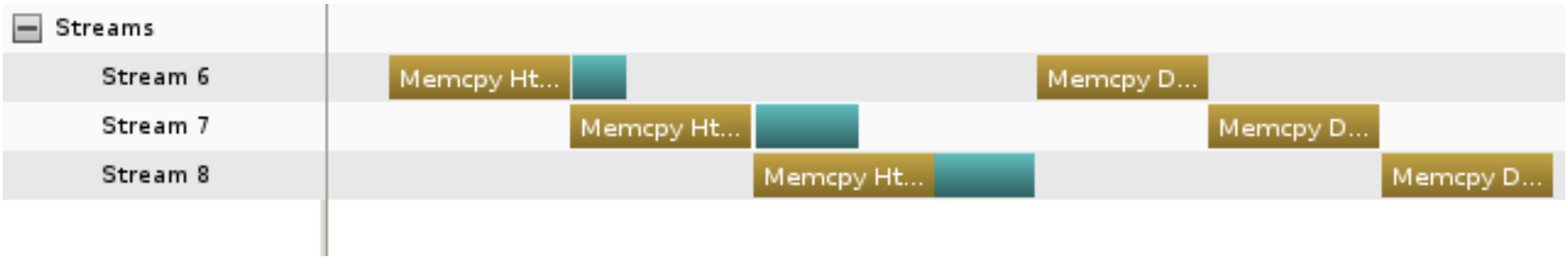
```
for (int i = 0; i < 3; ++i)
      cudaMemcpyAsync(inputDevPtr + i * size,
                      hostPtr + i * size, size,
                      cudaMemcpyHostToDevice, stream[i]);

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]);
```

Пример 2

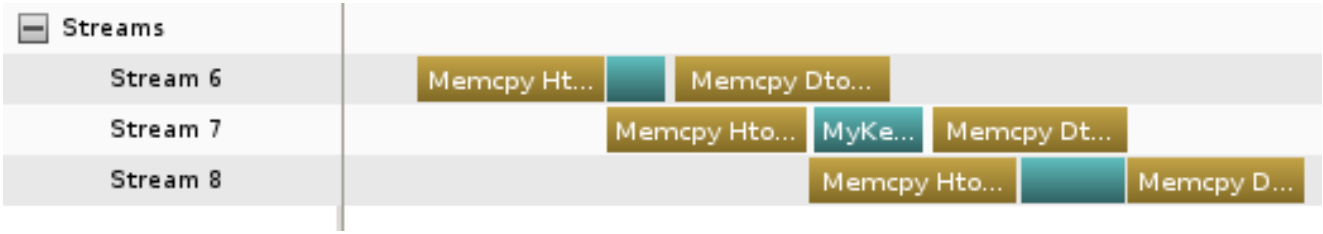
```
for(int i=0; i < 3; ++i) {
        cudaMemcpyAsync(inputDevPtr + i * size,
                        hostPtr + i * size, size,
                cudaMemcpyHostToDevice, stream[i]);

        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

Пример 1



Пример 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:
  * Любое event или stream может быть синхронизирован